Maturitní práce

30. března 2025

Gymnázium, Praha 6, Arabská 14

Arabská 14, Praha 6, 160 00

Předmět: Programování

Téma: Nástroj pro tvorbu myšlenkových map

Autor: *Vítězslav Procházka*

Třída: 4.E

Vyučující: Mgr. Jan Lána

Tř. vyučující: Mgr. Blanka Hniličková

Čestné prohlášení:

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené.

Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V Praze dne ....

Poděkování

Chtěl bych poděkovat Mgr. Janu Lánovi za vedení této práce. Dále bych chtěl poděkovat Mgr. Jiřímu Procházkovi za pomoc při navrhování ovládání a jeho následné testování . Velké díky také patří Ing. Václavu Chalupníčkovi za velmi srozumitelný úvod do tvorby webových aplikací.

Anotace

Tato dokumentace popisuje maturitní práci, na téma: Nástroj pro tvorbu myšlenkových map. Cílem tohoto projektu je vytvořit intuitivní nástroj pro tvorbu myšlenkových map v podobě webové stránky. Každý uživatel bude mít svůj vlastní workspace, kde bude mít online uložené své vlastní privátní myšlenkové mapy. Myšlenkové mapy by měly jít snadno vytvářet a upravovat. Dalším podstatným prvkem je přehlednost uživatelského rozhraní.

Klíčová slova

Myšlenková mapa; ; webová aplikace; graf; brainstorming; organizace myšlenek;

Abstract (English)

This documentation describes a year-end project on the topic: A Tool for Creating Mind Maps. The goal of this project is to create an intuitive tool for mind map creation in the form of a website. Each user will have their own workspace, where they can store their private mind maps online. Mind maps should be easy to create and edit. Another essential element is the clarity of the user interface.

Keywords

Mind map; web application; graph; brainstorming; thoughts organization;

Obsah

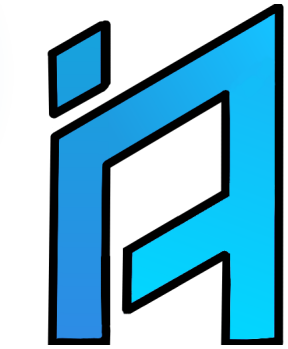
1 Úvod

Myšlenkové mapy představují efektivní nástroj pro vizualizaci a organizaci informací, podporující kreativní myšlení, analýzu problémů a strukturování myšlenek. Jejich využití se rozšířilo do různých oblastí, včetně vzdělávání, projektového řízení, vědeckého výzkumu a osobního rozvoje. Tradičně se myšlenkové mapy vytvářely na papíře, avšak s rozvojem digitálních technologií se stále více uplatňují webové aplikace umožňující dynamickou práci s uzly, propojeními a interaktivními prvky.

Tato práce se zaměřuje na návrh a implementaci webové aplikace pro tvorbu myšlenkových map, která poskytuje uživatelům intuitivní rozhraní pro vytváření a editaci myšlenkových schémat. Cílem je vytvořit prototyp webové aplikace, která přináší přehledné, čisté uživatelské rozhraní a snadné, zapamatovatelné ovládání.

Projekt je stavěn na moderních technologiích pro vývoj webových aplikací, včetně frameworků pro dynamické uživatelské rozhraní a nástrojů pro práci s grafovými strukturami a jejich vizualizaci.

Jméno projektu Idea-Atlas (z angličtiny: atlas idejí) má vyjadřovat účel projektu; tedy mapovat ideje (myšlenky).



Obrázek 1: Logo Idea-Atlas

1.1 Zadání projektu

IdeaAtlas bude online webový nástroj pro tvorbu myšlenkových map. Uživatelům umožní jednoduše uspořádat své myšlenky a nápady do grafů závislostí mezi pojmy. Uživatelé budou moci přidávat nové položky, měnit jejich vztahy, mazat je, a tím upravovat myšlenkovou mapu. Další funkcí tohoto nástroje bude generování souvislostí a pojmů pomocí ChatGPT. Každý uživatel bude mít svůj vlastní workspace, což znamená, že jeho mapy budou uloženy na serveru. Tento nástroj bude ideální pro brainstorming a vizualizaci souvislostí libovolné problematiky.

1.2 Důvod výběru tématu

Vybral jsem si téma nástroj pro tvorbu myšlenkových map, protože mě zaujal koncept vizuálního mapování nápadů a jejich propojení do dynamické struktury. Zároveň mi téma přišlo obtížné, nikoliv však nemožné. Práce s grafy, interaktivní vizualizací a databázemi přináší spoustu výzev. Dále jsem si chtěl vyzkoušet moderní technologie pro tvorbu webových aplikací a prozkoumat, jak je lze efektivně kombinovat.

2 Použité technologie

V mém projektu jsme se rozhodl využít tyto programovací jazyky, frameworky a významné knihovny.

1. Nuxt 3 - full stack framework[1]
    - Vue.js - front-end framework na kterém staví Nuxt[2]
    - V-network-graph - knihovna pro tvorbu relačních interaktivních grafů[3]
2. JavaScript, TypeScript – Logika uživatelského rozhraní, dotazy do databáze
    - D3.js - Knihovna pro tvorbu grafů (v mé práci použita k implementaci force layout) [4]
3. Supabase - open source alternativa k firebase, databáze, Baas (back-end as service)[5]
4. Tailwind CSS - jednodušší a organizovaný design UI[6]
5. Bun - rychlejší alternativa k Node.js; runitme, package manage, bundler [7]

3 Brainstorming a myšlenkové mapy

V dnešním světě, kde inovace hrají klíčovou roli, je schopnost generovat nové nápady zásadní. Existuje mnoho metod, jak podpořit kreativní myšlení, ale dvě z nejefektivnějších jsou brainstorming a myšlenkové mapy. Tyto techniky nejen usnadňují proces tvorby nápadů, ale také pomáhají organizovat myšlenky do srozumitelných struktur.[8]

3.1 Brainstorming: Svoboda Myšlení

Brainstorming je oblíbená metoda, která umožňuje jednotlivcům i skupinám přicházet s novými nápady bez obav z okamžitého hodnocení. Tento proces podporuje volné myšlení a často vede k překvapivým a inovativním řešením. Základní pravidlo brainstormingu spočívá v tom, že neexistují špatné nápady – jakýkoliv návrh může sloužit jako inspirace pro další myšlenky.[9]

Brainstorming obvykle probíhá v několika fázích. Nejprve je definován problém nebo téma, na které se skupina soustředí. Poté účastníci spontánně sdílejí své myšlenky, aniž by byly okamžitě analyzovány nebo kritizovány. Teprve v závěrečné fázi dochází k selekci a hodnocení nápadů s cílem vybrat ty nejefektivnější. Tento přístup eliminuje bariéry v myšlení a umožňuje vznik inovativních konceptů, které by jinak mohly být přehlédnuty.[10]

3.2 Myšlenkové Mapy: Strukturované Myšlení

Zatímco brainstorming podporuje rychlý tok nápadů, myšlenkové mapy slouží k jejich vizualizaci a organizaci. Myšlenková mapa je grafické znázornění informací, které propojuje jednotlivé pojmy a ukazuje jejich vzájemné vztahy. Tato metoda je velmi efektivní nejen při plánování projektů, ale také při učení nebo řešení složitých problémů.

Vytvoření myšlenkové mapy začíná ústředním pojmem, který je umístěn do středu diagramu. Od něj se větví hlavní témata, která se dále dělí na podtémata. Tento proces napodobuje přirozený způsob, jakým mozek zpracovává informace, což z něj činí intuitivní a účinný nástroj.

Používání myšlenkových map pomáhá lépe pochopit složité souvislosti a usnadňuje zapamatování informací. Díky své vizuální podobě umožňují snadnější orientaci v nápadech a podporují kreativní řešení problémů. [11]

3.3 Propojení Brainstormingu a Myšlenkových Map

Brainstorming a myšlenkové mapy se vzájemně doplňují. Po ukončení brainstormingu je možné převést výsledné nápady do myšlenkové mapy, což pomůže s jejich organizací a dalším rozpracováním. Tento postup umožňuje nejen efektivnější řízení kreativního procesu, ale také lepší pochopení a využití generovaných myšlenek.

V dnešní době, kdy jsou kreativita a inovace nezbytné pro úspěch, jsou tyto techniky cenným nástrojem pro každého, kdo se snaží přicházet s novými nápady a hledat neotřelá řešení. Ať už pracujeme na osobních projektech, nebo spolupracujeme v týmu, brainstorming a myšlenkové mapy nám pomáhají překonat bariéry v myšlení a nacházet nové perspektivy. [8, 12]

4 Instalace

4.1 Linux Setup

Pro správnou instalaci projektu IdeaAtlas na operační systém Linux postupujte podle následujících kroků:

Klonování repozitáře

Nejprve naklonujte repozitář pomocí příkazu:

```
git clone https://github.com/gyarab/2024-4e-prochazka-IdeaAtlas.git
```

Instalace Bun

Nainstalujte správce balíčků Bun spuštěním následujícího příkazu:

```
curl -fsSL https://bun.sh/install | bash
```

Přechod do adresáře projektu

Přejděte do hlavního adresáře projektu:

```
cd 2024-4e-prochazka-IdeaAtlas
cd idea-atlas
```

Instalace závislostí

Nakonec nainstalujte všechny závislosti projektu:

```
bun install
```

Po úspěšném dokončení těchto kroků by měl být projekt připraven k použití.

4.2 Widnows setup

Pro správnou instalaci projektu IdeaAtlas na operační systém Windows postupujte podle následujících kroků:

Klonování repozitáře

Nejprve naklonujte repozitář pomocí příkazu:

```
git clone https://github.com/gyarab/2024-4e-prochazka-IdeaAtlas.git
```

Instalace Bun

Nainstalujte správce balíčků Bun spuštěním následujícího příkazu:

```
powershell -c "irm␣bun.sh/install.ps1␣|␣iex"
```

Přechod do adresáře projektu

Přejděte do hlavního adresáře projektu:

```
cd 2024-4e-prochazka-IdeaAtlas
cd idea-atlas
```

Instalace závislostí

Nakonec nainstalujte všechny závislosti projektu:

```
bun install
```

Po úspěšném dokončení těchto kroků by měl být projekt připraven k použití.

4.3 Spuštění projektu

Pro spuštění projektu: v adresáři *2024-4e-prochazka-IdeaAtlas\idea-atlas* použijte příkaz

```
bun run dev
```

5 Ovládání

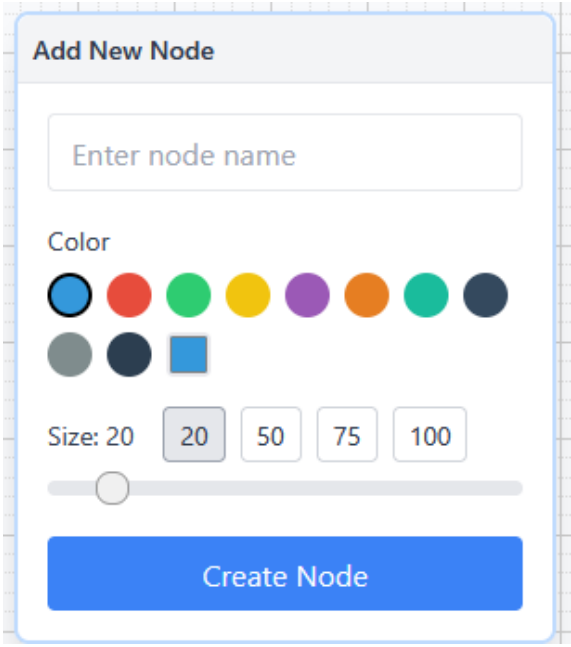
Ovládání výrazně závisí na klávesových zkratkách, nástroj tedy není vhodný k používání na mobilu. Snažil jsem se, aby ovládání bylo intuitivní a lehce se dalo zapamatovat. Seznam všech zkratek naleznete v kapitole 5.10

5.1 Pohyb po myšlenkové mapě

Pokud zmáčkneme levé tlačítko (do prázdného prostoru) a budeme ho držet, můžeme myšlenkovou mapou pohybem myši posouvat. Pro přibližování a oddalování slouží kolečko na myši.

5.2 Přidání nového vrcholu

Pro přidání nového vrcholu stačí namířit myšítkem na zvolené místo pro nový vrchol. Dvojklikněme levým tlačítkem myši nebo stiskněme klávesu Enter pro zobrazení dialogu pro tvorbu nového vrcholu (obrázek 2. Zde vyplňme jméno, zvolme barvu a velikost. Tento dialog se dá posouvat za jeho záhlaví a pro jeho zmizení stačí kliknout mimo něj.

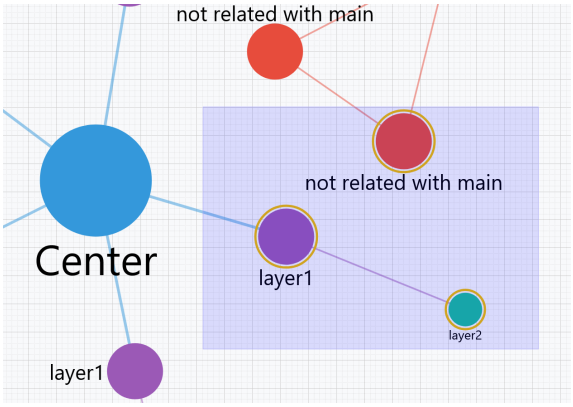


Obrázek 2: Dialog pro přidání nového vrcholu

5.3 Selekce vrcholů a hran

Existuje hned několik způsobů, jak vybrat prvky. Pro vybrání jedné hrany nebo vrcholu na ně stačí kliknout levým tlačítkem myši. Pokud jich chceme vybrat více, musíme u toho držet *LShift*.

Dále můžeme hrany vybrat pomocí obdélníkového výběru, který aktivujeme klávesou *Ctrl* a tažením myši při držení stisknutého levého tlačítka, viz obrázek 3.



Obrázek 3: Obdélník pro selekci vrcholů

Další speciální způsob, jak označit vrcholy, je pomocí *wave selectu*, který postupně rekurzivně označí vrcholy spojené s těmi již označenými. Vždy je zde časová prodleva, než se vybere další vrstva, postupně je tato prodleva snižována; pro případy, kdy chce uživatel takto označit velké množství vrcholů. Selekce probíhá, pokud uživatel drží klávesu *W*, pokud ji pustí, dojde k přerušení a další vrcholy nebudou vybrány. Tato funkcionalita se zejména hodí, pokud naše myšlenková mapa má strukturu lesa, má tedy hodně oddělených stromů - jednoduše se tak dá vybrat právě jeden celý strom.

Poslední způsob, jak vybrat vrcholy, je pomocí vyhledávání. Automaticky se budou vybírat ty vrcholy, které obsahují zadaný řetězec.

Pro odznačení jakéhokoliv výběru stačí kliknout levým tlačítkem myši na pozadí.

5.4 Přidání hran mezi vrcholy

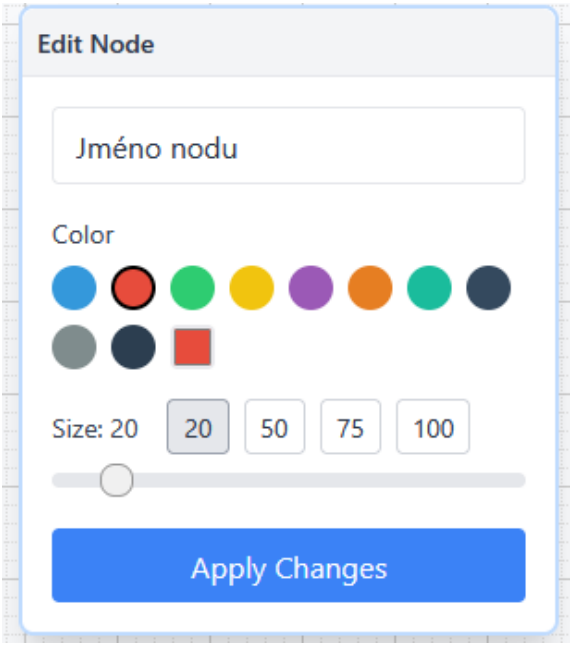
Pokud máme označené dva vrcholy, pomocí mezerníku je spojíme. Pokud máme označeny 3 nebo více vrcholů, po zmáčknutí mezerníku se spojí všechny vybrané vrcholy s tím, co bylo označeno jako první. Pokud chceme spojit všechny vrcholy mezi sebou, stačí zmáčknout najednou *Ctrl* a mezerník.

5.5 Odebrání vrcholu a hran

Po stisknutí klávesy *backspace* dojde k smazání vybraných prvků, nezáleží na tom, jestli jde o hrany nebo vrcholy. Pokud ovšem chceme smazat hrany, které mezi sebou sdílí vybrané vrcholy, dosáhneme toho tím stisknutím *ctrl* a *backspace*.

5.6 Úprava vrcholu

Pro úpravu vrcholu na něj dvojklikněme levým tlačítkem myši nebo pomocí klávesové kombinace *ctrl* a *Enter*. Objeví se nám dialog pro úpravu vrcholu (obrázek 4), který je předvyplněný jeho aktuálními parametry. Tento dialog se dá posouvat za jeho záhlaví a pro jeho zmizení stačí kliknout mimo něj.



Obrázek 4: Dialog pro úpravu vrcholu

5.7 Undo a Redo

Pro odčinění poslední akce stiskněte klávesovou kombinaci *Ctrl* a *Z*. Pokud jsme se vrátili o pár akcí do minulosti v historii provedených kroků, můžeme se pohybovat zpět k současnému bodu pomoci kombinace kláves *Ctrl*, *LShift* a *Z*

5.8 Změna layoutu

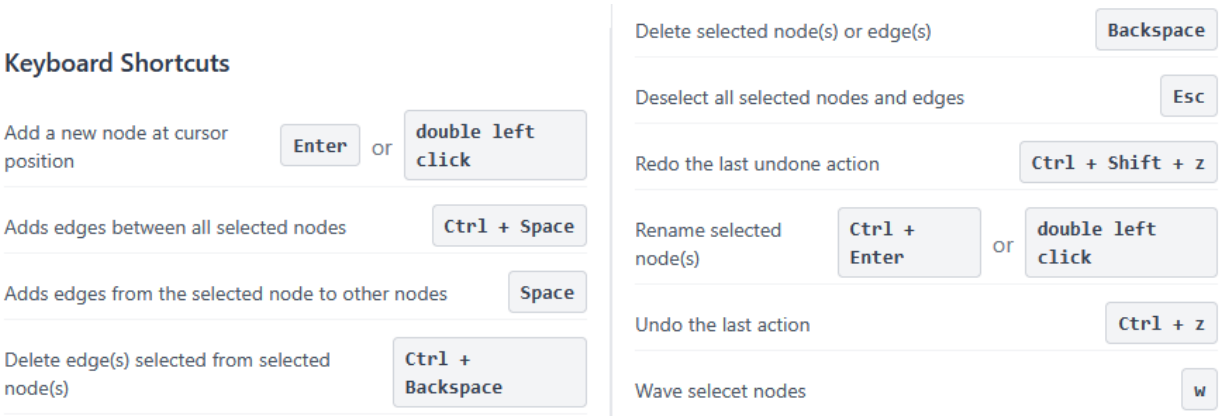
Pomocí tlačítka Force Layout a Grid Layout (záleží, co je aktuálně vybráno) můžeme tyto dva režimy přepínat.

5.9 Uložení

Pro uložení neslouží žádná klávesová zkratka, ale tlačítko Save. Nebo stačí, pokud nebudeme dělat žádné změny po dobu delší než 2 vteřiny.

5.10 Seznam všech klávesových zkratek

Veškeré klávesové zkratky naleznete na obrázku 5; samotná webová stránka nabízí možnost zobrazení této nápovědy.



Obrázek 5: Seznam klávesových zkratek

6 Frontend

Webová aplikace je primárně postavena na Vue.js, moderní JavaScriptové knihovně určené k tvorbě dynamických uživatelských rozhraní. Využívá komponentový přístup, který umožňuje rozdělit rozhraní na menší, znovu použitelné části, čímž usnadňuje správu kódu a zlepšuje čitelnost projektu. Tento přístup nejen zjednodušuje vývoj, ale také usnadňuje rozšiřování a údržbu aplikace, což je klíčové pro škálovatelnost projektu.

6.1 Routování

V Nuxtu funguje routování automaticky díky file-based routing, což znamená, že struktura souborů ve složce *pages/* určuje URL adresy aplikace. Například soubor *pages/index.vue* odpovídá domovské stránce */*, zatímco *pages/about.vue* vytváří routu */about*. Pro dynamické parametry se používají hranaté závorky, takže soubor *pages/blog/[id].vue* odpovídá cestě */blog/:id*, kde lze parametr id získat pomocí *useRoute()*.[13]

6.2 Komponenty

Vue.js i Nuxt staví na technologii komponentů. Komponenty umožňují rozdělit komplexní UI problémy do mnoha menších podproblémů; je to podobné jako například funkce nebo třídy v objektově orientovaném programování. Jednotlivé komponenty jsou samostatné .vue soubory ve složce *components*, obsahují HTML šablony, JavaScriptový (popřípadě TypeScriptový) kód i CSS styly. Komponenty pomáhají modulárně strukturovat aplikaci, což vede k lepší čitelnosti a opětovné použitelnosti kódu.[14] Dále bych chtěl podotknout, že Nuxt zajišťuje automatické importování komponent, není tedy potřeba je v každém souboru importovat manuálně. [15]

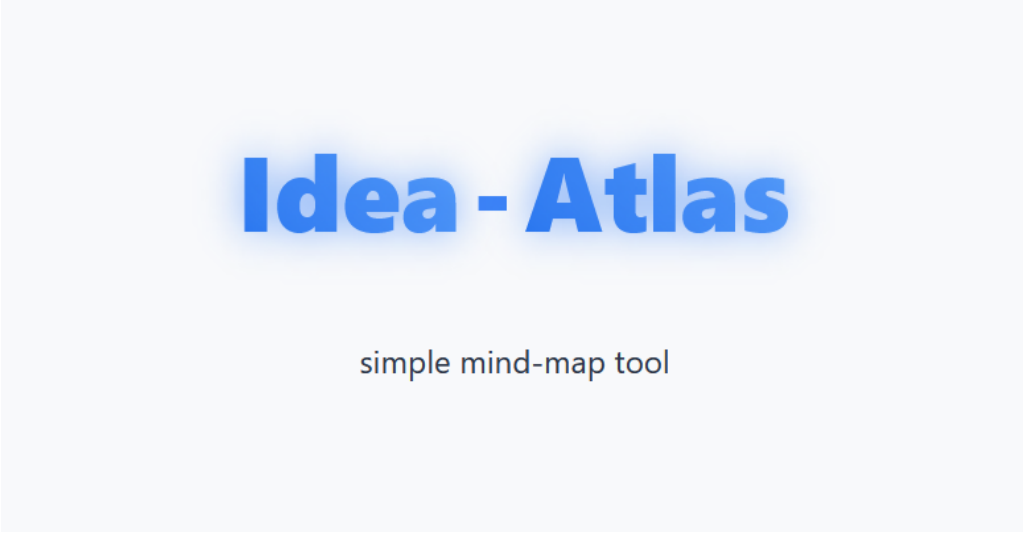
Nejvýznamnějším komponentem v tomto projektu je *MapNetwork.vue*. V tomto komponentu se do hromady skládá veškerá funkcionalita nástroje na tvorbu myšlenkových map. V jádru této komponenty stojí knihovna V-network-graph, která se stará o

reprezentaci i zobrazování relačních grafů viz kapitola 7. Jsou zde napojeny další UI komponenty, jak funkce spravující grafovou strukturu, tak funkce dotazující do databáze. Dále jsou zde handlery na obsluhování klávesových zkratek.
Dalšími významnými komponenty jsou *NodeInputDialog.vue* *NodeEditDialog.vue*, tyto UI komponenty zprostředkovávají uživatelské rozhraní pro editace (případně tvorbu) jednotlivých nodů.

6.3 Design

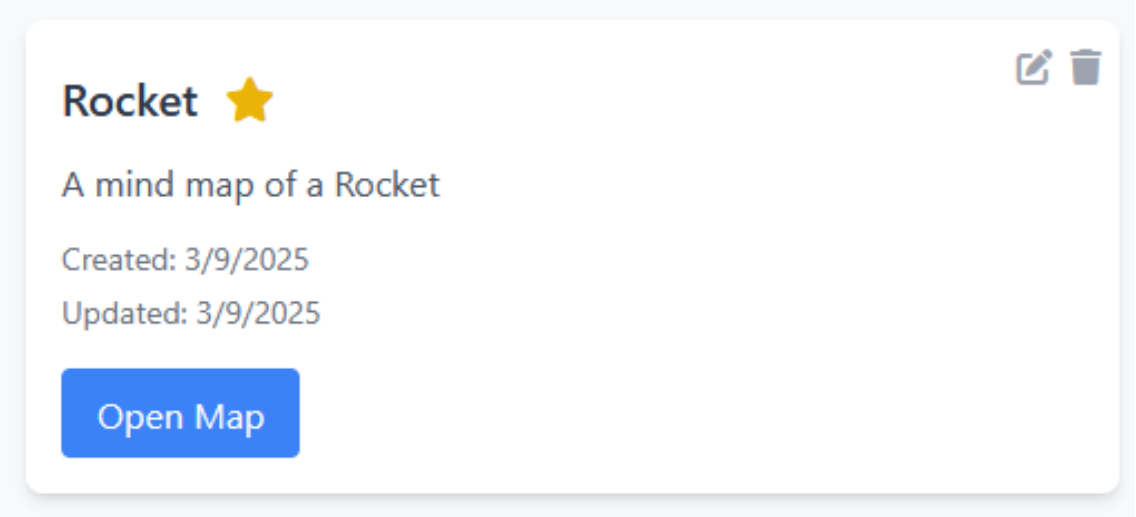
Idea-Atlas je proveden v jednoduchém, čistém stylu. Dominantními barvami jsou modrá, světle šedá a bílá. Snažil jsem se, aby uživatelské rozhraní působilo přehledně a moderně, ale nikoliv však rušivě. Schválně jsem se vyhnul komplikovaným designovým prvkům a animacím, protože to není předmětem této práce.

Zde jsou vyobrazeny ukázky těch zajímavějších částí webové aplikace. Na obrázku 6 je k nalezení hero nadpis úvodní domovské stránky. Na něm bych vypíchnul minimalistický design pomocí gradientů a poklidnou jednoduchou animaci pohupování.[16]



Obrázek 6: Hero nadpis

Na obrázku 7 je k vidění kartička jedné z map. Je na ní vidět, zda je bookmarknutá, kdy byla mapa vytvořena, kdy naposledy upravena, její jméno a popis.[17] Dále jsou zde ikonky pro smazání dané mapy nebo upravení jejích metadat.



Obrázek 7: Kartička mapy

7 Vizualizace myšlenkových map

O zobrazování myšlenkových map se stará knihovna V-network-graph. Práce s ní byla vcelku přímočará, veškeré informace jsem čerpal z oficiální dokumentace s ukázkami [18]. Postupně jsem si z jednotlivých ukázek zaměřených vždy na jednu problematiku vybral, co potřebuji, a sestavil jsem z toho finální komplexní nástroj.
Samotná knihovna poskytuje následující funkce:

1. zoom
2. přesouvání se po mapě
3. přidávání a odebírání vrcholu
4. přidávání a odebírán hrany
5. podpora customizace - barvy, velikosti, animace vrhcolů a hran
6. dynamické překreslování (například při změně barvy vrcholu)
7. interagování s grafem
8. integrace force layoutu pomocí D3.js
9. zobrazení mřížky

Jak již bylo zmíněno, vše dohromady přichází k "životu"v souboru *MapNetwork.vue* . Veškerá funkcionalita a interaktivita s grafem je zřízena v souboru *graphManager.js*, nacházející se zde funkce pro správu vrcholů, hran atd. Ukázka 1 dává za příklad jednoduchou funkci pro přidání nového vrcholu. Vypočítá nové id a namapuje hodnoty pro nový vrchol.

Ukázka 1: utils/graphManager.js, přidání vrcholu do grafu

```
// adds a new node to the graph
function addNewNode(data, nodeProps, xMousePos, yMousePos) {
    // Find the largest numeric part of the existing keys
    const maxNodeId = findCurrentMaxNodeId(data);
    // Generate the next key
    const nextNodeId = `node${maxNodeId + 1}`;

    // Add the new node with all properties
    data.nodes[nextNodeId] = {
        name: nodeProps.name,
        color: nodeProps.color,
        size: nodeProps.size
    };
    // Adds the new node to the layouts so its position is tracked
    data.layouts.nodes[nextNodeId] = { x: xMousePos, y: yMousePos };

    historyManager.addToHistory(data);

}
```

Nachází se zde ale i komplikovanější funkce, například *waveNodeSelect* viz ukázka 2. Tato funkce rekurzivně přidává vrcholy, které jsou potomky těch již označených, do označených vrcholů. Tato funkcionalita rozšiřuje možnosti editace myšlenkových map.

Ukázka 2: utils/graphManager.js, wave select

```
264 // Function which will recusively select all the nodes connected to the selected node
265 async function waveNodeSelect(data, selectedNodes, callback) {
266     const visited = new Set(selectedNodes);
267     const newlySelected = [];
268 
269     // Get connected nodes that haven't been visited
270     Object.values(data.edges).forEach(edge => {
271         const source = edge.source;
272         const target = edge.target;
273 
274         if (selectedNodes.includes(source) && !visited.has(target)) {
275             visited.add(target);
276             newlySelected.push(target);
277         }
278         if (selectedNodes.includes(target) && !visited.has(source)) {
279             visited.add(source);
280             newlySelected.push(source);
281         }
282     });
283 
284     if (newlySelected.length > 0) {
285         callback(newlySelected);
286     }
287 
288     return newlySelected;
289 }
```

Další zajímavá implementovaná funkce zařizuje automatické generování šířek a barev hran na základě spojených vrcholů. Tato funkce vždy vychází z většího vrcholu, přepočítá se jeho velikost na šířku hrany pomocí konstanty, dále hrana zdědí i barvu většího vrcholu; akorát má slabší alpha kanál, o tom se více dočtete v kapitole 7.2. Na ukázce 3 je možno vidět tuto funkcionalitu ve funkci pro přidání hran mezi všemi vybranými vrcholy *addEdges*.

Ukázka 3: utils/graphManager.js, přidání hran a jejich automatická konfigurace

```
92  // Function which will add edges between all selected nodes
93  function addEdges(data, selectedNodes) {
94      // Return if selected nodes are less than 2 - cannot create an edge
95      if (selectedNodes.length < 2) return;
96
97      const maxEdgeId = findCurrentMaxEdgeId(data);
98      // Tracks how many new edges are created
99      // So the next edge id can be generated
100     let newEdgeCounter = 0;
101
102     for (let srcIndex = 0; srcIndex < selectedNodes.length - 1; srcIndex++) {
103         for (let trgtIndex = srcIndex + 1; trgtIndex < selectedNodes.length; trgtIndex++) {
104
105             const source = selectedNodes[srcIndex];
106             const target = selectedNodes[trgtIndex];
107             if (!edgeExists(data, source, target)) {
108                 // Increase the counter by 1
109                 newEdgeCounter++;
110                 // Creates a new edge id
111                 const nextEdgeId = `edge${maxEdgeId + newEdgeCounter}`;
112                 const edgeProps = getLargerNodeProperties(data, source, target);
113                 data.edges[nextEdgeId] = {
114                     source,
115                     target,
116                     color: edgeProps.color,
117                     width: edgeProps.width
118                 };
119                 // .....
```

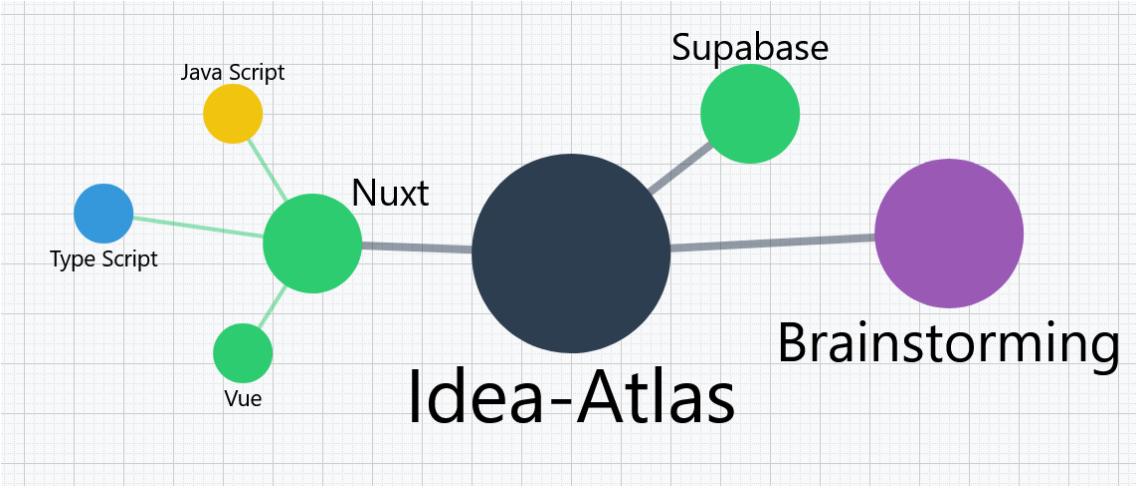
7.1 Implementace grafů

Grafy jsem implementoval tak, v jaké podobě je vyžaduje V-network-graph, tedy v podobě 3 seznamů: *nodes, edges, layouts*. Výsledný nástroj není koncipovaný pro velké množství záznamů v jednom grafu, nebylo tedy potřeba vytvářet žádnou speciální implementaci grafu. Ukázkový graf viz ukázka 4.

Ukázka 4: Implemetnace ukázkového grafu

```
const nodes: Nodes = {
  node1: { name: "Idea-Atlas", color: "#2c3e50", size: 100 },
  node2: { name: "Nuxt",color: "#2ecc71", size: 50 },
  node3: { name: "Vue", color: "#2ecc71", size: 30 },
  node4: { name: "Java Script", color: "#f1c40f", size: 30 }
}
const edges: Edges = {
  edge1: { source: "node1", target: "node2", color: "#2c3e50", width: 8 },
  edge2: { source: "node2", target: "node3", color: "#2ecc71", width: 4 },
  edge3: { source: "node4", target: "node2", color: "#2ecc71", width: 4 }
}
const layouts: Layouts = {
  nodes: {
    node1: { x: 200, y: 40 },
    node2: { x: -110, y: 80 },
    node3: { x: -410, y: 160 },
    node4: { x: -310, y: -160 }
  },
}
```

O trochu složitější graf pak může vypadat jako ten na obrázku 8.



Obrázek 8: Ukázkový graf

7.2 Konfigurace V-network-graph

Konfigurace této knihovny hraje v mém projektu zásadní roli. Konfigurace se postupně měnila a rostla spolu s komplexitou projektu. Zejména se jedná o upravené, personalizované výtažky z dokumentace V-network-graph [18]. Kompletní konfigurační soubor *mapNetworkConfig.ts* je k nalezení v adresáři *config*.

Na ukázce 5 se nachází první část konfigurace V-network-graph, konfiguruje se zde na příklad:

1. zdali je povolen box selection a jak vypadá
2. vzhled mřížky na pozadí
3. vypnutí zvětšování objektů podle přiblížení
4. minimální a maximální přiblížení, které může uživatel provést
5. zakázání přiblížení při dvojitém kliku; v mém projektu má dvojtý klik funkci přidání nového uzlu

Ukázka 5: config/mapNetworkConfig.ts, první část konfigurace vNG

```
56 view: {
57     /...
58       boxSelectionEnabled: true,
59       selection: {
60         box: {
61           color: "#0000ff20",
62           strokeWidth: 1,
63           strokeColor: "#aaaaff",
64           strokeDasharray: "0",
65         },
66       },
67       grid: {
68         visible: true,
69         interval: 10,
70         thickIncrements: 5,
71         line: {
72         //...
73         },
74         thick: {
75           color: "#cccccc",
76           width: 1,
77           dasharray: 0,
78         },
79       },
80       layoutHandler: new vNG.GridLayout({ grid: 10 }),
81       scalingObjects: true,
82       minZoomLevel: 0.1,
83       maxZoomLevel: 16,
84       doubleClickZoomEnabled: false,
85     },
86     /...
```

Na ukázce 6 se konfigurují vlastnosti a vzhled vrcholů:

1. jak vypadá vrchol v normálním stavu
2. jak se změní vrchol, když na něj uživatel najede myší
3. jak vypadá popisek vrcholu; dále se tam zapíná automatické posouvání popisku, aby nepřekážel hranám, které vedou z daného vrcholu

Ukázka 6: config/mapNetworkConfig.ts, druhá část konfigurace vNG

```
87    node: {
88      normal: {
89        type: "circle",
90        radius: node => node.size, // Use the value of each node object
91        color: node => node.color,
92      },
93      hover: {
94        radius: node => node.size + 4,
95        color: node => node.color,
96      },
97      selectable: true,
98      label: {
99        visible: true,
100       fontFamily: undefined,
101       fontSize: node => calculateFontSize(node.size),
102       lineHeight: 1.1,
103       color: "#000000",
104       margin: 4,
105       direction: "south",
106       directionAutoAdjustment: true,
107       background: {
108         visible: false,
109         color: "#ffffff",
110         padding: {
111           vertical: 1,
112           horizontal: 4,
113         },
114         borderRadius: 2,
115       },
116     },
117   },
```

Na ukázce 7 se nastavuje:

1. zda se dají vybrat hrany
2. jak vypadají hrany v normálním stavu
3. co se stane s hranami, když na ně uživatel namíří myšítkem
4. jak budou hrany vypadat, když je uživatel označí

Ukázka 7: config/mapNetworkConfig.ts, třetí část konfigurace vNG

```
118 edge: {
119     selectable: true,
120     normal: {
121       width: edge => edge.width, // Use the value of each edge object
122       color: edge => convertToRGBA(edge.color),
123     },
124     hover: {
125       width: edge => edge.width,
126       color: edge => edge.color, // Use original color without transparency
127     },
128     selected: {
129       width: edge => edge.width * 1.5, // Increase width by 50% when selected
130       color: edge => edge.color,
131       dasharray: "6", // Fixed dash pattern for selected edges
132       animate: true, // Always animate selected edges
133     },
134   },
```

8 Backend

Vybral jsem si Supabase jako řešení backendu pro svůj projekt, protože nabízí open-source alternativu k Firebase s výkonnou PostgreSQL databází, která umožňuje flexibilní práci s daty, dále nabízí i bezplatný tier služby pro vývoj. Díky vestavěné podpoře pro autentizaci, realtime synchronizaci a serverless funkce poskytuje moderní a škálovatelné řešení bez nutnosti správy složité infrastruktury. Díky modulům se Supabase dobře integruje s Nuxt 3 a usnadňuje vývoj díky přehlednému API a skvělé dokumentaci.[19, 20]

8.1 Ověření uživatele

Supabase využívá tabulku *auth.users* pro ověřování uživatelů, kterou spravuje automaticky. Když se někdo zaregistruje pomocí e-mailu a hesla, jeho účet se uloží do této tabulky. Zároveň mu systém automaticky odešle e-mail s potvrzovacím odkazem, který je nutné kliknutím ověřit, aby byl účet aktivován.[21, 22]
Kromě toho Supabase podporuje i další metody autentizace, jako je přihlášení přes externí poskytovatele (např. Google, GitHub) nebo přihlášení pomocí magic linku, přičemž Supabase zajišťuje bezpečnost a šifrování citlivých údajů.[21] Bohužel kvůli zbytečné složitosti ze strany Googlu při přidání Google authenticatoru jsem se rozhodl zůstat pouze u e-mailu a klasického hesla.

8.2 Middleware

Middleware je software nebo kód, který funguje jako prostředník mezi různými částmi aplikace, například mezi klientem a serverem nebo mezi různými vrstvami v aplikaci. Obvykle se používá k manipulaci s požadavky a odpověďmi, autentizaci, logování, zpracování chyb nebo modifikaci dat předtím, než se dostanou k hlavní aplikaci. [23]
V mém projektu slouží middleware k tomu, abych ověřil, zda je uživatel přihlášený, když se snaží dostat na části webové stránky, kde je to nutné; například na uživatelský profil.[24]. Na ukázce 8 je možné vidět jednoduchý kód, který ověří, zdali je uživatel přihlášen, a pokud není, přesměruje ho na login page.

Ukázka 8: middleware/auth.js, middleware

```
export default defineNuxtRouteMiddleware(() => {
    const user = useSupabaseUser();
    if (!user.value) {
        return navigateTo('/login');
    }
});}
```

Middleware pak jde snadno použít na libovolné stránce. Jeho názorné použití lze najít na ukázce 9

Ukázka 9: pages/profile.vue, ověření přihlášeného uživatele

```
definePageMeta({
    middleware: ["auth"],
});
```

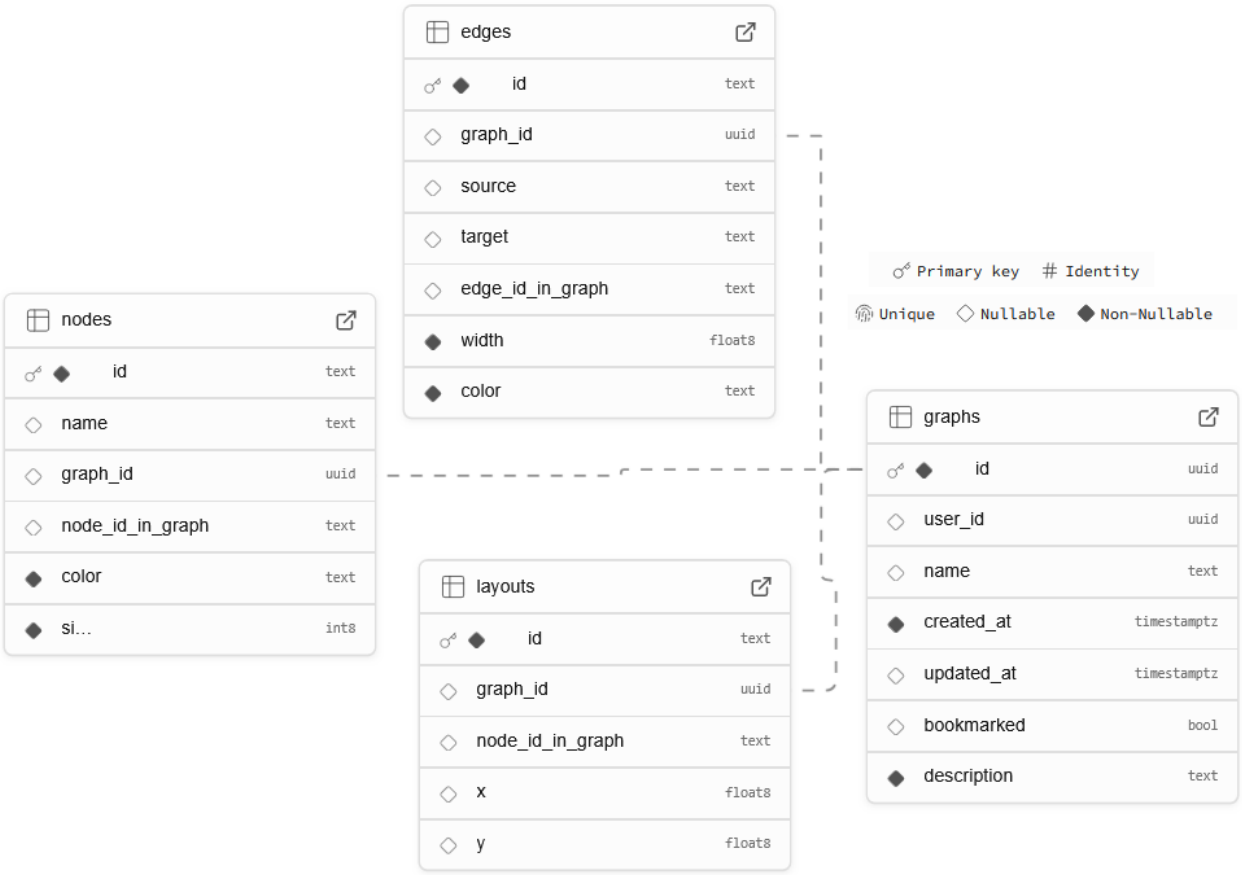
8.3 Struktura databáze

Data jsou ukládána do 4 tabulek viz obrázek 9. Tyto tabulky jsou propojené přes *graph id*. Tedy každý node, edge a layout spadá pod nějaký graf.

Za zmínku také stojí, že kvůli jednodušší implementaci s knihovnou v-network-graph, jsem se rozhodl pro propojení jednotlivých prvků grafu přes *id in graph* a nikoliv propojení samotných tabulek pomocí cizího klíče. Jelikož každý node je ve v-network-graph grafové struktuře používán jako klíč, tak někdy i jako hodnota objektu viz ukázka 10. Tohle mé řešení sice není konvenční, nicméně mi to ulehčilo práci při mapování získaných dat z databáze.

Ukázka 10: Demonstrace problému s mapováním

```
// Ukazkova data
const nodes: Nodes = {
  node1: { name: "Rocket", color: "#2c3e50", size: 100 },
  node2: { name: "Fuel",color: "#2ecc71", size: 50 },
}
const edges: Edges = {
  edge1: { source: "node1", target: "node2", color: "#2c3e50", width: 8 },
}
```



Obrázek 9: Schéma databáze

8.4 Dotazy do databáze

Dotazy do databáze zařizují soubory *graphService.js* a *graphMetadataService.js*. Soubor *graphService* se stará přímo o data konkrétního grafu. Kdežto *graphMetadataService* se stará o metadata jednotlivých grafů, tedy posílá dotazy na jméno myšlenkové mapy, poslední datum editace a podobné.

Na ukázce 11 je k nalezení část funkce *fetchData*. Tato ukázka slouží jako demonstrace jednoduchého dotazu do databáze. Jako argument potřebuje objekt *useSupabaseClient()* a id grafu, jehož data chceme dotazem získat.[25]

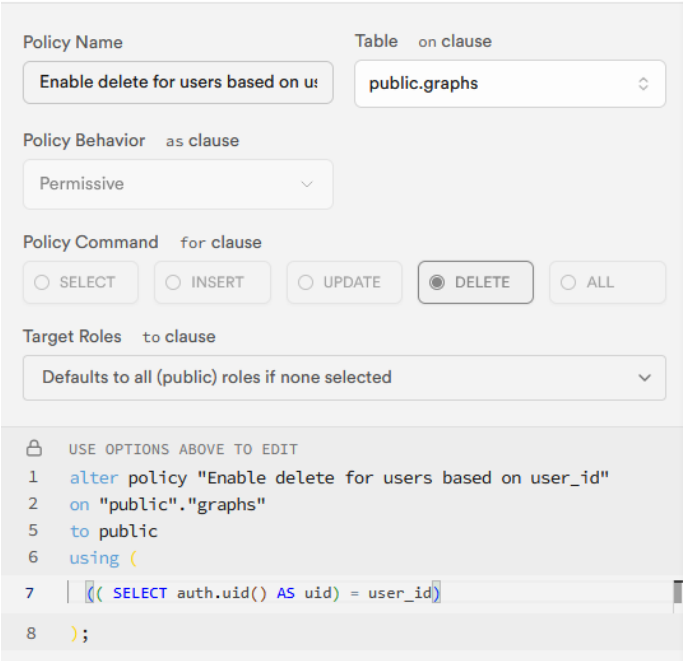
Ukázka 11: utils/graphService.js, dotaz do databáze

```
206 async function fetchData(supabase, graph_id) {
207   const { data: nodesData, error: nodesError } = await supabase
208     .from('nodes')
209     .select('*')
210     .eq('graph_id', graph_id);
211   if (nodesError) throw nodesError;
212 }
```

8.5 Row-level security

RLS (Row-Level Security) jsou v Supabase bezpečnostní pravidla, která určují, kdo může přistupovat k jednotlivým řádkům tabulky v databázi. Supabase používá PostgreSQL Row-Level Security (RLS) k omezení přístupu na základě definovaných podmínek.[26]

Na obrázku 10 je možné vidět RLC pro tabulku *graphs* . Tato podmínka je vyhodnocována vždy při dotazu a danou akci databáze provede pouze, jestliže je pravidlo splněno. V tomto konkrétním případě může smazat user jenom svůj vlastní graf.



Obrázek 10: Příklad Supabase RLC

8.6 Ukládání změn v grafu

Na ukázce 12 je popsána funkce, která kompletně řeší problematiku uložení dat grafu. Nejprve pošle dotaz do databáze na získání všech ID pro daný graf. Následně vyfiltruje pouze ta ID, která jsou uložena v databázi, ale už ne lokálně v prohlížeči [27]; uživatel je tedy smazal. Pošle delete dotaz do databáze na tato vybraná data. Následně upsertne všechna data, která graf v prohlížeči obsahuje. Nová data jsou tedy přidána a změněná jsou updatnuta. Jde o jednoduché v celku elegantní řešení, které je konzistentní a spolehlivé.

Ukázka 12: utils/graphService.js, uložení grafu

```
168 async function saveGraph(supabase, data, graph_id) {
169   try {
170     // Get current IDs from database
171     const ids_in_db = await fetchAllIds(supabase, graph_id);
172
173     // Find items to delete
174     // __ is a placeholder for the layoutsToDelete variable - they are not used
175     const { nodesToDelete, edgesToDelete, __ } = await filterDeletedData(data, ids_in_db);
176     // First delete edges
177     await deleteEdges(supabase, edgesToDelete, graph_id);
178     //....
179     // Now upsert all current data
180     await upsertGraphData(supabase, data, graph_id);
181
182     // Update the graph's updated_at timestamp
183     const { error: updateError } = await supabase
184       .from('graphs')
185       .update({ updated_at: new Date().toISOString() })
186       .eq('id', graph_id);
187 }
```

8.7 Automatické ukládání

Při práci s interaktivními grafy je důležité zajistit, aby uživatelské změny byly automaticky ukládány, aniž by docházelo k nadměrnému zatěžování databáze. V komponentě *MapNetwork.vue* je tato funkcionalita implementována pomocí debounce vzoru, který minimalizuje počet zbytečných požadavků na server a zároveň uchovává data uživatele v bezpečí. [28]

Aby bylo možné efektivně reagovat na časté změny v grafu, využívá implementace časovou prodlevu mezi úpravou a samotným uložením. *AUTO SAVE DELAY* je nastaven na 2 sekundy, což znamená, že pokud uživatel provede změnu, systém čeká, zda dojde k další úpravě. Teprve pokud během této doby žádná další změna nenastane, proběhne uložení.

Celý proces je řízen několika klíčovými prvky:

1. *saveTimeout* uchovává ID časovače, který řídí zpoždění ukládání.
2. *isSaving* sleduje, zda právě probíhá proces ukládání, což umožňuje zobrazit vizuální indikátor.

Při každé změně v grafu je nejprve zrušen existující časovač, aby se zabránilo vícenásobnému ukládání. Poté se nastaví nový časovač, který po uplynutí definované prodlevy spustí funkci *handleSave()*, jež provede uložení dat do Supabase.
Aby systém spolehlivě detekoval jakékoli změny, využívá Vue watcher, který sleduje uzly, hrany a rozložení grafu s nastavením deep: true. To znamená, že změny na jakékoli úrovni těchto objektů automaticky aktivují proces ukládání.

Ukázka 13: components/MapNetwork.vue, automatické ukládání

```
165 const saveTimeout = ref<NodeJS.Timeout | null>(null);
166 const isSaving = ref(false);
167
168 // Debounced auto-save function
169 const debouncedSave = () => {
170   if (saveTimeout.value) {
171     clearTimeout(saveTimeout.value);
172   }
173
174   saveTimeout.value = setTimeout(async () => {
175     isSaving.value = true;
176     try {
177       await handleSave();
178     } finally {
179       isSaving.value = false;
180     }
181   }, AUTO_SAVE_DELAY);
182 };
183
184 // Watch for changes in data and trigger auto-save
185 watch(
186   () => [data.nodes, data.edges, data.layouts],
187   () => {
188     debouncedSave();
189   },
190   { deep: true }
191 );
```

9 Klíčové problémy

Během mé práce jsem se setkal s mnoha problémy, spoustu z nich mělo jednoduché řešení, jiné byly o mnoho komplikovanější a zásadní pro mou vizi funkčnosti tohoto webového nástroje. V této kapitole jsou popsány ty nejvýznamnější z nich.

9.1 Automatické uspořádání

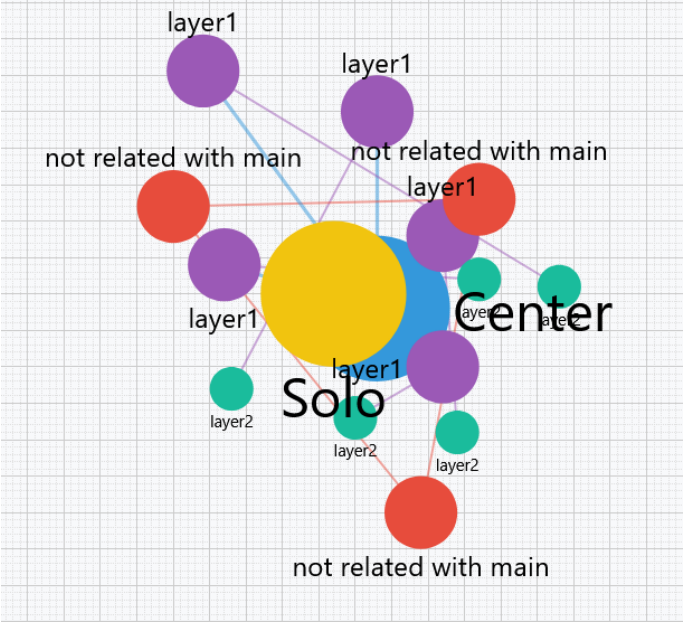
Chtěl jsem, aby Idea-Atlas disponoval funkcí automatického uspořádání vrcholů, tak aby vznikl na první pohled přehledný graf. Ukázalo se, že se jedná o velmi těžkou úlohu. Zprvu jsem se snažil přijít s vlastním algoritmem, ale to velmi rychle selhalo. Následně jsem se snažil na můj problém aplikovat *Force-directed graph drawing* algoritmus [29]; jde vcelku o komplexní algoritmy, bohužel má implementace selhala. Následně jsem zjistil, že v-network-graph má v sobě zabudované propojení s D3.js. Dohromady tak tyto 2 knihovny nabízejí konfigurovatelný force layout, který jsem metodou pokus-omyl nastavil.

Ukázka 14: config/mapNetworkConfig.ts, Implemetnace force layout

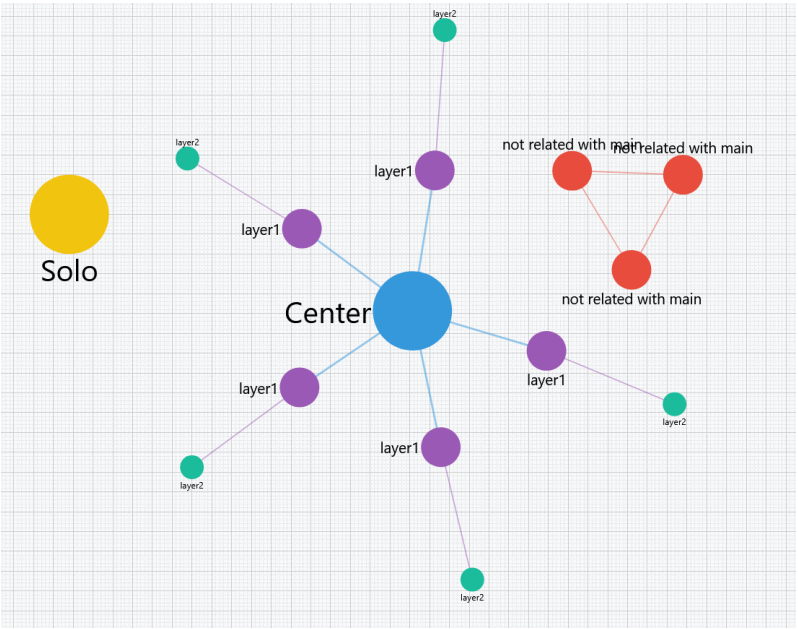
```
export const ForceConfig = new ForceLayout({
  positionFixedByDrag: false, // Nodes continue simulation after being dragged
  positionFixedByClickWithAltKey: true, // Alt+Click fixes node position
  createSimulation: (d3, nodes, edges) => {
    // Create links between nodes with specified parameters
    const forceLink = d3.forceLink<ForceNodeDatum, ForceEdgeDatum>(edges)
      .id((d: { id: any; }) => d.id) // Use node's id property to establish connections

    return d3
      .forceSimulation(nodes)
      // Edge force: maintains distance between connected nodes
      .force("edge", forceLink.distance(FORCE_LINK_DISTANCE).strength(FORCE_LINK_STRENGTH))
      // Charge force: makes nodes repel each other
      .force("charge", d3.forceManyBody().strength(FORCE_CHARGE_STRENGTH))
      // Center force: pulls nodes toward canvas center
      .force("center", d3.forceCenter().strength(FORCE_CENTER_STRENGTH))
      .alphaMin(FORCE_ALPHA_MIN) // Minimum energy level before simulation stops
  }
});
```

Force layout automaticky a dynamicky simuluje přitažlivé síly jednotlivých hran a odpudivé síly vrcholů, výsledkem je přehledné uspořádání grafu, viz obrázky 11 a 12. Uživatel si může přepínat mezi normálním grid layoutem nebo mezi force layoutem.



Obrázek 11: Chaotické uspořádání vrcholů



Obrázek 12: Uspořádání po použtí force layout

9.2 Historie editací

K správě historie editací je používán soubor *graphHistory.js*. Obsahuje jednu třídu *HistoryManager*. Je založena na hlavním poli *history*, elementy tohoto pole jsou kopie dat grafu v určitý okamžik. Historie může mít celkem až 100 záznamů (aby nedošlo k zbytečnému plýtvání pamětí). Dále je zde index, jenž určuje aktuální polohu v historii.

Ukázka 15: utils/graphHistory.js, proměné třídy HistoryManager

```
4  class HistoryManager {
5      constructor() {
6          this.history = [];
7          this.currentIndex = -1;
8          this.maxHistory = HISTORY_MAX_LENGTH; // Maximum number of saved states
9      }
```

Při každé změně dat grafu (například při přidání nového vrcholu) se udělá jejich kopie a přidá se do historie. Následně se smaže veškerá budoucí historie; jde o případ, kdy jsme se nejdříve vrátili do minulosti a pak provedly nové úpravy - tedy tu minulé musí být odstraněny. Následně zvětšíme index současného bodu v historii a pokud jsme přesáhli velikost historie, jsou data posunuta směrem na začátek a první záznam je smazán a index současnosti je zmenšen o jedna.

Ukázka 16: utils/graphHistory.js, přidání do historie

```
13 addToHistory(data) {
14        // Create deep copy using structuredClone
15        const dataCopy = {
16        // .....
17        };
18        // Remove any future history entries
19        this.history = this.history.slice(0, this.currentIndex + 1);
20
21        this.currentIndex++;
22        this.history[this.currentIndex] = dataCopy;
23        // If currentIndex is greater than maxHistory, remove the first element
24        // And shift all elements to the left
25        if (this.currentIndex > this.maxHistory) {
26            this.history.shift();
27            this.currentIndex--;
28        }
29    }
```

9.3 Převod sousřadnic mezi SVG a DOM

Aby program věděl, kde se má zobrazit dialog pro vytvoření vrcholu nebo jeho editaci. Musel jsem převést souřadnice z SVG souřadnicového systému (tedy toho, co používá vNG) do DOM souřadnicového systému (souřadnice v okně prohlížeče).[30]

Ukázka 17: components/MapNetwork.vue, převod z Svg souřadnic do Dom

```
385  // Translate from SVG to DOM coordinates
386 const domPoint = graph.value.translateFromSvgToDomCoordinates({ x: nodeLayout.x, y: nodeLayout.y });
```

Dále byla potřeba omezit, kde až se může dialog objevit. Například by nebylo přehledné a intuitivní, kdyby se dialog objevil příliš nahoře a nebyl by tak tedy vidět. Funkce posune souřadnice dolů, nahoru, doleva nebo doprava, podle vzájemné pozice od okrajů okna v prohlížeči, viz ukázka 18. [31]

Ukázka 18: components/MapNetwork.vue, výpočet offset dialogu

```
395   const windowWidth = window.innerWidth;
396   const windowHeight = window.innerHeight;
397
398   // Calculate horizontal offset
399   let xOffset = domPoint.x < windowWidth / 2 ? DIALOG_WIDTH/2 + PADDING : -(DIALOG_WIDTH/2 + PADDING);
400
401     // Calculate vertical offset
402   let yOffset = 0;
403   if (domPoint.y < DIALOG_HEIGHT + PADDING) {
404     // Too close to top - move down
405     yOffset = (DIALOG_HEIGHT/2 + PADDING) - domPoint.y;
406   } else if (domPoint.y > windowHeight - DIALOG_HEIGHT - PADDING) {
407     // Too close to bottom - move up
408     yOffset = (windowHeight - DIALOG_HEIGHT/2 - PADDING) - domPoint.y;
409   }
410
411   return {
412     x: domPoint.x + xOffset,
413     y: domPoint.y + yOffset
414     };
```

9.4 Správa UI stavů

Další důležitý problém, který bylo potřeba vyřešit, bylo, jak správně spravovat stavy UI. Například jak zařídit, aby se nesmazal vrchol, když klávesou *backspace* smažu znaky při editaci jména vrcholu. Vytvořil jsem samotný soubor *uiTracker.ts* ve složce *utils*, který kontroluje jednotlivé UI komponenty, zdali nejsou otevřené. Implementace je velice jednoduchá, za to účinná a přehledná.

9.5 Přetahovatelné UI

useDraggable.ts umožňuje přetahování HTML prvků. Poskytuje funkci *startDrag*, která se stará o výpočet počáteční pozice, správu událostí myši a aktualizaci pozice prvku. Po kliknutí na prvek se určí jeho aktuální souřadnice a vypočítá se posun mezi místem kliknutí a jeho skutečnou polohou. Během přetahování se aktualizují CSS vlastnosti *style.left* a *style.top*, přičemž *callback onPositionChange* umožňuje sledování nových souřadnic, viz ukázka 19. Jakmile uživatel pustí tlačítko myši, obslužné funkce pro pohyb a ukončení přetahování se odstraní, aby nedocházelo k nežádoucím interakcím. Tento hook poskytuje jednoduché a efektivní řešení pro přetahování prvků v rámci Vue komponentů.[32]

Ukázka 19: utils/useDraggable.ts, přetahovatelné UI

```
export function useDraggable() {
  const startDrag = (event: MouseEvent, element: HTMLElement,
  onPositionChange: (x: number, y: number) => void) => {
    // Calculate the initial position from the element's current position
    const currentX = parseInt(element.style.left) || 0;
    const currentY = parseInt(element.style.top) || 0;

    // Calculate the offset from where we clicked relative to the current position
    const offsetX = event.clientX - currentX;
    const offsetY = event.clientY - currentY;

    const handleMove = (moveEvent: MouseEvent) => {
      const newX = moveEvent.clientX - offsetX;
      const newY = moveEvent.clientY - offsetY;

      element.style.left = `${newX}px`;
      element.style.top = `${newY}px`;
      onPositionChange(newX, newY);
    };// ....
```

10 Návrhy na zlepšení

Ačkoliv je webová stránka ve funkčním stavu, vždy se najde, co vylepšit a jaké nové funkce implementovat. Zde je seznam pár věcí, které jsem nestihl nebo jsem od nich upustil.

10.1 Personalizace stránky

Uživatel by si mohl nastavit svůj preferovaný barevný vizuál. Dále by bylo na výběr z několika jazyků; zejména bych rád implementoval český překlad. Uživatelský profil by mohl hrát větší roli, mohl by se upravovat - profilový obrázek, jméno, osobní informace. To by bylo zejména užitečné s implementací popsanou v kapitole: 10.2

10.2 Realtime spolupráce uživatelů

Tato funkcionalita by umožnila uživatelům sdílet myšlenkové mapy mezi sebou. Mohli by je spolu v reálném čase upravovat. Takováto funkce by udělala můj nástroj ještě užitečnější.

10.3 ChatGPT api

Bohužel se mi nepodařilo zařadit ChatGPT API do projektu. ChatGPT API je placená služba a na rozdíl od Supabase nenabízí žádný bezplatný zkušební plán. Nepodařilo se mi najít žádné konkurenční služby, které by to umožnily. Jako další možnost se nabízela cesta vlastního hostování jazykového modelu; to by ovšem výrazně zvětšilo komplexitu projektu a navíc je to velmi výkonnostně velmi náročné.

10.4 Deploy

Nasazení (deploy) mého projektu by rozhodně zlepšilo jeho dostupnost a použitelnost. V současnosti je omezený na lokální prostředí, což komplikuje testování a sdílení s uživateli. Pokud bych jej nasadil na produkční server nebo cloudovou platformu, umožnilo by to nepřetržitý přístup odkudkoli, snadnější iteraci a případné škálování podle potřeby.

11 Závěr

Idea-Atlas je komplexní projekt s důrazem na detail. Na projektu jsem pracoval průběžně a strategicky jsem si jeho vyhotovení rozvrhl.

Zadání se mi podařilo téměř celé splnit. Vytvořil jsem plně funkční webovou aplikaci, která slouží jako přehledný a snadno ovladatelný nástroj pro tvorbu myšlenkových map. Bohužel se mi ze zmíněných důvodů nepodařilo implementovat ChatGPT API, a aplikace tak neumožňuje automatickou generaci pojmů a jejich závislostí. Podařilo se mi úspěšně nalézt i použít moderní technologie hodící se pro řešení zadané problematiky. Celkově vnímám Idea-Atlas jako velmi přínosný projekt, který mi pomohl prohloubit znalosti a posunout se dál nejen v oblasti vývoje webových aplikací.

12 Použité zdroje

[1] *NUXT: the Progressive Web Framework*. URL: https://nuxt.com/.

[2] *Vue.js*. URL: https://vuejs.org/.

[3] *v-network-graph*. URL: https://dash14.github.io/v-network-graph/.

[4] *D3 by Observable | The JavaScript library for bespoke data visualization*. URL: https://d3js.org/.

[5] *Supabase | the open source Firebase alternative*. URL: https://supabase.com/.

[6] *Tailwind CSS for Nuxt*. URL: https://tailwindcss.nuxtjs.org/.

[7] Bun. *Bun — A fast all-in-one JavaScript runtime*. URL: https://bun.sh/.

[8] *ChatGPT - brainstorming a myšlenkové mapy*. URL: https://chatgpt.com/share/67e4219c-845c-8000-b8fa-1a9d677b3e18.

[9] *What is Brainstorming? Techniques and Methods | Miro*. URL: https://miro.com/brainstorming/what-is-brainstorming/.

[10] Sprouts. *Brainstorming techniques: How to innovate in groups*. 28. lis. 2017. URL: https://www.youtube.com/watch?v=YXZamW4-Ysk.

[11] Mindmaps.com. *What is Mind Mapping? What Are Its Uses? | Mindmaps.com*. 18. srp. 2021. URL: https://www.mindmaps.com/what-is-mind-mapping/.

[12] Mindmaps.com. *How to Brainstorm with Mind Maps - Mindmaps.com*. 8. zář. 2020. URL: https://www.mindmaps.com/how-to-brainstorm-with-mind-maps/.

[13] *ChatGPT - Routování v Nuxtu*. URL: https://chatgpt.com/share/67e4652b-ee14-8000-a860-66efb9a5e7da.

[14] *Vue.js-Components*. URL: https://vuejs.org/guide/essentials/component-basics.

[15] *Auto-imports · NUXt Concepts*. URL: https://nuxt.com/docs/guide/concepts/auto-imports.

[16] *45+ CSS Glow effects*. 21. říj. 2022. URL: https://freefrontend.com/css-glow-effects/.

[17] Coding Nation. *Beautiful Simple Card Design using Tailwind CSS*. 12. zář. 2021. URL: https://www.youtube.com/watch?v=B2PVKb5L0wo.

[18] *v-network-graph examples and docs*. URL: https://dash14.github.io/v-network-graph/examples/.

[19] *Nuxt Supabase - home*. URL: https://supabase.nuxtjs.org/.

[20] *Supabase Docs*. 26. březen 2025. URL: https://supabase.com/docs.

[21] *AuTh | Supabase Docs*. 22. březen 2025. URL: https://supabase.com/docs/guides/auth.

[22] John Komarnicki. *Easily add authentication with NUXT 3 + Supabase*. 14. červen 2023. URL: https://www.youtube.com/watch?v=yO-JMkjc4oo.

[23] GeeksforGeeks. *Middleware: Categories, Types, working, Advantage and Disadvantage*. 1. červenec 2024. URL: https://www.geeksforgeeks.org/what-is-middleware/.

[24] LearnVue. *Auth in Nuxt 3 The EASY WAY*. 22. srpen 2022. URL: https://www.youtube.com/watch?v=IcaL1RfnU44.

[25] Net Ninja. *Supabase Tutorial 3 - Fetching data*. 1. září 2022. URL: https://www.youtube.com/watch?v=VjohMDwjty4.

[26] *Row Level Security | Supabase Docs*. 22. březen 2025. URL: https://supabase.com/docs/guides/database/postgres/row-level-security.

[27] *W3Schools.com*. URL: https://www.w3schools.com/jsref/jsref_filter.asp.

[28] *ChatGPT - Auto-Save Implementation Tips*. URL: https://chatgpt.com/share/67e56f7f-43fc-8000-b39e-6bc526cf8f8b.

[29] Wikipedia contributors. *Force-directed graph drawing*. 25. říjen 2024. URL: https://en.wikipedia.org/wiki/Force-directed_graph_drawing.

[30] *v-network-graph Miscellaneous - Coordinates translation*. URL: https://dash14.github.io/v-network-graph/examples/misc.html#coordinates-translation.

[31] *W3Schools.com - Current Screen Size*. URL: https://www.w3schools.com/howto/howto_js_get_current_window.asp.

[32] *W3Schools.com - Draggable DIV Element*. URL: https://www.w3schools.com/howto/howto_js_draggable.asp.

Seznam obrázků

Seznam ukázek kódu