Zadání bakalářské práce

Název:	Webová aplikace - Builder
Student:	Alena Ježková
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Cílem této práce je vytvoření finální webové aplikace, která zajišťuje kompletaci, kompilaci a vydání(nasazení) dané verze aplikace různých platforem. Aplikace bude jednou z komponent Apps Manageru, který se kromě této aplikace skládá také z License a Deployment Manageru.

Postupujte v těchto krocích:
1. Důkladně analyzujte prototypy Licence manageru a Builderu, které předcházely vývoji Apps Manageru. Dále se zaměřte na konkrétní potřeby společnosti Jagu v souvislosti s touto prací a popište je.
2. Na základě analýzy vytvořte vhodný návrh aplikace, která bude vhodně komunikovat s ostatními částmi Apps Manageru, na kterých pracují v rámci bakalářských prací Adam Staš a Tomáš Sládek.
3. Na základě návrhu vytvořte minimálně funkční(použitelný) prototyp aplikace.
4. Prototyp důkladně dokumentujte.
5. Výsledný prototyp řádně otestujte vhodně zvolenými testy.
6. Zhodnoťte vámi dosažené výsledky a navrhněte možná budoucí vylepšení.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 18. ledna 2023 v Praze.

Bakalářská práce

WEBOVÁ APLIKACE – BUILDER

Alena Ježková

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Jiří Hunka,˙
12. května 2023

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2022 Alena Ježková. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení, je nezbytný souhlas autora.

Odkaz na tuto práci: Ježková Alena. *Webová aplikace – Builder*. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Obsah

Poděkování vii

Prohlášení viii

Abstrakt ix

Seznam zkratek x

Úvod 1

1 Analýza 5
1.1 Proč dělat analýzu . . . 5
1.2 Metodika výzkumu . . . 5
1.3 Metodiky vývoje . . . 6
1.3.1 Vodopád . . . 6
1.3.2 Agilní vývoj . . . 6
1.3.3 Zvolená metodika . . . 6
1.4 Vznik domény . . . 6
1.5 Popis domény . . . 7
1.5.1 Slovník pojmů . . . 7
1.5.2 Zdroje informací kvalitativní metodiky . . . 7
1.5.3 Komunikace s LM . . . 8
1.5.4 Další služby . . . 8
1.5.5 Komunikace s hostingem . . . 8
1.5.6 Sestavení objednávky . . . 9
1.5.7 Deaktivace produktu zákazníka . . . 10
1.5.8 Reaktivace a zrušení produktu zákazníka . . . 10
1.5.9 Navýšení konfigurace produktu zákazníka . . . 11
1.6 Analýza implementace . . . 11
1.6.1 Zvolené technologie pro první prototyp . . . 11
1.6.2 Databáze . . . 12
1.6.3 Požadavky . . . 12
1.7 Výstup kapitoly . . . 15

2 Návrh 17
2.1 Komponenty . . . 17
2.2 Úprava požadavků . . . 17
2.3 Případy užití . . . 18
2.3.1 UC1 – Sestavení licence . . . 18
2.3.2 UC2 – Upgrade licence . . . 19
2.3.3 UC3 – Deaktivace licence . . . 19
2.3.4 UC4 – Zrušení licence . . . 20
2.3.5 UC5 – Obnovení licence . . . 20
2.3.6 UC6 – Zobrazení všech aplikací . . . 21

2.3.7 UC7 – Přidání produktu . . . 21
2.3.8 UC8 – Úprava produktu . . . 22
2.3.9 UC9 – Zobrazení produktu . . . 22
2.3.10 UC10 – Zobrazení všech produktů . . . 22
2.3.11 UC11 – Smazání produktu . . . 22
2.3.12 UC12 – Přidání komponenty . . . 23
2.3.13 UC13 – Úprava komponenty . . . 23
2.3.14 UC14 – Zobrazení komponenty . . . 23
2.3.15 UC15 – Zobrazení všech komponent . . . 23
2.3.16 UC16 – Smazání komponenty . . . 24
2.3.17 UC17 – Přiřazení produktu komponentě . . . 24
2.4 Komunikace s dalšími komponentami . . . 24
2.4.1 API . . . 24
2.5 Komunikace s dalšími službami . . . 24
2.5.1 Komunikace s hostingem . . . 24
2.5.2 Komunikace s GitLabem . . . 25
2.6 Návrh domény . . . 25
2.7 Licence . . . 25
2.8 Architektura a porovnání s předešlou architekturou . . . 25
2.9 Technologie . . . 26
2.9.1 Jazyk a prostředí . . . 27
2.9.2 Persistence dat . . . 27

3 Implementace **29**
3.1 Verzování . . . 29
3.2 Vývojové prostředí . . . 29
3.3 Běhové prostředí . . . 29
3.4 Statická analýza . . . 30
3.5 Konfigurace aplikace . . . 30
3.6 Persistence dat . . . 30
3.6.1 Modely . . . 30
3.6.2 Migrace . . . 31
3.6.3 Sequelize-cli . . . 31
3.6.4 Spouštění migrací . . . 32
3.6.5 Asociace . . . 32
3.6.6 Konfigurace . . . 32
3.6.7 Subjektivní hodnocení Sequelize . . . 32
3.7 Aplikace Builder . . . 33
3.7.1 MVC . . . 33
3.7.2 Dodržování SRP . . . 34
3.7.3 Objekty nebo třídy . . . 34
3.7.4 Továrna . . . 34
3.8 Kontrola vstupu . . . 34
3.9 Správa chyb a výjimek . . . 34
3.10 Logování . . . 35
3.11 Nové funkcionality . . . 35

4 Testování **37**
4.1 Druhy testování . . . 37
4.2 Manuální testování . . . 37
4.2.1 Průběžné testování . . . 37
4.2.2 Akceptační testování . . . 38

4.3 Automatické testování 38
4.3.1 Jednotkové testy 38
4.3.2 Integrační testování 39

A Databázový model prvotního prototypu 41

Seznam obrázků

1.1 Třívrstvá architektura . . . 11

2.1 Diagram komponent . . . 18
2.2 Zrušení licence na požadavek . . . 20
2.3 Zrušení licence po vypršení lhůty . . . 21
2.4 Datový model . . . 26
2.5 Stavový diagram licence . . . 26
2.6 MVC . . . 27

3.1 Struktura adresáře pro definování cest . . . 33

4.1 Postman – Autorizované požadavky . . . 38
4.2 Postman – Ostatní požadavky . . . 39

A.1 Databázový model prvotního prototypu . . . 42

Seznam tabulek

2.1 Pokrytí funkčních požadavků případy užití 1 . . . 18

Seznam výpisů kódu

1.1 Tělo požadavku pro sestavení objednávky . . . 9
1.2 Tělo požadavku po sestavení objednávky . . . 10
3.1 Sequelize-cli bash příkazy pro vytvoření modelů a migrací . . . 31
3.2 Zachytávání neodchycených výjimek . . . 35

Chtěla bych poděkovat především vedoucímu práce, Ing. Jiřímu Hunkovi, který mi umožnil zpracovávat toto téma ve své bakalářské práci a během její tvorby mi poskytoval cennou zpětnou vazbu. Dále pak panu Ing. Oldřichu Malcovi, který se účastnil některých konzultací a trpělivě dodával informace nutné pro pochopení požadovaných vlastností systému.
Dále bych chtěla poděkovat své rodině a svým kamarádům za jejich nepřetržitou podporu po dobu celého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracovala samostatně a že jsem uvedla 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 12. května 2023 ....

Abstrakt

Tato bakalářská práce se zabývá analýzou požadavků na komponentu systému Apps Manager, která se zabývá automatizovaným nasazením a správou těchto nasazených produktů na webových serverech. Na začátku práce je sběr požadavků, který se provádí především na základě stávajícího prototypu aplikace a dalších komponent systému, ze kterých aplikace získává vstupní data. Podle výsledků analýzy je proveden návrh systému – zejména návrh komponent, případů užití, návrh domény a databázový model. Na základě návrhu je implementován prototyp aplikace, který je podrobován průběžnému a finálnímu testování. V poslední části práce je popsán možný budoucí vývoj aplikace a shrnuty nedostatky aplikace.

Klíčová slova webová aplikace, automatizace, refaktorizace, Node.js, JavaScript

Abstract

This bachelor's thesis deals with the analysis of the requirements for the component of the Apps Manager system, which deals with the automated deployment and management of these deployed products on web servers. At the beginning of the work is the collection of requirements, which is carried out mainly on the basis of the existing prototype of the application and other components of the system, from which the application receives input data. According to the results of the analysis, the system design is carried out – especially the design of components, use cases, domain design and database model. Based on the design, a prototype of the application is implemented, which is subjected to continuous and final testing. In the last part of the work, the possible future development of the application is described and the shortcomings of the application are summarized.

Keywords web application, automatization, refactorization, Node.js, JavaScript

Seznam zkratek

AWS	Amazon Web Services
API	Application Programming Interface
CI	Cotinous Integration
CD	Continous Deployment
CLI	Command Line Interface
CRUD	Create Read Update Delete
DM	Deployment Manager
HTTP	Hyper Text Transfer Protocol
JS	JavaScript
JSON	JavaScript Object Notation
LM	License Manager
MVC	Model View Controller
OOP	Objektově Orientované Programování
ORM	Objektově relační mapování
PDF	Portable Document Format
REST	Representational State Transfer
SP1	Softwarový týmový projekt 1
SP2	Softwarový týmový projekt 2
SRP	Single Responsibility Principle
UML	Unified Modeling Language
URL	Uniform Resource Locator

Úvod

Každým dnem vzniká v současnosti velké množství softwarových aplikací. Tyto aplikace bohužel obvykle doprovází komplikovaný proces jejich nasazování zákazníkovi, který zabírá zbytečně dlouhotrvající manuální práci, například z důvodu komplexnosti aplikací a jejich závislostí, neboť je třeba konfigurovat aplikaci pro každého zákazníka zvlášť. Proces nasazování, tedy především vyplňování konfiguračních souborů a nasazování aplikací na webové servery, se neustále opakuje a je možné jej urychlit a také automatizovat. Proto vznikl nápad vytvořit systém Apps Manager pro automatické nasazování aplikací a to pro firmu Jagu s. r. o.

Práce je určena především pro prodejce softwarových aplikací, kterým vznikající aplikace Builder, jako komponenta systému Apps Manager, poslouží k automatizaci procesu nasazování prodejcem nabízených aplikací zákazníkovi.

Z výše uvedených důvodů jsem se rozhodla pro volbu tématu webová aplikace Builder.

Práce je věnována analýze, návrhu, implementaci a testování vyvíjené aplikace a dále je zaměřena na dokumentaci a budoucí vývoj.

Po seznámení se s cíli práce následuje kapitola věnována analýze aktuální situace, tzn. analýze prvotního prototypu aplikace, neboť aplikace byla vytvářena již v rámci předmětů, vyučovaných na FIT ČVUT, Softwarový týmový projekt 1 (zkr. SP1) a Softwarový týmový projekt 2 (zkr. SP2) již v roce 2022. Z výše uvedené kapitoly vyplývají otázky, jak se má aplikace dále vyvíjet, které požadavky stále nejsou splněny a co je třeba změnit pro správnou funkčnost aplikace, čemuž je věnována kapitola Návrh. Dále je v kapitole Implementace rozebrán samotný vývoj aplikace, v kapitole Testování je aplikace podrobena drobnohledu za účelem najití chyb. Celá aplikace je zdokumentována a na závěr je v kapitole Budoucí vývoj shrnuto, jaké jsou možnosti pro další zlepšení.

Jak již bylo zmíněno, tato práce navazuje na předměty SP1 a SP2, ale tyto dva semestrální projekty vycházely v první řadě z bakalářské práce Viktora Holého, který se věnoval návrhu první komponenty systému Apps Manager, License Manager. Tuto komponentu ve své bakalářské práci dále rozvíjí student FIT ČVUT Tomáš Sládek - Webová aplikace - License Manager II. V rámci systému Apps Manager existuje dále nová komponenta Deployment Manager, kterou ve své bakalářské práci, Webová aplikace Deployment Manager, popisuje student FIT ČVUT Adam Staš.

Cíle práce

Přínosem této práce je usnadnění nasazování softwarových aplikací softwarovým firmám, které řeší příliš dlouhý proces nasazování aplikací pro jednotlivé zákazníky, neboť nasazování těchto aplikací vyžaduje specifická data od zákazníků a tedy především upravování konfiguračních souborů. Tato práce se věnuje komponentě Builder, která je součástí systému Apps Manager, jež bude fungovat jako celek právě pro usnadnění celého procesu od zákazníka jako zájemce o software až po administrátora jako správce nasazování aplikací.

Cílem následující kapitoly je analýza existujících prototypů komponent systému Apps Manager, což je prototyp aplikace License Manager a prototyp aplikace Builder. Tato bakalářská práce se věnuje aplikaci Builder, proto je analýza komponenty License Manager soustředěna na části související s ní. Výstupem této části je být aktuální seznam požadavků, který platí pro Builder.

Cílem další kapitoly je vytvoření vhodného návrhu aplikace Builder pro celý systém v rámci nových požadavků a na základě konzultací s dalšími vývojáři celého systému a zadavatelem.

Cílem kapitoly implementace je vytvoření funkčního a v praxi použitelného prototypu aplikace na základě návrhu z předešlé kapitoly a důkladná implementace tohoto prototypu.

Cílem kapitoly testování je vytvoření sady testů na různých testovacích úrovních, které prošetří chování jednotlivých částí aplikace i chování celé aplikace jako celku.

Cílem kapitoly budoucí vývoj je shrnutí dosažených výsledků a možných vylepšení přínosných pro další vývojáře do budoucna.

Kapitola 1
Analýza

Počáteční fází celé práce je analýza. V ní je třeba se soustředit nejen přímo na samotnou aplikaci Builder, která je předmětem této práce, ale také na další komponenty celého systému Apps Manageru. Analýza je zaměřena na současný stav celého systému, ze kterého tato práce vychází, v porovnání s požadovanou verzí aplikace.

1.1 Proč dělat analýzu

Analýza přináší do vzniku software nový pohled. Jejím cílem je přesný popis zkoumaných dat, popis významu termínů, vazeb mez jednotlivými entitami, identifikace stavů entit a zachycení jednotlivých atributů [1]. Softwarová analýza je významný faktor při určování úspěšnosti projektu, neboť správná analýza dokáže vytvořit potřebná data k dlouhodobě udržitelnému projektu. Pokud není analýza řádně provedena, může to mít vážné následky jako je například nespokojenost zákazníka či příliš dlouhá doba vývoje software. [2]

Zaměřuje se na požadavky, které je třeba zahrnout a zkoumá, jaké jsou předpokládané funkcionality aplikace. Při použití metodik analýzy softwarového inženýrství dává systematický vhled do problematiky dané domény a poskytuje potřebná data pro návrh aplikace.

1.2 Metodika výzkumu

Pro důkladné popsání celého problému je třeba zvolit správnou metodiku výzkumu. Metodiky jsou děleny na kvalitativní a kvantitativní.

Kvantitativní metodika udává především data, která se dají kvantifikovat, tedy jsou udávána v číslech. Je to technika, která se v analýze soustředí především na statistická data, měření a podobně. [3]

Do této kategorie by se tedy dalo zařadit například měření rychlosti aplikace či pokrytí testy. Vzhledem k tomu, že na začátkem této bakalářské práce byl vytvořen pouze prvotní prototyp aplikace, který nepokrývá všechny funkcionality a prototyp aplikace nebyl testován, bylo rozhodnuto, že tato metodika zde nebude použita.

Tato kapitola je zaměřena především na kvalitativní metodiku, kterou představuje důkladná analýza již existující dokumentace a hotové implementace. Cílem je identifikovat chyby a nedostatky v předešlém návrhu aplikace, implementovaných funkcionalitách a vytvořit díky tomu aktualizovaný seznam požadavků pro vytvoření prototypu aplikace.

Kvalitativní metodika řeší oproti kvantitativní metodě veškerá data, která nejsou spojena s čísly. Zabývá se například uživatelským hodnocením systému, probíhá formou rozhovorů a je zaměřena na porozumění požadavků ze strany běžného uživatele i ze strany administrátora. [4]

Pro tuto část metodiky byla zvolena forma pravidelných konzultací se zadavatelem práce, který prezentuje především administrátorský pohled, ale také ideální uživatelský pohled na fungování celé aplikace.

Další částí kvalitativní metodiky je komunikace se členy týmu, kteří se podíleli na původním prototypu aplikace, a tedy mohou poskytnout chybějící informace, které nejsou uvedeny v dokumentaci.

1.3 Metodiky vývoje

1.3.1 Vodopád

Vodopád, neboli anglicky *waterfall* je druh metodiky vývoje software. Jedná se o sekvenční metodu vývoje software. Postupně se odvíjí jednotlivé kroky vývoje a to: analýza, návrh, implementace, testy a údržba. Po skončení všech těchto částí je ukončen celý projektu. [5]

Tento typ vývoje je již dnes méně používaný, neboť požadavky často nejsou známy na začátku vývoje a objevují se až v průběhu. Tato metodika tedy špatně reaguje na změny a navíc trvá velice dlouho, než zákazník od zadání projektu uvidí reálný výsledek. [5]

1.3.2 Agilní vývoj

Oproti vodopádu je agilní vývoj mnohem přizpůsobivější. Agilní vývoj spočívá opakování vodopádu v jednotlivých krátkých iteracích. Jednotlivé verze nemusí být tedy vždy přímo nasaditelné do produkce, ale hlavní výhodou je, že v průběhu vývoje je možné konzultovat vývoj produktu se zákazníkem a v případě potřeby aktualizovat požadavky. [5]

1.3.3 Zvolená metodika

Celá práce probíhá formou agilního vývoje a právě díky pravidelným konzultacím je možné v případě potřeby měnit během vývoje požadavky na funkcionality aplikace Builder či upravovat priority.

1.4 Vznik domény

Nápad pro vytvoření aplikace Builder vznikl s počátkem bakalářské práce Viktora Holého [6], License Manager (dále LM). LM je komponentou systému Apps Manager. Jedná se o webovou aplikaci, která zajišťuje správu licencí pro různé druhy softwaru.

Webová aplikace Builder, jež je tématem této práce, začala vznikat v roce 2022 v rámci předmětu SP1 skupinou studentů včetně autorky práce, kde proběhla prvotní snaha o vytvoření aplikace. V průběhu celého semestru probíhal sběr požadavků, návrh různých diagramů, z čehož vznikla již zmiňovaná dokumentace, a byla vytvořena první verze prototypu s omezenými funkcionalitami, které nenaplnily všechny požadavky. V průběhu vývoje aplikace byly provedeny menší změny v aplikaci LM pro funkční komunikaci s aplikací Builder. V předmětu SP2 stejná skupina studentů pokračovala na projektu, ale většina z nich se již věnovala jiným prioritám projektu – především automatizaci sestavování konkrétní aplikace. V té době byla autorce poskytnuta příležitost rozvíjet téma aplikace Builder pro bakalářskou práci, jelikož aplikace skrývala různé nedostatky, které bylo nutno více přezkoumat a vyřešit.

1.5 Popis domény

Builder je webová aplikace, jež je součástí vznikajícího systému Apps Manager. Tato komponenta zajišťuje kompletaci, kompilaci a nasazení dané verze aplikace různých platforem. Příkladem je konkrétní webová aplikace, která má svůj vlastní frontend, backend, ale existuje pro ni i verze electron či android. Tato aplikace je pomocí aplikace Builder automaticky sestavena, nasazena na server a zákazníkovi jsou přes LM zprostředkovány odkazy pro stažení aplikace pro různé platformy. Dále se o takovéto sestavované *aplikaci* mluví jako o produktu.

1.5.1 Slovník pojmů

Nejprve je třeba si vymezit slovník pojmů, jenž charakterizuje jednotlivé entity, které Builder obsahuje. Tento slovník je platný pro prvotní prototyp z něhož vychází tato analýza.

Objednávka je vytvořena zákazníkem přes LM, Builder přijímá od LM požadavek na sestavení objednávky. Objednávka obsahuje právě jeden produkt.

Produkt je konkrétní zboží, softwarová aplikace, kterou si chce zákazník pomocí objednávky koupit. Skládá se minimálně z jedné komponenty. V současné době existují čtyři typy komponent: frontend, backend, android a electron.

Komponenta je část produktu. Jedná se například o frontend či backend produktu.

Aplikace je běžící aplikace konkrétního zákazníka.

Soubor je soubor ke stažení, který může sloužit jak pro odkaz ke stažení souboru z cloudu, kde je například verze aplikace ve frameworku Electron či verze aplikace pro operační systém Android nahrána. Jinak slouží jako odkaz na běžící instanci aplikace. Těchto souborů může být uloženo více, protože mohou být uloženy předchozí sestavené verze.

V původním systému byly entity pojmenovány jinak, ale kvůli zmatečnosti došlo ke změně terminologie:

- Produkt byl vždy produktem.
- Komponenta byla v dřívější dokumentaci nazývaná jako aplikace.
- Aplikace byla v dřívější dokumentaci vedena jako Instance aplikace.
- Soubor byl vždy veden jako soubor.

Doménový model má svou verzi v dokumentaci, ta však zahrnuje i část patřící jiné komponentě Apps Manager, LM. Tento doménový model je porovnán s databázovým modelem používaným v prototypu, s nímž nekoresponduje. Stejně tak neodpovídá ani databázovému modelu z dokumentace. Nový doménový model je vytvořen v kapitole návrh v části 2.6.

1.5.2 Zdroje informací kvalitativní metodiky

1.5.2.1 Existující dokumentace

Během vývoje aplikace Builder vznikla v rámci předmětu SP1 při studiu na FIT ČVUT dokumentace, která obsahuje následující položky:

- diagramy aktivit - změna konfigurace, nová instance aplikace, nová objednávka, pravidelné faktury

- doménový model pro LM a Builder
- model požadavků na aplikaci Builder
- model případů užití
- stavový diagram aplikace, faktury, instance aplikace, objednávky
- databázový model

Největší prioritou analýzy je především analýza modelu požadavků. Pro každý požadavek je třeba určit zda byl splněn v již existující implementaci a jeho prioritu.

Dále je nutno zkontrolovat, zda ostatní modely odpovídají implementaci, neboť byly vytvářeny ve fázi návrhu a nemusí korespondovat.

1.5.2.2 Databáze

Aplikace Builder má svou vlastní databázi. Důležitým zdrojem informací je databázový model z prvotního prototypu (A), který lze porovnat s dokumentací.

1.5.2.3 Implementace

Jak již bylo zmíněno, aplikace Builder byla vyvíjena již od roku 2022 skupinou studentů v předmětech SP1 a SP2. Tato implementace obsahuje problémy, které porušují některé ze zásad softwarového inženýrství, čemuž se chci věnovat v části Analýza implementace (1.6).

1.5.3 Komunikace s LM

Builder je webová aplikace, která je přes REST API a HTTP protokol napojena na komponentu LM, od níž na základě uživatelských vstupů přijímá požadavky [6]:

- na sestavení objednávky
- na deaktivaci instance aplikace
- na reaktivaci instance aplikace
- na navýšení konfigurace aplikace
- na smazání instance aplikace

Builder zpětně komunikuje s LM a podává mu zprávu o jeho výsledcích, pro tento druh komunikaci je také využíváno REST API. Před vytvářením této bakalářské práce byl Builder ve stavu, že díky jeho implementaci bylo možné pouze sestavit objednávku. Každý požadavek od LM je analyzován z úhlu pohledu *AS IS*. *AS IS* je stav aplikace v momentální fázi vývoje. Je možné také požadavky analyzovat z pohledu *TO BE*, což je požadovaný stav na konci práce, avšak já tuto část přenechám do kapitoly Návrh, neboť v tu chvíli bude třeba se zaměřit na nové okolnosti vyplývající z návrhu.

1.5.4 Další služby

1.5.5 Komunikace s hostingem

Pro zpřístupnění webové aplikace na dané doméně, případně android aplikace založené na odpovídající webové aplikaci, je třeba, aby aplikace komunikovala s hostingovou službou. Hosting je pronájem webové stránky na cizím serveru [7].

Pro vývoj aplikace Builder byla zadavatelem stanovena doména testovací *ebbe.cz*, se kterou Builder spolupracuje a může si na ní vytvářet domény nižších řádů. Tato doména je od poskytovatele hostingu Hosting90 od *Webglobe* [8].

1.5.6 Sestavení objednávky

Sestavením objednávky se rozumí proces, který připraví produkt pro použití zákazníkem dle podaných informací pro definované platformy. Příkladem takovéto objednávky může být objednání webové aplikace Sklady, která se skládá z frontendu a backendu a je dostupná pro webový prohlížeč tak i jako aplikace pro android. Po sestavení objednávky bude k dispozici odkaz na běžící webovou aplikaci a odkaz ke stažení aplikace Android.

Adresa požadavku: *https://adresa-builderu/api/build*

Builder po přijetí požadavku na jeho adresu sestavuje požadovaný produkt pro zákazníka, vytvoří specifikované instance aplikací a předá je zpět LM. Builder obsahuje informace ke každému produktu, který je obsažen v LM. Tento požadavek nebyl specifikován v technické rovině a jednalo se pokaždé pouze o manuální kontrolu.

Builder dostane od LM data ve formátu JSON. Jedná se o vstupní informace, které jsou vyžadovány ve formátu *objednávka* (1.1):

Výpis kódu 1.1 Tělo požadavku pro sestavení objednávky

```
{
    "order": {
        "id": ID,
        "configuration": {
            "product": {
                "builder_id": BUILDER_PRODUCT_ID
            }
        },
        "billing_information": {
            "name": CUSTOMER_NAME
        }
    }
}
```

ID je identifikátor objednávky, jež byla vytvořena v LM. Tento identifikátor si Builder ukládá pro pozdější použití, jímž může být například požadavek na reaktivaci objednávky právě pomocí tohoto identifikátoru přijatého od LM. `BUILDER_PRODUCT_ID` je identifikátor produktu, který má ve své databázi uložen Builder. `CUSTOMER_NAME` je jméno zákazníka. Jméno zákazníka je využito pro tvorbu DNS záznamu, kde je instance aplikace po sestavení nasazena.

Proces sestavení objednávky probíhá následovně. Nejprve je v databázi vytvořena položka objednávky a následně se postupně sestaví všechny aplikace daného produktu. Pro webovou aplikaci je registrován nový DNS záznam na hostingu (v této implementaci využit od firmy Hosting90). Název DNS záznamu se odvíjí z názvu produktu a jména zákazníka, jak již bylo zmíněno. Samotné sestavení aplikace je v podstatě vytvoření nové instance aplikace a pak také aktivování vybrané Gitlab CI/CD pipeline.

Pro aktivování pipeline je v kódu knihovna Node.js *FormData*. Tato knihovna umožňuje zkonstruovat množinu klíčů s jejich hodnotami. Využívá se pro zasílání http požadavků [9].

V této implementaci je tato knihovna ve spojením s *Axios*, což je promise-based http klient pro Node.js a prohlížeč. Pro svoje fungování využívá *XMLHttpRequests* [10], které poskytuje právě knihovna FormData.

Po dokončení sestavování jsou na poskytované API LM odeslány informace o sestavených aplikacích. Ve formátu *objednávka* a k tomu ještě pole *url* (1.2).

Výpis kódu 1.2 Tělo požadavku po sestavení objednávky


```
{
    "urls": [
    \{
      "type": "android",
      "url": "gitlab.jagu.com"
    \},
    \{
      "type": "web",
      "url": "gitlab.com"
    \}
  ]
}
```


Výše definovaný stav je již implementovaný. Co se týká požadovaného stavu, od doby vývoje neproběhla žádná změna v požadavcích pro sestavování aplikace.

1.5.7 Deaktivace produktu zákazníka

Deaktivace produktu zákazníka nejčastěji nastává ve chvíli, kdy zákazník LM nezaplatil za svůj produkt a od komponenty LM přijde požadavek na deaktivaci konkrétní licence. Po deaktivaci není zákazníkovi produkt již přístupný, je možné jej však obnovit.

Adresa požadavku: *https://adresa-builderu/api/deactivate/order-id*

Builder dostane od LM požadavek na deaktivaci konkrétní objednávky dle jejího číselného identifikátoru. Tento identifikátor má Builder od vyřízení objednávky uložen ve své databázi a může proto snadno identifikovat, o které instance aplikací se jedná. Zpětnou vazbou pro LM by měl po deaktivaci objednávky potvrdit, že deaktivace proběhla úspěšně.

Požadavek na deaktivaci není v současné době stále ještě implementován a je pro něj potřeba vytvořit návrh funkčnosti, který bude v souladu s fungováním reaktivace a zrušení.

1.5.8 Reaktivace a zrušení produktu zákazníka

Reaktivace produktu zákazníka je proces, který probíhá na žádost zákazníka poté, co zákazník zaplatí požadovanou částku, kterou nesplácel. Reaktivace je možná do předem definované lhůty.

Po vypršení této lhůty dojde ke zrušení produktu zákazníka automaticky uvnitř aplikace Builder a veškerá data jsou smazána a produkt zákazníka již není možné obnovit. Druhou variantou je přímý požadavek na zrušení od administrátora LM.

Adresa požadavku reaktivace: *https://adresa-builderu/api/reactivate/order-id*

Adresa požadavku zrušení: *https://adresa-builderu/api/cancel/order-id*

Stejně jako u deaktivace produktu zákazníka obdrží Builder při požadavku na reaktivaci číselný identifikátor objednávky, který určí, jaké instance aplikace se mají obnovit. Reaktivace je však možná pouze do určeného času od deaktivace. Po uplynutí této časové lhůty dojde k celkovému zrušení objednávky, jejímu vymazání z databáze a smazání veškerých dat zákazníka, která se vážou ke konkrétní objednávce.

Reaktivace ani zrušení produktu zákazníka rovněž nebyly v rámci prvotní verze prototypu na-implementovány.

1.5.9 Navýšení konfigurace produktu zákazníka

Navýšení konfigurace produktu zákazníka nastává na žádost zákazníka, požadavek tedy přichází z LM. Navýšení konfigurace produktu zákazníka spočívá v tom, že zákazník potřebuje software pro více poboček a je tedy potřeba změnit omezení pro dostupný počet poboček.

Adresa požadavku: *https://adresa-builderu/api/upgrade/order-id*

Tento požadavek zatím nebyl řešen.

1.6 Analýza implementace

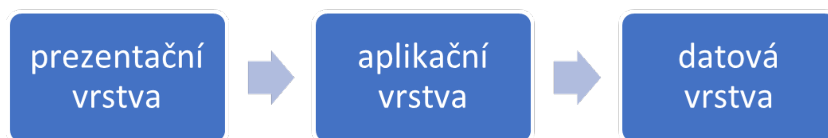
1.6.1 Zvolené technologie pro první prototyp

Pro implementaci celé webové aplikace byl vybrán systém Node.js. Node.js je softwarový systém pro psaní vysoce škálovatelných webových aplikací. Jeho programy jsou psané v programovacím jazyku JavaScript. [11] Webová aplikace Builder má také svou databázi. Pro ni byla zvolena varianta PostgreSQL, což je open-source objektově-relační databáze. [12]

Pro svou funkci využívá Builder Gitlab CI/CD. Jedná se o nástroj, který využívá následující metodiky: CI (*Continous Integrations*), CD (*Continous Deployment*), CD (*Continous Delivery*). Tento nástroj je využíván především pro CD. To znamená, že po integraci kódu je aplikace automaticky sestavena, otestována (v případě prvotního prototypu aplikace Builder testy nebyly implementovány) a nasazena. Většina zdrojových kódů aplikací je uložena právě v systému Gitlab firmy Jagu s.r.o. a díky využívání Gitlab CI/CD dává možnost vyžadovaného automatického nasazování. [13]

1.6.1.1 Architektura

Builder je postaven do třívrstvé architektury. Třívrstvá architektura se vyznačuje 3 lineárně postavenými částmi dle schématu 1.1. Prezentační vrstva slouží pro konkrétní interakci s uživatelem, aplikační vrstva (byznys, doménová) představuje jádro aplikace a datová vrstva (perzistentní) provádí veškerou práci s daty a tedy komunikaci s databází. [14]



Obrázek 1.1 Třívrstvá architektura

1.6.1.2 Programovací paradigma

Programovací paradigma je set konceptů, který jazyk dává [15]. Kromě objektového programovacího paradigma známe také funkcionální či paralelní a mnoho dalších. Jak je poznat již z architektury aplikace Builder, bylo zvoleno objektově orientované programovací paradigma. Objektově orientované programování se vyznačuje 4 pilíři: enkapsulací, abstrakcí, dědičností a polymorfismem. [15], [16]

Při analýze implementace lze jednoduše poznat, že mnoho tříd (objektů) porušuje jeden z principů objektově orientovaného programování: Single Responsibility Principle [17]. Tento princip udává, že každý objekt by měl mít jednu odpovědnost. Jeho nedodržení způsobuje těžkou testovatelnost kódu.

1.6.2 Databáze

Pro prvotní verzi prototypu Builder byl vytvořen databázový model v dokumentaci. Po bližším prozkoumání bylo však zjištěno, že je již neplatný a v reálném použití je model dle schématu v příloze A.

1.6.3 Požadavky

1.6.3.1 Kategorizace požadavků

Pro analýzu požadavků je využívána kategorizace FURPS a MoSCow. Kategorizace FURPS pro každý z požadavků uvádí, pro kterou z následujících kategorií se používá [18]:

F jako *functionality* (funkčnost) Tato kategorie odpovídá hlavně funkčnosti aplikace, tedy co software umí, jaké jsou jeho funkcionality.

U jako *usability* (použitelnost) Kategorie se dívá na požadavek z pohledu uživatele, jak se software dobře používá, jestli je intuitivní a srozumitelný.

R jako reliability (spolehlivost) Kategorizuje požadavky, které specifikující dostupnost systému, správnost výstupu a zotavování z chyb, či přesnost zpracování vstupů a výstupů.

P jako *performance* (výkon) Jsou to požadavky, jenž říkají, jak rychle musí systém reagovat, kolik požadavků zvládne zpracovávat najednou apod.

S jako *supportability* (podporovatelnost, rozšířitelnost) Tyto požadavky specifikují, jak dlouho je software dále použitelný, jak dlouho je poskytována údržba a jak moc se dokáže přizpůsobit.

Druhou kategorizací je metoda MoSCow, která slouží pro určení především priority jednotlivých požadavků. [19], [1]

M - *Must have* Jedná se o požadavky s nejvyšší prioritou, musí být splněny na konci projektu.

S - *Should have* Důležité požadavky, které mají určitý přínos, ale nejsou nezbytné.

C - *Could have* Požadavky, které by bylo dobré splnit, ale jedná se spíše o požadavky, které se splní po dokončení prvotní fázi nebo pokud ke konci projektu zbude čas.

W - *Won't have* Požadavky, jež vůbec nejsou prioritou a není potřeba je splnit.

1.6.3.2 Specifikace požadavků

V této části jsou vyjmenovány vyjmenovány aktuální známé požadavky na systém. Požadavky jsou založeny především na požadavcích, které byly specifikovány v již zmiňované dokumentaci. U každého požadavku je poznamenáno, zda je již požadavek splněn v implementaci a zároveň jsou požadavky seřazeny dle priority dle metody MoSCoW. Požadavky jsou dále dle metody FURPS děleny na požadavky funkční, což představuje kategorii F a nefunkční, kde je blíže specifikováno, o který další podtyp z kategorií U, R, P, S se jedná.

1.6.3.3 Funkční požadavky

F1 – Sestavení produktu zákazníka
Builder sestaví objednaný produkt pro daného zákazníka. Sestavení se provede na základě přijatých informací od volajícího požadavku. Builder informace zpracuje a podle nich vytvoří pro zákazníka produkt.
Implementováno: Ano
Priorita: *Must have*

F2 – Uchovávání sestavených aplikací
Builder uchovává informace o jím sestavených aplikacích, dokud nedojde ke zrušení licence.
Implementováno: Ano
Priorita: *Must have*

F3 – Ovládaní Builderu
Builder je ovládán pomocí požadavků z komponenty Apps Manager. Správce aplikace Builder má možnost přes komponenty systému Apps Manager ručně spravovat běžící aplikace, sledovat záznamy běžících produktů a jejich výpisy.
Implementováno: Ano
Priorita: *Must have*

F4 – Komunikace s Gitlabem
Nabízené produkty, které si zákazník může objednat, jsou umístěny na Gitlabu. Builder dokáže s Gitlabem komunikovat. Žádanou komponentu z Gitlabu získá, aby s ní mohl začít pracovat a sestavovat ji pro produkt.
Implementováno: Ano
Priorita: *Must have*

F5 - Instalační soubory přes cloud
Pokud zakoupený produkt nabízí i možnost instalace přímo na zařízení zákazníka, Builder vytvoří instalační soubory pro nabízené operační systémy a tyto instalační soubory zpřístupní zákazníkovi přes cloud. Builder jako odpověď k sestavení požadavku předá nazpět k těmto souborům odkazy, přes které jsou instalační soubory dostupné pro zákazníka.
Implementováno: Ano
Priorita: *Must have*

F6 – Odstavení produktu zákazníka
Builder dokáže zamezit zákazníkovi v používání jeho zakoupeného produktu na základě požadavku od jiné komponenty Apps Manageru.
Implementováno: Ne
Priorita: *Should have*

F7 – Obnovení přístupu k produktu zákazníka
Builder dokáže zákazníkovi deaktivovaný produkt znovu zpřístupnit přes původní přístupová rozhraní na základě požadavku od jiné komponenty Apps Manageru.
Implementováno: Ne
Priorita: *Should have*

F8 – Trvalé zrušení licence
Builder smaže celý produkt zákazníka společně se všemi jeho daty. Toto smazání je již ne nenávratné.
Implementováno: Ne
Priorita: *Should have*

F9 – Navýšení konfigurace produktu
Builder dokáže vylepšit konfiguraci běžící dříve zakoupené aplikace. Konfigurací produktu se rozumí například, že produkt byl zakoupen pro užívání na 2 pobočkách, pro více poboček by neměl být používán. Procesem vylepšení zákazník nepřijde o žádná data a je zachována i původní doména v případě, že produkt běží jako webová aplikace.
Implementováno: Ne
Priorita: *Should have*

F10 - Omezení aplikace
"V případě, že zákazník nedodržuje limity zakoupené konfigurace, dokáže Builder omezit používání aplikace tak, aby zákazník skutečně používal aplikaci pouze v rámci zakoupené konfigurace. Při tomto procesu může dojít ke smazání dat. Builder data před smazáním zazálohuje, aby se s zákazníkem později mohla osobně řešit jejich případná obnova."
Implementováno: Ne
Priorita: *Could have*

F11 – Kontrola dodržování konfigurací
Builder sleduje zakoupené produkty a vyhodnocuje u nich dodržování zvolené konfigurace. Neoprávněné navyšování konfigurace bude zaznamenávat a v případě opakování nebo dlouhotrvajícího porušování upozorní zákazníka i správce, aby spolu začali pracovat na vyřešení. Správce většinou nabídne zákazníkovi upgrade na vyšší konfiguraci nebo omezení neoprávněného používání.
Implementováno: Ne
Priorita: *Could have*

F12 – Webové aplikace a domény
V případě, že zakoupený produkt podporuje provoz jako webová aplikace, Builder takovou aplikaci vytvoří a umístí na danou doménu. Zákazník si při objednávce zvolí, jestli mu stačí doména od provozovatele Builderu, nebo jestli chce vlastní. První způsob Builder provádí automaticky. Pokud si zákazník v LM vybere druhý způsob, vlastní doména, Builder to pozná dle těla požadavku pro sestavení objednávky a danou doménu si uloží.
Implementováno: Ne
Priorita: *Could have*

F13 – Vizuální zrušení licence
Pokud přijde požadavek na zrušení licence, dojde k tzv. vizuálnímu zrušení licence. Builder deaktivuje produkt zákazníka (jestliže ještě není deaktivovaný) a odpoví, že se jedná o zrušenou licenci. Zároveň spustí hodiny, které po vypršení dané lhůty zavolají tzv. Trvalé zrušení licence. Data produktu nejsou smazána a je možnost si je před vypršením té samé lhůty obnovit. Jestliže je licence obnovena (na základě požadavku od komponenty Apps Manageru), zruší se časovač na trvalé zrušení licence, které pak už nikdy neproběhne.
Implementováno: Ne
Priorita: *Could have*

F14 – Výpis aplikací
Builder poskytne seznam všech běžících i pozastavených produktů včetně informací o nich. Součástí budou i produkty, které jsou ve stavu vizuálně zrušené a je možné je za poplatek obnovit.
Implementováno: Ne
Priorita: *Could have*

F15 – Komunikace s hostingem
Builder zprostředkovává komunikaci s hostingem pro správu dns záznamů. Při změně poskytovatele hostingu jiné komponenty mohou posílat stále stejné požadavky, mění se pouze implementace Builderu.
Implementováno: Ne
Priorita: *Could have*

1.6.3.4 Nefunkční požadavky

N1 – Opravení chyb v implementaci
Požadavek vyplývá z analýzy implementace. Je třeba opravit nedostatečnost implementace z hlediska udržitelnosti a připravit kód pro testování. Typ: S
Priorita: *Must have*

N2 – Zabezpečení
Ovládání aplikace Builder je zabezpečeno, aby nebylo možné provádět požadavky neoprávněným osobám a aplikacím.
Typ: R
Implementováno: Ano
Priorita: *Must have*

N3 – Počet souběžných požadavků
Builder dokáže v jeden okamžik zpracovávat pouze jediný požadavek. Dokáže ale přijmout více požadavků najednou a tyto požadavky pak zpracovat postupně.
Typ: P
Priorita: *Should have*

1.7 Výstup kapitoly

Hlavním výstupem této kapitoly je seznam aktuálních požadavků na aplikaci z předešlé části. Následující kapitola bere v potaz tyto požadavky a dle nich vytváří návrh aplikace. Dílčím výstupem je také popis předešlého stavu před návrhem a implementací nového prototypu.

Kapitola 2

Návrh

Před implementací aplikace je nutné přesněji určit, jak bude aplikace fungovat dle aktuálních požadavků. Tyto požadavky je nejprve třeba zadefinovat, jak již vyplývá z předchozí kapitoly a následně navrhnout způsob jejich řešení. Tomu se věnuje tato kapitola, která probírá návrh aplikace Builder.

2.1 Komponenty

V předešlé kapitole byla analyzována komunikace s komponentou LM. Během vzniku tématu této bakalářské práce se však během konzultací objevila další část celého systému, jež je třeba připravit. Této komponentě byl dán název Deployment Manager (dále zkráceně DM) a ve své bakalářské práci na ni pracuje Adam Staš. Tato komponenta umožňuje administrátorovi správu serverů, na kterých jsou aplikací Builder nasazené sestavené aplikace.

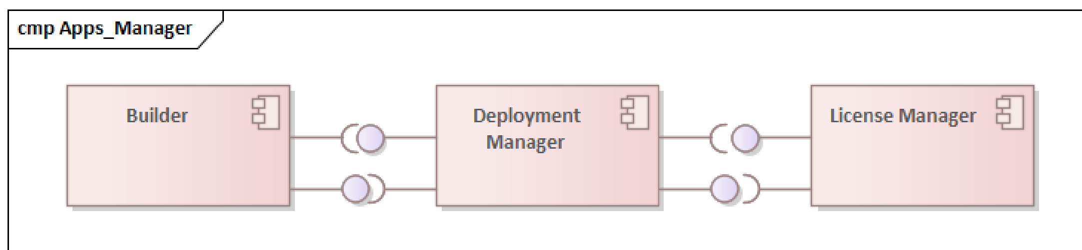
Díky přidání této nové komponenty bylo třeba rozmyslet návrh komunikace všech tří komponent v celém systému. Během společných konzultací s Adamem Stašem a Tomášem Sládkem, který se ve své bakalářské práci zabývá komponentou LM, bylo nejprve dohodnuto, že kvůli možným nejasnostem či úniku informací nebudou zaváděny cyklické závislosti a bude udržena pouze lineární závislost komponent.

Následně byl řešen další problém, pořadí závislostí. Prvotní návrh představoval pouze napojení DM za Builder, tzn. že by systém fungoval v následujícím schématu: LM by komunikoval pouze s aplikací Builder a Builder by komunikoval s LM i s DM a zprostředkovával by komunikaci mezi nimi. Tento návrh byl zamítnut po konzultaci s vedoucími prací, neboť, jak už ze samotného názvu aplikace Builder vypovídá, Builder má především sestavovat produkty a ne se zabývat tím, kde která verze běží. Proto bylo zvoleno jiné pořadí schématu komunikace: LM předává své požadavky na DM, který doplní požadavky o další potřebné informace - především verzi - a pošle svůj vlastní požadavek aplikaci Builder.

Pro přehlednost byl vytvořen diagram komponent (2.1).

2.2 Úprava požadavků

V předchozí kapitole byly specifikovány funkční a nefunkční požadavky na aplikaci Builder. Díky vzniku nové komponenty je třeba požadavky upravit tak, že místo přijímání požadavku od komponenty LM jsou požadavky přijímány od komponenty DM. V textu požadavků se tato specifikace neobjevila, ale je třeba tuto poznámku neopomenout při implementaci a návrhu komunikace.



Obrázek 2.1 Diagram komponent

2.3 Případy užití

Ve většině případů jsou modely případů užití součástí analýzy, nicméně díky vzniku nové komponenty již v době analýzy bylo rozhodnuto, že tato část bude přesunuta až do části návrhu.

Modelování případů užití konkretizuje funkční požadavky do scénářů, ve kterých jsou naplněny.

Tyto diagramy obsahují aktéry, kterými jsou v tomto případě LM a Čas. Dalo by se polemizovat, zda za aktéra nezvolit přímo zákazníka, ale on využívá komponentu LM ke zprostředkování a tedy se nejedná o aktéra.

Tabulka 2.1 zobrazuje pokrytí funkčních požadavků konkrétními případy užití, které jsou popsány dále v této kapitole společně s jejich scénáři.

Tabulka 2.1 Pokrytí funkčních požadavků případy užití 1

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15
UC1	X			X	X							X			X
UC2									X	X	X				
UC3						X									
UC4								X					X		
UC5							X								
UC6		X	X											X	
UC7			X												
UC8			X												
UC9			X												
UC10			X												
UC11			X												
UC12			X												
UC13			X												
UC14			X												
UC15			X												
UC16			X												
UC17			X												

2.3.1 UC1 – Sestavení licence

Od komponenty DM přijde požadavek na sestavení daného produktu s konkrétní konfigurací. Builder produkt pro zákazníka sestaví a předá jej zpět komponentě DM.
Aktér: DM

Hlavní scénář: Sestavení webové aplikace

1. Přes API přijde požadavek na sestavení produktu pro konkrétního zákazníka.
2. Builder zjistí, o jaký produkt se jedná, z informací dodaných DM dále zjistí, jaké verze komponent jsou potřeba sestavit.
3. Builder kontaktuje Gitlab pro sestavení jednotlivých komponent do aplikací.
4. Pokud se jedná o webovou aplikaci, je zaregistrována doména dle požadavku, který přišel od zákazníka. Pokud není doména specifikována, je registrována doména přidělená aplikací Builder dle jména zákazníka.
5. Webová aplikace je nasazena na server.
6. Je vytvořen DNS záznam propojující doménu a adresu serveru, kde je webová aplikace nasazena.
7. Komponentě DM je spolu s dalšími údaji o aplikacích předána nazpět URL na běžící webovou aplikaci.

Alternativní scénář: Sestavení aplikace android nebo electron
Scénář začíná ve 4. kroku hlavního scénáře, pokud se nejedná o webovou aplikaci, kterou je třeba sestavit.

1. Pokud se jedná o aplikaci android nebo electron patřící k webové aplikaci, je vytvořena nejprve webová aplikace.
2. Po sestavení webové aplikace je sestavena aplikace android (electron).
3. Aplikace android (electron) je uložena na cloudu.
4. Komponentě DM je spolu s dalšími údaji o aplikacích předána nazpět URL pro stažení funkční aplikace android (electron).

2.3.2 UC2 – Upgrade licence

Případ užití umožňuje upgrade licence, což znamená zvýšení počtu poboček na kterých může být produkt používán. Od komponenty DM přijde požadavek na licence s daným identifikátorem. Builder provede upgrade u potřebných aplikací, které patří k dané licenci a informuje o tom nazpět DM, případně informuje o problémech, které se vyskytly.
Aktér: DM
Podmínky pro spuštění: Existuje aktivní licence s daným identifikátorem.

2.3.3 UC3 – Deaktivace licence

Případ užití umožňuje deaktivovat licenci a zamezit tak zákazníkovi přístup k jeho zakoupenému produktu.
Aktér: DM
Podmínky pro spuštění: Existuje aktivní licence s daným identifikátorem.
Hlavní scénář

1. Builder přijme požadavek přes API na deaktivace licence s daným identifikátorem
2. Builder licenci deaktivuje.

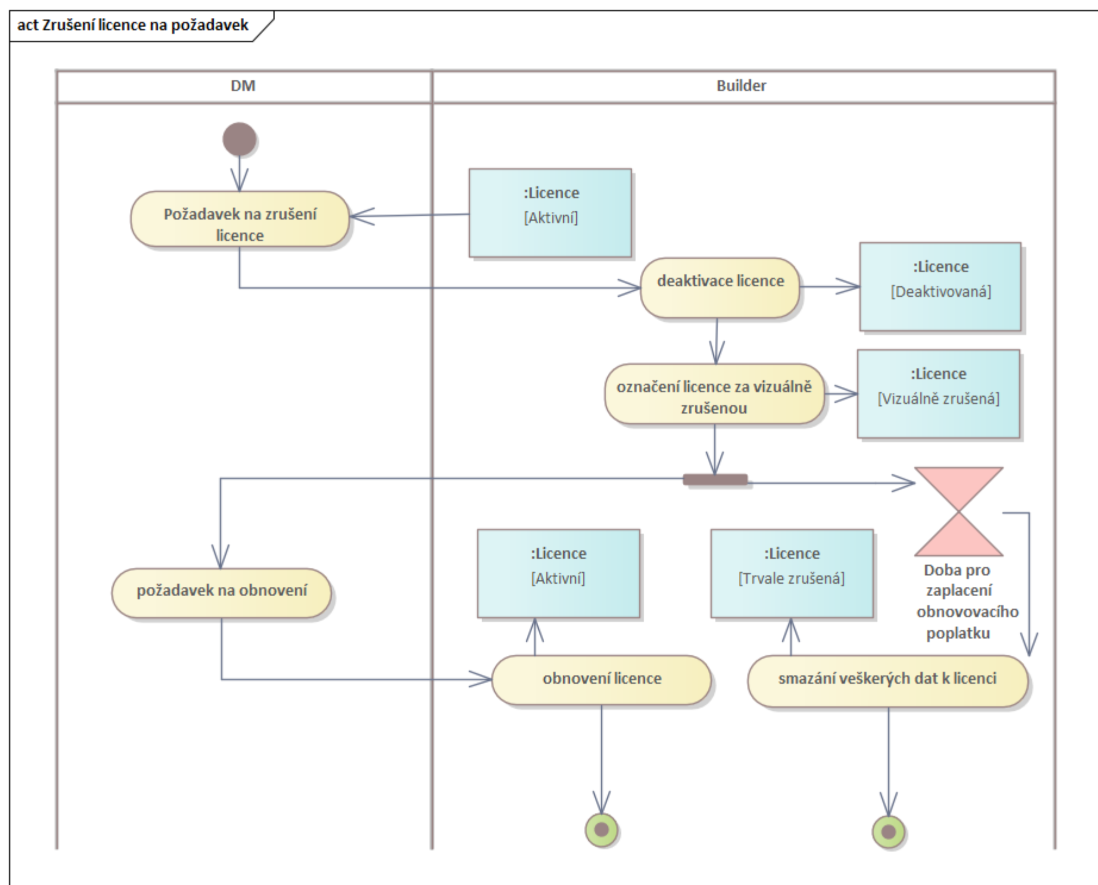
3. Builder zařídí, aby byl zrušen přístup zákazníka přes jeho doménu k webové aplikaci a dalším aplikacím.
4. Builder informuje DM o úspěšné deaktivaci licence.

2.3.4 UC4 – Zrušení licence

Případ užití umožňuje zrušit licenci, takže zákazník si zdánlivě před vypršením dané lhůty myslí, že již není možné produkt vůbec obnovit. Reálně se jedná ale stále o deaktivovaný produkt, ale po vypršení dané lhůty je produkt smazán ze serverů a není možné jej již nikdy obnovit.

Pro zrušení licence jsou možné dva různé scénáře, jelikož se jedná o složitější scénáře, byly vytvořen diagramy aktivit.

První scénář nastává ve chvíli, kdy se administrátor LM/DM rozhodne, že s daným zákazníkem nemá dobrou zkušenost a pošle požadavek na zrušení licence (2.2).

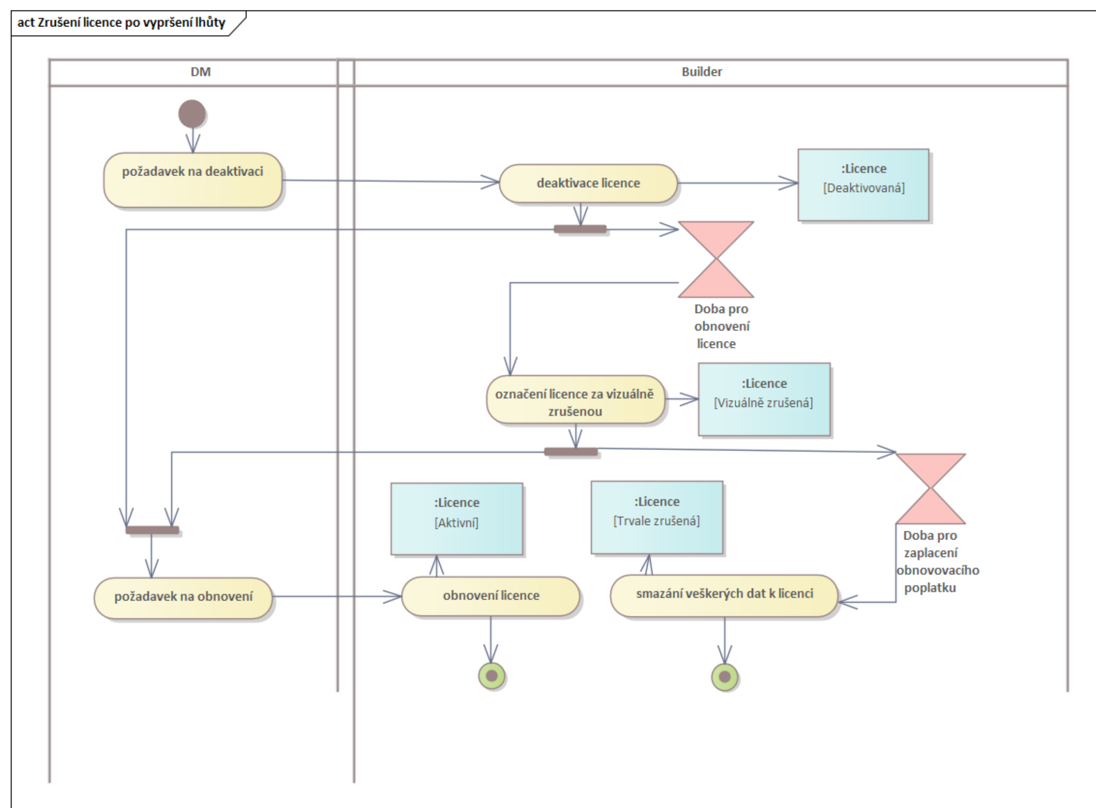


Obrázek 2.2 Zrušení licence na požadavek

Druhým případem je když je produkt danou dobu deaktivovaný a zrušení produktu probíhá interně pouze v aplikaci Builder (2.3).

2.3.5 UC5 – Obnovení licence

Obnovení licence úzce souvisí se zrušením licence, neboť obnovení licence je možné právě ve chvíli, kdy je licence deaktivovaná nebo vizuálně zrušená. Případ užití umožňuje DM obnovit



Obrázek 2.3 Zrušení licence po vypršení lhůty

přístup k danému produktu zákazníka.

2.3.6 UC6 – Zobrazení všech aplikací

Případ užití umožňuje zobrazení všech aplikací pro získání přehledu o běžících komponentách pro přehled.
Aktér: DM

Hlavní scénář

1. Přes API je z DM odeslán požadavek na zobrazení všech aplikací.
2. Builder ze své databáze vyčte seznam všech aplikací.
3. Builder odešle v odpovědi na požadavek seznam všech aplikací.

2.3.7 UC7 – Přidání produktu

Případ užití umožňuje vytvoření nového produktu v aplikaci Builder.
Aktér: DM

Hlavní scénář

1. Přes API je z DM odeslán požadavek na přidání produktu se všemi potřebnými informacemi.

2. Do aplikace Builder je přidán produkt.

3. Builder v odpovědi na požadavek informuje o úspěchu.

2.3.8 UC8 – Úprava produktu

Případ užití umožňuje upravit konkrétní produkt v případě změny jeho informací.
Aktér: DM

Hlavní scénář

1. Přes API je z DM odeslán požadavek na úpravu produktu se všemi potřebnými informacemi.

2. V aplikaci Builder je produkt upraven.

3. Builder v odpovědi na požadavek informuje o úspěchu.

2.3.9 UC9 – Zobrazení produktu

Případ užití umožňuje zobrazit si informace o produktu uložené v databázi aplikace Builder.
Aktér: DM

Hlavní scénář

1. Přes API je z DM odeslán požadavek na zobrazení produktu se všemi potřebnými informacemi.

2. Builder v odpovědi na požadavek odešle informace o konkrétním produktu.

2.3.10 UC10 – Zobrazení všech produktů

Případ užití umožňuje zobrazit si informace o všech produktech uložených v databázi aplikace Builder.
Aktér: DM

Hlavní scénář

1. Přes API je z DM odeslán požadavek na zobrazení všech produktů.

2. Builder v odpovědi na požadavek odešle informace o všech svých produktech.

2.3.11 UC11 – Smazání produktu

Případ užití umožňuje vymazat konkrétní produkt uložený v databázi aplikace Builder.
Podmínka před použitím: Produkt nesmí mít vazby na žádné licence. **Aktér:** DM

Hlavní scénář

1. Přes API je z DM odeslán požadavek na smazání produktu se všemi potřebnými informacemi.

2. Builder v odpovědi na požadavek odešle informace o úspěchu či selhání.

2.3.12 UC12 – Přidání komponenty

Případ užití umožňuje vytvoření nového produktu v aplikaci Builder.
Aktér: DM

Hlavní scénář

1. Přes API je z DM odeslán požadavek na přidání komponenty se všemi potřebnými informacemi.
2. Do aplikace Builder je přidána komponenta.
3. Builder v odpovědi na požadavek informuje o úspěchu.

2.3.13 UC13 – Úprava komponenty

Případ užití umožňuje upravit konkrétní komponentu v případě změny jejích informací.
Aktér: DM

Hlavní scénář

1. Přes API je z DM odeslán požadavek na úpravu komponenty se všemi potřebnými informacemi.
2. V aplikaci Builder je komponenta upravena.
3. Builder v odpovědi na požadavek informuje o úspěchu.

2.3.14 UC14 – Zobrazení komponenty

Případ užití umožňuje zobrazit si informace o komponentě uložené v databázi aplikace Builder.
Aktér: DM

Hlavní scénář

1. Přes API je z DM odeslán požadavek na zobrazení komponenty se všemi potřebnými informacemi.
2. Builder v odpovědi na požadavek odešle informace o konkrétní komponentě.

2.3.15 UC15 – Zobrazení všech komponent

Případ užití umožňuje zobrazit si informace o všech komponentách uložených v databázi aplikace Builder.
Aktér: DM

Hlavní scénář

1. Přes API je z DM odeslán požadavek na zobrazení všech komponent.
2. Builder v odpovědi na požadavek odešle informace o všech svých komponentách.

2.3.16 UC16 – Smazání komponenty

Případ užití umožňuje vymazat konkrétní komponentu uloženou v databázi aplikace Builder. **Podmínka před použitím:** Komponenta nesmí mít vazby na žádné aplikace. **Aktér:** DM

Hlavní scénář

1. Přes API je z DM odeslán požadavek na smazání komponenty se všemi potřebnými informacemi.

2. Builder v odpovědi na požadavek odešle informace o úspěchu či selhání.

2.3.17 UC17 – Přiřazení produktu komponentě

Případ užití umožňuje přiřazení produktu komponentě při přidávání nových produktů či komponent.
Aktér: DM

Hlavní scénář

1. Přes API je z DM odeslán požadavek na přiřazení komponenty k danému produktu.

2. Builder v odpovědi na požadavek odešle informace o úspěchu či selhání.

2.4 Komunikace s dalšími komponentami

2.4.1 API

Jako nejjednodušší a nejintuitivnější implementace komunikace aplikace Builder s DM se jeví dnes široce rozšířené a populární REST API. REST je architektura rozhraní navržená pro distribuované prostředí. REST je spojen s HTTP a je datově orientovaný a implementuje CRUD, což je zkratka pro 4 základní operace: *Create* (vytvoření informace), *Read* (čtení informace), *Update* (změna informace) a *Delete* (smazání informace). [20]

Toto API bylo navrhováno ve spolupráci s Adamem Stašem, protože bylo třeba se domluvit, jak spolu komponenty systému budou komunikovat. Vzhledem k tomu, že se spolupráce rozcházela, nebylo možné sjednotit návrhy. V určité chvíli byla komunikace pozastavena a implementována dohodnutá část, vzhledem k časovým možnostem. Je tedy možné, že komponenty Builder a DM nebudou na sebe přímo navazovat přes vytvořené API a bude nutné tuto změnu provést na základě další domluvy.

Pro co nejpřesnější návrh API byl využit nástroj SwaggerHub. Tento nástroj umožňuje s použití vlastní syntaxe definovat pro API jednotlivé koncové body a parametry, koncové body a další. SwaggerHub poskytuje jako součást tohoto návrhu také grafické rozhraní, které poskytuje vizuální zobrazení navrženého API. [21]

Toto API je zachyceno v dokumentaci v přiloženém archivu.

2.5 Komunikace s dalšími službami

2.5.1 Komunikace s hostingem

Co je to hosting bylo vysvětleno již v kapitole návrh v části Komunikace s hostingem 1.5.5, stejně jako to, že je využíván hosting od Hosting90 na doméně *ebbe.cz*. Tyto informace se od počátku vývoje aplikace nezměnily.

Komunikace s hostingem je praktikována na základě existující dokumentace hostingu [22]. Pro komunikaci s hostingem je vytvořena v aplikaci Builder samostatná třída, jež spravuje veškeré činnosti spojené s udržováním DNS záznamů.

Pro komunikaci byl implementovány metody zprostředkující konkrétně: přihlášení k hostingu, zaregistrování DNS záznamu, získání DNS záznamu s daným jménem a pro další vývoj byla připravena i metoda pro deaktivaci DNS záznamu, kdy je DNS záznam přesměrován na deaktivační stránku aplikace Builder.

2.5.2 Komunikace s GitLabem

Pro sestavování objednávky je třeba sestavit jednotlivé komponenty, které patří objednávanému produktu. Když od DM přijde požadavek na sestavení objednávky, přijdou v něm zároveň data pro sestavování jednotlivých komponent, tzv. tagy. Také přijdou data, která slouží jako jednoznačné identifikátory pro konkrétní aplikace.

Při sestavování jednotlivých komponent komunikuje Builder s GitLabem. Posílá GitLabu data pro sestavení potřebné komponenty a to dle typu komponenty.

Pro webové aplikace zasílá webovou url, port přes který aplikace běží, jednoznačný identifikátor slug. Dále také zasílá jméno zákazníka, port a název webu, na kterém aplikace poběží.

Pro android aplikace zasílá url, na které běží webová aplikace, ikonu aplikace, kulatou ikonu aplikace, adresu autorizačního serveru, url příponu a výslednou hodnotu url adresy a následně také příponu, která je udávána za verzí aplikace.

Pro elektron aplikace je zasílána url, na které běží webová aplikace, jednoznačný identifikátor – slug – aplikace a jméno zákazníka.

Tato část nebyla v rámci vytváření prototypu příliš analyzována, ani nebyly navrhovány možná vylepšení, neboť proces sestavování objednávky je ponechán především přímo na GitLab pipeline. Je tedy nutné pouze definovat, které parametry jsou nutné pro sestavování jednotlivých typů komponent, což již bylo implementováno v prvotním prototypu.

2.6 Návrh domény

V analytické části byl zobrazen databázový model aplikace, který byl využíván pro prvotní prototyp aplikace Builder. Tento databázový model je nutné upravit tak, aby korespondoval s právě navrženým API v dokumentaci Swagger. Výsledkem této podkapitoly je datový model v notaci UML - konkrétně diagramu tříd. Tento diagram slouží k objektově orientovanému modelování dat, popisuje strukturu systému zobrazením tříd a jejich atributů a především vztahů mezi jednotlivými třídami.

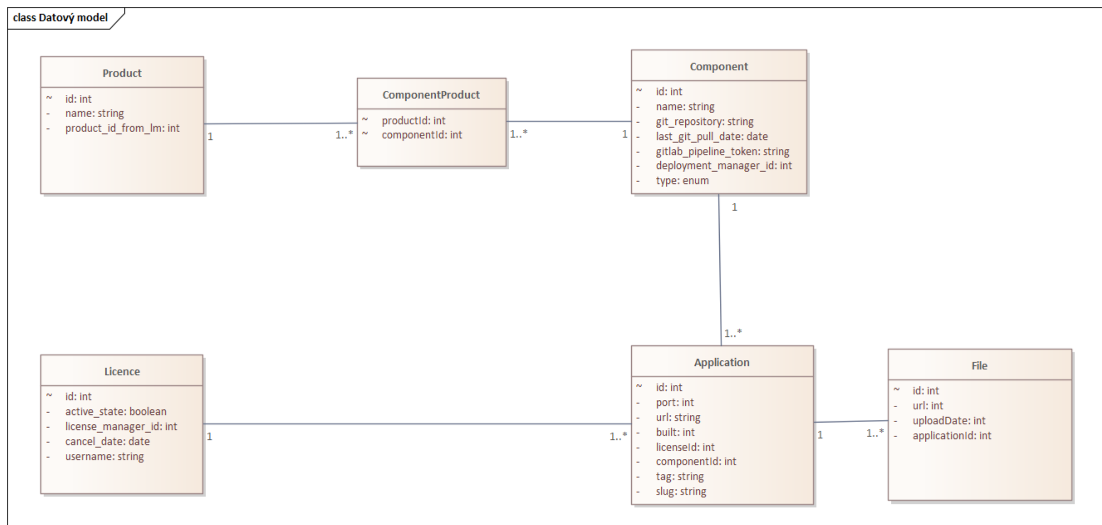
Na základě návrhu API a prvotního prototypu aplikace byl vytvořen nový, aktualizovaný doménový model obsahující všechna potřebná data (2.4).

2.7 Licence

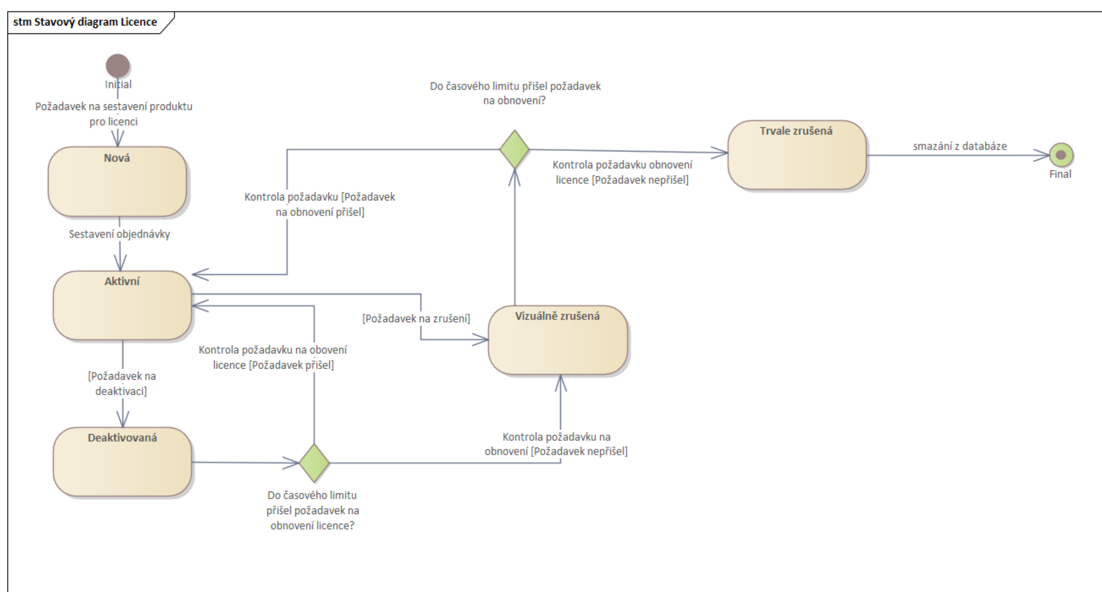
V předchozí kapitole byl zmíněn stavový diagram pro aplikaci. Tento stavový diagram nicméně již nebyl aktuální dle definovaných požadavků. Navíc je lepší vzít to z vyššího levelu abstrakce, protože lze říci, že se jedná o stavový diagram celé objednávky, tedy licence.

2.8 Architektura a porovnání s předešlou architekturou

Pro vytvoření nového prototypu aplikace byla zvolena architektura MVC (*Model View Controller*). Tato architektura funguje dle schématu 2.6.



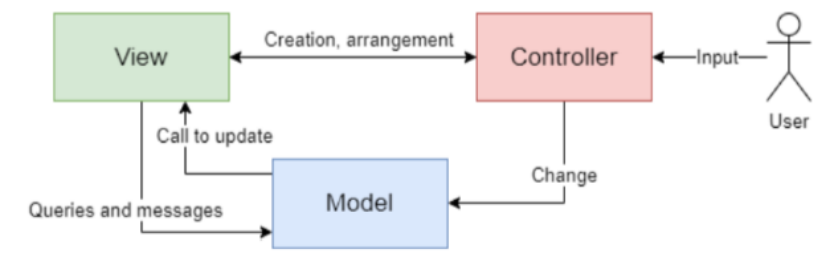
Obrázek 2.4 Datový model



Obrázek 2.5 Stavový diagram licence

2.9 Technologie

Před samotným návrhem a především implementací je třeba si určit technologie skrze které je webová aplikace Builder implementována. Již bylo zmíněno, že existuje prvotní prototyp aplikace, nicméně je třeba navrhnout technologie, přes které by měla být aplikace implementována dále, protože stávající řešení nemusí být řešením nejlepším.



Obrázek 2.6 MVC

2.9.1 Jazyk a prostředí

Pro vývoj webových aplikací jsou běžně používané jazyky Python, PHP, Java, C#, samotný JavaScript, React, Angular či VueJS. Prvotní prototyp aplikace byl vytvořen přes softwarový systém Node.js. V úvaze byl také Python s frameworkem Django. JavaScript se však po analýze jevil jako jednodušší na naučení a prostředí Node.js poskytovalo všechny potřebné vlastnosti. Django framework také nedokáže zpracovávat více požadavků najednou, protože postrádá asynchronní část.

Node.js je široce rozšířené open-source běhové prostředí, postavené na jazyku JavaScript. Toto prostředí je často využíváno pro event-driven servery, tedy backend aplikací, které reagují na požadavky poslané na API [23], což je přesně případ aplikace Builder. Má vysoký výkon pro real-time požadavky a jeho aplikace jsou známy tím, že se dají rychle vyvinout [24]. Další výhodou Node.js je NPM (Node Package Manager), který umožňuje pracovat s různými knihovnami a poskytuje jednoduchou práci s nimi, neboť instaluje nejen potřebné knihovny ale přímo i závislé knihovny. Node.js má single-threaded event loop architekturu. Při zapnutí aplikace se tato smyčka inicializuje a následně zpracovává jednotlivé instrukce. Node.js má také přímou podporu AWS, což je technologie, již zadavatel vytváří pro nasazování některých svých aplikací [23].

Naopak Node.js má problém s početně náročnějšími úkoly, což ale Builder dělat nebude, jelikož jeho hlavní náplní práce je komunikace s Gitlab CI/CD skrze pipelines. Pokud neexistují potřebné knihovny pro Node.js, kód tím může utrpět a stát se méně bezpečným. Další z problémových oblastí je asynchronní model, protože je náročný na údržbu [24]. Nebyly nalezeny žádné jiné technologické možnosti, které by umožňovaly to, co Node.js. Navíc prvotní prototyp byl již implementován v tomto prostředí a proto by bylo nevýhodné z časových důvodů předělávat prototyp do jiné technologie.

2.9.2 Persistence dat

Pro perzistentní ukládání dat bylo potřeba vybrat databázi. Díky ORM, objektově relačnímu mapování nezáleželo příliš na výběru databáze. Objektově relační mapování je programovací technika zajišťující automatickou konverzi dat mezi relační databází a objektově orientovaným programovacím jazykem. Jednotlivé instance tříd se mapují jako řádky tabulek databáze [25]. Jako nástroj pro toto mapování byl vybrán prostředek Sequelize, který zajišťuje mapování u Node.js aplikací s různými databázemi: Oracle, PostgreSQL, MySQL, MariaDB a další [26].

Kapitola 3

Implementace

Na základě návrhu z předešlé kapitoly je možné konečně přejít k samotné implementaci. Implementace přináší nicméně spoustu dosud neobjevených problémů a úskalí, z nichž ty nejzajímavější problémy jsou zahrnuty právě v této kapitole.

3.1 Verzování

V rámci celého vývoje aplikace Builder byl již od začátku zamýšlen nástroj Gitlab pro sestavování jednotlivých produktů pro konkrétní zákazníky. Tento nástroj se využívá jak pro verzování kódu, tak pro spouštění Gitlab CI/CD nástrojů, například pipeline, které fungují právě pro sestavování aplikací.

3.2 Vývojové prostředí

Před začátkem implementace je nejprve třeba vybrat správné vývojové prostředí, které díky svým funkcionalitám urychlí proces vývoje, usnadní rozpoznávání chyb a obecně zvýší produktivitu vývojáře. Samozřejmě by bylo možné vytvářet aplikace pouze v textovém editoru, nicméně vývojové prostředí poskytuje mnoho nástrojů pro vývoj, úpravu, překlad a ladění softwaru [27].

Při výběru vývojového prostředí byly zohledněny především mé zkušenosti s prostředím. Bylo zvoleno prostředí WebStorm, které je jedním z nejpoužívanějších vývojových prostředí pro JavaScript. Další variantou bylo používání IntelliJ IDEA. WebStorm je však podmnožinou prostředí IDEA a je tedy dostatečná. K oběma těmto prostředím poskytuje navíc licenci FIT ČVUT.

3.3 Běhové prostředí

Pro spuštění aplikace Builder byl již od začátku vývoje využíván virtulizační open-source nástroj Docker. Docker zabalí aplikaci do kontejneru společně s její databází a umožní jí běžet nezávisle na prostředí. V rámci toho spouští virtualizační prostředí, které se však na rozdíl od jiných nástrojů virtualizuje jen ve svém jádru a není třeba simulovat celý operační systém. [28]

Pro spouštění aplikace Builder je vytvořen Dockerfile, který sestavuje aplikaci [28] Builder a instaluje její potřebné balíčky.

Oproti tomu docker-compose.yml soubor specifikuje, jaké všechny části existují [28] a kromě aplikace Builder vytvoří také databázi – různé databáze pro produkci, testování a vývoj, kde každá běží na jiném portu.

Aplikaci Builder je možné spouštět pouze pomocí technologie Node.js. Zde však nastává problém, že aplikace nemá dostupnou databází a musela by být vytvořena lokální databáze běžící na specifikovaném portu.

3.4 Statická analýza

Během vývoje aplikací je důležité hlídat si styl psaní kódu, neboť inkonzistence způsobuje zmatení a odvádění pozornosti vývojářů, kteří chtějí pochopit, jak kód funguje. Proto existuje statická analýza kódu, která dokáže zabránit chybám z nepozornosti a upozornit na nepřesnosti v *code style*. *Code style* je styl kódu, který je možný nastavit pro každý projekt zvlášť, nicméně se využívají různé standardy. Pro statickou analýzu slouží jako vývojové prostředí, které poskytuje základní statickou analýzu, ale pro formátování byl vybrán balíček ESlint. Variantou byl také balíček JSLint. Tento balíček je však nekonfigurovatelný a nelze tedy určit si svůj vlastní *code style*. Další variantou byl balíček JSHint, který se vyvinul právě ze zmíněného JSLint. Tento balíček však nepodporuje tvorbu vlastních pravidel, takže byl zvolen nejvíce populární linter pro JavaScript v dnešní době, ESlint, který umožňuje vybrat si ze standardních *code styles* nebo si vytvořit svůj vlastní.

Pro používání ESlint byl balíček nainstalován do vývojového prostředí, kde se po spuštění zobrazily chybové hlášky v kódu. Zároveň s používáním tohoto balíčku byl přidán také balíček Prettier, který slouží k automatickým opravám chyb. Má svou vlastní konfiguraci a pokud je navolena tak, aby korespondovala s balíčkem ESlint, tak po stisknutí klávesové zkratky z dokumentace vývojového prostředí není již dále potřeba opravovat špatná odsazení, uvozovky apod. Při prvotním spuštění ESlint byly tázány otázky k mému stylu, které byly zodpovězeny a také byla možnost vybrat jeden z existujících stylů. [29] Na základě odpovědí byl vytvořen konfigurační soubor .eslintrc, který zobrazoval mé požadavky a bylo možné jej dále upravit. [30]

V dalším kroku byl aktualizován soubor .gitlab-ci.yml, který definuje skripty, jež běží při spuštění CI/CD pipeline a jejich konfigurace a závislosti. Do tohoto souboru byla přidána fáze kontroly přes balíček ESlint s využitím konfiguračního souboru .eslintrc.

3.5 Konfigurace aplikace

Pro konfiguraci aplikace je využit konfigurační soubor *.env*. Tento soubor obsahuje jak přístupové údaje do databáze, tak i ostatní proměnné aplikace, které se můžou měnit v závislosti na nasazení aplikace či jejích komponent. Tento soubor se udržuje v kořenovém adresáři repozitáře. [31]

Pro využívání tohoto konfiguračního souboru slouží široce užívaný balíček dotenv, který umožňuje udržovat citlivé informace v bezpečí a pomáhá je organizovat. [31]

Soubor *.env.example* se používá jako součást repozitáře, neboť je pro nasazení aplikace třeba ji konfigurovat a užívání tohoto souboru zajistí, že pokud je vytvořen *.env* soubor právě z tohoto příkladu, jsou v něm vytvořeny všechny požadované proměnné aplikace. [32]

3.6 Persistence dat

Jak již bylo zmíněno v předchozí kapitole, pro persistenci dat byl zvolen nástroj Sequelize zajišťující ORM. Dále je popsáno, jak je tento nástroj používán.

3.6.1 Modely

Modely jsou základem knihovny Sequelize. Každý model reprezentuje tabulku databáze. Každá instance modelu reprezentuje řádek tabulky databáze. Při definování jednotlivých atributů modelu jsou definovány sloupce tabulky. Pomocí možností sequelize je možné vytvořit i omezení v

modelech jako je například povolení nulové hodnoty ve sloupci, nebo nastavení defaultní hodnoty. Je také možné přímo nastavit zda se budou objevovat *timestamps*, tedy údaje o tom, kdy byl záznam tabulky vytvořen a kdy byl naposled aktualizován. [33]

3.6.2 Migrace

Migrace databází je proces, který přesouvá data z jedné nebo více databáze do cílové databáze. Tento proces je vhodný například při přecházení od jednoho databázového systému k jinému. Dále je vhodný také pro vyměňování databází dle účelu spuštění aplikace, tedy lze mít databázi samostatně pro vývoj, pro testování a pak také přímo pro produkci. Jedním z kroků pro databázové migrace je vytváření schématu v nové databázi dle schématu v původních databází. [34]

Pro vytvoření schématu v nové databázi pomocí migrací slouží právě migrační soubory knihovny Sequelize. Každá migrace obsahuje dvě metody, metodu pro projevení změny *up* a metodu pro zrušení změny *down*. [35]

3.6.2.1 Seeder

Seedery jsou migrace, které do databáze vkládají konkrétní data. Seedery jsou dobré pro používání, pro neustálé přechody od jedné databáze k druhé, kdy jsou již dopředu známá vstupní data. [34]

3.6.3 Sequelize-cli

Pro práci s nástrojem Sequelize byl nainstalován balíček sequelize-cli, který pomocí příkazové řádky umožňuje vytvářet v projektu modely, migrace, seedery a konfigurační soubor. V případě vygenerování modelu je automaticky vygenerována i migrace, která po svém zavolání pomocí metody *up* vytváří příslušný model v databázi a po zavolání metody *down* tento model zase z databáze smaže.

Pro vytváření modelů byly použity příkazy: 3.1. Po vygenerování těchto modelů však docházelo k dalším úpravám modelů, pro něž už další migrace ani modely nebyly generovány, ale raději byly přímo upravované ty již existující. Zajímavým případem, bylo zjištění, že při generování modelu musí být vždy alespoň jeden atribut. Proto případ generování modelu *ComponentProduct* obsahuje generování atributu *name*, který byl však následně ze samotného modelu i migrace vymazán. Dále je třeba vyzdvihnout, že v příkazu pro vytvoření komponenty se vyskytuje i datový typ enum, který umožňuje vyjmenovat všechny možnosti, které databáze pro daný sloupec dokáže přijmout.

Výpis kódu 3.1 Sequelize-cli bash příkazy pro vytvoření modelů a migrací

```
Licence:
 sequelize model:generate --name License
 --attributes active:boolean,licenseManager_id:integer,
 cancel_date:date,username:string --underscored

Component
 sequelize model:generate --name Component
 --attributes name:string,type:enum:
 '{'ANDROID', 'ELECTRON', 'WEB', 'WINDOWS', 'MACOS'}',
git_repository:string,last_git_pull_date:date,
gitlab_pipeline_token:string,git_ref_branch:string --underscored

Product
sequelize model:generate --name Product
--attributes name:string,product_id_from_lm:integer --underscored
```

```
Application
 sequelize model:generate --name Application
 --attributes port:integer,url:string,built:boolean --underscored

File
sequelize model:generate --name File
--attributes url:integer,upload_date:date --underscored

ComponentProduct
sequelize model:generate --name ComponentProduct
--attributes name:string
```

V rámci jednotlivých migrací a modelů bylo později definováno, které atributy mohou být prázdné a tedy nemusí být přítomny při vkládání do databáze. Dále byly specifikovány defaultní hodnoty, pokud existují a také které sloupce v databázi musí mít unikátní hodnoty. Při porušení těchto hodnot Sequelize vyhodí error.

Při dovytvořeni těchto vlastností je viditelný nedostatek knihovny. Při manuální aktualizaci souboru s modelem není aktualizován soubor s migrací a naopak.

3.6.4 Spouštění migrací

Spouštění migrací je možné definovat přímo pro konkrétní migraci dle souboru či všechny najednou. Spouštění seederů je odlišné od spouštění migrací pro schéma databáze, dle dokumentace [35].

3.6.5 Asociace

Pro vytvoření vztahů mezi tabulkami databáze jsou nutné asociace. Asociace musí být řešeny jak v modelech, tak i v migracích, neboť se jedná o součást schéma databáze.

Při vývoji byly implementovány různé druhy asociací: vazba 1:M a vazba M:N. Nejzajímavější částí asociací byla bezpochyby asociace mezi modely *Component* a *Product*, neboť tyto dva modely mají mezi sebou právě vazbu M:N. Dle dokumentace knihovny Sequelize [36] se však již nejedná o tak složitý problém a pomocí dekompozice asociace M:N na dvě vazby 1:N s pomocí modelu *ComponentProduct* byla vztahu mezi tabulkami docíleno.

3.6.6 Konfigurace

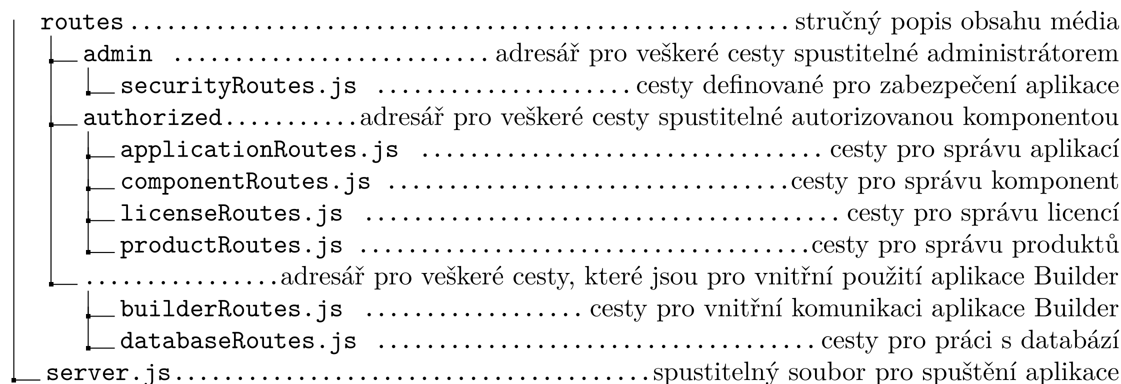
Pro vytváření jednotlivých databází existuje konfigurační soubor knihovny Sequelize. Tento soubor specifikuje přístupové údaje do databáze a také dialekt databáze. [34].

Pro konfiguraci knihovny Sequelize v celém projektu je v kořenovém adresáři uložen konfigurační soubor .sequelizerc.js, který vypovídá, kde Sequelize-cli najde jednotlivé složky pro konfiguraci databází, modely, migrace a seedery. [37]

3.6.7 Subjektivní hodnocení Sequelize

Při práci s nástrojem Sequelize byly zaznamenány časté problémy s nedostatečnou dokumentací [38], které způsobily nefunkčnost na první pohled jednoduchých metod, které byly zprostředkovány přes syntaxi nástroje.

Po delší implementaci se již jevilo jako větší problém vybírat jinou technologii pro implementaci ORM než vytvoření testování jednotlivých funkcí pracujících s databází.



Obrázek 3.1 Struktura adresáře pro definování cest

Jako další možnost řešení tohoto problému se jevilo přímé použití SQL dotazů přes nástroj Sequelize v kódu aplikace, nicméně to by způsobilo horší čitelnost kódu a pro neznalé SQL. či po změně databáze například z PostgreSQL do MySql by dotaz nemusel být funkční.

Konečným řešením problému bylo tedy průběžné integrační testování metod, které se napojují přímo na knihovnu Sequelize, toto testování je více popsáno v části integrační testování (4.3.2).

3.7 Aplikace Builder

Po definování databáze bylo konečně možné vrhnout se do práce na samotné aplikaci. Aplikace Builder disponovala různými nedostatky, které bylo třeba napravit a postup nápravy je popsán dále.

3.7.1 MVC

Jak již bylo zmíněno v kapitole Návrh 2.8, pro implementaci této aplikace byla zvolena architektura MVC.

Při změně struktury bylo potřeba zavést především strukturu pro složku *routes*, kde se před touto částí implementace vyskytovalo pár souborů, které rovnou řešili sestavování aplikací. Tento přístup však nerozděloval správně vrstvy a bylo nutné jej upravit tak, aby veškerou správu nechával na logické stránce aplikace, tj. na kontrolorech. Proto bylo třeba vytvořit kontrolery a upravit cesty *routes* dle návrhu.

Koncové body aplikace Builder byly specifikovány v kapitole Návrh v části API. Tyto koncové body v aplikaci Builder jsou upraveny tak, aby odpovídaly běžným technikám a jsou rozděleny dle svého zaměření. V 3.1 je zobrazena struktura adresáře pro specifikování jednotlivých koncových bodů.

Oproti původní implementaci se jednalo o přejmenování některých souborů, které plně nevyjadřovaly funkcionalitu, například soubor *lm.js* sloužil pro správu licencí.

Dále byly dle běžné praxe vytvořeny kontrolery. Kontrolery jsou součástí MVC architektury. Zde slouží pro navýšení abstrakce mezi přijímáním HTTP požadavků z API a logiku pro jejich naplnění. V prvotním prototypu tyto kontrolery nebyly implementovány a řešení HTTP požadavků bylo implementováno přímo ve specifikaci cest. [39], [40], [41]

Pro strukturu MVC byly přidány také *Services*, jenž mají za úkol starat se o business logiku aplikace. Tato vrstva taky přímo využívá přístup k datové vrstvě aplikace, jež v implementaci představuje knihovna Sequelize. [42]

3.7.2 Dodržování SRP

Předělávání aplikace bylo v první části zaměřeno na rozdělení složitých metod do menších srozumitelných částí a tříd, dle principu OOP SRP.

Jak již bylo zmíněno, byly vytvořena aplikace dle struktury MVC, byly tedy vytvořeny kontrolery a services. Při implementaci třídy zajišťující obsluhování komponent se objevil problém, že je třeba se chovat dle typu komponenty.

Proto byly vytvořeny podtypy třídy *ComponentService* pro jednotlivé typy komponent, které obstarávají práci s komponentou.

Zajímavým zjištěním v průběhu této části implementace bylo zjištění, že JavaScript neobsahuje interfaces [43] na rozdíl od jiných jazyků.

Jako varianta se jevil jazyk TypeScript, který běží v prostředí Node.js a umožňuje užití interface [44]. Proběhla proto krátká analýza, zda nepřejít k jazyku TypeScript, který je velice postavený na jazyku JavaScript s rozdílem statického typování místo dynamického a jeho součástí je tedy kompilace [45]. Tento proces se obecně nepovažuje jako složitý [46], nicméně by výrazně pozdržel tuto bakalářskou práci a proto od něj bylo opuštěno.

3.7.3 Objekty nebo třídy

Pro implementaci jednotlivých modulů bylo třeba vybrat, zda se bude jednat o třídy, které budou mít několik instancí či zda se bude jednat o objekty, které existují v celé aplikaci pouze jednou.

JavaScript vytváří objekty v podstatě jako singletony, tzn. instance tříd, které se vrátí při každém zavolání modulu. Tato varianta byla ze začátku prosazována, nicméně po delší implementaci bylo rozhodnuto, že v případě, že by se jednalo o asynchronní aplikaci, mohlo by být časem třeba více objektů jedné třídy, které by vše obstarávaly. Proto bylo rozhodnuto, že každý modul bude implementován jako singleton, nicméně za použití ES6 JavaScript tříd.

3.7.4 Továrna

Pro implementaci polymorfismu v jazyku JavaScript slouží rozšíření ES6. Toto rozšíření umožňuje, aby od sebe třídy dědily a umožňuje tak přejímat funkce. Pro implementaci jednotlivých podtypů *GitlabService*, tedy třídy obsluhující veškerou komunikaci s technologií Gitlab, se jako jednoduché řešení pro přípravu dat odesílaných na Gitlab jevil právě polymorfismus.

Následně se ale v implementaci aplikace se objevil problém, který byl vyřešen pomocí návrhového vzoru Továrna. Tento návrhový vzor dává možnost vytvářet instance objektů dle právě zvoleného argumentu za běhu aplikace. [47]

Tento problém se objevil právě při vytváření podtypů *GitlabService*. Před spuštěním aplikace totiž nebylo jasné, jakého je komponenta typu a nedala se tedy jednoduše staticky vytvořit podtřída třídy *GitlabService*. Továrna umožnila vytvořit

3.8 Kontrola vstupu

Pro správu vstupů byla využita expresss-validator. Tato knihovna dokáže kontrolovat data, která přicházejí v příchozím požadavku.

3.9 Správa chyb a výjimek

Pro správu chyb a výjimek byl uvažován nejprve handler pro celou aplikaci zajišťující řešení problémů centrálně. Následně se ale ukázalo, že se nejedná o tak jednoduchou implementaci a

od tohoto návrhu bylo upuštěno. Tuto část je tedy možné dále řešit v dalším vývoji aplikace Builder.

Pro zachytávání jinak nezachycených výjimek byl do implementace přidán kód 3.2.

Výpis kódu 3.2 Zachytávání neodchycených výjimek

```
process.on('unhandledRejection', (reason, p) => {
        console.log('Unhandled Rejection at:', p, 'reason:', reason);
        // application specific logging, throwing an error, or other logic here
});
```

3.10 Logování

Pro informace o běhu aplikace, o problémech, které vznikly a podobně slouží logování. Logování je implementováno pomocí knihovny winston, která je běžně používanou knihovnou v Node.js. Winston umožňuje logování na různých levelech a je možné si definovat i vlastní úrovně logování a poté následně logování při přílišném množství zpráv omezit pomocí změny úrovně.

Pro budoucí vývoj je určitě důležité neopomenout napojení aplikace na Sentry, což je software pro monitorování chyb a lze napojit přímo na knihovnu winston.

3.11 Nové funkcionality

V průběhu implementace byly vytvořeny funkcionality naplňující nově specifikované požadavky pro správu produktů a správu komponent. Byly vytvořeny koncové body pro zobrazení všech existujících komponent i produktů, pro zobrazení konkrétního produktu a pro zobrazení konkrétní komponenty. Také byl vytvořen koncový bod pro zobrazení všech existujících aplikací, tedy běžících komponent.

Kapitola 4
Testování

Po implementaci je na řadě testování chování aplikace, jež je nutné k dalšímu vývoji, aby při přidání nových funkcionalit či refaktorizaci nebyly ovlivněny již funkční části aplikace Builder.

Testování softwaru je v dnešní době nezbytnou součástí vývoje softwaru. Softwarové testování verifikuje splnění všech požadavků, kontroluje, zda software dělá co má a nedělá, co nemá. Přináší také zprávu o stavu softwaru i během vývoje a pomáhá nalézt chyby před vypuštěním softwaru do produkce.

4.1 Druhy testování

Testování je děleno na dvě základní kategorie. Dynamické testování a statické testování. Dynamické testování je takové, které probíhá na konci implementace a při jeho provádění je software spouštěn za účelem ověření kvality. Statické testování spočívá v kontrolách kódu, např. zda neexistují nedosažitelné části kódu či jestli kód neobsahuje chyby v logice. Dále se zaměřuje na zadání požadavků, zda jsou správně definované a nejsou vynechané určité části a zda je vše správně navrženo z technické strany.

Dále je testování možno dělit z pohledu automatizace. Testy se dají provádět jak manuálně, kdy se krok po kroku provádí požadované operace a kontrolují se výsledky. Proti tomu stojí automatizované testy, které sami verifikují požadované funkcionality. Tato automatizace přináší spoustu výhod, jakou je například znovupoužitelnost testů nebo také vysoké zvýšení efektivity práce.

4.2 Manuální testování

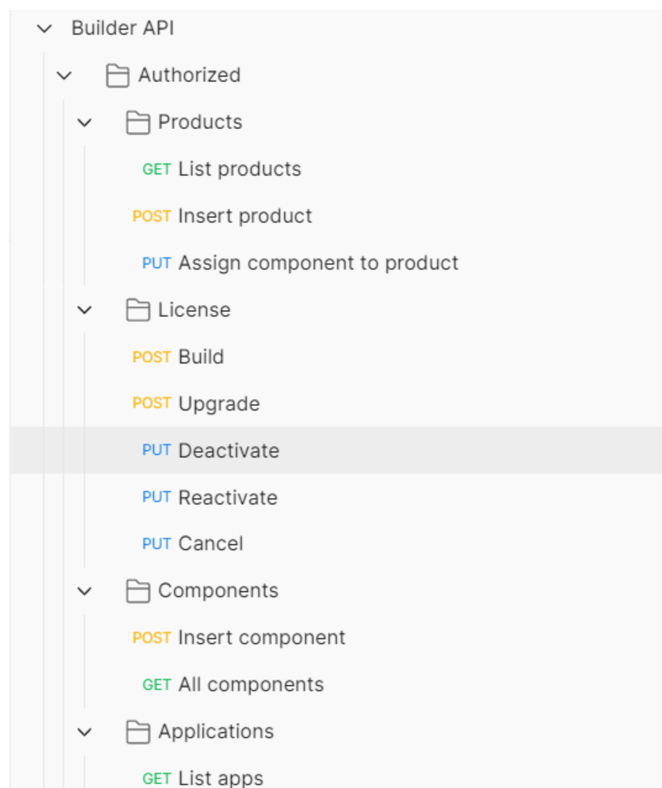
Manuální testování je časově náročnější druh testování, protože vyžaduje důkladnou kontrolu jednotlivých výsledků.

4.2.1 Průběžné testování

Průběžné testování představuje testování během vývoje aplikace, kdy je aplikaci posílán vstup a je kontrolováno, zda aplikace vrátí požadovaný výstup. Toto testování má výhodu, že podává výsledky o celkovém fungování požadavků na systém a je tedy možné říci, které požadavky jsou již splněny a které ne. Nevýhodou tohoto testování je to, že jsou snadno opomenuty konkrétní případy užití a může se tedy stát, že aplikace funguje správně jen v konkrétních případech.

Pro tento druh testování byl používán nástroj Postman. Nástroj Postman je API platforma pro sestavování a používání API [48].

V nástroji Postman byla vytvořena struktura dotazů odpovídajících dotazům definovaným v API 2.4.1, jak je vidět na 4.1 a 4.2. Pro simulaci požadavků je v přiloženém archivu poskytnut soubor, který lze importovat do nástroje Postman a pomocí nastavení proměnných *BUILDER_URL* a po registraci také *auth_token* lze aplikaci Builder testovat jak lokálně tak na serveru.



Obrázek 4.1 Postman – Autorizované požadavky

4.2.2 Akceptační testování

Akceptační testování je testování během kterého je aplikace předvedena zákazníkovi (zde zadavateli), který podá zpětnou vazbu na aplikaci a uvede, které funkcionality mu chybí a kde vidí skryté problémy.

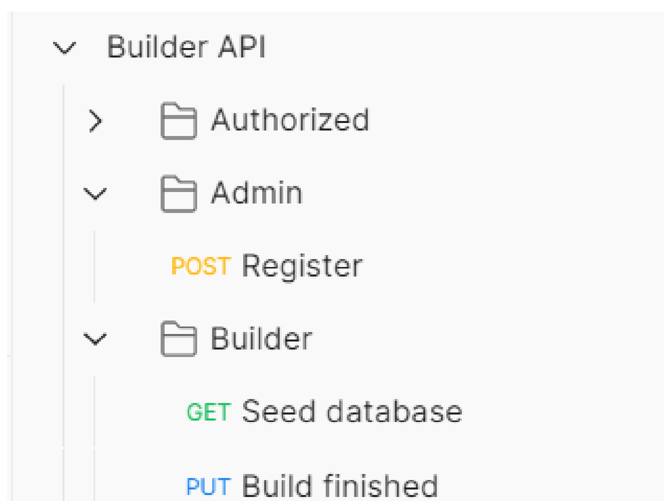
Akceptační testování nakonec nebylo provedeno.

4.3 Automatické testování

4.3.1 Jednotkové testy

Jednotkové, nebo také unit testy, jsou testy, které se zaměřují pouze na konkrétní část aplikace a ostatním částem aplikace je jejich chování nasimulováno, tzv. mockováno. [49]

Testování aplikace je tedy rozděleno na testování jednotlivých tříd. Je testováno API, jsou testovány kontrolery, jsou testovány services.



Obrázek 4.2 Postman – Ostatní požadavky

Pro testování aplikace byla vybrána knihovna Jest, což je široce rozšířená knihovna pro testování Node.js aplikací. [50]

Při běhu vícero testů se jednalo o poměrně pomalý průběh, proto byly hledány způsoby, jak testování urychlit. To existuje pomocí užití parametru *–maxWorkers=50%*, což upřesní počet vláken, které může Jest používat a proto poběží rychleji. [51]

Pro psaní testů pro Services byla využita knihovna Sequelize-test-helpers, pro testování zasílání požadavků zase knihovna Axios-mock-adapter.

4.3.2 Integrační testování

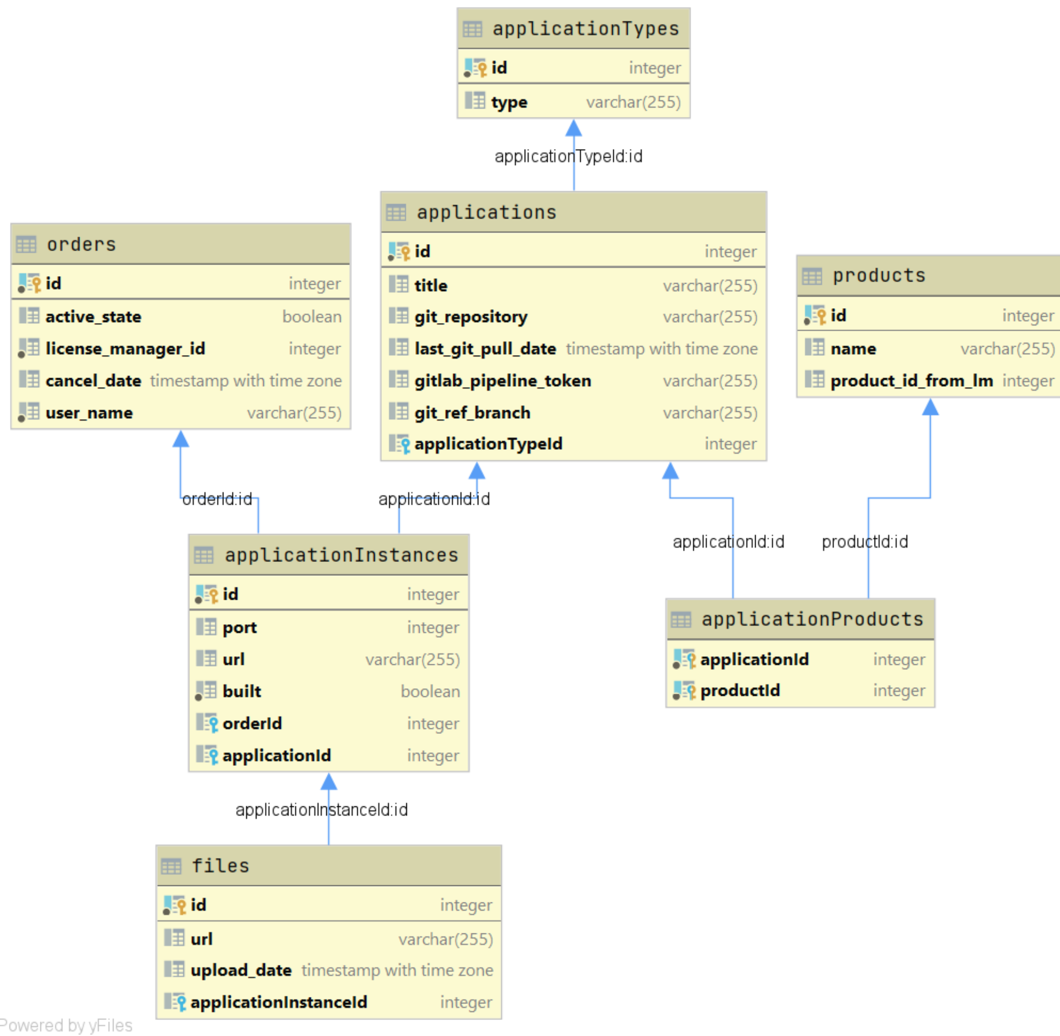
Integrační testování je testování, které slouží k verifikaci vztahů mezi jednotlivými částmi systému. Tedy pokud aplikace pracuje s databází nebo jinými službami jako je hosting nebo gitlab, integrační testování zkoumá právě to.

Integrační testování se stalo důležitou součástí pro funkčnost celé aplikace. Vzhledem ke špatné dokumentaci návratových hodnot Sequelize nebylo možné bez integračního testování pokračovat ve vývoji aplikace.

Integrační testování na rozdíl od unit testování nemockuje součást systému. Testuje se právě součinnost částí systému.

Příloha A

Databázový model prvotního prototypu



Obrázek A.1 Databázový model prvotního prototypu

Bibliografie

1. MLEJNEK, Ing. Jiří. *Analýza a sběr požadavků — Softwarové inženýrství BI-SI1* [online]. 2023. [cit. 2023-04-23]. Dostupné z: https://moodle-vyuka.cvut.cz/pluginfile.php/532096/mod_resource/content/7/03.prednaska.pdf.
2. ROPER, Kimone. *Importance of System Analysis in Software Development — LinkedIn* [online]. 2021-09. [cit. 2023-05-11]. Dostupné z: https://www.linkedin.com/pulse/importance-system-analysis-software-development-kimone-roper/?trk=public_profile_article_view.
3. *Quantitative Analysis (QA): What It Is, How It's Used in Finance* [online]. 2020. [cit. 2023-05-09]. Dostupné z: https://www.investopedia.com/terms/q/quantitativeanalysis.asp.
4. BHANDARI, Pritha. *What Is Qualitative Research? — Methods & Examples* [online]. 2020-06. [cit. 2023-05-11]. Dostupné z: https://www.scribbr.com/methodology/qualitative-research/.
5. PROFINIT. *Softwarový proces* [online]. 2023. [cit. 2023-05-11]. Dostupné z: https://moodle-vyuka.cvut.cz/pluginfile.php/530559/course/section/84272/2021_2022/01_SoftwareProcess.pdf.
6. HOLÝ, Viktor. *License manager – webová aplikace* [online]. 2021-05. [cit. 2023-04-23]. Dostupné z: https://dspace.cvut.cz/bitstream/handle/10467/95066/F8-BP-2021-Holy-Viktor-Thesis-15.pdf?sequence=-1&isAllowed=y.
7. S.R.O., BEST-HOSTING. *Co je to hosting? - BEST-HOSTING.cz* [online]. 2013. [cit. 2023-05-11]. Dostupné z: https://best-hosting.cz/cs/napoveda/co-je-to-hosting.
8. S.R.O, Webglobe. *Hosting90.cz by Webglobe — Managed servery, VPS, webhosting a domény* [online]. 2023. [cit. 2023-05-11]. Dostupné z: https://www.hosting90.cz/.
9. *FormData - Web APIs — MDN* [online]. 2023-02. [cit. 2023-04-23]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/FormData.
10. *Getting Started — Axios Docs* [online]. [cit. 2023-04-23]. Dostupné z: https://axios-http.com/docs/intro.
11. WIKIPEDIE. *Node.js — Wikipedie: Otevřená encyklopedie* [online]. 2022. [cit. 2023-04-23]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Node.js&oldid=21322165.
12. *PostgreSQL: The world's most advanced open source database* [online]. 1996. [cit. 2023-04-23]. Dostupné z: https://www.postgresql.org/.
13. *CI/CD concepts — GitLab* [online]. [cit. 2023-04-23]. Dostupné z: https://docs.gitlab.com/ee/ci/introduction/index.html/#continuous-integration.

14. *Analýza, diagramy – Disk Google* [online]. 2022. [cit. 2023-05-11]. Dostupné z: https://drive.google.com/drive/folders/1Lz1bcL7PmYwBXK1-PL9_ph88RcpmtpCI.
15. *BI-OOP 2022 - Lecture 1: Introduction - Prezentace Google* [online]. 2022. [cit. 2023-04-27]. Dostupné z: https://docs.google.com/presentation/d/1hAbK6ox6OPYwzwPOZxmTtXjcjEcBXlrtb-SgHfiUaPY/edit#slide=id.g615494dbd9_0_0.
16. *Objektově orientované programování – Wikipedie* [online]. 2023-01. [cit. 2023-04-27]. Dostupné z: https://cs.wikipedia.org/wiki/Objektov%C4%9B_orientovan%C3%A9_programov%C3%A1n%C3%AD.
17. *Single-responsibility principle - Wikipedia* [online]. 2022-12. [cit. 2023-04-27]. Dostupné z: https://en.wikipedia.org/wiki/Single-responsibility_principle.
18. WIKIPEDIE. *FURPS — Wikipedie: Otevřená encyklopedie* [online]. 2021. [cit. 2023-04-26]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=FURPS&oldid=20306564.
19. *What is MoSCoW Prioritization? — Overview of the MoSCoW Method* [online]. [cit. 2023-04-26]. Dostupné z: https://www.productplan.com/glossary/moscow-prioritization/.
20. *What is a REST API? — IBM* [online]. 2023. [cit. 2023-05-11]. Dostupné z: https://www.ibm.com/topics/rest-apis.
21. *SwaggerHub — API Design and Documentation with OpenAPI* [online]. 2023. [cit. 2023-05-09]. Dostupné z: https://swagger.io/tools/swaggerhub/.
22. S.R.O, Webglobe. *Nápověda a API dokumentace — Hosting90 by Webglobe* [online]. 2023. [cit. 2023-05-11]. Dostupné z: https://docs.hosting90.cz/.
23. KAREN, Ana. *Why use Node js? 5 Reasons to use Node.js for your App* [online]. 2022-05. [cit. 2023-04-23]. Dostupné z: https://www.clickittech.com/developer/why-use-node-js/.
24. TONDON, Sophia. *The Pros and Cons of Node.js Web App Development: A Detailed Look — by Sophia Tondon — JavaScript in Plain English* [online]. 2022-02. [cit. 2023-04-23]. Dostupné z: https://javascript.plainenglish.io/the-pros-and-cons-of-node-js-web-app-development-a-detailed-look-c91a22f013c.
25. WIKIPEDIE. *Objektově relační mapování — Wikipedie: Otevřená encyklopedie* [online]. 2021. [cit. 2023-04-23]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Objektov%5C%C4%5C%9B_rela%5C%C4%5C%8Dn%5C%C3%5C%AD_mapov%5C%C3%5C%A1n%5C%C3%5C%AD&oldid=20335589.
26. CONTRIBUTORS, Sequelize. *Sequelize — Feature-rich ORM for modern TypeScript & JavaScript* [online]. 2023. [cit. 2023-04-24]. Dostupné z: https://sequelize.org/.
27. WIKIPEDIE. *Vývojové prostředí — Wikipedie: Otevřená encyklopedie* [online]. 2023. [cit. 2023-04-23]. Dostupné z: https://cs.wikipedia.org/wiki/V%5C%C3%5C%BDvojov%5C%C3%5C%A9_prost%5C%C5%5C%99ed%5C%C3%5C%AD.
28. VONDRA, Marek. *Lekce 1 - Docker - Teorie a instalace* [online]. 2023. [cit. 2023-05-11]. Dostupné z: https://www.itnetwork.cz/site/docker/docker-teorie-a-instalace.
29. ORONA-CALVERT, Zachary. *Compare the Top 3 Style Guides and Set Them Up With ESLint — by Zachary Orona-Calvert — Better Programming* [online]. 2020-04. [cit. 2023-04-23]. Dostupné z: https://betterprogramming.pub/comparing-the-top-three-style-guides-and-setting-them-up-with-eslint-98ea0d2fc5b7.
30. *Eslint configuration for Node Project - DEV Community* [online]. 2021-10. [cit. 2023-04-23]. Dostupné z: https://dev.to/drsimplegraffiti/eslint-configuration-for-node-project-275l.
31. KOGAN, Daniel. *Why a .env?. Why are dotenv files important for your. . . — by Daniel Kogan — Dev Genius* [online]. 2022-11. [cit. 2023-05-10]. Dostupné z: https://blog.devgenius.io/why-a-env-7b4a79ba689.

32. STHALEKAR, Abhishek. *Why do you copy the .env.example file and create a .env file in Laravel? - Quora* [online]. 2019. [cit. 2023-05-10]. Dostupné z: https://www.quora.com/Why-do-you-copy-the-env-example-file-and-create-a-env-file-in-Laravel.
33. CONTRIBUTORS, Sequelize. *Model Basics — Sequelize* [online]. 2023. [cit. 2023-05-10]. Dostupné z: https://sequelize.org/docs/v6/core-concepts/model-basics/.
34. FATIMA, Nida. *Database Migration: What It Is and How It Is Done — Astera* [online]. 2023. [cit. 2023-05-10]. Dostupné z: https://www.astera.com/type/blog/database-migration-what-it-is-and-how-it-is-done/.
35. CONTRIBUTORS, Sequelize. *Migrations — Sequelize* [online]. 2023. [cit. 2023-05-10]. Dostupné z: https://sequelize.org/docs/v6/other-topics/migrations/.
36. CONTRIBUTORS, Sequelize. *Advanced M:N Associations — Sequelize* [online]. 2023. [cit. 2023-05-10]. Dostupné z: https://sequelize.org/docs/v6/advanced-association-concepts/advanced-many-to-many/.
37. *javascript - Migrations in Sequelize in my own folder structure - Stack Overflow* [online]. 2022-08. [cit. 2023-05-10]. Dostupné z: https://stackoverflow.com/questions/54404302/migrations-in-sequelize-in-my-own-folder-structure.
38. CONTRIBUTORS, Sequelize. *Model Querying - Basics — Sequelize* [online]. 2023. [cit. 2023-05-10]. Dostupné z: https://sequelize.org/docs/v6/core-concepts/model-querying-basics/.
39. *Express Tutorial Part 4: Routes and controllers - Learn web development — MDN* [online]. 2023. [cit. 2023-05-10]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes#create_the_route-handler_callback_functions.
40. *NodeJS + Express part 5: Routes and Controllers - DEV Community* [online]. 2021-01. [cit. 2023-05-10]. Dostupné z: https://dev.to/ericchapman/nodejs-express-part-5-routes-and-controllers-55d3.
41. *Understanding MVC pattern in Nodejs - DEV Community* [online]. 2021-06. [cit. 2023-05-10]. Dostupné z: https://dev.to/eetukudo_/understanding-mvc-pattern-in-nodejs-2bdn.
42. BECHTOL, Evan. *Node Service-oriented Architecture — Codementor* [online]. 2020-01. [cit. 2023-05-10]. Dostupné z: https://www.codementor.io/@evanbechtol/node-service-oriented-architecture-12vjt9zs9i.
43. *oop - Does JavaScript have the interface type (such as Java's 'interface')? - Stack Overflow* [online]. 2010. [cit. 2023-05-10]. Dostupné z: https://stackoverflow.com/questions/3710275/does-javascript-have-the-interface-type-such-as-javas-interface.
44. *TypeScript: Handbook - Interfaces* [online]. 2023. [cit. 2023-05-10]. Dostupné z: https://www.typescriptlang.org/docs/handbook/interfaces.html.
45. W3SCHOOLS. *TypeScript Introduction* [online]. 2023. [cit. 2023-05-10]. Dostupné z: https://www.w3schools.com/typescript/typescript_intro.php.
46. *TypeScript: Documentation - Migrating from JavaScript* [online]. 2023. [cit. 2023-05-10]. Dostupné z: https://www.typescriptlang.org/docs/handbook/migrating-from-javascript.html.
47. REFACTORING.GURU. *Factory Method* [online]. 2023. [cit. 2023-05-10]. Dostupné z: https://refactoring.guru/design-patterns/factory-method.
48. POSTMAN, Inc. *What is Postman? Postman API Platform* [online]. 2023. [cit. 2023-05-10]. Dostupné z: https://www.postman.com/product/what-is-postman/.

49. *Unit Testing Tutorial – What is, Types Test Example* [online]. [cit. 2023-05-11]. Dostupné z: https://www.guru99.com/unit-testing-guide.html.
50. *Jest · Delightful JavaScript Testing* [online]. [cit. 2023-05-11]. Dostupné z: https://jestjs.io/.
51. *Make Your Jest Tests up to 20% Faster by Changing a Single Setting - DEV Community* [online]. [cit. 2023-05-11]. Dostupné z: https://dev.to/vantanev/make-your-jest-tests-up-to-20-faster-by-changing-a-single-setting-i36.