Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Webový nástroj pro vizualizaci toku v síti

Plzeň 2021 Václav Werner

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 6. května 2021

Václav Werner

Abstract

This bachelor thesis is concentrated on the design and implementation of a web application, which would allow its users to visualize graph data. In addition to visualization, the application also enables the user to calculate node significance using user supplied libraries and to then graphically visualize their results. The theoretical part of this thesis contains an overview of the most commonly used node centrality measures and a list of the web technologies used during the implementation phase is also present here. The practical part of this work contains details regarding the implementation and a testing overview. The final part summarizes the test results and highlights key areas in which the application can be further extended.

Abstrakt

Cílem této bakalářské práce je navrhnout a implementovat webovou aplikaci, která umožňuje vizualizovat grafy a sítě. Tato aplikace dále umožňuje provádět výpočty ohodnocení vrcholů těchto sítí za pomoci uživateli dodávaných knihoven a následně graficky reprezentovat jejich výsledky. V teoretické části práce jsou popsány nejdůležitější metody hodnocení významnosti vrcholů grafu a dále je zde proveden výběr technologií, které byly při tvorbě aplikace použity. V praktické části práce je detailně popsána implementace jednotlivých částí aplikace. Následně je popsán průběh a výsledky testování. Na závěr je provedeno zhodnocení výsledků práce a návrh oblastí, ve kterých lze aplikaci v budoucnu dále rozvíjet.

Obsah

1 Úvod

V rámci mnoha výzkumných oborů se pracuje s daty, která lze reprezentovat grafovými strukturami. Například při analýze sociálních sítí si lze sociální síť představit jako graf, kde lidé tvoří jednotlivé vrcholy a sociální vazby jsou prezentovány hranami. Cílem výzkumníků v mnoha případech bývá navrhnout nové metody, kterými lze ohodnotit významnost jednotlivých vrcholů grafu.

Jednou z těchto metod je algoritmus PageRank, který firma Google využívá při řazení výsledků svého webového vyhledávače, kde vrchol představuje webovou stránku a hrana hypertextový odkaz. Jedním z cílů této práce je seznámit se s tímto a podobnými algoritmy a vysvětlit, na jakém principu fungují.

Při testování těchto metod je potřeba je vyzkoušet na ukázkových datech a srozumitelně vizualizovat jejich výsledky. Existuje řada programů, kterými lze tento úkol řešit, ty ale jsou z mého pohledu v mnoha ohledech omezené svou licencí, kompatibilitou, uživatelskou přívětivostí či specifickými požadavky na stroj, na kterém běží. Optimálním řešením může být specializovaná webová aplikace, která umožní uživateli nahrát vlastní program v podobě JAR knihovny. Tento program lze následně spustit na výkonném vzdáleném stroji a výsledky graficky prezentovat v okně prohlížeče nebo stáhnout ve strojově čitelné podobě pro další zpracování. Tím se minimalizují požadavky na výkon klientského stroje a rovněž odpadne nutnost instalovat separátní jednoúčelovou aplikaci s jejími případnými závislostmi.

Primárním cílem této práce je výše zmíněnou aplikaci navrhnout a implementovat. Aplikace by měla umožnit uživatelům nahrávat a spouštět různé programy pro nalezení ohodnocení vrcholů grafu na vzdáleném stroji a následně srozumitelně prezentovat jejich výsledky. Vzhledem k tomu, že aplikace je cílená na hodnocení významnosti vrcholů, může být jednodušší než jiné aplikace, které dokáží nad grafem provádět komplexní operace, a primárně bude zaměřená na přehlednost a uživatelskou přívětivost.

Byť dle zadání není persistence dat v aplikaci vyžadována, rozhodl jsem se do aplikace navíc přidat správu uživatelů a ukládání projektů na vzdálený server. Myslím, že tato funkcionalita činí aplikaci daleko bezpečnější a praktičtější pro uživatele, kteří často střídají stroje. Mým osobním cílem bylo se v rámci bakalářské práce seznámit s existujícími frameworky a jinými technologiemi, které moderní webové aplikace využívají, např. NodeJS, Angular, React nebo Docker.

Kapitola 2 obsahuje stručný přehled některých typů algoritmů, se kterými bude aplikace umět pracovat. Nejdůležitější je zde sekce 2.2, která vysvětluje princip algoritmu *PageRank*. V kapitole 3 jsou analyzovány požadavky na aplikaci, proveden návrh architektury aplikace a výběr technologií, které byly při implementaci aplikace použity. Samotný postup implementace je pak popsán v kapitole 4. Zde se lze například dozvědět jaké při implementaci nastaly problémy a jaké bylo jejich řešení, nebo jak funguje komunikace mezi jednotlivými komponentami aplikace.

V kapitole 5 je popsán postup a průběh testování aplikace. Rovněž je zhodnocena funkčnost finální aplikace, její omezení a navrženy metody jak tato omezení odstranit. Na závěr je v kapitole 6 provedeno celkové shrnutí práce a zhodnocení toho, zda bylo dosaženo všech stanovených cílů. Dále jsou v této kapitole navrhnuty oblasti, ve kterých lze aplikaci v budoucnu vylepšit.

2 Hodnocení významnosti vrcholů grafu

Cílem této kapitoly je seznámit čtenáře s některými algoritmy, které lze použít k analýze grafů a sítí. Protože aplikace nebude vázána na žádnou pevně danou knihovnu či specifický algoritmus, jsou zde stručně popsány ty nejznámějších metody, které lze pro ohodnocení vrcholů grafu využít. Pokud není v textu popsáno jinak, tak všechny zde definované formule ohodnocují vrchol $v \in V$, kde V je množina všech vrcholů daného grafu.

2.1 Centrality measures

Metody *Centrality measures* (CM) měří tzv. *centralitu*, tj. určují významnost jednotlivých vrcholů v dané síti. Například v sociální síti, kde vrchol představuje osobu a hrana vazbu mezi dvěma osobami, by měly celebrity vyšší hodnotu centrality než průměrné osoby vzhledem k velikosti své fanouškovské základny. Výsledky CM obvykle normalizujeme, čímž umožníme jejich porovnávání mezi různě velkými sítěmi. Metod CM existuje celá řada [1, 6], nicméně v následující části textu se budeme zabývat pouze nejznámějšími metodami *Degree centrality*, *Closeness centrality*, *Betweenness centrality* a *Eigenvector centrality*.

2.1.1 Degree centrality

Jedná se o nejjednodušší metodu výpočtu centrality, protože je určena pouze stupněm daného vrcholu [6]. Stupeň vrcholu je dán počtem vazeb na ostatní vrcholy. V orientované síti se stupeň vrcholu dělí na vstupní a výstupní a následně se i vrcholy dělí na prominentní (s vysokým vstupním stupněm) a vlivné (s vysokým výstupním stupněm).

Výhodou této metody je nízká výpočetní složitost (stačí projít všechny vrcholy grafu pouze jednou). Nevýhodou je, že vrcholy s nejvyšším stupněm mohou být významné pouze lokálně. Navíc lze tuto metodu použít pouze v případě že jsou všechny hrany grafu stejně významné.

Normalizovanou verzi této metriky lze vyjádřit pomocí vzorce 2.1, kde $d(v)$ je stupeň vrcholu $v \in V$ a n je počet vrcholů v grafu. Normanizaci provádí podíl $n - 1$, který zajišťuje, že výsledná hodnota se vždy nachází v intervalu $< 0, 1 >$.

$$C_D(v) = \frac{d(v)}{n-1} \tag{2.1}$$

2.1.2 Closeness centrality

Principem této metody je určení průměrné blízkosti specifického vrcholu vůči ostatním vrcholům [16]. Jako vzdálenost mezi dvěma vrcholy uvažujeme délku nejkratší cesty mezi danými vrcholy. Nejkratší cesta mezi dvěma vrcholy je definována jako posloupnost přechodů mezi danými dvěma vrcholy, která má minimální ohodnocení hran. Průměrná vzdálenost je pak suma vzdáleností pro všechny vrcholy grafu, dělená celkovým počtem vrcholů. Vrcholy s nízkou průměrnou vzdáleností jsou tudíž blíže ostatním vrcholům.

Normalizovanou verzi metriky lze vyjádřit vzorcem 2.2 kde funkce $f(v, u)$ značí délku nejkratší cesty mezi vrcholy v a u. Pro nalezení nejkratší cesty mezi dvěma vrcholy se tradičně používá algoritmus, který navrhl E. W. Dijkstra [3]. Případně lze použít Floyd-Warshallův algoritmus [5], který hledá všechny nejkratší cesty mezi všemi vrcholy grafu pomocí matice sousednosti.

$$C_C(v) = \frac{n-1}{\sum_{u \in V; u \neq v} f(v, u)} \tag{2.2}$$

Na rozdíl od metody *Degree centrality*, která sleduje pouze stupeň daného vrcholu, dokáže tato metoda odhalit i vrcholy menšího stupně, přes které ale proudí komunikace mezi mnoha ostatními vrcholy a jsou tak významné na základě své polohy v grafu.

2.1.3 Betweenness centrality

Tato metoda sleduje schopnost vrcholu propojovat různé skupiny vrcholů [6]. V sociální síti se může jednat o osobu, která současně náleží například do okruhů studentů, sportovců a matematiků, čímž vytváří spojení mezi těmito jinak nezávislými skupinami. Tato osoba je pak významná, protože dokáže mezi skupinami přenášet zprávy a tím ovlivňovat dění v nich.

$$C_B(v) = \sum_{u,w \in V; u \neq v \neq wj} \frac{g(u, v, w)}{g(u, w)} \tag{2.3}$$

Pro výpočet musíme, stejně jako v *Closeness centrality*, vypočítat nejkratší cesty mezi všemi vrcholy grafu, ale tentokrát nás nezajímá délka cest do daného vrcholu, nýbrž počet cest, které tento vrchol obsahují, viz vzorec 2.3. Protože se mezi dvěma vrcholy může nacházet více nejkratších cest

stejné délky, dělíme $g(u, v, w)$, tj. počet nejkratších cest mezi vrcholy u a w vedoucí přes vrchol v, celkovým počtem nejkratších cest mezi těmito dvěma vrcholy, tj. $g(u, w)$.

Na rozdíl od ostatních vzorců není vzorec 2.3 normalizován, tudíž jej nelze použít pro porovnání různě velkých sítí.

2.1.4 Eigenvector centrality

Jedná se o metodu odvozenou od *Closeness centrality*. Na rozdíl od předchozích metod se jedná o iterační metodu, která svůj výpočet ukončí po dosažení určité úrovně přesnosti, nebo po provedení pevně stanoveného počtu iterací. Vzhledem k tomu, že je tato metoda odvozena od C_C ji lze rovněž využít pouze za předpokladu že jsou všechny hrany grafu stejně významné.

Výpočet jedné iterace normalizované verze této metody pro vrchol v lze zapsat vzorcem 2.4 kde λ je normalizační hodnota a $U(v)$ je množina vrcholů sousedících s vrcholem v.

$$C_{E_{t+1}}(v) = \lambda \sum_{u \in U(v)} C_{E_t}(u) \tag{2.4}$$

Pokud budeme uvažovat matici sousednosti A, tak po přepsání rovnice do maticové podoby zjistíme, že platí rovnice $x_{t+1} = \lambda A x_t$, kde je vektor x vektorem vlastních čísel (eigenvector) matice A, po kterém tato metoda dostala své jméno.

Finální řešení lze tedy nalézt opakovaným násobením $x_{t+1} = A x_t$ a následnou normalizací $x_{(t)} = \lambda x_{(k)}$. Normalizační hodnota $\lambda = 1/||x_{t+1}||$. Výpočet lze ukončit poté, co $||x_{t+1} - x_t|| < \epsilon$, kde ϵ je požadovaná úroveň přesnosti. Bohužel takto definovaný algoritmus nemusí konvergovat, viz problémy zmíněné v sekci 2.2.1.

2.2 PageRank

Jedná se variantu metody *Eigenvector centrality*, kterou firma Google používá při řazení výsledků ve svém internetovém vyhledávači. Algoritmus PageRank [2, 14] byl navržen pro orientované grafy, kde každý vrchol představuje webovou stránku a každá hrana představuje hypertextový odkaz. Byť původním účelem tohoto algoritmu je řazení webových stránek, lze jej aplikovat na libovolný problém vyžadující ohodnocení významnosti vrcholů grafu, například pro analýzu sociálních sítí, citační analýzu, či v oboru neurovědy [4]. Při studiu tohoto algoritmu jsem čerpal z práce [13].

Základní verzi algoritmu lze zapsat podobně jako *Eigenvector centrality*, viz vzorec 2.5. $N(u)$ udává počet výstupních hran vrcholu u. Výsledná hodnota $R_{(t+1)}(v)$ udává hodnotu PageRanku pro vrchol v v iteraci $t+1$.

$$R_{(t+1)}(v) = \sum_{u \in B(v)} \frac{R_{(t)}(u)}{N(u)} \tag{2.5}$$

2.2.1 Problémy konvergence

Tato základní verze algoritmu (viz vzorec 2.5) se potýká se dvěma problémy, které výpočet komplikují. Prvním problémem je případ, kdy se hodnota PageRanku přelévá do tzv. slepých vrcholů, tj. vrcholů, ze kterých nevede hrana ven. Původním řešením [14] bylo normalizovat hodnoty všech ostatních vrcholů a slepé vrcholy ignorovat, protože se převážně jednalo o části webu, které ještě nebyly do vyhledávače přidány. Toto řešení ale nelze použít pro grafy, ve kterých jsou slepé vrcholy jejich platnou součástí, proto se zavedlo řešení, kde se hodnota slepých vrcholů přerozděluje mezi všechny vrcholy grafu. Proto je ale napřed nutné definovat množinu všech slepých vrcholů S.

Druhý problém nastává, když dvě stránky odkazují pouze na sebe navzájem a nevede z nich cesta ven. Tomuto problému se přezdívá *Rank sink*, protože v těchto případech do těchto vrcholů bude 'odtékat' ohodnocení z vrcholů, které do nich vedou. Takové vrcholy pak mají velice vysoké ohodnocení, zatímco ohodnocení ostatních vrcholů padá k nule. Navíc si tyto vrcholy mohou hodnotu mezi sebou 'předávat', důsledkem čehož algoritmus diverguje. Jako řešení se zavádí tzv. *faktor tlumení* značený d. Ten ve webovém grafu reprezentuje pravděpodobnost přechodu mezi stránkami bez použití odkazu (pomocí zadání adresy). Protože se jedná o pravděpodobnost, hodnota d je z intervalu $(0, 1)$. Upravený výpočet najdeme ve vzorci 2.6.

$$R_{(t+1)}(v) = \frac{(1-d)}{n} + d\left(\sum_{u \in B(v)} \frac{R_{(t)}(u)}{N(u)} + \sum_{s \in S} \frac{R_{(t)}(s)}{n})\right) \tag{2.6}$$

2.2.2 Rozšíření o personalizaci a vážené hrany

Dále lze do výpočtu přidat zvýhodňování hran a vrcholů, například v případě, kdy ve webovém grafu vede z jedné stránky více odkazů na stránku druhou, nebo když je stránka zvýhodněna jinou metodou řazení. Hranu lze zvýhodnit pomocí váhy, kdy nadefinujeme $w_{u\text{out}}$ jako součet vah všech výstupních hran vrcholu u a $w_{u,v}$ jako váhu hrany z u do v. Místo podílu výstupních hran $1/N(u)$ pak můžeme použít podíl vah $w_{u,v}/w_{u\text{out}}$.

Zvýhodnění vrcholu můžeme docílit nerovnoměrným rozdělením tlumícího faktoru, proto nadefinujeme množinu tzv. *personalizací* jednotlivých vrcholů P a $p_v \in P$ jako *personalizaci* vrcholu v. *Faktor tlumení* pro jednotlivé vrcholy následně upravíme dle podílu těchto hodnot, viz vzorec 2.7.

$$R_{(t+1)}(v) = \frac{(1-d)p_v}{\sum_{p \in P} p} + d\left(\sum_{u \in B(v)} \frac{R_{(t)}(u)w_{u,v}}{w_{u\text{out}}} + \sum_{s \in S} \frac{R_{(t)}(s)}{n})\right) \tag{2.7}$$

Při vývoji aplikace byla používána knihovna implementující tento algoritmus včetně rozšíření o vážené hrany a personalizaci vrcholů. Tato knihovna není součástí finální aplikace, byla vytvořena vedoucím práce, ale posloužila k otestování její funkčnosti.

3 Návrh aplikace

Jelikož cílem bylo navrhnout aplikaci, která nezávisí na žádné již existující funkcionalitě (mimo uživateli dodávané separátní knihovny psané v jazyce Java), návrh architektury aplikace a výběr použitých technologií byly zcela v mé kompetenci. Při výběru technologií jsem se zaměřoval především na jejich aktuální popularitu, kvalitu dokumentace a případně mé stávající zkušenosti s nimi.

3.1 Existující software

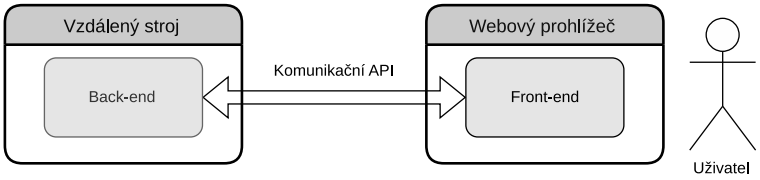
Mezi software, kterým lze provádět vizualizaci a manipulaci grafů, patří například programy Gephi [1], který obsahuje silnou vizualizační část, ale neumožňuje jednoduše spouštět vlastní metody ohodnocení, Pajek [2], který je dostupný pouze pro operační systém Windows, nebo balík igraph [3], kterým lze provádět analýzu a vizualizaci grafů v jazycích R, Python a C++.

3.2 Architektura aplikace

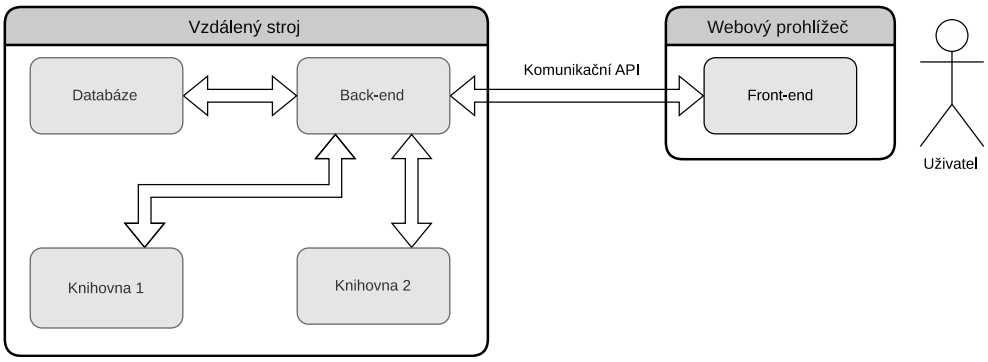
Webová aplikace běží na vzdáleném webovém serveru a uživatel k ní přistupuje přes webový prohlížeč. Taková aplikace je pak rozdělena na kód, který běží na vzdáleném stroji, a kód, který běží v prohlížeči klienta. Mezi těmito dvěma částmi se nachází komunikační API. Jak jde vidět z obr. 3.1, tak část, která běží na vzdáleném serveru, nazýváme **back-end**. Ten se stará o správu dat, přístupu a vystavuje své API, které následně využívá **front-end**. **Front-end** interaguje s uživatelem, přebírá od něj data a předává je **back-endu** k vykonání. Ten výsledky vrací zpět **front-endu**, který je následně prezentuje uživateli. Každá část aplikace může používat rozdílné technologie a mají na vývojáře rozdílné požadavky. Například **back-end** obvykle vyžaduje vysokou propustnost, zatímco **front-end** je zaměřen na uživatelskou přívětivost.

Ve vytváření aplikaci se budou nacházet dodatečné části. Pro ukládání dat použijeme externí databázový software, který běží nezávisle na ostatních částech programu. Byť by se pro uchovávání dat daly použít soubory, které

[1] https://gephi.org/
[2] http://mrvar.fdv.uni-lj.si/pajek/
[3] https://igraph.org/



Obrázek 3.1: Základní struktura aplikace.



Obrázek 3.2: Finální struktura aplikace.

by přímo upravoval **back-end**, databáze bývá odolnější vůči ztrátě dat a obsahuje protokol, který usnadňuje jejich manipulaci.

Další dodatečné části jsou pak samotné uživatelské knihovny obsahující algoritmy, které zpracovávají data grafu. Každá knihovna je vlastně nezávislá Java aplikace, která načítá data ze vstupu a výsledky vypisuje na výstup na základě pevně daného protokolu (viz příloha B). Tyto knihovny by měly být izolovány od zbytku systému, nebo by alespoň mělo být omezeno jejich nahrávání na server, aby se zamezilo problémům s bezpečností, kdy uživatel nahraje do systému aplikaci, která vykonává nějakou nežádoucí činnost. Na obr. 3.2 jde pak vidět finální architektura aplikace.

3.3 Návrh řešení back-endu

Technologií, které lze využít pro tuto část aplikace existuje velké množství. Výběr jsem tudíž omezil pouze na technologie, se kterými už jsem se někdy setkal, nebo na technologie, které využívají mnou ovládaný programovací jazyk a co nejvíce mi usnadňují práci.

PHP

Jedná se skriptovací jazyk, který je používaný velkou většinou webových stránek [18]. Jedná se o jeden z prvních jazyků, který umožnil tvorbu dyna-

mických stránek. Zkratka *PHP* znamená *Hypertext Preprocessor* a vystihuje původní účel tohoto jazyka, kdy jazyk generoval HTML, které následně poslal klientovi k vykreslení. Dnes pro jazyk existuje mnoho frameworků a knihoven, které umožňují tvorbu nejrůznorodějších aplikací a komunikaci s množstvím dalších technologií. Aby bylo možné jazyk použít, je nutné mít na serveru nainstalovaný webový server s jeho podporou, například Apache, nebo využít jeho GCI rozhraní.

Byť je PHP dlouhodobě tím nejpoužívanějším jazykem v rámci webových aplikací, rád bych zmínil, že se mnohdy jedná o stránky existující již delší dobu, nebo stránky využívající některý z již předpřipravených obsahových systémů. Většina nově založených projektů využívá místo PHP některou z dalších zde zmíněných technologií a popularita jazyka PHP jako základu nových projektů tak pozvolna upadá [4].

Dalším důvodem proč jazyk PHP v aplikaci nepoužívám je to, že s ním nemám zkušenosti a nechci se učit z mého pohledu zcela novou technologii, která nemá v současné době jistou budoucnost.

Node.js

Tato relativně nová technologie [5] umožňuje spouštění JavaScriptu i mimo webový prohlížeč. Je navržena na základě architektury řízené událostmi (event-driven architecture) a umožňuje spouštění příkazů paralelně, kdy se nečeká na jejich průběh, ale o jejich výsledku se informuje na základě události. Díky tomu jde Node.js dobře horizontálně škálovat. Navíc si technologie, navzdory krátkému času strávenému na trhu, vypěstovala masivní základnu uživatelů, kteří ji obohatili o tisíce open-source knihoven a dalších balíků s dodatečnou funkcionalitou, které lze jednoduše instalovat pomocí nástroje *NPM*. Protože jediným podporovaným jazykem je JavaScript, tak je možné mít back-end i front-end napsaný v jednom jazyce.

Přestože je tato technologie v současné době jedna z těch nejrychleji se rozvíjejících, rozhodl jsem se ji v projektu nepoužít, protože s ní mám již zkušenosti a jedním z mích cílů, co se týče této práce, je vyzkoušet něco nového.

ASP.NET

Jedná se o framework navržený firmou Microsoft pro tvorbu dynamických webových stránek [6]. V minulosti byl omezen pouze na operační systém Win-

[4] `https://www.tiobe.com/tiobe-index/php/`

[5] `https://nodejs.org/`

[6] `https://dotnet.microsoft.com/apps/aspnet`

dows skrze platformu .NET Framework, ale od roku 2016 jej jeho nástupce ASP.NET Core re-implementuje jako modulární framework běžící pod platformou .NET Core, která je dostupná pro vícero operačních systémů. Platforma .NET Core v současné době podporuje jazyky C#, F# a v budoucnu se plánuje podpora Visual Basic .NET. Výhodou této platformy je jednoduchá integrace na služby firmy Microsoft.

Podobně jako v případě Node.js již mám s touto technologií zkušenosti z předchozího studia. Byť se tehdy ještě neuskutečnil přechod z .NET Framework na .NET Core, při přechodu mezi frameworky došlo pouze k minimu změn. Tuto technologii jsem nakonec zavrhl, protože je příliš komplexní pro jednoduché aplikace, které nebudou využívat napojení na ostatní služby firmy Microsoft.

Jakarta EE

Dříve *Java Enterprise Edition* [7]. Jedná se o rozšíření Javy SE o 'enterprise' vlastnosti, například webové služby nebo distribuované výpočty. Pro backend se z těchto rozšíření dají použít Jakarta servlety, které definují jak zacházet s HTTP požadavky. Aby šlo servlety a další technologie používat, je nutné mít nainstalovaný webserver, který s nimi umí zacházet, například Apache Tomcat.

Spring

Jedná se o nejpoužívanější [17] Java framework [8], který v sobě implementuje princip *Inversion of Control* (IoC). Tento princip nám říká, že jednotlivé komponenty aplikace by měly být modulární a navzájem nezávislé. Řešení závislostí je přenecháno IoC kontejneru a nastává až za běhu. Například komponenta, která čte data z databáze, si nevytváří svou vlastní instanci databázového ovladače, nýbrž o ni pouze zažádá IoC kontejner. Kontejner se podívá do své konfigurace a podle ní vrátí specifickou instanci ovladače. Díky tomu lze jednoduše vyměnit databázi, aniž by bylo nutné přepisovat všechny závislé komponenty.

Framework jde v základu používat pro jakoukoliv Java SE aplikaci, ale pro účely webových aplikací se používá nad Jakartou EE. Pro framework je dostupných mnoho modulů, které se starají o základní funkčnost aplikace, například zabezpečení nebo komunikaci se zdrojem dat. Dále pro něj existuje tzv. Spring Boot, který umožňuje rychlý vývoj standalone webových apli-

[7] `https://jakarta.ee/`
[8] `https://spring.io/`

kací bez nutnosti složité konfigurace a dokáže využívat embedded Tomcat (případně jiný) server pro řešení síťové komunikace.

Vzhledem k tomu že aplikace má být zaměřena na uživatelskou přívětivost, kterou řeší front-end jsem se rozhodl na back-endu strávit pokud možno co nejméně času. Proto jsem se rozhodl používat právě Spring Boot, který mi umožňuje se zaměřit na samotnou funkcionalitu aniž bych musel implementovat například individuální dotazy na databázi nebo ověřování přístupu jednotlivých uživatelů. Navíc je, vzhledem k jeho popularitě, velice dobře zdokumentován a lze tak rychle nalézt řešení většiny problémů, které mohou při implementaci back-endu nastat.

3.3.1 Databázové technologie

Protože jsem se rozhodl používat databázi pro ukládání dat, je nutné si zvolit ten nejvhodnější databázový systém z pohledu výkonu, funkcionality a jednoduchosti implementace. Do databáze se budou ukládat tři druhy dat:

1. Data uživatelů:
    - přihlašovací údaje
    - práva v systému
    - seznam projektů
2. Data projektů:
    - data grafu (seznam vrcholů a hran)
    - historie výpočtů (seznam hodnot pro každý vrchol)
3. Data knihoven:
    - cesta ke knihovně
    - seznam parametrů knihovny

Relační databáze

Relační databázový systém je systém, který ukládá data do vzájemně provázaných tabulek. Každá tabulka má pevně danou strukturu sloupců, každý řádek představuje jednu datovou položku, která má svůj unikátní klíč. Tabulky jsou provázány pomocí relací, kdy jedna tabulka může obsahovat klíče z jiné tabulky, čímž se mezi nimi vytváří spojení. Pro manipulaci dat se většinou využívá jazyk SQL (*Structured Query Language*). Existuje celá řada

systémů pracujících na tomto modelu, například Oracle, MySQL, PostgreSQL a další.

Po rozboru dat jsem se rozhodl RDBS nepoužít, a to z několika důvodů. Tím prvním důvodem je to, že relační databáze není navržená pro ukládání dat, která mají grafovou strukturu. Pokud bych chtěl do relační databáze uložit seznam vrcholů a hran se zachováním relací, musel byt tak učinit do dvou dlouhých tabulek pro všechny projekty společně, což by, vzhledem k tomu že jeden graf může mít tisíce vrcholů a hran, systém zbytečně zatěžovalo. Další možností je uložit seznamy do tabulky projektů v textovém, či binárním formátu a následně je konvertovat po jejich přečtení z databáze. Toto řešení rovněž není optimální.

Dalším důvodem, proč se v tomto případě RDBS vyhnout je, že tato aplikace z větší části nebude využívat relačního modelu. Jediné místo, kde by se relací využívalo, je přiřazování projektů uživatelům, což se dá jednoduše obejít pomocí datových rozhraní ve Springu. Všechny ostatní druhy dat jsou na sobě nezávislé.

Posledním důvodem je to, že některá data uložená v databázi nebudou mít pevně danou strukturu, například každá knihovna může mít neurčité množství parametrů a každý vrchol grafu může mít několik dodatečných hodnot historie. Z tohoto důvodu by byla relační databáze, která vyžaduje pevně danou strukturu, značně omezující.

NoSQL databáze

Pod tímto označením najdeme celou řadu rozličných technologií [8], jejichž jediným pojítkem je to, že nepoužívají relační model. Řadí se mezi ně vše od jednoduchých programů v obsažených v jádře operačního systému, až po komplexní znalostní a vyhledávací systémy rozdělené mezi stovky instancí. Kategorie NoSQL databází nejsou pevně dané a někdy se navzájem překrývají. Zde jsem se pokusil o jejich základní rozdělení podle používaného datového modelu:

Key-value

- Používá tabulky klíčů a jim přiřazených hodnot.
- Často jsou tyto druhy databází používané jako cache.
- Některé systémy běží pouze v operační paměti bez persistence.
- Příklady: Redis, Berkeley DB

Wide-column

- Data jsou uložena po sloupcích.
- Jednotlivé záznamy se řadí do tabulek.
- Každý záznam v tabulce může obsahovat data z rozdílných sloupců.
- Výhodou je škálování, jednotlivé sloupce jsou navzájem nezávislé a mohou být uložené na různých instancích.
- Příklady: Apache Cassandra, HBase

Document

- Dokument je datová jednotka, např. objekt v JSON formátu.
- Dokumenty jsou rozděleny do kolekcí podle svého typu.
- Jednotlivé záznamy jsou částečně strukturované. Například záznam uživatele vždy obsahuje uživatelské jméno, ale telefon je dobrovolný.
- Velmi jednoduché na použití, lze ukládat všechna serializovatelná data.
- Příklady: MongoDB, CouchDB, ElasticSearch

Graph

- Podobají se dokumentovým systémům, ale navíc obsahují relace.
- Relace spojují jinak nezávislé záznamy (např. uživatel -> projekty).
- Umožňují rychlejší průchod daty než dokumentové systémy.
- Použití například pro znalostní systémy.
- Příklady: AllegroGraph, JanusGraph

Multi-model

- Velice flexibilní systém, využívá několika datových modelů současně.
- Vývojář nadefinuje strukturu dat a systém si pro ně sám následně vybere nejvhodnější model.
- Například databáze ArangoDB a OrientDB v sobě kombinují key-value, dokumentový a grafový systém.

Object

- Jedná se o databáze, které se řídí principy OOP [9].
- Data jsou uložena pomocí předdefinovaných objektů.
- Objekt může dědit vlastnosti od jiných objektů. Například objekty 'klient' a 'zaměstnanec' mohou zdědit vlastnosti od objektu 'člověk'.
- Většinou lze použít pouze se specifickými programovacími jazyky.
- Příklady: Perst (Java, .NET), Objectivity/DB (C++, C#, Python)

Výběr databázového systému

Jak jde vyčíst ze seznamu v sekci 3.3.1, tak použitá databáze by měla umět využívat datový model **Document**, **Graph**, nebo **Multi-model** (model **Key-value** je příliš jednoduchý a **Wide-column** neřeší všechny problémy zmíněné v sekci 3.3.1). Dále by systém měl být podporovaný frameworkem Spring [10], aby jsem si co nejvíce usnadnil práci. Také jsem ze rozhodl vyřadit systémy s licencí, díky které by mohly nastat problémy s jejich redistribucí. Následuje seznam systémů, nad kterými jsem uvažoval.

MongoDB Jedná se databázový systém na základě dokumentového modelu. Jednotlivé záznamy jsou ukládány ve formátu *BSON* (Binární JSON), systém pro komunikaci používá jazyk, který je rovněž založen na JSONu. Spring MongoDB podporuje již v základním nastavení. Systém je rovněž dobře otestovaný, zdokumentovaný a dostupný pod otevřenou licencí. Z těchto důvodů jsem se rozhodl ve své aplikaci používat právě tento systém.

ArangoDB Tento multi-model systém kombinuje dokumentový, grafový a key-value model. Díky tomu lze jednu databázi použít k mnoha různým účelům. Je podporovaný Springem a lze jej ve své komunitní edici redistribuovat. Systém je napsaný v jazyce C++ a pro manipulaci s daty využívá svůj vlastní query jazyk.

OrientDB Co se týče funkcionality, tak se jedná o systém podobný *ArangoDB*. Je napsaný v jazyce Java a manipulaci dat lze provádět pomocí jazyka odvozeného od SQL. Na rozdíl od předchozích systémů nemá oficiální podporu ve Springu, což by mohlo dělat potíže při implementaci.

[9] Object Oriented Programming

ElasticSearch Na rozdíl od ostatních systémů se nejedná o čistě databázový systém, místo toho se jedná o systém nabízející rychlé vyhledávání různých druhů dokumentů pomocí knihovny Lucene. Pro nás není rychlost vyhledávání příliš důležitá, tudíž nemáme důvod jej používat. Zde je zmíněn pouze z toho důvodu, že s ním mám předchozí zkušenosti a krátce jsem ho používal při tvorbě prototypu aplikace.

3.3.2 Komunikační protokol

Aby bylo možné mezi back-endem a front-endem úspěšně komunikovat, je nutné aby obě části používaly identický komunikační protokol. Aplikace, které vystavují svoje API na webu za použití známého protokolu, nazýváme webové aplikace. V případě navrhované aplikace je back-end webová služba, pro kterou musíme daný protokol navrhnout.

Representational State Transfer

REST (*Representational State Transfer*) je architektura, která využívá standartních HTTP metod ke komunikaci mezi webovými službami. REST protokol by měl být bez stavový, což znamená že při stejném vstupu vždy dostaneme ze serveru stejný výstup. Server vystavuje svůj obsah skrze unikátní URL identifikátory, přičemž pod každým identifikátorem se nachází buď kolekce dat, nebo specifický datový prvek. Například pokud URL `/users` obsahuje seznam uživatelů, pak `/users/admin` obsahuje informace o daném uživateli.

REST není sám o sobě plnohodnotný protokol, pouze definuje způsob komunikace skrze HTTP. Strukturu jednotlivých požadavků a jejich funkcionalitu navrhuje vývojář. Pokud služba vystavuje API navržené na základě RESTu, tvrdíme, že se jedná o tzv. RESTful webovou službu.

Metoda	**Kolekce**	**Prvek**
GET	Vrátit seznam prvků v kolekci	Vrátit data daného prvku
POST	Vytvořit nový záznam v kolekci	Vložit do prvku novou hodnotu
PUT	Nahradit celou kolekci	Upravit hodnoty prvku
DELETE	Smazat celou kolekci	Smazat prvek z kolekce

Tabulka 3.1: Typické REST operace.

Pro jednotlivé HTTP metody se typicky na serveru implementují unikátní operace (viz tabulka 3.1). Obsah zpráv je typicky ve formátu JSON. Jednotlivé operace lze ve Springu implementovat jednoduše pomocí předpřipravených rozhraní, proto jsem se rozhodl při návrhu API řídit právě touto architekturou. Výsledné API si lze prohlédnout v příloze B.

3.3.3 Další způsoby komunikace

SOAP

Jedná se o protokol navržený pro výměnu strukturovaných dat mezi webovými službami [10]. Na rozdíl od RESTu se jedná o pevně strukturovaný protokol, který není závislý na specifickém způsobu přenosu dat mezi službami. Pro komunikaci skrze SOAP lze využít jak HTTP, tak WebSocket, nebo např. SMTP.

Jednotlivé zprávy jsou ve formátu XML. Kořenová část zprávy je tzv. obálka `soap:Envelope`, která říká, že se jedná o SOAP zprávu. Obálka může obsahovat tzv. hlavičku zprávy `soap:Header`, která udává o jaký typ zprávy se jedná, a tělo `soap:Body`, které obsahuje samotný obsah zprávy.

Protokol SOAP je také podporován frameworkem Spring, ale na rozdíl od RESTu není nativně podporován frameworkem Angular. Z tohoto důvodu jsem se rozhodl jej nepoužívat.

WebSocket

Na rozdíl od ostatních zde uvedených technologií se nejedná o protokol, kterým lze nadefinovat jak probíhá komunikace mezi dvěma službami. Namísto toho se jedná o protokol, který umožňuje duplexní komunikaci mezi dvěma službami skrze jedno TCP spojení. WebSocket[11] (WS) může sloužit jako alternativa k HTTP, které je použito pouze k založení spojení.

V porovnání s protokolem HTTP poskytuje WS nižší latence [15] a vyšší propustnost, díky tomu, že nepřenáší data, která nejsou samotným obsahem zprávy, a má jednodušší dekódování než HTTP. Nevýhodou může být nutnost implementovat nad WS vlastní komunikační protokol (např. SOAP).

Frameworky Spring i Angular WebSocket podporují, ale vzhledem k nutnosti implementace vlastního protokolu a následně těžšímu hledání chyb při komunikaci jsem se rozhodl tuto technologii ve finální verzi nevyužít. WS byl v jedné vývojové verzi aplikace použit pro získání výsledků po spuš-

[10] https://www.w3.org/TR/2000/NOTE-SOAP-20000508/
[11] https://tools.ietf.org/html/rfc6455

tění knihovny, ale vzhledem k tomu, že aplikace není limitována latencí, či propustností, nedávalo jejich použití smysl.

3.4 Návrh řešení front-endu

Vzhledem k tomu, že front-end aplikace běží ve webovém prohlížeči, musí být vytvořen pouze za použití technologií, které jsou moderními prohlížeči podporovány. Byť výběr podporovaných technologií je značně omezen, existuje nad nimi řada nadstaveb, které tvorbu webových aplikací usnadňují.

3.4.1 Základní technologie

HTML a CSS

Hyper Text Markup Language (HTML) je jazyk navržený pro tvorbu webových stránek. HTML dokument je tvořen stromovou strukturou HTML elementů. Každý element může obsahovat další vnořené elementy (potomky), nebo svůj vlastní obsah (např. text, obrázek apod.). Z HTML dokumentů se v paměti webového prohlížeče sestavuje tzv. HTML DOM (*Document Object Model*), který je následně prohlížečem vykreslován.

První verze HTML byla vydána v roce 1990, v současné době je nejnovější verze HTML5, vydaná v roce 2014. Tato verze obsahuje nové prostředky zaměřené na tvorbu komplexních webových aplikací.

Zatímco HTML definuje obsah stránky, kaskádové styly (CSS) mu dodávají vzhled. CSS soubor obsahuje sadu pravidel, kde každé pravidlo obsahuje selektor říkající, na jaký element je aplikováno, a seznam stylových hodnot (např. barva textu, pozadí, velikost, atd.). Slovo 'kaskádové' vystihuje postup aplikování těchto stylů na element, kdy se styly aplikují na element postupně na základě specificity daného selektoru (napřed se aplikují méně specifické styly).

Webový prohlížeč pak z původního HTML kódu vytvoří HTML DOM ve své paměti, aplikuje na něj CSS a výsledek vykreslí.

JavaScript

Jedná se o skriptovací jazyk, který je spouštěn prohlížečem na straně klienta. Umožňuje prohlížeči reagovat na vstupy klienta a modifikovat DOM bez nutnosti komunikace se serverem. Jedná se o interpretovaný dynamický jazyk, který prohlížeč kompiluje a vykonává za běhu. Navzdory svému jménu nejsou JavaScript a Java příbuzné jazyky.

3.4.2 Další knihovny

Byť by se front-end aplikace dal implementovat pouze za pomoci HTML, CSS a JavaScriptu, rozhodl jsem se také využít jednu z moderních knihoven, které umožňují tvorbu uživatelských rozhraní typu *Single page application* (SPA). SPA je druh webové aplikace, kdy uživatel nepřechází mezi různými webovými stránkami v rámci jedné aplikace, ale aplikace je místo toho tvořena pouze jednou stránkou, která upravuje svůj obsah na základě vstupů od uživatele a back-endu.

Takových frameworků existuje několik, liší se v obsažené funkcionalitě a způsobu vytváření uživatelského rozhraní. Já jsem si k tomuto účelu zvolil framework Angular, který obsahuje robustní UI systém a přidává k němu další nástroje, které usnadňují vývoj SPA. Mezi další populární knihovny patří např. React a Vue.js, které jsou ale zaměřené čistě na uživatelská rozhraní a všechna další nezbytná funkcionalita se musí řešit skrze dodatečné knihovny.

NPM

Nástroj *NPM* [12] je správce JavaScriptových balíků, který umožňuje instalovat balíky s dodatečnou funkcionalitou ze vzdáleného repositáře pomocí jednoduchých příkazů. Nainstalované balíky se ukládají do seznamu, ze kterého je lze později nainstalovat znovu. Protože *NPM* je napsán v jazyce JavaScript, je nutné mít nainstalované runtime prostředí JavaScriptu, obvykle Node.js. *NPM* je v tomto projektu použit pro správu balíků front-endu.

Angular

Jedná se o webový framework vyvíjený firmou Google [14], který slouží k tvorbě uživatelských rozhraní. Jedná se o kompletní řešení front-endu, které obsahuje mimo nástrojů na tvorbu uživatelských rozhraní i řadu dodatečné funkcionality, například routování do specifického stavu aplikace podle URL, nebo vestavěnou ochranu proti Cross-site scriptingu [15].

Na rozdíl od ostatních podobných knihoven, které jsou založené na JavaScriptu, je Angular navržen pro použití s jazykem TypeScript. Stejně jako frameworky React a Vue.js je v Angularu uživatelské rozhraní reprezentováno komponentami. Komponenta je rozdělena na HTML šablonu, která

[12] NodeJS Package Manager [13]

[14] https://angular.io/

[15] Spouštění skriptů třetích stran z webové stránky, která jim umožňuje přístup do ostatních služeb.

udává, jak se má komponenta zobrazovat, a TypeScriptový kód, který říká, jaká obsahuje komponenta data a jak má reagovat na jejich změny. V HTML kódu šablony lze použít speciální Angular syntax, který umožňuje propojení mezi HTML a kódem šablony. Komponenta je zkompilována za běhu a do obsahu stránky se následně vloží její HTML. Pokud se změní data komponenty, provede se vyhodnocení podle šablony a případná aktualizace jejího HTML.

Angular dále obsahuje vlastní CLI aplikaci pro prostředí Node.js, která usnadňuje časté operace, jako je tvorba nových komponent, nebo správa webového serveru pro vývoj.

AngularJS

První verze Angularu byla vydána v roce 2010 pod názvem AngularJS [16]. Oproti dnešnímu Angularu se jednalo se o velice rozdílný framework, původně byl navržen pro jazyk JavaScript (využíval knihovnu JQuery). V roce 2016 byl framework od základu přepracován a vydán jako Angular 2. Tato verze byla zcela nekompatibilní s původní verzí, ale aby se zamezilo potížím s podporou, byl původní AngularJS nadále vyvíjen jako zcela samostatný framework. V současné době již vývoj AngularJS není aktivní a jeho stávajícím uživatelům se doporučuje provést migraci na nejnovější verzi Angularu.

S frameworkem AngularJS mám velmi dobré zkušenosti, a proto jsem se rozhodl využít jeho nástupce pro tvorbu front-endu.

TypeScript

Jedná se o rozšíření [17] jazyka JavaScript, které vyvinula firma Microsoft. Rozšíření přidává do JavaScriptu statické typování a některé principy objektově orientovaného programování, např. třídy a rozhraní. Jazyk byl navržen pro použití ve velkých projektech a v těchto případech znatelně zlepšuje přehlednost kódu, protože díky statickému typování je na první pohled zřejmé, jak jsou na sobě jednotlivé části kódu závislé. Protože webové prohlížeče podporují pouze jazyk JavaScript, je do něj nutné při sestavení TypeScript překompilovat.

[16] https://angularjs.org/
[17] https://www.typescriptlang.org/

Bootstrap

Jedná se o CSS framework [18] sloužící k tvorbě responzivních webových aplikací. Jedná se o sbírku CSS souborů, které obsahují předdefinované styly často používaných komponent (např. navigace, dialogové okno, atd.). Díky tomu jej lze použít jak pro aktivní tak statické stránky. Předdefinované styly reagující na rozměry okna zaručují, že stránky budou použitelné na většině zařízení.

Mimo Bootstrap existuje řada dalších podobných frameworků, například Foundation, Materialize, Bulma a další. V této aplikaci je použita knihovna *ng-bootstrap*, která zpřístupňuje některé Bootstrap komponenty jako Angularovské komponenty.

Chart.js

Jedná se o JavaScriptovou knihovnu určenou pro vizualizaci dat [19]. Vizualizaci značně zjednodušuje, protože stačí nadefinovat data, a knihovna se sama postará o jejich optimální vykreslení. Bohužel toto zaměření na jednoduchost je omezující v případě, že je potřeba vizualizace méně častých typů dat. V této aplikaci byla zpočátku tato knihovna použita pro vizualizaci grafu, ale v průběhu vývoje od ní bylo upuštěno ve prospěch knihovny D3.js (viz sekce 3.4.2).

D3.js

Stejně jako Chart.js se jedná o knihovnu určenou pro vizualizaci dat [20]. Samotná knihovna neobsahuje žádné předdefinované datové rozložení, namísto toho přenechává definici zobrazení na vývojáře. Tato definice probíhá pomocí napojení dat na SVG elementy skrze CSS selektory. Toto ruční napojování umožňuje vývojářům vytvořit vizualizaci libovolného druhu dat.

Porovnání knihoven d3.js a Chart.js

Na počátku vývoje byla pro vizualizaci používána knihovna *Chart.js*, která byla později nahrazena knihovnou *d3.js*. K této změně došlo z několika důvodů. Knihovna *Chart.js* je primárně určena pro vykreslování statických diagramů, přičemž síťový diagram, kterým by šlo graf vizualizovat, není v základní konfiguraci dostupný. Vizualizace grafu lze docílit skrze plugin *chartjs-chart-graph* a interakce s uživatelem skrze *chartjs-plugin-dragdata*.

[18] https://getbootstrap.com/
[19] https://www.chartjs.org/
[20] https://d3js.org/

Tato funkcionalita však není v pluginech implementována od základu, nýbrž tyto dva pluginy využívají právě metod z knihovny *d3.js*. Knihovna *Chart.js* se tak stává z části redundantní.

Největší výhodou *Chart.js* oproti *d3.js* je zjednodušená definice dat. Zatímco v *d3.js* je nutné ručně definovat, jak se mají jednotlivé druhy dat vykreslovat (viz sekce 4.3.5), při použití *Chart.js* stačí knihovně předat data v předem definované struktuře a vybrat, o jaký druh diagramu se jedná. Knihovna sama se postará o jejich inteligentní vykreslení.

Tato přívětivost byla ale v případě vytvářené aplikace omezující, protože umožňuje měnit pouze některé předpřipravené vlastnosti vizualizace. Aby bylo možné vizualizaci upravit kompletně, bylo by nutné vytvořit vlastní plugin do `Chart.js`, který by definoval, jak se má vykreslovat graf. To by znamenalo provedení přibližně stejného množství práce jako při použití knihovny `d3.js`. Knihovna `d3.js` není v tomto ohledu zdaleka tak omezující, protože umožňuje navázat libovolná data na livovolné DOM prvky. O vizualizaci se pak stará sám prohlížeč, který dané prvky vykresluje.

Nakonec je nutné zmínit, že zatímco knihovna *d3.js* pracuje s DOM, *Chart.js* vykresluje výsledky do HTML5 prvku `canvas`. Tento přístup má výhody i nevýhody. Teoreticky je vykreslování do `canvas` rychlejší než vykreslování SVG, ale tato výhoda je při aplikaci na vykreslování grafů téměř neznatelná [9]. Nevýhodou vykreslování do `canvas` je například nemožnost jednoduše detekovat uživatelskou interakci nad specifickým vrcholem, protože samotný `canvas` prvek neobsahuje informace o vrcholech, které vykresluje.

3.5 Kontejnerové systémy

Kontejnerové systémy jsou používány řadou webových aplikací pro nasazení a izolaci jejich jednotlivých instancí a části. Zde se nachází stručný popis základní funkčnosti kontejnerového systému.

Kontejner

Při použití kontejnerového systému neběží aplikace přímo pod operačním systémem, ale místo toho se nachází v izolovaném prostředí, které obsahuje všechny její závislosti. Toto prostředí nazýváme kontejner a pod jedním operačním systémem může běžet více kontejnerů. Tento systém je podobný virtualizaci, ale na rozdíl od ní jednotlivé kontejnery neobsahují vlastní operační systém, všechna volání na funkce OS z kontejneru jsou kontejnerovým systémem předávána na OS hostitele. Díky tomu, že není nutné v kontejneru

virtualizovat celý další OS, je takový systém méně náročný na prostředky než standartní virtualizace.

Obraz

Další výhoda je jednoduchost nasazení, kdy není nutné server předem konfigurovat, stačí na něj pouze nainstalovat kontejnerový systém. Nasazení pak probíhá pomocí tzv. obrazů, které obsahují vzor, ze kterého se nový kontejner vytvoří. Obrazy lze obvykle stahovat ze vzdáleného repositáře nebo načítat ze souborů a z jednoho obrazu lze vytvořit několik kontejnerů.

Existující systémy

V současné době je nejpoužívanějším systémem systém Docker [21], proto jsem se rozhodl v aplikaci používat právě tento systém. Mezi další systémy patří například ContainerD, Podman, CRI-O apod.

[21] `https://sysdig.com/blog/sysdig-2021-container-security-usage-report/`

4 Implementační detaily

Protože aplikace je rozdělena do dvou separátních částí, je i tato kapitola rozdělena do několika sekcí. První sekce popisuje rozdělení projektových souborů aplikace. V sekcích 4.2 a 4.3 je rozepsán způsob implementace back-endu a front-endu. Poslední sekce 4.4 se zabívá problematikou nasazení aplikace v Dockeru.

4.1 Struktura projektu

V kořenové složce projektu (značeno **/**) najdeme množství souborů, které byly z větší části automaticky vygenerovány používanými nástroji. Kód aplikace samotné se nachází pouze ve složkách **/src/main/java/**, která obsahuje kód back-endu, a **/src/main/angular**, která obsahuje kód front-endu.

4.1.1 Sestavení aplikace

Projekt se sestavuje pomocí nástroje *Maven*. Tento nástroj dokáže automaticky stáhnout prostředí *Node.js* včetně nástroje *NPM*, tudíž je možné celou aplikaci sestavit bez nutnosti ručně instalovat tyto dodatečné balíky.

Sestavení aplikace probíhá tak, že se napřed stáhnou požadované balíky a nástroje. Následně se provede sestavení front-endové části aplikace. Sestavená data se vloží do složky statického obsahu. Následně se provede sestavení back-endu. Nakonec se aplikace zabalí do spustitelného JAR archivu. Tento archiv obsahuje jak kód aplikace, tak embedded Tomcat server a statický obsah, který je následně tímto serverem servírován jako front-end aplikace.

Aplikaci jde sestavit ve dvou konfiguracích. Výchozí je konfigurace 'test', která obsahuje embedded MongoDB databázi. Tato databáze není perzistentní, ale je užitečná při testování aplikace. Druhá konfigurace má název 'prod', a neobsahuje embedded databázi, místo toho využívá databázi externí. Tento profil je určený pro použití v Docker kontejneru (viz sekce 4.4).

4.2 Implementace back-endu

Protože bylo pro back-end použito nástroje Spring Boot, byla po jeho spuštění automaticky vytvořena již funkční základní webová aplikace. Do této aplikace jsem přidal další funkcionalitu pomocí dodatečných tříd.

4.2.1 Nastavení

Nastavení databáze

Toto nastavení se nachází v souboru **`/config/MongoConfiguration.java`**. Nastavení je rozděleno do dvou tříd, každá obsahuje odlišné nastavení. Tato nastavení jsou Springem aplikována na základě jeho současného profilu. Profil je načítán z `.properties` souboru, který je generován při sestavení.

Profil 'prod' je aplikován v případě, že je aplikace sestaveno pomocí 'prod' konfigurace. Tato konfigurace pouze udává jméno databáze, a adresu, na které bude vzdálená databáze dostupná.

Profil 'test' je složitější, protože nezná adresu databáze předem. Místo toho musí počkat na spuštění embedded databáze (viz sekce 4.1.1) a až poté zjistit, na jakém portu databáze naslouchá. Tato konfigurace dále využívá třídu *TestSeeder*, která se stará o vložení defaultních dat do databáze.

Nastavení bezpečnosti

Prvním úkolem bylo nakonfigurovat back-end tak, aby se zabránilo možnosti připojit se na něj z venčí. Toto nastavení je provedeno v souboru **`/config/SecurityConfiguration.java`**. Zde je rozšířena třída *WebSecurityConfigurerAdapter*, která se stará o zabezpečení. Vzhledem k tomu, že String implementuje IoC, nebylo nutné provádět změny nikde jinde, framework si tuto konfiguraci najde sám za běhu.

Objekt vracený metodou *corsConfigurationSource()* se stará o kontrolu CORS headerů. Tyto headery slouží k omezení komunikace s neznámými aplikacemi, protože aplikace automaticky odmítá žádosti ze zdroje, který není povolen. Povolené headery nastavíme pouze na `localhost:8080`, což umožňuje komunikaci pouze s aplikací běžící na stejném stroji a komunikující na stejném portu jako back-end. Rovněž zde povolíme HTTP autentikaci.

V metodě *configure(HttpSecurity http)* nastavíme přístup na jednotlivé API end-pointy podle práv uživatele. Také zde nastavíme, jaké odpovědi má back-end vracet po vyřízení nebo selhání požadavku. Dále je zde nastavena ochrana proti CSRF [1]. Té docílíme odesláním tokenu skrze cookies při prvním pokusu o připojení. Pokud klient pošle v následujících požadavcích tento token zpět, víme, že se skutečně jedná o klienta, který si o připojení zažádal.

Ostatní vlastnosti, metody a třídy definují další detaily zabezpečení, jako je například způsob šifrování hesel uživatelů, nebo službu, která ověřuje uži-

[1] Cross Site Request Forgery - zranitelnost, při které se do kódu stránky vloží skript třetí strany, čímž se obejde autentizace uživatele

vatelská data.

Autentizace uživatelů

Autentizace uživatelů probíhá skrze základní HTTP autentizaci. Po přijetí požadavku o autentizaci si back-end zjistí detaily uživatele pomocí služby `MongoUserDetailsService`. Pokud služba daného uživatele nenajde, nebo se přihlašovací údaje neshodují, pak back-end vrátí odpověď s chybovým stavem. Pokud se podaří uživateli přihlásit, vrátí se mu v odpovědi cookie, která udává identifikátor jeho sezení.

4.2.2 Řešení požadavků

O řešení jednotlivých požadavků na REST API se starají třídy ze složky **`/controllers/`**. Každá z těchto tříd řeší požadavky na specifickou část API. Požadavky, které zde nejsou definované vrací klientovi chybový status a neprovádějí žádné změny.

REST API je vytvořeno pomocí Springem definovaných anotací. Anotace typu `@RestController` říká, že se jedná o třídu starající se o REST požadavky, a `@RequestMapping` udává, na jakém end-pointu třída požadavky řeší. Anotace nad jednotlivými Java metodami pak udávají URL a typ HTTP požadavku, který daná metoda využívá.

Samotné metody jsou spouštěné Springem a ten do nich dodává argumenty z několika zdrojů na základě jejich anotace, případně datového typu. Argumenty se přebírají z URL pomocí `@PathVariable`, obsahu požadavku (`@RequestBody`), parametru požadavku (`@RequestParam`) nebo lze využít argument datového typu `Principal`, do kterého se vkládá současný uživatel.

Data pocházející z obsahu požadavku se automaticky konvertují z formátu JSON na standartní Java objekty dle typu specifikovaném v hlavičce metody. A stejně tak se data, které metoda vrací, automaticky konvertují na JSON. Definice jednotlivých datových modelů se nacházejí ve složce **`/models/`**. Tyto modely se používají i při komunikaci s databází (viz sekce 4.2.3).

Pokud při vyřizování požadavku nastane chyba, je vyhozena výjimka typu `ResponseStatusException`. Spring následně vrací odpověď na základě obsahu této výjimky. Z pohledu klienta má úspěšná odpověď serveru vždy HTTP kód 200 a jejím případným obsahem je objekt v JSON formátu.

4.2.3 Komunikace s databází

K tomuto účelu slouží technologie *Spring Data Repositories*, která minimalizuje množství práce, kterou je nutné provést, aby bylo možné s databází komunikovat. Pro použití této technologie je nutné napřed provést nastavení datových repositářů (viz sekce 4.2.1), a následně nadefinovat rozhraní, která se budou pro komunikaci využívat. Tato rozhraní se nacházejí ve složce **`/repos/`**. Každé rozhraní zde rozšiřuje rozhraní `MongoRepository`, které pracuje s databází *MongoDB*.

Do rozhraní je možné nadefinovat dodatečné metody, které lze použít pro manipulaci s daty. Spring vytváří instance objektů implementujících tyto instance automaticky za běhu. Jednotlivé dotazy na databázi se generují za pomoci názvů a vstupních/výstupních typů jednotlivých metod rozhraní. Například pro metodu `User findByEmail(String email)` by se vygeneroval dotaz vyhledávající v databázi uživatele dle jeho emailu. Jelikož instance jednotlivých rozhraní vytváří Spring automaticky, není nutné psát jejich implementace ručně a lze je následně používat v dalších částech back-endu pomocí IoC.

Definice jednotlivých datových modelů se nacházejí ve složce **`/models/`**. Anotace `@Document` u každého modelu značí jméno kolekce, do které se v databázi data používající daný model uloží.

4.2.4 Správa knihoven

Aby bylo možné uživatelem dodávané knihovny v aplikaci používat, je nutné znát cestu ke spustitelnému souboru knihovny a parametry potřebné pro jeho spuštění. K tomuto účelu slouží datové typy `LibraryPath` a `LibraryModel`. Samotné soubory knihoven jsou uloženy do souborového systému serveru. Data typu `LibraryPath` obsahují ID knihovny a cestu ke spustitelného souboru. Data typu `LibraryModel` obsahují jméno a popis knihovny, případně další parametry nutné k jejímu spuštění. Toto rozdělení teoreticky umožňuje pro jeden soubor knihovny vytvořit několik metod spuštění, ale tato funkcionalita není v tuto chvíli implementována.

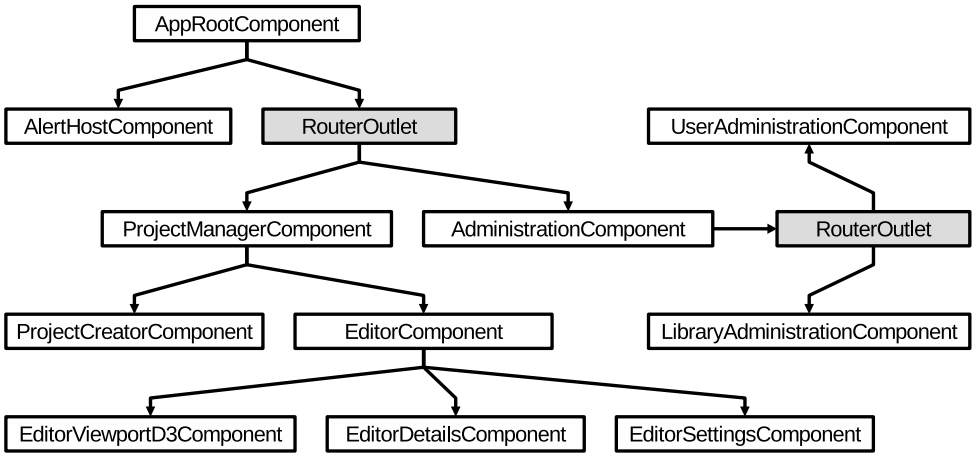
Tato třída dále umožňuje získat z knihovny informace o podporovaných parametrech spuštění za pomoci jejího spuštění se speciálním parametrem **-helpMachine**. Pokud knihovna podporuje tento parametr, pak je jejím výstupem seznam ostatních podporovaných parametrů ve formátu, který je popsán v příloze B.

Samotné spuštění knihovny probíhá ve třídě `LibraryRunnerService`, která JAR soubor knihovny spouští jako externí proces a komunikuje s ním

za pomoci přesměrování vstupu a výstupu. Komunikační protokol je založen na CSV formátu (viz příloha B).

4.3 Implementace front-endu

Jak již bylo nastíněno v kapitole 3, tak front-end byl implementován pomocí frameworku Angular. Projekt je tak tvořen hierarchií komponent (viz obr. 4.1), kde každá komponenta představuje specifickou část uživatelského rozhraní.



Obrázek 4.1: Hierarchie primárních Angularovských komponent.

4.3.1 Základy frameworku Angular

Vzhledem k tomu, že Angular může být pro začínající vývojáře matoucí, zde jsou stručně popsány jeho základní koncepty a další funkcionalita, která je v aplikaci využívána. Na koncepty a funkce zde obsažené bude odkazováno v následujících kapitolách.

Moduly, komponenty a služby

Modul frameworku Angular je založený na modulech známých z novějších verzí jazyka JavaScript. Moduly udávají informace o funkcionalitě, kterou poskytují nebo vyžadují. Angular modul se definuje pomocí dekorátoru `@NgModule`, do kterého se zadá seznam modulů a komponent, které daný modul využívá

nebo vystavuje. Každá Angular aplikace má alespoň jeden tzv. kořenový modul, který udává komponentu sloužící jako základ webové stránky. V případě naší aplikace se jedná modul `AppModule` a komponentu `AppRootComponent`.

Z modulů lze také do aplikace přidat dodatečnou funkcionalitu, například modul `RouterModule` obsahuje komponenty a služby týkající se směrování skrze URL.

Komponenta je základní jednotka uživatelského rozhraní, která se skládá z HTML šablony a funkčního kódu. V kódu se komponenta definuje pomocí dekorátoru `@Component`, kterému se předá odkaz na soubor HTML šablony, nebo se HTML zapíše rovnou do jejího kódu. Dále může mít komponenta tzv. selektor, který říká, že pokud se při sestavování v HTML nalezne prvek s daným typem, bude nahrazen danou komponentou. Například pokud se v HTML naší aplikace nalezne prvek `<graph-editor>`, bude tento prvek nahrazen komponentou `EditorComponent`.

Služba představuje data a funkcionalitu sdílenou mezi více komponentami. Definuje se pomocí dekorátoru `@Injectable` a do jednotlivých komponent se obvykle vkládá skrze jejich konstruktor. Například v naší aplikaci služba `SessionService` obsahuje informace o současném uživateli a tuto službu používá několik komponent.

Direktivy

Angular dodává jednotlivým HTML prvkům dodatečnou funkcionalitu pomocí tzv. direktiv. Direktivy se prvkům přidávají v HTML šabloně, kdy se prvku přidá speciální předdefinovaný atribut. Direktivy dělíme na direktivy strukturální, které upravují rozložení stránky tím, že do ní přidávají a odstraňují prvky, a atributové, které pouze upravují stávající prvky.

V naší aplikaci používáme dvě strukturální direktivy. Direktiva `*ngIf` vytváří daný prvek pouze pokud je splněna v ní uvedená podmínka. Například direktiva `*ngIf="project.ready"` vytvoří prvek, na který je aplikována pouze v případě, že hodnota `project.ready` bude pravdivá. Druhou strukturální direktivou, která se v naší aplikaci nachází, je `*ngFor`, která vytváří řadu kopií daného prvku na základě určité hodnoty. Například direktiva `*ngFor="let project of projects"` zopakuje prvek tolikrát, kolik je hodnot v seznamu `projects`. Hodnota `project` každého prvku je pak unikátní a rovna hodnotě načtené ze seznamu.

Atributové direktivy mohou prvek upravit mnoha způsoby. Například mu mohou přidat novou funkci, což si lze ukázat např. na použití direktivy `routerLink`, která pochází z `RouterModule` a umožňuje po kliknutí na prvek přesměrovat prohlížeč na URL v ní uvedenou. Dalším příkladem, kde lze

tyto direktivy použít, je například změna vzhledu prvku na základě nějaké proměnné skrze direktivy `ngStyle`, nebo `ngClass`.

Datové vazby

Aby bylo možné aktualizovat hodnoty na stránce automaticky, je nutné napřed nastavit datové vazby. Datová vazba udržuje data komponenty v souladu s daty na stránce tak, že při změně dat provede opětovné sestavení HTML šablony komponenty. K vytvoření vazby lze využít atributových direktiv.

Jednosměrnou vazbu, která aktualizuje data na stránce dle komponenty, ale nikoliv opačně, lze definovat na specifické vlastnosti daného prvku pomocí hranatých závorek. Například pro HTML prvek tlačítka lze použít `[disabled]="!isValid"`, který nastaví atribut `disabled` podle hodnoty proměnné `isValid`. Tento typ vazeb lze také zapisovat to těla prvku pomocí výrazu `{{ }}`. Například do prvku `<span>{{ project.title }}</span>` by se vložilo jméno projektu.

Dále lze tvořit vazby reagující na události pomocí kulatých závorek, například `(click)="send()"` zavolá v komponentě metodu `send` po stisknutí tlačítka.

Posledním typem vazeb jsou obousměrné vazby, kdy Angular načítá hodnoty z prvku, ale rovněž aktualizuje prvek, pokud nastane změna dat v komponentě. Příkladem může být vazba `[(ngModel)]="projectId"`, která načítá z prvku ID projektu.

Vazby lze také používat k výměně dat mezi komponentami. V komponentě lze nadefinovat libovolnou proměnnou jako vstupní nebo výstupní pomocí atributu `Input`, resp. `Output`. Následně můžeme na tyto proměnné nasadit vazby v šabloně nadřazené komponenty. Za použití šablonových proměnných lze vyměňovat data i mezi sousedícími komponentami.

Šablonové proměnné a fragmenty

Šablonové proměnné slouží k předávání dat v HTML šabloně komponenty bez nutnosti tato data definovat v jejím kódu. Proměnnou definujeme tak, že libovolnému prvku HTML šablony přidáme atribut začínající znakem `#`. Na takto označený prvek se následně můžeme odkazovat v atributech sousedících prvků šablony. Například pokud označíme HTML vstup atributem `#nameInput`, můžeme jeho obsah zobrazit v jiné části šablony pomocí `{{ nameInput.value }}`. Dále lze takto označený prvek předat do kódu komponenty pomocí proměnné s dekorátorem `ViewChild`.

Šablonové fragmenty definují části HTML šablony, které se normálně nevykreslují. Pokud je použijeme společně se šablonovými proměnnými, můžeme tak nadefinovat HTML, které se zobrazuje v jiné části komponenty.

4.3.2 Rozvržení stránky

Při návrhu rozhraní jsem se snažil, aby editor grafu zabíral co nejvíce místa na stránce, protože se jedná o tu nejdůležitější část aplikace a je tak nezbytně nutné, aby byl přehledný. Finální rozvržení stránky lze vidět na obrázku 4.2. Všechny komponenty obsažené v tomto diagramu budou detailněji popsány v následujícím textu.

Obrázek 4.2: Vizualizace jednoduchého grafu.

Komponenta `AppRootComponent` je kořenovou komponentou aplikace. Tato komponenta obsahuje HTML kód záhlaví, ve kterém se nachází navigace mezi jednotlivými částmi aplikace, komponentu `RouterOutlet`, která v sobě zobrazuje obsah vybrané části, a komponentu `AlertHostComponent`, která zobrazuje upozornění. Aplikace je rozdělena na projektovou část, kde lze vytvářet, upravovat a vizualizovat grafy, a administrativní část, kde lze manipulovat uživatele a knihovny. Do administrativní části mají přístup pouze uživatelé s právy administrátora. Pokud uživatel není administrátor, navigační rozhraní se mu nezobrazí, čehož je docíleno direktivou `*ngIf`.

V záhlaví se dále nachází tlačítka pro přihlášení, odhlášení a registraci uživatele. Tato tlačítka po stisknutí vytvářejí Bootstrap dialog, který žádá po uživateli přihlašovací údaje. Při pokusu o přihlášení ze zobrazí dialog

obsahující komponentu `LoginModalComponent`, která žádá od uživatele přihlašovací jméno a heslo. Pokud se uživatel rozhodne zaregistrovat nový účet, zobrazí se dialog obsahující komponentu `UserDetailsModalComponent`, do které lze vypsat informace o uživateli.

Zobrazení obsahu je implementováno pomocí modulu `RoutingModule`, který je součástí Angularu. Tento modul obsahuje služby a komponenty, které umožňují zobrazovat různé komponenty na základě aktuálního URL. Nastavení směrování je provedeno v modulu `AppModule`. Tento modul slouží jako vstupní bod aplikace a obsahuje deklarace všech používaných komponent a služeb. Směrování je zde provedeno pro URL začínající `/projects` a `/administration` na komponentu obsahující projektový manažer, resp. administrativní prostředí. Ostatní URL jsou přesměrovány na `/projects`. V HTML šabloně `AppRootComponent` se pak nachází Angularovská komponenta `RouterOutlet`, do které se budou dané komponenty zobrazovat.

4.3.3 Projektový manažer

Jelikož uživatel může mít otevřeno několik projektů současně, komponenta `ProjectManagerComponent` se stará o to, aby s nimi mohl uživatel volně manipulovat.

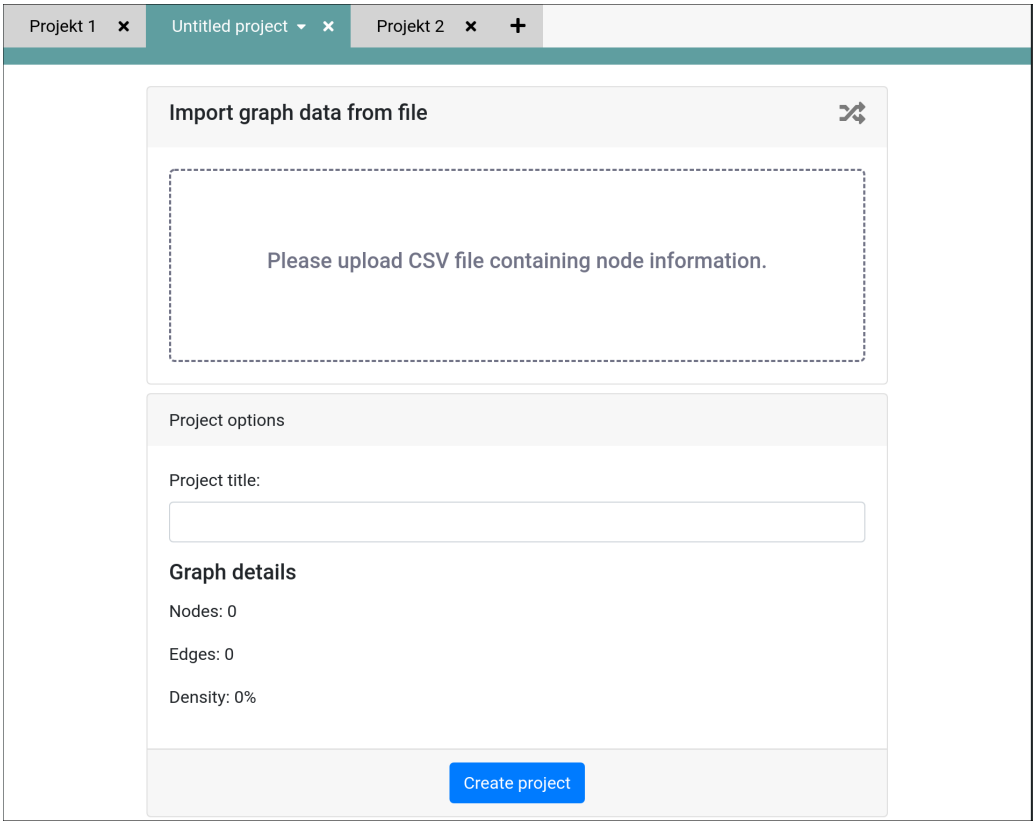
Správa projektů

Aby bylo možné zachovávat stav projektů i po případném odstranění komponenty projektového manažera, současný stav projektů se nachází ve službě `ProjectManagerService`. Samotná komponenta pak spoléhá na tuto službu v případě, že je nutné seznam projektů upravit.

Seznam projektů je v rozhraní reprezentován sadou záložek, které se chovají podobně jako záložky v prohlížeči (viz horní část obrázku 4.3). Po kliknutí na záložku se zobrazí její obsah. Jednotlivé záložky lze řadit pomocí táhnutí myší, čehož je docíleno pomocí knihovny *sortable.js*. Jednotlivé záložky se generují automaticky na základě seznamu projektů pomocí direktivy `*ngFor`. V poslední záložce se vždy nachází tlačítko, skrze které je možné vytvořit nový projekt. Po jeho stisknutí se do seznamu přidá nový prázdný projekt a zobrazí se průvodce vytvořením nového projektu.

Po výběru nějakého projektu se pod záložkami zobrazí jeho obsah. Tento obsah může být tvořen buď editorem grafu, nebo průvodcem vytvoření nového projektu v případě, že se jedná o nově vytvořený projekt. Přepínání mezi těmito komponentami probíhá pomocí direktivy `*ngIf`.

Záložka právě aktivního projektu obsahuje menu, ze kterého lze současný projekt exportovat, přejmenovat, případně uložit na server. Export projektu



Obrázek 4.3: Lišta správce projektů (nahoře) a průvodce vytvořením nového projektu.

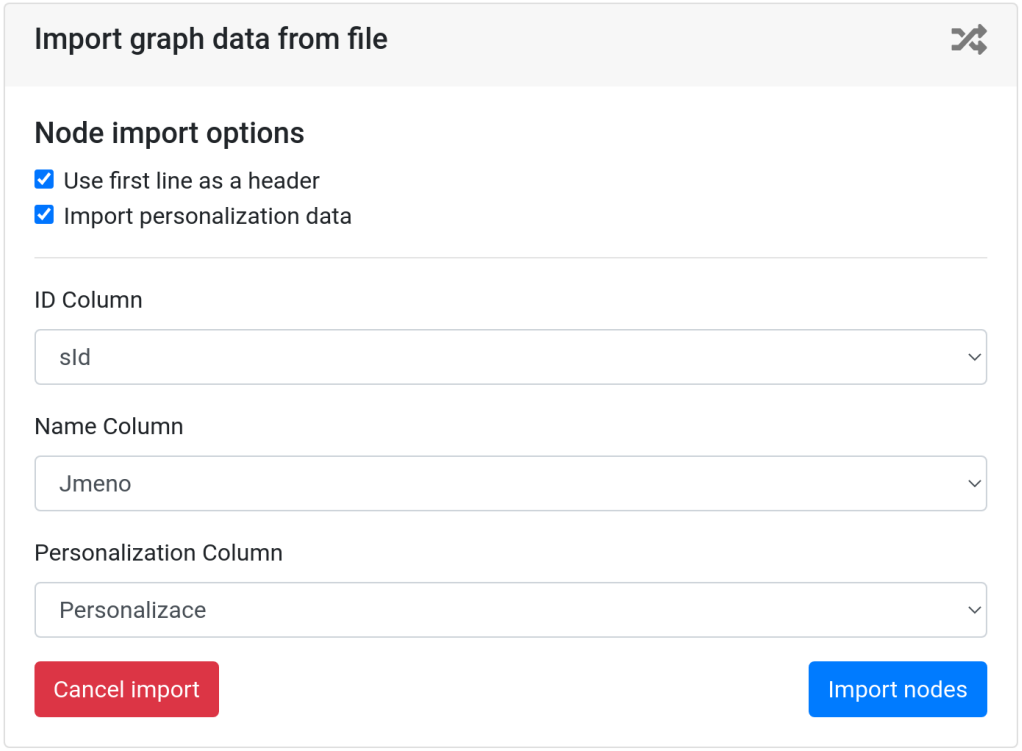
se provádí přes službu `ProjectSaverService`, která vytváří *.zip* archiv, ve kterém se nachází CSV soubory vrcholů a hran daného grafu. K přejmenování projektu slouží komponenta `TextPromptModalComponent`, která obsahuje generický textový vstup, do kterého lze v tomto případě zadat nové jméno projektu. V každé záložce se dále nachází tlačítko, které daný projekt odstraní se seznamu projektů.

V případě že je uživatel do aplikace přihlášen, jsou do menu skrze direktivu `*ngIf` přidány položky, kterými lze uložit nebo načíst exitující projekt z databáze na serveru. Načítání ze serveru je implementováno v komponentě `ProjectSelectionModalComponent`, kde si uživatel může zvolit, který ze svých dříve uložených projektů chce načíst. Tato komponenta se zobrazuje jako obsah dialogu generovaného knihovnou *ng-bootstrap*.

Průvodce vytvořením nového projektu

Komponenta `ProjectCreatorComponent` obsahuje průvodce, ze kterého je do aplikace možné importovat existující graf ze souborů.

Rozložení této komponenty jde vidět na obr. 4.3. V horní polovině komponenty se nachází oblast do které lze nahrát soubory grafu. Tato oblast je vytvořena pomocí knihovny *ngx-dropzone*. Tato knihovna umožňuje vytváření oblastí, do kterých lze myší přetáhnout soubory z klientského počítače. Tyto soubory musí být ve formátu CSV a pro každý graf je potřeba dvou souborů. V prvním souboru se nachází seznam jednotlivých vrcholů a ve druhém se nachází seznam hran v grafu. Vždy po nahrání souboru se oblast dropzone nahradí sadou přepínačů, ve které lze nastavit dodatečné vlastnosti importu (viz obr. 4.4). Pomocí těchto přepínačů lze například změnit pořadí sloupců v souboru, nebo nastavit, zda se mají ze souboru importovat dodatečné hodnoty.



Obrázek 4.4: Dodatečné možnosti po nahrání souboru obsahujícího vrcholy.

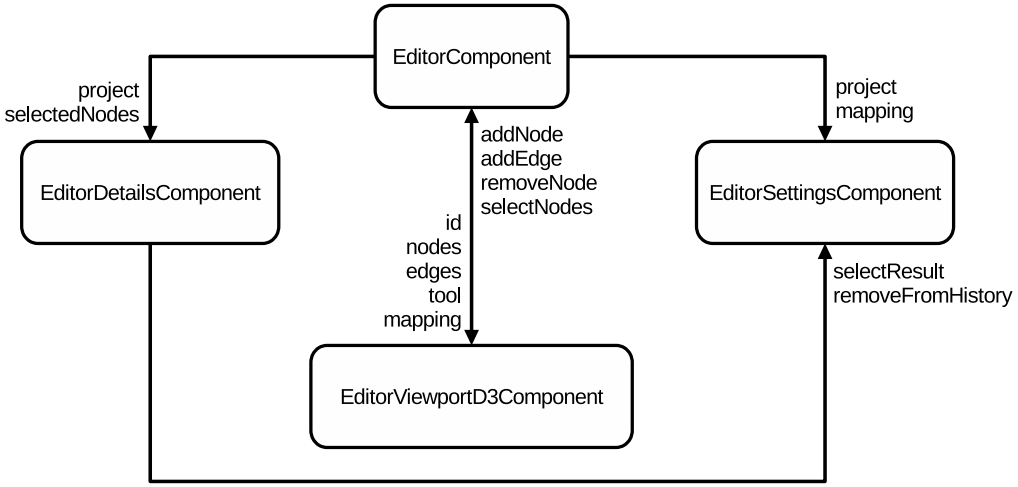
V dolní části průvodce se nachází pole pro pojmenování projektu, tlačítko pro jeho vytvoření a statistiky počtu vrcholů a hran v grafu. Pro vytvoření grafu není nutné aby graf obsahoval nějaké vrcholy, je možné vytvořit i prázdný graf. Po stisknutí tlačítka vytvoření projektu se průvodce uzavře a je nahrazen editorem grafu, viz obr. 4.2.

V horním pravém rohu komponenty se dále nachází tlačítko, které otevírá dialog, ze kterého je možné vygenerovat náhodný graf. Tato funkcionalita je užitečná například při testování aplikace. Do dialogu lze zadat počet vrcholů a maximální počet hran, které mohou z jednoho vrcholu vézt. Algoritmus, který graf generuje, je velmi jednoduchý, a nezaručuje, že graf bude spojitý. Snaží se jen z každého vrcholu vytvořit náhodný počet hran na ostatní vrcholy s náhodnou váhou.

Dialog generování náhodného grafu je vytvářen pomocí knihovny *ng-bootstrap*, ale tentokrát se mu nepředává odkaz na separátní komponentu. Na místo toho se mu předá odkaz na šablonový fragment, který je následně v dialogu zobrazen. Generování grafu nadále probíhá v komponentě `ProjectCreatorComponent`.

4.3.4 Editor grafu

Jedná se o nejsložitější část aplikace, která je tvořena několika navzájem provázanými komponentami. Diagram 4.5 ukazuje vazby mezi těmito komponentami. Vazby jsou provedeny v HTML šabloně `EditorComponent` pomocí šablonových proměnných.



Obrázek 4.5: Diagram vazeb mezi komponentami editoru.

Komponenta `EditorComponent` se stará o základní funkcionalitu editoru. Obsahuje funkce pro přidávání/mazání hran a vrcholů a dokáže nad grafem provádět výpočet ohodnocení vrcholů pomocí externích uživatelských JAR knihoven. Jak jde vypozorovat z obr. 4.2, téměř celou komponentu editoru vyplňuje vizualizační komponenta, která je detailně popsána v sekci 4.3.5. Na levé straně editoru se nachází soubor tlačítek, která slouží k manipulaci grafu. Horní modrá skupina tlačítek představuje různé nástroje manipulace grafu, které slouží k výběru, přidávání a mazání částí grafu. Šedé prostřední tlačítko otevírá dialog, ze kterého lze spustit externí knihovnu.

Zelená dolní tlačítka spouští přepočet rozložení grafu (viz sekce 4.3.5) a otevírají panel nastavení editoru. Tlačítko na pravé straně otevírá panel vlastností vybraného objektu. Tyto panely jsou řešené separátními komponentami `EditorDetailsComponent` a `EditorSettingsComponent`.

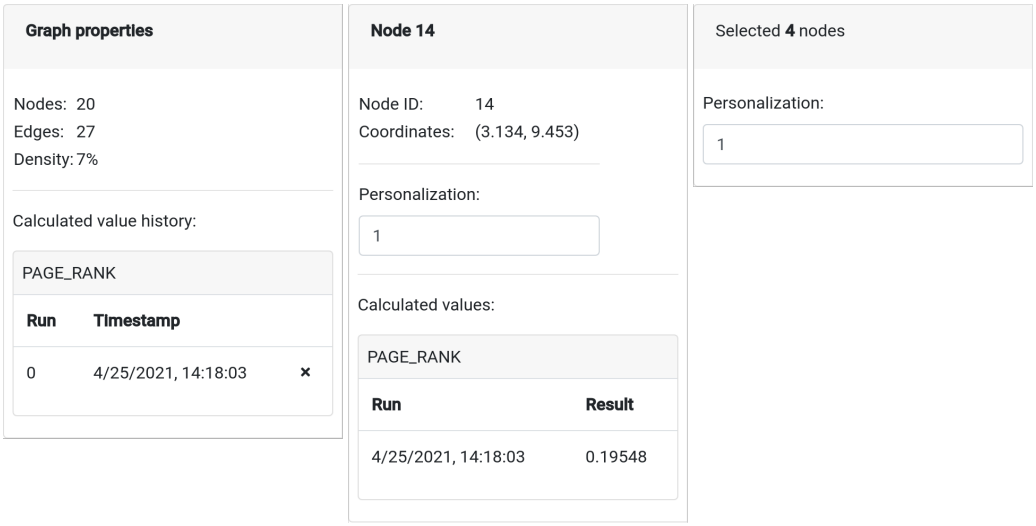
Výběr externí knihovny

Výběr, zadání parametrů a spuštění externí knihovny v sobě implementuje komponenta `LibrarySelectionModalComponent`. Seznam knihoven je načí-

tán z back-endu a uživateli je prezentován jako výběr z listu. Poté, co si uživatel vybere požadovanou knihovnu, zobrazí se v komponentě seznam možných parametrů knihovny. Tento seznam je generovaný direktivou `*ngFor`. Po zadání povinných parametrů může uživatel knihovnu spustit. Dialog spuštění se uzavře poté, co z back-endu dorazí výsledek.

Panel vlastností

Komponenta `EditorDetailsComponent` slouží k zobrazení všech informací o objektech obsažených ve výběru editoru. Výběr se získává z nadřazené komponenty `EditorComponent`. Při výběru můžou nastat tři situace, při každé z nich se v tomto panelu zobrazí odlišná data.



Obrázek 4.6: Panel vlastností v následujících případech (zleva doprava): Prázdný výběr, vybrán je jeden vrchol, vybráno je několik vrcholů.

Pokud je vybrán pouze jeden vrchol, zobrazí se v tomto panelu všechny informace o daném vrcholu (viz obr. 4.6 uprostřed). Tabulka hodnot vypočítaných externí JAR knihovnou je interaktivní a po kliknutí na nějaký výsledek se daný výsledek vizualizuje v obsahu editoru. Toho je docíleno změnou aktivního výsledku v komponentě nastavení editoru. Dále je v tomto panelu možné přiřadit danému vrcholu personalizaci. Pokud je vybráno více vrcholů, panel umožňuje přiřadit hodnotu personalizace všem vrcholům najednou (viz obr. 4.6 vpravo).

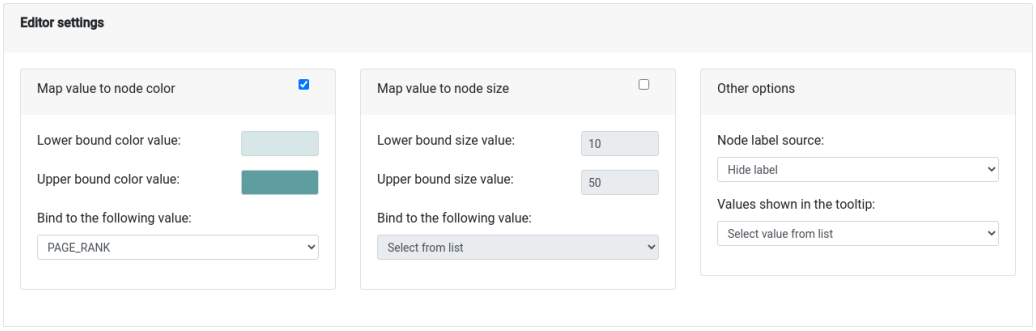
Pokud není vybrán žádný vrchol, panel zobrazuje základní informace o grafu, včetně tabulky, kde je vypsána historie výpočtů externí JAR knihovny

(viz obr. 4.6 vlevo). Tato tabulka je interaktivní, podobně jako tabulka v informacích vrcholu, ale navíc umožňuje mazat jednotlivé průběhy z historie.

Šablona obsahuje všechny tři verze panelu, mezi nimiž se přepíná skrze direktivu `*ngIf`.

Nastavení editoru

V panelu nastavení editoru je možné nastavit, jaké hodnoty se budou vizualizovat a jakým způsobem se bude vizualizace provádět, viz obr. 4.7.



Obrázek 4.7: Panel nastavení editoru.

Komponenta `EditorSettingsComponent` přebírá z informačního panelu typ vizualizované hodnoty a z nadřazené komponenty přebírá objekt, který představuje mapování hodnoty na vlastnost prvku vizualizace. Mapování je možné pouze na barvu a velikost vrcholu, lze ale na každou vlastnost namapovat rozdílnou hodnotu. Nadřazená komponenta tento mapovací objekt předává vizualizační komponentě, která podle něj upravuje vizualizaci. Komponenty výběru barvy pochází z knihovny *ngx-color-picker*.

4.3.5 Vizualizace grafu

K vizualizaci grafu slouží komponenta `EditorViewportD3Component`. Tato komponenta graf vizualizuje pomocí knihovny *d3.js* a informace o uživatelské interakci předává zpět nadřazené komponentě `EditorComponent`.

Použití knihovny d3.js

Knihovna *d3.js* (*Data Driven Documents*) umožňuje na DOM komponenty navázat libovolná data, a následně pomocí těchto dat DOM transformovat. Pro vizualizaci grafu bylo využito SVG elementu jako plátna, do kterého jsou vložené dva kontejnerové prvky. První kontejner obsahuje sadu vrcholů, které

jsou reprezentované SVG kruhem. Druhý kontejner obsahuje hrany, které jsou vizualizované SVG čárou. Dvou kontejnerů je potřeba, aby se zabránilo případům, kdy hrana překrývá vrchol.

O přidávání jednotlivých prvků a změnu jejich vlastností se stará *d3.js*, která ale změny dat nedekuje automaticky. Detekce a aktualizace se provádí ve funkci `doCheck()`, která je volána pokaždé, když je detekován uživatelský vstup nebo změna vstupních dat. Detekce změn v datových objektech se provádí pomocí Angularovských tříd `IterableDiffer` a `KeyValueDiffer`.

Pokaždé když je v datech detekována změna, je prováděn následující sled operací:

1. Provede se výběr všech vrcholů skrze volání metod `D3.selectAll()` a `D3.data()`.

2. Pomocí `D3.exit()` se odstraní vrcholy, které již nejsou součástí dat.

3. Do dokumentu se skrze metodu `D3.enter()` přidají vrcholy, které jsou součástí dat, ale nemají v dokumentu reprezentaci. Pro každý vrchol jsou přidány dva nové prvky pomocí `D3.append()`. Prvním prvkem je SVG kruh, který graficky prezentuje vrchol. Druhým prvkem je SVG text, který slouží jako jeho popisek. Rovněž se zde provedou vazby reakcí na uživatelskou interakci.

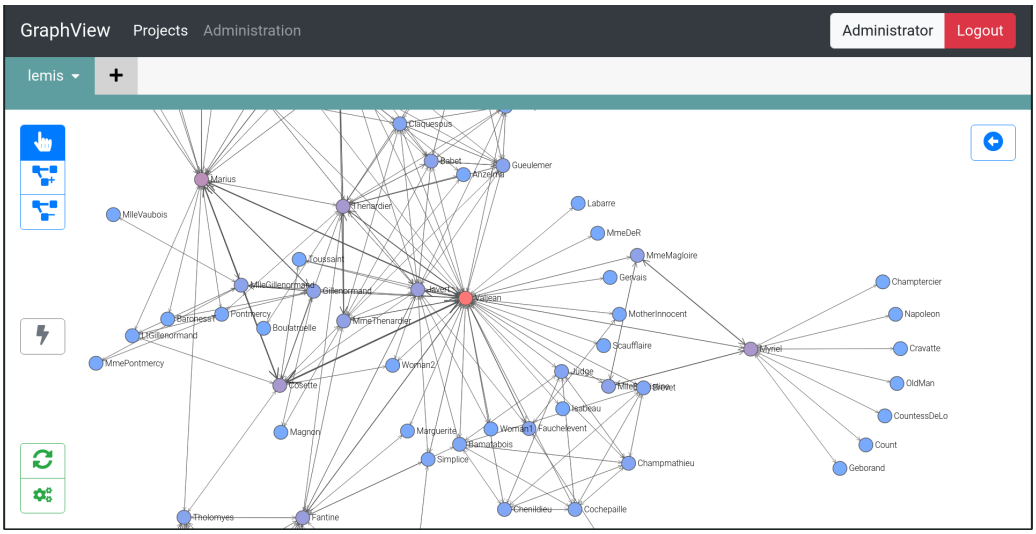
4. Vyberou se všechny vrcholy, které jsou označené jako změněné a provede se aktualizace jejich vlastností pomocí `D3.attr()` a `D3.style()`. Veškeré hrany vedoucí z/do těchto vrcholů se rovněž označí jako změněné.

5. Nakonec se vyberou všechny právě vybrané vrcholy a upraví se jejich styl tak, aby byly vizuálně zvýrazněné.

Aktualizace hran probíhá stejným způsobem, jen je místo kruhu každá hrana reprezentována čárou se šipkou na konci. Některé vypočtené hodnoty vrcholu/hrany mohou být mapovány na hodnotu vizualizačního prvku. Například na obr. 4.8 je tloušťka čáry, která reprezentuje hranu, přímo úměrná její váze a barva vrcholu vizualizuje vypočtenou hodnotu PageRank. Toto mapování je prováděno ve třídě `PropertyMapping`, jejíž vlastnosti jsou definovány v komponentě `EditorSettingsComponent`.

Rozložení vrcholů

Protože načtené nebo vygenerované grafy většinou neobsahují informace o rozložení jednotlivých vrcholů, je nutné vrcholy automaticky rozložit tak,



Obrázek 4.8: Ukázka vizualizace sítě.

aby se navzájem nepřekrývaly a aby byla zřetelně vidět celková struktura grafu.

V aplikaci jsem se rozhodl používat tzv. *Force Directed Layout*, který generuje rozložení vrcholů na základě simulace sil, které na ně působí. Tato simulace je pro knihovnu *d3.js* dostupná skrze plugin *d3-force*. V aplikaci jsem se rozhodl pro výpočet rozložení využít následujících sil:

1. Síla spojující vrcholy skrze hrany, která se stará o to, aby si byly spojené vrcholy co nejblíže.
2. Síla působící směrem do středu plátna, která se snaží o to, aby graf ostatní síly neposunuly mimo obrazovku.
3. Síla odpuzující jednotlivé vrcholy, která se snaží o to, aby byly vrcholy rovnoměrně rozložené.
4. Síla kolize mezi vrcholy, která se snaží o to, aby se spolu jednotlivé vrcholy nepřekrývaly.

Samotný výpočet probíhá iterativně a jeho ukončení je nastaveno na dosažení určené hranice přesnosti. Jednotlivé iterace jsou prováděny asynchronně, aby se zamezilo případnému zamrznutí okna prohlížeče.

Řešení interakce s uživatelem

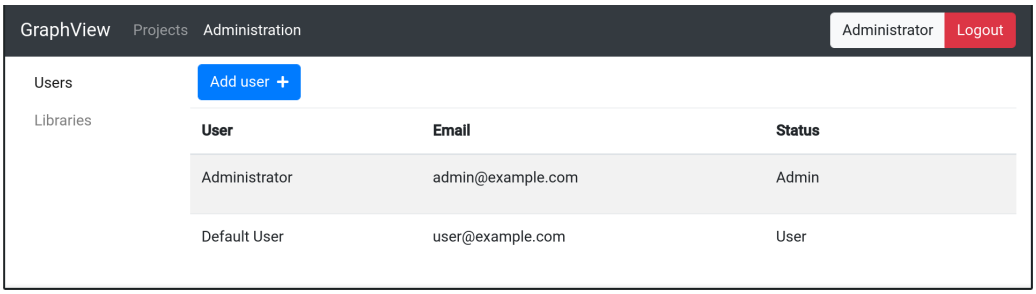
Reakce na uživatelské vstupy se provádějí odposloucháváním událostí na jednotlivých SVG prvcích. V závislosti na nástroji zvoleném v nadřazené

komponentě se při reakci provádějí různé operace. Navázání reakcí na události se v jednotlivých prvcích provádí voláním metody `D3.on()` při jejich přidávání do DOM. Reakce se navazují na následující prvky a události:

- Pohyb a přibližování plátna je řešeno v kořenovém SVG prvku pomocí pluginu *d3-zoom*, který je nastaven tak, aby reagoval na stisk pravého, nebo prostředního tlačítka myši a na její kolečko. Tlačítka myši posouvají obrazovku a kolečko myši ji přibližuje/oddaluje.
- Pokud je v komponentě `EditorComponent` vybrán nástroj pro přidávání vrcholů a hran, tak kliknutím na kořenový SVG prvek se tato komponenta informuje o přidání nového vrcholu na současnou pozici myši.
- Kliknutí na vrchol daný vrchol přidá do výběru `EditorComponent`. V případě že je vybrán nástroj mazání vrcholů a hran se daný vrchol smaže.
- Pokud bylo nad vrcholem detekováno kliknutí a následný posun, tak se daný vrchol přesune na aktuální pozici myši. Případně se provede pokus o vytvoření nové hrany, pokud byl vybrán nástroj pro přidávání vrcholů a hran.

4.3.6 Administrativní část aplikace

V této časti aplikace je možné přidávat, mazat a upravovat uživatele a uživatelské JAR knihovny, které se nacházejí v systému. Na obr. 4.9 je ukázáno rozložení komponenty `AdministrationComponent`, která slouží jako kontejner obsahující komponenty `UserAdministrationComponent` (administrace uživatelů) a `LibraryAdministrationComponent` (administrace knihoven). Na levé straně se nachází menu, které přepíná obsah mezi těmito dvěma komponentami.



Obrázek 4.9: Obrazovka administrace uživatelů.

Přepínání obsahu je implementováno skrze modul `RoutingModule` Je zde využito vnořeného směrování, kdy lze pod každou cestu zapsat její potomky. V případě směrování na URL `/administration` jsou to směrování na `/administration/users` a `/administration/libs`. Tyto vnořené cesty jsou následně delegovány na komponentu `RouterOutlet`, která se nachází v šabloně komponenty `AdministrationComponent`.

Komponenta `UserAdministrationComponent` obsahuje rozhraní pro administraci uživatelů. Ta se provádí pomocí tabulky, kde každý řádek představuje jednoho uživatele v systému. Řádky tabulky se generují automaticky pomocí direktivy `*ngFor`. Po kliknutí na nějaký z řádků se zobrazí dialog obsahující komponentu `UserDetailsModalComponent`, kde lze upravovat uživatelské údaje. V každé řádce se také nachází tlačítko, kterým se daný uživatel ze systému smaže. Po stisknutí tohoto tlačítka se zobrazí dialog s komponentou `ConfirmModalComponent`, do které je nutné ručně napsat, že chceme změnu provést. Nad tabulkou se nachází tlačítko, kterým lze vytvořit nového uživatele.

Administrace knihoven funguje podobně jako administrace uživatelů, jen se v tabulce místo uživatelů nacházejí jednotlivé knihovny a k případným úpravám se využívá komponenty `LibraryDetailsModalComponent`. Tato komponenta obsahuje dropzone, kam lze nahrát JAR soubor knihovny. Po jeho načtení se soubor automaticky odešle na back-end, který se pokusí z něj načíst vstupní parametry (viz sekce 4.2.4).

4.3.7 Další funkcionalita

Mimo funkcionalitu popsanou v předchozích kapitolách je v aplikaci implementováno několik dalších pomocných služeb.

REST služby

Aby nebylo nutné v každé komponentě ručně psát požadavky na back-end, byly tyto požadavky implementovány do separátních služeb. Jedná se o služby `RestLibs`, `RestUtils` a `RestProjects`. Všechny tyto služby jsou potomci třídy `RestBase`, která obsahuje společné vlastnosti všech požadavků. Každá služba implementuje pouze určitý obor požadavků, například `RestProjects` obsahuje pouze požadavky, které manipulují projekty na serveru.

Session služba

Služba `SessionService` obsahuje informace o současném uživateli a umožňuje odeslat požadavek o jeho přihlášení a odhlášení. Je využívána všude, kde je nutné ověřit práva uživatele.

Systém upozornění

Jedná se o systém, který je tvořený službou `AlertService` a komponentami `AlertComponent` a `AlertHostComponent`. Služba `AlertService` poskytuje metodu `pushAlert`, pomocí které lze službu požádat o zobrazení upozornění. O zobrazování upozornění se stará komponenta `AlertHostComponent`, která se u služby zaregistruje a následně při každém požadavku do sebe přidá komponentu upozornění `AlertComponent`. Tato komponenta v sobě obsahuje Bootstrap upozornění. Jediná instance komponenty `AlertHostComponent` se nachází v kořenové komponentě aplikace, což zaručuje, že upozornění se zobrazují ve všech částech front-endu. Mezi zobrazovaná upozornění patří například chybové hlášení při komunikaci s back-endem, nebo potvrzení o úspěšném vykonání akce.

4.4 Sestavení a nasazení aplikace

Pro jednodušší nasazení v cloudu byla provedena tzv. dockerizace aplikace, což je pojem, který značí, že aplikace byla převedena do Docker obrazu a lze ji spustit v kontejneru (viz sekce 3.5). V této kapitole je popsáno, jak byla dockerizace provedena, a jsou zde zmíněny alternativní možnosti nasazení.

4.4.1 Sestavení pro Docker

Aby bylo možné aplikaci dockerizovat, je napřed nutné vytvořit v kořenovém adresáři aplikace soubor `Dockerfile`. Tento soubor v nejzákladnější podobě říká, jaký předdefinovaný obraz se má použít jako základ vytvářeného obrazu, jaká data se do něj mají zkopírovat a jakým příkazem se spouští aplikace v kontejneru. Po spuštění příkazu `docker build` se obraz aplikace sestaví podle tohoto souboru. V tomto příkazu lze nastavit i výsledné jméno obrazu.

V `Dockerfile` vytvářené aplikace je sestavení rozděleno na dvě části. Nejdříve se stáhne obraz obsahující nástroj *Maven*, zkopírují se do něj zdrojové kódy aplikace a spustí se příkaz `mvn install -P prod`, čímž se spustí sestavení, jak bylo vysvětleno v sekci 4.1.1. Následně se stáhne obraz obsahující prostředí Java, do kterého se zkopírují výsledné soubory z obrazu

obsahujícího *Maven* do složky `/app`. Dále se port 8080 označí jako výstupní a nadefinuje se spouštěcí příkaz `java -jar /app/app.jar`.

Díky tomu, že celé sestavení aplikace i její běh probíhá v Docker kontejneru, není nutné na stroj, kde sestavení probíhá, instalovat běhové prostředí Javy, nástroj *Maven*, ani žádný webový server. Stačí mít nainstalovaný a nakonfigurovaný Docker.

4.4.2 Spouštění v Dockeru

Po sestavení obrazu aplikace ji lze spustit z lokálního repositáře Docker obrazů. Pro to, aby aplikace v Dockeru fungovala správně, je nutné, aby bylo splněno několik prerekvizit:

1. Musí existovat kontejner, ve kterém běží MongoDB. Tento kontejner musí být dostupný z kontejneru aplikace skrze hostname `mongo` a naslouchat na portu 27017.

2. Na cestě `/data/libs` musí být napojena složka, do které se budou ukládat soubory knihoven. Nejlepší je zde použít Docker volumes, což jsou složky spravované Dockerem.

3. Port 8080 musí být vystaven mimo kontejner, protože aplikace komunikuje právě na tomto portu. Pokud tento port není z kontejneru dostupný, nelze se na aplikaci připojit.

Po splnění těchto prerekvizit lze kontejner spustit příkazem `docker run` a kombinací přepínačů, kterou je nutné získat z nastavení prerekvizit na serveru.

4.4.3 Nástroj Docker Compose

Protože sestavovat a spouštět Docker obrazy manuálně je relativně složitý proces, existuje nástroj *Docker Compose*, který umožňuje sestavení obrazů a spuštění kontejnerů automatizovat pomocí tzv. služeb. Každá služba pracuje s rozdílným Docker obrazem/kontejnerem a *Docker Compose* umožnuje spustit několik služeb zároveň. Konfigurace tohoto nástroje se nachází v souboru `docker-compose.yml`.

V tomto souboru jsou nejdříve popsány dvě služby. První služba se stará o vytvářenou aplikaci. Jsou zde definovány příkazy sestavení, vystavení portů z kontejneru ven, název obrazu/kontejneru a připojení složky `/data/libs` na Docker volume (viz níže). Druhá služba se stará o kontejner obsahující databázi MongoDB. Je zde definován obraz, který se má použít, vystavení

portu 27017 do lokální Docker sítě, nastavení jména kontejneru na `mongo` a připojení Docker volumů, které obsahují data databáze. Rovněž je zde provedena její konfigurace skrze připojení souboru `initialize-mongodb.js` na cestu specifikovanou v dokumentaci obrazu.

V souboru `docker-compose.yml` je dále provedena definice *volumes*, což jsou souborové systémy plně spravované Dockerem. Díky tomu jsou nezávislé na operačním systému, což umožňuje jejich izolaci a zjednodušuje jejich migraci mezi stroji.

Příkazem `docker-compose build` lze celou aplikaci sestavit a její obraz přidat do lokálního registru obrazů. Obraz lze následně spustit v kontejneru příkazem `docker-compose up`, čímž se provedou všechny ostatní operace popsané ve skriptu. Aplikaci lze ukončit příkazem `docker-compose down -v`, čímž se zastaví a odstraní všechny kontejnery a *volumes* vytvořené předchozími příkazy.

4.4.4 Nasazení na server

Aby bylo možné aplikaci nasadit bez nutnosti jejího sestavení na serveru, je nutné napřed získat její obraz. To lze provézt dvěma způsoby. Pokud je na serveru i na stroji, kde je aplikace sestavována, nastavený v Dockeru stejný repozitář obrazů, lze do repozitáře sestavený obraz nahrát příkazem `docker push graphviewapp`. Na serveru poté stačí obraz stáhnout a načíst příkazem `docker pull graphviewapp`.

Pokud není nastavený repozitář, lze obraz uložit do TAR archivu příkazem `docker save graphviewapp -o app.tar`. Tento archiv pak lze ručně zkopírovat na server a následně načíst příkazem `docker load -i app.tar`.

Následně lze aplikace spustit manuálně pomocí kroků popsaných v sekci 4.4.2, nebo automaticky pomocí nástroje Docker Compose. Při použití nástroje Docker Compose je nutné na server rovněž zkopírovat dvojici souborů `docker-compose.yml` a `initialize-mongodb.js` (viz příloha C). Oba soubory jsou důkladněji popsány v sekci 4.4.3. Aplikaci lze následně spustit příkazem `docker-compose up -d`.

4.4.5 Provozování aplikace bez Dockeru

Vzhledem k tomu, že ne každý stroj má nainstalovaný Docker, je aplikaci možné spustit i bez jeho použití. Tento způsob spouštění je primárně určený pro testování aplikace, protože uživatelská data nejsou persistentní. Pro sestavení aplikace v testovacím režimu je nutné mít nainstalované vývojářské prostředí Java a nástroj *Maven*.

Aplikaci lze sestavit příkazem `maven install -P test`, jehož výstup je JAR soubor ve složce `target/`. Tento JAR je možné spustit, aniž by bylo potřeba instalovat jakékoli další závislosti, mimo běhového prostředí Java. Aplikace si při spuštění stáhne a spustí vlastní instanci *MongoDB* pro uchovávání dat. Tato data jsou ale uložená pouze v paměti, nikoli na disku.

5 Testování a zhodnocení výsledků

Všechny části aplikace byly důkladně manuálně otestovány a následně bylo provedeno testování výkonu editoru na reálných [11, 12] i syntetických testovacích datech. Testovací data musela být před importem do aplikace zkonvertována do formátu CSV, což bylo provedeno ručně. Testování výkonu bylo prováděno na dvojici strojů, jejichž specifikace jsou popsané v tabulce 5.1. Pro testování byl použit prohlížeč Google Chrome 89.

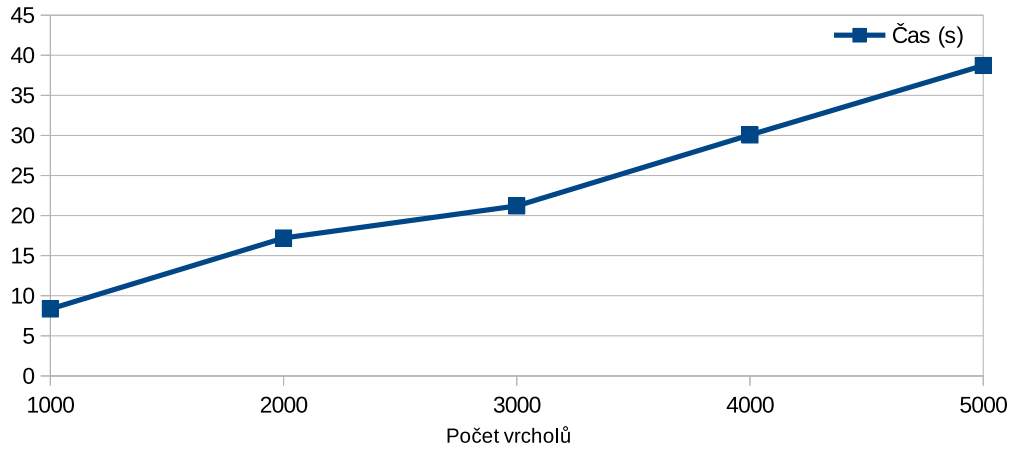
CPU	GPU	RAM
AMD Ryzen 3600 @ 3600 Mhz	AMD RX Vega 56	16GB
Intel Core i7-10710U @ 1600 Mhz	NVidia GTX 1650 Max-Q	16GB

Tabulka 5.1: Specifikace testovacích strojů.

Při testech výkonu se ukázalo, že aplikace dokáže bez větších problémů importovat rozměrnou síť s 81306 vrcholy a 2420766 hranami [11]. Při pokusu o uložení dokumentu na server se však narazilo na omezení databáze MongoDB, kde může mít jeden dokument maximální velikost 16MB. Toto omezení lze v budoucnu obejít použitím technologie *GridFS*, která soubory ukládá do databáze po částech. Export grafu do souboru funguje bezproblémově i pro výše zmíněný rozměrný graf a lze jej použít jako alternativu k ukládání na server.

Při importu předchozího grafu si aplikace vyžádala cca. 300MB paměti, což je přijatelná hodnota vzhledem k tomu, že se celý graf nachází v paměti prohlížeče. Velice rozměrné grafy, které mají desítky milionů vrcholů a hran mohou způsobit nedostatek paměti, proto takto rozměrné grafy nejsou aplikací podporované.

Dalším bottleneckem v aplikaci je výpočet rozložení grafu. Vzhledem k tomu, že výpočet je prováděn iterativní simulací sil, nastává problém, kdy délka simulace roste lineárně s počtem vrcholů (viz obr. 5.1). Z této závislosti vyplývá problém, kdy výpočet rozložení pro rozměrné grafy trvá neúměrně dlouho. Řešením by mohlo být implementovat rozdílný algoritmus rozložení grafu [7], nebo se pokusit rozložení vypočítat vícevláknově technologí *Web Workers*, která umožňuje spouštět kód mimo hlavní vlákno aplikace.



Obrázek 5.1: Délka výpočtu rozložení grafu v závislosti na počtu vrcholů.

Finálním omezením aplikace je samotná výkonnost vykreslování grafu. Vzhledem k tomu že graf je prezentován DOM prvky, je jeho vykreslování přenecháno prohlížeči. Prohlížeč automaticky optimalizuje vykreslování například tím, že nevykresluje prvky mimo oblast okna, ale i přesto může v rozměrných grafech nastat situace, kdy prohlížeč nedokáže vykreslovat všechny prvky v rozumném čase. Aplikace se pak jeví jako neresponzivní. Například při vizualizaci grafu s 5000 vrcholy a 15000 hranami trvalo vykreslení jednoho snímku v průměru 75ms. Prohlížeč tudíž překresloval okno s frekvencí cca. 13.3 snímků za vteřinu. Nicméně pro to aby byla aplikace plynulá je potřeba, aby bylo okno překreslováno s frekvencí alespoň 30 snímků za vteřinu. Vizualizaci lze v budoucnu optimalizovat například použitím technologie *WebGL* [9], která k vykreslování používá *OpenGL* kód, který běží na grafické kartě.

Vzhledem k výsledkům testování výkonu lze konstatovat, že aplikace nejlépe funguje pro grafy, které celkově obsahují méně jak 5000 položek (suma počtu vrcholů a hran). Grafy, které obsahují mezi 5000 a 20000 položkami mohou způsobovat potíže s výkonem, ale neohrožují stabilitu aplikace. Pokud graf obsahuje více jak 20000 položek, je aplikace téměř nepoužitelná.

6 Závěr

Vytvořená aplikace splňuje hlavní cíl práce. Uživatel může do aplikace nahrát grafová data v CSV formátu. Aplikace tato data následně vizualizuje v editoru grafu. Externí knihovny ve formě JAR souborů lze nahrávat na server z rozhraní aplikace a následně spouštět nad daty z grafu. Cílem těchto knihoven je hodnocení významnosti vrcholů grafu, jako je například *PageRank*, nebo *Degree Centrality*. Výsledky výpočtů jsou následně zobrazeny přímo v editoru grafu.

Editor dále umožňuje manipulaci vrcholů, vytváření nových vrcholů a hran a také jejich mazání. Vizualizaci výsledků si lze přizpůsobit a je možné v ní přepínat mezi různými dříve vypočítanými hodnotami. Vypočtené výsledky je rovněž možné z aplikace exportovat ve formátu CSV.

Navíc jsem aplikaci obohatil o správu uživatelů a možnost ukládat rozpracované projekty na server do NoSQL databáze. Tato funkcionalita sice nebyla součástí zadání, ale myslím, že se jedná o důležitou část každé webové aplikace. Díky správě uživatelů je aplikace bezpečnější, protože uživatelé mají omezená práva. Persistence dat v databázi umožňuje uživatelům pracovat na více strojích bez nutnosti exportu a importu dat.

Vzhledem k tomu, že aplikace byla implementována za použití modulárních frameworků Spring a Angular, lze ji v budoucnu relativně jednoduše vylepšovat. V první řadě se nabízí vyřešit výkonnostní omezení popsané v kapitole 5 (např. umožnění plynulé práce na grafech s více jak 20000 položkami). Dále lze aplikaci rozšířit například o možnost importovat/exportovat i jiné druhy souborů než-li CSV. V budoucnu by bylo možné aplikaci rozšířit na komplexní grafový editor, který by umožňoval spouštět nejen externí knihovny ohodnocující vrcholy grafu, ale rovněž psát vlastní algoritmy pracující s vrcholy i hranami. Taková aplikace by byla užitečná například při výuce předmětů, které se grafy zaobírají.

Věřím, že zadání se mi podařilo splnit ve všech bodech, a že vytvořená aplikace bude v budoucnu užitečným nástrojem při výzkumu sociálních sítí.

Literatura

[1] Borgatti, S. P. Centrality and network flow. *Social networks*. 2005, 27, 1.

[2] Brin, S. – Page, L. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*. 1998, 30, 1-7.

[3] Dijkstra, E. W. – others. A note on two problems in connexion with graphs. *Numerische mathematik*. 1959, 1, 1.

[4] Fletcher, J. M. – Wennekers, T. From Structure to Activity: Using Centrality Measures to Predict Neuronal Activity. *International Journal of Neural Systems*. 2018, 28, 02. doi: 10.1142/S0129065717500137. PMID: 28076982.

[5] Floyd, R. W. Algorithm 97: Shortest Path. *Commun. ACM*. June 1962, 5, 6. ISSN 0001-0782. doi: 10.1145/367766.368168. Dostupné z: `https://doi.org/10.1145/367766.368168`.

[6] Freeman, L. C. Centrality in social networks conceptual clarification. *Social networks*. 1978, 1, 3.

[7] Gansner, E. R. et al. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*. 1993, 19, 3, s. 214–230.

[8] Han, J. et al. Survey on NoSQL database. In *2011 6th international conference on pervasive computing and applications*, s. 363–366. IEEE, 2011.

[9] Horak, T. – Kister, U. – Dachselt, R. Comparing rendering performance of common web technologies for large graphs. In *Poster Program of the 2018 IEEE VIS Conference*, 2018.

[10] Inc., V. *Spring Data Documentation* [online]. Dostupné z: `https://spring.io/projects/spring-data/`.

[11] J. McAuley, J. L. Learning to Discover Social Circles in Ego Networks. In *NIPS*, 2012.

[12] Knuth, D. E. *The Stanford GraphBase: A Platform for Combinatorial Computing*. Addison-Wesley, 1993.

[13] Nykl, M. *Hodnocení významnosti variantami PageRanku*. PhD thesis, Západočeská univerzita v Plzni, 2016.

[14] Page, L. et al. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report 1999-66, Stanford InfoLab, November 1999. Dostupné z: http://ilpubs.stanford.edu:8090/422/. Previous number = SIDL-WP-1999-0120.

[15] Pimentel, V. – Nickerson, B. G. Communicating and displaying real-time data with websocket. *IEEE Internet Computing*. 2012, 16, 4.

[16] Sabidussi, G. The centrality index of a graph. *Psychometrika*. 1966, 31, 4.

[17] Snyk.io. *JVM ecosystem report* [online]. Dostupné z: https://snyk.io/blog/spring-dominates-the-java-ecosystem-with-60-using-it-for-their-main-applications/

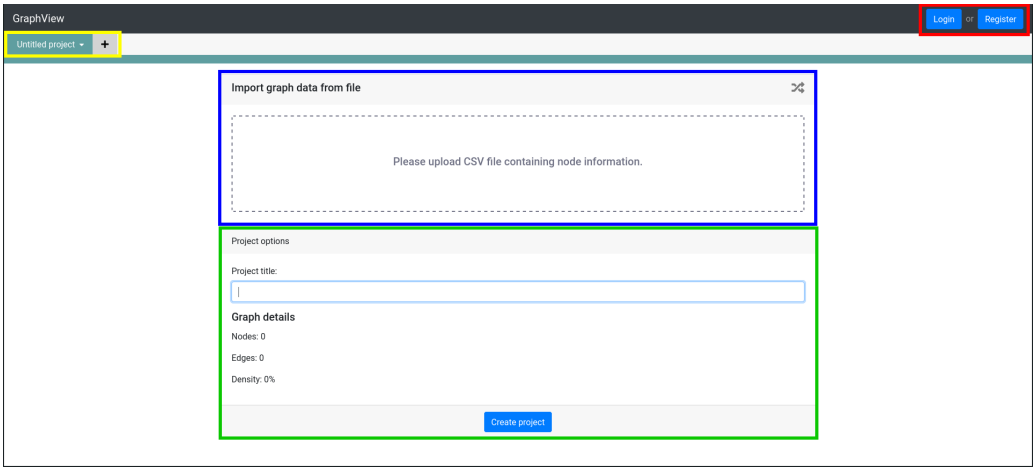
[18] W3Techs. *Usage statistics of PHP for websites* [online]. Dostupné z: https://w3techs.com/technologies/details/pl-php.

A Uživatelská příručka

V této příloze se nachází stručný návod k tomu, jak aplikaci používat.

A.1 Hlavní obrazovka

Po prvním spuštění aplikace budete přívítáni následující obrazovkou:



Obrázek A.1: Úvodní obrazovka aplikace.

V horní pravé části obrazovky se nachází dvojice tlačítek, pomocí kterých je možné se do aplikace přihlásit, nebo si zažádat o vytvoření účtu (vyznačeno červenou barvou na obr. A.1).

Žlutou barvou je na obr. A.1 vyznačená lišta projektů. V aplikaci je možné mít otevřeno více projektů najednou, každý projekt má svou vlastní lištu. Pomocí tlačítka + je možné vytvořit nový prázdný projekt. Jednotlivé lišty lze přesouvat pomocí táhnutí myší. Po kliknutí na aktivní lištu se zobrazí menu, ze kterého lze projekt uložit/načíst, nebo přejmenovat. Možnost ukládání na server je povolena pouze pro přihlášené uživatele.

A.1.1 Průvodce vytvořením nového projektu

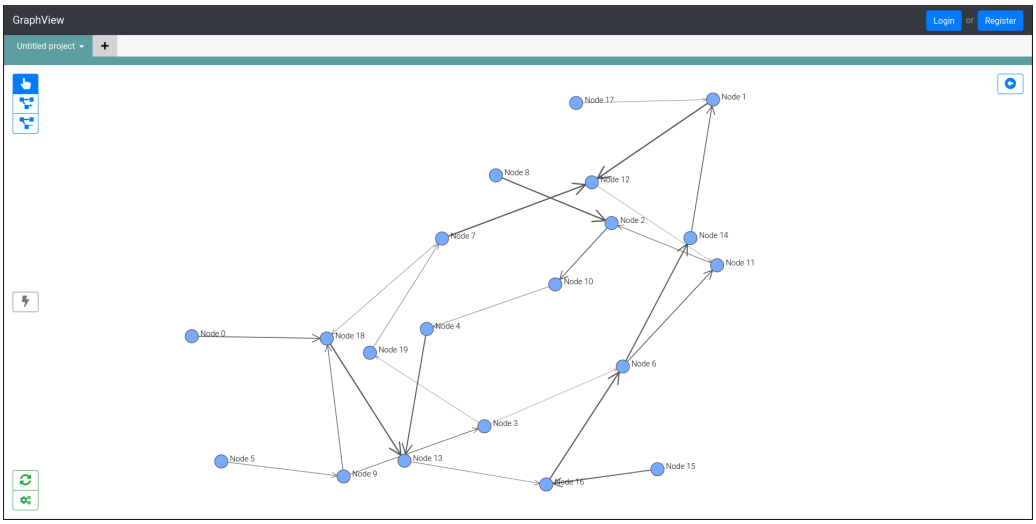
Pokud je aktivní projekt prázdný, zobrazí se průvodce vytvořením nového projektu, ve kterém lze importovat do projektu existující graf z CSV souboru. Na obrázku A.1 je modrou barvou označena plocha, do které lze soubor

grafu přetáhnout. Po nahrání souboru se zobrazí seznam přepínačů, kterými lze import grafu přizpůsobit. Lze například nastavit, zda soubor obsahuje záhlaví, nebo přiřadit druhy dat ke sloupcům CSV tabulky. Aby byl naimportován celý graf, je nutné aby byly nahrány dva soubory. První soubor obsahuje definice vrcholů a druhý definice hran.

Pod oblastí pro nahrávání souborů se nachází oblast, ze které lze projekt pojmenovat a vytvořit (vyznačeno na obr. A.1 zeleně). V této oblasti jsou rovněž vypsány základní grafové statistiky. Tlačítko vytvoření projektu umožňuje vytvořit i prázdný projekt, který neobsahuje žádné vrcholy.

A.2 Grafový editor

Po vytvoření projektu se zobrazí editor grafu (viz obr. A.2). Při prvním otevření projektu se provede výpočet optimálního rozložení grafu.



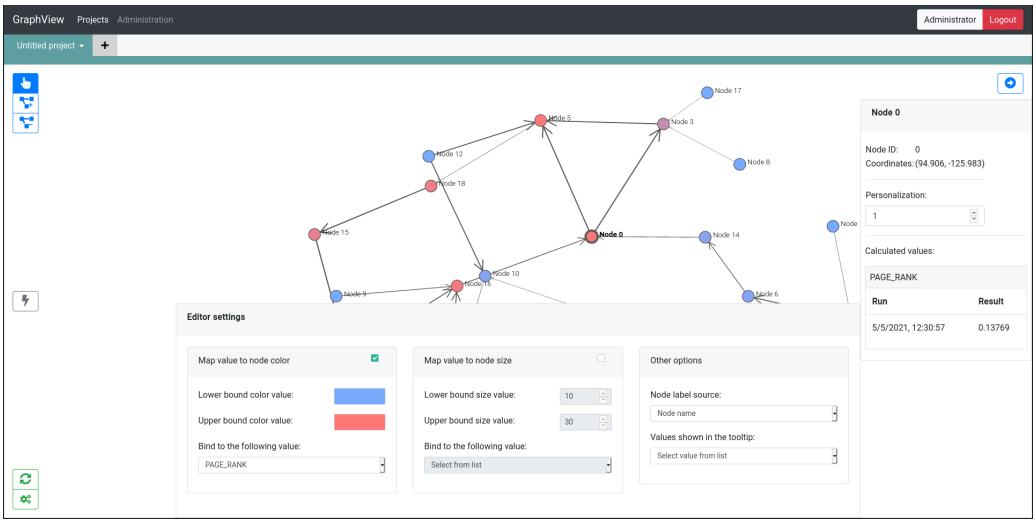
Obrázek A.2: Obrazovka grafového editoru.

Většinu obrazovky zabírá vizualizace grafu. Vrcholy jsou reprezentovány jako kruhy, a šipky mezi nimi představují hrany. Graf lze přibližovat a oddalovat pomocí kolečka myši. Pohled na graf lze posouvat pomocí stisku pravého, nebo prostředního tlačítka a následným táhnutím myši. Po najetí myší na vrchol se zobrazí popisek obsahující informace o daném vrcholu. Při kliknutí na vrchol se daný vrchol přidá do výběru. do výběru lze přidat více vrcholů najednou pomocí táhnutí myši. Detaily výběru se zobrazují v panelu na pravé straně editoru (viz A.2.1).

Po levé straně se nacházejí tři skupiny tlačítek. Modrá tlačítka obsahují nástroje pro manipulaci jednotlivých vrcholů/hran, např. posun, přidávaní a mazání. Prostřední tlačítko umožňuje spustit nad grafem externí knihovnu (viz sekce A.2.3 a A.3.2). Zelenými tlačítky lze přepočítat rozložení a zobrazit/skrýt panel nastavení vizualizace (viz sekce A.2.2).

A.2.1 Detaily výběru

Na pravé straně editoru se nachází panel detailů výběru (viz obr. A.3). Tento panel zobrazuje dodatečné informace o současném výběru. Panel lze skrýt a opětovně zobrazit pomocí tlačítka, které se nachází nad panelem. V případě, že je výběr prázdný se v panelu zobrazují základní informace o grafu a historie výpočtů externích knihoven.



Obrázek A.3: Editor s vysunutými panely detailů a nastavení.

Pokud je vybrán právě jeden vrchol, v panelu se zobrazují informace o daném vrcholu, včetně hodnot, které byly vypočítané dříve skrze externí knihovny. Po kliknutí na nějakou z těchto hodnot se v grafu vizualizují všechny výsledky, které byly touto knihovnou vypočteny.

V případě, že je vybráno více vrcholů, umožňuje panel přiřadit každému z nich personalizaci.

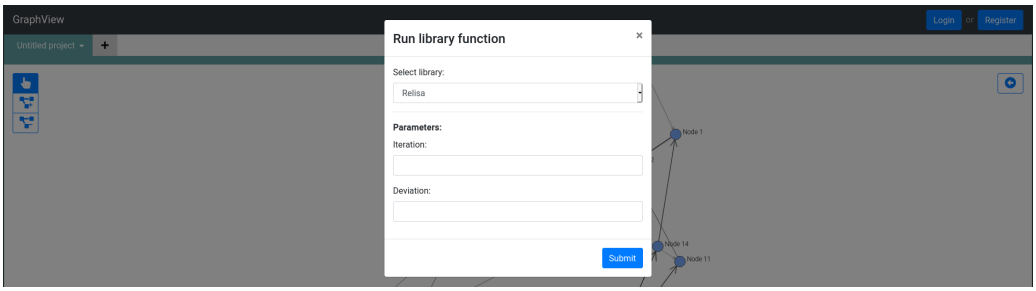
A.2.2 Nastavení vizualizace

V tomto panelu lze přizpůsobit vizualizaci grafu a vypočtených hodnot. Jak jde vidět na obrázku A.3, panel se nachází na spodní části editoru. První

sekce panelu umožňuje navázat vypočtenou hodnotu vrcholu na jeho barvu. Druhá část umožňuje navázat rozdílnou hodnotu na jeho velikost. Aby bylo možné navázání provézt, musí se napřed provézt výpočet ohodnocení vrcholů skrze externí knihovnu. Poslední sekce panelu umožňuje nastavit jaké hodnoty se zobrazují v názvu a popisku vrcholu.

A.2.3 Spuštění externí knihovny

Po kliknutí na tlačítko spuštění externí knihovny se zobrazí dialog (viz obr. A.4), ve kterém lze vybrat knihovnu, která bude spuštěná. Tyto knihovny musí do systému nahrát administrátor (viz sekce A.3.2). V tomto dialogu lze rovněž knihovně předat parametry, se kterými se bude spouštět. Po spuštění knihovny a vrácení výsledků se výsledky v grafu automaticky vizualizují.



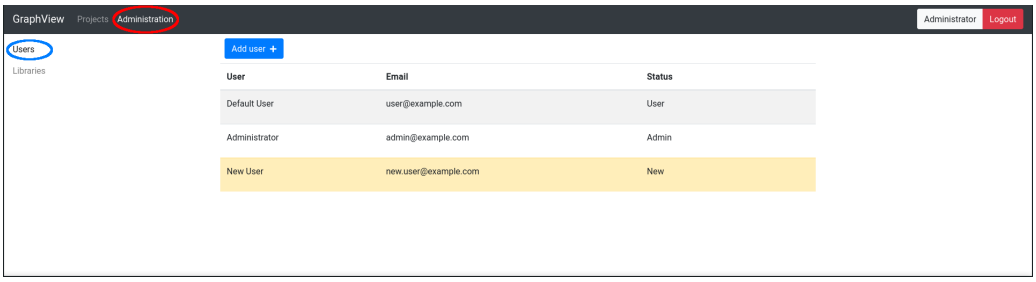
Obrázek A.4: Dialog spuštění externí knihovny.

A.3 Administrativní část aplikace

Do administrativní části aplikace mají přístup pouze uživatelé s právem administrátora. Pokud je uživatel administrátor, zobrazí se v horní části obrazovky položka, pomocí které se lze do administrativní části aplikace přesunout. Tato položka je na obr. A.5 zvýrazněna červenou barvou. V administraci lze přepínat mezi administrací uživatelů a knihoven pomcí menu na pravé straně.

A.3.1 Administrace uživatelů

V této části aplikace lze přidávad, odebírat a měnit uživatele v systému. Kliknutím na položku, která je na obr. A.5 zvýrazněná modrou barvou se zobrazí tabulka uživatelů. Každá položka v této tabulce představuje jednoho uživatele v systému. Kliknutím na libovolnou položku se zobrazí dialog, ve

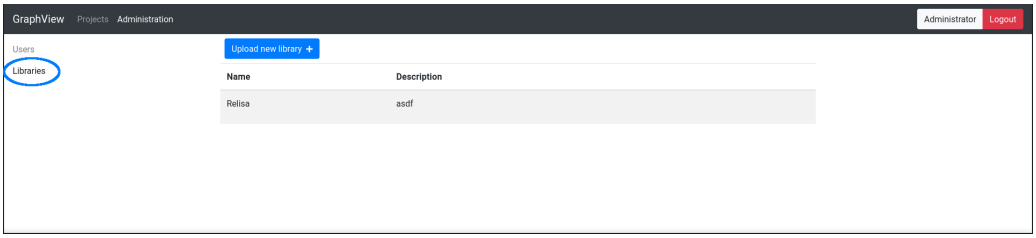


Obrázek A.5: Obrazovka administrace uživatelů.

kterém lze upravit data daného uživatele. Kliknutím na tlačítko `X` na pravé straně položky v tabulce se daný uživatel smaže ze systému. Uživatelé, kteří jsou v systému nově registrovaní, jsou v tabulce zvýraznění a čekají na přidělení uživatelských práv. Nad tabulkou uživatelů se nachází tlačítko, kterým lze do systému přidat nového uživatele.

A.3.2 Administrace knihoven

Na obr. A.6 je popsána obrazovka administrace knihoven. Administrace knihoven pracuje podobně jako administrace uživatelů, jen je nutné při vytváření nové knihovny do systému nahrát JAR souboru knihovny. Dialog vytvoření knihovny rovněž umožňuje přidat ke každé knihovně seznam argumentů, které knihovna používá při spuštění. Tyto parametry jsou detekovány automaticky při nahrání knihovny do systému, ale je možné je přidat i ručně.



Obrázek A.6: Obrazovka administrace knihoven.

B Komunikační protokoly

B.1 REST API aplikace

V následujícím seznamu se nachází seznam jednotlivých endpointů a akcí, které provádějí. Princip REST je popsán v sekci 3.3.2.

`/api/login` :

POST Přihlášení uživatele do systému.

`/api/user` :

GET Vrací informace o přihlášeném uživateli.

`/api/users` :

GET Vrací seznam všech uživatelů v systému.

`/api/users/{username}` :

POST Vytvoří v systému nového uživatele.

PUT Upravuje data daného uživatele.

DELETE Smaže daného uživatele ze systému.

`/api/libs` :

GET Vrací seznam všech knihoven v systému.

POST Nahraje do systému novou knihovnu.

`/api/libs/{id}` :

DELETE Smaže danou knihovnu ze systému.

`/api/libs/{id}/run` :

POST Spustí danou knihovnu.

`/api/projects` :

GET Vrátí seznam všech dříve uložených projektů, které patří přihlášenému uživateli.

POST Uloží projekt na server.

`/api/projects/{id}` :

GET Vrátí data daného projektu ze serveru.

DELETE Smaže daný projekt ze serveru.

B.2 Komunikace s knihovnou

Všechny knihovny jsou aplikací spouštěny s parametrem `-useStreamIO`. Data jsou knihovně předávána na její `stdin` a výsledky jsou očekávány na `stdout`. Do `stdout` by se nemělo vypisovat nic, co nepatří k výsledkům. Pokud knihovna potřebuje něco sdělit uživateli, lze tak provézt do `stderr`. Pokud knihovna při ukončení vrátí nenulový stavový kód, je její `stderr` přečten a odeslán uživateli místo výsledků.

B.2.1 Formát vstupních dat

Mezi příkazy `DATA` a `END_DATA` se nachází definice dat. Mezi příkazy `NODES` a `END_NODES` se nachází seznam vrcholů a mezi `EDGES` a `END_EDGES` se nachází seznam hran. V seznam vrcholů každý vrchol obsahuje své ID, jméno a personalizaci. V seznamu hran každá hrana obsahuje ID výstupního vrcholu, ID vstupního vrcholu a svou váhu.

```
DATA
NODES
NODE1_ID;NODE1_NAME;NODE1_PERSONALIZATION
NODE2_ID;NODE2_NAME;NODE2_PERSONALIZATION
...
END_NODES
EDGES
NODE1_ID;NODE2_ID;EDGE1_WEIGHT
NODE2_ID;NODE3_ID;EDGE2_WEIGHT
...
END_EDGES
END_DATA
```

B.2.2 Formát výstupních dat

Za příkazem `NODES` se nachází seznam hodnot, které byly pro každý vrchol vypočteny. Každý vrchol v seznamu vrcholů obsahuje pouze své ID a seznam vypočtených hodnot.

```
RESULTS
NODES;NODE_VAL1_LABEL;NODE_VAL2_LABEL
NODE1_ID;NODE1_VAL1;NODE1_VAL2
NODE2_ID;NODE2_VAL1;NODE2_VAL2
...
END_NODES
END_RESULTS
```

Finální výstup tudíž může vypadat například takto:

```
RESULTS
NODES;PAGE_RANK
0;0.041
1;0.359
2;0.600
END_NODES
END_RESULTS
```

B.2.3 Formát strojově čitelné nápovědy

Po spuštění knihovny s parametrem `-helpMachine` back-end očekává výstup na `stdout` v následujícím formátu:

```
ARGS
NAME;OPTION;TYPE;MANDATORY;DESCRIPTION
END_ARGS
```

Mezi příkazy `ARGS` a `END_ARGS` je možné zadefinovat více argumentů, každý argument musí být na separatní řádce. Pole `NAME` značí jméno argumentu zobrazované v systému. `OPTION` značí přepínač, který se přidá za příkaz ke spuštění knihovny. V poli `TYPE` se nachází typ hodnoty, která bude knihovně předávána. Typy jsou podporovány tři:

`number` hodnota je celočíselná

`normal` hodnota je z intervalu <0, 1>

`string` hodnota je textový řetězec

Pole `MANDATORY` definuje, zda se jedná o povinný argument. Obsahuje `true`, pokud je k běhu knihovny tento argument vyžadován, v opačném případě `false`. Do pole `DESCRIPTION` je možné napsat detailní popis parametru.

Finální popis argumentů může vypadat například takto:

```
ARGS
Damping;-d;normal;false;damping factor
Termination;-termination;string;false;kriterum ukonceni
Deviation;-deviation;normal;false;odchylka pro ukonceni
Iteration;-iteration;number;false;pocet iteraci pro ukonceni
END_ARGS
```

C Obsah CD

Zde je popsána adresářová struktura souborů na CD:

`app/` - Soubory potřebné pro spuštění aplikace

`app.tar` - Docker obraz aplikace

`docker-compose.yml` - Soubor nastavení nástroje Docker Compose

`initialize-mongodb.js` - Skript používaný k inicializaci databáze

`GraphAppView.jar` - Soubor aplikace spustitelný bez Dockeru

`data/` - Ukázková vstupní data aplikace

`small_project.csv` - Data jednoduchého grafu

`large_project.csv` - Data složitého grafu

`PageRank_ZCU.jar` - JAR knihovna implementující PageRank

`src/` - Adresář obsahující zdrojové soubory aplikace