**Univerzita Hradec Králové**

**Fakulta informatiky a managementu**

**Katedra informatiky a kvantitativních metod**

# Vizuální podpora výuky předmětů zabývajících se teorií grafů a grafovými algoritmy

Diplomová práce

Autor: Markéta Šťastná
Studijní obor: Aplikovaná informatika

Vedoucí práce: RNDr. Andrea Ševčíková

Hradec Králové duben 2017

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracovala samostatně a s použitím uvedené literatury.

V Hradci Králové dne 30. 4. 2017 Markéta Šťastná

Poděkování:

Děkuji vedoucí diplomové práce, paní doktorce Ševčíkové, za cenné připomínky a odborné vedení při tvorbě této práce.

# Anotace

Diplomová práce se zabývá tvorbou aplikace vhodné pro výuku důkazů matematických vět a související výrokové logiky. Aplikace se skládá z vizualizací důkazů z teorie grafů, ale její součástí jsou také vizualizace pro podporu výuky negací výroků a vět. Práce nejdříve představuje teorii související s důkazy v diskrétní matematice, poté obsahuje analýzu aplikace a ukázku podobných nástrojů. Následně popisuje technologie použité k implementaci aplikace a výsledné řešení.

# Annotation

## Title: Visual learning support for courses dealing with graph theory and graph algorithms

This Diploma Thesis focuses on creation of application suitable for learning proofs of mathematical theorems and related propositional calculus. The application consists of theory of graphs proofs visualization, but it also includes visualizations to support teaching of propositions and theorems negations. The thesis first introduces the theory related to discrete mathematics proofs. Then it contains an analysis of application and shows similar tools as well. The paper also describes technologies used to implement the application and the final solution.

# Obsah

# Seznam tabulek



# Seznam grafů



# Seznam obrázků

# Seznam zdrojových kódů

# 1 Úvod

Vývoj v oblasti webových technologií umožňuje stále rozvinutější a dynamičtější aplikace použitelné k různým účelům a cílům. Tyto nové možnosti v oblasti webových stránek, aplikací a prohlížečů se dají velmi dobře využít i v oblasti výuky, a to jak v učebně, tak i v prostředí domova. Takové moderní e-learningové nástroje mohou posloužit pro oživení výukových hodin, ale taktéž zaujmou studenty, kteří se díky nim mohou lépe a s větším nadšením připravovat i doma. Obor teorie grafů, který spadá pod odvětví diskrétní matematiky, se jeví být vhodnou oblastí pro vizualizační aplikace, a to díky možnému grafickému znázornění mnoha jeho problémů. Nedílnou součástí matematiky je důkaz. Také ve studiu teorie grafů na Fakultě informatiky a managementu UHK se studenti zabývají dokazováním tvrzení, což je velice důležité pro správné porozumění matematické věty a její interpretace či využití. Tato práce se zabývá tématem důkazů matematických vět v oblasti teorie grafů, protože důkazy jsou pro studenty náročné a obtížně představitelné. Praktická část práce se věnuje tvorbě aplikace, která by mohla studentům pomoci lépe pochopit dokazování vět v této oblasti. Teoretická část práce vysvětlí následující témata:

- výroková logika a její znalost potřebná při dokazování matematických vět,
- základní pojmy teorie grafů,
- typy důkazů v matematice.

V poslední době dochází k velkému rozvoji front-endových nástrojů, jako je například jazyk JavaScript a jeho nadstavby TypeScript a Babel, a zároveň se také zvětšuje jejich podpora v prohlížečích. Proto je možné kvalitněji vyvíjet webové aplikace, jejichž kód je vykonáván na straně klienta. Ke spuštění takové aplikace není nutný server ani databáze, což ulehčuje správu hardwarových prostředků. Vzniklo také mnoho frameworků nad výše zmíněnými jazyky, díky kterým je možné lépe pracovat s vizuální stránkou programů. To je velmi vhodné pro použití v edukačních materiálech. Při tvorbě aplikace pro podporu výuky důkazů vyvstaly otázky, které jsou v práci zodpovězené:

- Jaké nástroje je vhodné použít k naprogramování aplikace vzhledem k jejich rozšířenosti a vhodnosti použití pro daný účel?

- Jaké jsou rozdíly při použití těchto nástrojů při vykreslování požadovaných vizualizací, a to zejména z hlediska rychlosti vykreslování?
- Jaké vyvstaly problémy při programování vizualizační aplikace s pomocí vybraných nástrojů?

## 1.1 Cíl práce

Cílem práce je vytvoření aplikace s vizualizacemi důkazů z diskrétní matematiky, které jsou často součástí výukové látky v předmětech zabývajících se teorií grafů a grafovými algoritmy. Tato oblast se odvíjí od znalostí souvisejících s výrokovou logikou, protože žádný důkaz nelze provést bez znalosti základních pojmů, mezi které patří například implikace, ekvivalence, negace a tak dále. Proto se tato práce zaměřuje taktéž na výukovou podporu těchto pojmů a vizualizaci negací, které bývají leckdy pro studenty složité na pochopení. To se může změnit při použití vhodné interaktivní vizualizace, která studentovi pomůže s vizuální představou dané věty a její negace. Výsledná aplikace by měla být uživatelsky přívětivá, dobře použitelná a přístupná, přičemž zvolené příklady by měly být dostatečně názorné pro studenta.

Vytvoření moderního vizualizačního nástroje může být docíleno pouze použitím aktuálních technologií, které je nutné dobře zvolit. Proto je součástí práce také jejich porovnání a výběr té nejvhodnější pro daný účel.

## 1.2 Metodika zpracování

První část práce obsahuje úvod do problematiky výuky důkazů a jejich vizualizací. Při její tvorbě bylo čerpáno z domácích i zahraničních vědeckých článků jako například Vizuální důkazy ve výuce matematiky [1]. Tématu se ve svých článcích věnují autoři jako Hanna a Sidoli [2] nebo Kilic [3].

Teoretická část práce se zabývá výrokovou logikou, typy důkazů a představením základních pojmů teorie grafů. Základní teoretické informace byla čerpána z knih Teorie grafů a grafové algoritmy [4], Diskrétní matematika [5], Kapitoly z diskrétní matematiky [6] a dalších. O důkazech v matematice pojednává R. Thiele v knize Matematické důkazy [7], která byla také využita ke zpracování dané části práce.

Praktická část se zabývá tvorbou aplikace. Tato nová vizualizační aplikace by měla být moderním výukovým nástrojem, vytvořeným s pomocí inovativních technologií. Celá

oblast dynamických klientských webových aplikací je ve velkém rozvoji, nové knihovny přicházejí doslova každý den. Proto k této oblasti neexistuje velké množství aktuálních knižních zdrojů, a to ani v anglickém jazyce, natož v jazyce českém. Při tvorbě této části práce bylo proto čerpáno zejména z internetových zdrojů, a to hlavně z dokumentací frameworků a dalších technologických nástrojů v anglickém jazyce. Příkladem je příručka jazyka JavaScript [8], webové stránky jazyka TypeScript [9], články o frameworku Bobril [10] a další zdroje zmíněné v závěru práce.

# 2 Výuka důkazů z teorie grafů

Tato část práce obsahuje úvod do problematiky výuky důkazů a shrnuje, proč jsou důkazy pro studenty sice náročnou, ale přesto důležitou součástí matematiky. Teoretické pozadí problému je představeno v kapitolách 3 Výroková logika, 4 Matematické důkazy a 5 Teorie grafů.

Mnoho studií se zabývá přístupem studentů k důkazům. Nejrozšířenějším poznatkem těchto studií je, že pro studenty je dokazování obtížné a pohled studentů na účel důkazu je omezený. Někteří studenti nevědí, že je potřeba důkazů pro ověření tvrzení a pro dokázání se odvolávají na autority (učitele nebo knihu). [11]

## *2.1 Důležitost důkazu*

J. Grabinerová [12] shrnuje, proč jsou důkazy důležité, a uvádí, že důkazy oplývají intelektuálními výzvami a s pomocí matematických důkazů se učíme uvažovat logicky. Štrausová uvádí, že důkazy jsou klíčové nejen při budování matematických pojmů, ale i pro porozumění matematice jako takové. Dále zmiňuje, že uvědomění žáků ohledně nutnosti argumentování a dokazování může pomoci i v jiných oblastech, než je matematika. [1]

## *2.2 Vizualizace ve výuce*

Použití vizualizace ve výuce, a to zejména v e-learningu, vychází z faktu, že lidský mozek si dokáže více zapamatovat informace získané z obrazových dat, než informace získané z textu. Tento fenomén bývá označován jako efekt nadřazenosti obrazu (resp. picture superiority effect). To potvrzuje Snodgrass a kol. [13] (volně přeloženo): *„Zjištění, že obrázkový stimulus je lepší pro poznávací paměť než stimulus verbální, není překvapením. To bylo opakovaně demonstrováno, a to jak mezi obrázky a slovy, tak i mezi slovy vysoce obrazotvornými a méně obrazotvornými. Podle všeho, více asociativních vazeb může být vytvořeno ke stimulům, které mají bohatou vizuální, stejně tak jako verbální stránku, takže se obě doplňují a vylepšují poznání.“*

Bílek a Slabý se v článku [14] zabývají učením z grafické reprezentace v oblasti přírodních a technických věd a zmiňují, že vizuální vnímání se zde zdá být podstatnou součástí

získávání znalostí. Milková [15] uvádí své zkušenosti s použitím e-learningu na Univerzitě Hradec Králové v článku E-learning: Postřehy a zkušenosti. Zde podotýká, že multimediální prezentace a programy bývají součástí e-learningových univerzitních kurzů a slouží k tomu, aby studenti hlouběji pronikali do dané látky. Také zmiňuje výhody virtuálního studijního prostředí, mezi které patří možnost studovat svým tempem a realizovat studium dle svých potřeb.

## 2.3 Vizuální forma důkazů

Srovnáním geometrického myšlení, znalostí a dokazovacích schopností u dvou skupin studentů se zabývala Hulya Kilic. Studenti několika tureckých škol byli v její studii rozděleni na skupinu experimentální a kontrolní. Součástí výuky experimentální skupiny bylo používání dynamického geometrického softwaru. Členové této skupiny dosáhli v podaných testech zřetelného zlepšení v uvažování nad geometrickými problémy včetně lepší schopnosti jejich dokazování. [3]

Hanna a Sidoli [2] se zabývají užitečností vizualizace v určitých aspektech matematického dokazování. Zmiňují, že existují dva hraniční názory na vizualizaci důkazů. Autoři zastávající první z nich uznávají vizualizaci pouze jako přídavek k důkazu, zatímco výzkumníci zastávající druhý postoj si myslí, že je možné vizualizacemi úplně nahradit tradiční důkaz. V závěru zmiňují, že ještě nejsou prozkoumané všechny možné role vizualizace ve výuce matematiky, ale role vizualizace jako důležité pomoci v porozumění matematice je všeobecně přijímána.

Článek [16] o formách důkazů při vyučování se zabývá formou vizualizace důkazů. Zmiňuje, že sémantický potenciál vizualizací není totální a důležitou dovedností matematika je rozeznat, kdy při konstrukci důkazu postupovat sémanticky a kdy syntakticky. To záleží také na daném důkazu a objektech, které se v něm vyskytují. Závěrem uvádí: *„Stručně řečeno, navzdory výhradám které jsme nastínili, uvedené práce se zdají být základem uznání vizuálního uvažování, a to zejména schopnosti snímků zahrnout nápady, které nemohou být provedeny ve formální definici nebo v algebraickém výpočtu.“* [16] (volně přeloženo)

# 3 Výroková logika

Výroková nebo také formální logika je odvětví logiky, které úzce souvisí s matematikou a matematickými důkazy. R. Thiele v knize Matematické důkazy [7] informuje o této logice následovně: *„Formální logika zkoumá souvislost mezi výroky jisté teorie, např. jaké výroky lze odvodit z jiných nebo zda jsou odvozeny korektně.“*

Protože se výroková logika zabývá studiem výroků, je třeba nejdříve definovat pojem výrok.

## *3.1 Výrok*

Výrokem se rozumí sdělení, o kterém je smysluplné prohlásit, zda je pravdivé či nepravdivé (takzvaný princip dvouhodnotovosti). Pokud je toto tvrzení pravdivé, pak je jeho pravdivostní hodnota 1 (v informatice je možné označení hodnotou TRUE datového typu boolean). Pokud nepravdivé, pak je jeho pravdivostní hodnota 0 (FALSE). [7]

Dá se tedy konstatovat, jestli výrok nastane nebo nenastane. Není možné, aby platilo obojí současně (věta o vyloučeném sporu), ale je jisté, že platí alespoň jedna z možností (věta o vyloučeném třetím) [7]. Příkladem takového výroku může být například jednoduchá rovnice ukazující distributivitu násobení vůči sčítání reálných čísel, tj. roznásobení součtu. Rovnice

$$6 \cdot (1 + 2) = (6 \cdot 1) + (6 \cdot 2)$$

může být označena za výrok, což ale neznamená, že výroky můžeme vytvořit pouze o matematických příkladech a operacích. Za výrok se dá označit i obyčejná oznamovací věta popisující svět kolem nás, jako například:

- „Sofie je hlavní město Bulharska.“
- „Petr dostal na vysvědčení tři dobré známky.“
- „Číslo jedna je prvočíslo.“

Věty výše jsou příkladem nejjednodušších výroků. Výroky ale mohou být i složitější, jako například speciální skupina kvantifikovaných výroků a výroky složené, které jsou vysvětleny dále. Za výroky není možné uvažovat například věty rozkazovací či otázky,

protože je nelze ohodnotit z hlediska jejich pravdivosti nebo nepravdivosti [17]. Následují příklady vět, které není možné označit za výroky.

- Ticho!
- Kolik je hodin?
- $x = y$

### 3.1.1 Logický úsudek

Na základě výroků můžeme vytvářet logické úsudky neboli argumenty, protože pravdivost jednoho výroku může být důsledkem pravdivosti jiných výroků. Takovouto činnost, při které usuzujeme pravdivost výroku na základě pravdivosti jiných výroků, nazýváme argumentací. Pražák [18] uvádí: *„Každá argumentace má dvě části: a) výrok, který zdůvodňujeme, nazýváme závěr nebo důsledek; b) výroky, kterými závěr odůvodňujeme, nazýváme předpoklady. Vzniklému celku pak říkáme argument nebo úsudek. Jestliže konjunkce předpokladů implikuje závěr, říkáme, že argument je správný.“*

Příkladem jsou následující výroky, kde první dva výroky lze označit za předpoklady a poslední výrok za závěr.

- „Všichni savci jsou teplokrevní.“
- „Všechny velryby jsou savci.“
- „Všechny velryby jsou teplokrevné.“

Zde je vhodné zmínit se o platnosti logických závěrů. R. Thiele zmiňuje: *„Logický zákon se nevztahuje na pravdivost obsahu nějakého výroku, vztahuje se pouze na platnost formy výroku, ať už se přitom jedná o jev geometrický, číselně teoretický či jiný (třeba biologický). Zákony formální logiky, které umožňují převést pravdivé výroky bez ohledu na jejich obsah na jiné pravdivé výroky, jsou přirozeně také základem důkazových postupů.“* [7]

Platnost závěru o teplokrevných velrybách tedy závisí pouze na jeho tvaru a nezávisí na pravdivosti použitých výroků. Tím, že se slovo „velryby“ nahradí například slovem „šneci“ není narušena formální platnost závěru [7]. Pokud by tedy byly předpoklady pravdivé, byl by pravdivý i závěr. Pokud předpoklady pravdivé nejsou, pak závěr nebude pravdivý, což ale neznamená, že není formálně platný na základě logického úsudku.

### 3.1.2 Negace výroku

Negace výroku, psáno $\neg p$, označuje výrok, který má opačnou pravdivostní hodnotu než původní výrok $p$. Tento výrok je interpretován jako pravdivý, když je původní výrok $p$ nepravdivý, podobně i pro opačný případ (výrok $p$ pravdivý a jeho negace je nepravdivá) [7]. Negaci lze značit také $p'$ a textově pomocí slova $non$.

Negaci výroku v českém jazyce označuje slovní spojení „Není pravda, že…“, které lze při negování umístit před původní výrok. Většinou jde ale utvořit vhodnější formulaci, jak ukazuje následující příklad.

- Výrok: "Sofie je hlavní město Bulharska."
- Negace: "Není pravda, že Sofie je hlavní město Bulharska."
- Vhodnější negace: "Sofie není hlavní město Bulharska."

### 3.1.3 Kvantifikovaný výrok

*„Za kvantifikované výroky považujeme takové oznamovací věty, které udávají přesný počet nebo určitý odhad počtu předmětů, osob apod., které mají uvedenou vlastnost.“* [17] (volně přeloženo)

Kvantifikovaný výrok, jak již název napovídá, označuje spojení výroku s kvantifikátorem. Právě kvantifikátory vyjadřují údaje o počtech objektů ve výroku, které disponují určitou vlastností. Kvantifikátor může být buď univerzální (velký), nebo existenční (malý).

#### 3.1.3.1 Univerzální kvantifikátor

Univerzální, obecný nebo také velký kvantifikátor se značí symbolem ∀. Tento kvantifikátor vyjadřuje, že každý objekt má danou vlastnost (popřípadě žádný prvek nemá danou vlastnost) [17]. Zápis $\forall x$ lze do českého jazyka přeložit jako slovní spojení „pro všechna $x$“, „pro libovolné $x$“ nebo „pro každé $x$“ [18]. Příkladem kvantifikovaných výroků s univerzálním kvantifikátorem mohou být výroky:

- $\forall n \in N: n > 0$
- „Každé prvočíslo je dělitelné pouze číslem jedna a sebou samým.“
- „Všichni savci jsou teplokrevní.“

#### 3.1.3.2 Existenční kvantifikátor

Existenční nebo malý kvantifikátor se značí symbolem ∃. Tento kvantifikátor oznamuje, že existuje alespoň jeden objekt, který má danou vlastnost (popřípadě alespoň jeden objekt nemá danou vlastnost) [17]. Zápis $\exists x$ lze přeložit jako „existuje alespoň jeden prvek $x$“ nebo „pro alespoň jedno $x$ platí“ [18]. Následují příklady výroků s existenčním kvantifikátorem.

- $\exists n \in N: n < 5$
- „Existuje alespoň jeden trojúhelník, který je rovnostranný.“
- „Existuje město, které má více než jeden milion obyvatel.“

#### 3.1.3.3 Ostatní údaje o počtu

Kvantifikované výroky mohou také obsahovat další údaje o počtu objektů splňujících danou vlastnost. Je možné například vyjádřit, že existují právě tři prvky, alespoň čtyři prvky nebo nejvýše dva prvky, které danou vlastnost mají. Takové výroky je možné rozdělit na následující kategorie.

- Právě $n$ prvků má danou vlastnost.
- Alespoň $n$ prvků má danou vlastnost.
- Nejvýše $n$ prvků má danou vlastnost.

Příkladem použití těchto údajů o počtu jsou věty:

- „Snědl jsem právě tři kotlety.“
- „Petr má alespoň dvě dobré známky.“
- „Nejvýše čtyři studenti rozumí negacím.“

### 3.1.4 Negace kvantifikovaného výroku

Jak již bylo řečeno, negace výroku $\neg p$ má opačnou pravdivostní hodnotu než původní výrok $p$. Negování kvantifikovaných výroků může být náročnější z hlediska správné formulace negace. Lze sice před původní výrok umístit slovní spojení „Není pravda, že…“, taková formulace ale nemusí vždy odpovídat logickému užití jazyka, proto se zvláště u kvantifikovaných výroků přistupuje k jinému slovnímu formulování negací, jak ukazuje

následující příklad. Tato forma negace výroku se používá v matematických důkazech, proto je vhodné si ji blíže představit.

- Výrok: „Každé prvočíslo je dělitelné pouze číslem jedna a sebou samým.“
- Negace výroku: „Není pravda, že každé prvočíslo je dělitelné pouze číslem jedna a sebou samým.“
- Vhodnější negace výroku: „Existuje alespoň jedno prvočíslo, které není dělitelné pouze číslem jedna a sebou samým.“

Práce s kvantifikovanými výroky může být někdy pro studenty méně intuitivní, a to hlavně při tvorbě negací. Pražák [18] uvádí následující příklad:

- Výrok: „Všichni skřeti slouží Sauronovi.“
- Nesprávná negace výroku: „Všichni skřeti neslouží Sauronovi.“ Tato věta by znamenala, že nikdo ze skřetů Sauronovi neslouží. Přitom pro popření původního výroku stačí, aby pouze někteří skřeti Sauronovi nesloužili.
- Správná negace výroku: „Existuje alespoň jeden skřet, který neslouží Sauronovi.“

Při negování kvantifikovaných výroků je možné se řídit tabulkou 1. Tabulka 1U negování výroků s kvantifikátory platí, že negace výroku má opačný kvantifikátor než původní výrok.

| $p$ | $\neg p$ |
|---|---|
| Právě $n$ prvků má danou vlastnost | Nejvýše $n-1$ nebo alespoň $n+1$ prvků má danou vlastnost |
| Alespoň $n$ prvků má danou vlastnost | Nejvýše $n-1$ prvků má danou vlastnost |
| Nejvýše $n$ prvků má danou vlastnost | Alespoň $n+1$ prvků má danou vlastnost |
| Každý prvek má danou vlastnost (resp. všechny prvky) | Existuje alespoň jeden prvek, který nemá danou vlastnost |
| Existuje alespoň jeden prvek, který má danou vlastnost | Každý prvek nemá danou vlastnost (resp. všechny prvky). |

**Tabulka 1: Negace kvantifikovaných výroků**

## 3.2 Složený výrok

Z jednoduchých, atomárních výroků lze utvořit výroky složené, které se mohou rozložit na původní výroky. Atomárním výrokem je výrok, který již rozložit nelze. Složené výroky je možné dále skládat do dalších složených výroků. Skládání výroků probíhá pomocí logických spojek zvaných konjunkce, disjunkce, implikace a ekvivalence. [18]

- Konjunkce se značí symbolem ∧ a znamená „a“, „i“ nebo „a zároveň“.
- Disjunkce se značí symbolem ∨ a znamená „nebo“.
- Implikace se značí symbolem ⇒ a znamená „jestliže, pak“.
- Ekvivalence se značí symbolem ⇔ a znamená „právě tehdy, když“.

Následují příklady složených výroků s využitím výše popsaných logických spojek (ve stejném pořadí).

- „V Pardubicích prší a v Hradci Králové svítí slunce.“
- „Půjdeme do školy nebo půjdeme do práce.“
- „Jestliže je dané číslo větší než pět, pak je kladné.“
- „V grafu existuje cesta z vrcholu $v$ do vrcholu $w$ právě tehdy, když v grafu existuje sled z vrcholu $v$ do vrcholu $w$.“

### 3.2.1 Tabulka pravdivostních hodnot

Tabulka 2 obsahuje pravdivostní hodnoty negací výroku a složených výroků při použití výše zmíněných logických spojek.

| $p$ | $q$ | $\neg q$ | $p \wedge q$ | $p \vee q$ | $p \Rightarrow q$ | $p \Leftrightarrow q$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 |

**Tabulka 2: Pravdivostních hodnoty složených výroků**

### 3.2.2 Negace složeného výroku

I u složených výroků je možné vytvořit negaci, jejíž utvoření je složitější a může se řídit tabulkou 3. Pro účely této práce stačí blíže ukázat negovanou implikaci, jelikož tato je často používána při dokazování matematických vět. Negaci složeného výroku

$$p \Rightarrow q,$$

tedy výrok „$p$ implikuje $q$“, jinak řečeno „jestliže platí výrok $p$, pak platí výrok $q$“, lze vytvořit pomocí negace druhého výroku jako

$$p \wedge \neg q.$$

Následuje příklad složeného výroku s implikací a jeho negace.

- Výrok: „Má-li na sobě béžový oblek, pak má hnědé boty.“
- Negace: „Má na sobě béžový oblek a zároveň nemá hnědé boty.“

Tabulka 3 ukazuje negace složených výroků obsahujících konjunkci, disjunkci, implikaci a ekvivalenci (v tomto pořadí), jejich ekvivalentní formu zápisu (na druhém řádku) a odpovídající pravdivostní hodnoty.

| $p$ | $q$ | $\neg p$ | $\neg q$ | $\neg(p \wedge q)$ | $\neg(p \vee q)$ | $\neg(p \Rightarrow q)$ | $\neg(p \Leftrightarrow q)$ |
|---|---|---|---|---|---|---|---|
| | | | | $\neg p \vee \neg q$ | $\neg p \wedge \neg q$ | $p \wedge \neg q$ | $(p \wedge \neg q) \vee (\neg p \wedge q)$ |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

**Tabulka 3: Negace složených výroků a jejich pravdivostní hodnoty**

## *3.3 Matematická věta a axiom*

Jako axiom se označuje výrok, který je pokládán za pravdivý a není nutné ho dále dokazovat. Axiom slouží jako výchozí bod pro další uvažování o důkazech matematických vět.

*„V matematice je zvykem slovem věta označovat každý pravdivý výrok, někdy určitěji říkáme matematická věta. … Matematické věty, tzn. platné výroky o matematických pojmech, jsou skutečnými nositeli matematických poznatků.“* [19]

Matematická věta je tedy tvrzení, jehož pravdivost se musí dokázat, a to pomocí axiomů, definic nebo vět již dříve dokázaných. Jedna věta se může dokázat i několika různými důkazy. *„Logická forma matematických vět je logickou formou výroků. Mimo jednoduché výroky jsou nejčastější logické formy vět implikace nebo ekvivalence. Symbolicky tyto věty zapisujeme $V_1 \Rightarrow V_2$ nebo $V_3 \Leftrightarrow V_4$, kde $V_1$, $V_2$, $V_3$, $V_4$ jsou označení určitých výroků. $V_1$ se nazývá předpoklad a $V_2$ tvrzením věty $V_1 \Rightarrow V_2$. Výroky $V_3$, $V_4$ nazýváme členy ekvivalence $V_3 \Leftrightarrow V_4$.“* [19]

Věty se nejčastěji vyjadřují implikací nebo ekvivalencí a mohou mít i následující formu:

- $A(x) \Rightarrow B(x), \forall x \in M,$
- $A(x) \Leftrightarrow B(x), \forall x \in M.$

### 3.3.1 Věta vyjádřená implikací

Taková věta je tvořena dvěma výroky, které jsou spojené pomocí logické spojky zvané implikace. Větu složenou z výroků pomocí implikace je možné nejen znegovat, ale také provést její obměnu a definovat větu obrácenou.

#### 3.3.1.1 Obměna věty

Obměna věty má stejnou pravdivostní hodnotu jako původní věta. Proto se používá v nepřímých důkazech, jinak také nazývaných důkazy obměněnou implikací, které jsou blíže vysvětleny v kapitole 4.3. Obměnu lze definovat následovně [20]:

$$B'(x) \Rightarrow A'(x), \forall x \in M.$$

#### 3.3.1.2 Obrácená věta

Oproti obměně, obrácená věta nemusí mít stejnou pravdivostní hodnotu jako původní věta. Obrácená věta vypadá dle [20] takto:

$$B(x) \Rightarrow A(x), \forall x \in M.$$

#### 3.3.1.3 Negace věty

Negace věty s implikací odpovídá negaci složeného výroku s touto spojkou, která byla již zmíněna v kapitole o složených výrocích. Negace věty má opačnou pravdivostní hodnotu než původní věta a používá se při důkazech sporem, kterým se věnuje kapitola 4.4. Negace implikace je definována jako:

$$A(x) \wedge B'(x), \forall x \in M.$$

### 3.3.2 Věta vyjádřená ekvivalencí

Tato věta je tvořena dvěma výroky spojenými pomocí logické spojky zvané ekvivalence. V případě důkazu věty založené na ekvivalenci je možné tuto spojku rozložit na dvě implikace a každou dokázat zvlášť. Tím je dokázána celá původní ekvivalence. Rozložení je naznačeno níže. Ekvivalenci

$$A(x) \Leftrightarrow B(x), \forall x \in M,$$

lze rozložit na implikaci „jestliže platí $A$, pak platí $B$“,

$$A(x) \Rightarrow B(x), \forall x \in M,$$

a „jestliže platí $B$, pak platí $A$“,

$$B(x) \Rightarrow A(x), \forall x \in M.$$

# 4 Matematické důkazy

*„Matematický důkaz je deduktivní argument, tj. předpoklady jsou pravdivé nebo za pravdivé považované výroky a pomocí logických pravidel se odvodí tvrzení, které je pak nutně pravdivé.“* [18]

Dle [7] se při důkazu ukazuje, že pokud platí předpoklady, pak je platné i tvrzení, což je málokdy přímo patrné, a proto se důkaz rozkládá na několik postupných kroků, které jsou prováděny podle logických pravidel. Na konci řetězce úsudků je právě dokazované tvrzení, které je díky důkazu přijato jako věta. Důkazy jsou prováděny podle několika ověřených schémat, které jsou v práci dále rozebrány. Patří mezi ně důkaz přímo, nepřímo, sporem a matematickou indukcí.

## *4.1 Axiomatický systém*

Jak již bylo zmíněno, axiom je výrok, který je považován za pravdivý. Při dokazování matematických vět je třeba právě axiomů, které je možné umístit na začátek jako základ dané teorie a pomocí těchto axiomů dokazovat další matematické věty.

*„Aby bylo možno pojmout nějaký výrok do teorie jako větu, musí být toto tvrzení dokázáno v rámci této teorie. V ideální teorii by tedy všechny věty musely být dokazatelné. To však nelze uskutečnit, neboť abychom dokázali nějakou větu, používáme jako předpokladů vět již dokázaných. K důkazu těchto vět však opět potřebujeme dokázané věty a tak dále. Tímto způsobem buď vytvoříme nekonečný řetězec důkazů (regressus in infinitum), anebo se v našich důkazech objeví ve formě předpokladů právě ty věty, které máme dokázat (circulus vitiosus).“* [7]

Jak je vidět, v ideální teorii by nebylo možné vyhnout se nekonečnému dokazování nebo takzvanému úsudku kruhem. Proto existuje určitý počet axiomů neboli základních vět, což jsou zřejmá tvrzení o základních vlastnostech objektů a jejich vztazích, která není možné dokázat [7]. Množina axiomů tvoří axiomatický systém. Raclavský [21] uvádí, že axiomatický systém je zadán formálním jazykem, axiomy a odvozovacími pravidly.

## *4.2 Přímý důkaz*

Přímý důkaz se používá k dokázání tvrzení nebo věty ve tvaru implikace.

### 4.2.1 Přímý důkaz tvrzení

*„Přímý důkaz matematického výroku, tzv. tvrzení, spočívá v tom, že z již dokázaných výroků (tj. z vět) získáme tvrzení po konečném počtu korektních úsudků. Přímé důkazy se v některých speciálních případech nazývají též konstrukce.“* [7]

Pokud je cílem dokázat tvrzení $T$, tak je třeba najít pravdivý výrok $T_1$ a následně dokázat, že platí implikace $T_1 \Rightarrow T$. To lze zajistit sestavením konečného řetězce implikací:

$$T_1 \Rightarrow T_2 \Rightarrow T_3 \Rightarrow \dots \Rightarrow T_n \Rightarrow T,$$

kde $T_1, \dots, T_n$ jsou axiomy nebo dokázaná tvrzení, kde každé je logickým důsledkem předchozího. Na základě této úvahy usoudíme, že platí poslední člen, což je dokazované tvrzení $T$ [22].

### 4.2.2 Přímý důkaz věty ve tvaru implikace

Přímý důkaz může sloužit i k důkazu složeného výroku s implikací. V tomto případě je cílem dokázat platnost implikace $A \Rightarrow B$. Základem je předpoklad, že $A$ platí a následné sestavení řetězce implikací:

$$A \Rightarrow B_1 \Rightarrow B_2 \Rightarrow \dots \Rightarrow B_n \Rightarrow B,$$

kde $B_1, \dots, B_n$ jsou axiomy nebo dokázaná tvrzení. Na základě hypotetického sylogismu je dokázána platnost původní implikace [18].

## *4.3 Nepřímý důkaz*

Pod kategorii nepřímých důkazů se někdy zařazuje i důkaz sporem, který je v této práci uveden zvlášť. Tyto dva typy důkazů spolu mají unikátní vztah, jelikož každý nepřímý důkaz je možné převést na důkaz sporem [23]. Z toho důvodu je důkaz tvrzení principiálně totožný a je zmíněn až v kapitole o důkazu sporem. Nepřímý důkaz věty ve tvaru implikace se někdy nazývá důkaz obměněnou implikací.

### 4.3.1 Nepřímý důkaz věty ve tvaru implikace

Při nepřímém dokazování věty ve tvaru implikace sestavíme její obměnu, tedy $B' \Rightarrow A'$. Jak již bylo řečeno, obměna má stejnou pravdivostní hodnotu jako původní věta. Tuto obměnu pak dokazujeme přímo:

$$B' \Rightarrow B_1 \Rightarrow B_2 \Rightarrow \ldots \Rightarrow B_n \Rightarrow A',$$

kde $B_1, \ldots, B_n$ jsou axiomy nebo dokázaná tvrzení. Tak dojdeme k závěru o platnosti obměny. Protože obměna je ekvivalentní s původní větou, tak jsme dokázali i implikaci $A \Rightarrow B$ [18].

## 4.4 Důkaz sporem

Základní myšlenkou dokazování sporem je vyjít z negace a dojít ke spornému tvrzení (tvrzení, které je sporem s předpokladem nebo s jiným dokázaným tvrzením).

### 4.4.1 Důkaz tvrzení sporem

Důkaz tvrzení $T$ sporem spočívá v předpokládání pravdivosti negace výroku $T'$ a v sestavení řetězce implikací:

$$T' \Rightarrow T_1 \Rightarrow T_2 \Rightarrow \ldots \Rightarrow T_n \Rightarrow S,$$

kde $T_1, \ldots, T_n$ jsou axiomy nebo dokázaná tvrzení. Posledním členem je odvozené tvrzení S, o kterém víme, že je určitě nepravdivé. Tím jsme došli ke sporu, tvrzení $T'$ není pravdivé a pravdivé je naopak původní tvrzení $T$ [18].

### 4.4.2 Důkaz věty ve tvaru implikace sporem

Při důkazu sporem předpokládáme, že platí negace věty $A \wedge B'$. Sestavíme řetězec implikací:

$$(A \wedge B') \Rightarrow B_1 \Rightarrow B_2 \Rightarrow \ldots \Rightarrow B_n \Rightarrow S,$$

kde $B_1, \ldots, B_n$ jsou axiomy nebo dokázaná tvrzení. Stejně jako u výše zmíněného důkazu tvrzení dojdeme k výroku $S$, který je nepravdivý. Negace věty tedy neplatí, platí původní implikace $A \Rightarrow B$.

## 4.5 Důkaz matematickou indukcí

U důkazu matematickou indukcí dokazujeme, že dané tvrzení platí pro všechny prvky množiny. Často se používá u důkazu platnosti tvrzení pro všechna přirozená čísla [7], tedy:

$$\forall\, n \in \mathbb{N}: V(n).$$

Důkaz se rozděluje na dva kroky. Nejdříve je třeba dokázat tvrzení pro nejmenší prvek množiny $n_0$, mnohdy $n_0 = 1$. Následovně se dokáže platnost implikace: „Jestliže platí $V(n)$, pak platí $V(n+1)$“ [24].

1. Platnost $V(n_0)$.
2. Platnost implikace $\forall\, n \in \mathbb{N}: V(n) \Rightarrow V(n+1)$ při platnosti indukčního předpokladu $V(n)$.

Pokud je nejmenším prvkem $n_0 = 1$, pak dle prvního kroku je pravdivý výrok $V(1)$. Podle druhého kroku je pravdivý výrok $V(2)$. Když znovu použijeme druhý krok, dokážeme i pravdivost výroku $V(3)$. Soudíme, že výrok je pravdivý pro všechna $n$. [18]

# 5 Teorie grafů

Teorie grafů je jedním z oborů diskrétní matematiky. John Renze a Eric W. Weisstein [25] představují tento obor následovně: *„Diskrétní matematika je odvětvím matematiky zabývající se objekty, u kterých lze předpokládat odlišné, oddělené hodnoty. Výraz „diskrétní matematika" se proto používá v kontrastu s výrazem „spojitá matematika", což je odvětví zabývající se objekty, které lze měnit plynule (a které zahrnuje například počet). Diskrétní objekty mohou být charakterizovány celými čísly, oproti tomu spojité objekty vyžadují reálná čísla."* (volně přeloženo)

Do diskrétní matematiky bývá zahrnována teorie množin a relací, Booleova algebra, teorie grup, teorie grafů a další. Motivací ke studování teorie grafů může být následující citace z článku Grafové algoritmy v školskej praxi [26] od Pavla Híce a Martina Pokorného: *„Výhodou teorie grafů je totiž fakt, že jejími metodami je možné často jednoduchým a přehledným způsobem řešit problémy z jiných oblastí (fyzika, chemie, sociologie, ekonomie, …), které se na první pohled mohou zdát poměrně komplikované."* (volně přeloženo)

## *5.1 Grafy*

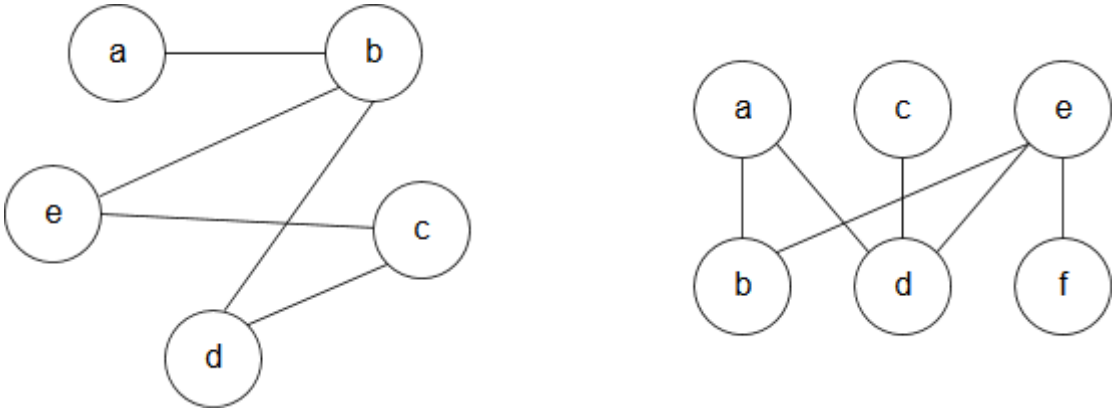
V této části jsou definovány základní pojmy teorie grafů, které jsou využity při popisu vytvořené aplikace. Další pojmy a definice je možné nalézt v knize Teorie grafů a grafové algoritmy [4] nebo Diskrétní matematika [5]. Teorie grafů je založena na zkoumání diskrétních matematických struktur nazývaných grafy. Graf je definován množinou vrcholů a hran, které tyto vrcholy propojují. V této práci je pojmem graf nazýván obyčejný graf, jehož definici předkládá Milková [4]:

*„Obyčejný graf je uspořádaná dvojice* $(V, E)$*, kde V je neprázdná množina vrcholů a E je množina hran, přičemž hrana e z E je dvouprvková podmnožina množiny V. Obyčejný graf nazýváme obyčejným konečným grafem, jestliže množiny V a E jsou konečné. Počet prvků množiny V značíme* $|V|$ *nebo n a počet prvků množiny E značíme* $|E|$ *nebo m."*

### 5.1.1 Grafická reprezentace (nakreslením)

Nejnázornější reprezentací grafu pro člověka je grafická reprezentace, jinak také reprezentace „obrázkem" nebo nakreslením. V tomto případě jsou vrcholy grafu

znázorněny jako body v rovině a jsou spojeny pomocí rovných nebo zkřivených čar (také takzvaných oblouků), které reprezentují hrany [4].




**Obrázek 1: Ukázka grafické reprezentace grafů**

### 5.1.2 Reprezentace maticí sousednosti

V případě složitějších grafů nebo v případě potřeby jiného znázornění (například pro zpracování počítačem nebo pro ilustraci některých grafových algoritmů) se používá reprezentace grafu pomocí matice sousednosti [6]. Tato matice je čtvercová a její řádky a sloupce odpovídají vrcholům v grafu. Pokud jsou vrcholy spojeny hranou, pak je v odpovídající buňce uvedena hodnota 1, pokud nejsou, tak je uvedena hodnota 0. V tabulce 4 je umístěna matice sousednosti odpovídající levému grafu na obrázku 1.

| | *a* | *b* | *c* | *d* | *e* |
|---|---|---|---|---|---|
| *a* | 0 | 1 | 0 | 0 | 0 |
| *b* | 1 | 0 | 0 | 1 | 1 |
| *c* | 0 | 0 | 0 | 1 | 1 |
| *d* | 0 | 1 | 1 | 0 | 0 |
| *e* | 0 | 1 | 1 | 0 | 0 |

**Tabulka 4: Ukázka matice sousednosti**

### 5.1.3 Incidentní a sousední vrcholy

*„Nechť e = {v, w} je hrana grafu G. Vrcholy v, w nazýváme koncovými vrcholy hrany e nebo též vrcholy incidentní s hranou e. Vrchol v nazýváme sousedním vrcholem vrcholu w, resp. vrchol w nazýváme sousedním vrcholem vrcholu v.“* [4]

V matici sousednosti (tabulka 4) je možné vidět, které vrcholy grafu jsou sousedními vrcholy. Například vrchol $a$ je sousedním vrcholem vrcholu $b$, protože jsou spojeny hranou (to je možné vidět v grafické reprezentaci). V matici sousednosti je tak v dané buňce umístěna hodnota 1.

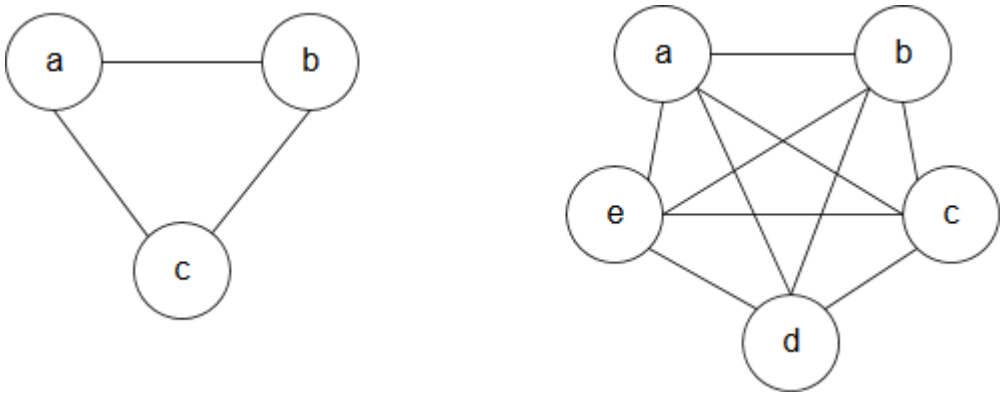
### 5.1.4 Úplný graf a diskrétní graf

Některé specifické typy grafů mají zvláštní označení, jako například úplný nebo diskrétní graf. Předpokládáme, že počet vrcholů $n$ je větší nebo rovný jedné. Definice úplného grafu dle knihy [5] poté zní:

*„Úplný graf na n vrcholech (značí se $K_n$) obsahuje jako hrany všechny neuspořádané dvojice prvků n, takže*

$$V(K_n) = [n],$$
$$E(K_n) = \binom{[n]}{2}.“$$

Jinak řečeno, každý vrchol je spojen hranou s každým dalším vrcholem grafu. Příkladem takových úplných grafů mohou být grafy $K_3$ a $K_5$, které jsou vidět na obrázku 2.




**Obrázek 2: Úplné grafy $K_3$ (vlevo) a $K_5$**

Definice diskrétního grafu dle [5]:

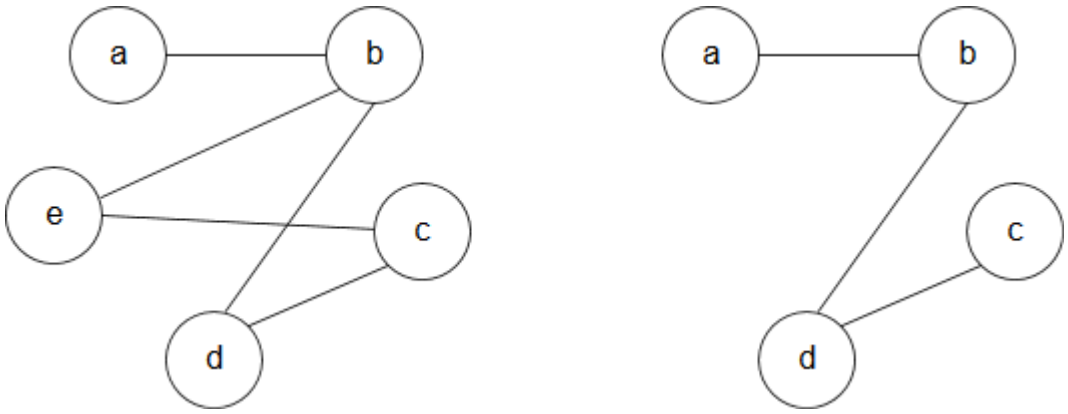
*„Diskrétní graf $D_n$ na n vrcholech nemá žádné hrany:*

$$V(D_n) = [n],$$
$$E(D_n) = \emptyset.“$$

### 5.1.5 Podgraf

*„Graf $G = (V, E)$ je podgraf grafu $G' = (V', E')$, jestliže $V \subseteq V'$ a $E \subseteq E'$.“* [4]

Variantou podgrafu je indukovaný (nebo také plný) podgraf. Ten musí obsahovat všechny hrany, které v původním grafu existují na množině vrcholů tohoto podgrafu [5].




**Obrázek 3: Graf a jeho indukovaný podgraf**

### 5.1.6 Stupeň vrcholu

*„Stupeň vrcholu $v$ v grafu $G$ je číslo rovnající se počtu hran incidentních s vrcholem $v$. Značíme jej $deg_G(v)$ nebo krátce $d_G(v)$.“* [4]

U grafu výše vlevo odpovídá stupeň vrcholu $a$ hodnotě jedna, protože z tohoto vrcholu vychází pouze jedna hrana, zatímco například vrchol $b$ je stupně tři, protože hrany vycházející z toho vrcholu (resp. hrany incidentní s tímto vrcholem) jsou tři.

## 5.2 Souvislost grafu

Ještě před vyslovením definice souvislého grafu je potřeba představit pojmy sled, tah a cesta, protože pomocí cesty lze zavést pojem souvislosti grafu.

### 5.2.1 Sled

*„Sled délky $k, k \geq 0$ v grafu $G$ je posloupnost $(v_0, e_1, v_1, \dots, e_k, v_k)$ kde $e_i = \{v_{i-1}, v_i\}, i = 1, \dots, k$.“* [4]

Sled je tedy posloupnost vrcholů a hran, která prochází grafem. Není možné, aby sled skončil nebo začínal pouze hranou, vždy musí začínat a končit vrcholem. Posloupnost vrcholů a hran se někdy zapisuje pouze jako posloupnost vrcholů.

### 5.2.2 Tah

*„Tah délky $k, k \geq 0$ v grafu G je sled délky k s navzájem různými hranami, tj. pro $i \neq j$ platí $e_i \neq e_j$."* [4]

Tah je také posloupnost vrcholů a hran, ve které se tentokrát nesmí opakovat hrany. Vrcholy se opakovat můžou.

### 5.2.3 Cesta

*„Cesta z u do v v grafu G je sled $u = (v_{,0}, v_1, \ldots, v_k = v)$, ve kterém se každý vrchol $v_i$ objevuje pouze jednou."* [5]

Tedy pro $i \neq j$ platí $v_i \neq v_j$. Nesmí se opakovat jak vrcholy, tak samozřejmě i hrany. Souvisejícím pojmem je pojem kružnice v grafu, ve které je první a poslední vrchol totožný.

### 5.2.4 Souvislý graf

*„Graf G je souvislý, pokud pro každé dva vrcholy x, y existuje v grafu G cesta z x do y."* [5]

Grafy, které výše uvedenou definici nesplňují, jsou grafy nesouvislé. S pojmem souvislosti souvisí i pojem komponenty grafu. Komponenta je maximálním souvislým podgrafem grafu, je to tedy podgraf, který by rozšířením o další vrchol již nebyl souvislý. [4]

# 6 Analýza aplikace

Při tvorbě softwaru je vhodné nejdříve specifikovat, jaké jsou na software kladeny požadavky. Kniha UML2 a unifikovaný proces vývoje aplikací [27] zmiňuje v této souvislosti pojem inženýrství požadavků: *„Inženýrství požadavků (requirements engineering) je termín používaný k popisu aktivit zapojených do zjišťování, dokumentování a údržby množiny požadavků na softwarový systém. Zastupuje odhalování způsobu, jak a k čemu uživatelé daný systém potřebují.“*

## *6.1 Systémové požadavky*

Požadavky jsou určením toho, co by měl systém dělat, ale naopak ne toho, jak bude toto chování implementováno. Je také potřeba rozlišit mezi dvěma druhy požadavků, a to funkčními a nefunkčními. Funkční požadavky (functional requirements) říkají, jaké funkce bude systém nabízet uživateli, zatímco nefunkční požadavky (non-functional requirements) určují vlastnosti daného systému. Při návrzích systémů nelze opomenout pojem UML (Unified Modeling Language, unifikovaný jazyk pro tvorbu diagramů). Tento jazyk ale s požadavky pracuje pouze ve formě případů užití, pro psaní systémových požadavků doporučení neposkytuje. [27]

Arlow a Neustadt [27] přesto doporučují tvorbu a správu systémových požadavků, a to v jednoduché formě znázorněné na obrázku 4.

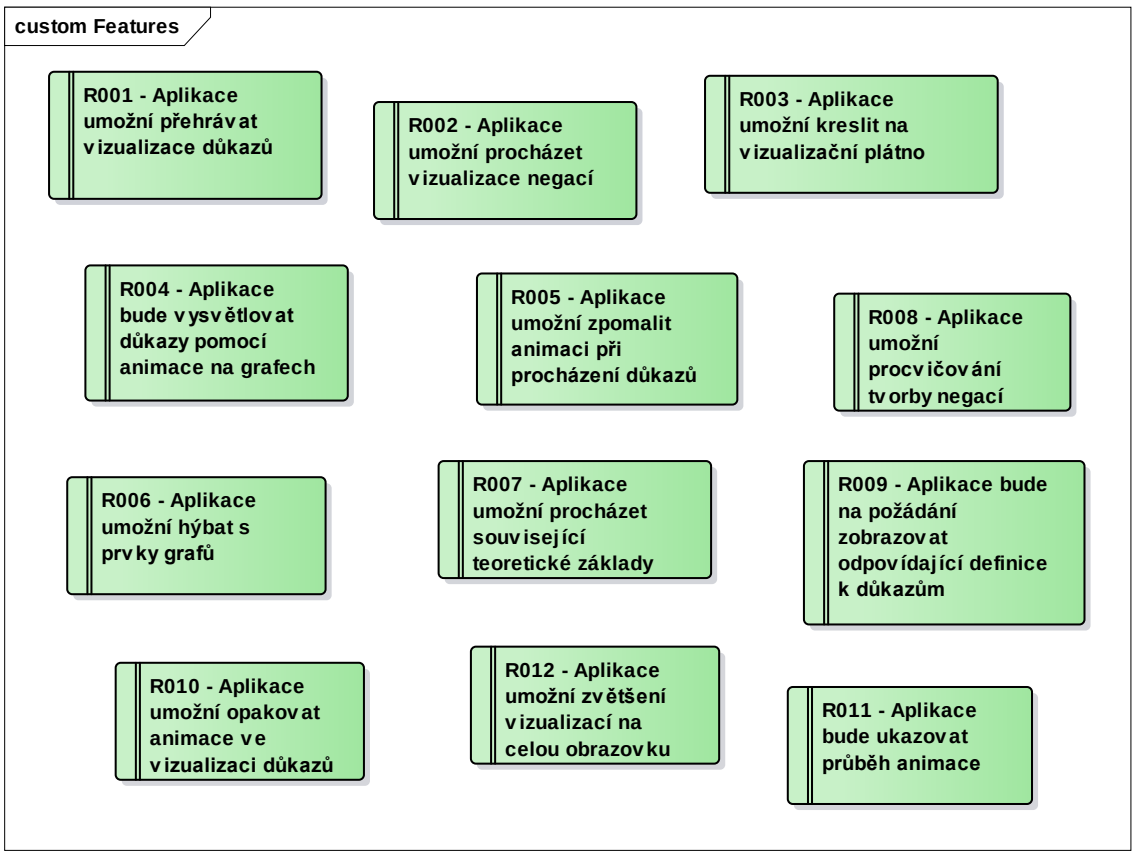

**Obrázek 4: Forma požadavků, zdroj: překresleno dle [27]**

Primárním požadavkem na aplikaci, jejíž tvorbou se tato práce zabývá, je možnost vizualizace vybraných důkazů z teorie grafů. Dále by aplikace měla sloužit hlavně k vizualizaci negací u kvantifikovaných výroků. Všechny požadavky na aplikaci byly rozděleny do zmíněných kategorií funkčních a nefunkčních požadavků.

### 6.1.1 Funkční požadavky

*„Funkční požadavky specifikují, co produkt musí umět udělat. Týkají se akcí, které produkt musí provádět tak, aby byly splněny základní důvody jeho existence.“* [28]

Funkční požadavky byly průběžně konzultovány s vedoucí práce jakožto zadavatelkou projektu. Požadavky na funkce aplikace byly vytvořeny v programu Enterprise Architect a jejich souhrn je vidět na obrázku 5.

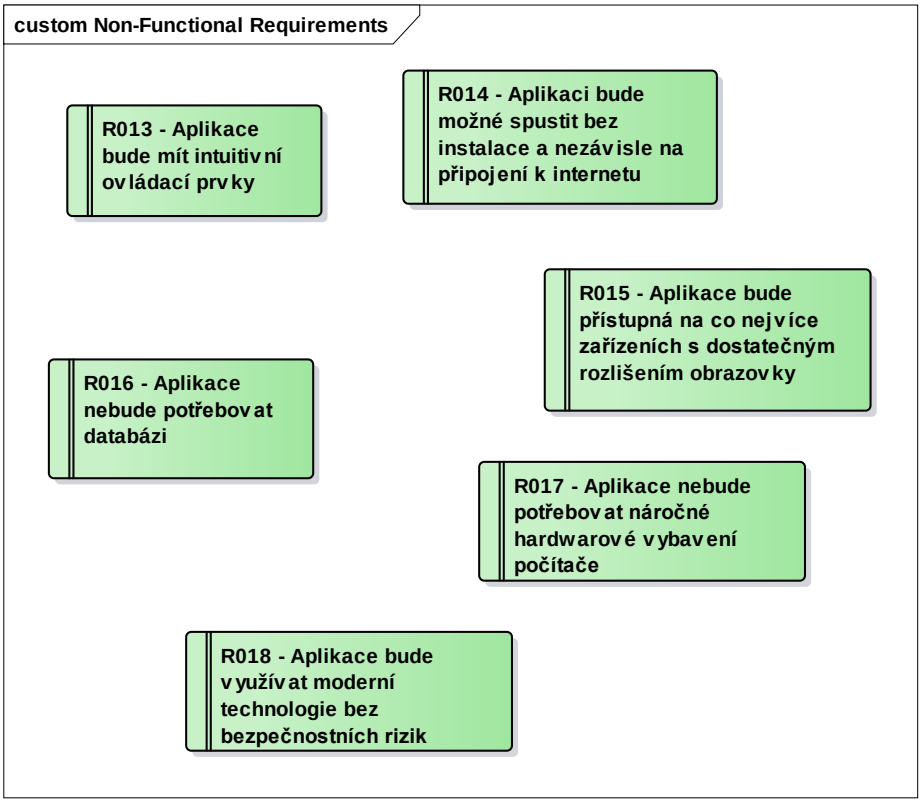



**Obrázek 5: Funkční požadavky**

### 6.1.2 Nefunkční požadavky

Nefunkční požadavky jsou, jednoduše řečeno, vlastnosti, které by systém měl mít. Je to důležitá část specifikace softwaru. Dají se vysvětlit jako soubor charakteristik, díky kterým je nový program lépe použitelný, rychlý, bezpečný, výkonný, rozšiřitelný nebo spolehlivý. Nefunkční požadavky se také dají chápat jako podmínky, které by měly být splněny při tvorbě systému. Tyto požadavky již pomáhají určit způsob, jakým bude program implementován. [27] [28]

Nefunkční požadavky byly také sepsány pomocí známého softwarového nástroje Enterprise Architect, který je nezanedbatelným pomocníkem v návrhové fázi vývoje softwaru.

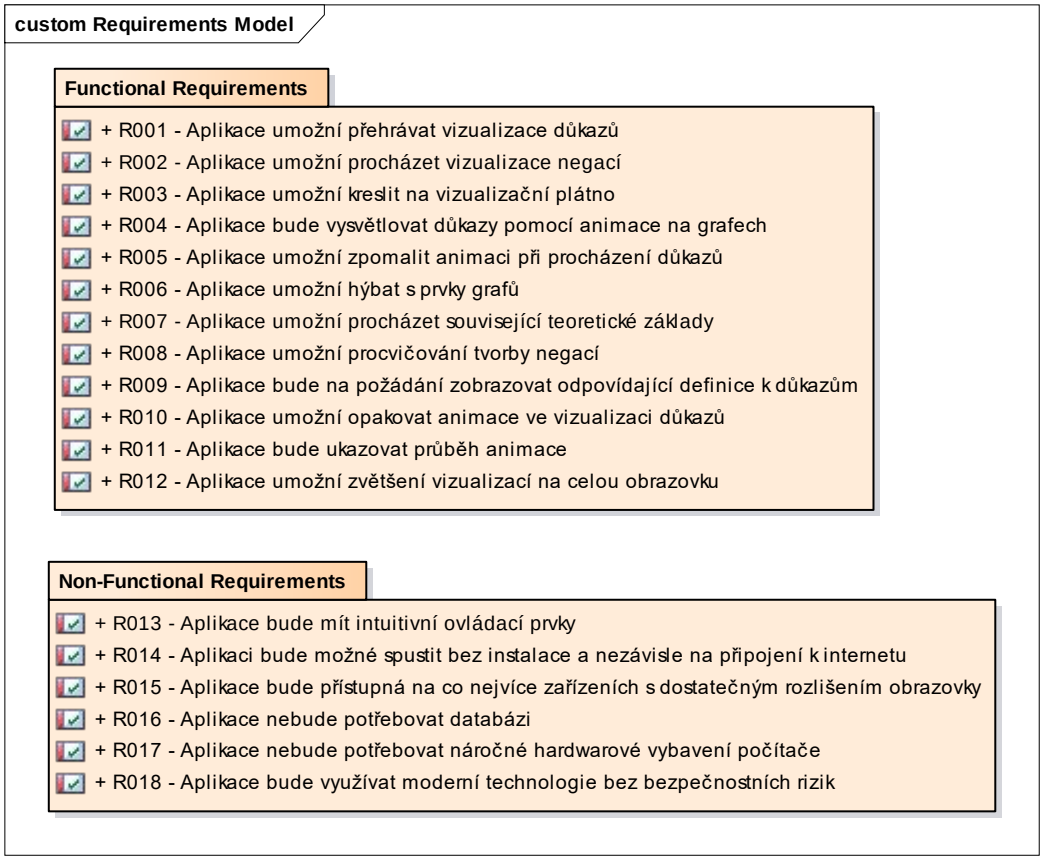



**Obrázek 6: Nefunkční požadavky**

### 6.1.3 Model požadavků

Funkční i nefunkční požadavky lze shrnout do jednoho modelu, který stručně a výstižně ukazuje všechny požadavky na nově vznikající aplikaci a zároveň umožňuje všimnout si rozdílu mezi těmito dvěma kategoriemi požadavků. Tento model byl opět vytvořen v programu Enterprise Architect.

Při návrhu požadavků je možné se setkat s uspořádáním prvků do hierarchie (tzv. taxonomie). Základní taxonomií je rozdělení požadavků na funkční a nefunkční, ale zejména nefunkční požadavky je možné ještě více kategorizovat. V případě této práce nejsou tyto požadavky rozebírány do větší hloubky, protože pro dané množství nefunkčních požadavků by další tříštění nebylo vhodné, a tedy stačí tato jednoduchá taxonomie.




**Obrázek 7: Model požadavků**

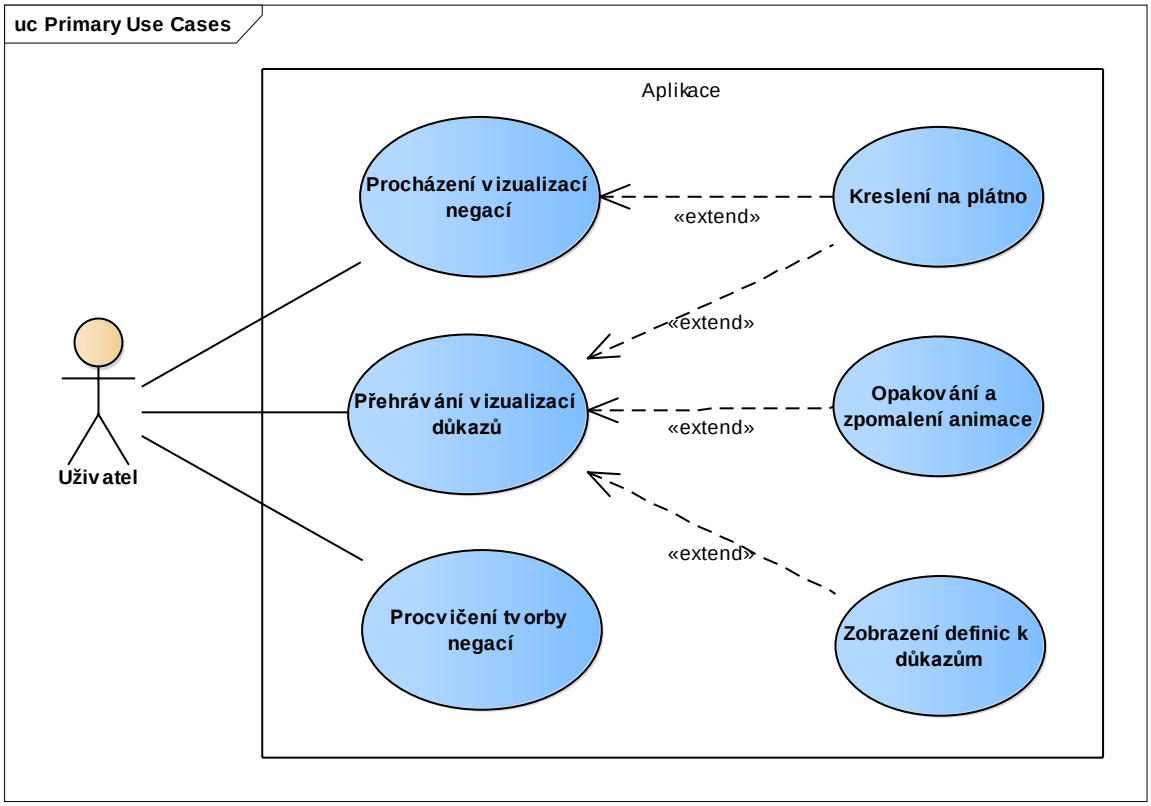
## 6.2 Případy užití

Při tvorbě softwaru se od specifikace požadavků postupuje k modelování případů užití, které se označují anglickým souslovím „use cases“. Případ užití je nějaká činnost, kterou aktér (osoba používající systém) vykonává s pomocí systému. *„Modelování případů užití je jiným, doplňkovým způsobem získávání a dokumentování požadavků. Modelování případů užití se skládá z následujících aktivit:*

- *Nalezení hranic systému,*
- *Vyhledání aktérů (actors),*
- *Nalezení případů užití:*
    - *Specifikace případu užití,*
    - *Určení alternativních scénářů.“* [27]

Vytvářená aplikace má jednoho aktéra – uživatele. Tímto uživatelem může být student, který se s aplikací připravuje doma pomocí e-learningu, nebo vyučující, který s aplikací

pracuje v rámci výukové hodiny. Není nutné uživatele dále rozdělovat, protože obě tyto persony budou s aplikací pracovat obdobným způsobem, a není tedy potřeba v aplikaci vytvářet omezení rolí pro učitele a pro studenta.




**Obrázek 8: Model případů užití**

V modelu případů užití výše je možné vidět šest základních případů užití a hranice aplikace. Uživatel může procházet vizualizace negací, v rámci této činnosti je také možné kreslit tužkou na plátno a nějakou informaci si tak ve vizualizaci zvýraznit. Kreslení je uskutečnitelné i při přehrávání vizualizací důkazů, kde je navíc možnost opakovat určitý krok důkazu a zpomalit animaci, která se při kroku spouští.

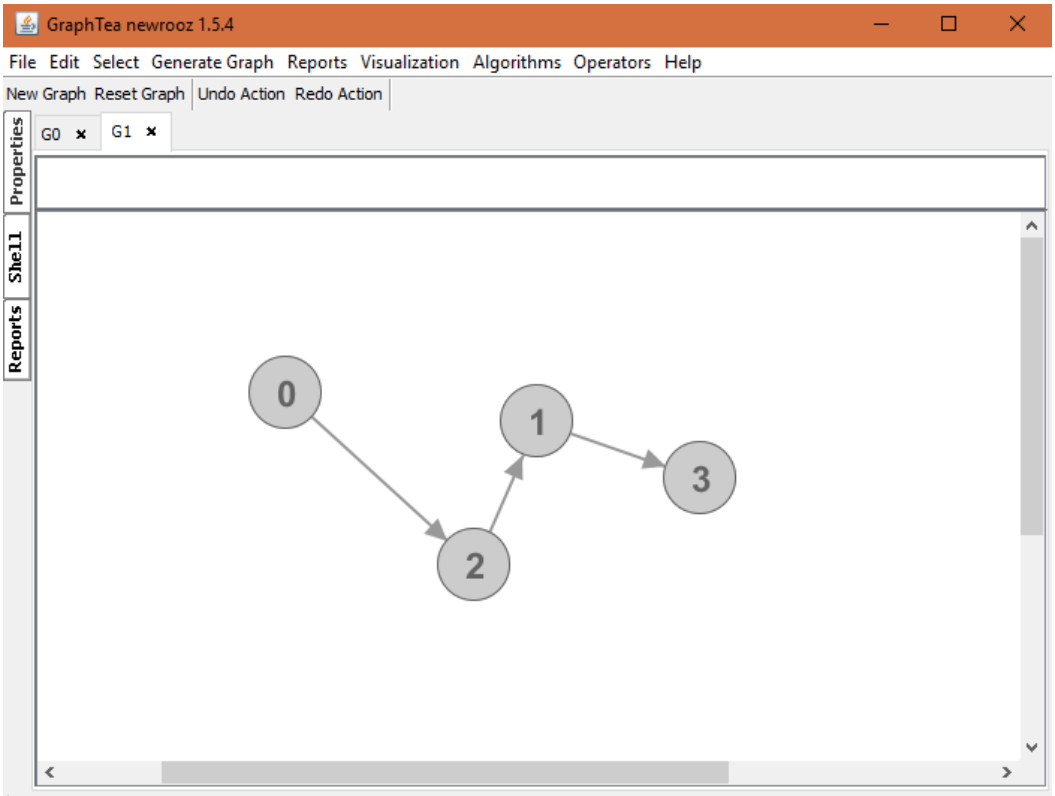
Při procházení jednotlivých kroků důkazu si uživatel může zobrazit definice, které souvisí s daným důkazem, a ihned si tak připomenout základní pojmy nejen z teorie grafů, ale i z výrokové logiky. Dále si uživatel může procvičovat tvorbu negací pomocí jednoduchého cvičení typu otázka (původní výrok) a správná odpověď (negace výroku).

## 6.3 Podobné aplikace

Existuje více aplikací, které se zabývají vizualizací grafů a grafových algoritmů, ale žádná z nich se neorientuje přímo na vizualizaci důkazů v tomto oboru. Taktéž nebyla nalezena aplikace přímo pro výuku negací výroků.

### 6.3.1 GraphTea

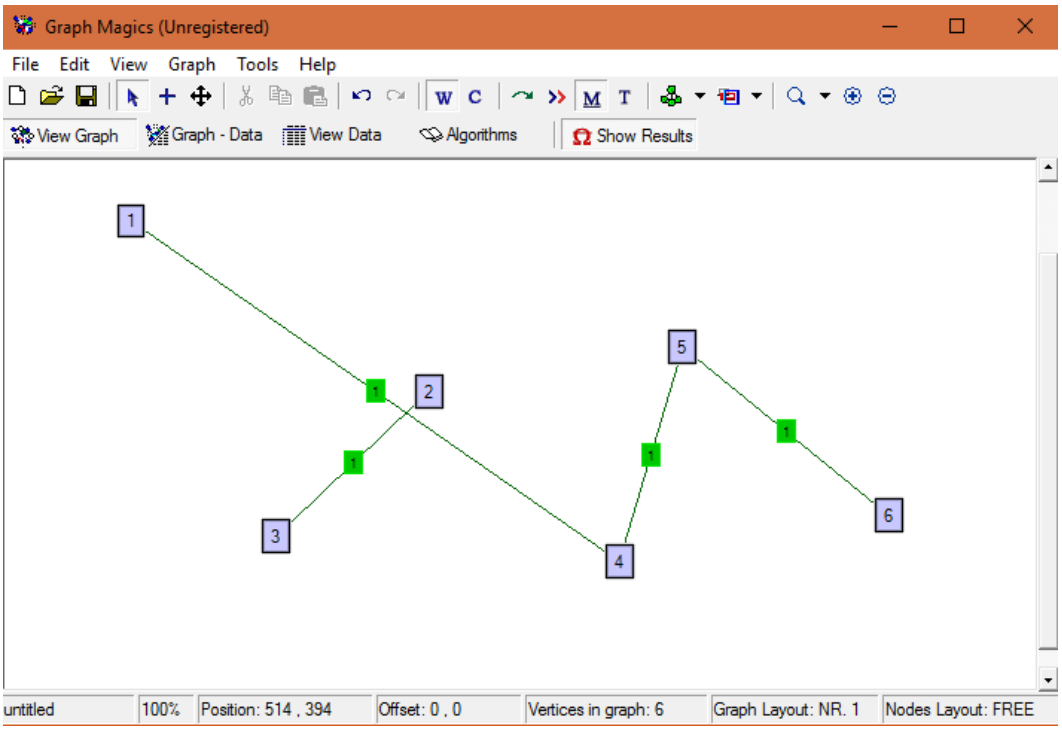
Program GraphTea [29] může sloužit jako podpora výuky teorie grafů, a to jak pro studenty, tak pro učitele. Tento software je open-source (s otevřenými zdrojovými kódy) a je potřeba ho nainstalovat na svůj počítač dle daného operačního systému. Program má široké možnosti užití, je možné zde kreslit orientované i neorientované grafy a zjišťovat o nich informace jako například jestli se jedná o bipartitní graf, eulerovský graf a tak dále. Na nakresleném grafu je možné spouštět algoritmy, mezi které patří například prohledávání do šířky nebo do hloubky. Program ale neobsahuje žádné důkazy z teorie grafů ani jejich vizualizace a definice pro výuku těchto důkazů. Stejně tak se nezaobírá jakoukoliv související výrokovou logikou.




**Obrázek 9: Software GraphTea**

### 6.3.2 Graph Magics

Dalším obdobným programem je Graph Magics [30]. Svou funkcionalitou se podobá již zmíněnému GraphTea, oproti němu je však ovládání tohoto programu méně intuitivní. Umožňuje tvořit grafy a spouštět na nich grafové algoritmy, kterých je celkem sedmnáct (nalezení nejkratší cesty, problém čínského listonoše, eulerovský tah a další). Tento program, stejně jako předchozí, neumožňuje zobrazovat důkazy, které souvisí s danými pojmy. Je placený, zdarma lze využít pouze trial verzi na třicet dní. Stejně jako u programu GraphTea se jedná o desktopový program, u kterého je potřeba instalace.

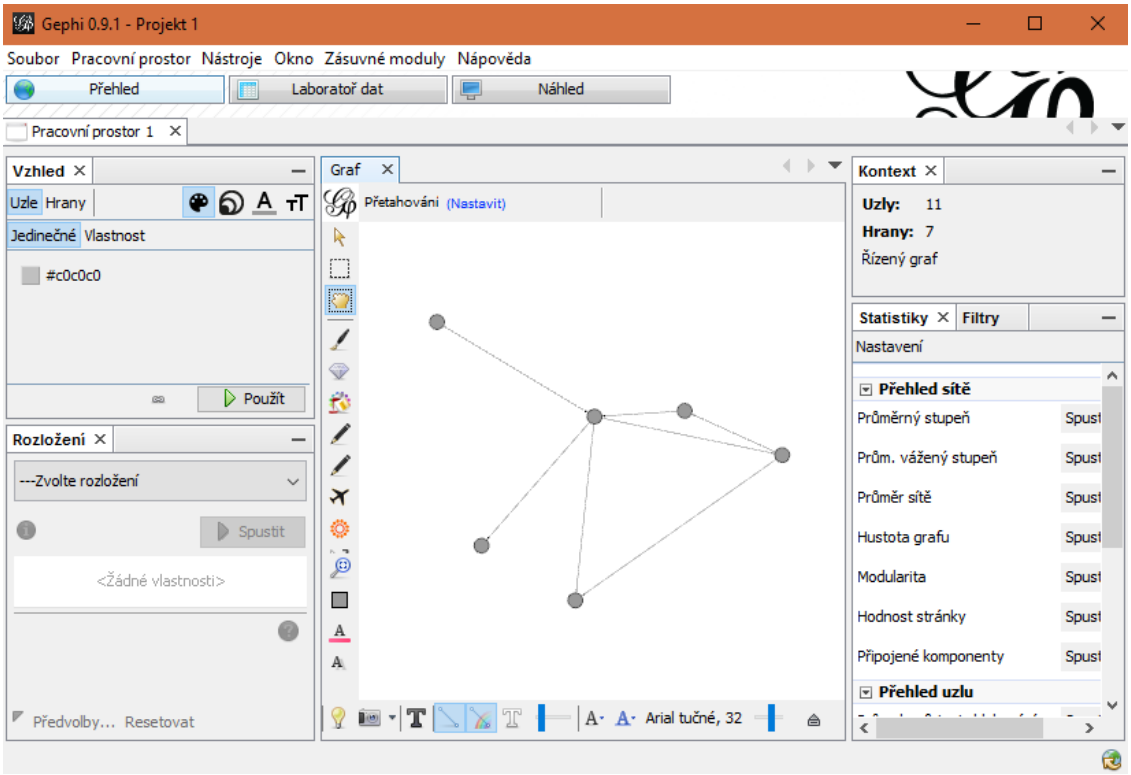



**Obrázek 10: Software Graph Magics**

### 6.3.3 Gephi

Gephi [31] je open-source software, který slouží k vizualizaci grafů a sítí. Z vybraných příkladů se jedná o nejvíce komplexní software, který se zaměřuje na vizualizaci dat pro vědce a datové analytiky. V programu je možné tvořit grafy a manipulovat s nimi, obarvovat a měnit vrcholy a hrany, pouštět vizualizace i velmi rozsáhlých grafů (100 000 vrcholů a 1 000 000 hran) v reálném čase. Z grafů je možné odvozovat statistiky a metriky. Software je zdarma. Ze zde zmíněných programů se ale nejméně hodí pro

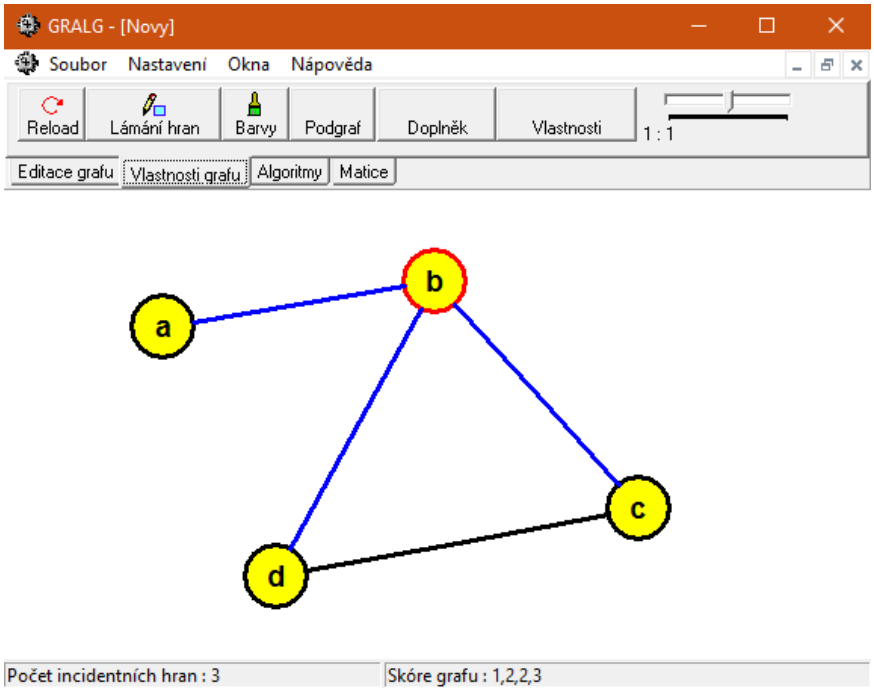
výuku teorie grafů a jeho součástí také není dokazování matematických vět a tvrzení z této oblasti.




**Obrázek 11: Software Gephi**

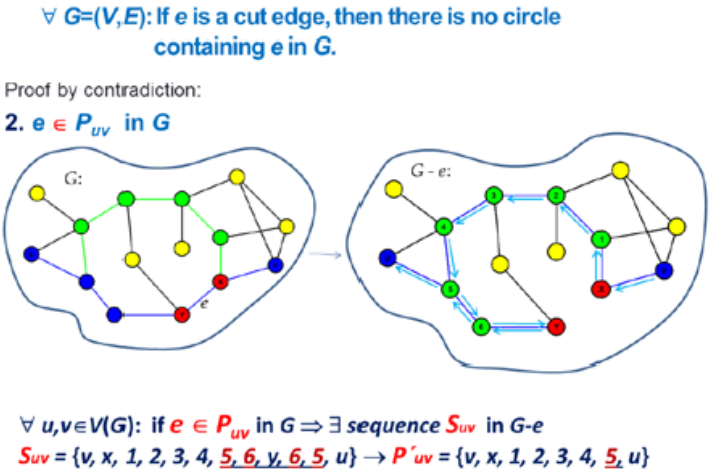
### 6.3.4 GrAlg

Program GrAlg byl vytvořen ve vývojovém prostředí Delphi a již několik let slouží na Univerzitě Hradec Králové k výuce předmětů zabývajících se teorií grafů [32]. V tomto programu je možné vytvářet a editovat orientované i neorientované grafy a výsledek práce uložit ve formátu XLS nebo BMP. Program také umožňuje spouštět na vytvořených grafech velké množství grafových algoritmů (algoritmy hledání minimální kostry, prohledávání do šířky i do hloubky a další), přičemž v průběhu algoritmu umožňuje sledovat i aktuální stav zásobníku. [33]




**Obrázek 12: Software GrAlg**

Tento software je možné použít i při přípravě výukových materiálů nápomocných při výuce důkazů z teorie grafů, kde se uplatní jeho schopnost obarvovat vrcholy nebo hrany. Příklad takového využití udávají Milková a Ševčíková [34].




**Obrázek 13: Použití programu GrAlg při výuce důkazů, zdroj: [34]**

## 6.4 Výběr technologií a přístupu

Výběr technologií a přístupu, jakým je k vývoji programu přistupováno, záleží zejména na nefunkčních požadavcích. Mezi tyto požadavky patřila dostupnost programu i bez instalace, použití moderních nástrojů a nenáročnost hardwarového vybavení strojů. Proto bylo rozhodnuto o implementaci projektu formou webové aplikace se zaměřením na klientskou část. To splňuje princip takzvané single page aplikace (single-page application, zkratkou SPA). Nejdříve se ale hodí představit vhodné programovací jazyky pro tvorbu tohoto typu aplikací.

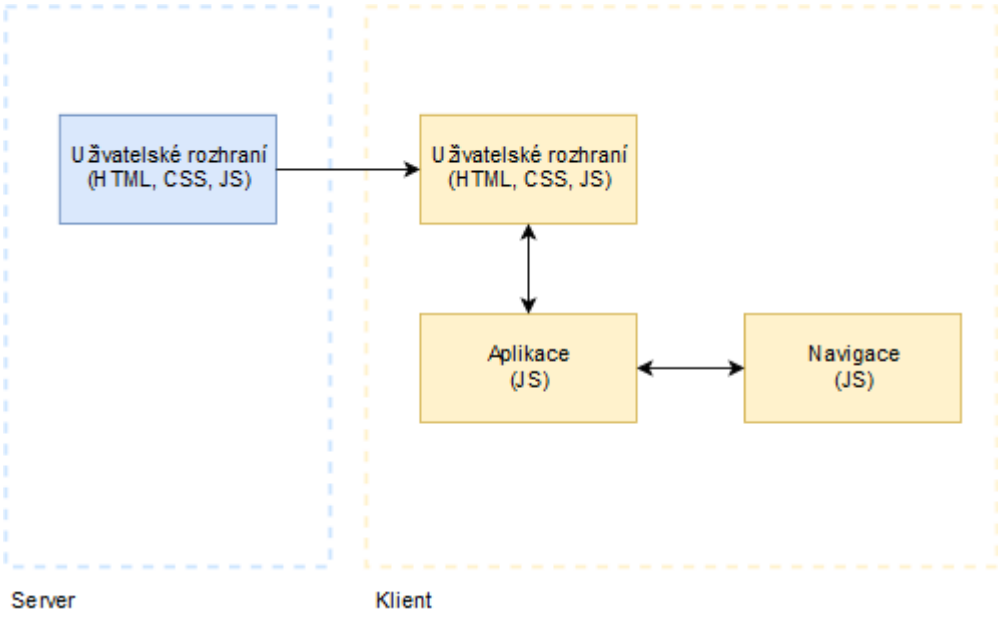
### 6.4.1 JavaScript a TypeScript

TypeScript je programovací jazyk, dá se říci nadstavba jazyka JavaScript. Vyznačuje se možností použít třídy, datové typy a další prvky objektově orientovaného programování. TypeScript je kompilován do jednoduchého, čistého JavaScriptu, aby mohl být jeho kód proveden v prohlížeči, a to v jakémkoliv. TypeScript také není závislý na použitém operačním systému a je open-source. [9]

JavaScript je skriptovací jazyk, který se často používá pro programování uživatelských prvků webových stránek, ale může být použit i na straně serveru. Byl vytvořen Brendanem Eichem v roce 1995 a spolu s HTML a CSS je základem tvorby webových stránek. [8]

### 6.4.2 Single page aplikace

Single page aplikace je označení pro speciální typ webové aplikace, která je tvořena jednou stránkou a píše se pomocí jazyka JavaScript. Při spuštění je třeba načíst pouze tuto jedinou stránku, popřípadě jsou ostatní prostředky zaváděny dle potřeby. Tím je zajištěna rychlejší odezva na uživatelovy podněty, protože prohlížeč nemusí překreslovat celou stránku, ale pouze její část. Uživatelská zkušenost je pak podobná jako při užívání aplikace desktopové. Na obrázku 14 je možné vidět zjednodušený náčrt architektury SPA (zde bez nutnosti ukládání dat). [35]




**Obrázek 14: Single page aplikace, zdroj: vlastní zpracování dle [36]**

Tento typ aplikace lze programovat pomocí několika frameworků, které vzaly principy single page aplikace za své. Mezi nejznámější patří například frameworky AngularJS, React, Meteor.js nebo Aurelia. Negativní stránkou použití této architektury může být problém se SEO (optimalizace pro vyhledávače neboli search engine optimization), protože díky svému principu načítání jsou tyto stránky hůře zpracovatelné pro roboty vyhledávačů. To ale není problém v případě zde vytvářené aplikace, která má být primárně spouštěna lokálně. Lokální umístění je u single page aplikace možné, protože veškerá její logika je prováděna v prohlížeči.

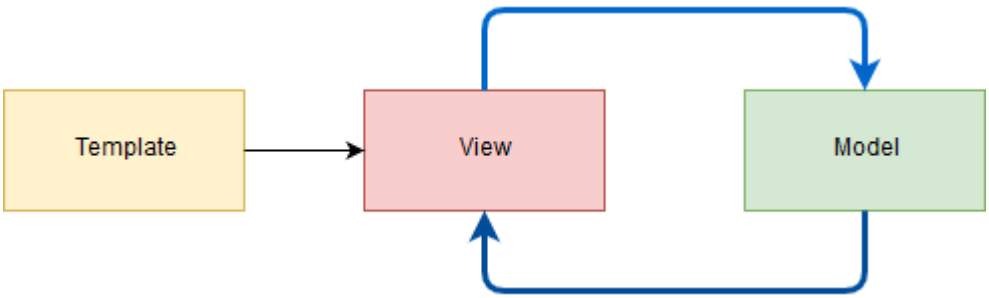
### 6.4.3 Frameworky

Jak již bylo řečeno, existuje několik frameworků, které se zaměřují na tvorbu single page aplikací. Pro vývoj aplikace byli vybráni následující kandidáti: AngularJS, React a Bobril. První dva patří mezi nejznámější a nejrozšířenější. Původní český framework Bobril byl vybrán jako vhodný protikandidát, protože autorka této práce s ním má největší zkušenost, i když celosvětově ještě tolik známý není. Tato kapitola se zabývá představením a porovnáním těchto frameworků.

#### 6.4.3.1 AngularJS

AngularJS[1] je javascriptový framework, který je open-source. Slouží k programování frontendu u webových aplikací. V době psaní této práce byla již vydána verze AngularJS 2, která není kompatibilní s předchozími verzemi. Tento framework je nezávislý na platformě, oproti svým předchozím verzím je rychlejší a jednodušší. Zaměřuje se hlavně na mobilní aplikace. [37]

Nově vznikající aplikaci je možné psát ve starší nebo v nové verzi. AngularJS 1 přinesl do webového vývoje myšlenku naprogramování vlastních javascriptových komponent, kterou přebraly i frameworky následující po něm, jako například React. Oproti Reactu je ale výkonnostně horší, a to hlavně díky tzv. obousměrnému provázání dat (two-way data binding). Jakékoliv změny, které se provedou v modelu, se ihned promítají do pohledu (neboli view) a stejně tak i obráceně. Model s pohledem je tedy propojen vazbou v obou směrech, což se promítá do rychlosti reakcí uživatelského rozhraní. [38] [39]




**Obrázek 15: Obousměrné provázání dat, zdroj: vlastní zpracování dle [40]**

Druhá verze Angularu se jeví mnohem příznivěji, a to hlavně právě ohledně rychlosti vyřízení uživatelských požadavků, protože tato dvojitá vazba v něm již není přednastavena. Místo toho je nahrazena jednou vazbou z modelu na pohled (view). Je ale důležité podotknout, že Angular spolu s nástrojem aplikační architektury RxJS je dle [38] až čtyřikrát větší knihovnou než React spolu s nástrojem Redux, takže načtení Angularu

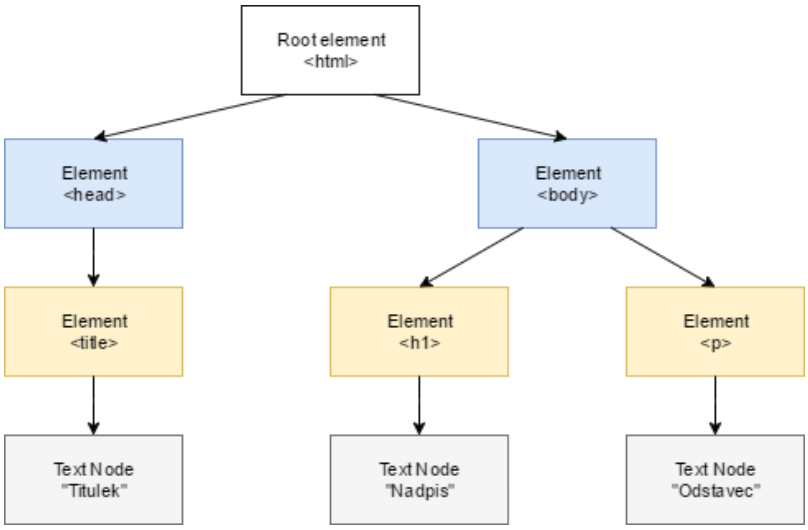
[1] https://angularjs.org/

trvá déle. Stejně tak článek [38] uvádí komplikovanost, s jakou se framework stará o vkládání závislostí (dependency injection, DI).

#### 6.4.3.2 React

React[2] (někdy také React.js) je javascriptová knihovna pro tvorbu uživatelských rozhraní vyvinutá ve společnosti Facebook. Zabývá se vrstvou pohledu (view) v návrhovém vzoru MVC (Model-View-Controller), takže může být použit i v kombinaci s frameworkem AngularJS, který není zaměřen pouze na komponenty uživatelského rozhraní, ale i na ostatní části tohoto návrhového vzoru.

React využívá technologii takzvaného virtuálního objektového modelu dokumentu neboli Virtual DOM (Document Object Model). Klasický HTML Document Object Model má podobu stromu (viz obrázek 16), kde jsou HTML elementy reprezentovány uzly. Virtual DOM je abstrakcí HTML objektového modelu dokumentu. Do tohoto zjednodušeného modelu se promítají změny v elementech stránky a na požádání vývojáře se do výsledného skutečného Document Object Modelu vykreslí jen nově provedené změny, takže se překreslí jen změněné elementy, a ne celá webová stránka. Komponenty mohou být psány pomocí JSX, což je rozšíření JavaScriptu přidávající syntaxi XML do JavaScriptu. [38]




**Obrázek 16: Document Object Model, zdroj: vlastní zpracování dle [8]**

[2] https://facebook.github.io/react/

#### 6.4.3.3 Bobril

Bobril[3] je komponentově orientovaný javascriptový framework původem z České republiky (autorem je Boris Letocha). Je inspirovaný knihovnami React a Mithril, využívá již zmíněný virtuální DOM a zaměřuje se na co nejmenší velikost a co největší rychlost. [10]

Aplikaci využívající Bobril je vhodné psát s pomocí TypeScriptu (i samotný Bobril je napsán v tomto jazyce) a s použitím nástroje Bobril-build. Bobril-build je nástroj pro sestavování aplikace, který byl speciálně vyvinutý pro kombinaci s frameworkem Bobril, ale, jak uvádí článek [41], může být použitý i s jakoukoliv aplikací napsanou v TypeScriptu.

### 6.4.4 Benchmark

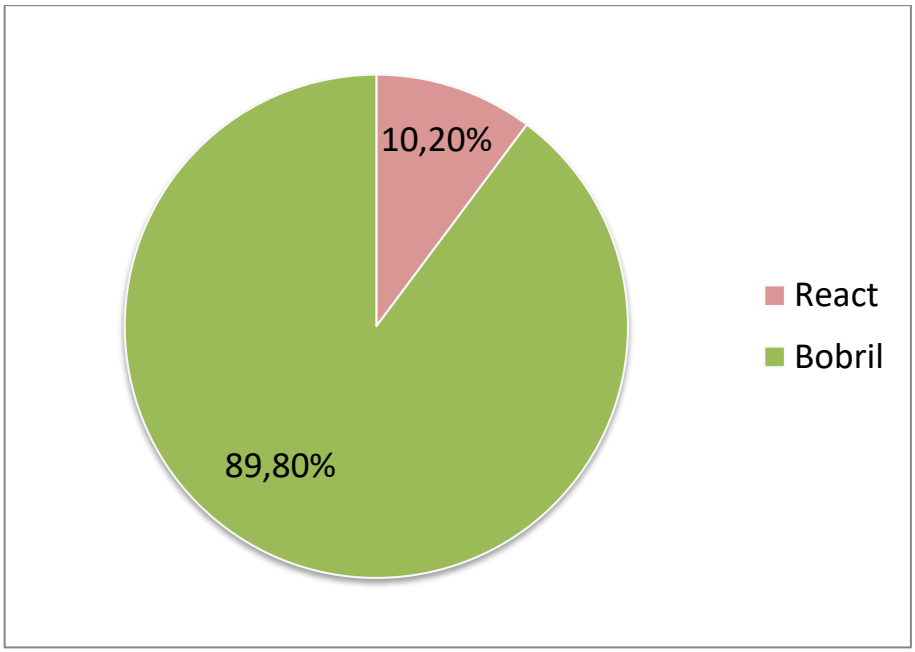
Benchmarkem se v oblasti výpočetní techniky označuje porovnání výkonu počítačových programů nebo operací, často se užívá při porovnání hardwarových prostředků. Díky projektu UI Benchmark [42] je možné porovnat i výkon zde zmíněných knihoven pro vykreslování prvků uživatelského rozhraní. Součástí tohoto testu je několik knihoven, mezi kterými jsou i frameworky React a Bobril, které jsou svým obdobným zaměřením vhodné pro srovnání a jsou tedy v této práci výkonnostně porovnány.

Je důležité zmínit konfiguraci počítače, na kterém byly testy prováděny a také prohlížeč, ve kterém byly spouštěny. Benchmark byl proveden v prohlížeči Mozilla Firefox verze 52.0.2 (32bit) na následujícím stroji:

- Procesor: Intel Core i5-2520M CPU 2.50GHz 2.50GHz
- RAM: 8 GB
- OS: Windows 10 Pro
- 64bitový operační systém
- Grafické čipy: Intel HD Graphics 3000, NVIDIA NVS 4200M

[3] https://github.com/Bobris/Bobril

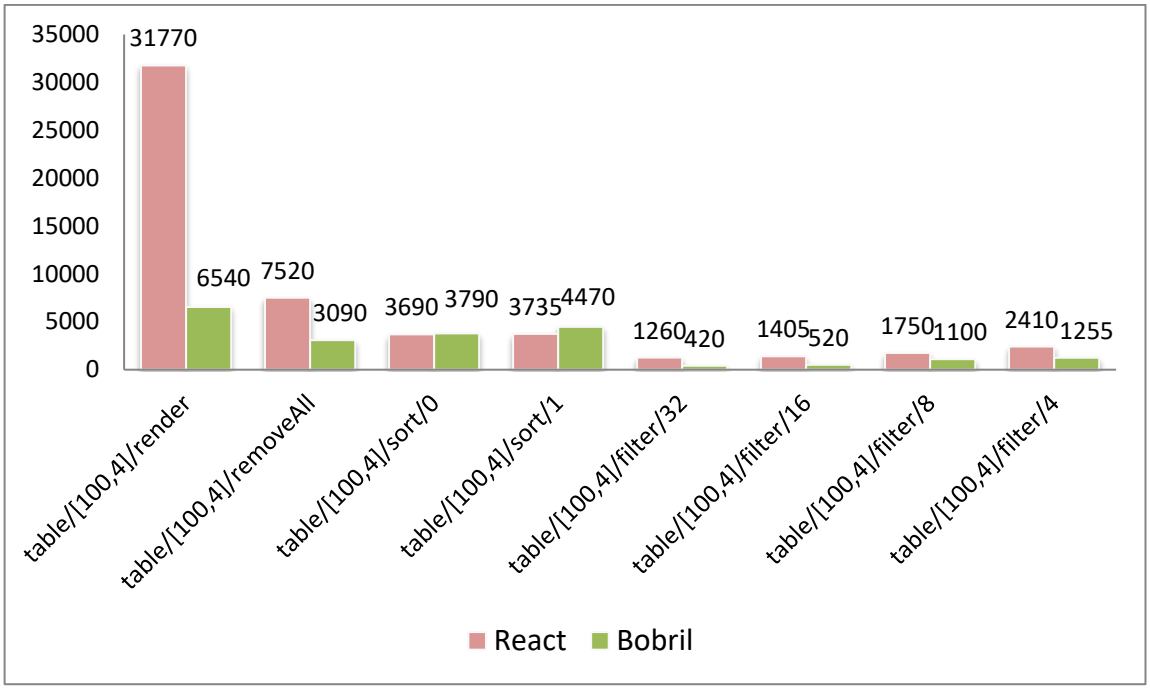
Podrobné výsledky srovnání lze nalézt v příloze A této práce. Výsledky lze shrnout následujícím grafem, který ukazuje, v kolika procentech případů benchmarku byl rychlejší framework Bobril a v kolika framework React, a to v prohlížeči Mozilla Firefox.




**Graf 1: React a Bobril - benchmark, shrnutí výsledků**

Jak je vidět, v necelých devadesáti procentech byl při vykreslování rychlejší framework Bobril, zatímco přibližně v deseti procentech to byl React. To je dáno zaměřením Bobrilu na rychlost a simplicitu.

Pro představu, které prvky byly vykreslovány a jak rychle, lze uvést sloupcový graf (graf 2). Ten ukazuje rychlost vykreslení daných objektů v milisekundách (jedná se pouze o část objektů z podstoupeného benchmarkového testu). Prvním případem značeným `table/[100,4]/render` je například aktualizace z prázdné tabulky na tabulku se sto řádky a čtyřmi sloupci. Dalším testovacím scénářem je odebrání takové tabulky. Následuje řazení a filtrování.




**Graf 2: React a Bobril - benchmark, testování vykreslování tabulek**

### 6.4.5 Výběr frameworku

Na základě porovnání výhod a nevýhod zmíněných frameworků byl pro tvorbu aplikace zvolen framework Bobril. Tomu nahrává i provedené výkonnostní srovnání a také zkušenosti autorky s daným frameworkem a jeho komunitou.

# 7 Použité technologie

Tato kapitola se věnuje podrobnějším charakteristikám vybraných technologií, které byly při tvorbě aplikace použity.

## *7.1 Bobril*

Stručně byl framework Bobril představen již v předchozí kapitole. Tato část je zaměřena na jeho implementaci virtuálního Document Object Modelu a podrobnější vysvětlení s příkladem. Dále uvede komponenty a jejich životní cyklus.

### 7.1.1 Virtuální DOM

Uzel virtuálního Document Object Modelu je v Bobrilu reprezentován rozhraním `IBobrilNode`, které obsahuje nepovinné vlastnosti (jinak také property). Mezi ně patří tyto vlastnosti:

- `tag` – udává název HTML tagu (například `'div'`),
- `className` – udává název CSS třídy,
- `style` – udává styly přidané k HTML tagu,
- `attrs` – udává atributy daného HTML tagu (například `id`), které jsou dány rozhraním `IBobrilAttributes,`
- `children` – zde jsou uvedeny child uzly elementu, může zde být jak pouze text, tak i pole uzlů.

Na ukázce zdrojového kódu 1 lze vidět příklad části objektového modelu dokumentu, který je vytvořen pomocí několika objektů TypeScriptu implementujících právě výše zmíněné rozhraní `IBobrilNode`. Objekt uvedený v příkladu reprezentuje element s tagem `div`, který má v sobě vnořeny další dva elementy (takzvané child uzly) pomocí vlastnosti `children`. V poli vnořených uzlů je nadpis s tagem `h1` a odstavec s tagem `p`. Je možné vidět atributy přidané k daným elementům (zde `id`) a také textové uzly. Ty mohou patřit mezi children uzly jiných elementů. V příkladu je možné vidět textové uzly vnořené do odstavce stejně tak jako další element `em`, který značí druh písma zvaný kurzíva.

```
{
    tag: 'div',
    children: [
        {
            tag: 'h1', attrs: { id: 'heading-1' }, children: 'Nadpis'
        },
        {
            tag: 'p', attrs: { id: 'paragraph-1' }, children: [
                'Text odstavce',
                {
                    tag: 'em', attrs: { id: 'em-1' }, children: 'Kurzíva'
                }
            ]
        }
    ]
},
```

**Zdrojový kód 1: Virtuální DOM**

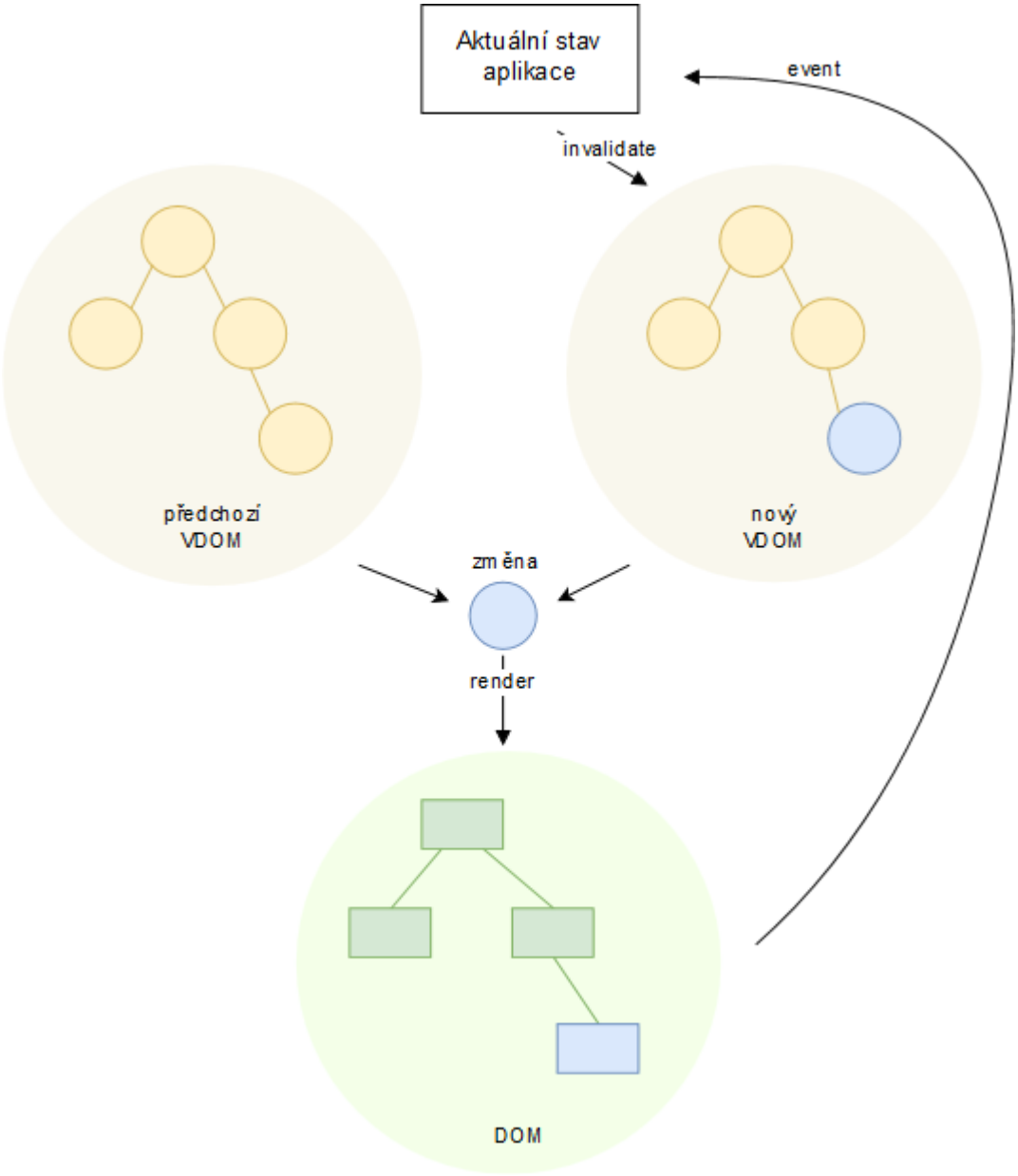
Při vykreslení takového virtuálního DOMu v prohlížeči by výsledný reálný DOM vypadal v HTML následovně.

```
<div>
    <h1 id="heading-1">Nadpis</h1>
    <p id="paragraph-1">
        Text odstavce
        <em id="em-1">Kurzíva</em>
    </p>
</div>
```

**Zdrojový kód 2: Reálný DOM**

Princip, jakým funguje v Bobrilu vykreslování změn pomocí virtuálního DOMu, je obdobný jako ve frameworku React. Aplikace se nachází v aktuálním stavu, vývojář provede nějakou změnu a vyvolá překreslení stránky pomocí metody `invalidate()`. V tu chvíli je připraven nový virtuální Document Object Model. Bobril se zaměří na změnu, která byla provedena oproti minulému virtuálnímu modelu, což může být například přidání nového elementu do stránky (na obrázku 17 je změna znázorněna modrým kolečkem). Framework nyní překreslí reálný HTML DOM pouze v tom, co se změnilo, takže není potřeba překreslovat celou stránku. Další výhodou virtuálního DOMu je, že v případě implementace v Bobrilu mají uzly virtuálního modelu mnohem méně vlastností než u reálného Document Object Modelu, takže změna ve virtuálním modelu není tak

náročná, jako by byla v případě reálného DOMu. Výpočetně náročnější vlastnosti elementů jsou vypočteny až při vykreslení reálného modelu. Celý cyklus je vidět na obrázku 17. V případě interakce uživatele se stránkou se spustí událost (event), která změní aktuální stav aplikace.




**Obrázek 17: VDOM a vykreslení změny, zdroj: vlastní zpracování dle [43]**

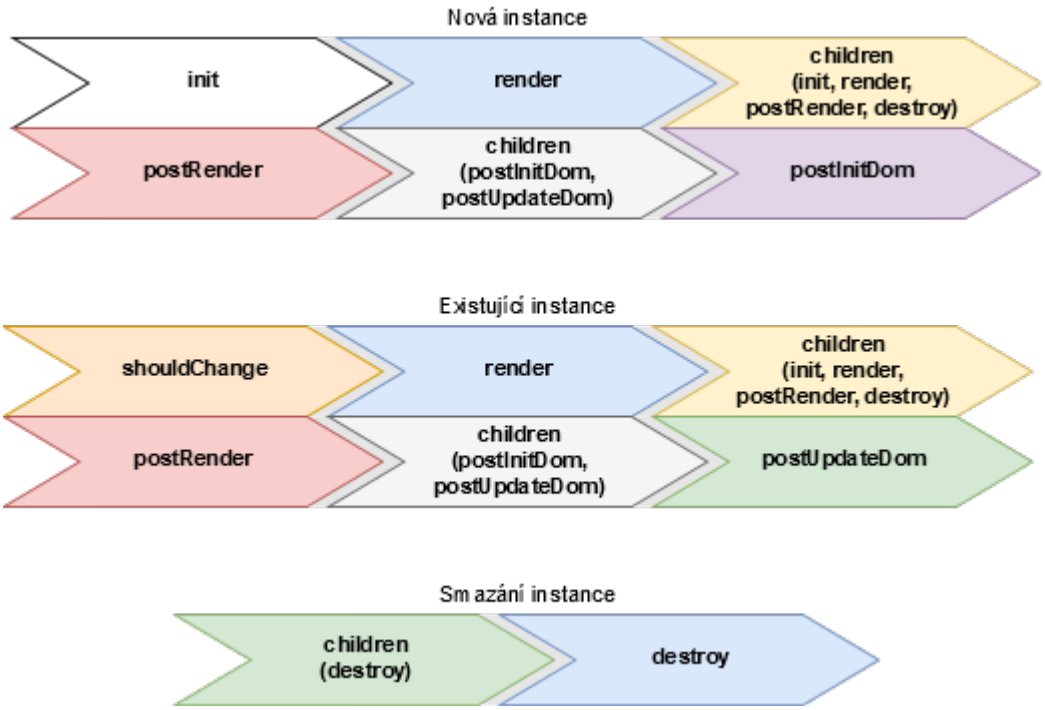
### 7.1.2 Komponenty

Bobril je komponentově orientovaný framework. Komponenta je základním stavebním prvkem stránky napsané v Bobrilu a pomocí komponent je možné poskládat celé

uživatelské rozhraní aplikace. Komponenty implementují interface `IBobrilComponent` a jsou charakterizovány svým životním cyklem, jehož částem odpovídají nepovinné metody právě v `IBobrilComponent`. Mezi tyto metody patří:

- `init` – tato metoda je volána před vytvořením uzlu ve virtuálním DOMu,
- `render` – stará se o přípravu virtuálního uzlu,
- `postRender` – volá se po vykreslení všech child uzlů
- `postInitDom` – volána po vytvoření uzlu ve skutečném DOMu,
- `shouldChange` – volá se před každým render (mimo prvotní vytvoření),
- `postUpdateDom` – volá se po změně uzlu ve skutečném DOMu,
- `destroy` – volána před odstraněním uzlu z DOMu.

Životní cyklus komponenty je znázorněn na obrázku 18.




**Obrázek 18: Životní cyklus komponenty, zdroj: vlastní zpracování dle [44]**

První průběh metod odpovídá vytvoření nové instance komponenty. Metody `init` a `postInitDom` jsou pro každou instanci volány pouze jednou. Pokud je provedena změna, již se provádí druhý řádek metod a na základě výsledku metody `shouldChange` se volá metoda `render` pro překreslení změny v komponentě. Metody `postInitDom`

a `postUpdateDom` již mohou pracovat s HTML elementem v reálném Document Object Modelu, takže je vhodné je využívat při použití externích knihoven, které neumí fungovat s komponentou, se kterou pracuje Bobril. Při smazání instance (třetí řádek v obrázku 18) se nejdříve smažou její child uzly a pak se teprve volá `destroy` metoda komponenty pro danou instanci. Jednou z komponent ve vytvořené aplikaci je například tlačítko pro mazání nakreslených čar na vykreslovací plátno. Komponenta vypadá následovně.

**Obrázek 19: Komponenta tlačítka**

Zdrojový kód k této komponentě je umístěn v samostatném souboru s koncovkou `.ts`. Exportovaná továrna (také factory) na uzly virtuálního DOMu má název buttonErase a je vytvořena pomocí metody `createVirtualComponent`, kterou poskytuje Bobril. Tato metoda je generická a je jí předáno rozhraní `IButtonEraseData`, které je definováno v souboru komponenty. Parametrem této metody je typescriptový objekt, který implementuje rozhraní `IBobrilComponent`. Ve zdrojovém kódu je tedy také vidět již zmíněná metoda `render` a mohly by se tam vyskytovat i ostatní metody volané v životním cyklu komponenty. Parametry metody `render` jsou `ctx` typu `IButtonEraseCtx` a `me` typu `IBobrilNode`. Díky nim je možné se v tělu metody dostat k vlastnostem komponenty, jako jsou například child uzly `me.children`, a tak definovat, jak se má komponenta vykreslit.

```
export const buttonErase = b.createVirtualComponent<IButtonEraseData>({
    render(ctx: IButtonEraseCtx, me: b.IBobrilNode) {
        me.children = b.style(
            bs.button(
                {
                    option: bs.ButtonOption.Primary,
                    onClick: ctx.data.onClick
                },
                bs.glyphicon({ icon: bs.GlyphIcon.Erase })
            ),
            ctx.data.fullScreen ? css.buttonFullStyle : css.buttonStyle);
}});
```

**Zdrojový kód 3: Komponenta tlačítka**

Komponenty lze definovat i pomocí metody `createComponent`, která se od metody `createVirtualComponent` liší tím, že je tvořená („obalená“) elementem s tagem `div`.

Data a kontext komponenty z příkladu jsou definovány jako rozhraní. Rozhraní `IButtonEraseData` reprezentuje data, která přijdou do komponenty z vnějšího světa, tedy z nadřazené komponenty. Kontext typu `IButtonEraseCtx` implementující rozhraní `IBobrilCtx` udržuje informace o stavu komponenty, které by neměly být přístupné z vnějšího světa, což neplatí pouze pro vlastnost `data` typu `IButtonEraseData`, díky které je možné dostat se k datům komponenty.

```
export interface IButtonEraseData {
    onClick?: (event: b.IBobrilMouseEvent) => boolean | void;
    fullScreen: boolean;
}

interface IButtonEraseCtx extends b.IBobrilCtx {
    data: IButtonEraseData;
}
```

**Zdrojový kód 4: Kontext a data komponenty**

## 7.2 Bobril-build

Bobril-build je sestavovací nástroj, který je určený pro build aplikace napsané s pomocí Bobrilu. Je možné ho nainstalovat pomocí nástroje Npm. Po instalaci stačí na začátku spustit příkaz `bb` a Bobril-build se už postará o spuštění serveru a sledování změn v souborech se zdrojovými kódy, přičemž při změnách informuje o úspěchu nebo neúspěchu buildu. Bobril-build dále umí:

- spouštět testy a generovat XML soubor s výsledky,
- instalovat externí závislosti,
- transpilovat,
- kompilovat TypeScript,
- spravovat překlady,
- spouštět externí Bobril pluginy. [41]

## 7.3 NPM

Npm[4] je manažer pro správu balíčků v JavaScriptu. Díky němu je možné instalovat a využívat části kódu, které již byly někým napsány a jsou znovupoužitelné. Všechny závislosti, které se vztahují k vyvíjenému projektu, lze definovat v souboru `package.json`. Následuje příklad závislostí ve vytvářené aplikaci, kde je také možné vidět závislost na frameworku Bobril (verze 6.0.2).

```
  "dependencies": {
    "@types/bootstrap-slider": "^4.8.32",
    "bobril": "^6.0.2",
    "bobrilstrap": "^1.2.3",
    "bootstrap-slider": "^9.7.2",
    "jquery.scrollintoview": "^1.9.4"
  },
```

**Zdrojový kód 5: Závislosti v package.json**

## 7.4 Bootstrap a Bobrilstrap

Bootstrap[5] je známý framework obsahující prvky tvořené pomocí HTML, CSS a JavaScriptu, jako jsou například tlačítka, stránkování, modální okna, menu, panely, formuláře a další. Slouží pro jednoduchou tvorbu uživatelských rozhraní webových stránek. Například vložit piktogram reprezentující zvětšení na celou obrazovku do webové stránky je při použití tohoto frameworku snadné.

```
<span class="glyphicon glyphicon-fullscreen" aria-hidden="true"></span>
```

**Zdrojový kód 6: Vložení prvku uživatelského rozhraní s použitím Bootstrapu**

Bobrilstrap je framework, který obaluje všechny tyto prvky tak, aby je bylo možné použít ve webové aplikaci psané v Bobrilu, a to jako již zmíněné komponenty. Příklady použití jsou uvedeny na jeho stránkách [45] a přesně odpovídají prvkům Bootstrapu, s tím

[4] https://www.npmjs.com/

[5] http://getbootstrap.com/

rozdílem, že zápis není v HTML, ale v TypeScriptu. Piktogram odpovídající příkladu výše se s pomocí Bobrilstrapu naprogramuje následovně.

```
bs.glyphicon({ icon: bs.GlyphIcon.Fullscreen })
```

**Zdrojový kód 7: Vložení prvku uživatelského rozhraní s použitím Bobrilstrapu**

## 7.5 SVG

SVG je zkratka pro Scalable Vector Graphics (česky škálovatelná vektorová grafika). SVG obrázky jsou definovány pomocí souborů typu XML, díky čemuž mohou být vytvořeny i v libovolném textovém editoru. Jedná se o grafiku vektorového typu, takže grafické prvky definované pomocí tohoto formátu neztrácí na kvalitě, pokud jsou zvětšovány či přibližovány.

SVG je podporováno ve všech nejnovějších verzích známých prohlížečů k datu vydání této práce. Jeho podpora není zajištěna v prohlížeči Internet Explorer 8 a nižších a je pouze částečná v prohlížeči pro zařízení Android verze 3 až 4.3. Tyto prohlížeče ale již téměř nejsou používány. Internet Explorer 8 užívá celosvětově pouze 0,36 procent uživatelů (resp. zařízení). Prohlížeč Android 4 se nachází na 0,03 procentech zařízení, verzi 4.1 používá 0,18 procent uživatelů a Android 4.3 je aktuálně na 0,43 procentech zařízení. Celkově je SVG plně podporováno na 97,31 procentech zařízení, což je plně dostačující pro požadavky nově vzniklé aplikace. [46]

SVG může být součástí HTML stránek díky tagu `svg` a je možné ho dále upravovat pomocí CSS nebo Javascriptu. Vnořenými elementy pak mohou být například následující prvky:

- `circle` – kruh,
- `rect` – obdélník,
- `line` – čára,
- `path` – cesta skládající se z různých typů čar.

SVG prvky je možné měnit pomocí filtrů, které jim dodávají různé typy efektů. Tak se může vytvořit efekt rozmazání, stínu, světla a podobně. Všechny filtry se udávají do tagu `defs` a dále se vnořují do tagu `filter`. V příkladu zdrojového kódu 8 je možné vidět element Gaussovského rozostření `feGaussianBlur` nadefinovaný pomocí Bobrilstrapu, který je součástí filtrů v nově vytvořené aplikaci. Atribut `stdDeviation` uvádí velikost rozostření.

```
bs.elem(
    {
        tag: 'feGaussianBlur', attrs: { result: 'blurOut', stdDeviation: 10 }
    },
    []),
```

**Zdrojový kód 8: Gaussovské rozostření**

## 7.6 Verzovací systém a vývojové prostředí

Verzování je důležitou součástí vývoje softwaru, a to zejména, pokud je vyvíjen týmově. I v případě jednoho vývojáře je ale verzování vhodné, protože umožňuje vrátit se před nově provedené změny nebo změny zahodit. K verzování se používají verzovací systémy. Mezi známé systémy patří například Apache Subversion, Git nebo Mercurial.

Při vývoji aplikace byl používán systém Mercurial[6], který patří mezi takzvané distribuované (decentralizované) verzovací systémy. Ty jsou opakem systémů centralizovaných, které používají jedno centrální úložiště dat. Verzovací systém je možné použít ve spojitosti se vzdáleným serverem, který umožňuje přistoupit ke zdrojovým kódům odkudkoliv. Pro zálohování a verzování aplikace bylo vybráno řešení Atlassian Bitbucket[7], které je možné použít jak se systémem Git, tak se systémem Mercurial.

Vývojové prostředí umožňuje vyvíjet software a nabízí k tomuto účelu nástroje, které programování činí pro programátora jednodušším a efektivnějším. Vývojových prostředí je velké množství, mezi známé patří například Eclipse, NetBeans (hlavně pro vývoj v programovacím jazyku Java) a Microsoft Visual Studio (zejména pro jazyky C, C++, C#). Pojem vývojové prostředí se prolíná s pojmem editoru zdrojového kódu. Ten slouží k psaní a editaci zdrojových kódů, což ulehčuje zvýrazněním syntaxe kódu nebo automatickým doplňováním. Editor bývá součástí vývojových prostředí. Stejně tak existují i samostatné editory, které jsou oproti vývojovým prostředím odlehčené o jistou funkcionalitu a jsou vhodné například pro tvorbu klientských aplikací. Mezi známé editory patří Atom, Visual Studio Code nebo také Notepad++.

[6] https://www.mercurial-scm.org/

[7] https://bitbucket.org/

Aplikace byla vytvořena v editoru zdrojového kódu jménem Visual Studio Code[8] od společnosti Microsoft. V něm je možné nainstalovat i rozšíření, které pomáhají programovat v daném jazyce nebo frameworku. Takové rozšíření existuje i pro framework Bobril.

[8] https://code.visualstudio.com/

# 8 Implementace aplikace

Tato kapitola se zabývá samotnou implementací aplikace s použitím nástrojů zmíněných v předchozích kapitolách.

## 8.1 Složení aplikace

Aplikace je rozdělená do několika složek. Zde jsou stručně představeny části aplikace, které obsahují typescriptový kód (jsou vynechány složky s knihovnami, obrázky a podobně):

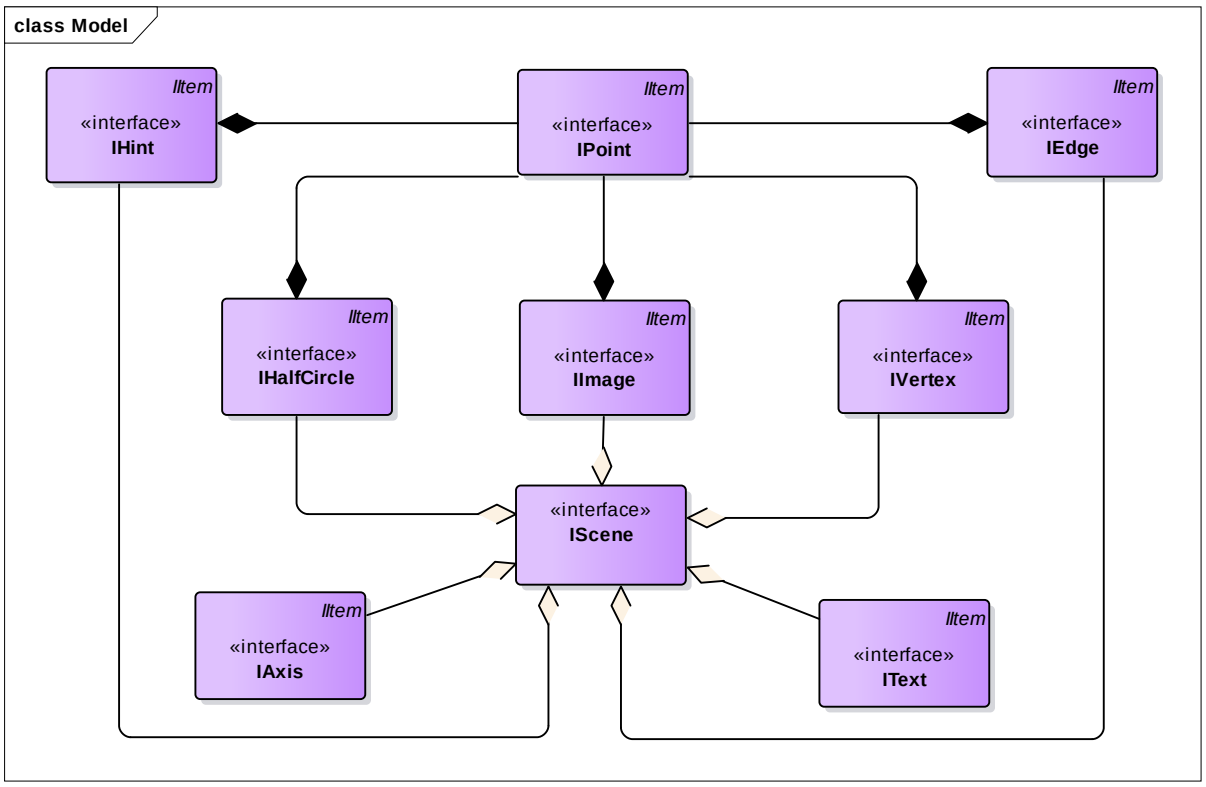
- `main` – obsahuje základní stránky aplikace (například úvodní stranu),
- `statements` – obsahuje stránky zabývající se výrokovou logikou, negacemi, jejich vizualizací a procvičením,
- `proofTypes` – obsahuje teoretickou část aplikace zahrnující typy důkazů,
- `proofs` – obsahuje vizualizace důkazů v teorii grafů,
- `viewer` – obsahuje nejen komponentu „přehrávače“ důkazů a negací, ale také akce, pomocí kterých je možné přidávat nebo odebírat komponenty vizualizací,
- `model` – obsahuje rozhraní modelu,
- `components` – obsahuje komponenty, ze kterých je aplikace složena,
- `common` – obsahuje pomocné algoritmy a konstanty.

## 8.2 Návrh rozhraní

Na obrázku 20 je vidět model návrhu rozhraní, která jsou základem vizualizační části aplikace. Rozhraní neboli interface má v TypeScriptu obdobný význam jako v ostatních objektově orientovaných jazycích, ale přeci jen je jeho pojetí trochu odlišné. V TypeScriptu rozhraní definuje typ objektu a jeho očekávané vlastnosti, takže TypeScript může ověřit, že daný objekt odpovídá určitému typu.

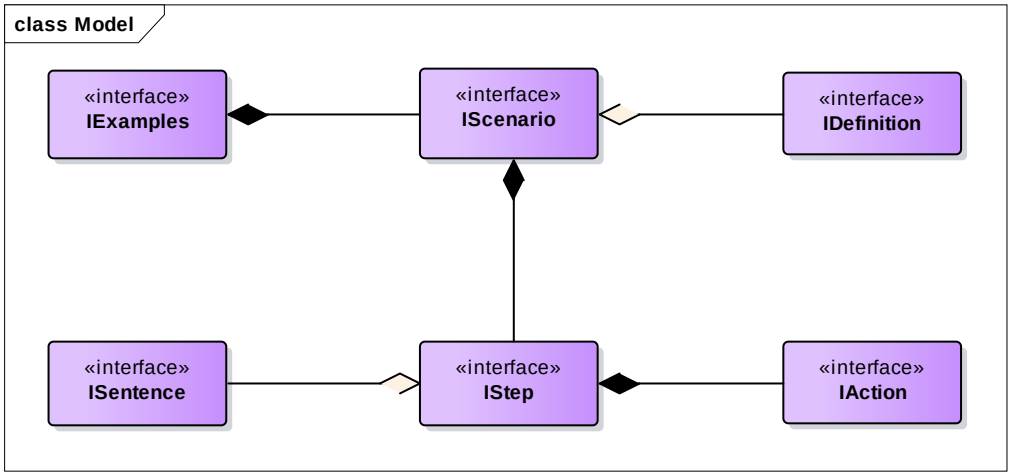
Rozhraní se týkají buď komponent, nebo vizualizační logiky jako takové. V nové aplikaci je rozhraní `IScene`, které reprezentuje scénu vizualizace, jejíž součástí mohou být různé prvky (komponenty). To mohou být vrcholy a hrany grafu implementující rozhraní `IVertex`, resp. `IEdge`, pomocné boxy s rozhraním `IHint`, obrázky s rozhraním `IImage`, osa `IAxis` a další. Tyto součásti vizualizace implementují rozhraní `IItem`, které obsahuje

vlastnosti `id` a `priority`. Vlastnost `id` zadaná řetězcem slouží jako identifikátor, tedy k dohledání konkrétního prvku. Vlastnost `priority` typu `number` není povinná a označuje prioritu při vykreslování objektů na plátně.




**Obrázek 20: Model rozhraní – komponenty**

Rozhraní, která tvoří vizualizační jádro, lze vidět v modelu na obrázku 21. Rozhraní `IExamples` je kolekcí scénařů `IScenario`. Každý scénář odpovídá jednomu průběhu vizualizace (například jednomu důkazu z teorie grafů). Rozhraní `IDefinition` reprezentuje popis jednotlivých kroků důkazu, který se zobrazuje uživateli, zatímco rozhraní `IStep` označuje krok důkazu obsahující určité akce definované pomocí rozhraní `IAction`. Tyto akce označují, který prvek má být přidán na vykreslovací plátno, který odebrán anebo upraven. Toto rozhraní nemá žádné vlastnosti, ale zato obsahuje dvě metody, a to `update` a `rollback`. Obě metody přijímají jako parametr scénu a nic nevracejí. Rozhraní `ISentence` reprezentuje matematické věty, výroky, definice a tak dále, které si může uživatel zobrazit jako připomenutí již naučené látky.




**Obrázek 21: Model rozhraní – viewer**

## 8.3 Princip vizualizace

Pro přidání vizualizačního okna do stránky je třeba vložit do ní komponentu `viewer` s nadefinovanými kroky vizualizace. Tyto kroky obsahují akce (`actions`), které se v nich provádí. V příkladu zdrojového kódu 9 jsou vidět akce jednoho kroku z důkazu o souvislosti grafu při odebrání hrany z jeho kružnice. V tomto kroku se odebrán text s identifikátorem `'graf1'`, přidán nový text (lze si všimnout použití komponenty písma kurzíva) a odebrána hrana `'v-w'`, tedy hrana mezi vrcholem $v$ a vrcholem $w$.

```
actions: [
    a.removeWhiteboard('graf1'),
    a.addWhiteboard(
        'graf2', { x: 560, y: 50 },
        ['Graf ', bs.em({}, 'G - e')], undefined, undefined, 0
    ),
    a.removeEdge('v-w')
],
```

**Zdrojový kód 9: Nadefinování akcí pro jeden krok vizualizace**

Pro představu, jak jsou definovány samotné akce, je uveden příklad funkce `removeEdge`, která se stará o odstranění hrany. Tato funkce vrací objekt implementující rozhraní `IAction`. Jak již bylo řečeno, toto rozhraní obsahuje dvě metody, `update` a `rollback`. Metoda `update` se stará o provedení akce (zde se jedná o odstranění hrany), zatímco metoda `rollback` se stará o vrácení akce (v příkladu jde o přidání původně odstraněné

hrany). Tento mechanismus je nutný proto, aby bylo možné se ve vizualizaci vrátit k předchozímu kroku důkazu.

```
export function removeEdge(id: string): IAction {
    let originalEdge: IEdge = null;
    return {
        update: scene => originalEdge = removeById(scene.edges, id),
        rollback: scene => {
            originalEdge.waitForAnimation = undefined;
            scene.edges.push(originalEdge);
        }
    };
}
```

**Zdrojový kód 10: Funkce vracející objekt akce**

## 8.4 Základní prvky aplikace

V této kapitole jsou stručně nastíněny jednotlivé základní prvky, které tvoří vizualizace ve vytvořené aplikaci. S vrcholy může uživatel hýbat, což způsobí i pohyb hran. Stejně tak se dá hýbat s pomocnými panely informujícími například o sledu vrcholů.

### 8.4.1 Vertex

Uzel v grafu je reprezentován komponentou `vertex`, pojmenovaný uzel pak komponentou `vertexWithLabel`. Komponenta vertex je tvořena SVG elementem `circle`, přičemž jeho vlastnosti jsou předávány z nadřazené komponenty (umístění elementu, poloměr kruhu a jeho barva, šířka a barva ohraničení a další).


**Obrázek 22: Vrchol grafu**

Stejně tak je této komponentě předávána funkce `onMove`, která definuje, jak se komponenta zachová v případě, že s ní uživatel hýbe. Volání této funkce se přitom děje ve vnitřní části komponenty. Komponenta `vertexWithLabel` je složena ze dvou komponent vertex, přičemž jedna z nich je pouze průhledným kruhem a slouží pro překrytí popisu

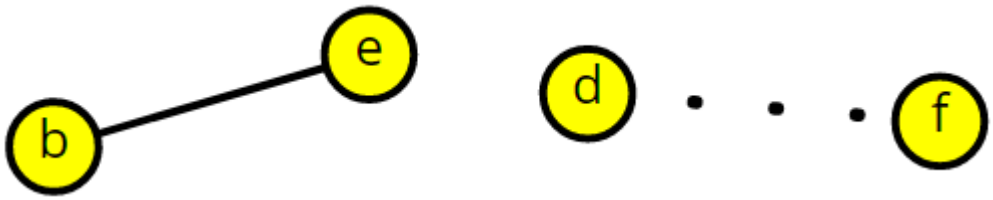
vrcholu (pro lepší ovládání pohybu). Náhled na metodu `render` je vidět v příkladu zdrojového kódu číslo 11.

```
render(ctx: IVertexWithLabelCtx, me: b.IBobrilNode) {
    me.tag = 'g';
    me.children = [
        vertex(ctx.data),
        label({
            text: ctx.data.label,
            x: ctx.data.cx,
            y: ctx.data.cy
        }),
        b.style(vertex(ctx.data), vertexInvisibleStyle)
    ];
}
```

**Zdrojový kód 11: Metoda render komponenty vertexWithLabel**

### 8.4.2 Edge

Hrana grafu je reprezentována komponentou `edge`. Tato komponenta je tvořena SVG elementem zvaným `line`. Vlastnosti elementu jako pozice čáry, barva nebo šířka jsou předávány z nadřazené komponenty přes data komponenty. Stejně tak se předává informace o tom, jestli se jedná o tečkovanou hranu, popřípadě jestli se má hrana animovat.




**Obrázek 23: Hrany v grafu**

Animace hrany je způsobena změnou CSS hodnoty `opacity` (tedy neprůhlednost). Pokud se má hrana animovat, pak je v její metodě `init` nastavena hodnota neprůhlednosti na malé číslo (0,005). V metodě `render` je pak umístěno nastavení časového limitu, po kterém se animace spustí, a to pomocí metody `setTimeout`. Jakmile tento limit uběhne, tak se zavolá metoda `createInterval`, ve které se vytvoří interval pomocí metody `setInterval`. Tento interval v časových úsecích zvyšuje hodnotu neprůhlednosti, a to až dokud není hrana úplně neprůhledná.

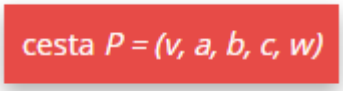
```
function createInterval() {
    let inter = setInterval(
        () => {
            ctx.opac += (0.04 / constants.slowedTime);
            b.invalidate(ctx);
            if (Math.floor(ctx.opac) === 1) {
                clearInterval(inter);
            }
        },
        100);
}

if (ctx.opac === 0.005) {
    setTimeout(createInterval,ctx.data.waitForAnimation*constants.slowedTime);
}
```

**Obrázek 24: Animace hrany**

### 8.4.3 Hint

Jako `hint` je označena komponenta, která reprezentuje pomocný panel s informacemi o sledu nebo cestách v grafu. Stejně jako u předchozí komponenty je možné i s touto pohybovat nebo ji animovat. Je tvořena HTML elementem `div`.




**Obrázek 25: Komponenta pomocného panelu**

## 8.5 Implementační problémy

Při tvorbě aplikace se vyskytlo několik problémů, jejichž implementaci je vhodné představit.

### 8.5.1 Interpolace

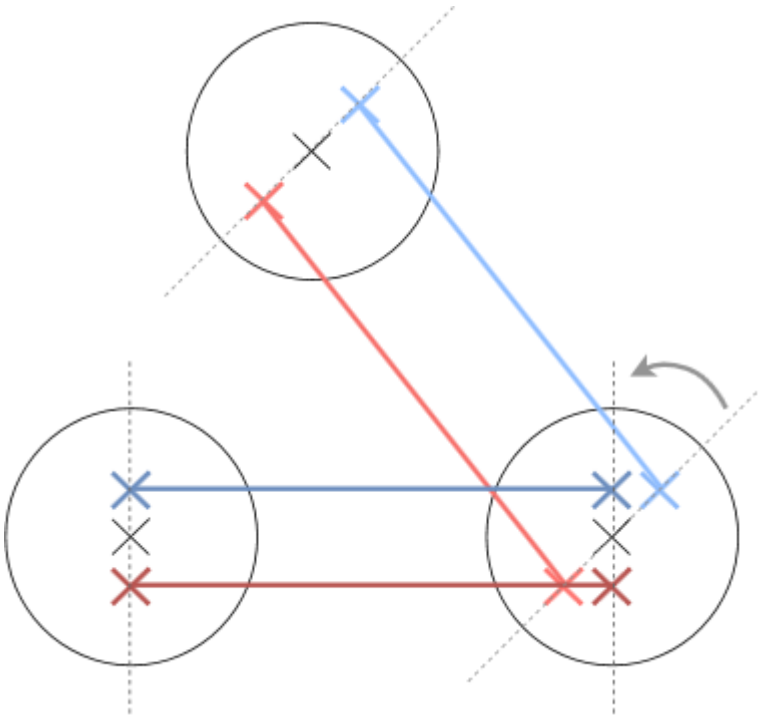
Interpolace je v aplikaci využívána na více místech, například při výpočtu pozice nového bodu u dvou hran, což je podrobně popsáno v další kapitole. Také se pomocí interpolace počítá obarvení určité části hrany (například když je potřeba zvýraznit pouze jednu polovinu hrany).

```
export function linearInterpolation(a: number, b: number, t: number) {
    return (1 - t) * a + t * b;
}
```

**Obrázek 26: Lineární interpolace**

### 8.5.2 Dvě hrany

V některých vizualizacích důkazů jsou dva vrcholy propojeny dvěma hranami. Protože je s vrcholy možno hýbat, tak by se i hrany tomuto pohybu měly přizpůsobit. To není problém v případě jedné hrany, jejíž počáteční bod odpovídá pozici počátečního vrcholu a koncový bod odpovídá koncovému vrcholu. U dvou hran toto není možné, protože by se překrývaly. Proto bylo potřeba odvodit jejich počáteční a koncové body v závislosti na pozici vrcholů, která se může měnit, a to tak, aby vzdálenost nového bodu od středu vrcholu byla pořád stejná (mezi novým i starým bodem by měla procházet přímka stejná u obou vrcholů). Pro ilustraci je předložen obrázek.



**Obrázek 27: Dvě hrany spojující dva vrcholy**

Prvním krokem je vypočtení směrového vektoru, který odpovídá směru hrany (od počátečního ke koncovému vrcholu). Poté se z něj odvodí normálový vektor, a to prohozením jeho souřadnic a změněním znaménka u první souřadnice.

```
export function computeNormal(x1: number, x2: number, y1: number, y2: number)
    : IPoint {
    let v1 = x2 - x1;
    let v2 = -(y2 - y1);
    let u1 = -v2;
    let u2 = v1;
    return { x: u1, y: u2 };
}
```

**Zdrojový kód 12: Souřadnice směrového a normálového vektoru**

Dalším krokem je vypočtení délky normály, díky které je možné odvodit interpolační parametr *t*. Ten je vypočten dělením parametru `moved` (vzdálenost od středu vrcholu) proměnnou `lengthOfNormal` (délka normály). Interpolační parametr je následně použit při vypočtení souřadnic nového bodu.

```
export function computeNewPoint(
    normal: IPoint, x1: number,
    y1: number, moved: number): IPoint {
    let lengthOfNormal = Math.hypot(normal.x, -normal.y);
    let interpolationParameter = (!!moved && moved) / lengthOfNormal;
    let newX = linearInterpolation(x1, x1 + normal.x, interpolationParameter);
    let newY = linearInterpolation(y1, y1 - normal.y, interpolationParameter);
    return { x: newX, y: newY };
}
```

**Zdrojový kód 13: Výpočet nového bodu hrany**

Protože má hrana dva vrcholy, tak je třeba zavolat výše zmíněnou metodu dvakrát – pro vypočtení pozice obou vrcholů. V druhém případě je volána s obrácenými hodnotami souřadnic vrcholů, protože se koncový vrchol tentokrát ustanoví jako počáteční a obráceně.

```
export function computeMovedPosition(
    x1: number, x2: number,
    y1: number, y2: number, moved: number): IPoint[] {
    let firstNormal = computeNormal(x1, x2, y1, y2);
    let secondNormal = computeNormal(x2, x1, y2, y1);
    let firstPoint = computeNewPoint(firstNormal, x1, y1, moved);
    let secondPoint = computeNewPoint(secondNormal, x2, y2, -moved);
    return [firstPoint, secondPoint];
}
```

**Zdrojový kód 14: Vypočtení obou bodů hrany**

### 8.5.3 Přepočtení souřadnic

Když uživatel zvětší vizualizaci na celou obrazovku nebo změní velikost okna, pak je potřeba přepočítat pozici prvků vizualizace. To se týká pouze komponent tvořených HTML elementy, protože SVG elementy se přepočítávají samy na základě atributu `viewBox` tagu `svg`. Při přepočtu pozice prvků se používá následující část kódu (zde přepočet pozice komponenty nápomocného panelu `hint`).

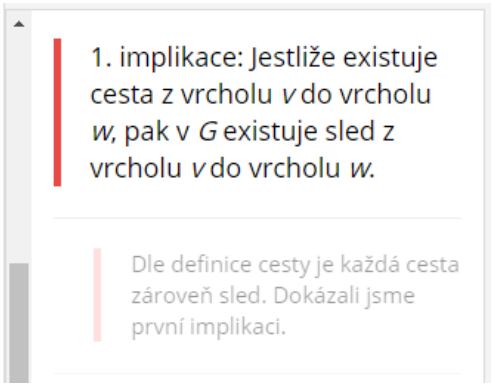
```
hintData.countedPoint.y = (
    (hintData.point.y * svg.getBoundingClientRect().height) / constants.height
);
hintData.countedPoint.x = (
    (hintData.point.x * svg.getBoundingClientRect().width) / constants.width
);
```

**Obrázek 28: Přepočtení pozice prvku**

Metoda `getBoundingClientRect` vrací pozici a velikost elementu `svg`. Konstanty `height` a `width` odpovídají původní velikosti `svg` prvku na stránce, od které se výpočty odvíjejí. Tento přepočet musí probíhat v metodě `postUpdateDom`, protože jinak by se nebylo možné dostat k vypočtené velikosti prvků ve skutečném Document Object Modelu.

### 8.5.4 Animace posuvníku

V aplikaci je také vytvořena animace posuvníku, který je umístěn v komponentě s vypsanými kroky důkazu. Aktuální krok vizualizace se vždy automaticky zvětší a zvýrazní, aby upoutal pozornost uživatele. Stejně tak je posunut tak, aby byl vidět na obrazovce (když je to možné, tak je umístěn nejvýše). Animace posunutí je dosaženo pomocí vlastností `scrollTop` a `offsetTop`.




**Obrázek 29: Zvětšení aktuálního kroku důkazu**

# 9 Popis a ovládání aplikace

Tato kapitola obsahuje uživatelskou příručku k aplikaci, která byla pojmenována GraPro (zkratka ze slov graph proofs, tedy důkazy grafů). Vytvořená aplikace slouží k výuce důkazů v teorii grafů a obsahuje vizualizace negací a důkazů, se kterými se dá interaktivně pracovat. Také je možné si negace procvičit. Aplikaci lze spouštět lokálně nebo je možné ji umístit na server.

## *9.1 Požadavky na konfiguraci*

Aplikace nemá výraznější požadavky na konfiguraci systému z hlediska výkonu. Pro nejlepší uživatelskou zkušenost je doporučeno spouštět aplikaci v prohlížeči Google Chrome, popřípadě Mozilla Firefox, a to v nejnovějších verzích prohlížečů. Minimální rozlišení obrazovky by mělo být 1024x768.

Aplikace funguje i na mobilních zařízeních. V tomto případě je doporučen prohlížeč Google Chrome, Firefox Focus nebo Safari. V těchto prohlížečích by se webová stránka měla přizpůsobit velikosti displeje a žádná funkcionalita by neměla být omezena. Přesto je doporučeno aplikaci používat minimálně na zařízeních velikosti tabletu, aby bylo možné ji pohodlně ovládat.

## *9.2 Složení aplikace*

Aplikace GraPro se skládá ze tří hlavních částí:

- Výroková logika - tato část obsahuje základní teoretický úvod do této oblasti, který je nutný pro pochopení principu matematických důkazů.
- Typy důkazů - zde jsou představeny základní typy matematických důkazů.
- Důkazy - tato část obsahuje samotné vizualizace vybraných důkazů z teorie grafů.

### 9.2.1 Výroková logika

Tato část obsahuje šest menších částí:

- výroky,
- kvantifikované výroky,
- vizualizace negací,

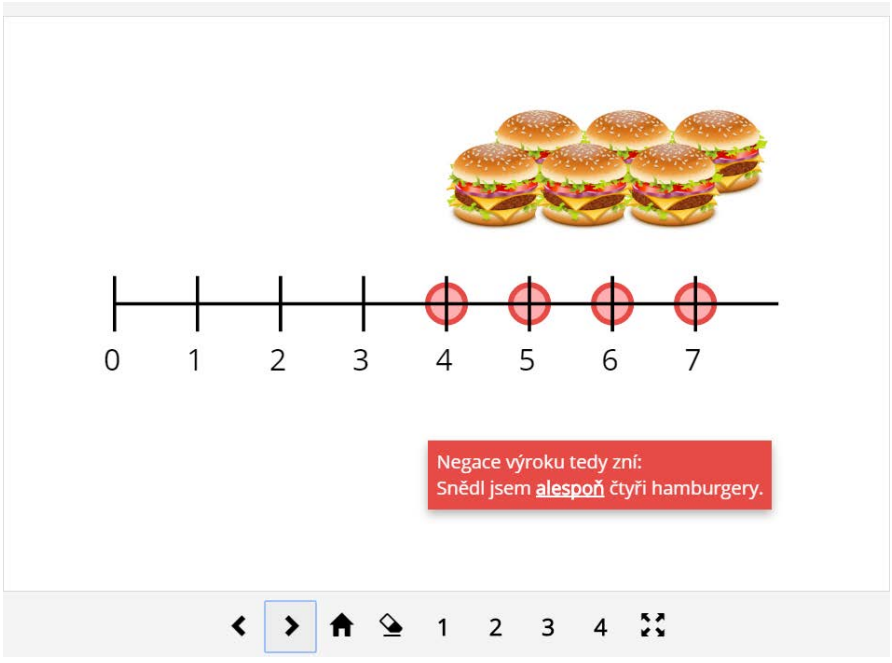
- složené výroky,
- matematické věty,
- procvičení.

Části s vizualizacemi a procvičením negací jsou interaktivní, proto se jim příručka věnuje podrobněji.

#### 9.2.1.1 Vizualizace negací

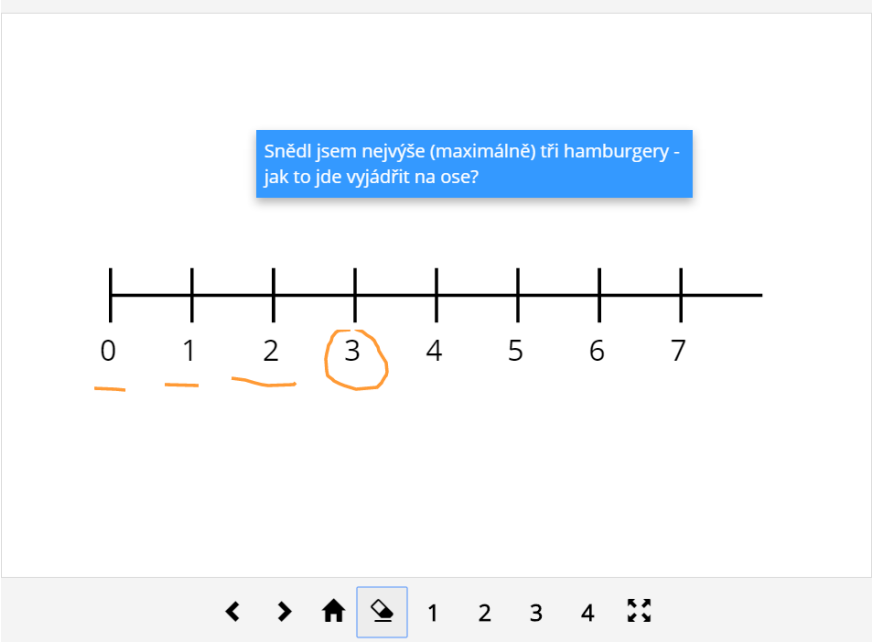
V této části je možné si interaktivně projít vizualizace negování čtyř typů kvantifikovaných výroků, které by měly usnadnit pochopení negací těchto výroků. Obrazovka vizualizace obsahuje několik ovládacích prvků:

- tlačítka pro pohyb zpět a vpřed,
- tlačítko pro návrat na první krok,
- tlačítko pro mazání nakreslených tvarů,
- tlačítka 1-4 pro přechod na další typy kvantifikovaných výroků,
- tlačítko pro zvětšení na celou obrazovku.




**Obrázek 30: Vizualizace negací**

Na zobrazovací plochu je možné kreslit oranžovým perem, což může být užitečné zejména pro vyučujícího při výuce:




**Obrázek 31: Kreslení na plátno**

### 9.2.2 Typy důkazů

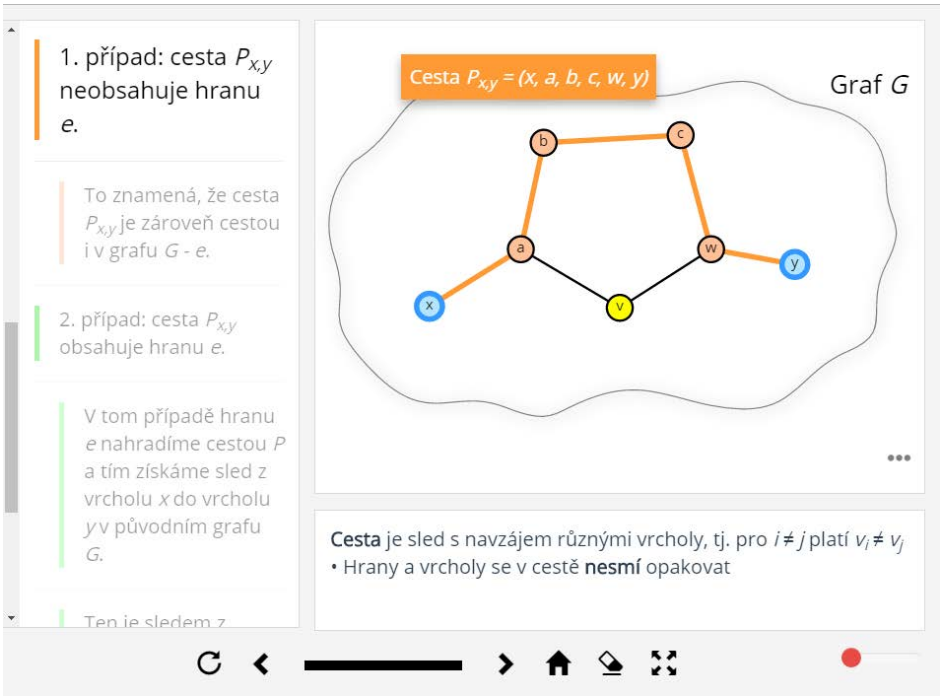
Tato část obsahuje teoretické představení obecných typů matematických důkazů.

### 9.2.3 Důkazy

V této části jsou vizualizace vybraných důkazů z teorie grafů, jako je například princip sudosti a další. "Přehrávač" důkazů obsahuje kroky důkazu na levé straně, hlavní vykreslovací plochu na pravé straně a tlačítko pro zobrazení definic a dalších pomocných informací, které se zobrazují pod hlavní obrazovkou. S prvky jako jsou vrcholy a barevné tipy lze hýbat s pomocí myši. Obrazovka vizualizace důkazů obsahuje několik ovládacích prvků:

- tlačítko pro opakování animace v daném kroku důkazu,
- tlačítka pro pohyb zpět a vpřed,
- ukazatel průběhu animace,
- tlačítko pro návrat na první krok,
- tlačítko pro mazání nakreslených tvarů,

- tlačítko pro zvětšení na celou obrazovku,
- posuvník pro zpomalení animace.




**Obrázek 32: Vizualizace důkazů z teorie grafů**

# 10 Testování aplikace a návrhy rozšíření

Aplikace byla testována pomocí manuálního testování a unit testů, které byly napsány zejména pro funkce poskytující výpočet hodnot.

## *10.1 Jednotkové testy Jasmine*

Framework Jasmine slouží k programování Unit testů, někdy také nazývaných jednotkové testy. Unit testy slouží k testování jednotlivých částí (jednotek) zdrojového kódu. Jako příklad je zde uveden test testující výpočet dříve zmíněné lineární interpolace. Funkce, která se o výpočet stará, je zavolána s určitými hodnotami a její výsledek je porovnán s výsledkem očekávaným.

```
describe('computer', () => {
    describe('interpolation', () => {
        it('should compute interpolated value', () => {
            let result = computer.linearInterpolation(5, 10, 0.5);
            expect(result).toEqual(7.5);
        });
    });
});
```

**Zdrojový kód 15: Příklad Jasmine testu**

## *10.2 Manuální testování*

Aplikace byla manuálně testována jak autorkou práce, tak průběžně i zadavatelkou práce. Ještě v průběhu vývoje byla také předložena studentům předmětu Diskrétní matematika na Univerzitě Hradec Králové, díky jejichž zpětné vazbě bylo možné přizpůsobit aplikaci přímo studentům, tedy uživatelům, kteří ji budou používat. Ti poskytli zejména následující podněty pro vylepšení aplikace:

- možnost přeskočit animaci bez jejího dokončení,
- upozornění na ukončení vizualizace,
- ovládání rychlosti animace,
- více vizualizací důkazů.

Výše zmíněné připomínky byly zohledněny při tvorbě aplikace a potřebná funkcionalita byla do aplikace doplněna. Studenti také vyjádřili, co jim na aplikaci nejvíce vyhovovalo. Výhody aplikace dle studentů se dají shrnout následovně:

- kvalitní provedení vizualizace,
- snadné ovládání a přehlednost,
- interaktivní prostředí a animace,
- možnost vlastního pohybu v čase.

## *10.3 Návrhy rozšíření do budoucna*

Aplikaci by bylo možné rozšířit o další důkazy matematických vět nebo o vizualizace dalších příkladů negací. Zajímavé by bylo rozšíření o editor vizualizací, což by mohl být námět například další závěrečné práce. Editor by uživateli umožňoval vytvářet a editovat vizualizace důkazů tak, aby je „přehrávač“ vizualizací v aplikaci mohl zobrazovat.

# 11 Shrnutí výsledků

Cílem práce bylo vytvoření aplikace pro vizualizaci důkazů z teorie grafů a dále také pro vizualizaci negací, a to s pomocí moderních technologií. Cíle zmíněné ve druhé kapitole práce byly naplněny naprogramováním webové aplikace GraPro, která odpovídá zmíněným uživatelským požadavkům (jak funkčním, tak nefunkčním).

Součástí práce je úvod do problematiky výuky důkazů a představení vizualizace jako názorné formy výuky. Práce dále obsahuje teoretický úvod do oblasti výrokové logiky, důkazů a teorie grafů. Praktická část práce se skládá z analýzy projektu a porovnání technologií pro tvorbu aplikace. Jako součást srovnání byl proveden benchmark frameworků React a Bobril, ve kterém byl ve vykreslování rychlejší framework Bobril. Práce obsahuje popis implementačních problémů, které při vývoji vznikly a nabízí jejich řešení. V neposlední řadě byla vytvořena uživatelská příručka, která je i součástí zhotovené aplikace.

# 12 Závěry a doporučení

Jak bylo zmíněno v úvodu, výuka důkazů je důležitou součástí matematiky, která je ale pro studenty látkou náročnou a těžko uchopitelnou. V tom může pomoci vizualizace důkazů, a to i jako součást e-learningového prostředí. S pomocí moderních programovacích nástrojů lze vytvořit aplikaci, která kombinuje interaktivní vizualizace s textovými zdroji.

Ukázalo se, že komponentově orientované frameworky s použitím virtuálního Document Object Modelu jsou vhodné pro programování e-learningových aplikací, a to zejména těch, týkajících se vizualizace učební látky. Znovupoužitelné komponenty umožňují takovou aplikaci sestavit snadněji a předností je také přehlednost jejich kódu. Princip single page aplikací je vhodný pro takové projekty, které není nutné umístit na server, a tedy i pro výukový software, který si uživatel může spustit pouze lokálně, což odpovídá daným požadavkům. Pro tvorbu takové klientské aplikace je možné použít framework Bobril, který poskytuje dostatečné prostředky pro plnohodnotný vývoj, a to včetně sestavovacího nástroje Bobril-build.

Vytvořená aplikace byla vyvinuta i otestována a je připravena pro použití ve výuce předmětů zabývajících se teorií grafů, důkazy a související výrokovou logikou. V budoucnosti by bylo možné ji rozšířit například o editor vizualizací, který by uživatelům umožňoval vytvářet si vlastní vizualizace důkazů.

# Seznam použité literatury


[1] ŠTRAUSOVÁ, Irena. Vizuální důkazy ve výuce matematiky. *South Bohemia Mathematical Letters*. 2012, 20(1), 48-56.

[2] HANNA, Gila a Nathan SIDOLI. Visualisation and proof: a brief survey of philosophical perspectives. *ZDM* [online]. 2007-3-2, 39(1-2), 73-78 [cit. 2017-04-22]. DOI: 10.1007/s11858-006-0005-0. ISSN 1863-9690. Dostupné z: http://link.springer.com/10.1007/s11858-006-0005-0

[3] KILIC, Hulya. The effects of dynamic geometry software on learning geometry. *Proceedings of CERME*. 2009, 8. Dostupné také z: http://cerme8.metu.edu.tr/wgpapers/WG15/WG15_Kilic.pdf

[4] MILKOVÁ, Eva. *Teorie grafů a grafové algoritmy*. Hradec Králové: Gaudeamus, 2013. ISBN 978-80-7435-267-6.

[5] ČADA, Roman, Tomáš KAISER a Zdeněk RYJÁČEK. *Diskrétní matematika*. Plzeň: Západočeská univerzita v Plzni, 2004. ISBN 80-7082-939-7.

[6] MATOUŠEK, Jiří a Jaroslav NEŠETŘIL. *Kapitoly z diskrétní matematiky*. 3., upr. A dopl. Vyd. V Praze: Karolinum, 2007. ISBN 978-80-246-1411-3.

[7] THIELE, Rüdiger. *Matematické důkazy*. Praha: Státní nakladatelství technické literatury, 1985. Polytechnická knižnice (SNTL).

[8] JavaScript Tutorial. *W3Schools* [online]. Refsnes Data, 2017 [cit. 2017-03-19]. Dostupné z: https://www.w3schools.com/js/

[9] *TypeScript: JavaScript that scales* [online]. Redmond: Microsoft, 2016 [cit. 2017-03-19]. Dostupné z: https://www.typescriptlang.org/

[10] RŮT, Tomáš. Bobril – I: Getting Started. *CodeProject: For those who code* [online]. 2017 [cit. 2017-03-19]. Dostupné z: https://www.codeproject.com/Articles/1044425/Bobril-I-Getting-Started

[11] CABASSUT, Richard, AnnaMarie CONNER, Filyet Aslı İŞÇIMEN, Fulvia FURINGHETTI, Hans Niels JAHNKE a Francesca MORSELLI. Conceptions of Proof – In Research and Teaching. In: *Proof and Proving in Mathematics Education: The 19th ICMI Study*. Springer, 2012, s. 169. DOI: 10.1007/978-94-007-2129-6_7. ISBN 10.1007/978-94-007-2129-6. ISSN 1387-6872. Dostupné také z: http://www.springerlink.com/index/10.1007/978-94-007-2129-6_7

[12] GRABINER, Judith V. Why Proof? A Historian's Perspective. In: *Proof and Proving in Mathematics Education: The 19th ICMI Study*. Springer, 2012, s. 147. DOI: 10.1007/978-94-007-2129-6_6. ISSN 1387-6872. Dostupné také z: http://www.springerlink.com/index/10.1007/978-94-007-2129-6_6

[13] SNODGRASS, Joan Gay, Rochelle VOLVOVITZ a Eleanor Radin WALFISH. Recognition memory for words, pictures, and words + pictures. *Psychonomic Science* [online]. 1972, 27(6), 345-347 [cit. 2017-03-19]. DOI: 10.3758/BF03328986. ISSN 0033-3131. Dostupné z: http://link.springer.com/10.3758/BF03328986

[14] BÍLEK, Martin. Learning from Graphical Presentation in Science and Technical Education. In: *Visualization in Science and Technical Education*. Gaudeamus, Hradec Králové, 2003, s. 9 – 12. ISBN 80-7041-497-9.

[15] MILKOVÁ, Eva. E-learning: postřehy a zkušenosti. In: *Sborník příspěvků ze semináře a soutěže eLearning*. Hradec Králové: Gaudeamus, Univerzita Hradec Králové, 2004, s. 320-324. ISBN 80-7041-798-6.

[16] DREYFUS, Tommy, Elena NARDI a Roza LEIKIN. Forms of Proof and Proving in the Classroom. In: *Proof and Proving in Mathematics Education: The 19th ICMI Study*. Springer, 2012, s. 191. DOI: 10.1007/978-94-007-2129-6_8. ISBN 10.1007/978-94-007-2129-6. ISSN 1387-6872. Dostupné také z: http://www.springerlink.com/index/10.1007/978-94-007-2129-6_8

[17] ŠEDIVÝ, Jaroslav, Júlia LUKÁTŠOVÁ, Oldřich ODVÁRKO a Michal ZÖLDY. *Úlohy o výrokoch a množinách pre 1. ročník gymnázia.* Bratislava: Slovenské pedagogické nakladateľstvo, 1970.

[18] PRAŽÁK, Pavel. *Matematika 1*. Hradec Králové: Gaudeamus, 2012. ISBN 978-80-7435-227-0.

[19] ČECH, Vlastimil. *Proč děláme důkazy v matematice.* Praha: Státní pedagogické nakladatelství, 1971. Knižnice všeobecného vzdělávání mládeže.

[20] Základy matematiky: Výroková logika. In: *MATEMATIKA online* [online]. Ústav matematiky FSI VUT Brno, 2005 [cit. 2017-04-05]. Dostupné z: http://mathonline.fme.vutbr.cz/Logika-dukazy-mnoziny-relace-zobrazeni-algebraicke-struktury/sc-15-sr-1-a-31/default.aspx

[21] RACLAVSKÝ, Jiří. Úvod do logiky (VL): 13. Axiomatické systémy VL a pojem důkazu. In: *Projekty Operačního programu Vzdělávání pro konkurenceschopnost* [online]. Plzeň: Katedra filozofie FF ZČU v Plzni, 2014 [cit. 2017-04-05]. Dostupné z: https://www.esf.kfi.zcu.cz/logika/opory/phk1102/VL_13_Formalni_systemy.pdf

[22] MIŠKOVSKÝ, Pavel. Důkazy. In: *Gymnázium Na Vítězné pláni* [online]. Praha 4: Gymnázium Na Vítězné pláni, 2007 [cit. 2017-04-05]. Dostupné z: http://www.gvp.cz/~vinkle/mafynet/M_opakovaci_seminar/02_studijni_texty/Dukazy.pdf

[23] Co je to důkaz. *Matematika.cz* [online]. Brno-město: Vydavatelství Nová média, 2014 [cit. 2017-04-05]. Dostupné z: http://www.matematika.cz/dukazy

[24] HABALA, Petr. Diskrétní matematika: 5. Indukce a rekurze. In: *ČVUT: Katedra matematiky* [online]. 2012 [cit. 2017-04-06]. Dostupné z: https://math.feld.cvut.cz/habala/teaching/dma/dmknih05.pdf

[25] Discrete Mathematics. *Wolfram MathWorld* [online]. Wolfram Research, ©1999-2017 [cit. 2017-04-05]. Dostupné z: http://mathworld.wolfram.com/DiscreteMathematics.html

[26] HÍC, Pavel a Milan POKORNÝ. Grafové algoritmy v školskej praxi. In: *E-learning 2004: Sborník příspěvků ze semináře a soutěže e-learning 2004*. Hradec Králové: Gaudeamus, 2004, s. 94-101. ISBN 80-7041-798-6.

[27] ARLOW, Jim a Ila NEUSTADT. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky*. 2., aktualiz. a dopl. vyd. Brno: Computer Press, 2007. ISBN 978-80-251-1503-9.

[28] Functional and nonfunctional requirements. *Search Software Quality* [online]. TechTarget, 2017 [cit. 2017-04-08]. Dostupné z: http://searchsoftwarequality.techtarget.com/answer/Functional-and-nonfunctional-requirements

[29] *GraphTea: Your buddy to teach, learn and research on graph theory.* [online]. [cit. 2017-04-08]. Dostupné z: http://www.graphtheorysoftware.com/

[30] *Graph Magics* [online]. 2004-2017: Dumitru Ciubatii [cit. 2017-04-08]. Dostupné z: http://www.graph-magics.com/

[31] *Gephi: The Open Graph Viz platform* [online]. 2017 [cit. 2017-04-08]. Dostupné z: https://gephi.org/

[32] ŠEVČÍKOVÁ, Andrea a Eva MILKOVÁ. Multimedia applications: Graph algorithms visualization. In: *2016 IEEE 17th International Symposium on Computational Intelligence and Informatics (CINTI)* [online]. IEEE, 2016, s. 000231-000236 [cit. 2017-04-22]. DOI: 10.1109/CINTI.2016.7846409. ISBN 978-1-5090-3909-8. Dostupné z: http://ieeexplore.ieee.org/document/7846409/

[33] ŠITINA, Jiří. *Grafové algoritmy a jejich vizualizace*. 2010. Diplomová práce. Univerzita Hradec Králové. Vedoucí práce Eva Milková.

[34] MILKOVÁ, Eva a Andrea ŠEVČÍKOVÁ. Deeper insight into graph theory using multimedia applications. *International Journal of Circuits, Systems and Signal Processing*. 2016, 10, 448-454. ISSN 1998-4464. Dostupné také z: http://www.naun.org/main/NAUN/circuitssystemssignal/2016/b602005-aap.pdf

[35] ČÁPKA, David. 1. díl - Úvod do Single Page Application v ASP.NET. In: *ITnetwork.cz* [online]. 2017 [cit. 2017-04-10]. Dostupné z: http://www.itnetwork.cz/csharp/asp-net/single-page-application/tutorial-uvod-do-asp-net-single-page-application

[36] NENOV, Simeon. Single-Page Application with Kendo UI. In: *Simeon Nenov: Programming and many more* [online]. 2013 [cit. 2017-04-10]. Dostupné z: https://ssnenov.wordpress.com/2013/03/24/single-page-up-with-kendo-ui/

[37] Angular 2 - Overview. In: *Tutorialspoint* [online]. India, 2017 [cit. 2017-04-15]. Dostupné z: https://www.tutorialspoint.com/angular2/angular2_overview.htm

[38] ELLIOTT, Eric. Angular 2 vs React: The Ultimate Dance Off. In: *Medium* [online]. 2016 [cit. 2017-04-15]. Dostupné z: https://medium.com/javascript-scene/angular-2-vs-react-the-ultimate-dance-off-60e7dfbc379c

[39] STEIGERWALD, Daniel. What's wrong with Angular 1. In: *Medium* [online]. 2014 [cit. 2017-04-15]. Dostupné z: https://medium.com/este-js-framework/whats-wrong-with-angular-js-97b0a787f903

[40] TRIVEDI, Jignesh. Data Binding in AngularJS. In: *C# Corner: A Social Community of Developers and Programmers* [online]. C# Corner, 2017 [cit. 2017-04-15]. Dostupné z: http://www.c-sharpcorner.com/UploadFile/ff2f08/data-binding-in-angularjs/

[41] RŮT, Tomáš. Bobril - V - Bobril-build. In: *CodeProject: For those who code* [online]. 2017 [cit. 2017-04-15]. Dostupné z: https://www.codeproject.com/Articles/1167901/bobril-build

[42] UI Benchmark. In: *GitHub Pages* [online]. GitHub, 2017 [cit. 2017-04-15]. Dostupné z: https://localvoid.github.io/uibench/

[43] PANDEY, Harshit. ReactJS | Learning Virtual DOM and React Diff Algorithm. In: *Oyecode* [online]. 2016 [cit. 2017-04-16]. Dostupné z: http://www.oyecode.com/2015/09/reactjs-learning-virtual-dom-and-react.html

[44] Bobril core. *GitHub* [online]. 2017 [cit. 2017-04-16]. Dostupné z: https://github.com/Bobris/Bobril/blob/master/src/bobril.md

[45] *Bobrilstrap: Bobril wrapper of the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web - Bootstrap.* [online]. 2017 [cit. 2017-04-16]. Dostupné z: http://keeema.github.io/bobrilstrap/

[46] SVG (basic support). In: *Can I use* [online]. 2017 [cit. 2017-04-17]. Dostupné z: http://caniuse.com/#feat=svg

# Seznam příloh

A. Benchmark – výsledky

B. Obsah kompaktního disku

# A Benchmark - výsledky

Tabulka obsahuje výsledky výkonnostního srovnání frameworků React (verze 15.3.1) a Bobril (verze 4.38.0) s využitím nástroje UI Benchmark [42].

| | **React** 15.3.1 | **Bobril** 4.38.0 |
|---|---|---|
| **Flags:** | | |
| Measure Full Render Time | | |
| Preserve State | | |
| sCU Optimization | | |
| DOM Recycling | | |
| Spec Tests | | |
| **Times:** | | |
| JS Init Time | 46755 (3.13) | 14920 |
| First Render Time | 82295 (4.17) | 19755 |
| Overall Tests Time | 469530 | 180020 |
| Iterations | 5 | 5 |
| table/[100,4]/render | 31770 (4.86) | 6540 |
| table/[50,4]/render | 15355 (4.40) | 3490 |
| table/[100,2]/render | 20890 (4.34) | 4810 |
| table/[50,2]/render | 10410 (4.55) | 2290 |
| table/[100,4]/removeAll | 7520 (2.43) | 3090 |
| table/[50,4]/removeAll | 3775 (2.28) | 1655 |
| table/[100,2]/removeAll | 4735 (2.11) | 2245 |
| table/[50,2]/removeAll | 2910 (2.46) | 1185 |
| table/[100,4]/sort/0 | 3690 | 3790 (1.03) |
| table/[50,4]/sort/0 | 1710 | 2010 (1.18) |
| table/[100,2]/sort/0 | 3455 (1.36) | 2540 |
| table/[50,2]/sort/0 | 1415 | 1445 (1.02) |
| table/[100,4]/sort/1 | 3735 | 4470 (1.20) |
| table/[50,4]/sort/1 | 2360 (1.19) | 1990 |
| table/[100,2]/sort/1 | 3710 (1.11) | 3345 |
| table/[50,2]/sort/1 | 1580 (1.08) | 1465 |
| table/[100,4]/filter/32 | 1260 (3.00) | 420 |
| table/[50,4]/filter/32 | 790 (3.22) | 245 |
| table/[100,2]/filter/32 | 1460 (3.89) | 375 |
| table/[50,2]/filter/32 | 700 (3.33) | 210 |
| table/[100,4]/filter/16 | 1405 (2.70) | 520 |
| table/[50,4]/filter/16 | 945 (2.86) | 330 |

| | | |
|---|---|---|
| table/[100,2]/filter/16 | 1260 (2.47) | 510 |
| table/[50,2]/filter/16 | 850 (3.04) | 280 |
| table/[100,4]/filter/8 | 1750 (1.59) | 1100 |
| table/[50,4]/filter/8 | 1025 (1.54) | 665 |
| table/[100,2]/filter/8 | 1610 (1.86) | 865 |
| table/[50,2]/filter/8 | 1340 (2.85) | 470 |
| table/[100,4]/filter/4 | 2410 (1.92) | 1255 |
| table/[50,4]/filter/4 | 1375 (2.05) | 670 |
| table/[100,2]/filter/4 | 2420 (2.23) | 1085 |
| table/[50,2]/filter/4 | 1170 (2.13) | 550 |
| table/[100,4]/activate/32 | 1790 (3.28) | 545 |
| table/[50,4]/activate/32 | 685 (3.19) | 215 |
| table/[100,2]/activate/32 | 1900 (5.85) | 325 |
| table/[50,2]/activate/32 | 665 (3.24) | 205 |
| table/[100,4]/activate/16 | 1605 (2.51) | 640 |
| table/[50,4]/activate/16 | 810 (2.75) | 295 |
| table/[100,2]/activate/16 | 1465 (2.90) | 505 |
| table/[50,2]/activate/16 | 765 (2.78) | 275 |
| table/[100,4]/activate/8 | 2295 (3.30) | 695 |
| table/[50,4]/activate/8 | 1100 (2.39) | 460 |
| table/[100,2]/activate/8 | 1655 (2.61) | 635 |
| table/[50,2]/activate/8 | 935 (2.53) | 370 |
| table/[100,4]/activate/4 | 2920 (1.81) | 1615 |
| table/[50,4]/activate/4 | 1740 (2.72) | 640 |
| table/[100,2]/activate/4 | 2250 (1.89) | 1190 |
| table/[50,2]/activate/4 | 1250 (2.25) | 555 |
| anim/100/32 | 1630 (4.02) | 405 |
| anim/100/16 | 1480 (3.57) | 415 |
| anim/100/8 | 1995 (3.12) | 640 |
| anim/100/4 | 3240 (2.32) | 1395 |
| tree/[500]/render | 22305 (2.76) | 8070 |
| tree/[50,10]/render | 22240 (3.07) | 7245 |
| tree/[10,50]/render | 20100 (4.18) | 4805 |
| tree/[5,100]/render | 19585 (4.75) | 4125 |
| tree/[2,2,2,2,2,2,2,2,2,2]/render | 88810 (5.00) | 17750 |
| tree/[500]/removeAll | 6655 (1.11) | 6010 |
| tree/[50,10]/removeAll | 4785 (2.43) | 1970 |

| | | |
|---|---|---|
| tree/[10,50]/removeAll | 4120 (2.42) | 1705 |
| tree/[5,100]/removeAll | 3675 (2.23) | 1645 |
| tree/[2,2,2,2,2,2,2,2,2,2]/removeAll | 12295 (2.94) | 4175 |
| tree/[500]/[reverse] | 9620 | 10565 (1.10) |
| tree/[50,10]/[reverse] | 3590 (1.56) | 2305 |
| tree/[10,50]/[reverse] | 2400 (1.55) | 1550 |
| tree/[5,100]/[reverse] | 1705 (1.37) | 1245 |
| tree/[500]/[insertFirst(1)] | 5080 (9.77) | 520 |
| tree/[50,10]/[insertFirst(1)] | 730 (2.86) | 255 |
| tree/[10,50]/[insertFirst(1)] | 475 (2.71) | 175 |
| tree/[5,100]/[insertFirst(1)] | 480 (3.10) | 155 |
| tree/[500]/[insertLast(1)] | 3665 (6.32) | 580 |
| tree/[50,10]/[insertLast(1)] | 745 (3.47) | 215 |
| tree/[10,50]/[insertLast(1)] | 435 (2.90) | 150 |
| tree/[5,100]/[insertLast(1)] | 395 (2.72) | 145 |
| tree/[500]/[removeFirst(1)] | 4400 (8.54) | 515 |
| tree/[50,10]/[removeFirst(1)] | 865 (3.20) | 270 |
| tree/[10,50]/[removeFirst(1)] | 895 (1.69) | 530 |
| tree/[5,100]/[removeFirst(1)] | 1515 (1.73) | 875 |
| tree/[500]/[removeLast(1)] | 4330 (7.22) | 600 |
| tree/[50,10]/[removeLast(1)] | 745 (3.63) | 205 |
| tree/[10,50]/[removeLast(1)] | 645 (2.63) | 245 |
| tree/[5,100]/[removeLast(1)] | 1710 (5.10) | 335 |
| tree/[500]/[moveFromEndToStart(1)] | 9085 (15.53) | 585 |
| tree/[50,10]/[moveFromEndToStart(1)] | 3105 (14.79) | 210 |
| tree/[10,50]/[moveFromEndToStart(1)] | 1950 (7.65) | 255 |
| tree/[5,100]/[moveFromEndToStart(1)] | 1860 (5.24) | 355 |
| tree/[500]/[moveFromStartToEnd(1)] | 3700 (6.27) | 590 |
| tree/[50,10]/[moveFromStartToEnd(1)] | 745 (2.57) | 290 |
| tree/[10,50]/[moveFromStartToEnd(1)] | 795 (1.51) | 525 |
| tree/[5,100]/[moveFromStartToEnd(1)] | 1065 (1.16) | 920 |
| tree/[500]/[kivi_worst_case] | 9565 | 10810 (1.13) |
| tree/[500]/[snabbdom_worst_case] | 3625 | 8800 (2.43) |
| tree/[500]/[react_worst_case] | 8035 (6.84) | 1175 |
| tree/[500]/[virtual_dom_worst_case] | 3750 | 7925 (2.11) |
| tree/[10,10,10,10]/no_change | 170 | 570 (3.35) |
| tree/[2,2,2,2,2,2,2,2,2,2]/no_change | 120 | 145 (1.21) |

# B Obsah kompaktního disku

K práci je přiložen kompaktní disk, který obsahuje výslednou webovou aplikaci. Na disku se také nachází zdrojové kódy a soubor README k této aplikaci.

Univerzita Hradec Králové
Faculty of Informatics and Management
Akademický rok: 2016/2017

Studijní program: Applied Informatics
Forma: Full-time
Obor/komb.: Aplikovaná informatika (ai2-p)

**Podklad pro zadání DIPLOMOVÉ práce studenta**

| PŘEDKLÁDÁ: | ADRESA | OSOBNÍ ČÍSLO |
|---|---|---|
| Bc. Šťastná Markéta | J. Jabůrkové 274, Pardubice - Trnová | I1500698 |

**TÉMA ČESKY:**

Vizuální podpora výuky předmětů zabývajících se teorií grafů a grafovými algoritmy

**TÉMA ANGLICKY:**

Visual learning support for courses dealing with graph theory and graph algorithms

**VEDOUCÍ PRÁCE:**

RNDr. Andrea Ševčíková - KIKM

**ZÁSADY PRO VYPRACOVÁNÍ:**

Cíl práce: Vytvoření aplikace pro výuku důkazů matematických vět a související výrokové logiky v předmětech zabývajících se teorií grafů.

Osnova práce:
- Úvod
- Teoretické zpracování typů důkazů
- Důkazy v teorii grafů
- Výběr technologie, ve které bude aplikace vytvořena
- Implementace aplikace
- Popis a ovládání aplikace
- Testování aplikace a návrh rozšíření aplikace do budoucna
- Závěr
- Seznam použité literatury

**SEZNAM DOPORUČENÉ LITERATURY:**

THIELE, Rüdiger. Matematické důkazy. 1. vyd. Praha: Státní nakladatelství technické literatury, 1985. Polytechnická knižnice (SNTL).

MATOUŠEK, Jiří a Jaroslav NEŠETŘIL. Kapitoly z diskrétní matematiky. V Praze: Karolinum, 2000. ISBN 978-80-246-1740-4.

MILKOVÁ, Eva. Teorie grafů a grafové algoritmy. Vyd. 1. Hradec Králové: Gaudeamus, 2013. ISBN 978-80-7435-267-6.

**Podpis studenta:** ..... **Datum:** 14. 10. 2016

**Podpis vedoucího práce:** ..... **Datum:** 14. 10. 2016