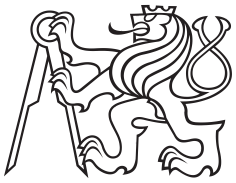
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Generování kódu stubů z modelů Enterprise Architect

Vít Nademlejnský

Vedoucí: Ing. Jiří Šebek
Studijní program: Otevřená informatika
Specializace: Software
Květen 2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Nademlejnský** Jméno: **Vít** Osobní číslo: **499187**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Software**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Generování kódu stubů z modelů Enterprise Architect

Název bakalářské práce anglicky:

Generating stub code from Enterprise Architect models

Pokyny pro vypracování:

Cílem práce je zanalyzovat databázovou strukturu sparxEA databáze a na základě toho vytvořit generátor kódu. Generátor by měl umět generovat swagger, stuby pro frontend a backend.
Motivací je generování kódu, který se vždy vyskytuje v kódu a pro generování speficikací v jednom modelu.
Požadavky na funkcionalitu:
1) Možnost konfigurace atributů v rámci modelu v Enterprise Architect
2) Možnost generování i struktur DTO, parametrů endpointů, LDM modelu
3) Možnost generování struktury pro API endpointy
4) Srovnání s běžným vývojem aplikací (React, Springboot)
Výsledná práce bude obsahovat analýzu, návrh, implementaci a dokumentaci pro používání.
Implementace bude obsahovat generátor pro Springboot stub, react stub a swagger.

Seznam doporučené literatury:

Cao, Hanyang, Jean-Rémy Falleri, and Xavier Blanc. "Automated generation of REST API specification from plain HTML documentation." Service-Oriented Computing: 15th International Conference, ICSOC 2017, Malaga, Spain, November 13–16, 2017, Proceedings. Springer International Publishing, 2017.
SparxEA documentation - https://sparxsystems.com/enterprise_architect_user_guide/15.2/guidebooks/tools_ba_documentation.html
B. Selic, "The pragmatics of model-driven development," in IEEE Software, vol. 20, no. 5, pp. 19-25, Sept.-Oct. 2003, doi: 10.1109/MS.2003.1231146.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jiří Šebek kabinet výuky informatiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **20.02.2023** Termín odevzdání bakalářské práce: **26.05.2023**

Platnost zadání bakalářské práce: **16.02.2025**

Ing. Jiří Šebek podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____ _____

Datum převzetí zadání Podpis studenta

Poděkování

Chtěl bych poděkovat svému vedoucímu práce Ing. Jiřímu Šebkovi za hodnotnou pomoc v průběhu vypracovávání práce a Vítu Gardoňovi za účast na uživatelských testech.

Prohlášení

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

.....
Vít Nademlejnský

V Praze, 26. května 2023

Abstrakt

Tato bakalářská práce řeší generování stubů na základě analytických modelů.

K řešení byl použit OpenAPI Generator, který na základě vygenerovaných OpenAPI specifikací aplikací z modelů vytvořených v Enterprise Architect dokáže generovat Frontend stuby na straně klienta a Backend stuby na straně serveru. Při vývoji byly použity různé návrhové vzory, které řeší čitelnost a práci s pamětí.

Provedeným výzkumem jsem zjistil, že aplikace šetří čas a práci všem potenciálním uživatelům ze softwarového inženýrství.

Výsledná aplikace byla po detailním otestování ohodnocena pozitivně v rámci uživatelské přívětivosti a splňuje všechny stanovené požadavky.

Klíčová slova: softwarové inženýrství, generátor kódu, Swagger, OpenaAPI Generator, Enterprise Architect, Spring Boot, Java, MySQL

Vedoucí: Ing. Jiří Šebek
Katedra počítačů,
Karlovo náměstí 13,
Praha 2, 121 35

Abstract

This bachelor thesis deals with the generation of stubs based on analytical models.

OpenAPI Generator was used for the solution, which can generate Frontend stubs on the client side and Backend stubs on the server side based on OpenAPI specification generated by the application from models created in Enterprise Architect. Various design patterns were used in the development process to address readability and memory handling.

Through the research, I found out that the application saves time and work for all potential users from the software engineering industry.

After detailed testing, the resulting application was rated positively regarding user-friendliness and met all the initial requirements.

Keywords: software engineering, code generator, Swagger, OpenaAPI Generator, Enterprise Architect, Spring Boot, Java, MySQL

Obsah

Obrázky

Tabulky

Kapitola 1

Úvod

V průběhu let, kdy se zvětšila komplexita a velikost vyvíjeného softwaru a začalo se více dbát na kvalitu a ušetřený čas při vývoji, začaly vznikat nové výzvy pro společnosti dodávající různé druhy softwaru. Těmto společnostem se daří tyto výzvy plnit za pomocí Model-driven developmentu (MDD), který je postupem let čím dál tím více osvojován velkými společnostmi na trhu a zajišťuje zlepšení v oblasti kvality, chybovosti a časové náročnosti vytváření kódu. Jelikož každá z těchto společností má ve svých řadách profesionální analytiky, kteří dokáží provést detailní analýzu zadaného projektu, který následně převedou do podoby analytických modelů, je pro vývojáře a testery mnohem jednodušší pochopit řešenou tématiku a interakci mezi entitami.[1]

Nepostradatelnou součástí metodik MDD jsou různé nástroje podporující tento druh vývoje. Mezi ně patří například generátory kódu, které zajišťují transformaci abstraktních modelů do rozvedeného kódu. Tyto nástroje jsou v dnešní době nedílnou součástí vývoje softwaru. Jejich použití je toiž jednoduché, rychlé a velmi efektivní.[2]

1.1 Cíle práce

Cílem této bakalářské je návrh, implementace a otestování aplikace, která bude schopna tyto metodiky využít. Společně s nástrojem pro tvorbu modelů bude aplikace připojena na společnou databázi, ze které bude aplikace z vytvořených modelů generovat kódy a k nim i příslušnou dokumentaci ve

formě Swaggerů.

1.2 Výhody

Mezi výhody, jež práce přináší, se řadí:

- Urychlení času na vývoj aplikace za pomocí generátorů, jelikož vývojáři nebudou muset implementovat opakující se kód.
- Úbytek času a také počtu chyb, které se mohou v kódu vyskytnout, jelikož se jedná o automatizovaný proces.
- Vždy dodaná vygenerovaná dokumentace k vygenerovanému kódu, která bude pomocí různých UI (User Interface) nástrojů přehledně zobrazitelná.
- Standard pro vývoj - vývojáři si nemůžou dělat co chtějí. Generování zamezuje vývojářům mít větší volnost a tím se zvyšuje bezpečnost samotného kódu a aplikace, která z něj poté vzniká.

1.3 Příklad ušetření času

Příkladem pro ušetření času je vývoj systému pro školu, kde se přes Frontend aplikace budou moci dozvědět studenti pomocí napojení databáze a logiky, která probíha na Backendu, různé informace o sobě, učitelích, studovnách, studijních materiálech, předmětech, školním řádu a plno dalších informacích, co může školní systém nabídnout.

Nejprve zkušený analytik vypracuje analýzu celého systému. Nastolí cíle, business a systémové požadavky projektu, případy užití a k nim scénáře, které mohou nastat průchodem aplikace uživatelem. Na základě toho vytvoří Analytický doménový model, ze kterého je aplikace schopna vygenerovat kód a po dodání modelů rozhraní a metod API (Application Programming Interface) by proběhlo generování Frontendu i Backendu.

1.4 Porovnání časů odvedené práce s a bez generátoru

Analytikovi by zpracování komplexní analýzy celého systému trvala přibližně 2 MD. Vypracování analýzy je provedeno v obou případech užití.

V případě generování by analytik dodal modely rozhraní a metod. Tvorba modelů by trvala přibližně půl MD, a jelikož nastavení aplikace a vygenerování trvá jednotky minut, můžeme tento časový údaj zanedbat. Celkem bychom se dostali k vytvoření kódu pomocí generátoru za 2.5 MD.

V případě vývoje bez generátoru kódu by vývojář musel zajistit 4 věci. Vytvoření Swagger souboru, vytvoření Frontend a Backend stubů a vytvoření Data Transfer Object (DTO). Vytvoření Swagger souboru by vývojáři trvalo půl MD. Vývoj Frontend stubu by vývojáři trval 1 MD a vývoj Backend stubu také 1 MD. Tvorba DTO entit by vývojáři trvala půl MD. Celkem by vytvoření stejného výstupu jako při použití generátoru trvalo 5 MD viz Tabulka 1.1.

	S generátorem	Bez generátoru
Analýza	2 MD	2 MD
Modelace	0.5 MD	—
Tvorba Swagger souboru	—	0.5 MD
Frontend stub	—	1 MD
Backend stub	—	1 MD
DTO	—	0.5 MD
Strávený čas na vypracování	2.5 MD	5 MD

Tabulka 1.1: Porovnání časů

Kapitola 2

Rešerše

V této části práce je důležité, aby byly vybrány potřebné nástroje, se kterými bude potřeba pracovat. Je potřeba vybrat mezi nejlepšími nabídkami na trhu v oblastech:

- Generátory
- Vývojové nástroje
- Modelovací nástroje
- Databáze

2.1 Generátory

Generátory se už delší dobu vyskytují v prostředích zabývajícími se softwarovým vývojem. Ulehčují práci a čas strávený na vývoji produktu, zároveň také snižují chybovost napsaného kódu a tím i urychlují dodání finálního produktu zákazníkovi. Pro generování existují všechny možné nástroje, knihovny či frameworky.[3]

2.1.1 Laravel Generator

Jedná se o framework generující jednoduché CRUD (zkratka pro create, read, update, delete) funkcionality pro aplikace. Lze využít pouze na jazyk PHP.[4]

2.1.2 Eclipse Modeling Framework

Zkratkou EMF je framework zabudovaný v integrovaném vývojovém prostředí Eclipse, který se stará o generování kódu pro model-driven aplikace. Obsahuje hodně nástrojů a APIs pro vytváření, editaci a práci s modely jako například vlastní grafický editor. Také podporuje práci s nejpoužívanějšími modelovacími jazyky jako je UML nebo XML.[5]

2.1.3 Selenium

Selenium je knihovna pro generování různých testovacích scénářů a automatických testů pro webové aplikace. Je používán k udržení kvality a zmenšení chybovosti aplikací. Díky němu je možné psát různé uživatelské testy, které dokáží interagovat s aplikací na úrovni běžného uživatele a tím aplikaci otestovat před nasazením do produkčního prostředí.[6]

2.1.4 Yeoman

Yeoman je nástroj pro generaci webových aplikací, který zajišťuje například generování běžných funkcionalit jako je autentizace nebo routing. Je v něm možné také používání populárních frameworků jako React, Angular nebo Vue.js. Jedná se o běžný nástroj používaný během vývojů webových aplikací.[7]

2.1.5 Javadoc

Součástí Java Development Kitu je nástroj javadoc, který zajišťuje generaci dokumentace kódu. Díky speciálním komentářům použitými ve zdrojovém

kódu aplikace napsané v Javě dokážeme vygenerovat dokumentaci kódu v HTML, PDF nebo XML formátu popisující vlastnosti, atributy, návratové hodnoty funkcí, či funkcionalitu tříd.[8]

2.1.6 OpenAPI Generator

Jedním z nástrojů pro generování kódu je také OpenAPI Generator. Jedná se o nástroj pro automatickou generaci kódu na základě OpenAPI specifikace (neboli Swaggeru) umožňující generování kódu v mnoha různých jazycích a frameworcích. Pomocí něj dochází k urychlení vývoje API, zároveň jsou také poskytnuty šablony pro tvorbu dokumentací, testů a dalších nástrojů pro správu API. Generátor také nabízí možnost různých konfigurací a rozšiřitelností, což vede k jednoduchým úpravám pro potřeby uživatelů.[9]

2.2 Požadavky na vývoj

V dnešní době je na trhu vysoká dostupnost všelijakých jazyků, knihoven či frameworků pro vývoj různých softwarů. Naším cílem je vybrat si ze všech možných nabízených možností tu, která by nejlépe zajišťovala následující parametry.

2.2.1 Rychlost běhu programu

Aplikace by měla být schopna provádět výpočty velmi rychle, chceme se tudíž vyhnout redundantním operacím, které by mohly běžet na pozadí.

2.2.2 Rychlost vývoje

Vývoj by neměl trvat příliš dlouhý časový úsek. Je důležité, aby vývojář, který nástroj využívá, nemusel řešit samozřejmé věci, a aby nástroj řešil určité úseky vývoje za vývojáře. Tím je myšlen také počet řádků kódu nezbytný k vytvoření určité funkcionality.

2.2.3 Vývojářská přívětivost

V dnešní době je důležité vybrat nástroj, se kterým se lehce pracuje a místo toho, aby vývojářům přidělával práci, ji ubírá.

2.3 Vývojové nástroje

Nyní, když jsou stanovené požadavky na vývojové nástroje, je nutné porovnat nabízené nástroje mezi sebou a zjistit, které by pro účely aplikace byly nejideálnější.

2.3.1 C#

C# je objektově orientovaný jazyk vyvinutý firmou Microsoft. Je to také součást .NET frameworku, což nabízí přístup k mnoha knihovnám pro vývoj různých webových nebo mobilních aplikací. Jedná se o silně typovaný jazyk a díky vestavěnému Garbage Collectoru se narozdíl od jeho nižších C verzí stará sám o správu paměti. Použitelnost C# se však pohybuje hlavně okolo operačních systémů firmy Microsoft.[10]

2.3.2 Java

Java je narozdíl od C# multiplatformní díky tvorbě bytekódu, který může být spustitelný na jakémkoliv zařízení s JVM (Java Virtual Machine). Má velkou komunitní základnu a k tomu se pojí mnoho různých knihoven a frameworků, které vysoce ulehčují vývoj aplikací. Programy napsané v Javě obsahují většinou plno řádků kódu pro správný běh programu, což může navyšovat čas potřebný na vývoj aplikace.[11]

2.3.3 Spring Boot

Jedná se o framework, který je dnes nepostradatelnou součástí jazyku Java. Zajišťuje jednoduchou tvorbu a udržování webových aplikací a přináší nové techniky jakými jsou například Aspektově-orientované programování, Dependency Injection nebo Plain Old Java Object. Spring Boot je nástroj stavící své základy na Springu. Pomáhá vytvářet samostatně stojící aplikace, které v sobě mají zabudovaný servlet kontejner (Tomcat). Všechna funkcionalita aplikace se řídí přes nastavování konfigurací. Spring Boot je velmi výhodné zvolit, pokud chceme při vývoji použít REST API, jelikož s ním umí dobře pracovat. Vytváření produkční verze je velmi jednoduché a zabere méně času. Je zde také nabídka vestavěného databázového systému H2.[12]

2.3.4 Django

Django je populární webový framework pro jednoduché a efektivní vytváření webových aplikací. Zajišťuje interakci s databází pomocí jazyku Python namísto klasických SQL dotazů. Nabízí velké množství jedinečných vlastností a přehlednou dokumentaci, díky čemuž se stal jedním z nejoblíbenějších frameworků pro tvorbu webových aplikací za pomocí velice jednoduše použitelného a pochopitelného jazyku Python.[13]

2.4 Modelovací nástroje

V této části budou různé možnosti pro modelovací nástroj. Je nutné vybrat takový, který je uživatelsky přívětivý, má jednoduché napojení na databázi a je oblíbený mezi analytiky v profesionálním prostředí.

2.4.1 Enterprise Architect

Jedná se o nástroj pro zprostředkování Model-driven developmentu velkopodnikového stylu, který je jednoduchý na použití, podporuje týmovou kolaboraci na projektech a je vhodný pro velké a komplexní projekty. Nabízí plno možností modelovacích jazyků, v případě naší aplikace je nejdůležitější UML. Pro

většinu velkých společností je to první volba pro tvorbu modelů a náčrtu architektury projektu ještě před počátkem vývoje.[14]

2.4.2 Rational Software Architect

Rational Software Architect je další nástroj pro generování modelů a pro podporu MDD založený na principu Eclipse Modelling Frameworku. Jde o integrované vývojové prostředí vyvinuté firmou IBM pro softwarové vývojáře a architekty k vytváření různých modelů pomocí UML. Nabízena je i podpora generování kódu a také podpora různých programovacích jazyků.[15]

2.5 Databáze

V této části bude proveden výčet možných příkladů databází, které budou použity jako propojení mezi aplikací a nástrojem pro modelování.

2.5.1 MySQL

MySQL je open-source databáze s velkou uživatelskou základnou, která nabízí plno knihoven a frameworků. Je zde také vysoká podpora bezpečnosti díky šifrování dat, uživatelské autentifikaci nebo kontrole přístupu, což zajišťuje ochranu uživatelských dat. MySQL je také lehce použitelná, nabízí výbornou škálovatelnost a multiplatformní použití.[16]

2.5.2 PostgreSQL

PostgreSQL je stejně jako MySQL open-source databáze s velkou uživatelskou bází, která nabízí tvrobu vlastních datových typů, funkcí či operátorů. Podporuje také například vyhledávání v celém textu nebo použití JSON datových formátů, zároveň podporuje použití komplexních dotazů nebo optimalizaci, což zaručuje spolehlivou práci nad velkým objemem dat.[17]

2.6 Výsledky výběru technologií

Na základě výčtu možností v různých kategoriích je potřeba vybrat ty možnosti, které budou nejlépe vyhovovat při vývoji aplikace. Na základě funkcionality jednotlivých okruhů bylo uděleno hodnocení od 1 do 5 a položky s nejvíce body byly vybrány.

2.6.1 Generátor: OpenAPI Generator

Ze všech nabízených možností generátorů byl zvolen OpenAPI Generator. To hlavně z důvodu možnosti generování stubů nebo knihoven v mnoha různých jazycích a frameworcích a také vždy dodané, graficky zobrazitelné dokumentaci, která je základem pro generování stubů. Důležité bylo také rozhodnutí ve vlastním přizpůsobení generátoru viz Tabulka 2.1.

	Laravel	EMF	Selenium	Yeoman	Javadoc	OpenAPI
Generování pro více jazyků	2	2	4	4	1	5
Možné výstupy generování	3	2	2	3	2	4
Nastavitelnost přizpůsobení	3	3	2	2	2	4
Výsledky hodnocení	8	7	8	9	5	13

Tabulka 2.1: Porovnání generátorů

2.6.2 Nástroj pro vývoj: Spring Boot

Spring Boot se stal volbou číslo jedna, jelikož nejlépe splňoval stanovené požadavky na vývoj aplikace. Je velice uživatelsky přívětivý díky anotacím, které řeší plno věcí za vývojáře a tím narozdíl od C# a Javy šetří počet úkonů a nastavení, které musí vývojář provést viz Tabulka 4.2.

	Java	C#	Spring Boot	Django
Rychlost běhu	5	5	4	3
Rychlost vývoje	3	3	5	5
Uživatelská přívětivost	3	2	4	4
Výsledky hodnocení	11	10	13	12

Tabulka 2.2: Porovnání vývojových nástrojů

2.6.3 Modelovací nástroj: Enterprise Architect

Enterprise Architect je využívaný a oblíbený napříč mnoha společnostmi v analytickém či architektonickém odvětví softwarového inženýrství díky jeho nabízeným funkcionalitám a také jeho finanční dostupnosti. Navíc je velice jednoduchý na pochopení a použití viz Tabulka 4.3.

	EA	RSA
Nabídka funkcí	5	3
Uživatelská přívětivost	4	3
Dostupnost na trhu	5	4
Výsledky hodnocení	14	10

Tabulka 2.3: Porovnání modelovacích nástrojů

2.6.4 Databáze: MySQL

MySQL nenabízí takovou funkcionalitu jako PostgreSQL, ale je více populární a výkonný. Tato databáze bude použita jako propojení mezi aplikací a nástrojem pro zprostředkování Model-driven developmentu viz Tabulka 4.4.

	MySQL	PostgreSQL
Nabídka funkcí	3	5
Popularita	5	4
Výkon	5	4
Ekosystém	5	3
Výsledky hodnocení	18	16

Tabulka 2.4: Porovnání databází

Kapitola 3

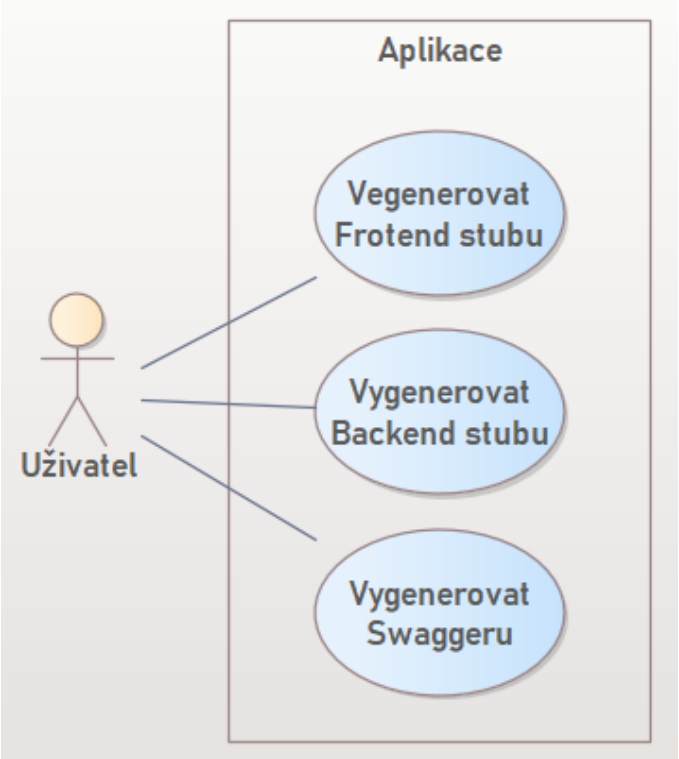
Analýza požadavků

V této části je hlavním cílem analyzovat všechny požadavky této práce.

3.1 Funkční požadavky

Hlavními funkčními požadavky práce je vygenerování stubů z analytických modelů. Na základě volby generátoru učiněné v 2.6.1 bude aplikace využívat OpenAPI Generator, který jako vstup zpracovává OpenAPI specifikaci neboli Swagger. Primárním výstupem aplikace je soubor Swagger, na jehož základě vygenerujeme potřebné stuby. Vstupem aplikace jsou modely vytvořené v Enterprise Architect na základě výběru v 2.6.3. Funkční požadavky jsou vyobrazeny v Obrázku 3.1. Výčet funkčních požadavků:

- FR1 **Vygenerování stubů Frontendu a Backendu v různých nástrojích:** Systém umožní uživateli vygenerovat Frontend a Backend stuby na základě jím zadaných parametrů a vstupního modelu.
- FR2 **Vygenerování Swaggeru pomocí aplikace:** Systém umožní uživateli vygenerovat Swagger na základě jím vytvořeného modelu.



Obrázek 3.1: Případy užití aplikace

3.1.1 Stub

Stub je označení pro část kódu s unikátní funkcionalitou, který ještě doposud nebyl plně implementován. Takové označení se používá pro vývojáře i testery pro simulaci určitého chování podsystémů nebo uzavření částí pro testování.[20]

3.1.2 Swagger

Dnes také známý jako OpenAPI je specifikace pro popis REST API, která je použita pro automatickou generaci dokumentace, klientské knihovny a kódu pro validaci. Specifikace je v tomto případě jedním souborem obsahující informace o endpointech, metodách a parametrech poskytované API. Jedná se nejčastěji o soubory typu JSON nebo YAML, které jsou dobře hierarchicky a strukturálně uspořádané a tudíž jednoduché na sestavení. OpenAPI je vstupem pro OpenAPI Generator.[19]

3.1.3 REST API

Celým názvem Representational State Transfer je architektonický styl pro webové API. Slouží k přístupu ke zdrojům, objektům, datům nebo servisám na webu skrz URL (Uniform Resource Locator) a pomocí různých HTTP (Hypertext Transfer Protocol) metod s nimi provádět operace. Metodami HTTP jsou:[18]

- GET - metoda pro získávání dat
- POST - metoda pro vytváření nových dat
- PUT - metoda pro plnou aktualizaci již existujících dat
- PATCH - metoda pro částečnou aktualizaci již existujících dat (v některých případech existuje jen metoda PUT)
- DELETE - metoda pro mazání dat

3.1.4 Struktura Swaggerů

Swaggery mají několik podčástí, které obsahují důležité informace o API. Každá z těchto podčástí obsahuje povinné i nepovinné položky.

OpenApi

První položka každého Swagger souboru, která dává vědět, o jakou verzi OpenAPI se jedná. V našem případě budeme používat verzi 3.0.3. Jedná se o jednu z nejnovějších verzí OpenApi, která je kompatibilní s nejnovější verzí OpenAPI Generatoru.[21]

Info

Tato část Swagger souboru nabízí obecné informace týkající se API jako je hlavní nadpis, popis nebo verze.[21]

Povinné položky:

- Title - jde o hlavní nadpis API
- Version - vypovídá o tom, o jakou verzi API se jedná

Nepovinné položky:

- Description - popis API
- Contact - kontakt na vývojáře

Servers

Specifikuje URL adresu serverů API. Přístup k jeho specifickým endpointům je potom relativní vzhledem k URL příslušných serverů. Jedná se o nepovinný údaj.[22]

Tags

Tagy pod sebe sjednocují HTTP metody přistupující k datům na stejných endpointech. Každý tag patří ke specifickému endpointu a je popsán svým jménem (na který je později odkazováno v dalších částech Swaggeru) a popisem.[21]

Paths

V Paths se identifikují všechny HTTP metody sjednocené pod jednotlivé endpointy. Každá cesta začíná cestou k endpointu. Pod cestou se nachází všechny metody týkající se samotného endpointu a k nim vždy popisy a tag, ke kterému patří. Ke každé cestě taky nesmí chybět všechny možné odpovědi a požadavky, které se vážou k dané metodě.[23]

Components - Schemas

Sekce components obsahuje hlavní část schemas, která nese všechny informace o objektech vyskytujících se v API. Každý objekt je popsán všemi atributy, které nese, které jsou povinné či nepovinné při jeho inicializaci. Také může obsahovat odkaz na další objekty vyskytující se v sekci Schemas.

3.1.5 Vstupní model

Primární vstupem aplikace je model vytvořený v Enterprise Architect. Pro uživatele jsou možné dva typy modelace. Prvním typem je Logický datový model (LDM), který popisuje pouze datové schéma. Druhým typem je modelace API, která popisuje chování rozhraní. K tomu se váže definice metod, což zahrnuje jejich požadavky a odpovědi.

Schéma

Schéma je složeno z hierarchie entit, které uživatel použije ve svoji vygenerované aplikaci. Součástí schématu jsou pouze třídy a primitivní typy neboli jejich atributy spojené vazbou k třídám.

Rozhraní

Základem každého API je poskytované rozhraní neboli interface. Na interface potom navazují všechny další metody, které rozhraní poskytuje.

Metody

Metody jsou přímo navázané vazbou na interface a vážou se k určitému endpointu. Mohou být typu GET, POST, PUT, PATCH nebo DELETE.

Požadavky a odpovědi

Každá metoda potom obsahuje požadavky (requests) a odpovědi (responses). Požadavky a odpovědi na sebe mají potom navázané jednotlivé entity ze schématu.

3.2 Nefunkční požadavky

Mezi nefunkční požadavky se řadí všechny požadavky, které nemají přímý vliv na funkcionalitu aplikace. Výčet nefunkčních požadavků:

- NFR1 **Grafické zobrazení Swaggeru:** Uživatel bude schopen OpenAPI specifikace zobrazovat pomocí přehledného grafického nástroje SwaggerUI.
- NFR2 **Korekce Swaggeru:** Uživatel bude schopen zjistit, zda je Swagger korektně vygenerovaný či ne. Uživatel bude mít možnost tyto chyby zachytit a opravit pomocí Swagger Editoru.
- NFR3 **Tvorba modelu:** Uživatel bude vytvářet model v nástroji Enterprise Architect.
- NFR4 **Použitá databáze:** Aplikace i modelovací nástroj budou napojeny na MySQL server.
- NFR5 **Vývojové nástroje:** Aplikace bude vyvíjena v jazyku Java společně s použitím Spring Boot frameworku.

3.2.1 Swagger Editor

Hlavní nástroj pro editaci Swaggerů. Jedná se o rozšíření nástroje SwaggerUI společně s možností upravovat obsah Swaggeru. Swagger Editor také upozorňuje na to, jestli jsou některé prvky obsahu špatně použité, nebo zda některé povinné položky nechybí.[28]

3.2.2 SwaggerUI

SwaggerUI je plně dostupný nástroj pro vizualizaci Swagger souborů.[19]

Kapitola 4

Návrh

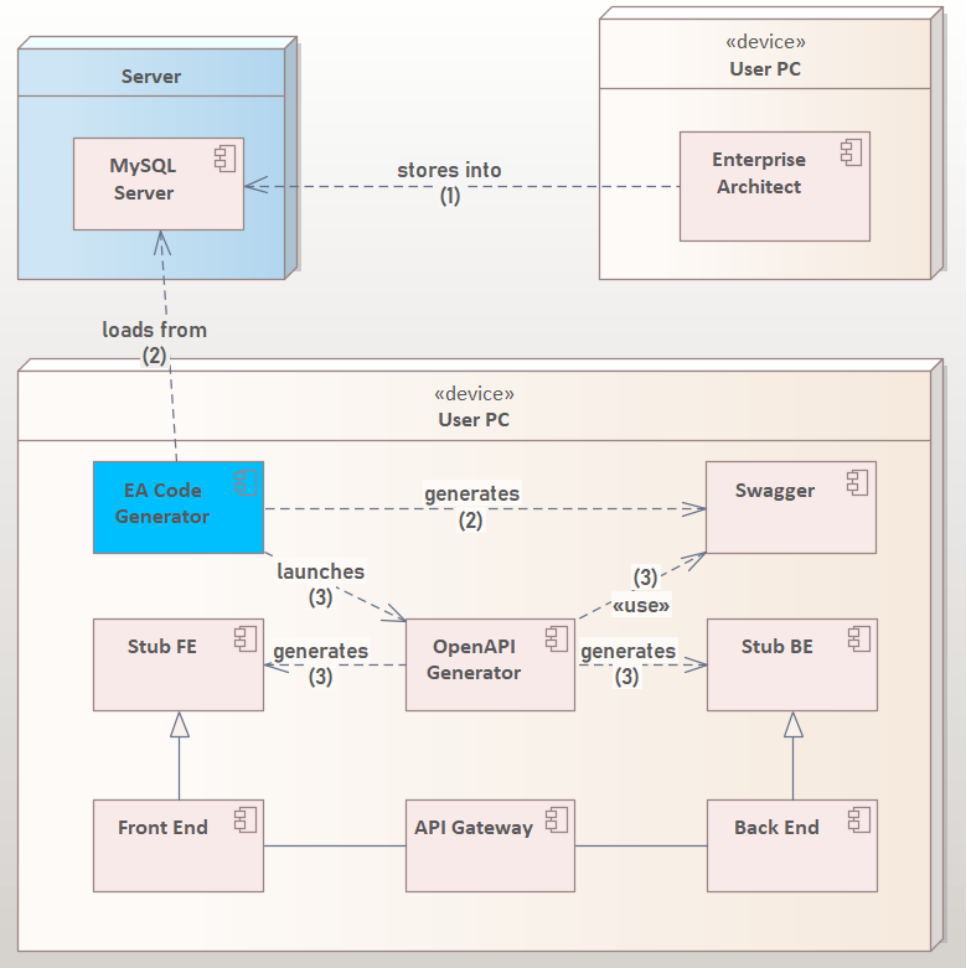
V návrhové části je hlavní představit architektonický návrh aplikace, řešení pomocí diagramu tříd a chování aplikace pomocí Sekvenčních diagramů.

4.1 Architektonický návrh aplikace

V rešerši jsme si stanovili nástroje, které budou při práci využity a nyní je důležité stanovit interakci mezi nimi. Začátkem procesu aplikace je vytvoření modelu v Enterprise Architect, který je napojený na MySQL databázi. Z databáze poté aplikace čte a generuje Swagger. Zároveň také spouští OpenAPI Generator, který z vytvořeného Swaggeru generuje stuby.

4.2 Postup použití aplikace

Následující kroky použití aplikace popsané níže se vážou k Obrázku 4.1.



Obrázek 4.1: Architektura aplikace

4.2.1 Modelování

(1) Prvním krokem, který uživatel musí učinit, jelikož se jedná o Model driven development, je vymodelovat si svoji službu. Model by měl obsahovat pouze schéma v případě LDM či v případě API rozhraní a na něj napojené dostupné metody, jakými jsou GET, POST, PUT, PATCH a DELETE. S těmito operacemi se vážou různé odpovědi rozdělené podle HTTP status kódů a požadavky na ně. Model je průběžně ukládán do MySQL databáze.

4.2.2 Vygenerování Swaggerů

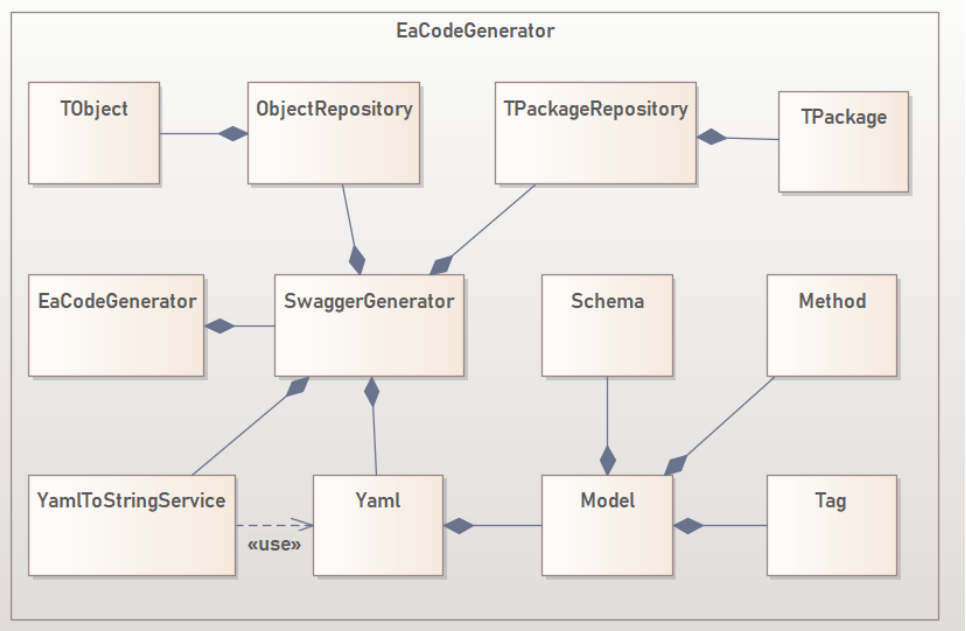
(2) Poté, co je model vytvořen, se aplikace po spuštění připojí na databázi a podle zadaných parametrů v konfiguraci, jimiž je v případě LDM cesta k balíčku obsahující schéma a v případě API název rozhraní, načte data. Na základě načtených dat vygeneruje Swagger.

4.2.3 Použití OpenAPI Generátoru

(3) Po vygenerování Swaggerů použije aplikaci na vygenerovanou specifikaci OpenAPI Generator, který na základě její definice a definovaných parametrů v konfiguraci, jimiž jsou jazyk, framework či knihovna pro klientskou a serverovou stranu, vygeneruje Frontend a Backend stuby. Tyto stuby uživatel použije a podle svých potřeb upraví či rozšíří ve své službě.[9]

4.3 Model aplikace

Hlavní třídou aplikace je třída reprezentující generátor, která po spuštění načítá data z databáze pomocí Repository. Objekty, které načítá, jsou entity typu TObject a TPackage. Ostatní objekty, se kterými pracujeme z Enterprise Architect, jsou získávány z těchto objektů, mezi ně patří například konektory. Načtená data zpracovává a ukládá do třídy Yaml, který má Model se všemi potřebnými částmi jako jsou například Tagy, Schéma nebo Metody. Po dokončení zpracování vstupu je Yaml použit pro převod do Stringu pomocí YamlToStringService, která data uložená v modelu převede na String, který uloží do souboru viz Obrázek 4.2. Datový model aplikace je namodelovaný zjednodušeně a neobsahuje všechny možné třídy aplikace pro lepší přehlednost a pochopení.

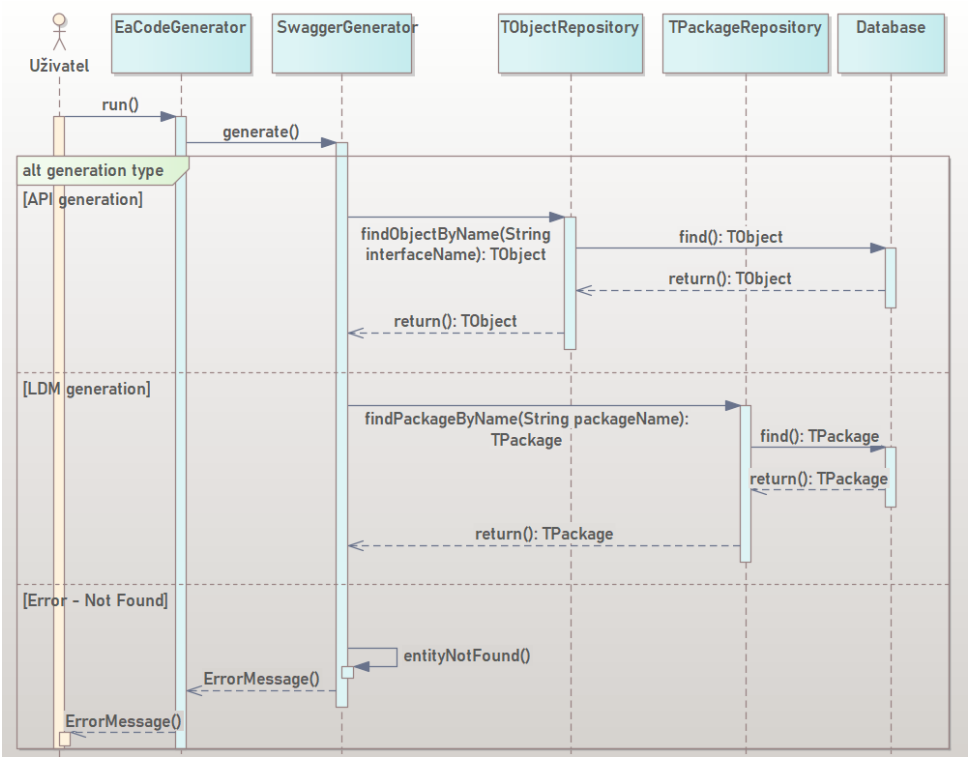


Obrázek 4.2: Package diagram aplikace

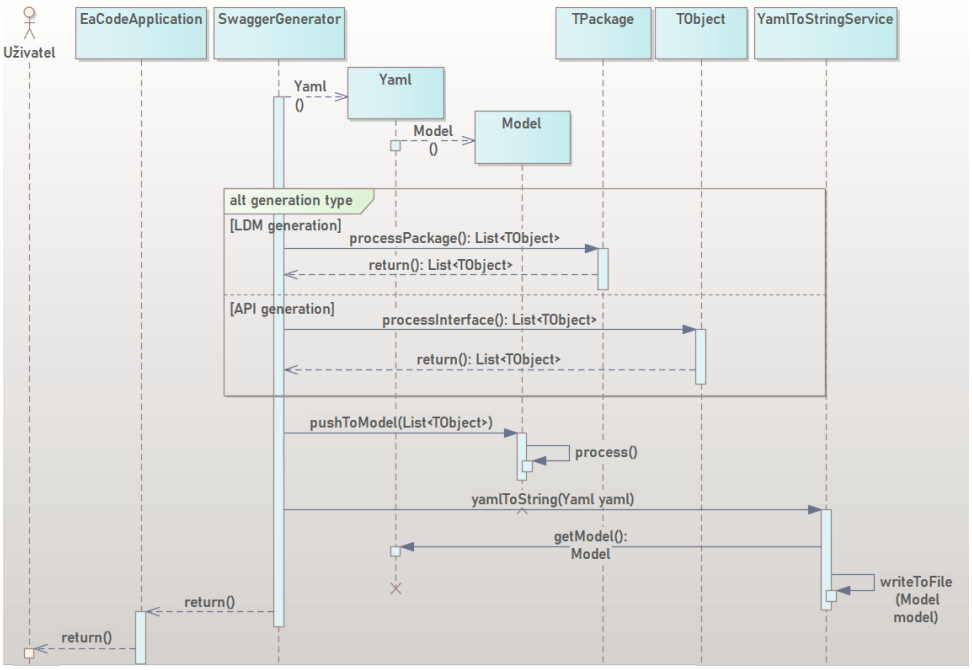
4.4 Chování Aplikace

Nejprve dojde ke spuštění aplikace. Ta začne generování na třídě SwaggerGenerator. Jako první nastane načtení dat v podobě TObject nebo TPackage objektů na základě zadaných vstupních parametrů viz Obrázek 4.3.

Vytvoří se objekt třídy Yaml, ve kterém bude Model naší aplikace z Enterprise Architect. Ten obsahuje všechny možné podčásti a data týkající se našeho Swagger souboru. Pokud bude probíhat generování LDM, projdou se všechny balíčky a soubory v zadaném balíčku, který byl vstupním parametrem. Pokud bude probíhat generování API, projdou se rekurzivně všechny objekty navázané na rozhraní. Všechna tato prošlá data jsou potom uložena v Modelu v podobě různých objektů. Potom, co jsou všechna data prošlá, nastává vznik Swagger souboru, o který se stará třída YamlToStringService, která z objektu Yaml získá Model a pomocí všech zpracovaných dat vygeneruje Swagger soubor viz Obrázek 4.4.



Obrázek 4.3: Sekvenční diagram - Načítání dat



Obrázek 4.4: Sekvenční diagram - Zpracování a výstup dat

Kapitola 5

Použití aplikace

Na základě návrhového popisu použití aplikace v části 4.2 se v této části rozvede detailnější použití aplikace z uživatelského hlediska.

5.1 Tvorba Modelu

První krok pro použití aplikace je vytvoření modelu v Enterprise Architect. Model je vytvořen v balíčku se jménem rozhraní, který v sobě obsahuje další balíčky s číslem verze API. Např. `Library_System/1.0.4/"model"` Pokud však uživatel chce generovat pouze LDM, tak stačí model uložit pouze do určitého balíčku. Pro správné fungování aplikace je důležité, aby uživatel nastavoval objektům v modelu tzv. Stereotypy. Jejich pojmenování musí být přesné, aby aplikace poznala, o jaký typ objektu se jedná. Pro vytvoření diagramů používá uživatel vždy diagram typu UML Structural - Class.

5.1.1 Tvorba Child Diagramu

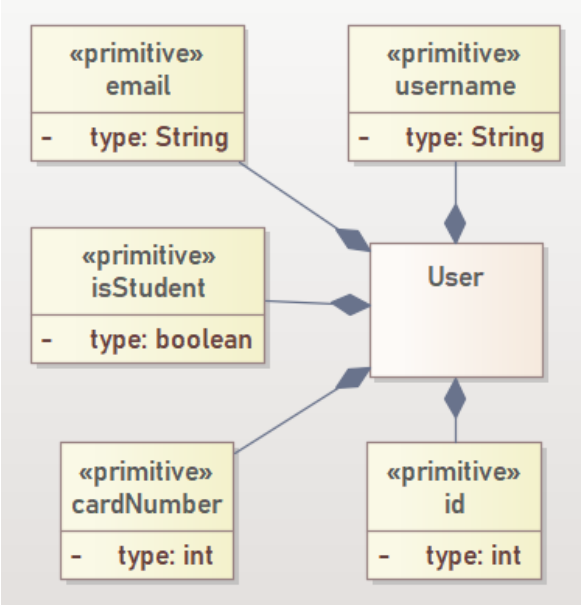
Na začátek je důležité vysvětlit, jak zabalit jednotlivé diagramy pod objekty kvůli větší přehlednosti a strukturalizaci. Značka, která říká, že daný objekt obsahuje v sobě diagram, se vyznačuje dvěma elipsami spojenýma čarou vpravo dole viz Obrázek 5.1. Přiřazení tzv. child diagramu objektu je možné

pomocí kliknutí pravého tlačítka myši na daný objekt, přejít na sekci "New Child Diagram"a zvolit Add Diagram. Tímto krokem se diagram přímo vytvoří a zároveň propojí s daným objektem. Přejití na diagram je poté možné pomocí dvojího kliku myši na daný objekt obsahující diagram.

Obrázek 5.1: Třída obsahující Child Diagram

5.1.2 Schéma

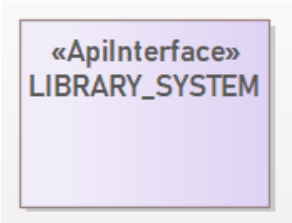
V modelaci schématu nebudou potřeba žádné Stereotypy. Všechny třídy budou vytvořeny pomocí objektu typu Class a všechny jejich atributy budou objekty typu Primitive, které budou mít napřímo jeden atribut. Atribut se jménem “type” s typem podle toho, o jaký typ atributu se jedná. Atributy budou na třídy přímo navázané pomocí vazby Compose viz Obrázek 5.2. Závislost tříd je vytvořená také pomocí vazby Compose. Pokud chcete vytvořit oboustrannou závislost, použijte vazbu Compose dvakrát jedním i druhým směrem.



Obrázek 5.2: Schéma

5.1.3 Interface

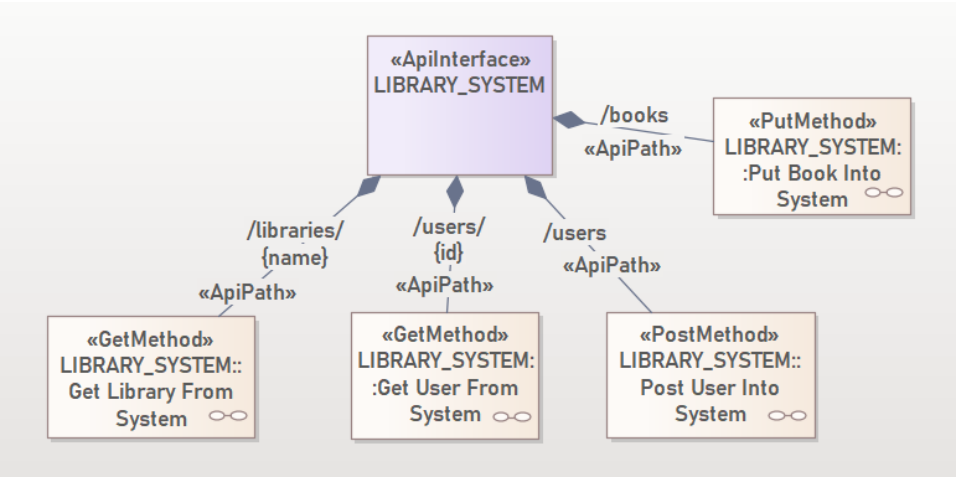
Interface je základ, pokud uživatel usiluje o generování API. Interface je objekt typu Interface. Pojmenování rozhraní a jeho popis budou promítnuty do nadpisu a popisu výsledného Swaggeru. Důležité je označit Interface Stereotypem “ApiInterface” viz Obrázek 5.3.



Obrázek 5.3: Interface

5.1.4 Metody

Metody mohou být pěti druhů a to GET, PUT, POST, DELETE nebo PATCH. Podle toho, jaký druh metody chceme použít, určíme objektu Stereotyp. Například v případě metody typu GET bychom použili Stereotyp “GetMethod”. Všechny metody jsou objekty typu Class a jsou na Interface navázány vazbou Compose, která má Stereotyp “ApiPath” a je pojmenovaná podle URL cesty endpointu metody viz Obrázek 5.4.

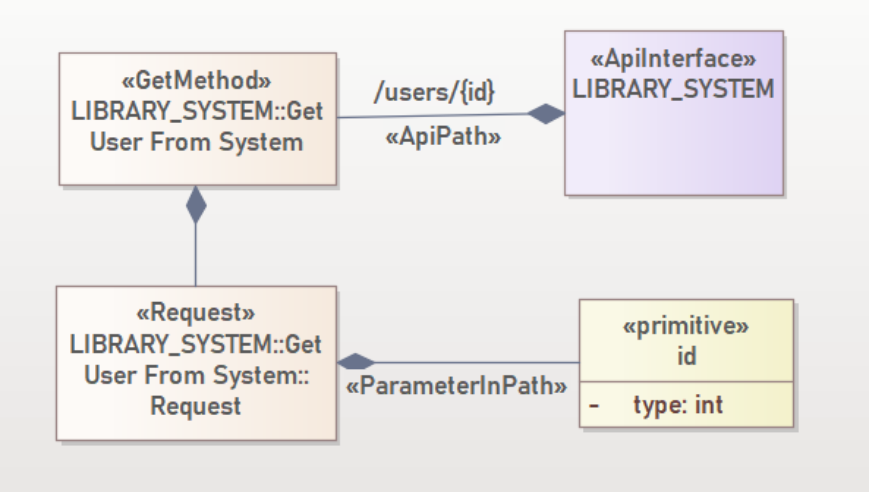


Obrázek 5.4: Metody

5.1.5 Request

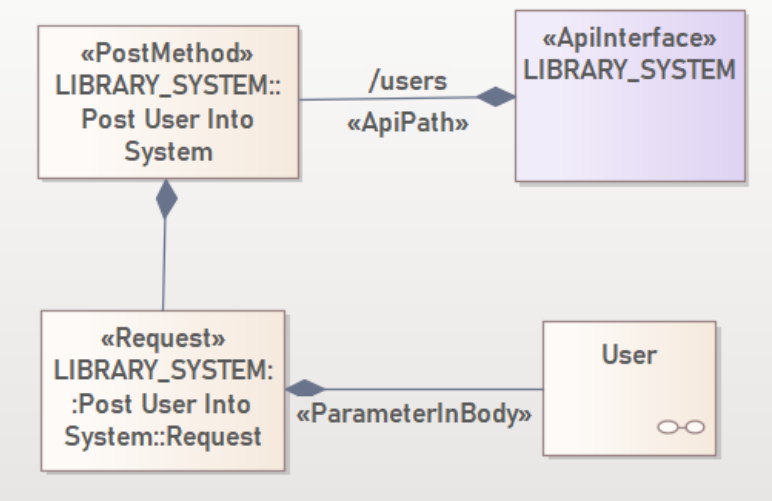
Na request neboli požadavek jsou navázány vstupy vytvořené metody, které je potřeba zadat. Všechny vstupy navazujeme na request přímo a to pomocí vazby Compose, která má vždy Stereotyp podle toho, o jaký druh vstupu se jedná. Existují tři druhy Stereotypů a tedy vstupů, které uživatel může použít:

1. Path Variable - jde o vstup primitivního typu, který slouží jako argument v endpointu. Například měli bychom metodu Get na endpoint `/users/{id}`, na metodu musí být navázán objekt typu Primitive, jako bylo použito ve schématu v 5.1.2. Pokud použijeme endpoint, který požaduje parametr a nezadáme ho do modelu, bude se jednat o neplatný model. Vazba, která půjde z primitivního typu do requestu musí mít Stereotyp ParameterInPath a být typu Compose viz Obrázek 5.5.



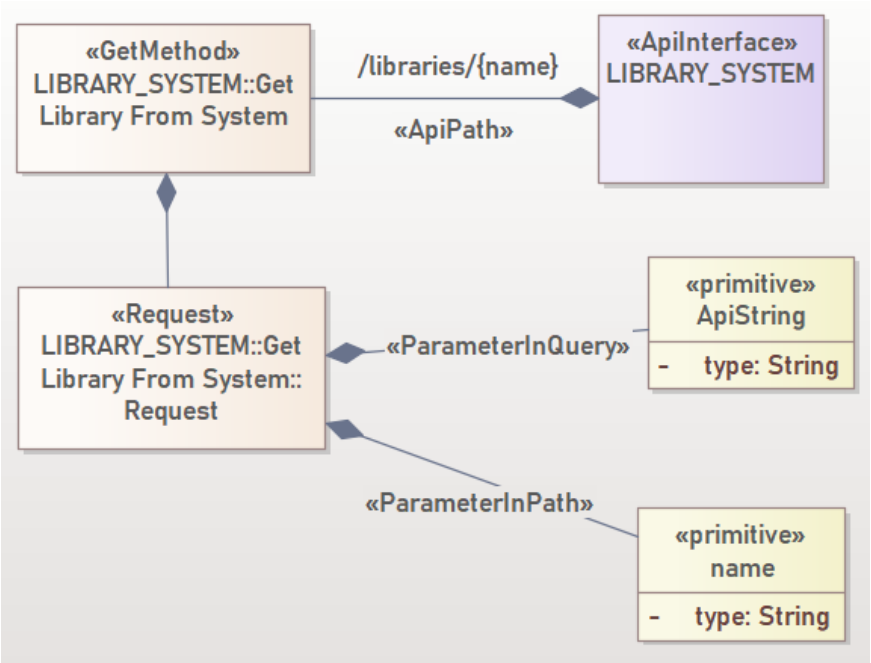
Obrázek 5.5: Path Variable

2. Request Body - jde o vstup objektového typu. Pravděpodobně se jedná o třídu, kterou už máme vytvořenou ve schématu. Nevytvářejte novou třídu, jinak by mohla nastat duplikace tříd. Nakopírujte třídu z vašeho schématu jako link. Vazba, která jde z objektu do requestu, musí mít Stereotyp ParameterInBody a být typu Compose viz Obrázek 5.6.



Obrázek 5.6: Request Body

3. Query Parameter - jde o typ, který je užitečný jako parametr v SQL dotazech. Vazba, která jde z dotazového parametru do requestu, musí mít Stereotyp ParameterInQuery a být typu Compose viz Obrázek 5.7.

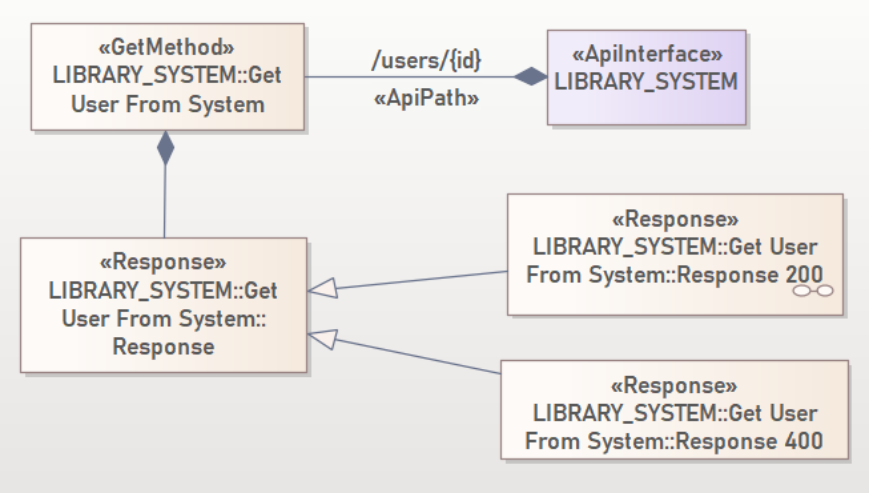


Obrázek 5.7: Query Parameter

5.1.6 Response

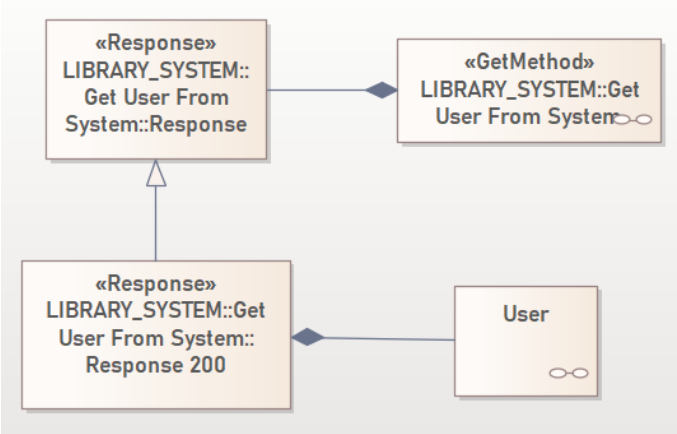
Response neboli také jinak nazýváno odpověď. Může nastat více různých odpovědí s různými úspěchy a neúspěchy nabývajících různých HTTP status kódů, proto je nutné vytvořit rodičovskou odpověď typu Class, kterou pojmenujeme Response. Od té pak budou dědit všechny odpovědi, které mohou nastat v metodě pomocí vazby Generalize. Každá rodičovská Response by měla obsahovat alespoň Response 200 a Response 400 viz Obrázek 5.8. Aplikace je schopna zpracovat Response kódu 200, 204 a kódů 400-405.

Jak rodičovská třída, tak dědicové této třídy, musí obsahovat Stereotyp Response. Podle toho, jaký HTTP status kód vrací response, pojmenujeme danou třídu. Například pro úspěch bychom třídu dědící ze třídy Response pojmenovali Response 200.



Obrázek 5.8: Obecný Response

Na odpovědi, které vracejí známé chybové hlášky, není potřeba navazovat cokoliv dalšího. Aplikace sama ví, jaké kódy vracejí jaké hlášky. Na odpovědi, které jsou úspěšné a vracejí jeden nebo více různých objektů, je potřeba objekty navázat. To pomocí nalinkování objektu ze schématu. Navázat je potřeba objekty na jednotlivé odpovědi vazbou typu Compose viz Obrázek 5.9.



Obrázek 5.9: Response 200

5.2 Databázový server

Výše vytvořený model je potřeba mít napojený na nastavený MySQL server, který si uživatel musí nainstalovat a napojit na svůj model v Enterprise Architect. K databázi bude připojena i aplikace.[26]

5.3 Použití aplikace

Pro použití aplikace je nutné nastavit parametry v konfiguraci, podle kterých aplikace pozná, zda se jedná o LDM nebo o API a další parametry. Vše se nastavuje v souboru `src/main/resources/application.properties`.

5.3.1 Databáze

Pro nastavení databáze je nutné nastavit parametry sekce spring.datasource. Databázové properties:

- spring.datasource.url - URL MySQL serveru
- spring.datasource.username - přihlašovací jméno k MySQL serveru

- spring.datasource.password - přihlašovací heslo k MySQL serveru

5.3.2 Typ generování

- API - pro typ generování API nastavíme proměnné sekce ea.interface. Mezi ně patří jméno rozhraní a verze balíčku, ve kterém se nachází tak, jak je popsáno v 5.1. API properties:
  - ea.interface.name - název rozhraní
  - ea.interface.version.main - hlavní verze balíčku, ve kterém se nachází model
- LDM - pro typ generování LDM je nutné nastavit pouze absolutní cestu balíčku, ve kterém se nachází vymodelované schéma a také zakomentovat všechny ea.interface parametry. Pro generování typu LDM existují také další properties, které lze nastavit:
  - ea.ldm.package - absolutní cesta k balíčku se schématem
  - ea.ldm.version.main - hlavní verze
  - ea.ldm.version.minor - vedlejší verze
  - ea.ldm.description - popis

5.3.3 Parametry výstupu

Nyní už je potřeba nastavit pouze parametry výstupu generování stubů. Podle toho, v jakém jazyce či frameworku chcete mít vygenerované Frontend a Backend stuby, nastavte proměnné typu generation.language. Všechny možnosti naleznete na stránkách OpenAPI generátoru[25]. Properties pro generování:

- generation.language.frontend - název nástroje pro generování Frontendu
- generation.language.backend - název nástroje pro generování Backendu

Před spuštěním generování je také důležité si nainstalovat OpenAPI Generator. Pro jeho instalaci použijte instalační balíček npm. Návod na instalaci naleznete na github repositáři OpenAPI Generatoru[9]. Vygenerovaný Swagger najdete po dokončení generování ve složce export. Vygenerované stuby naleznete ve složce gen.

Kapitola 6

Implementace

V implementační části bude představena struktura aplikace a způsoby řešení různých problémů v průběhu implementace.

6.1 Spouštěcí soubor aplikace

Spuštění aplikace probíhá přes třídu EaCodeGeneratorApplication. Ta zpracovává všechny vstupní parametry a na základě nich i rozhoduje, zda proběhne generování typu LDM či API.

6.2 Entity z Enterprise Architect

Balíček s názvem metamodel obsahuje všechny potřebné třídy, které ztělesňují entity v databázi propojené s Enterprise Architect. Hlavními entitami, se kterými pracujeme v aplikaci, jsou třídy TObject a TPackage, díky kterým je možné zachytit počáteční body při generování typu LDM či API.

Objekty jsou mezi sebou propojené vazbami, které popisuje třída TConnector, která obsahuje různé atributy jako popis, směr vazby, kardinality na obou stranách nebo typ vazby. Poslední důležitou zmíněnou třídou je

TAttribute, která je stejná jako atributy u různých objektů v Enterprise Architect. Z informacích zjistěných pomocí těctho entit se aplikace rozhoduje, jakým způsobem vygeneruje Swagger soubor.

6.3 Repositories

Nepostradatelnou součástí každé backend architektury jsou DAO nebo Repositories. Ve svém projektu jsem se rozhodl pro implementaci CRUDRepositories, které se nachází v balíčku repository. Pomocí repositories získává aplikace z Enterprise Architect entity.

6.4 Konfigurační třídy

Konfigurace aplikace jsou určovány tím, jaký typ generování chce uživatel použít. Na základě konfiguračních tříd se aplikace rozhoduje, co je důležité vyhledat a zpracovat. Pokud použijeme typ generování API, tak jsou důležité pouze informace o jméně a verzi rozhraní API v modelu v Enterprise Architect.

Cheme-li použít typ generování LDM jsou pro konfiguraci důležité tyto parametry:

- jméno balíčku, ve kterém se náš model nachází
- ignorované balíčky
- hlavní verze
- vedlejší verze
- popis

6.5 Mapované objekty

Mapované objekty, které se opakovaně vyskytují ve Swagger souboru nebo v Enterprise Architect jsou v aplikaci řešené pomocí výčtových typů. Každý

výčtový typ jako například Stereotypy ve formě druhů metod obsahuje svůj přesný textový popis jak z Enterprise Architect, tedy vstupní, tak ve vygenerovaném Swagger souboru, tedy výstupní viz Obrázek 6.1.

```
3 usages
METHOD_GET( name: "GetMethod",  yaml: "get"),
2 usages
METHOD_POST( name: "PostMethod",  yaml: "post"),
2 usages
METHOD_PUT( name: "PutMethod",  yaml: "put"),
2 usages
METHOD_DELETE( name: "DeleteMethod",  yaml: "delete"),
2 usages
METHOD_PATCH( name: "PatchMethod",  yaml: "patch"),
```

Obrázek 6.1: Mapované objekty

6.6 Generování

Po zpracování načtených dat aplikací je spuštěno generování Swagger souboru. Aplikace vezme svá načtená data, která má uložená a struktualizovaná v objektech a převádí je do String proměnné. Aplikace také řeší použití tabulátorů a nových řádků pro správné vytvoření Swagger souboru.

Poté, co je Swagger soubor vytvořen, nastává generování stubů, které je spuštěno pomocí procesu a příkazů pro příkazovou řádku na základě uživatelského operačního systému.

Kapitola 7

Testování

7.1 Unit Testy

Pro správné fungování aplikace byly vytvořené Unit testy, které testovaly funkcionality na úrovni funkcí.

7.1.1 Test Coverage

Test coverage aplikace je přibližně 40%. Test coverage by mohla být vyšší, ale aplikace obsahuje plno metod, které se starají pouze o strukturalizaci a neprobíhá v nich žádná logika jako set a get metody tříd. Testované jsou hlavně třídy, které zajišťují správný výstup ze získaných dat z balíčku service nebo třída starající se o tvorbu Tagů, ve kterých probíhají složitější operace a jsou náchylnější na chyby viz Obrázek 7.1.

7.1.2 Testovací scénáře

Třída PackageNamePredicate má metodu, která se stará o porovnávání názvů balíčků s možností vypnutí nebo zapnutí ignorování velkých a malých písmen. Pro testovací třídu této třídy bylo vytvořeno několik různých scénářů:

Element	Class, %	Method, %	Line, %
cz	71% (35/49)	39% (209/533)	40% (584/1448)
cvut	71% (35/49)	39% (209/533)	40% (584/1448)
fel	71% (35/49)	39% (209/533)	40% (584/1448)
metamodel	0% (0/6)	0% (0/75)	0% (0/140)
generator	100% (2/2)	18% (9/48)	15% (54/354)
yaml	72% (16/22)	31% (81/254)	40% (208/515)
service	100% (1/1)	60% (38/63)	55% (99/179)
util	85% (12/14)	84% (58/69)	81% (147/181)
EaCodeGeneratorApp	100% (1/1)	90% (9/10)	94% (51/54)
repository	100% (0/0)	100% (0/0)	100% (0/0)
configuration	100% (3/3)	100% (14/14)	100% (25/25)

Obrázek 7.1: Test Coverage Aplikace

- Porovnání dvou stejných textových řetězců - metoda vrátí true.
- Porovnání dvou různých textových řetězců - metoda vrátí false.
- Testování stejných řetězců, ale s jinými velkými a malými písmeny se zapnutím ignorování velikosti písmen - vrátí se true.
- Vstupní objekt není textový řetězec - metoda spadne s vyhozením výjimky IllegalArgumentException.

7.1.3 Ukázka

O tvorbu Swagger souboru se stará třída YamlToStringService. Ta na základě zpracovaného modelu zpracuje data a převede je na String, který má správné řádkování a odsazení takové, jaké by měl Swagger soubor mít. V ukázce je zobrazeno testování výstupu modelu, který nemá žádné Tagy, Metody nebo Schema. Výstupem jsou v tomto případě základní informace, které by měl každý Swagger soubor ve správném odsazení a odřádkování viz Obrázek 7.2.

```
@DisplayName("Creates String based on model.")
@Test
public void toStringModel_ModelOfNoParts_ReturnsCorrectString() {
      ARRANGE
    Model model = mock(Model.class);

    when(model.getOpenApiVersion()).thenReturn( t: "3.0.3");
    when(model.getDescription()).thenReturn( t: "description");
    when(model.getTitle()).thenReturn( t: "title");
    when(model.getVersion()).thenReturn( t: "1.0");

    when(model.getTags()).thenReturn(Collections.emptyList());
    when(model.getMethods()).thenReturn(Collections.emptyMap());
    when(model.getSchemas()).thenReturn(Collections.emptyList());


    String expectedValue =
            "openapi: \"3.0.3\"\n" +
                    "info:\n" +
                    TAB + "description: \"description\"\n" +
                    TAB + "title: \"title\"\n" +
                    TAB + "version: \"1.0\"\n" +
                    "tags:\n" +
                    "paths:\n" +
                    "components:\n" +
                    TAB + "schemas:";

      ACT
    String result = yamlToStringService.toString(model);

      ASSERT
    assertEquals(expectedValue, result);
}
```

Obrázek 7.2: Test na tvorbu výstupu Swagger souboru

7.2 Uživatelské testy

V následující části bude testujícímu uživateli přiděleno zadání společně s návodem tvorby modelu v Enterprise Architect a použití samotné aplikace.

7.2.1 Obecné zadání uživatelského testu

Zadáním je vytvořit model API pro systém spravující zoo, který bude mít tři různé endpointy, kterými budou `/animals/{animal_id}/enclosure`, `/animals` a `/zookeepers/{id}`. Na interface budou navázány tři metody typu GET, PUT, POST. Schéma bude obsahovat tři různé třídy. Pro inspiraci můžete použít defaultně vytvořenou službu Petstore od OpenAPI.[27]

7.2.2 Konkrétní zadání uživatelského testu

1. Pomocí návodu v kapitole 5 si vymodelujte svoji službu a následně vygenerujte stuby.

2. V modelu Enterprise Architect si vytvořte balíček se schématem, ve kterém budou tři třídy - Animal, Enclosure a Zookeeper.

3. Každá třída bude mít atribut id typu int, třída Animal bude mít atribut birthDate typu date a Zookeeper atribut name typu String. Mezi třídami vytvořte konektory podle potřeby.

4. Na stejné úrovni balíčků si vytvořte balíček s názvem vašeho API a v něm balíček s číslem verze API. V balíčku s verzí si vytvořte rozhraní s názvem vašeho API.

5. Nyní vytvoříte a na rozhrání navážete tři metody, které budou přistupovat k různým endpointům.

6. Metoda GET, která bude přistupovat k endpointu `/animals` a bude získávat všechny zvířata v zoo.

7. Metoda POST, která vloží nového chovatele na endpoint `/zookeepers/{id}`.

8. Metoda PUT, která na endpointu `/animals/{animal_id}/enclosure` aktualizuje výběh podle zvířete.

9. Tento model potom nechte vygenerovat aplikací.

7.2.3 Hodnocení návodu

Pro testujícího uživatele bylo důležité si přečíst návod celý, aby pochopil řešení a vytváření celistvého modelu a všech jeho částí. Před přečtením návodu měl uživatel jasnou teoretickou vizi a návod mu pomohl ji převézt do praktického hlediska. Uživatel podotkl, že pro správné splnění zadání a práce s aplikací je nezbytné umět pracovat s Enterprise Architect. Uživatel zhodnotil návod celkově velice pozitivně a ocenil ho, jelikož pro něj bylo jednoduché se v něm vyznat a pochopit všechny principy, v čemž mu napomáhali přiložené ukázky v podobě obrázků.

7.2.4 Zpětná vazba aplikaci

Uživateli přišlo zdlouhavé u každého typu určovat Stereotyp, představoval by si mít pro každý typ samostatný objekt. Například by byly různé typy, které by znamenaly přímo jednu z pěti metod. Stejně tak přišlo uživateli zdlouhavé vytváření atributů tříd jako objekty typu Primitive a ke každému takovému objektu přiřadit atribut "type". Později to uživateli přišlo užitečné v případě určování parametrů v requestech. Pro uživatele bylo obecně složité nastavování a instalace Enterprise Architect.

7.2.5 Celkové zhodnocení

Uživateli se nápad a zpracování velice líbilo. Byl si jistý, že by to bylo užtečné při jeho každodenní práci vývojáře. Jako hlavním klíčovým pozitivem mu přišel ušetřený čas a také přehledná dokumentace, která je ke generovaným stubům dodávána, jelikož má problém se ve větších aplikacích vyznat. Uživateli přišel návod plně srozumitelný a pochopení použití aplikace jasné.

Kapitola 8

Závěr

V rámci této práce byla zpočátku provedena analýza požadavků na vyvíjený software a na základě nich vyvinuta aplikace, která usnadňuje práci všem potenciálním uživatelům v oblasti softwarového inženýrství.

Na základě výběru technologií se jako modelovací nástroj, kde se tvoří vstup aplikace, vybral Enterprise Architect. Ten je společně s aplikací napojený na MySQL server. Celá aplikace byla sepsána v jazyce Java za pomocí Spring Boot frameworku, který při vývoji usnadňoval práci. Generátorem stubů byl zvolen OpenAPI Generator, který aplikací generované Swagger převáděl na stuby Frontendu a Backendu.

Při práci jsem využíval různé metodiky, znalosti a zkušenosti získané při studiu, mezi které náleží Návrhové vzory, Modelovací jazyk UML nebo Návrh softwarového systému. Ostatní informace, které jsem potřeboval při práci využít, jsem čerpal z internetu nebo použité literatury.

Kapitola 9

Budoucí práce

Práce už v této podobě nabízí plno příležitostí a výhod. Přesto je možné ji v budoucnosti posouvat v mnoha směrech, kde by mohla potenciálním uživatelům více pomoci nebo jim ještě více ušetřit čas a práci.

9.1 Přidání výčtového typu

V této verzi není aplikace schopna z modelů Enterprise Architect zpracovat výčtové typy. Uživatel si je musí doplnit do stubů nebo Swagger souboru sám. V budoucích verzích by aplikace mohla být schopna tento problém řešit sama.

9.2 Automatická aplikační kontrola Swaggeru

Pokud uživatel provede nekorektní modelaci své služby a vznikne tím neplatný Swagger soubor, je uživatel schopen tyto nedostatky objevit pomocí Swagger Editoru.[28] V budoucích verzích by aplikace mohla rozeznávat, zda model vytvořený uživatelem je korektní a obsahuje vše, co je potřeba.

9.3 Zjednodušení modelování

Na základě zpětné vazby v uživatelských testech se ukázalo, že by modelování mohlo probíhat rychleji, jelikož bylo pro uživatele moc zdlouhavé k mnoha objektům vždy opakovaně přiřazovat Stereotyp. Tento problém by se v budoucích verzích dal také vyřešit, aby uživatel nemusel vždy přiřazovat stereotypy a aplikace by mohla na základě jiných vlastností poznat, o jaký objekt se jedná.

Literatura

[1] Hailpern, Brent, and Peri Tarr. *Model-driven development: The good, the bad, and the ugly.* IBM systems journal 45.3 (2006): 451-461.

[2] Selic, Bran. *The pragmatics of model-driven development.* IEEE software 20.5 (2003): 19-25.

[3] Tolvanen, Juha-Pekka, and Matti Rossi. *Metaedit+ defining and using domain-specific modeling languages and code generators.* Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. 2003.

[4] He, Ren Yu. *Design and implementation of web based on Laravel framework.* 2014 International Conference on Computer Science and Electronic Technology (ICCSET 2014). Atlantis Press, 2015.

[5] Steinberg, Dave, et al. *EMF: eclipse modeling framework.* Pearson Education, 2008.

[6] Xu, Dianxiang, et al. *An automated test generation technique for software quality assurance.* IEEE transactions on reliability 64.1 (2014): 247-268.

[7] Ambler, Tim, et al. *Yeoman.* JavaScript Frameworks for Modern Web Dev (2015): 37-52.

[8] Leslie, Donald M. *Using Javadoc and XML to produce API reference documentation.* Proceedings of the 20th annual international conference on Computer documentation. 2002.

[9] *OpenAPI Generator* OpenAPI Generator Documentation `https://github.com/OpenAPITools/openapi-generator`

[10] Hejlsberg, Anders, Scott Wiltamuth, and Peter Golde. *C# language specification.* Addison-Wesley Longman Publishing Co., Inc., 2003.

[11] Arnold, Ken, James Gosling, and David Holmes. *The Java programming language.* Addison Wesley Professional, 2005.

[12] Boot, Spring. *Spring Boot.* (2020).

[13] Liawatimena, Suryadiputra, et al. *Django web framework software metrics measurement using radon and pylint.* 2018 Indonesian Association for Pattern Recognition International Conference (INAPR). IEEE, 2018.

[14] Bernard, Scott A. *An introduction to enterprise architecture.* AuthorHouse, 2012.

[15] Leroux, Daniel, Martin Nally, and Kenneth Hussey. *Rational Software Architect: A tool for domain-specific modeling.* IBM systems journal 45.3 (2006): 555-568.

[16] DuBois, Paul. *MySQL.* Pearson Education, 2008.

[17] Drake, Joshua D., and John C. Worsley. Practical PostgreSQL. "O'Reilly Media, Inc.", 2002.

[18] Masse, Mark. *REST API design rulebook: designing consistent RESTful web service interfaces.* O'Reilly Media, Inc.", 2011.

[19] Espinoza-Arias, Paola, Daniel Garijo, and Oscar Corcho. *Mapping the web ontology language to the openapi specification.* Advances in Conceptual Modeling: ER 2020 Workshops CMAI, CMLS, CMOMM4FAIR, CoMoNoS, EmpER, Vienna, Austria, November 3–6, 2020, Proceedings 39. Springer International Publishing, 2020.

[20] Li, Hongwei, et al. *Research of "Stub" remote debugging technique.* 2009 4th International Conference on Computer Science & Education. IEEE, 2009.

[21] *OpenAPI Basic Structure* `https://swagger.io/docs/specification/basic-structure/`

[22] *OpenAPI API Server* `https://swagger.io/docs/specification/api-host-and-base-path/`

[23] *OpenAPI Paths and Operations* `https://swagger.io/docs/specification/paths-and-operations/`

[24] *OpenAPI Schemas* `https://swagger.io/docs/specification/data-models/`

[25] *OpenAPI generators list* `https://openapi-generator.tech/docs/generators/`

[26] *MySQL Server* `https://www.mysql.com`

[27] *PetstoreAPI* REST API example with SwaggerUI `https://petstore3.swagger.io`

[28] *Swagger Editor* `https://editor.swagger.io`

Příloha A

Seznam použitých zkratek

- API - Applicatiion Programming Interface
- CRUD - Create Read Update Delete
- DAO - Data Access Object
- DTO - Data Tranfer Object
- EMF - Eclipse Modeling Framework
- HTML - Hypertext Markup Language
- HTTP - Hypertext Transfer Protocol
- JSON - JavaScript Object Notation
- JVM - Java Virtual Machine
- LDM - Logický Datový Model
- MD - Man-day
- MDD - Model Driven Development
- PDF - Portable Document Format
- REST - Representational State Transfer
- UI - User Interface
- UML - Unified Modeling Language
- URL - Uniform Resource Locator

- XML - Extensible Markup Language
- YAML - Yaml Ain't Markup Language