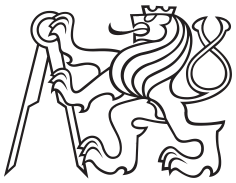
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Modul pro UML Aktivity Diagram do platformy Rhino

Václav Smítka

Školitel: Ing. Matěj Klíma
Květen 2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Smítka** Jméno: **Václav** Osobní číslo: **478205**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Modul pro UML Activity Diagram do platformy Rhino

Název bakalářské práce anglicky:

Module for the UML Activity Diagram for the Rhino Platform

Pokyny pro vypracování:

Navrhněte a implementujte modul do platformy Rhino, vyvíjené výzkumnou skupinou STILL na Katedře počítačů ČVUT FEL.
Modul bude umožňovat vytvoření a editaci zjednodušeného UML diagramu modelujícího proces nebo sled funkcí v testovaném systému, perzistentní uložení diagramu do datového modelu platformy Rhino, přidání konfigurovatelných metadat k uzlům a hranám modelu, generování testovacích scénářů z modelu za použití již existujících algoritmů v platformě, vizualizaci vygenerovaných testovacích scénářů v modelu, import a export modelu ve formátech podporovaných platformou Rhino a export vygenerovaných testovacích scénářů ve formátech CSV, XML a JSON.
Vytvořený modul otestujte vhodnou sadou uživatelských testů.

Seznam doporučené literatury:

Ammann, P., & Offutt, J. (2016). Introduction to software testing. Cambridge University Press.
Miroslav Bures, Bestoun S. Ahmed, Employment of multiple algorithms for optimal path-based test selection strategy, Information and Software Technology, Volume 114, 2019, pp. 21-36.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Matěj Klíma, katedra počítačů FEL

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

Datum zadání bakalářské práce: **13.02.2020** Termín odevzdání bakalářské práce: **22.05.2020**

Platnost zadání bakalářské práce: **30.09.2021**

Ing. Matěj Klíma 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

Děkuji vedoucímu mé práce panu Ing. Matěji Klímovi za odbornou pomoc s vypracováním této bakalářské práce. Jeho rady byly zejména při zhotovování wireframů, diagramů, testovacích scénářů a struktury této práce velmi přínosné. Rovněž děkuji panu doc. Ing. Miroslavu Burešovi, Ph.D za pomoc s volbou literatury pro teoretickou část této práce. Na závěr chci poděkovat všem mým blízkým, kteří mi pomohli s korekturou práce - především Kristýně Tabačíkové, která mi pomohla s pravopisnou částí práce.

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.

V Praze, 21. 5. 2020

Abstrakt

Tato bakalářská práce se věnuje návrhu a vývoji modulu pro UML Aktivity diagram do webové platformy Rhino. Dále se také věnuje návrhu a vývoji rozšíření platformy o možnost importu diagramů z desktopové aplikace Oxygen ve formátech json, xml, celých projektů ve formátech prj, a rovněž webové platformy Draw.io ve formátu xml. Dalším předmětem práce je návrh a vývoj exportu diagramů a vygenerovaných testovacích scénářů ve formátech xml, json a csv. Posledním z cílů práce je navržení a implementace práce s metadaty jednotlivých elementů diagramů.

Text práce obsahuje seznámení se s teorií testování. Poté je blíže specifikováno testování procesů, pro který je platforma Rhino vytvořena.

Implementace využívá již zavedených knihoven a jazyků platformy Rhino; tedy jazyku JavaSript a knihoven React, Redux a Cytoscape.js.

Klíčová slova: UML aktivity diagram, testování procesů, platforma Rhino, uživatelské akceptační testy, automatizované generování testovacích scénářů, webová aplikace

Školitel: Ing. Matěj Klíma
Laboratoř inteligentního testování softwaru,
Katedra počítačů,
Karlovo náměstí 13,
121 35 Praha 2

Abstract

The aim of this bachelor thesis is design and module development for the UML Activity diagram into the web platfrom Rhino. It is also dedicated to design and development of the platform extension to get the ability to import diagrams from the desktop application Oxygen in json and xml formats, complete projects in prj formats and also the web platform Draw.io in xml format. The next aim of the thesis is design and development of exporting diagrams and generated test cases in xml, json and csv formats. The last purpose of the thesis is design and implementation of the work with metadata of individual diagram elements.

The text contains introduction to the testing theory, it focuses especially on detailed description of the process testing which is the Rhino platform created for.

Implementation uses established libraries and Rhino platform languages, thus JavaScript and React, Redux and Cytoscape.js libraries.

Keywords: UML Activity diagram, Model-based Testing, Platfrom Rhino, User Acceptance Test, Automated Test Case Generation, Web Application

Title translation: Module for the UML Activity Diagram for the Rhino Platform

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Přestože je testování velmi důležité, mnohdy se z nejrůznějších důvodů jedná o upozaďovanou část vývoje softwaru. V dnešní době, kdy je software ve většině společností využívaný jako klíčový nástroj pro řízení chodu nebo zdroj zisku, je o to důležitější zaměřit se na vývoj podpůrných nástrojů pro jeho testování. Metod testování softwaru je celá řada a vznikají různé nástroje pro jejich zjednodušení či automatizaci. Jedním z nich je webová platforma Rhino[4], jež se zaměřuje na simplifikaci procesního testování. V rámci platformy je možné nakreslit proces v orientovaného grafu[1] a na jeho základě vygenerovat testovací scénáře.

V rámci této bakalářské práce jsem do této platformy vytvořil další modul, který umožní vytvářet procesy v základní UML aktivity notaci[2]. Dále jsem obohatil funkcionalitu platformy o rozšířenou práci s metadaty jednotlivých uzlů. Bude tak možné vybírat další, předem zvolené atributy, jež bude moci uživatel dle vlastní potřeby vkládat či odebírat. Na závěr jsem přidal modul pro import a export diagramů a testovacích scénářů. Veškeré zásahy musely být v souladu s momentální architekturou z důvodu přímé návaznosti nových komponent na již fungující systém.

Text bakalářské práce je rozdělen do čtyř základních celků, v nichž popisuji celý proces vývoje nového modulu a zbylých funkcionalit. V první části se zabývám procesním testování a teorií spojenou s testováním obecně. Kladu důraz především na vysvětlení základních pojmů a propojení s platformou Rhino. V druhé části se věnuji popisu stávajícího stavu platformy Rhino, návrhu stávajícího modulu a doplňkových funkcionalit. Ve třetí část se zaměřuji na samotnou implementaci a dokumentaci způsobu realizace jednotlivých funkcionalit. Zvýšenou pozornost věnuji zejména ověření kvality implementace a kontrole splnění všech cílů práce.

Smyslem této práce je nejen demonstrace vědomostí získaných během studia, ale především vylepšení platformy za účelem optimalizace její funkčnosti pro možné využití běžnými uživateli.

[1] Tento pojem je ekvivalentním s „flow graph“ využívaný v platformě Rhino(4.2.1) a s „directed graph“ vyžívaný v nástroji Oxygen(4.2.2). Orientovaný graf je množina uzlů a množina hran, které mají definovaný směr.[6] Pro účely generování testovacích scénářů je navíc důležité mít označený počáteční uzel.

[2] Blíže popsáno v kapitole Požadavky na modul 4.1.1

Kapitola 2
Teorie testování

V dnešní době si nelze představit život bez informačních technologií – např. počítač používáme doma i v práci takřka každý den. Společnosti vydělávají peníze poskytováním služeb a produktů související s IT nebo využíváním IT systémů k podnikání.[7]

Chyba v IT systému může mít velmi vážné důsledky: společnost nemusí být schopna dodávat produkty nebo vyúčtování za provedené služby. Zákazník, který je zklamaný kvalitou služby nebo produktu, hledá jiného dodavatele. Software se tak stal subjektem, který je třeba nebrat na lehkou váhu.[7]

2.1 Základní termíny

Definice pojmů je klíčová pro testování softwaru. Nejeden IT systém zkolaboval na různých definicích základních pojmů nebo odlišném pochopení požadavků.[8] Níže jsou uvedeny základní pojmy teorie testování.

2.1.1 Chyba

Pojem chyba je velmi skloňovaný termín, vyskytuje se v mnoha podobách a je nazýván různými způsoby. Českými ekvivalenty jsou např. slova: problém, defekt, závada, nesrovnalost, nekonzistentnost apod. V angličtině jsou nejvíce rozšířené pojmy: „error“, „fault“, „failure“, „mistake“ nebo „bug“.[7]

Chybou se rozumí rozdílný stav zhotoveného produktu nebo jeho části oproti požadavkům. Riziko chyby v IT systému je hlavní motivací pro vznik teorie testování.

Chyba má rovněž vlastní hodnotu, již lze vyčíslit pomocí vynaložených financí na její opravu. Je zjištěno, že cena chyby roste s dobou její existence.[1] Vzhledem k tomu, že se v dnešní době životní cyklus IT systémů výrazně zkrátil, je o to důležitější chyby efektivně vyhledávat a opravovat.[7]

2.1.2 Testování

Jedná se o proces, který probíhá v průběhu vývoje softwaru. V nejlepším případě by měl začít hned při získávání požadavků a skončit po ukončení

[1] Boehmův první zákon[8]

podpory systému. Lze ho různorodě členit a zároveň k němu přistupovat různými způsoby.[9]

Například pro metodiku Test driven development[10] jsou testy klíčové pro vývoj softwaru. Na jejich základě teprve vzniká software, který je výsledek kladného průchodu testů.

Testování podává hlášení o tom, jak realistické je dosažení očekávaného cíle. Poukazuje na rizika, která cíl ohrožují a studuje dopad provedených změn.[7]

2.1.3 Tester

Při vývoji softwaru se na testování podílí celý tým lidí. Velikost týmu zpravidla záleží na velikosti projektu. Základní pozicí je tester, který má zodpovědnost za otestování konkrétní části softwaru.[7]

2.1.4 Test

Nizozemský slovník van Dale uvádí následující definici: *„Test je posouzení kvality schopností lidí nebo věcí.“*[7]

Definicí pojmu test však existuje celá řada. Důležité je, že se jedná o nástroj, pomocí něhož tester testuje IT systém pro odhalení chyb či ověření systémové kvality.[7]

2.2 Strategie testování

Pro úspěšný proces testování je klíčové strategii testování zadefinovat. Tato definice je obecný popis jednotlivých testovacích úrovní a testů prováděných v rámci zmíněných úrovní pro danou organizaci nebo program.[7]

2.2.1 Úrovně testování

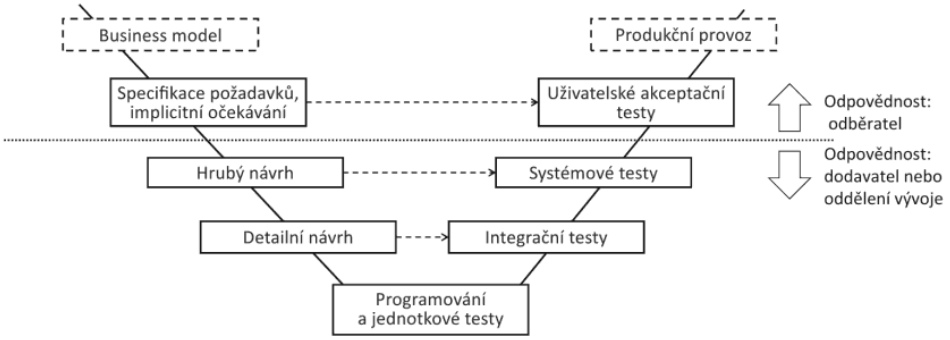
Pro určení jednotlivých úrovní testování je třeba konkrétního návodu, jenž pomůže rozčlenit celý proces do jednotlivých částí, v nichž je třeba testovat daným způsobem. K tomuto účelu jsou vytvořené metodiky.[1] Formální definice metodiky je následující:

Metodika je souhrn vodítek a principů, které mohou být přizpůsobeny a aplikovány na řešení určité situace. Může jít jak o jednoduchý soupis akcí, které mají vykonat, tak o specifický přístup, šablony, formuláře, popřípadě konkrétní seznamy.[11]

V model

Jeden z možných způsobů rozdělení definuje V-model, který byl definován již v osmdesátých letech 20. stol. Je znázorněn na obrázku 2.1.[1]

V-model tvoří několik dvojic aktivit (specifikace požadavků – uživatelské akceptační testy, hrubý návrh – systémové testy, detailní návrh – integrační testy, programování – jednotkové testy). Na levé straně se nachází aktivity



Obrázek 2.1: V-model - převzato[1]

související se specifikací a návrhem na pravé aktivity testování. Tímto je zdůrazněna úloha testování ve vazbě na odpovídající analytické aktivity a související artefakty.[1]

Tento model je variací na všeobecně známý vodopádový model pro vývoj softwaru, kde testování probíhá až v poslední fázi vývoje. Z těchto důvodů byly vytvořeny i další modely, jež se snaží začlenit testování do všech fází vývoje jako například W-model a další.[1]

2.2.2 Plán testování

Uvádí se dva typy plánů testování. Plán testů, nebo také Detailní testovací plán, je dokument popisující rozsah, přístup, zdroje a harmonogram zamýšlených testovacích aktivit. Identifikuje položky testování, vlastnosti k otestování, testovací úkoly, kdo je bude dělat, stupeň nezávislosti testerů, testovací prostředí, techniky návrhů testů, použitá vstupní a výstupní kritéria, odůvodnění pro jejich volbu a rizika, na jejichž základě je nutné definovat plány pro výjimečné situace. Testovací plán je výstupem procesu plánování testů. Plán testování je typicky vytvářen pro jednu úroveň testů.[1]

Hlavní plán testování je obvykle zaměřen na více úrovní testů. Oproti detailnímu testovacímu plánu je zde kladen důraz na projekt jako celek a s ním spojené cíle, financování, přístup k testování, definice rozsahu apod.[7]

2.2.3 Komunikace

Po IT systémech roste poptávka, nové IT technologie přibývají a pole možností vývojářů se tak značně rozšiřuje. S příchodem nových technologií přichází nutnost zavedení zcela nových pojmů. U starších technologií dochází k vývoji a jednotlivé modifikace rovněž potřebují unikátní pojmenování. Tyto aspekty vedou k definici nových pojmů a využívání pojmů starých v novém kontextu. Je tedy nezbytné ujasnit si terminologii a sjednotit slovník, který se bude v daném projektu využívat. Komunikace se musí řídit pravidly, jež jsou pevně stanovena, jinak hrozí riziko nedorozumění a nekonzistentnosti názvů.[1]

2.3 Rizika

Dennodenně se každý z nás potýká s riziky. Většinou intuitivně zvažujeme možné následky, promýšlíme, jak se jim vyhnout, a následně vymyšlenou strategii realizujeme. Například riziko ztráty klíčů minimalizujeme zvýšenou pozorností věnovanou skutečnosti, že je máme stále někde při ruce. V každém projektu tak rovněž číhají rizika, jež mohou způsobit zpoždění, navýšení nákladů nebo dodání nekvalitního či neúplného díla.[1]

2.3.1 Definice

Formálních definic je jako u většiny pojmů v testování scénářů je více. Jedna z definic je podle Glosáře ISTQB: *„Faktor, který by mohl v budoucnu vést k negativním následkům; je obvykle vyjádřen jako dopad a pravděpodobnost.“*[1]

Při řízení projektu je důležité rizika shromáždit a pracovat s nimi. Je důležité je popsat a následně rozdělit do kategorií. Pro určení vážnosti se rizika měří pomocí kombinace pravděpodobnosti uskutečnění hrozby nebo příležitosti a závažnosti jejího dopadu na cíl.[1]

Rizika dělíme na:

- **Projektová rizika** – mohou mít dopad na dodávku výstupů projektu v plánovaném čase, za plánovanou cenu či v plánovaném rozsahu. Mezi tato rizika typicky patří např. onemocnění důležitého člena týmu.[1]
- **Produktová rizika** – ovlivňují některou z požadovaných či očekávaných vlastností vytvářeného produktu. Tato rizika jsou někdy nazývána kvalitativní, neboť mohou mít dopad na očekávanou kvalitu produktu. Mezi tato rizika řadíme např. nesprávnou funkčnost systému.[1]

2.3.2 Přístup

Projektová i produktová rizika je nutné řídit a pravidelně kontrolovat. Je nezbytné, aby byl v průběhu projektu kontrolován stav všech rizik a v případě konkrétních scénářů aplikován navržený krizový plán.

Proces řízení rizik je cyklický. Začíná u identifikace, kdy se rizika sbírají. Nasbíraná rizika poté prochází analýzou. Na základě analýzy poté dochází k určení vážnosti rizik a k jejich ohodnocení. Dále se plánují opatření, jejichž účelem je riziko minimalizovat nebo odstranit. Následně se dané opatření realizuje a poté je nezbytné rizika monitorovat. Po určité době se tento cyklus opakuje.[1]

Kapitola 3

Testování procesů

Testování procesů aplikací je jedním z nejčastěji používaných konceptů testování. Většina testovacích návrhů se provádí ručně a automatizovaná podpora se neprovádí v míře, jež by byla efektivní. Pro větší množství procesů a ruční generování testovacích scénářů může být testování časově náročné a rovněž může vést k chybám.[12]

Platforma Rhino je nástrojem pro automatizaci generování testovacích scénářů pro orientované grafy pomocí algoritmu PCT, jemuž se budu věnovat v samostatné kapitole. Dále se v této práci zaměřím na popis způsobu rozšíření platformy o možnost generování testovacích scénářů na základě aktivity diagramů.

V této kapitole popisuji základ testování procesů, který je rozdělen do dvou částí. V první části se zabývám vytvářením modelů procesů. Druhá část představuje dva základní postupy pro generování testovacích scénářů.

3.1 Model procesů

Přirozený způsob, jak postavit testovací scénář, je řetězit sekvence specifických volání na různé funkce testovaného systému. Aby bylo možné vytvořit daný testovací scénář, je nutné vhodně vytvořit model testovaného systému jako podklad pro vytváření scénářů.[13]

Pro tento účel se využívá modelů, které umožňují znázornění chování systému. Zde uvádím tři druhy znázornění.

3.1.1 Orientovaný graf

Jedná se o nejjednodušší způsob znázornění procesu. Graf je uspořádaný pár G = (V, E). Množina V obsahuje vrcholy, kterým se též říká uzly a množina E obsahuje hrany. Orientovaný graf je takový graf, který má směrované hrany e = (x, y), kde hrana vede z x do y.[13]

Testovací scénář je následně definován jako cesta v grafu od zadefinovaného počátečního uzlu. Cestou v grafu je myšlena taková posloupnost uzlů, která mezi každými dvěma sousedními uzly umožňuje existenci právě jedné hrany. Takto lze definovat i cestu grafem pomocí hran.[13]

3.1.2 Aktivity diagram

UML aktivity diagramy se používají k modelování dynamických aspektů skupiny objektů a řídí tok operací. Základní myšlenkou aktivity diagramů je modelování aktivit a jejich možné posloupnosti vykonání. Aktivity diagram zároveň vyjadřuje klíčové chování systému a hojně se využívá pro modelování celých systémů a byznys procesů – proto je vhodný pro návrh testovacích scénářů. Na základě jednoho aktivity diagramu vzniká celá sada testovacích scénářů.[14]

3.1.3 IFML

Existují i další možnosti modelování softwarových aplikací. Jedním z nich je standardizovaný Web Modeling Language (WebML), který oproti UML aktivity diagramům taktéž podporuje modelování uživatelských rozhraní (UI) a uživatelské interakce. WebML se vyvinul v Interaction Flow Modeling Language (IFML), jenž pokrývá širší spektrum uživatelských rozhraní a tok dat mezi jednotlivými komponentami uživatelského rozhraní.[15]

Na základě tohoto typu modelu lze také vytvářet testovací scénáře.[16]

3.2 Generování testovacích scénářů

Automatizovaný způsob generování testovacích scénářů způsobem nalezení cesty v grafu je intenzivně studovaný problém. K tomuto účelu lze mimo jiné použít již publikované algoritmy a strategie zaměřující se na průchod grafem. Jedná se například o prohledávání do hloubky (DFS), prohledávání do šířky (BFS) nebo Ford-Fulkersonův algoritmus.[17]

Níže jsou uvedeny dvě techniky, které jsou využívány v nástroji Oxygen.

3.2.1 PCT

Process Cycle Test (PTC) je technika, která se používá zejména při testování integrace mezi administrativní organizací a automatizovaným informačním systémem. Zaměřuje se na pokrytí variací v procesu.[18]

Na základě strukturovaných informací o požadovaném chování systému ve formě cest a rozhodovacích bodů vytváří testovací scénáře. Tato technika je vytvořena metodologií TMap a blíže popsána v knize „TMap next: for result-driven testing“.[18]

3.2.2 PPT

Prioritized Process Test (PPT) je technika, která generuje testovací scénáře zaměřené na pokrytí prioritní části pracovních toků s vysokou intenzitou a pokrytí neprioritní části s nižší intenzitou. “.[19]

Na základě seznamu hran, úrovně priorit (PTL) a hloubky testování (TDL) jsou vygenerovány testovací scénáře. Konkrétní popis algoritmu i jeho ověření je v článku „Prioritized Process Test: More Efficiency in Testing of Business Processes and Workflows“.[19]

Kapitola 4

Návrh modulu do platformy Rhino

Předmět této bakalářské práce lze rozdělit na několik částí. V první části se jedná o samotný aktivity modul, jenž umožní uživateli vytvářet UML aktivity diagramy[1]. Z nich se budou následně, stejně jako u orientovaných grafů, generovat testovací scénáře. Další část se věnuje práci s doplňkovými metadaty, jež jsou předem definovaná. Poslední část se soustředí především na rozšíření funkčnosti platformy.

Filosofie platformy je postavena na svobodě uživatele. Není zde kladen důraz na omezování kreslení diagramů pravidly notace. Jsou zavedena pouze základní omezení, jež by mohla narušit chod platformy nebo generování testovacích scénářů.

4.1 Požadavky na modul

Jak bylo zmíněno v úvodu, platforma Rhino je již existující a funkční platformou, na základě čehož vzniká několik základních požadavků a omezení ohledně nových částí aplikace. Především je důležité zachovat architekturu stávajících částí a napojit na ně nové tak, aby společně komunikovaly a nedocházelo ke konfliktům. Dalším z klíčových požadavků je umožnit zpětný import všech exportovaných diagramů do platformy Rhino.

Dále je velký důraz kladen především na oddělení funkcionality od vizualizace za účelem udržení aplikace čisté. Z toho rovněž vyplývá nevyčleňování jednotlivých nových částí platformy a znovu využívání již vzniklých částí aplikace. Veškerá nová funkcionalita musí být připravena pro možné další rozšíření nebo napojení nových modulů, které budou kooperovat se zbytkem platformy.

Vzhled nově vzniklých částí musí vycházet z vzhledu stávajících. Klíčová je zejména jednoduchost a přehlednost. Nové prvky tedy budou využívat již integrované knihovny React Bootstrap[2].

[1] Je to jeden ze skupiny UML diagramů chování. Ukazuje tok řešení nebo tok objektu s důrazem na podmínky toku.[20]

[2] Nejpopulárnější framework pro klientskou část přepracován pro React[21]

4.1.1 Aktivity diagram

Hlavní částí práce je rozšíření platformy Rhino a možnost kreslení diagramů pomocí UML aktivity notace[3]. Použiji základní aktivity prvky, jmenovitě počáteční uzel, rozhodující uzel, koncový uzel a uzel aktivity. Jednotlivým uzlům se po vytvoření automaticky vygenerují názvy, které lze později změnit. Uzly se budou tvořit podle pravidel:

- Názvy malými písmeny („a,“ „b,“, ...)
- Priorita nastavena na nezadefinovanou („Undefined“)

U jednotlivých uzlů bude možné, stejně jako název měnit i prioritu. Prioritu bude rovněž možné změnit na jednu z hodnot: „Undefined,“„Low,“ „Medium,“ nebo „High“.

Jednotlivé uzly bude možné spojit jednou nebo více hranami, stejně jako u orientovaných grafů. Hrany se budou tvořit podle pravidel:

- Názvy psány formou čísel („0“, „1“, ...)
- Priorita nastavena na nízkou („Low“)

Název hrany bude možné dle uživatele libovolně měnit. Prioritu bude opět možné upravit na jednu z hodnot: „Undefined“, „Low“, „Medium“ nebo „High“.

Kreslení diagramu bude mít několik základních omezení. V prvé řadě nebude možné mít v jednom diagramu více počátečních uzlů. To je klíčová podmínka pro algoritmus generující testovací scénáře. Dále nebude možné vést hranu do počátečního uzlu a koncového uzlu. Tato dvě omezení jsou stanovena za účelem dodržení základních pravidel pro UML aktivity diagramy.

Posledním požadavkem z pohledu High level[4] požadavků je možnost vygenerování testovacích scénářů na základě nakresleného diagramu. Je tedy potřeba využít již integrovaný algoritmus Oxygen PCT[12].

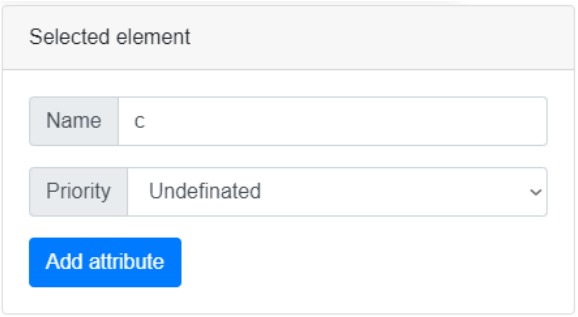
Z architektury již vzniklé platformy vyplývá, že pro kreslení diagramů je třeba využít javascriptové knihovny Cytoscape.js[24]. Zároveň pokud možno co nejméně změnit zápis diagramu do databáze tak, aby se do stejného zápisu mohly ukládat jak orientované grafy, tak aktivity diagramy.

4.1.2 Metadata

Platforma už podporuje u hran i uzlů úpravu dvou základních atributů - název a prioritu. Oba atributy jsou pro platformu klíčové. Rovněž každý diagram musí mít svůj unikátní název. Ač to není pro vnitřní chod aplikace nezbytně nutné, je to obzvláště u hran dobré pro přehlednost. Při vygenerování testovacích scénářů by se mohlo jednat o riziko duplicitních názvů v testovacím scénáři.

[3] Obecně rozvedeno v The Unified Modeling Language User Guide[22]
[4] Požadavky lze rozdělit do kategorií podle úrovně podrobnosti[23]

Je třeba navrhnout způsob zadefinování doplňkových atributů, mezi kterými si bude moci uživatel vybrat a zvolit jejich hodnotu. Veškeré nově vytvořené parametry je nutné uložit do databáze[5] pro další použití. Je nutné, aby je uživatel mohl dále měnit a případně smazat.



Obrázek 4.1: Stávající vzhled panelu atributů v Rhinu

Po kliknutí na daný element se na pravé straně pod stávajícími atributy zobrazí atributy dodatkové. Je důležité dodržet již zavedený způsob vizualizace atributů, který můžete vidět na obrázku 4.1.

Cílem této práce není vytvářet skládající mechanismus pro vkládaní konfiguračního souboru[6] s výčtem všech možných dodatkových atributů. Dodatkové atributy jsou již definované předem v rámci konstant uvnitř kódu.

4.1.3 Import a Export

Důležitou rozšiřující funkcionalitou platformy bude import a export grafů a export testovacích scénářů. Hlavním přínosem platformy nemá být možnost diagram nakreslit, ale na základě diagramu vytvořit testovací scénáře. Z tohoto důvodu je důležité mít možnost importu diagramů.

Protože platforma Rhino je úzce spjata s desktopovým softwarem Oxygen[7], který též generuje testovací scénáře na základě diagramů, je dalším úkolem navrhnout formát pro export tak, aby bylo možné přenést diagramy vytvořené v rámci Rhino do Oxygenu a obráceně.

Bude rovněž možné importovat a exportovat orientované grafy. Je důležité pracovat s možností rozšíření funkcionality i pro další typy diagramů, které v tuto chvíli ještě nejsou naimplementované. V plánu je například implementace možnosti vytvářet stavové diagramy[8].

[5] Více o databázi v kapitole MongoDB 4.2.1
[6] Soubor, který mění nastavení softwaru.
[7] Více v kapitole Oxygen 4.2.2
[8] Je to jeden ze skupiny UML diagramů chování. Ukazuje diskrétní chování části navrženého systému prostřednictvím přechodů konečných stavů.[25]

Diagramy

Diagramy[9] mají v platformě Rhino již zadefinovaný formát, jakým se ukládají do databáze a jakým způsobem se používají. Hlavním požadavkem je možnost importu diagramů z platformy Oxygen a možnost importu exportovaných diagramů do Oxygenu. Další požadavek je možnost importu diagramu z webové aplikace Draw.io[26].

Pro export diagramu definoval školitel formáty json, xml a csv[10]. Json a xml ponesou plnohodnotnou informaci o diagramu, tedy i pozici jednotlivých uzlů v souřadnicovém systému a doplňkové atributy. Formát csv bude zaměřen především na jednoduchost a budou do něj vloženy pouze nezbytné informace pro zpětnou rekonstrukci diagramu. Spojení uzlů bude zadefinováno maticí sousednosti[27].

Do platformy bude možné importovat více diagramů najednou – ať už v jednotlivých souborech nebo v projektovém souboru vyexportovaném z Oxygenu.

Testovací scénáře

Klíčový výstup celé platformy jsou testovací scénáře. Pro uživatele je důležité, aby měl možnost je z platformy exportovat za účelem dalšího použití. Pro pohodlí uživatelů je požadavkem školitele možnost exportovat testovací scénáře ve formátech json, xml a csv. Všechny formáty ponesou stejnou informaci - výčet všech testovacích scénářů. Jednotlivé testovací scénáře budou vyobrazeny jako posloupnost názvů hran oddělené pomlčkou.

4.2 Použité technologie a knihovny

Jak je zmíněno v úvodu, platforma byla již vytvořena jako praktická část diplomové práce Ing. Ruslana Bakeyeva. V této sekci popíšu použité technologie platformy a její stávající architekturu.

4.2.1 Platforma Rhino

Jedná se o webovou platformu, která je vytvořena jako klient-server aplikace[11]. Dle toho byly i zvoleny technologie – jedná se o moderní a dnes využívané způsoby[12]. Technologie lze rozdělit na dvě skupiny podle toho, zda jsou vyžívány na straně klienta nebo serveru.

[9] V této práci využívám českého slova diagram, který v platformě má anglický ekvivalent „graph“.

[10] Více o formátech v pod kapitole Implementace Export diagramů 5.3.1

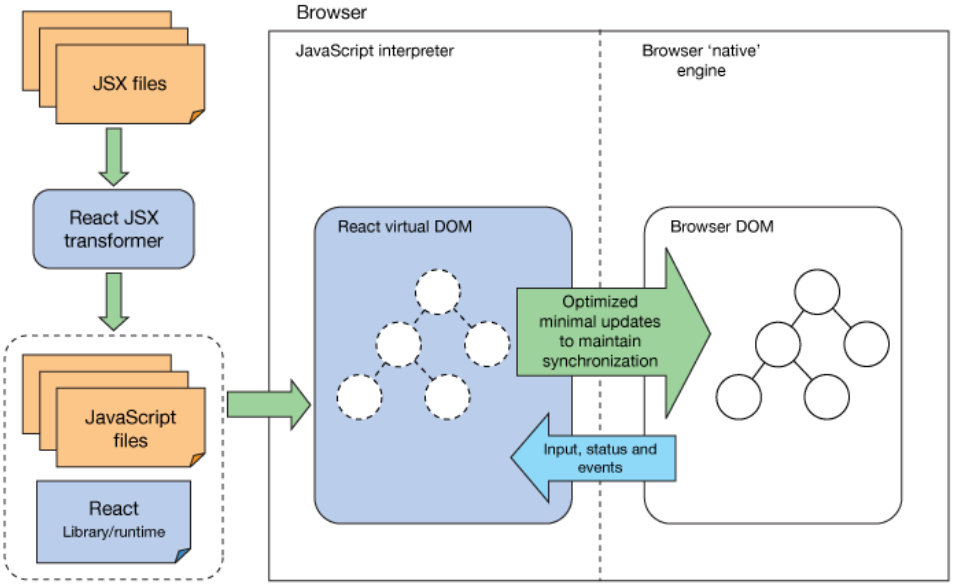
[11] Návrhový vzor pro dnešní webové aplikace[28]

[12] Autor platformy ve své práci v kapitole „Defining client-side technology stack“ zabývá výběrem technologií[4]

Uživatelské rozhraní je vytvořeno pomocí Javascriptu[13] a jeho knihoven[14]. V dnešní době je hojně využíván především díky možnosti vykonávání v prohlížeči na straně uživatele. Umožňuje dynamický obsah a větší interakci. Pro tyto účely byla rovněž vybrána knihovna React doplněna knihovnou Redux, které mají na starosti správu stavu aplikace. Tyto knihovny jsou použity nejen pro vystavění celé platformy a uživatelského rozhraní, ale i pro samotnou tvorbu diagramů. Pro kreslení diagramů je využito specializované knihovny Cytoscape.js, která se zaměřuje na kreslení a vizualizaci všech druhů diagramů.

React

Javascript je v dnešní době podporovaný ve všech moderních prohlížečích a stal se ve své podstatě hlavním programovacím jazykem pro uživatelské prostředí webových aplikací. Na základě vzrůstající komplexity javascriptových aplikací se však začalo objevovat mnoho problémů. Kód se začal stávat nepřehledným a jeho výkon se začal snižovat. V roce 2013 firma Facebook zveřejnila project React jako open source UI komponentovou knihovnu jako způsob, kterým se vyrovnávají s komplexitou javascriptových aplikací.[2]



Obrázek 4.2: Fungování Reactu - převzato[2]

Obrázek 4.2 ukazuje, jak React funguje. Projekt je rozdělen do jednotlivých komponent, jež mohou být realizované pomocí funkce nebo třídy. Tyto komponenty tvoří hierarchickou strukturu, která je postupně celá transformovaná do čistého Javascriptu a poté teprve převedena do DOM struktury, jak se

[13] Lehký, interpretovaný, objektově orientovaný jazyk s funkcemi první třídy a je nejlépe známý jako skriptovací jazyk webových stránek.[29]

[14] Soubor funkcí, které již někdo naprogramoval a zveřejnil pro veřejné využití. V platformě Rhino jsou knihovny stahovány z npm[30]

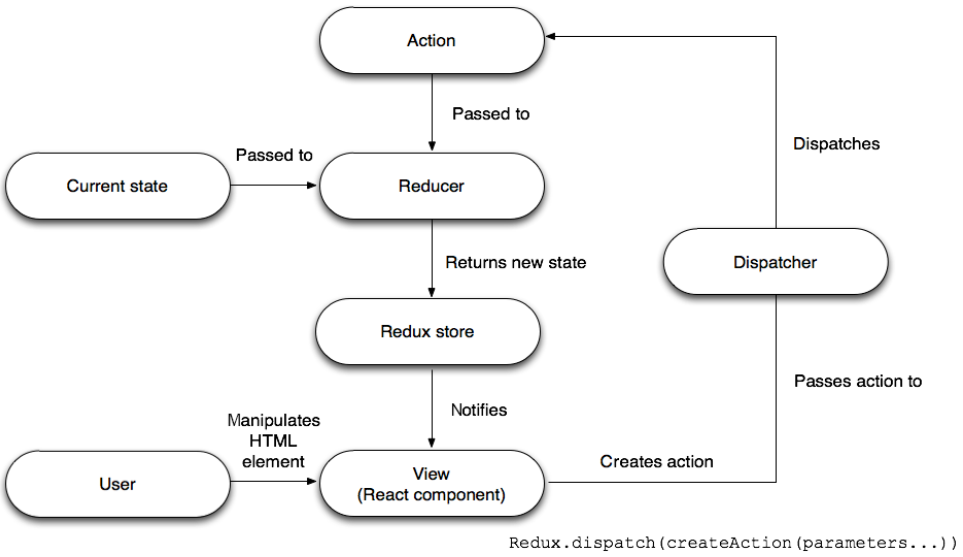
vizualizuje v prohlížeči. V rámci transformace dochází též k optimalizaci kódu, aby aplikace byla co nejvýkonnější.

Redux

Stavový management je jedním z nejtěžších aspektů softwarového vývoje. Nekonzistentní stav aplikace způsobuje velké množství bugů, které jsou často obtížně vystopovatelné. Za účelem vyřešení těchto problémů vznikla javascriptová knihovna Redux. Jedná se o předvídatelný stavový kontejner pro javascriptové aplikace. Redux byl vyvinutý React komunitou, ale zároveň na ni není vázaný. Redux je možné využít s jakoukoliv jinou javascriptovou knihovnou nebo frameworkem.[3]

Jedná se o zjednodušenou implementaci Facebookové Flux architektury. Staví na třech hlavních principech:[3]

- Stav aplikace je uložen v jediném objektu
- Stav aplikace je needitovatelný a lze změnit pouze skrz akci popisující změnu stavu
- Reducers tvoří další stav na základě momentálního stavu a akcí



Obrázek 4.3: Struktura Reduxu - převzato[3]

Na obrázku 4.3 je vidět znázorněný tok dat v rámci Redux architektury na základě uživatelské činnosti ve View komponentě, která je v našem případě komponentou Reactu. Zde se uskuteční akce, která pomocí Dispacheru spustí aktivitu. Tato aktivita po dokončení své činnosti spustí Reducer, jenž na základě výsledku a stávajícího stavu aktualizuje stav aplikace v Redux storu.

Cytoscape

Jedná se o opensource softwarovou platformu pro vizualizaci sítí, biologických drah a interakci těchto sítí s anotacemi a dalšími údaji. Přesto byl Cytoscape původně navržen pro biologický výzkum, nyní je obecnou platformou pro komplexní síťovou analýzu a vizualizaci.[31]

Na základě softwarové platformy též vznikla javascriptová knihovna, která je využita pro kreslení diagramů v Rhinu. Knihovna je rozdělena do balíčků, do nichž je rozdělena jednotlivá funkcionalita.[32]

Pro jádro editoru diagramu je využíván balíček CytoscapeCore. Balíček EdgeHandlers slouží pro veškerou práci s hranami. Pohyb po ploše editoru zajišťuje balíček Panzoom, jenž zároveň umožňuje přibližovat a oddalovat diagramy. Posledním balíčkem je GridGuide, který utváří čtverečkovanou síť na pozadí editoru.[15]

Dropzone

Lehká knihovna pro použití tzv. „přetáhni a pusť“[16]. Na poměry ostatních knihoven sloužících k témuž účelu má jednoduché rozhraní. Podporuje velké množství různých nastavení v rámci okna, kam se soubory vkládají, a rovněž úpravu designu. Také má velmi kvalitně zpracovanou dokumentaci obohacenou o velké množství ukázek jednotlivých implementací.[33]

Xml-js

Jedná se o jednoduchou knihovnu pro převod mezi javascriptovými objekty json a xml. Zejména pro výstupní formát xml knihovna umožňuje opravdu bohaté nastavení. Její další velkou výhodou je zejména kvalitní dokumentace a jednoduchost použití. Interface knihovny je velmi intuitivní a nároky na vstupní objekty pro převod do xml jsou minimální. Pro oddělení atributů od vložených elementů využívá speciálně pojmenovaných objektů „_attribute“.[34]

Spring

Rychlý a jednoduchý open source Javový framework Spring je využit pro implementaci serveru. Konkrétně je využito Spring Reactive, který podporuje vývoj neblokových a asynchronních aplikací. [35]

Na serveru se diagramy zpracovávají a ukládají do databáze. Jeho druhou důležitou funkcí je generování testovacích scénářů. Pro jejich vytvoření se využívá algoritmu z Oxygenu.

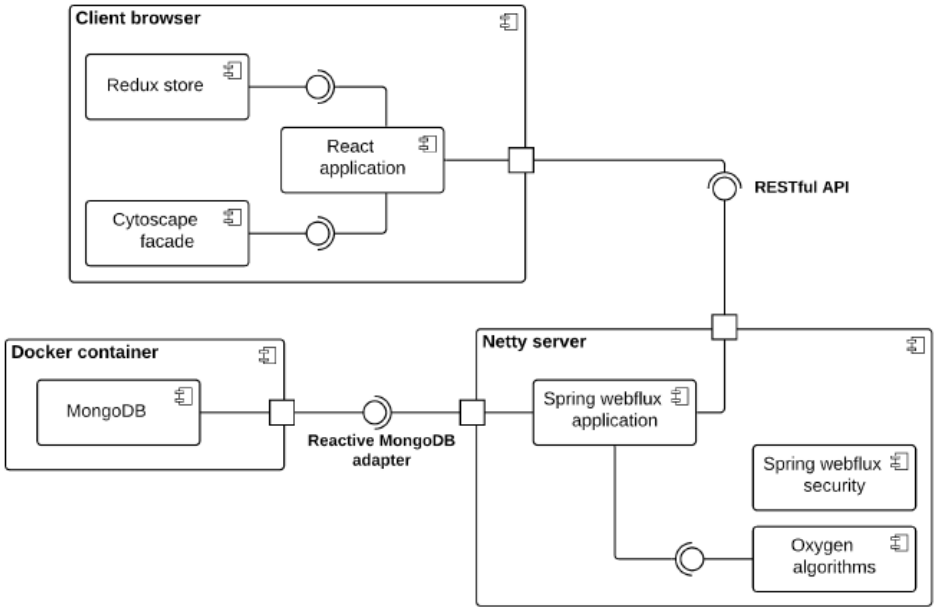
[15] Více v kalitole Implementace aktivity diagramu 5.1
[16] V originále: „Drag and Drop“

MongoDB

Vzhledem k charakteru ukládaných dat se používá grafová databáze MongoDB. Jedná se o jednu z nejrozšířenějších dokumentově orientovaných databází. Cílem je vytvořit přirozenější způsob práce s daty, který bude zároveň výkonnější než klasické SQL databáze.[36] Tento způsob je zde využit, protože se diagramy lépe ukládají do json objektů než do tabulek.

Architektura platformy Rhino

Architektura platformy staví na výše zmíněných technologiích, jak je vidět na obrázku 4.4 Klientská část je se serverem propojená pomocí RESTful API[17]. MongoDB databáze je pak spojena se serverem přes reactivní adapter frameworku Spring.

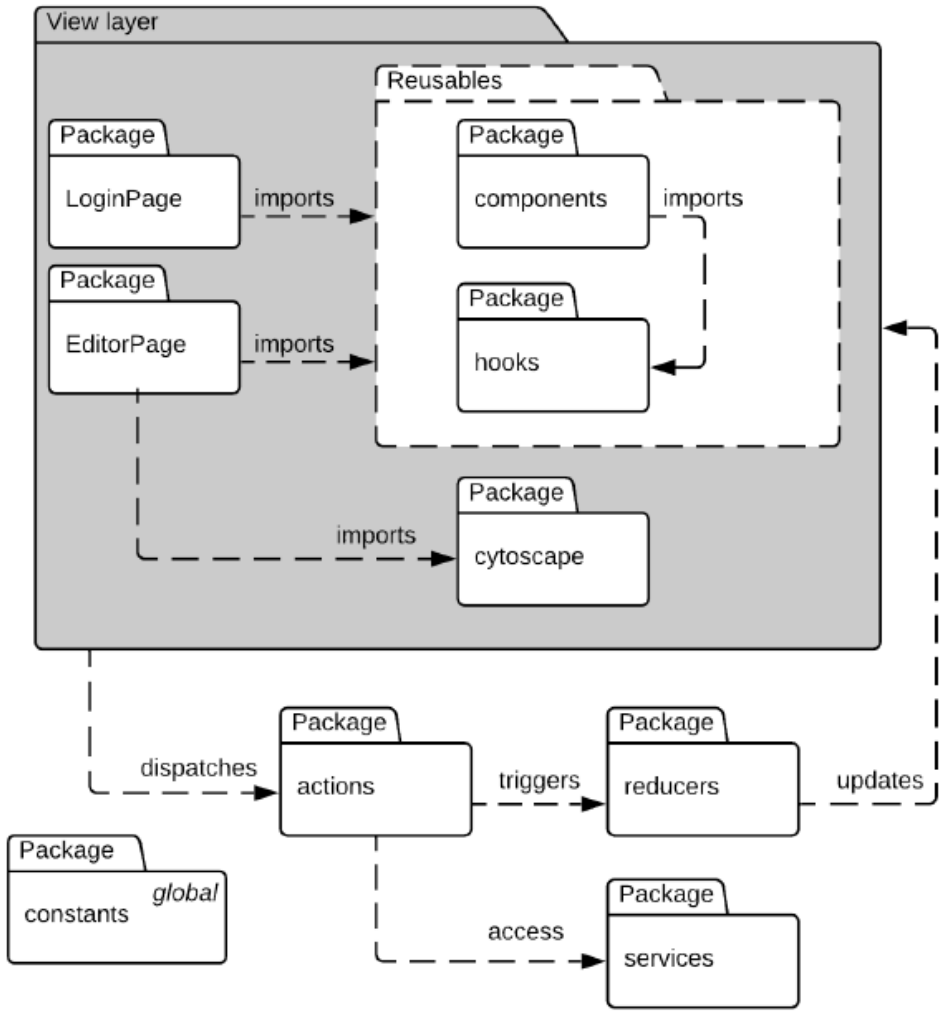


Obrázek 4.4: Diagram architektury platformy Rhino - převzato[4]

Nyní se blíže zaměřím na popis klientské části platformy Rhino, neboť většina nové funkcionality a nového modulu ovlivní především tuto část. Celá klientská část je postavena jako one page aplikace a je spravována předvídatelným stavovým kontejnerem Redux. Na základě použití těchto technologií je strukturovaný i projekt.

Z obrázku 4.5 je vidět použití knihovny Redux podle rozdělení balíčků na view, actions a reduces. Dále jsou zobrazeny dvě základní stránky platformy, a to Přihlašovací stránka (LoginPage) a stránka s diagramy (EditorPage). Obě stránky jsou hierarchicky rozděleny na komponenty, jež jsou uloženy

[17] Celým názvem Representational State Transfer[37]. Užívaný pro architektonický styl nabržený pro komunikaci aplikací pomocí HTTP. Aby API mohlo být RESTful, musí splňovat daná pravidla.[38]



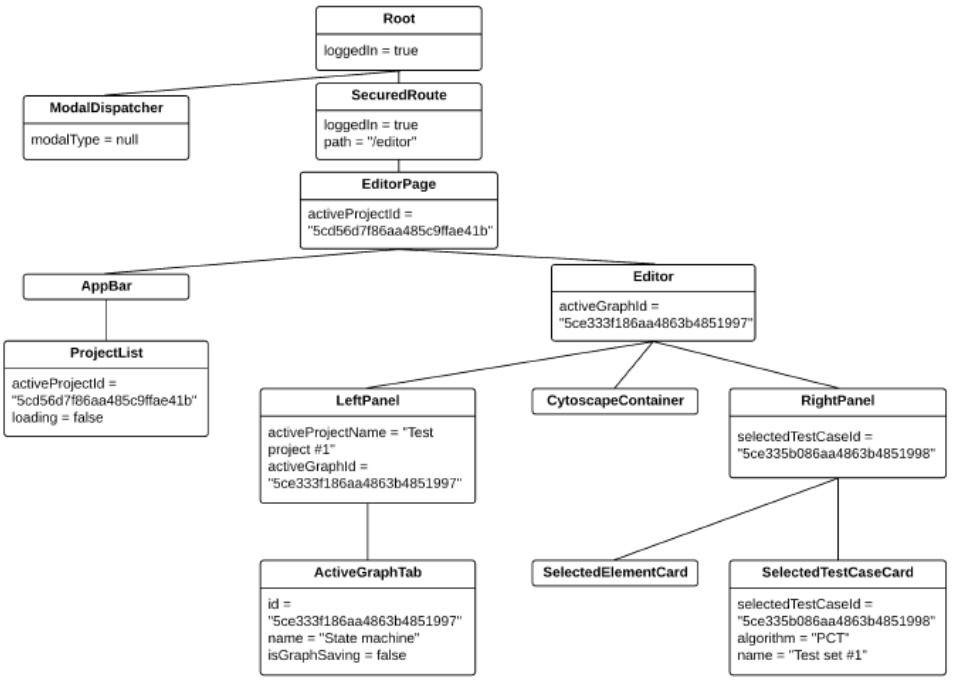
Obrázek 4.5: Diagram hierarchie modulů klientské části - převzato[4]

zvlášť. Je rovněž vidět, že je Cytoscape knihovna vyčleněna mimo zbytek aplikace pro přehlednost.

Na obrázku 4.6 je znázorněna hierarchická struktura aplikace ve chvíli, kdy se uživatel přihlásí a má již vytvořený projekt, v němž se nachází diagram s vygenerovanými testovacími scénáři. Je zde vidět souvislost komponent editoru a napojení knihovny Cytoscape.js na zbytek aplikace.

4.2.2 Oxygen

Celá platforma vznikla jako webová verze desktopového nástroje Oxygen. Tento nástroj je naprogramovaný v Javě a vznikl na Katedře počítačů pod vedením skupiny STILL v rámci mnoha závěrečných prací studentů. Je to nástroj pro generování testovacích scénářů na základě vytvořených diagramů. Vhledem k tomu, že se nástroj vyvíjí již od roku 2015, je podstatně rozšířenější než platforma Rhino, která se z tohoto důvodu nedá považovat za jeho



Obrázek 4.6: Zjednodušený příklad stromu React komponent - převzato[4]

plnohodnotnou náhradu.[12]

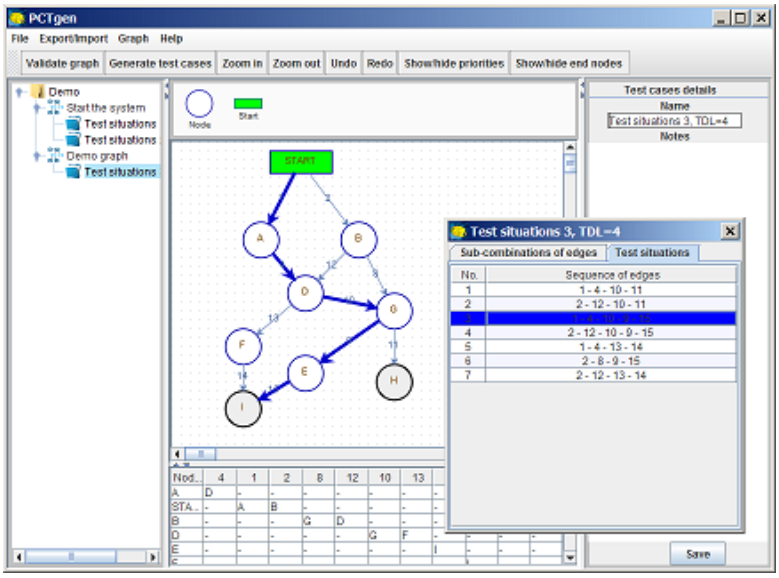
V Oxygenu je oproti Rhinu možné kreslit tři druhy diagramů, a to orientované grafy, stavové diagramy a aktivity diagramy.[19] Dále je zde umožněn import diagramů z formátu xml a csv a export diagramů ve formátech xml, csv a json. Oxygen rovněž disponuje možností ukládat celé projekty, které obsahují nejen diagramy, ale i testovací scénáře. Export testovacích scénářů je rozpracovaný do dvou druhů exportovaného formátu. Poslední významnou odlišností Oxygenu od Rhina je výběr z pěti experimentálních algoritmů vyvíjených na STILL.[13]

Důležitost nástroje Oxygen pro platformu Rhino není jen ve vzoru, co se funkcionality týče, ale především v roli, jež hraje sdílení algoritmu pro generování testovacích scénářů. V této chvíli je využíván pouze algoritmus PCT[12], ale do budoucna se uvažuje o integraci dalších algoritmů.

4.3 Architektura modulu

Na základě požadavků na nový modul, analýzy již vzniklé platformy a architektury knihovny Cytoscape.js jsem se rozhodl zvolit řešení nejméně duplikující kód, který již platforma obsahuje a zároveň požadovat co nejméně změn v již vzniklé struktuře.

Stejný algoritmus se stejnými vstupy a výstupy bude generovat testovací scénáře i pro nový modul. Klíčové pro tento algoritmus je znát počáteční uzel - dále se bude řídit jen podle hran a priorit. Díky tomu není nutné



Obrázek 4.7: Uživatelské rozhraní desktopového nástroje Oxygen - převzato[5]

měnit vnitřní strukturu aplikace. Je potřeba je vytvořit uživatelské rozhraní, jež bude odpovídat požadavkům UML aktivity diagramům

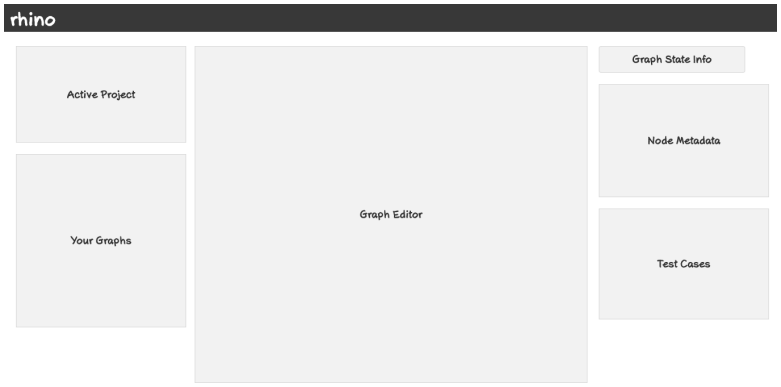
O vzhled diagramů se stará knihovna Cytoscape.js, proto se nabízí řešení v podobě zdědění již vzniklého jádra editoru a využití stejných balíčků, které využívá i orientovaný graf. Nový modul vyžaduje změnu balíčku pro obsluhu hran a balíček pro kontextové okno. Balíčky pro zoom a mřížku na pozadí není nutné měnit. Veškerá logika je pak skryta pod rozhraním Cytoscapu, jak můžete vidět na obrázku 5.1. Pro zbytek aplikace pak vše probíhá stejně jako u orientovaných grafů.

4.4 Uživatelské rozhraní

Platforma Rhino má své uživatelské rozhraní rozděleno na dvě hlavní obrazovky. Na první obrazovce jsou dva formuláře. Jeden pro registraci a druhý pro přihlášení. Druhá obrazovka pak představuje zbytek platformy a nachází se zde všechny funkční prvky. Opět je kladen hlavní důraz na jednoduchý design a přehlednost jednotlivých komponent. Pro návrh uživatelského rozhraní byly použity wireframy[18].

Na obrázku 4.8 je schématicky znázorněno rozložení druhé obrazovky. Ve středu se nachází prostor pro kreslení diagramů a v okolí informativní nebo pomocná okna. Levý panel ve své horní části obsahuje informaci o právě zobrazeném projektu a pod ním se nachází seznam grafů umístěných v tomto projektu. Tyto grafy se zde zobrazují pod sebe podle toho, v jakém pořadí byly do projektu vytvářeny. V okně „Your Graphs“ jsou umístěny ikony pro import a export grafů. Vzhledem k možnosti vkládání více diagramů najednou, se bude ikona pro import nacházet někde nad seznamem diagramů. Ikonu pro

[18] Neinteraktivní zjednodušený návrh uživatelského prostředí[39]



Obrázek 4.8: Wireframe celého uživatelského prostředí platformy Rhino

export diagramu má každý diagram vlastní vedle sebe v sousedství dalších ikon, které již v tuto chvíli platforma obsahuje.

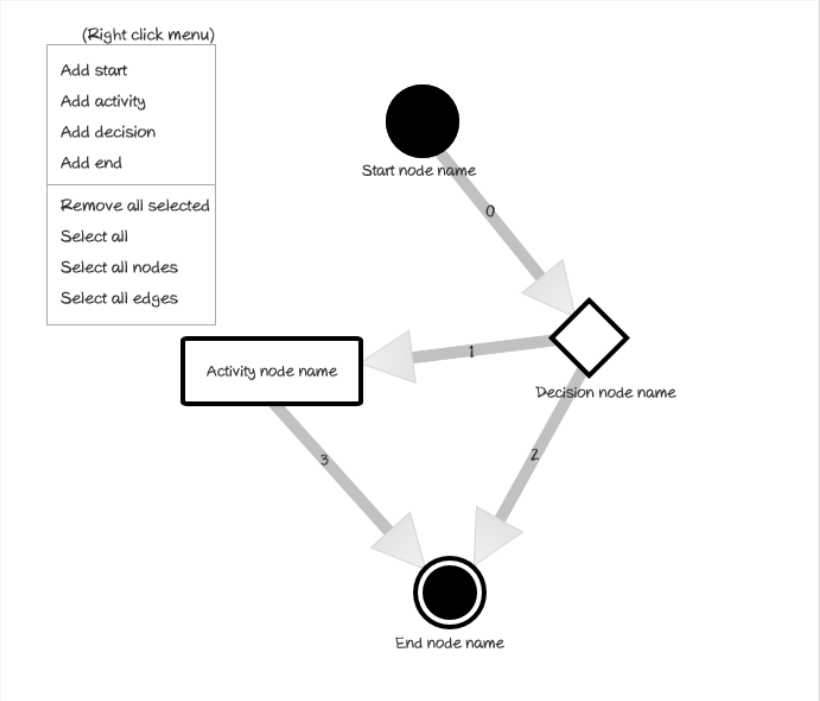
V pravém panelu se hned nahoře nachází informativní okno, které signalizuje, v jakém stavu se právě otevřený diagram nachází. Zda je uložen do databáze nebo zda během ukládání nastala nějaká chyba. Zbytek panelu zůstává v základním zobrazení prázdný. Okno s metadaty jednotlivých elementů se zobrazuje po kliknutí na element pravým tlačítkem myši a okno s testovacími scénáři pouze potom, co se uživatel rozhodne je vygenerovat nebo znovu zobrazit. Veškerá práce s metadaty se bude odehrávat v okně s metadaty a ikona pro export testovacích scénářů se bude nacházet v hlavičce okna testovacích scénářů.

4.4.1 Aktivity diagram

Jak již bylo zmíněno v požadavcích, editor bude podporovat kreslení základních čtyř uzlů UML aktivity notace. Na obrázku 4.9 jsou všechny vyobrazeny ve smyšleném diagramu, jak budou ve finále vypadat v platformě Rhino. Vzhled hran zůstane stejný, jaký se využívá v Rhinu pro hrany v orientovaných grafech.

Názvy uzlů se nacházejí pod daným uzlem. Výjimkou je pouze aktivity uzel, který má název uvnitř. Všechny tři uzly mají stejnou výšku a šířku. Pro aktivity uzel se šířka i výška vypočítává dynamicky na základě délky, počtu řádků názvu a pevně definovaného odsazení. Názvy hran se centrují do středů délky hrany.

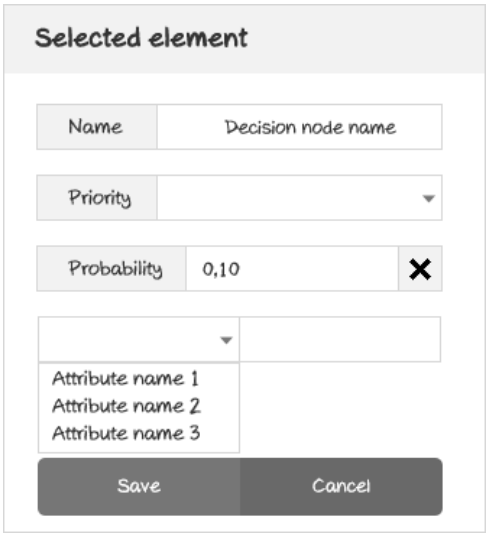
Důležitou součástí editoru diagramu je také menu, které se zobrazuje po kliknutí do prostoru editoru pravým tlačítkem menu. Je to jediný způsob, jímž lze do prostoru přidávat uzly nebo z něj odebírat elementy. Nacházejí se zde ještě rozšiřující možnosti v podobě označení nebo smazání všech elementů a označení všech uzlů nebo hran. Po kliknutí pravým tlačítkem myši na element se zobrazí možnost daný element odstranit.



Obrázek 4.9: Wireframe editoru UML aktivity diagramu

4.4.2 Metadata

Dodatkové atributy budou přidány do okna v pravém panelu pod již stávající klíčové atributy. Na základě požadavků budou dodržovat stejný design jako jméno a priorita - jen s tím rozdílem, že musí mít i křížek pro případné odstranění.



Obrázek 4.10: Wireframe pro přidávání dodatkových atributů k elementu diagramu

Na obrázku 4.10 je vidět fáze, kdy uživatel je v procesu přidávání nového atributu. Z vyjíždějícího menu bude mít možnost vybrat z předem zadefinovaných atributů, kterým může následně určit hodnotu dle svého výběru. Nebude nijak omezeno, zda se jedná o číslo či řetězec.

4.4.3 Import a Export

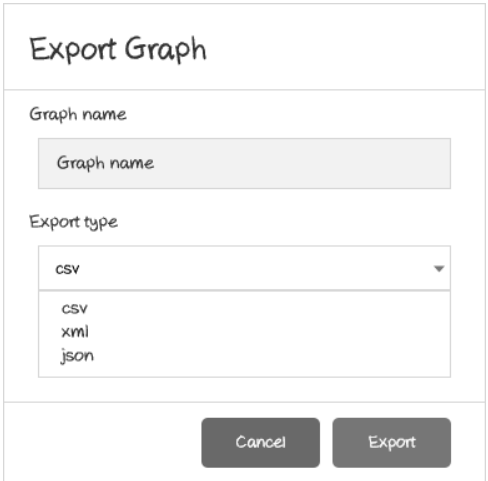
Pro uživatelské rozhraní importu a exportu bylo klíčové zvolit formu, kterou se zobrazí formulář. Tento formulář pak dospecifikuje uživatelské parametry pro export, nebo nastíní informace o importovaném souboru, aby mohl proběhnout úspěšný import diagramu.

Z důvodu rozhodnutí o zachování dvojstránkového rozložení aplikace jsou formuláře realizovány jako modální okna[19]. Ta se zobrazí na popředí po kliknutí na dotyčnou ikonu v obrazovce editoru. Jejich pozice je blíže popsána výše, v obecném popisu uživatelského prostředí.

Diagramy

Je třeba mít možnost exportovat diagramy ve více formátech a importovat z více platforem. Export, bude možný ve formátech xml, json a csv. Dle platformy bude ve zmíněných formátech možný i import.

U exportu je situace jednodušší než u importu – především z důvodu jednotného zdroje diagramů. Zde je klíčové pouze vybrat exportní typ souboru. Název diagramu je pro jednoduchost pevně stanoven jako název souboru. Pokud není zvolen typ vyexportovaného souboru, není možné daný diagram vyexportovat.



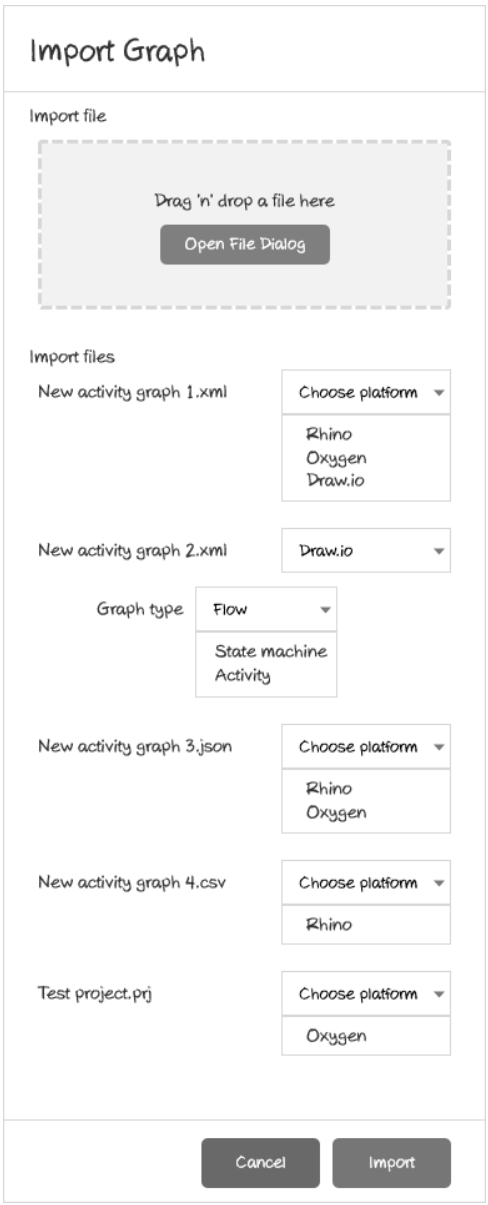
Obrázek 4.11: Wireframe modálního okna pro export diagramu

Podstatně složitější oproti exportu diagramů je jejich import. Zde je od uživatele třeba doplnit více údajů, které jsou pro import zásadní. Veškeré

[19] Okno aplikace, zobrazující se v popředí. Využívá se ve chvílích, kdy je třeba neztratit povědomí o zbytku aplikace, ale zároveň zobrazit něco důležitého.[40]

možnosti, které mohou nastat, jsou znázorněny na obrázku 4.11.

Pro import je v prvé řadě podstatné vybrán způsob vkládání souborů. V tomto ohledu je zvolen způsob přetahni a pust pro pohodlí uživatelů. Zároveň je zde možnost vybrání z adresáře počítače. Tento adresář se zobrazí po kliknutí na tlačítko „Open File Dialog“



Obrázek 4.12: Wireframe modálního okna pro import diagramu – všechny možnosti

Níže se pak bude nacházet výčet vložených souborů. Ve chvíli, kdy uživatel otevře modální okno a nebude vložený žádný soubor, objeví se text informující o tom, že žádný soubor nebyl vložený. Pokud uživatel soubor vloží, objeví se jeho název i s koncovkou. Vedle názvu se zobrazí vyjíždějící menu pro volbu

platformy, z níž byl diagram exportován. Pokud nebude zvolena platforma, nebude možné soubory importovat.

Jak je rovněž vidět na obrázku 4.12, možnosti platforem jsou pro každý podporovaný formát odlišné.

V případě, že bude zvolena platforma „Draw.io,“ objeví se ještě jedno vyjíždějící menu, které vybere, o jaký typ diagramu jde. Je to z toho důvodu, že existuje celá řada způsobů, jakými lze nakreslit v Draw.io aktivity nebo orientované grafy. Na základě zvoleného způsobu se potom liší konkrétní struktura souboru. Proto se Rhino nesnaží na základě souboru detekovat typ diagramu, ale vyžaduje jeho zvolení uživatelem.

Testovací scénáře

Vzhled modálního okna pro export testovacích scénářů bude řešen stejným způsobem jako modální okno pro export diagramů. Opět je třeba vybrat pouze typ vyexportovaného souboru. Název zůstává předdefinovaný na základě názvu testovacích scénářů, které uživatel volí při požadavku na jejich generování. Není-li typ zvolen, testovací scénáře není možné exportovat.

Obrázek 4.13: Wireframe modálního okna pro export testovacích scénářů

Kapitola 5
Implementace

V této kapitole podrobně popisuji způsob implementace všech nově přidaných funkcionalit do platformy Rhino a zároveň se věnuji i popisu jejich funkčnosti.

5.1 Aktivity diagram

Prvním a zároveň hlavním předmětem praktické části této bakalářské práce bylo naimplementovat modul pro rozšíření platformy Rhino o základní kreslení UML aktivity diagramů a následnou možnost z nich generovat testovací scénáře. Tento úkol byl zahrnut již do Semestrálního projektu, na němž jsem pracoval minulý semestr.

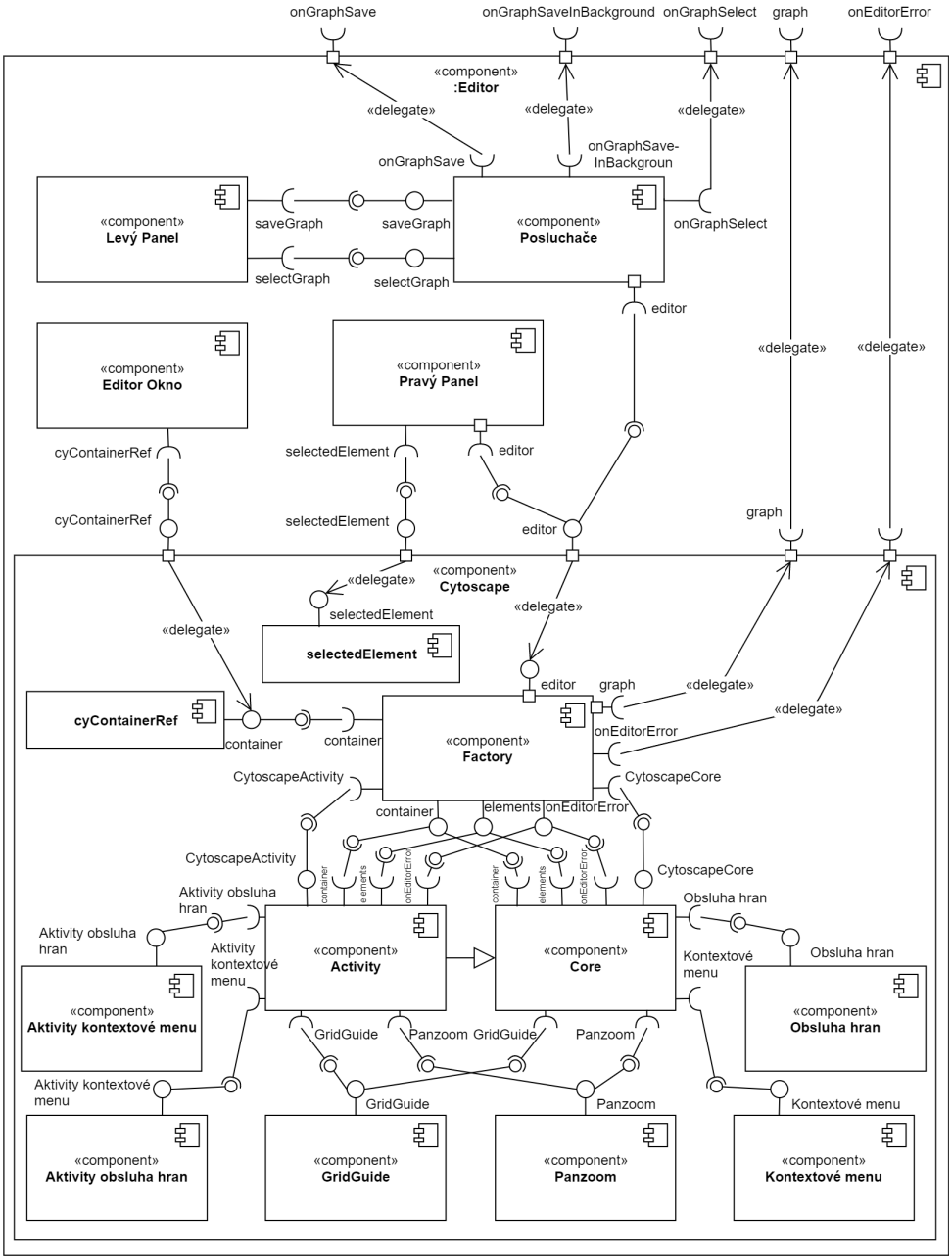
Na základě analýzy vzniklé architektury platformy vyšlo najevo, že je vytvořena s ohledem pro rozšíření kreslení více typů diagramů. To celou implementaci zjednodušilo a též snížilo riziko chyb na minimum.

Z hlediska uživatelského rozhraní bylo třeba v modálním okně pro tvorbu nového diagramu rozšířit vyjíždějící menu. Zde se nachází výčet možných typů diagramů, které jsou podporované platformou. Dále bylo třeba vytvořit vzhled aktivity diagramů dle návrhu. Posledním úkolem bylo upravit kontextové menu tak, aby odpovídalo požadavkům diagramu.

Struktura aplikace obhospodařující diagramový editor je zobrazena na obrázku 5.1. Pro zbytek aplikace se chová jako jedna komponenta. Její vnitřní struktura se dále dělí podle funkčnosti. Editor potřebuje mít z jádra aplikace možnost volat základní funkce „onGraphSave“, „onGraphSaveInBackground“ a „onGraphSelect“, které jsou zadefinovány v Redux aktivitách diagramu. Potřebuje též volat funkci „onEditorError“, která se nachází v Redux aktivitách editoru. Tato funkce zobrazuje chybová hlášení v modálním okně v případě, že uživatel bude chtít nakreslit jednu ze zakázaných možností v aktivity diagramu. (odkaz na kapitolu s návrhem) V poslední řadě je potřeba objekt diagramu, který je uložen v Redux store.

Pro jednoduchost přístupu k proměnným diagramu v případě levého panelu a možnosti úpravy diagramu nebo příjímání informací o změně v diagramu v případě pravého panelu jsou pravý a levý panel zanořeny až do komponenty editoru. Oba panely jsou z pohledu funkčnosti s diagramem úzce spjaty.

Levý panel obsahuje informaci o seznamu všech diagramů v daném projektu. Uživatel může ze seznamu vybrat konkrétní diagram a ten pomocí tlačítka



Obrázek 5.1: Diagram komponent pro architekturu implementace UML aktivity diagramu do platformy Rhino

„Save Graph“ uložit do databáze. Levý panel tak umožňuje více možností práce s diagramem, ale ostatní funkcionalita je předávána přímo z Redux aktivit.

Pravý panel obsahuje okno s metadaty konkrétního elementu z právě otevřeného diagramu, dále má přístup k dalším metodám ovlivňujícím stav diagramu. Na obrázku 5.1 jsou znázorněny jedním rozhraním „editor“. Mezi výše zmíněné metody patří:

- selectedElement – získání konkrétního uzlu
- getElementNameById – vyhledání uzlů dle id[1] v testovacím scénáři
- highlihtPath – zvýraznění testovacího scénáře v diagramu
- validateName – kontrola, zda nový název elementu je možný
- setElementAttribute – přidání dodatkového atributu elementu
- removeElementAttribute – odebrání dodatkového atributu elementu

Stejně jako v levém, i v pravém panelu jsou využívány další funkce, které jsou definovány v Redux aktivitách diagramu a testovacích scénářů.

Samotné okno editoru diagramu je již pod kontrolou knihovny Cytoscape, která mimo sebe vystavuje pouze HTML uzel div[2]. Je možné mu přidat vlastní vzhled a vše, co se zobrazí uvnitř, závisí pouze na referenci do knihovny Cytoscape. Konkrétně se stav okna editoru drží v komponentně „cyContainerRef“, která je součástí struktury Cytoscape.

Vznik vnitřní struktury editoru je ovlivněn návrhovým vzorem „Factory“. Na základě vstupních parametrů vytvoří orientovaný graf a přidá knihovny s rozšiřujícími vlastnostmi editoru. Zde bylo třeba rozšířit možnosti „Factory“ o možnost vytváření aktivity diagramů. To je realizováno pomocí proměnné typu diagramu, která si v sobě drží instance diagramu, nacházející se na vstupu do „Factory“.

Jádro diagramu je definováno v komponentě „Core“. Zde jsou definovány všechny potřebné metody, které upravují diagram. Mezi ně například patří:

- createNode – vytvoří uzel
- isElement – odlišuje zda se jedná o cytoscape element nebo jinou entitu
- generateNodeParams – vytváří strukturu uzlu, která je používána pro ukládání do databáze
- generateEdgeParams – vytváří strukturu hrany, která se používá pro ukládání do databáze
- getAllElements – vrátí všechny elementy diagramu
- removeElement – odstraní element z diagramu
- updateElementData – aktualizuje metadata elementu

Těchto metod je však mnohem více. Některé z nich se volají z přidaných balíčků, jiné se volají z pravého nebo levého panelu a další se využívají pouze vně komponenty „Core“. Například přidání nebo odebrání elementu z diagramu lze uskutečnit pouze přes kontextové menu, které je realizováno samostatným balíčkem.

[1] Zkratka pro unikátní identifikátor

[2] Celám názvem Hypertext Markup Language je způsob zápisu obsahu webových stránek. Využívá se stromové struktury, kde uzly představují části stránky.[41]

Aktivity modul je samostatná komponenta, jež dědí všechny metody od Komponenty „Core“ a pouze přepisuje ty, které se na základě typu diagramu liší. Mezi tyto metody patří:

- createNode – je zde navíc kontrolováno, zda již neexistuje počáteční uzel, aby v diagramu mohl být maximálně jeden
- generateNodeParams – je definován navíc atribut type, pro udržení typu uzlu
- setStartNode – definuje povinný atribut automaticky, bez vědomí uživatele
- createConfig – definuje vzhled jednotlivých elementů a dodatkové designové změny pro jednotlivé priority nebo při najetí myší na element

Dále jsou zde přidány dvě metody. Obě metody slouží pro interní práci s hranami. Jsou též volány z příslušného balíčku „EdgeHandlers“ zobrazeného v obrázku 5.1 jako komponenta „Aktivity obsluha hran“. Metoda „checkEdgeSourceNode“ kontroluje, zda hrana nevede z koncového uzlu. Metoda „checkEdgeAfterAdd“ zase kontroluje, zda nějaká hrana nevede do uzlu počátečního. Nastane-li chyba je volána metoda „onEditorError“, která zobrazí modální okno s příslušnou chybovou hláškou.

V případě druhé metody balíček bohužel nepodporuje možnost průběžného stopování kreslení hrany. Proto je potřeba kontrolu provést až po vytvoření hrany a v případě chyby je nutné hranu smazat.

Z obecnosti balíčků bylo možné využít stejný balíček „Panzoom“[42] a „GridGuide“[43] pro oba typy diagramů. Oproti nim bylo třeba vytvořit nové využití balíčku pro kontextová menu[44] a pro obsluhu hran[45]. Z toho důvodu vznikly dvě nové komponenty, které mají předponu „Aktivity“. V kontextovém okně bylo oproti orientovaným grafům potřeba změnit způsob vkládání uzlů. Kontextové menu orientovaných grafů podporuje pouze možnost „Add node“. V případě, že se jedná o první vložený uzel, automaticky se zobrazí jako počáteční, jinak se vkládají uzly obyčejné. Aktivity diagram vyžaduje možnost vkládání více odlišných uzlů. Oproti orientovaným grafům je taktéž odebrána možnost nastavení uzlu jako počátečního. Zbytek kontextového okna zůstává pro oba diagramy stejný.

Obsluha hran je rozšířena o posluchač na události[3], kdy uživatel chce začít vytvářet hranu. Z tohoto posluchače se pak volá metoda na kontrolu, zda se uživatel nepokouší vytvořit hranu z koncového uzlu. Dále je rozšířen o posluchač, kdy je hrana vytvořena a je možné zkontrolovat, zda daná hrana nevede do počátečního uzlu. Pokud by bylo třeba vytvořit další omezení pro kreslení hran mezi uzly diagramu, stačilo by do posluchačů přidat další kontrolní metody.

[3] Speciální funkce, která je vykonána v tu chvíli, kdy se stane daná událost[46]

5.2 Metadata

Implementace dodatkových atributů elementů je vymyšlena tak, aby se co nejméně zasahovalo do již funkční správy atributů názvu a priority. Elementy obsahují ještě několik interních atributů, které se nevizualizují. Uzly obsahují z pohledu databáze ještě jeden povinný atribut „type“ pro aktivity diagramy a nastaven na „null“ pro orientované grafy, který určuje typ uzlu. Orientovaný graf je nepotřebuje, protože zde je klíčový atribut „nodeStart“, který je buď „true“ nebo „false“.

Pro implementaci libovolného množství dodatkových atributů k elementům byl zvolen způsob přidání dalšího povinného atributu pro všechny uzly. Tento atribut v sobě bude obsahovat všechny vložené dodatkové atributy. Je pojmenován „restAttributes“. Je ukládán do databáze jako „BasicDBObject“. Jedná se o základní implementaci bson[4] objektu pro Mongo databáze[48]. Více nebylo třeba do databáze zasahovat, protože přenos dat mezi serverem a klientem tato úprava nijak neovlivnila.

V rámci klientské části aplikace jsou metadata elementů velmi úzce spjata s diagramem. Veškeré operace s nimi se dějí na úrovni logiky editoru. Jednotlivé metody na zobrazení, tvorbu, úpravu i smazání jsou vyvedeny do pravého panelu, kde s nimi lze v rámci okna pro metadata pracovat. Stahování metadat z databáze a jejich ukládání zpět se děje současně s celým diagramem.

Důležitým aspektem je též definice možných dodatkových atributů, z kterých bude moci uživatel vybrat a přidat je k danému elementu. Výčet možných atributů se definuje v interním souboru konstant editoru. Je ho možné najít zde: „src/EditorPage/cytoscape/constants.js“. Konkrétně se objekt nazývá „CUSTOM_ELEMENT_ATTRIBUTES“.

Aby nemohl mít jeden element dva stejné atributy, rozdělují se dodatkové atributy na již přidané a ještě nepřidané. Ty nepřidané se vloží do vyjíždějícího menu pro přidání nových dodatkových atributů.

Logika operací s dodatkovými atributy pak funguje na operacích s objektem „restAttributes“. Pokud se jedná vytvoření, je atribut se svou hodnotou přidán ze seznamu možných atributů. Když se hodnota atributu změní, je hodnota v objektu daného atributu nahrazena a v případě smazání je hodnota z objektu smazána.

5.3 Import a Export

Oproti implementaci modulu pro aktivity diagramy a práce s metadaty elementů bylo třeba vytvořit logiku úplně od začátku a napojit veškerý nový mechanismus na mechanismus stávající. Protože se jedná o možnost přenášení dat skrz další platformy, byl kladen důraz i na volbu exportního formátu diagramů. Na export testovacích scénářů však není kladen takový důraz jako na export diagramů.

[4] Zratka pro „Binary JSON“[47]

5.3.1 Diagramy

Jak již bylo zmíněno v návrhu, platforma Rhino vychází z desktopového nástroje Oxygen. Z potřeby mít možnost přenášet diagramy mezi platformami bez ztráty dat bylo nutné vyřešit několik problémů.

První problém byl v rozdílném formátu ukládání diagramů interně. Z pohledu struktury uloženého diagramu má Oxygen oproti Rhinu o poznání plošší způsob ukládání jednotlivých elementů. Dalším souvisejícím problémem bylo větší množství atributů, které Oxygen narozdíl od Rhina podporuje. Mimo jméno a prioritu standardně definuje ještě tyto atributy elementů: „description“, „limitedConnectionProbability“, „height“ a „width“. „Height“ a „width“ jsou vzhledové atributy bez hran, definující rozměry uzlů.

Druhý problém představovalo různé pojmenování totožných věcí. Jednalo se např. o pojmenování xové a ylonové souřadnice. V Oxygenu se pojmenovávají „xpos“ a „ypos,“ a zatímco v Rhinu „x“ a „y,“ přičemž jsou zanořeny do objektu „position“. Další rozdíl se nachází v pojmenování priorit. V Oxygenu jsou pojmenovány s malými písmeny „(not defined)“, „low“, „medium“ a „high“ a v Rhinu jsou úplně stejné názvy jen s velkými písmeny na začátku. S prioritami souvisí ještě jeden rozdíl. V Oxygenu je možné, aby element měl prázdnou hodnotu priority. To ale v Rhinu není možné. Musí být nastavena vždy na jednu ze čtyř možností. Dalším rozdílem jsou různě pojmenované druhy diagramů. V Oxygenu jsou pojmenovány jako „ORIENTED_GRAPH“ a „ACITIVITY_DIAGRAM“ a v platformě Rhino jako „Flow“ a „Activity“. Poslední rozdíl spočívá v definici druhu uzlu. V Oxygenu jsou definované atributem „style“, který může nabývat pro aktivity diagramy hodnot:

- STYLE_ACTIVITY_START
- STYLE_ACTIVITY_DECISION
- STYLE_ACTIVITY_NODE
- STYLE_ACTIVITY_END

Dobré je, že Oxygen neobsahuje více druhů uzlů pro UML aktivity diagramy, než bylo požadavkem pro UML aktivity diagramy v platformě Rhino.

Třetí problém byl rozdílný souřadnicový systém Oxygenu a Rhina. Oxygen má počátek souřadnicového systému[5] nahoře v levém rohu. Rhino má sice také zadefinovaný souřadnicový systém, ale ten se vždy vycentruje vůči všem vloženým elementům v daném diagramu.

Přestože lze platformu Rhino využít i ke kreslení diagramů, je daleko lépe uzpůsobená pro generování testovacích scénářů. K tomu je třeba mít možnost importovat diagramy z některé ze specializovaných webových aplikací pro kreslení diagramů. Na základě požadavků byla k tomuto účelu zvolena aplikace Draw.io. Jedná se o specializovanou aplikaci pro kreslení nejrůznějších typů grafů. Podporuje mnoho různých druhů diagramů a způsob kreslení. Je zde možné kreslit neomezeně velké diagramy a uživatel má na výběr z mnoha

[5] Počátkem se rozumí bod, kde x-ová i y-ová souřadnice je nulová.([0,0])[49]

druhů elementů. Draw.io rovněž podporuje export svých diagramů ve formátu xml.

Velká variabilita a mnoho možností aplikace však činí v případě snahy o import diagramů do jiné platformy značné potíže. Jednotlivé typy diagramů mají různé podoby výstupního xml. Z těchto důvodů bylo zapotřebí vymyslet jiný způsob identifikace typů elementů především pro UML aktivity diagramy.

Export

Pro export je velmi důležité zvolit formát a data, která je třeba exportovat. Jak již bylo zmíněno, formát ukládání diagramů v platformě Rhino do databáze je poměrně rozvětvený a bohatý na interní proměnné. Data se ukládají do databáze jako ve formátu json. Ukázka struktury uloženého diagramu, jenž obsahuje počáteční uzel, uzel aktivity a koncový uzel spojené hranami, je zobrazena na obrázku 5.2.


```
{
  "id": "5ec11e8857087c250c59273e",
  "name": "Check Data",
  "type": "Activity",
  "elements": {
    "nodes": [
      {
        "classes": "", "grabbable": true, "group": "nodes", "locked": false,
        "removed": false, "selectable": true, "selected": false,
        "position": {"x": 432.8113116339027, "y": 241.07766677408958},
        "data": {
          "id": "e51e0283-14ed-49c2-941d-f0155c342cdf",
          "name": "a",
          "priority": "Undefinated",
          "startNode": true,
          "type": "startNode",
          "restAttributes": {}
        }
      },
      {...},
      {...}
    ],
    "edges": [
      {
        "classes": "", "grabbable": true, "group": "edges", "locked": false,
        "removed": false, "selectable": true, "selected": false,
        "position": {"x": 0, "y": 0},
        "data": {
          "id": "c417c17f-7d8e-44fb-be6d-c4c3d5164d84",
          "name": "a",
          "priority": "Low",
          "from": "a",
          "source": "e51e0283-14ed-49c2-941d-f0155c342cdf",
          "to": "c",
          "target": "d331b555-0183-459d-a207-0e1050a98961",
          "restAttributes": {}
        }
      },
      {...}
    ]
  },
  "testCases": []
}
```


Obrázek 5.2: Aktivity diagram v interním formátu platformy Rhino - zjednodušen a formátován ve Visual Studio Code

Uzly a hrany jsou v polích. Data o poloze jsou uložena zvlášť v objektu „position“. Všechny klíčové informace jsou uloženy v objektu „data“. Zde je též objekt „restAttributes“, který obsahuje přidaná metadata.

Formáty. Z důvodu požadavku možnosti exportované diagramy importovat do Oxygenu byl zvolen formát exportu. Na základě analýzy funkcionality

Oxygenu jsem zjistil, že funkční import do Oxygenu je pouze pomocí formátu xml[6]. Oxygen podporuje také import ve formátu csv[7], ale podporovaný formát souboru neobsahuje důležité informace o typu diagramu, a tedy se automaticky všechny naimportované diagramy zobrazí jako orientované grafy.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<graph name="Check Data" description="" id="0" linkage="" type="ACTIVITY_DIAGRAM" version="">
  <node ypos="100" xpos="200" name="a" width="25" description="" style="STYLE_ACTIVITY_START"
  id="1" limitedConnectionProbability="0" priority="(not defined)" height="25"/>
  <node ypos="270" xpos="203" name="b" width="35" description="" style="STYLE_ACTIVITY_END"
  id="2" limitedConnectionProbability="0" priority="(not defined)" height="35"/>
  <node ypos="182" xpos="261" name="c" width="80" description="" style="STYLE_ACTIVITY_NODE"
  id="3" limitedConnectionProbability="0" priority="(not defined)" height="40"/>
  <edge expectedResult="" name="1" description="" style="" id="4"
  limitedConnectionProbability="0" source="1" priority="low" target="3"/>
  <edge expectedResult="" name="2" description="" style="" id="5"
  limitedConnectionProbability="0" source="3" priority="low" target="2"/>
</graph>
```

Obrázek 5.3: Aktivity diagram exportovaný do formátu xml - formátován ve Visual Sturio Code

Na obrázku 5.3 je ukázka totožného diagramu jako na obrázku 5.2. Je uložen v platformě Rhino a exportován do formátu xml. V této podobě je možné diagram importovat také do Oxygenu. Formát json[8] je rovněž převzat z platformy Oxygen, oproti xml jej nelze do Oxygenu importovat.[9] Pro export formátu do csv jsem navrhl rozšířený zápis diagramu pro možnou specifikaci typu diagramů a jednotlivých uzlů tak, aby se co nejméně lišila od používaného formátu v Oxygenu. Oproti zápisu na obrázku 5.4 formát v Oxygenu neobsahuje druhý sloupec a první buňku.

Simple graf	ACTIVITY_DIAGRAM	1	2	3	4
a	STYLE_ACTIVITY_START	-	c	-	-
b	STYLE_ACTIVITY_NODE	d	-	-	-
c	STYLE_ACTIVITY_DECISION	-	-	d	b
d	STYLE_ACTIVITY_END	-	-	-	-

Obrázek 5.4: Aktivity diagram exportovaný do formátu csv - formátován v MS Excel

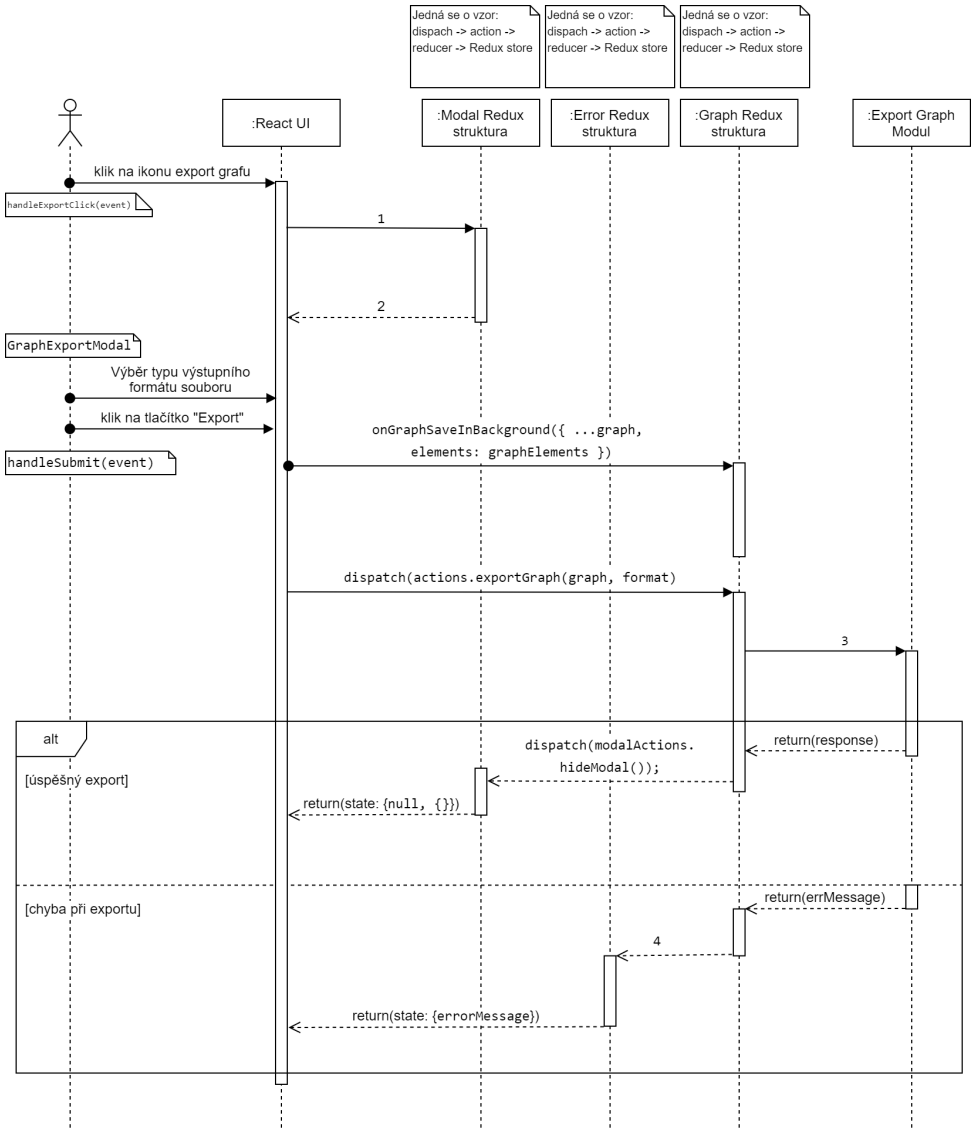
Obrázek 5.4 znázorňuje exportovaný diagram do formátu csv. Struktura souboru je definována:

- první buňka - název diagramu
- druhá buňka - typ diagramu
- první sloupeček - názvy uzlů
- druhý sloupeček - typy uzlů

[6] Extensible Markup Language[50]
[7] Comma Separated Values[51]
[8] JavaScript Object Notation[52]
[9] Formáty json a xml jsou převeditelné mezi sebou bez ztráty informace[53]

- první řádek - názvy hran
- matice sousednosti - definice pozice hrany pomocí uzlů, které spojuje (Např. hrana 2 vede z počátečního uzlu „a“ do rozhodujícího uzlu „c“.)

Integrace. Klíčovými aspekty pro integraci funkcionality pro export diagramu jsou získání dat diagramu pro export a napojení na stávající strukturu Reduxu. Zbytek logiky se odehrává ve vlastním modulu. Zpět je třeba vrátit jen informaci, zda byl export úspěšný nebo zda došlo k chybě.



Obrázek 5.5: Sekvenční diagram pro export diagramu

Celý proces exportu je zachycen v sekvenčním diagramu[10] na obrázku 5.5. Po kliknutí uživatele na ikonu exportu daného diagramu se odešle požada-

[10]Nejběžnější druh interakčního diagramu, který se zaměřuje na výměnu zpráv mezi řadou životních linií.[54]

Čísla	Popis
1	dispatch(modalActions.showModal({ modalType: modalTypes.GRAPH_EXPORT, modalProps: {key: graphId, graphId})
2	return(state: {modalType, modalProps})
3	exportManagerGraph(resolve, reject, dispatch, getState, graph, format)
4	dispatch(error(EXPORT_GRAPH, { errMessage }))

Tabulka 5.1: Vysvětlivky k sekvenčnímu diagramu pro export diagramu 5.5

vek na otevření modálního okna viz. obrázek 4.11. Poté uživatel zvolí typ exportovaného souboru a klikne na tlačítko exportovat. Následně je potřeba odeslat aktuální diagram do databáze, aby zůstala perzistentní. Platforma tuto operaci dělá automaticky po krátkých časových intervalech. Pokud by uživatel rychle něco nakreslil a chtěl své dílo okamžitě exportovat, exportovaný soubor bude prázdný.

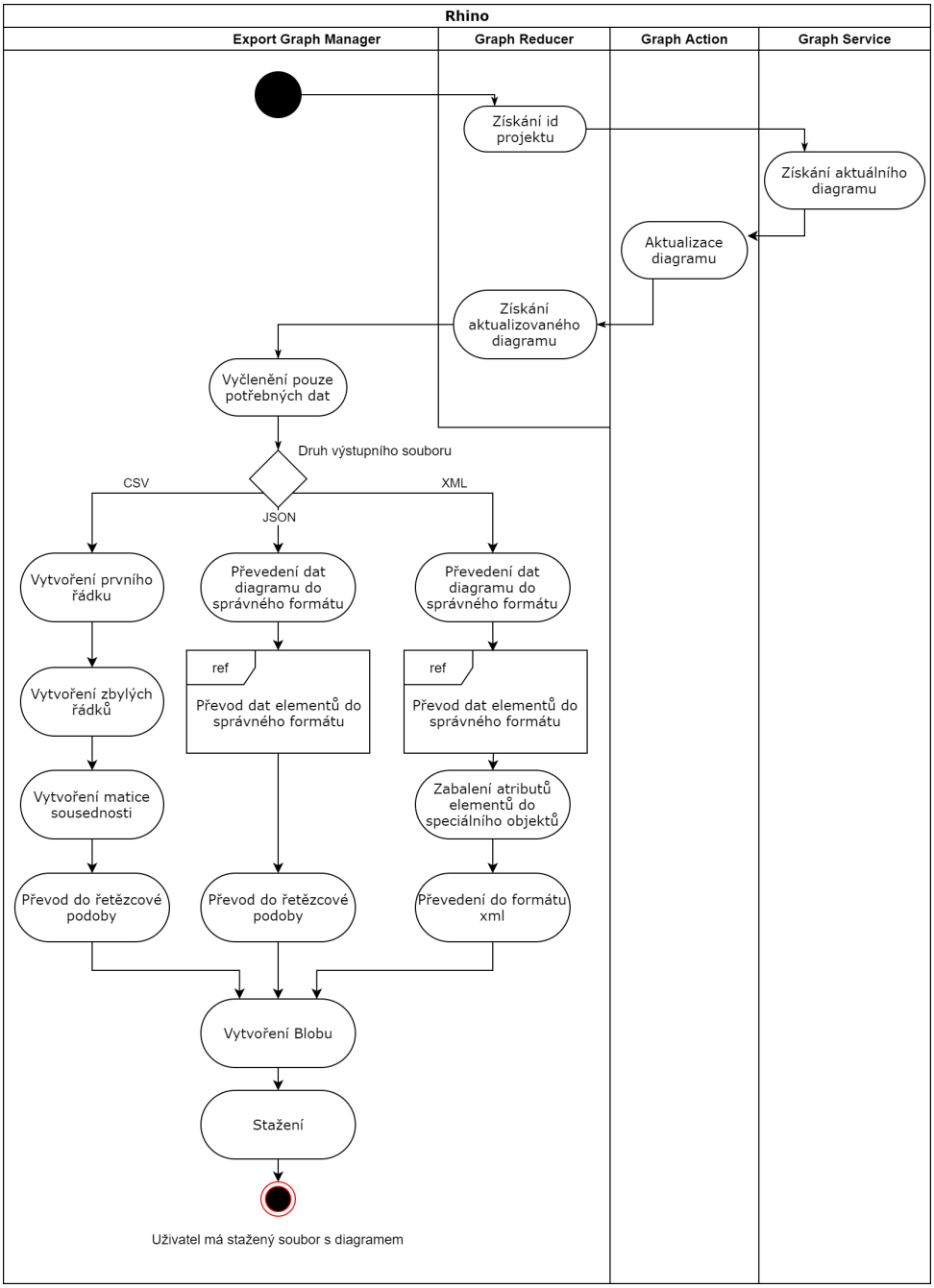
V dalším kroku je diagram z aktuálního stavu aplikace předán do modulu pro export, kde se na základě formátu výstupního souboru provedou dané operace a vrátí odpověď. Pokud vše proběhne v pořádku, zavře se modální okno pro export diagramu a uživatel může dále pokračovat v práci v platformě. Dojde-li při samotném exportu dojde k chybě, chybová hláška se uloží do stavu aplikace, modální okno se nezavře a chybová hláška se v něm následně zobrazí. Uživatel má poté možnost pokusit se daný diagram exportovat znovu nebo tlačítkem „Cancel“ modální okno zavřít.

Proces exportu. Pro zachování podobnosti s ostatními funkcemi, které implementují aktivity grafu a mění stav aplikace, je celý proces exportu pouštěn asynchronně pomocí funkcí Promise[11]. Jedná se o řešení pro zpětnou vazbu ke změně stavu aplikace. V případě úspěšného konce se volá funkce „resolve“ a případě neúspěchu se volá funkce „reject“. V těchto funkcích se Promise používají zdůvodu, že většina aktivit souvisí s komunikací se serverem.

Na začátku procesu je důležité získat aktuální verzi diagramu. I když je diagram ve vstupní promněné, je využito pouze jeho id. Ve skutečnosti se jedná jen o diagram ve stavu, který je uložen v Reduxstore. Pomocí dotazu do databáze si stáhneme aktuální verzi, kterou jsme tam před spuštěním samotného procesu exportu nahráli. Poté je nutné aktualizovat stav aplikace, aby obsahoval aktuální verzi diagramu. Následně je možné pracovat se správnou verzí diagramu a exportovat ho.

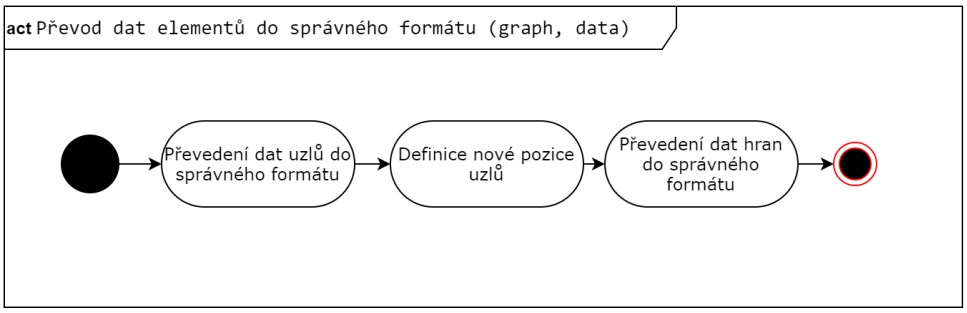
Jak je vidět na obrázku 5.6, dále se proces rozvětvuje dle typu exportního souboru. Z důvodu, že xml a json budou obsahovat veškeré informace, je jejich proces podobný. Nejdříve je v obou případech nutné převést data grafu do nového json objektu, který bude odpovídat nové struktuře. Poté do dané

[11] Objekt Promise představuje případné dokončení (nebo selhání) asynchronní operace a její výslednou hodnotu.[55]



Obrázek 5.6: Aktivity diagram exportu diagramu

struktury přidáme data elementů. V případě xml je dále potřeba upravit strukturu pro převod do formátu xml. To je realizováno pomocí javascriptové knihovny Xml-js. Pro využití knihovny je rovněž třeba zabalit data atributů do speciálního objektu „_attributes“, čímž je pro knihovnu možné identifikovat, kde se jedná o atribut a kde o vnořený element. Pro výstup ve formátu json je javascriptový objekt převeden do textového řetězce. Pro výstupní formát



Obrázek 5.7: Aktivity podproces pro převod dat do správného formátu

xml je aplikována knihovna, která na základě javasctiptového objektu vytvoří textový řetězec obsahující data v xml formátu.

V případě výstupního souboru ve formátu csv je potřeba postupovat dle struktury formátu. Nejdříve je zadefinovaný první řádek tabulky. Ten dle navrženého formátu obsahuje název diagramu, typ diagramu a výčet názvů všech hran. Poté jsou vytvořeny všechny zbylé řádky. Jsou zde uvedeny názvy jednotlivých uzlů a jejich typy. V dalším kroku je třeba po řádcích vytvořit matici sousednosti. Kde je klíčové, zda hrana vychází z daného uzlu. Pokud ano, je nutné najít a vložit název uzlu, do něhož daná hrana vede. Pak jen stačí opět převést celou vytvořenou tabulku do řetězce, kde jsou jednotlivé buňky v řádku oddělené středníkem.

Konec procesu probíhá pro všechny tři varianty výstupních formátu stejně. Řetězec se vkládá do Blobu[12]. Na tento Blob se vytvoří url adresa, která se vloží do html elementu odkazu. Tento odkaz se vloží jako neviditelný do html DOM[13] stránky a pomocí javascriptu se proklikne. Tím dochází ke stažení souboru do počítače uživatele. Neviditelný odkaz se následně z DOMu smaže, čímž se zajistí původní stav DOMu.

Import

Celý proces importu diagramů opět začíná kliknutím uživatele na ikonu import diagramů. Pak je zpracován požadavek na zobrazení modálního okna, které umožní uživateli vkládat do platformy své soubory.

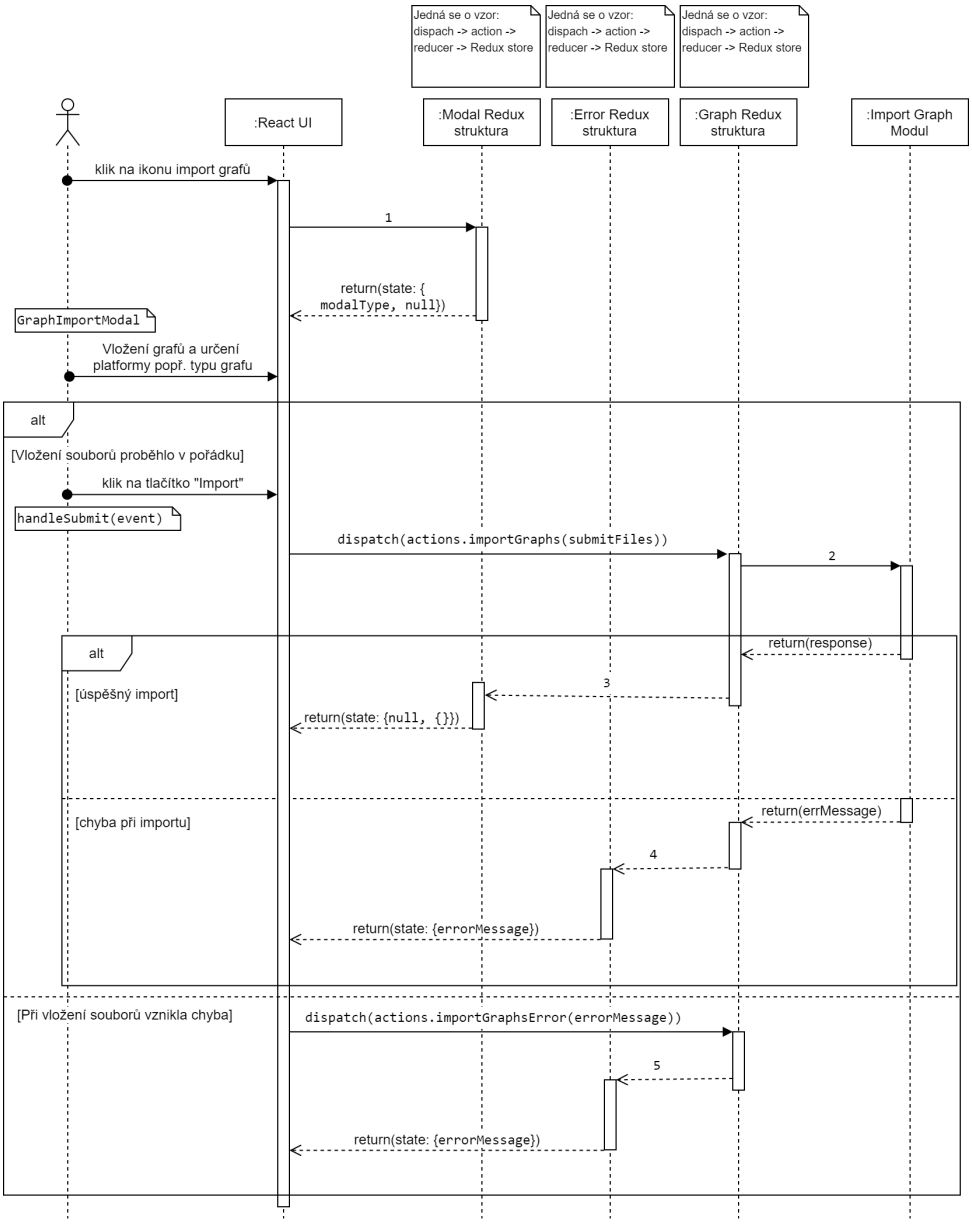
Uživatelské rozhraní. Knihovna Dropzone umožňuje mimo jiné nastavit podmínky, za kterých nebude možné vložit soubor do platformy. Zde se kontroluje formát souboru na základě přípony. Pokud se nejedná o jeden z podporovaných formátů, je zobrazena chybová hláška. Další možností vložení souboru je otevření dialogu výběru souboru. Otvírá se po kliknutí na tlačítko, jež se nachází uvnitř pole, kam se vkládají soubory. Díky knihovně lze nastavit, aby se v dialogu zobrazovaly pouze soubory podporovaných formátů.

[12] Souborový objekt neměnných nezpracovaných dat; mohou být čteny jako textová nebo binární data.[56]

[13] DOM (Document Object Model) je datová reprezentace objektů, které tvoří strukturu a obsah dokumentu na webu.[57]

Po úspěšném vložení podporovaného souboru do modálního okna se zobrazí název nově přidaného souboru a možnost výběru, z jaké platformy byl daný soubor exportovaný. V případě zvolení Draw.io se navíc objeví vyjíždějící menu pro volbu typu diagramu.

Pokud jsou zvoleny všechny platformy, popřípadě i všechny typy diagramů, je možné kliknout na tlačítko „Import“ a přesunout data souborů k vnitřnímu zpracování. Celý proces přesunu dat je znázorněn na obrázku 5.8.



Obrázek 5.8: Sekvenční diagram pro import diagramů

Integrace. Z praktických důvodu se samotné soubory a informace o platformách drží oddělené. Souvisí to s tím, že soubory obhospodařuje knihovna

Čísla	Popis
1	dispatch(modalActions.showModal({ modalType: modalTypes.GRAPH_IMPORT})
2	importManager(resolve, reject, dispatch, getState, submitFiles)
3	dispatch(modalActions.hideModal())
4	dispatch(error(IMPORT_GRAPH, { errMessage }))
5	exportManagerTestCases(resolve, reject, graph, selectedTestCase, format)

Tabulka 5.2: Vysvětlivky k sekvenčnímu diagramu pro import diagramu 5.8

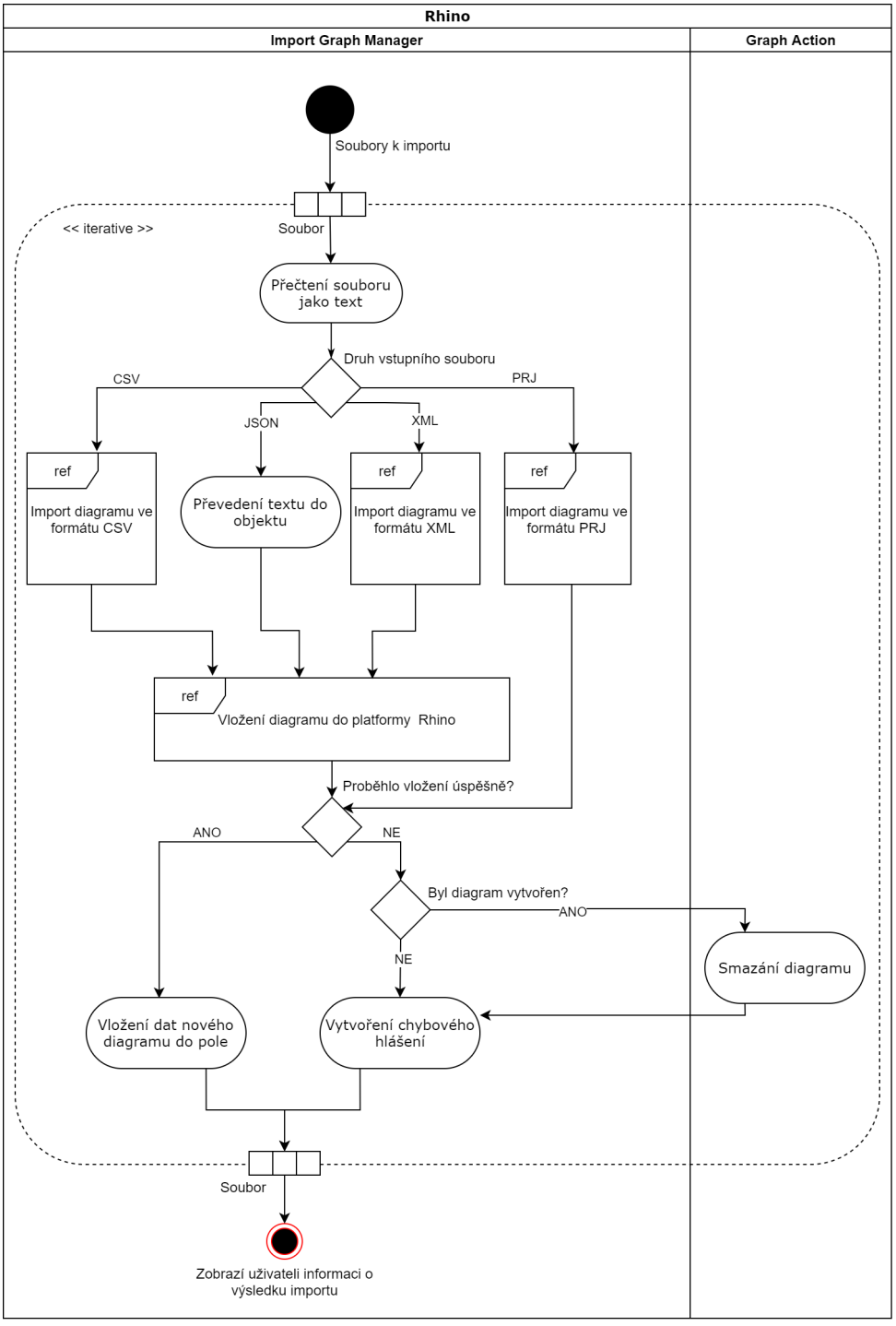
a informace o platformách jsou zpracovávány mimo. Aby se však udržela souvislost, jsou informace a soubory uchovávány v polích. Informace k souborům jsou uloženy v druhém poli na stejné pozici. Po kliknutí uživatelem na tlačítko importu se informace spárují. Data se odesílají do příslušné reduxové aktivity.

Aktivita je implementovaná po vzoru všech ostatních diagramových aktivit. Nejdříve vznikne požadavek na import diagramu, poté je asynchronně spuštěn modul pro import diagramů, kam jsou vloženy všechny informace o vložených souborech. Pak se jen čeká na zpětnou vazbu. Pokud je import souborů úspěšný, modální okno se uzavře. V opačném případě zůstane otevřené a zobrazí se chybová hláška. V případě, kdy je do platformy importováno více diagramů, stačí, aby byl úspěšný pouze jeden a modální okno se zavře.

Vzhledem k možnosti importovat do platformy více souborů najednou, je třeba moci tentýž proces pro import jednoho souboru aplikovat pro každý vložený soubor zvlášť. Z povahy řešení se nabízela možnost daný problém vyřešit asynchronně, což by sice ulehčilo řádky kódu, ale na druhou stranu by tím docházelo k pozdějším kolizím v procesu vkládání diagramu do databáze. Z tohoto důvodu je zde zvoleno řešení synchronního[14] iterátoru[15], kdy se diagramy importují postupně. Protože je více aktivit v procesu asynchronních[16], bylo zde potřeba zvolit řešení pomocí asynchronního iterátoru. Jedná se o uměle vytvořený objekt, jenž díky poslední verzi JavaScriptu podporuje tzv. Symboly[17], v rámci nichž je možné vytvořit iterátor, který bere jednotlivé vložené soubory z pole. Pro každý soubor pouští proces importu diagramu, a teprve když skončí první, spustí se druhý. Daný iterátor se poté pouští pomocí „for await (const value of objectWithIterator)“.

Proces importu. Obrázek 5.9 zachycuje celý proces importu. Na začátku importu jednotlivých diagramů je nutné vzít data souboru a převést je do textového řetězce. Jedná se o asynchronní operace, na níž je třeba čekat. Z toho důvodu je celý mechanismus čtení dat ze souboru zapouzdřen do javasriptové Promise.

[14] Jednotlivé akce na sebe navazují.
[15] Jedná se o návrhový vzor pro průchod daty.[58]
[16] Jednotlivé akce se vykonávají bez ohledu ostatních.
[17] Typ proměnné, který může hodnota v Javascriptu nabývat.[59]

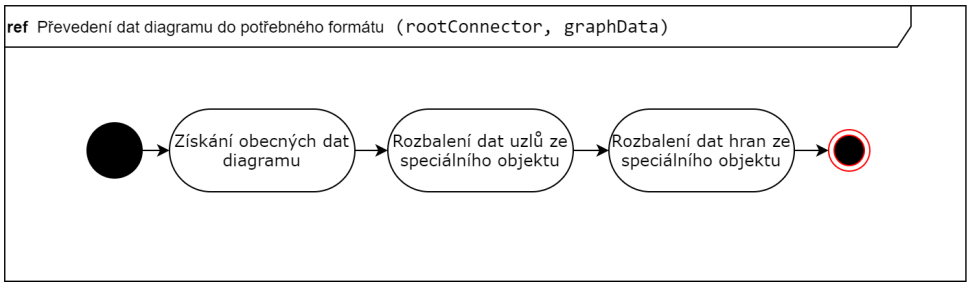


Obrázek 5.9: Aktivity diagram importu diagramu

Když jsou data souboru převedena, proces se rozvětvuje podle toho, o jaký typ souboru se jedná. Jde-li json formát, jsou data z textového řetězce převedena do jsonového objektu a kontroluje se, zda daný diagram obsahuje uzly a hrany. Pokud ne, jsou do objektu doplněna prázdná pole, aby násle-

dující proces nespadl na nedefinovaných atributech objektu. Pole elementů jsou procházena pomocí metody „forEach“[60], která v případě nedefinované proměnné, přes kterou iteruje, vrací chybu.

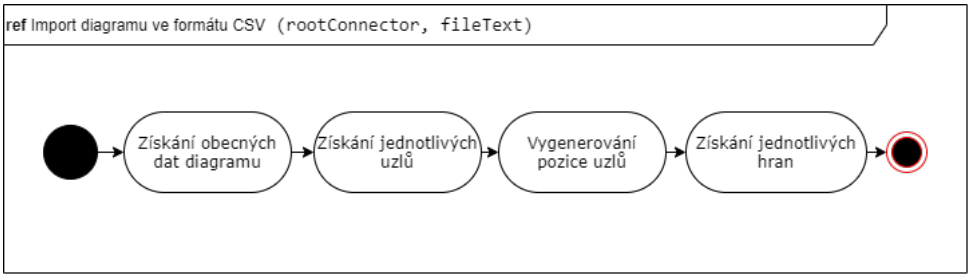
Jedná-li se o csv formát, je potřeba rozdělit řetězec na řádky a poté ještě na buňky, aby vzniklo dvojrozměrné pole. Dále je třeba vytvořit objekt, do kterého budou data z pole převedena. Nejprve se zadefinuje název a typ diagramu. Poté jsou do objektu přidány uzly, u kterých je nutné zadefinovat provizorní id a zvolit pozici, protože ta není v diagramu přenášeném ve formátu csv určena. Pro tento účel slouží jednoduchý mechanismus, v němž je první uzel vložen na souřadnice [500, 500] a další dva uzly se vkládají na pozice +100 v případě ylonových souřadnic a +/-100 u souřadnic xsových. Další dva uzly se tedy budou nacházet na souřadnicích [400, 600] a [500, 600]. Následně jsou do objektu vloženy hrany, které již mají definovaný název, a na základě matice sousednosti je určeno, odkud kam hrana vede. Poté se vytvořený javascriptový objekt předává dál pro vložení do platformy.



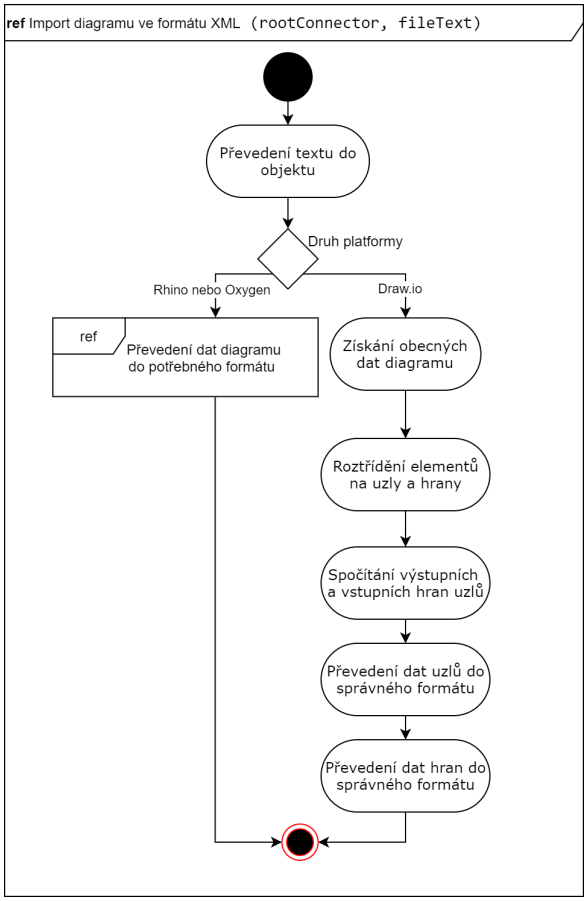
Obrázek 5.10: Aktivity podproces pro dat diagramu do správného formátu

Dalším možným formátem je xml. Pro možnost práce s daty diagramu je nejprve potřeba převést data z textového řetězce ve formátu xml do javascriptového objektu. K tomu je opět využita knihovna Xml-js. Dále se proces rozděluje podle toho, zda se jedná o platformu Rhino a Oxygen nebo Draw.io. V případě Rhina a Oxygenu je situace jednodušší, neboť jsou data v již známém formátu. Proto stačí pouze ze získaného objektu převést data do objektu nového, přičemž objekt „_attributes“ využívaný knihovnou pro oddělení atributů uzlů od vnořených elementů, bude odebrán a nahrazen daty zevnitř objektu. Rovněž je potřeba kontrolovat, zda existují uzly a hrany – v opačném případě je třeba vložit prázdné pole. Objekt se následně podává dál pro vložení do platformy.

V případě importu xml souboru z platformy Draw.io je třeba se nejdříve skrz získaný javascriptový objekt dostat ke kořeni diagramu. Ten je zanořen, protože soubor obsahuje spoustu doplňkových atributů používaných v platformě. Diagram může mít vlastní název, ale pro větší přehlednost je využit shodný název s názvem souboru. Typ diagramu není možné na základě struktury dat s jistotou poznat, proto se zde využívají informace vložené uživatelem. Protože jsou všechny elementy vloženy do jednoho pole, je třeba dané elementy roztřídit na hrany, uzly a zbytek, který je nepoužit. Hrany jsou identifikovány pomocí atributu „edge“ a uzly jsou identifikovány pomocí atributu „parent“, který musí mít hodnotu „1“.



Obrázek 5.11: Aktivity podproces pro zpracování souboru ve formátu csv



Obrázek 5.12: Aktivity podproces pro zpracování pro souboru ve formátu xml

Z důvodů, které jsou sepsány na začátku této kapitoly, bylo potřeba vytvořit způsob, jímž se budou identifikovat jednotlivé typy uzlů aktivity diagramů:

- uzel do kterého nevede žádná hrana - počáteční uzel
- uzel z kterého nevede žádná hrana - koncové uzel
- uzel do kterého vede alespoň jedna hrana a vychází alespoň dvě - rozhodující uzel
- ostatní uzly - aktivity uzel

Uzly orientovaných grafů se budou identifikovat pomocí pravidel:

- uzel do kterého nevede žádná hrana - počáteční uzel
- ostatní uzly - obyčejné uzly

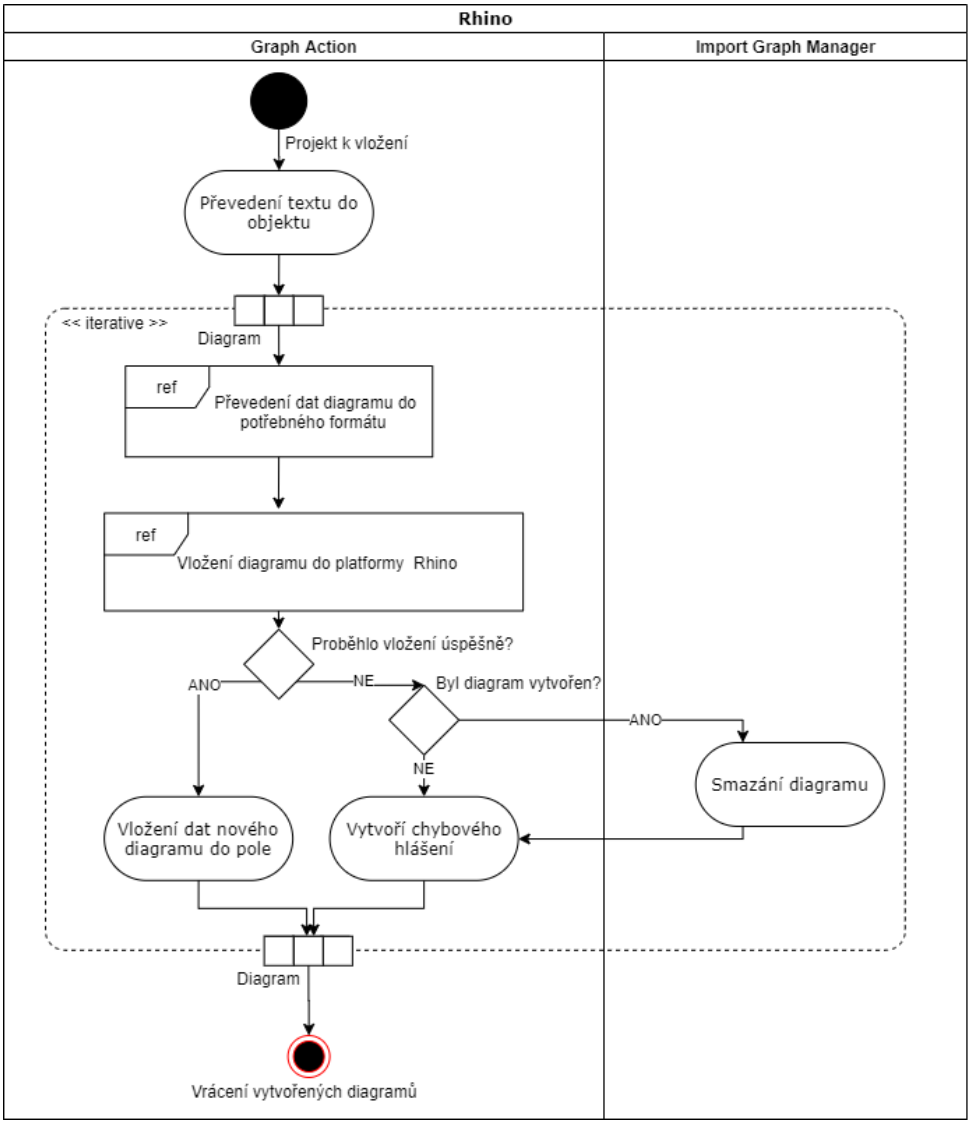
Proto je třeba v dalším kroku pro každý uzel sečíst počet výstupních a vstupních hran. Důležité je také, aby při kreslení diagramu v Draw.io uživatel dbal na propojení hran s uzly. Poté lze do nového javascriptového objektu, který je v požadovaném formátu, vložit jednotlivé uzly a hrany a předat objekt dál pro vložení do platformy. Při vkládání elementů je důležité ještě zkontrolovat, zda mají jednotlivé elementy definovaný název. Platforma vyžaduje unikátní názvy elementů, a proto v případě absence názvu je nezbytné ho doplnit. Využívá k tomu již naimplementovaný generátor. Název první chybějící hrany bude „1“, a další název bude vždy o jednu větší. Jen místo čísla „10“ bude pokračovat „00“. Pro uzly je zvolen odlišný způsob, používá písmena. Prvnímu uzlu je přidělen znak „a“, dalším uzlům budou přidány znaky podle toho jak jdou po sobě v abecedě. V případě využití celé abecedy se bude pokračovat znakem „aa“.

Posledním typem podporovaného formátu souboru je prj. Tento formát obsahuje celý projekt z Oxygenu. Může tedy obsahovat více než jen jeden diagram a obsahuje více informací než pouze diagramy. Data jsou uložena ve formátu xml. Proto je třeba spustit proces vkládání jednotlivých diagramů postupně, stejně jako na začátku celého procesu pro jednotlivé soubory pomocí asynchronního iterátoru. Lze využít již naimplementovaný způsob pro import diagramů ve formátu xml. Dále se pro každý diagram zvlášť dál předávají data pro vložení do platformy. Diagramy je nutné vkládat postupně.

Po rozdělení procesu podle jednotlivých typů souborů se proces opět sjednocuje v podprocesu Vložení diagramu do platformy Rhino až na část, která se věnuje souborům ve formátu prj. Tento podproces se děje uvnitř podprocesu Import diagramu ve formátu PRJ. Zde se vytvořený javascriptový objekt, který má pro všechny formáty stejný vzhled, a který je uzpůsobený pro jednodušší vkládání do platformy. Z důvodu, že zde probíhá též vkládání do databáze, je nutné, aby tato metoda byla asynchronní. Celý podproces je znázorněn na obrázku 5.14.

Zde se nacházela možnost dvojí implementace. První možností bylo vytvořit jeden nový endpoint pro vložení javascriptového objektu na server, který by ho zpracoval a uložil. Potom by se do klientské sekce vrátil již uložený diagram, a pak by se pouze aktualizoval stav aplikace. To by znamenalo rozšíření funkcionality serveru. Druhou (aplikovanou) možností je použít již vzniklé endpointy pro tvorbu a aktualizaci diagramů. Je nutné zahrnout oba, protože při použití endpointu pro aktualizaci diagramu je potřeba id aktualizujícího se diagramu spárovat s id již uloženého diagramu v databázi. Toto id však prozatím není známé – proto je proces rozdělen do několika fází.

V první fázi je potřeba získat id aktuálního projektu a sestavit strukturu diagramu podle toho, jak bude odeslána na server. Dále je nutné ošetřit, aby nedošlo ke kolizi názvů diagramů. Proto je třeba získat ze stavu aplikace seznam všech aktuálně vytvořených diagramů. Pokud již nějaký z vytvořených

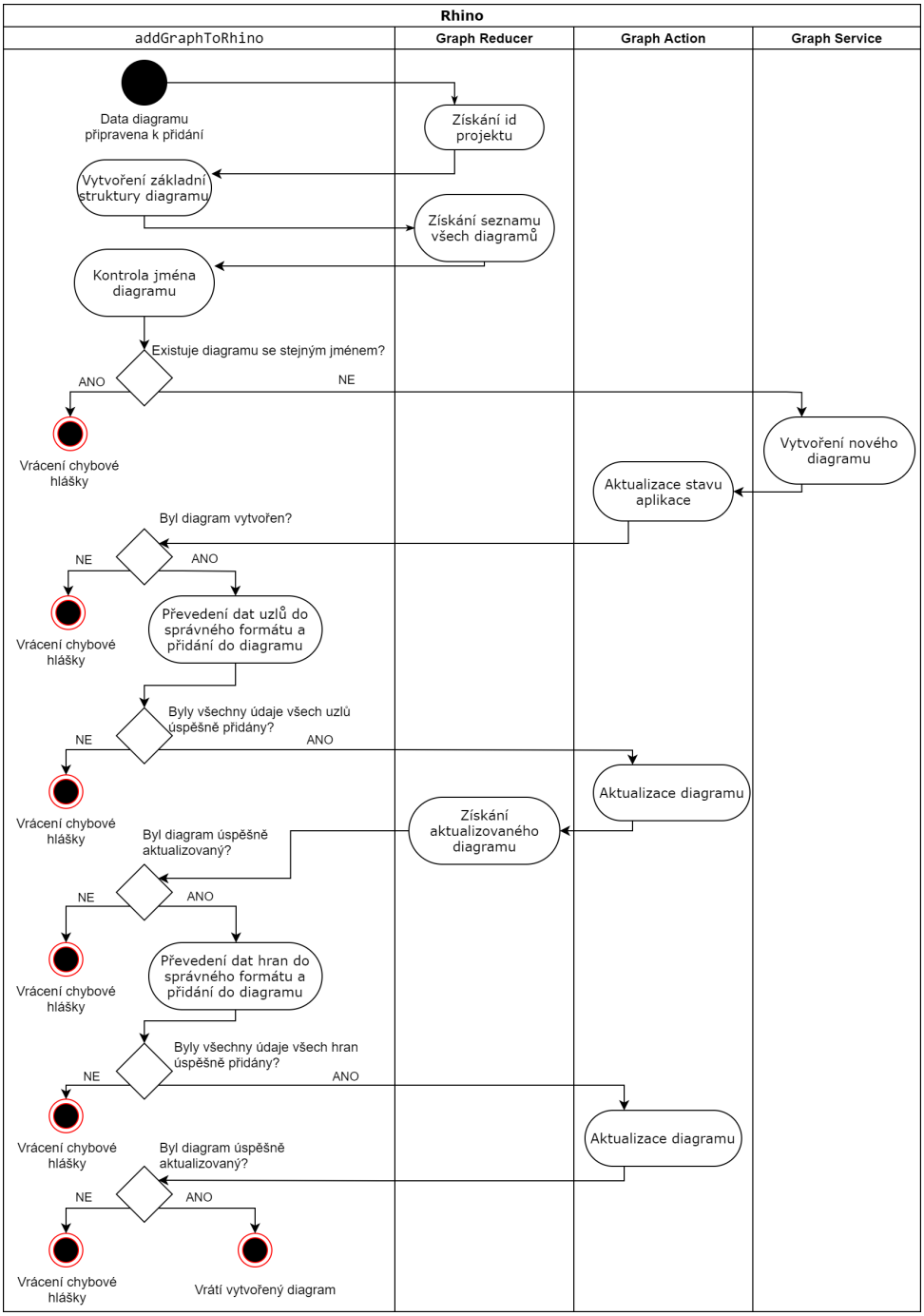


Obrázek 5.13: Aktivity podproces pro zpracování souboru ve formátu prj

diagramů má stejný název, podproces končí chybovou hláškou. V opačném případě je poslán název a typ diagramu na server a čeká se na vygenerované id. Následně je stav aplikace aktualizován – existuje tedy nový diagram v seznamu diagramů.[18] Pokud vytvoření diagramu proběhlo v pořádku, jsou do vytvořené struktury diagramu přidány uzly. Na základě překladače je nutné převést priority do správného formátu, určit počáteční uzel na základě typu uzlů, v případě aktivity diagramů určit typ uzlů a zbytek atributů schovat do objektu „restAttributes“.

Pokud všechny převody proběhnou úspěšně, je nový diagram poslán na server, aby aktualizoval svůj stav. V opačném případě podproces končí a je

[18] Právě kvůli této části je nutné importovat diagramy synchronně, aby se nestalo, že budou obsahovat v databázi dva diagramy stejný název.



Obrázek 5.14: Aktivity podproces vložení diagramu

nutné nově vytvořený diagram smazat z databáze. Stejně tak se stane, pokud během aktualizace diagramu dojde k chybě na serveru nebo následné aktualizaci stavu aplikace. Je nutné aktualizovat diagram před vložením hran do struktury, protože díky tomu jsou na serveru vygenerovány id, jež hrany potřebují do svých atributů. Pokud nedojde k chybě, je z aktuálního stavu

diagramu získán nový diagram. Dále jsou do diagramu vloženy hrany, které potřebují do svých atributů jak id, tak název uzlu, z kterého vychází, tak i název uzlu do kterého vedou. Podobně jako u uzlů je třeba převést prioritu do správného formátu dle překladače, a zbytek atributů je zabalen do objektu „restAttributes“. Poté je diagram znovu odeslán na server, aby se aktualizovala databáze. Po úspěšné aktualizaci databáze se aktualizuje i stav aplikace. Následně se jen vytvořená struktura diagramu vloží do pole všech úspěšně vytvořených diagramů. V případě chyby při aktualizaci databáze nebo aplikace proces končí, ale je ještě nutné smazat nově vytvořený diagram bez hran. Nakonec se jen všechny úspěšně importované diagramy zobrazí v konzoli prohlížeče, a jako aktivní diagram se nastaví poslední importovaný.

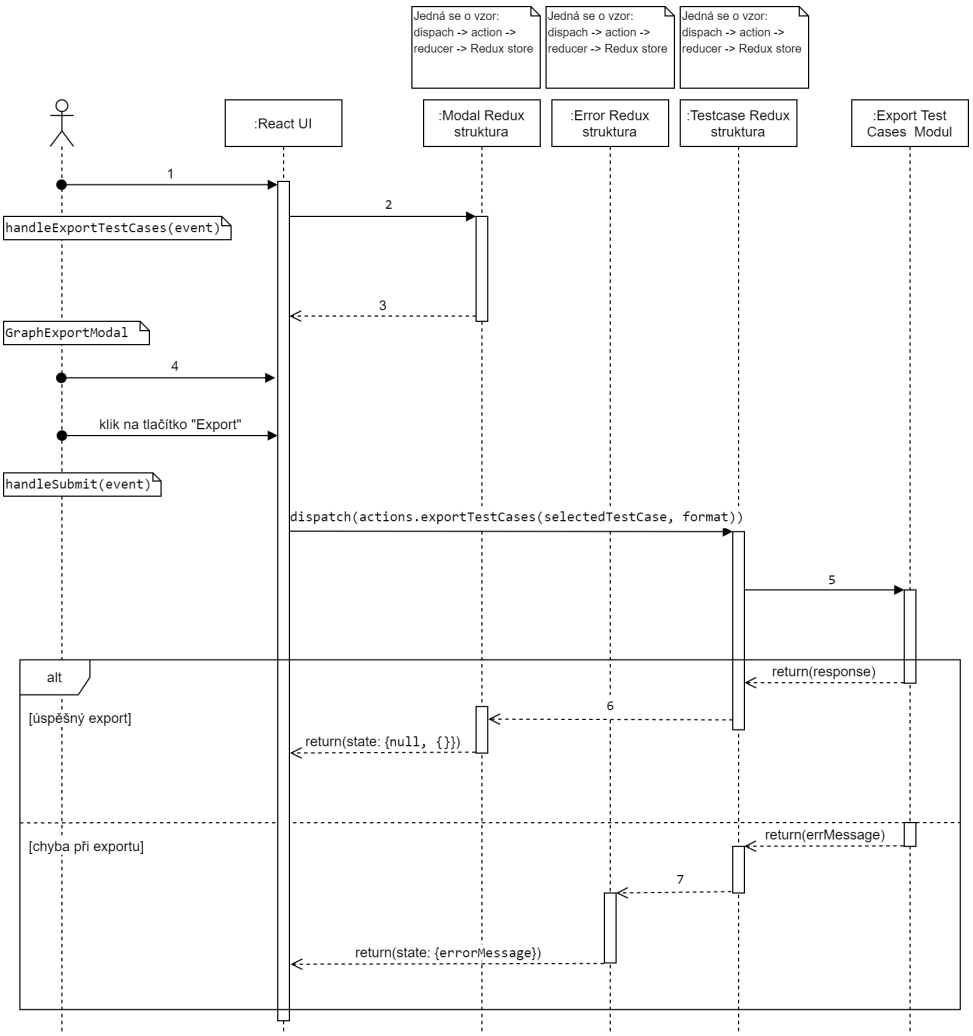
5.3.2 Testovací scénáře

Proces exportu testovacích scénářů není nijak limitovaný v podobě formátu z pohledu následující potřeby importu do jiného nástroje. Z pohledu platformy Rhino se jedná o konec procesu. I tak je zvolen stejný formát jako pro export testovacích scénářů z Oxygenu, pro zachování jednotnosti. Tento formát obsahuje seznam všech scénářů, které jsou uloženy jako jeden textový řetězec skládající se z názvů hran oddělených pomlčkami.

Čísla	Popis
1	klik na ikonu export testovacích scénářů
2	dispatch(modalActions.showModal({modalType: modalTypes.TEST_CASES_EXPORT, modalProps: {selectedTestCase}}))
3	return(state: {modalType, modalProps})
4	Výběr typu výstupního formátu souboru
5	exportManagerTestCases(resolve, reject, graph, selectedTestCase, format)
6	dispatch(modalActions.hideModal)
7	dispatch(error(EXPORT_TEST_CASES { errMessage }))

Tabulka 5.3: Vysvětlivky k sekvenčnímu diagramu pro export testovacích scénářů 5.15

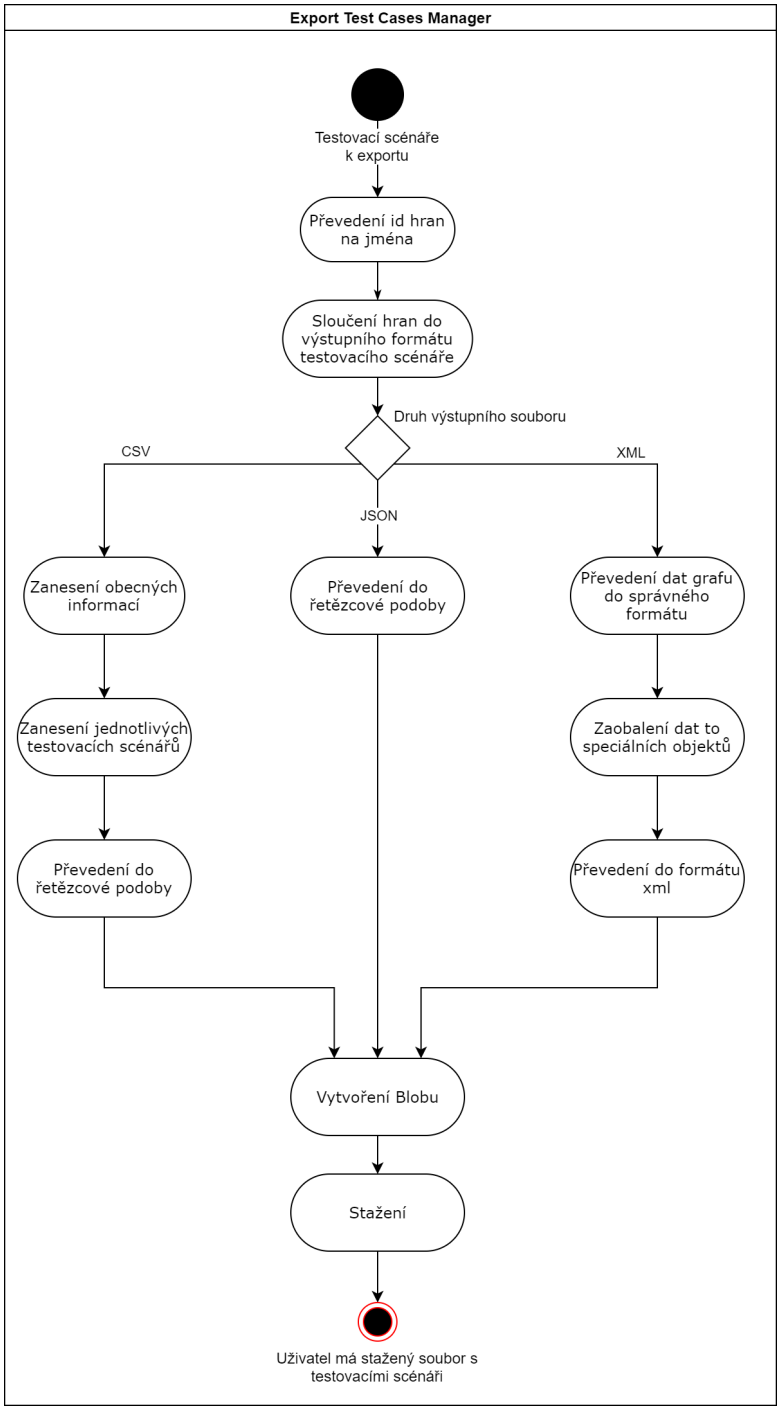
Napojení procesu na stávající aplikaci je vidět na sekvenčním diagramu, který je znázorněn na obrázku 5.15. Po kliknutí na ikonu exportu testovacích scénářů se nejdříve zobrazí modální okno, kde je třeba zvolit typ exportního formátu. Poté je požadavek poslán do příslušné reduxové aktivity testovacích scénářů. Zde je požadavek zpracován asynchroně, po vzoru všech ostatních reduxových aktivit. Logika převodu dat do definovaného formátu a samotný export se vykonává ve speciálních funkcích. Ve chvíli, kdy je soubor úspěšně exportován, dochází k uzavření modálního okna a uživatel může v platformě pracovat dále. Pokud dojde k chybě, v modálním okně se objeví chyba a uživatel jej uzavře sám pomocí tlačítka „Cancel,“ nebo se může pokusit



Obrázek 5.15: Sekvenční diagram exportu testovacích scénářů

exportovat diagram znovu.

Proces průběhu samotného exportu testovacích scénářů je zobrazen v diagramu na obrázku 5.16. Na začátku je potřeba převést data do správného formátu, protože z aplikace je předáno pole s id jednotlivých hran testovacího scénáře. Id jsou generována serverem a pro uživatele nemají žádnou vypovídající hodnotu, proto je z aplikace též potřeba předat diagram, kterému náleží dané testovací scénáře. Na základě hran diagramu jsou poté podle id těchto hran získávána jejich jména. Všechny potřebné informace se následně vloží do javascriptového objektu, který se pak uloží ve formátu dle volby uživatele. Pro export do xml je potřeba opět zabalit atributy uzlů do speciálních objektů „_attributes“, protože je využito knihovny Js-xml. Formát csv je rozdělen do dvou sekcí. Jedna se nachází na začátku a jsou zde zadefinovány základní informace o testovacích scénářích. Druhá sekce je oddělena prázdným řádkem a nachází se zde výčet jednotlivých testovacích scénářů.



Obrázek 5.16: Aktivity diagram exportu testovacích scénářů

Kapitola 6
Ověření kvality

V této kapitole se vracím k průběhu testování naimplementovaného aktivity modulu a zbytku naimplementované funkcionality. V první části se věnuji testování funkčnosti interních funkcí zajišťujících jednotlivé části importu a exportu diagramů a exportu testovacích scénářů. V druhé části popisuji testování dodržení požadavků této práce pomocí případů užití[1] a jejich testů.

6.1 Jednotkové testy

Jedná se o nejnižší úroveň testování softwaru, kde se testují samostatné jednotky nebo nejmenší komponenty zvlášť. Účelem je ověřit, zda každá jednotka pracuje právě tak, jak byla navržena a jestli při její práci nevznikají chyby. Každý test kontroluje pouze jednu jednotku a pouze jeden možný scénář. Test mívá pouze pár vstupů a jeden výstup. Samotné testování zpravidla se skládá ze tří částí. První část zahrnuje definované vstupy, ve druhé je volána testovaná jednotka a nakonec ve třetí části dochází ke kontrole, zda výsledek odpovídá očekávání.[9]

Pro otestování platformy pomocí jednotkových testů bylo využito frameworku Jest[62], který je integrován v balíčku Create React App[2]. Jedná se o javascriptový framework se zaměřením na jednoduchost a rychlost. Využívá ho přes 1 200 000 veřejných GitHub[3] projektů a mnoho známých společností např. Facebook či Airbnb. Při práci s Create React App není třeba náročných konfigurací. V základu podporuje code coverage[4] a mockování[5], a navíc ji lze rozšířit o další funkcionalitu.

Pro jednotkové testy jsem zvolil jednotný způsob pojmenování, který je důležité pro rychlé zorientování se ve velkém množství testů a snadnou identifikaci místa chyby. Zde je použit způsob „MethodName_StateUnderTest-_ExpectedBehavior“[66]. Udávám tento název způsobu v původním znění,

[1] Je to jeden ze skupiny UML diagramů chování. Ukazují soubor akcí (případy použití), které by některý systém nebo systémy (subjekt) měly nebo mohou provádět ve spolupráci s jedním nebo více externími uživateli systému (aktéry).[61]

[2] Balíček využíván pro tvorbu React projektů[63]

[3] Platforma pro ukládání repositářů a prostor pro komunikaci mezi vývojáři.[64]

[4] Procentuální množství otestovaných funkcí kódu.[9]

[5] Vytváření umělých datových struktur a dalších komponent, pro testování[65]

protože jsou klíčové i pozice velkých a malých písmen. Na prvním místě se udává název testované metody, poté se udává stav, který se testuje, a nakonec očekávaný výsledek.

Vzhledem k architektuře nově implementovaných částí platformy bylo nutné otestovat funkce zařizující import a export diagramů a export testovacích scénářů. Otestování vytváření UML aktivity diagramů nebo práci s metadaty by byla komplikované. Jednotlivé funkce, které realizují tyto funkcionalitu, jsou hluboko zapouzdřeny do editoru diagramu, nebo jsou úzce spjaty s knihovnou Cytoscape a bylo by zde potřeba velkých mocků. Proto je na tyto části kladen hlavní důraz v druhé fázi testování, a to pomocí uživatelských testů.

Pomocí jednotkových testů jsou otestovány všechny funkce, které jsou vyčleněny z procesu importu a exportu do složky „src/transferModule/helpers.js“. Zde se nachází funkce, které jsou používány napříč jednotlivými procesy importu a exportů a které hrají klíčovou roli pro správný chod. Překládají jednotlivá data z formátu do formátu nebo vyhledávají ve specifických polích elementů potřebná data pro další zpracování. Pro celkem 12 funkcí bylo vytvořeno 30 jednotkových testů, které testují jak pozitivní, tak negativní scénáře jednotlivých funkcí.

Dále byla otestována funkčnost implementace interní chyby, která je využívána v procesu importu diagramů. Využívá k předává informaci o id již vytvořeného diagramu. Co je následně rovněž využito pro poslání požadavku na smazání daného diagramu. Pro tuto chybu byly napsány celkem 3 testy související s možnostmi volání dané chyby.

Při psaní testů jsem narazil na nekonzistentnosti jednotlivých chybových hlášení funkcí a neošetřené případy vstupů - chyby jsem napravil. Tyto chyby se však ve výsledku daly označit za přínosné pro celé procesy importu a exportů. Upravení dílčích funkcí vedlo k větší čitelnosti kódu a jednotnosti volání dílčích funkcí.

6.2 Uživatelské testy

Ověření, zda byly splněny požadavky implementace, je téma nejvyšší úrovně testování. Pro tento účel byly zvoleny uživatelské testy. Je to technika, při které se testuje platforma fyzicky. Druhů uživatelských testování může být celá řada. Od ponechání testovateli úplné svobody v platformě a stopování jeho průchodu aplikací, až po striktní dodržování scénáře průchodu aplikací pro otestování konkrétní funkcionality nebo části aplikace.[65] Zde je zvolen způsob dle TMAP Next[6], který je definován v rámci metodiky pro testování případů užití.

Tato metoda staví na případech užití, protože v sobě obsahují interakci mezi uživatelem a systémem. Případy užití popisují úplnou funkčnost, kterou systém nabízí a která přináší uživateli pozorovatelný výsledek. Proto je pro metodiku velmi důležitý diagram případů užití.[18]

[6]TMAP je soubor znalostí společnosti Sogeti pro kvalitní inženýrství v oblasti IT a staví na praktických zkušenostech od tisíců lidí od roku 1995.[18]

V metodice je uvedený pomocný kontrolní seznam, který má pomoci autorovi uživatelských testů. Je důležité, aby jednotlivé případy užití byly dostatečně detailní. Zároveň je nutné, aby bylo možné pro každý případ užití vytvořit alespoň jeden logický uživatelský test. Pokud jsou případy užití příliš obecné nebo naopak příliš konkrétní, může se stát, že vytvoření testovacích scénářů nebude možné. Tato situace může nastat buď z nedostatku informací nebo z nemožnosti vytrhnout daný případ užití z kontextu ostatních případů užití[18].

Pro sestavení jednotlivých testovacích scénářů pro jednotlivé případy užití se v metodice využívá scénářů případů užití. Je zde znázorněno, jak lze rozčlenit daný případ užití do jednotlivých akcí, jaké jsou podmínky pro realizaci a výsledek případu užití, negativní scénáře a jejich podmínky vzniku a proměnné s kterými případ užití operuje.[18]

Případ užití	**Přidat metadata**
Testovací scénář ID	**1**
Účel	Zkontrolovat, že lze přepnout v pravém panelu metadat do režimu přidávání atributů
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram alespoň s jedním uzlem. Má rozkluknuté stávající metadata uzlu.
Aktivita	Kliknutí na tlačítko "Add attribute"
Výsledek	Zobrazí se další políčko, kde bude vyjížděcí menu a místo tlačítka "Add attribute" se objeví tlačítko "Cancel" a tlačítko "Save"
Testovací scénář ID	**2**
Účel	Zkontrolovat, že je možné zobrazit metadata z kterých je možné vybrat, které přidat.
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram alespoň s jedním uzlem. Má rozkliknuté stávající metadata uzlu a klikl na tlačítko "Add attribute"
Aktivita	Kliknutí na vyjížděcí menu v novém políčku.
Výsledek	Zobrazí se možné atributy na přidání

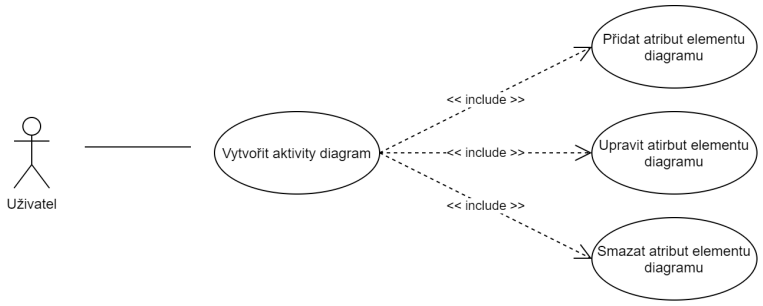
Obrázek 6.1: Ukázka testovacích scénářů pro případ užití „Přidat metadata“

Na obrázku 6.1 je vidět příklad dvou testovacích scénářů pro případ užití „Přidat metadata“. Struktura je definována v TMAP Next a pro účely této práce je přeložena do češtiny. Jednotlivé testovací scénáře představují dílčí kroky, které je důležité zkontrolovat zvlášť. To připadá v úvahu ve chvíli, kdy daný případ užití obsahuje testovacích scénářů více, kde scénáře jsou napsány v návaznosti za sebou. V testovacích scénářích rozděluji prioritu na nízkou, střední a vysokou. Hodnotu priority určuje důležitost dané funkce pro fungování celého procesu jako celku.[18]

Z pohledu požadavků a povahy jednotlivých částí implementace jsem rozdělil uživatelské testování do tří celků. První se zabývá aktivity diagramem a metadaty uzlů, druhý se zaměřuje na import diagramů a třetí na export diagramů a testovacích scénářů.

6.2.1 Aktivity modul a metadata elementů

Tato část je klíčovou částí celé práce s důrazem na funkčnost kreslení aktivity diagramů. Zároveň se pracuje s metadaty elementů, protože jsou úzce spjata s diagramy.[7]



Obrázek 6.2: Diagram případů užití pro vytvoření UML aktivity dagramu a práce s metadaty

Návrh jednotkových testů

Na obrázku 6.2 je vidět diagram případů užití pro aktivity diagram vztah k operacím s metadaty. Tento diagram by mohl být mnohem větší, protože by bylo možné rozdělit případ užití „Vytvořit aktivity diagram“ na mnoho menších. Tento případ užití v sobě obsahuje mnoho funkcionalit spojených s tvorbou jednotlivých uzlů a prací s nimi. Formát diagramu je zvolen za účelem zobrazení vztahu aktivity diagramu a jeho metadaty.

Diagram na obrázku 6.2 nezobrazuje některé další případy užití, které by zde rovněž mohly být uvedeny, avšak nejsou primárně testovány. Mezi tyto případy užití patří: Zobrazení aktivity diagramu, smazání aktivity diagramu, uložení aktivity diagramu nebo přejmenování aktivity diagramu. Je to z toho důvodu, že tyto funkcionality platformy zůstaly zcela nezměněny a zároveň platforma byla funkční již ve chvíli, kdy jsem na ní začal pracovat. Některé případy užití jsou navíc testovány pomocí jiných uživatelských testů. Např. uložení aktivity diagramu se testuje pomocí funkčnosti exportu diagramu a zobrazení aktivity diagramů se testuje pomocí importu diagramů.

Testování metadat je také o něco zjednodušeno. Není zde zahrnuto testování práce se základními atributy, a to s názvem a prioritou elementu. Tyto operace byly již naimplementované. Testování je omezeno na práci s metadaty elementů pro aktivity diagramy. Důvodem je totožnost práce s atributy. Pro oba diagramy zajišťuje práci s metadaty elementů totožná komponenta. Jediný rozdíl v implementaci metadat elementů je základní atribut „type“, který pro orientované grafy zůstává „null“ a pro aktivity diagramy nabývá hodnot dle typu uzlu.

[7] Všechny testovací scénáře použity pro uživatelské testování se nacházejí v příloze A

Implementace jednotkových testů

Pro vytvoření testovacích scénářů nebylo třeba scénářů případů užití. Využil jsem již dříve vypracovanou analýzu kódu a wireframů. Scénáře by byly výše zmiňované duplicitním.

Výsledky testování

Testování vytváření aktivity diagramů a práce s metadaty poukázaly na několik nedostatků, jež se v aplikaci nacházelo už od začátku. Je možné, že se nejednalo o záměr, ale v tuto chvíli se projevily jako nedostatky. Stalo se tak především proto, že nebyly v testech brány předem v potaz.

Čtvrtý testovací scénář vytváření aktivity diagramů ukázal, že po vytvoření diagramu se nový diagram nezobrazil v editoru, ale pouze v levém panelu. Pro otevření editoru bylo nutné kliknout na název vytvořeného diagramu. V 9. testovacím scénáři občas vznikl problém s vytvořením hrany. Po kliknutí na uzel a následném tažení se nevytvářela hrana, ale uzel se začal hýbat. Překvapivý výsledek přinesl 14. testovací scénář vytváření aktivity diagramů. V případě, kdy uživatel smazal počáteční uzel, se jako počáteční nastavil první vytvořený uzel, který počáteční nebyl. V případě ostatních uzlů se však toto nedělo. Poslední neočekávaný výsledek přinesl 5. testovací scénář pro přidávání atributu po kliknutí na tlačítko „Save“. Možnost přidávání nových atributů se nezavírala a hodnota, která byla přidána nově vznikajícímu atributu, zůstala předvyplněna pro přidávání nového atributu. Z toho vyplývá, žejediná možnost, jak ukončit přidávání nových atributů, bylo kliknutím na tlačítko „Cancel,“ popř. kliknutí kamkoliv do editoru diagramu.

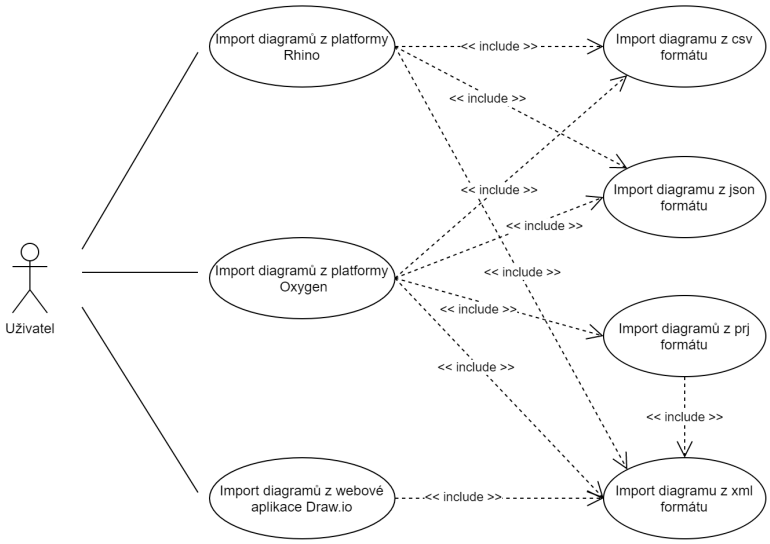
6.2.2 Import diagramů

Ze zbytku testované funkcionality je import diagramů vyčleněn z exportů kvůli komplikovanosti a zároveň velkému množství procesů, v nichž lze udělat chyba.

Návrh jednotkových testů

Obrázek 6.3 znázorňuje diagram případů užití pro import diagramů. Diagram bylo možné vytvořit více způsoby. Použitý způsob klade důraz na rozdílnost jednotlivých platforem a zároveň ukazuje shodnost v podobě formátů importovaných souborů. Stejný diagram bylo možné vytvořit s důrazem na rozdílnost jednotlivých způsobů importu různých formátů souborů pro jednotlivé platformy. Bylo také možné vytvořit celkem osm případů užití pro každý podporovaný formát souboru a konkrétní platformu zvlášť.

Z pohledu uživatelského rozhraní byl vybrán jako demonstrativní diagram ten, který byl vytvořen podle prvního způsobu. Protože uživatel po vložení jakéhokoliv podporovaného souboru udává pouze platformu a s typem souboru, pracuje platforma samotná bez vědomí uživatele.



Obrázek 6.3: Diagram případů užití pro import UML aktivity dagramů - formátován v MS Excel

Uživatelské testy této části jsem omezil na testování čtyř případů užití, protože mezi importem z platformy obecně a importem jednotlivých formátů je velmi úzké spojení. V diagramu na obrázku 6.3 testuji pouze případy užití importující konkrétní soubory. V těchto testech zohledňuji jednotlivé platformy zvlášť.

Implementace jednotkových testů

Jednotlivé testovací scénáře byly navrženy podle wireframu na obrázku 4.12 a aktivity digramů v sekci implementace importu diagramů 5.3.1. Tam je znázorněn celý průběh procesu a zároveň všechny možné vstupy a konce procesu. Je nutné jednotlivé testy provést s aktivity diagramy i s orientovanými grafy.

Výsledky testování

Výsledky testování importu diagramů byly úspěšné pro všechny testy jednotlivých případů užití. Byly otestovány aktivity diagramy i orientované grafy. Tyto testy testují pozitivní průchod procesem a všechny možnosti, které jsou platformou podporované.

Dále byly vytvořeny testy pro kontrolu zpětné vazby v případě chybných vstupů nebo poškozených dat diagramů v souboru. Celkově tyto testy nedopadly příliš dobře. V žádném z nich sice chybná data nezpůsobila pád aplikace, ale zároveň neposkytla uživateli zpětnou vazbu, jež byla očekávána. První test se tedy stal jediným úspěšným testem.

Ve snaze importovat znovu tentýž soubor do platformy aplikace zamrzne a nezobrazí se žádná chyba. Je nutné obnovit stránku, a poté zmizí modální okno a je možné dále bez problémů pracovat. Čtvrtý test ukázal, že při

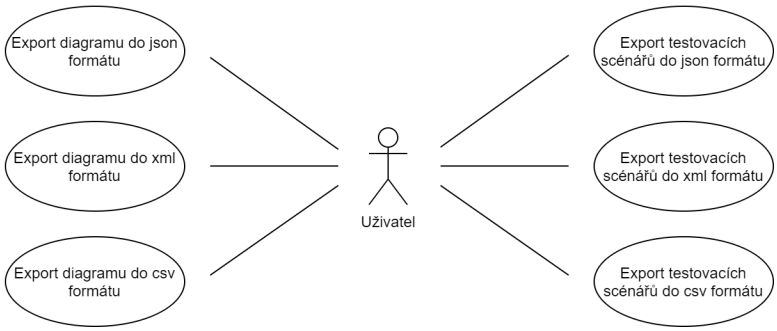
chybné definici stylu uzlu v souboru se diagram i přesto importuje, avšak daný uzel má nedefinovaný typ. Je vizualizován šedým kolečkem s černým okrajem. V případě špatného id uzlu se diagram neimportuje a modální okno se zavře. To samé se stane v případě špatné definice hrany v pátém testu. Zajímavý výsledek přinesl šestý test, kde se ukázalo, že závisí na pořadí vložení do aplikace poškozeného a nepoškozeného souboru. Když je první soubor poškozený, tak se nenaimportuje soubor žádný, a pokud se soubory vloží v opačném pořadí, naimportuje se pouze ten nepoškozený, což se však nezobrazí v editoru.

6.2.3 Export diagramů a testovacích scénářů

Poslední část obsahuje testování exportu dat z platformy Rhino.

Návrh jednotkových testů

Diagramy a testovací scénáře lze exportovat ve třech různých formátech. Všechny možnosti jsou znázorněny v diagramu případů užití na obrázku 6.4.



Obrázek 6.4: Diagram případů užití pro exportt UML aktivity dagramů a testovacích scénářů

Implementace jednotkových testů

Jednotlivé testovací scénáře pro případy užití jsou vytvořeny na základě wireframů modálních oken k exportu diagramů a testovacích scénářů a k nim vytvořeným aktivity diagramům v kapitole Implementace 5.3. Stejně jako pro proces importu diagramů je třeba stejné testy provést ještě jednou pro orientované grafy.

Výsledky testování

Pro export diagramů se ukázalo, že vše podle testů funguje bez problému. Nejsou zde definovány žádné chybové testy, protože pokud nedojde k nečekané chybě na serveru nebo uvnitř stavu aplikace, uživatel by neměl mít možnost chybu vytvořit.

Testy pro export testovacích scénářů ukázaly drobný problém v definici druhého testu pro každý exportní formát. Je zde vyžadována kontrola automaticky vyplněného názvu exportního souboru, který má být shodný s názvem testovacího scénáře zobrazeného pod tlačítkem „Show TDL“. Výčet testovací scénářů v prostoru pod tlačítkem ovšem není zobrazen, pokud na tlačítko uživatel dříve nekliknul. Druhý problém, který se zobrazil, je mnohem závažnější. Souvisí s nekonzistentností v aktuálnosti stavu aplikace a stavu databáze. V případě, že v aplikaci není záznam o novém diagramu, dochází ke spadnutí platformy. Pokud je v aplikaci uložen neaktuální stav diagramu, soubor obsahuje prázdná místa pro názvy hran, která nejsou v aplikaci uvedena, avšak jsou již uložena v databázi.

Kapitola 7

Závěr

V této bakalářské práci se mi podařilo rozšířit možnosti webové platformy Rhino o kreslení UML aktivity diagramů, přidávání a úpravu metadat elementů, import a export diagramů a export vygenerovaných testovacích scénářů. Všechny naimplementované funkce platformy je možné aplikovat jak pro aktivity diagramy, tak i pro orientované grafy.

Rhino je platformou, jenž je vyvíjena pomocí závěrečných prací studentů pod záštitou Fakulty elektrotechnické ČVUT. Tato práce je v pořadí druhá, která na této platformě pracuje. Další očekávaný vývoj platformy bude spočívat v podobě integrace dalších možností kreslení diagramů nebo rozšíření sady algoritmů pro generování testovacích scénářů. Protože je platforma vyvíjena s vědomím možného rozšíření, není vyloučeno přidání další funkcionality související s účely platformy, které v tuto chvíli zatím nejsou vymyšleny.

Pro mě osobně byla tvorba této bakalářské práci velmi přínosná. Vyzkoušel jsem si celý vývojový proces modulu pro webovou aplikaci: zpracování požadavků, návrh řešení, implementace, otestování zhotovených částí a veškerá jejich dokumentace. Výzvou pro mě bylo zejména vypracování jednotlivých diagramů a snaha udržet konzistentnost mezi jednotlivými diagramy, což při jejich velikosti nebylo vůbec jednoduché. Jsem rád, že jsem mohl uplatnit své zkušenosti s Reactem při vývoji klientské části aplikace a zároveň se naučit správu stavu aplikace pomocí knihovny Redux. Uživatelské testy mi ukázaly nový pohled na mnou odvedenou práci. Uvědomil jsem si, že některé věci nefungují přesně tak, jak bych očekával, což je velmi přínosné hlavně pro další práci na platformě. V poslední fázi jsem si osvojil práci v latexovém editoru Overleaf, který mi velmi usnadnil práci s celkovým formátováním textu práce.

Na základě uživatelských testů lze říci, že cíle této práce jsou splněny – tedy že všechny požadavky jsou úspěšně naimplementovány. Chyby, které byly odhaleny, ukazují na nedostatky při negativních scénářích pro import diagramů. Zároveň se však ukázalo, že chyby jsou interně ošetřeny a neovlivňují chod aplikace. Uživatel se pouze nedozví, kde nastala chyba. Vážnější problém se ukázal při generování testovacích scénářů, které nejsou uloženy ve stavu aplikace. Zde může hrozit i pád aplikace. Po obnovení aplikace se ovšem diagramy neztratí, neboť se ukládají správně. Bude tak dokonce možné úspěšně exportovat i testovací scénáře.

Do budoucna nebude problém všechny tyto chyby opravit. Upravení chování

při chybě importu diagramů je důležité nejdříve specifikovat. Na základě aktivity diagramů zmíněných v kapitole Implementace importu diagramů 5.3.1 lze v kódu vystopovat, v jakých místech chyby jsou ignorovány. Chyby při exportu testovacích scénářů bude možné dohledat v diagramech pro export testovacích scénářů a následně navrhnout implementační řešení v komunikaci s Redux diagramovou aktivitou „upgradeGraph“.

Literatura

[1] M. Bureš, M. Renda, M. Doležel *et al.*, *Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu.* Grada Publishing as, 2016.

[2] "React: Create maintainable, high-performance ui components," Feb 2015. [Online]. Available: <https://developer.ibm.com/technologies/javascript/tutorials/wa-react-intro/>

[3] "Introducing redux," Jul 2016. [Online]. Available: <https://developer.ibm.com/tutorials/wa-manage-state-with-redux-p1-david-geary/>

[4] R. Bakeyev, "Webový editor modelu procesů testovaného systému," Jun 2019. [Online]. Available: <https://dspace.cvut.cz/handle/10467/82852>

[5] M. Bures, "Experimental model-based testing environment," Mar 2018.

[6] J. Bang-Jensen and G. Z. Gutin, *Digraphs: theory, algorithms and applications.* Springer Science & Business Media, 2008.

[7] D.-J. De Grood, *Testgoal: Result-driven testing.* Springer Science & Business Media, 2008.

[8] B. W. Boehm, R. K. Mcclean, and D. Urfrig, "Some experience with automated aids to the design of large-scale reliable software," *IEEE Transactions on Software Engineering*, no. 1, pp. 125–133, 1975.

[9] P. Ammann and J. Offutt, *Introduction to software testing.* Cambridge University Press, 2016.

[10] K. Beck, *Test-driven development: by example.* Addison-Wesley Professional, 2003.

[11] J. Charvat, *Project management methodologies: selecting, implementing, and supporting methodologies and processes for projects.* John Wiley & Sons, 2003.

[12] M. Bures, "Pctgen: automated generation of test cases for application workflows," in *New Contributions in Information Systems and Technologies*. Springer, 2015, pp. 789–794.

[13] M. Bures and B. S. Ahmed, "Employment of multiple algorithms for optimal path-based test selection strategy," *Information and Software Technology*, vol. 114, pp. 21–36, 2019.

[14] M. Shirole and R. Kumar, "Uml behavioral model based test case generation: a survey," *ACM SIGSOFT Software Engineering Notes*, vol. 38, no. 4, pp. 1–13, 2013.

[15] K. Frajták, M. Bureš, and I. Jelínek, "Transformation of ifml schemas to automated tests," in *Proceedings of the 2015 Conference on research in adaptive and convergent systems*, 2015, pp. 509–511.

[16] N. Yousaf, F. Azam, W. H. Butt, M. W. Anwar, and M. Rashid, "Automated model-based test case generation for web user interfaces (wui) from interaction flow modeling language (ifml) models," *IEEE Access*, vol. 7, pp. 67 331–67 354, 2019.

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[18] T. Koomen, B. Broekman, L. van der Aalst, and M. Vroon, *TMap next: for result-driven testing*. Uitgeverij kleine Uil, 2013.

[19] M. Bures, T. Cerny, and M. Klima, "Prioritized process test: More efficiency in testing of business processes and workflows," in *International Conference on Information Science and Applications*. Springer, 2017, pp. 585–593.

[20] K. Fakhroutdinov, "Activity diagrams," Mar 2011. [Online]. Available: `<https://www.uml-diagrams.org/activity-diagrams.html>`

[21] "React bootstrap," Mar 2020. [Online]. Available: `<https://react-bootstrap.github.io/>`

[22] G. Booch, *The unified modeling language user guide*. Pearson Education India, 2005.

[23] B. A. C. . R. . M. Analyst, "Keeping high-level requirements high-level," Jun 2019. [Online]. Available: `<https://modernanalyst.com/Resources/Articles/tabid/115/ID/5384/Keeping-High-Level-Requirements-High-Level.aspx>`

[24] M. Franz, "Cytoscape.js," Feb 2020. [Online]. Available: `<https://js.cytoscape.org/#introduction>`

[25] K. Fakhroutdinov, "State machine diagrams," Apr 2011. [Online]. Available: `<https://www.uml-diagrams.org/state-machine-diagrams.html>`

[26] "Diagrams for everyone, everywhere," Apr 2020. [Online]. Available: <https://drawio-app.com/>

[27] E. W. Weisstein, "Adjacency matrix," May 2020. [Online]. Available: <https://mathworld.wolfram.com/AdjacencyMatrix.html>

[28] H. S. Oluwatosin, "Client-server model," *IOSRJ Comput. Eng*, vol. 16, no. 1, pp. 2278–8727, 2014.

[29] "About javascript," May 2020. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript>

[30] "About npm," Nov 2015. [Online]. Available: <https://docs.npmjs.com/about-npm/>

[31] M. Franz, C. T. Lopes, G. Huck, Y. Dong, O. Sumer, and G. D. Bader, "Cytoscape. js: a graph theory library for visualisation and analysis," *Bioinformatics*, vol. 32, no. 2, pp. 309–311, 2016.

[32] "Cytoscape.js," Apr 2019. [Online]. Available: <https://js.cytoscape.org/#introduction/about>

[33] "dropzone.js," Feb 2020. [Online]. Available: <https://www.dropzonejs.com/>

[34] "xml-js," Feb 2019. [Online]. Available: <https://www.npmjs.com/package/xml-js>

[35] "Web reactive framework," Jul 2016. [Online]. Available: <https://docs.spring.io/spring-framework/docs/5.0.0.M1/spring-framework-reference/html/web-reactive.html>

[36] "What is mongodb?" 2020. [Online]. Available: <https://www.mongodb.com/what-is-mongodb>

[37] R. T. Fielding, "Architectural styles and the design of network-based software architectures. university of california, irvine, 2000," 2016.

[38] "Rest," 2018. [Online]. Available: <https://restfulapi.net/rest-architectural-constraints/>

[39] O. S. Ramón, J. G. Molina, J. S. Cuadrado, J. Vanderdonckt *et al.*, "Gui generation from wireframes," in *14th Int. Conference on Human-Computer Interaction Interaccion*, 2013.

[40] M. Otto and J. Thornton, "Modal," Feb 2019. [Online]. Available: <https://getbootstrap.com/docs/4.0/components/modal/>

[41] G. Henke, "Html 5.0," Nov 2009.

[42] Cytoscape, "cytoscape/cytoscape.js-panzoom," Mar 2019. [Online]. Available: <https://github.com/cytoscape/cytoscape.js-panzoom>

[43] "cytoscape-grid-guide," Mar 2020. [Online]. Available: <https://www.npmjs.com/package/cytoscape-grid-guide>

[44] "cytoscape-context-menus," Mar 2020. [Online]. Available: <https://www.npmjs.com/package/cytoscape-context-menus>

[45] Cytoscape, "cytoscape/cytoscape.js-edgehandles," Oct 2019. [Online]. Available: <https://github.com/cytoscape/cytoscape.js-edgehandles>

[46] "Eventtarget.addeventlistener()," Mar 2020. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener>

[47] B. Spec, "Bson (binary json): Specification [electronic resource]," *Mobile access: http://bsonspec. org/spec. html.*

[48] "Basicdbobject," May 2011. [Online]. Available: <https://api.mongodb.com/java/2.6/com/mongodb/BasicDBObject.html>

[49] "Cartesian coordinates," May 2020. [Online]. Available: <https://mathworld.wolfram.com/CartesianCoordinates.html>

[50] "Extensible markup language (xml) 1.0 (fifth edition)," Nov 2008. [Online]. Available: <https://www.w3.org/TR/xml/>

[51] Y. Shafranovich, "Common format and mime type for comma-separated values (csv) files.(2005)," *Disponıvel em:< http://www. ietf. org/rfc/rfc4180. txt*, 2005.

[52] D. Crockford, *How JavaScript Works*. Virgule-Solidus, 2018.

[53] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, "Comparison of json and xml data interchange formats: a case study." *Caine*, vol. 9, pp. 157–162, 2009.

[54] K. Fakhroutdinov, "Uml sequence diagrams," Jan 2013. [Online]. Available: <https://www.uml-diagrams.org/sequence-diagrams.html>

[55] "Promise," Apr 2020. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise>

[56] "Blob," Feb 2020. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/API/Blob>

[57] "Introduction to the dom," Jan 2020. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction>

[58] J. Gibbons and B. C. d. S. Oliveira, "The essence of the iterator pattern," *Journal of functional programming*, vol. 19, no. 3-4, pp. 377–402, 2009.

[59] "Symbol," Apr 2020. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Glossary/Symbol>

[60] "Array.prototype.foreach()," May 2020. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach>

[61] K. Fakhroutdinov, "Uml use case diagrams," Jan 2014. [Online]. Available: <https://www.uml-diagrams.org/use-case-diagrams.html>

[62] "Getting started · jest," Oct 2019. [Online]. Available: <https://jestjs.io/docs/en/getting-started>

[63] L. Llobera, "Create react app · set up a modern web app by running one command." Feb 2020. [Online]. Available: <https://create-react-app.dev/docs/getting-started>

[64] "Build software better, together," 2020. [Online]. Available: <https://github.com/about>

[65] B. Beizer, *Software testing techniques*. Dreamtech Press, 2003.

[66] R. Osherove, "Naming standards for unit tests," Apr 2005. [Online]. Available: <https://osherove.com/blog/2005/4/3/naming-standards-for-unit-tests.html>

Příloha A

Testovací scénáře

Případ užití	**Vytvořit Activity Diagram**
Testovací scénář ID	**1**
Účel	Zkontrolovat, že platforma zobrazuje modální okno pro tvoření nového diagramu
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy
Aktivita	Kliknutí na ikonu plus vedle názvu "Your Graphs"
Výsledek	Otevře se modální okno s názvem "Create Graph"
Testovací scénář ID	**2**
Účel	Zkontrolovat, že lze vybrat typ diagramu Activity
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má zobrazené modální okno pro vytvoření diagramu
Aktivita	Ve vyjížděcím menu pro "Graph type" zvolení "Activity"
Výsledek	V kolonce pro typ diagramu se objeví Acitivity
Testovací scénář ID	**3**
Účel	Zkontrolovat, že lze zadat název nového diagramu
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má zobrazené modální okno pro vytvoření diagramu
Aktivita	V poli pro "Graph name" zadání názvu grafu
Výsledek	Zobrazení názvu diagramu v poli pro "Create Graph"
Testovací scénář ID	**4**
Účel	Zkontrolovat, že po vytvoření diagramu se uživateli zobrazuje název nového diagramu v levém seznamu diagramů a objeví se mu prázdný editor pro vytváření nového diagramu
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vyplněný název a typ nového diagramu
Aktivita	Kliknutí na tlačítko "Submit"
Výsledek	V levém seznamu se zobrazí název nového diagramu a zobrazí se prázdný editor pro vytvoření nového diagramu

Testovací scénář ID	**5**
Účel	Zkontrolovat, že lze vytvořit počáteční uzel
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořený nový activity diagram
Aktivita	Pravým tlačítkem zobrazení menu a kliknutí na "Add start"
Výsledek	V místě, kde bylo kliknuto pravým tlačítkem se zobrazí počáteční uzel UML Activity diagramu
Testovací scénář ID	**6**
Účel	Zkontrolovat, že lze vytvořit aktivity uzel
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořený nový activity diagram
Aktivita	Pravým tlačítkem zobrazení menu a kliknutí na "Add activity"
Výsledek	V místě, kde bylo kliknuto pravým tlačítkem se zobrazí aktivity uzel UML Activity diagramu
Testovací scénář ID	**7**
Účel	Zkontrolovat, že lze vytvořit rozhodovací uzel
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořený nový activity diagram
Aktivita	Pravým tlačítkem zobrazení menu a kliknutí na "Add decision"
Výsledek	V místě, kde bylo kliknuto pravým tlačítkem se zobrazí rozhodovací uzel UML Activity diagramu
Testovací scénář ID	**8**
Účel	Zkontrolovat, že lze vytvořit koncový uzel
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořený nový activity diagram
Aktivita	Pravým tlačítkem zobrazení menu a kliknutí na "Add end"
Výsledek	V místě, kde bylo kliknuto pravým tlačítkem se zobrazí koncový uzel UML Activity diagramu

Testovací scénář ID	**9**
Účel	Zkontrolovat, že lze vytvořit hranu mezi uzly
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořený nový activity diagram, ve kterém má minimálně dva uzly
Aktivita	Najetí myší na libovolný uzel a po podržení levého tlačítka myši přejetí na libovolný uzel mimo počáteční uzel
Výsledek	Mezi hranami zůstane hrana
Testovací scénář ID	**10**
Účel	Zkontrolovat, že nelze vytvořit hranu vedoucí do počátečního uzlu
Priorita	Nízká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořený nový activity diagram ve kterém má minimálně dva uzly. Jeden musí být počáteční uzel
Aktivita	Najetí myší na libovolný uzel a po podržení levého tlačítka myši přejetí na počáteční uzel
Výsledek	Vyskočí okno s chybovou hláškou: "No edge can go to the start node."
Testovací scénář ID	**11**
Účel	Zkontrolovat, že nelze vytvořit druhý počáteční uzel v jednom diagramu
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořený nový activity diagram ve kterém má vytvořený jeden počáteční uzel
Aktivita	Pravým tlačítkem zobrazení menu a kliknutí na "Add start"
Výsledek	Vyskočí okno s chybovou hláškou: "Graph already contains start node."
Testovací scénář ID	**12**
Účel	Zkontrolovat, že nelze vytvořit hranu vedoucí z koncového uzlu
Priorita	Nízká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořený nový activity diagram ve kterém má minimálně dva uzly. Jeden musí být koncový uzel
Aktivita	Najetí myší na koncový uzel a po podržení levého tlačítka myši přejetí na libovolný uzel
Výsledek	Vyskočí okno s chybovou hláškou: "No edge can go from the end node."

Případ užití	Přidat metadata
Testovací scénář ID	**1**
Účel	Zkontrolovat, že lze přepnout v pravém panelu metadat do režimu přidávání atributů
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram alespoň s jedním uzlem. Má rozkluknuté stávající metadata uzlu.
Aktivita	Kliknutí na tlačítko "Add attribute"
Výsledek	Zobrazí se další políčko, kde bude vyjížděcí menu a místo tlačítka "Add attribute" se objeví tlačítko "Cancel" a tlačítko "Save"
Testovací scénář ID	**2**
Účel	Zkontrolovat, že je možné zobrazit metadata z kterých je možné vybrat, které přidat.
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram alespoň s jedním uzlem. Má rozkliknuté stávající metadata uzlu a klikl na tlačítko "Add attribute"
Aktivita	Kliknutí na vyjížděcí menu v novém políčku.
Výsledek	Zobrazí se možné atributy na přidání
Testovací scénář ID	**3**
Účel	Zkontrolovat, že lze vybrat nový atribut uzlu
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram alespoň s jedním uzlem. Má rozkliknuté stávající metadata uzlu a klikl na tlačítko "Add attribute"
Aktivita	Ve vyjížděcí menu vybere libovolný atribut
Výsledek	V daném poli zůstane zobrazen daný atribut
Testovací scénář ID	**4**
Účel	Zkontrolovat, že je možné přidat hodnotu k danému atributu
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram alespoň s jedním uzlem. Má rozkluknuté stávající metadata uzlu a klikl na tlačítko "Add attribute". Vybral jeden z atributů
Aktivita	V políčku vedle atributu vyplnění libovolné hodnoty
Výsledek	V daném poli hodnota zůstane
Testovací scénář ID	**5**
Účel	Zkontrolovat, že lze uložit nový atribut s vyplněnou hodnotou
Priorita	Střední

Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram alespoň s jedním uzlem. Má rozkliknuté stávající metadata uzlu a klikl na tlačítko "Add attribute". Vybral jeden z atributů a napsal k němu hodnotu.
Aktivita	Kliknutí na tlačítko "Save"
Výsledek	Hodnota se zapíše se stejným formátem jako předchozí atributy a tlačítko "Save" a tlačítko "Cancel" zmizí a zobrazí se opět tlačítko "Add attribute"
Testovací scénář ID	**6**
Účel	Zkontrolovat, že nelze přidat nový atribut bez vyplněné hodnoty
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram alespoň s jedním uzlem. Má rozkluknuté stávající metadata uzlu a klikl na tlačítko "Add attribute". Vybral jeden z atributů
Aktivita	Kliknutí na tlačítko "Save"
Výsledek	V místě, nevyplněného hodnoty atributu se zobrazí hláška: "Vyplňte prosím toto pole"
Testovací scénář ID	**7**
Účel	Zkontrolovat, že nelze znovu vložit stejný atribut
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram alespoň s jedním uzlem. Má rozkliknuté stávající metadata uzlu a klikl na tlačítko "Add attribute".
Aktivita	Kliknutí na vyjížděcí menu v novém políčku.
Výsledek	Ve vyjížděcím menu bude o jednu možnost mé a to přesně o tu, která již byla využita
Testovací scénář ID	**8**
Účel	Zkontrolovat, že po vložení posledního atributu nepůjde kliknout na tlačítko "Add attribute"
Priorita	Nízká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram alespoň s jedním uzlem. Má rozkliknuté stávající metadata uzlu a již jsou přidány všechny možné doplňkové atributy
Aktivita	Kliknutí na tlačítko "Add attribute"
Výsledek	Tlačítko má světlejší barvu a po kliknutí na něj se vůbec nic nestane

Případ užití	Upravit metadata
Testovací scénář ID	**1**
Účel	Zkontrolovat, že lze upravit hodnotu nově přidaného atributu
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram alespoň s jedním uzlem. Má rozkliknuté stávající metadata uzlu a je tam alespoň jeden nový atribut.
Aktivita	Změnění hodnoty atribut.
Výsledek	Hodnota atributu zůstane změněna.

Případ užití	Smazat metadata
Testovací scénář ID	**1**
Účel	Zkontrolovat, že lze smazat nově přidaný atribut
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram alespoň s jedním uzlem. Má rozkliknuté stávající metadata uzlu a je tam alespoň jeden nový atribut.
Aktivita	Kliknutí na tlačítko na pravo od hodnoty atributu s křížkem.
Výsledek	Atribut i s hodnotou zmizí.
Testovací scénář ID	**2**
Účel	Zkontrolovat, že po smazání atributu lze znovu přidat
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram alespoň s jedním uzlem. Má rozkliknuté stávající metadata uzlu a jeden uzel uživatel smaže.
Aktivita	Kliknutí na tlačítko "Add attribute" a rozkliknutí vyjížděcího menu
Výsledek	Zde se mezi ostatními nevyužitými atributy objevý i ten smazaný

Případ užití	**Import diagramu z csv formátu**
Testovací scénář ID	**1**
Účel	Zkontrolovat, že po kliknutí na ikonu import, tak se zobrazí modální okno
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy.
Aktivita	V levém okně vedle názvu "Your Graphs" kliknutí na ikonu import
Výsledek	Zobrazí se modální okno s názvem "Import Graph"
Testovací scénář ID	**2**
Účel	Zkontrolovat, že lze dropnout do modálního okna soubor s příponou csv
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a klikl na ikonu importu diagramů
Aktivita	Přetahnutí do vyznačeného pole soubor s diagramem ve formátu csv
Výsledek	Zobrazí se název grafu a vyjížděcí menu pod názvem "Import files"
Testovací scénář ID	**3**
Účel	Zkontrolovat, že lze přidat do modálního okna soubor s příponou csv přes tlačítko "Open File Dialog"
Priorita	Nízká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a klikl na ikonu importu diagramů
Aktivita	Po kliknutí na tlačítko "Open File Dialog" vybere soubor s diagramem ve formátu csv
Výsledek	Zobrazí se název grafu a vyjížděcí menu pod názvem "Import files"
Testovací scénář ID	**4**
Účel	Zkontrolovat, že lze po přidání souboru ve formátu csv zvolit jako platformu Rhino
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a klikl na ikonu importu diagramů. Vložil do modálního okna soubor ve formátu csv
Aktivita	Pod názvem "Import files" vedle názvu importovaného souboru vybrání ve vyjížděcím menu možnost Rhino
Výsledek	Platforma Rhino zůstane vybrána

Testovací scénář ID	**5**
Účel	Zkontrolovat, že vložení souboru a vybrání platformy popř. typu diagramu se modální okno zavře a zobrazí se importovaný diagram
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a klikl na ikonu importu diagramů. Vložil do modálního okna soubor a zvolil platformu
Aktivita	Kliknutí na tlačítko "Import"
Výsledek	Modální okno se zavře a zobrazí se importovaný diagram
Testovací scénář ID	**6**
Účel	Zkontrolovat, že lze vložit více diagramů zároveň v různých formátech
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a klikl na ikonu importu diagramů. Vložil do modálního okna soubory a zvolil jejich platformy
Aktivita	Kliknutí na tlačítko "Import"
Výsledek	Modální okno se zavře a zobrazí se poslední importovaný diagram

Případ užití	**Import diagramu z json formátu**
Testovací scénář ID	**1**
Účel	Zkontrolovat, že po kliknutí na ikonu import, tak se zobrazí modální okno
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy.
Aktivita	V levém okně vedle názvu "Your Graphs" kliknutí na ikonu import
Výsledek	Zobrazí se modální okno s názvem "Import Graph"
Testovací scénář ID	**2**
Účel	Zkontrolovat, že lze dropnout do modálního okna soubor s příponou json
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a klikl na ikonu importu diagramů
Aktivita	Přetahnutí do vyznačeného pole soubor s diagramem ve formátu json
Výsledek	Zobrazí se název grafu a vyjížděcí menu pod názvem "Import files"
Testovací scénář ID	**3**
Účel	Zkontrolovat, že lze přidat do modálního okna soubor s příponou json přes tlačítko "Open File Dialog"
Priorita	Nízká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a klikl na ikonu importu diagramů
Aktivita	Po kliknutí na tlačítko "Open File Dialog" vybere soubor s diagramem ve formátu json
Výsledek	Zobrazí se název grafu a vyjížděcí menu pod názvem "Import files"
Testovací scénář ID	**4**
Účel	Zkontrolovat, že lze po přidání souboru ve formátu json zvolit jako platformu Rhino
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a klikl na ikonu importu diagramů. Vložil do modálního okna soubor ve formátu json
Aktivita	Pod názvem "Import files" vedle názvu importovaného souboru vybrání ve vyjížděcím menu možnost Rhino
Výsledek	Platforma Rhino zůstane vybrána

Testovací scénář ID	**5**
Účel	Zkontrolovat, že lze po přidání souboru ve formátu json zvolit jako platformu Oxygen
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a klikl na ikonu importu diagramů. Vložil do modálního okna soubor ve formátu json
Aktivita	Pod názvem "Import files" vedle názvu importovaného souboru vybrání ve vyjížděcím menu možnost Oxygen
Výsledek	Platforma Oxygen zůstane vybrána
Testovací scénář ID	**6**
Účel	Zkontrolovat, že vložení souboru a vybrání platformy popř. typu diagramu se modální okno zavře a zobrazí se importovaný diagram
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a klikl na ikonu importu diagramů. Vložil do modálního okna soubor a zvolil platformu
Aktivita	Kliknutí na tlačítko "Import"
Výsledek	Modální okno se zavře a zobrazí se importovaný diagram
Testovací scénář ID	**7**
Účel	Zkontrolovat, že lze vložit více diagramů zároveň v různých formátech
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a klikl na ikonu importu diagramů. Vložil do modálního okna soubory a zvolil jejich platformy
Aktivita	Kliknutí na tlačítko "Import"
Výsledek	Modální okno se zavře a zobrazí se poslední importovaný diagram

Případ užití	Import diagramu z xml formátu
Testovací scénář ID	**1**
Účel	Zkontrolovat, že po kliknutí na ikonu import se zobrazí modální okno
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy.
Aktivita	Kliknutí na ikonu import v levém okně vedle názvu "Your Graphs"
Výsledek	Zobrazí se modální okno s názvem "Import Graph"
Testovací scénář ID	**2**
Účel	Zkontrolovat, že lze dropnout do modálního okna soubor s příponou xml
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a klikl na ikonu importu diagramů
Aktivita	Přetáhnutí do vyznačeného pole soubor s diagramem ve formátu xml
Výsledek	Zobrazí se název grafu a vyjížděcí menu pod názvem "Import files"
Testovací scénář ID	**3**
Účel	Zkontrolovat, že lze přidat do modálního okna soubor s příponou xml přes tlačítko "Open File Dialog"
Priorita	Nízká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a klikl na ikonu importu diagramů
Aktivita	Po kliknutí na tlačítko "Open File Dialog" volba souboru s diagramem ve formátu xml
Výsledek	Zobrazí se název grafu a vyjížděcí menu pod názvem "Import files"
Testovací scénář ID	**4**
Účel	Zkontrolovat, že lze po přidání souboru ve formátu xml zvolit jako platformu Rhino
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a klikl na ikonu importu diagramů. Vložil do modálního okna soubor ve formátu xml
Aktivita	Zvolení možnosti Rhino ve vyjížděcím menu pod názvem "Import files" vedle názvu importovaného souboru
Výsledek	Platforma Rhino zůstane vybrána

Testovací scénář ID	**5**
Účel	Zkontrolovat, že lze po přidání souboru ve formátu xml zvolit jako platformu Oxygen
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy, klikl na ikonu importu diagramů a vložil do modálního okna soubor ve formátu xml
Aktivita	Zvolení možnosti Oxygen ve vyjížděcím menu pod názvem "Import files" vedle názvu importovaného souboru
Výsledek	Platforma Oxygen zůstane vybrána
Testovací scénář ID	**6**
Účel	Zkontrolovat, že lze po přidání souboru ve formátu xml zvolit jako platformu Draw.io
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy, klikl na ikonu importu diagramů a vložil do modálního okna soubor ve formátu xml
Aktivita	Zvolení možnosti Draw.io ve vyjížděcím menu pod názvem "Import files" vedle názvu importovaného souboru
Výsledek	Platforma Draw.io zůstane vybrána a objeví se další vyjížděcí menu s názvem "Graph type"
Testovací scénář ID	**7**
Účel	Zkontrolovat, že lze po přidání souboru ve formátu xml a zvolení platformy Draw.io vybrat jako typ diagramu Activity
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a klikl na ikonu importu diagramů. Vložil do modálního okna soubor ve formátu xml a zvolení platformy Draw.io
Aktivita	Zvolení "Activity" ve vyjížděcím menu pod názvem souboru
Výsledek	V daném vyjížděcím menu zůstane hodnota "Activity"
Testovací scénář ID	**8**
Účel	Zkontrolovat, že lze po přidání souboru ve formátu xml a zvolení platformy Draw.io vybrat jako typ diagramu Flow
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy, klikl na ikonu importu diagramů, vložil do modálního okna soubor ve formátu xml a zvolil platformu Draw.io

Aktivita	Zvolení "Flow" ve vyjížděcím menu pod názvem souboru
Výsledek	V daném vyjížděcím menu zůstane hodnota "Flow"
Testovací scénář ID	**9**
Účel	Zkontrolovat, že vložení souboru a vybrání platformy popř. typu diagramu se modální okno zavře a zobrazí se importovaný diagram
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy, klikl na ikonu importu diagramů, vložil do modálního okna soubor a zvolil platformu
Aktivita	Kliknutí na tlačítko "Import"
Výsledek	Modální okno se zavře a zobrazí se importovaný diagram
Testovací scénář ID	**10**
Účel	Zkontrolovat, že lze vložit více diagramů zároveň v různých formátech
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy, klikl na ikonu importu diagramů, vložil do modálního okna soubory a zvolil jejich platformy
Aktivita	Kliknutí na tlačítko "Import"
Výsledek	Modální okno se zavře a zobrazí se poslední importovaný diagram

Případ užití	**Import diagramů z prj formátu**
Testovací scénář ID	**1**
Účel	Zkontrolovat, že po kliknutí na ikonu import, tak se zobrazí modální okno
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy.
Aktivita	V levém okně vedle názvu "Your Graphs" kliknutí na ikonu import
Výsledek	Zobrazí se modální okno s názvem "Import Graph"
Testovací scénář ID	**2**
Účel	Zkontrolovat, že lze dropnout do modálního okna soubor s příponou prj
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a klikl na ikonu importu diagramů
Aktivita	Přetahnutí do vyznačeného pole soubor s diagramemy ve formátu prj
Výsledek	Zobrazí se název grafu a vyjížděcí menu pod názvem "Import files"
Testovací scénář ID	**3**
Účel	Zkontrolovat, že lze přidat do modálního okna soubor s příponou prj přes tlačítko "Open File Dialog"
Priorita	Nízká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a klikl na ikonu importu diagramů
Aktivita	Po kliknutí na tlačítko "Open File Dialog" vybere soubor s diagramemy ve formátu prj
Výsledek	Zobrazí se název grafu a vyjížděcí menu pod názvem "Import files"
Testovací scénář ID	**4**
Účel	Zkontrolovat, že lze po přidání souboru ve formátu prj zvolit jako platformu Oxygen
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a klikl na ikonu importu diagramů. Vložil do modálního okna soubor ve formátu prj
Aktivita	Pod názvem "Import files" vedle názvu importovaného souboru vybrání ve vyjížděcím menu možnost Oxygen
Výsledek	Platforma Oxygen zůstane vybrána

Testovací scénář ID	**5**
Účel	Zkontrolovat, že vložení souboru a vybrání platformy popř. typu diagramu se modální okno zavře a zobrazí se importovaný diagram
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a klikl na ikonu importu diagramů. Vložil do modálního okna soubor a zvolil platformu
Aktivita	Kliknutí na tlačítko "Import"
Výsledek	Modální okno se zavře a zobrazí se importovaný diagram
Testovací scénář ID	**6**
Účel	Zkontrolovat, že lze vložit více diagramů zároveň v různých formátech
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a klikl na ikonu importu diagramů. Vložil do modálního okna soubory a zvolil jejich platformy
Aktivita	Kliknutí na tlačítko "Import"
Výsledek	Modální okno se zavře a zobrazí se poslední importovaný diagram

	Import diagramu negativní scénáře
Testovací scénář ID	**1**
Účel	Zkontrolovat, že pokud není vybrána platforma, tak se diagram neimportuje
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a klikl na ikonu importu diagramů. Vložil do modálního okna soubory
Aktivita	Kliknutí na tlačítko "Import"
Výsledek	Modální okno se nezavře a zobrazí hláška u volby platformy "Vyberte prosím ze seznamu některou položku"
Testovací scénář ID	**2**
Účel	Zkontrolovat, že nelze importovat soubor v nepodporovaném formátu
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a klikl na ikonu importu diagramů.
Aktivita	Pomocí drag and drop vloží do platformy lobovolný soubor v nepodporovaném formátu
Výsledek	Modální okno zobrazí chybovou hlášku ve formátu "Unaccepted file format: nazev souboru.pripona" a nezobrazí soubor v seznamu vložených souborů
Testovací scénář ID	**3**
Účel	Zkontrolovat, že nelze vložit dva diagramy se stejným názvem
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a již importoval úspěšně jeden diagram a opět otevřel modální okno a vložil stejný soubor a zadal stejné údaje
Aktivita	Kliknutí na tlačítko "Import"
Výsledek	Modální okno se nezavře a chybovou hlášku "IMPORT: unaccepted to have two graphs with same name." a soubor zůstane v seznamu vložených souborů

Testovací scénář ID	**4**
Účel	Zkontrolovat, že nelze vložit diagram s vadnými daty uzlů
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a klikl na ikonu importu diagramů. Vložil do modálního okna soubor s diagramem, co má špatně zadefinované atributy jednoho uzlu.
Aktivita	Kliknutí na tlačítko "Import"
Výsledek	Modální okno se nezavře a zobrazí se chybová hláška "IMPORT: add nodes to graph faild, check if your file have right structure."
Testovací scénář ID	**5**
Účel	Zkontrolovat, že nelze vložit diagram s vadnými daty hran
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a klikl na ikonu importu diagramů. Vložil do modálního okna soubor s diagramem, co má špatně zadefinované atributy jedné hrany.
Aktivita	Kliknutí na tlačítko "Import"
Výsledek	Modální okno se nezavře a zobrazí se chybová hláška "IMPORT: edges target or source is incorrect, check if your file have right structure."
Testovací scénář ID	**6**
Účel	Zkontrolovat, že v případě nahrávání více diagramů se nahrají ty správné a smažou ty špatné
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a klikl na ikonu importu diagramů. Vložil do modálního okna soubor s nepoškozeným diagramem a druhý s poškozeným diagramem
Aktivita	Kliknutí na tlačítko "Import"
Výsledek	Modální okno se zavře a zobrazí se importovaný diagram

Případ užití	**Export diagramu do json formátu**
Testovací scénář ID	**1**
Účel	Zkontrolovat, že po kliknutí na ikonu export diagramu se zobrazí modální okno pro export diagramu
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram.
Aktivita	V aktivním diagramu kliknutí na ikonu exportu
Výsledek	Zobrazí se modální okno s názvem "Export Graph"
Testovací scénář ID	**2**
Účel	Zkontrolovat, že se zobrazí jako název souboru název diagramu
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram.
Aktivita	V aktivním diagramu kliknutí na ikonu exportu
Výsledek	V poli pod názvem "Graph name" je stejný jako v levém sloupci nad tlačítkem "Save graph"
Testovací scénář ID	**3**
Účel	Zkontrolovat, že nelze změnit hodnota v poli "Graph name"
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram. Kliknul na ikonu export aktivního diagramu.
Aktivita	Kliknutí do poled pod názvem "Graph name"
Výsledek	Po kliknutí se vůbec nic nestane
Testovací scénář ID	**4**
Účel	Zkontrolovat, že lze vybrat možnost json jako formát exportovaného souboru
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram. Kliknul na ikonu export aktivního diagramu.
Aktivita	V modálním okně ve vyjížděcím menu pod názvem "Export type" vybrání možnosti json
Výsledek	Zůstane zobrazena hodnota json

Testovací scénář ID	**5**
Účel	Zkontrolovat, že po kliknutí na tlačítko "Export" se uloží do počítače soubor ve formátu json se zvoleným názvem.
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram. Kliknul na ikonu export aktivního diagramu. V modálním okně byl vybrán typ exportovaného souboru json
Aktivita	Kliknutí na tlačítko "Export"
Výsledek	Do počítače se uloží soubor ve formátu json a jmenuje se tak, jak se jmenuje graf

Případ užití	**Export diagramu do xml formátu**
Testovací scénář ID	**1**
Účel	Zkontrolovat, že po kliknutí na ikonu export grafu se zobrazí modální okno pro export diagramu
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram.
Aktivita	V aktivním diagramu kliknutí na ikonu exportu
Výsledek	Zobrazí se modální okno s názvem "Export Graph"
Testovací scénář ID	**2**
Účel	Zkontrolovat, že se zobrazí jako název souboru název diagramu
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram.
Aktivita	V aktivním diagramu kliknutí na ikonu exportu
Výsledek	V poli pod názvem "Graph name" je stejný jako v levém sloupci nad tlačítkem "Save graph"
Testovací scénář ID	**3**
Účel	Zkontrolovat, že nelze změnit hodnota v poli "Graph name"
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram. Kliknul na ikonu export aktivního diagramu.
Aktivita	Kliknutí do poled pod názvem "Graph name"
Výsledek	Po kliknutí se vůbec nic nestane

Testovací scénář ID	**4**
Účel	Zkontrolovat, že lze vybrat možnost xml jako formát exportovaného souboru
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram. Kliknul na ikonu export aktivního diagramu.
Aktivita	V modálním okně ve vyjížděcím menu pod názvem "Export type" vybrání možnosti xml
Výsledek	Zůstane zobrazena hodnota xml
Testovací scénář ID	**5**
Účel	Zkontrolovat, že po kliknutí na tlačítko "Export" se uloží do počítače soubor ve formátu xml se
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram. Kliknul na ikonu export aktivního diagramu.
Aktivita	Kliknutí na tlačítko "Export"
Výsledek	Do počítače se uloží soubor ve formátu xml a jmenuje se tak, jak se jmenuje diagram

Případ užití	**Export diagramu do csv formátu**
Testovací scénář ID	**1**
Účel	Zkontrolovat, že po kliknutí na ikonu export grafu se zobrazí modální okno pro export diagramu
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram.
Aktivita	V aktivním diagramu kliknutí na ikonu exportu
Výsledek	Zobrazí se modální okno s názvem "Export Graph"
Testovací scénář ID	**2**
Účel	Zkontrolovat, že se zobrazí jako název souboru název diagramu
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram.
Aktivita	V aktivním diagramu kliknutí na ikonu exportu
Výsledek	V poli pod názvem "Graph name" je stejný jako v levém sloupci nad tlačítkem "Save graph"

Testovací scénář ID	**3**
Účel	Zkontrolovat, že nelze změnit hodnota v poli "Graph name"
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram. Kliknul na ikonu export aktivního diagramu.
Aktivita	Kliknutí do poled pod názvem "Graph name"
Výsledek	Po kliknutí se vůbec nic nestane
Testovací scénář ID	**4**
Účel	Zkontrolovat, že lze vybrat možnost csv jako formát exportovaného souboru
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram. Kliknul na ikonu export aktivního diagramu.
Aktivita	V modálním okně ve vyjížděcím menu pod názvem "Export type" vybrání možnosti csv
Výsledek	Zůstane zobrazena hodnota csv
Testovací scénář ID	**5**
Účel	Zkontrolovat, že po kliknutí na tlačítko "Export" se uloží do počítače soubor ve formátu csv se zvoleným názvem.
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram. Kliknul na ikonu export aktivního diagramu. V modálním okně byl vybrán typ exportovaného souboru csv
Aktivita	Kliknutí na tlačítko "Export"
Výsledek	Do počítače se uloží soubor ve formátu csv a jmenuje se tak, jak se jmenuje diagram

Případ užití	Export testovacích scénářů do json formátu
Testovací scénář ID	**1**
Účel	Zkontrolovat, že po kliknutí na ikonu export testovacích scénářů se zobrazí modální okno pro export testovacích scénářů
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram s několika uzlemi spopojenými hranami a jedním počátečním uzlem. Vygenerované testovací scénáře s libovolnými parametry.
Aktivita	Napravo v okně s testovacími scénáři kliknutí na ikonu exportu
Výsledek	Zobrazí se modální okno s názvem "Export Test Cases"
Testovací scénář ID	**2**
Účel	Zkontrolovat, že se zobrazí jako název souboru název testovacích scénářů
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram s několika uzlemi spopojenými hranami a jedním počátečním uzlem. Vygenerované testovací scénáře s libovolnými parametry.
Aktivita	Napravo v okně s testovacími scénáři kliknutí na ikonu exportu
Výsledek	V poli pod názvem "Test Cases name" je stejný jako v levém sloupci pod tlačítkem "Show TDL"
Testovací scénář ID	**3**
Účel	Zkontrolovat, že lze zvolit název exportovaného souboru
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram s několika uzlemi spopojenými hranami a jedním počátečním uzlem. Vygenerované testovací scénáře s libovolnými parametry.
Aktivita	V modálním okně v poli pod názvem "Test Cases name" změněna názvu
Výsledek	Zůstane zobrazena nový název
Testovací scénář ID	**4**
Účel	Zkontrolovat, že lze vybrat možnost json jako formát exportovaného souboru
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram s několika uzlemi spopojenými hranami a jedním počátečním uzlem. Vygenerované testovací scénáře s libovolnými parametry.

Aktivita	V modálním okně ve vyjížděcím menu pod názvem "Export type" vybrání možnosti json
Výsledek	Zůstane zobrazena hodnota json
Testovací scénář ID	**5**
Účel	Zkontrolovat, že po kliknutí na tlačítko "Export" se uloží do počítače soubor ve formátu json se zvoleným názvem.
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram s několika uzlemi spopojenými hranami a jedním počátečním uzlem. Vygenerované testovací scénáře s libovolnými parametry. V modálním okně byl vybrán typ exportovaného souboru json
Aktivita	Kliknutí na tlačítko "Export"
Výsledek	Do počítače se uloží soubor ve formátu json a jmenuje se tak, jak byly pojmenovány testovací scénáře

Případ užití	**Export testovacích scénářů do xml formátu**
Testovací scénář ID	**1**
Účel	Zkontrolovat, že po kliknutí na ikonu export testovacích scénářů se zobrazí modální okno pro export testovacích scénářů
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram s několika uzlemi spopojenými hranami a jedním počátečním uzlem. Vygenerované testovací scénáře s libovolnými parametry.
Aktivita	Napravo v okně s testovacími scénáři kliknutí na ikonu exportu
Výsledek	Zobrazí se modální okno s názvem "Export Test Cases"
Testovací scénář ID	**2**
Účel	Zkontrolovat, že se zobrazí jako název souboru název testovacích scénářů
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram s několika uzlemi spopojenými hranami a jedním počátečním uzlem. Vygenerované testovací scénáře s libovolnými parametry.
Aktivita	Napravo v okně s testovacími scénáři kliknutí na ikonu exportu
Výsledek	V poli pod názvem "Test Cases name" je stejný jako v levém sloupci pod tlačítkem "Show TDL"

Testovací scénář ID	**3**
Účel	Zkontrolovat, že lze zvolit název exportovaného souboru
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram s několika uzlemi spopojenými hranami a jedním počátečním uzlem. Vygenerované testovací scénáře s libovolnými parametry.
Aktivita	V modálním okně v poli pod názvem "Test Cases name" změněna názvu
Výsledek	Zůstane zobrazena nový název
Testovací scénář ID	**4**
Účel	Zkontrolovat, že lze vybrat možnost xml jako formát exportovaného souboru
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram s několika uzlemi spopojenými hranami a jedním počátečním uzlem. Vygenerované testovací scénáře s libovolnými parametry.
Aktivita	V modálním okně ve vyjížděcím menu pod názvem "Export type" vybrání možnosti xml
Výsledek	Zůstane zobrazena hodnota xml
Testovací scénář ID	**5**
Účel	Zkontrolovat, že po kliknutí na tlačítko "Export" se uloží do počítače soubor ve formátu xml se zvoleným názvem.
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram s několika uzlemi spopojenými hranami a jedním počátečním uzlem. Vygenerované testovací scénáře s libovolnými parametry. V modálním okně byl vybrán typ exportovaného souboru xml
Aktivita	Kliknutí na tlačítko "Export"
Výsledek	Do počítače se uloží soubor ve formátu xml a jmenuje se tak, jak byly pojmenovány testovací scénáře

Případ užití	**Export testovacích scénářů do csv formátu**
Testovací scénář ID	**1**
Účel	Zkontrolovat, že po kliknutí na ikonu export testovacích scénářů se zobrazí modální okno pro export testovacích scénářů
Priorita	Vysoká
Aktér	Uživatel

Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram s několika uzlemi spopojenými hranami a jedním počátečním uzlem. Vygenerované testovací scénáře s libovolnými parametry.
Aktivita	Napravo v okně s testovacími scénáři kliknutí na ikonu exportu
Výsledek	Zobrazí se modální okno s názvem "Export Test Cases"
Testovací scénář ID	**2**
Účel	Zkontrolovat, že se zobrazí jako název souboru název testovacích scénářů
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram s několika uzlemi spopojenými hranami a jedním počátečním uzlem. Vygenerované testovací scénáře s libovolnými parametry.
Aktivita	Napravo v okně s testovacími scénáři kliknutí na ikonu exportu
Výsledek	V poli pod názvem "Test Cases name" je stejný jako v levém sloupci pod tlačítkem "Show TDL"
Testovací scénář ID	**3**
Účel	Zkontrolovat, že lze zvolit název exportovaného souboru
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram s několika uzlemi spopojenými hranami a jedním počátečním uzlem. Vygenerované testovací scénáře s libovolnými parametry.
Aktivita	V modálním okně v poli pod názvem "Test Cases name" změněna názvu
Výsledek	Zůstane zobrazena nový název
Testovací scénář ID	**4**
Účel	Zkontrolovat, že lze vybrat možnost csv jako formát exportovaného souboru
Priorita	Střední
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram s několika uzlemi spopojenými hranami a jedním počátečním uzlem. Vygenerované testovací scénáře s libovolnými parametry.
Aktivita	V modálním okně ve vyjížděcím menu pod názvem "Export type" vybrání možnosti csv
Výsledek	Zůstane zobrazena hodnota csv

Testovací scénář ID	5
Účel	Zkontrolovat, že po kliknutí na tlačítko "Export" se uloží do počítače soubor ve formátu csv se zvoleným názvem.
Priorita	Vysoká
Aktér	Uživatel
Předpoklad	Uživatel je přihlášen do platformy a má vytvořen diagram s několika uzlemi spopojenými hranami a jedním počátečním uzlem. Vygenerované testovací scénáře s libovolnými parametry. V modálním okně byl vybrán typ exportovaného souboru csv
Aktivita	Kliknutí na tlačítko "Export"
Výsledek	Do počítače se uloží soubor ve formátu csv a jmenuje se tak, jak byly pojmenovány testovací scénáře

Příloha B

Struktura příloh elektronické verze práce

Součástí elektronické verze práce jsou zdrojové soubory platformy Rhino. Příloha je rozdělena do složky „frontend“, kde se nachází klientská část platformy, ve složce „backend“ je uložena serverová část platformy a v souboru „README.md“ je návod jak platformu zprovozni[1].

[1]Pozor! V případě spouštění platformy lokálně je třeba změnit údaje v souboru „backend/src/main/resources/application.properties“ pro úspěšné spojení s lokální databází.