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: Bc. Tomáš Skořepa
Studijní obor: Aplikovaná informatika

Vedoucí práce: RNDr. Andrea Ševčíková

Hradec Králové srpen 2018

Prohlášení:

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

V Hradci Králové dne 16.8.2018 Bc. Tomáš Skořepa

Poděkování:

Děkuji RNDr. Andree Ševčíkové za metodické vedení, cenné rady, podněty a věnovaný čas při zpracování diplomové práce.

Anotace

Diplomová práce se zaměřuje na podporu výuky dokazování matematických vět v předmětech zabývajících se teorií grafů a grafovými algoritmy. V posloupnosti seznamuje čtenáře se základy teorie grafů, neformálními metodami dokazování a v jejich souvislosti se soustřeďuje na přínosy a možnosti využití vizualizací ve výuce.

Dále jsou uvedeny teoretické zásady tvorby moderních single page aplikací a principy funkcionálního programování, představeny novinky v JavaScriptu specifikace ES6 a porovnány výhody knihovny React s frameworky Angular a Vue.js.

Výsledkem vlastní práce autora je tvorba výukových materiálů ve formě webové single page aplikace s využitím principů funkcionálního programování, novodobé JavaScriptové syntaxe a komponentově orientované knihovny React. S podporou dalších moderních nástrojů je popsán proces tvorby interaktivních, dynamických vizualizací grafů a dalších komponent. V závěru práce jsou uvedeny podrobnosti o testování nástroji Jest a Enzyme a shrnuta zpětná vazba z uživatelského testování.

Annotation

Title: Visual support for teaching subjects dealing with Graph theory and Graph algorithms

The diploma thesis focuses on the support of teaching proving of mathematical theorems in subjects dealing with the graph theory and graph algorithms. The reader is sequentially acquainted with the fundamentals of graph theory, informal proof methods, and in sequence it concentrates on the benefits and possibilities of using visualizations in teaching.

Furthermore, the thesis introduces the theoretical principles of creating modern single page applications, principles of functional programming and innovations of JavaScript ES6 specification and compares the benefits of the React library with frameworks Angular and Vue.js.

The author's own work is the creation of teaching materials in the form of a single page web application using the principles of functional programming, the modern JavaScript syntax and the component-oriented React library. With the support of other modern tools, the process of creating interactive dynamic visualizations of graphs and other components is described. In the final chapter of the thesis, test details by Jest and Enzyme tools and a summarized feedback from user experience testing are provided.

Obsah

1 Úvod

Jako poznatek se v matematice uznává a zařazuje do teorie pouze to, co bylo dokázáno. Důkaz zaručuje platnost tvrzení a dokázané matematické věty se stávají nositeli matematických poznatků, které tak mohou být předávány následujícím generacím. V průběhu tvorby matematického důkazu je procvičováno logické uvažování a důležitost výuky dokazování se proto projevila i jejím začleněním do rámcových vzdělávacích programů. [53][6]

Ačkoli jsou studentům většinou předkládány neformální, snáze pochopitelné důkazy v přirozeném jazyce, mívají s porozuměním a především pak s vlastní konstrukcí důkazu potíže. Při tvorbě důkazu musí mít matematik dobrý přehled v dané matematické oblasti, a přestože existují ověřené metody dokazování, nelze je bezmyšlenkovitě aplikovat jako vzorec zaručeně vedoucí k výsledku. Právě při tvorbě důkazů se ukazuje, že i matematik musí být kreativní, a že dané látce důkladně rozumí. [9][30]

Zřejmě i z těchto důvodů je dokazování u studentů často látkou neoblíbenou a opomíjenou. Studenti pravdivosti předloženého tvrzení raději uvěří, než aby se o ní sami přesvědčili. Je proto vhodné hledat způsoby, jak studentům cestu k výukovým materiálům o dokazování ulehčit, důkazy zatraktivnit a jejich pochopení usnadnit. [49]

Ani oblast teorie grafů není v tomto směru výjimkou. Otázky, o kterých pojednává, je ovšem většinou možné graficky reprezentovat. Protože lidé používají zrak jako hlavní vjem pro vstřebávání informací, je možné porozumění důkazům posílit vizualizacemi. Matematické zápisy se stávají snáze pochopitelné a důkazy atraktivnější. [32][10]

Při dnešní dostupnosti počítačů a Internetu se stalo jejich využití ve výuce běžné a to jak ve školním prostředí, tak formou e-learningu. Nutnost instalovat software či stahovat soubory prezentací pomíjí se stále častějším využitím všestrannosti webových prohlížečů a moderních webových aplikací, které umožňují interaktivitu a dynamiku vizualizací na různých druzích koncových zařízení. Snad právě touto cestou lze zpřístupnit obsah studentům v přívětivější a snadno dostupné formě.

2 Cíl a metodika práce

Cílem diplomové práce je vytvoření studijních materiálů pro výuku matematických důkazů v předmětech zabývajících se teorií grafů a grafovými algoritmy.

Čtenář diplomové práce je nejprve seznámen s teoretickými základy teorie grafů a neformálních metod dokazování, přičemž je opodstatněn jejich význam v matematice.

Podstatné téma diplomové práce tvoří vizualizace a hledání způsobu jejich efektivního využití při dokazování tvrzení z oblasti teorie grafů. V kapitole 4 je proto postupně prozkoumáno jak obecné využití vizualizací ve výuce matematiky, tak využití ve spojitosti s dokazováním a grafy. Představeny jsou teoretické postupy tvorby vizualizací grafů, možné způsoby využití jejich atributů pro potřeby dokazování, a zásady, které je pro dosažení snadné srozumitelnosti grafu vhodné dodržovat. V závěru kapitoly jsou analyzovány požadavky na zpracovávanou aplikaci a jsou porovnány s aktuální nabídkou v rešerši nástrojů zabývajících se podobnou tématikou.

Kapitola 5 je věnována aktuálním trendům ve vývoji webových aplikací a technologiím použitým v praktické části. Prozkoumán je rozmach funkcionálního programování, front-endových frameworků a možnosti využití dalších nástrojů při tvorbě výsledné aplikace. V průběhu textu jsou v souvislosti se stanovenými požadavky opodstatňovány provedené volby těchto technologií.

V poslední obsahové části je představena webová single page aplikace realizovaná v praktické části a jsou popsány principy a konkrétní postupy použité při její tvorbě. Na projektu je uvedena řada příkladů použití technik z teoretické části a text je proto doprovázen ukázkami zdrojového kódu z aplikace. Tyto ukázky mohou být v některých případech zjednodušeny a zkráceny tak, aby vynikla podstata myšlenky popisované v textu, a ukázka nebyla nadbytečně dlouhá. V závěru kapitoly je připojena uživatelská příručka a jsou popsány postupy získávání zpětné vazby a testování aplikace.

Vzhledem k povaze tématu práce bylo nutné opatřit literaturu z různých oborů a zdrojů. V části věnované teorii grafů je čerpáno zejména ze skript „Teorie grafů

a grafové algoritmy“ [34] od paní profesorky Evy Milkové, jakožto předního materiálu doporučeného ke studiu studentům Fakulty informatiky a managementu Univerzity Hradec Králové. Literaturu v českém znění lze nalézt také k oblasti dokazování. Zde je čerpáno například z knihy „Matematické důkazy“ [53], jejímž autorem je Rüdiger Thiele, nebo ze skript „Matematika 1“ [40] pana docenta Pavla Pražáka.

V oblasti vizualizace ve vztahu k výuce je čerpáno z výzkumných studií a sbírek konferenčních příspěvků. Ve spojitosti s vizualizací grafů nabízí cenné poznatky sbírka „Handbook of graph drawing and visualization“ [52], ale pro jejich technickou realizaci lze čerpat i z publikací věnovaných zpracování dat jako je například kniha „Graph analysis and visualization“ [12].

V současnosti je oblast single page aplikací ve velkém rozvoji. Vzniká řada nových nástrojů, boří se mýty a zavádí nové syntaxe. Vydávané publikace nedokáží reagovat na rychlost aktuálního vývoje a představované techniky mohou být zastaralé již v momentě, kdy se dostanou do tisku. V případě populární knihovny React s početnou komunitou jsou však nové zdroje publikovány neustále a je tak možné čerpat například z knih pod názvem „Learning React“ z roku 2017 od Alexe Bankse [7] a z roku 2018 od Kirupy Chinnathambiho [25]. Hlavními internetovými zdroji jsou především oficiální dokumentace.

Z těchto důvodů jsou některé technické pojmy v práci ponechány v původním anglickém tvaru, jelikož dosud nebyla vydána žádná česká literatura, která by zavedla počeštěné ekvivalentní pojmy, nebo tyto pojmy ještě nejsou ve vývojářské komunitě dostatečně zažity.

3 Teoretická východiska

Úvodní kapitola si klade za cíl uvést čtenáře v obecné rovině do problematiky vizualizace důkazů z oblasti teorie grafů a představit základní souvislosti témat rozebíraných v diplomové práci. V této kapitole budou také vymezeny základní pojmy nutné k porozumění jak následujícím kapitolám teoretické části, tak části praktické.

3.1 Teorie grafů a diskrétní matematika

Teorie grafů je odvětvím matematiky, jehož kořeny sahají do 18. století. Za první práci je považován článek [20] Leonharda Eulera z roku 1736, ve kterém se Euler věnoval tzv. „problému sedmi mostů města Královce“. Teprve o dvě stě let později, v roce 1936 [28], však vyšla první kniha kompletně věnovaná teorii grafů. Byla jí monografie *„Theorie der endlichen und unendlichen Graphen“* maďarského matematika Dénese Königa a sloužila dlouhou dobu jako jediná učebnice. [46]

Bouřlivý rozvoj této mladší matematické disciplíny pak nastal až ve druhé polovině 20. století. V této době našly výsledky teorie grafů uplatnění nejen v řadě matematických oblastí, ale také například ve fyzice, v chemii, v elektrotechnice, v ekonomii a v lingvistice. [46]

Dnes řadíme teorii grafů pod obor Diskrétní matematiky – moderní a dynamické disciplíny, úzce související jak s ostatními matematickými obory (např. s lineární algebrou nebo analýzou), tak s teoretickou informatikou. [14]

Protože diskrétní matematika zahrnuje mnoho témat, je obtížné stanovit jednotnou definici. Příklady témat jsou výroková logika, teorie množin, kombinatorika a právě teorie grafů. Nicméně, pod diskrétní matematiku jsou zařazovány také počítačové vědy, abstraktní a lineární algebra, teorie čísel, teorie her, pravděpodobnost nebo geometrie, ačkoli některá ze jmenovaných (především poslední dvě) mají i svou nediskrétní variantu. [30]

Obecněji lze říci, že diskrétní matematika je využitelná kdykoli jsou počítány objekty, když jsou studovány vztahy mezi konečnými (nebo spočítatelnými) množinami, a když jsou analyzovány procesy zahrnující konečný počet kroků. [43]

Klíčovým důvodem pro nárůst důležitosti diskrétní matematiky je fakt, že diskrétní cestou jsou uloženy a manipulovány informace v počítačích. Za řadu impulsů z druhé poloviny 20. století proto vděčí také prudkému rozvoji počítačů a komunikačních technologií. Z těchto důvodů poskytuje matematické základy pro obory z počítačových věd zahrnujících strukturování dat, algoritmy, teorie databází, formální jazyky, počítačovou bezpečnost nebo operační systémy. [43][14]

V následujících podkapitolách budou uvedeny základní definice z teorie grafů nutné k porozumění příkladům důkazů zpracovaných v praktické části.

3.1.1 Grafy

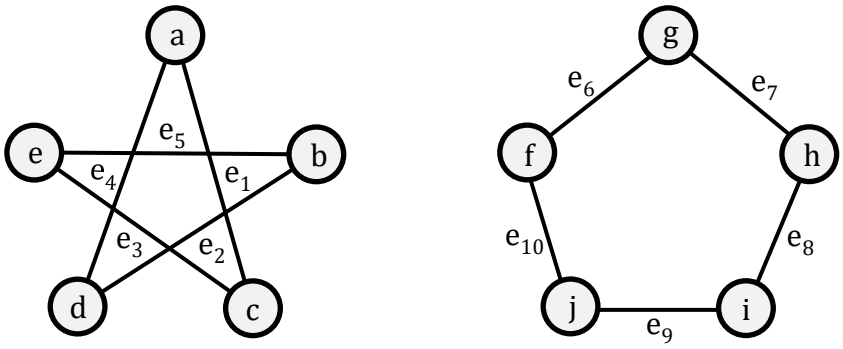
Pojem graf, v tom smyslu, jak jej chápeme dnes, použil poprvé J. J. Sylvester v roce 1878 [46] a dostal svůj název podle možnosti grafické reprezentace. Je to právě grafická reprezentace, která nám pomáhá porozumět mnoha jejich vlastnostem a lze jím vyjádřit prakticky jakékoli vztahy mezi dvojicemi prvků [10].

Blíže nespecifikovaným pojmem graf bude v této práci uvažována definice obyčejného grafu, kterou Milková [34] uvádí takto:

„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*.“*

3.1.1.1 Reprezentace grafu kreslením

Jak již bylo uvedeno v předchozím textu, výhodou grafů je možnost jejich reprezentace kreslením. V učebnicích jsou nejčastěji vrcholy zakresleny jako kroužky (případně jen body) a hrany jako čáry (úsečky, případně oblouky), které spojují příslušné dvojice vrcholů. Grafy kreslíme do roviny, a protože je často možné nakreslit je mnoha různými způsoby, volíme obvykle ten s co nejmenším počtem průsečíků. [18]



Obrázek 1 Příklady nákresů grafu [vlastní zpracování]

K předchozímu Matoušek [32] dodává, že: *„Kreslení grafů je důležitou pomůckou při studiu grafů.“*

3.1.1.2 Seznamy vrcholů a hran

Dalším způsobem, jak definovat graf, je úplný popis přesně dle definice grafu. *„Množina vrcholů je popsána prostým výčtem (seznamem) prvků, množina hran je popsána seznamem trojic tvořených jménem hrany a jejím počátečním a koncovým vrcholem,“* vysvětluje Demel [18]. Jména hran není nutné uvádět, a protože uvažujeme neorientované grafy, pořadí vrcholů a hran můžeme zvolit libovolně.

Graf hvězdice z obrázku 1 lze popsat například takto:

Vrcholy: v_1, v_2, v_3, v_4, v_5.

Hrany: (e_1, v_1, v_3), (e_4, v_1, v_4),

(e_2, v_2, v_4), (e_5, v_2, v_5).

(e_3, v_3, v_5),

Stejný graf lze pouhým pojmenováním vrcholů zadat například takto:

Vrcholy: a, b, c, d, e.

Hrany: (a, c), (a, d),

(b, d), (b, e).

(c, e),

Demel [18] dále uvádí, že: *„K výhodám tohoto popisu grafu patří univerzálnost a relativní úspornost.“* Díky tomu je tento způsob v praxi velmi často používán, ačkoli se detaily provedení mohou lišit. [18]

3.1.2 Základní a odvozené pojmy teorie grafů

Tak jako jiné matematické disciplíny, používá teorie grafů základní pojmy jako stavební kameny pro definování těch sofistikovanějších. [1]

Následují definice několika základních pojmů vztahujících se k této myšlence podle Milkové [34]:

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

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

„Cesta délky $k, k \geq 0$, v grafu G je sled délky k s navzájem různými vrcholy, tj. pro $i \neq j$ platí $v_i \neq v_j$.“

„Kružnice délky $k, k \geq 3$, v grafu G je posloupnost $(v_0, e_1, v_1, \dots, e_k, v_0)$, kde $e_i = \{v_{i-1}, v_i\}, i = 1, \dots, k-1, e_k = \{v_{k-1}, v_0\}$ a pro $i \neq j$ platí $v_i \neq v_j$.“

Se znalostí předchozích definic je možné objasnit definice vztahujících se k souvislosti grafu. Matoušek [32] uvádí, že *„graf G je souvislý, jestliže pro každé dva jeho vrcholy x a y v něm existuje cesta z x do y.“*

Nesouvislý graf je složen z několika souvislých částí, nazývaných komponenty [1]. *„Komponenta grafu je každý jeho maximální souvislý podgraf, tj. souvislý podgraf, který není podgrafem žádného jiného souvislého podgrafu.“* [34]

Speciálními názvy označujeme vrcholy a hrany, jejichž odebráním vznikne v grafu G více komponent.

„Vrchol v je artikulace grafu G, jestliže graf $G - v$ má více komponent než graf G.“ [34]

„Hrana e je most grafu G, jestliže graf $G - e$ má více komponent než graf G.“ [34]

Pojmy uvedené v kapitole věnované teorii grafů budou v praktické části práce hojně využívány. Kromě nich, lze definice dalších pojmů najít například v české literatuře od Demela [18], Matouška [32] nebo Milkové [34].

3.2 Důkazy

Důkaz je platný argument, který potvrzuje pravdivost matematického tvrzení, a jako poznatek se v matematice se uznává a zařazuje do teorie pouze to, co bylo dokázáno. [53]

Tvorba důkazů ukazuje, že i matematikové musí být kreativní, jelikož při dokazování není úspěch zvolené cesty nikdy zaručen. Odhodlání přesvědčit svět, že dané tvrzení je pravdivé, tak bývá často velkou výzvou. [30]

Je také nutné si uvědomit, že ačkoli nalezneme postupy k zodpovězení položených otázek o pravdivosti výroku, je ještě nutné zvážit, jak danou odpověď prezentovat pokud možno v co nejsrozumitelnější a stručné podobě. [9]

3.2.1 Základní terminologie

Formálně je matematický výrok, u kterého lze dokázat jeho pravdivost, nazýván *věta* avšak méně významné věty bývají označovány jako *tvrzení*. Pravdivost matematické věty demonstruje právě *důkaz*. [43]

K důkazům Pražák [40] uvádí: *„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“*

Jen v málo případech je však správnost tvrzení bezprostředně patrná, a tak se běžně důkaz rozkládá na více kroků, jejichž správnost již bezprostředně patrná je. Na základě korektních logických pravidel se v jednotlivých krocích důkazu vyvozují závěry, přičemž na konci takového řetězce je dokázané tvrzení přijato do teorie jako věta [53]. V praxi je posledním krokem obvykle prosté shrnutí dokázané věty. Pro srozumitelnost je však někdy navíc rekapitulován původní výrok [43]. Užitečné je také označení konce důkazu symbolem čtverce □ [9].

Všechna předešlá tvrzení, na kterých je výsledná věta vystavěna, by tedy ideálně mělo být možné dokázat. Jak uvádí Thiele [53]: *„To však nelze uskutečnit, neboť abychom dokázali nějakou větu, používáme jako předpokladů vět již dokázaných. K důkazů těchto vět však opět potřebujeme dokázané věty a tak dále.“*

Kromě dokázaných tvrzení se proto k dokázání nových vět používají ještě tzv. *axiómy*. Axiómy Thiele [53] definuje jako *„bezprostředně zřejmá tvrzení, která není třeba dokazovat a ani je dokázat nelze“*. Jsou v nich vyjádřeny základní vlastnosti objektů a základní vztahy mezi těmito objekty. [53]

Axiomy neoznačujeme nutně za pravdivé, abychom z nich však mohli vycházet při určování platnosti dalších výroků, považujeme je pro danou matematiku za platné [15]. Čech [15] proto upozorňuje, že: *“Platnost určité matematické věty je závislá na zvolených axiómech.“*

3.2.2 Význam dokazování

Matematické věty jsou skutečnými nositeli matematických poznatků a naši předchůdci nám pomocí vět předali mnoho svých vědomostí. Obdobně předáváme získané poznatky také dnes. [15]

Thiele uvádí ve své knize [53] citát A. Einsteina: *„Matematika požívá oproti všem jiným vědám mimořádné vážnosti; její věty jsou absolutně jisté a nepopiratelné, zatímco ve všech ostatních vědách jsou důkazy do jisté míry sporné a jsou vždy vystaveny nebezpečí, že nově odhalené skutečnosti je vyvrátí.“* Matematické důkazy nejsou založeny na názorech nebo zkušenostech, které lze zlepšovat nebo měnit. Naopak, jak zdůrazňuje Čech [15]: *„Dbá se na to, aby odvozování bylo skutečně logické, jinak by nebyla zaručena použitelnost vzniklé matematické teorie ve všech vědách, ve kterých platí axiómy.“* Obvykle se navíc snažíme o důkaz v obecném případě tak, aby byl výsledek platný pro všechny případy dané domény [43]. Protože si učitelé, odborníci na vzdělávání i vládní činitelé důležitost dokazování a procvičování logického uvažování uvědomují, jsou vybrané techniky zakotveny již v Rámcovém vzdělávacím programu pro gymnázia [6][49].

Kromě dokazování matematických vět, nacházejí důkazy využití v mnoha aplikacích do počítačových věd. Tyto aplikace zahrnují například ověřování bezchybnosti

počítačového programu, zhodnocení zabezpečenosti operačního systému nebo dedukci v umělé inteligenci. Proto je porozumění technikám dokazování nezbytné nejen v matematice, ale také v počítačových vědách. [43]

Kdykoli student prochází důkaz v učebnici, měl by se ujistit, že rozumí tomu, co každý řádek vyjadřuje, a chápe, proč je považován za pravdivý. Stejně tak je důležité, aby rozuměl celkové struktuře důkazu. [30]

Nicméně, formální důkazy užitečných vět mohou být extrémně dlouhé a těžko srozumitelné. V praxi jsou proto často čtenářům předkládány důkazy neformální, kde sice může být v jednom kroku použito více deduktivních pravidel či mohou být některé kroky zcela přeskočeny, ale postup důkazu je pro ně srozumitelnější než třeba formální důkaz vyprodukovaný automatizovanou dedukcí znalostního systému. Z těchto důvodů se někdy v důkazech objevují slova jako „očividně“, „nepochybně“ nebo spojení jako „je zřejmé“, kdy autor očekává, že čtenář bude schopný domyslet si vynechané kroky. [43]

Přesto má mnoho studentů s pochopením a především pak s vlastní konstrukcí matematických důkazů potíže [9]. Naštěstí, existuje pouze několik standardních metod dokazování, které se objevují znovu a znovu. Tyto metody objasňují celkový přístup a strategie k dokazování a jejich znalost může k pochopení a samostatné konstrukci důkazů značně pomoci [30].

V následujících podkapitolách jsou krátce představeny základní metody dokazování, se kterými se student setká nejen ve studiu vysokoškolských matematických předmětů.

3.2.3 Důkaz přímo

„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.“ [53]

Z logické perspektivy se jedná o nejjednodušší metodu dokazování. Vše, co důkaz vyžaduje, je často jen pouhé vysvětlení provedených kroků. Přímé důkazy jsou zvláště užitečné při dokazování implikací. [30]

Dokazujeme-li platnost implikace $p \Rightarrow q$ sestrojíme řetězec platných implikací

$$p \Rightarrow q_1 \Rightarrow q_2 \dots q_n \Rightarrow q$$

Na základě tzv. hypotetického sylogismu uzavřeme, že platí původní implikace $p \Rightarrow q$. [40]

Máme-li dokázat platnost výroku v, vyhledáme pravdivý výrok p a stejně jako u důkazu implikace dokážeme, že platí $p \Rightarrow v$. Z platnosti výroku p a implikace $p \Rightarrow v$ usoudíme na základě tzv. oddělovacího pravidla, že platí také výrok v. [40]

Často se zdá, že přímý důkaz byl konstruován přímočaře očividnou sekvencí kroků a vedl od hypotézy snadno až k závěru. Přímé dokazování však někdy vyžaduje hluboký vhled do problematiky a může tak být v některých případech ve výsledku obtížné. [43]

3.2.4 Důkaz nepřímo

Existuje mnoho výroků, které je obtížné dokázat přímou metodou, zato dokazování nepřímo může být snadné. Z výrokového počtu platí, že implikace $p \Rightarrow q$ je logicky ekvivalentní k výroku $\neg q \Rightarrow \neg p$. A přesně z tohoto poznatku metoda dokazování nepřímo vychází. Snažíme se přímo dokázat negaci původního výroku, což je z důvodu ekvivalence s původním výrokem dostačující. [30]

Dokazujeme-li tedy platnost implikace $p \Rightarrow q$, sestavíme řetězec platných implikací

$$\neg q \Rightarrow q_1, q_1 \Rightarrow q_2, \dots q_n \Rightarrow \neg p$$

a opět uzavřeme, že platí $\neg q \Rightarrow \neg p$, tedy výrok ekvivalentní s původním $p \Rightarrow q$. [40]

3.2.5 Důkaz sporem

Předpokládejme, že chceme dokázat pravdivost výroku p. Navíc, jsme schopni nalézt nepravdivý výrok q takový, že implikace $\neg p \Rightarrow q$ je pravdivý výrok. Pokud ovšem víme, že q je nepravdivý výrok a přitom implikace $\neg p \Rightarrow q$ je pravdivý výrok, znamená to, že $\neg p$ je výrok nepravdivý a tedy p je výrok pravdivý. Takto stavěné důkazy jsou nazývány důkazy sporem. Protože důkaz sporem nedokazuje výrok p přímo, je někdy označován jen jako druh nepřímého důkazu. [43]

Pokud jsme tedy schopni dokázat, že výrok $\neg p$ vede ke sporu, je jediným závěrem, že se jednalo o výrok nepravdivý, tudíž výrok p je pravdivý, a to jsme původně chtěli dokázat. Jinými slovy, pokud není možné, aby p, byl nepravdivý výrok, pak je to výrok pravdivý. [30]

3.2.6 Protipříklady

Téměř nikdy není správné dokazovat pravdivost výroku pouhý uvedením platného příkladu. Důkazy se většinou snažíme konstruovat platné pro obecnou množinu případů a nalezení jediného příkladu nemusí nic znamenat. Avšak existenciální výroky tímto způsobem dokázat lze. Chceme-li dokázat, že existuje případ, kdy výrok p není pravdivý, vše co nám stačí je takový případ nalézt. [30]

Nicméně, jak uvádí Thiele [53]: *„Existuje celá řada slavných domněnek, pro které dodnes neznáme důkaz, které se však ani nepodařilo vyvrátit protipříkladem.“* Ani nalézt protipříklad tedy není vždy snadné.

3.2.7 Důkaz matematickou indukcí

Za pokročilou, ale důležitou důkazovou metodu diskrétní matematiky je považována metoda matematické indukce, která umožňuje pohodlně dokazovat i složitá tvrzení po jednotlivých (diskrétních) krocích. Je používána například pro ověřování správnosti některých typů počítačových programů, a coby iterace cyklů je jim svou podstatou blízká. Lze ji proto považovat za důležitou zejména pro informatiky. [23][43]

Definici pro tzv. slabý princip matematické indukce uvádí Habala [22] následovně:

Nechť $n_0 \in \mathbb{Z}$, nechť $V(n)$ je vlastnost celých čísel, která má smysl pro $n \geq n_0$.

Předpokládejme, že jsou splněny následující předpoklady:

(0) $V(n_0)$ platí.

(1) Pro každé $n \in \mathbb{Z}, n \geq n_0$ je pravdivá následující implikace: Jestliže platí $V(n)$, pak platí i $V(n+1)$.

Potom $V(n)$ platí pro všechna $n \in \mathbb{Z}, n \geq n_0$.

Matematická indukce se obvykle používá, chceme-li dokázat, že nějaká vlastnost platí pro všechna přirozená čísla. Nejprve ukážeme, že vlastnost $V(n)$ platí pro nejmenší číslo n_0 (většinou $n_0 = 1$), a následně, že jestliže vlastnost $V(n)$ platí pro všechna $n \geq n_0$ (nazýváno indukční předpoklad), pak platí také $V(n + 1)$. [8]

Metod dokazování existuje mnohem více. Z těch, které zde nebyly představeny lze jmenovat například metodu zpětného úsudku, důkaz rozborem případů nebo důkaz myšlenou konstrukcí. Informace o těchto i dalších metodách uvádí například Thiele [53], Rosen [43] nebo Levin [30].

3.3 Vizualizace

Vizualizaci lze definovat jako předávání informací využitím grafické reprezentace. Obrázky byly využívány jako mechanismus komunikace dávno před formalizováním zápisu jazyků. Jeden obrázek může zahrnovat spoustu informací, jejichž význam lze vstřebat mnohem rychleji v porovnání se stejnými informacemi sepsanými v textu jedné stránky. Je to tím, že interpretace obrazu je prováděna paralelně v rámci lidského vjemového systému, zatímco rychlost textové analýzy je limitována sekvenčním procesem čtení. Obrázky mohou být také nezávislé na jazyce, a jejich sdělení tedy mohou porozumět i lidé znalí jen rozdílných jazyků. Důvodů, proč je vizualizace důležitá, je mnoho. Jedním z nejzřejmějších je, že lidé využívají zrak jako hlavní smysl pro vstřebávání informací a jejich porozumění. [55]

Z hlediska výuky se ukazuje, že vizualizace je účinnou metodou, kterou lze posílit pochopení konceptů v mnoha oborech, jako je informatika, chemie, fyzika, biologie, inženýrství, aplikovaná statistika a matematika. Především ve výuce matematiky bylo nalezeno mnoho důvodů, které opodstatňují využití vizualizace na všech úrovních školství, od škol základních přes střední až po univerzity. [42]

Literatura také naznačuje, že schopnost uvědomit si, co daný obrázek představuje, není vrozenou a samozřejmou, ale něco, čemu se lze učit a osvojit si. Jak upozorňují kognitivní vědy, vidět a utvářet si představu o tom co vidíme, se učíme. A pokud vizuální uvažování je dovednost, které se lze učit, je možné ji také vyučovat. [56]

Tím pádem učitelé, kteří dovedou principy vizualizace začlenit do vyučování, mohou posílit vysvětlení matematických konceptům a ještě více tak podpořit studenty v procesu učení. Proto studie z kognitivních věd doporučují široké využití vizualizací v různých směrech matematikům a obecně všem lidem, se zájmem o matematiku. [13]

Teorie grafů v tomto směru není výjimkou. Jak uvádí Matoušek [32]: *„Většina teorie grafů pojednává o otázkách, které se dají geometricky znázornit.“* A tudíž i v případě mezi studenty ne příliš oblíbeného dokazování bude vhodné vizualizace využít a co nejvíce jim tak usnadnit cestu ve studiu k jejich pochopení.

4 Možnosti a význam vizualizace ve výuce

4.1 Vztah vizualizace a počítačové grafiky

Současné vizualizace poskytují vizuální reprezentaci objektů zahrnujících například data, algoritmy, výsledky výpočtů, procesy, uživatelské ovládací prvky a mnohé další komponenty aplikace. Tyto vizuální reprezentace jsou zobrazovány využitím počítačově generované grafiky. V interaktivních vizualizacích mohou uživatelé dokonce interagovat přímo se zobrazením raději než přes menu aplikace. [55]

Právě protože vizualizace využívá grafiky k zobrazení informací, byla původně považována za podobor počítačové grafiky. Ve všech vizualizacích lze pozorovat využití grafických primitiv, jako jsou body, čáry nebo barevné plochy. Oproti využití grafiky, je však důležitým aspektem vizualizací jejich souvislost s reprezentací informací. Právě z nich vizualizace vychází a využívá k tomu kromě počítačové grafiky i dalších disciplín jako je psychologie vnímavosti nebo interakce člověka s počítačem. Zatímco snahou počítačové grafiky je především realistické ztvárnění tří-dimenzionálního světa, vizualizace oproti realističnosti upřednostňuje efektivnost komunikace informace. [55]

4.2 Vizualizace ve výuce matematiky

Historie vizualizace ve výuce matematiky je velmi dlouhá. Především od 80. let 20. století se matematici zajímali o její praktické využití, jako prostředku ve školní výuce a v rámci pedagogické psychologie začali zkoumat teoretické zásady a tak pokračují i dnes. Článek [19] Theodora Eisenberga „On the understanding the reluktance to visualize“ a mnohé další nedávné studie se shodují, že úspěch každého matematika závisí v nezanedbatelné míře na jeho schopnosti vizualizovat. Přesto od dob Leonharda Eulera a úspěchů využití vizualizace k důkazům mnoha matematických vět nastalo mezi 19. a 20. století období, kdy matematici omezili využití vizualizace, aby při řešení matematických problémů nabyli nových myšlenek. Zřejmě právě mezera v tomto období je v současnosti podnětem k novým výzkumům. [27]

Mnoho výzkumů, experimentů a vědeckých studií z oblasti výuky matematiky dokazuje, že role vizuální reprezentace je zásadní při řešení slovně zadaných problémů. Vizuální reprezentace často pomáhá pochopit problém a přispívá nejen k porozumění, ale také k lepšímu matematickému uvažování. [17]

Debrenti [17] představil studii realizovanou na univerzitních studentech, kteří byli požádáni o vyřešení logického problému bez časového limitu. Část studentů mohla k řešení využít předpřipravené karty vizuálně reprezentující objekty v zadání, další část počítačový program a zbylí studenti měli k dispozici pouze tužku a papír. Problém se podařil vyřešit 100% studentům z první skupiny s kartami, 64% studentům s počítači a 62,5% studentů ze třetí skupiny. Navíc se ukázalo, že motivace vyřešit zadání vydržela studentům z první skupiny s dostupnou vizuální reprezentací déle. Byli trpělivější oproti ostatním dvěma skupinám studentů, kteří měli tendenci vzdát úkol dříve. [17]

Tvrzení, že vizualizace může být užitečným nástrojem při řešení problému, podporuje také práce Röskenové [44], kde se mimo jiné ukázalo, že některým studentům postačí k řešení vytvoření vizuální reprezentace v jejich mysli a nepotřebují ji realizovat fyzicky na papír. U vizualizací ve výuce nicméně upozorňuje na jejich možnou mnohoznačnost. Zatímco zkušený matematik dokáže správně interpretovat myšlenku, kterou vizualizace reprezentuje, student s tím může mít potíže. Je proto důležité, aby vizualizaci doprovázel bližší popis a zamezilo se tak její dezinterpretaci. [44]

4.3 Vizualizace důkazů v matematice

Ve výzkumu matematických důkazů je samotné dokázání tvrzení až konečnou fází celého procesu. Ještě před dokázáním tvrzení, musí vzniknout myšlenka o nové větě, kterou by stálo za to dokázat nebo o tvrzení, které by mohlo být pravdivé. Tato průzkumná fáze matematického myšlení profituje z dobře ucelené představy o matematických vztazích a k získání takové představy může dopomoci vizualizace. [51]

Pojem „vizuální důkaz“ je někdy míněn v kontextu jeho speciálního odvětví, ve kterém je důkaz tvrzení patrný přímo z obrázku.

Jak uvádí Štrausová [48]: *„Vizuálním důkazům se také někdy říká „důkazy beze slov“, při jejichž využití ve výuce je ale důležité s žáky nad těmito důkazy diskutovat.“* Diskuze je nutná, jelikož obrázek znázorňuje většinou pouze výsledek. Student tudíž musí sám nalézt posloupnost myšlenek, která vede k samotnému důkazu a to může být v některých případech značně složité. *„Jednou z možností je, rozložit vizuální důkaz do více obrázků a tím ulehčit pochopení myšlenky důkazu,“* navrhuje dále Štrausová [48]. Sérii obrázků je ještě vhodnější zpracovat do animace, kde pozorovatel v každém kroku snadno vidí, co se změnilo [52]. Tento typ důkazů pak zpravidla bývá pro studenty více atraktivní než klasické algebraické důkazy [49].

Oproti klasickým důkazům, které odvozují pomocí logické posloupnosti výroků, tak využívají k dokázání pravdivosti tvrzení odlišný přístup. Přirozeně se tyto důkazy týkají především geometrie, jelikož ta se zabývá operacemi s čísly v prostoru. Jedním z nejznámějších příkladů je pak vizuální důkaz Pythagorovy věty. Výskyt těchto tzv. „důkazů beze slov“ lze tedy vysledovat až do dávné minulosti před naším letopočtem. [35]

V dnešní době jsou nejznámějšími publikacemi knihy Rogera Nelsena [36][37], které jsou kolekcemi vizuálních důkazů z různých oblastí matematiky. [49]

V každém z těchto případů poskytuje vizualizace náhradu textové či verbální informace. Je zřejmé, že vizualizace mohou vystihnout předávanou informaci mnohem šířeji než slovně založený popis a často také mohou poskytnout odlišný pohled. [55]

Ačkoli jsou matematické důkazy považovány za důležitou součást budování systému matematických znalostí, jsou žáky mnohdy považovány za zbytečné, velmi složité a jedná se tak mnohdy o neoblíbenou látku. O to větší smysl má hledání cest, jak jim dodat motivaci a usnadnit pochopení důkazu. K tomu Štrausová [48] navrhuje, že: *„Jednou z možností může být využití větší pestrosti při volbě typu důkazu, nebo využití některého z matematických softwarů.“*

4.4 Vizualizace grafů ve výuce

Jak bylo uvedeno v předchozí části práce, také kresby abstraktních grafů se začaly objevovat již před několika staletími. Nyní jsou samozřejmou součástí učebnic týkajících se matematiky nebo počítačových věd, ať už popisují koncepty vztahující se ke grafům nebo grafovým algoritmům. Každá z učebnic tykající se teorie grafů, diskrétní matematiky nebo datových struktur bude takových ilustrací obsahovat mnoho. Kresby grafů se také používají k ilustrování grafově strukturovaných informací, jako jsou topologie počítačových sítí nebo vývojové diagramy popisující běh programu. [52]

Zavádění počítačů do výuky vedlo k novým aplikacím kreslení grafů například v animaci algoritmů, simulaci algoritmů nebo vizualizaci softwaru. V hodině může vyučující využít předpřipravených animací jako demonstrace, aby žákům pomohli žákům pochopit nový koncept. Animace využité při vyučování je poté vhodné studentům zpřístupnit, aby mohli průběh zastavit, krokovat či zopakovat a vstřebávat tak nový materiál vlastním tempem. Animace na grafech poté přirozeně využívají kreslení grafu a začleňují změnu barev nebo jiných vizuálních efektů k demonstraci průběhu algoritmu. Obdobně lze využít animací na stromech k demonstraci manipulací v datových strukturách jako je vložení či odebrání prvku. [52]

4.5 Postupy vizualizace grafů

Grafové vizualizace vynikají především v charakterizování komplexních směsí vztahů, které nelze vystihnout černobílými termíny. Vizualizace, které jsou zpracovány vkusně, mohou být velmi intuitivní. Jsou například přirozenou volbou pro zobrazování sítí, ale mnoho nového do vizualizace grafů přineslo jejich využití v datové analýze. [12]

Grafy se stovkami až stovkami tisíc hran mohou být zpracovány na lokálním počítači, zatímco grafy s miliony až miliardami hran vyžadují zpracování spoluprací několika počítačů. Grafy s tisíci hran mohou být vizualizovány přímo, zatímco větší grafy vyžadují strategii nebo rozhraní pro zvolení výběru k zobrazení či filtrování na menší podgraf. [12]

4.5.1 Obecenstvo vizualizace a její interpretace

Tvoříme-li vizualizaci, jejímž účelem je podat vysvětlení nějakého konceptu nebo dat, je důležité poznat naše obecenstvo. Ať už jsou naším obecenstvem specialisté konkrétního oboru, studenti nebo široká veřejnost, je vhodné se vcítit do jejich role a uzpůsobit vizualizaci jejich potřebám. Kromě identity je dobré znát například jejich motivaci ke studiu dané vizualizace, jazyk a znalosti nebo terminologii z oboru, kterým se vizualizace zabývá. Dále je nutné provést určitá rozhodnutí například o tom, které informace ponechat, a které jsou naopak irelevantní nebo by mohly odvádět pozornost od účelu vizualizace. [26]

Do vizualizace jako komunikačního média je snaha zaznamenat informace tak, aby čtenář byl schopen poselství správně interpretovat. Proto je nutné, aby tvůrce vizualizovaným informacím dobře rozuměl, a byl schopen odhadnout pohled čtenáře tak, aby nedošlo k jejich dezinterpretaci. Tvůrce vizualizace je často zatížen přemírou vědomostí, které mu umožňují správně si domýšlet veškeré souvislosti, protože ví přesně, co chce vizualizací vyjádřit. Musí však brát v úvahu vědomosti čtenáře, který vizualizaci vidí poprvé, a pokoušet se podat vysvětlení co nejjednodušeji. Veškerá tato uzpůsobení mohou ve výsledku kladně přispět k lepšímu porozumění vizualizaci. [26]

Aplikace implementovaná v praktické části bude uvažovat jako primárního uživatele studenta předmětů zabývajících se teorií grafů, který zná definice základních pojmů a věty uvedené v literatuře doporučené ke studiu.

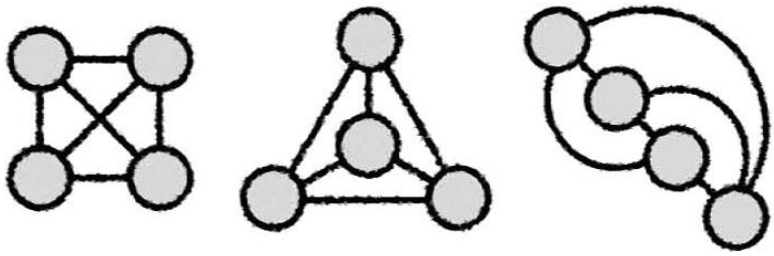
4.5.2 Definování cíle vizualizace

Pro to, aby vizualizace splnila svůj účel je vhodné před započetím implementace definovat její cíl. Tento cíl je poté brán v úvahu při celém procesu implementace vizualizace. Cíl vizualizace je obvykle ovlivňován záměrem autora a potřebami čtenáře. Nejčastějším cílem bývá poskytnout čtenáři specifickou informaci, ale cílem může být také změnit jeho pohled na rozebíraný kontext. Definice cíle vizualizace by neměla zahrnovat reference na žádný specifický obsah nebo detaily implementace. Naopak, měla by být dostatečně obecná, aby vyjadřovala záměr a priority, ale nebránila případné kreativitě při konkrétní realizaci cíle. [26]

4.5.3 Uspořádání grafu

Grafy vrcholů a hran jsou téměř výhradně kresleny do 2D prostoru. S tří dimenzionálním zobrazením se lze setkat jen velmi zřídka. Důležitým aspektem těchto grafů je, že konkrétní souřadnice umístění vrcholů nevypovídají nic o jejich o jejich vlastnostech. Hlavním záměrem rozmístění vrcholů je dosažení co nejlepší přehlednosti. [29]

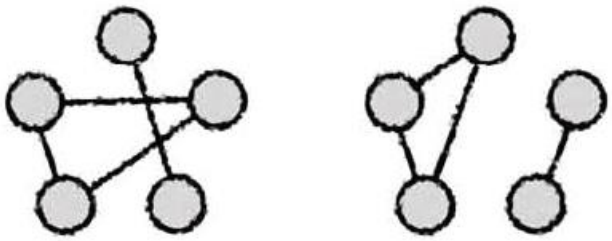
Volba uspořádání grafu je důležitá, jelikož může také ovlivnit, jak jej vnímáme. Na obrázku 3 lze vidět tři vzájemně izomorfní grafy se čtyřmi vrcholy a šesti hranami. V levém grafu jsou vrcholy rozmístěny do čtverce, ale hrany se kříží. V prostředním grafu se žádné hrany nekříží, avšak vrchol umístěný uprostřed se jeví jako odlišný od zbylých a mohl by vést k chybnému mínění, že je vyššího stupně. Graf vpravo využívá zakřivených hran a má vrcholy umístěny v lajně, kde dva jsou ve středu a dva na koncích. [12]



Obrázek 2 Vzájemně izomorfní grafy, které se vinou uspořádání jeví jako odlišné [12]

Pouhá volba uspořádání může svádět k mínění, že se jedná o odlišné grafy.

Při vizualizaci grafu s více komponentami je důležité zvolit uspořádání tak, aby byly komponenty viditelně separovány. Křížení hran ztěžuje odlišení částí komponent. [12]



Obrázek 3 Komponenty je jednodušší odlišit, pokud se nepřekrývají [12]

4.5.4 Vizuální atributy grafů

Po uspořádání grafu je v dalších krocích možné aplikovat vizuální atributy, kterými lze v grafu zachytit další informace. Příkladem atributů, které lidské oko vnímá, jsou například různé barvy a názvy vrcholů a hran, odlišná šířka hran nebo velikost vrcholů. [12]

Dostupnost těchto druhů vizuálních atributů v softwarových balíčcích je různá. V některých lze parametry nastavit jen velmi omezeně jiné jsou maximálně flexibilní. Ve většině z nich lze nalézt místo, kde je možné nastavit vizuální atributy globálně aplikované na celý graf. Aplikace na konkrétní prvky grafu bývá komplikovanější. [12]

4.5.4.1 Klíčové atributy vrcholů

Výrazným visuálním indikátorem dodatečných údajů o vrcholech je barva jejich výplně. Zavedením určité sémantiky barev můžeme nejen předat další informace, ale také usnadnit orientaci v celém grafu. Jsou-li k reprezentaci vrcholů použity kroužky, je možné upravovat také barvu a tloušťku jejich obrysů. [12]

Barvy vrcholů budou v důkazech v praktické části využity pro vyznačení prvků z teorie grafů, jako jsou tahy, cesty, kružnice, ale také například k zvýraznění odebírané artikulace.

Změnu velikosti vrcholu je možné využít pro poukázání na stupeň vrcholu, kdy vrcholy s vyššími stupni budou zastoupeny většími kroužky [12]. Nicméně, na rozdíl od datové analýzy se tento atribut se pro potřeby naznačení prvků z teorie grafů běžně nevyužívá a vrcholy v celém grafu mívají stejnou velikost.

Jedním z nejdůležitějších obohaceních grafů jsou nápisy, které jednoznačně identifikují konkrétní vrchol. Problémy mohou vyvstat s umístěním a čitelností nápisů. Obvykle jsou nápisy umístěny uvnitř vrcholů tak, aby nápis nepřečníval přes okraje vrcholu. Pak ovšem délka nápisu a velikost písma ovlivňují také průměr kruhu vrcholu. Vhodné je proto použití úzkých typů písma, jejichž mezery mezi písmeny jsou zúžené, nebo zvolit umístění nápisu mimo vrchol tak, aby velikost písma zachovávala čitelnost nápisu, ale neovlivňovala negativně

velikost vrcholu. Další podmínkou zajištění dobré čitelnosti je volba dostatečně kontrastní barvy nápisu vůči barvě výplně vrcholu. [12]

4.5.4.2 Klíčové atributy hran

Obdobně jako u vrcholů bývá možné měnit barvu čáry reprezentující hranu a přidat k ní asociovaný nápis. Navíc lze ovlivňovat šířku čáry, zakřivení a především její styl. [12]

V určitých krocích dokazování je vhodné využít přerušovanou čáru, například k vyznačení hrany odebírané v následujícím kroku. Šipky jsou v neorientovaných grafech užitečné například pro zachycení konstrukce cest a zakřivení čar při vícenásobném použití stejné hrany při konstrukci tahu.

Kombinací atributů vrcholů a hran můžeme předat dodatečné informace, stejně jako zvýšit přehlednost a pochopitelnost grafu. Zvolení správných designových postupů je proto klíčem ke zpracování vizualizace, která úspěšně splní své cíle. Při její tvorbě je však nutné mít na vědomí, že efektivní a pochopitelné vyjádření informací je důležitější než na pohled hezká vizualizace. Vizuální aspekt má za účel podpořit smysl vizualizace, ne od něj vyrušovat. [26]

Při využívání vizuálních atributů je také nutné být obezřetný, aby výsledný obraz grafu nebyl příliš komplexní a nebylo obtížné jej interpretovat. Účinnou metodou je poskytnutí legendy nebo popisu, který jasně udává, co daná vizualizace grafu vyjadřuje. [12]

4.5.5 Interaktivní vizualizace grafů

Interaktivní funkce zahrnují práci s kamerou jako je přibližování a oddalování grafu nebo její přesun v různých směrech a umožňují zkoumat graf ve větším detailu. Běžné je k tomuto účelu využití tahu myší k přesunu a otáčení kolečka k přibližování a oddalování, kdy kurzor myši slouží jako středový bod, ale možné je také použít klávesnici či v uživatelském rozhraní připravit ovládací tlačítka. V praktické části bude využito kombinace těchto možností.

Změna měřítka přiblížením či oddálením může být užitečná, pokud se představuje pouze část grafu nebo k efektivnímu využití celého prostoru grafu bez nutnosti měnit pozici a velikost vrcholů s šířkou hran. [12]

Uživatelsky přirozené je zvýraznění vrcholu a hrany po najetí myší nebo možnost jejich přesunu tahem s přidrženým tlačítkem myši. Pokročilou funkcí je kombinace přidržení klávesy (například CTRL) a označování podgrafu stisknutím tlačítka myši. To lze využít nejen k přesunu celého podgrafu, ale ve školní hodině především k jeho jednorázovému vyznačení pro potřeby dodatečného vysvětlení teorie.

Pod interakce lze dále zařadit možnosti úpravy prvků grafu přes uživatelské rozhraní [12]. Takové úpravy mohou zahrnovat změnu velikosti, barvy nebo nápisu vrcholu, ale také jeho úplné odstranění. V tom případě je nutné automaticky odstranit také hrany, které jsou k vrcholu připojené.

Žádné z těchto druhů interakcí by nebylo možné využít, pokud by graf byl reprezentován obrázkem například v PDF souboru nebo PowerPointové prezentaci. Přitom právě tyto funkce snadno nacházejí využití ve výuce.

4.6 Využití vizualizačního softwaru ve výuce

Digitální zařízení, které stále více rozšiřují možnosti interakce mezi člověkem a technologií a přinášejí nové prvky, které mohou pomoci studentům při vzdělávání. Otázkou zůstává způsob jejich uplatnění. Studenti mohou využít jejich benefitů mimo vyučování nebo je třeba hledat způsoby, jak tato zařízení začlenit do vyučování. [11]

Ilustrace a diagramy hrají v matematice důležitou roli a stejně tak ve výuce matematiky. Nicméně grafická reprezentace v učebnicích matematiky je vždy statická. A proto umožňuje ilustrovat pouze konkrétní příklad nebo omezený počet příkladů. S použitím počítačového softwaru k vizualizaci matematických principů, teoreticky není omezení v počtu demonstrací konkrétních příkladů. Momentálně existuje několik softwarových balíčků použitelný pro vizualizaci matematiky při výzkumech, ale také několik speciálně vyvinutých pro výuku matematiky. [16]

Také z těchto důvodů Palais [39] usuzuje, že vysoká interaktivita vyobrazení produkovaných prostřednictvím moderního matematického vizualizačního softwaru, umožňuje matematikům realizovat experimenty snadněji než kdy dříve. Povzbuzuje tak matematiky k využívání matematických vizualizací stejně tak jako k začlenění počítačové grafiky do výuky matematiky.

Když Daugherty [16] analyzoval limity obou typů vizualizačního softwaru matematiky, dospěl k závěru, že hlavními důvody zamezujícími jejich většímu rozšíření jsou přílišná komplexita, nedostatečná flexibilita a vysoké pořizovací náklady, přičemž se ukázalo, že komplexita a flexibilita spolu vzájemně souvisí. Vysoce flexibilní software jako je MATLAB nebo Mathematica jsou sice flexibilní, ale zároveň značně komplexní. Jednodušší software je zase zpravidla méně flexibilní.

4.7 Analýza zpracovávané aplikace

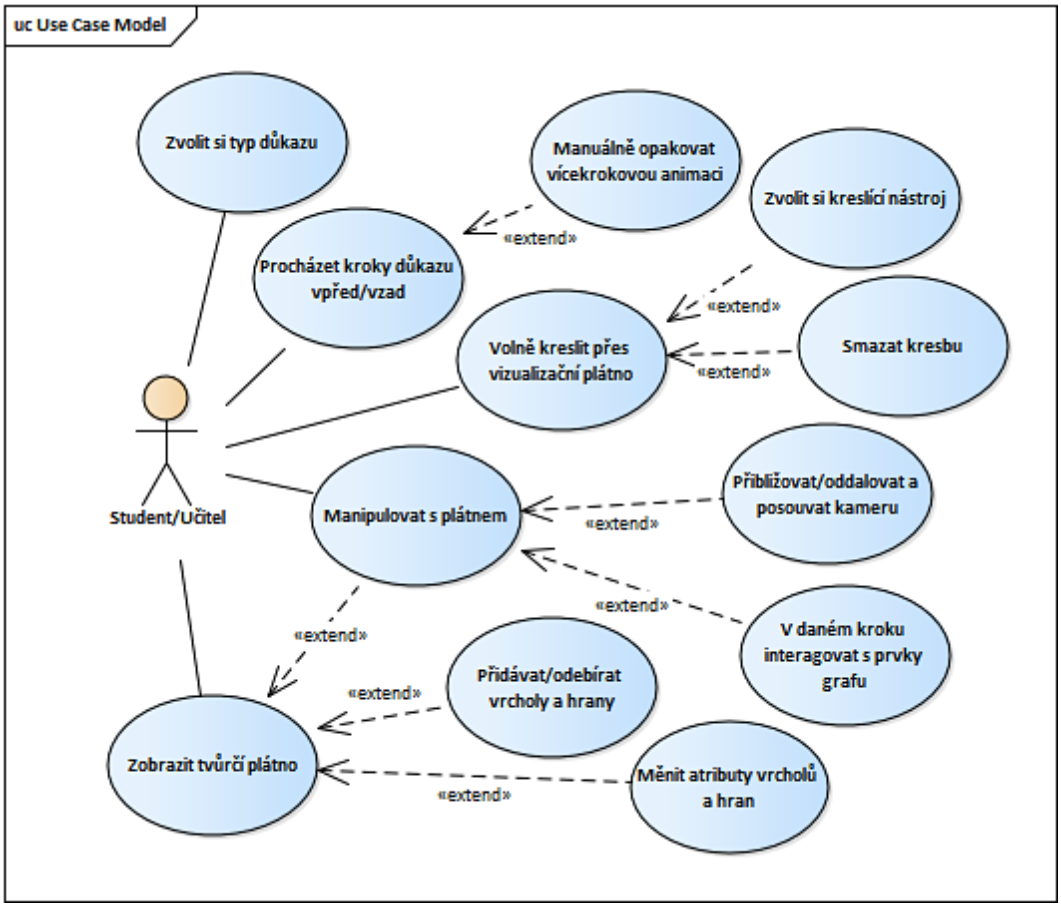
Při vývoji každé aplikace je dobrým zvykem postupovat podle jedné ze standardních metodik procesu vývoje softwaru. V jejich rámci dojde na podrobnou analýzu cílů, které má aplikace naplňovat, požadavků, které přitom má splňovat, což zvyšuje šance na úspěch výsledného softwaru. Jednou z takových metodik je například metodika Unified Process začleňující jazyk UML (Unified Modeling Language), což je přístup běžně vyučovaný nejen na fakultě FIM UHK.

Metodika Unified Process začleňuje několik pracovních postupů jako je zjišťování požadavků na software, analýza softwaru, návrh, implementace a průběžné testování implementace. K tvorbě diagramů využívá jazyk UML. [4]

4.7.1 Požadavky na aplikaci

Nároky vyvstaly především z cíle projektu, a to vytvořit ucelenou aplikaci, která by fungovala jako cvičebnice dokazování tvrzení z oblasti teorie grafů primárně pro studenty předmětu DIMA. Dále byly zohledněny konkrétní požadavky vznesené paní RNDr. Andreou Ševčíkovou, jakožto zadavatelkou projektu, a připomínky studentů z průběžného uživatelského testování aplikace.

V aplikaci by měly být dostupné příklady základních typů metod dokazování. Pro snadnější pochopení postupu v důkazu, by jednotlivé kroky měly být podpořeny interaktivní vizualizací, která bude vykazovat větší dynamiku než například vizualizace zpracované formou obrázků do dokumentu typu PDF nebo PowerPointové prezentace. K tomuto účelu je zapotřebí vytvořit vizualizační plátno, které bude schopné zobrazovat grafy a vizualizovat na nich myšlenky prováděné v jednotlivých krocích důkazu.



Obrázek 4 Diagram případů užití (Use Case) [vlastní zpracování]

Pro účely výuky patří k dalším požadavkům například možnost volného kreslení přes plátno, aby vyučující mohl k dodatečnému výkladu naznačovat své poznámky přímo v aplikaci. Pro tyto účely je vhodné kromě pera pro volné kreslení poskytnout také například nástroj čára a kruh využitelné jako rychlé naznačení hrany či vrcholu. Za výhodnou by byla považována také možnost úpravy pozic a atributů vrcholů a hran grafu. Pro účely výuky je vhodné také zavést koncept kamery, která by umožnila pohyb po plátně s přibližováním a oddalováním. Z hlediska kroků

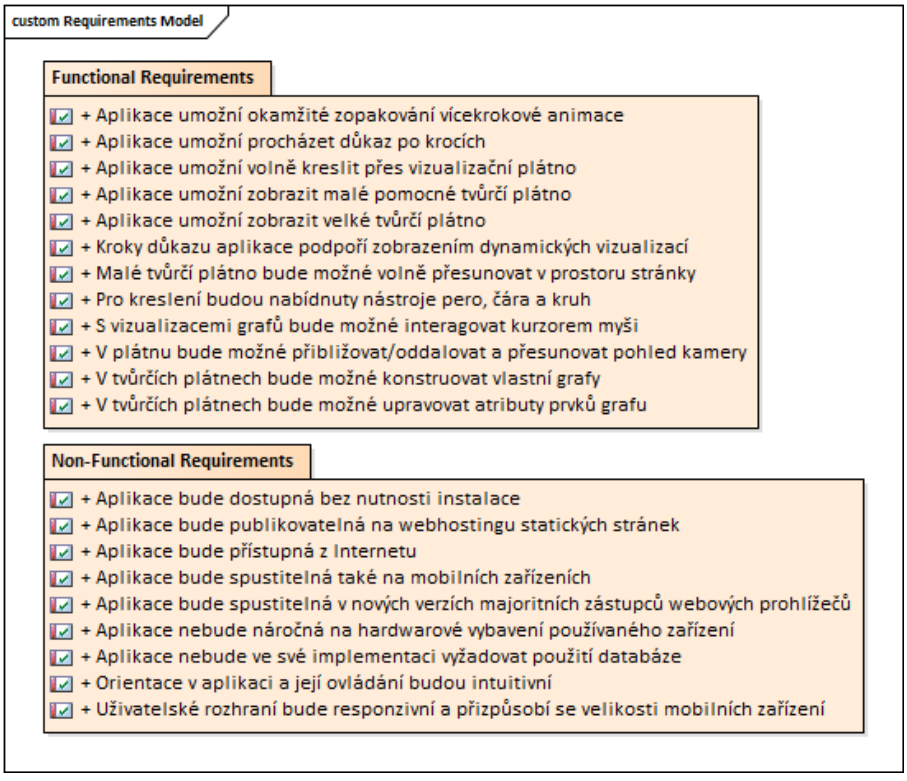
je u delších animací dobré umožnit jejich okamžité zopakování od začátku, pokud by se student v postupu ztratil.

4.7.2 Funkční a nefunkční požadavky

Z funkcí, které jsou po aplikaci požadovány, vyplynuly funkční požadavky. Na aplikaci jsou ovšem kladeny nároky ještě z pohledu tzv. nefunkčních požadavků.

Aplikace má být spustitelná bez nutnosti instalace a měla by být snadno dostupnou, například z webového prohlížeče. Postačí, bude-li aplikace spustitelná v aktuálních verzích majoritních zástupců webových prohlížečů. Především pro potřeby studentů bude vhodné, pokud aplikace bude spustitelná na mobilních zařízeních. Z tohoto důvodu je nutné, aby aplikace byla responzivní a uživatelské rozhraní se tak přizpůsobovalo i velikosti mobilních zařízení.

Přehled funkčních a nefunkčních požadavků zachycuje model na obrázku 5.



Obrázek 5 Model funkčních a nefunkčních požadavků [vlastní zpracování]

Z těchto požadavků se jako nejlepšími technologickými kandidáty jeví JavaScriptové frameworky pro tvorbu webových Single Page aplikací. Pro účely odkazování se v textu byla aplikace nazvána „ProofVis“.

4.8 Průzkum nástrojů na podporu výuky důkazů z oblasti teorie grafů

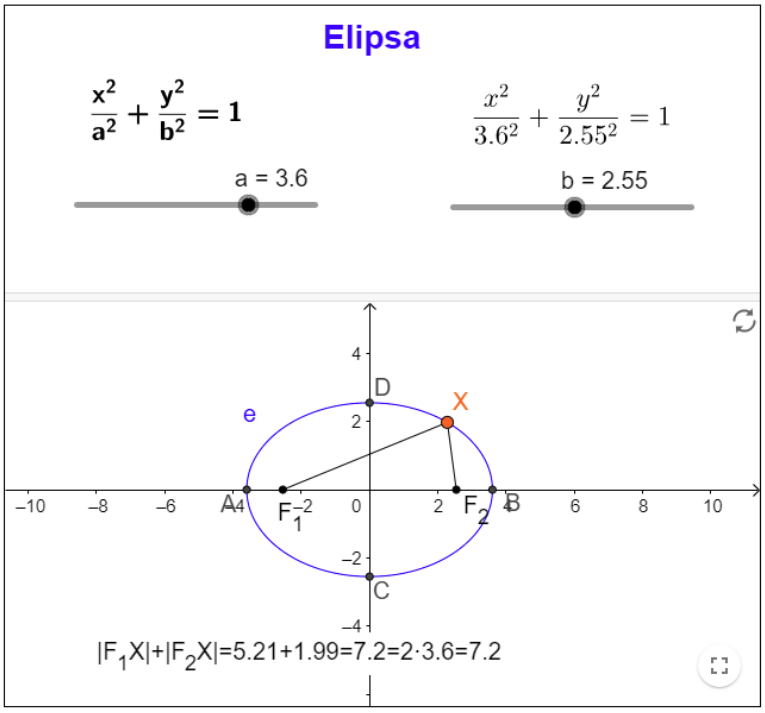
V následujících podkapitolách budou představeny programy a softwarové balíčky a jejich možnost využití ve výuce důkazů v matematice.

4.8.1 GeoGebra

„GeoGebra je dynamický matematický software pro všechny úrovně vzdělávání, který spojuje geometrii, algebru, tabulkový procesor, grafy, statistiku a analýzu do jednoho snadno použitelného balíčku.“ [68]

Aplikace umožňuje tvorbu interaktivních výukových materiálů dostupných z webových stránek a zároveň funguje jako komunita sdružující miliony uživatelů, kteří ji využívají v různých směrech [68]. Někteří plátno aplikace začleňují do online výukových prostředí a jiní se snaží najít způsoby, jak do plátna GeoGebry začlenit textové poznámky [11].

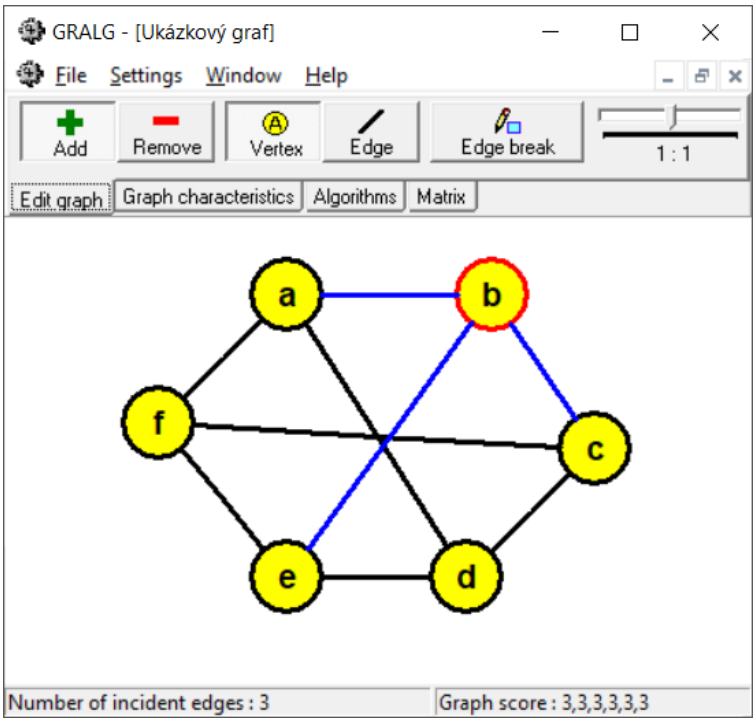
Svou geometrickou povahou však není příliš vhodná pro zpracování důkazů z teorie grafů.



Obrázek 6 Ukázka vizualizace v aplikaci GeoGebra [72]

4.8.2 GrAlg

GrAlg [47] je desktopový program určený pro podporu výuky předmětů zabývajících se grafovými algoritmy. Byl vyvinut v rámci diplomové práce publikované v roce 2010 na půdě Univerzity Hradec Králové a umožňuje základní funkce přidávání vrcholů a hran pomocí tlačítka myši a na vlastním vytvořeném grafu spustit jeden z implementovaných algoritmů nebo zobrazit informace o základních charakteristikách grafu. Grafy lze exportovat do obrázkového souboru.

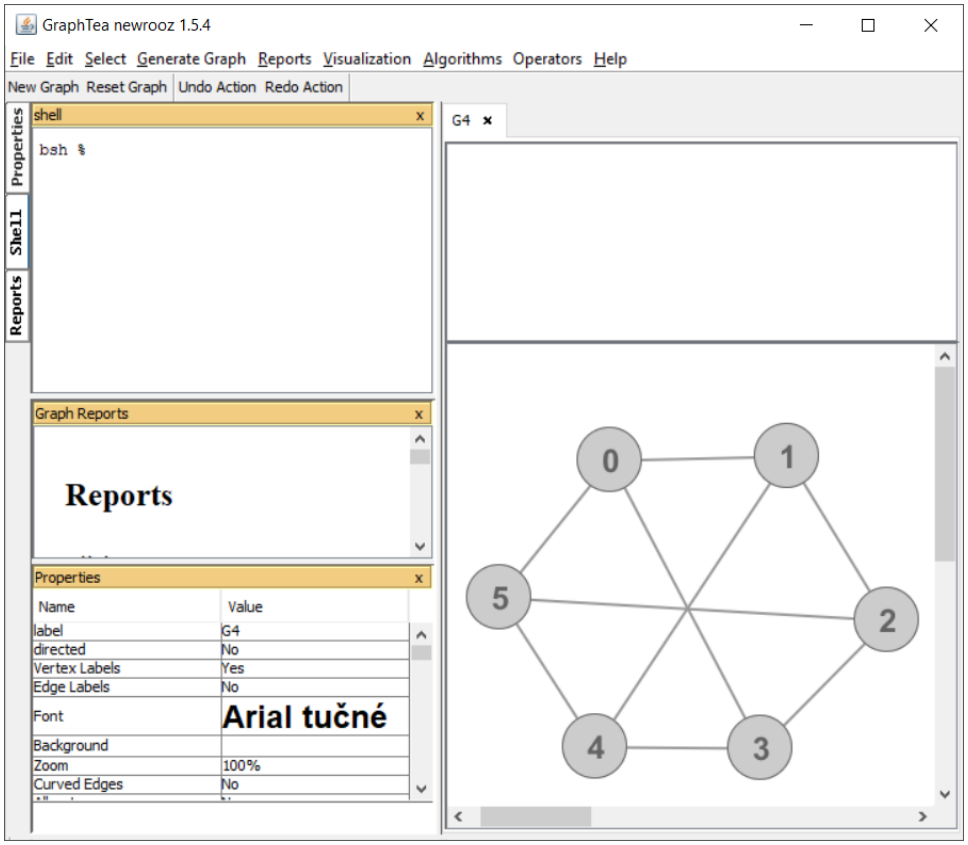


Obrázek 7 Ukázka grafu konstruovaného v programu GrAlg [vlastní zpracování]

GrAlg však nemá žádnou podporu pro jakékoli textové zpracování důkazů.

4.8.3 GraphTea

Širšími a sofistikovanějšími možnostmi disponuje program GraphTea. Nabízí více možností kreslení grafů, změny barev a jiných vlastností vrcholů i hran, automatické generování grafů. Poskytuje širokou nabídku aplikace algoritmů a možnost exportovat nejen obrázkový soubor grafu, ale také report ve formátu LaTeX dokumentu. [71]



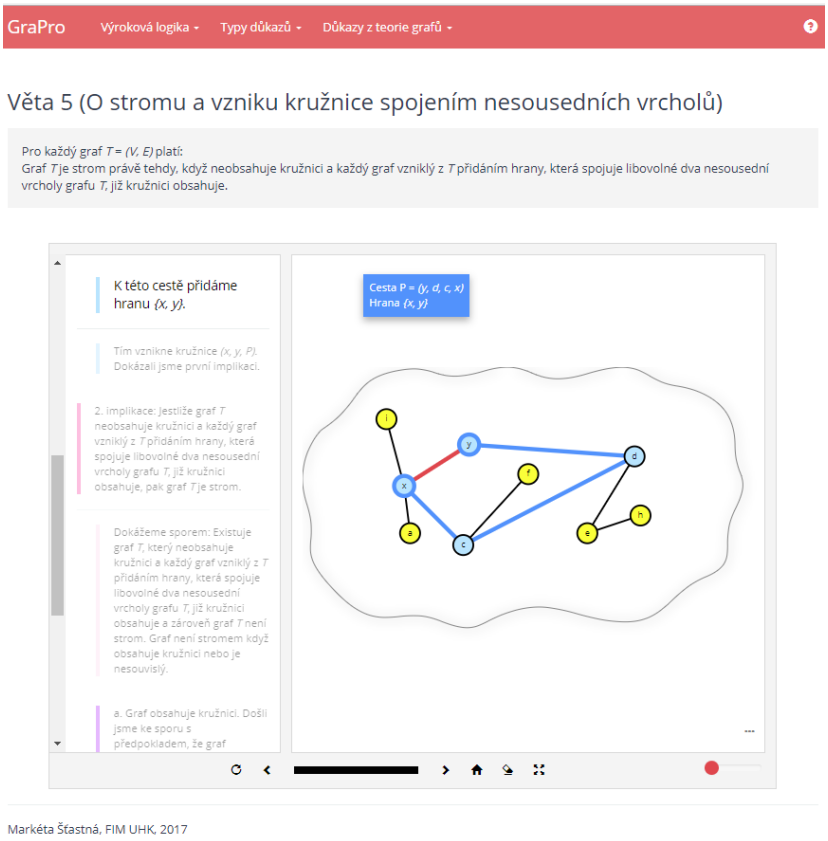
Obrázek 8 Ukázka grafu konstruovaného v GraphTea [vlastní zpracování]

Ačkoli je program zamýšlen jako podpora pro účely výzkumu i výuky, opět se jedná pouze o desktopovou aplikaci bez webové verze, která nepočítá s využitím pro zpracování matematických důkazů

4.8.4 GraPro

Paralelně s vývojem aplikace ProofVis, zpracovávané v praktické části této práce, probíhal také vývoj aplikace GraPro [50] studentky Markéty Šťastné. Od počátku zadání obou projektů paní RNDr. Ševčíkovou bylo důležitou myšlenkou, aby se vzniklé aplikace vhodně doplňovaly. Zatímco aplikace GraPro se více soustřeďuje na úvod do výrokové logiky a dokazování základních matematických vět, aplikace ProofVis byla zamýšlena jako cvičebnice důkazů, s jejichž obdobami by se student mohl setkat například u zkoušek, a které by bylo možné využít jako ukázkové příklady jak ve výukových hodinách, tak pro domácí samostudium. Aby se příklady v aplikacích neopakovaly, bylo jejich rozdělení konzultováno.

Společně tak obě aplikace tvoří celistvější výukový materiál, který ještě více podpoří studenta v přípravě.



Obrázek 9 Ukázka z aplikace GraPro [vlastní zpracování]

Vývoj aplikace ProofVis tudíž nestaví na základech aplikace GraPro. Přesto byla snaha v závěru vývoje reagovat na přínosné myšlenky a případně přinést vylepšení. Ačkoli jsou použité technologie, postupy dosažení výsledku i možnosti obou aplikací odlišné, což se projevuje jak ve zpracování vizualizací, tak v designu uživatelského rozhraní, student by neměl mít potíže s orientací při přechodu z jedné aplikace do druhé. Aplikace ProofVis se například více soustřeďuje na rozdělení příkladů důkazů dle metod dokazování, aby si student mohl vybrat přímo dle metody, kterou chce procvičovat. Dále je kladen větší důraz na interaktivitu vizualizací pro výukové účely například v podobě zvýrazňování prvků grafu při najetí myší nebo možnostech jejich vyznačování podržením klávesy CTRL. Přináší více nástrojů pro kreslení přes graf a v neposlední řadě možnost zobrazení pomocného plátna, ve kterém může vyučující sám konstruovat grafy přidáváním vrcholů a hran a měnit jejich atributy.

5 Technologie pro tvorbu studijních materiálů

Vývoj moderních webových aplikací vyžaduje znalost širokého okruhu technologií, nástrojů a postupů. Především oblast front-endového vývoje se rychle mění. K novým technologiím a nástrojům často nestačí vznikat řádná dokumentace a knihy o průlomových technikách mohou být v některých případech zastaralé už v momentě, kdy se dostanou do tisku. Současné trendy přinášejí řadu změn do zažitých postupů.

5.1 Funkcionální programování

Ačkoli se nejedná o nový koncept, funkcionální programování aktuálně posiluje svůj vliv na komunitu okolo programování. Techniky jako *immutable* (neměnné) *proměnné* a *pure* (čisté) *funkce* se ukázali jako užitečné při ladění kódu, zatímco *high-order funkce* nám umožňují extrahovat vnitřní logiku funkcí a díky efektivnější znovu-použitelnosti psát méně kódu. [5]

Techniky funkcionálního programování jsou založeny na myšlence matematické funkce. Tak jako ony mají svůj název, přijímají parametry, vrací výsledek a přitom nejsou ovlivněny ničím „z vnějšího světa funkce“ ani nic vně funkce nemění. Navíc, pro stejné vstupy vrací taková funkce vždy stejný výstup. Tato důležitá vlastnost je nazývána *referenční transparence*. Je to dáno právě tím, že funkce operuje pouze se svými vstupními argumenty a neodkazuje uvnitř na žádné globální reference. Díky tomuto předpokladu je možné například spouštět funkce ve více vláknech bez nutnosti synchronizace, a protože pro stejné vstupy vrací stejný výstup, můžeme je cachovat. Jinými slovy, funkcionální programování vede k možnosti spouštění paralelního kódu a ke cachování. [3]

5.1.1 Deklarativní paradigma

S funkcionálním programování se pojí koncept deklarativního paradigma a psaní abstraktního kódu. Většina programátorů je dnes obeznámena především s imperativním paradigma, jelikož se jedná o nejběžnější styl programování. Velmi rozšířené je také objektově orientované programování (OOP), které zahrnuje například jazyk Java, i když primárně staví na imperativním paradigma. [5]

Rozdílnost obou přístupů lze předvést již na dvou krátkých ukázkách výpisu pole čísel do konsole. K zápisu je použit jazyk JavaScript, který není nutně funkcionální, již v základu však podporuje funkcionální techniky. [7]

```
var array = [1, 2, 3];
for(i = 0; i < array.length; i++)
  console.log(array[i])
```

Ukázka kódu 1 Imperativní výpis pole čísel [3]

Tímto stylem přímo specifikujeme, „jak“ má kompilátor vypsat dané pole čísel. Tedy, že má využít smyčky pro procházení pole prvků od prvního až po poslední a postupně každý prvek vypsat. Principem deklarativního programování je však kompilátoru zadat pouze „co“ má s polem udělat. Část „jak“ toho má docílit je abstrahována použitím běžně dostupných nebo nově vytvořených abstraktních funkcí. V tomto případě lže využít vestavěné funkce `forEach` k projití a vypsání pole. [3]

```
var array = [1, 2, 3];
array.forEach((element) => console.log(element))
```

Ukázka kódu 2 Deklarativní výpis pole čísel [3]

Výstup z obou ukázek bude naprosto stejný. Smyslem funkcionálního programování je tedy tvorba abstraktních funkcí, které poté mohou být znovupoužity také v jiných částech kódu. [31]

5.1.2 Výhody funkcionálního programování

Obecně, imperativním stylem je možné psát kód „za pochodu“ bez jasné představy, jak bude výsledná implementace vypadat. To může vyústit v neudržovatelný základ plný nadbytečných tříd a tzv. „špagety kódu“. Oproti tomu funkcionální programování nutí vývojáře více se zamyslet nad výslednou podobou implementace ještě před tím, než začne samotný kód psát. Lépe se tak dá identifikovat, kde bude možné zkrátit kód znovupoužitím stejných funkcí pro dosažení požadované funkcionality. Možnost změny kódu funkce na jednom místě při potřebě úpravy funkcionalit aplikace také přispívá k lepší udržovatelnosti. [5]

Někteří lidé považují funkcionální programování za náhradu objektově orientovaného programování, nicméně, OOP bude nepochybně dále využíváno

k používání objektů a udržování jejich metod [5]. V jazycích jako JavaScript můžeme navíc prvky z obou světů vhodně kombinovat [31].

Ačkoli od vzniku funkcionálního programování uplynulo již mnoho let, až v poslední době získává výraznou popularitu. Příčinou může být, že pominula doba, kdy jakkoli napsané weby byly lepší než žádné a kdy nebylo tolik hleděno na volbu paradigma či jazyka, ve kterém byla aplikace naprogramována. [5]

Výhody efektivnějšího a lépe udržovatelného kódu, lepší testovatelnosti a výkonnosti aplikací nabývají ve světě moderního programování na větším významu. [7]

5.2 Vzestup JavaScriptu

JavaScript prošel od svého vzniku v roce 1995 řadou změn. Nejprve zjednodušil přidávání interaktivních elementů do webových stránek, s příchodem AJAXu a dynamického HTML se rozrostl v komplexní nástroj [7]. Ačkoli byl primárně navržen pro skriptování ve webovém prostředí, souhrou historických událostí se v jeho vývoji zkombinovaly nezvyklé vlastnosti [57]. Jak dále uvádí Žára [57]: *„Jeho syntaxe vychází z jazyka C, některá standardní rozhraní (vestavěné funkce, objekty, proměnné) se podobají Javě, objektový a funkcionální model je inspirován jazyky Scheme a Self.“*

Rada, která řídí změny v JavaScriptu, se nazývá European Computer Manufacturers Association (ECMA). Ta přijímá návrhy kohokoli z členů vývojářské komunity a přiděluje jim prioritu v jejich začlenění do nadcházející specifikace. [7]

Koncem roku 1999 byla přijata třetí verze specifikace jazyka, nazvaná ECMA-262, a nedostala nástupce po dlouhých 10 let. V roce 2008 totiž společnost Google vyvinula pro svůj prohlížeč Chrome novou výkonnější implementaci nazvanou V8, která přinesla myšlenku využít JavaScript i pro programování serverového kódu. S touto myšlenkou obalil v roce 2009 projekt Node.js implementaci V8 sadou rozhraní a knihoven. Získal silnou uživatelskou základnu a stal se standardem pro serverové vykonávání JavaScriptu. [57]

Protože se využití JavaScriptu rozrostlo až do serverové části, stal se z něj jazyk používaný k budování tzv. full-stack aplikací. [7]

5.3 Node.js a správce balíčků npm

Pomocí svého open source nástroje pro příkazovou řádku umožnilo Node.js spouštět JavaScriptové programy také mimo prohlížeč na lokálním počítači a serveru [2]. Vývojáře přitáhlo především možností provozovat command-line aplikace na operačních systémech Microsoft Windows, Mac OS X, Linux i SunOs bez potřeby dalších vývojářských kroků, čímž se takové aplikace stávají skutečně multiplatformní [38].

Node.js je také bohatý ekosystém modulů volně dostupných k použití. Dá se říci, že moduly Node.js jsou samostatné Node.js aplikace, které mezi sebou vývojáři dobrovolně sdílí, aby je mohly začlenit jako části do komplexnějších projektů. Takových modulů je v současnosti dostupných více než 120 000 a mohou na sobě být i závislé. K jejich správě se používá správce balíčků zvaný Node Package Manager (npm), který je zakomponován již v instalaci Node.js. [21]

Společně tvoří mocný nástroj pro lokální vývoj čistě JavaScriptových aplikací, protože umožňují vývojáři vytvářet skripty a provozovat úlohy, jako například přesun souborů nebo spuštění vývojářského serveru, a přitom se automaticky aktualizují využité balíčky [2]. Právě nástroje pro frontendový vývoj a lokální sever jsou příklady, kde bývá využití Node.js a npm zapotřebí, i kdyby samotná vyvíjená aplikace nebyla provozována na produkčním prostředí s Node.js severem [63].

5.4 ES6

V současnosti svět JavaScriptu zažívá nejhojnější období. Okolo ekosystému vzniká řada nových nástrojů, boří se mýty a zavádí se nové syntaxe a best practices. Důležité změny přinesla specifikace schválená v červnu 2015. K té se váže hned několik názvů. Nejčastěji se lze setkat s označením ES6, avšak objevují se také označení ECMAScript 6 nebo ES6Harmony. Ukazuje se, že do budoucna se organizace ECMA plánuje držet konvence ES s připojeným rokem přijetí, v tomto případě tedy oficiálně ES2015. [7]

Nové techniky, které ES6 zavedlo, jsou klíčovým prvkem využitým napříč praktickou částí. Ty nejdůležitější budou krátce rozebrány v následujících podkapitolách s podporou ukázek z implementované aplikace.

5.4.1 Nové typy proměnných const a let

Před příchodem ES6 bylo jediným způsobem, jak deklarovat proměnnou, přes klíčové slovo `var`. Nyní jsou preferovány nové způsoby. Podobně jako ostatní programovací jazyky, zavedl JavaScript konstanty. Ty lze deklarovat klíčovým slovem `const` a používají se pro proměnné, jejichž hodnotu nelze měnit. Při pokusu o jejich změnu bude ohlášena chyba. [7]

Pro proměnné, jejichž hodnotu je potřeba v budoucnu měnit, bylo zavedeno klíčové slovo `let`. Jeho implementace se však od původního `var` liší. S oběma deklaracemi zavedl totiž JavaScript tzv. scope. Tyto deklarované proměnné jsou tudíž platné pouze v rámci bloku kódu definovaného složenými závorkami (`{}`) a nemohou tudíž ovlivnit hodnoty proměnných deklarovaných mimo tento blok kódu. [65]

5.4.2 Arrow funkce

Arrow funkce dostaly svůj název podle šipky (znaménko „rovná se“ = a „větší než“ >), která je použita pro její definici. Při její definici lze vynechat klíčové slovo `function` a pokud funkce přijímá pouze jeden parametr, pak také jeho závorky. Při vracení primitivního objektu je možné dokonce vynechat příkaz `return`. [7]

Příkladem je jednoduchá funkce z Ukázky kódu 3, která zpracovávaná aplikace využívá pro zobrazení a skrytí dialogového okna po kliknutí na tlačítko.

```
export const handlerDrawingDialog = isDrawingDialogOpen => {
  return { isDrawingDialogOpen: !isDrawingDialogOpen };
};
```

Ukázka kódu 3 Příklad arrow funkce [vlastní zpracování]

Pokud parametr `isDrawingDialogOpen` indikuje, že dialogové okno je otevřené, vrátí funkce objekt, který slouží k zavření okna a naopak.

Nejzásadnější odlišností arrow funkce od funkcí původních je, že nepřerušují platnost klíčového slova `this`. [65]

5.4.3 Třídy v JavaScriptu

V minulosti specifikace JavaScriptu oficiálně nezahrnovala třídy. K vytváření tříd se používaly funkce, kterým se využitím tzv. prototypování nadefinovaly metody. Ačkoli JavaScript takto v jádru funguje i nadále, ES6 umožňuje používání syntaxe, která je obdobná té z klasických objektově orientovaných jazyků. [7]

V ukázce kódu 4 je uveden princip zápisu třídy `SingleDrawing`. Jak si lze všimnout, u tříd je možné klasicky využívat klíčové slovo `extends`, tak, aby podtřída zdědila vlastnosti a metody rodičovské třídy [7].

Vlastnosti lze třídě přidělit přes klíčové slovo `this` uvnitř konstruktoru. Zajímavá je syntaxe metod, kde se v případě metody `render(){}` z ukázky již nepoužívá klíčové slovo `function`. [45]

```
import React from 'react';

class SingleDrawing extends React.Component {
  constructor(props) {
    super(props);
    this.state = { /* ... */ };
  }

  render() {
    return;
  }
}

export default SingleDrawing;
```

Ukázka kódu 4 Příklad zápisu třídy v JavaScriptu [vlastní zpracování]

Ačkoli v praktické části bude upřednostňováno použití funkcionálních technik před technikami z objektově orientovaného paradigma, znalost syntaxe zápisu JavaScriptových tříd je stále důležitá. Pomocí nich budou totiž vytvářeny komponenty frameworku React.

5.4.4 Moduly v JavaScriptu

V rozsáhlejších JavaScriptových aplikacích je potřeba zajistit dobrou organizaci kódu. Proto specifikace ECMAScript přijala návrhy na zavedení myšlenky modulů. [45]

Modul je samostatná část znovupoužitelného kódu, který lze jednoduše začlenit do ostatních JavaScriptových souborů. V minulosti byla tato funkce dostupná pouze s využitím externí knihovny. S příchodem ES6 JavaScript moduly podporuje nativně. [7]

Pro zpřístupnění modulu ostatním souborům bylo v JavaScriptu zavedeno klíčové slovo `export`. Naopak pro deklaraci závislosti na jiném modulu se v daném souboru použije klíčové slovo `import`. Vždy je vhodné dodržovat konvenci, kdy importování požadovaných modulů uvádíme na začátku souboru a exportování objektů naopak na konci. [45]

JavaScriptové moduly jsou odděleny v samostatných souborech, ze kterých lze exportovat buď více JavaScriptových objektů nebo jeden objekt za celý modul. V případě, kdy z modulu exportujeme pouze jeden objekt, je možné použít spojení `export default`. Při importování modulu pak není nutné specifikovat, o které jeho části máme zájem. [7]

V předchozí ukázce kódu 4 je importován modul `React` a na konci souboru defaultně exportován celý objekt `SingleDrawing`. V ukázce kódu 3 je naopak použito samotné klíčové slovo `export`, jelikož jde o exportování funkce, která je jednou z mnoha v daném souboru.

5.5 Babel

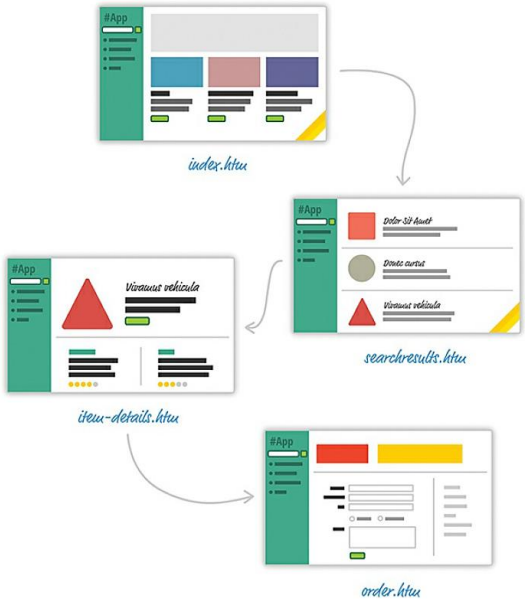
Mnoho nových vlastností je již nativně podporováno nejnovějšími webovými prohlížeči. Informace o aktuální podpoře jednotlivých novinek v různých verzích prohlížečů jsou zaznamenávány například do tabulky kompatibility na GitHub stránkách Juriye Zaytseva [92]. Ne všechny vlastnosti mají ve všech webových prohlížečích plnou podporu. Navíc budou neustále přibývat nové a není v silách vývojářů prohlížečů, aby okamžitě reagovali na nejnovější schválené specifikace.

Z těchto důvodů se v současnosti programátoři JavaScriptu spoléhají na nástroje jako je Babel, které zaručují spolehlivou funkčnost nové syntaxe ve všech prohlížečích.

Babel je nástroj, který transformuje veškerý kód zapsaný novodobou syntaxí do JavaScriptu verze ES5, kterému moderní prohlížeče rozumí, a umožňuje tak plně využívat i novější verze JavaScriptu již dnes. Je velmi modulární, postavený na pluginové architektuře s rozsáhlými možnostmi konfigurace. K malému jádru tak lze prostřednictvím npm balíčků připojit pluginy, které se vztahují například k použití ES6. [54]

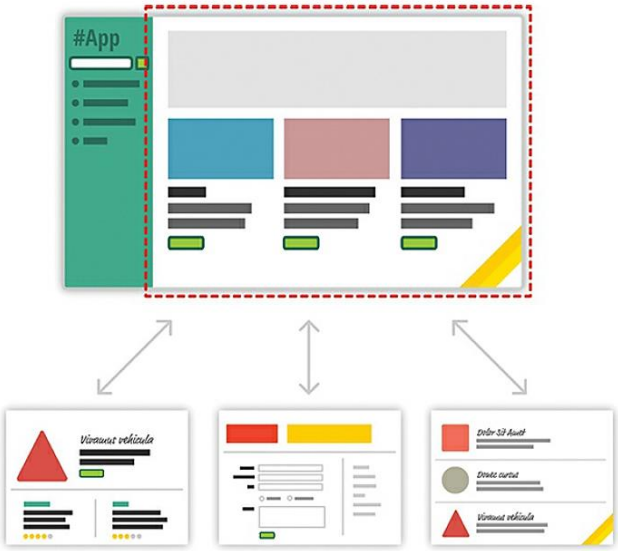
5.6 Single page aplikace

Nehledě na to, že dnešní webové aplikace vypadají navenek lépe, nastal posun také v jádru jejich fungování a architektuře. Protože se webové aplikace většinou skládají z několika individuálních stránek, bylo ještě před několika lety běžné, že odkazy v navigaci vedly k načtení a zobrazení celého dokumentu stránky. To způsobovalo nadbytečné zpomalení aplikace při procházení webu, jelikož stránky často obsahují stejné prvky a tedy redundantní kód, který nebylo nutné vždy znovu načítat. V některých případech tento jev způsoboval probliknutí stránky, které bylo viditelné, kdykoli došlo ke kompletnímu překreslení. [25]



Obrázek 10 Klasický model vícestránkové aplikace [25]

V současnosti se proto vývojáři moderních webových aplikací raději přiklánějí k využití tzv. single page modelu. Podle něj nazýváme single page aplikacemi (SPA) ty, které dokážou na webu zprostředkovat rychlost, funkcionality a další vlastnosti, jaké máme běžně spojeny spíše s desktopovými a nativními aplikacemi. Mimo jiné toho dosahují tím, že do prohlížeče je načten pouze jeden počáteční dokument stránky. Ostatní zdroje, jako skripty, styly, obrázky a jiná data jsou načítány asynchronně podle potřeby a původní dokument již nemění. Tím je také zamezeno znovunačítání kódu, který se opakuje. [24]



Obrázek 11 Ilustrace modelu single page aplikace [25]

Naplnění požadavků tohoto modelu lze dosáhnout například využitím JavaScriptových front-endových frameworků.

5.7 Front-endové frameworky

JavaScriptové front-endové frameworky se vyvíjí velmi rychle, což lze pozorovat z četnosti přibývajících commitů a vydávaných verzí. [88]

Do globálního povědomí se dostaly především frameworky Angular, Vue.js, React, Ember, KnockoutJS a Polymer. Porovnání jejich syntaxe na stejném příkladu aplikace uvádí například web todomvc.com. Lišit se mohou nejen syntaxí, ale také preferovaným jazykem i celým pojetí architektury. Tři nejpopulárnější zástupci budou představeny v následujících podkapitolách.

5.7.1 Angular

Angular je open source framework vytvořený a udržovaný společností Google, která jej v roce 2010 vydala pod názvem Angular 1. Framework ve svém vývoji prošel zásadními změnami, které se podílely na ztrátě důvěry vývojářské komunity k týmu okolo Angularu. [33]

Zatímco Angular 1 byl založen na architektuře MVC, následující verze, Angular 2, je založena na komponentově-servisní architektuře inspirované frameworkem React a odlišují se také API. Vývojáři přecházející na druhou verzi tudíž nemohli zužitkovat své znalosti z verze první. [41]

Oznámení, že verze Angular 2 bude od základů přepsána a nebude zpětně kompatibilní s Angular 1 způsobilo významný odliv komunity především ku prospěchu frameworku React. V současnosti se Angular nachází ve své 5. verzi, která poskytuje kompletní ekosystém nástrojů, ve snaze zjednodušit vývojáři tvorbu tím, že nemusí sami tyto nástroje shromažďovat formou stahování knihoven. [33]

Framework je založen na architektuře MVC a ačkoli je možné k vývoji použít jazyky Dart či JavaScript ES6, většina dostupných zdrojů upřednostňuje jazyk TypeScript. K základu tvořeným již všemi dostupnými nástroji a plnou architekturou dodává také nástroj pro command line interface (CLI), který přispívá k zrychlení vývoje. [33]

5.7.2 Vue.js

Vue je progresivní framework pro vytváření uživatelských rozhraní. Oproti Angularu je jádro zaměřeno pouze na vrstvu view z architektury MVC, a je proto principiálně snazší jej integrovat do dalších knihoven nebo existujících projektů. Vývojáři sami uvádí, že Vue má mnoho společného s Reactem, a inspiraci si bral také z Angularu 1 i Angularu 2. [91]

Za jeho výhody jsou pokládány například dobře zpracovaná dokumentace a snazší proniknutí do frameworku pro začátečníky se zkušenostmi s programováním z jiných front-endových i back-endových oblastí. [58]

Komunita kolem tohoto frameworku je však výrazně menší než u obou konkurentů a také za jeho rozvojem a podporou nestojí renomovaná společnost, což může vzbuzovat obavy z nestabilní budoucnosti frameworku. Z důvodu menší komunity je dostupných mnohem méně nástrojů a rozšiřujících knihoven. [88]

5.7.3 React

Ačkoli je React často zařazován do srovnání mezi JavaScriptové frameworky, ve skutečnosti se nejedná o plnohodnotný framework, ale o knihovnu určenou pro budování uživatelského rozhraní (často uváděno pod zkratkou UI z anglického user interface). V architektuře MVC bychom řekli, že zajišťuje roli View. React byl uvolněn společností Facebook v roce 2013 a mnozí na něj zprvu nahlíželi skepticky, jelikož zaváděl několik unikátních zvyklostí. [7]

React si zprvu získal vývojáře rychlostí, s jakou zprostředkovával načítání webových stránek, a kterou hravě předčí Angular. Přišel například s návrhem rozebrat uživatelské rozhraní aplikace na znovupoužitelné celky, tzv. komponenty, a touto myšlenkou se časem nechaly inspirovat nejen oba konkurenční frameworky, Angular i Vue.js. [58]

Fakt, že React je jen malou knihovnou, která hned v základu nezahrnuje vše, co by vývojář mohl potřebovat, je mnohými považováno naopak za výhodu. React lze snadno začlenit do existujícího projektu a naopak jej lze rozšířit o moduly, které vývojář vskutku využije. [60]

V současnosti je React stále považován za poměrně novou knihovnu. Verze React 16 uvolněná koncem září 2017 [64] zahrnula dlouho očekávaný React Fiber, reimplementaci algoritmu jádra, která měla za cíl zásadně urychlit vykreslování. Dosáhl tak milníku, kdy funkce jádra jsou poměrně stabilní, ačkoli novinky okolo Reactu rychle přibývají [7]. Komunita okolo Reactu je totiž obrovská. Ačkoli jeho první vydání nastalo až 3 roky po vydání Angularu, již v roce 2016 dosáhla celosvětově jeho poptávka podle Google Trends srovnatelné dynamiky [88], přičemž v současnosti jej již předčil [70]. Jeho popularitu a příslib stabilní budoucnosti potvrzuje také poptávka na trhu práce, kde například podle celosvětových údajů ze serveru Indeed.com k březnu 2018 o aktuálních

otevřených pozicích vyžadujících znalost konkrétního frameworku, připadá podíl 78,1% na React, 21,0% na Angular a 0,8% na Vue.js [88].

Společnost Facebook uvolnila React verze 16 pod licencí MIT [64], investuje čas do práce s komunitou a zdroje do jejích projektů. React proto používají i další velké společnosti jako Yahoo, Mozilla, Airbnb nebo Netflix, které Reactive komunitu také podporují. Pro Facebook bylo navíc od začátku důležité, aby React bylo možné snadno integrovat s existujícím kódem, a byl proto navržen i pro použití s jQuery nebo jiným JavaScriptovým frameworkem. Pro React tak vzniká mnoho užitečných open source knihoven a nástrojů, které každý vývojář může nejčastěji pod licencí MIT volně využít, a protože je knihovna Reactu navržena velmi flexibilně, má možnost volby, které knihovně dá přednost. [93]

Dostupnost široké nabídky open source knihoven je tak jedním z hlavních důvodů, proč byl pro implementaci zpracovávané aplikace zvolen právě React.

5.7.3.1 Virtuální DOM

Struktura HTML elementů každé webové stránky je určena jejím data object modelem (DOM). Protože v single page aplikacích pracujeme pouze s jedním dokumentem stránky, u kterého měníme elementy podle aktuální potřeby, je nutné manipulovat s jeho DOM. Bohužel, přidávání elementů, odebírání celých stromů elementů a jiné operace s DOM jsou jednou z výpočetně nejnáročnějších věcí, kterou můžeme po prohlížeči požadovat. [25]

Proto React přišel s návrhem zjednodušeného DOM, tzv. virtuálního DOM, který je tvořen React elementy, což jsou JavaScriptové objekty konceptuálně podobné HTML elementům. S Reactem tedy nepracujeme přímo s DOM API, ale s virtuálním DOM nebo se sadou instrukcí, které React použije ke konstrukci UI a interakci s prohlížečem. Práce s JavaScriptovými objekty je totiž mnohem rychlejší než práce přímo s DOM API. Změny provedené na virtuálním DOM poté React vykreslí použitím DOM API co nejefektivnějším způsobem. [7]

5.7.3.2 Komponenty

Každé uživatelské rozhraní aplikace je možné rozdělit na části a v některých případech vycházejí tyto části ze stejné šablony. Pro takové části UI zavedl React pojem komponenta. Komponenty Reactu jsou znovupoužitelné kusy JavaScriptového kódu, jejichž výstupem jsou HTML elementy. Již při návrhu uživatelského rozhraní je vhodné přemýšlet nad možnostmi jej rozdělit na tyto znovupoužitelné komponenty. [25]

Dále je vhodné do komponenty zapouzdřovat vše související s danou komponentou. Díky tomu pak komponenty nejen omezují opakování kódu, ale je možné je použít v různých částech UI bez hrozby, že by jejich podoba ovlivněna čímkoli nežádoucím. Z tohoto důvodu bývá ve složce komponenty umístěn také soubor kaskádových stylů (CSS) dané komponenty. [7]

5.7.3.3 Immutabilita

Pro psaní kódu v Reactu je možné použít jazyk TypeScript. Nicméně, React těží maximum z JavaScriptu specifikace ES6, staví základy na jeho syntaxi, je upřednostňován v dokumentaci a je proto také vývojářům doporučen týmem okolo Reactu. Vývojář by měl v kódu dále preferovat koncepty funkcionálního programování, které byly představeny v podkapitole 5.1. Těm napomáhá právě JavaScript ve specifikaci ES6, který pro koncepty funkcionálního programování přinesl řadu vylepšení představených v podkapitole 5.4. [87]

Jedním ze zdůrazňovaných funkcionálních konceptů v Reactu je immutabilita, neboli neměnnost. Ve funkcionálním programu by správně data neměla být měněna přímo. Jak uvádí Banks [7]: *„Místo změny originálních datových struktur, vytvoříme jejich pozměněné kopie, které použijeme namísto originálu.“* (volně přeloženo)

Díky dodržování immutability mohou být požadavky vyvolané na staré verzi dat dokončeny bez hrozby, že by data mezi tím byla změněna. To vede k jasné předvídatelnosti a tím pádem také k snazšímu odhalování vzniklých chyb. Pokud se určité proměnné nemohly změnit, dokážeme lépe odhadnout, co se mohlo objevit na vstupu funkce a co na jejím výstupu. [5]

React umožňuje různé styly managementu dat včetně mutace, nicméně, použití immutable dat ve výkonově náročných částech aplikace vede k výraznému zrychlení aplikace. [75]

I s použitím nových prostředků ES6 je v JavaScriptu immutabilita obtížně implementovatelná technika. Proto komunita kolem Reactu připravila knihovny, které dosažení immutability usnadňují. Jednou z nich je knihovna immutability-helper [75], která poskytuje například immutable verze funkcí pro operace s poli.

5.7.3.4 Stav komponenty

Komponenty, které pouze zobrazují jen zmíněná neměnná data, jsou v Reactu označovány jako *stateless* (bezstavové) a data jsou jim předána z jejich rodiče formou proměnné, v Reactu označované jako `props` (z anglického properties). Právě z důvodu rychlosti dané immutabilitou je snaha využívat stateless komponenty co nejvíce. V mnoha případech je však potřeba měnit aspekty komponenty. Například v reakci na interakci s uživatelem nebo na data vrácená ze serveru. [25]

Proto React zavedl koncept stavu, což jsou data, která komponenta využívá k vykreslení sebe sama. Komponenty, které implementují svůj objekt stavu formou proměnné `state`, jsou pak označovány jako *stateful*. [45]

Protože v SPA je udržování informací o aktuálním stavu UI náročné, pro usnadnění využívá React tzv. automatické řízení stavu UI. Jak uvádí Chinnathambi [25]: *„S Reactem je možné soustředit se pouze na finální stav UI. Nezáleží, jaký byl jeho počáteční stav. Nezáleží, jaké kroky uživatelé provedly ke změně UI. Vše na čem záleží, je konečná podoba UI. O vše ostatní se postará sám React.“* (volně přeloženo)

Zatímco v objektově orientovaném programování je stav zapouzdřen uvnitř objektu a měníme jej pomocí metod daného objektu, ve funkcionálním programování je v souvislosti s immutabilitou snaha skládat stav pomocí funkcí, které přijímají aktuální stav jako argument a vrací stav nový. [33]

Z důvodu zamezení nepředvídatelného chování aplikace by tedy objekt `state`, podobně jako objekt `props`, neměl být měněn přímo, ale voláním funkce `setState()`, která je v Reactu pro tyto účely určena. Po zavolání funkce `setState()` je aktualizace UI řízena mechanismem fronty, který efektivně dávkuje provedené změny. [45]

Kombinace konceptu virtuálního DOM, komponent a řízení stavu s důrazem na immutabilitu umožňuje Reactu dosahovat rychlostí vykreslování změn v UI nerozeznatelných od změn v desktopových a nativních aplikacích.

5.7.3.5 JSX

Aby byla tvorba komponent jednodušší, má React ještě svou vlastní syntaxi zvanou JSX. Jedná se o JavaScriptové rozšíření, které umožňuje definovat React elementy formou zápisu podobného HTML. [24]

V JSX je typ elementu specifikován tagem a atributy tagu reprezentují vlastnosti elementu. Potomky elementu lze vnořit mezi otevírající a uzavírající tagy. Díky JSX se tak kód komponent stává přehledný podobně jako HTML a XML, ačkoli se v jádru stále jedná o JavaScript. [7]

```
const StepCounter = (props) => (
  <span className="step-counter">
    {props.currentStep}/{props.stepSum}
  </span>
);
```

Ukázka kódu 5 Definice stateless komponenty StepCounter zápisem JSX [vlastní zpracování]

Pro vnoření JavaScriptových hodnot do kódu komponent se používají složené závorky. Protože je klíčové slovo `class` v JavaScriptu rezervované, pro definici atributu ho React nahrazuje klíčovým slovem `className`. [7]

5.8 Linting a Airbnb React/JSX style guide

Kód lze formátovat mnoha způsoby, nicméně aby byl co nejlépe čitelný pro člověka, je vhodné dodržovat domluvené syntaktické zásady. O to větší význam to má u nově vzniklého JSX s nezvyklou syntaxí. Proto vznikl „Airbnb React/JSX Style Guide“ [59], který je při využití Reactu jedním ze světově nejdodržovanějších průvodců

a je také oficiálně doporučen React týmem [87]. Jeho zajímavým doporučením je například nerozdělovat dlouhé textové řetězce na více řádků, jelikož to snižuje jejich přehlednost a komplikuje úpravu. Právě z pravidel tohoto průvodce bude vycházet formátování kódu praktické části.

Pro kontrolu dodržování stanovených zásad formátování, tzv. linting, vznikl nástroj ESLint. Při správném nastavení dokáže ve spolupráci s editorem například upozornit vývojáře na potencionálně nebezpečné vzory v zápisu, překlepy, zapomenuté nevyužité proměnné a pomáhá udržet konzistenci v odsazení kódu. [45]

5.9 Webpack

Webpack je nástroj, který dokáže rozličné soubory projektu (JavaScriptové, CSS, JSX, ES6 a podobně) svázat do jednoho a umožňuje tak vývojáři organizovat kód do modulů, ve kterých je snazší se orientovat. Díky tomu je snížena síťová zátěž, protože každý `script` tag s sebou nese potřebu odeslání HTTP požadavku, který zvyšuje celkovou latenci. Svázání všech souborů do jednoho balíčku dovolí načíst vše za cenu pouze jediného HTTP požadavku. [7]

Kromě toho dokáže Webpack minifikovat soubory nebo jinak optimalizovat kód a integrovat tzv. loadery pro traspilaci kódu. V případě ES6 a JSX se jedná o babel-loader, který specifikuje, na které soubory má být použit transpilátor Babel představený v kapitole 5.5. [7]

5.10 Projekt Create React App

Jak si lze představit z předchozích kapitol, pro vývoj v Reactu je zapotřebí několik nástrojů a knihoven a samozřejmě konfigurace jejich vzájemné spolupráce. Z těchto důvodů vznikl projekt Create React App, který po zadání několika příkazů do příkazové řádky Node terminálu automaticky vytvoří nový projekt s výchozí konfigurací a ulehčí tak vývojáři založení nového projektu. [25]

Vytvořené prostředí zahrnuje vše potřebné pro sestavení moderní single page React aplikace. Jmenovitě se jedná o podporu syntaxe Reactu, JSX, ES6 a novinek

v JavaScriptu nad rámec ES6, jako je například tzv. spread operátor, který je v kódu projektu reprezentován třemi tečkami a slouží vytváření kopií objektů. Dále zahrnuje vývojový server, podporu pro tvorbu unit testů a konfigurace pro Webpack, Babel i ESLint. Celý balíček je poté spravován komunitou jako jediná závislost, což ulehčuje aktualizace na nové verze. V případě potřeby je prostřednictvím příkazu `eject` možné přesunout všechny závislosti přímo do projektu a spravovat tak veškerou konfiguraci po svém. [69]

5.11 Routování a knihovna React Router

V klasické vícestránkové aplikaci odpovídá jedna URL jedné stránce. Protože je v SPA využíván pouze jeden dokument přináší tento přístup komplikace při změnách URL v adresním řádku webového prohlížeče. Také zde je ovšem potřeba zajistit, aby odkaz v navigaci na jinou stránku upravil URL adresu a aby byla zajištěna synchronizace historie prohlížeče. Takové techniky jsou souhrnně označovány pod pojmem routování, kdy je snaha mapovat URL adresy na cíle, kterými nejsou fyzické stránky. [25]

Možnosti deklarativního routování lze do SPA vyvíjené v Reactu přinést instalací JavaScriptové knihovny React Router [82].

5.12 Bootstrap a Font Awesome

Bootstrap [62] je populární front-endový open source framework pro usnadnění vývoje webového uživatelského rozhraní. Zahrnuje vzory nejpoužívanějších webových komponent, jako jsou navigace, tlačítka, formuláře, tabulky, modální okna, ale také typografické styly. Jeho systém mřížky je významným pomocníkem při tvorbě responzivního designu.

Častým způsobem integrace Bootstrapu do Reactu je využití knihovny React Bootstrap [76], která implementuje klasické komponenty Bootstrapu jako React komponenty. Tak lze v kombinaci s komponentami Bootstrapu využít naplno výhody Reactu. Pomocí knihovny react-router-bootstrap [83] lze navíc zajistit integraci mezi knihovnami React Bootstrap a React Router. [69]

Pro implementaci vektorových ikon lze využít například knihovnu Font Awesome [67] obsahující sadu ikon zdarma. Jejím týmem je pro kombinaci s Reactem oficiálně doporučeno použít knihovnu react-fontawesome [79] a její React komponentu pro Font Awesome verze 5.

5.13 MathJax

Při zpracovávání matematických učebnic, knih a jiných výukových materiálů je pro zápis matematických symbolů běžné používat matematický editor se specifickým stylem písma. Ve webové aplikaci není možné takový zápis replikovat prostou volbou příslušného stylu písma. Řešení tohoto problému přináší JavaScriptový zobrazovací engine MathJax.

MathJax je dostupný formou open source JavaScriptové knihovny a používá CSS (kaskádové styly) s webovými styly písma a SVG (Scalable Vector Graphics) tak, aby jeho výpočty mohly přizpůsobit velikost písma okolí bez ztráty kvality. K matematickému zápisu je možné použít syntaxi MathML, TeX nebo ASCIImath. MathJax je funkční ve většině prohlížečů včetně IE 6. [74]

Pro integraci MathJaxu do React aplikací vytvořila komunita řadu volně dostupných knihoven. Jednou z nich je například knihovna react-mathjax [81], která zpracovává MathJax do podoby React komponenty.

5.14 Vis.js

Vis.js je JavaScriptová vizualizační knihovna umožňující zpracovávat rozsáhlá dynamická data a dále s nimi manipulovat a interagovat. Skládá se z několika komponent a jednou z nich je také komponenta Network sloužící k zobrazování grafových sítí. [89]

Pro vykreslování grafů používá HTML canvas a v moderních webových prohlížečích tak dokáže hladce vizualizovat grafy o počtu i několika tisíc vrcholů a hran. Dále umožňuje například měnit tvary a velikosti vrcholů, barvy v grafu a mnoho dalšího. Pro tyto účely je komponenta Network ještě dále rozdělena do samostatných modulů pro jednotlivé prvky grafu. Tyto moduly pak obsahují vlastní dokumentaci k jejich možnostem nastavení, metodám a eventům. [90]

K využití modulu Network knihovny Vis.js s Reactem byla komunitou implementována samostatná knihovna react-graph-vis [80]. Ta zpracovává plátno Vis.js jako samostatnou React komponentu a umožňuje tak využít přínosy z kombinace obou knihoven.

Vývoj v Reactu zapadá do moderního JavaScriptového ekosystému. Vývojář by měl být obeznámen s funkcionálním paradigma v JavaScriptu, stejně jako s novými vlastnostmi verze ES6 [2]. V současnosti se React těší velké popularitě a udává trendy a směr i pro konkurenční JavaScriptové frameworky. Forma jeho návrhu do podoby knihovny a priorita v jednoduchosti jeho integrace do projektů stavících na různých technologiích přispívá k šíři jeho využití. Zároveň se velikost jeho komunity odráží ve velkém počtu vznikajících volně dostupných knihoven a přídavných balíčků, o které lze React rozšířit.

5.15 Testovací nástroje Jest a Enzyme

Již v průběhu vývoje je vždy doporučeno testovat kód pro odhalení potencionálních chyb. Jedním ze způsobů takového testování je psaní automatizovaných jednotkových testů. Jednotkové testy neprobíhají přímo v prohlížeči, ale pomocí Node.js balíčků jsou emulovány tak, jako by probíhaly v prostředí prohlížeče. Za provádění kódu připravených testů v tomto emulovaném prostředí je zodpovědný tzv. test runner. Ten také poskytuje validační knihovnu, která umožňuje porovnávat výsledky testů s očekávanými výsledky a případně oznámit chyby. Projekt Create React App již obsahuje předkonfigurované testovací prostředí a využívá test runner zvaný Jest [73], který není omezený pouze na testování React aplikací, ale je v jeho spojitosti často využíván. [69]

Při testování React aplikací jsou dále užitečné nástroje pro testování komponent a jejich emulaci. Jedním z nich je volně dostupný nástroj Enzyme [66] od společnosti Airbnb, který je také oficiálně doporučen React týmem. [69]

6 Tvorba studijních materiálů

Tato kapitola popisuje konkrétní postupy zvolené při tvorbě výukové aplikace. Postupně je představen návrh uživatelského rozhraní, rozebrána je struktura aplikace, detailněji je popsáno fungování vizualizace důkazů a místy jsou vysvětlena a zdůvodněna zvolená řešení zajímavých problémů, které při vývoji vyvstaly. Realizovány jsou požadavky na aplikaci, které vyvstaly z analýzy aplikace představené v podkapitole 4.7, a k jejich implementaci jsou využity technologie představené v předchozí kapitole 5.

6.1 Struktura aplikace

Díky modulům v JavaScriptu verze ES6 je možné zdrojový kód single page aplikací v Reactu, který může obsahovat několik tisíc řádků, rozdělit do samostatných souborů. Aplikace jsou obvykle složeny z mnoha komponent umístěných v samostatných složkách a je proto vhodné stanovit adresářovou strukturu projektu. Základní přehled adresářové struktury aplikace ProofVis vypadá následovně:

- `public`
  - `assets`
  - `index.html`
- `src`
  - `components`
    - `dialogs`
    - `pages`
    - `proofs`
  - `functionality`
  - `UI`
  - `App.jsx`
  - `index.js`

Aby mohl být sestaven produkční build React projektu, musí s přesným názvem existovat soubor `public/index.html`, který je šablonou dokumentu SPA,

a soubor `src/index.js`, který je vstupním bodem JavaScriptu. Veškeré soubory, využívané přímo v kódu dokumentu `index.html`, musí být umístěny ve složce `public`. Zde lze proto nalézt například ikonu pro záložku webového prohlížeče a další obrázky.

JavaScriptové soubory a soubory kaskádových stylů jsou umístěny v adresáři `src`, aby je mohl zpracovat Webpack. V souboru `index.js` je komponentě `ReactDOM` předán tzv. `root` element, který ukazuje na místo v dokumentu `index.html`, do kterého se mají vykreslovat výstupy z aplikace. Pro inicializaci aplikace je podle zažitých konvencí vytvořen soubor `App.jsx`.

Složka `components` obsahuje stateful komponenty, tedy potomky třídy `React.Component`, které ke svému fungování využívají stav. Ty jsou dále složkami rozdělené na dialogy, důkazy a ostatní stránky aplikace. Pro každou komponentu důkazu je vytvořená samostatná složka, ve které je umístěn soubor `constants` sdružující konstantní data využitá v komponentně důkazu. Toto oddělení části dat zpřehledňuje a usnadňuje především tvorbu a úpravu textů důkazu.

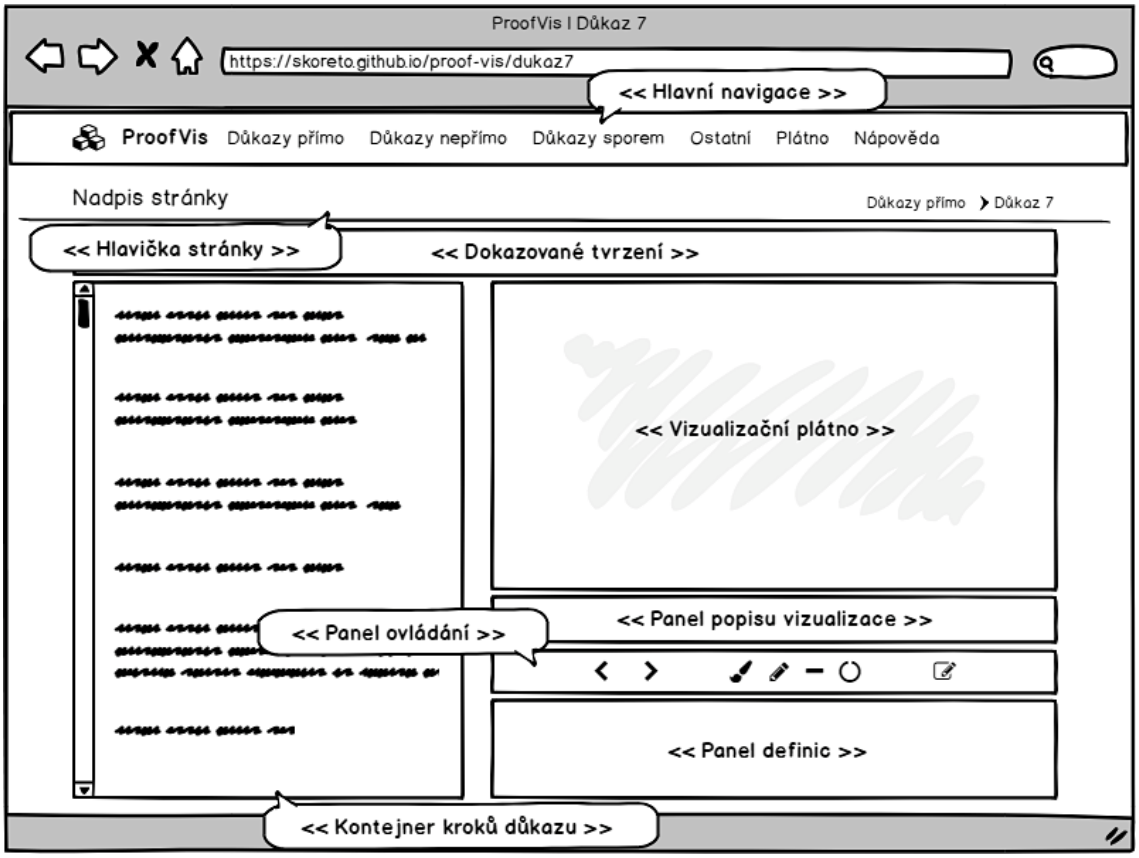
Stateless komponenty definované jako funkce jsou umístěny v samostatné složce `UI` ve vrchním adresáři `src`, neboť se týkají výhradně budování uživatelského rozhraní. Pro každou stateless komponentu je vytvořena samostatná složka, kde může být přiložen její vlastní soubor CSS. To zajišťuje její nezávislost. Pro lepší přehlednost jsou složky komponent zanořovány do sebe dle skutečného využití. Například do složky šablony pro tělo stránky důkazu (`ProofWrapper`) jsou proto zanořeny komponenty panelů, které jsou využity výhradně na stránkách důkazů, ačkoli mohou být v případě potřeby využity v kterékoli části aplikace.

V adresáři `functionality` jsou umístěny soubory zajišťující provádění operací potřebných z různých komponent a globálně dostupné konstanty. Jsou zde definovány například funkce pro úpravy grafů, počáteční stavy důkazů nebo paleta barev.

V případě použití defaultního exportu jsou soubory pojmenovány stejně, jako objekty, které jsou z nich defaultně exportovány. Po vzoru Airbnb Style Guide mají soubory obsahující JSX zápis přidělenu příponu `.jsx`.

6.2 Návrh uživatelského rozhraní

Při využití komponentového frameworku je vhodně již návrhu uživatelského rozhraní přemýšlet nad jeho rozdělením na části. Nejvýraznější komponenty jsou popsány wireframem stránky důkazu na obrázku 12, který byl zpracován v aplikaci Balsamiq [61]. Tyto komponenty se mohou dále skládat z menších komponent.



Obrázek 12 Návrh UI a identifikace komponent [vlastní zpracování]

Pro přepínání mezi důkazy zahrnuje návrh horizontální navigaci připnutou navrch stránky tak, aby co nejméně omezovala prostor prohlížeče. Hlavička stránky obsahuje po levé straně nadpis stránky. V případě stránky důkazu tedy název důkazu, a po pravé straně drobečkovou navigaci, která pomáhá uživateli s orientací v aplikaci. Pod hlavičkou se nachází výrazný panel s dokazovaným tvrzením, který již spadá pod hlavní tělo stránky důkazu. To dále tvoří kontejner kroků důkazu. Po konzultacích se studenty byl celý kontejner pro lepší intuitivnost přemístěn z pravé strany aplikace na levou. Člověk je přirozeně zvyklý číst zleva doprava, což navádí to studenta přečíst si nejprve probíraný krok důkazu, a teprve

poté se soustředit na vizualizaci. Postup důkazu může skládat z mnoha kroků, je mu proto věnován zbývající podélný prostor stránky. Na pravé straně je tedy umístěno vizualizační plátno s panelem poskytujícím bližší popis. Průběh důkazu lze ovládat z panelu nástrojů, kde jsou umístěny také tlačítka pro kreslení přes plátno a tlačítko pro zobrazení pomocného plátna, ve kterém je možné sestrojovat vlastní grafy. Podle potřeby se v průběhu důkazu může objevit panel definic, jehož účelem je připomenout studentovi znění důležitých definic a vět použitých v daném kroku důkazu.

Cílem návrhu bylo co nejefektivněji využít prostor prohlížeče. Protože pozice komponent v těle stránky nejsou stanoveny absolutně, mohou se v zájmu responzivity na obrazovkách mobilních zařízení přemístit nad sebe a využít tak naplno jejich šířku.

6.3 Realizace uživatelského rozhraní

Části uživatelského rozhraní se skládají zejména ze stateless komponent, tedy komponent tvořených arrow funkcí, která může přijímat jako argument objekt `props` předaný z rodičovské komponenty. Pokud není nutné v komponentě využívat objekt stavu, je využití stateless komponent preferováno před stateful komponentami z důvodu snadnější testovatelnosti a urychlení jejich vykreslování.

6.3.1 Hlavní navigace

Hlavní horizontální navigace je tvořena stateless komponentou `MainNavbar`, jejíž zjednodušený úryvek zdrojového kódu lze vidět v příloze č. 1 na konci této práce. Názvy React komponent jako `Navbar`, `NavDropdown` nebo `NavItem` jsou obdobou názvů CSS tříd v Bootstrapu. Použitím knihovny react-bootstrap lze tak dosáhnout plně komponentové obdoby klasické bootstrapové navigace. Díky tomu, je horizontální menu plně responzivní a na úhlopříčkách displejů mobilních zařízení se změní na uživatelům věrně známé rozbalitelné menu.

6.3.2 Routování

V příloze 1 si lze povšimnout také komponenty `LinkContainer` z knihovny react-router-bootstrap, která souvisí s logikou routování. Komponenta `LinkContainer` zastupuje chování komponenty `Link` knihovny React Router pro obalení komponent knihovny React Bootstrap. Stejně jako `Link`, má vlastnost `to`, ve které přijímá řetězec odpovídající vlastnosti `path` komponenty `Route`. Tím `LinkContainer` definuje odkazu v menu místo v aplikaci, na které má vést.

Samotné routování, tedy přepínání mezi routami na různá místa aplikace, a historii pro webový prohlížeč zajišťuje router. V případě této práce je konkrétně zvolen typ `BrowserRouter`, který používá standardní formát URL adres bez přidávání hash symbolu (#). Pod tuto komponentu jsou uzavřeny komponenty Route, kde každá z nich určuje, která komponenta se má vykreslit po odkázání na příslušný řetězec cesty. Úryvek této části zdrojového kódu lze vidět na ukázce kódu 6.

```
<BrowserRouter basename={process.env.PUBLIC_URL}>
  <div>
    <MainNavbar />
    <Grid>
      <Switch>
        <Route exact path="/" component={Overview} />
        <Route path="/dukaz1" component={Proof1} />
        <Route path="/dukaz2" component={Proof2} />
        <Route path="/dukaz3" component={Proof3} />
        /* ... */
        <Route path="/napoveda" component={Help} />
        <Route path="/platno" component={SingleDrawing} />
      </Switch>
    </Grid>
  </div>
</BrowserRouter>
```

Ukázka kódu 6 Definice komponenty BrowserRouter v souboru App.jsx [vlastní zpracování]

Po kliknutí na odkaz v navigaci tedy `BrowserRouter` porovná řetězec vlastnosti `to` v komponentě `LinkContainer` s řetězci vlastností `path` v komponentách `Route` a vykreslí komponentu definovanou ve vlastnosti `component`. Jak lze dále z ukázky vidět, při přepínání mezi routami se vykreslují komponenty s těly stránek. Hlavní navigace reprezentovaná komponentou `MainNavbar` zůstává pro každou stránku stejná. Z tohoto důvodu je definice komponenty `BrowserRouter` umístěna již ve vrchní komponentně `App`. Protože každá stránka obsahuje tělo pouze jednou

a tedy jedna cesta vede vždy na jednu komponentu těla, jsou komponenty `Route` obaleny ještě komponentou `Switch`, která zajistí, že `BrowserRouter` vykreslí hned první komponentu, která odpovídá požadavku.

Aby routování správně fungovalo jak ve vývojovém prostředí, tak na produkčním serveru, kde je základ URL adresy dán hostingovou službou, je komponentě `BrowserRouter` přidána vlastnost `basename` s hodnotou proměnné `process.env.PUBLIC_URL`, která základ URL adresy na produkčním serveru doplní. První routě je navíc přidána vlastnost `exact`, což značí, že komponenta `Overview` reprezentující úvodní stránku bude zobrazena pouze, pokud URL adresa odpovídá přesně kořenovému adresáři `/`.

6.3.3 Ikony v Reactu

Již v příloze č. 1 bylo možné si povšimnout použití ikon v navigaci. Pro vložení Font Awesome ikon do React aplikace je využita knihovna react-fontawesome, zmíněná v kapitole 5.12. Její implementace je provedena formou vytvoření objektu referencí na ikony. V souboru `App.jsx`, kde je celá aplikace inicializována, je importován objekt `library` z knihovny fontawesome-svg-core a dále objekty vybraných ikon z volně dostupné sady ikon free-solid-svg-icons. Vybrané ikony jsou poté přidány do objektu `library`, který tak obsahuje jejich reference.

```
import { library } from '@fortawesome/fontawesome-svg-core';
import {
  faChevronRight,
  faChevronLeft,
  /* ... */
  faEdit,
} from '@fortawesome/free-solid-svg-icons';

library.add(
  faChevronRight,
  faChevronLeft,
  /* ... */
  faEdit,
);
```

Ukázka kódu 7 Úryvek inicializace sady ikon v souboru App.jsx [vlastní zpracování]

Díky tomuto přístupu, není nutné importovat ikony zvlášť do každé komponenty, kde se používají. V JSX zápisu je možné se poté odkázat na ikonu pomocí komponenty `FontAwesomeIcon` a názvu ikony odpovídajícího názvu v oficiální

dokumentaci sady Font Awesome ikon. Přehledné použití samostatné ikony v tlačítku pro přepnutí na následující krok důkazu lze vidět na ukázce kódu 8.

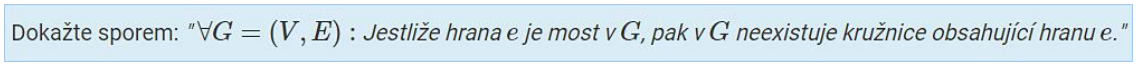
```
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';

<Button clicked={props.nextStep} disabled={props.btnNextD}>
  <FontAwesomeIcon icon="chevron-right" />
</Button>
```

Ukázka kódu 8 Využití ikony šipky vpravo pro tlačítko dalšího kroku [vlastní zpracování]

6.3.4 Zápis matematických symbolů

Aplikace je tvořena primárně pro studenty, kteří mají znalosti z teorie grafů. Lze proto využívat patřičnou terminologii včetně matematického zápisu. Pro tyto účely byl pomocí knihovny react-mathjax2 implementován zobrazovací engine MathJax. Nejjednodušší příklad jeho použití je v panelu dokazovaného tvrzení na obrázku 13.



Obrázek 13 Panel dokazovaného tvrzení [vlastní zpracování]

Pro vložení matematického zápisu do řádku s okolním textem je ve zdrojovém kódu nutné ohraničit zvolený text komponentou `MathJax.Node`, přidat vlastnost `inline` a celý blokový prvek, obsahující matematické zápisy obalit komponentou `MathJax.Context`. V její vlastnosti `input` je dále potřeba určit, jestli je matematický zápis proveden ve formátu AsciiMath nebo LaTeX.

```
import MathJax from 'react-mathjax2';

<MathJax.Context input="tex">
  <div>
    Hrana <MathJax.Node inline>{'e=\\{x,y\\}'}</MathJax.Node> je
    most v grafu <MathJax.Node inline>G</MathJax.Node>.
  </div>
</MathJax.Context>
```

Ukázka kódu 9 Surový matematický zápis z pohledu aplikace [vlastní zpracování]

Zdrojový kód z ukázky 9 by pak na výstupu v prohlížeči zobrazil zformátovanou větu: „Hrana $e = \{x, y\}$ je most v grafu G.“ Protože React využívá složené závorky (`{}`) pro ohraničení JavaScriptového kódu, je možné si v ukázce kódu 9 také všimnout, že některé matematické výrazy je nutné escapovat.

Zápis zdrojového kódu tímto způsobem by však byl značně neefektivní a nepřehledný. Proto byla mimo jiné v aplikaci implementována pomocná stateless komponenta `MathJaxNode`, která je do míst s matematickým zápisem zpravidla importována pod zkratkou `MN`, a která tak zkracuje kód pro jeho ohraničení.

```
const MathJaxNode = (props) => (
  <span className={props.classes}>
    <MathJax.Node inline>
      {props.children}
    </MathJax.Node>
  </span>
);
```

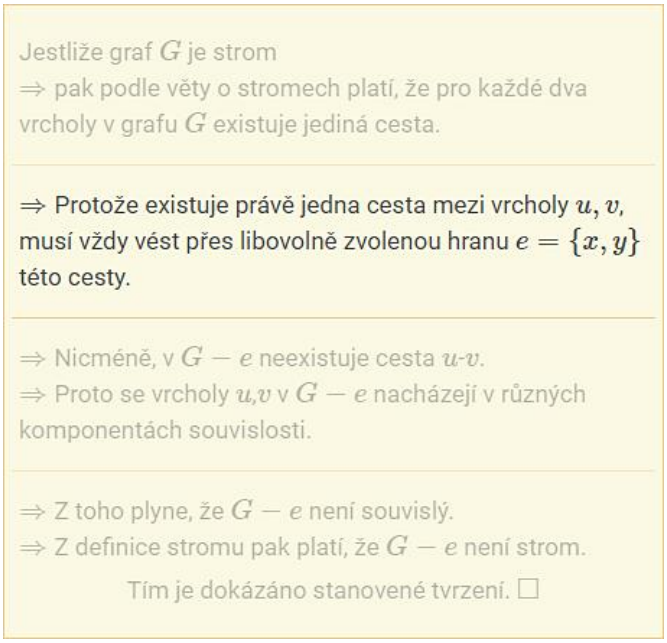
Ukázka kódu 10 Komponenta pro ohraničení matematického zápisu [vlastní zpracování]

Pro vyznačení symbolu *G* v textu je tak možné zapsat pouze `<MN>G</MN>` a kód textu tvrzení z obrázku 13 je tak ohraničením matematického zápisu dotčen minimálně. Sofistikovaněji je řešeno také ohraničení matematických zápisů bloku kódu komponentou `MathJax.Context`. To zajišťují až samotné hlavní komponenty UI, kterým je předáván pouze textový obsah pro vykreslení.

Kosmetickým problémem při použití MathJaxu v klasické více stránkové aplikaci je, že při každém načtení nové stránky je analyzován její obsah a teprve po nalezení úryvků MathJaxu jsou matematické zápisy transformovány do správné podoby. Tento proces se projevuje zpožděným naformátováním matematických symbolů v momentě načítání stránky, při němž si lze povšimnout viditelné transformace z LaTeX formátu a v horším případě i posunu celého textu o několik řádků. Protože SPA zpracovaná Reactem přednačítá analyzované prvky všech stránek již při prvním vstupu do aplikace, tento problém je z velké části vyřešen. V případě, že první vstup uživatele vede na stránku bez prvků MathJaxu, odpadá tento problém zcela, jelikož pro každou další stránku byly již prvky MathJaxu analyzovány a zformátovány.

6.3.5 Kontejner kroků důkazu

Jednou z hlavních komponent uživatelského rozhraní aplikace je kontejner kroků důkazu. Ten seskupuje menší bloky kódu, panely důkazu, které popisují kroky prováděné při dokazování. Tyto panely jsou pro každý důkaz definovány v poli `proofStepPanels`.



Obrázek 14 Kontejner kroků důkazu [vlastní zpracování]

Každý objekt panelu důkazu má definovány 3 vlastnosti. Textový řetězec `name` identifikuje panel pro účely posunu posuvníku a zároveň je využit jako jedna složka klíče při mapování panelů do kontejneru panelů. Hodnotu vlastnosti `activeForSteps` tvoří pole čísel kroků důkazu, pro které má být panel zvýrazněn jako aktivní. A konečně hodnotu vlastnosti `content` tvoří obsah kroku důkazu, který má být v panelu zobrazen. Díky využití zápisu JSX v Reactu, lze tento obsah definovat na pohled podobně, jako elementy v HTML kódu.

```
export const proofStepPanels = [
  /* ... */
  {
    name: 'proofStepPanel2',
    activeForSteps: [2, 3],
    content:
      <p>
        Protože uvažujeme souvislý graf <MN>G</MN>, musí mezi
        libovolně zvolenými vrcholy <MN>u</MN> a <MN>v</MN> existovat
        cesta.
      </p>
  },
  /* ... */
];
```

Ukázka kódu 11 Úryvek objektu panelu z pole panelů [vlastní zpracování]

React ve vykreslování elementů z pole vyniká. V ukázce 12 je možné vidět úryvek zdrojového kódu komponenty `ProofStepsBox`, ve které je pomocí JavaScriptové

metody `.map()` immutabilně transformován obsah všech kroků důkazu z předaného pole na vnitřní elementy kontejneru důkazů. Navíc je porovnán stav aktuálního kroku důkazu s čísly kroků, pro které má být daný panel zvýrazněn, a podle toho je panelu přidělena CSS třída `proof-active`. Tím je umožněno zvýraznění jednoho panelu po více kroků důkazu, ve kterých může být postup podrobněji vysvětlen například několika různými vizualizacemi.

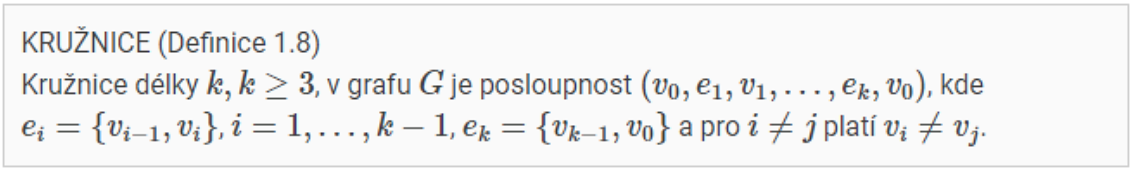
```
const ProofStepsBox = (props) => (
  <MathJax.Context input="tex">
    <Element id="proof-steps-box">
      {props.proofStepPanels.map((proofStepPanel, index) =>
        <Element
          key={index + "-" + proofStepPanel.name}
          name={proofStepPanel.name}
          className={
            proofStepPanel.activeForSteps.includes(props.currentStep)
            ? "proof-step-active"
            : ""
          }
        >
          {proofStepPanel.content}
        </Element>
      )}
    </Element>
  </MathJax.Context>
);
```

Ukázka kódu 12 Úryvek stateless komponenty ProofStepsBox [vlastní zpracování]

Z důvodu, že React volá tzv. diffing algoritmus, není vhodné použít jako `key` přímo číselnou hodnotu indexu panelu v předávaném poli panelů. V případě odebrání jednoho z panelů by došlo ke změně indexů prvků v poli a klíče by nadále již neodpovídaly. Pro zaručení unikátnosti klíče je možné použít kombinaci indexu s nějakými daty prvku z pole. V tomto případě je využit textový řetězec z kombinace indexu a vlastnosti `name`. Tím je zaručena unikátnost klíče i v případě, kdy by omylem dva panely obsahovaly stejnou hodnotu `name`.

6.3.6 Panel definic a panel popisu vizualizace

Podobným způsobem jako kontejner kroků důkazu je řešen panel popisu vizualizace, jehož účelem je ujasnit studentům význam zobrazené vizualizace, a panel definic, který jim má zase připomenout znění důležitých definic a vět použitých v daném kroku důkazu.



Obrázek 15 Definice kružnice v panelu definic [vlastní zpracování]

Rozdílným požadavkem na tyto komponenty však je, že v jednom kroku důkazu mají zobrazovat vždy obsah maximálně jednoho panelu. Před mapováním objektů z pole panelů je proto v aktuálním kroku příslušné pole nejprve přefiltrováno metodou `.filter()`, která propustí pouze ten panel, který má ve svém číselném poli s kroky pro jaké má být zobrazen (hodnota `showForSteps`), uveden aktuální krok. Především panel definic není potřeba zobrazovat u každého z kroků důkazu. Číselné pole kroků, pro které má být definice zobrazena, navíc umožňuje zobrazení stejné definice v různých krocích a tím omezuje opakování stejného kódu pro různé kroky.

```
const DefinitionPanel = (props) => (
  <MathJax.Context input="tex">
    <div>
      {props.definitionPanels
      .filter(
        definitionPanel =>
        definitionPanel.showForSteps.includes(props.currentStep)
      )
      .map((definitionPanel, index) =>
        <div
          key={index + "-" + definitionPanel.id}
          className="definition-panel"
        >
          {definitionPanel.content}
        </div>
      )}
    </div>
  </MathJax.Context>
);
```

Ukázka kódu 13 Úryvek stateless komponenty DefinitionPanel [vlastní zpracování]

Na rozdíl od panelů v kontejneru kroků důkazu, kde každý panel musel mít pro funkci posunu posuvníku přidělen textový řetězec jako hodnotu vlastnosti `name`, je u těchto panelů jako `key` pro mapování využita kombinace indexu v poli a číselného `id`.

6.3.7 Šablona stránky důkazu

Protože se rozvržení těla stránky každého důkazu shoduje, byla vytvořena komponenta `ProofWrapper`, která slouží jako šablona. Ta sdružuje rozvržení menších komponent, jejichž složky jsou proto ve struktuře projektu umístěny jako podadresáře složky `ProofWrapper`.

Ve stateful komponentě každého důkazu jsou poté na výstupu v metodě `render()` data o stavu důkazu předána stateless komponentě `ProofWrapper`, která se postará o rozvržení stránky a další předání dat do příslušných menších komponent. Komponenta `ProofWrapper` tak odstiňuje logiku průběhu důkazu od řešení výsledného vzhledu stránky.

6.4 Vizualizační plátno

K vizualizaci plátna je použit balíček react-graphvis, který prostřednictvím komponenty `GraphVis` zpřístupňuje do React aplikace HTML canvas z modulu „network“ knihovny Vis.js a jeho metody. Komponenta GraphVis přijímá mimo jiné jako `props` objekt se seznamem vrcholů (`nodes`) a hran (`edges`) a objekt nastavení vizualizace (`options`). Graf je tedy definován seznamem vrcholů a hran tak, jak bylo teoreticky popsáno v kapitole 3.1.1.2.

Každému vrcholu je nutné přidělit unikátní hodnotu `id` a pozici určenou souřadnicemi `x` a `y`. Volitelně je možné definovat například barvu, velikost a tvar vrcholu, nápis na vrcholu a parametry nápisu.

Každé hraně je také nutné přidělit unikátní hodnotu `id` a poté pomocí vlastností `from` a `to` uvést `id` vrcholů, které má spojovat. Hranám je také možné přiřadit nápis. Dále lze ovlivňovat například jejich barvu, tloušťku, styl a případně zakřivení formou Bézierovy křivky. Zakřivení je užitečné zejména při sestrojení více hran mezi dvěma vrcholy, jak je vidět na obrázku 16 v následujícím textu. Směr hrany daný vlastnostmi `from` a `to` je důležitý při použití šipek.

```
{
  nodes: [
    { id: 1, x: -180, y: -40 },
    { id: 2, x: -40, y: -100 },
    { id: 3, x: -30, y: 50 },
    { id: 4, x: 110, y: -50 },
    { id: 5, x: 120, y: 80 },
  ],
  edges: [
    { id: 1, from: 1, to: 2 },
    { id: 2, from: 2, to: 3 },
    { id: 3, from: 2, to: 4 },
    { id: 4, from: 3, to: 5 },
    { id: 5, from: 4, to: 5 },
  ],
}
```

Ukázka kódu 14 Objekt graph definující seznam vrcholů a hran grafu [vlastní zpracování]

Výchozí nastavení vizualizací, které je primárně použito pro vizualizace u důkazů, lze vidět ve zdrojovém kódu v příloze č. 2. Kromě výchozí podoby přidávaných vrcholů a hran jsou zde provedeny předvolby interaktivity grafu tak, aby uživatel mohl volně přesunovat vrcholy, zvýrazňovat vrcholy a hrany, a ovládat přibližování, oddalování a posun kamery, k čemuž slouží také zelená tlačítka ve spodní části vizualizačního plátna. Lze si povšimnout, že je zde také nastavena lokalizace kontextových nabídek do českého jazyka a předán objekt s překlady, který byl pro tyto účely vytvořen.

6.5 Přechod mezi kroky důkazu

Pro každý důkaz v aplikaci je vytvořen samostatný soubor obsahující deklaraci stateful komponenty. Ve stavu komponenty, definovaném objektem `state`, jsou uloženy informace o podobě grafu prostřednictvím seznamu vrcholů a hran. Díky možnosti měnit stav na základě událostí či interakcí uživatele lze nejen manipulovat s podobou grafu, ale řídit celý průběh důkazu. Každá změna stavu totiž v Reactu vyvolá aktualizaci stavů a překreslení všech komponent, které jsou potomky komponenty, jejíž stav byl změněn.

Jednou z důležitých vlastností stavu je aktuální krok průběhu důkazu daný číselnou hodnotou `currentStep`. V momentě, kdy uživatel klikne na tlačítko pro posun v důkazu o krok dopředu či zpět, je zavolána funkce `nextStep()` resp. `previousStep()`, která mimo jiné změní hodnotu aktuálního kroku.

Hodnota aktuálního kroku má vliv na překreslení panelu kroků důkazu a vykreslení panelu popisu vizualizace a v případě potřeby také panelu definic. Podle hodnoty aktuálního kroku je dále provedena příslušná sekvence funkcí a příkazů, které definují změny ve vizualizaci. V následujícím textu bude jako příklad objasněna ukázka kódu funkce `nextStep()` z přílohy č. 3 na konci dokumentu.

6.5.1 Funkce pro změny podoby vizualizace

Kroky prováděné pro potřebné změny podoby vizualizace jsou definovány formou JavaScriptových arrow funkcí přímo v komponentě věnované příslušnému důkazu. Pomocí těchto funkcí je předávána posloupnost změn vedoucí k nové podobě vizualizace. Pro dodržení immutability není možné upravovat objekt `state` přímo. K tomuto účelu poskytuje React metodu `setState()`, která přijímá jako parametr novou podobu stavu a provede změnu stavu bezpečně. Protože ve funkcionálním programování lze jako parametr funkce předat jinou funkci, je vhodnou praktikou metodě `setState()` předat celou funkci, která teprve na svém výstupu vrací novou podobu stavu. Ve zdrojovém kódu aplikace je pro tyto arrow funkce zavedena konvence pojmenování `stepX()`, kde X značí číslo kroku v dokazování, kterého je při použití funkce dosaženo.

Podle principu deklarativního programování je potřeba se při konstrukci takové funkce soustředit na definici kýžené podoby výsledku, spíše než na implementační postup, který vede k jeho dosažení. Modelujeme tedy, jak má vypadat stav na konci kroku, nebo stavy, ze kterých lze sestavit výsledný stav kroku. Ve zdrojovém kódu se tento princip odráží tím, že pro dosažení žádoucí podoby vizualizace je možné za sebe zařadit i více funkcí, jejichž postupné zpracování zobrazí na plátně požadovanou vizualizaci.

Vícenásobné volání metody `setState()` v reakci na kliknutí na tlačítko posunu v důkazu o krok dopředu či zpět není v Reactu problém, jelikož na konci události vyprodukují všechna volání pouze jedno překreslení komponent. React spojí veškeré úpravy stavu do jedné při zachování jejich pořadí. Tato myšlenka je pro výkon React aplikace stěžejní, protože nedochází k nadbytečným překreslením komponent. Zároveň je zaručeno, že při skládání stavů vizualizace

bude vidět pouze výsledná podoba. Například přesun v důkazu na předchozí krok lze proto poskládat z posloupnosti funkcí předchozích kroků, které vedou k požadované podobě vizualizace.

6.5.2 Připravené operace pro úpravy grafu

Ačkoli lze metodě `setState()` předat pouze vybrané objekty pro změnu jen některých vlastností stavu komponenty, v určitých případech je potřeba vybranou vlastnost stavu pozměnit jen z části. Příkladem je situace, kdy má být obarvena pouze půlka ze všech vrcholů grafu. Při dodržení immutability by pro změnu stavu vrcholů musel být metodě `setState()` předán objekt s definovanou podobou všech vrcholů, i těch, které nebylo potřeba měnit.

Zde nastává prostor pro využití knihovny immutability-helper. Díky jejím metodám lze měnit pouze vybrané vlastnosti původního objektu, přičemž je vrácen nový objekt se zanesenými změnami a původní objekt zůstane nezměněn. Pro pohodlnější provádění úprav grafu jsou na základě metod této knihovny připraveny funkce pro operace s vrcholy a hranami grafu.

```
export function updateNode(nodesState, nodeIndex, background, label) {
  return imHelp(nodesState, {
    [nodeIndex]: {
      color: { $set: { background: background } },
      label: { $set: label },
    }
  });
}
```

Ukázka kódu 15 Příklad operace pro změnu barvy a nápisu vrcholu [vlastní zpracování]

Tyto funkce jsou exportovány ze souboru `graphFunctions.js`, importovány do souboru komponenty každého důkazu a svázány v konstruktoru funkcí `.bind()`, aby mohly být využity k úpravám grafů ve funkcích pro jednotlivé kroky důkazu.

```
step3 = (state) => {
  let newNodes = this.updateNode(state.nodes, 0, palette.orange, 'x');

  return { nodes: newNodes };
};
```

Ukázka kódu 16 Příklad funkce měnící barvu a nápis vrcholu v kroku 3 [vlastní zpracování]

Funkce vracející nový seznam vrcholů je poté předána metodě `setState()` jejíž pomocí provede React efektivně potřebné změny ve stavu komponenty.

6.5.3 Posun posuvníku panelu kroků důkazu

Protože se důkaz může skládat z mnoha kroků a panel kroků důkazu by mohl být příliš dlouhý pro současné sledování vizualizačního plátna, je jeho délka omezena a k přesahujícímu textu se lze dostat posunutím postranního posuvníku. Při procházení kroků důkazu by však nutnost posunovat posuvník mohla být pro uživatele nepohodlná. Tento proces je proto automatizován.

K posunu je použita komponenta `scroller` z volně dostupného balíčku react-scroll [84], která pomocí metody `scrollTo()` dokáže posunout posuvník na začátek komponenty `Element` z téhož balíčku. Použití metody lze vidět v příloze č. 3. Metoda přijímá název komponenty `Element`, na kterou je požadováno posunout posuvník, formou `props` označené `name`, a objekt nastavení způsobu posunutí. Že jsou bloky textů v panelu kroků důkazu tvořeny komponentami `Element` a mají přiděleno `name`, bylo možné si všimnout v ukázce kódu 12. Objekt nastavení způsobu posunu se pak skládá například z vlastnosti `duration` pro změnu délky trvání posunu, `containerId` pro předání identifikátoru elementu s posuvníkem, a `offset` pro dodatečný posun o zadané množství pixelů. Pro jeho generování je připravena funkce `getScrollOptions()` z následující ukázky kódu 17.

```
export function getScrollOptions(windowScrollY) {
  return {
    duration: 800,
    delay: 0,
    smooth: 'linear',
    containerId: 'proof-steps-box',
    offset: (-230 + windowScrollY),
    // Prevent canceling of scroll by fast switching between steps
    ignoreCancelEvents: true,
  }
}
```

Ukázka kódu 17 Nastavení posunu posuvníku panelu kroků důkazu [vlastní zpracování]

Při implementaci se ukázalo, že posun celé stránky ve webovém prohlížeči způsobuje nepřesnost v posunu posuvníku panelu kroků důkazu. K řešení tohoto problému je vytvořena právě funkce `getScrollOptions()`, která přijímá hodnotu aktuálního posunu posuvníku celého webového prohlížeče. Tato hodnota je poté přičtena ve vlastnosti `offset` pro korekci nepřesnosti v posunu.

6.5.4 Pohyb kamery

Jednou z věcí, která dodává vizualizacím důkazů dynamiku, je pohyb kamerou. Ten lze využít k mnoha účelům. Umožňuje oddálení v momentě, kdy je graf rozšířen o další vrcholy, které by přesahovaly velikost plátna, ale také přiblížení na část grafu, na kterou se student má v daném kroku důkazu soustředit.

Pro pohyb kamerou je využívána metoda `moveTo()` aplikovatelná na instanci objektu `network` vizualizačního plátna. Tu lze získat z `props` komponenty `GraphVis`. K její propagaci zpět do komponenty důkazu je implementována funkce `initNetworkInstance()`.

Instance objektu `network` je poté přístupná v celé komponentě důkazu. Pro účely pohybu kamery je metoda `moveTo()` využívána nejčastěji po provedení změn stavu komponenty v daném kroku, jak lze vidět také v příloze č. 3. Metoda `moveTo()` přijímá jako parametr objekt definující novou pozici kamery a způsob animace k jejímu dosažení.

```
const cameraPosition1 = {
  position: { x: 170, y: -10 },
  scale: 1.4,
  animation: { duration: 1000, easingFunction: 'easeInOutQuad' },
};
```

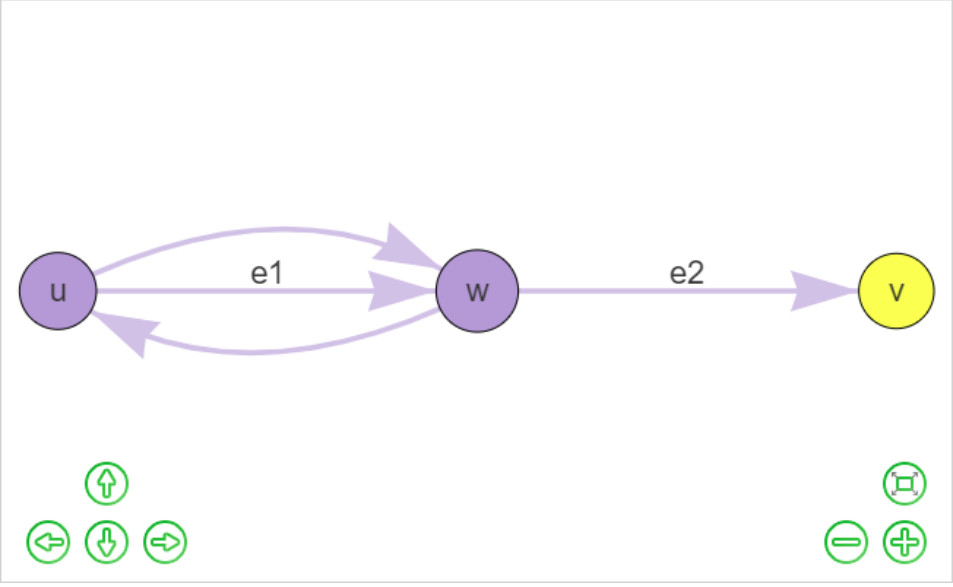
Ukázka kódu 18 Objekt definující animaci na novou pozici kamery [vlastní zpracování]

Tyto objekty jsou do komponenty důkazu importovány společně s doprovodnými texty z externího souboru `constants.jsx`.

6.5.5 Animace vizualizace

Pro lepší názornost při vysvětlování průběhu důkazu je v některých případech vhodné použít vícekrokovou animaci. K těmto účelům je v aplikaci využíváno JavaScriptových funkcí `setTimeout()` a `setInterval()`.

Jak lze vidět opět v příloze č. 3, ve druhém kroku důkazu je nastaven interval pro periodické spouštění funkce `step2Animation()`. Tato funkce zahrnuje příkazy pro nastavení prodlev mezi spouštěním funkcí pro úpravy grafu. V určitém časovém intervalu je tedy spuštěna funkce, která v nastavených časových úsecích upravuje graf, čímž je přehrávána animace. Reference na spuštěné časovače je ukládána do stavu komponenty. Při přechodu na následující krok důkazu je zavolána funkce `clearAllTimers()`, která byla implementována za účelem přerušit všechny časovače a vymazat jejich reference.



Obrázek 16 Snímek z průběhu vícekrokové animace konstrukce sledu [vlastní zpracování]

Pomocí animací je možné například v jednom kroku důkazu ilustrovat vícenásobný průchod grafem přes stejné vrcholy a hrany. Ke znázornění lze použít například barvy a šipky, tak jako tomu je na obrázku 16, zachycujícím konstrukci sledu $S = (u, e_1, w, e_1, u, e_1, w, e_2, v)$.

V panelu ovládání je dále implementováno tlačítko pro okamžité opakování animace, které studenti mohou použít v případě, že chtějí animaci přerušit a zahájit od začátku.

6.6 Animovaný text na vizualizačním plátnu

Zejména pro metody nepřímého dokazování a důkazy sporem je vhodné v důkazu důkladněji rozebrat sestavení obměn a negací původního tvrzení. Aby pro studenty byl tento rozbor co nejsnáze pochopitelný, je nutné jej rozdělit do mnoha kroků. K tomuto účelu dobře slouží animace rozboru znění tvrzení přímo na vizualizačním plátnu.

Animování textu na vizualizačním plátnu je dosaženo pomocí CSS přechodů z knihovny react-css-transition-replace [77] a knihovny react-transition-group [86], která byla z Reactu oddělena do formy samostatného npm balíčku. Pro umístění řádku textu na plátno je implementována komponenta `VisualTextRow`. Pro každý důkaz jsou řádky sdruženy v poli objektů v příslušném souboru `constants.js`, aby i tyto texty byly odděleny od logiky vizualizace grafu v důkazu. U každého řádku je definováno, ve kterých krocích důkazu má být na plátně viditelný. Pole řádků je předáno komponentě `VisualTextsPanel`, která je implementována tak, že rozpoznává, který řádek má být v daném kroku viditelný.

Prostřednictvím komponent `TransitionGroup` a `CSSTransition` z animační knihovny sama provádí animaci při přidávání a odebírání textových řádků v průběhu procházení kroků důkazu.

```
const VisualTextsPanel = (props) => (
  <MathJax.Context input="tex">
    <div className="vis-texts-panel">
      <TransitionGroup>
        {props.visualTextRows
        .filter(
          textRow => textRow.showForSteps.includes(props.currentStep)
        )
        .map(({ id, content }) => (
          <CSSTransition key={id} timeout={500} classNames="row">
            {content}
          </CSSTransition>
        ))}
      </TransitionGroup>
    </div>
  </MathJax.Context>
);
```

Ukázka kódu 19 Úryvek komponenty pro animaci textů na plátně [vlastní zpracování]

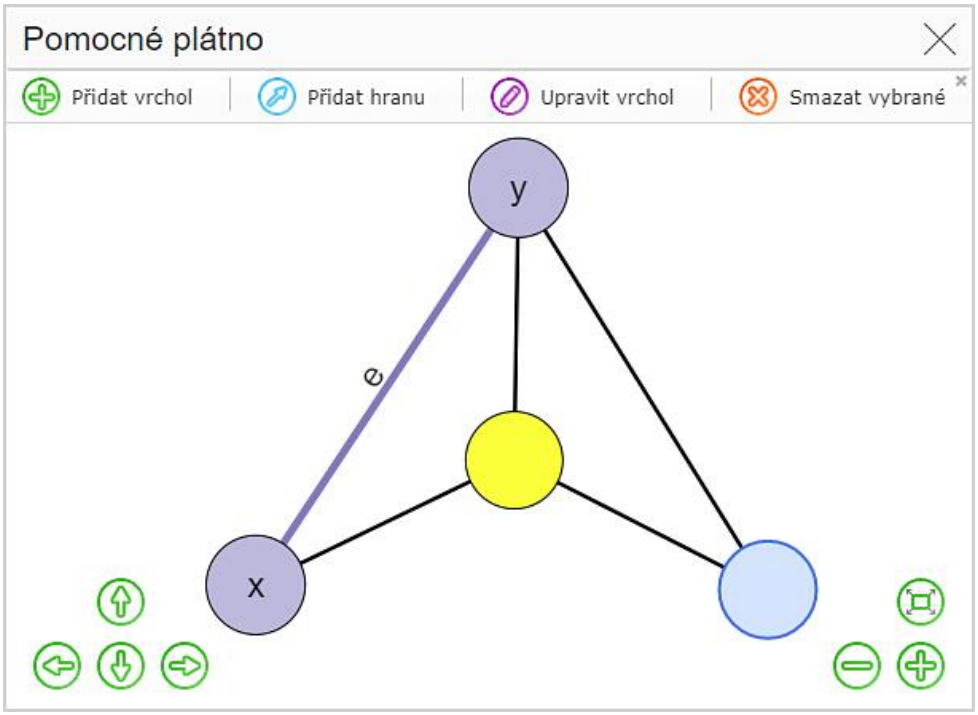
6.7 Kreslící nástroje

Pro účely výuky byla jedním z požadavků na aplikaci možnost volně kreslit přes vizualizaci tak, aby vyučující mohl znázornit doplňující poznámky. Pro tyto účely je v panelu ovládání implementována skupina tlačítek kreslících nástrojů. Při aktivaci kreslení je pomocí CSS před vizualizaci do popředí umístěna průhledná vrstva. Tím je zároveň zamezeno ovlivňovat kurzorem myši rozmístění vrcholů pohybovat s kamerou nebo jinak interagovat s vizualizací. Při kreslení přes plátno tedy nemůže dojít k nežádoucím změnám ve vizualizaci.

Pro implementaci kreslících nástrojů je využit balíček react-sketch [85] volně dostupný pod licencí MIT. Ve výchozím stavu je po aktivaci kreslení kurzor myši změněn na kříž a je vybrán nástroj tužka pro kreslení přes plátno volným tahem. Zpřístupněny jsou nástroje kruh a čára. Volba nástroje je zaznamenávána do stavu komponenty a podle toho je implementovanými metodami předán příslušný objekt nástroje komponentě `SketchField` ze zmiňovaného balíčku. Taktéž deaktivace kreslení je zaznamenávána do stavu komponenty a projeví se promazáním a skrytím kreslící vrstvy.

6.8 Prázdná tvůrčí plátna

Dalším z nástrojů implementovaných pro účel výuky jsou pomocná tvůrčí plátna. V nich má vyučující možnost sám sestrojovat vlastní grafy přidáváním vrcholů a jejich propojováním hranami.



Obrázek 17 Konstrukce vlastního grafu v malém tvůrčím plátnu [vlastní zpracování]

Požadována byla možnost zobrazit na stránce důkazu malé pomocné plátno, které by umožnilo vyučujícímu doplnit výklad důkazu příkladem na jeho vlastním grafu. K tomuto účelu je do aplikace implementována komponenta dialogu ve formě nemodálního okna z volně dostupného npm balíčku react-dialog [78]. Jeho podobu lze vidět na obrázku 17. Tento dialog je možné uchopením za horní panel volně přesunovat v prostoru celé stránky a po jeho zavření křížkem v pravém horním rohu je jeho obsah vymazán.

Do dialogu je vložena stateful komponenta `DrawingDialog`, která zahrnuje implementaci plátna s unikátním nastavením a modálních oken pro úpravu atributů vrcholů a hran. Obdobou je komponenta `SingleDrawing`, která slouží k zobrazení prázdného tvůrčího plátna přes celou stránku. Toto velké plátno je užitečné v případě, kdy vyučující chce pro konstrukci grafu využít celý prostor prohlížeče.

Pro tvorbu a úpravy grafu slouží nabídka s tlačítky v horní části plátna. Ta slouží v průběhu tvorby zároveň jako nápověda tím, že například popisuje uživateli, jakým způsobem umístí nový vrchol do plátna nebo jak propojí dva vrcholy hranou.

Po označení vrcholu se v nabídce zobrazí tlačítko pro úpravu jeho atributů. Pro tyto účely byly do manipulačního modulu plátna doimplementovány funkce obsluhující modální okno tvořené stateless komponentou `EditNodeDialog`. Tyto funkce jsou umístěny v externím souboru `editNodeFunctions.js`, jehož zjednodušený úryvek lze najít v příloze č. 4, a zajišťují zobrazení a skrytí modálního okna, předvyplnění hodnot atributů vrcholu do formuláře a možnost uložení úprav atributů. Uživatel má možnost změnit textový popis vrcholu, barvu textu a barvu výplně vrcholu.

Obrázek 18 Modální okno pro úpravu atributů vrcholu [vlastní zpracování]

Obdobně je implementováno také modální okno pro úpravu atributů hran. Po označení hrany je v nabídce zobrazeno tlačítko pro úpravu hrany. Nejprve má uživatel možnost přetáhnout konec hrany pro její propojení s jiným vrcholem, poté je zobrazeno modální okno s možností úpravy textového popisu, barvy a tloušťky hrany.

6.9 Verzování zdrojového kódu a zprovoznění na GitHub Pages

V rámci vývoje aplikace je využíván systém správy verzí Git a na webové službě GitHub byl založen repositář s názvem „proof-vis“ publikující veškeré změny prováděné ve zdrojovém kódu aplikace.

Repositář je veřejně dostupný na adrese https://github.com/Skoreto/proof-vis. Aktuální verze je udržována ve větvi `default`. Zdrojový kód je také přiložen na CD.

Pro účely publikování aplikace je využíván webhosting GitHub Pages, který GitHub nabízí pro open source projekty zdarma. Pro JavaScriptovou single page aplikaci je webhosting podporující soubory typu HTML, CSS a JS dostačující. Ošetření funkčnosti routování bylo popsáno v podkapitole 6.3.2 a konfigurace publikování je definována v souboru `package.json` v hlavním adresáři projektu. K publikaci buildu sestaveného z větve `default` využívají GitHub Pages větev `gh-pages` ze stejného repositáře.

Aplikace je veřejně dostupná na adrese https://skoreto.github.io/proof-vis/.

6.10 Uživatelská příručka

Následující podkapitoly poskytují seznámení s aplikací formou uživatelské příručky. Tato příručka je také standardní součástí aplikace.

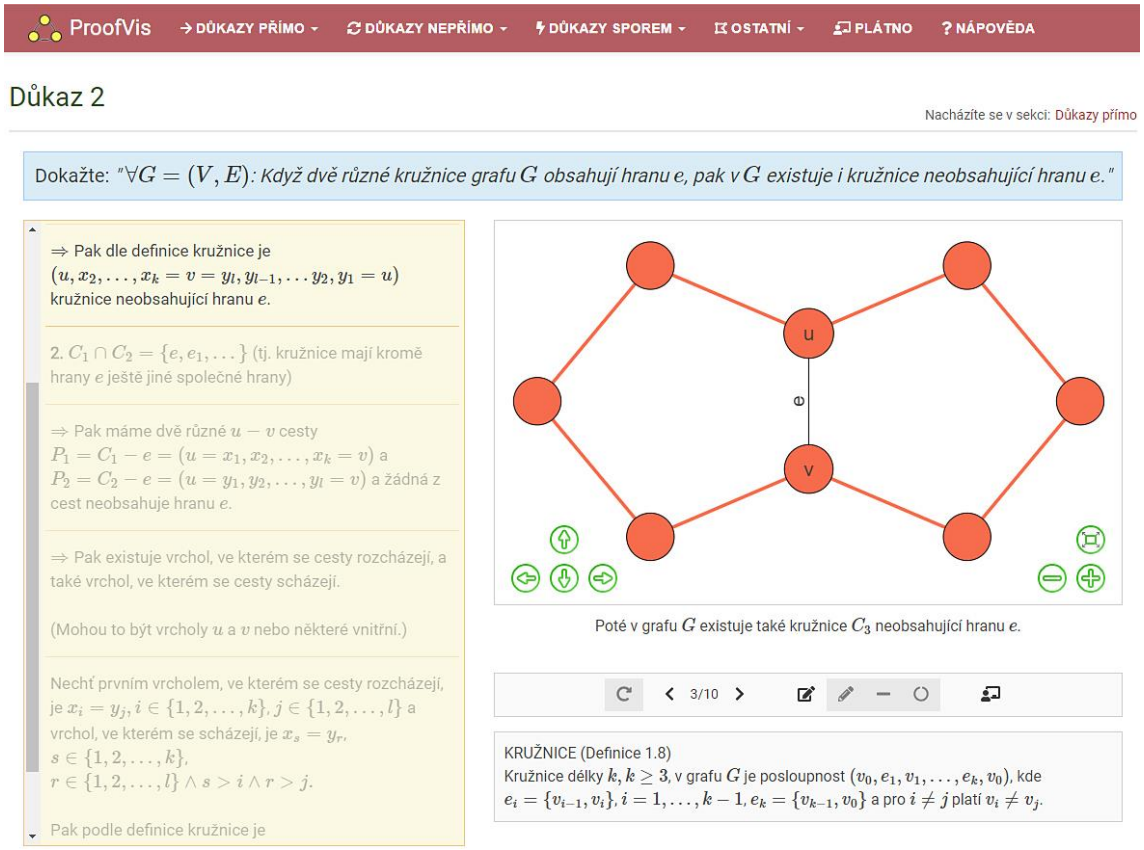
6.10.1 Požadavky na provoz aplikace

Vývoj aplikace byl cílen na použití v nových verzích majoritních zástupců webových desktopových a mobilních prohlížečů. Z desktopových variant jsou doporučeny prohlížeče Google Chrome a Mozilla Firefox, ale podporován je také prohlížeč Opera, Microsoft Edge a Safari. Na mobilních zařízeních je doporučen prohlížeč Google Chrome. Pro pohodlné používání aplikace by úhlopříčka displeje zařízení měla dosahovat alespoň rozměrů tabletu s rozlišením alespoň 1024x768 pixelů. Uživatelské rozhraní se však přizpůsobí i displejům chytrých telefonů.

6.10.2 Orientace v aplikaci

V horní části stránek je umístěna horizontální navigace obsahující rozbalitelné nabídky s odkazy na stránky důkazů. Důkazy jsou rozděleny do nabídek

podle metody dokazování, která v nich je využita. Navigace je dostupná z každé stránky a je proto možné ihned přejít na kterýkoli z důkazů. V navigaci se dále nachází odkaz na stránku s velkým plátnem pro tvorbu vlastního grafu a odkaz na stránku s nápovědou. Přechod na úvodní stranu je proveden po kliknutí na logo s názvem aplikace.



Obrázek 19 Výřez z celkového pohledu na stránku důkazu [vlastní zpracování]

6.10.3 Popis hlavních prvků stránky důkazu

Rozvržení těla každé stránky důkazu zahrnuje v horní části název důkazu a drobečkovou navigaci pro usnadnění orientaci v aplikaci. Dále široký panel se zněním dokazovaného tvrzení, levý postranní panel s kroky prováděnými v průběhu dokazování, vizualizační plátno s popisem aktuální vizualizace, panel ovládání a v případě potřeby šedý panel definic, který slouží k připomenutí znění důležitých definic použitých při dokazování.

6.10.4 Ovládání vizualizačního plátna

Součástí plátna jsou zelená tlačítka pro ovládání kamery. Šipky v levém dolním rohu slouží k posunu po ose x a y. Tlačítka + a – v pravém dolním rohu umožňují pohled kamery přiblížit a oddálit, což lze provést také otáčením kolečka myši. Tlačítko roztahovaného čtverce vycentruje graf na střed a resetuje oddálení kamery tak, aby byl komfortně zobrazen celý aktuální obsah plátna.

S grafem na plátnu je možné interagovat. Vrcholy a hrany reagují na přejetí kurzorem myši a stisknutím levého tlačítka myši je možné je zvýrazňovat. Jejich zvýraznění zůstane permanentní, pokud je levé tlačítko podrženo déle než jednu sekundu nebo pokud je při klikání podržena klávesa CTRL.

V daném kroku lze vrcholy přesunout na jinou pozici.

6.10.5 Panel ovládání

V šedém panelu ovládání se nacházejí tlačítka pro ovládání průběhu důkazu a pomocných nástrojů pro výuku.

Obrázek 20 Panel ovládání [vlastní zpracování]

Černými šipkami vlevo a vpravo se lze pohybovat v důkazu o krok zpět či vpřed. Aktuální krok z celkového počtu kroků zobrazuje ukazadlo mezi šipkami. V průběhu vícekrokových animací je aktivní tlačítko „šipky zopakovat“, které slouží k okamžitému zopakování animace od začátku.

6.10.6 Kreslící nástroje

Panel ovládání dále obsahuje skupinu tlačítek pro kreslení přes plátno. Po aktivaci kreslení prvním tlačítkem „editace“ je kurzor změněn na kříž a jsou zpřístupněny tlačítka pro výběr kreslícího nástroje. Ve výchozím stavu je aktivní nástroj „tužka“, který umožní kreslit přes plátno volným tahem. Další dva nástroje umožní kreslení přímek a kružnic například pro naznačení vrcholů a hran grafu. Deaktivací kreslení opětovným stisknutím tlačítka „editace“ je kresba vymazána.

6.10.7 Tvůrčí plátna

Pro účely výuky byly dále do aplikace přidány prázdná tvůrčí plátna, ve kterých může vyučující sám sestrojovat grafy přidáváním vrcholů a jejich propojováním hranami. Na stránku velkého plátna lze přejít tlačítkem „PLÁTNO“ v hlavní navigaci.

Na stránce každého důkazu je v panelu ovládání dostupné tlačítko pro zobrazení malého pomocného plátna z obrázku 17. Plátno lze volně přesouvat v prostoru celé stránky uchopením za horní panel a po zavření dialogu křížkem v pravém horním rohu je jeho obsah vymazán.

Pro tvorbu a úpravy grafu slouží tlačítka z nabídky v horní části plátna. Po jejich zvolení se v nabídce zobrazí nápověda k provedení požadovaného úkonu. Po označení vrcholu je v nabídce zobrazeno také tlačítko „Upravit vrchol“ po jehož stisknutí se zobrazí dialog z obrázku 18, ve kterém lze změnit text popisu vrcholu, barvu textu a barvu vrcholu. Obdobně je možné upravit také textový popis, barvu a tloušťku hrany. Nejprve je možné propojit hranu s jiným vrcholem, poté se zobrazí formulář pro úpravu zmíněných atributů.

6.10.8 Řešení problémů

V případě výskytu chyby či špatného vykreslení některého z prvků je doporučeno přejít na úvodní stránku a obnovit ji například klávesou F5.

6.11 Testování

V průběhu zpracování projektu probíhalo testování jak z hlediska vývoje pro odhalení chyb, tak testování uživatelské pro získání zpětné vazby od studentů a zadavatelky práce.

6.11.1 Unit testy

Přímo ve složkách komponent, kterých se dané testování týká, jsou umístěny soubory s příponou `.test.js`. Ty jsou automaticky rozpoznány Reactem a přidány do testování. Díky kombinaci nástrojů Jest a Enzyme je možné testovat vykreslování komponent. Toho je využito zejména pro testování komponent panelů,

jako jsou `DescriptionPanel`, `DefinitionPanel` a `ProofStepsBox`, kterým jsou předávána pole s objekty, jejichž vykreslování je podmíněné aktuálním krokem v důkazu.

Ukázku jednoho z testů lze vidět v příloze č. 5. V testovém bloku je nejprve metodou `.shallow()` získána komponenta `DefinitionPanel` bez vykreslování jejích potomků a v samotném testu jsou jí nastaveny `props`. Na konci testu je očekáváno, že v kroku 4 bude vykreslena definice, která je pro tento krok určena, a že definice pro kok 2 a 3 naopak vykresleny nebudou. Pokud v průběhu vývoje nedošlo k poškození komponenty, proběhne test bez ohlášení chyby.

6.11.2 Uživatelské testování

Již v průběhu vývoje byli oslovováni studenti předmětu DIMA a KDIMA za účelem získání zpětné vazby. Oslovení studenti měli možnost na vlastních zařízeních otestovat funkčnost aplikace a přinést návrhy na vylepšení. Studentům byly předkládány důkazy v různých nastaveních a způsobech zpracování tak, aby bylo možné poznat jejich preference. Na základě podnětů a připomínek byla aplikace vylepšována.

Další forma testování se studenty a vyučujícím proběhla na konci výuky přímo v učebně na Fakultě informatiky a managementu s využitím projektoru. Prozkoumáno bylo výsledné zobrazení a viditelnost prvků na projektoru i použitelnost aplikace ve výukovém prostředí. Takto bylo možné získat zpětnou vazbu jak z domácího tak školního prostředí.

Testování například odhalilo, že většina studentů preferuje umístění vizualizačního plátna na pravé straně aplikace, nebo pomohlo vybrat ikony, které z pohledu studentů nejlépe vystihují své funkce. Při zkoumání vícekrokové animace konstrukce sledu z obrázku 16, byla připravena varianta využívající ztmavování barev hran a vrcholů při jejich opětovném procházení a varianta zachycená na obrázku 16, která k naznačení postupu využívá šipek. Ačkoli by šipky mohly v grafu evokovat multihrany, protože byla podoba grafu jasně známa z předchozích kroků důkazu, studenti takřka bez výjimky upřednostňovali právě variantu se šipkami.

Ve výsledku byla zpětná vazba od studentů pozitivní, někteří aplikaci údajně již využili při přípravě na zkouškové termíny. Studenti se vyjadřovali i k funkcím cíleným především pro učitele a na aplikaci oceňují zejména:

- snadnou dostupnost a zprovoznění aplikace ve webovém prohlížeči bez nutnosti instalace
- intuitivnost ovládání
- popis průběhu důkazu přímo u vizualizace usnadňující pochopení důkazu
- dynamičnost vizualizací a interaktivnost
- možnost tvorby vlastních grafů a kreslení přes vizualizaci

Dále byla uvedena například výhoda v ucelenosti souboru důkazů, kdy není nutné zanášet PC stahováním souborů PowerPointových prezentací pro jednotlivé důkazy. Oproti zpracování formou videa, které by bylo nutné pozastavovat, bylo vyzdvihnuto krokování, díky kterému se student může snadno nad krokem pozastavit a případně se vrátit k předchozímu.

Při některých rozhodnutích nepanovala u studentů jednomyslná shoda, nicméně u nejednoznačných voleb vždy preference jedné z variant výrazně převažovala ty ostatní. V průběhu vývoje bylo studenty dále zmiňováno, že by uvítali více podobně zpracovaných důkazů.

Testování poskytlo cenné informace pro eliminaci problémů a určovalo správný směr vývoje. Zejména uživatelské testování pak mělo zásadní dopad na výsledné zpracování některých prvků aplikace.

7 Výsledky

V diplomové práci byly prozkoumány možnosti podpory výuky dokazování v oblasti teorie grafů. Jelikož je otázky teorie grafů často možné graficky reprezentovat, důraz byl kladen na zkoumání podpory výuky dokazování vizualizacemi. Pro zatraktivnění důkazů a usnadnění jejich pochopitelnosti byly prostudovány teoretické aspekty tvorby dynamických a interaktivních vizualizací.

Na základě těchto zjištění a nároků kladených zadavatelkou projektu pro účely školní výuky byly stanoveny požadavky na zpracování podpůrné výukové aplikace. Ukázalo se, že nejvhodnější formou zpracování bude single page aplikace, spustitelná bez nutnosti instalace z majoritních zástupců webových prohlížečů na různých druzích koncových zařízení.

Především pro své inovativní postupy a širokou nabídku open source nástrojů a rozšiřujících balíčků od početné komunity byla pro účely implementace zvolena komponentově orientovaná knihovna React. Při implementaci byly využity například principy funkcionálního programování, novodobá syntaxe JavaScriptu ve specifikaci ES6 a její transformace do podoby srozumitelné prohlížečům nástrojem Babel.

Již v průběhu vývoje byla aplikace pravidelně testována a byla získávána zpětná vazba od studentů a zadavatelky projektu jak z domácího tak školního prostředí, aby co nejlépe naplnila jejich potřeby. Takto se podařilo implementovat komponenty a funkce pro prezentaci matematických důkazů doprovázených dynamickými a interaktivními vizualizacemi. Rozšíření jejich počtu je tedy pouze otázkou použití stejných postupů. Aplikace je vhodná pro samostudium, ale disponuje také řadou nástrojů využitelných ve vyučování, jako jsou kreslící nástroje a prázdná tvůrčí plátna, ve kterých může vyučující konstruovat vlastní grafy a podpořit tak svůj dodatečný výklad.

Tím byly splněny stanovené požadavky na aplikaci a naplněn cíl diplomové práce.

8 Závěr a doporučení

Nepopiratelnou důležitost dokazování v matematice dokládá řada studií a publikací. Pozornost je věnována také snahám zatraktivnit pro studenty dokazování ve výuce a hledáním nových způsobů, jak porozumění důkazům usnadnit.

Ukazuje se, že moderní technologie mají potenciál do řešení této problematiky přispět. Výukové materiály zpracované formou single page aplikace s vizualizačním plátnem přinášejí v souvislosti s výukou důkazů řadu výhod oproti jiným postupům. V porovnání se zpracováním formou videa umožňují studentovi při procházení kroků důkazu kdykoli se pozastavit či vrátit, aby si mohl předchozí provedené kroky znovu ujasnit. Oproti statickým obrázkům je dynamika vizualizace užitečná při tvorbě animací, které zpřehledňují a ulehčují pochopení prováděných kroků. Interaktivita vizualizace společně s funkcemi kreslení a konstrukce vlastních grafů jsou pak cennými nástroji při využití ve školní výuce. Rozšíření Internetu a zařízení s webovým prohlížečem zaručuje snadnou dostupnost materiálů bez nutnosti instalací a v ucelené podobě.

Na realizovaném projektu bylo těchto aspektů dosaženo využitím řady moderních technologií a inovativních postupů. Využita byla knihovna React, jejíž popularita v rychle se měnící oblasti webových technologií předvídá stabilní budoucnost a důraz na flexibilitu umožňuje snadné rozšíření aplikace o nové funkce. Zajímavým rozšířením ke konstrukci grafů by byla možnost přidávání textů a vytváření celých vlastních důkazů. Rešerše naznačila, že také přidání překladů by mohlo najít své uplatnění, jelikož i ve světovém měřítku je podobných aplikací nedostatek.

Ze zpětné vazby vyplynulo, že studenti vznik aplikace oceňují a mají o používání výukových materiálů v podobné formě zájem.

9 Seznam použité literatury

9.1 Literární zdroje

[1] ANDERSON, Ian. A first course in discrete mathematics. London: Springer, c2002. ISBN 1852332360.

[2] ANTONIO, Cássio de Sousa. Pro React. Berkeley, Calif.: Apress, 2015. ISBN 978-1-4842-1261-5.

[3] ARAVINTH, Anto. Beginning functional JavaScript: functional programming with JavaScript using EcmaScript 6. Berkeley, California: Apress, 2017. Books for professionals by professionals. ISBN 978-1-4842-2655-1.

[4] ARLOW, Jim a Ila NEUSTADT. UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky. 2. aktual. a dopln. vyd. Přeložil Bogdan KISZKA. Brno: Computer Press, 2011. ISBN 978-80-251-1503-9.

[5] BACKFIELD, Joshua. Becoming functional. United States of America. Sebastopol: O'Reilly, 2014. ISBN 978-1-449-36817-3.

[6] BALADA, Jan. Rámcový vzdělávací program pro gymnázia: RVP G. Praha: Výzkumný ústav pedagogický v Praze, c2007. ISBN 978-80-87000-11-3. Dostupné také z: http://www.msmt.cz/file/10427_1_1

[7] BANKS, Alex a Eve PORCELLO. Learning React: functional web development with React and Redux. Sebastopol, CA: O'Reilly Media, 2017. ISBN 978-1-491-95462-1.

[8] BENDOVÁ, Háňa. Matematická indukce. In: Matematický korespondenční seminář [online]. Praha: Matematicko-fyzikální fakulta UK, 2007, 2007, s. 13-14 [cit. 2018-07-27]. Dostupné z: https://mks.mff.cuni.cz/library/MatematickaIndukceHB/MatematickaIndukceHB.pdf

[9] BIGGS, Norman. Discrete mathematics. 2nd ed. New York: Oxford University Press, 2002. ISBN 978-0-19-850718-5.

[10] BONDY, J. A. a U. S. R. MURTY. Graph theory with applications. Fifth Printing. New York: North Holland, 1982. ISBN 04-441-9451-7.

[11] BORBA, Marcelo C. a Hans-Georg WEIGAND. Analysis of Uses of Technology in the Learning of Mathematics. In: CHO, Sung Je. The Proceedings of the 12th International Congress on Mathematical Education. Cham: Springer International Publishing, 2015, 2015-2-11, s. 479-483. DOI: 10.1007/978-3-319-12688-3_44. ISBN 978-3-319-10685-4. Dostupné také z: http://link.springer.com/10.1007/978-3-319-12688-3_44

[12] BRATH, Richard. Graph analysis and visualization. Indianapolis, IN: John Wiley, 2015. ISBN 978-1-118-84584-4.

[13] BROWN, James Robert. Philosophy of mathematics: a contemporary introduction to the world of proofs and pictures. 2nd ed. New York: Routledge, 2008. ISBN 978-0415960472.

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

[15] ČECH, Vlastimil. Proč děláme důkazy v matematice. Praha: SPN - pedagogické nakladatelství, 1971.

[16] DAUGHERTY, Sarah C. A Study of Visualization for Mathematics Education. In: PINELLI, Thomas, Shannon SULLIVAN a Alicia SANCHEZ, ed. MODSIM World 2007 Conference and Expo: Select Papers and Presentations from the Education and Training Track. VA, United States: Old Dominion Univ., 2008, s. 521-526. Dostupné také z: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20080040970.pdf

[17] DEBRENTI, Edith. Visual Representations in Mathematics Teaching: An Experimtnt with Students. In: Acta Didactica Napocensia. Volume 8. Partium Christian University Oradea (Romania), 2015, s. 21-26. ISSN 2065-1430.

[18] DEMEL, Jiří. Grafy a jejich aplikace. Praha: Academia, 2002. ISBN 8020009906.

[19] EISENBERG, Theodore a Tommy DREYFUS. On the reluctance to visualize in mathematics. In: ZIMMERMANN, Walter a Steve CUNNINGHAM. Visualization in teaching and learning mathematics. Washington, DC, USA: Mathematical Association of America, 1991, s. 25-37. ISBN 0-88385-071-0.

[20] EULER, Leonhard. Solutio problematis ad geometriam situs pertinentis. Commentarii Academiae Scientiarum Imperialis Petropolitanae. 1736, 8, 128-140.

[21] FEDOSEJEV, Artemij. React.js Essentials: A fast-paced guide to designing and building scalable and maintainable web apps with React.js. Birmingham, UK.: Packt Publishing, 2015. ISBN 978-1-78355-162-0.

[22] HABALA, Petr. Diskrétní matematika: Indukce a rekurze [online]. Praha: ČVUT, 2012 [cit. 2018-06-02]. Dostupné z: https://math.feld.cvut.cz/habala/teaching/dma/dmknih05.pdf

[23] HLINĚNÝ, Petr. Úvod do Informatiky [online]. Verze 1.20. Masarykova Univerzita, 2010 [cit. 2018-06-02]. Dostupné z: https://is.muni.cz/do/1499/el/estud/fi/js10/uinf/web/UInf-text10.pdf

[24] HORTON, Adam a Ryan VICE. Mastering React: Master the art of building modern web applications using React. Birmingham: Packt Publishing, 2016. ISBN 978-1-78355-856-8.

[25] CHINNATHAMBI, Kirupa. Learning react: a hands-on guide to building web applications using react and redux. 2nd edition. Indianapolis, IN: Addison-Wesley Professional, 2018. ISBN 978-0-13-484355-1.

[26] ILIINSKY, Noah, STEELE, Julie, ed. Designing data visualizations. Sebastopol, CA: O'Reilly, 2011. ISBN 9781449312282.

[27] KADUNZ, Gert a Michal YERUSHALMY. Visualization in the Teaching and Learning of Mathematics. In: CHO, Sung Je. The Proceedings of the 12th International Congress on Mathematical Education. Cham: Springer International Publishing, 2015, s. 463-465. DOI: 10.1007/978-3-319-12688-3. ISBN 978-3-319-10685-4.

[28] KÖNIG, Dénes. Theorie der endlichen und unendlichen Graphen. Leipzig: Akademische Verlagsgesellschaft m. b. h., 1936.

[29] LANUM, Corey. Visualizing graph data. Version 8. New York: Manning, 2017. ISBN 9781617293078.

[30] LEVIN, Oscar. Discrete Mathematics: An Open Introduction. 2nd Edition. Colorado: University of Northern Colorado, 2017. ISBN 978-1534970748.

[31] MANTYLA, Dan. Functional Programming in JavaScript. Birmingham: Packt Publishing, 2015. ISBN 978-1-78439-822-4.

[32] MATOUŠEK, Jiří a Jaroslav NEŠETŘIL. Kapitoly z diskrétní matematiky. 4., upr. a dopl. vyd. V Praze: Karolinum, 2009. ISBN 978-80-246-1740-4.

[33] MEZZALIRA, Luca. Front-end reactive architectures. New York, NY: Apress, 2018. ISBN 978-1-4842-3179-1.

[34] MILKOVÁ, Eva. Teorie grafů a grafové algoritmy. Hradec Králové: Gaudeamus, 2013. ISBN 9788074352676.

[35] MILLER, Robin. On Proofs Without Words [online]. Whitman College, 2012, 14.5.2012, 1-4 [cit. 2018-06-19]. Dostupné z: https://www.whitman.edu/Documents/Academics/Mathematics/Miller.pdf

[36] NELSEN, Roger B. Proofs without words: exercises in visual thinking. Washington, D.C.: The Mathematical Association of America, c1993. ISBN 0-88385-700-6.

[37] NELSEN, Roger B. Proofs without words II: more exercises in visual thinking. Washington, DC: Mathematical Association of America, c2000. Classroom resource materials (Unnumbered). ISBN 0-88385-721-9.

[38] ODELL, Den. Pro JavaScript development: coding, capabilities, and tooling. Berkeley: Apress, [2014]. ISBN 978-1-4302-6268-8.

[39] PALAIS, Richard. The visualization of mathematics: Towards a mathematical exploratorium. Notices of the American Mathematical Society. In: Notices of the American Mathematical Society [online]. 1999, July 1999, s. 647-658 [cit. 2018-08-14]. Volume 46, Number 6. Dostupné z: https://researchgate.net/publication/216874406_The_Visualization_of_Mathematics_Towards_a_Mathematical_Exploratorium

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

[41] PRUSTY, Narayan. Modern JavaScript Applications: An example-driven guide that explores the world of modern web development with JavaScript. Birmingham: Packt Publishing, 2016. ISBN 978-1-78588-144-2.

[42] RAHIM, Medhat a Radcliffe SIDDO. The use of visualization for learning and teaching mathematics. In: Proceedings of the 10th International Conference: The Mathematics Education into the 21st Century Project [online]. Dresden, Saxony, Germany, 2009, September 2009, pp. 496-497 [cit. 2018-06-20]. Dostupné z: http://math.unipa.it/~grim/21_project/Rahim496-500.pdf

[43] ROSEN, Kenneth H. Discrete mathematics and its applications. 7th ed. New York: McGraw-Hill, c2012. ISBN 00-733-8309-0.

[44] RÖSKEN, Bettina a Katrin ROLKA. A Picture is Worth a 1000 Words: The Role of Visualization in Mathematics Learning. In: KRÁTKÁ, M. a N. STEHLÍKOVÁ, NOVOTNÁ, J. a H. MORAOVÁ, ed. Proceedings 30th Conference of the International Group for the Psychology of Mathematics Education. Volume 4. Prague: PME, 2006, s. 457-464.

[45] STEFANOV, Stoyan. React: up & running: building web applications. Sebastopol, California: O'Reilly, 2016. ISBN 978-1-491-93182-0.

[46] ŠIŠMA, Pavel. Teorie grafů 1736-1963. Praha: Prometheus, 1997. ISBN 80-7196-065-9.

[47] ŠITINA, Jiří. Grafové algoritmy a jejich vizualizace [online]. Hradec Králové, 2010 [cit. 2018-06-12]. Dostupné z: https://theses.cz/id/x3oakx/. Diplomová práce. Univerzita Hradec Králové, Fakulta informatiky a managementu. Vedoucí práce Eva Milková.

[48] ŠTRAUSOVÁ, Irena. GeoGebra a OK Geometry jako pomocníci při dokazování. In: HAŠEK, Roman. Sborník příspěvků 6. konference: Užití počítačů ve výuce matematiky. České Budějovice: Jihočeská univerzita v Českých Budějovicích, 2013, s. 350-353. ISBN 978-80-7394-448-3.

[49] ŠTRAUSOVÁ, Irena. Vizuální důkazy ve výuce matematiky. In: South Bohemia Mathematical Letters. České Budějovice: Jihočeská univerzita v Českých Budějovicích, 2012, s. 48-52. Volume 20.

[50] ŠŤASTNÁ, Markéta. Vizuální podpora výuky předmětů zabývajících se teorií grafů a grafovými algoritmy [online]. Hradec Králové, 2017 [cit. 2018-06-13]. Dostupné z: https://theses.cz/id/ab65wp. Diplomová práce. Univerzita Hradec Králové, Fakulta informatiky a managementu. Vedoucí práce RNDr. Andrea Ševčíková.

[51] TALL, David. Intuition and rigour: the role of visualization in the calculus. In: ZIMMERMANN a CUNNINGHAM, ed. Visualization in Mathematics. 1991, s. 106-107. MAA Notes Volume 19. Dostupné také z: https://homepages.warwick.ac.uk/staff/David.Tall/pdfs/dot1991a-int-rigour-maa.pdf

[52] TAMASSIA, Roberto. Handbook of graph drawing and visualization. Boca Raton: CRC Press, Taylor & Francis Group, [2014]. ISBN 978-1-58488-412-5.

[53] THIELE, Rüdiger. Matematické důkazy. 2., nezm. vyd. Praha: Státní nakladatelství technické literatury, 1986. Polytechnická knižnice (SNTL). ISBN 04-006-86.

[54] VIPUL, A M a Sonpatki PRATHAMESH. ReactJS by Example - Building Modern Web Applications with React: Get up and running with ReactJS by developing five cutting-edge and responsive projects. Birmingham, UK: Packt Publishing, 2016. ISBN 978-1-78528-964-4.

[55] WARD, Matthew, Georges G. GRINSTEIN a Daniel KEIM. Interactive data visualization: foundations, techniques, and applications. Natick: A K Peters, c2010. ISBN 978-1-56881-473-5.

[56] WHITELEY, Walter. Visualization in Mathematics: Claims and Questions towards a Research Program [online]. York University, Toronto, Canada, 2004, June 2004 [cit. 2018-06-17]. Dostupné z: https://www.researchgate.net/publication/237595057_Visualization_in_Mathematics_Claims_and_Questions_towards_a_Research_Pr

[57] ŽÁRA, Ondřej. JavaScript: programátorské techniky a webové technologie. Brno: Computer Press, 2015. ISBN 978-80-251-4573-9.

9.2 Internetové zdroje

[58] AHUVIA, Yogev. React vs. Vue (vs. Angular). In: Medium [online]. Medium, c2018, 28.6.2018 [cit. 2018-07-02]. Dostupné z: https://medium.com/fundbox-engineering/react-vs-vue-vs-angular-163f1ae7be56

[59] Airbnb React/JSX Style Guide [online]. Airbnb, c2018 [cit. 2018-07-03]. Dostupné z: https://github.com/airbnb/javascript/tree/master/react

[60] ALEXSEYENKO, Andrew. Angular 2 vs React: What to chose in 2017?. In: TechMagic: Blog [online]. 2017, 20.9.2017 [cit. 2018-07-01]. Dostupné z: https://blog.techmagic.co/angular-2-vs-react-what-to-chose-in-2017

[61] Balsamiq: Balsamiq Cloud [online]. Balsamiq Studios, c2008-2018 [cit. 2018-07-08]. Dostupné z: https://balsamiq.com

[62] Bootstrap: The most popular HTML, CSS, and JS library in the world [online]. Twitter, c2018 [cit. 2018-07-03]. Dostupné z: https://getbootstrap.com

[63] CEDDIA, Dave. Do I need Node.js in the backend?. In: Daveceddia.com [online]. 2016, 8.6.2016 [cit. 2018-08-14]. Dostupné z: https://daveceddia.com/do-i-need-nodejs-backend-for-react-angular

[64] CLARK, Andrew. React v16.0. React: Blog [online]. Facebook, 2017, 26.9.2017 [cit. 2018-06-27]. Dostupné z: https://reactjs.org/blog/2017/09/26/react-v16.0.html

[65] Ecma International: ECMAScript 2015 Language Specification [online]. Geneva: Ecma International, c2015 [cit. 2018-06-21]. Dostupné z: https://ecma-international.org/ecma-262/6.0/#sec-declarations-and-the-variable-statement

[66] Enzyme [online]. Airbnb, c2018 [cit. 2018-07-06]. Dostupné z: http://airbnb.io/enzyme

[67] Font Awesome: React [online]. Fonticons, c2018 [cit. 2018-07-04]. Dostupné z: https://fontawesome.com/how-to-use/on-the-web/using-with/react

[68] GeoGebra: What is GeoGebra? [online]. International GeoGebra Institute, c2018 [cit. 2018-06-12]. Dostupné z: https://geogebra.org/about

[69] GitHub: Create React App [online]. Facebook, c2018 [cit. 2018-07-03]. Dostupné z: https://github.com/facebook/create-react-app

[70] Google Trends: Porovnání React a Angular [online]. Google, c2018 [cit. 2018-08-12]. Dostupné z: https://trends.google.com/trends/explore?date=today%205-y&q=react,angular

[71] GraphTea: Your buddy to teach, learn and research on graph theory. [online]. [cit. 2018-06-12]. Dostupné z: http://graphtheorysoftware.com/about

[72] HAŠEK, Roman. Užití GeoGebry II: Elipsa. In: GeoGebra [online]. c2018 [cit. 2018-06-12]. Dostupné z: https://geogebra.org/m/HpAQuSES#material/fVXDjDXN

[73] Jest: Delightful JavaScript Testing [online]. Facebook, c2018 [cit. 2018-07-06]. Dostupné z: https://jestjs.io

[74] MathJax: A JavaScript display engine for mathematics [online]. MathJax, c2009-2018 [cit. 2018-07-04]. Dostupné z: https://mathjax.org

[75] NPM: immutability-helper [online]. c2018 [cit. 2018-07-02]. Dostupné z: https://npmjs.com/package/immutability-helper

[76] NPM: React-Bootstrap [online]. c2018 [cit. 2018-07-03]. Dostupné z: https://npmjs.com/package/react-bootstrap

[77] NPM: React CSS Transition Replace [online]. c2018 [cit. 2018-07-11]. Dostupné z: https://npmjs.com/package/react-css-transition-replace

[78] NPM: react-dialog [online]. c2018 [cit. 2018-07-12]. Dostupné z: https://npmjs.com/package/react-dialog

[79] NPM: react-fontawesome [online]. Fort Awesome, c2018 [cit. 2018-07-04]. Dostupné z: https://npmjs.com/package/@fortawesome/react-fontawesome

[80] NPM: React graph vis [online]. c2018 [cit. 2018-07-05]. Dostupné z: https://npmjs.com/package/react-graph-vis

[81] NPM: React MathJax [online]. c2018 [cit. 2018-07-04]. Dostupné z: https://npmjs.com/package/react-mathjax2

[82] NPM: react-router [online]. c2018 [cit. 2018-07-03]. Dostupné z: https://npmjs.com/package/react-router

[83] NPM: react-router-bootstrap [online]. c2018 [cit. 2018-07-03]. Dostupné z: https://npmjs.com/package/react-router-bootstrap

[84] NPM: React Scroll [online]. c2018 [cit. 2018-07-09]. Dostupné z: https://npmjs.com/package/react-scroll

[85] NPM: react-sketch [online]. c2018 [cit. 2018-08-14]. Dostupné z: https://npmjs.com/package/react-sketch

[86] NPM: react-transition-group [online]. c2018 [cit. 2018-07-11]. Dostupné z: https://npmjs.com/package/react-transition-group

[87] React Documentation [online]. Facebook, c2018 [cit. 2018-07-02]. Dostupné z: https://reactjs.org/docs/getting-started.html

[88] TECHMAGIC. ReactJS vs Angular5 vs Vue.js: What to choose in 2018? [online]. Medium, c2018 [cit. 2018-06-23]. Dostupné z:

https://medium.com/@TechMagic/reactjs-vs-angular5-vs-vue-js-what-to-choose-in-2018-b91e028fa91d

[89] Vis.js [online]. Almende B.V., c2015-2017 [cit. 2018-07-05]. Dostupné z: http://visjs.org

[90] Vis.js Documentation: Network [online]. Almende B.V., c2015-2017 [cit. 2018-07-05]. Dostupné z: http://visjs.org/docs/network/

[91] Vue.js: Guide [online]. Evan You, c2014-2018 [cit. 2018-06-25]. Dostupné z: https://vuejs.org/v2/guide

[92] ZAYTSEV, Juriy. ECMAScript 6: Compatibility Table [online]. GitHub, c2018 [cit. 2018-06-22]. Dostupné z: http://kangax.github.io/compat-table/es6

[93] Zdroják.cz: Každý programátor by mal poznať týchto 7 výhod React.js [online]. Devel.cz Lab, 2016 [cit. 2018-06-25]. Dostupné z: https://zdrojak.cz/clanky/kazdy-programator-mal-poznat-tychto-7-vyhod-react-js

10 Seznam obrázků

11 Seznam ukázek kódů

12 Přílohy

1) Zjednodušený úryvek zdrojového kódu komponenty `MainNavbar`

```
import { Navbar, Nav, NavItem, NavDropdown, MenuItem, Image }
from 'react-bootstrap';
import { Link } from 'react-router-dom';
import { LinkContainer } from 'react-router-bootstrap';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';

const MainNavbar = () => (
  <Navbar className="main-nav" default collapseOnSelect>
    <Navbar.Header>
      <Navbar.Brand>
        <Link to="/">
          <Image src="assets/image/logo.png" />ProofVis
        </Link>
      </Navbar.Brand>
      <Navbar.Toggle />
    </Navbar.Header>
    <Navbar.Collapse>
      <Nav>
        <NavDropdown
          title={
            <span>
              <FontAwesomeIcon icon="arrow-right" />
              Důkazy přímo
            </span>
          }
          className="nav-item"
        >
          <LinkContainer to="/dukaz1">
            <MenuItem>Důkaz 1</MenuItem>
          </LinkContainer>
          /* ... */
        </NavDropdown>
        /* ... */
        <LinkContainer to="/napoveda">
          <NavItem className="nav-item">
            <FontAwesomeIcon icon="help " /> Nápověda
          </NavItem>
        </LinkContainer>
      </Nav>
    </Navbar.Collapse>
  </Navbar>
);

export default MainNavbar;
```

2) Objekt nastavení vizualizace předaný vlastnosti `options` komponenty `GraphVis`

```
export const graphVisOptions = {
  autoResize: true,
  height: '100%',
  width: '100%',
  // Set localization to Czech language
  locale: 'cs',
  // Pass object with Czech translations.
  locales: graphVisLocales,
  // Default node properties
  nodes: {
    shape: 'circle',
    color: { background: palette.yellow, border: palette.black },
    label: '   ',
    margin: 12,
    font: { size: 18 },
  },
  // Default edge properties
  edges: {
    arrows: {
      to: { enabled: false, scaleFactor: 2 },
      from: { enabled: false, scaleFactor: 2 },
    },
    color: { color: palette.black, hover: palette.black },
    width: 1,
    dashes: false,
    label: '   ',
    font: { align: 'top', size: 18 },
  },
  // Allow to interact with the graph
  interaction: {
    dragNodes: true,
    dragView: false,
    // Use node hover colors when to mouse moves over it
    hover: true,
    hoverConnectedEdges: false,
    // Longheld click or CTRT+click will add to the selection
    multiselect: true,
    navigationButtons: true,
    selectable: true,
    // Do not highlight connecting edges on selecting a node
    selectConnectedEdges: false,
    zoomView: true,
  },
  // Turn automatic graph rearranging off
  physics: false,
  // Turn configuration panel off
  configure: false,
};
```

3) Příklad implementace funkce `nextStep()`

```
nextStep = () => {
  if (this.state.currentStep < constants.stepSum) {
    switch (this.state.currentStep) {
      case 0: {
        this.setState(this.step1);
        this.network.moveTo(cameraPositions[0]);
        break;
      }
      case 1: {
        this.step2Animation();
        let interval1 = setInterval(this.step2Animation, 6000);
        this.setState({ intervals: [interval1] });
        this.network.moveTo(cameraPositions[1]);
        scroller.scrollTo(
          'proofStepPanel2',
          getScrollOptions(window.scrollY)
        );
        break;
      }
      case 2: {
        this.clearAllTimers(this.state);
        this.setState(this.step3);
        this.network.moveTo(cameraPositions[2]);
        scroller.scrollTo(
          'proofStepPanel3',
          getScrollOptions(window.scrollY)
        );
        break;
      }
      /* ... */
      case 9: {
        this.setState(this.step10);
        scroller.scrollTo(
          'proofStepPanel7',
          getScrollOptions(window.scrollY)
        );
        break;
      }
      default: {
        break;
      }
    }

    this.updateCurrentStep(this.state.currentStep, 1);
  }
};
```

4) Zjednodušený úryvek souboru `editNodeFunctions.js` s funkcemi pro obsluhu modálního okna pro úpravy atributů vrcholu

```
/**
 * Displays dialog with form for editing selected node.
 */
export function showEditNodeDialog(nodeData, callback) {
  // Fill node edit dialog's inputs by selected node data
  document.getElementById('inpNodeLabel').value = nodeData.label;
  document.getElementById('inpNodeColor').value =
    nodeData.color.background;
  document.getElementById('inpLabelColor').value =
    nodeData.font.color;
  // Bind saveNode and cancelEditNode functions
  document.getElementById('btnSave').onclick =
    saveNode.bind(this, nodeData, document, callback);
  document.getElementById('btnCancel').onclick =
    cancelNodeEdit.bind(this, document, callback);
  document.getElementById('editNodeDialog').style.display = 'block';
}

/**
 * Sets inputed data to the selected node, saves the node and hides
the Node Edit dialog.
 */
function saveNode(nodeData, document, callback) {
  nodeData.label = document.getElementById('inpNodeLabel').value;
  nodeData.color.background =
    document.getElementById('inpNodeColor').value;
  nodeData.font.color =
    document.getElementById('inpLabelColor').value;
  clearEditNodeDialog(document);
  callback(nodeData);
}

/**
 * Cancels editing of node and hides Edit Node dialog.
 */
function cancelNodeEdit(document, callback) {
  clearEditNodeDialog(document);
  callback(null);
};

/**
 * Clears and hides Edit Node dialog.
 */
function clearEditNodeDialog(document) {
  document.getElementById('btnSave').onclick = null;
  document.getElementById('btnCancel').onclick = null;
  document.getElementById('editNodeDialog').style.display = 'none';
}
```

5) Zjednodušená ukázka jednotkového testu komponenty panelu definic

```
import React from 'react';
import { configure, shallow } from 'enzyme';
import Adapter from 'enzyme-adapter-react-16';

import DefinitionPanel from './DefinitionPanel';

// Connect enzyme to React
configure({ adapter: new Adapter() });

describe('DefinitionPanel', () => {
  let wrapper = shallow(
    <DefinitionPanel definitionPanels={[]} currentStep={0} />
  );

  it('should render only the panel with showForSteps property
      equal to 4', () => {
    wrapper.setProps({
      definitionPanels: [
        {
          id: 1,
          showForSteps: [2, 3],
          content: <p>Fiktivni definice pro krok 2 a 3.</p>
        },
        {
          id: 2,
          showForSteps: [4],
          content: <p>Fiktivni definice pro krok 4.</p>
        },
      ],
      currentStep: 4,
    });

    expect(
    wrapper.contains(<p>Fiktivni definice pro krok 2 a 3.</p>))
    .toBe(false);

    expect(
    wrapper.contains(<p>Fiktivni definice pro krok 4.</p>))
    .toBe(true);
  });
});
```

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2016/2017

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai2-p)

Podklad pro zadání DIPLOMOVÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Bc. Skořepa Tomáš	U Struhy 1016, Poděbrady - Poděbrady III	I1500897

TÉMA ČESKY:

Vizuální podpora výuky předmětů zabývajících se teorií grafů a grafovými algoritmy

TÉMA ANGLICKY:

Visual support for teaching subjects dealing with Graph theory and Graph algorithms

VEDOUCÍ PRÁCE:

RNDr. Andrea Ševčíková - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl práce: Vytvoření studijních materiálů pro výuku důkazů matematických vět v předmětech zabývajících se teorií grafů a grafovými algoritmy.

Osnova práce:
1. Úvod
2. Cíl a metodika práce
3. Teoretická východiska
4. Možnosti a význam vizualizace ve výuce
5. Technologie pro tvorbu studijních materiálů
6. Tvorba studijních materiálů
7. Výsledky
8. Závěr a doporučení
9. Literární zdroje

SEZNAM DOPORUČENÉ LITERATURY:

DEMEL, Jiří: Grafy a jejich aplikace, Academia 2002.
FULTON, Steve: HTML5 Canvas, O'Reilly 2013.
MILKOVÁ, Eva: Teorie grafů a grafové algoritmy, Gaudeamus 2013.
MURRAY, Scott: Interactive Data Visualization for the Web, O'Reilly 2013.
ODELL, Den: Pro JavaScript Development, Apress 2014.
PHILIPS, Linda: Visualization in Mathematics, Reading and Science Education, Springer 2010.
TAMASSIA, Roberto: Handbook of Graph Drawing and Visualization, CRC Press 2014.
THIELE, Rüdiger: Matematické důkazy, SNTL 1986.

Podpis studenta: ..... **Datum:** .....

Podpis vedoucího práce: ..... **Datum:** .....

(c) IS/STAG , Portál - Podklad kvalifikační práce , I1500897 , 12.10.2016 19:18