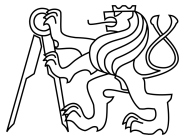
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Webový nástroj pro kreslení UML diagramů tříd
Student:	Iuliia Evseenko
Vedoucí:	Ing. Petr Špaček, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Pro potřeby projektu Laplace-IDE vytvořte prototyp nástroje pro kreslení UML diagramu tříd s využitím vlastností HTML5 SVG a frameworku React JS.

1. Seznamte se s technologiemi HTML5, HTML5 SVG a jazykem JavaScript.
2. Navrhněte model uložení diagramu tříd v prostředí JavaScript.
3. Navrhněte proces vykreslení takového modelu pomocí HTML5 SVG.
4. Na základě návrhu implementujte prototyp takového nástroje.
5. Proveďte uživatelské otestování výsledku a vyhodnoťe míru pokrytí všech vlastností UML diagramu tříd.

Práce je tématem vypsaným výzkumnou skupinou #CCMi.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 6. listopadu 2017

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Webový nástroj pro kreslení UML diagramů tříd

Evseenko Iuliia

Katedra softwarového inženýrství
Vedoucí práce: Ing. Petr Špaček Ph.D

11. května 2018

Poděkování

Chtěla bych poděkovat vedoucímu mé bakalářské práce Ing. Petru Špačkovi Ph.D za nasměrování a pomoc při psání této bakalářské práce. Zvláštně chtěla bych poděkovat přítelovi, který mě podporoval celou dobu mého studia. Poděkování patří taky mé rodině, která mě podporovala a pomáhala během studia a při psaní této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické 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. § 46 odst. 6 tohoto zákona 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 11. května 2018 .....

Odkaz na tuto práci

Evseenko, Iuliia. *Webový nástroj pro kreslení UML diagramů tříd.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Cílem této práce je navrhnout a implementovat prototyp nástroje pro kreslení UML diagramů tříd s využitím technologií HTML5 SVG a frameworku ReactJS. Výstupem této bakalářské práce je funkční a otestovaná webová aplikace, která umožňuje kreslení UML diagramů tříd.

Klíčová slova kreslič UML diagramů tříd, navrh a implementace nástroje, prototyp, webový nástroj, HTML5, SVG, ReactJS, JavaScript

Abstract

The objective of this thesis is to design and implement prototype tool for drawing UML class diagrams using HTML5 SVG and ReactJS framework. The output of this bachelor thesis is a functional and tested web application.

Keywords drafter of UML class diagrams, design and implementation of web application, prototype, web application, HTML5, SVG, ReactJS, JavaScript

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Většina softwarových inženýrů se alespoň jednou v životě setkala s UML diagramy tříd a problémem v jakém prostředí diagramy vytvořit. UML diagramy tříd slouží k návrhu systému a v současnosti existují desktopové aplikace a webové nástroje, které pomáhají daný problém řešit. V této práci se soustředím na oblast webových aplikací.

Největší přínos z mé práce budou mít analytici a programátoři, kterým aplikace umožní vytvářet UML diagramy tříd bez nutnosti instalace specializovaného software. V neposlední řadě výstup práce pomůže i studentům, kteří tyto diagramy potřebují často vytvářet v rámci studia.

Téma jsem si zvolila, protože v rámci analýzy stávajících řešení se mi nepodařilo nalézt žádné kvalitně zpracované řešení tohoto problému v oblasti webových aplikaci s využitím knihovny ReactJS. Většina stávající řešení je určena nejenom ke kreslení UML diagramů tříd, ale také pro modelování dalších diagramů. Výstupem mé bakalářské práce bude prototyp webové aplikace pro modelováni pouze UML class diagramů, avšak aplikaci se pokusím navrhnout tak, aby v budoucnu bylo možné snadno přidat i další diagramy jazyka UML. Hlavní motivací je tedy návrh a implementace takového nástroje v populární technologii ReactJS, který bude jednou ze stěžejních komponent projektu webového IDE s pracovním názvem Laplace-IDE, s jehož vizí vedoucí práce téma navrhl.

Bakalářská práce obsahuje pět kapitol: cíl práce, analýza, návrh, implementace a testování.

Kapitola 1 obsahuje cíl mé bakalářské práce.

V kapitole 2 se zabývám analýzou stávajících řešení. Dále analyzuji případy užití systému, funkční a nefunkční požadavky na systém. Táto kapitola bude obsahovat kontrolu splnění všech požadavků na systém. Taktéž se seznámím s technologiemi, ve kterých budu webový nástroj implementovat.

V kapitole 3 navrhuji model uložení UML diagramů tříd v prostředí JavaScript. Navrhnu proces vykreslení takového modelu pomocí značkovacího jazyka SVG a popíšu návrhový vzor, který mi to vykreslení umožní. Dále v této kapitole

ukáži systémovou architekturu prototyp a návrh grafického uživatelského rozhraní.

V kapitole 4 na základě návrhu implementuji prototyp webového nástroje pro kreslení UML diagramů tříd. Popíšu nejdůležitější kroky implementace.

Kapitola 5 obsahuje testování aplikace a vyhodnocení míry pokrytí všech vlastností UML diagramu tříd.

Kapitola 1

Cíle práce

Cílem rešeršní části práce je analýza stávajících řešení pro kreslení UML diagramů tříd. Zvážení výhod a nevýhod současných řešení a vyjmenování vlastností, které musí splňovat webový nástroj, jenž bude implementován v rámci této bakalářské práce.

Dílčím cílem rešeršní části je nalezení všech případů užití systému spolu s definicí funkčních a nefunkčních požadavků. Taktéž se v rešeršní části pokusím seznámit se s technologiemi, které mají být použity při implementaci, konkrétně: technologie HTML5, SVG, ECMAScript; a frameworky ReactJS a Express.js.

Cílem praktické části práce je návrh a implementace prototypu webového nástroje pro kreslení UML diagramů tříd. Praktická čast obsahuje návrh architektury aplikace, která je do značné míry dána použitými technologiemi. Dále v praktické části uvádím návrh modelu uložení diagramů tříd v prostředí JavaScript. Následuje popis návrhového vzoru, který jsem použila pro řešení problému vykreslování. Do praktické části taky patří implementace prototypu kreslícího nástroje a popis jednotlivých implementačních kroků. Poslední sekce praktické části je testování, jejíž cílem ukázat, že v rámci požadavků se nástroj chová korektně.

Kapitola 2

Analýza

2.1 Analýza stávajících řešení

V rámci mé bakalářské práce jsou analyzovány řešení, která podporují modelování UML diagramů tříd online a tudíž není třeba v dalším textu popisovat hlavní výhodu těchto řešení, tedy, že jsou přístupné přes webový prohlížeč bez nutnosti instalace specializovaného software lokálně. Společnou výhodou analyzovaných nástroje je, že všechny nabízejí start modelování pomocí šablony.

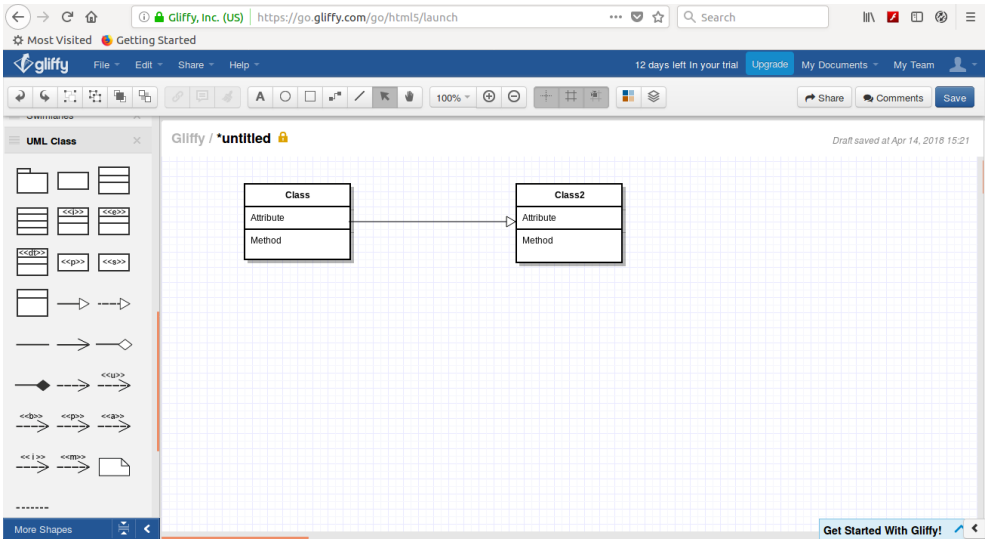
2.1.1 `www.gliffy.com`

`www.gliffy.com` je to webová aplikace, která umožňuje modelování nejen UML diagramů, ale i modelování diagramů business procesů, diagramů počítačových sítí, ER diagramů a mnoha dalších. Webová stránka `www.gliffy.com` se nachází na první pozici ve vyhledávači Google na dotaz „uml diagram tool online“. Pro modelování jakéhokoli diagramu je nutné se registrovat. Existuje možnost využití 14-ti denní free trial s omezenými možnostmi vývoje anebo přihlášení se ke každoměsíční platbě pro získání plné verze.

Mezi výhody patří:

- možnost využití free trial,
- možnost sdílení svého diagramu s kolegy, kteří mohou následně pokračovat v úpravě [1],
- možnost exportu diagramů do různých formátů [2],
- možnost „Load Draft“ při přerušení Session,
- přehledné uživatelské rozhraní (viz obrázek 2.1).

`www.gliffy.com` je primárně napsán v jazyce JavaScript [3].



Obrázek 2.1: Uživatelské rozhraní Gliffy

2.1.2 `www.draw.io`

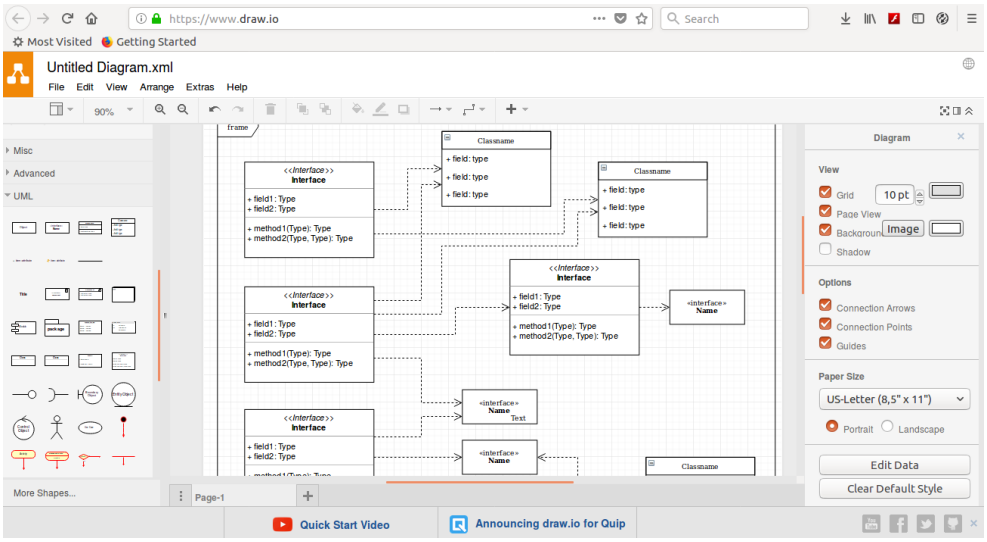
Webová stránka `www.draw.io` se nachází na druhé pozici ve vyhledávači Google na dotaz „uml diagram tool online“. Stejně jako předchozí aplikace podporuje více druhů diagramů, takových jako UML diagramy, Business diagramy, Network diagramy atd. Pro modelování se nevyžaduje žádná registrace. Modelování v online a offline verzi je zdarma.

Mezi výhody patří:

- možnost modelování zdarma,
- žádná registrace pro počáteční modelování,
- možnost exportu diagramů do různých formátů [4],
- možnost importu diagramů z různých formátů a různých zdrojů, takových jako Google Drive, GitHub, DropBox atd. [5].

Mezi nevýhody patří:

- není možnost sdílení diagramu s kolegy
- není možnost „Load Draft“ při přerušení Session, jenom „Load existing diagram“,
- uživatelské rozhraní se podobá Google Drive (viz obrázek 2.2).



Obrázek 2.2: Uživatelské rozhraní Draw.io

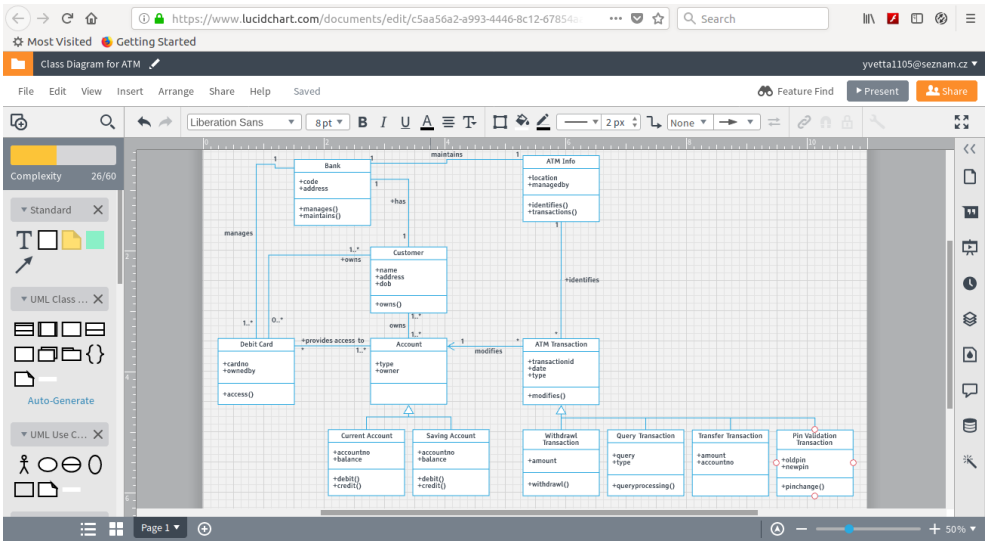
`www.draw.io` je primárně napsán v jazycích JavaScript a Java s využitím knihovny mxGraph [6].

2.1.3 `www.lucidchart.com`

Webová stránka `www.lucidchart.com` se nachází na třetí pozici ve vyhledávači Google na dotaz „uml diagram tool online“. Stejně jako předchozí aplikace nabízí širokou škálu druhů diagramů, takových jako UML diagramy, Business diagramy, Wireframy, Network diagramy atd. Pro modelování je vyžadována registrace, ale je zde i možnost využití 7-mi denní free trial pro modelování bez omezení. Po skončení free trial Váš účet se zůstává, ale s omezenými možnostmi vývoje. Taky se lze přihlásit ke každoměsíční platbě pro plnou verze.

Mezi výhody patří:

- možnost modelování zdarma,
- možnost exportu diagramů do různých formátů [7],
- možnost importu diagramů z dvou předchozích webových aplikací [8].
- možnost sdélení svého diagramu s kolegy [9],
- možnost „Load Draft“ při přerušení Session.
- přehledné uživatelské rozhraní (viz obrázek 2.3).



Obrázek 2.3: Uživatelské rozhraní Lucidchart

2.1.4 `www.creately.com`

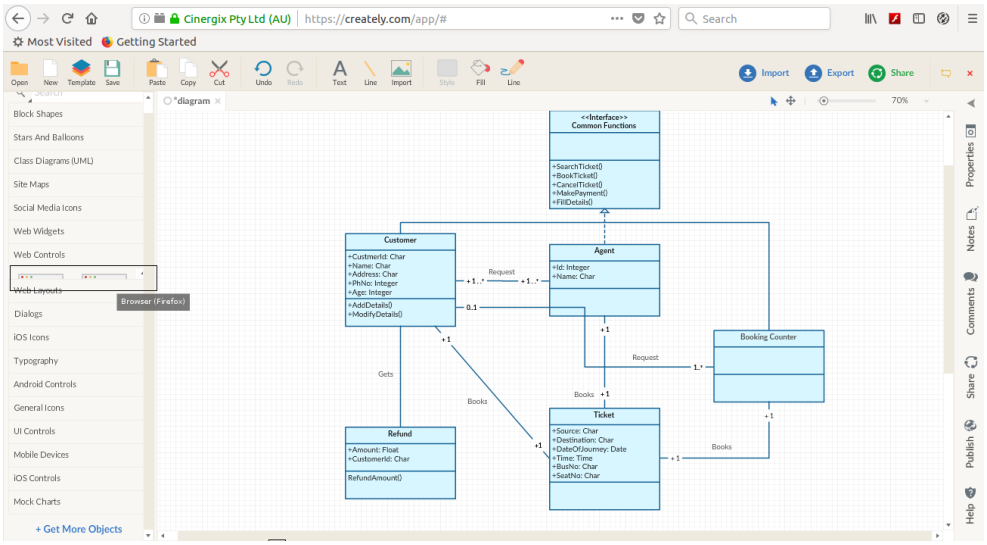
`www.creately.com` je webová aplikace, která stejně jako předchozí aplikace nabízí velké množství diagramů pro modelování. Mezi nimi patří Data Flow diagramy, UML diagramy, Business diagramy, UI návrh atd. Webová stránka `www.creately.com` se nachází na čtvrté pozici ve vyhledávači Google na dotaz „uml diagram tool online“. Pro modelování není vyžadována žádná registrace, avšak poté nejsou některé funkce dostupné. Pro plnou verzi lze zakoupit předplatné.

Mezi výhody patří:

- žádná registrace pro počáteční modelování,
- možnost modelování zdarma,
- možnost sdílení diagramů a modelování „Real Time“ [10],
- možnost exportu diagramů do různých formátů,
- možnost importu diagramů z různých formátů,
- možnost „Load Draft“ při přerušení Session.

Mezi nevýhody patří:

- import a export jenom po registraci,



Obrázek 2.4: Uživatelské rozhraní Creately

- „Share diagram“ je dostupný po registraci,
- při použití velkého množství modelovacích nástrojů se levá část aplikace, sloužící pro zobrazení jednotlivých prvků diagramů, stává nepoužitelnou (viz obrázek 2.4).

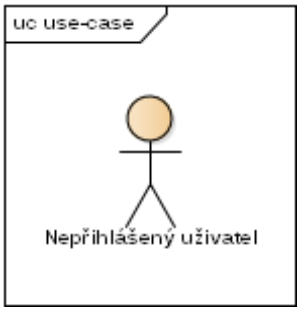
`www.creately.com` je primárně napsán v jazycích TypeScript a JavaScript [11].

2.1.5 Shrnutí aktuálních řešení

V současnosti existuje velké množství řešení tohoto problému, proto jsem analyzovala ty nejpopulárnější projekty, dle Google search engine. Z analýzy vyplývá, že každé z probraných řešení obsahuje možnost modelování zdarma, ale ne každé řešení podporuje možnost modelování bez registrace. Každá webová aplikace z analýzy nabízí velké množství diagramů, včetně UML diagramů. Žádná z analyzovaných aplikací však nevyužívá knihovnu ReactJS, což bude hlavním přínosem mé práce.

2.2 Případy užití

V této části jsou uvedený případy užití, které ukazují jaké funkcionality systém musí obsahovat a jak jsou spouštěny (viz obrázek 2.6). Případy užití jsou specifikované krátkým popisem, aktéry, kteří se případu účastní a scénářem. Taktéž je u případů užití evidována priorita.



Obrázek 2.5: Seznam účastníků

2.2.1 Seznam účastníků

Webový nástroj pro kreslení UML diagramů tříd nevyžaduje žádné přihlášení, tím pádem je identifikován jeden účastník - nepřihlášený uživatel (viz obrázek 2.5).

- **Nepřihlášený uživatel** - uživatel, který není přihlášen do aplikace

2.2.2 Scénáře případů užití

V této sekce jsou uvedený scénáře případů užití nástroje (viz obrázek 2.6). Pro každý scénář jsou uvedený aktéři, kteří se účastní případ užití.

UC1 - Založení nového diagramu

Krátký popis:

Use-case umožňuje založit si nový UML diagram tříd.

Aktéři:

- Uživatel
- Systém

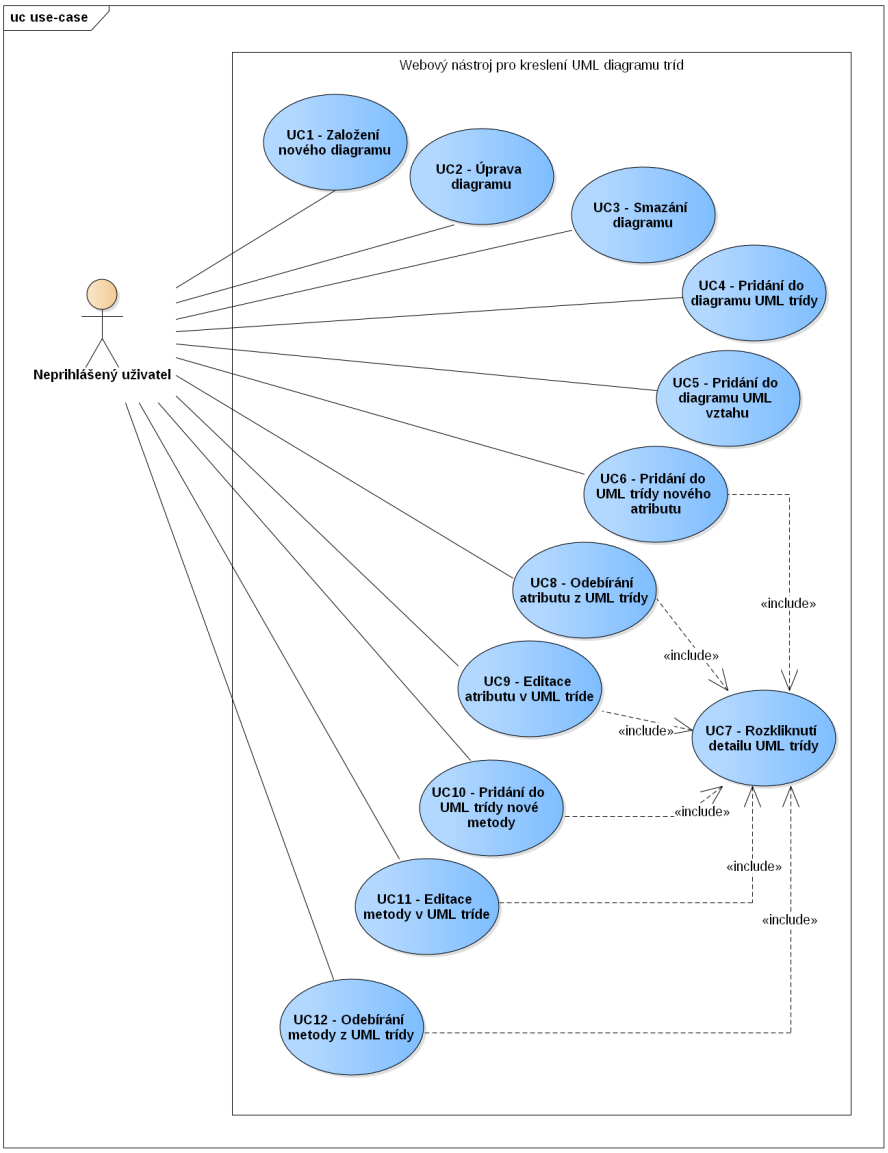
Hlavní scénář:

1. Případ užití se začíná, když si uživatel chce založit nový UML diagram tříd.
2. Systém bude mít nový diagram již na vstupní obrazovce a uživatel může pokračovat v modelování.

Alternativní scénář:

2.1. Uživatel už něco v diagramu má a chce si založit nový diagram.
2.2. Uživatel klikne na tlačítko „New Diagram“.
2.3. Systém se zeptá jestli uživatel si chce opravdu založit diagram.
2.4. Uživatel potvrdí, že si chce založit nový diagram.
2.5. Systém zobrazí nový diagram.

Priorita: vysoká



Obrázek 2.6: Use-case model

UC2 - Úprava diagramu

Krátký popis:

Use-case umožňuje upravit UML diagram tříd, například změnit polohu nebo velikost elementu v diagramu.

Aktéři:

- Uživatel
- Systém

Hlavní scénář:

1. Případ užití se začíná, když uživatel si chce upravit zobrazený UML diagram tříd.
2. Uživatel přetáhne nějaký element nebo změní velikost elementu na diagramu.
3. Systém uloží změny.

Priorita: vysoká

UC3 - Smazání diagramu

Krátký popis:

Use-case umožňuje smazat zobrazený UML diagram tříd.

Aktéři:

- Uživatel
- Systém

Hlavní scénář:

1. Případ užití se začíná, když uživatel si chce smazat zobrazený UML diagram tříd.
2. Uživatel klikne na tlačítko „Delete Diagram“.
3. Systém se zeptá jestli uživatel opravdu chce smazat diagram.
4. Uživatel potvrdí že chce smazat diagram.
5. Systém smaže diagram.

Priorita: vysoká

UC4 - Přidání do diagramu UML třídy

Krátký popis:

Use-case umožňuje přidat UML třídu to zobrazeného UML diagramu tříd.

Aktéři:

- Uživatel
- Systém

Hlavní scénář:

1. Případ užití se začíná, když uživatel si chce přidat UML třídu do diagramu tříd.
2. Uživatel klikne na tlačítko „Add Class“.
3. Systém přidá do diagramu novou UML třídu.

Priorita: vysoká

UC5 - Přidání UML vztahu do diagramu

Krátký popis:

Use-case umožňuje přidat UML vztah to zobrazeného UML diagramu tříd.

Aktéři:

- Uživatel
- Systém

Hlavní scénář: 1. Případ užití začíná, když uživatel chce přidat vztah do zobrazeného UML diagramu tříd.
2. Uživatel klikne na tlačítko „New Relation“.
3. Systém zobrazí formulář do kterého uživatel musí napsat názvy nebo ID tříd pro relace.
4. Uživatel vyplní názvy nebo ID tříd.
5. Systém zobrazí nový vztah na diagramu tříd.

Priorita: vysoká

UC6 - Přidání nového atributu do UML třídy v diagramu

Krátký popis:

Use-case umožňuje přidat atribut do libovolné třídy v zobrazeném UML diagramu tříd.

Aktéři:

- Uživatel
- Systém

Hlavní scénář:

1. Případ užití se začíná, když uživatel chce přidat atribut do nějaké UML třídy v diagramu.
2. Include UC7 - Rozkliknutí detailu UML třídy.
3. V detailu třídy uživatel rozklikne položku „Attributes“.
4. Systém zobrazí položku „Attributes“.
5. Uživatel stiskne tlačítko „New Attribute“.
6. Systém zobrazí formulář pro nový atribut.

7. Uživatel vyplní formulář a stiskne tlačítko „Save“.
8. Systém uloží nový atribut.

Priorita: střední

UC7 - Rozkliknutí detailu UML třídy

Krátký popis:

Use-case umožňuje rozkliknout detail pro libovolnou UML třídu v diagramu tříd.

Aktéři:

- Uživatel
- Systém

Hlavní scénář:

1. Případ užití začíná, když uživatel chce rozkliknout detail UML třídy v zobrazeném diagramu tříd.
2. Uživatel rozklikne detail UML třídy kliknutím pravým tlačítkem myši.
3. Systém zobrazí detail třídy.

Priorita: střední

UC8 - Odebírání atributu z UML třídy

Kratký popis:

Use-case umožňuje odebrat libovolný atribut z UML třídy.

Aktéři:

- Uživatel
- Systém

Hlavní scénář:

1. Případ užití začíná, když uživatel chce odebrat atribut z nějaké UML třídy v diagramu.
2. Include UC7 - Rozkliknutí detailu UML třídy.
3. V detailu třídy uživatel rozklikne položku „Attributes“.
4. Systém zobrazí položku „Attributes“.
5. Uživatel stiskne tlačítko „Delete Attribute“ vedle příslušného atributu, který chce smazat.
6. Systém zobrazí dialog pro potvrzení této akce.
7. Uživatel podtvrdí, že chce smazat atribut.
8. Systém smaže atribut.

Priorita: střední

UC9 - Editace atributu v UML třídě

Krátký popis:

Use-case umožňuje editovat libovolný atribut v UML třídě.

Aktéři:

- Uživatel
- Systém

Hlavní scénář:

1. Případ užití začíná, když uživatel chce editovat atribut v nějaké UML třídě v diagramu.
2. Include UC7 - Rozkliknutí detailu UML třídy.
3. V detailu třídy uživatel rozklikne položku „Attributes“.
4. Systém zobrazí položku „Attributes“.
5. Uživatel stiskne tlačítko „Edit Attribute“ vedle příslušného atributu, který chce editovat.
6. Systém zobrazí formulář pro editace atributu.
7. Uživatel vyplní formulář a stiskne tlačítko „Save“.
8. Systém uloží změny v atributu.

Priorita: střední

UC10 - Přidání nové metody do UML třídy

Krátký popis:

Use-case umožňuje přidat novou metodu do libovolné UML třídy.

Aktéři:

- Uživatel
- Systém

Hlavní scénář:

1. Případ užití začíná, když uživatel chce přidat metodu do nějaké UML třídy v diagramu.
2. Include UC7 - Rozkliknutí detailu UML třídy.
3. V detailu třídy uživatel rozklikne položku „Methods“.
4. Systém zobrazí položku „Methods“.
5. Uživatel stiskne tlačítko „New method“.
6. Systém zobrazí formulář pro novou metodu.
7. Uživatel vyplní formulář a stiskne tlačítko „Save“.
8. Systém uloží novou metodu.

Priorita: střední

UC11 - Editace metody v UML třídě

Krátký popis:

Use-case umožňuje editovat libovolnou metodu v UML třídě.

Aktéři:

- Uživatel
- Systém

Hlavní scénář:

1. Případ užití začíná, když uživatel chce editovat metodu v nějaké UML třídě v diagramu.
2. Include UC7 - Rozkliknutí detailu UML třídy.
3. V detailu třídy uživatel rozklikne položku „Methods“.
4. Systém zobrazí položku „Methods“.
5. Uživatel stiskne tlačítko „Edit method“ vedle příslušné metody, kterou chce editovat.
6. Systém zobrazí formulář pro editace metody.
7. Uživatel vyplní formulář a stiskne tlačítko „Save“.
8. Systém uloží změny v metodě.

Priorita: střední

UC12 - Odebrání metody z UML třídy

Krátký popis:

Use-case umožňuje odebrat libovolnou metodu z UML třídy.

Aktéři:

- Uživatel
- Systém

Hlavní scénář:

1. Případ užití začíná, když uživatel chce odebrat metodu z nějaké UML třídy v diagramu.
2. Include UC7 - Rozkliknutí detailu UML třídy.
3. V detailu třídy uživatel rozklikne položku „Methods“.
4. Systém zobrazí položku „Methods“.
5. Uživatel stiskne tlačítko „Delete Method“ vedle příslušné metody, kterou chce smazat.
6. Systém zobrazí dialog pro potvrzení této akce.
7. Uživatel potvrdí, že chce smazat metodu.
8. Systém smaže metodu.

Priorita: střední

2.3 Analýza požadavků

Na základě případů užití (use-cases) byly sestaveny následující požadavky na systém.

2.3.1 Funkční požadavky

F1 - Založení UML diagramu tříd
Uživatel bude schopen založit nový diagram.

Priorita: vysoká

F2 - Úprava UML diagramu tříd
Uživatel bude schopen upravit zobrazený diagram a přidat nějaký element do diagramu. Uživatel bude také schopen změnit polohu elementů v diagramu.

Priorita: vysoká

F3 - Smazání UML diagramu tříd
Uživatel bude moci smazat zobrazený UML diagram tříd.

Priorita: vysoká

F4 - Editace UML třídy v diagramu
Uživatel bude schopen upravovat UML třídu v diagramu. Uživatel bude mít možnost rozkliknout detail UML třídy, přidat, smazat, editovat atribut ve třídě. Stejně tak bude moci přidat, smazat nebo upravit metodu ve třídě.

Priorita: střední

2.3.2 Nefunkční požadavky

N1 - Architektura aplikace
Webová aplikace v architektuře klient – server

N2 - Backend v Node.js
Backend musí být implementován v prostředí Node.js s použitím frameworku Express.js

N3 - Frontend v React.js
Frontend musí být implementován v jazyce JavaScript (ES6) pomocí React.js

N4 - REST
Klient bude komunikovat s serverem skrze REST rozhraní

2.4 Kontrola splnění všech požadavků

Daná tabulka 2.1 ilustruje míru pokrýti požadavků na systém. Všechny požadavky jsou pokryté.

Tabulka 2.1: Tabulka pokrytí požadavků

	Požadavky			
Případy užití	F1	F2	F3	F4
UC1	+			
UC2		+		
UC3			+	
UC4		+		
UC5		+		
UC6				+
UC7				+
UC8				+
UC9				+
UC10				+
UC11				+
UC12				+

2.5 Použité technologie

V této kapitole se budeme věnovat výběru vhodných technologií pro implementaci webového nástroje pro kreslení UML diagramů tříd a jejích popisu.

2.5.1 HTML5

HTML je značkovací jazyk pro tvorbu webových stránek. HTML5 je poslední verze jazyka HTML. Oproti předcházejícím specifikacím je rozšířená o nové sémantické tagy, definující strukturu stránky. Dále byla přidána podpora offline aplikací. Na závěr nové specifikace jsou navrženy perzistentní úložiště formou asociativního pole a relační databáze s podporou transakcí. Další vývoj HTML přímo ovlivňuje rozvoj internetových aplikací a umožňuje vývojářům realizovat více nápadů. Hlavním přínosem poslední specifikace jazyka HTML pro tuto bakalářskou práci je možnost použití SVG grafiky pomocí HTML tagu `<svg>`, sloužící jako kontejner pro vektorovou grafiku. [12]

2.5.2 SVG

SVG (Scalable Vector Graphics) je značkovací jazyk, který popisuje dvojrozměrnou vektorovou grafiku na základě formátu XML. Obrovská výhodou je, že HTML5 umožňuje vkládat SVG kód přímo do kódu webové stránky. Grafika SVG neobsahuje obrazová data ve formátu pixelů, ale seznam svých součástí – grafických objektů, pomocí kterých lze obrázek vykreslit, a proto je pro naše účely SVG je ideální. Klíčové vlastnosti zahrnují tvary, text a vestavěnou rastrovou grafiku s mnoha různými styly malování. Podporuje skriptování

pomocí jazyků, jako je ECMAScript, a má komplexní podporu pro animaci. [13]

2.5.3 ECMA6

ECMAScript je standartizovaná verze skriptovací jazyka JavaScript, který je využíván především na webových stránkách pro vytváření skriptů na straně klienta. ECMAScript6 je šestá edice téhož jazyka ECMAScript, která posunula ECMAScript dále ve směru objektově orientovaného programovacího jazyka. Syntaxe obsahuje třídy a moduly, jež jsou ale definovány sémanticky stejně jako v ECMAScript5 strict mode. Dále jsou doplněny iterátory, generátory výjimek a kolekce. [14]

2.5.4 ReactJS

ReactJS je JavaScriptová knihovna pro vytváření webových komponent a uživatelského rozhraní. V rámci architektury (či vzoru) Model-View-Controller představuje View vrstvu. ReactJS je odpovědný za prezentaci dat, přijímání a zpracování vstupů uživatele. Přináší zásadní změnu paradigmatu. S Reactem nepíšeme kód, který něco mění, ale kód, který popisuje, jak má vypadat výsledek, což je řádově snazší úloha. React umožňuje vytvářet komponenty, které umí zpravovat svůj stav. Jelikož komponenty můžeme vnořovat do jiných komponent, lze snadno udělat složitější interaktivní části uživatelského rozhraní. Obrovskou výhodou Reactu je to, že umožňuje používat JSX syntaxe a vytvářet komponenty, s nimimž lze snadno manipulovat v rámci DOM modelu. [15]

2.5.5 Node.js

Node.js je softwarový systém navržený pro psaní vysoce škálovatelných internetových aplikací, především webových serverů. Programy pro Node.js jsou psané v jazyce JavaScript, využívající model událostí a asynchronní vstupně-výstupní operace pro minimalizaci režie procesoru a maximalizaci výkonu. Balíkový ekosystém Node.js, npm, je největším ekosystémem open source knihoven na světě. [16] Podrobněji nástroj npm je popsán v kapitole 4.4.

2.5.6 Express.js

Express.js je minimalistický a flexibilní webový framework pro Node.js, který poskytuje robustní sadu funkcí pro webové a mobilní aplikace. S nepřeberným množstvím nástrojů HTTP a middlewaru, které má Express.js k dispozici, lze rychlé a snadno vytvořit robustní rozhraní API. Express.js je vydán jako bezplatný a otevřený software pod licencí MIT. [17]

2.5.7 Sequelize

Sequelize je ORM nástrojem pro Node.js. Objektově relační zobrazení (ORM) je programovací technika v softwarovém inženýrství, která zajišťuje automatickou konverzi dat mezi relační databází a objektově orientovaným programovacím jazykem. Sequelize podporuje dialekty PostgreSQL, MySQL, SQLite a MSSQL a nabízí solidní podporu transakcí, vztahy, replikaci čtení a další. V rámci této bakalářské práce se Sequelize používa ve spojení s frameworkem Express.js a databázovým systémem PostgreSQL. [18]

Kapitola 3

Návrh

3.1 Architektura

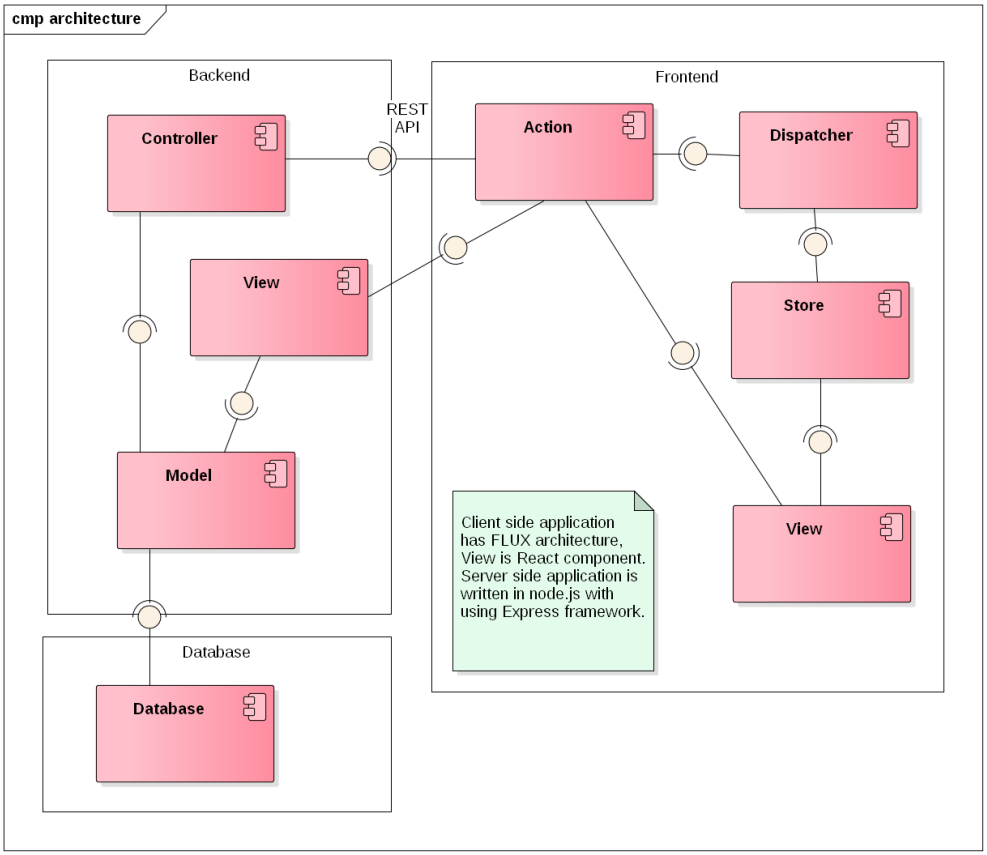
Architekturu lze rozdělit na tři části: klientskou, serverovou a datovou. Jedná se o model klient-server, protože klient a server mají na starosti různé věci a mají striktně oddělené zodpovědnosti (viz obrázek 3.1). Klientská část aplikace, tzv. frontend, běží v prohlížeči uživatele a má za odpovědnost prezentaci dat. Serverová část aplikace, tzv. backend, běží na jiném serveru a má na starost zpracování uživatelských požadavků a práci s daty. Datová vrstva zahrnuje databázi, která běží na databazovém serveru. Má odpovědnost za uchování dat a poskytnutí rozhrání pro přístup k datům. Komunikace mezi klientskou a serverovou části aplikace bude probíhat pomocí rozhraní REST (více v sekci 3.6).

3.1.1 Frontend

Jako prezentační vrstva bude sloužit React aplikace stavěna na základě architektury FLUX. Jedná se o návrhový vzor, který se aplikuje při návrhu jednosměrného toku dat v UI. Vychází z předpokladu, že výsledné UI je reprezentovatelné pomocí dat, které do něj tečou z úložišť (Store). Interakce uživatelů jsou reprezentovány pomocí akcí. Akce zpracovává Dispatcher, který následně notifikuje o změnách jednotlivá úložiště (viz obrázek 3.1). Tím pádem, hlavní zodpovědností frontendu je prezentování dat uživateli, posílání uživatelských požadavků na backend a prezentace backend odpovědí.

3.1.2 Backend

Na stráně serveru je architektura Model-View-Controller, která je dáná použitím frameworku Express.js (viz obrázek 3.1). Controller slouží pro přijímání uživatelských požadavků a View k sestavení odpovědí. Model reprezentuje datový model aplikace (více o Modelu 3.3). Na straně backendu se používá



Obrázek 3.1: Architektura aplikace

ORM nástroj, který mapuje prvky navrženého modelu na skutečné tabulky uložené v databázi. ORM nástroj slouží pro zjednodušení práce s databází, protože umožňuje pracovat pouze s objektovým modelem. Tím pádem, hlavní zodpovědností backendové části aplikace je zpracování požadavků od klienta, komunikace s databází a poskytnutí REST rozhraní.

3.1.3 Datová vrstva

Datová vrstva obsahuje databázi, běžící na databázovém serveru (více o databázi v sekci 4.3). Databáze má na starosti uchování dat dle schéma a poskytnutí rozhrání pro práce s daty.

3.2 Návrhový vzor řešení

Návrhový vzor je to obecné řešení problému, který se využívá při návrhu aplikace. Hlavním přínosem návrhového vzoru je garance dobré rozšiřitelnosti a

snadné údržby navržené aplikace, což zjednodušuje samotný vývoj a umožňuje efektivně pracovat s kódem vývojářům, kteří budou později kód dále rozvíjet.

Pro řešení problému vykreslování modelu UML diagramu tříd, v rámci mé bakalářské práce, jsem zvolila návrhový vzor Visitor. Tento vzor je velice užitečný v situacích, když počet se tříd modelu už nemění, ale je potřeba k nim přidávát nějakou další funkcionalitu [19], v našem případě např. funkcionalitu pro vykreslení pomocí HTML5 SVG. V mé práci je třídní hierarchie modelu UML diagramů tříd neměnná, ale může nastat okamžik, kdy bude třeba přidat dalších metodu do jednotlivých tříd modelu. V rámci vzoru Visitor, je pro každou novou funkcionalitu třeba přidávat zvláštní třídu vycházející z abstrakce Návštěvníka (Visitor). V mém případe, v roli návštěvníků budou vystupovat dvě třídy: první třída bude vykreslovat jednotlivé prvky diagramu; druhá třída slouží pro generování databázového modelu.

Principem je, že pro každou novou funkčnost, kterou chceme dodat původnímu modelu, vytvoříme novou třídu. Tato nová třída představuje „návštěvníka“. Instanci tohoto návštěvníka pak předáme původní třídě a ta v podstatě sama na sebe, pomocí double-dispach [19], zavolá odpovídající metodu návštěvníka. Původní třída tedy představuje „navštíveného“. Návštěvník umí vykonat novou akci, navštívený ho přijme a nechá ho se sebou vykonat tuhle novou akci. Takže platí, že kolik bude dodatečných akcí, tolik bude nových návštěvnických tříd.

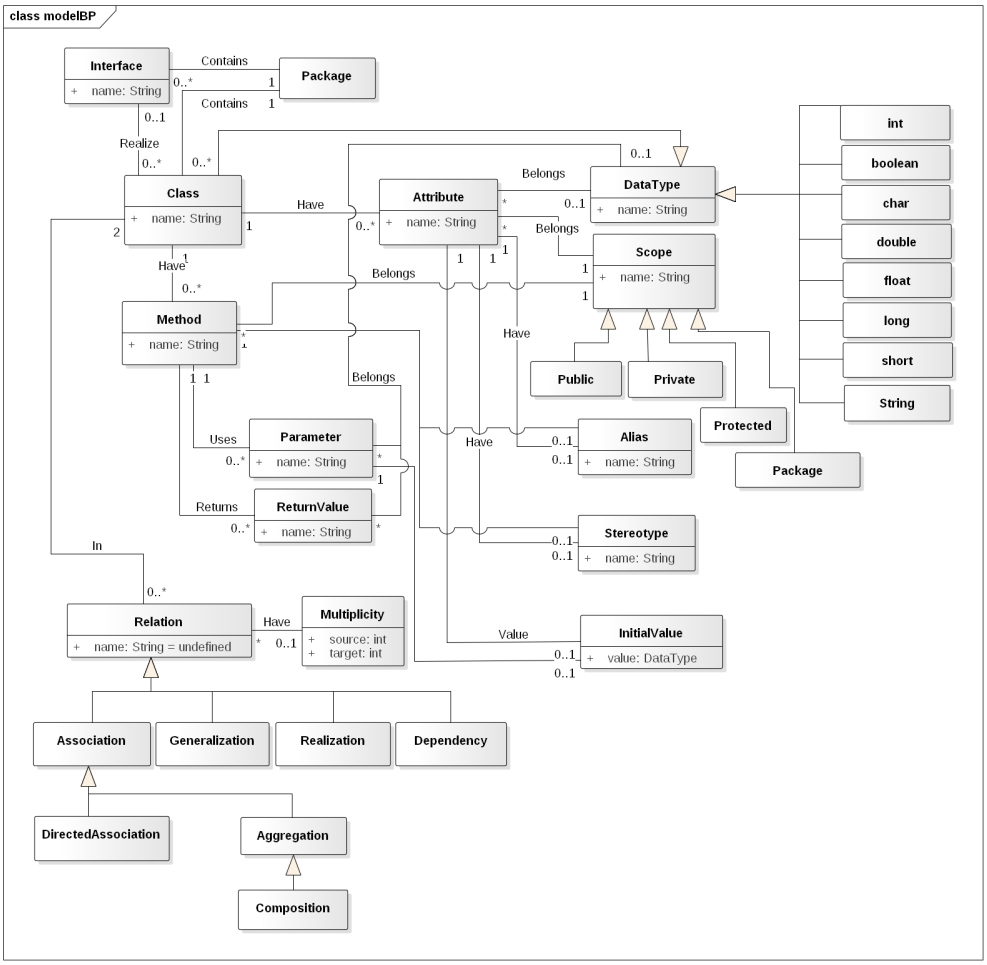
V případě mé bakalářské práce bude Visitor použit pro vykreslení jednotlivých prvků modelu 3.2 a pro generování databázového modelu. Tzn. každý prvek z modelu bude mít metodu `acceptVisitor(aVisitor)` která bude volat metodu pro vykreslení odpovídajícího prvku. Např., pro třídu `Class`, která reprezentuje UML třídu v modelu 3.2 metoda `acceptVisitor(aVisitor)` bude vypadat následujícím způsobem

```
acceptVisitor(aVisitor){
    aVisitor.visitClass(this);
}
```

Metoda `visitClass(this)` s pomocí instanci `aVisitor` buď vykreslí SVG element (konkretně pro UML třídu) anebo vratí definice tabulky pro databáze. Konkretní implementace metody `visitClass(this)` záleží na tom, za jakého návštěvníka zrovna systém používá.

3.3 Model

V této sekce popíši model, který se v mé aplikaci využívá. Pro každý element z UML class diagramu jsem navrhla JS třídu, reprezentující tento element, která obsahuje vlastností patřící do tohoto elementu. Každá třída obsahuje metodu `acceptVisitor(aVisitor)`, která umožňuje Návštěvníku „navštívit“ jednotlivé třídy.



Obrázek 3.2: Doménový model

3.3.1 Doménový model

V této části se věnuji návrhu doménového modelu pro uložení UML diagramů tříd v prostředí JavaScript. Třídy v doménovém modelu neobsahují metody a mají pouze důležité atributy. Doménový model je tedy náčrtem základních entit systému a vztahů mezi nimi. Je platformově nezávislý.

Na diagramu 3.2 je vidět jednotlivé prvky modelu a vztahy mezi nimi. Doménový model popisuje možné elementy UML diagramu tříd. Model je návržen na základě UML specifikace [20].

Diagram tříd obsahuje třídy systému, jejich atributy, metody či operace a také vztahy mezi jednotlivými objekty. Dále popíšu nejdůležitější elementy doménového modelu.

Doménový model obsahuje tyto elementy:

Class je klíčovým elementem UML diagramu tříd. Na UML diagramu tříd se zobrazuje jako obdélník, který obsahuje svůj název a do třídy patřící atributy a metody. Proto je atributem Class název třídy.

Attribute jsou atributy jednotlivých tříd. Samozřejmě atributy taky mají své názvy a proto je v tabulce uveden atribut name. Jedná třída může mít více než jeden anebo žádný atribut, proto multiplicita 0..* na stráně atributu ve vztahu s třídou.

Method jsou metody jednotlivých tříd. Metoda má vždy nějaký název a proto je v tabulce uveden atribut name. Analogicky jako u atributu, jedna třída může mít více než jednu anebo žádnou metodu, proto multiplicita 0..* na stráně metody ve vztahu s třídou.

Parameter metody je hodnota, kterou může metoda vyžadovat pro zpracování. Parametr má svůj název a datový typ do kterého patří. Metoda může nepožadovat žádný parametr anebo jich může vyžadovat několik, tím pádem ve vztahu s metodou má 0..* na stráně parametru.

ReturnValue je návratová hodnota. Metoda může po zpracování vracet nějaké hodnoty, které lze využít pro další běh programu. Stejně jako u parametru, návratová hodnota má název a patří do nějakého datového typu. Na stráně ReturnValue ve vztahu s metodou je 0..*.

DataType je datový typ, který je používán v rámci diagramu. Obecně je datový typ doménou hodnot, kterých smí nabývat proměnná. Datový typ může byt jak primitivní, např. int nebo boolean, tak referenční, což znamená, že je reprezentován nějakou jinou třídou z diagramu. DataType má vztah s atributem, parametrem a s návratovou hodnotou, což znamená že každý ze zmíněných tříd určitě patří do nějakého datového typu.

Scope označuje viditelnost jednotlivých částí diagramu a tříd. Podle doménového modelu má vztah s atributem a metodou, což znamená, že pro tyto je třeba nadefinovat viditelnost.

Relation je relace mezi třídami. Vztah má atribut name, a defaultně je nastaven na undefined, protože název pro relace není povinná vlastnost. Nejdůležitější pro vztah je tvar vztahu a multiplicita. Vztah v UML diagramu tříd je binární relace, to znamená, že vždy spojuje dvě třídy.

Multiplicity vztahu udává, kolik instancí jedné třídy může být svázáno s instancí druhé třídy. Obsahuje dva atributy: source a target, které udávají multiplicity pro počáteční a cílovou třídu ve vztahu.

3.4 Backend

Pojem backend často označuje část aplikace. která je především zodpovědná za komunikaci s klientskou části aplikace, řešení business logiky a správu databázového úložiště.

Databáze obsahuje tabulky a vztahy mezi nimi pro každý element z UML diagramu tříd. ORM nástroj Sequelize slouží pro generování tabulek z mého modelu, přístupu k datům a editaci záznamů v databáze. Pro generování tabulek z mého modelu, navrženému v předchozí sekci, je nutné získat Sequelize definici tabulky pro každou třídu vyskytující se v modelu. Pro tenhle účel jsem použila návrhový vzor Visitor, který mi umožní získáni definice jednotlivých Sequelize tabulek. Navštěvnik, který prochází jednotlivými třídami z modelu a generuje pro ně definice Sequelize modelu. Pomocí ORM nástroje Sequelize můžeme k datům, uloženým v databázi, přistupovat, vkládat nové záznamy, editovat anebo mazat stávající záznamy.

Pro poskytování rozhraní slouží webový framework Express.js. REST rozhraní (více o rozhraní v sekci 3.6) slouží pro komunikace s frontendovou části aplikace. Backend poskytuje rozhraní pro komunikaci, frontend může díky němu posílat požadavky na backendem definované URI a čekat na odpovědi. Pod požadavkem si můžeme představit požadavek od frontendu na výpis všech tříd, které jsou uložené v databázi, odpovědí z backendu by mělo být pole obsahující všechny třídy z databáze. Formát odpovědi z backendu záleží na konkretní implementaci, ale nejčastěji je to formát JSON. Ve své bakalářské práci proto budu používat formát JSON pro komunikace mezi frontendem a backendem.

3.5 Frontend

Pojem frontend slouží k označení části webu viditelné běžným návštěvníkům. Hlavní zodpovědností frontendu je prezentování dat uživateli webové stránky.

Frontend prototypu běží v prohlížeči uživatele a slouží k zobrazování UML diagramu tříd. Uživatel bude moci přidávat a mazat jednotlivé komponenty v diagramu, posouvat anebo měnit velikost komponent atd. Pro implementace frontendové části aplikace je použita knihovna ReactJS. Pro reprezentaci jednotlivých komponent jsem si využila technologii SVG, která je podporována v HTML5.

Hlavní komponentou mého prototypu je komponenta `Diagram`. `Diagram` je zodpovědný za zobrazení UML diagramu tříd. V rámci `Diagramu` lze elementy posouvat a měnit jejích velikost.

`Diagram` jako React komponenta má na vstupu několik parametrů, jeden z nich je `model`, reprezentující obsah `Diagramu`. akt přidání dalšího elementu do diagramu tříd, v důsledku znamená přidání instance do `modelu` diagramu a jeho následné vykreslení komponentou `Diagram`.

Pro každý element z modelu existuje React komponenta, která pomocí SVG reprezentuje pohled na element. Např. pro Class existuje React komponenta obsahující SVG definici obdélníku s názvem třídy. Implementace jednotlivých komponent umožňuje korektní zobrazení prvků diagramu a taky slouží k přidání takových vlastností jako: pohyblivost a změna velikosti.

Další React komponentou v mém systému je `Panel`. `Panel` obsahuje ovladácí prvky pro práci s diagramem. Např. tlačítko na přidání třídy do `Diagramu`.

Frontendová část aplikace realizuje architektonický vzor: single page application (SPA), což znamená že se používá jenom jedná webová stránka a obsah stránky se dynamicky, pomocí JavaScriptu, mění. Pomáhá v tom technologie AJAX (Asynchronous Javascript and XML), jejíž hlavním principem je komunikace s backend na pozadí, tím pádem odpadá nutnost znovunačítání celé webové stránky při jakékoliv změně. Pro komunikace pomocí AJAX jsem si zvolila formát JSON. A tedy, v případě, že na frontendu vznikne nutnost načtení nových dat z databáze, pošle se na pozadí požadavek na některý z end-pointů REST rozhraní backendu (více v sekci 3.6) a nová informace se objeví na webové stránce pomocí JavaScriptu bez nutnosti znovu-načitání této webové stránky.

3.6 REST API

Základním způsobem komunikace mezi klientskou a serverovou části aplikace je REST komunikace.

Representational State Transfer (REST) je koncept pro návrh distribuované architektury. Distribuovaná architektura v tomto smyslu znamená, že části programu běží na různých strojích a pro svojí komunikaci využívají síť. Pod programem si můžete představit například webovou aplikaci, kde internetový prohlížeč komunikuje s webovým serverem, aplikaci pro výměnu dat mezi finančními institucemi, kde dochází k vzájemnému volání mezi servery. [21]

Všechny zdroje, tzv. end-pointy, mají vlastní identifikátor URI a REST definuje čtyři základní metody pro přístup k nim.

V této sekce popíšu jednotlivé end-pointy RESTového rozhraní, které poskytuje backendová část mé aplikace.

Backend podporuje následující end-pointy:

- /api/classes[/:id],
- /api/attributes[/:id],
- /api/methods[/:id],
- /api/interfaces[/:id],
- /api/packages[/:id],
- /api/datatypes[/:id],

- /api/packages[/:id],
- /api/parameters[/:id],
- /api/return_values[/:id],
- /api/relations[/:id],
- /api/multiplicities[/:id],
- /api/scopes[/:id],
- /api/aliaces[/:id],
- /api/stereotypes[/:id],
- /api/relations_classes[/:id],

Každý end-point poskytuje HTTP metody:

GET vrací kolekci entit,

GET s parametrem *id* vrací entitu s konkretním *id*,

POST vytvoří nonou entitu,

PUT s parametrem *id* edituje entitu s konkretním *id*,

DELETE s parametrem *id* smaže entitu s konkretním *id*

Pro nejdůležitější end-pointy krátce popíšu pro jaké účely slouží, ukážu které HTTP metody podporují, co jednotlivé metody vrátí a jaké informace vyžadují. End-pointy bez parametru *id* podporují HTTP metody GET a POST. Koncové body s parametrem *id* podporují HTTP metody GET, PUT a DELETE.

api/classes[**/:id**]

`api/classes`

Koncový bod, obsahující kolekci všech tříd, existujících v databázi. Více o koncovém bodu lze nalézt v tabulce 3.1.

Tabulka 3.1: api/classes

Metoda	Popis	Tělo požadavku	Úspěch	Neúspěch
GET	Koncový bod pro výpis všech tříd existujících v databázi	žádné	200	500
POST	Koncový bod pro přídání nové třídy	žádné	204	500

`api/classes/id`
Koncový bod, obsahující jednu třídu z databáze, s zadaným *id*. Více o koncovém bodu lze nalézt v tabulce 3.2.

Tabulka 3.2: api/classes/id

Metoda	Popis	Tělo požadavku	Úspěch	Neúspěch
GET	Koncový bod pro výpis třídy s zadyným *id*	žádné	200	500
PUT	Koncový bod pro změnu třídy existující v databázi s daným *id*	údaje třídy v formátu JSON	201	500
DELETE	Koncový bod pro smazání třídy existující v databázi s daným *id*	žádné	204	500

api/attributes[**/:id**]
`api/attributes`
Koncový bod, obsahující kolekci všech atributů, existujících v databázi. Více o koncovém bodu lze nalézt v tabulce 3.3.

Tabulka 3.3: api/attributes

Metoda	Popis	Tělo požadavku	Úspěch	Neúspěch
GET	Koncový bod pro výpis všech atributů existujících v databázi	žádné	200	500
POST	Koncový bod pro přídání nového atributu	žádné	204	500

`api/attributes/id`
Koncový bod, obsahující jeden atribut z databáze, s zadaným *id*. Více o koncovém bodu lze nalézt v tabulce 3.4.

api/methods[**/:id**]
`api/methods`
Koncový bod, obsahující kolekci všech metod, existujících v databázi. Více o koncovém bodu lze nalézt v tabulce 3.5.

Tabulka 3.4: api/attributes/id

Metoda	Popis	Tělo požadavku	Úspěch	Neúspěch
GET	Koncový bod pro výpis atributu s zadyným *id*	žádné	200	500
PUT	Koncový bod pro změnu atributu existujícího v databázi s daným *id*	údaje atrubutu v formátu JSON	201	500
DELETE	Koncový bod pro smazání atributu existujícího v databázi s daným *id*	žádné	204	500

Tabulka 3.5: api/methods

Metoda	Popis	Tělo požadavku	Úspěch	Neúspěch
GET	Koncový bod pro výpis všech metod existujících v databázi	žádné	200	500
POST	Koncový bod pro přídání nové metody	žádné	204	500

`api/methods/id`
Koncový bod, obsahující jednu metodu z databáze, s zadaným *id*. Více o koncovém bodu lze nalézt v tabulce 3.6.

Tabulka 3.6: api/methods/id

Metoda	Popis	Tělo požadavku	Úspěch	Neúspěch
GET	Koncový bod pro výpis metody s zadyným *id*	žádné	200	500
PUT	Koncový bod pro změnu metody existující v databázi s daným *id*	údaje metody v formátu JSON	201	500
DELETE	Koncový bod pro smazání metody existující v databázi s daným *id*	žádné	204	500

api/relations[**/:id**]

`api/relations`

Koncový bod, obsahující kolekci všech relací, existujících v databázi. Více o koncovém bodu lze nalézt v tabulce 3.7.

Tabulka 3.7: api/relations

Metoda	Popis	Tělo požadavku	Úspěch	Neúspěch
GET	Koncový bod pro výpis všech relací existujících v databázi	žádné	200	500
POST	Koncový bod pro přídání nové relace	žádné	204	500

`api/relations/id`

Koncový bod, obsahující jednu relaci z databáze, s zadaným *id*. Více o koncovém bodu lze nalézt v tabulce 3.8.

Tabulka 3.8: api/relations/id

Metoda	Popis	Tělo požadavku	Úspěch	Neúspěch
GET	Koncový bod pro výpis relace s zadyným *id*	žádné	200	500
PUT	Koncový bod pro změnu relace existující v databázi s daným *id*	údaje relace v formátu JSON	201	500
DELETE	Koncový bod pro smazání relace existující v databázi s daným *id*	žádné	204	500

api/multiplicities[**/:id**]

`api/multiplicities`

Koncový bod, obsahující kolekci všech multiplicit pro relace, existujících v databázi. Více o koncovém bodu lze nalézt v tabulce 3.9.

`api/multiplicities/id`

Koncový bod, obsahující jednu multiplicitu z databáze, s zadaným *id*. Více o koncovém bodu lze nalézt v tabulce 3.10.

api/relations_classes[**/:id**]

Tabulka 3.9: api/multiplicities

Metoda	Popis	Tělo požadavku	Úspěch	Neúspěch
GET	Koncový bod pro výpis všech multiplicit existujících v databázi	žádné	200	500
POST	Koncový bod pro přídání nové multiplicity	žádné	204	500

Tabulka 3.10: api/multiplicities/id

Metoda	Popis	Tělo požadavku	Úspěch	Neúspěch
GET	Koncový bod pro výpis multiplicity s zadyným *id*	žádné	200	500
PUT	Koncový bod pro změnu multiplicity existující v databázi s aným *id*	údaje multiplicity v formátu JSON	201	500
DELETE	Koncový bod pro smazání multiplicity existující v databázi s daným *id*	žádné	204	500

Třída a relace, podle navrženého modelu mají vztah M:N, z čěhož vyplývá existence pomocné tabulky, reprezentující tuto vazbu.

`api/relations_classes`
Koncový bod, obsahuje kolekci všech relací mezi třídy Class a Relation, existujících v databázi. Více o koncovém bodu lze nalézt v tabulce 3.11.

Tabulka 3.11: api/relations_classes

Metoda	Popis	Tělo požadavku	Úspěch	Neúspěch
GET	Koncový bod pro výpis všech relací existujících v databázi	žádné	200	500
POST	Koncový bod pro přídání nové relace	žádné	204	500

`api/relations_classes/id`

Koncový bod, obsahující jednu relaci mezi třidy Class a Relation z databáze, s zadaným *id*. Více o koncovém bodu lze nalézt v tabulce 3.12.

Tabulka 3.12: api/relations_classes/id

Metoda	Popis	Tělo požadavku	Úspěch	Neúspěch
GET	Koncový bod pro výpis relace s zadyným *id*	žádné	200	500
PUT	Koncový bod pro změnu relace existující v databázi s daným *id*	údaje relace v formátu JSON	201	500
DELETE	Koncový bod pro smazání relace existující v databázi s daným *id*	žádné	204	500

api/parameters[**/:id**]

`api/parameters`

Koncový bod, obsahující kolekci všech vstupních parametrů, existujících v databázi. Více o koncovém bodu lze nalézt v tabulce 3.13.

Tabulka 3.13: api/parameters

Metoda	Popis	Tělo požadavku	Úspěch	Neúspěch
GET	Koncový bod pro výpis všech vstupních parametrů existujících v databázi	žádné	200	500
POST	Koncový bod pro přídání nového vstupního parametru	žádné	204	500

`api/parameters/id`

Koncový bod, obsahující jeden vstupní parametr z databáze, s zadaným *id*. Více o koncovém bodu lze nalézt v tabulce 3.14.

api/return_values[**/:id**]

`api/return_values`

Koncový bod, obsahující kolekci všech navratových hodnot, existujících v databázi. Více o koncovém bodu lze nalézt v tabulce 3.15.

Tabulka 3.14: api/parameters/id

Metoda	Popis	Tělo požadavku	Úspěch	Neúspěch
GET	Koncový bod pro výpis vstupního parametru s zadyným *id*	žádné	200	500
PUT	Koncový bod pro změnu vstupního parametru existující v databázi s daným *id*	údaje vstupního parametru v formátu JSON	201	500
DELETE	Koncový bod pro smazání vstupního parametru existující v databázi s daným *id*	žádné	204	500

Tabulka 3.15: api/return_values

Metoda	Popis	Tělo požadavku	Úspěch	Neúspěch
GET	Koncový bod pro výpis všech navratových hodnot existujících v databázi	žádné	200	500
POST	Koncový bod pro přídání nové navratové hodnoty	žádné	204	500

`api/return_values/id`
Koncový bod, obsahující jednu navratovou hodnotu, s zadaným *id*. Více o koncovém bodu lze nalézt v tabulce 3.16.

3.7 Návrh uživatelského rozhraní

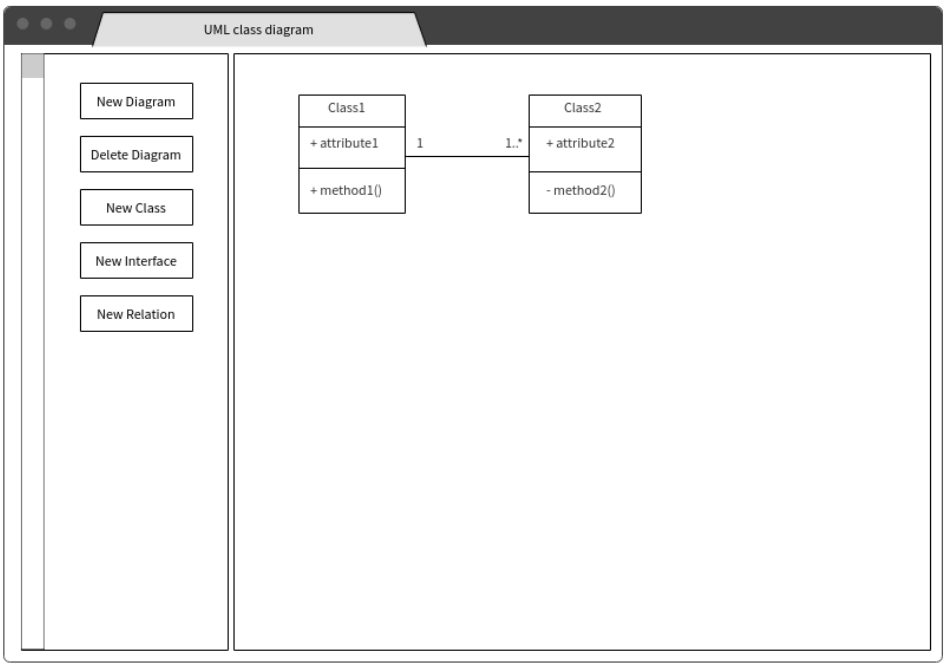
Prototyp nástroje realizuje architekturu SPA, což znamená, že všechno bude běžet na jedné webové stránce a tím pádem musím navrhnout jen jednu obrazovku. Návrh uživatelského rozhraní je znázorněn na obrazce 3.3. Aplikace pro kreslení diagramů bude běžet v prohlížeči uživatele. Obrazovka se dělí na dvě části: tzv. Panel vlevo a samotný Diagram vpravo.

Panel obsahuje ovládací prvky pro práci s diagramem. Podle návrhu bude obsahovat tlačítka, která jsou zodpovědná za funkčnost mé aplikace.

Diagram jak je zřejmé z názvu je samotný UML diagram tříd, kde jsou znázorněné elementy diagramu. Elementy lze v rámci Diagramu posouvat a měnit jejich velikost.

Tabulka 3.16: api/return_values/id

Metoda	Popis	Tělo požadavku	Úspěch	Neúspěch
GET	Koncový bod pro výpis navratové hodnoty s zadyným *id*	žádné	200	500
PUT	Koncový bod pro změnu navratové hodnoty existující v databázi s daným *id*	údaje navratové hodnoty v formátu JSON	201	500
DELETE	Koncový bod pro smazání navratové hodnoty existující v databázi s daným *id*	žádné	204	500



Obrázek 3.3: Návrh uživatelského rozhraní

Kapitola 4

Implementace

V této kapitole je popsán postup při implementaci prototypu webového nástroje pro kreslení UML diagramů tříd. Při implementaci prototypu jsem se řídila návrhem popsaným v předchozí kapitole. Dále tato kapitola obsahuje popis vývojového prostředí, popis použitého verzovacího systému a popis použité databáze. Taktéž se v této kapitole věnuji popisu použitých nástrojů při implementace.

4.1 Vývojové prostředí

Jako vývojové prostředí jsem si zvolila WebStorm protože je to mocný nástroj pro moderní vývoj v jazyce JavaScript. Toto IDE nabízí podporu moderních frameworků pro JavaScript, přistup k databázi, možnost debuggování a spoustu dalších funkcionalit, potřebných pro pohodlný vývoj. Velkou vyhodou je integrace s Version Control Systems, jako např. GIT. Nejdůležitější vlastnosti WebStormu, které se mi budou při vývoji, je možnost využívání frameworku ReactJS na stráně klienta, integrace běhového prostředí Node.js pro JavaScript a podpora frameworku Express.js [22].

4.2 Verzovácí systém

Verzování se používá pro uchování historie veškerých změn, primárně u zdrojového kódu. Verzování umožňuje sledovat historii změn v kódu, porovnávat progres během vývoje a taky mít vždy přehled o aktuálním stavu. Pokud se něco stane v lokálním prostředí, například přijdete o zdrojový kód na Vašem počítači, verzovácí systém vám dokáže s tímto problémem pomoci. Je tedy velmi výhodné nějaký takový systém při vývoji použít, už jen proto, že tak lze zabezpečit svou odvedenou práci a strávený čas. Verzování a sledování historie změn taktéž velmi pomáhá při odhadu časové náročnosti práce.

Pro verzování aplikace, která je napsaná v rámci této bakalářské práce, byl použit verzovácí systém GIT. Konkretně používám školní GitLab. Pomocí GitLabu uchovávám změny v kódu jak na backendové části (repozitář backend), tak a na frontendové části (repozitár frontend, historicky bohužel pojmenovaný BP). Navíc GitLab podporuje issue tracking, který usnadňuje management pracovních úkolů. V jednom systému tedy GitLab nabízí přehled o kódu a seznam toho co ještě zbývá udělat.

4.3 Databáze

Pro účely své bakalářské práce jsem si zvolila objektově-relační databázový systém PostgreSQL. Hlavní důvody pro tuto volbu jsou následující.

Za prvé, PostgreSQL je primárně vyvíjen pro GNU/Linux resp. pro unixové systémy obecně. Tím že při vývoje používám operační systém Ubuntu, použití PostgreSQL je tedy snadné a výhodné. Tento systém je free a open-source, tudíž jej mohu používat bez nutnosti platit licenční poplatky, přitom však přináší vlastnosti dobře navrženého a podporovaného systému. V rámci mé bakalářské práce nepracuji s velikým množství dat, a proto i z tohoto hlediska, je pro mě funkcionalita PostgreSQL dostatečná.

Za druhé, zvolené vývojové prostředí podporuje integrace databáze PostgreSQL a umožňuje přistupovat k datům přímo během vývoje. Na změny dat v databázi při vývoji se mohu okamžitě podívat přímo v IDE WebStorm. Toto usnadňuje a urychluje vývoj aplikace. I v případě, že jiné IDE neumožňuje rychlý přístup k databázi PostgreSQL, existuje velké množství rozhraní pro správu této databáze. Jeden z takových příkladu je grafické administrační rozhraní pro PostgreSQL s názvem pgAdmin. Je to free a open source rozhraní, které umožňuje mít pod kontrolou všechno co se děje v databázi. Program pgAdmin má přehledné uživatelské rozhraní. Existence takových rozhraní je tedy také argumentem pro použití databázového systému PostgreSQL.

Za třetí, argumentem pro volbu tohoto databázového systému je i to, že ORM nástroj Sequelize tento BDS podporuje a existuje mnoho dokumentace ohledně používáni PostgreSQL ve spojení s Express a Sequelize.

Posledním důvodem jsou mé předchozí dobré zkušenosti s PostgreSQL.

4.4 npm

Nástroj npm je balíčkovací systém pro prostředí Node.js. Nástroj npm slouží ke stahování závislostí, tedy knihoven a modulů potřebných pro projekt. *Nástroj npm není jediný správce balíčků pro Node.js, ale je zdaleka nejpoužívanější a je součástí základní instalace Node.js* [23].

Modul, nebo taky balík, je jeden nebo několik JavaScriptových souborů, které dohromady tvoří nějakou knihovnu nebo nástroj. S npm se pracuje přes příkazový řádek a libovolný modul se instaluje příkazem `npm install`.

Jakmile použiji příkaz `npm install`, pak se automaticky v projektu objeví složka node_modules, do které npm stáhne všechny požadované moduly. Existuje dvě možnosti stahování modulů: lokálně a globálně. Lokální instalace modulů znamená, že npm udělá složku node_modules přímo v projektu a dotáhne všechny potřebné moduly přímo tam, to mimo jiné znamené, že moduly které byly stáhnuty pro jeden projekt, jiný projekt neuvidí. Druhá varianta je globální stáhnutí, kdy moduly po instalaci budou dostupné všem projektům, které to potřebují.

Ke konfiguraci npm se používá soubor package.json. Tento soubor obsahuje popis závislostí, potřebných pro projekt, název a číslo verze projektu a další meta informace. Pokud si nějaký projekt s tímto souborem stáhnete, pak se po zadaní `npm install` všechny moduly automaticky nainstalují. Dále je možnost definovat v tomto souboru skripty, které v procesu vývoje chcete spouštět. Např. můžete definovat skript pro build, start anebo deploy vaší aplikace.

4.5 Babel

Babel je kompilátor, který umí převádět ES6 do čistého JavaScriptu, podporovaného všemi moderními prohlížeči. Tento kompilátor potřebujeme, protože některé prohlížeče nemají podporu ES6 a tedy nelze moji aplikaci v takovém prohlížeči, bez využití dodatečného nástroje pro překlad, sputit. Babel plně integruje JSX a React. JSX je syntaktické rozšíření JavaScriptu umožňující vkládat HTML tagy přímo uprostřed JavaScriptového kódu. JSX syntaxe ve spojení s Reactem nabízí mocný nástroj, jenž umožňuje použití React komponent vkládáním do HTML tagů, což vede na pochopitelným a přehledným kód. Hlavní výhodou kompilátoru je tedy možnost využit všech nových funkcionalit Reactu a ES6 bez ohledu na to, co současně prohlížeče podporují, o vše se za mě postará Babel.

4.6 ESLint

ESLint je nástroj na kontrolu JavaScript kódu umožňující kontrolu syntaxe. Pro tento nástroj existují React pluginy, které kontrolují dodržování best practices. Další výhoodou je, že je nástroj ESLint podporován v mém vývojovém prostředí – Webstromu. Konfigurační soubor .eslintrc.json obsahuje sadu pravidel, podle kterých bude ESLint kontrolovat kód. Např. jaké uvozovky se používájí pro řetězce, jaké je odsazení pro new line a další. Kromě zmíněných nastavení konfigurační soubor obsahuje konfigurace prostředí ve kterém je ESLint používán, definice pluginu a nastavení jakou specifikaci JavaScriptu chceme používat. V rámci konfigurace je možné dědit od jiných konfiguračních souborů, což usnadňuje přenositelnost a udržovatelnost pravidel. [24]

4.7 Struktura projektu

Táto sekce obsahuje popis členění zdrojového kódu mého projektu. Patří sem dvě hlavní, již zmíněné, části: backend a frontend, kde každá z nich má svou strukturu.

Pro sestavení backend a frontend části aplikace jsem používala nástroje pro generování aplikací, proto je struktura částečně dána použitím těchto nástrojů. Pro backendovou část jsem použila `express-generator`. Pro frontendovou část jsem použila `create-react-app`.

4.7.1 Backend

Struktura backend části je znázorněna na obrázku 4.1.

Backend obsahuje tyto složky:

- db-scripts,
- node_modules,
- public,
- src

Nejdůležitější složkou backend části je složka src. Tato obsahuje veškerou logiku serverové části aplikace. Složka src obsahuje další podsložky:

api obsahuje logiku routování aplikace. Do složky patří konfigurace dostupných URL, definice routování, implementace REST rozhraní, podle návhru ze sekce 3.6.

middleware obsahuje middleware soubory, které mají přístup k objektům HTTP požadavků a HTTP odpovědím. Umožňují mezizpracování HTTP požadavků, než se odešle odpověď. Hlavní výhodou je to, že middleware má přístup k HTTP požadavku a muže ho změnit předtím, než proběhne vlastní zpracování požadavku pomocí správného routu.

Model obsahuje definice elementů v Modelu podle návrhu (viz 3.3). Frontend a backend využívají stejný model, pro zajištění konzistence. Model, který je využíván v obou částech mé aplikace patří do front části a v backendu je model symbolickým odkazem na frontend model.

Složka db-scripts obsahuje databázové skripty, které pomáhají při manipulaci se zvolenou databází (viz 4.3).

Složka public obsahuje generovaný kód po buildování mé aplikace.

Složka node_modules obsahuje všechny potřebné moduly pro sestavení mé aplikace, které jsou dotahovány nástrojem npm (viz 4.4).

Dále backend obsahuje následující soubory:

config.json obsahuje konfigurace pro ORM nástroj Sequelize a pro Express. Do config.json patří konfigurace databáze, popis portu na kterém aplikace bude běžet a další nastavení. Jak je zřejmě z přípony jedná se o JSON soubor.

index.js je hlavním souborem Express aplikace.

AbstractVisitor.js je odkazem na třídu `AbstractVisitor` na frontendu. Je to rozhraní pro práci se zvoleným návrhovým vzorem, více v kapitole 4.8.

GetSequelizeDef.js je implementací Visitoru pro získání definice ORM Sequelize, více v sekce 4.8.3.

.gitignore obsahuje seznam adresářů a souborů, které nechceme verzovat pomocí verzovacího nástroje (viz 4.2).

makeLinkToModel.sh je skript který slouží pro správné nastavení složky Model. Skript obsahuje příkaz pro vytvoření symbolického linku na frontend model.

package.json konfigurace pro nástroj npm (viz 4.4) a pro celý projekt. Obsahuje popis závislostí, které jsou nutné pro sestavení aplikace, a které jsou dotahované nástrojem npm. Dále obsahuje skripty pro build a start aplikace. Poslední co obsahuje, je obecný popis projektu, např. název a verze.

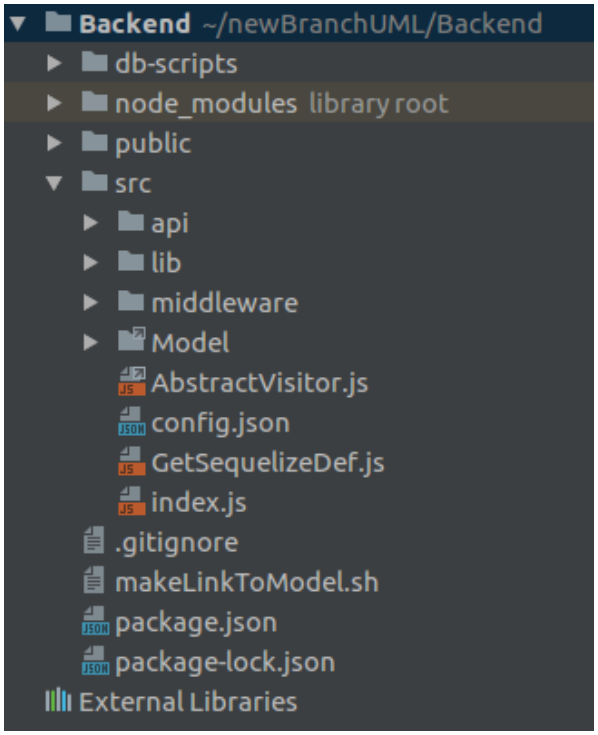
4.7.2 Frontend

Struktura frontend části je znázorněna na obrazce 4.2.

Frontend obsahuje tyto složky:

- lib
- Model
- node_modules
- public
- RenderedModel
- src

Nejdůležitější složkou frontendové části aplikace je složka src. Tato obsahuje veškerou logiku pro klientskou část aplikace, hlavně logiku vykreslení UML diagramů tříd a jednotlivých dílčích komponent.



Obrázek 4.1: Backend

Složka Model obsahuje definice jednotlivých tříd podle návrhu 3.3 v prostředí JavaScript. Právě táto složka je model kam odkazuje model z backend části aplikace.

Složka node_modules, stejně jako u backendové části, obsahuje moduly a knihovny, stáhnuté pomocí nástroje npm (viz 4.4). Této složka se generuje automaticky po zadání příkazu `npm install`.

Do složky public patří vygenerovaný kód po spuštění build skriptu ze souboru package.json. Podobně, jako předchozí složka je generovaná automaticky.

Složka RenderedModel obsahuje definice React komponent pro vykreslení jednotlivých části diagramu. Např. do této složky patří třída RenderedClass, která pomocí HTML5 SVG grafiky udává jak třída v rámci mého diagramu musí vypadat. Definice každého elementu zvlášť je velmi důležitá z několika důvodů. První je, že definice elementů takovým způsobem pomáhá zajistit snadnou udržovatelnost mé aplikace a zároveň i snadnou rozšiřitelnost. Druhý důvod je v tom, že definice každého elementu diagramu jako React komponentu umožňuje manipulace s částmi obdobně jako s běžnými React komponentami a ne jako pouze s prostou SVG grafikou, tak jak ji definuje HTML5. Toto se projevuje v tom, že můžu každému elementu přidávat tzv. eventy, jež umožňují např. přetahování elementu napříč diagramem. Poslední důvod je v tom, že definice grafických elementů v jednotlivých třídách pomáhá zacho-

vat konzistenci pohledu a usnadňuje ovládání grafické části aplikace. Pokud by definice pohledů elementů byla někde uprostřed kódu a na různých místech, tak by to velmi komplikovalo případné a pravděpodobné změny pohledů na elementy diagramu.

Frontend část aplikace obsahuje další soubory:

.babelrc je konfigurační soubor pro nástroj Babel, který je popsán v sekci 4.5. Babel slouží pro sestavení React aplikace.

.eslintrc.json je konfigurační soubor pro nástroj ESLint, popsaný v sekci 4.6.

.gitignore obsahuje seznam adresářů a souborů, které nechceme verzovat pomocí verzovacího nástroje (viz 4.2).

package.json stejně jako u backendu obsahuje konfiguraci pro nástroj npm (viz 4.4) a pro celý projekt. Obsahuje popis závislostí, které jsou nutné pro sestavení aplikace a které jsou dotahované nástrojem npm. Dále obsahuje skripty pro build a start aplikace. Poslední co obsahuje, je obecný popis projektu, např. název a verze.

webpack.config.js je konfigurační soubor pro nástroj Webpack. Nástroj Webpack slouží k vytváření JavaScriptových balíčků z modulárního JS kódu pro použití v prohlížeči. Jelikož pro pro sestavení projektu jsem dříve používala Webpack, ale ne npm skripty, ten konfigurační soubor zde zůstal spíše jako opatření pro situaci, když vznikne nutnost zpětného používání Webpacku.

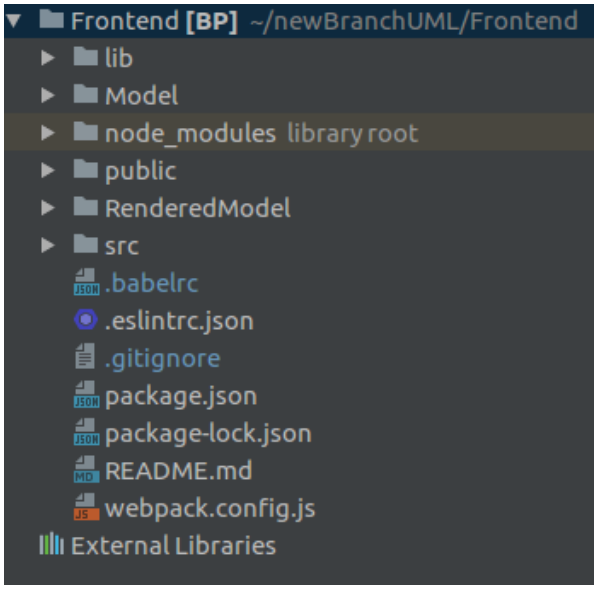
4.8 Využití návrhového vzoru

Jak jsem zmiňovala v sekci 3.2, pro účely své aplikace jsem si použila návrhový vzor Visitor. V této kapitole podrobně popíšu implementaci tohoto vzoru a využití v kontextu mé aplikace. Návrhový vzor Visitor byl použit, protože elegantně řeší problém přidání nějaké nové funkčnosti pro každou třídu navrženého modelu. Nejdůležitější je si uvědomit, že se model už nemění, a proto můžeme navrhnout abstraktní rozhraní *AbstractVisitor*, které musí splňovat každá další konkrétní implementace.

4.8.1 AbstractVisitor

`AbstractVisitor` je abstraktní rozhraní Visitoru, které musí splňovat každá další třída, implementující nějakou novou funkčnost. `AbstractVisitor` obsahuje určitou sadu metod, a využívá fakt, že se navržený model již nemění. `AbstractVisitor` je definován na frontendu, na backendu se používá odkaz na tuto třídu.

`AbstractVisitor` obsahuje metody:



Obrázek 4.2: Frontend

- `visitClass()`
- `visitAttribute()`
- `visitMethod()`
- `visitRelation()`
- `visitInterface()`
- `visitPackage()`
- `visitParameter()`
- `visitReturnValue()`
- `visitRelationClass()`
- `visitMultiplicity()`

Každá třída z modelu obsahuje metodu `acceptVisitor(aVisitor)`, která zavolá konkretní odpovídající metodu z abstraktního rozhraní pro zpracování. Výhodou je to, že `acceptVisitor(aVisitor)` se volá z nějaké konkretní třídy, což znamená, že je známo jaká konkrétní metoda bude volána. Např. pokud zavoláme metodu `acceptVisitor(aVisitor)` z třídy pro atribut, tak metoda bude vypadat takhle:

```
acceptVisitor(aVisitor){
    aVisitor.visitAttribute(this);
}
```

Jak bude implementována metoda `visitAttrribute(this)` záleží na konkretním Visitoru. Např. pro Visitor vykreslení elementů se vrátí v metodě `visitAttribute(this)` definice jak se má atribut vykreslit, ale Visitor pro ORM nástroj ve stejné metodě vrátí Sequelize definice tabulky pro atribut.

4.8.2 ReactSvgRenderer

`ReactSvgRenderer` je návštěvník, implementující rozhraní `AbstractVisitor` a slouží pro vykreslení jednotlivých elementů na diagramu. Ve třídě `Diagram`

```
const renderer = new ReactSvgRenderer(model, this);

var renderedModel = [];
if(model.length)
{
        model.forEach(function (el){
                el.acceptVisitor(renderer);
        });

        renderedModel = renderer.getRenderModel();
}
```

Ve třídě `Diagram` se inicializuje nová instance třídy `ReactSvgRenderer` do které se předává `model` a instance samotného `Diagramu`. `Model` je pole obsahující prvky diagramu. Pokud v `Modelu` něco je, spustí se cyklus, ve kterém se pro každý element z `Modelu` zavolá metoda `acceptVisitor(renderer)`. Jako parametr dostává tato metoda instanci `ReactSvgRenderer`. Pro ukázku dalšího průběhu vykreslování budu používat třídu `Attribut`. Ve třídě `Attribute`

```
acceptVisitor(aVisitor) {
        aVisitor.visitAttribute(this);
}
```

Metoda `acceptVisitor(aVisitor)` přijímá instanci `ReactSvgRenderer` a volá metodu `visitAttribute(this)`, kde jako parameter vloží svou instanci.

Ve tříde `ReactSvgRenderer`

```
visitAttribute(attr, attrPosDimY) {
        const self = this;
        const classPosDim = self.getPosDimByKey(attr.class.key);
        if(!classPosDim) return;
```

```
        let element = (
                <g key={attr.key} fill="#f0f0f0" stroke="#000000" >
                        <text x={classPosDim.x + 30} y={attrPosDimY}
                        stroke="none" fill="black">
                                {attr.name}
                        </text>
                </g>
        );

        this.renderedAttributes.push(element);
}
```

Metoda `visitAttribute()` se volá z metody `visitClass()`, tím pádem dojde k přidání dalšího parametru pro vykreslování atributu v rámci konkrétní třídy. `classPosDim` obsahuje parametry zadané třídy, takové jako výška, šířka a počáteční souřadnice. Metoda `visitAttribute()` vrací textový element pro atribut a přidává ho do pole všech elementů, které musí se vykreslit v rámci zadané třídy na diagramu. Jakmile všechny elementy budou navštíveny, `ReactSvgRenderer` vratí pole s SVG elementy zpět do `Diagramu`, kde budou vykresleny.

4.8.3 GetSequelizeDef

`GetSequelizeDef` je návštěvník, implementující rozhraní `AbstractVisitor`. Postup získání definice tabulky pomocí návrhového vzoru Visitor probíhá stejně jako vykreslení u `ReactSvgRenderer`. Jen pro ukázku zde uvedu metodu `visitAttribute()` ve třídě `GetSequelizeDef`.

```
visitAttribute(){
        const seqModelName = 'attribute';
        let seqModelOfAttribute = this.sequelize.isDefined(seqModelName)
                ? this.sequelize.model(seqModelName)
                : this.sequelize.define(seqModelName, {
                name: this.sequelizeDataTypes.STRING
        });


        /* define 1:N relation between Attribute and Class */

        seqModelOfAttribute.belongsTo(this.visitClass());
        return seqModelOfAttribute;
}
```

4.9 BackendFacade

Třída `BackendFacade` se nachází ve frontendové části aplikace a má za odpovědnost komunikaci s backendem. Umí posílat požadavky na end-pointy backendu a umí zpracovávat odpovědi. Třída `BackendFacade` slouží hlavně pro práci s databází, získání, vkládaní a editace dat, existujících v databázi, pomocí REST komunikace s backendem (viz 3.6).

`BackendFacade` volá se, když frontend požaduje nějaké informace z backendové databáze. Nejčastěji dochází k volání metody `loadDiagram()`, která vrací seznam všech tříd a vztahů v databázi. Zejména táto metoda se volá, když je potřeba překreslit diagram, např. když došlo k přenačtení stránky.

```
loadDiagram(successFnc, failFnc){
        const self = this;

        this.classes = [];
        this.attributesCache = [];
        this.methodsCache = [];

        this.relations = [];
        this.relationsCache = [];
        this.relationsClassesCache = [];
        this.multiplicityCache = [];

        this.loadingDoneFunctions = {
                success : [successFnc]
                ,failure : [failFnc]
        };

        fetch(this.url + '/api/classes')
                .then(res => res.json())
                .then(res => {
                        self.classesCache = res.map( e => Class.fromJson(e) );

                        self.classesCache.forEach((aClass)=>{
                                self.loadMethodsForClass(aClass);
                                self.loadAttributesForClass(aClass);
                                self.loadRelationsForClass(aClass);
                        });
                }).catch( () => self.loadingFailed() );
}
```

V této metodě se používá metoda `fetch()`, která odpovídá za komunikaci s backendem. Metoda `fetch()` je alternativa pro XmlHttpRequest. AJAX je

technologie vývoje webových aplikací s použitím XMLHttpRequest pro komunikaci se servery. Může odesílat a přijímat informace v různých formátech, včetně JSON, XML, HTML a textových souborů. Nejpozoruhodnější charakteristikou AJAXu je její „asynchronní“ povaha, což znamená, že může komunikovat se serverem, vyměňovat data a aktualizovat stránku bez nutnosti obnovovat stránku. XMLHttpRequest je rozhraní umožňující webovým aplikacím komunikaci mezi serverem a klientem prostřednictvím protokolu HTTP. [25]

Metoda `fetch()` má za parametry povinný URL, kam se posílají požadavky a nepovinný parametr options. Do options mohou patřit údaje jako např. metoda HTTP požadavku, tělo požadavku nebo hlavička požadavku. Pokud parametr options není uveden, posílá se defaultně GET request na zadaný URL.

Metoda `fetch()` je založená na slibech (Promises), což zvyšuje přehlednost rozhraní oproti XMLHttpRequest a zjednodušuje syntaxi. Pro práce s vracenou odpovědí je nutné nejdříve zavolat metodu `json()`.

Všchna potřebná pole jsou prazdná, protože je to první metoda, která se volá, když je nutnost načtení celého diagramu, např. při obnovení webové stránky.

4.10 Implementace přidání nového elementu do diagramu

Procesu přidávaní nového elementu do diagramu se zúčastní backend a frontend aplikace. Přidání elementu proběhne, když uživatel stiskne tlačítko pro přidání na ovládacím panelu, viz návrh 3.3.

V rámci této kapitoly podrobně popíši přidávání nové třídy do diagramu. Pro přidání třídy uživatel stiskne tlačítko „Add Class“ v levé části webu na ovládacím panelu (v kódu třída `Panel`).

HTML kód pro tlačítko v `Panelu` vypádá následujícím způsobem:

```
<button onClick={this.props.addClassBtnClick}>Add Class</button>
```

Třída `App` odpovída za interakce `Panelu` a `Diagramu`. Pokud se něco změní v `Panelu`, např. uživatel stiskne tlačítko pro přidání nové třídy a vyplní jméno nové třídy, tak požadavek uživatele bude zpracován právě v třídě `App`. Třída `App` obsahuje model, který se má vykreslit na diagramu a předává ten model jako props do React komponenty `Diagram`. Z čehož plyne, že pokud se přídá nový element do diagramu, nejdříve se musí objevit v modelu třídy `App`, poté `Diagram` zjistí, že model se změnil a překreslí se.

Jakmile se stiskne tlačítko „Add Class“, zavolá se metoda `addClassBtnClick()`, která se nachází v třídě `App`:

```
addClass(btnEvent, state){
        const self = this;

        this.backend.addClass(state.className, self);
}
```

`this.backend` je instancí třídy `BackendFacade` (viz 4.9), která odpovídá za komunikaci s backendem. Posíla na URL adresu backendu požadavky, přijímá a zpracovává odpovědi (více o komunikace frontendu a backendu v 3.6). Metoda `addClass()` posílá HTTP POST požadavek pro přidání nové třídy do databáze.

Jako parametry má `state.className` a odkaz na třídu `App`. Parametr `state` obsahuje současný stav (state) React komponenty `Panel`. Jakékoliv změny v formulářích při přidávání nových elementů se zaznamenají v objektu `state`. Tím pádem, parametr `state.className` obsahuje název třídy, kterou uživatel chce pridat.

Parametr `self` přidáváme pro možnost přidání nového elementu diagramu do modelu bez nutnosti načítání všech elementů z databáze. Ušetří to čas na překreslení diagramu, protože na backend nebudou se posílat požadavky na načtení celého modelu, ale jenom na nově vytvořenou třídu, která se následně přidá do modelu se nacházející v třídě `App`.

Po volání této metody se v databázi objeví nová třída.

Metoda `addClass()` v třídě `BackendFacade`:

```
addClass(nameOfClass, app){
        const self = this;

        let parameters = { method: 'POST'};

        fetch(this.url + '/api/classes', parameters)
                .then(res =>res.json())
                .then(res => {
                        if (nameOfClass){
                                self.updateClass(res.idOfNewClass,
                                        nameOfClass, app);
                        }
                })
                .catch(error => console.log('There has been a problem
                         with your fetch operation: ' + error.message));
}
```

Metoda `fetch` umožňuje jednoduché dotazování restových end-pointů backendu. Pro dotazování používám end-point `/api/classes`, více o end-pointy v 3.6. Jako další parametr metody `fetch` zadám HTTP metodu, kterou chci použít. Metoda `fetch` umožňuje posílání požadavků na pozadí.

Mám možnost vkládání nové třídy nemající jméno, tím pádem vytvoření nové třídy na backendu nevyžaduje ten parametr. Požadavek na přidání nové třídy se uspěšně zpracuje backendem, a jakmile se nová třída objeví v databázi, backend pošle odpověď, obsahující *id* nově přídané třídy. A pokud mám zadané jméno, musím nově zadanou třídu změnit.

Metoda `updateClass()` v třídě `BackendFacade`:

```
updateClass(idOfClass, name, app){
        let body = {'name': name };

        let parameters = { method: 'PUT',
                body: JSON.stringify(body),
                headers: {
                        'Content-Type': 'application/json'
                }};

        fetch(this.url + '/api/classes/' + idOfClass, parameters)
                .then(res =>res.json())
                .then(res => {

                        app.updateModel(app.state.model
                                .concat(Class.fromJson(res)));

                })
                .catch(error => console.log('There has been a problem
                        with your fetch operation: ' + error));
}
```

Metoda `updateClass()` se posíla HTTP PUT požadavek na end-point backendu s zadaným *id*. Jako `options` se do metody `fetch()` posílám objekt obsahující parametry pro HTTP požadavek a tělo požadavku. Tělo požadavku obsahuje název třídy, který chceme nastavit.

Pro překreslení diagramu a přidání nového elementu, musíme změnit stav třídy `App`. Zavoláme metodu `app.updateModel()`, kam jako parametr pošleme nový stav.

Po zaslání požadavků na jeden z end-pointů, jej backend musí zpracovat. Požadavek přijímá konkretní metoda z třídy, odpovědné za požadovaný end-point. Tím pádem, když chceme přidat novou třídu do diagramu, třída, která zpracovává požadavky na end-pointu `/api/classes` se nazývá `classes.js` a nachází se ve složce api. Metoda zodpovědná za přidání nové třídy:

```
create({ body }, res) {
        SeqDef.visitClass().create()
                .then((aClass) => {
```

```
                res.send( { idOfNewClass : aClass.get().id });
        });
}
```

`SeqDef` je instancí třídy `GetSequelizeDef` (více 4.8.3) a metoda `visitClass()` vrací Sequelize definice tabulky pro třídu. Metoda `create()` přidá novou třídu do databáze. Po volání této metody se v databázi objeví nově přidaná třída a jak jsem zmiňovala výše, jako odpověď se odešle *id* nové třídy.

Metoda zodpovědná za změnu již existující třídy:

```
update({ aRecord, body }, res) {
        SeqDef.visitClass().findById(aRecord.id)
                .then( aClass => {
                        aClass.update({name: body.name});
                        res.send(aClass);
                });
}
```

Po volání této metody se změní jméno třídy s zadaným *id* a jako odpověď se odešle změněná třída. V metodě `app.updateModel()` přidá se nová třída do stávajícího modelu a na diagramu se objeví nová třída.

Kapitola 5

Testování

Testování je velmi důležitou částí vývoje jakéhokoliv informačního systému. Pomáhá ověřit korektní chování systému a odhalit chyby, aniž by se objevily v produkci. Cílem je ukázat funkčnost systému, splnění uživatelských požadavků či případné nalezení chyb a jejich oprava.

Pro účely mé bakalářské práce bylo provedeno jenom uživatelské testování. Výhodou uživatelského testování je, že pozoruje chování samotných uživatelů, čímž může odhalit chyby, které zůstaly vývojářům skryté. Testování probíhá formou manuálního testování.

Tohoto testování se zúčastnili tři uživatelé mé webové aplikace. Na základě scénářů, popsaných v kapitole 2.2.2, bylo provedeno testování. Testování se týkalo scénářů, které mají vysokou prioritu.

Každý uživatel musel ověřit chování systému v souladu se scénářem. Pokud popsané chování systemu v scénáři je v souladu se skutečním chováním systému, tak můžeme říct, že systém se chová korektně a že jsou splněné všechny požadavky na systém.

5.1 Uživatele webové aplikace

- žena, 21 roků. vývojář prototypu.
- muž, 22 roků. nemá zkušenosti s kreslením UML diagramů tříd.
- muž, 35 roků. má velké zkušenosti s kreslením UML diagramů tříd.

5.2 Testovácí scénáře

V této sekci se věnuji popisu testovacích scénářů, hodnocení vysledků testů a případně, popisu chyb, které byly během testování zjištěny.

TC1 - Založení nového diagramu

Hlavní scénář:

1. Při otevření vstupní obrazovky systému je již otevřen nový diagram a uživatel může pokračovat v modelování.
2. Přidat do diagramu nějaký element.
3. Zkusit založit nový diagram.
4. Kliknout na tlačítko „New Diagram“.
5. Systém se zeptá, jestli si opravdu chcete založit nový diagram.
6. Kliknout „Yes“.
7. Systém zobrazí nový diagram.

Hodnocení testu:

`Passed`. Všechny kroky scénáře jsou v souladu se skutečným chovaním systému.

TC2 - Úprava diagramu

Hlavní scénář:

1. Zkusit posunout nějaký element na diagramu.
2. Element se změní svou polohu.
3. Zkusit zmenit velikost elementu.
4. Element se změní svou velikost.
5. Systém uloží změny.

Hodnocení testu:

`Passed`. Všechny kroky scénáře jsou v souladu se skutečným chovaním systému.

TC3 - Smazání diagramu

Hlavní scénář:

1. Kliknout na tlačítko „Delete Diagram“.
3. Systém se zeptá jestli uživatel opravdu chce smazat diagram.
4. Kliknout „Yes“.
5. Systém smaže diagram.

Hodnocení testu:

`Passed`. Všechny kroky scénáře jsou v souladu se skutečným chovaním systému.

TC4 - Přidání do diagramu UML třídy

Hlavní scénář:

1. Zkusit přidat novou třídu kliknutím tlačítka „Add Class“.
2. Nová třída se objeví na diagramu.

Hodnocení testu:

`Passed`. Všechny kroky scénáře jsou v souladu se skutečným chovaním systému.

TC5 - Přidání do diagramu UML vztahu

Hlavní scénář:

1. Kliknout na tlačítko „New Relation“.
2. Systém zobrazí formulář do kterého uživatel musí napsat id tříd pro vztah.
3. Vyplnit id tříd (Pro testovácí účely je nutné zadat id v rozmezí od 1 do 5).
4. Systém zobrazí nový vztah na diagramu tříd.

Hodnocení testu:

`Passed`. Všechny kroky scénáře jsou v souladu se skutečným chovaním systému.

5.3 Shrnutí testování

Uživatelské testování ukázalo, že nejdůležitější funkcionality nástroje pro vykreslovaní UML diagramů tříd, se chovají korektně. Uživatel, který neměl žádné zkušenosti s modelováním UML diagramů tříd, při práci se systémem neměl žádné problémy s orientací v systému, což ukazuje, že návrh uživatelského rozhraní byl úspěšný.

Závěr

V práci jsem se zabývala analýzou, návrhem, implementací a testováním aplikace. Prozkoumala jsem již existující řešení dané problematiky a zvážila jsem výhody a nevýhody stávajících řešení. Dále jsem detekovala případy užití systému, na jejíchž základě jsem odvodila funkční a nefunkční požadavky na systém. Na základě analýzy jsem navrhnula model uložení UML diagramů tříd v prostředí JavaScript. Navrhnula jsem architekturu webové aplikace, která má být vyvinutá v rámci této bakalářské práce. Popsala jsem návrhový vzor, který mi umožnil dosáhnout cíle mé práce. Na základě návrhu jsem naimplemenovala prototyp webového nástroje pro kreslení UML diagramů tříd a provedla jsem testování této aplikace.

Vytvořená aplikace umožňuje vytvoření, editování a smazání UML diagramů tříd. V diagramu lze přidávat prvky UML class diagramu, zadávat mezi nimi vazby, měnit polohu a velikost jednotlivých prvků.

V budoucnu by bylo možné aplikaci rozšířit o další diagramy, např. Network diagramy anebo Business diagramy. Dobrým nápadem je přidání importu a exportu diagramů. Dále bylo by možné umožnit uživatelům sdělování svých diagramů a editování diagramu v režimu „Real Time“. Tím že backendová část poskytuje REST API je možné rozšíření anebo případná výměna klienta.

Hlavní přínos mé práce je v tom, že prototyp byl na-implementován pomocí knihovny ReactJS s využitím vlastností SVG, což je zcela nový přístup k vytváření podobných aplikací, jak je to ukázala analýza stávajících řešení. Naimplementovaný webový nástroj se stane jednou ze stěžejních komponent projektu webového IDE s pracovním názvem Laplace-IDE.

Literatura

[1] Sharing a Diagram. [online], 5 dubna 2017, ©2017 Gliffy Inc., [cit. 2018-04-15]. Dostupné z: https://guide.gliffy.com/user-manual/gliffy-online-user-manual/1/en/topic/sharing-a-diagram

[2] Exporting Diagrams as an Image. [online], 5 dubna 2017, ©2017 Gliffy Inc., [cit. 2018-04-15]. Dostupné z: https://guide.gliffy.com/user-manual/gliffy-online-user-manual/1/en/topic/exporting-diagrams-as-an-image

[3] Gliffy Inc. [online], n.d., © 2018 GitHub, Inc., [cit. 2018-04-16]. Dostupné z: https://github.com/gliffy

[4] GAUDENZ, A.: Exporting Files. [online], 17 června 2017, [cit. 2018-04-16]. Dostupné z: https://support.draw.io/display/DO/Exporting+Files

[5] BENSON, J.: Importing Files. [online], 17 července 2015, [cit. 2018-04-16]. Dostupné z: https://support.draw.io/display/DO/Importing+Files

[6] GitHub - jgraph/drawio. [online], n.d., © 2018 GitHub, Inc., [cit. 2018-04-16]. Dostupné z: https://github.com/jgraph/drawio

[7] Import and Export Your Diagrams. [online], n.d., © 2017 Lucid Software Inc., [cit. 2018-04-16]. Dostupné z: https://lucidchart.zendesk.com/hc/en-us/articles/207299716-Import-and-Export-Your-Diagrams

[8] Import and Export Your Diagrams. [online], n.d., © 2017 Lucid Software Inc., [cit. 2018-04-16]. Dostupné z: https://lucidchart.zendesk.com/hc/en-us/articles/207299716-Import-and-Export-Your-Diagrams

[9] Share your document. [online], n.d., © 2017 Lucid Software Inc., [cit. 2018-04-16]. Dostupné z: https://lucidchart.zendesk.com/hc/en-us/articles/207300466-Share-files-and-folders

[10] SAMSOODEEN, S.: Share Your Projects Real Time. [online], červen 2017, [cit. 2018-04-16]. Dostupné z: https://support.creately.com/hc/en-us/articles/227794367-Share-Your-Projects-Real-Time

[11] Creately. [online], n.d., © 2018 GitHub, Inc., [cit. 2018-04-16]. Dostupné z: https://github.com/creately

[12] W3Schools: *HTML5 SVG*. n.d., Copyright 1999-2018 by Refsnes Data, [cit. 2018-05-05]. Dostupné z: https://www.w3schools.com/html/html5_svg.asp

[13] LILLEY, C.; JACKSON, D.: Scalable Vector Graphics (SVG). [online], říjen 2004, [cit. 2018-05-05]. Dostupné z: https://www.w3.org/Graphics/SVG/About.html

[14] ECMAScript: *ECMAScript 6: New Features: Overview and Comparison*. n.d., Copyright © 2015-2017 Ralf S. Engelschall, [cit. 2018-05-05]. Dostupné z: http://es6-features.org/

[15] React: *React – A JavaScript library for building user interfaces*. n.d., Copyright © 2018 Facebook Inc., [cit. 2018-05-05]. Dostupné z: https://reactjs.org/

[16] Node.js: *Node.js*. n.d., © Node.js Foundation, [cit. 2018-05-05]. Dostupné z: https://nodejs.org/en/

[17] Express: *Express – Node.js web application framework*. n.d., Copyright © 2017 StrongLoop, IBM, and other expressjs.com contributors, [cit. 2018-05-05]. Dostupné z: http://expressjs.com/

[18] Sequelize: *Manual | Sequelize | The node.js ORM for PostgreSQL, MySQL, SQLite and MSSQL*. n.d., [cit. 2018-05-05]. Dostupné z: http://docs.sequelizejs.com/

[19] PECINOVSKÝ, R.: *Návrhové vzory*. Brno : Computer Press, 2007, první vydání, ISBN 978-80-251-1582-4, 453-454 s.

[20] The Object Management Group®: *Unified Modeling Language, 2.5.1*. prosinec 2017, [cit. 2018-04-21]. Dostupné z: https://www.omg.org/spec/UML/2.5.1/PDF/changebar

[21] PICHLIK, R.: A REST. *dagblog [online]*, říjen 2007, [cit. 2018-04-19]. Dostupné z: https://dagblog.cz/a-rest-c5156313d79e

[22] JetBrains s. r. o.: *WebStorm: The smartest JavaScript IDE by JetBrains*. n.d., Copyright © 2000–2018 JetBrains s. r. o., [cit. 2018-04-28]. Dostupné z: https://www.jetbrains.com/webstorm/

[23] MROZEK, J.: JavaScript na serveru: moduly a npm. *Zdroják [online]*, říjen 2012, ISSN 1803-5620, [cit. 2018-04-29]. Dostupné z: https://www.zdrojak.cz/clanky/javascript-na-serveru-moduly-a-node-package-manager/

[24] MENGER, D.: ESLint: Cesta ke kvalitnímu JavaScriptu. *NodeJsFan.com [online]*, květen 2016, [cit. 2018-04-29]. Dostupné z: http://nodejsfan.com/eslint-pro-nodejs-a-react/

[25] Mozilla Foundation: *Getting Started*. duben 2018, © 2005-2018 Mozilla and individual contributors, [cit. 2018-05-03]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX/Getting_Started

Příloha **A**

Seznam použitých zkratek

AJAX Asynchronous Javascript and XML

API Application Programming Interface

DOM Document Object Model

ER Entity Relationship

ES6 ECMAScript6

GUI Graphical User Interface

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

JS JavaScript

JSON JavaScript Object Notation

JSX JavaScript Syntax eXtension

MIT MIT License

ORM Object-Relational Mapping

REST Representational State Transfer

SVG Scalable Vector Graphics

UML Unified Modeling Language

URI Uniform Resource Identifier

XML eXtensible Markup Language

Příloha B

Návod na spuštění

Předpoklady pro spuštění:

- Nainstalovaný node.js. Pokud nemáte, stáhnete z `https://nodejs.org/en/download/` a nainstalujte na svůj počítač.
- Nainstalovaný npm. Pokud máte nainstalovaný node.js, npm se nainstaluje automaticky.
- Nainstalovaný databázový systém PostgreSQL. Pokud používáte jiný systém, musíte předělat create a insert script pro Váš systém. Stránka pro stážení PostgreSQL `https://www.postgresql.org/download/`, je tam možnost stáhnutí grafického rozhraní.

Návod na spuštění:

1. Nainstalovat si závislosti pro frontendovou a backendovou část aplikace zvlášť. Dělá se to příkazem `npm install`. Ten příkaz musí být spuštěn z příkazové řádky v příslušné složce. Tím se vygeneruje složka node_modules. Pro správný build frontendu musíte si modul „browserify“ nainstalovat globálně příkazem `npm install browserify -g`.
2. Nainstalovat si http-server příkazem `npm install -g http-server`.
3. Přidat do konfiguračního souboru údaje Vaši databázi. Konfigurační soubor se nachází v src/impl/backend/src/config.js.
4. Vytvořit si tabulky v databázi použitím create scriptu (pro PostgreSQL). Script se nachází src/impl/backend/db-scripts/create.sql
5. Naplňit tabulky v databázi daty použitím insert scriptu (pro PostgreSQL). Script se nachází src/impl/backend/db-scripts/insert.sql
6. Udělat symbolické linky v backendu na složku Model a třídu AbstractVisitor z frontendu. Příkazy jsou v souboru src/impl/backend/makeLink.

7. V složce backend spustit příkaz `npm run-script build`. Tím se backendová část zkompiluje. Následně spustít příkaz `npm run-script start`. Po spuštění tohoto příkazu začne backend naslouchat na portu 3000 na lokalním serveru. Na tuto adresu `http://localhost:3000` posílá požadavky frontend.

8. V složce frontend spustit příkaz `npm run-script build`. Tím se frontendová část zkompiluje. Následně spustít příkaz `npm run-script start`. Po spuštění tohoto příkazu frontendová část aplikace začne běžet na `http://localhost:8080`. V webovém prohlížeči si otevřit `http://localhost:8080` a můžete začít modelovat.

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `BP_Evseenko_Iuliia_2018.pdf` ..... text práce ve formátu PDF