TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Vytvoření programu řešícího síťovou analýzu (CPM, PERT)

Diplomová práce

Studijní program: N2612 Elektrotechnika a informatika
Studijní obor: Informační technologie

Autor práce: **Bc. Karel Drnec**
Vedoucí práce: Ing. Josef Chudoba, Ph.D.
Ústav nových technologií a aplikované informatiky

Liberec 2022

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání diplomové práce

Vytvoření programu řešícího síťovou analýzu (CPM, PERT)

Jméno a příjmení: **Bc. Karel Drnec**
Osobní číslo: M20000163
Studijní program: N2612 Elektrotechnika a informatika
Studijní obor: Informační technologie
Zadávající katedra: Ústav nových technologií a aplikované informatiky
Akademický rok: **2021/2022**

Zásady pro vypracování:

1. Rešerše stávajících komerčních SW.
2. Vytvoření vlastního programu řešícího komplexní projekty (CPM, PERT) s využitím stochastických přístupů.
3. Ověření vytvořeného programu na příkladech a automatické provedení a zpracování textových a grafických výstupů.
4. Kritické zhodnocení vytvořeného programu s komerčními produkty.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	40-50 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] IHRIG, Colin J. Pro node.js for developers. [Berkeley]: Apress, [2013]. Expert's voice in Web development. ISBN 1430258608.

[2] EAST, William. Critical Path Method (CPM) Tutor for Construction Planning and Scheduling. McGraw-Hill Education, 2015. ISBN 0071849238.

[3] LEWIS, James. Project Planning, Scheduling, and Control: The Ultimate Hands-On Guide to Bringing Projects in On Time and On Budget. 5th Edition. McGraw-Hill Education, 2010. ISBN 0071746528.

[4] ROBERT, Christian a George CASELLA. Monte Carlo Statistical Methods. 2nd ed. New York: Springer-Verlag New York, 2004. ISBN 978-1-4757-4145-2.

Vedoucí práce:	Ing. Josef Chudoba, Ph.D. Ústav nových technologií a aplikované informatiky
Datum zadání práce:	12. října 2021
Předpokládaný termín odevzdání:	16. května 2022

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

Ing. Josef Novák, Ph.D.
vedoucí ústavu

V Liberci dne 19. října 2021

Prohlášení

Prohlašuji, že svou diplomovou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Jsem si vědom toho, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má diplomová práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

16. května 2022 Bc. Karel Drnec

Poděkování

Chtěl bych poděkovat panu Ing. Josefu Chudobovi, Ph.D. za odborné vedení, cenné rady, trpělivost a ochotu, kterou mi v průběhu diplomové práce věnoval. V neposlední řadě bych chtěl poděkovat svým blízkým za podporu.

Abstrakt

Tato diplomová práce řeší metody užívané v řízení projektů. Jedná se o metodu kritické cesty (zkráceně CPM) a její stochastickou nadstavbu PERT. Cílem práce v teoretické části je zpracování informací o metodách, které budou využity v praktické části práce. V praktické části je hlavním cílem udělat rešerši existujících softwarů a knihoven. Dále vytvořit aplikaci, která se zabývá oběma metodami v grafickém uživatelském rozhraní. Pro praktickou část byla vytvořena webová aplikace za pomoci technologie Node.js a webového frameworku Express.js. V praktické části dále je porovnání vytvořené aplikace s komerčními softwary a ověření výpočtů vytvořené aplikace.

Klíčová slova: CPM, PERT, pravděpodobnost, kritický, Monte Carlo, JavaScript

Abstract

This diploma thesis addresses the methods used in project management. It is a critical path method (abbreviated CPM) and its stochastic superstructure PERT. The goal of the work in the theoretical part is to process information about methods that will be used in the practical part of the work. In the practical part, the main goal is to search for existing software and libraries. Next, create an application that deals with both methods in the graphical user interface. For the practical part, a web application was created using Node.js technology and the Express.js web framework. The practical part also compares the created application with commercial software and verifying the calculations created by the application.

Keywords: CPM, PERT, probability, critical, Monte Carlo, JavaScript

Obsah

Seznam obrázků 7
Seznam tabulek 7
Seznam zkratek a použitého značení 8
Úvod 9
1 Metody užívané v síťové analýze 10
1.1 Metoda kritické cesty (CPM – Critical Path Method) 10
1.2 PERT (Project Evaluation and Review Technique) 12
1.3 Monte Carlo 13
2 Rešerše dostupných aplikací a knihoven 15
2.1 Lucidchart 15
2.2 Microsoft Project 16
2.3 PERT Calculator 18
2.4 Knihovny zabývající se metodou CPM/PERT 19
2.5 Statistické knihovny 21
3 Aplikace 23
3.1 O aplikaci 23
3.2 Struktura vytvořené aplikace 25
3.3 Grafické objekty v aplikaci 27
3.4 Zobrazení projektu 29
3.5 Výpočet metody CPM a PERT 38
3.6 Simulace Monte Carlo 44
3.7 Nastavení aplikace 48
3.8 Další funkce aplikace 49
3.9 Zabezpečení 54
3.10 Porovnání s komerčními softwary 58
4 Ověření správnosti výpočtu výsledků 59
4.1 Testovací příklad 59
Závěr 61
Seznam použité literatury 62
Přílohy 64
A Ukázka aplikace pro srovnání knihoven SciPy a jStat 64
B Obsah přiloženého CD 65
C Generovaný projekt 66
D Testovací příklad 67

Seznam obrázků

Obrázek 1: Graf projektu po proběhlé analýze CPM 10
Obrázek 2: Diagram vytvořený v aplikaci Lucidchart 16
Obrázek 3: Síťový diagram v Microsoft Project 17
Obrázek 4: Prostředí webové aplikace Microsoft Project 17
Obrázek 5: Prostředí aplikace PERT Calculator 18
Obrázek 6: Zobrazení výsledků aplikace PERT Calculator 19
Obrázek 7: Diagram metody PERT knihovny AnyChart 20
Obrázek 8: Návrh aplikace v MVC 26
Obrázek 9: Aktivita, fiktivní aktivita a nápověda 28
Obrázek 10: Stav a stav po analýze systému 29
Obrázek 11: Systém v prostředí vytvořené aplikace 29
Obrázek 12: Zobrazení paralelních hran v grafu 30
Obrázek 13: Adresář s vytvořenými projekty 33
Obrázek 14: Formuláře pro přidání stavu a činnosti 34
Obrázek 15: Chybové hlášení formulářů 35
Obrázek 16: Formulář pro nastavení výpočtu a chybové hlášení 39
Obrázek 17: Vykreslení kritické cesty v prostředí aplikace 42
Obrázek 18: Zobrazení činností v tabulce výsledků 44
Obrázek 19: Vstupní formulář simulace Monte Carlo v aplikaci 46
Obrázek 20: Graf zobrazující průběh simulace Monte Carlo 47
Obrázek 21: Vstupní formulář pro generátor projektu 52

Seznam tabulek

Tabulka 1: Projekt definovaný tabulkou 11
Tabulka 2: Výsledky výpočtu metody PERT 22
Tabulka 3: Porovnání vytvořené aplikace s komerčními softwary 58
Tabulka 4: Výsledky testovacího příkladu ve vytvořené aplikaci 59
Tabulka 5: Porovnání výsledných hodnot metody PERT 60

Seznam zkratek a použitého značení

CPM	Critical Path Method
y_{ij}	délka trvání činnosti
ES	nejdříve možný začátek činnosti (Earliest Start Time)
LS	nejpozději přípustný začátek činnosti (Latest Start Time)
EF	nejdříve možný konec činnosti (Earliest Finish Time)
LF	nejpozději přípustný konec činnosti (Latest Finish Time)
PERT	Project Evaluation and Review Technique
a_{ij}	optimální odhad délky činnosti
b_{ij}	pesimistický odhad délky činnosti
m_{ij}	modální odhad délky činnosti
μ_{ij}	střední doba trvání činnosti
σ_{ij}	směrodatná odchylka střední doby trvání činnosti
σ_{ij}^2	rozptyl
HTML	Hypertext Markup Language
HTTP	Hyper Text Transfer Protocol
MVC	Model-View-Controller
NoSQL	nerelační databáze
JSON	JavaScript Object Notation
Blob	nestrukturovaný soubor binárních dat
SVG	Scalable Vector Graphics
PNG	Portable Network Graphics
UI	User Interface
URI	Uniform Resource Identifier („jednotný identifikátor zdroje“)
API	Application Programming Interface
DOM	Document Object Model

Úvod

Plánovat projekty ve firmě tak, aby se stihnuly v požadovaném termínu, může být obtížné. To samé platí pro plánování dílčích činností projektu. Zpoždění projektu může ve firmě způsobit finanční ztráty. Řízení (plánování) projektů označuje proces, ve kterém se koordinují jednotlivé činnosti projektů pro dodržení stanového termínu. Pro řízení projektů se využívá tzv. síťové analýzy, která hledá nejvhodnější uspořádání činností projektu tak, aby byl projekt stihnut v požadovaném termínu. Síťová analýza zkoumá časové rezervy dílčích činností projektu, kdy vhodným uspořádáním úkolů lze i docílit menších nákladů na realizaci.

Tato diplomová práce se zabývá metodami používanými v síťové analýze a to metodou kritické cesty a její stochastickou nadstavbou PERT. Motivací pro výběr tohoto tématu bylo vytvoření aplikace s grafickým rozhraním a výstupy pro nové uživatele v problematice síťové analýzy. Nejdříve jsou v první části práce teoreticky popsány obě metody. Je zde zmíněna práce s metodami, grafické objekty využité v grafu pro reprezentaci projektu a vstupní či výstupní hodnoty obou metod. Dále je v této části popsána simulační metoda Monte Carlo a její použití s metodou PERT. V druhé části práce jsou rozebrány existující komerční softwary a knihovny, které se zabývají problematikou metod používaných v síťové analýze.

Ve třetí části je detailně popsána vytvořená aplikace, která se zabývá metodou kritické cesty a PERT. Je zde popsána její struktura, algoritmus při tvorbě grafu reprezentujícího projekt, vstupní a výstupní hodnoty, výpočty metod a simulace Monte Carlo. Dále jsou zde popsány další funkce aplikace. Na závěr této části je celkové srovnání vytvořené aplikace s komerčními softwary, které se zabývají stejnou problematikou.

V závěrečné části této práce se nachází ověření správnosti výpočtu výsledků ve vytvořené aplikaci, kdy jsou výsledky z aplikace porovnány s analyticky vypočítanými výsledky a s výsledky z komerčních softwarů.

1 Metody užívané v síťové analýze

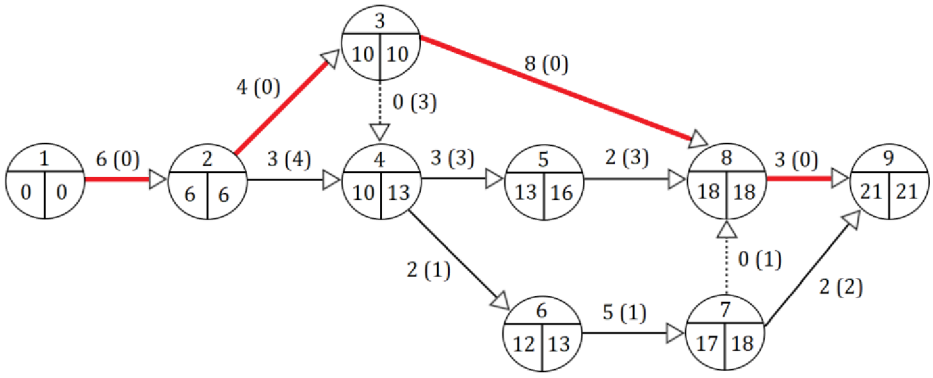
V následující části jsou popsány metody užívané v síťové analýze. Jedná se o metodu kritické cesty a její stochastickou nadstavbu PERT. Následuje popis metody Monte Carlo a jejího použití s metodou PERT.

1.1 Metoda kritické cesty (CPM – Critical Path Method)

Metoda kritické cesty je deterministická metoda, u které se předpokládá pevně daná délka dílčích činností y_{ij}. Změny délek činností jsou možné, je ale potřeba přepočítat celý projekt. Do metody kritické cesty vstupují následující parametry:

- nejdříve možný začátek činnosti – činnost nelze začít dříve, dokud neskončí všechny předcházející činnosti (anglicky Earliest Start Time – ES),
- nejdříve možný konec činnosti – lze dopočítat sečtením nejdříve možného začátku činnosti a doby trvání činnosti (anglicky Earliest Finish Time – EF),
- nejpozději přípustný konec činnosti – okamžik, ve kterém se musí činnost ukončit, aby nedošlo k zpoždění (anglicky Latest Finish Time – LF),
- nejpozději přípustný začátek činnosti – lze dopočítat odečtením nejpozději přípustného konce činnosti a doby trvání činnosti (anglicky Latest Start Time – LS).

Graf metody kritické cesty (viz Obrázek 1) obsahuje ohodnocené hrany, které značí konkrétní činnosti projektu. Ohodnocení hrany označuje délku činnosti. Vrchol grafu označuje stav, ve kterém se projekt nachází. Vstupní vrchol představuje začátek projektu. Koncový vrchol značí ukončení projektu. Do grafu metody kritické cesty se nejčastěji uvádí nejdříve možný začátek činnosti (ES) a nejpozději přípustný začátek činnosti (LS), ze kterých lze kritickou cestu nalézt.



Obrázek 1: Graf projektu po proběhlé analýze CPM[1]

[1] Časové rezervy činností jsou umístěny za ohodnocením hrany v závorkách. Výsledná kritická cesta je vyznačena červeně.

Při přidávání nových hran (činností) do grafu (projektu) mohou nastat následující situace:

- činnosti předchází pouze jedna činnost,
- činnosti předchází více činností.

Každá činnost může mít pouze jednu hranu. Pokud činnosti předchází pouze jedna činnost, tak se ke stavu, kam vstupuje předcházející činnost, přidá další ohodnocená hrana s novou činností. Pokud činnosti předchází více činností, využívá se tzv. fiktivních hran. Fiktivní hrana je hrana, která je ohodnocená nulovým časem. V grafu se fiktivní hrana značí přerušovaně. Při vkládání fiktivních hran může dojít k problému dvou různých směrů fiktivní hrany. Pokud prochází fiktivní hrana obousměrně, je možné dílčí stavy sloučit. [1]

Projekt může být definován také pomocí tabulky (viz Tabulka 1), ve které jsou zapsány dílčí činnosti projektu. Každá činnost má své označení, popis, označení přechozích činností a dobu trvání.

Tabulka 1: Projekt definovaný tabulkou

činnost	popis činnosti	předchozí činnosti	doba trvání (hodiny)
A	start projektu	-	0
B	nákup materiálu	A	10
C	návrh zpracování	A	50
D	zhotovení výrobku	B, C	30
E	dokončení projektu	D	0

Princip nalezení kritické cesty

Hledání kritické cesty spočívá ve výpočtu jednotlivých parametrů (časů). Výpočet jednotlivých časů probíhá ve dvou fázích:

- dopředný výpočet nejdříve možných začátků činnosti,
- zpětný výpočet nejpozději přípustných začátků činnosti.

Při dopředném výpočtu se nejdříve možný začátek činností začínajících v konkrétním stavu rovná maximu z nejdříve možných konců činností do konkrétního stavu vstupujících. Nalezení maximálně ohodnocené hrany (činnosti) je založeno na algoritmu průchodu do šířky. Výsledné maximum lze určit pouze tehdy, jsou-li známy všechny časy hran do stavu vstupujících.

Při zpětném výpočtu se postupuje inverzně od koncového vrcholu. Při průchodu grafem do šířky se hledají minima, která se od předchozího stavu odečítají. Pokud stav obsahuje obě hledané hodnoty, lze určit tzv. časovou rezervu.

Časová rezerva (v angličtině slack) vznikne odečtením časů a musí být vždy nezáporná. Odečítá se nejdříve možný začátek činnosti (ES) od nejpozději přípustného začátku činnosti (LS). Pokud časová rezerva je rovna 0, jedná se o stav, který je součástí kritické cesty.

Kritická cesta je cesta tvořená stavy a hranami od vstupu do výstupu, kde platí, že oba časy se rovnají (ES a LS) a po jejich odečtení vzniká časová rezerva rovna 0. Je to souhrn činností, kde dojde vlivem opoždění činnosti ke zpoždění celého projektu. Každý graf má minimálně jednu kritickou cestu. [1]

1.2 PERT (Project Evaluation and Review Technique)

Metoda PERT byla navržena jako pravděpodobnostní nadstavba metody kritické cesty. U metody kritické cesty se předpokládá, že činnosti mají pevně danou délku trvání. V praxi se tento předpoklad dá těžko splnit. V metodě PERT se předpokládá, že doba trvání každé činnosti je náhodnou veličinou, která je definována na nějakém intervalu $<a_{ij}, b_{ij}>$. Veličina a_{ij} představuje předpokládanou nejkratší možnou dobu trvání činnosti (tzv. optimistický odhad) a veličina b_{ij} je předpokládaná nejdelší doba provedení této činnosti (tzv. pesimistický odhad). Skutečná doba trvání činnosti se nachází někde uvnitř intervalu $<a_{ij}, b_{ij}>$. V metodě PERT se dále předpokládá, že lze pro jednotlivé činnosti určit jejich nejpravděpodobnější dobu trvání činnosti m_{ij} (tzv. modální odhad). V každém projektu je nutné pro činnost stanovit odhady a_{ij}, b_{ij}, m_{ij}. [1]

Výpočet metody PERT

Doba trvání činnosti je spojitá náhodná veličina, u které pravděpodobnostní rozdělení není předem známé. Toto neznámé rozdělení lze aproximovat pomocí β-rozdělení. Střední hodnota μ_{ij} β-rozdělení se vypočítá podle následujícího vzorce:

$$\mu_{ij} = \frac{a_{ij} + 4m_{ij} + b_{ij}}{6} \tag{1}$$

Směrodatná odchylka σ_{ij} β-rozdělení je stanovena podle vzorce:

$$\sigma_{ij} = \frac{b_{ij} - a_{ij}}{6} \tag{2}$$

Při výpočtu kritické cesty metodou PERT se algoritmus liší od metody CPM tím, že místo pevně daných délek trvání činností y_{ij} se pracuje se středními hodnotami dob trvání činností

μ_{ij}. Výsledkem je kritická cesta, která je ohodnocená hodnotou M, která vznikla součtem středních dob trvání kritických činností projektu. Hodnota M udává střední dobu trvání celého projektu. Skutečná doba trvání celého projektu T je spojitá náhodná veličina, u které je střední hodnota M a rozptyl σ^2 lze určit jako součet rozptylů všech činností na kritické cestě. Výsledná pravděpodobnost, zda projekt bude ukončen ve stanoveném čase T se aproximuje distribuční funkcí normálního rozdělení $N(M, \sigma)$ v bodě $t = T$. V praxi se hledá v tabulkách hodnota $N(0,1)$ v transformovaném bodě z, který je dán vzorcem:

$$z = \frac{T_s - M}{\sigma} \tag{3}$$

Úpravou výše uvedeného vzorce lze určit, v jakém čase T_s bude projekt ukončen se stanovenou pravděpodobností:

$$T_s = z\sigma + M \tag{4}$$

1.3 Monte Carlo

Metoda Monte Carlo představuje souhrnné označení pro stochastické metody sloužící k řešení různých matematických úloh (např. integrace, matematická statistika, modelování systémů a fyzikálních procesů, aj.). V těchto aplikacích je metoda využívána především k simulaci (modelování) distribuce náhodné veličiny X s použitím náhodně generovaných čísel z určitého statistického rozdělení (generování tzv. pseudonáhodných čísel).

Průběh metody

Průběh metody spočívá ve vkládání náhodně generovaných hodnot z určitého statistického rozdělení do funkce náhodné veličiny X a zaznamenávání výstupních hodnot. Následně je tento postup aplikován znovu s dostatečným počtem opakování pro lepší přesnost simulace.

Výsledkem simulace (modelování) je model dat, ze kterého lze získat odhady statistických údajů o náhodné veličině X, jako je například její střední hodnota $E(X)$ nebo směrodatná odchylka σ. Nevýhodou metody Monte Carlo může být malá přesnost. S větším počtem opakování algoritmu lze docílit přesnějšího modelu dat. Pokud se zvýší počet opakování n^2-krát, zvýší se n-krát přesnost. Přesnost metody je dále ovlivněna dalšími okolnostmi, jako je například kvalita generátoru náhodných čísel určitého statistického rozdělení nebo výpočetní náročnost algoritmu použitého v simulaci. [2]

Typickou úlohou metody Monte Carlo je simulování náhodné veličiny X pomocí náhodných čísel z veličiny X a zjištění parametrů jako je například střední hodnota, rozptyl, aj.

Při simulování náhodné veličiny X s n nezávislými realizacemi X_1, X_2, …, X_n lze neznámou střední hodnotu $E(X)$ odhadnout pomocí aritmetického průměru:

$$E(X) = \frac{1}{n}(X_1 + X_2 + \cdots + X_n) \tag{5}$$

Použití v metodě PERT

V metodě PERT je doba trvání činnosti spojitá náhodná veličina bez předem známého rozdělení. Toto neznámé rozdělení lze aproximovat pomocí β-rozdělení. β-rozdělení je spojité rozdělení pravděpodobnosti definované na intervalu $<0, 1>$, ve kterém vystupují kladné parametry α a β. Tyto parametry řídí tvar distribuce a jsou označovány jako exponenty náhodné veličiny X:

$$X \sim Beta(\alpha, \beta) \tag{6}$$

Činnost v metodě PERT je definována odhady délek trvání:

- a_{ij} – optimistický odhad délky trvání činnosti,
- m_{ij} – modální odhad délky trvání činnosti (nejpravděpodobnější),
- b_{ij} – pesimistický odhad délky trvání činnosti.

Pro odhady délky trvání činnosti platí, že optimistický odhad a_{ij} je menší nebo roven modálnímu odhadu m_{ij}, který je menší nebo roven pesimistickému odhadu b_{ij}. Transformací odhadů délek trvání činnosti jsou získány parametry β-rozdělení podle vzorce:

$$X \sim Beta\left(1 + \lambda\frac{m_{ij} - a_{ij}}{b_{ij} - a_{ij}}, 1 + \lambda\frac{b_{ij} - m_{ij}}{b_{ij} - a_{ij}}\right) \tag{7}$$

Veličina λ je tradičně zadána jako konstanta rovna číslu 4. Hodnota délky trvání činnosti je poté generována jako $a_{ij} + X \cdot (b_{ij} - a_{ij})$, kde X představuje náhodnou hodnotu náhodné veličiny X z β-rozdělení. [2] [3] [4]

Vygenerované hodnoty jsou z intervalu $<a_{ij}, b_{ij}>$. Pravděpodobnost, že projekt ukončí svou činnost v krajních mezích intervalu, je nulová. Daleko pravděpodobnější je, že projekt ukončí činnost v hodnotě (době), která se nachází uvnitř tohoto intervalu.

2 Rešerše dostupných aplikací a knihoven

V následující části této práce jsou popsány softwary a knihovny, které se zabývají problematikou spojenou s plánováním projektů. Dále jsou porovnány statistické knihovny pro výpočet metody PERT. Rešerše dostupných softwarů a knihoven byla nezbytnou součástí pro tvorbu vlastní aplikace. Porovnání vytvořené aplikace s komerčními softwary je dále popsáno v kapitole 3.10.

Pro rešerši dostupných softwarů byly vybrány následující komerční aplikace:

- Lucidchart (https://www.lucidchart.com/),
- Microsoft Project (https://project.microsoft.com/),
- PERT Calculator (https://goodcalculators.com/pert-calculator/).

V následující části je popsána webová aplikace Lucidchart, její rozhraní a hodnocení.

2.1 Lucidchart

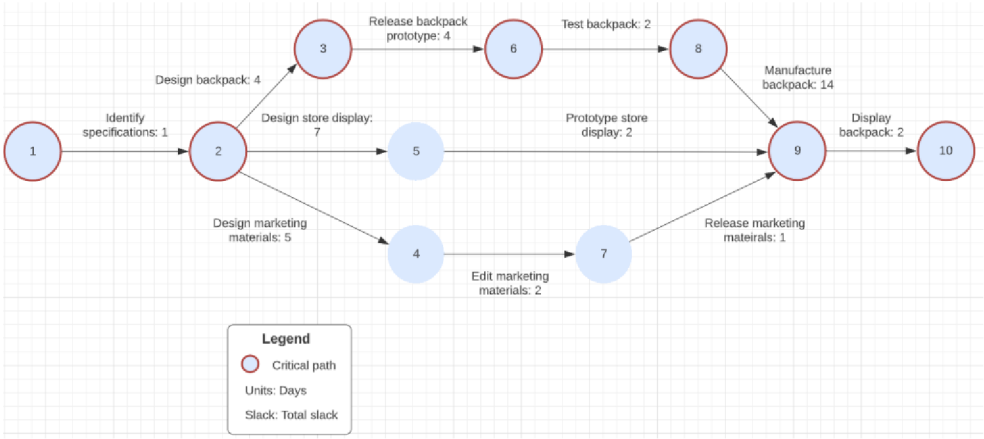
Lucidchart je webová aplikace od společnosti Lucid Software Inc., která umožňuje tvorbu a sdílení digramů mezi uživateli. Pro tvorbu diagramů lze využít předpřipravených šablon diagramů, mezi které například patří vývojové, stromové, systémové nebo Ganttovy diagramy. Aplikace Lucidchart nabízí také šablony pro diagram metody PERT. [5]

Rozhraní aplikace

Po přihlášení do aplikace se uživateli zobrazí prostředí aplikace s adresářem projektů. Hlavní nabídka aplikace se nachází ve sloupci vlevo vedle adresáře. Pro vytvoření nového diagramu musí uživatel zvolit položku *New* a z nabídky vybrat *Lucidchart*. Zde může vytvořit prázdný diagram pomocí položky *Blank Document* nebo vytvořit diagram z předpřipravené šablony pomocí položky *Create from Template*. Při výběru položky *Create from Template* se zobrazí výběr předdefinovaných šablon diagramů. Šablona pro metodu PERT je definována buď jako základní struktura diagramu, která obsahuje pár zobrazených uzlů a hran, anebo jako ukázkový příklad projektu metody PERT, ve kterém jsou popsány všechny uzly a hrany (viz Obrázek 2).

Po výběru šablony se aplikace přesměruje do prostředí diagramu, ve kterém se nové objekty do diagramu přidávají pomocí levé postranní lišty. V horní části této lišty se nachází objekty, které jsou v diagramu použity. Pro metodu PERT se jedná o uzly a legendu diagramu. Uzel diagramu má světle modré pozadí s červeným ohraničením. Hrana diagramu lze přidat výběrem objektu *Line* z postranní lišty. Vytvořený diagram může uživatel exportovat

například do formátu PDF, PNG, SVG nebo JPEG. Dále může vytvořený diagram převést do nové šablony, kterou může sdílet dalším uživatelům aplikace.



Obrázek 2: Diagram vytvořený v aplikaci Lucidchart

Hodnocení

Výhodou této aplikace je, že se dá využívat bezplatně a online. Pokud si uživatel předplatí funkce navíc, má přístup k předpřipraveným šablonám diagramů, které usnadňují práci při tvorbě diagramu. Další výhodou je sdílení diagramu mezi uživateli nebo možnost spolupráce uživatelů na jednom diagramu.

Mezi nevýhody aplikace patří složité prostředí pro vyhledávání předpřipravené šablony diagramu. Další možnou nevýhodou může pro uživatele být zjištění kritické cesty obsáhlejšího projektu, kdy aplikace nabízí pouze zobrazení diagramu. Konkrétními výpočty se nezabývá a výslednou kritickou cestu v diagramu musí vykreslit sám uživatel.

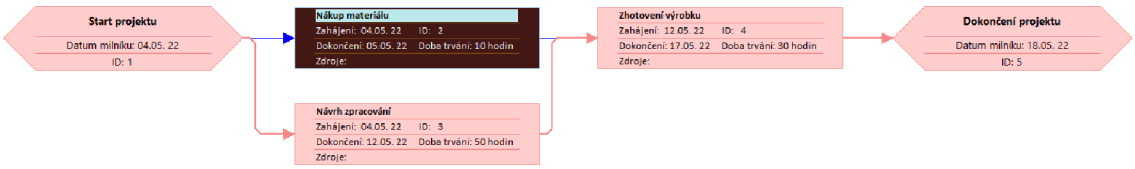
2.2 Microsoft Project

Microsoft Project je aplikace od společnosti Microsoft určená pro správu a řízení projektů. Aplikace má desktopovou i webovou variantu. Aplikace je zpoplatněná a uživatel si může vybrat mezi místním a cloudovým řešením. [6]

Rozhraní aplikace

Hlavní prostředí desktopové aplikace Microsoft Project tvoří adresář projektů. Nový projekt lze vytvořit kliknutím na tlačítko *Nový*, které otevře nabídku pro vytvoření nového prázdného nebo načtení již existujícího projektu. Po vytvoření projektu je uživatel aplikace přesměrován do hlavního pracovního prostředí projektu, ve kterém je zobrazena tabulka pro definici úkolů (činností) projektu současně s časovou osou projektu (Ganttův diagram). Úkoly projektu jsou v diagramu znázorněny jako pruhy, kdy délka pruhu označuje délku trvání úkolu. Po přidání nového úkolu (např. v tabulce) se v diagramu přidá nový pruh.

Hlavní nabídka aplikace obsahuje sekci *Formát Ganttového diagramu*, ve které může uživatel upravovat vlastnosti diagramu (např. barva pruhů). Dále lze zobrazit projekt do síťového diagramu (viz Obrázek 3) výběrem možnosti *Síťový diagram* z nabídky v sekci *Zobrazení*. Po výběru této možnosti se projekt zobrazí v síťovém diagramu s kritickou cestou vyznačenou červenou barvou.

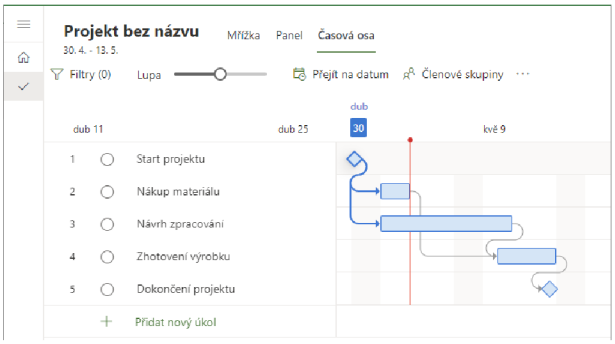


Obrázek 3: Síťový diagram v Microsoft Project

Webová aplikace Microsoft Project nabízí zjednodušené prostředí pro řízení projektů (viz Obrázek 4). Uživatel může projekt zobrazit ve třech variantách:

- Mřížka – výchozí varianta zobrazující úkoly projektu v tabulce,
- Panel – varianta zobrazující úkoly v kontejnerech,
- Časová osa – zobrazení úkolů v časové ose (Ganttův diagram).

Webová varianta nenabízí jako desktopová aplikace změnu zobrazení diagramu. Projekt může být exportován do souboru formátu *XLS*. Dále aplikace umožňuje exportovat projekt v zobrazené variantě *Časová osa* do formátu *PDF*.



Obrázek 4: Prostředí webové aplikace Microsoft Project

Hodnocení

Hlavní výhodou aplikace je její propracovanost. Jedná se o aplikaci, kterou běžně firmy využívají při plánování projektů. Další výhodou je, že obsahuje webovou a desktopovou variantu, mezi kterými lze projekt libovolně přenášet. V desktopové variantě má uživatel aplikace více možností pro definování a zobrazení projektu a jeho dílčích úkolů.

Možnou nevýhodou aplikace může být pro uživatele její cena, pokud by aplikaci chtěl využívat pro soukromé účely. Nejlevnější licence nabízí pouze webovou variantu aplikace, která obsahuje méně funkcí než desktopová varianta. Aplikace je ale primárně určená pro firmy a správu velkých projektů, ve kterých plánování hraje důležitou roli a cena softwaru je minimální oproti možné ztrátě výdělku firmy způsobené zpožděním projektu.

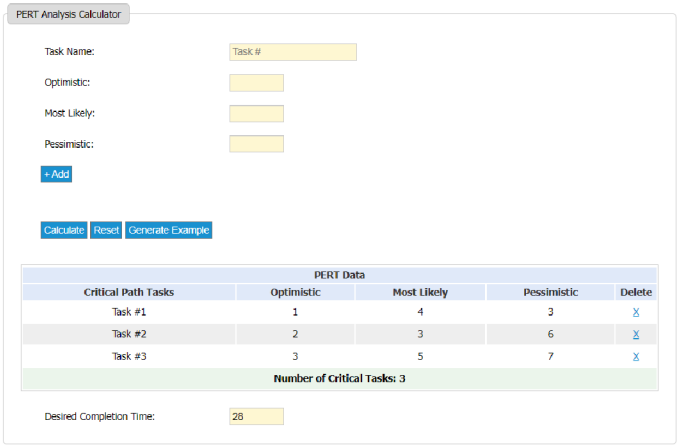
2.3 PERT Calculator

PERT Calculator je webová aplikace od společnosti Good Calculators, která nabízí zdarma pomocné aplikace a kalkulačky pro nejrůznější odvětví (statistika, finance, zdraví, aj). Aplikace PERT Calculator slouží k výpočtu metody PERT v zjednodušené grafické podobě. [7]

Rozhraní aplikace

Prostředí aplikace PERT Calculator se skládá z části pro zadávání vstupních činností projektu (viz Obrázek 5) a z části pro zobrazení výsledků metody PERT (viz Obrázek 6). Do aplikace se na vstupu zadávají činnosti projektu, které jsou součástí kritické cesty. Pro každou činnost je potřeba zadat optimistický, nejpravděpodobnější a pesimistický odhad délky činnosti. Název činnosti (položka *Task Name*) je volitelná. Přidané činnosti jsou zobrazené v tabulce a nelze je upravovat, ale pouze smazat.

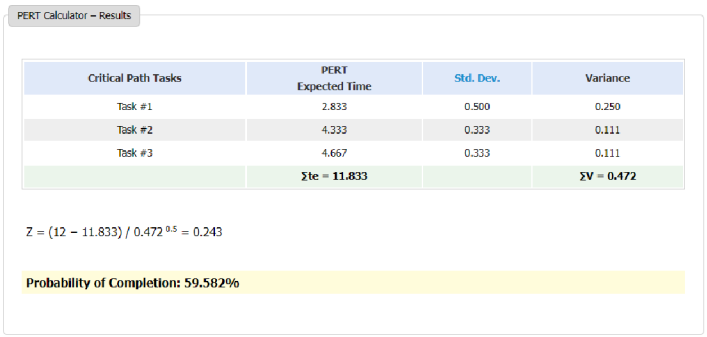
Pod formulářem pro přidání činnosti se nachází tlačítko *Reset* pro odstranění všech přidaných činností v tabulce, tlačítko *Generate Example*, který vygeneruje novou tabulku s náhodnými činnostmi a tlačítko *Calculate*, které spustí výpočet metody PERT. Pro výpočet je potřeba vyplnit požadovaný čas dokončení ve vstupu s názvem *Desired Completion Time*.



Obrázek 5: Prostředí aplikace PERT Calculator

Po proběhlém výpočtu se výsledky zobrazí v tabulce *Results* (viz Obrázek 6). V tabulce jsou vypsány pro jednotlivé činnosti jejich střední hodnoty délky trvání, směrodatné odchylky

a rozptyl. V posledním řádku tabulky je vypsán součet středních hodnot a rozptylů. Pod tabulkou s výsledky se nachází výpočet hodnoty *z* potřebné pro distribuční funkci normálního rozdělení. Pod tímto údajem se nachází vypočtená pravděpodobnost zaokrouhlená na tři desetinná místa.


PERT Calculator – Results

Critical Path Tasks	PERT Expected Time	Std. Dev.	Variance
Task #1	2.833	0.500	0.250
Task #2	4.333	0.333	0.111
Task #3	4.667	0.333	0.111
	∑te = 11.833		**∑V = 0.472**

$Z = (12 - 11.833) / 0.472^{0.5} = 0.243$

Probability of Completion: 59.582%

Obrázek 6: Zobrazení výsledků aplikace PERT Calculator

Hodnocení

Výhodou této aplikace je bezesporu, že je bezplatná a online. Nabízí jednoduché uživatelské prostředí, které obsahuje i teorii a vzorce týkající se dané problematiky.

Mezi nevýhody patří hlavně to, že uživatel zadává pouze činnosti, které jsou součástí kritické cesty projektu. Aplikace se tedy pro řízení projektů samotná použít nedá a je tedy spíš určená pro uživatele, kteří si kritickou cestu v zadaném projektu naleznou pomocí jiné aplikace a chtějí pouze zjistit pomocí metody PERT pravděpodobnost, s kterou projekt ukončí svou činnost do stanovaného času.

2.4 Knihovny zabývající se metodou CPM/PERT

V této podkapitole jsou popsány knihovny, které se zabývají problematikou metody kritické cesty nebo její stochastické nadstavby (PERT). Pro popis byly vybrány následující knihovny:

- CriticalPath (https://pypi.org/project/criticalpath/),
- AnyChart (https://www.anychart.com/).

V následující části je popsána knihovna CriticalPath.

CriticalPath

CriticalPath je knihovna vytvořená pro programovací jazyk Python, která slouží ke zjištění kritické cesty v projektu. Knihovna obsahuje hlavní objekt *Node* (uzel), který slouží k definici projektu a zároveň jeho stavů.

Projekt je definován vytvořením hlavního uzlu, který představuje celý projekt. Dále se hlavnímu uzlu přidávají další uzly, ve kterých se definuje *duration* označující délku trvání.

Po přidání všech uzlů hlavními uzlu se jednotlivé uzly hlavního uzlu propojí pomocí funkce *link()*, do které jako parametry přichází název proměnné počátečního a koncového uzlu. Knihovna nabízí funkci pro nalezení kritické cesty, kdy danou cestu sestavenou z názvu uzlů vypíše do konzole. Dále obsahuje funkce pro zjištění celkové doby trvání projektu nebo nalezení parametrů *ES* a *LS*.

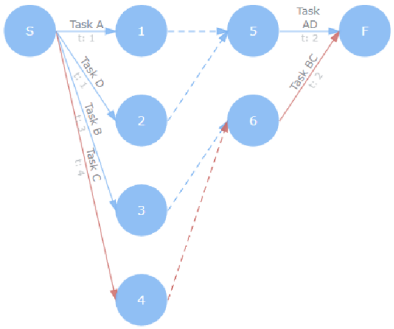
Hodnocení

Hlavní výhodou knihovny je, že se dá snadno použít. Definování projektu je jednoduché a pro základní uvedení uživatele do problematiky dostačující.

Nevýhodou může být v případě obsáhlejších projektů absence grafického rozhraní, kdy by uživatel mohl snadno udělat chybu při propojování uzlů z důvodu obsáhlosti projektu.

AnyChart

AnyChart je JavaScriptová knihovna, která slouží k tvorbě interaktivních grafů pro webové, desktopové a mobilní aplikace. Knihovna obsahuje předdefinované šablony pro základní spojnicové nebo sloupcové grafy. Obsahuje také ale šablony pro diagramy užívané při řízení projektů, Ganttův diagram a diagram metody PERT (viz Obrázek 7). [8]



Obrázek 7: Diagram metody PERT knihovny AnyChart

Údaje o činnostech projektu (hran diagramu) jsou uloženy v poli objektů typu JSON. Objekt typu JSON (JavaScript Object Notation) obsahuje parametry uložené jako klíč a hodnota. Každá hrana obsahuje ID, délku trvání a název. Pokud závisí na předchozí činnosti, obsahuje dále klíč *dependsOn*, v jehož hodnotě se definují ID hran, které této činnosti předchází. Sestavením diagramu se automaticky vykreslí kritická cesta, která je vyznačena červenou barvou hran diagramu.

Hodnocení

Hlavní výhodou knihovny AnyChart je, že obsahuje šablony konkrétních grafů. Navíc knihovna má již implementované funkce pro export nebo tisk grafu. Další výhodou je,

že na stránkách knihovny je testovací prostředí ke každé šabloně, kdy si ukázkový příklad šablony může uživatel knihovny upravit nebo vytvořit zcela nový.

Nevýhodou knihovny je u diagramu typu PERT, že je diagram statický. S uzly, které se automaticky vytvořily, nelze pohybovat a v případě obsáhlejšího projektu může nastat problém se zobrazením.

2.5 Statistické knihovny

V této podkapitole jsou popsány a porovnány statistické knihovny, které obsahují moduly použitelné v síťové analýzy při výpočtu metody PERT. Pro porovnání byly vybrány následující knihovny:

- SciPy (https://www.scipy.org/),
- jStat (https://jstat.github.io/all.html).

V následující části jsou popsány knihovny SciPy a jStat. Dále je zde popsán způsob testování knihoven, porovnání výsledných hodnot a hodnocení.

SciPy

SciPy je bezplatná a open source vědecká knihovna Pythonu využívaná pro matematické výpočty. Obsahuje moduly pro statistiku, lineární algebru, zpracování signálů a obrazů, aj. Mimo jiné obsahuje statistický modul s názvem *norm* pro normální rozdělení, který se nachází v sekci *stats* (sekce s moduly využívanými ve statistice) knihovny SciPy. Tento modul obsahuje funkce potřebné pro výpočet metody PERT a to následující:

- stats.norm.cdf(x) – funkce, která vrací hodnotu distribuční funkce normálního rozdělení v bodě x (hodnota *z*),
- stats.norm.ppf(q) – inverzní funkce, která vrací na základě vstupní požadované pravděpodobnosti hodnotu *z*.

jStat

jStat je statistická knihovna napsaná v JavaScriptu obsahující moduly určené pouze pro statistiku. Modul s normálním rozdělením se zde nachází pod názvem *normal.* Tento modul obsahuje funkce potřebné pro výpočet metody PERT a to následující:

- jStat.normal.cdf(x,mean,std) – zjištění hodnoty distribuční funkce normálního rozdělení v bodě x (požadovaný čas) ze vstupních hodnot (požadovaný čas, střední hodnota, směrodatná odchylka),

- jStat.normal.inv(p,mean,std) – inverzní funkce, kde se na základě požadované vstupní pravděpodobnosti získá výsledný čas, ve kterém projekt bude ukončen se zadanou pravděpodobností ze vstupních hodnot (požadovaná pravděpodobnost, střední hodnota, směrodatná odchylka).

Způsob testování knihoven

Pro otestování obou knihoven byla vytvořena aplikace obdobná komerční aplikaci PERT Calculator. Aplikace (viz Příloha A) byla vytvořena v programovacím jazyce Python a jeho webovém frameworku Flask. Aplikace od uživatele na vstupu přijímá činnosti, které jsou součástí kritické cesty. Pro každou činnost se musí definovat název a odhady délky činnosti. Přidané činnosti jsou zobrazeny v tabulce. Pro spuštění výpočtu metody PERT musí být ještě zadán požadovaný čas dokončení projektu v kolonce *Desired Completion Time*. Po stisknutí tlačítka *Calculate* se provede výpočet a dopočítané hodnoty se zobrazí v nově vzniklé tabulce. Na konci tabulky jsou tři řádky obsahující souhrn výpočtu metody PERT. V prvním řádku se nachází zadaný čas dokončení a zjištěná hodnota *z*. Druhý a třetí řádek obsahují dopočítanou pravděpodobnost pomocí knihovny SciPy a jStat.

Porovnání výsledných hodnot

K porovnání výsledných hodnot byl do aplikace zadán projekt obsahující čtyři kritické činnosti (viz Příloha A). Požadovaný čas dokončení a výsledky metody PERT jsou shrnuty v tabulce (viz Tabulka 2). Výsledné hodnoty jsou zaokrouhleny na tři desetinná místa.

Tabulka 2: Výsledky výpočtu metody PERT

Požadovaný čas dokončení projektu	28
Celková střední hodnota projektu	26,834
Celkový rozptyl projektu	5,194
Hodnota z	0,512
Pravděpodobnost (SciPy)	69,567 %
Pravděpodobnost (jStat)	69,564 %

Hodnocení

Obě knihovny nabízí široké možnosti užití. Zatímco knihovna jStat se zabývá pouze statistikou, knihovna SciPy obsahuje mnoho dalších modulů pro různé vědecké disciplíny. Při testování knihoven byly použity moduly pro normální rozdělení, kdy se výsledek obou knihoven lišil na třetím desetinném místě.

3 Aplikace

V této části práce je popsána vytvořená aplikace (https://naa-app.herokuapp.com/). Nejdříve jsou zde rozepsány informace o aplikaci a použité technologie. Následuje popis struktury aplikace, použitých grafický objektů, zobrazení diagramu, výpočtů, nastavení, popis dodatečných funkcí aplikace a zabezpečení aplikace. V poslední části této kapitoly je aplikace porovnána s komerčními softwary.

3.1 O aplikaci

Vytvořená aplikace byla implementována jako webová pomocí značkovacího jazyka HTML, skriptovacího jazyka JavaScript, technologie Node.js (https://nodejs.org/) a webového aplikačního frameworku Express.js (https://expressjs.com/). Hosting aplikace na webu zajišťuje cloudová platforma Heroku (https://devcenter.heroku.com/).

K perzistentnímu uložení dat byla použita multiplatformní dokumentová databáze MongoDB (https://www.mongodb.com/), která spadá do NoSQL databází, ve kterých jsou data ukládána do dokumentů ve formátu JSON (nebo BSON). Dále se pro dočasné ukládání dat využívá sessionStorage a cookies. Uložiště sessionStorage uchovává data pouze v rámci jedné platné session. Data v sessionStorage jsou uloženy jako název proměnné a hodnota. V aplikaci se do sessionStorage vkládají především výsledky proběhlé analýzy pro následné zobrazení výsledků (dále Kapitola 3.5). Uložiště cookies obsahuje malé množství dat, které server zašle klientovi (prohlížeči), který je má uložené v paměti. Cookies byly vytvořeny jako řešení problému s pamatováním informací o uživateli. Ve vytvořené aplikaci se cookies využívají k ukládání dat o nastavení aplikace pro konkrétního uživatele (dále Kapitola 3.7). [9]

Vzhled aplikace je zajištěn pomocí kaskádových stylů (CSS) a front-end frameworku Bootstrap (https://getbootstrap.com/), který je open-source a obsahuje sadu předloh a komponent pro snadnější definici webové stránky a hlavně jejího vzhledu. Tyto definované předlohy se předávají HTML elementu v jeho třídě (atribut *class*). V aplikaci je Bootstrap používán zejména pro navigační lišty, formuláře, tabulky a modální okna s formulářem nebo zobrazením nápovědy. Pro ikony, využívané v ovládacích prvcích aplikace, byla použita knihovna Font Awesome (https://fontawesome.com/), která obsahuje sadu fontů a ikon. V bezplatné verzi je sada ikon omezená, ale pro základní značení ovládacích prvků je dostačující.

Handlebars

Handlebars (https://handlebarsjs.com/) je šablonovací systém, který umožňuje tvorbu šablon a opakovaně použitelných částí (komponent) HTML kódu pro jednodušší implementaci webových stránek. V aplikaci je vytvořen základní layout s názvem *main*, ve kterém je definována základní struktura HTML stránky. V HTML elementech *<head>* a *<body>* jsou skripty, které obsahují použité JavaScriptové knihovny společné pro všechny stránky webové aplikace. Dále v elementu *<head>* je přidán odkaz na soubor *styles.css* s použitými kaskádovými styly a odkaz na ikonu aplikace, která se zobrazuje u titulku webové stránky. Všechny webové stránky dědí tento základní layout. V každé webové stránce je pak dodatečně definován jen vlastní obsah HTML elementu *<body>* (těla), název titulku *<title>* a skripty, které jsou použity na dané stránce (inicializace grafu, simulace, aj.).

Dále byly v aplikaci vytvořeny znovupoužitelné komponenty (tzv. partials) pro 3 navigační lišty hlavní nabídky aplikace. Společným prvkem všech tří navigačních lišt je ikona aplikace sloužící k přesměrování uživatele na úvodní (hlavní) stránku aplikace *projectsDirectory* se seznamem všech projektů. První lišta je použita v hlavním pracovním prostředí aplikace, ve kterém probíhá práce s diagramem (systémem) a obsahuje kompletní ovládání aplikace:

- Project – adresář projektů, nový projekt, editace, smazání, export vytvořeného projektu,
- Analysis – výpočet analýzy (CPM nebo PERT), zobrazení výsledků, simulace Monte Carlo,
- Settings – nastavení aplikace, manuál, nápověda, o aplikaci,
- Profile[2] – můj profil, odhlášení z aplikace.

Druhá lišta je použita ve všech ostatních stránkách webové aplikace, ve kterých musí být uživatel přihlášen a na rozdíl od první lišty neobsahuje položky *Project* a *Analysis*. Poslední lišta obsahuje pouze ikonu určenou k přesměrování a je zobrazena na stránkách s přihlášením existujícího nebo registrací nového uživatele.

Handlebars umožňují také tvorbu speciálních funkcí (tzv. helpers), které jsou definované v serverové části aplikace a doplňují funkcionalitu na klientské části. V aplikaci byly takto definované například funkce pro překlad nebo porovnání dvou hodnot v jazyce HTML. [10]

[2] Položka Profile je v aplikaci pojmenována jako kombinace jména a příjmení uživatele.

3.2 Struktura vytvořené aplikace

V této podkapitole je nejdříve popsán návrhový vzor Model View Controller, který byl použit při návrhu vytvořené aplikace (viz Obrázek 8). Následuje popis HTTP komunikace, jejího úskalí ve formulářích HTML a způsobu řešení v aplikaci.

Model View Controller

Pro aplikaci byl využit architektonický návrhový vzor Model-view-controller (zkráceně MVC), který patří k nejčastěji používaným architektonickým vzorům ve webových aplikacích. Základní ideou vzoru MVC je oddělení aplikační logiky od výstupu aplikace, kdy tímto způsobem lze předejít problému tzv. špagetového kódu, kdy se právě aplikační logika a renderování výstupu nachází v jednom souboru (popř. třídě). [11]

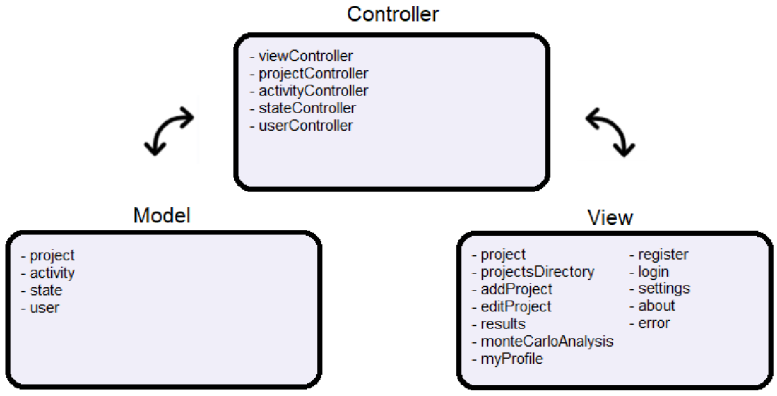
Vzor MVC se skládá ze tří funkčních logických komponent:

- Model – definice datových struktur objektů a datová logika aplikace,
- View – tzv. „pohled“ prezentující data uživateli,
- Controller – tzv. „prostředník“, se kterým komunikují komponenty Model a View.

Pro tvorbu modelů datových struktur byl ve vytvořené aplikaci použit nástroj Mongoose (https://www.npmjs.com/package/mongoose), který slouží k tvorbě objektových schémat (modelů) používaných v dokumentové databázi MongoDB, jejich validaci a zajištění přístupu k nim přes databázové dotazy. Ve vytvořené aplikaci byly definovány následující schémata (modely):

- Project – ID, název, typ, informace o projektu, datum vytvoření, datum poslední úpravy, ID uživatele,
- Activity – ID, název, typ činnosti (normální nebo fiktivní), ID počátečního stavu, ID konečného stavu, jednotka času, hodnoty, popis činnosti, ID projektu,
- State – ID, název, ID projektu, popis stavu,
- User – ID, jméno, příjmení, emailová adresa, heslo, datum vytvoření účtu.

V schématech *State* a *Activity* byl dodatečně implementován index k zajištění unikátnosti stavu nebo činnosti v rámci vytvořeného projektu. V obou schématech se jedná o unikátní kombinaci názvu stavu (*stateName*), anebo činnosti (*activityName*) a identifikátoru projektu (*projectID*).



Obrázek 8: Návrh aplikace v MVC

V komunikaci mezi webovou aplikací a kontrolorem byl použit tzv. router, který přijímá požadavek (anglicky request) z URL (konkrétní cesta v aplikaci), ten zpracuje a určí podle parametrů, která funkce kontroleru má být použita pro obsluhu. Funkce kontroleru podle parametrů a vstupních dat (např. z formuláře) zavolá daný model kvůli datům. Data buď načítá z databáze, aktualizuje, maže nebo ukládá nová do databáze. Po zpracování v obsluhující funkci, jsou data (pokud nebyla smazána) vyslána zpět do připravené šablony v komponentě View (renderování stránky s daty). V případě výskytu chyby při práci s daty a databází, musí být obslužná funkce v kontroleru ošetřena. Ve vytvořené aplikaci je ošetření řešeno pomocí bloku *try-catch*, kdy je při výskytu chyby v kódu ohraničeném blokem *try* automaticky volán kód v bloku *catch*, ve kterém je buď aplikace přesměrována na stránku zobrazující obsah konkrétní chyby (stránka *error*) nebo je chyba zobrazena jako zpráva ve stránce, kde se chyba vyskytla. Pro komunikaci mezi daty a pohledy bylo definováno v aplikaci pět kontrolerů:

- userController – registrace nového uživatele, přihlášení, odhlášení, zobrazení a změna údajů účtu,
- projectController – zobrazení adresáře projektů, vytvoření projektu, zobrazení projektu, aktualizace, smazání, generování náhodného projektu, načtení projektu ze souboru,
- activityController – přidání činnosti, aktualizace, smazání,
- stateController – přidání stavu, aktualizace, smazání,
- viewController – zobrazení informací o aplikaci, manuálu a nastavení aplikace.

HTTP komunikace

HTTP (Hypertext Transfer Protocol) je internetový protokol zajišťující komunikaci na webu mezi klientem (např. internetový prohlížeč) a serverem. Samotná komunikace funguje na principu žádosti a odpovědi (HTTP request a HTTP response). Klient zašle žádost

na server, server danou žádost obdrží, vyhodnotí, zpracuje a zašle odpověď, kterou následně klient obdrží. [12]

V protokolu HTTP je komunikace mezi klientskou a serverovou částí aplikace zajištěna pomocí HTTP metod. Ve vytvořené aplikaci se využívají následující metody protokolu HTTP:

- GET – žádost o data,
- POST – zaslání dat na server,
- PUT – zaslání dat na server (aktualizace dat),
- DELETE – smazání vybraných dat.

Rozdíl mezi metodami POST a PUT je v tom, že metoda PUT je tzv. idempotentní. Opakované volání stejného požadavku metody PUT vždy vrátí stejný výsledek. Naopak opakovaným voláním metody POST mohou s výsledkem vzniknout vedlejší účinky (vytvoření duplicitních dat). Pro aktualizaci dat se volí metoda PUT. [13]

V aplikaci se nová data vytváří pomocí formulářů (HTML element *<form>*), které podporují pouze metody GET a POST. V případě použití dalších metod je potřeba žádost před odesláním na server upravit. V aplikaci byly použity formuláře pro přidání nových dat, úpravu a smazání vytvořených dat. Pro úpravu a smazání dat (metoda PUT a DELETE) byl v aplikaci použit modul method-override (https://www.npmjs.com/package/method-override), který dokáže upravit žádost před zasláním na server. Úprava žádosti probíhá pomocí textového řetězce, který je přidán do atributu *action* (akce formuláře) elementu *<form>*. Textový řetězec je ve tvaru *_method=* a názvu HTTP metody, na kterou se má akce formuláře přepsat (PUT nebo DELETE). Výsledný doplňkový řetězec atributu *action* je tvaru */?_method=DELETE* pro metodu DELETE a */?_method=PUT* pro metodu PUT protokolu HTTP. Po odeslání formuláře je následně tato hodnota zpracována a v routeru je správně zvolena příslušná cesta pro obsloužení požadavku. [13] [14]

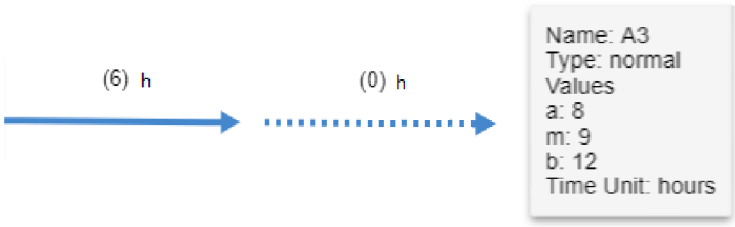
3.3 Grafické objekty v aplikaci

V aplikaci jsou definovány dva různé grafické objekty, se kterými se dá v analýze pracovat. Jedná se o aktivitu a stav. V následující části jsou tyto objekty popsány.

Aktivita

Aktivita (činnost) je reprezentována jako ohodnocená hrana grafu, která propojuje dva stavy (vrcholy grafu). Ohodnocení je dáno typem analýzy. Pokud se projekt zabývá pouze metodou kritické cesty, ohodnocení hrany představuje délku trvání činnosti y_{ij}. U metody PERT se jedná o odhady délky činnosti a_{ij} (optimistický), m_{ij} (modální) a b_{ij}

(pesimistický). Popis hrany je dán závorkou, ve které je zapsána délka činnosti nebo odhady délky činnosti. Za závorkou se nachází informace o jednotce času, ve které bylo ohodnocení zadáno (viz Obrázek 9). V grafu lze pracovat buď s normální aktivitou, nebo fiktivní aktivitou. Normální aktivita je značena plnou čárou. Fiktivní je označena přerušovanou čárou.



Obrázek 9: Aktivita, fiktivní aktivita a nápověda

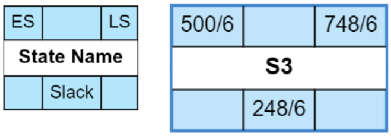
Pro každou aktivitu bylo nastaveno menu a nápověda (anglicky tooltip). Menu je zobrazeno po kliknutí pravým tlačítkem myši na zvolenou aktivitu a zobrazuje ovládací prvky pro úpravu (položka *Edit*) nebo smazání aktivity (položka *Delete*). Po stisknutí položky se otevře modální okno s formulářem pro zvolenou akci. Nápověda (viz Obrázek 9) byla v grafu vytvořena pro rychlé zobrazení všech důležitých informací o činnosti a zobrazí se umístěním kurzoru myši na aktivitu. Nápověda obsahuje název, typ, hodnoty činnosti a údaj o jednotce času, ve které byly hodnoty zadány.

Stav

Stav (viz Obrázek 10) je reprezentován jako vrchol grafu a propojuje předchozí a následující aktivitu. Popis je dán názvem stavu (*State Name*) a výsledky po zvolené analýze. Po provedené analýze se ve stavu zobrazí následující hodnoty:

- Earliest Start Time (ES) – čas, kdy nejdříve mohou začít následující aktivity (činnosti),
- Latest Start Time (LS) – čas, kdy nejpozději musí začít následující aktivity (činnosti), aby se projekt stihnul v plánovaném čase,
- Slack – časová rezerva.

Pro každý stav bylo nastaveno menu obsahující ovládací prvky stavu. Zobrazí se po kliknutí pravým tlačítkem myši na zvolený stav a obsahuje ovládací prvky pro úpravu (položka *Edit*) nebo smazání stavu (položka *Delete*). Dále obsahuje položku *Add State*, která slouží k otevření formuláře pro přidání nového stavu a položku *Add Activity*, která otevře formulář pro přidání nové aktivity propojující dva stavy, který byly vybrány položkou *Add Activity*.



Obrázek 10: Stav a stav po analýze systému

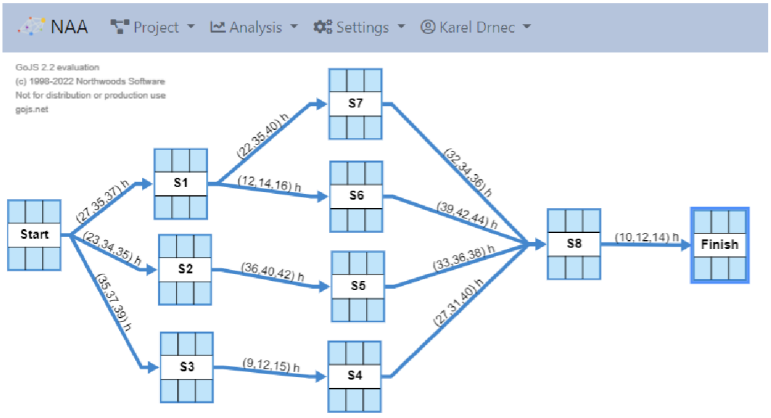
3.4 Zobrazení projektu

Pro zobrazení diagramu (systému) byla ve vytvořené aplikaci použita JavaScriptová knihovna GoJS (https://gojs.net/latest/index.html) od společnost Northwoods Software. Nejdříve je v této části práce popsána knihovna GoJS. Následně je zde popsána problematika vykreslení paralelních hran, formát dat pro vytvoření projektu, vytvoření projektu a práce s grafickými objekty projektu (stavy a činnosti).

GoJS

Knihovna GoJS se zabývá reprezentacemi interaktivních grafů (např. stromové grafy, rodokmeny, vývojové diagramy, UML diagramy, aj.). K zobrazení systému do grafu šlo zvolit i knihovnu AnyChart (viz Kapitola 2.4). Pro aplikaci byla ale vybrána knihovna GoJS z důvodu, že systém lze do grafu zobrazit v interaktivnější podobě pro uživatele.

Knihovna GoJS obsahuje předdefinované vzorky (anglicky samples), které představují základní zobrazení dané problematiky do grafové podoby. Mimo jiné obsahuje šablonu pro problematiku PERT (https://gojs.net/latest/samples/PERT.html). Tato šablona byla využita pro zobrazení systému v grafické podobě ve vytvořené aplikaci a dodatečně modifikována pro potřeby aplikace (viz Obrázek 11).



Obrázek 11: Systém v prostředí vytvořené aplikace

Knihovna GoJS je placená a bezplatně ji lze využít pouze pro své soukromé účely. Pro tento účel obsahuje HTML element, ve kterém je vykreslen graf za použití této knihovny,

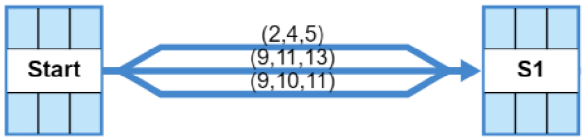
informaci o knihovně s dodatkem, že nesmí být použita pro komerční účely. V případě jejího užití pro komerční účely nabízí GoJS následující licence:

- Team License – týmová licence pro maximálně 3 developery (cena 6 990 $),
- Group License – skupinová licence pro maximálně 50 developerů (cena 9 950 $),
- Enterprise License – pro zakázkové potřeby (pro více informací nutno kontaktovat společnost),
- Individual License – pro jednotlivce, začínající společnosti (cena 3 995 $).

Týmová, skupinová a individuální licence mají platnost po dobu tří let s podporou ze strany společnosti na jeden rok. Společnost Northwoods Software nově vydala také knihovnu GoDiagram, která představuje knihovnu pro framework .NET pro tvorbu aplikací zabývající se reprezentacemi grafu v prostředí WinForms. [15]

Vykreslení paralelních hran grafu

Použitý předdefinovaný model PERT (https://gojs.net/latest/samples/PERT.html) knihovny GoJS umí pracovat pouze s jednou hranou mezi dvěma různými uzly grafu. Paralelní hrany zobrazuje jako jednu hranu, která je překreslena dalšími hranami. V nastavení zobrazení grafu není definována možnost, která by správné vykreslení paralelních hran zajistila. Z tohoto důvodu byl pro hrany grafu použit skript *ParallelRouteLink*, který definuje zobrazení, ve kterém se dvě hrany obsahující stejný počáteční a koncový uzel vzájemně nepřekreslují (viz Obrázek 12). Tento skript je vydaný stejnou společností a není součástí základní verze knihovny GoJS. Dále pro lepší zobrazení hran a jejich popisů byl u uzlů grafů nastavena větší mezera mezi dvěma sousedícími uzly a to pomocí vlastnosti *layerSpacing*, která byla nastavena na hodnotu 200. Ve výchozím nastavení má vlastnost *layerSpacing* hodnotu 50 a zobrazuje sousedící uzly blízko sebe tak, že popis jednotlivých hran je v určitých případech špatně čitelný.



Obrázek 12: Zobrazení paralelních hran v grafu

Formát dat pro vytvoření grafu

K zobrazení grafu jsou data o objektech grafu (uzly a hrany) uložena v polích, která obsahují objekty typu JSON. První pole obsahuje uzly grafu (stavy projektu) a každý prvek tohoto pole obsahuje následující klíče:

- key – ID uzlu (ID stavu z databáze),
- text – název stavu,
- critical – údaj, zda stav je součástí kritické cesty či nikoliv (hodnota *true* nebo *false*).

Klíč *critical* určuje, je-li uzel grafu součástí kritické cesty či nikoliv. Pokud má hodnotu *true*, je uzel vykreslen červenou barvou. V případě hodnoty *false* je vykreslen výchozí barvou nebo barvou, která byla zvolena v nastavení aplikace (dále Kapitola 3.7). Před analýzou mají všechny uzly v poli nastavenou hodnotu klíče *critical* na *false*.

Po proběhlé analýze každý uzel v poli obsahuje další klíče:

- earliestStart – nejdříve možný začátek činnosti (Earliest Start Time – ES),
- latestStart – nejpozději přípustný začátek činnosti (Latest Start Time – LS),
- slack – časová rezerva stavu.

Hodnota klíče *critical* je po analýze závislá na hodnotě klíče *slack*, která určuje časovou rezervu stavu. Pokud je rovna 0, je hodnota klíče *critical* změněna na *true* a uzel (stav) se v grafu vykreslí červenou barvou. Klíče *earliestStart*, *latestStart* a *slack* jsou také důležité pro vypsání informací o stavu, kdy jejich hodnoty jsou zobrazeny v příslušných polích uzlu grafu (viz Obrázek 10).

Druhé pole obsahuje hrany grafu (činnosti projektu) a každý jeho prvek obsahuje následující klíče:

- id – ID hrany (ID činnosti z databáze),
- from – ID uzlu, který představuje počáteční stav hrany,
- to – ID uzlu, který představuje koncový stav hrany,
- text – popis hrany,
- color – barva hrany,
- tooltip – nápověda hrany (další informace o činnosti).

Popis hrany je definován délkou činnosti pro metodu kritické cesty nebo odhady délky činnosti pro metodu PERT. Klíč *color* může nabývat hodnoty *N*, která představuje výchozí barvu nebo barvu zvolenou v nastavení aplikace, anebo hodnoty *R*, která označuje činnost, která je součástí kritické cesty. Hodnoty *R* může nabývat pouze po provedené analýze. Hodnota klíče *tooltip* představuje nápovědu hrany (viz Obrázek 9), ve které jsou vypsány dodatečné informace o činnosti. Textový řetězec nápovědy byl v aplikaci sestaven pomocí vytvořené funkce *parseLinkTextTooltip()*, která jako vstup přijímá název činnosti, typ činnosti, hodnoty a jednotku času, ve kterých byly hodnoty činnosti zadány. Pokud je

činnost fiktivní, obsahuje objekt v poli další klíč s názvem *dash*, díky kterému je fiktivní hrana v grafu vykreslena přerušovanou čarou (viz Obrázek 9).

Obě pole obsahující data grafu jsou poté přidány do diagramu pomocí modelu *GraphLinksModel()*, který na vstupu přijímá daná pole dat. Při aktualizaci dat stačí pouze zavolat tento model s aktualizovanými daty.

Vytvoření projektu

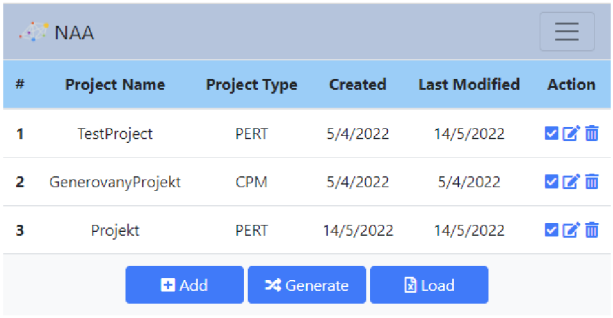
Hlavní stránku aplikace představuje adresář (viz Obrázek 13), ve kterém se nachází uživatelem vytvořené projekty. Adresář s projekty je tvořen tabulkou pomocí HTML elementu s tagem *<table>*, kde každý řádek tabulky obsahuje následující sloupce:

- Project Name – název projektu,
- Type of Project – typ projektu (CPM nebo PERT),
- Created – datum vytvoření projektu (ve formátu den/měsíc/rok),
- Last Modified – datum poslední úpravy projektu (ve formátu den/měsíc/rok),
- Action – skupina ovládacích prvků projektu (otevření, úprava nebo smazání projektu).

Popis jednotlivých sloupců se nachází v záhlaví tabulky. V zápatí tabulky jsou ovládací prvky (tlačítka) pro vytvoření nového projektu. První tlačítko s názvem *Add* slouží pro přesměrování na stránku s definováním nového projektu. Druhé a třetí tlačítko je pro obsluhu generování náhodného projektu a načtení projektu ze souboru (dále Kapitola 3.8).

Po stisknutí tlačítka *Add* je aplikace přesměrována na stránku s formulářem pro přidání nového projektu, kde uživatel aplikace musí vyplnit název projektu (*Project Name*) a vybrat typ projektu (*Type of Project*), ve kterém je na výběr mezi metodou CPM nebo PERT. Následně může doplnit další informace o projektu do položky *Information* (nepovinný údaj). Po potvrzení formuláře jsou údaje o projektu poslány na serverovou část aplikace, kde jsou zaslané údaje z formuláře zpracovány asynchronní funkcí *addProject*, která byla vytvořena v *projectController* (kontroler projektu). V této funkci je vytvořen projekt a jeho dva základní stavy (*Start* a *Finish*). Nově vytvořené objekty jsou následně uloženy do databáze a aplikace je přesměrována zpět do adresáře projektů (viz Obrázek 13). Nově vytvořený projekt se nachází na konci tabulky a je po dobu tří vteřin zvýrazněn žlutým pozadím řádku tabulky. Název projektu musí být v rámci aplikace pro konkrétního uživatele unikátní. Pokud není, nový projekt se nevytvoří z důvodu duplicity dat. Nově vytvořený projekt

lze otevřít pomocí tlačítka *Select*, které se nachází v ovládacích prvcích projektu ve sloupci *Action* nebo pomocí dvojkliku na řádek tabulky s projektem.

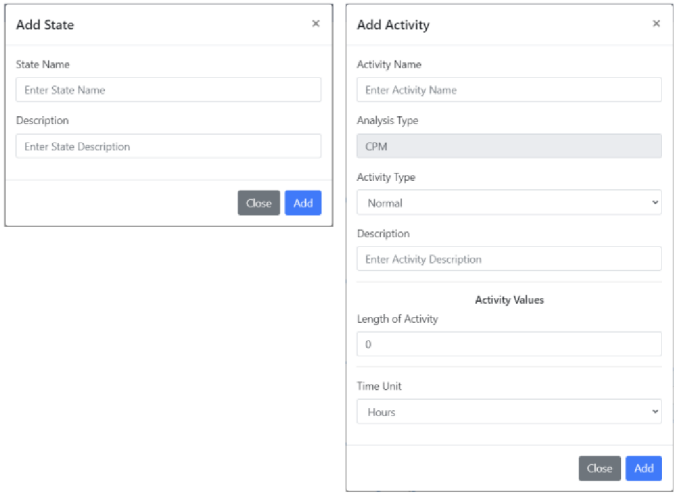


Obrázek 13: Adresář s vytvořenými projekty

Přidání objektu do projektu

Nové uzly a hrany jsou do projektu přidány pomocí tlačítka *Add State* (přidání nového stavu) a tlačítka *Add Activity* (přidání nové činnosti). Obě tato tlačítka se nachází v menu uzlu (stavu) a zobrazí se s pravým kliknutím na zvolený stav. Přidat nový uzel lze i pomocí pravého kliknutí do pozadí diagramu, kdy se otevře menu, které obsahuje položky *Help* (zobrazení nápovědy) a právě položku *Add State* pro přidání nového stavu.

Po stisknutí tlačítka pro přidání nového stavu nebo činnosti se otevře v modálním okně formulář pro definici nového stavu nebo činnosti (viz Obrázek 14). Ve formuláři pro nový stav musí uživatel vyplnit název stavu (*State Name*), který v rámci projektu nesmí být duplicitní. Dále může vyplnit dodatečné informace o stavu v položce *Description*. Pro přidání nové činnosti musí ve formuláři vyplnit název činnosti (*Activity Name*), který musí být v rámci projektu unikátní. Dále musí zvolit typ činnosti (*Activity Type*), kde je na výběr mezi normální a fiktivní činností. Mezi poslední povinné údaje patří hodnoty činnosti (*Activity Values*) a jednotka času (*Time Unit*), ve které byly hodnoty činnosti zadány. Hodnoty činnosti závisí na typu projektu. Ve formuláři v položce *Analysis Type* je vypsán typ analýzy, se kterou projekt pracuje (CPM nebo PERT). V případě, že se jedná o metodu CPM, musí uživatel ve formuláři definovat délku činnosti y_{ij} v položce *Length of Activity*. V metodě PERT musí stanovit optimistický odhad délky činnosti a_{ij} (*Optimistic Time*), modální odhad délky činnosti m_{ij} (*Modal Time*) a pesimistický odhad délky činnosti b_{ij} (*Pessimistic Time*). Dále může doplnit dodatečné informace o činnosti v položce *Description*. Po stisknutí tlačítka *Add* v zápatí formuláře se data o novém stavu nebo aktivitě pošlou na serverovou část aplikace.



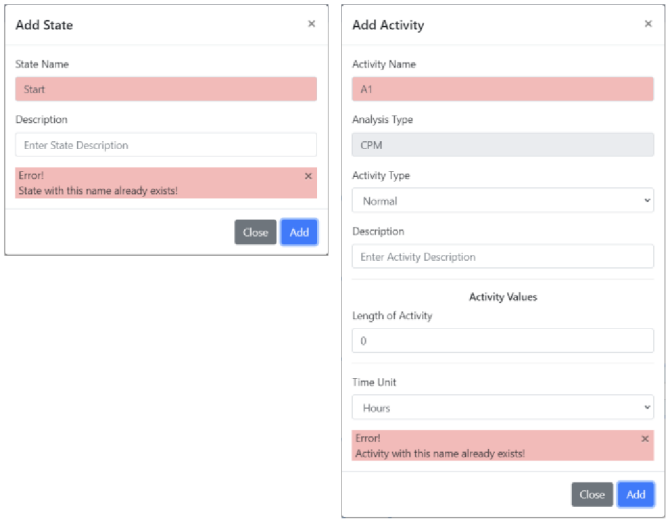
Obrázek 14: Formuláře pro přidání stavu a činnosti

V komunikaci mezi klientskou a serverovou částí aplikace byl pro práci se stavy a činnostmi projektu použit real-time aplikační framework Socket.io (https://socket.io/), který umožňuje obousměrnou komunikaci mezi serverem a klientem v reálném čase. Framework se skládá z knihovny, která běží na straně serveru a z knihovny běžící na straně klienta v prohlížeči. Komunikace probíhá v zasílání tzv. zpráv, kdy jedna strana (server nebo klient) vyšle zprávu a druhá strana tuto zprávu zachytí, zpracuje a případně zašle odpověď v podobě další zprávy. V aplikaci byl tento framework použit z důvodu snížení počtu přístupů do databáze, kdy aplikace v hlavním prostředí projektu (tvorba grafu) načítá všechny stavy a činnosti jen při prvotním načtení projektu a při aktualizaci stránky. Přidání nových objektů do diagramu probíhá tak, že se z formuláře pro přidání objektu vyšle zpráva s daty pomocí Socket.io do kontroleru stavů (*stateController*) nebo do kontroleru činností (*activityController*), zde se zpráva s daty o objektu vyhodnotí a zašle se odpověď zpět do klientské části. Pro zaslání zprávy se využívá funkce *emit()* a pro zachycení zprávy se využívá funkce *on()*. [16]

Přidání stavu

Po vyplnění formuláře s daty o novém stavu se před odesláním zkontroluje, zda byl vyplněn název stavu. Následně jsou data z formuláře zaslána funkcí *emit("new state", data stavu)* na server do kontroleru (*stateController*), kde jsou data zpracována. Pokud údaje o stavu byly zadány správně, je nový stav uložen do databáze a je zaslána zpráva (odpověď) funkcí *emit("new state", data stavu)* ze strany serveru zpět do klientské části aplikace. V klientské části po obdržení odpovědi je nový stav (uzel grafu) přidán do diagramu pomocí vytvořené funkce *addCreatedState()*, která nový uzel vloží do pole uzlů a pomocí vytvořené funkce *reload()* znovu vykreslí graf s aktualizovanými daty. Po úspěšném přidání je formulář pro nový stav zavřen. V případě chyby při zpracování dat na straně serveru (např. duplicitní

název stavu) vyšle server zprávu pomocí funkce *emit("error state", kód chyby, výskyt chyby)*, kde kód chyby nabývá buď hodnoty *1*, pokud se jedná o chybu způsobenou duplicitou dat u názvu stavu, anebo hodnoty *0*, pokud se jedná o jinou chybu způsobenou ukládáním nového stavu do databáze. Výskyt chyby označuje, v jakém formuláři při práci se stavem se chyba objevila (přidání, aktualizace nebo smazání stavu). Zde nabývá textové hodnoty *"add"*. Klientská část aplikace přijme zprávu s chybou a zobrazí ji na konci formuláře (viz Obrázek 15). Pokud se jedná o chybu způsobenou duplicitou dat, je položka *State Name* po dobu tří vteřin zvýrazněna červenou barvou.



Obrázek 15: Chybové hlášení formulářů

Přidání činnosti

Ve formuláři pro přidání nové činnosti (hrany grafu) se před odesláním dat na serverovou část aplikace kontroluje, zda byl vyplněn název činnosti (*Activity Name*) a zda byly správně vyplněny hodnoty činnosti v případě metody PERT. Optimistický odhad délky činnosti a_{ij} musí být menší nebo roven modálnímu odhadu délky činnosti m_{ij}, který musí být menší, anebo roven pesimistickému odhadu délky činnosti b_{ij}. Pokud není toto dodrženo, je zobrazena chyba na konci formuláře spolu s vyznačenou položkou formuláře, ve které chyba nastala. Po kontrole jsou data z formuláře zaslána pomocí funkce *emit("new activity", data činnosti)* na server do kontroleru (*activityController*), kde jsou data zpracována. Pokud se jedná o fiktivní činnost, proběhne navíc kontrola, zda mezi dvěma stavy (uzly) neexistuje už fiktivní činnost (hrana), anebo není vstupní stav činnosti počátečním stavem projektu nebo výstupní stav koncovým stavem projektu. V těchto případech nemůže být činnost fiktivní.

Pokud nenastala chyba během kontroly, je nová činnost (hrana) uložena do databáze a zaslána zpráva s činností zpět do klientské části funkcí *emit("new activity", data činnosti)*.

V klientské části aplikace je poté volána vytvořená funkce *addCreatedActivity()*, která novou hranu vloží do pole hran a volá funkci *reload()*, ve které se graf znovu vykreslí s aktualizovanými daty. Následně je modální okno formuláře pro přidání nové činnosti zavřeno. Pokud na straně serveru dojde k chybě, je klientovi zaslána zpráva pomocí *emit("error activity", kód chyby, výskyt chyby)*, kde kód chyby nabývá hodnoty *1* pro duplicitu názvu činnosti, hodnoty *2* nebo *3* pro fiktivní činnost, která nemůže mezi dvěma zvolenými stavy být přidána. Poslední hodnotu chyby je hodnota *0* pro jiný druh chyb, který nastal při ukládání dat o činnosti do databáze. Výskyt chyby označuje, v jakém formuláři při práci s činností se chyba objevila (přidání, aktualizace nebo smazání činnosti). Zde nabývá textové hodnoty *"add"*. V klientské části aplikace je zaslaná zpráva zobrazena na konci formuláře (viz Obrázek 15). Pokud se navíc jednalo o chybu způsobenou duplicitním názvem činnosti, je položka pro zadání názvu činnosti (*Activity Name*) zvýrazněna po dobu tří vteřin červeným pozadím.

Editace objektu v projektu

Pro editaci stavu (uzlu) nebo činnosti (hrany) musí uživatel vybraný objekt zvolit stisknutím pravého tlačítka myši a vybrat z nabídky položku *Edit*. Po stisknutí položky *Edit* se zobrazí formulář pro editaci stavu nebo činnosti.

Editace stavu

Formulář pro editaci stavu obsahuje název stavu (*State Name*) s názvem vybraného stavu a popis (*Description*) stavu. Pomocí tlačítka *Edit* jsou data o upraveném stavu zaslána do kontroleru (*stateController*) pomocí funkce *emit("edit state", data stavu)*. V kontroleru je stav s novými daty aktualizován v databázi. Pokud se při aktualizaci stavu nevyskytla chyba (např. duplicita dat), je zaslána zpět do klientské části aplikace odpověď pomocí funkce *emit("edit state", data stavu)* s aktualizovaným stavem. V klientské části po obdržení zprávy jsou data upraveného stavu vložena do vytvořené funkce *editSelectedState()*, která pomocí vstupních dat (ID stavu) vyhledá stav v poli uzlů pro zobrazení grafu a ten aktualizuje. Následně je volána funkce *reload()* pro znovunačtení grafu s aktualizovanými daty a formulář pro editaci stavu je uzavřen.

V případě výskytu chyby při aktualizaci stavu v databázi není stav aktualizován a do klienta aplikace je zaslána odpověď funkcí *emit("error state", kód chyby, výskyt chyby)*. Kód chyby nabývá hodnoty *1*, pokud se jedná o chybu způsobenou duplicitou dat u názvu stavu, anebo hodnoty *0*, pokud se jedná o jinou chybu způsobenou aktualizací stavu v databázi. Výskyt chyby zde nabývá textové hodnoty *"edit"* a označuje, že se chyba stala

ve formuláři pro editaci stavu. Chyba je následně ve formuláři zobrazena stejně jako chyby ve formuláři pro přidání stavu (viz Obrázek 15).

Editace činnosti

Formulář pro editaci činnosti obsahuje název činnosti (*Activity Name*) s názvem vybrané činnosti, typ činnosti (*Activity Type*), popis (*Description*), hodnoty (*Activity Values*) a časovou jednotku (*Time Unit*), ve které byly hodnoty činnosti zadány. Pokud se jedná o projekt typu PERT, proběhne před odesláním dat na serverovou část aplikace kontrola hodnot činnosti stejně, jako tomu bylo u přidání nové činnosti. V případě chyby je ve formuláři zobrazena chyba na konci formuláře s výrazněnou položkou formuláře, ve které chyba nastala. Po kontrole jsou aktualizovaná data činnosti zaslána do kontroleru (*activityController*) pomocí funkce *emit("edit state", data činnosti)*. V kontroleru je činnost s novými daty aktualizována. Pokud aktualizace činnosti má změnit činnost (normální) na fiktivní, proběhne před aktualizací kontrola stejně jako u přidání nové činnosti. Pokud se při aktualizaci činnosti v databázi nevyskytla chyba (např. duplicita dat), je do klientské části aplikace zaslána odpověď pomocí funkce *emit("edit activity", data činnosti)* s aktualizovanou činností. V klientské části aplikace jsou data upravené činnosti po obdržení zprávy vložena do vytvořené funkce *editSelectedActivity()*, která pomocí vstupních dat (ID činnosti) vyhledá činnosti v poli hran pro zobrazení grafu a tu aktualizuje. Následně je volána funkce *reload()* pro načtení grafu s aktualizovanými daty a formulář pro editaci činnosti je uzavřen.

V případě výskytu chyby během aktualizace činnosti projektu v databázi není činnost aktualizována a do klientské části aplikace je zaslána zpět odpověď pomocí funkce *emit("error activity", kód chyby, výskyt chyby)*, kde kód chyby nabývá hodnoty *1* pro duplicitu názvu činnosti, hodnoty *2* nebo *3* pro fiktivní činnost, která se nemůže nacházet mezi počátečním a koncovým uzlem činnosti. Poslední hodnotou chyby je hodnota *0* pro jiný druh chyby, který nastal během aktualizace činnosti v databázi. Výskyt chyby zde nabývá textové hodnoty *"edit"* a značí, že se chyba stala ve formuláři pro editaci činnosti. Chyba je následně ve formuláři zobrazena stejně jako chyby ve formuláři pro přidání činnosti (viz Obrázek 15).

Smazání objektu z projektu

Pro smazání stavu (uzlu) nebo činnosti (hrany) musí uživatel vybraný objekt zvolit pomocí pravého tlačítka myši a vybrat z nabídky položku *Delete*. Po stisknutí položky *Delete* se zobrazí formulář, kde uživatel musí potvrdit, že vybraný objekt chce smazat.

Smazání stavu

Po potvrzení formuláře pro smazání zvoleného stavu je do kontroleru (*stateController*) zasláno ID stavu s ID projektu pomocí funkce *emit("delete state", ID stavu, ID projektu)* a formulář je zavřen. V kontroleru jsou nejdříve smazány všechny činnosti v databázi, u kterých je zvolený stav počátečním nebo koncovým stavem. Dále je smazán vybraný stav a u projektu je aktualizována položka poslední úpravy (*lastModified*). Pokud se při smazání stavu nevyskytla chyba, je zpět do klientské části aplikace zaslána odpověď pomocí funkce *emit("delete state", ID stavu)*. Po zachycení odpovědi je v klientské části volána vytvořená funkce *deleteSelectedState()*, která na vstupu přijímá ID smazaného stavu. V této funkci je v poli uzlů smazán uzel podle ID smazaného stavu. Dále jsou v poli hran smazány všechny hrany, u kterých je smazaný uzel počátečním nebo koncovým uzlem. Po smazání uzlů a hran je volána funkce *reload()* pro znovunačtení grafu bez smazaných objektů.

V případě výskytu chyby při mazání stavu nebo činností, ve kterých je vybraný stav počátečním nebo koncovým stavem, je zaslána pomocí funkce *emit("error state", kód chyby, výskyt chyby)* odpověď s chybou do klientské části aplikace. Kód chyby nabývá hodnoty *0* a výskyt chyby je textové hodnoty *"delete"*. Text chyby je v prostředí aplikace zobrazen pomocí metody *alert()*.

Smazání činnosti

Po potvrzení formuláře pro smazání zvolené činnosti je do kontroleru (*activityController*) zasláno ID činnosti s ID projektu funkcí *emit("delete activity", ID činnosti, ID projektu)* a formulář je zavřen. V kontroleru je smazána vybraná činnost v databázi. Následně je u projektu aktualizována položka poslední úpravy (*lastModified*) a pokud nenastala chyba, je do klientské části zaslána odpověď pomocí funkce *emit("delete activity", ID činnosti)*. V klientské části je volána vytvořená funkce *deleteSelectedActivity()*, která na vstupu přijímá ID smazané činnosti. V této funkci je v poli hran smazána hrana podle ID smazané činnosti. Poté je volána funkce *reload()* pro znovunačtení grafu bez smazané činnosti.

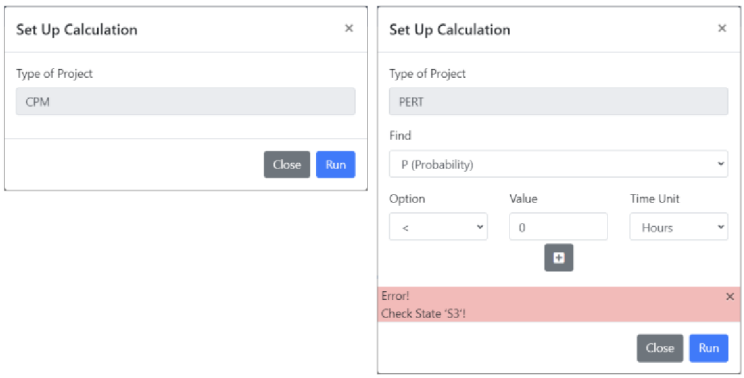
V případě výskytu chyby při mazání činnosti je z kontroleru (*activityController*) zaslána odpověď do klientské části aplikace pomocí *emit("error activity", kód chyby, výskyt chyby)*, kde kód chyby nabývá hodnoty 0 a výskyt chyby obsahuje textovou hodnotu *"delete"*. Text chyby je zobrazen pomocí metody *alert()* do prostředí aplikace.

3.5 Výpočet metody CPM a PERT

V této podkapitole je popsán způsob nalezení kritické cesty v projektu a výpočet metody PERT. Dále je zde popsáno zobrazení kritické cesty v grafu a výsledků obou metod. Následující část obsahuje popis nastavení výpočtu.

Nastavení výpočtu

Výpočet metody kritické cesty (CPM) a PERT probíhá přes položku *Calculate*, která se nachází v položce *Analysis* hlavní nabídky aplikace. Po zvolení položky *Calculate* se zobrazí formulář s názvem *Set Up Calculation* pro nastavení výpočtu (viz Obrázek 16).



Obrázek 16: Formulář pro nastavení výpočtu a chybové hlášení

Kontrola projektu před výpočtem

Projekt je před samotnou analýzou zkontrolován, zda byl správně vytvořen. Pro kontrolu diagramu byla vytvořena funkce *checkDiagram()*, která jako vstupní data přijímá pole stavů a pole činností. V této funkci probíhá kontrola projektu, zda splňuje následující podmínky:

- počáteční stav projektu (*Start*) musí obsahovat alespoň jednu vystupující činnost,
- koncový stav projektu (*Finish*) musí obsahovat alespoň jednu vstupující činnost,
- ostatní stavy projektu musí obsahovat alespoň jednu vstupující a jednu vystupující činnost.

Pokud alespoň jeden stav tyto podmínky nesplňuje, je algoritmus přerušen a funkce vrátí formuláři název prvního stavu, u kterého se vyskytla chyba. Chybové hlášení je potom zobrazeno na konci formuláři pro nastavení výpočtu (viz Obrázek 16).

Formát dat pro nalezení kritické cesty

Při hledání kritické cesty se pracuje se dvěma poli obsahující objekty typu JSON. První pole obsahuje všechny stavy projektu. Každý prvek pole (objekt typu JSON) obsahuje následující klíče:

- ID – ID stavu,
- name – název stavu,
- ES – nejdříve možný začátek činnosti (anglicky Earliest Start Time),
- LS – nejpozději přípustný začátek činnosti (anglicky Latest Start Time),
- slack – časová rezerva stavu.

Klíče *ES*, *LS* a *slack* mají před analýzou nastavenou hodnotu na *-1*. Druhé pole obsahuje všechny činnosti projektu. Každý prvek pole (objekt typu JSON) obsahuje následující klíče:

- ID – ID činnosti,
- fromState – ID počátečního stavu činnosti,
- toState – ID koncového stavu činnosti,
- value – hodnota činnosti,
- critical – údaj pro určení kritické cesty.

Klíč *critical* je před analýzou nastaven na hodnotu *false*. Pro metodu kritické cesty je hodnotou klíče *value* délka trvání činnosti y_{ij}. V případě metody PERT je hodnotou klíče *value* střední hodnota délky činnosti μ_{ij} a každý prvek pole obsahuje navíc klíče *std* a *variance*, pro které jejich hodnota označuje směrodatnou odchylku a rozptyl délky činnosti.

Nalezení kritické cesty

Algoritmus pro nalezení kritické cesty se skládá z dopředného a zpětného průchodu grafem, který představuje projekt (systém). Při dopředném průchodu grafem se vypočítává pro každý stav hodnota klíče *ES* (nejdříve možný začátek činnosti). Při zpětném průchodu jsou vypočítávány hodnoty klíče *LS* (nejpozději přístupný začátek činnosti). Pokud jsou stanoveny hodnoty klíče *ES* a *LS*, je určena hodnota klíče *slack* (časová rezerva stavu).

Dopředný průchod

Pro dopředný průchod grafem byla vytvořena v aplikaci funkce *forwardCalculation()*, která jako argumenty přijímá pole stavů a pole činností. Počátečnímu stavu (stav s názvem *Start*) je nastavena hodnota klíče *ES* na 0. Poté jsou nalezeny činnosti projektu, pro které je stav *Start* počátečním stavem. Z nalezených činností jsou podle hodnoty klíče *toState* (ID koncového stavu činnosti) nalezeny další stavy k výpočtu, které jsou uchovány v poli.

Poté výpočet probíhá v cyklu, ve kterém jsou dopočítány hodnoty klíče *ES* pro všechny stavy z pole dalších stavů. V cyklu se kontroluje, zda aktuální stav má ve všech svých předcích dopočítanou hodnotu klíče *ES*. Pokud není u všech předchozích stavů tato hodnota známa, stav zůstává v poli dalších stavů pro další iteraci cyklu. Pokud je známa u všech předků aktuálního stavu, je dopočítána hodnota klíče *ES* vytvořenou funkcí *findMaxValue()*. Funkce *findMaxValue()* hledá z činností, které vstupují do aktuálního stavu, takovou činnost, která po přičtení hodnoty klíče *value* k hodnotě klíče *ES* předchozího stavu, ze kterého vychází, dává maximální hodnotu. Po výpočtu hodnoty klíče *ES* pro aktuální stav v cyklu, je stav z pole dalších stavů odstraněn. Dále jsou nalezeny činnosti, pro které byl stav počátečním

stavem a stavy, které jsou koncovými stavy nalezených činností. Nalezené nové stavy jsou přidány do pole dalších stavů a cyklus pokračuje v nové iteraci. Pokud je dopočítána hodnota *ES* pro koncový stav projektu *Finish*, je pole dalších stavů prázdné a cyklus je ukončen. Poté funkce *forwardCalculation()* vrací pole stavů s dopočítanou hodnotou *ES* pro každý stav projektu.

Zpětný průchod

Pro zpětný průchod grafem byla vytvořena funkce *backwardCalculation()*, která přijímá jako argumenty pole činností a pole stavů, které mají dopočítanou hodnotu klíče *ES* z dopředného průchodu. Ve zpětném průchodu představuje koncový stav (stav s názvem *Finish*) počáteční stav výpočtu. Stavu *Finish* je nastavena hodnota klíče *LS* na hodnotu klíče *ES* a hodnota klíče *slack* (časová rezerva stavu) na hodnotu 0. Poté jsou nalezeny činnosti projektu, pro které je stav *Finish* koncovým stavem. Z nalezených činností jsou podle hodnoty klíče *fromState* (ID počátečního stavu činnosti) nalezeny další stavy k výpočtu, které jsou uchovány v poli.

Poté probíhá výpočet v cyklu, ve kterém jsou dopočítány hodnoty klíčů *LS* a *slack* pro všechny stavy z pole dalších stavů. Pokud není u všech následujících stavů hodnota *LS* známa, stav zůstává v poli dalších stavů pro další iteraci cyklu. Pokud je známa u všech potomků aktuálního stavu, je dopočítána hodnota klíče *LS* vytvořenou funkcí *findMinValue()*. Funkce *findMinValue()* hledá z činností, která vystupují z aktuálního stavu, takovou činnost, která po odečtení hodnoty klíče *value* od hodnoty klíče *LS* následujícího stavu, do kterého vstupuje, dává minimální hodnotu. Po výpočtu hodnoty klíče *LS* je dopočítána hodnota klíče *slack* odečtením hodnoty *LS* od hodnoty *ES* a aktuální stav je z pole dalších stavů odstraněn. Dále jsou nalezeny činnosti, pro které byl stav koncovým stavem a stavy, které jsou počátečními stavy nalezených činností. Nalezené nové stavy jsou přidány do pole dalších stavů a cyklus pokračuje v nové iteraci. Pokud jsou dopočítány hodnoty klíčů *ES* a *slack* pro počáteční stav projektu Start, je pole dalších stavů prázdné a cyklus ukončil svou činnosti. Následně funkce *backwardCalculation()* vrací pole stavů, ve kterém jsou dopočítány hodnoty *LS* a *slack* pro každý stav projektu.

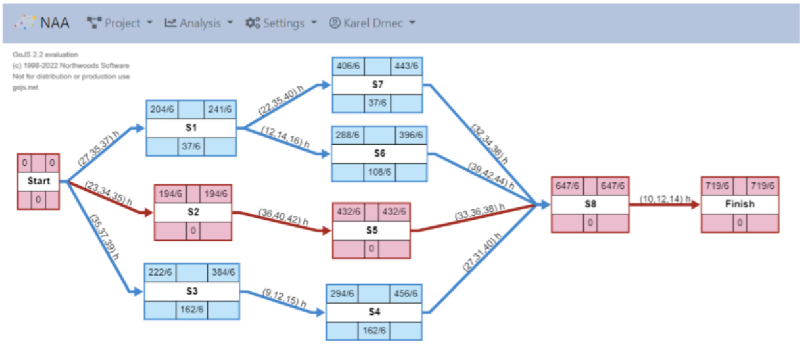
Výsledkem dopředného a zpětného průchodu je pole stavů, ze kterého lze určit kritické činnosti. Stav, který je součástí kritické cesty, má časovou rezervu (*slack*) rovnou 0. Kritické činnosti se nachází mezi stavy s časovou rezervou rovnou 0 a po přičtení délky trvání činnosti k hodnotě klíče *ES* počátečního stavu činnosti vyjde hodnota klíče *ES* koncového stavu činnosti. Po úspěšně proběhlé analýze jsou výsledky výpočtu uloženy do uložiště sessionStorage, kde byl nastaven název proměnné obsahující výsledky hodnotou ID projektu. Proměnná s výsledky je pole objektů typu JSON a obsahuje pole činností, pole

dopočítaných stavů a dodatečné informace o výpočtu uložené v objektu s názvem *project*. Po uložení hodnot do uložiště sessionStorage je v grafu vykreslena kritická cesta.

Vykreslení kritické cesty

Po proběhlé analýze se vykreslí nový diagram (viz Obrázek 17), ve kterém je kritická cesta vyobrazena červenou barvou a jednotlivé stavy obsahují dílčí výpočty časů (*Earliest Start Time* a *Latest Start Time*), ze kterých byla v analýze vypočtena jejich časová rezerva (*slack*). Činnosti, které se nachází mezi jednotlivými kritickými stavy, jsou knihovnou GoJS automaticky vykreslovány červenou barvou. Při takto definovaném způsobu vykreslování může dojít k chybnému zobrazení kritických činností v diagramu.

Ve vytvořené aplikaci je správné vykreslení zajištěno pomocí klíče *critical*, který obsahují všechny činnosti v poli po výpočtu analýzy. Pokud se jedná o kritickou činnost, hodnota klíče *critical* je nastavena na hodnotu *true*. Následně při zobrazení celého diagramu je u činností hodnota tohoto klíče kontrolována a v případě hodnoty *true*, je činnost vykreslena červenou barvou.



Obrázek 17: Vykreslení kritické cesty v prostředí aplikace

Výpočet metody PERT

Při výpočtu metody PERT se v modálním okně pro spuštění výpočtu kromě typu analýzy zobrazí formulář k vyplnění vstupních hodnot, které jsou potřebné pro výpočet analýzy PERT.

Formulář k vyplnění vstupních hodnot se nachází pod typem projektu (položka *Type of Project*) a obsahuje řádek s názvem *Find*, který slouží k výběru hledané veličiny (pravděpodobnost nebo čas). Druhý řádek slouží ke specifikaci výpočtu a obsahuje následující sloupce:

- *Option* – volba operátoru pro hledání pravděpodobnosti nebo času,

- *Value* – zadávaná hodnota představující požadovaný čas, pro který je hledána pravděpodobnost ukončení projektu v tomto čase nebo pravděpodobnost, pro kterou je hledán čas ukončení projektu s touto pravděpodobností,
- *Unit* – volba jednotky, ve které byla hodnota pole *Value* zadána.

V případě volby hledání pravděpodobnosti lze zvolit operátor „<“ v sloupci *Option*, kdy je hledána pravděpodobnost, se kterou projekt ukončí svou činnost do stanovené hodnoty v poli *Value*. Pokud je vybrán operátor „>”, je hledána pravděpodobnost, se kterou projekt neukončí svou činnost do stanové hodnoty v poli *Value*. Pro hledání pravděpodobnosti lze zobrazit druhý řádek a to pomocí tlačítka označeného plusem. Tímto způsobem lze poté hledat:

- pravděpodobnost, že se daný projekt stihne mezi dvěma požadovanými časy (kombinace operátorů „>“ a „<”),
- pravděpodobnost, že projekt bude ukončen v čase menším než první zadaná hodnota nebo v čase větším než druhá zadaná hodnota (kombinace operátorů „<“ a „>“).

Jednotka zvolená v poli *Time Unit* představuje jednotku času, ve které byla hodnota v poli *Value* zadána. Pokud se hledá čas, se kterým bude projekt ukončen v požadované pravděpodobnosti, obsahuje pole *Option* operátor „=“. Hodnota v poli *Value* musí být v rozmezí 0-100 (pravděpodobnost) a jednotka v poli *Unit* je fixně nastavena na „%“ (procenta).

Pro výpočet metody PERT byla vytvořena funkce *calculatePERT()*, která jako argumenty přijímá střední hodnotu délky trvání projektu a rozptyl. Dále přijímá argumenty, které byly zadány v modálním okně pro nastavení výpočtu PERT. K výpočtu byla použita statistická knihovna jStat (viz Kapitola 2.5), ze které byly v aplikaci použity dvě funkce:

- jStat.normal.cdf(x,mean,std) – zjištění hodnoty distribuční funkce normálního rozdělení v bodě x (požadovaný čas) ze vstupních hodnot (požadovaný čas, střední hodnota, směrodatná odchylka),
- jStat.normal.inv(p,mean,std) – inverzní metoda, kde se na základě požadované vstupní pravděpodobnosti získá výsledný čas, ve kterém projekt bude ukončen se zadanou pravděpodobností ze vstupních hodnot (požadovaná pravděpodobnost, střední hodnota, směrodatná odchylka).

Výsledky jsou zobrazeny ve spodní části formuláře pro nastavení vstupních hodnot metody PERT nebo v tabulce s výsledky na stránce *Results*.

Zobrazení výsledků v tabulce

Pro zobrazení výsledků obou metod byla vytvořena tabulka (viz Obrázek 18), na kterou se uživatel přesměruje výběrem možnosti *Results* ze sekce *Analysis* hlavní nabídky pracovního prostředí aplikace. V tabulce byl vytvořen přepínač, který umožňuje zobrazit tři různé obsahy tabulky:

- Summary – souhrnné informace o proběhlé analýze projektu,
- States – výsledné hodnoty dílčích stavů projektu,
- Activities – výsledné hodnoty dílčích činností projektu.

#	Activity Name	Activity Type	a_{ij}	m_{ij}	b_{ij}	μ_{ij}	σ_{ij}	σ_{ij}^2
1	A1	Normal	27	35	37	204/6	10/6	100/36
2	A2	Normal	23	34	35	194/6	12/6	144/36
3	A3	Normal	35	37	39	222/6	4/6	16/36
4	A5	Normal	36	40	42	238/6	6/6	36/36

Obrázek 18: Zobrazení činností v tabulce výsledků

Souhrnná tabulka (*Summary*) obsahuje informace o projektu v horní části tabulky. Dále obsahuje informace o kritické cestě, a pokud se jedná o projekt typu PERT, obsahuje také informace o proběhlém výpočtu metody PERT. V části tabulky se stavy (*States*) jsou vypsány stavy projektu s vypočtenými hodnoty pro *ES* (*Earliest Start Time*), *LS* (*Latest Start Time*) a *slack* (časová rezerva). Stavy, které jsou součástí kritické cesty, jsou v tabulce zobrazeny s červenou barvou pozadí řádku. V poslední části tabulky s činnostmi projektu (*Activities*) jsou vypsány pro každou činnost její údaje, které byly zadány při její definici. Pokud se jedná o projekt typu CPM, je v tabulce vypsána pro každou činnosti její délka trvání. V případě projektu typu PERT jsou v tabulce vypsány odhady délky trvání činnosti a vypočítané hodnoty pro střední hodnotu, směrodatnou odchylku a rozptyl (viz Obrázek 18). Kritické činnosti projektu jsou v tabulce zobrazeny s červenou barvou pozadí řádku.

3.6 Simulace Monte Carlo

V této části práce je detailněji popsáno použití simulace Monte Carlo s analýzou typu PERT ve vytvořené aplikaci. Nejdříve je zde popsáno Web Workers API řešící zásadní problém skriptovacího jazyka JavaScript v práci s více vlákny (multithreading), kdy JavaScript byl vytvořen jako jednovláknový (single-threaded). Následuje popis simulace Monte Carlo ve vytvořené aplikaci s použitím Web Workers API a zobrazením výsledků v grafu pomocí knihovny Chart.js (https://www.chartjs.org/).

Web Workers API

Web Workers je API, které umožňuje vykonávat skripty v pozadí stránky za účelem nesnížení výkonu hlavní stránky aplikace. Toto API je využíváno zejména pro spouštění náročných skriptů, které by při spuštění na hlavní stránce snižovaly výkon nebo by mohly způsobit úplné „zamrznutí“ stránky do doby, než spuštěný skript dokončí svou činnost. Skripty spuštěné v pozadí nemají přístup k obsahu stránky. Komunikace mezi stránkou a skriptem spuštěným v pozadí probíhá ve vyměňování zpráv.

Nový objekt typu Worker se inicializuje pomocí konstruktoru *Worker(URI)*, kde URI (Uniform Resource Identifier) představuje cestu ke skriptu, který má být spuštěn v pozadí. Data, která mají být spuštěna ve skriptu v pozadí, jsou poslána jako zpráva z popředí aplikace pomocí metody *postMessage()*. Zpracovaná data se zpět získají pomocí události *onmessage*, ve které se definuje funkce, která se má vykonat po zachycení této události. Ve skriptu, který je spuštěn v pozadí, se zpráva zachytává opět pomocí události *onmessage*, pro kterou se definuje funkce, která se má vykonat po jejím zachycení. Po zpracování dat se data odesílají zpět ze skriptu pomocí metody *postMessage()*, do které jako vstup přichází výsledky skriptu. Pro přerušení objektu typu Worker je volána metoda *terminate()*, která přeruší vykonávání skriptu v pozadí aplikace.

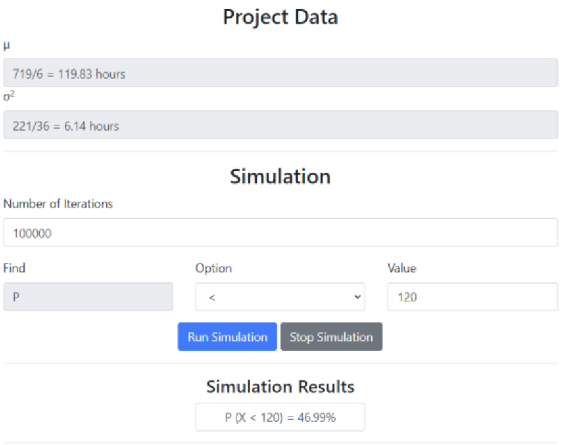
Skripty, které jsou vložené v hlavním prostředí aplikace, nejsou ve skriptu spuštěném v pozadí viditelné a to z důvodu, že skript v pozadí nepracuje ve stejném vlákně. Skripty spuštěné v pozadí mají přístup ke globální funkci *importScripts()*, která importuje skripty a jako parametr přijímá libovolný počet objektů URI představující cestu k importovanému skriptu. [17]

Simulace v aplikaci

V hlavním prostředí projektu se simulace Monte Carlo nachází v položce *Analysis* hlavní nabídky aplikace, kde v sekci *Analysis* je položka *Monte Carlo*, která přesměruje aplikaci na stránku se simulací. Před přesměrováním aplikace je projekt zkontrolován, zda se jedná o projekt typu PERT a zda byla provedena analýza metody PERT. Pokud nedojde během kontroly k chybě, je aplikace přesměrována na stránku se simulací Monte Carlo. V případě chyby je v aplikaci zobrazeno chybové hlášení pomocí metody *alert()*.

Stránka *monteCarlo*, ve které probíhá simulace Monte Carlo, obsahuje vstupní formulář pro nastavení simulace (viz Obrázek 19) a graf, ve kterém je zobrazen průběh simulace. V horní části formuláře (*Project Data*) je zobrazena střední hodnota a rozptyl projektu. V další části formuláře se nachází položka *Number of Iterations*, která představuje počet iterací simulace. Minimální počet iterací je nastaven na hodnotu 1000. Dalším vstupní

hodnotou formuláře je hodnota, pro kterou je hledána pravděpodobnost. V případě volby hledání pravděpodobnosti lze zvolit operátor „<“ v sloupci *Option*, kdy je hledána pravděpodobnost, se kterou projekt ukončí svou činnost do stanovené hodnoty v poli *Value*. Pokud je vybrán operátor „>”, je hledána pravděpodobnost, se kterou projekt neukončí svou činnost do stanové hodnoty v poli *Value*. Výsledná pravděpodobnost je v průběhu simulace zobrazena v sekci *Simulation Results*.



Obrázek 19: Vstupní formulář simulace Monte Carlo v aplikaci

Graf (viz Obrázek 20) zobrazuje průběh simulace Monte Carlo. Na ose x jsou zobrazeny hodnoty představující délky projektu, které vznikly v průběhu simulace. Osa y představuje pravděpodobnost výskytu jednotlivých délek na ose x. Pro graf byla vytvořena dodatečná funkce pro změnu maximální zobrazené hodnoty (pravděpodobnosti) na ose y. Tato funkce se nachází nad osou y a umožňuje měnit zobrazení grafu v průběhu simulace.

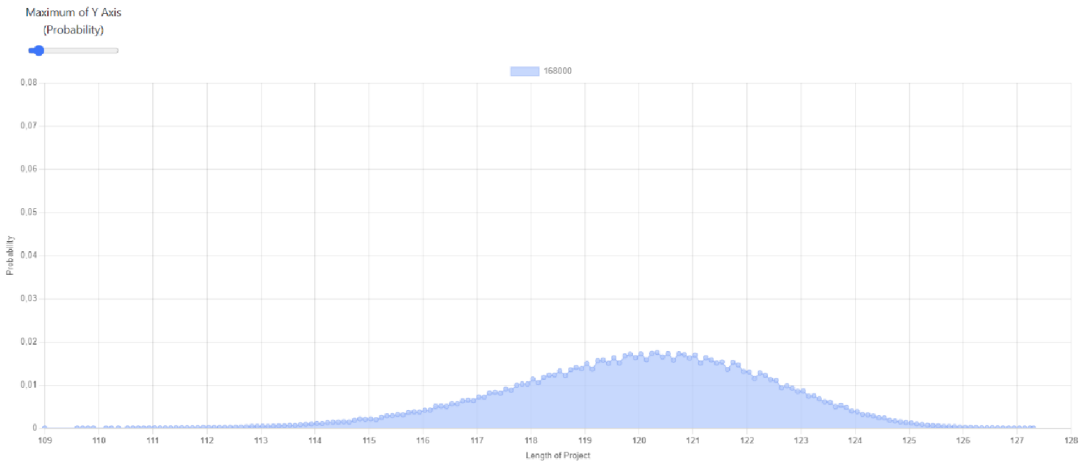
Průběh simulace

Po potvrzení vstupního formuláře je před vytvořením objektu typu Worker zkontrolováno, zda webový prohlížeč objekty typu Worker podporuje. Pokud nepodporuje, je vypsáno chybové hlášení pomocí metody *alert()*. Pokud podporuje, je vytvořen nový objekt typu Worker pomocí konstruktoru *Worker()*, do kterého je vložen jako vstupní parametr skript, který se má vykonat. Pro simulaci Monte Carlo byl vytvořen skript *monteCarlo.js*, který byl vložen jako parametr do objektu typu Worker. Pro objekt typu Worker byla nastavena metoda *postMessage()*, která posílá do skriptu, který se má vykonat v pozadí, data o počtu iterací, činnostech a stavech projektu. Dále byla nastavena událost *onmessage*, která po zpracování dat v pozadí aplikace zpracuje příchozí data v popředí aplikace (úprava hodnot v grafu).

Ve skriptu *monteCarlo.js* je importován skript *jstat.min.js*, obsahující funkce knihovny jStat a skript *calculations.js*, ve kterém jsou prováděny výpočetní operace vytvořené aplikace.

Ve skriptu je nastavena událost pro zpracování zprávy z popředí aplikace. Pokud přijde požadavek (zpráva) na novou simulaci z popředí aplikace, vykoná se funkce, která byla v této události definována. Vytvořená funkce zpracovává každých 1000 iterací ze zadaného celkového počtu iterací simulace. V každé iteraci je volána funkce *simulateMonteCarlo()*, která byla vytvořena ve skriptu *calculations.js* a přijímá na vstupu stavy a činnosti projektu. V této funkci je pro každou činnost projektu vygenerována délka trvání pomocí funkce *monteCarloActivities()*, ve které se vygenerují pro každou činnost parametry α, β a náhodné číslo z *<0, 1>* představující pravděpodobnost. Následně je zjištěna hodnota X distribuční funkce β-rozdělení pomocí funkce *jStat.beta.inv()* z knihovny jStat, do které jsou vloženy vygenerované parametry β-rozdělení a pravděpodobnost. Získaná hodnota X je následně použita pro výpočet délky trvání činnosti, která je dopočítána podle vzorce $a_{ij} + X \cdot (b_{ij} - a_{ij})$. Po vygenerování délky trvání pro všechny činnosti projektu je proveden dopředný průchod grafem. Po skončení dopředného průchodu vrací funkce *simulateMonteCarlo()* hodnotu *ES* koncového stavu *Finish* zpět do skriptu *monteCarlo.js*, který běží v pozadí aplikace.

Získaná hodnota *ES* koncového stavu je následně vložena do pole objektů typu JSON, kde každý objekt obsahuje klíč *value* představují délku trvání projektu. Hodnota klíče *value* představuje počet výskytů dané délky trvání projektu v simulaci. Po provedení každých 1000 iterací vrací skript v pozadí zprávu (odpověď) do popředí aplikace pomocí metody *postMessage()*. Zpráva obsahuje pole objektů typu JSON s délkami trvání projektu a jejich četnostmi. Po přijetí odpovědi jsou na stránce se simulací Monte Carlo zobrazeny délky trvání projektu na ose x. Na ose y jsou zobrazeny četnosti jednotlivých délek trvání dělené počtem proběhlých iterací simulace. Tyto hodnoty byly využity i pro výpočet hledané pravděpodobnosti (sekce *Simulation Results*), kdy se provádí suma těchto hodnot do stanové délky projektu, která byla zadána ve vstupním formuláři.



Obrázek 20: Graf zobrazující průběh simulace Monte Carlo

3.7 Nastavení aplikace

V aplikaci bylo vytvořeno nastavení, ve kterém lze upravit uživatelské prostředí. Nachází se v hlavní nabídce v sekci *Settings*, kde po zvolení položky *Change* je aplikace přesměrována na stránku s formulářem pro nastavení aplikace. V nastavení lze změnit jazyk, ve kterém je aplikace prezentována uživateli nebo upravit rozložení barev uživatelského prostředí. Jednotlivé změny se nastaví pomocí HTML elementu s tagem *<select>*. V následující části je popsána implementace jazykového rozhraní.

Jazykové rozhraní

Aplikaci lze používat v anglickém jazyce (výchozí možnost) nebo v českém jazyce. Změna jazykového rozhraní se nachází v nastavení aplikace (*Settings*) v sekci *Language*. Ke změně byl použit modul i18n (https://www.npmjs.com/package/i18n), který je využíván k tvorbě vícejazyčných webových aplikací.

V konfiguraci aplikace je v modulu i18n nastaven výchozí jazyk na angličtinu. Vlastní proměnná, podle které se určuje, v jakém jazyce má být aplikace zobrazena, je uložena v cookies pod názvem *locale*. Při změně jazyka v nastavení se v proměnné *locale* nachází buď hodnota *en* pro angličtinu nebo hodnota *cz* pro češtinu. Vlastní překlad jednotlivých frází se nachází ve dvou souborech formátu JSON ve složce *locale*. Soubor *en.json* obsahuje fráze zapsané v anglickém jazyce a v souboru *cz.json* jsou uloženy fráze v českém jazyce. Aplikace na základě proměnné *locale* v cookies určí, v jakém jazyce má být zobrazena jaká fráze. Pro použití modulu i18n v klientské části aplikace musela být v konfiguraci aplikace nastavena dodatečná pomocná funkce v šablonovacím systému Handlebars, díky které je možné tento modul s překlady využívat i v klientské části aplikace.

Další překlady lze přidat vytvořením dalšího souboru formátu JSON s překlady frází v požadovaném jazyce a dodatečnou úpravou v konfiguraci aplikace (přidání další možné proměnné *locale* označující zvolený jazyk).

Rozložení barev

Ve formuláři nastavení lze změnit rozložení barev diagramu nebo navigační lišty aplikace (hlavní nabídka). Rozložení barev diagramu se ve formuláři nachází pod položkou *Color of States (Nodes)*, která slouží ke změně barvy pro stavy (uzly) systému (grafu) a pod položkou *Color of Activities (Edges)*, která slouží ke změně barvy pro činnosti (hrany) systému (grafu). Barva navigační lišty lze změnit v položce *Color of Navigation Bar*.

Pro uložení nastavení barev byly použity cookies. Alternativní možnost pro uložení uživatelského nastavení byla v použití uložiště sessionStorage nebo localStorage, které se

od sessionStorage liší tím, že nemá expirační dobu. Naopak není doporučováno používat pro uložení uživatelského nastavení databázi z důvodu, že tyto údaje nejsou složitě strukturované a důležité, aby pro jejich uložení byla použita databáze. V cookies je nastavení barev uloženo pod proměnnou *graphSettings* představující pole obsahující dvě hodnoty pro zvolenou barvu uzlů a hran grafu. Pro barvu navigační lišty byla použita proměnná s názvem *colorNav*. Obě tyto proměnné se v cookies zpočátku nenachází, vytvoří se potvrzením formuláře nastavení aplikace. Pokud jsou v cookies tyto proměnné přítomny, aplikace je použije a zobrazí diagram nebo lištu hlavní nabídky v požadovaných barvách. Pokud nejsou proměnné přítomny, jsou v aplikaci zvoleny výchozí hodnoty barev. Výchozí barva uzlů a hran grafu je světle modrá. V případě lišty hlavní nabídky je zvolena světle ocelově modrá (*lightsteelblue*). [18]

3.8 Další funkce aplikace

V této podkapitole jsou popsány další funkce vytvořené aplikace. Konkrétně se jedná o export diagramu, uložení projektu do souboru, načtení projektu ze souboru a generování náhodného projektu. V poslední části je popsána nápověda a manuál, který je součástí vytvořené aplikace.

Export diagramu

Diagram, představující vytvořený systém, může uživatel exportovat do následujících grafických formátů:

- SVG – vektorový grafický formát,
- PNG – rastrový grafický formát.

Export diagramu se nachází pod položkou *Export* v sekci *Project* hlavní nabídky. Po vybrání položky *Export* se zobrazí modální okno s formulářem pro nastavení exportu (název souboru a typ grafického formátu). Název souboru je volitelný. Pokud název souboru není uživatelem specifikován, je pro stažení souboru zvolen výchozí název *myDiagram*. Kliknutím na tlačítko *Export* se diagram převede do požadovaného formátu a následně se stáhne uživateli do počítače.

Pro samotný export do obou grafických formátů byly použity implementované funkce knihovny GoJS a to konkrétně *makeSvg()* pro formát SVG a *makeImageData()* pro formát PNG. V obou formátech bylo nastaveno pozadí exportovaného diagramu na bílou barvu. Po volání funkce *makeSvg()* musela být následně proměnná obsahující diagram ve formátu SVG převedena na datový textový řetězec pomocí *XMLSerializer().serializeToString()*. Pro formát PNG funkce *makeImageData()* automaticky převádí diagram do datového

textového řetězce. Následně je tento řetězec vložen do tzv. Blobu. Blob (nestrukturovaný soubor binárních dat) představuje dodatečný výpomocný objekt ve webových aplikacích, pomocí kterého lze ukládat data různých datových formátů do počítače uživatele. Blob se skládá z typu (*type*) a částí (*blobParts*). Typem je definován formát výstupních dat souboru. Části Blobu představují pole, které obsahuje buď další objekty typu Blob nebo přímo zdroj převáděných dat (Buffer nebo datový textový řetězec String). Ve vytvořené aplikaci je u exportu diagramu v typu Blobu hodnota *image/svg+xml* (pro formát SVG) nebo hodnota *image/png* (pro formát PNG). V částech Blobu je obsažen datový textový řetězec obsahující exportovaný diagram formátu SVG nebo PNG. [19]

Pro samotné stažení souboru byl na stránce pomocí JavaScriptu vytvořen HTML element s tagem *<a>*, který na stránce není zobrazen. Pomocí *window.URL.createObjectURL()*, který jako vstupní data přijímá definovaný Blob, je vytvořeno URL, které je umístěno do elementu *<a>*. V elementu *<a>* je nastaven parametr *download* na požadovaný název staženého souboru (pokud je ve formuláři specifikován). Následně byla použita metoda *requestAnimationFrame()*, pomocí které byl nasimulován klik na element *<a>* obsahující v jeho URL vytvořený Blob s exportovaným diagramem. Simulací kliku se exportovaný diagram stáhne do uložiště uživatele aplikace. V metodě *requestAnimationFrame()* je po simulaci nastaveno odebrání použitého elementu *<a>* ze stránky.

Uložení projektu do souboru

Pro uložení projektu lze spolu s databází použít soubor typu JSON. Uložení projektu se nachází pod položkou *Save* v sekci *Project* hlavní nabídky. Po vybrání položky *Save* se zobrazí modální okno s formulářem, ve kterém se definuje název výstupního souboru (*filename*). Dále formulář obsahuje informační pole s typem výstupního souboru (JSON). Pokud není název výstupního souboru uživatelem specifikován, je pro výstupní soubor zvolena výchozí hodnota obsahující název projektu, který má být do souboru uložen. Po potvrzení formuláře stisknutím tlačítka *Save* je projekt převeden do formátu JSON a uložen jako soubor do uložiště uživatele aplikace.

Pro uložení souboru se opět využívá Blob, ve kterém je položka *type* nastavena na hodnotu *application/json* a *blobParts* obsahuje datový textový řetězec String, do kterého byl převeden obsahující informace o projektu pomocí metody *JSON.stringify()*. Objekt typu JSON, který je vložen do Blobu, obsahuje následující klíče:

- project – informace o projektu,
- states – stavy projektu,
- activities – aktivity (činnosti) projektu.

Hodnoty jednotlivých klíčů se skládají z dalších objektů typu klíč/hodnota. Hodnota pro klíč *project* obsahuje hodnoty pro klíč *name* (název projektu), *type* (typ projektu), *description* (popis) a *created* (vytvoření). Hodnota pro klíč *states* se skládá z pole objektů, kde jednotlivý objekt pole obsahuje hodnoty pro klíč *name* (název stavu) a *description* (popis). V případě klíče *activities* je hodnota opět složena z pole objektů, kde každý objekt pole obsahuje hodnoty pro klíč *name* (název aktivity), *type* (typ aktivity), *description* (popis), *fromState* (název počátečního stavu), *toState* (název koncového stavu), *values* (hodnoty aktivity) a *timeUnit* (jednotka času hodnot aktivity). Po vložení JSON objektu obsahujícího informace o projektu do Blobu, se samotné uložení zprostředkuje opět pomocí vytvoření HTML elementu s tagem *<a>*, vložení Blobu do URL elementu, vložením názvu souboru do parametru *download* a simulováním kliku na daný element.

Načtení projektu ze souboru

Projekt uložený v souboru typu JSON lze do aplikace znovu načíst a to v hlavní stránce aplikace s adresářem projektů (*Project Directory*). Pod tabulkou s vytvořenými projekty se nachází tlačítko *Load*, které v modálním okně zobrazí formulář pro načtení souboru. Ve formuláři musí být specifikován název projektu (*Project Name*), jelikož v rámci aplikace je název projektu unikátní a soubor může obsahovat název projektu, který se v adresáři projektů uživatele aplikace již vyskytuje. Dále formulář obsahuje vstup (tlačítko) pro výběr souboru, kdy se po stisknutí tlačítka otevře prostředí počítače s adresáři (složkami) pro výběr souboru.

Po výběru souboru probíhá v klientské části aplikace kontrola typu souboru a načtení obsahu souboru do formuláře. Při kontrole typu je validní hodnotou pouze hodnota *application/json* pro soubor typu JSON. Pokud zvolený soubor je jiného typu, není dále zpracován a aplikace zobrazí chybové hlášení obsahující zprávu o nepodporovaném formátu vstupního souboru. K načtení obsahu souboru bylo použito API pro čtení souborů s názvem FileReader. Po validní kontrole typu vstupního souboru je volán objekt FileReader pomocí konstruktoru *FileReader()*. Objektu jsou následně nastaveny následující události:

- onload – událost, která je volána po úspěšném dokončení načítání ze souboru,
- onerror – událost, která je volána při výskytu chyby během načítání ze souboru.

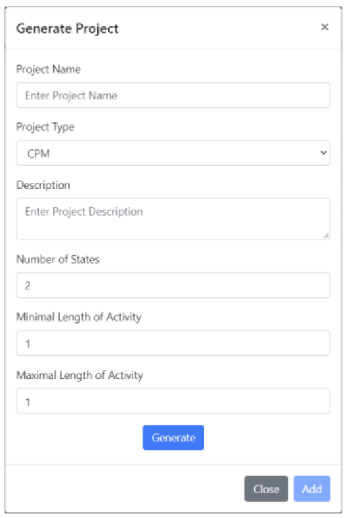
V události *onload* se data načtená ze souboru vloží do formuláře jako textová hodnota (*value*) v HTML elementu s tagem *<input>*, který je v samotném formuláři skryt. Pokud nastane během načítání chyba, je volána událost *onerror*, která zobrazí chybové hlášení ohledně chyby načítání. Pro samotné spuštění načítání ze souboru je volána metoda *readAsText()* objektu FileReader, která jako vstupní argument přijímá zvolený soubor.

Po úspěšném načtení a vložení dat do formuláře, jsou data po potvrzení formuláře tlačítkem *Load* zaslána na serverovou část aplikace.

Zpracování dat ze souboru na serveru obstarává asynchronní funkce *loadProjectFromFile*, která byla vytvořena v kontroleru (*projectController*). V této funkci je textový vstup z formuláře převeden na objekt typu JSON pomocí metody *JSON.parse()*. Z tohoto objektu je následně vytvořen projekt společně s jeho stavy a aktivitami. Projekt je definován podle schématu, ve kterém jsou použity jak data ze souboru, tak i z formuláře, kde byl specifikován název souboru. Po vytvoření projektu jsou vytvořeny stavy, pro které byla použita data ze souboru a ID vytvořeného projektu. Dále jsou vytvořeny aktivity (činnosti), ve kterých byla použita data ze souboru, ID vytvořeného projektu, ID počátečního stavu a ID koncového stavu. Po úspěšném vytvoření všech objektů jsou tyto objekty uloženy do databáze, modální okno s formulářem pro načtení souboru je zavřeno a aplikace je přesměrována do adresáře projektů se zprávou o úspěšném přidání nového projektu.

Generování náhodného systému

Uživatel aplikace si může nechat vygenerovat náhodný systém a to v hlavní stránce aplikace s adresářem projektů (*Project Directory*). Pod tabulkou s vytvořenými projekty se nachází tlačítko *Generate*, které po stisknutí otevře modální okno s formulářem pro nastavení vstupních hodnot generátoru (viz Obrázek 21).



Obrázek 21: Vstupní formulář pro generátor projektu

Ve formuláři musí uživatel vyplnit název projektu (*Project Name*), typ projektu (*Project Type*), počet stavů (*Number of States*), minimální (*Minimal Length of Activity*) a maximální délku činnosti (*Maximal Length of Activity*). Dále může vyplnit popis projektu v položce *Description*. Minimální počet stavů je nastaven na hodnotu 2, kdy jsou vytvořeny pouze stavy *Start* a *Finish*. Maximálně lze pomocí generátoru vytvořit 100 stavů

(viz Příloha C). Pomocí položek minimální délka činnosti a maximální délka činnosti se nastavuje rozsah pro generování náhodného čísla představující délku činnosti. Tlačítkem *Generate* se spustí generování náhodného systému, které probíhá v klientské části aplikace ve vytvořeném skriptu s názvem *projectGenerator*.

Generování náhodného systému je voláno pomocí vytvořené funkce *generateProject()*, která jako vstupní data přijímá počet stavů, typ analýzy (CPM nebo PERT), minimální a maximální délku činnosti. V této funkci jsou nejdříve generovány stavy a to pomocí funkce *generateStates()*, která jako vstupní data přijímá počet stavů zadaných ve formuláři pro generování náhodných systémů. Funkce *generateStates()* vygeneruje počáteční stav *Start*, koncový stav *Finish* a *n-2* dalších stavů. Po vygenerování stavů jsou tyto stavy vloženy do funkce *getStatesParts()*, která tyto stavy rozdělí do tzv. vrstev, které jsou následně propojeny činnostmi. První vrstva obsahuje pouze počáteční stav *Start*. V poslední vrstvě se nachází koncový stav *Finish*. Počet stavů v ostatních vrstvách je generován náhodně v rozmezí 1 až 7.

Po vytvoření vrstev je v hlavní funkci *generateProject()* dále volána funkce *createActivities()*, která jako vstupní data přijímá vrstvy s jednotlivými stavy, typ analýzy, minimální a maximální délku činnosti. V této funkci jsou jednotlivé vrstvy stavů propojeny činnostmi a to způsobem, že aktuální vrstva je propojena s vrstvou následující. Pro propojení je volána funkce *connectLayers()*, která jako vstupní parametr přijímá index, který označuje číslo vygenerované činnosti. Dále přijímá aktuální a následující vrstvu stavů, typ analýzy, minimální a maximální délku činnosti. Ve funkci *connectLayers()* se vrstvy stavů mezi sebou propojují činnostmi. Činnost je generována funkcí *generateActivity()*, která na vstupu přijímá index (označení) poslední vytvořené činnosti, typ analýzy, název počátečního stavu z aktuální vrstvy stavů, název koncového stavu z následující vrstvy stavů, minimální a maximální délku činnosti. V této funkci je vytvořena činnost propojující dva stavy. Délka činnosti je generována funkcí *generateValuesForActivity()*, která na základě typu analýzy (CPM nebo PERT), minimální a maximální délky činnosti vygeneruje délku trvání činnosti. Pokud je analýza typu CPM, je vygenerována délka trvání činnosti y_{ij}. V případě typu PERT je vygenerován optimistický odhad délky trvání činnosti a_{ij}, modální odhad délky trvání činnosti m_{ij} a pesimistický odhad délky trvání činnosti b_{ij}. Následně funkce *generateProject()* vrací objekt typu JSON s vygenerovanými stavy a činnostmi zpět do formuláře.

Ve formuláři je objekt s vygenerovaným systémem převeden pomocí metody *JSON.stringify()* do textového řetězce a vložen jako hodnota atributu *value* do HTML elementu s tagem *<input>*. Tento element je ve formuláři skryt. Po vložení vytvořeného systému do formuláře je navíc zobrazena informace o systému na konci formuláře, ve které

je vypsán název projektu, typ analýzy, počet stavů a počet činností systému. Po stisknutí tlačítka *Add* se formulář s daty o systému zašle na serverovou část aplikace, kde je vytvořen projekt se stavy a činnostmi. Objekty jsou následně uloženy do databáze a aplikace je přesměrována zpět do hlavního prostředí s adresářem projektů, ve kterém se na konci nachází nově vytvořený náhodný systém. V případě chyby (např. duplicita názvu projektu) není projekt uložen do databáze a aplikace zobrazí zprávu s textem chyby.

Uživatelský manuál

V aplikaci byla vytvořena nápověda obsahující informace o grafických objektech, které jsou použity k sestavení grafu. Uživatel ji může zobrazit v hlavním prostředí projektu, kdy se v hlavní nabídce aplikace pod položkou *Settings* nachází sekce označena názvem *Help*. Tato sekce obsahuje položku *Hint* (nápověda), která otevře do prostředí projektu modální okno s nápovědou pro uživatele. Modální okno s nápovědou lze otevřít i pomocí pravého stisknutí na myši v pozadí grafu, kdy se otevře nabídka obsahující položku *Help*. V modálním okně jsou popsány informace o značení stavů a činností v grafu. Dále obsahuje odkaz (*Complete User Manual*) na kompletní uživatelský manuál aplikace.

Pro aplikaci byl napsán kompletní uživatelský manuál ve formátu PDF, ve kterém je v češtině popsáno kompletní ovládání aplikace. Manuál se nachází v položce *Settings* hlavní nabídky aplikace. V této položce je v sekci *Help* položka *Manual*, která otevře soubor formátu PDF s manuálem v nové kartě prohlížeče. Odkaz na manuál je vytvořen pomocí HTML elementu s tagem *<a>*, kde atribut *href* obsahuje cestu k souboru s manuálem v PDF. Navíc byl nastaven atribut *target* na hodnotu *_blank* pro otevření dokumentu (manuálu) v nové kartě prohlížeče. Kompletní uživatelský manuál lze zobrazit také v nápovědě, kdy odkaz s názvem *Complete User Manual* také otevře manuál v nové kartě prohlížeče.

3.9 Zabezpečení

V této části práce je detailněji popsáno zabezpečení vytvořené aplikace. Nejdříve je zde zmíněno řešení bezpečnosti přenosu přihlašovacích údajů uživatele mezi klientskou a serverovou částí aplikace. Následně jsou zde zmíněna bezpečnostní rizika, která mohou nastat při práci v uživatelském prostředí aplikace a jejich řešení.

Autentifikace uživatele

Webovou aplikaci lze použít pouze pod vytvořeným uživatelským účtem. V databázi se o uživateli uchovává jméno, příjmení, emailová adresa, heslo a datum vytvoření účtu. ID uživatele je generováno automaticky. Uživatelský účet slouží k zajištění bezpečnosti a správného fungování aplikace, kdy ID uživatele slouží k rozpoznání uložených projektů.

Schéma projektu obsahuje povinný atribut *userID*, který představuje ID konkrétního uživatele.

V kontroleru *userController.js* byly definovány funkce pro autentifikaci (přihlášení) a uložení (registrace) nového uživatele. K registraci uživatele byla použita knihovna password-validator (https://www.npmjs.com/package/password-validator), ve které lze definovat validační schéma, podle kterého musí být vytvořeno uživatelské heslo. Validační schéma představuje pravidla, která musí být splněna pro vytvoření uživatelského hesla. V aplikaci bylo definováno schéma v souboru *passwordValidator.js*, které definuje pro heslo následující pravidla:

- minimálně 8 znaků,
- maximálně 30 znaků,
- alespoň jeden znak velkým písmenem,
- alespoň jeden znak malým písmenem,
- alespoň 2 číslice,
- bez mezer.

V komunikaci mezi klientskou a serverovou částí nelze zasílat heslo v původním tvaru. Z důvodu bezpečnosti se hesla zasílají v tzv. hashovaném tvaru. V aplikaci byla použita knihovna bcryptjs (https://www.npmjs.com/package/bcryptjs), která obsahuje funkce *hash()* pro registraci a *compare()* pro autentifikaci uživatele. Funkce *hash()* přijímá jako vstup heslo v textovém řetězci a tzv. sůl, která představuje náhodně vygenerovaný řetězec, který je k heslu přidán navíc. Sůl je generována pomocí funkce *genSalt()* knihovny bcryptjs, která na vstupu přijímá požadovanou délku řetězce (soli). V aplikaci je tento řetězec nastaven na délku 10. Heslo v hashovaném tvaru je následně uloženo do databáze. Do aplikace se přihlašuje pomocí emailové adresy a hesla. Funkce *compare()* knihovny bcryptjs slouží k porovnání hesla zadaného při přihlášení s heslem uloženým v databázi. Po úspěšné autentifikaci je ID uživatele uloženo do session a uživatel zůstane přihlášen v aplikaci, dokud nevyprší platnost session nebo se z aplikace neodhlásí. Expirační doba (platnost) session je v aplikaci nastavena na 30 minut. Pro odhlášení uživatele z aplikace byla v kontroleru *userController.js* vytvořena funkce, ve které dochází ke smazání aktuální session. Po smazání session přesměruje aplikace uživatele zpět na stránku s přihlášením.

Bezpečnostní rizika

Vytvořená aplikace využívá databázi pro persistentní ukládání dat uživatele. Data, která jsou odesílána z klientské části aplikace na server, nemohou vstupovat do databáze bez dodatečné kontroly. Spousta útoků na databáze probíhá přes neošetřené vstupy

aplikace, kdy útočník vloží (vsune) pozměněný dotaz na data v databázi. Přes pozměněný dotaz může útočník získat citlivé osobní informace o uživatelích aplikace. Další bezpečnostní hrozba spočívá ve vložení cizího objektu (např. skript) s nebezpečným kódem do textového výstupu aplikace, kdy se tento objekt stane součástí výstupu aplikace a může způsobit uživatelům problémy (ztráta dat, stažení nebezpečného softwaru). V následující části je detailněji popsána bezpečnostní hrozba NoSQL Injection.

NoSQL Injection

NoSQL Injection představuje bezpečnostní hrozbu, ve které útok probíhá vsunutím pozměněného (nebezpečného) dotazu na data v databázi. Pozměněný dotaz může následně útočníkovi vrátit data z databáze, která mu nepatří. Zejména se jedná o citlivé osobní údaje uživatelů aplikace (např. informace o kreditní kartě). Pozměněný (nebezpečný) dotaz může začínat znakem „$" nebo obsahovat znak „.". Tyto znaky se v databázích typu NoSQL často využívají jako operátory pro hledání konkrétních dat. Prevence proti útokům typu NoSQL Injection spočívá ve filtrování nebezpečných znaků dotazu před vyhledáváním dat v databázi. Nebezpečné znaky dotazu jsou buď odstraněny anebo jsou nahrazeny jiným znakem. [20]

Ve vytvořené aplikaci byl proti bezpečnostní hrozbě NoSQL Injection použit modul Express Mongoose Sanitize (https://www.npmjs.com/package/express-mongo-sanitize), který filtruje nebezpečné dotazy na straně serveru. Výchozím voláním modulu Express Mongoose Sanitize jsou všechny nebezpečné znaky z dotazu odstraněny. V modulu byla nastavena volba *replaceWith*, která nebezpečný znak nahradí jiným znakem. V aplikaci jsou nebezpečné znaky v dotazech nahrazeny znakem „_".

Zabezpečení HTTP hlavičky

Hypertext Transfer Protocol (zkráceně HTTP) funguje na principu request a response (dotaz a odpověď). Jednotlivé zprávy, které se předávají v rámci protokolu, obsahují tzv. hlavičky (anglicky headers). Hlavičky protokolu HTTP obsahují dodatečné informace o komunikaci probíhající skrze server (např. informace o použité technologii, informace o typu posílaných dat, aj.). Ne všechny informace, které jsou obsaženy v hlavičce, můžou zůstat veřejné pro uživatele aplikace. Dost útoků na webové aplikace probíhá skrze HTTP hlavičky, jelikož obsahují informace, které mohou útočníkovi ulehčit práci. Hlavička obsahuje například i informaci o technologii, pomocí které byla webová aplikace vytvořena (např. Express). V hlavičce se jedná o řádek s názvem klíče *X-Powered-By*. Pokud útočník zjistí konkrétní typy technologií, které aplikace využívá, může se cíleně zaměřit pouze

na bezpečnostní hrozby, které tyto technologie postihují. Minimálně tato informace je doporučována skrýt před uvedením vytvořeného softwaru na trh.

Pro zabezpečení hlavičky protokolu HTTP byl ve vytvořené aplikaci použit modul Helmet.js (https://helmetjs.github.io/). Helmet.js je kolekce middlewarů, které umožňují nastavit HTTP hlavičku v komunikaci mezi klientem a serverem, popřípadě pozměnit zobrazené údaje v ní a tak ji zabezpečit před útoky. Výchozím voláním *helmet()* jsou použity všechny middlewary kolekce s výchozími parametry. Z kolekce Helmet.js nemusí být použity všechny middlewary. Například pro schování řádku *X-Powered-By* stačí z kolekce zavolat middleware s názvem *Hide Powered By*. Jednotlivé middlewary se dají nainstalovat i samostatně jako moduly. Ve vytvořené aplikaci byl modul Helmet.js použit s výchozími parametry až na middleware s názvem *Content Security Policy*, který chrání před útoky z cizích skriptů. V *Content Security Policy* byly nastaveny skripty a zdroje dat, které webová aplikace využívá.

Cross-site scripting

Cross-site scripting (zkráceně XSS) je typ bezpečnostní hrozby, kdy útočník vloží cizí skript (JavaScriptový kód) do neošetřeného vstupu webové aplikace. Útočníkův skript se může stát dočasně (non-persistent XSS) nebo trvale (persistent XSS) součástí napadené webové aplikace. Dalším typem útoku XSS je tzv. DOM based XSS, který napadá aplikaci lokálně pomocí existujícího klientského skriptu, který je aplikací považován jako bezpečný. V tomto typu je útok situován přes DOM (Document Object Model), který umožňuje přístup pomocí JavaScriptu k prvkům HTML stránky.

Prevence proti XSS útokům spočívá v ošetření vstupů do aplikace a filtrování nebezpečných (podezřelých) znaků. Častým způsobem ochrany před XSS útokem je nahrazení podezřelého znaku jeho HTML entitou. HTML entita je sekvence znaků, která začíná znakem „&“ a končí středníkem a využívá se pro reprezentaci znaků, které jsou chápany jako součást samotného jazyka HTML. Zejména se jedná o znaky „<“ a „>“, které slouží k definici HTML tagu.

Ve vytvořené aplikaci byl pro zabezpečení proti bezpečnostní hrozbě XSS použit middleware *XSS Filter*, který je součástí kolekce Helmet.js. Nebezpečné znaky jsou nahrazeny HTML entitou. Například znaky „<“ a „>” jsou nahrazeny entitou „<“ a „>“. [21] [22]

3.10 Porovnání s komerčními softwary

V této podkapitole je porovnána vytvořená aplikace s komerčními softwary (viz Kapitola 2). Porovnání se nachází v tabulce (viz Tabulka 3), ve které první sloupec představuje vybrané funkce, které jsou mezi softwary navzájem porovnány.

Tabulka 3: Porovnání vytvořené aplikace s komerčními softwary

	Vytvořená aplikace	LucidChart	Microsoft Project	PERT Calculator
Grafické rozhraní	✓	✓	✓	✓
Grafické výstupy	✓	✓	✓	×
Tvorba diagramu	✓	✓	✓	×
Výpočet CPM	✓	×	✓	×
Výpočet PERT	✓	×	×	✓
Výpočet Monte Carlo	✓	×	×	×
Uložení souboru do počítače	JSON	×	CSV, Project, XLS, XML	×
Načtení souboru z počítače	JSON	×	CSV, Project, XLS, XML	×
Export diagramu	PNG, SVG	JPEG, PDF, PNG, SVG	PDF, XPS	×
Generátor náhodného projektu	✓	×	×	✓
Uživatelský manuál	✓	✓	✓	✓

4 Ověření správnosti výpočtu výsledků

V poslední kapitole této práce je popsáno ověření správnosti výpočtu aplikace na základě testovacího příkladu a porovnání výsledků z aplikace s komerčními softwary na trhu. Nejdříve je zde uveden testovací příklad projektu, pro který byla ve vytvořené aplikaci nalezena kritická cesta. Dále jsou porovnány výsledné hodnoty z vytvořené aplikace s hodnotami z komerčních softwarů a s hodnotami vlastního analytického řešení.

4.1 Testovací příklad

Ve vytvořené aplikaci byl vytvořen projekt typu PERT reprezentující testovací příklad (viz Příloha D), který se skládá z následujících objektů:

- 30 činností (z toho 3 fiktivní),
- 17 stavů.

Odhady délky trvání činnosti byly u všech činností projektu zadány v hodinách. Dále byl ve vytvořené aplikaci proveden výpočet metody PERT a nalezení kritické cesty projektu. Výsledky analýzy jsou shrnuty v tabulce (Tabulka 4).

Tabulka 4: Výsledky testovacího příkladu ve vytvořené aplikaci

Počet kritických činností v projektu	10
Počet kritických cest v projektu	1
Střední hodnota délky trvání projektu	940/6
Rozptyl	344/36

Pro porovnání kritické cesty byl vybrán software Microsoft Project, ve kterém byl testovací projekt definován v tabulce. U každé činnosti byla délka trvání zadána jako střední hodnota délky trvání činnosti μ_{ij} v nezkrácené formě bez dělení. Dále byl projekt zobrazen jako síťový diagram. V tomto zobrazení byla již vykreslena kritická cesta projektu a po sečtení délek trvání dílčích činností projektu vyšla délka projektu s hodnotou *940*, která odpovídá hodnotě získané z vytvořené aplikace. Zobrazená kritická cesta v aplikaci Microsoft Project odpovídá zobrazené kritické cestě ve vytvořené aplikaci.

Pro výpočet metody PERT byla vytvořená aplikace porovnána s komerčním softwarem PERT Calculator (viz Kapitola 2.3) a dále s vlastním analytickým řešením, při kterém byly využity tabulkové hodnoty pro hodnotu *z*. Pravděpodobnosti získané z vytvořené aplikace,

softwaru PERT Calculator a vlastního analytického výpočtu byly porovnány v následující tabulce (viz Tabulka 5).

Tabulka 5: Porovnání výsledných hodnot metody PERT

Typ výpočtu	Vytvořená aplikace	PERT Calculator	Vlastní výpočet
P(X < 145)	0,01 %	0,01 %	0,01 %
P(X < 150)	1,55 %	1,55 %	1,55 %
P(X < 155)	29,49 %	29,49 %	29,49 %
P(X < 160)	85,96 %	85,96 %	85,96 %
P(X < 165)	99,65 %	99,65 %	99,65 %
P(X < 170)	100 %	100 %	100 %

Závěr

Cílem diplomové práce bylo seznámit se s metodami užívanými v síťové analýze a vytvořit vlastní aplikaci řešící metodu kritické cesty a její pravděpodobnostní nadstavbu PERT. V kapitole 1 byly obě metody popsány společně s jejich použitím při plánování projektů. Dále v této kapitole byly zpracovány informace o simulační metodě Monte Carlo, která slouží k modelování náhodné veličiny X a o jejím použití s metodou PERT.

V kapitole 2 byla zpracována rešerše existujících komerčních softwarů a knihoven, které se zabývají problematikou síťové analýzy. V kapitole 2.5 byly porovnány statistické knihovny SciPy a jStat. Pro porovnání byla vytvořena aplikace v programovacím jazyce Python, která počítala pravděpodobnost s použitím knihovny SciPy a s použitím knihovny jStat.

Ve třetí části (Kapitola 3) byla popsána vlastní aplikace, která byla vytvořena jako webová s použitím technologií Node.js a Express.js. Pro trvalé ukládání dat byla využita NoSQL databáze MongoDB a k zobrazení diagramu byla použita knihovna GoJS. Projekt lze vytvořit jak v prostředí aplikace, tak i načtením ze vstupního souboru. Diagram lze uložit do souboru typu JSON nebo exportovat do souboru grafického formátu SVG nebo PNG. Aplikace dále nabízí generátor náhodných systémů, který na základě počtu vstupních stavů a rozsahů délky trvání činností vygeneruje náhodný projekt. Pro simulaci Monte Carlo byla použita technologie Web Workers, která spouští skript se simulací v pozadí aplikace, kdy během běžící simulace lze nadále využívat hlavní prostředí aplikace. Aplikace byla zabezpečena proti bezpečnostním hrozbám a to jak kontrolou vstupů do aplikace, tak i s pomocí modulů zajišťující bezpečnost webových aplikací. V kapitole 3.10 byla aplikace porovnána s komerčními softwary, kdy byla vytvořena tabulka obsahující vybrané funkce, které byly mezi softwary porovnány.

V závěrečné kapitole 4 byly ověřeny výsledky vytvořené aplikace s analyticky zjištěnými výsledky a s výsledky komerčních softwarů. Pro ověření byl vytvořen v aplikaci testovací příklad pro metodu PERT. Ke kontrole správného vykreslení kritické cesty v projektu byl testovací příklad zadán v aplikaci Microsoft Project, kde se zobrazená kritická cesta shodovala s kritickou cestou zobrazenou ve vytvořené aplikaci. Pro porovnání výsledných pravděpodobností byly výsledky z vytvořené aplikace porovnány s výsledky ze softwaru PERT Calculator a s vlastními výpočty, kdy bylo použito tabulkových hodnot pro výpočet pravděpodobnosti. Výsledné hodnoty se mezi vytvořenou aplikací, komerčním softwarem a vlastními výpočty shodovaly.

Seznam použité literatury

[1] RÁLEK, Petr, Josef NOVÁK a Josef CHUDOBA. *Metody užívané v logistice*. Technická univerzita v Liberci: skripta, 2009 [cit. 2021-11-15].

[2] DŘÍMAL, Jiří, David TRUNEC a Antonín BRABLEC. *Úvod do metody Monte Carlo* [online]. Masarykova univerzita, Brno, 2006 [cit. 2021-12-12]. Dostupné z: https://www.physics.muni.cz/~trunec/mc.pdf

[3] HERRERÍAS-VELASCO, José Manuel, Rafael HERRERÍAS-PLEGUEZUELO a Johan RENÉ VAN DORP. *Revisiting the PERT mean and variance* [online]. [cit. 2021-12-14]. Dostupné z: doi: https://doi.org/10.1016/j.ejor.2010.08.014

[4] MALCOLM, D. G., J. H. ROSEBOOM, C. E. CLARK a W. FAZAR. *Application of a Technique for Research and Development Program Evaluation* [online]. 1959 [cit. 2021-12-14]. Dostupné z: doi: https://doi.org/10.1287/opre.7.5.646

[5] *Lucidchart* [online]. [cit. 2022-01-15]. Dostupné z: https://www.lucidchart.com/pages/

[6] Microsoft Project. *Microsoft* [online]. [cit. 2022-01-15]. Dostupné z: https://www.microsoft.com/cs-cz/microsoft-365/project/project-management-software

[7] *Good Calculators: Free Online Calculators* [online]. [cit. 2022-01-15]. Dostupné z: https://goodcalculators.com/

[8] *AnyChart* [online]. [cit. 2022-01-23]. Dostupné z: https://www.anychart.com/

[9] Window.sessionStorage. *MDN Web Docs* [online]. [cit. 2022-01-30]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage

[10] What is Handlebars? *Handlebars* [online]. [cit. 2022-02-10]. Dostupné z: https://handlebarsjs.com/guide/#what-is-handlebars

[11] The Model View Controller Pattern – MVC Architecture and Frameworks Explained. *FreeCodeCamp* [online]. [cit. 2022-02-14]. Dostupné z: https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/

[12] What is HTTP? *W3Schools* [online]. [cit. 2022-02-19]. Dostupné z: https://www.w3schools.com/whatis/whatis_http.asp

[13] HTTP Request Methods. *W3schools* [online]. [cit. 2022-03-15]. Dostupné z: https://www.w3schools.com/tags/ref_httpmethods.asp

[14] Method-override. *Npm* [online]. [cit. 2022-03-15]. Dostupné z: https://www.npmjs.com/package/method-override

[15] Get Started with GoJS. *GoJS* [online]. [cit. 2022-04-10]. Dostupné z: https://gojs.net/latest/learn/

[16] Introduction. *Socket.IO* [online]. [cit. 2022-04-10]. Dostupné z: https://socket.io/docs/v4/

[17] Using Web Workers. *MDN Web Docs* [online]. [cit. 2022-04-20]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers

[18] Window.localStorage. *MDN Web Docs* [online]. [cit. 2022-04-23]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

[19] Blob. *MDN Web Docs* [online]. [cit. 2022-04-25]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Blob

[20] What is NoSQL Injection and How Can You Prevent It? *Invicti* [online]. [cit. 2022-05-10]. Dostupné z: https://www.invicti.com/blog/web-security/what-is-nosql-injection/

[21] Cross-site scripting. *PortSwigger* [online]. [cit. 2022-05-10]. Dostupné z: https://portswigger.net/web-security/cross-site-scripting

[22] Cross Site Scripting (XSS). *OWASP* [online]. [cit. 2022-05-10]. Dostupné z: https://owasp.org/www-community/attacks/xss/

Přílohy

A Ukázka aplikace pro srovnání knihoven SciPy a jStat

PERT

Add Task | Clear Data

#	Task Name	Optimistic	Most Likely	Pessimistic
1.	A1	1	6	9
2.	A2	2	5	9
3.	A3	5	6	10
4.	A4	5	10	12

Number of Critical Tasks: 4

Desired Completion Time [ ] Calculate

Results

#	Task Name	Expected Time	Standard Deviation	Variance
1.	A1	5.667	1.333	1.778
2.	A2	5.167	1.167	1.361
3.	A3	6.5	0.833	0.694
4.	A4	9.5	1.167	1.361
		Σ		Σ
		26.834		5.194

Desired Completion Time: 28, Z: 0.512

(SciPy) P: 69.567 %

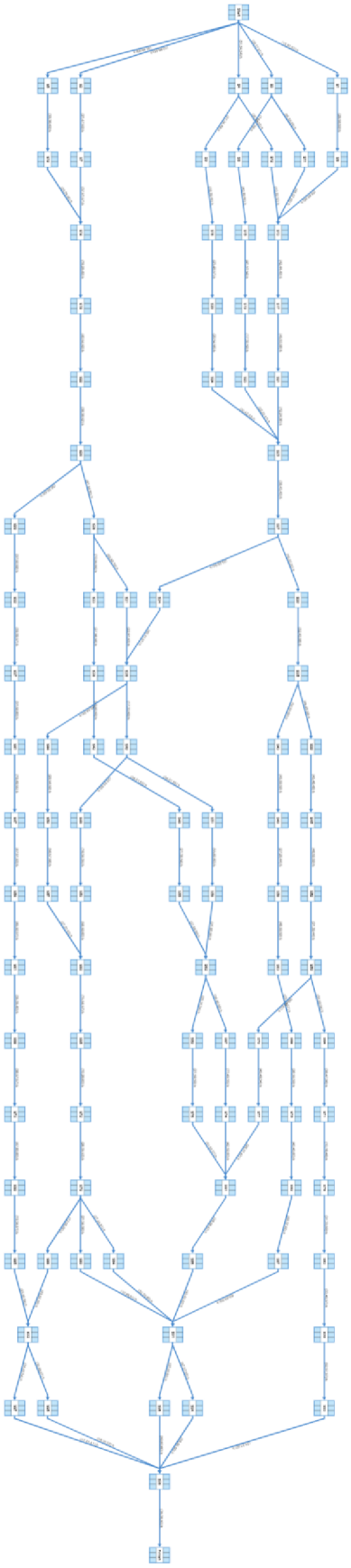
(jStat) P: 69.564 %

B Obsah přiloženého CD

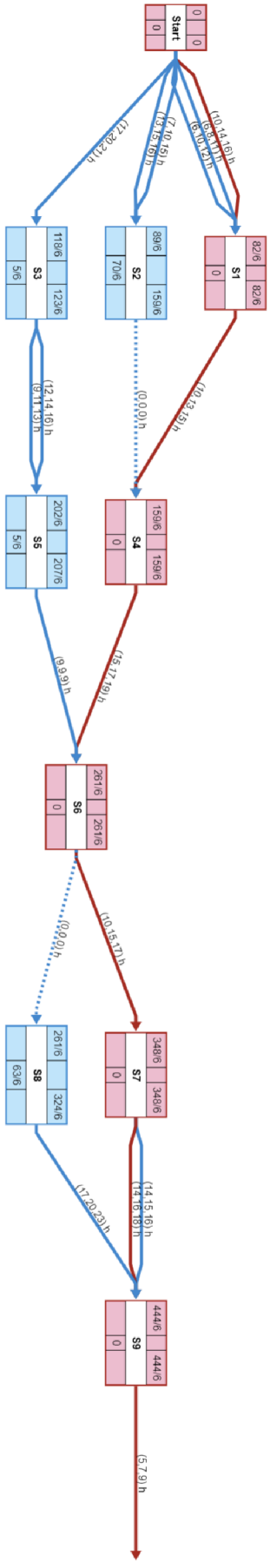
Obsah přiloženého CD je členěn do následujících adresářů:

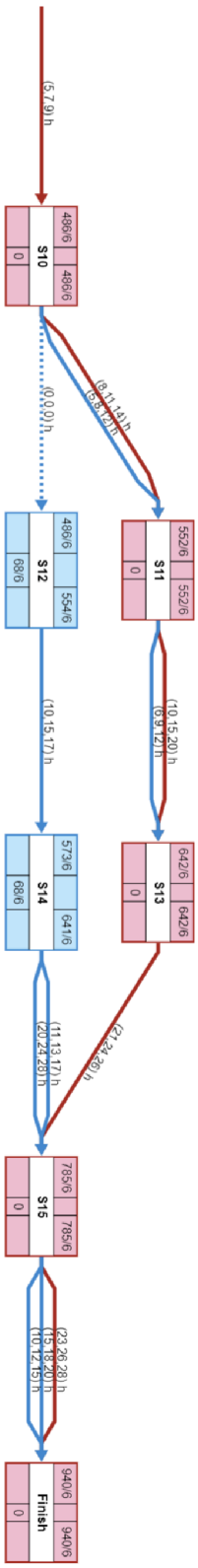
- diplomova_prace_text (textová zpráva)
    - diplomova_prace_drnec.pdf
- manual (manuál k aplikaci)
    - manual_naa.pdf (kompletní uživatelský manuál)
- porovnani_knihoven
    - PERT_Calculator (složka obsahující aplikaci v Python s frameworkem Flask pro porovnání knihoven SciPy a jStat)
- testovaci_priklad (testovací příklad pro vytvořenou aplikaci a Microsoft Project)
    - TestovaciPriklad.json
    - TestovaciPriklad.mpp
- zdrojovy_kod (zdrojový kód programu)
    - NAA (složka obsahující kompletní aplikaci vytvořenou s Node.js, Express.js)

C Generovaný projekt



D Testovací příklad




(5,7,9) h
S10
0
486/6
486/6
(0,0,0) h
(8,11,14) h
(9,9,17) h
S12
68/6
486/6
554/6
S11
0
552/6
552/6
(10,15,17) h
(10,15,20) h
(9,9,12) h
S14
66/6
573/6
641/6
S13
0
642/6
642/6
(11,13,17) h
(20,24,28) h
(21,24,29) h
S15
0
785/6
785/6
(23,26,28) h
(15,18,20) h
(10,12,15) h
Finish
0
940/6
940/6