

UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI

FAKULTA PRÍRODNÝCH VIED



Efektívna vizualizácia grafových databáz v trojrozmernom priestore

Diplomová práca

74bf893e-af7e-4273-8952-09fa0bb7d9f8

Študijný program: Aplikovaná informatika

Študijný odbor: 18. informatika

Školiace pracovisko: KIN FPV - Katedra informatiky

Vedúci práce: doc. Mgr. Adam Dudáš, PhD.

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Adam Kleinedler
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: Magisterská záverečná práca
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Efektívna vizualizácia grafových databáz v trojrozmernom priestore

Anotácia: V rámci doposiaľ dosiahnutých výsledkov v oblasti vizualizácie grafových databáz boli vždy využívané len dve dimenzie. Pri takýchto vizualizáciách je zjavné, že neplanárne databázy budú obsahovať pretínajúce sa elementy (objekty alebo vzťahy medzi nimi). Táto práca je práve preto zameraná na návrh a implementáciu nástroja vizualizácie grafových databáz v trojrozmernom priestore s cieľom odstránenia všetkých pretínajúcich sa grafových elementov, zvýšenou možnosťou interakcie s databázou a s kladením dôrazu na princípy efektívnej vizualizácie dát. Na experimentálne overenie vytvoreného nástroja použite najmenej tri grafové databázy rôznych veľkostí a porovnajte vytvorenú vizualizáciu s konvenčne používanými nástrojmi.

Vedúci: Mgr. Adam Dudáš, PhD.
Katedra: KIN FPV - Katedra informatiky
Vedúci katedry: doc. PaedDr. Patrik Voštinár, PhD.
doc. Ing. Jarmila Škrinárová, PhD.

Dátum zadania: 19.02.2025

Dátum schválenia smerovej katedry: 07.04.2025 prof. Dr. Ing. Miroslav Svítek, Dr.
osoba zodpovedná za realizáciu študijného programu

Čestné vyhlásenie

Čestne vyhlasujem, že som túto diplomovú prácu spracoval samostatne na základe získaných teoretických vedomostí a že všetku použitú literatúru a ďalšie pramene som v diplomovej práci vyznačil.

Podpis študenta

PodĎakovanie

V tejto časti by som rád vyjadril svoju vďaku vedúcemu mojej diplomovej práce, doc. Mgr. Adamovi Dudášovi, PhD., za jeho vedenie, odbornú podporu a cenné pripomienky, ktoré výrazne prispeli k dokončeniu mojej diplomovej práce.

Abstrakt

KLEINEDLER, Adam: Efektívna vizualizácia grafových databáz v trojrozmernom priestore. [Diplomová práca]. Univerzita Mateja Bela v Banskej Bystrici Fakulta prírodných vied, Katedra informatiky. Vedúci práce: doc. Mgr. Adam Dudáš, PhD. Stupeň odbornej kvalifikácie: magister – Mgr. Banská Bystrica, 2026. 58 strán.

Táto práca sa zaoberá návrhom, implementáciou a experimentálnym overením aplikácie na efektívnu vizualizáciu dát v grafových databázach v trojrozmernom priestore. V praxi je prístup k nástrojom, ktoré ponúkajú pokročilé metódy zobrazenia a interaktívnu prácu s veľkými grafmi, často obmedzený, pričom tradičné dvojrozmerné vizualizácie narážajú na problémy ako vysoká miera prekrytia a pretínania hrán, čo znižuje čitateľnosť a sťažuje analýzu. Navrhnutá aplikácia rozširuje možnosti bežne dostupných riešení tým, že poskytuje trojrozmerné zobrazenie grafu a viacero komplementárnych režimov vizualizácie: štandardný režim zameraný na prehliadanie jednotlivých uzlov a hrán, zhukový režim, ktorý zoskupuje množinu vrcholov do jedného kompozitného vrcholu za účelom zjednodušenia rozsiahlych grafov, prístup založený na Radial-BFS pre zhukovanie vrcholov rovnakého typu a analytické zobrazenie umožňujúce zorad'ovať a filtrovať vrcholy na základe hodnôt v nich vnorených kľúčov. Dôraz je kladený na interaktivitu a lepšie zobrazenie v priestore. Experimentálne overenie na vybraných dátach ukazuje, že trojrozmerný prístup zlepšuje čitateľnosť vizualizácie a podporuje rýchlejšiu interpretáciu komplexných vzťahových štruktúr v porovnaní s tradičnými prístupmi.

Kľúčové slová: Grafové Databázy, Vizualizácia dát, Nerelačné databázy, Zhluk, 3D vizualizácia.

Abstract

KLEINEDLER, Adam: Effective visualization of graph databases in threedimensional space. [Master's thesis]. University of Matej Bel in Banská Bystrica, Faculty of Natural Science, Department of Computer Science. Thesis supervisor: doc. Mgr. Adam Dudáš, PhD. Qualification Level: Masters – Mgr. Banská Bystrica, 2026. 58 pages.

This thesis focuses on the design, implementation, and experimental evaluation of an application for efficient visualization of data in graph databases within a three-dimensional space. In practice, access to tools that provide advanced rendering methods and interactive work with large graphs is often limited, while traditional two-dimensional visualizations face issues such as a high degree of occlusion and edge crossings, which reduce readability and hinder analysis. The proposed application extends the capabilities of commonly available solutions by providing a three-dimensional graph representation and several complementary visualization modes: a standard mode aimed at browsing individual nodes and edges, a clustered mode that groups a set of vertices into a single composite vertex in order to simplify large graphs, a Radial-BFS-based approach for clustering vertices of the same type, and an analytical view that enables grouping and filtering of nodes based on their attributes. The solution emphasizes interactivity and improved spatial representation. Experimental evaluation on selected data shows that the three-dimensional approach improves visualization readability and supports faster interpretation of complex relational structures compared to traditional approaches.

Keywords: Graph Databases, Data visualization, Non-relational Databases, Cluster, 3D Vizualization.

Zoznam obrázkov

1	Cloudové rozhranie Amazon Neptune	4
2	Cloudové rozhranie Neo4j AuraDB	6
3	Neo4j Desktop	7
4	Neo4j Desktop Dopytovanie	8
5	Data explorer yWorks	9
6	HeatMap yWorks	10
7	Gephi	11
8	Prvotný návrh aplikácie	14
9	Návrh Štandardného dopytovania v aplikácii	15
10	Návrh Formulárového vytvárania dopytu	16
11	Návrh pre spracovanie prirodzeného jazyka do dopytov	17
12	Návrh pre prihlásenie sa do databázy	18
13	Návrh pre prihlásenie sa do databázy	18
14	Topologické rozloženie v 3D priestore s kolíznou zložkou	20
15	Detail vybraného uzla s vypísaním jeho vlastností a súhrnných údajov. . .	20
16	Návrh zhlukovaného zobrazenia	22
17	Návrh podgrafu v zhlukovom zobrazení	22
18	Návrh filtrovaného zobrazenia pre jeden kľúč	23
19	Návrh filtrovaného zobrazenia pre dva kľúče	24
20	Návrh filtrovaného zobrazenia pre tri kľúče	25
21	Návrh sub-zhlukovaného zobrazenia	26
22	Implementovaná aplikácia 3D Graph database viewer	38
23	Hlavná navigačná lišta (Zmenšená pre účely čitateľnosti)	38
24	Bočná lišta	39
25	Metódy vykreslenia grafu	39
26	Modálne okno na dopyty	40
27	Výsledný podgraf po vykonaní dopytu	40
28	Syntakticky zlý dopyt	41
29	Low-code dopytovanie	41
30	LLM dopytovanie	42
31	Kontrola existujúceho databázového systému	42
32	Vkladanie mena a hesla	43
33	Databáza movies v topologickom rozložení	45
34	Rozklinutý vrchol	46
35	Databáza Messages v topologickom rozložení	47
36	Databáza shop v topologickom rozložení	48
37	Databáza movies v zhlukovanom rozložení	48

38	Modálne okno objektu Movies	49
39	Databáza Messages v zhlukovanom rozložení	51
40	Nastavenia analytického režimu	52
41	Zapnutie a zvolenie kľúčov na filtrovanie	52
42	Jeden filtrovaný atribút objektu Movie	53
43	Dva filtrované kľúče objektu Movie	54
44	Tri filtrované kľúče objektu Movie	55
45	Databáza movies v sub-zhlukovom rozložení	56
46	Databáza messages v sub-zhlukovom rozložení	57

Zoznam tabuliek

1	<i>Štruktúra vrcholu "User"</i>	2
2	<i>Štruktúra hrany "WATCHES"</i>	2
3	Porovnanie vybraných nástrojov na vizualizáciu grafových dát.	11
4	<i>Porovnanie vybraných vlastností medzi prostredím Neo4j a navrhnutým nástrojom.</i>	58

Zoznam zdrojových kódov

1	Vytvorenie 5 vrcholov a 2 hrán v jazyku Gremlin	3
2	Dopytovanie v jazyku Gremlin	3
3	Dopytovanie v jazyku openCypher	3
4	Dopytovanie v jazyku GSQL	5
5	Implementácia OpenAI požiadavky	27
6	Implementácia sťahovania snímkov obrazovky	28
7	Pohyb kamery pomocou WASD (pseudokód)	29
8	Priblíženie pomocou OrbitControls (pseudokód)	29
9	Inicializácia a aktualizácia priblíženia (úryvok z implementácie)	30
10	Výber uzla a prepnutie detailu (úryvok z implementácie)	30
11	Kolízna korekcia medzi dvomi uzlami (pseudokód)	31
12	Zhlukovanie vrcholov podľa typu (skrátená ukážka)	32
13	Kliknutie v zhlukovanom zobrazení (skrátene)	33
14	Vstup filtrov a priradenie osí X/Y/Z	34
15	Sub-zhlukované rozloženie pomocou BFS (pseudokód)	35
16	Sub-zhlukované rozloženie pomocou BFS (úryvok z implementácie)	35
17	Dopyt na vytvorenie databázy Messages columns	44
18	Dopyt na vytvorenie databázy Shop	45

Obsah

Úvod	1
1 Grafové databázy a ich vizualizácia	2
1.1 Implementácie grafových databáz	3
1.1.1 Amazon Neptune	3
1.1.2 TigerGraph	4
1.2 Neo4j	5
1.3 Vizualizácie grafových databáz	8
1.3.1 yWorks	9
1.3.2 Gephi	10
1.4 Porovnanie a vyhodnotenie vizualizačných nástrojov	11
2 Návrh a implementácia nástroja na trojdimenzionálnu vizualizáciu grafových databáz	13
2.1 Návrh	13
2.1.1 Topologické trojdimenzionálne rozloženie	19
2.1.2 Zhlukované trojdimenzionálne rozloženie	21
2.1.3 Filtrované trojdimenzionálne rozloženie	23
2.1.4 Sub-zhlukované trojdimenzionálne rozloženie	25
2.2 Implementácia	26
2.2.1 Metódy pohybu, priblíženia a detailného zobrazenia	28
2.2.2 Topologické trojdimenzionálne rozloženie	31
2.2.3 Zhlukované trojdimenzionálne rozloženie	32
2.2.4 Filtrované trojdimenzionálne rozloženie	33
2.2.5 Sub-zhlukované trojdimenzionálne rozloženie	34
3 Overenie nástroja na trojdimenzionálnu vizualizáciu grafových databáz	38
3.1 Použité databázy	43
3.2 Topologické trojdimenzionálne rozloženie	45
3.3 Zhlukované trojdimenzionálne rozloženie	46
3.4 Filtrované trojdimenzionálne rozloženie	52
3.4.1 Filtrovanie s jedným kľúčom	53
3.4.2 Filtrovanie s dvoma kľúčmi	54
3.4.3 Filtrovanie s tromi kľúčmi	55
3.5 Sub-zhlukované trojdimenzionálne rozloženie	56
3.6 Porovnanie implementovaného nástroja	57
Záver	59
Literatúra	60

Úvod

Rastúci objem dát a najmä zvyšujúca sa komplexnosť vzťahov medzi nimi spôsobujú, že klasické relačné ukladanie a dopytovanie už často nestačí na efektívnu analýzu. V mnohých doménach – od sociálnych sietí, cez biologické a medicínske dáta, až po dopravné siete či informačné systémy – majú dáta prirodzene „vzťahový“ charakter. Práve v takýchto prípadoch sa grafové databázy stávajú vhodnou alternatívou k tradičným relačným prístupom, keďže umožňujú reprezentovať entity a ich väzby priamo a pracovať s nimi vo forme grafových štruktúr.

Aj keď grafové databázy uľahčujú modelovanie a spracovanie vzťahových dát, prinášajú zároveň výzvy v oblasti interpretácie výsledkov a porozumenia štruktúre dátového modelu. Pri väčších grafoch je typickým problémom vizuálny „šum“ – najmä vysoká miera prekrytia prvkov grafu a časté pretínanie hrán v dvojrozmerných (ďalej len: *2D*) zobrazeniach. Toto znižuje čitateľnosť vizualizácie, zhoršuje orientáciu používateľa a komplikuje úlohy ako hľadanie súvislostí, identifikácia vrcholov, porovnávanie podštruktúr alebo kontrola konzistencie dát.

V praxi je navyše prístup k nástrojom, ktoré by poskytovali pokročilé zobrazovacie metódy, škálovanie na veľké grafy a prirodzenú interakciu, často limitovaný. Bežne dostupné riešenia síce umožňujú základné prehliadanie grafu, no pri rozsiahlejších dátach narážajú na limity čitateľnosti a flexibility. Táto práca preto reaguje na potrebu účinnejšej vizualizácie grafových databáz tým, že sa zameriava na využitie trojrozmerného priestoru, ktorý môže pomôcť redukovať niektoré nedostatky dvojrozmernej vizualizácie (napr. pretínanie hrán a prekrytie), a zároveň otvára priestor pre nové formy interakcie a viacúrovňové zobrazenie.

Účelom tejto práce je navrhnúť, implementovať a experimentálne overiť nástroj na trojdimenzionálnu vizualizáciu grafov nad dátami uloženými v grafovej databáze. Práca sa zameriava na vizualizáciu výsledkov dopytovania v systéme *Neo4j* a na návrh viacerých režimov zobrazenia, ktoré zlepšujú čitateľnosť a interpretáciu komplexných vzťahov v dátach. Výsledkom je aplikácia umožňujúca pripojenie k databáze, vykonávanie *Cypher* dopytov a interaktívnu analýzu grafu v 3D priestore vrátane zhlukovania, filtrovania podľa kľúčov a detailného zobrazenia vybraných entít.

Kapitola 1 prezentuje grafové databázy a ich vizualizáciu. V kapitole 2 je uvedený návrh a implementácia aplikácie na zobrazovanie grafových databáz. Napokon v kapitole 3 experimentálne túto aplikáciu overujeme

1 Grafové databázy a ich vizualizácia

Grafové databázy predstavujú pokročilý typ nerelačných databáz, ktorých cieľom je čo najefektívnejšie ukladať, vizualizovať a modelovať vzťahy medzi entitami v databáze. Práve preto sa často využívajú pri dátach s prepojeniami, ktoré sú zložité a navzájom bohato previazané [1]. Takéto grafové štruktúry možno matematicky vyjadriť takto:

$$G = (V, E), E \subset V^2 \quad (1)$$

kde V je množina vrcholov, každý vrchol predstavuje entitu v databáze s kľúčmi uloženými v nasledujúcom formáte: *klúč:hodnota*, ktorý je veľmi podobný iným nerelačným databázam. Pričom E reprezentuje hrany medzi vrcholmi. Tieto hrany označujú vzťahy medzi dvoma entitami, uloženými v databáze.

klúč	Hodnota
<i>Id</i>	1
meno	"Adam"
vek	27

Tabuľka 1: Štruktúra vrcholu "User"

V tabuľke 1 vidieť že každý vrchol má svoju štruktúru. Okrem samotných kľúčov má vrchol aj typ (*z angl. label*), ktorý určuje význam objektu a zároveň naznačuje, aký druh údajov sa pri danom objekte očakáva. Z pohľadu zápisu teda vrchol nie je chápaný ako riadok s pevnou schémou, ale ako samostatný objekt s vlastnou množinou polí, pričom rôzne vrcholy môžu mať aj odlišnú vnútornú štruktúru. Objekty toho istého typu nemusia mať stabilnú schému, a preto ich možno ďalej deliť na štrukturálne podtypy podľa toho, akú množinu kľúčov obsahujú.

klúč	Hodnota
<i>Id</i>	0
start	0
end	2
<i>type</i>	"WATCHES"

Tabuľka 2: Štruktúra hrany "WATCHES"

V tabuľke 2 znázorňujeme hranu s jej kľúčmi. Hrany predstavujú vzťahy medzi jednotlivými vrcholmi. Môžeme ich definovať ako prvky množiny E , ktoré prepájajú dvojice objektov v databáze a tým vyjadrujú ich vzájomné väzby. Hrany môžu byť orientované a označené, teda nesú informáciu o smere vzťahu aj o jeho type. To znamená, že vrchol reprezentuje čo je v databáze uložené, zatiaľ čo hrana reprezentuje ako spolu jednotlivé objekty súvisia [2].

1.1 Implementácie grafových databáz

Grafové databázy sú dostupné vo viacerých implementáciách, ktoré sa líšia nielen použitým dátovým modelom, ale aj spôsobom nasadenia, podporovanými dopytovacími jazykmi a ponúkanými nástrojmi na správu a vizualizáciu. V tejto podkapitole preto stručne predstavíme vybrané grafové databázové systémy a ich základné charakteristiky, aby bolo možné porovnať prístupy jednotlivých riešení. Následne sa zameriame na konkrétne platformy (napr. *Amazon Neptune*, *TigerGraph* a *Neo4j*), ktoré reprezentujú typické prístupy k práci s grafovými dátami v praxi.

1.1.1 Amazon Neptune

Takto definovaný model grafovej databázy nám umožňuje pracovať s vrcholmi a hranami prostredníctvom špecializovaných dopytovacích jazykov. Jeden z veľmi často používaných je *Amazon Neptune*, ktorý pri práci s dátami podporuje jazyky *Gremlin* a *openCypher*.

Z hľadiska spôsobu dopytovania je *Gremlin* vhodný na sériu na seba nadväzujúcich krokov. Každý krok spracúva aktuálny medzivýsledok a následne pokračuje k ďalším vrcholom alebo hranám.

```
1 g.addV('N').property('uid','n1').as('n1')
2 .addV('N').property('uid','n2').as('n2')
3 .addV('N').property('uid','n3').as('n3')
4 .addV('N').property('uid','n4').as('n4')
5 .addV('N').property('uid','n5').as('n5')
6 .addE('r1').from('n1').to('n2')
7 .addE('r2').from('n1').to('n3')
8 .iterate()
```

Ukážka 1: Vytvorenie 5 vrcholov a 2 hrán v jazyku *Gremlin*

Na ukážke 1 je zobrazená základná schéma na vytvorenie piatich vrcholov a dvoch hrán. Následne ich v ďalších ukážkach dopytujeme.

```
1 g.V().limit(5)
```

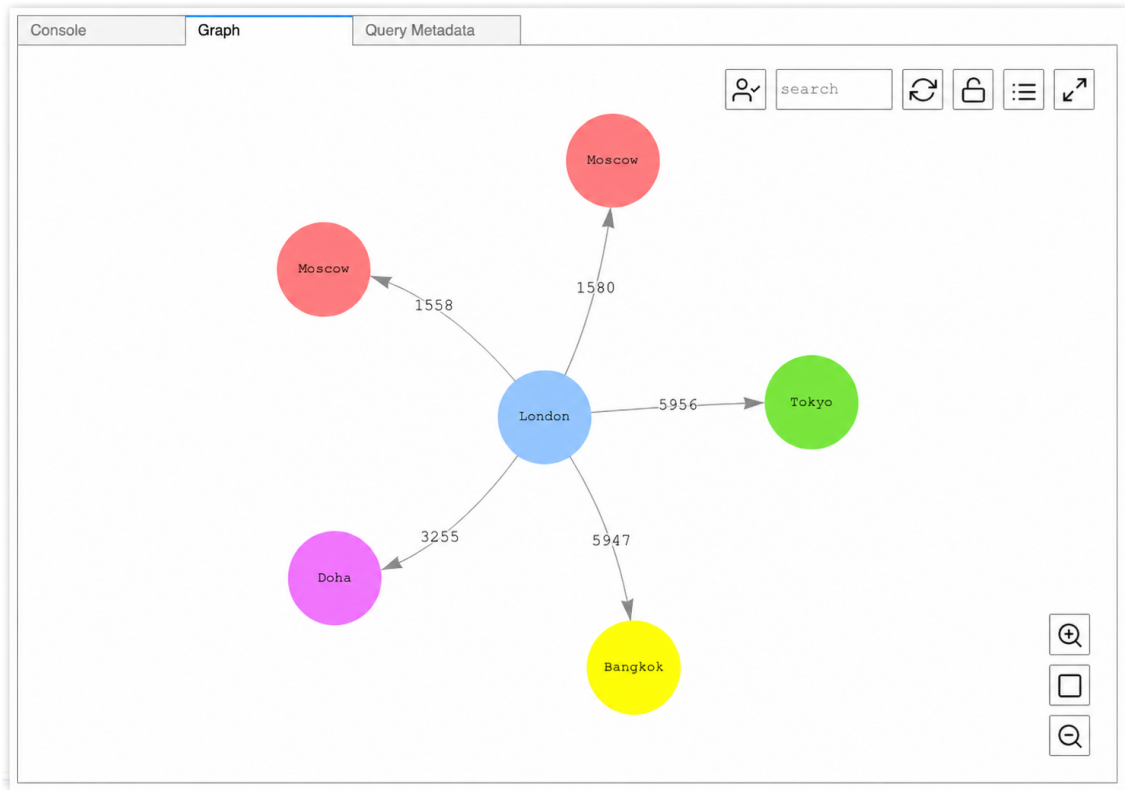
Ukážka 2: Dopytovanie v jazyku *Gremlin*

```
1 MATCH (n)
2 RETURN n
3 LIMIT 5;
```

Ukážka 3: Dopytovanie v jazyku *openCypher*

Tieto dva dopyty predstavujú najjednoduchší spôsob, ako získať obmedzený počet vrcholov z grafu. V prípade jazyka *openCypher* sa deklaratívne určí vzor, ktorý má databáza nájsť, pričom *MATCH(n)* znamená výber ľubovoľného vrcholu a *RETURN n* zabezpečí

jeho zobrazenie vo výsledku. Naopak, v jazyku Gremlin ide o prehľadavací prístup, kde $g.V()$ vytvorí množinu všetkých vrcholov a následný krok $limit(5)$ obmedzí počet vrátených záznamov.



Obrázok 1: Cloudové rozhranie Amazon Neptune

Na obrázku 1 [3] je znázornené rozhranie *Amazon Neptune*. Na rozdiel od niektorých iných grafových databázových systémov, *Amazon Neptune* nie je navrhnutý ako lokálne inštalovateľná desktopová aplikácia. Ide o plne spravovanú cloudovú službu, ktorej databázové clustre sú prevádzkované v prostredí *Amazon VPC*. Hoci *AWS* poskytuje nástroje ako *Neptune Workbench* a *Graph Explorer* na interaktívne dopytovanie a vizualizáciu dát, ich použitie je viazané na cloudové prostredie alebo na doplnkové kontajnerové riešenia. Z tohto dôvodu nebol tento systém zvolený ako primárny nástroj pre jednoduchú lokálnu demonštráciu funkcionality [4].

1.1.2 TigerGraph

TigerGraph predstavuje grafový databázový systém zameraný na prácu s grafom vlastností databázy, v ktorom sú dáta reprezentované prostredníctvom vrcholov, hrán a ich kľúčov. Z praktického hľadiska je jeho dôležitou vlastnosťou využitie natívneho jazyka *GSQL*, ktorý je určený na dopytovanie a analýzu grafových dát. Na rozdiel od niektorých cloudovo orientovaných riešení ponúka *TigerGraph* aj vlastné používateľské rozhranie nástroja *GraphStudio*, ktorý je možné spustiť v internetovom prehliadači. Vďaka čomu

je vhodný aj na lokálne alebo self-managed nasadenie. Tento prístup umožňuje nielen definovanie schémy grafu a prácu s vrcholmi a hranami, ale aj jednoduchšie testovanie a demonštráciu správania databázy v praxi.

Z pohľadu dopytovania sa v systéme *TigerGraph* používa najmä jazyk *GSQL*, ktorý kombinuje prvky deklaratívneho zápisu. V jednoduchej ukážke možno vybrať obmedzený počet vrcholov konkrétneho typu a následne ich vypísať ako výsledok dopytu. Nasledujúci príklad zobrazuje základný *GSQL* dopyt, ktorý vyberie prvých päť vrcholov typu *Person* a vypíše ich do výstupu:

```
1 INTERPRET QUERY () FOR GRAPH graph_name SYNTAX v2 {  
2     results = SELECT p  
3         FROM Person:p  
4         LIMIT 5;  
5     PRINT results;  
6 }
```

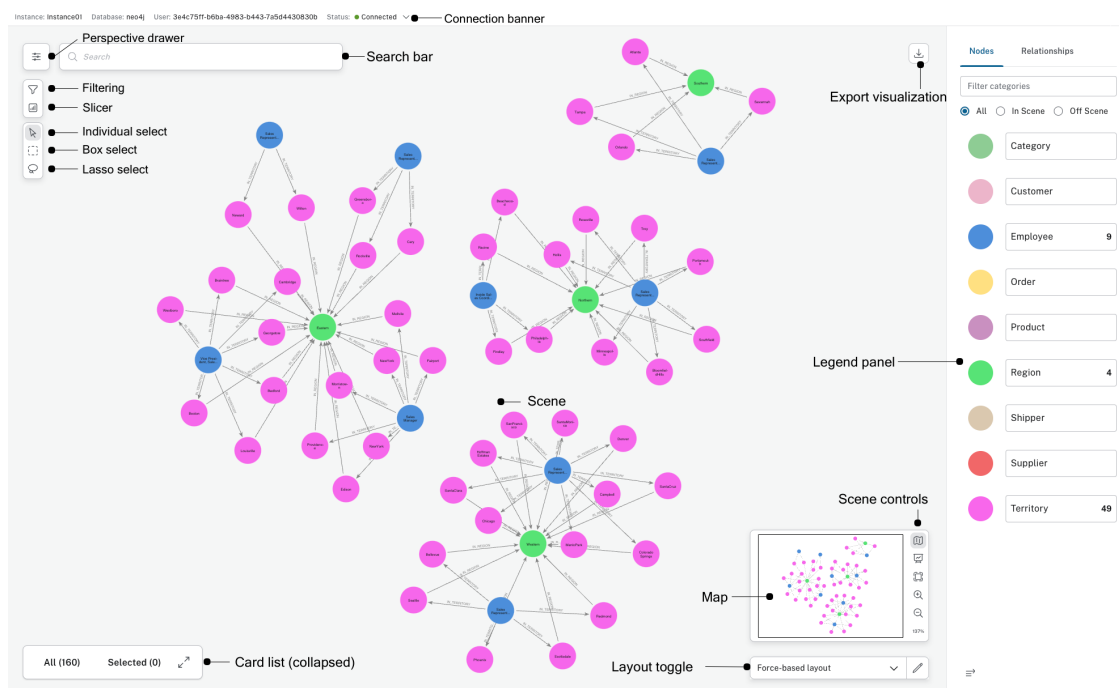
Ukážka 4: Dopytovanie v jazyku GSQL

Uvedený dopyt je možné chápať ako jednoduchý ekvivalent základného výberu vrcholov, pričom v prostredí *TigerGraph* je potrebné explicitne uviesť typ vrcholu, z ktorého sa výber realizuje. Príkaz *SELECT* vytvorí množinu výsledných vrcholov, *LIMIT* obmedzí ich počet a príkaz *PRINT* zabezpečí ich zobrazenie. Aj na takomto jednoduchom príklade je viditeľné, že *GSQL* je navrhnutý tak, aby podporoval nielen samotné dopytovanie, ale aj ďalšie analytické operácie nad grafom [5].

1.2 Neo4j

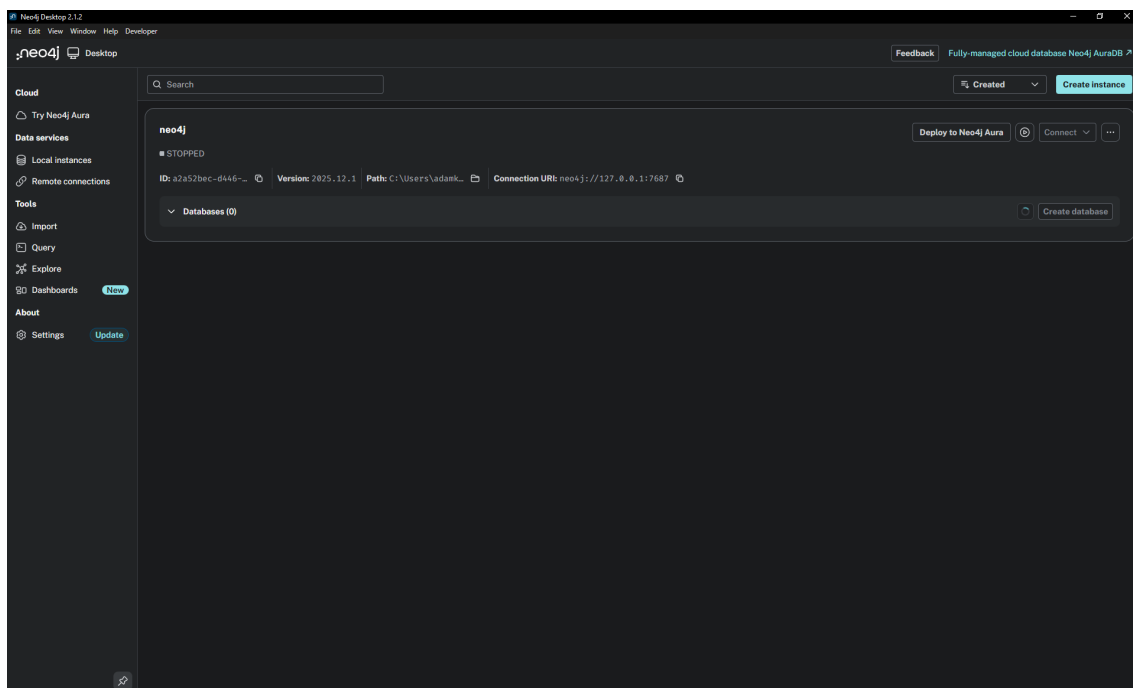
Neo4j predstavuje natívnu grafovú databázu. Na rozdiel od systémov, ktoré iba vytvárajú grafovú abstrakciu nad iným typom úložiska, *Neo4j* pracuje s dátami priamo vo forme grafu, čo zodpovedá prirodzenému modelovaniu entít a vzťahov v doméne.

Na dopytovanie dát v *Neo4j* sa používa jazyk *openCypher*, ktorý je deklaratívnym jazykom určeným na vytváranie, vyhľadávanie a úpravu grafových štruktúr. Praktická výhoda *Neo4j* je viacero spôsobov nasadenia a implementácie. Je možné použiť docker, lokálnu aplikáciu na manažovanie databázy alebo cloudové riešenia pre väčšie projekty. Práve pre toto je *Neo4j* vhodný ako systém nie len pre produkčné použitie ale aj pre vývoj a overenie implementácií [6].



Obrázok 2: Cloudové rozhranie Neo4j AuraDB

V prípade cloudového nasadenia sa pri *Neo4j* najčastejšie využíva služba *Neo4j AuraDB*. Databázu je takto možné vytvoriť priamo vo webovom rozhraní bez potreby lokálnej inštancie databázového systému. Na obrázku 2 [7] je zobrazené rozhranie *AuraDB*, výstupy z tejto databázy je možné exportovať priamo ako .png súbory. Z používateľského hľadiska je dôležité, že cloudové riešenie *Neo4j* neponúka iba samotnú databázu, ale aj integrované nástroje priamo v *Aura Console* [8].



Obrázok 3: Neo4j Desktop

Obrázok 3 zobrazuje klientsku aplikáciu *Neo4j Desktop* určenú na lokálnu prácu s databázovým systémom. Pričom jednou zo silných stránok aplikácie je, že v nej je možné spravovať ľubovoľný počet databáz súčasne.

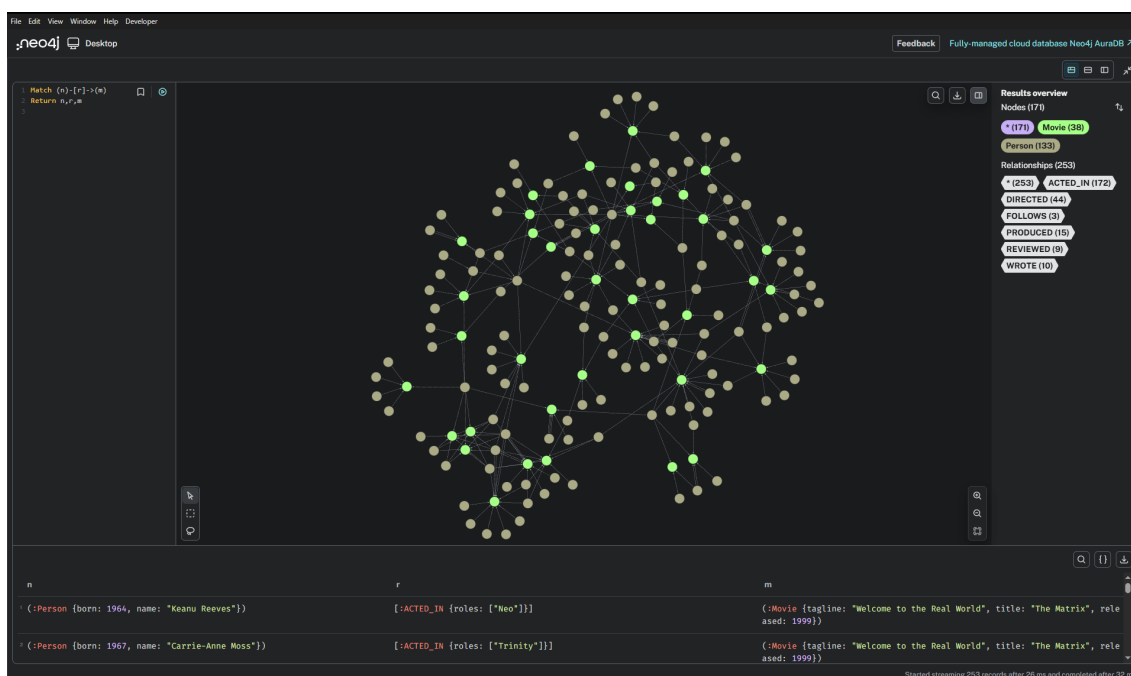
Súčasťou prostredia sú integrované nástroje:

- Import: importovanie dát.
- Query: dopytovanie v rámci aktívnej databázy.
- Explore: vizualizáciu grafu.
- Dashboards: jednoduchú prezentáciu výsledkov.

Neo4j Desktop je ako nástroj vhodný pre lokálne experimentálne použitie, kde uľahčuje správu databázy a robí prácu s *Neo4j* prístupnejšou než čistý príkazový riadok. Je vhodný najmä na lokálne použitie, zatiaľ čo cloudové alebo produkčné nasadenie by zbytočne zvyšovalo nároky na infraštruktúru a konfiguráciu [9].

1.3 Vizualizácie grafových databáz

Vizualizácia v grafových databázach predstavuje dôležitú súčasť práce s dátami, pretože výsledok dopytu nie je v mnohých prípadoch iba množina izolovaných záznamov, ale prepojená štruktúra uzlov a vzťahov. Práve grafické zobrazenie tak umožňuje používateľovi lepšie vnímať súvislosti medzi entitami, identifikovať centrálné objekty, sledovať smery väzieb a rýchlejšie pochopiť celkový kontext dát. V porovnaní s tabuľkovým výstupom je vizualizácia vhodná najmä pri úlohách, kde je dôležité sledovať susedstvá vrcholov, viacnásobné prepojenia alebo menšie podgrafy vytvorené výsledkom dopytu. Z hľadiska praktického využitia preto vizualizácia nepredstavuje iba doplnok k dopytovaniu, ale samostatný analytický nástroj, ktorý pomáha pri interpretácii výsledkov a pri objavovaní vzorov v dátach [10].

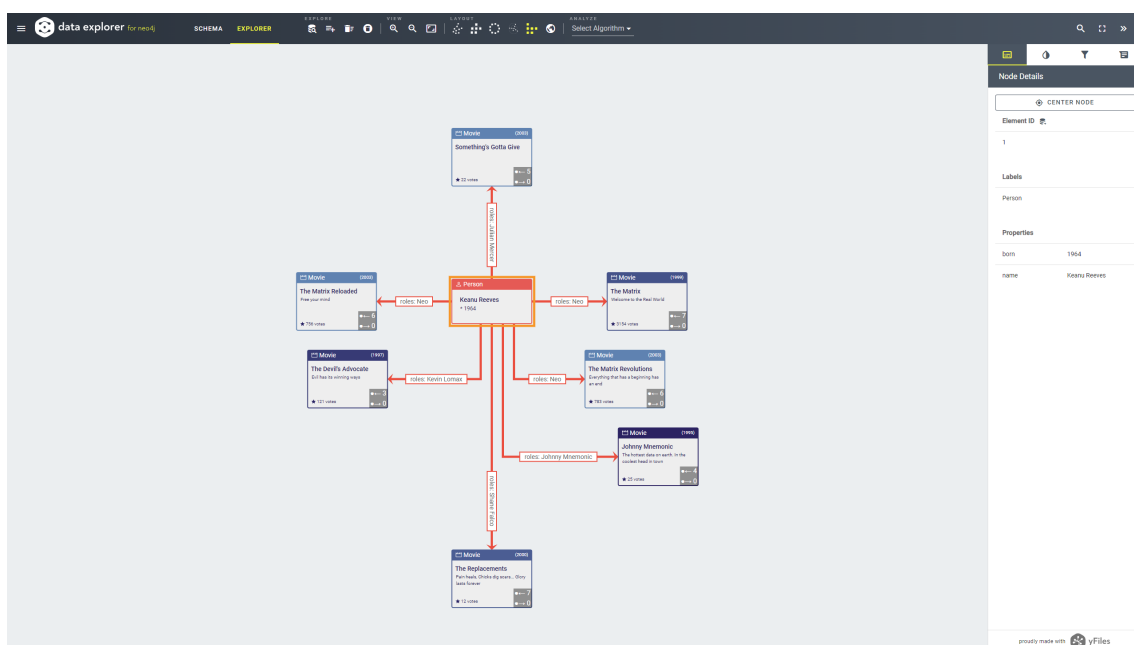


Obrázok 4: Neo4j Desktop Dopytovanie

Na obrázku 4 znázorňujeme databázu *Movies*, ktorý zobrazuje vzťahy medzi filmami, hercami, režisérmi a ďalšími entitami. Po vykonaní dopytu sa vo vizuálnom zobrazení objavia len tie uzly a hrany, ktoré zodpovedajú zadanému vzoru, teda nejde o zobrazenie celej databázy, ale o konkrétny výsledný podgraf. Dopyt ktorý bol použitý v obrázku 4 ukazuje celá databáza so všetkými vzťahmi.

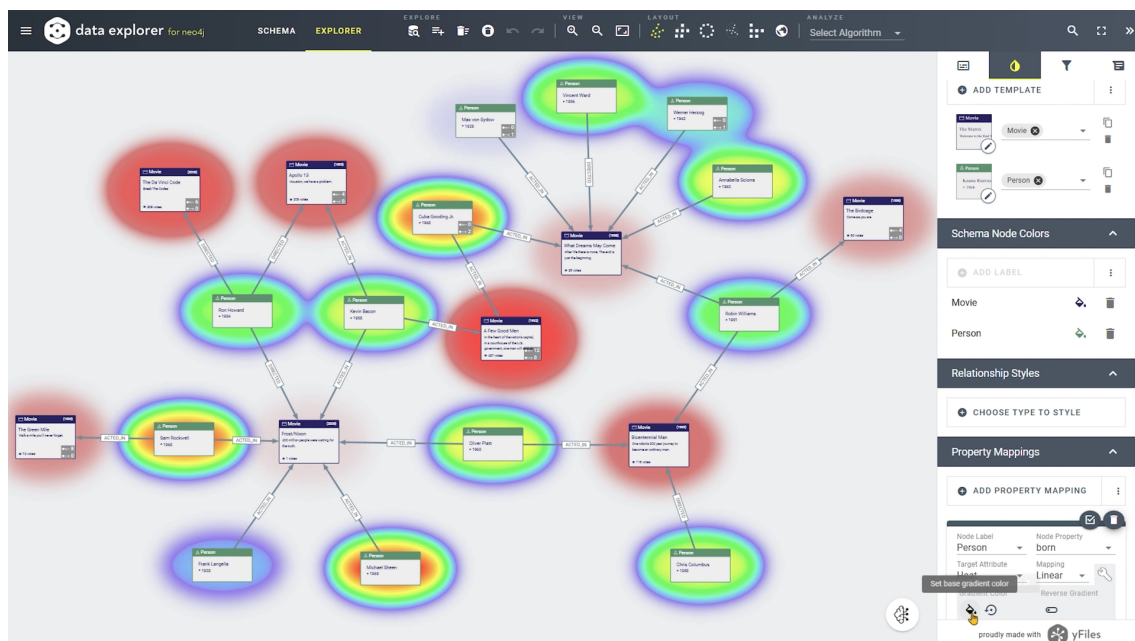
1.3.1 yWorks

Okrem prostredia *Neo4j Desktop* však existujú aj ďalšie nástroje, ktoré sú vhodné na vizualizáciu grafových databáz a ktoré možno využiť buď ako alternatívu, alebo ako doplnok pri špecializovanejších úlohách. *yWorks Data Explorer for Neo4j*, je navrhnutý priamo na vizuálne prehliadanie *Neo4j* databáz. Ide o bezplatný nástroj pre prehliadač, ktorý umožňuje interaktívne skúmanie databázy bez potreby písania *Cypher* dopytov, hoci ich podporuje, ak ich používateľ chce využiť. Nástroj poskytuje automatické rozloženia grafu vlastností, grafové analytické algoritmy a možnosť prispôbiť vizualizáciu uzlov podľa typov a vlastností. Z praktického hľadiska je zaujímavé aj to, že sa vie pripojiť k lokálnej aj vzdialenej *Neo4j* databáze a funguje priamo v prehliadači bez inštalácie. Takýto nástroj je vhodný najmä v situáciách, keď je cieľom rýchla a prehľadná vizualizácia dát bez potreby detailnejšieho programátorského zásahu do rozhrania [11].



Obrázok 5: Data explorer yWorks

Štandardné zobrazenie rozkladá vrcholy do štvorcov a prepája ich pomocou vzťahov, na obrázku 5 [12] je použitá databáza *Movies*.

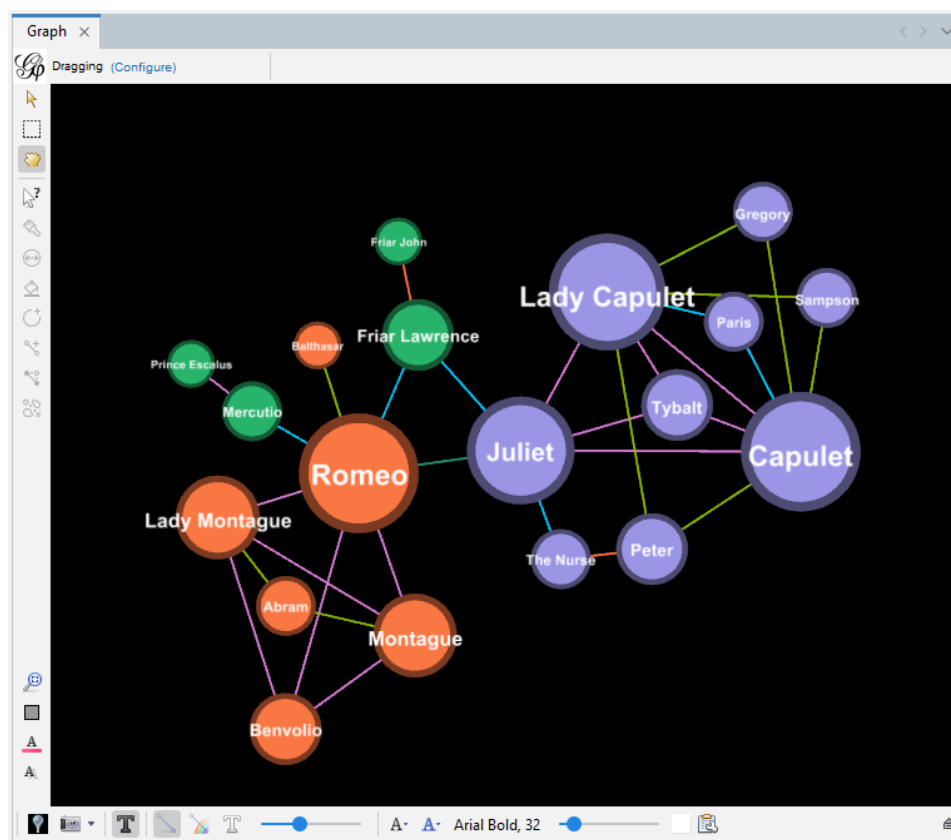


Obrázok 6: HeatMap yWorks

Ako príklad, jedným z nástrojov dostupných v prostredí *yWorks* uvádzame grafové tepelné mapy. Ako je znázornené na Obrázku 6 [13], tepelná mapa vo vizualizačných nástrojoch *yWorks* slúži na zvýraznenie vybraných častí grafu. Farebná intenzita je viazaná na hodnotu vypočítanú z dát uzlov a hrán: napríklad najintenzívnejšie toky alebo najviac zaťažené väzby. Takéto zobrazenie je vhodné najmä preto, že používateľovi umožňuje rýchlo identifikovať významné alebo preťažené časti grafu bez nutnosti detailne analyzovať každý prvok samostatne.

1.3.2 Gephi

Gephi, je open-source platformou na vizualizáciu a prieskum grafov a sietí. Jeho výhodou je, že umožňuje pracovať s exportovanými grafmi vo formátoch, ktoré sú vhodné na ďalšiu analýzu a prezentáciu. Dokumentácia uvádza, že na vytvorenie siete v *Gephi* postačuje aj jednoduchý *CSV* súbor so stĺpcami *Source* a *Target*, čo znamená, že dáta z grafovej databázy možno po exporte relatívne jednoducho preniesť do samostatného vizualizačného prostredia. Samotné rozhranie *Gephi* je postavené okolo centrálného okna *Graph*, ktoré slúži ako hlavný priestor na zobrazovanie siete, jej približovanie, posúvanie a manuálnu manipuláciu s uzlami. *Gephi* je preto vhodný najmä vtedy, keď je cieľom vytvoriť prehľadnejší statický alebo analyticky upravený obrázok grafu, napríklad pre publikačné alebo prezentačné účely [14].



Obrázok 7: Gephi

Na obrázku 7 [15] znázorňujeme lokálnu inštanciu *Gephi* aplikácie. Napriek svojej jednoduchosti je to veľmi efektívny a jednoducho použiteľný nástroj na dvojdimenzionálnu vizualizáciu grafových databáz.

1.4 Porovnanie a vyhodnotenie vizualizačných nástrojov

Predstavili sme viacero vizualizačných nástrojov na vizualizáciu grafových databáz. Jednotlivé riešenia sa líšia cieľovou skupinou, mierou integrácie s databázou, spôsobom práce s dopytmi a rozsahom vizualizačných možností.

Nástroj	Neo4j pripojenie	Dopytovanie v Cypher	Zhlukovanie	Analytické prvky	Export vizualizácie
<i>Neo4j Desktop</i>	áno (pripojenie)	áno	základné	základné	PNG
<i>Neo4j Aura</i>	áno (cloud)	áno	základné	základné	PNG
<i>yWorks</i>	áno (pripojenie)	nie	áno	obmedzené	PNG
<i>Gephi</i>	nie	nie	áno	pokročilé	PNG/SVG/PDF

Tabuľka 3: Porovnanie vybraných nástrojov na vizualizáciu grafových dát.

Zo zistení z tabuľky 3 vyplýva, že *Neo4j Desktop* je pre účely tejto práce najvhodnejší najmä vďaka priamej integrácii s databázou *Neo4j*, podpore dopytovania v jazyku *Cypher* a dostupnosti vizualizačných nástrojov v jednom lokálnom prostredí. V praxi to umožňuje

jednoduchý a reprodukovateľný pracovný postup: používateľ môže databázu spustiť lokálne, vykonať dopyt a okamžite vizualizovať výsledný podgraf bez potreby exportu dát do externých formátov. To je dôležité najmä pri experimentovaní, ladení dopytov a pri demonštrácii výsledkov v rámci diplomovej práce.

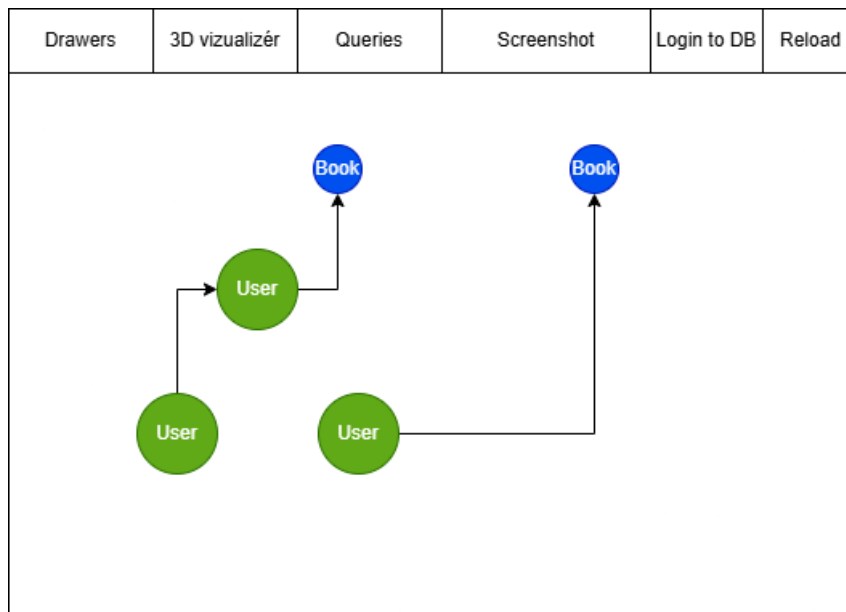
2 Návrh a implementácia nástroja na trojdimenzionálnu vizualizáciu grafových databáz

V dvojrozmernom priestore pri vizualizácii grafov nastáva problém prekrývania medzi uzlami a hranami, ktorý pri rastúcej veľkosti a hustote grafu vedie k vizuálnemu preťaženiu a zníženej čitateľnosti výsledkov. Keď sa na jednu rovinu premietne veľké množstvo objektov, zvyšuje sa počet kolízií uzlov, krížení hrán a vzájomné zakrývanie prvkov. Trojdimenzionálne (ďalej 3D) zobrazenie preto predstavuje prirodzený spôsob, ako časť tejto komplexity rozložiť do ďalšieho stupňa voľnosti: uzly a vzťahy je možné priestorovo separovať, zoskupovať alebo vrstviť tak, aby sa znížil vizuálny šum, pričom interaktívna manipulácia kamerou (rotácia, priblíženie, zmena perspektívy) umožňuje odhaľovať štruktúru grafu z viacerých pohľadov. Súčasne však 3D vizualizácia prináša nové výzvy:

- prvky sa môžu skrývať za inými (oklúzia),
- vyššie nároky na orientáciu v priestore,
- potreba vhodných interakčných techník,
- potreba hĺbkových vodidiel, aby 3D zobrazenie podporilo porozumenie dátam a nezvyšovalo kognitívnu záťaž používateľa [16].

2.1 Návrh

V tejto časti práce sa zameriame na používateľské rozhranie a ovládanie aplikácie *Graph Database Viewer*. Cieľom je stručne predstaviť hlavné prvky rozhrania, ktoré používateľ využíva pri bežnej práci: pripojenie k databáze, zadávanie dopytov, prepínanie režimov vizualizácie a manipuláciu so zobrazeným grafom. Popis je organizovaný podľa jednotlivých častí rozhrania tak, aby bolo zrejmé, kde sa konkrétne funkcionality nachádzajú a aký je ich účel pri práci s 3D vizualizáciou.



Obrázok 8: Prvotný návrh aplikácie

Na obrázku 8 sa nachádza prvotný návrh aplikácie, následne preberieme jednotlivé funkcie:

- **Horná navigačná lišta.** Obsahuje základné akcie aplikácie a prepínače režimov; v návrhu sú zobrazené položky *Bočný navigačný panel: Drawers, Názov aplikácie: 3D Vizualizér, Dopytovanie databázy: Queries, Vytvorenie obrázku zo scény: Screenshot, Pripojenie na lokálnu/vzdialenú databázu: Login to DB a Tlačítko na opätovné načítanie databázy: Reload.*
- **Drawers (bočné panely).** Tlačidlo slúži na otvorenie bočného panelu, v ktorom budú používateľovi ponúknuté nastavenia pre 3D vizualizáciu. V našom riešení sem patria najmä *Analytický režim* (voľby pre analýzu grafu) a *3D nastavenia* (nastavenia algoritmov rozloženia a parametrov vizualizácie).
- **Queries.** Pre účely vkladania dát a ich vizualizáciu je potrebné vytvoriť dopytované okno. Pre naše účely má dopytované okno 3 rôzne režimy: Štandardný, No-code a GPT builder.
- **Práca s dátami a aktualizácia.** *Login to DB* slúži na pripojenie k databáze (autentifikácia a získanie dát), zatiaľ čo *Reload* zabezpečuje opätovné načítanie dát a prekreslenie scény po zmene nastavení, takže nanovo urobíme dopyt na návrat celej databázy.
- **Vizualizačné plátno (3D scéna).** Hlavný priestor, kde sa zobrazuje graf. Používateľ tu sleduje uzly a orientované hrany, pričom zobrazenie predstavuje výsledný podgraf získaný z databázy.

- **Vizuálne odlíšenie prvkov.** Typ vrchola je odlíšený farbou a hĺbka je vyjadrená škálovaním veľkosti vrchola (perspektívny efekt). Smer hrán je jednoznačne vyznačený šípkami.

Drawers	3D vizualizér	Queries	Screenshot	Login to DB	Reload
---------	---------------	---------	------------	-------------	--------

Queries

Query Builder

LLM Queries

MATCH(N) RETURN N;

bad query

Text_Input

send

Obrázok 9: Návrh Štandardného dopytovania v aplikácii

Po zvolení položky *Queries* sa v aplikácii otvorí modálne okno určené na prácu s databázovými dopytmi. V hornej časti tohto okna sa nachádza prepínanie medzi tromi režimami: *Queries* pre priame zadanie dopytu, *Query Builder* pre zostavenie dopytu pomocou formulárového rozhrania a *LLM Queries* pre generovanie dopytov na základe textového zadania.

V režime *Queries* je zobrazený panel s aktuálnym dopytom (zelené pole) a pod ním stavový panel určený na spätnú väzbu, napríklad chybové hlásenia pri neplatnom dopyte (červené pole). V spodnej časti tohto okna sa nachádza textové vstupné pole spolu s tlačidlom *send*, ktoré slúži na odoslanie zadaného dopytu a jeho následné spracovanie systémom. Po spracovaní dopytu sa okno automaticky uzavrie a vykoná sa daná operácia.

Drawers	3D vizualizér	Queries	Screenshot	Login to DB	Reload
---------	---------------	---------	------------	-------------	--------

Queries	Query Builder	LLM Queries
---------	---------------	-------------

Operation: Display

Object: Movie

Filter by value: released

Field value: 1999

Create query

MATCH(N:Movie) WHERE N.released=1999 RETURN N;

Obrázok 10: Návrh Formulárového vytvárania dopytu

Režim *Query Builder* predstavuje formulárový spôsob vytvárania dopytov bez nutnosti ručného zapisovania celého dopytu. Používateľ v tomto okne postupne vyberá jednotlivé parametre dopytu pomocou výberových polí: najprv zvolí typ operácie (napr. *Display* ako ekvivalent výpisu dát, prípadne *Insert* na vkladanie dát, *Update* na úpravu dát pomocou podmienky alebo *Delete*, ktorý vymaže objekty na základe podmienenej operácie), následne vyberie cieľový objekt/typ uzla (v ukážke *Movie*) a doplní podmienky filtrovania cez zvolené pole a jeho hodnotu (napr. kľúč *released* a hodnota *1999*). Po stlačení tlačidla *Create query* aplikácia na základe týchto volieb automaticky vygeneruje výsledný dopyt, ktorý sa zobrazí v spodnej časti okna. Toto znižuje riziko syntaktických chýb, urýchľuje tvorbu jednoduchých dopytov a zároveň umožňuje používateľovi pochopiť, ako sa zvolená kombinácia parametrov premieta do výslednej textovej podoby dopytu.

Drawers	3D vizualizér	Queries	Screenshot	Login to DB	Reload
---------	---------------	---------	------------	-------------	--------

Queries	Query Builder	LLM Queries
---------	---------------	-------------

OpenAI API key:

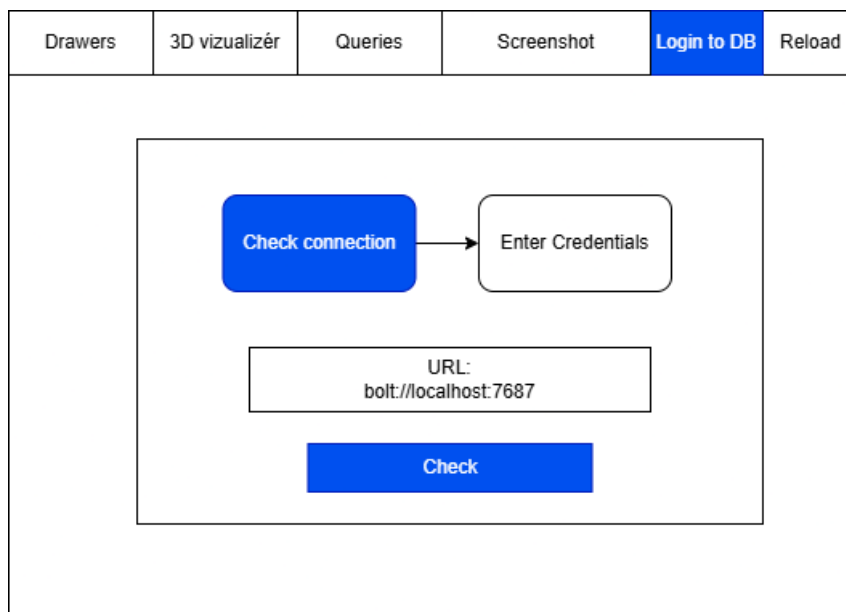
What do you want to do?
"Create a match query that will return all movies that were released before the year 2000.

Generate Cypher Query

MATCH(m:Movie) WHERE m.released < 2000 RETURN m;

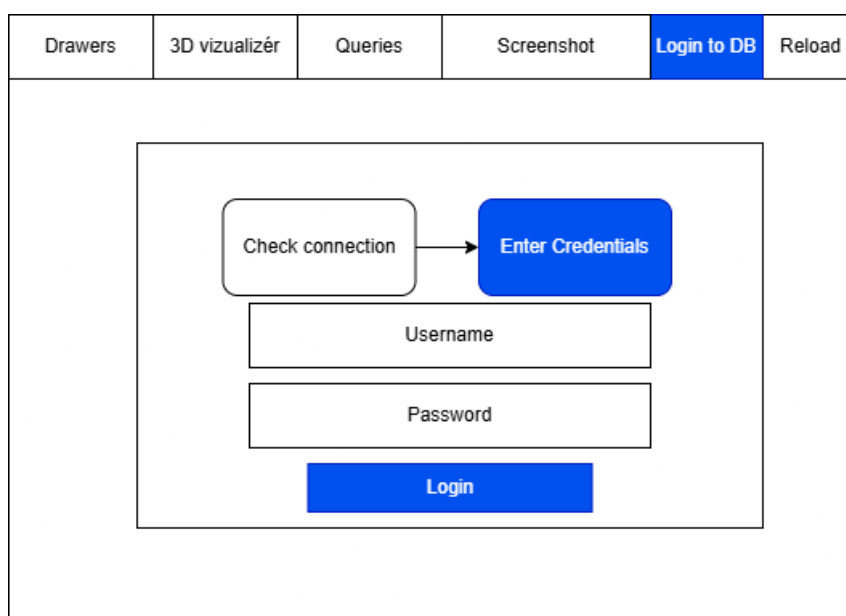
Obrázok 11: Návrh pre spracovanie prirodzeného jazyka do dopytov

Režim *LLM Queries* slúži na generovanie databázových dopytov na základe textového zadania dopytu v prirodzenom jazyku. Používateľ najprv vloží svoj *OpenAI API key*, ktorý aplikácia využije na autorizáciu volania externého modelu. Následne do vstupného poľa zadá opis toho, čo chce z databázy získať alebo vykonať (napr. výber filmov podľa roku vydania). Po stlačení tlačidla *Generate Cypher Query* systém spracuje textovú požiadavku a v spodnej časti okna zobrazí vygenerovaný dopyt v jazyku *Cypher*. Takto vytvorený dopyt môže používateľ následne skopírovať alebo ďalej upraviť a použiť ho ako vstup pre vykonanie v aplikácii, pričom hlavnou výhodou tohto režimu je zjednotenie tvorby dopytov pre používateľov, ktorí nepoznajú detailnú syntax jazyka *Cypher*.



Obrázok 12: Návrh pre prihlásenie sa do databázy

Pri pripájaní k databáze aplikácia používa dvojkrokový proces. V prvom kroku (*Check connection*) používateľ zadá URL databázy (napr. `bolt://localhost:7687`) a tlačidlom *Check* overí, či je databázový server dostupný a či je možné nadviazať spojenie.



Obrázok 13: Návrh pre prihlásenie sa do databázy

Po úspešnom overení sa sprístupní druhý krok (*Enter Credentials*), v ktorom používateľ zadá používateľské meno a heslo a zvolí *Login*. Po prihlásení aplikácia načíta potrebné dáta z databázy (napr. uzly, hrany a ich kľúče) tak, aby mohli byť ďalej použité v dopytovaní aj v 3D vizualizéri. V ďalšej sekcii sa budeme zamieriavať na spôsoby vizualizácie ktoré budú v aplikácii dostupné.

2.1.1 Topologické trojdimenzionálne rozloženie

Základným (predvoleným) spôsobom rozloženia grafu v aplikácii je topologické trojdimenzionálne rozloženie, ktorého cieľom je poskytnúť používateľovi rýchly a prehľadný „prvý pohľad“ na štruktúru výsledného podgrafu. V praxi to znamená, že uzly sa rozmiestňujú tak, aby boli v scéne čitateľné, minimálne sa prekrývali a aby bolo možné jednoducho sledovať vzťahy medzi entitami. Rozloženie vychádza z princípu založeného na fyzikálnom modelovaní síl pôsobiacich medzi objektami (z angl. *force-collide*), pričom kľúčovým prvkom pre zachovanie čitateľnosti je kolízna zložka (*force-collide*), ktorá zabezpečuje minimálnu vzdialenosť medzi vrcholmi a tým bráni ich zlievaniu do jedného miesta [17].

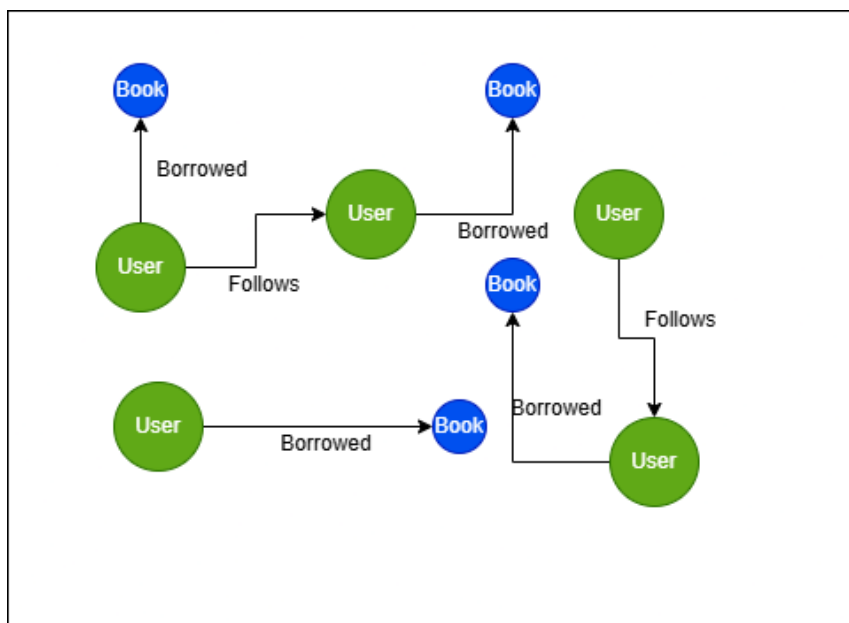
Pri vizualizácii grafu v 2D zobrazení vzniká pri väčšej hustote prirodzený problém: uzly a hrany sa začnú prekrývať a používateľ stráca schopnosť rozlíšiť jednotlivé prvky. V 3D priestore síce získavame dodatočný rozmer, avšak prekrývanie sa stále môže prejaviť v projekcii na obrazovku (najmä pri nevhodnom uhle pohľadu alebo pri zhutnení uzlov v jednej oblasti).

Z tohto dôvodu rozloženie kombinuje viacero jednoduchých mechanizmov: kolíznú silu, ktorá udržiava minimálnu vzdialenosť medzi uzlami, topologickú zložku viazanú na hrany, ktorá udržiava susediace uzly relatívne blízko, a tlmenie pohybu, ktoré stabilizuje simuláciu a zabraňuje neustálemu „kmitaniu“ scény. Výsledkom je rozloženie, ktoré zachováva lokálne susedstvá, no zároveň sa snaží o vizuálnu separáciu uzlov tak, aby boli jednotlivé entity ľahko identifikovateľné [18].

Z pohľadu používateľského zážitku je dôležité, že topologické rozloženie je univerzálne: nevyžaduje predchádzajúce znalosti o dátach ani špeciálne nastavenia a dáva zmysluplný výstup aj pri malých alebo stredne veľkých grafoch. V praxi sa tento režim používa najmä v úvodnej fáze práce s dátami, keď používateľ potrebuje rýchlo pochopiť, aké typy entít sa v grafe vyskytujú, ako sú medzi sebou prepojené a či dopyt vrátil očakávané vzťahy. Toto ilustruje typický výstup tohto režimu: uzly rôznych typov sú odlíšené farbou a smer vzťahov je vyjadrený orientáciou hrán.

Kolízna zložka (*force-collide*) je v tomto kontexte kľúčová najmä preto, že rieši najviditeľnejší problém vizualizácie: vzájomné „narážanie“ uzlov a ich prekrývanie. V princípe sa pre každý uzol uvažuje jeho efektívny polomer (resp. polomer s prídavným odstupom) a simulácia sa snaží zabezpečiť, aby vzdialenosť medzi dvojicou uzlov neklesla pod súčet ich polomerov. Ak k tomu dôjde, uzly sú v simulácii jemne odtlačené od seba v smere spojnice ich pozícií. Vďaka tomu sú uzly v scéne vizuálne oddelené, čo priamo zlepšuje čitateľnosť ich označení a znižuje riziko, že používateľ bude mylne interpretovať dve odlišné entity ako jeden prvok.

Keďže rozloženie prebieha v trojdimenzionálnom priestore, každému uzlu je priradená pozícia vo forme vektora (x, y, z) . Pri renderovaní sa následne používa perspektívna



Obrázok 14: Topologické rozloženie v 3D priestore s kolíznou zložkou

projekcia, ktorá prirodzene vyjadruje hĺbku scény: objekty ďalej od kamery pôsobia menšie a bližšie objekty väčšie (viď obrázok 14). Tento efekt je v návrhu využitý zámerne, pretože pomáha používateľovi intuitívne vnímať priestorové rozloženie bez potreby explicitných hĺbkových osí. Praktickým dôsledkom je, že aj pri relatívne hustej časti grafu môže používateľ zmenou uhla pohľadu alebo priblížením odhaliť prepojenia, ktoré by v čisto 2D zobrazení ostali prekryté.



Obrázok 15: Detail vybraného uzla s vypísaním jeho vlastností a súhrnných údajov.

Z implementačného hľadiska možno topologické rozloženie chápať ako iteratívnu simuláciu. V každom kroku sa z aktuálnych pozícií vypočítajú korekcie (sily) a tie sa ná-

sledne premietnu do aktualizácie pozícií uzlov. Simulácia sa vykonáva buď po stanovený počet iterácií, alebo do momentu, keď sa zmeny pozícií stanú zanedbateľné. V praxi je dôležité udržať kompromis medzi kvalitou rozloženia a časom výpočtu: príliš málo iterácií vedie k nestabilnému alebo „zhluknutému“ grafu, zatiaľ čo príliš veľa iterácií zvyšuje latenciu pri prekreslení (napr. po vykonaní nového dopytu).

Topologické rozloženie má aj svoje prirodzené limity. Pri veľmi hustých grafoch môžu hrany vytvoriť vizuálne „zväzky“, ktoré znižujú čitateľnosť, aj keď sú uzly od seba dostatočne oddelené. V takých prípadoch je vhodné prejsť na zhlukované alebo filtrované rozloženie. Napriek tomu je topologický režim dobrým východiskom: poskytuje okamžitú vizuálnu spätnú väzbu, podporuje interaktívny prieskum dát a slúži ako referenčný režim, voči ktorému možno porovnávať ďalšie typy rozložení (zhlukované, filtrované a sub-zhlukované).

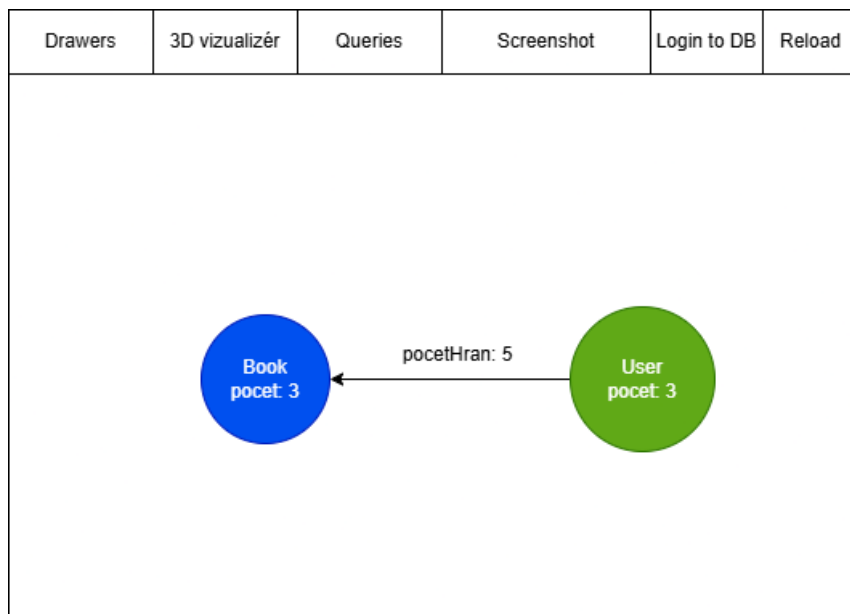
Dôležitou súčasťou topologického režimu je aj spôsob prezentácie informácií o prvkoch. Samotná scéna poskytuje globálny pohľad na štruktúru, avšak pri analýze je často potrebné poznať aj kľúče konkrétneho uzla (napr. identifikátor, názov alebo odvodené počty vzťahov). Preto aplikácia po výbere uzla zobrazuje detailný informačný panel so všetkými vlastnosťami vybraného objektu. Obrázok 15 ukazuje príklad takéhoto detailu, kde sa pri uzle typu *User* zobrazujú kľúče a súhrnné hodnoty (napr. počet vypožičaných kníh alebo počet väzieb typu *Follows*). Takáto interakcia prepája „globálne“ vizuálne vnímanie topológie s „lokálnym“ detailom konkrétnej entity.

2.1.2 Zhlukované trojdimenzionálne rozloženie

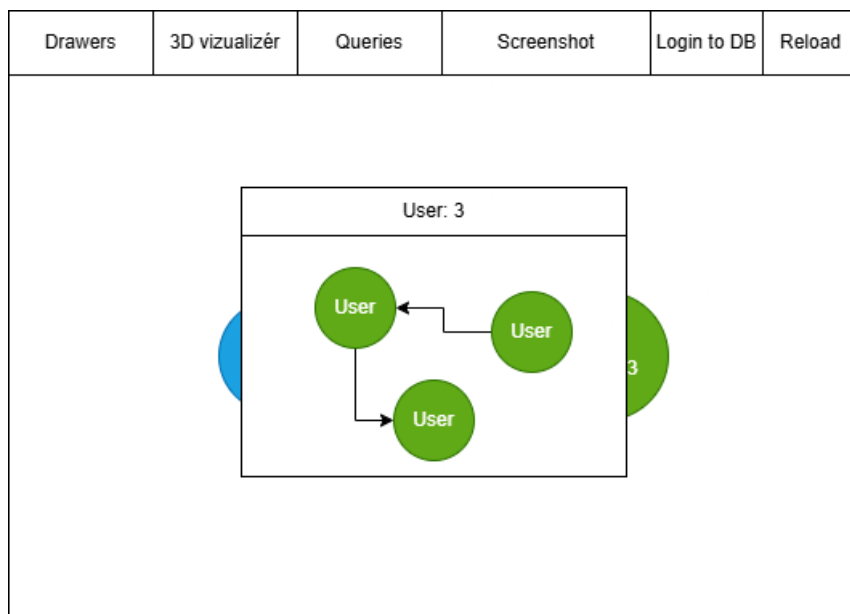
Pre komplexnejšie grafy a zobrazenia predstavujeme metódu agregácie vrcholov do spoločných celkov. Namiesto zobrazenia všetkých vrcholov a hrán zobrazíme počet vrcholov ekvivalentný počtu objektov v databáze.

Na obrázku 16 je výsledný návrh pre implementáciu zhlukovaného zobrazenia. Základnou myšlienkou je znížiť vizuálnu komplexitu tým, že namiesto jednotlivých entít zobrazíme agregované reprezentácie skupín vrcholov. To znamená, že vrcholy rovnakého typu sú zlúčené do jedného spoločného uzla.

Na obrázku 17 je znázornený návrh zobrazenia podgrafu v rámci zhlukového režimu. V tomto prípade už kliknutie na vrchol neslúži primárne na vypísanie detailných kľúčov jedného objektu, ale mení svoju funkcionálnosť na tzv. „rozbalenie“ zhuku. Po výbere agregovaného vrchola aplikácia zobrazí podgraf, ktorý tento vrchol reprezentuje, teda množinu pôvodných entít patriacich do danej skupiny spolu s ich relevantnými väzbami. Takýto prístup umožňuje plynulý prechod od prehľadového zhlukovaného zobrazenia k detailnejšej analýze konkrétnej časti grafu bez nutnosti manuálne filtrovať alebo znova formulovať dopyt.



Obrázok 16: Návrh zhlukovaného zobrazenia

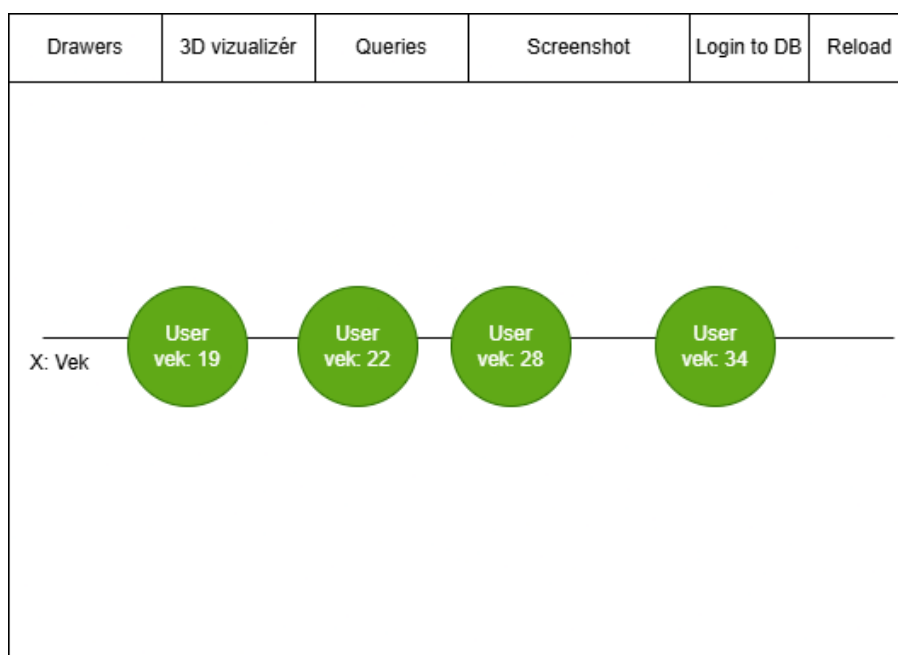


Obrázok 17: Návrh podgrafu v zhlukovom zobrazení

Kliknutie sa interpretuje ako navigačná akcia, ktorá používateľa presunie z agregovanej úrovne na úroveň vnútorného zobrazenia zhľuku. Detailné informácie o jednotlivých uzloch sa tak v tomto režime získavajú až následne, po rozbalení zhľuku, kde je možné opäť pracovať s konkrétnymi entitami a ich kľúčmi.

2.1.3 Filtrované trojdimenzionálne rozloženie

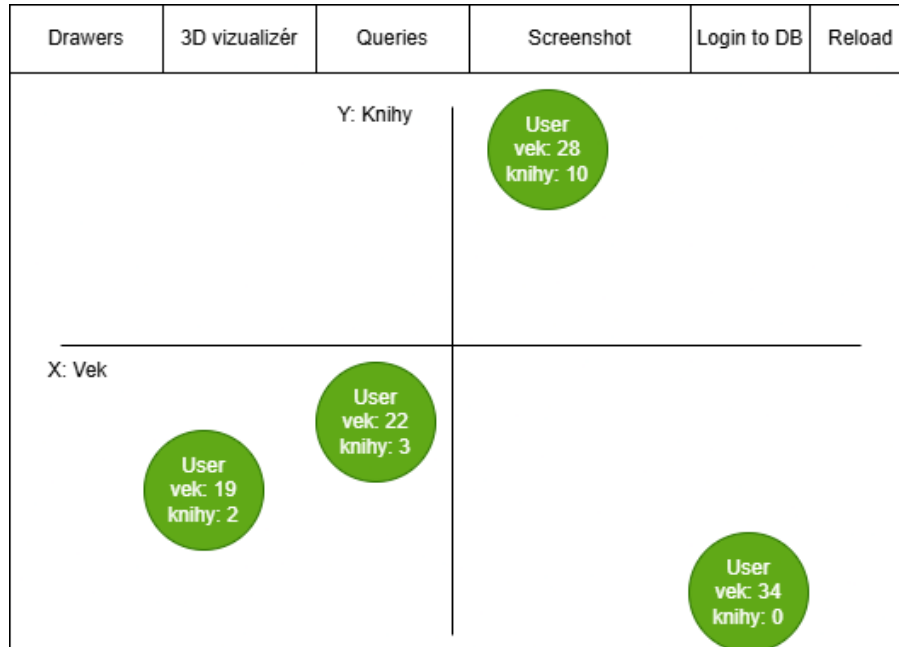
Filtrované trojdimenzionálne rozloženie je navrhnuté ako alternatíva k čisto topologickému rozmiestňovaniu, pričom jeho cieľom je zobraziť graf tak, aby priestorové umiestnenie vrcholov priamo filtrovalo v závislosti od zvolených kľúčov. Používateľ si zvolí jeden alebo viac filtrov, ktoré reprezentujú konkrétne vlastnosti vrcholov. Následne aplikácia transformuje hodnoty týchto kľúčov do súradnicového systému a každý vrchol umiestni na pozíciu, ktorá je určená kombináciou vybraných filtrov. To znamená, že vrcholy sa prestanú rozmiestňovať iba podľa topológie hrán a namiesto toho sa vizualizácia správa ako priestorový bodový graf nad zvoleným objektom a ich korešpondujúcimi filtermi.



Obrázok 18: Návrh filtrovaného zobrazenia pre jeden kľúč

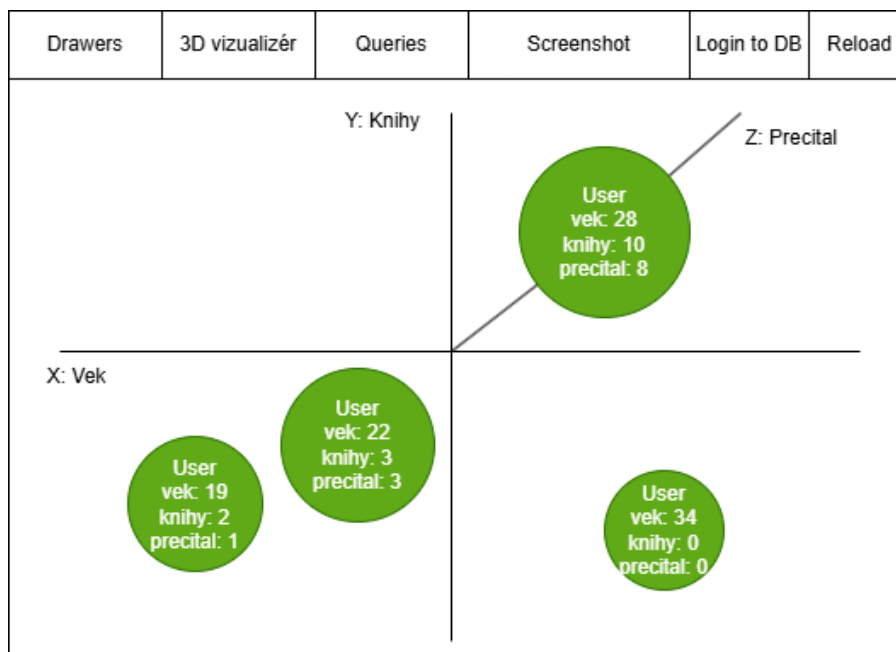
Na obrázku 18 je zobrazený prípad, keď používateľ zvolí iba jeden filter, takže rozloženie sa redukuje na jednorozmerné zobrazenie pozdĺž osi X . Každému vrcholu sa priradí súradnica x , ktorá vznikne mapovaním hodnoty vybraného kľúča do rozsahu zobrazovacej osi, pričom ostatné súradnice sa v tomto režime neuvažujú (resp. sú nastavené na konštantnú hodnotu). Výsledkom je usporiadanie vrcholov na jednej priamke, kde ich poloha priamo vyjadruje hodnotu filtrovaného kľúča. Takéto zobrazenie je vhodné najmä pri rýchлом porovnávaní objektov podľa jednej metriky, keďže používateľ okamžite vidí poradie prvkov a dokáže identifikovať extrémne hodnoty. Zároveň ide o najprehľadnejší

režim z hľadiska čitateľnosti, pretože minimalizuje vizuálnu zložitosť scény a znižuje riziko prekrytia. Hrany medzi vrcholmi sa zachovávajú ako doplnková informácia a môžu prepájať aj vrcholy, ktoré sú na osi X vzdialené, čím zostáva zachovaný kontext vzťahov aj pri výrazných rozdieloch v hodnote kľúču.



Obrázok 19: Návrh filtrovaného zobrazenia pre dva kľúče

Na obrázku 19 sú zvolené dva filtre, v takomto prípade sa rozloženie rozšíri do roviny XY , kde každému vrcholu prislúcha dvojica súradníc odvodená z hodnôt zvolených kľúčov. Os X reprezentuje prvý filter (v ukážke vek) a os Y druhý filter (počet kníh), takže výsledné zobrazenie má charakter bodového grafu, v ktorom poloha vrchola priamo vyjadruje kombináciu oboch hodnôt. Takéto zobrazenie umožňuje rýchlo porovnávať objekty podľa dvoch metrík naraz a identifikovať skupiny vrcholov s podobnými charakteristikami. V praxi je možné jednoducho rozlíšiť extrémny (napr. vysoký vek a vysoký počet kníh) a zároveň odhaliť odľahlé prvky, ktoré sa od ostatných výrazne odlišujú.



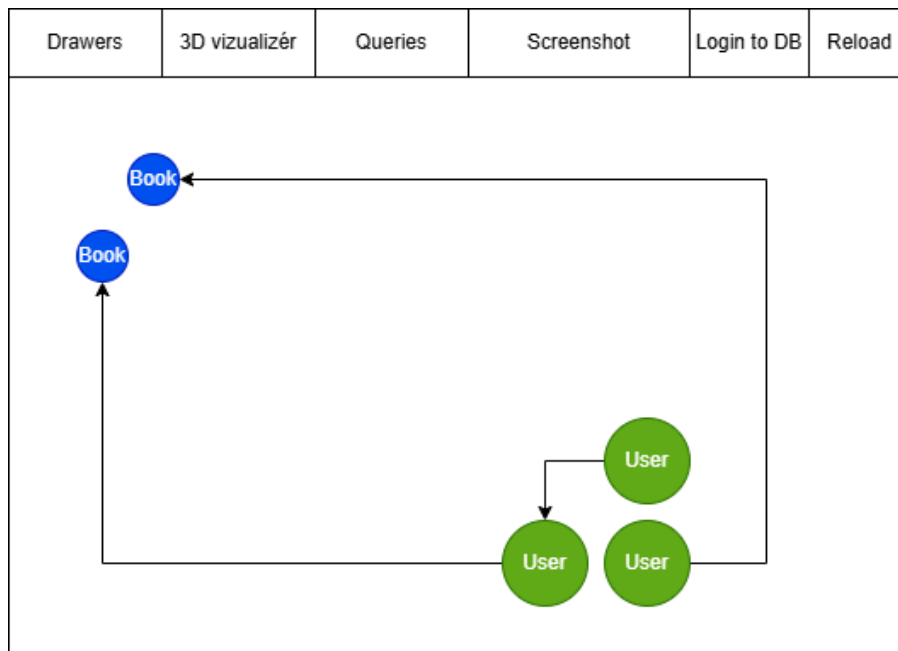
Obrázok 20: Návrh filtrovaného zobrazenia pre tri kľúče

Pri použití troch filtrov sa zobrazenie rozšíri do plnohodnotného trojdimenzionálneho priestoru XYZ , kde každému vrcholu prislúcha trojica súradníc (x,y,z) odvodená z hodnôt vybraných kľúčov. Os X reprezentuje prvý filter, os Y druhý filter a os Z tretí filter, takže poloha vrchola vyjadruje kombináciu troch metrík naraz. Takéto rozloženie umožňuje analyzovať vzájomné vzťahy medzi tromi vlastnosťami bez toho, aby bolo potrebné vytvárať viacero samostatných 2D pohľadov. Keďže výsledok je zobrazený perspektívne, používateľ môže pomocou rotácie a priblíženia meniť uhol pohľadu a tým odhaľovať štruktúru dát aj v prípadoch, keď by sa niektoré prvky v projekcii prekrývali. Hrany zostávajú zachované ako kontext, pričom vizuálne ukazujú prepojenia medzi vrcholmi aj v situáciách, keď sa objekty v priestore nachádzajú na výrazne odlišných hodnotách filtrov.

2.1.4 Sub-zhlukované trojdimenzionálne rozloženie

Posledná navrhovaná vizualizácia je *sub-zhlukové zobrazenie*, ktorého cieľom je priestorovo zoskupiť (zhlukovať) vrcholy do logických celkov, avšak bez toho, aby sa jednotlivé vrcholy agregovali do jedného spoločného uzla. Na rozdiel od zhlukového zobrazenia, kde sa celé skupiny nahrádzajú reprezentantom a v scéne sa zobrazuje len súhrnný počet prvkov, v sub-zhlukovom režime zostávajú viditeľné všetky uzly aj hrany. Zoskupenie sa preto prejavuje iba v rozložení. Vrcholy patriace do rovnakej skupiny sa nachádzajú priestorovo bližšie k sebe a vytvárajú lokálne „ostrovčeky“ v rámci scény, pričom medzi jednotlivými skupinami vznikajú väčšie vzdialenosti. Takéto zobrazenie je vhodné najmä v situáciách, keď chceme zachovať detail (konkrétne entity a ich vzťahy), no zároveň potrebujeme zlepšiť čitateľnosť a odhaliť prirodzené štrukturálne členenie grafu. Teda sub-zhlukové zobrazenie predstavuje kompromis medzi plne topologickým rozložením

a agregovaným zhlukovým režimom: používateľ vidí celý graf, ale zároveň získava informáciu o jeho členení na skupiny už na úrovni samotného rozmiestnenia prvkov.



Obrázok 21: Návrh sub-zhlukovaného zobrazenia

Na obrázku 21 je ilustrovaný výsledok sub-zhlukového rozloženia, v ktorom sa vrcholy automaticky zoskupujú do priestorovo oddelených oblastí podľa zvolenej logiky zhlukovania, pričom všetky entity zostávajú naďalej zobrazené samostatne. V rámci jednotlivých „ostrovčekov“ sú vrcholy umiestnené bližšie k sebe, čo uľahčuje vizuálne rozpoznanie skupín, zatiaľ čo väčšie vzdialenosti medzi zhlukmi pomáhajú oddeliť jednotlivé časti grafu a znižujú vizuálny šum v scéne. Zobrazenie zároveň zachováva hrany medzi vrcholmi, takže je možné sledovať nielen štruktúru v rámci zhluku, ale aj prepojenia medzi rôznymi skupinami.

2.2 Implementácia

V tejto časti práce sa zameriame na implementáciu aplikácie a na opis hlavných komponentov, ktoré zabezpečujú jej funkcionality. Postupne budú rozobraté kľúčové časti vizualizačného modulu, najmä inicializácia 3D scény, konfigurácia kamery, ovládanie používateľa a výpočet rozloženia grafu. V ďalšej časti popíšeme jednotlivé prvky implementácie, ako sú *OrbitControls* pre rotáciu, priblíženie a posun v scéne, klávesové ovládanie (WASD) pre plynulý pohyb kamery, mechanizmus výberu uzlov pomocou vrhania lúčov a detailné zobrazenie kľúčov vybraného objektu. Zároveň bude vysvetlené, ako aplikácia využíva silové rozloženie nad grafom, rieši prekrytie uzlov pomocou kolíznej sily a ako aktualizuje polohy hrán počas interakcie a renderovania.

```

1 const system = [
2     'You generate Cypher for Neo4j.',
3     'Return ONLY the Cypher query text. No markdown, no backticks,
4     no explanations.',
5     'Use ONLY the provided schema (labels/relationship types/
6     properties).',
7     'Return exactly ONE Cypher statement.',
8     'Do not add trailing semicolons.',
9     'Prefer inline literals (no parameters).',
10 ].join('\n');
11
12 const user = [
13     'Database schema:',
14     schema,
15     '',
16     'User request:',
17     userRequest,
18     '',
19     `Operation hint: ${op}`,
20 ].join('\n');
21
22 const body = {
23     model: 'gpt-4.1-mini',
24     input: [
25 { role: 'system', content: [{ type: 'input_text', text: system }]},
26 { role: 'user', content: [{ type: 'input_text', text: user }]},
27     ],
28     temperature: 0,
29 };

```

Ukážka 5: Implementácia OpenAI požiadavky

Na ukážke 5 je znázornená štruktúra požiadavky, ktorú aplikácia odosiela na *OpenAI* API pri generovaní dopytu v režime *LLM Queries*. V prvej časti sa zostaví systémová inštrukcia (*system*), ktorej úlohou je striktne určiť formát výstupu a obmedzenia generovania. V našom prípade sa model explicitne inštruuje, aby generoval výhradne dopyt v jazyku *Cypher* pre *Neo4j*, aby vrátil iba samotný text dopytu bez ďalších vysvetlení, aby používal výlučne poskytnutú schému databázy (labels, typy vzťahov a vlastností) a aby výsledkom bol presne jeden *Cypher* príkaz bez ukončujúcej bodkočiarky. Týmto spôsobom sa minimalizuje riziko, že model vráti nežiadaný formát (napr. komentáre alebo viacero dopytov) a zároveň sa zvyšuje pravdepodobnosť, že výsledok bude priamo vykonateľný v aplikácii.

```

1 function downloadScreenshotPNG(scale, filename):
2   if renderer/scene/camera/container not ready: return
3
4   (w, h) <- container size
5   name <- ensure ".png" extension
6
7   oldPR <- renderer.pixelRatio
8   oldSize <- renderer.size
9
10  maxDim <- min(GPU_MAX_RENDERBUFFER, GPU_MAX_TEXTURE)
11  prLimit <- floor(maxDim / max(w, h))
12  pr <- clamp(floor(scale), 1, prLimit)
13
14  screenshotInProgress <- true
15  renderer.setPixelRatio(pr); renderer.setSize(w, h)
16  updateEdgePositions(); renderer.render(scene, camera)
17
18  blob <- canvas.toPNGBlob()
19  if blob exists: triggerDownload(blob, name)
20
21  renderer.setPixelRatio(oldPR); renderer.setSize(oldSize)
22  screenshotInProgress <- false
23  updateEdgePositions(); renderer.render(scene, camera)

```

Ukážka 6: Implementácia sťahovania snímkov obrazovky

Funkcionalita vytvorenia snímky obrazovky je v implementácii na ukážke 6 realizovaná metódou *downloadScreenshotPNG(...)*, ktorá exportuje aktuálne zobrazenú 3D scénu do súboru PNG. Aplikácia dočasne zvýši interné rozlíšenie renderovania (cez *pixelRatio*), vykreslí scénu vo vyššej kvalite, uloží obsah renderovacieho plátna (*canvas*) do PNG a následne obnoví pôvodné nastavenia renderera, aby sa neznížil výkon pri ďalšom používaní.

2.2.1 Metódy pohybu, priblíženia a detailného zobrazenia

Keďže vizualizácia grafu prebieha v trojdimenzionálnom priestore, súčasťou nástroja musia byť aj vhodné mechanizmy navigácie. V aplikácii sú implementované dva odlišné spôsoby ovládania kamery:

- myšou pomocou *OrbitControls* (rotácia okolo cieľa, priblíženie/oddialenie a panning)
- klávesnicou pomocou režimu WASD, ktorý umožňuje plynulý posun pohľadu.

Cieľom je, aby sa používateľ vedel rýchlo presúvať po scéne, meniť vzdialenosť zobrazenia a zároveň si zachovať orientáciu pri skúmaní obrovských databáz.

```

1 each frame:
2   dt <- time since last frame
3
4   forward <- camera.viewDirection()
5   right   <- normalize(cross(forward, camera.up))
6   upScr   <- normalize(cross(right, forward))
7
8   want <- (0,0,0)
9   if W: want += upScr
10  if S: want -= upScr
11  if A: want -= right
12  if D: want += right
13  if want != 0: want <- normalize(want)
14
15  dist <- distance(camera.position, controls.target)
16  distScale <- clamp(dist / 150, 0.35, 3.0)
17  sprint <- (Shift pressed) ? 1.8 : 1.0
18  speed <- baseMoveSpeed * distScale * sprint
19
20  moveVel += want * (speed * dt)
21  moveVel *= 0.90    // tlmenie
22
23  camera.position += moveVel
24  controls.target += moveVel

```

Ukážka 7: Pohyb kamery pomocou WASD (pseudokód)

Na ukážke 7 je zobrazený pohyb pomocou **WASD**. Klávesové ovládanie je navrhnuté tak, aby sa posun realizoval v „view-plane“, teda v rovine kolmej na smer pohľadu. V každom snímku sa z kamery získa vektor smeru pohľadu (*forward*), z neho sa odvodí pravý smer (*right*) a následne smer hore (*upScreen*). Podľa stlačených klávesov (W,S,A,D) sa vytvorí požadovaný smer pohybu *want*, ktorý sa škáluje rýchlosťou závislou od vzdialenosti kamery od cieľa (aby bol pohyb prirodzenejší pri rôznych úrovniach priblíženia). Následne sa pohyb realizuje ako rýchlostný vektor *moveVel* s tlmením, čo spôsobí plynulý dojazd a eliminuje trhané skoky. Dôležité je, že sa naraz posúva **kamera aj cieľ OrbitControls**, takže používateľ prechádza po scéne bez zmeny orientácie.

```

1 on mouseWheel(delta):
2   dir <- normalize(camera.position - target)
3   dist <- length(camera.position - target)
4   dist <- clamp(dist * zoomFactor(delta), minDist, maxDist)
5   camera.position <- target + dir * dist
6 each frame:
7   controls.update()

```

Ukážka 8: Priblíženie pomocou OrbitControls (pseudokód)

Na ukážke 8 je uvedený pseudokód princípu priblíženia a oddialenia scény. Priblíženie je realizované prostredníctvom *OrbitControls*, ktoré upravujú vzdialenosť kamery od cieľového bodu `controls.target`. Zmena mierky zobrazenia sa uskutočňuje posunom kamery po priamke smerujúcej k cieľu (resp. od cieľa), pričom smer posunu je definovaný normalizovaným vektorom medzi kamerou a cieľom. Hodnota vzdialenosti je ohraničená na prípustný interval, aby sa zabránilo nežiaducej kolízii kamery s objektmi scény alebo príliš veľkému oddialeniu. V každom snímku sa následne volá `controls.update()`, ktoré zabezpečuje plynulosť pohybu a tlmenie zmien, vďaka čomu sa priblíženie správa stabilne aj pri rýchlych vstupoch používateľa.

```
1 this.controls = new OrbitControls(this.camera, this.renderer.domElement);
2 this.controls.enableDamping = true;
3
4 // v render čsluke
5 this.controls.update();
6 this.renderer.render(this.scene, this.camera);
```

Ukážka 9: Inicializácia a aktualizácia priblíženia (úryvok z implementácie)

Na ukážke 9 je zobrazený úryvok implementácie zodpovedný za inicializáciu a priebežnú aktualizáciu navigácie cez *OrbitControls*. Ovládacie prvky sú priamo naviazané na renderovací *canvas*, čo umožňuje používateľovi ovládať scénu myšou bez potreby dodatočných vstupných komponentov. Aktivácia tlmenia (`enableDamping`) prispieva k tomu, že zmeny polohy a orientácie kamery prebiehajú plynulo, čím sa znižuje vizuálna „trhavosť“ a zlepšuje sa používateľská orientácia pri manipulácii so scénou.

```
1 const raycaster = new THREE.Raycaster();
2 raycaster.setFromCamera(mouse, this.camera);
3 const hits = raycaster.intersectObjects(Object.values(this.nodeObjects));
4
5 if (hits.length > 0) {
6   const clicked = hits[0].object as THREE.Mesh;
7
8   if (this.selectedNode === clicked) {
9     this.resetNode(clicked);
10    this.selectedNode = null;
11  } else {
12    if (this.selectedNode) this.resetNode(this.selectedNode);
13    this.transformNodeToRectangle(clicked);
14    this.selectedNode = clicked;
15  }
16 }
```

Ukážka 10: Výber uzla a prepnutie detailu (úryvok z implementácie)

Na ukážke 10 je zobrazený úryvok implementácie, ktorý priamo realizuje logiku prepínania medzi štandardným a detailným stavom uzla. Po výbere objektu sa porovnáva, či ide o rovnaký uzol ako predtým. Ak áno, vykoná sa reset geometrie a odstráni sa stav „selected“. V opačnom prípade sa najprv zruší predchádzajúci výber a následne sa aktivuje transformácia uzla do obdĺžnikového panelu s textovým obsahom. Takéto riešenie umožňuje zobrazovať detail priamo v scéne bez nutnosti externého panelu, pričom sa zachováva možnosť plynule pokračovať v navigácii a interpretovať detail v kontexte okolitého podgrafu.

2.2.2 Topologické trojdimenzionálne rozloženie

Topologické trojdimenzionálne rozloženie predstavuje predvolený spôsob zobrazenia grafu v našom nástroji. Jeho cieľom je zobraziť štruktúru výsledného podgrafu tak, aby bol graf čitateľný, uzly sa v scéne neprekrývali a aby bolo možné vizuálne sledovať vzťahy medzi entitami. Rozloženie je implementované ako silová simulácia v 3D priestore, kde každý uzol je predstavený polohou (x,y,z) a simulácia iteratívne upravuje tieto polohy podľa definovaných síl. To znamená, že vrcholy sa prirodzene rozťahujú do priestoru, susediace uzly ostávajú relatívne blízko a celé rozloženie sa stabilizuje do stavu, ktorý je vhodný pre vizuálnu interpretáciu.

Kľúčovou súčasťou tohto režimu je kolízna sila *force-collide*. Jej úlohou je zabezpečiť minimálnu vzdialenosť medzi uzlami tak, aby sa v projekcii na obrazovku nezlievali alebo neprekryli. Pre každý uzol sa uvažuje jeho efektívny polomer, ktorý v našej implementácii závisí od veľkosti uzla (odvodenej napríklad od počtu kľúčov). Ak sa dva uzly v priebehu simulácie priblížia na vzdialenosť menšiu než súčet ich polomerov, kolízna sila vygeneruje korekciu, ktorá uzly od seba odtlačí. Výsledkom je rozloženie, v ktorom sú uzly vizuálne separované, čo zlepšuje čitateľnosť popisov aj presnosť interakcie.

```

1 for each pair of nodes (u, v):
2   du <- position[u] - position[v]
3   d <- length(du)
4   minD <- radius[u] + radius[v]
5   if d < minD:
6     push <- (minD - d) / 2
7     dir <- normalize(du)
8     position[u] += dir * push
9     position[v] -= dir * push

```

Ukážka 11: Kolízna korekcia medzi dvoma uzlami (pseudokód)

Na ukážke 11 je schematicky uvedený princíp kolíznej korekcie. V praxi sa tento mechanizmus neaplikuje ako jednorazový zásah, ale ako súčasť iteratívnej simulácie, v ktorej sa kolízna sila kombinuje s ďalšími zložkami rozloženia. Okrem kolízií je dôležitá aj väzbová sila (*force-link*), ktorá udržiava uzly spojené hranou v primeranej vzdialenosti. Táto

vzdialenosť je v implementácii odvodzovaná aj od veľkosti uzlov, čím sa znižuje riziko, že veľké uzly budú „pretáhovať“ hrany cez menšie uzly alebo vytvárať neprehľadné zhluky. Tretím prvkom je odpudivá sila (*force-many-body*), ktorá zabraňuje globálnemu zhutneniu grafu do jednej oblasti. Poslednou stabilizačnou zložkou je centrovanie (*force-center*), ktoré udržiava rozloženie v okolí stredu scény a zlepšuje kontrolu nad kamerou.

2.2.3 Zhlukované trojdimenzionálne rozloženie

Počas vývoja aplikácie začal vznikať problém s vizuálnym šumom. Tento šum bol spôsobený nadmerným množstvom jednotlivých vrcholov a prekrytia hrán, aj keď prekryvanie hrán je v 3D priestore ľahšie riešiteľný, pri nadmernom množstve vrcholov bolo potrebné implementovať lepšie riešenie.

```
1 const types = Array.from(new Set(this.rawNodes.map(n => n.type)));
2 const META_BASE = 1_000_000;
3
4 const countByType = new Map<string, number>();
5 for (const n of this.rawNodes) {
6   countByType.set(n.type, (countByType.get(n.type) ?? 0) + 1);
7 }
8
9 const metaNodes = types.map((t, i) => ({
10   id: META_BASE + i,
11   type: t,
12   isMeta: true,
13   attributes: { count: countByType.get(t) ?? 0 }
14 }));
15
16 this.nodes = metaNodes; // nahradenie pôvodných uzlov agregovanými
17 this.edges = metaEdges; // (agregované hrany sa pripraví samostatne)
18 this.rebuildGraph();    // spustenie vykreslenia zhlukovaného grafu
```

Ukážka 12: Zhlukovanie vrcholov podľa typu (skrátaná ukážka)

Na ukážke 12 je zobrazená skrátaná časť implementácie zhlukovania, ktorej cieľom je zredukovať počet zobrazených uzlov pri vizualizácii veľkých grafov. Najskôr sa zo vstupných dát určí množina typov uzlov (type) a následne sa v prechode cez všetky vrcholy vypočíta počet výskytov každého typu (countByType). Pre každý typ sa vytvorí jeden agregovaný uzol (metaNodes), ktorého kľúč count reprezentuje počet pôvodných vrcholov v databáze patriacich do danej kategórie. Takto vytvorené uzly nahradia pôvodnú množinu uzlov a po doplnení agregovaných hrán sa zavolá metóda rebuildGraph(), ktorá znovu zostaví scénu a zobrazí výsledné zhlukované zobrazenie.

```
1 if (this.overviewMode && clickedNode?.userData?.isMeta) {  
2   const t = String(clickedNode.userData.type);  
3   this.openClusterDialog(t); // zobrazí podgraf pre zvolený typ  
4 }
```

Ukážka 13: Kliknutie v zhlukovanom zobrazení (skrátene)

Na ukážke 13 vidíme implementáciu kliknutia. Keďže bolo potrebné prepísať metódu kliknutia, nezobrazujeme už dáta pre jeden vrchol po kliknutí zobrazíme v modálnom okne podgraf v ktorom sa nachádzajú všetky vrcholy a hrany v danom podgrafe.

2.2.4 Filtrované trojdimenzionálne rozloženie

Najinovatívnejšia metóda v aplikácii je trojdimenzionálne filtrovanie. Pre účely dátovej analýzy je častokrát potrebné filtrovať a zoradovať jednotlivé kľúče. V bežných nástrojoch sa takáto analýza realizuje najmä tabuľkovo alebo pomocou 2D grafov, čo však pri prepojených dátach často vedie k strate kontextu vzťahov. V našom riešení je preto cieľom prepojiť kľúčovú analýzu s grafovou vizualizáciou tak, aby poloha vrcholov v priestore priamo vyjadrovala hodnoty zvolených kľúčov. Používateľ si zvolí jeden až tri kľúče, ktoré sa mapujú na osi X,Y,Z, pričom počet zvolených filtrov určuje dimenzionalitu výsledného zobrazenia (1D/2D/3D). Vďaka tomu je možné porovnávať objekty podľa viacerých metrík naraz, identifikovať extrémne hodnoty a zároveň zachovať vizuálny kontext prepojení medzi objektmi.

```

1 // vstup: zoradený zoznam filtrov (0->X, 1->Y, 2->Z)
2 const attrs = this.analyticsSpec?.attributes ?? [];
3 const ax = attrs[0], ay = attrs[1], az = attrs[2];
4 const hasX = !!ax, hasY = !!ay, hasZ = !!az;
5
6 // hodnota kluca z uzla
7 const getVal = (n, a) => (n?.attributes ?? {})[a?.name];
8
9 // priradenie pozícií uzlov podľa filtrov (1D/2D/3D)
10 for (const n of visibleNodes) {
11     const x = hasX ? mapAxis(n, ax) : 0;
12     const y = hasY ? mapAxis(n, ay) : 0;
13     const z = hasZ ? mapAxis(n, az) : 0;
14     n.x = x; n.y = y; n.z = z;
15     this.nodeObjects[n.id]?.position.set(x, y, z);
16 }

```

Ukážka 14: Vstup filtrov a priradenie osí X/Y/Z

Ukážka 14 demonštruje, že filtrovací režim pracuje s pevne definovaným vstupom, ktorý určuje kľúče a poradie osí. V prípade, že je zvolený iba jeden kľúč, všetky uzly sa rozložia pozdĺž osi X. Pri dvoch kľúčoch vznikne rozloženie v rovine XY a pri troch kľúčoch plnohodnotné rozloženie v priestore XYZ. Samotná funkcia `mapAxis(...)` následne zabezpečuje prepočet konkrétnej hodnoty kľúča na súradnicu osi (napr. normalizáciou numerických hodnôt do rozsahu osi alebo mapovaním textových kategórií do poradia A–Z).

2.2.5 Sub-zhlukované trojdimenzionálne rozloženie

Sub-zhlukované (sub-clusterové) rozloženie predstavuje režim, ktorého cieľom je zvýrazniť štruktúru grafu bez toho, aby sa jednotlivé vrcholy agregovali do jedného spoločného reprezentanta. Na rozdiel od zhlukovaného zobrazenia (kde sa znižuje počet zobrazených vrcholov), v sub-zhlukovanom režime zostávajú všetky vrcholy aj hrany naďalej viditeľné, avšak rozmiestnenie je navrhnuté tak, aby sa v scéne prirodzene vytvorili „vnútorné skupiny“ a aby bol graf čitateľnejší. V našej implementácii je tento efekt dosiahnutý radiálnym rozložením založeným na prechode grafom pomocou algoritmu hľadanie do šírky (z angl. *Breadth-First Search*, BFS). BFS rozdelí uzly do vrstiev podľa ich najkratšej vzdialenosti od zvoleného koreňového uzla, pričom každá vrstva je následne umiestnená na samostatnej kružnici. Vzniká tak hierarchická štruktúra sústredných „prstencov“, ktoré prirodzene tvoria sub-zhluky bez straty detailu.

Základný princíp je nasledovný: najprv sa z grafu vytvorí susednostná štruktúra (z angl. *adjacency*) a vyberie sa koreňový uzol. Ak používateľ predtým vybral konkrétny uzol, tento uzol sa použije ako koreň; v opačnom prípade sa koreň zvolí automaticky ako uzol s najvyšším stupňom (počtom susedov), aby sa zvolil čo najreprezentatívnejší „centrálny“

bod siete. Následne BFS priradí každému uzlu hodnotu *level*, ktorá vyjadruje vzdialenosť v počte hrán od koreňa. Uzly s rovnakým *level* sa umiestnia na spoločný prstenec s polomerom závislým od čísla vrstvy, čím vznikne prehľadné rozloženie, ktoré zvýrazňuje lokálne okolie koreňa aj postupné rozširovanie siete smerom navonok.

Keďže graf môže obsahovať viacero nesúvislých komponentov, implementácia počíta BFS aj pre ďalšie komponenty. Každému ďalšiemu komponentu sa prideli nový interval vrstiev s medzerou (*gap*), aby boli komponenty vizuálne oddelené. Výsledkom je, že v sub-zhlukovanom zobrazení sú komponenty v priestore separované a zároveň v rámci každého komponentu vzniká radiálna štruktúra vrstiev. Z pohľadu čitateľnosti je to výhodné najmä v prípadoch, keď graf pozostáva z viacerých menších častí alebo keď používateľ potrebuje analyzovať „dosah“ určitého uzla do okolia (napr. komunikáciu, odporúčania, transakčné väzby).

```

1 build adjacency from edges
2 choose root (selected node OR highest-degree node)
3
4 level[root] <- 0
5 BFS from root to assign level for all reachable nodes
6
7 for each remaining connected component:
8   choose next start node
9   BFS from start node with level offset (add gap)
10
11 group nodes by level
12 for each level L:
13   r <- baseR + L * ringStep
14   place nodes of level L evenly on circle of radius r
15   assign z (e.g., by type layer) and small jitter

```

Ukážka 15: Sub-zhlukované rozloženie pomocou BFS (pseudokód)

Na ukážke 15 je uvedený pseudokód rozloženia, ktorý schematicky zachytáva kroky BFS vrstvenia a následného radiálneho umiestnenia uzlov.

```

1 private applyRadialBFS(): void {
2   if (this.sim) { this.sim.stop(); this.sim = null; }
3   if (!this.nodes?.length) return;
4
5   // adjacency (undirected)
6   const adj = new Map<number, Set<number>>();
7   for (const n of this.nodes) adj.set(n.id, new Set<number>());
8   for (const e of this.edges) {
9     const s = Number(e.source), t = Number(e.target);
10    adj.get(s)?.add(t); adj.get(t)?.add(s);
11  }
12
13  // root: selected node or highest-degree

```

```

14 let rootId: number | undefined = this.selectedNode?.userData?.id;
15 if (rootId == null) {
16     const deg = this.nodes.map(n => [n.id, adj.get(n.id)?.size ?? 0]);
17     deg.sort((a,b)=>b[1]-a[1]);
18     rootId = deg[0]?.[0] ?? this.nodes[0].id;
19 }
20
21 // BFS levels (handle multiple components)
22 const level = new Map<number, number>();
23 const unvisited = new Set(this.nodes.map(n => n.id));
24 const queue: number[] = [];
25
26 const bfsFrom = (start: number, startLevel: number) => {
27     level.set(start, startLevel);
28     unvisited.delete(start);
29     queue.push(start);
30     while (queue.length) {
31         const u = queue.shift()!;
32         const uL = level.get(u)!;
33         for (const v of (adj.get(u) ?? [])) {
34             if (unvisited.has(v)) {
35                 level.set(v, uL + 1);
36                 unvisited.delete(v);
37                 queue.push(v);
38             }
39         }
40     }
41     return Math.max(...Array.from(level.values()));
42 };
43
44 let maxLevel = bfsFrom(rootId!, 0);
45 while (unvisited.size > 0) {
46     const next = Array.from(unvisited)[0];
47     maxLevel = bfsFrom(next, maxLevel + 2); // gap between components
48 }
49
50 // group by level + radial placement
51 const groups = new Map<number, number[]>();
52 for (const n of this.nodes) {
53     const l = level.get(n.id)!;
54     (groups.get(l) ?? groups.set(l, [])).get(l)!.push(n.id);
55 }
56
57 const ringStep = 28, baseR = 8;
58 for (const [lvl, ids] of Array.from(groups.entries()).sort((a,b)=>a[0]-b[0])) {
59     const r = baseR + lvl * ringStep;

```

```

60     const k = ids.length;
61     const dTheta = (2 * Math.PI) / k;
62
63     ids.forEach((id, i) => {
64         const theta = i * dTheta;
65         const x = Math.cos(theta) * r;
66         const y = Math.sin(theta) * r;
67         const node = this.nodes.find(n => n.id === id)!;
68         node.x = x; node.y = y; node.z = node.z ?? 0;
69         this.nodeObjects[node.id]?.position.set(x, y, node.z);
70     });
71 }
72 }

```

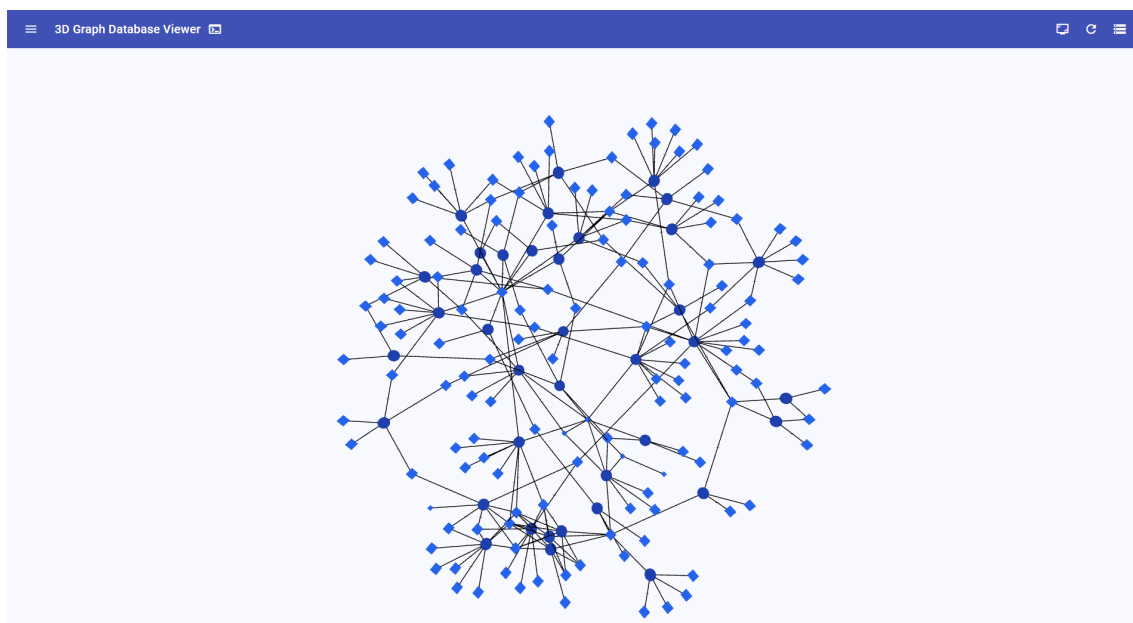
Ukážka 16: Sub-zhlukované rozloženie pomocou BFS (úryvok z implementácie)

V implementácii je rozloženie realizované metódou `applyRadialBFS()`, ktorá najprv zostaví susednosti, vykoná BFS pre všetky komponenty a následne uzly rozmiestni na prstence. Na ukážke 16 je zobrazený skratený úryvok priamo z implementácie (výpočet vrstiev BFS a radiálne rozmiestnenie). Z pohľadu interpretácie výsledku je dôležité, že BFS rozloženie vytvára „sub-zhluky“ prirodzene: uzly s rovnakou vzdialenosťou od koreňa sú v jednom prstenci, a preto sú vizuálne vnímané ako skupina. Toto je užitočné napríklad pri analýze dosahu (koľko uzlov je v 1., 2. alebo 3. kroku od vybraného uzla), pri skúmaní šírenia informácie (komunikačné grafy, správy), alebo pri hľadaní okolia významného uzla (odporúčacie scenáre). Na rozdiel od silového rozloženia, kde uzly „hľadajú“ svoju pozíciu iteratívne a výsledok závisí od parametrov síl, BFS rozloženie je deterministickejšie v tom, že vrstva uzla je daná štruktúrou grafu a nie fyzikálnou simuláciou.

Implementácia zároveň zachováva možnosť analyzovať viacero komponentov. Ak graf nie je súvislý, jednotlivé komponenty sa v rozložení neprekrývajú, pretože každý ďalší komponent dostane posunuté číslovanie vrstiev. V praxi to zlepšuje čitateľnosť pri databázach, ktoré obsahujú viacero oddelených komponentov (napr. rôzne skupiny zákazníkov v shop databáze, izolované konverzácie v messages databáze). V spojení s 3D navigáciou (rotácia, priblíženie, posun) je následne možné preskúmať jednotlivé prstence, identifikovať uzly v rovnakej vrstve a sledovať hrany medzi vrstvami, ktoré vizuálne reprezentujú postupné „preskakovanie“ z centra do okolia.

3 Overenie nástroja na trojdimenzionálnu vizualizáciu grafových databáz

Táto kapitola sa zameriava na experimentálne overenie navrhnutého 3D vizualizačného nástroja a na prezentáciu jeho správania pri rôznych typoch vstupných dát a režimoch zobrazenia. Cieľom overenia je ukázať, že aplikácia dokáže stabilne vykresliť graf v trojdimenzionálnom priestore, poskytuje konzistentnú interakciu (kamera, výber prvkov, prepínanie režimov) a že jednotlivé vizualizačné režimy vedú k očakávanému výsledku. Overenie je postavené na reálnej implementácii komponentu vizualizéra, ktorý využíva *Three.js* na renderovanie scény, *OrbitControls* a klávesové ovládanie (WASD) na navigáciu, a knižnicu *d3.js* na výpočet rozloženia pomocou silového modelu (vrátane kolízií a väzbovej sily).



Obrázok 22: Implementovaná aplikácia 3D Graph database viewer

Na obrázku 22 je zobrazená naimplementovaná aplikácia pripojená na *Neo4j* databázu.



Obrázok 23: Hlavná navigačná lišta (Zmenšená pre účely čitateľnosti)

Hlavná navigačná lišta obsahuje všetky dostupné funkcionality aplikácie. V nasledujúcich bodoch tieto funkcionality popíšeme:

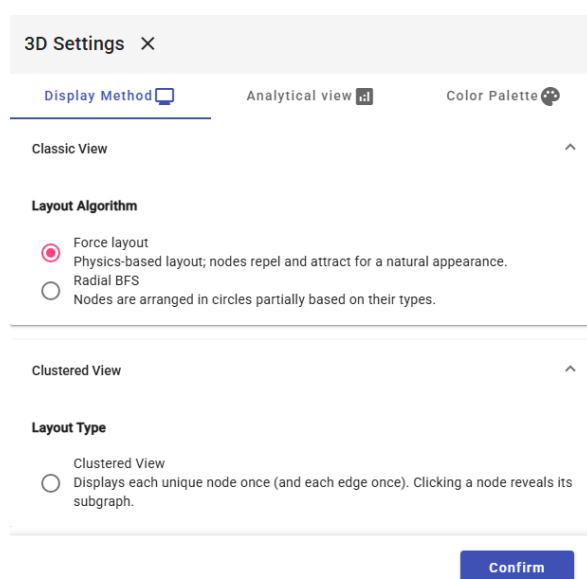
- **Bočné panely** - Vľavo od názvu aplikácie sa nachádza menu ktoré po kliknutí otvorí menu s *3D Settings* a *Data analytics*.

 Data Analytics

 3D Settings

Obrázok 24: Bočná lišta

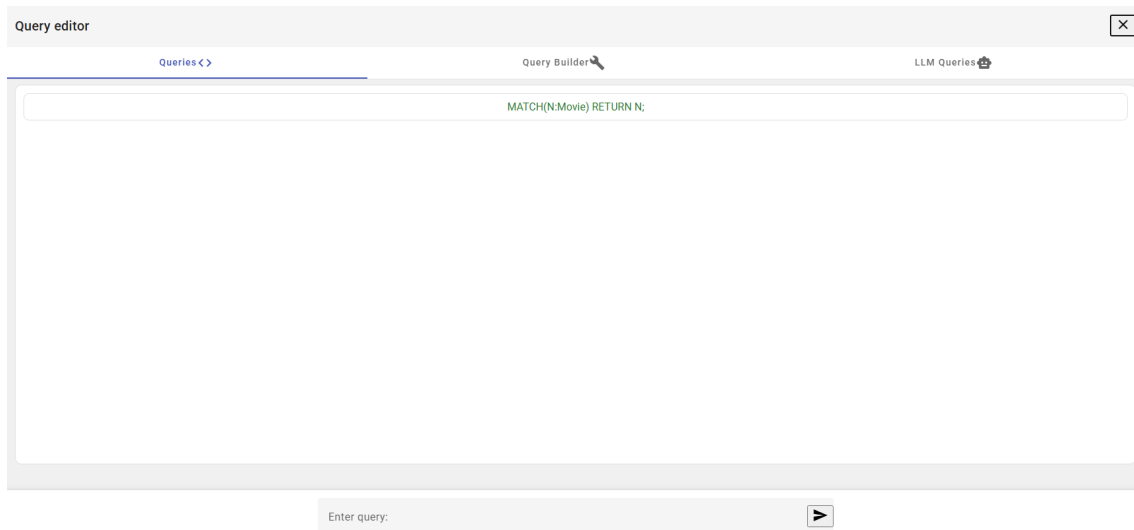
V bočnej lište aplikácie sa nachádzajú dve funkcionality a to podrobná analýza dát vykonaná nad databázou a *3D settings*. *3D Settings* nám umožňujú nastaviť a upraviť vizualizáciu dát.



Obrázok 25: Metódy vykreslenia grafu

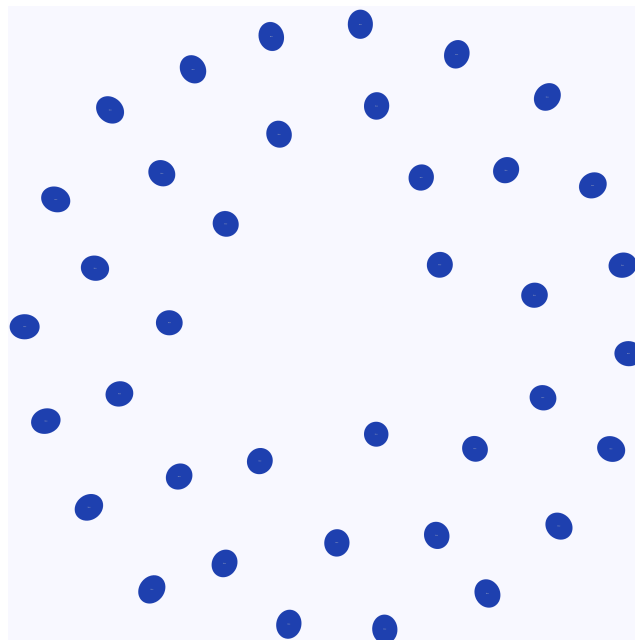
Na obrázku 25 vidieť *3D Settings*, kde si vieme nastaviť metódu zobrazenia, tieto metódy preberieme pri jednotlivých ukážkach databáz v nasledujúcich podkapitolách.

- **Dopyty** - Otvorenie modálneho okna na zobrazenie okna na vkladanie dopytov.



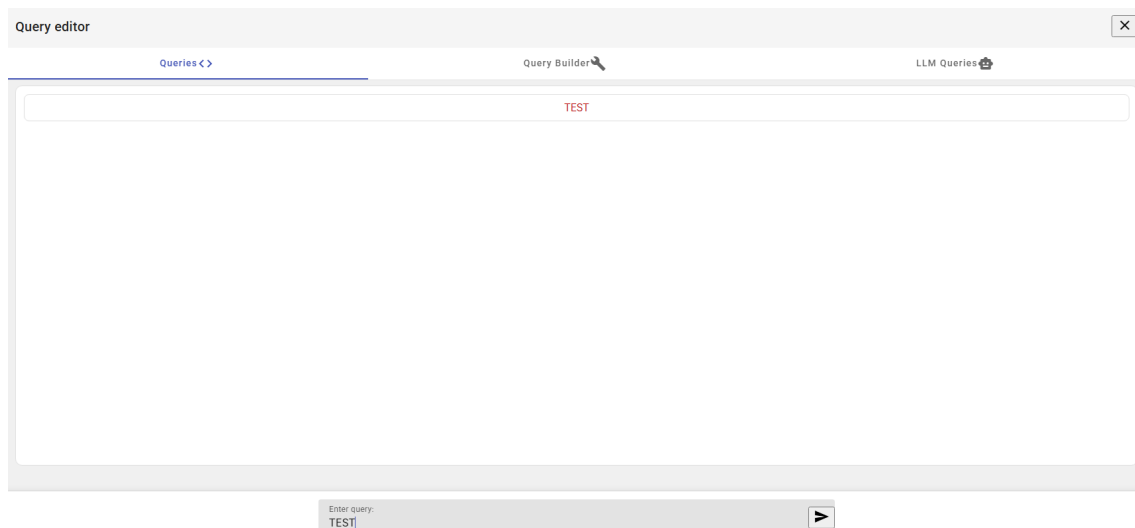
Obrázok 26: Modálne okno na dopyty

Na obrázku 26 je zobrazené modálne okno na dopytovanie do databázy. Manuálne dopytovanie umožňuje používateľovi napísať príkaz v jazyku *Cypher* v prípade že príkaz je syntakticky korektný a nachádza sa v dopyte kľúčové slovo *RETURN* sa okno automaticky uzavrie a zobrazí sa výsledok v aplikácii.



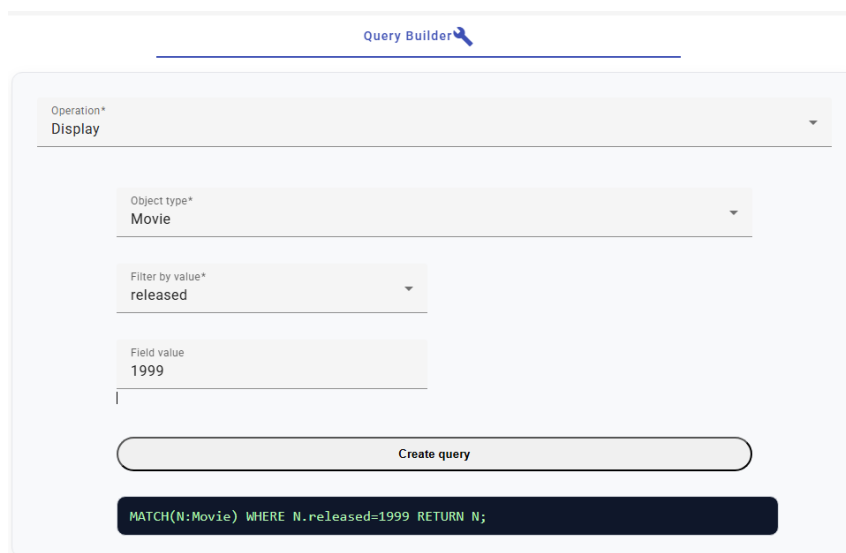
Obrázok 27: Výsledný podgraf po vykonaní dopytu

Kedže príkaz vykonaný na obrázku 26 bol syntakticky správny, na obrázku 27 môžeme vidieť výsledok.



Obrázok 28: Syntakticky zlý dopyt

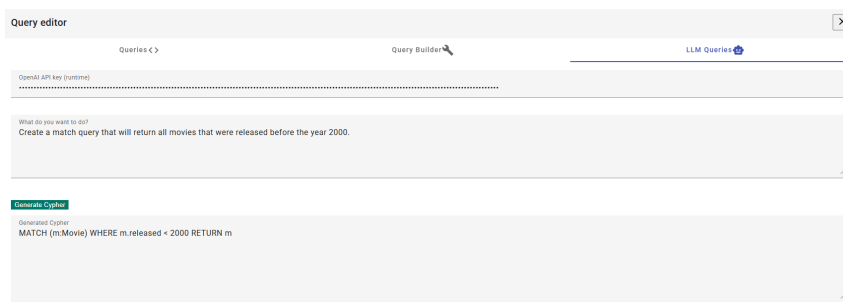
Každý dopyt je odoslaný na databázový systém, ktorý odpovedá vo forme JSON, dáta sú načítané a zobrazené. Keď zadáme syntakticky nesprávny dopyt ako je zobrazené na obrázku 28 databázový systém vracia chybové hlásenie 500, v tomto prípade sa okno nezavrie a používateľ môže ďalej písať dopyty.



Obrázok 29: Low-code dopytovanie

Na obrázku 29 prezentujeme ďalší spôsob vytvárania dopytov a to formou *low-code* respektíve *no-code*. Ako prvý krok v rámci tohto typu dopytovania, používateľ volí: **DISPLAY**, **UPDATE**, **DELETE**, **INSERT**. Po zvolení operácie **DISPLAY** je potrebné vybrať objekt, na ktorom sa vykoná operácia *MATCH*. Každý objekt má vlastné kľúče, preto je pre každý objekt vytvorený zoznam s jeho zoznamom kľúčov.

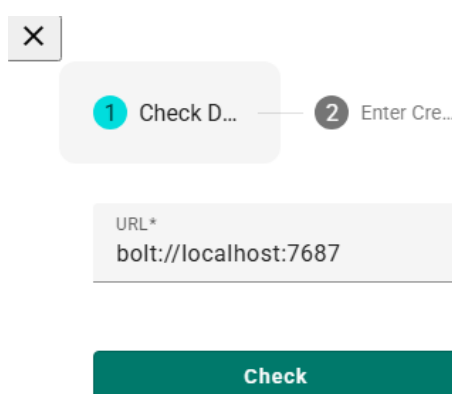
Po výbere kľúča, na základe ktorého bude dopyt filtrovať objekty, je zadaná hodnota, po kliknutí *Create query* sa vytvorí v čiernom ráme výsledný dopyt.



Obrázok 30: LLM dopytovanie

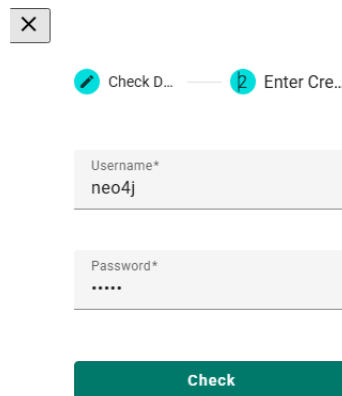
Na obrázku 30 Môžeme vidieť dopytovanie v prirodzenom jazyku, ktoré ja na pozadí zabezpečené cez *OpenAI* API. *OpenAI* poskytuje možnosť vytvorenia náhodne vygenerovaného kľúča, ktorý po vložení môžeme použiť na komunikáciu s API. Po vložení platného kľúča napíšeme vetu v prirodzenom jazyku, ktorá sa odošle na API, po spracovaní vety sa naspäť vráti výsledný dopyt z *OpenAI* [19].

- **Snímka obrazovky** - Vytvorí snímku obrazovky z aktuálnej vizualizácie.
- **Opätovné načítanie** - nanovo načítame všetky dáta z databázy.
- **Prihlásenie do databázy** - pripojenie do databázového systému.



Obrázok 31: Kontrola existujúceho databázového systému

Najdôležitejšia funkcionálna je pripájanie na databázový systém. Na obrázku 31 je zobrazené modálne okno ktoré kontroluje odpoveď z databázového systému na lokálnej adrese: **bolt://localhost7687**.



The image shows a login interface. At the top, there is a close button (X) and two progress indicators: a green checkmark labeled 'Check D...' and a blue question mark labeled 'Enter Cre...'. Below these are two input fields: 'Username*' containing the text 'neo4j' and 'Password*' with masked characters '*****'. At the bottom is a green button labeled 'Check'.

Obrázok 32: Vkladanie mena a hesla

Ak databáza odpovedá, presúvame sa cez krokovač na vloženie mena a hesla, ako je zobrazené na obrázku 32. Po úspešnej autentifikácii sa modálne okno uzavrie a zobrazí sa celá databáza.

3.1 Použité databázy

Na overenie funkcionality vytvoreného nástroja boli použité 3 databázy. Hlavná databáza na ktorej bolo robené testovanie je **Movies** [20], táto databáza je štandardne dostupný pre všetkých používateľov, ktorí si stiahnu *Neo4j Desktop*, príkazom *:play Movies*, a pozostáva z:

- 171 Vrcholov - typu osoba (Person) a film (Movie), pričom každá osoba je opísaná kľúčmi *meno {name}* a *rok narodenia {born}*. Pre každý film je zaznamenané *názov {title}*, *rok vydania {released}* a *stručný popis {tagline}*
- 253 Hrán - označujúcich vzťahy medzi osobami a filmami, konkrétne vzťah *hral v {ACTED_IN}* *režiroval {DIRECTED}* *tvoril {PRODUCED}* *napísal {WROTE}*, *hodnotil {REVIEWED}*

Databáza **Movies** je vhodná na skúšanie a testovanie dopytov, prepája medzi sebou ľudí (hercov a režisérov) a filmy.

Ďalšia použitá databáza bola **Messages**, ktorá simuluje komunikáciu medzi používateľmi a prihlasovanie na stránku, obsahuje:

- 197 Vrcholov - typu používateľ (User), správa (Message), skupinový chat (Group-Chat), token (Token), relácia (Session) a zariadenie (Device), pričom používateľ je opísaný kľúčmi *id*, *používateľské meno {username}* a *dátum vytvorenia {createdAt}*. Pre každú správu je zaznamenané *identifikátor {id}*, *typ správy {kind}*, *telo správy {body}* a *čas odoslania {sentAt}*. Skupinový chat má *identifikátor {id}* a *názov {name}*. Token má *identifikátor {id}*, *čas vydania {issuedAt}* a *čas expirácie*

{expiresAt}. Relácia má *identifikátor {id}* a *čas vytvorenia {createdAt}*. Zariadenie má *identifikátor {id}* a *operačný systém {os}*.

- 200 Hráč - označujúcich vzťahy medzi objektmi, konkrétne vzťah *odoslal {SENT}*, *adresované {TO}*, *v chate {IN_CHAT}*, *má token {HAS_TOKEN}*, *má reláciu {HAS_SESSION}*, *autentifikované tokenom {AUTHED_BY}* a *má zariadenie {HAS_DEVICE}*.

Messages ukazuje ako používatelia sociálnej siete môžu komunikovať medzi sebou alebo posilať správy do iných skupín. Táto databáza bola vygenerovaná s umelými dátami. Prostredníctvom nástroja *ChatGPT*. Pričom bol použitý nasledujúci dopyt (viď. ukážku 17).

```
1 I want you to build a Neo4j query that creates a large database.
2 Include users, messages between users, login tokens,
3 and group chat nodes.
4 Use at least 4 different node types and multiple relationship types.
5 Model communication as: User -> Message -> User.
6 Create at least 100 nodes and a high number of relationships.
```

Ukážka 17: Dopyt na vytvorenie databázy Messages columns

Online obchod alebo skrátene iba **Shop** je taktiež vygenerovaná databáza, kde si používatelia vedia kupovať dostupné produkty, databáza obsahuje:

- 56 Vrcholov - typu zákazník (Customer), produkt (Product), kategória (Category) a objednávka (Order), pričom každý zákazník je opísaný kľúčmi *identifikátor {id}*, *meno {name}*, *krajina {country}*, *mesto {city}*, *segment {segment}*, *dátum registrácie {signupDate}* a *marketingový súhlas {marketingOptIn}*. Každý produkt je opísaný kľúčmi *kód {code}*, *názov {name}*, *značka {brand}*, *cena {price}*, *náklad {cost}*, *marža {margin}*, *hodnotenie {rating}* a *skladom {stock}*. Každá kategória je opísaná kľúčmi *identifikátor {id}*, *názov {name}* a *oddelenie {department}*. Každá objednávka je opísaná kľúčmi *identifikátor {id}*, *dátum objednávky {orderDate}*, *kanál {channel}*, *stav {status}*, *platobná metóda {paymentMethod}*, *poplatok za dopravu {shippingFee}*, *zľava {discount}*, *mena {currency}*, *suma položiek {itemsTotal}* a *celková suma {total}*.
- 100 Hráč - označujúcich vzťahy medzi objektmi, konkrétne vzťah *patri do kategórie {IN_CATEGORY}*, *vytvoril objednávku {PLACED}*, *obsahuje produkt {CONTAINS}*, *prezeral {VIEWED}*, *pridal do wishlistu {WISHLISTED}* a *kupovaný spolu s {ALSO_BOUGHT_WITH}*.

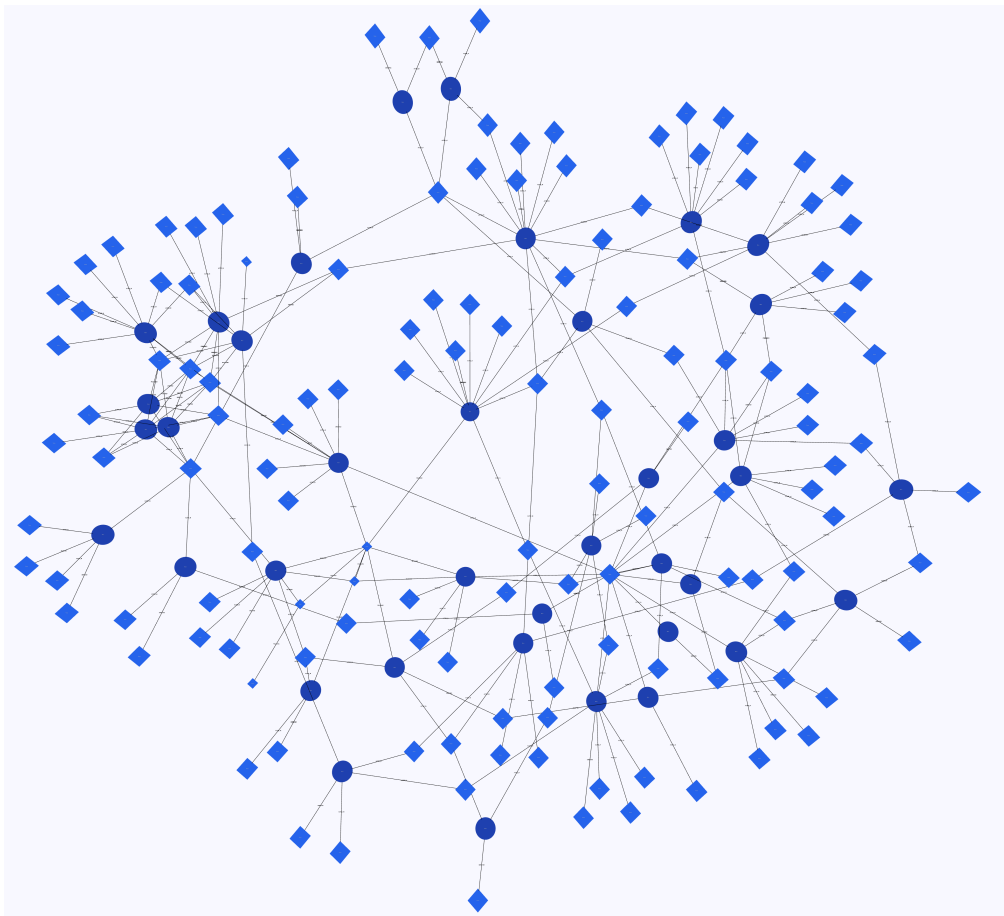
Shop je databáza na analýzu tržieb, marže, nákupných košíkov a prezerania nákupov a vzťahov medzi produktami, ktoré boli spolu kúpené. Rovnako ako Messages ide o umelo

generovanú databázu ktorá bola vytvorená v nástroji *chatGPT* s nasledujúcim dopytom (viď. ukážku 18).

- 1 Create an online shop database (categories, products, customers, orders).
- 2 Make the data meaningful so it can be used for data analysis.
- 3 Use 4 types of nodes and 6 different types of edges.
- 4 Maximum 80 nodes and maximum 100 edges.

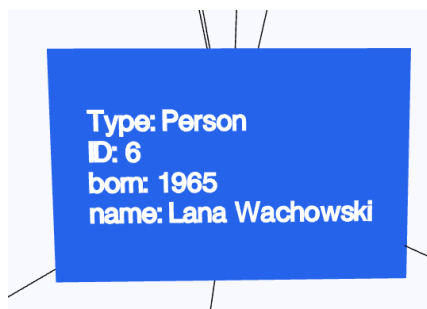
Ukážka 18: Dopyt na vytvorenie databázy Shop

3.2 Topologické trojdimenzionálne rozloženie



Obrázok 33: Databáza movies v topologickom rozložení

Na obrázku 33 je zobrazená databáza *Movies* v topologickom rozložení založenom na silovej simulácii. Vizuálne je možné pozorovať typickú štruktúru databázy, kde sa okolo výraznejších uzlov (uzly s vyšším počtom väzieb) vytvárajú lokálne zhluky a menšie „hviezdicové“ podštruktúry. Kolízna zložka rozloženia zabezpečuje, že uzly ostávajú dostatočne oddelené, vďaka čomu je graf čitateľný aj pri väčšom počte hrán. Zároveň je zachovaný topologický kontext: uzly, ktoré spolu súvisia cez viacero vzťahov, sa v priestore prirodzene približujú, zatiaľ čo menej previazané časti grafu ostávajú viac rozptýlené.



Obrázok 34: Rozklinutý vrchol

Obrázok 34 demonštruje detailné zobrazenie vybraného uzla po interakcii používateľa. Po kliknutí sa pôvodný tvar uzla dočasne zmení na obdĺžnikový panel, na ktorom sú zobrazené základné informácie o entite: typ uzla, jeho identifikátor a vybrané kľúče (konkrétne: *born* a *name*). Takýto spôsob zobrazenia umožňuje sprístupniť konkrétne údaje priamo v kontexte scény bez potreby externého panelu alebo ďalšieho okna. Toto eliminuje potreby modálnych okien. Detail je pritom vhodný najmä pri analýze databáz, kde je často potrebné rýchlo overiť, ktorú konkrétnu osobu alebo objekt používateľ vybral.

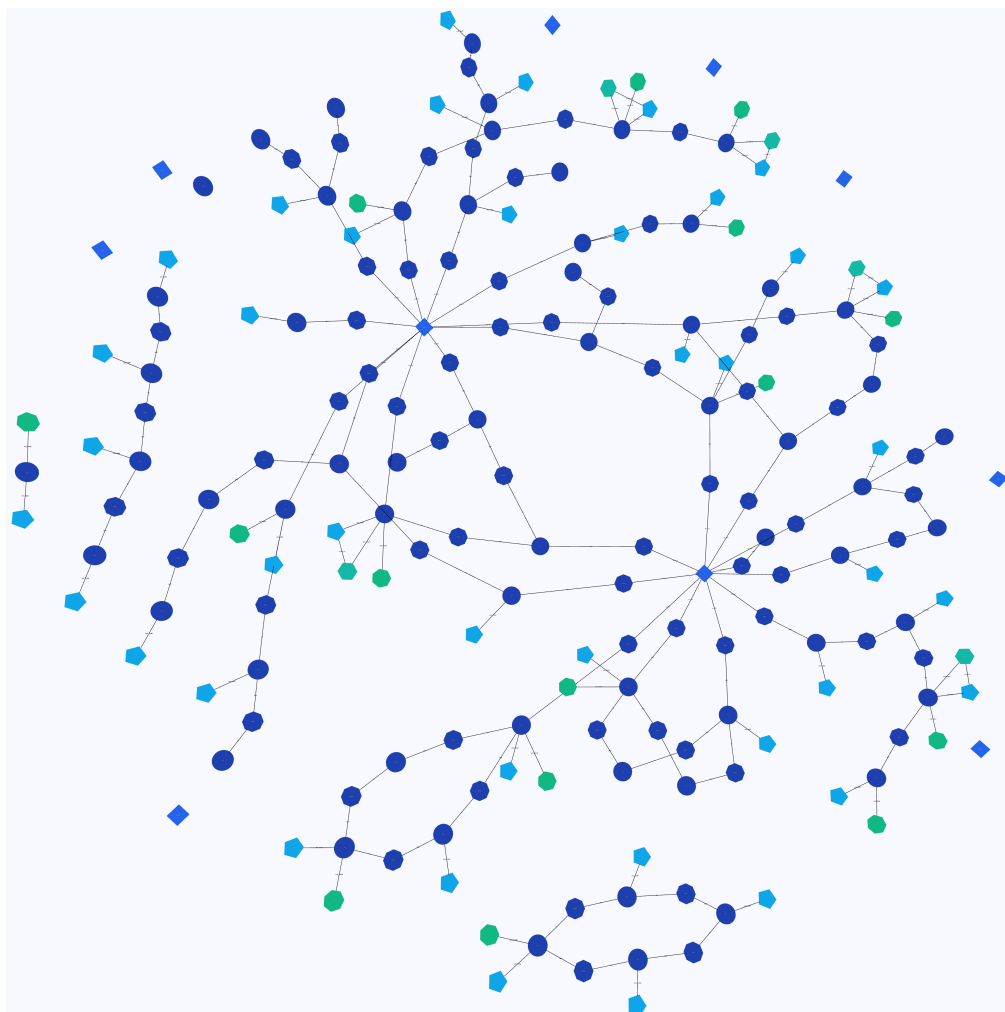
Na obrázku 35 je zobrazená databáza *Messages* v topologickom rozložení. V porovnaní s jednoduchšími databázami je v tomto prípade viditeľných viacero farebných skupín, keďže graf obsahuje vyšší počet typov objektov. Typ vrchola sa však v aplikácii neodlišuje iba farbou, ale aj tvarom. Prvý typ je zobrazený ako kruh a ďalšie typy sú reprezentované pravidelnými mnohouholníkmi, pričom s rastúcim poradím typu sa zvyšuje aj počet strán (napr. štvoruholník, päťuholník, šesťuholník a pod.). Takéto kódovanie zvyšuje čitateľnosť vizualizácie najmä v situáciách, keď sa farby v hustej scéne vizuálne miešajú alebo keď používateľ pracuje s viacerými typmi uzlov naraz. Kombinácia farby a tvaru zároveň umožňuje jednoduchšie rozlíšenie objektov aj pri rôznych úrovniach priblíženia, keď farba nemusí byť jediným spoľahlivým rozlišovacím znakom.

Na obrázku 36 je zobrazená databáza v ktorom je možné pozorovať rozdelenie grafu na viacero menších komponentov, ktoré sú v priestore jasne oddelené, čo zvyšuje čitateľnosť a zjednodušuje interpretáciu štruktúry. V rámci jednotlivých komponentov sa vytvárajú kompaktnejšie zhľuky s viditeľnými lokálnymi vzťahmi, pričom kolízna zložka zabráňuje prekrytiu aj pri hustejšom prepojení.

3.3 Zhľukované trojdimenzionálne rozloženie

Na obrázku 37 vidíme rovnakú databázu *movies* ale tentokrát používame zhľukované zobrazenie. Oproti sebe sú dva vrcholy typu *Movies* a *Person*. Tu strácame informáciu o konkrétnych objektoch ale v prípade obrovského počtu typov vrcholov vieme analyzovať graf z iného pohľadu.

Na obrázku 38 po kliknutí na vrchol označujúci objekty typu *Movies* sa nám zobrazí



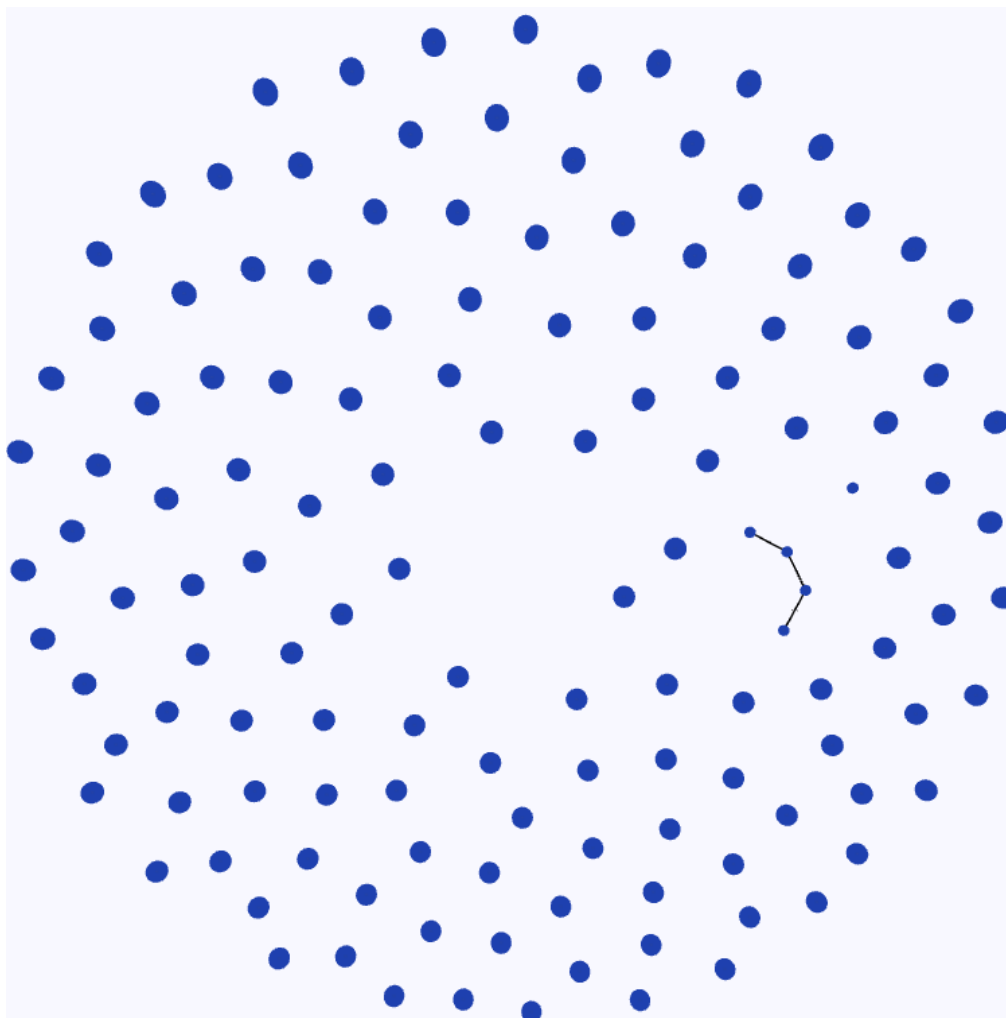
Obrázok 35: Databáza Messages v topologickom rozložení



Obrázok 36: Databáza shop v topologickom rozložení

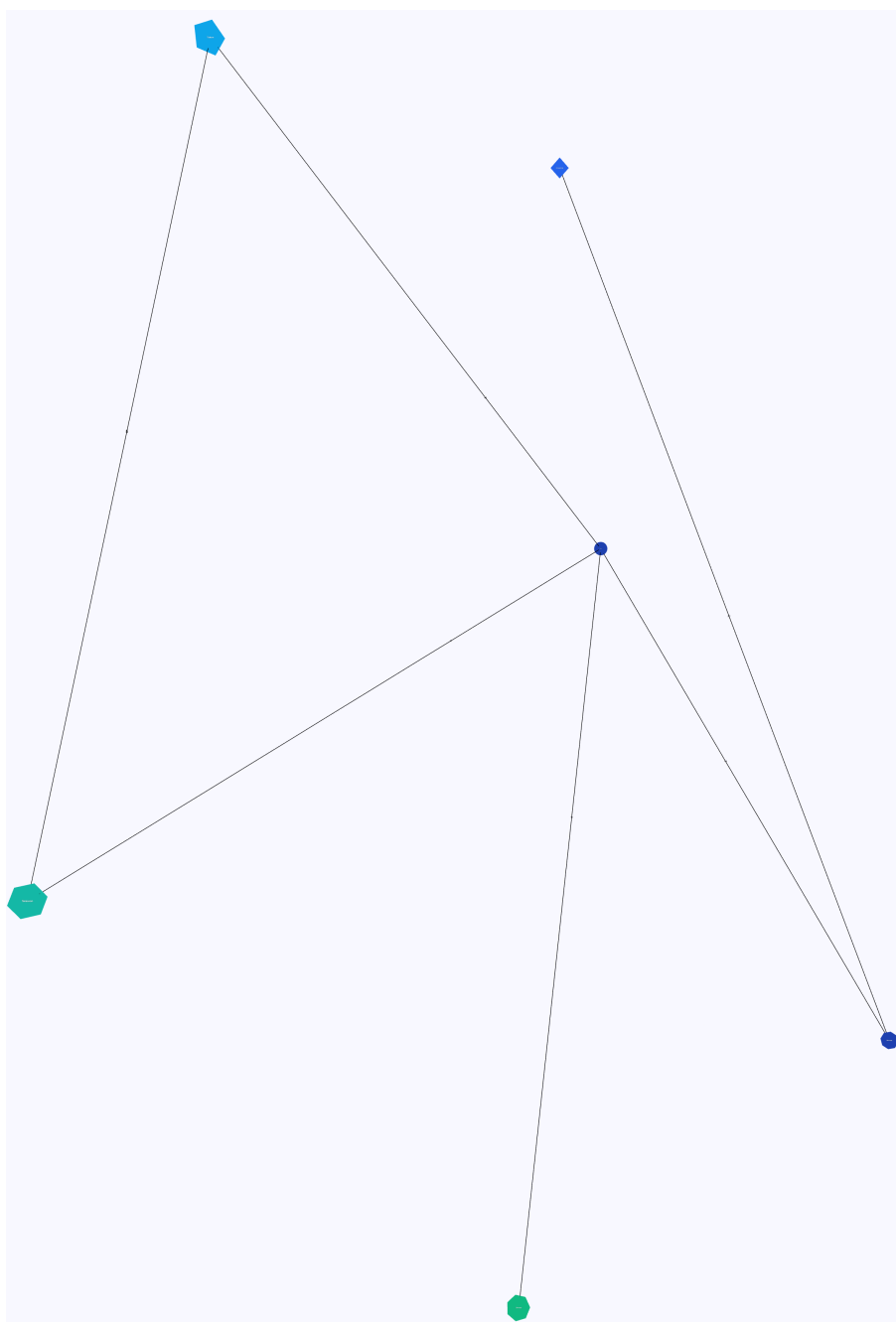


Obrázok 37: Databáza movies v zhlukovanom rozložení



Obrázok 38: Modálne okno objektu Movies

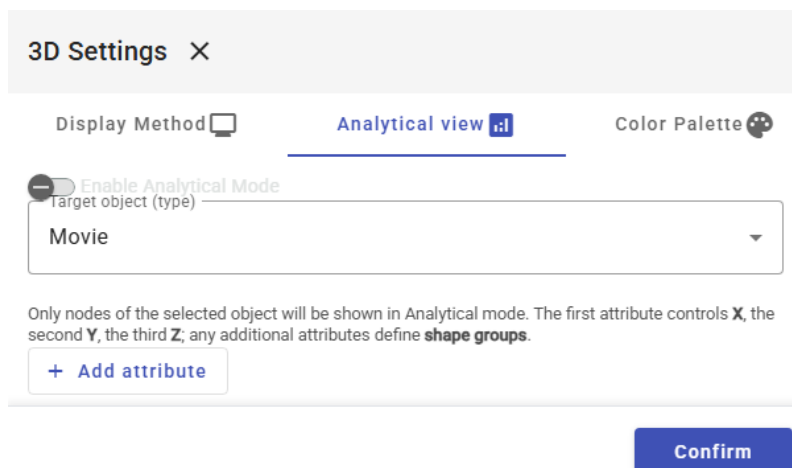
podgraf z objektu Movie. Keďže sa jedná o jeden typ vrcholu nie je potrebný iný vizualizačný algoritmus ako force-collide. Taktiež zobrazujeme aj hrany medzi jednotlivými entitami. Tu je dobre viditeľné ako force-collide zanecháva vrcholy, ktoré majú spoločné hrany pri sebe.



Obrázok 39: Databáza Messages v zhlukovanom rozložení

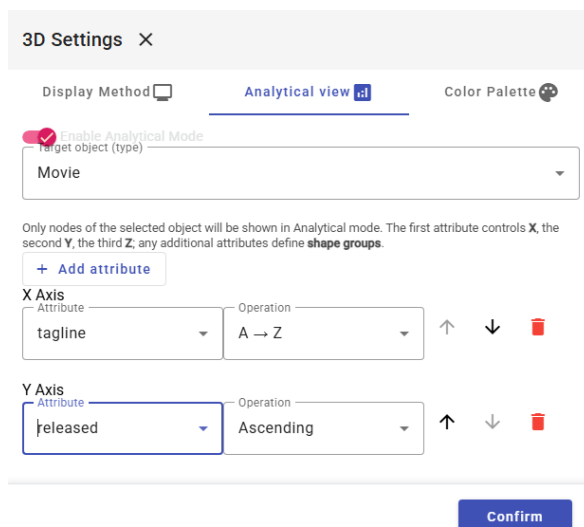
Na obrázku 39 znázorňujeme databázu Messages, ktorý obsahuje 5 typov vrcholov. Zobrazenie znázorňuje efektívny pohľad na vizualizáciu pri väčšom počte vrcholov, Messages obsahuje najväčší počet objektov, ktoré boli testované.

3.4 Filtrované trojdimenzionálne rozloženie



Obrázok 40: Nastavenia analytického režimu

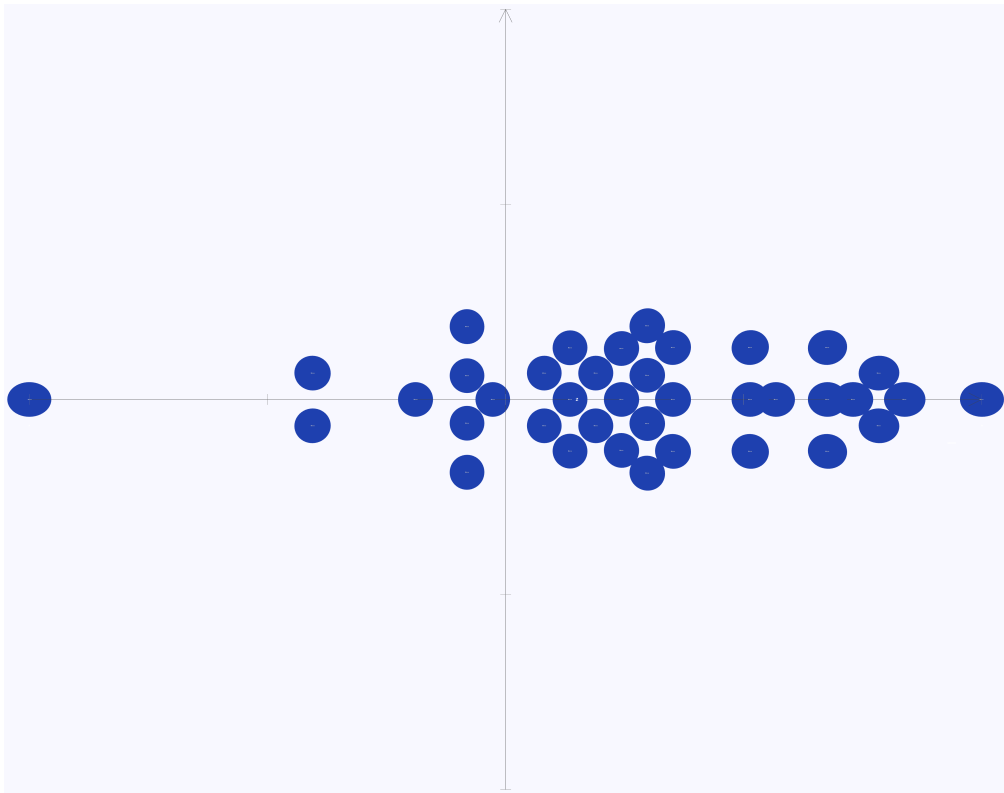
Na obrázku 40 je zobrazené modálne okno na prácu s analytickým režimom. Je najprv potrebné ho zapnúť cez slider *Enable Analytical Mode*, následne si zvolíme objekt na ktorom chceme dané filtrovanie vykonávať.



Obrázok 41: Zapnutie a zvolenie kľúčov na filtrovanie

Na obrázku 41 je viditeľný už zvolený objekt Movie do ktorého sme pridali dva kľúče. Po zvolení kľúčov a kliknutí *confirm* sa vykoná daná operácia.

3.4.1 Filtrovanie s jedným kľúčom

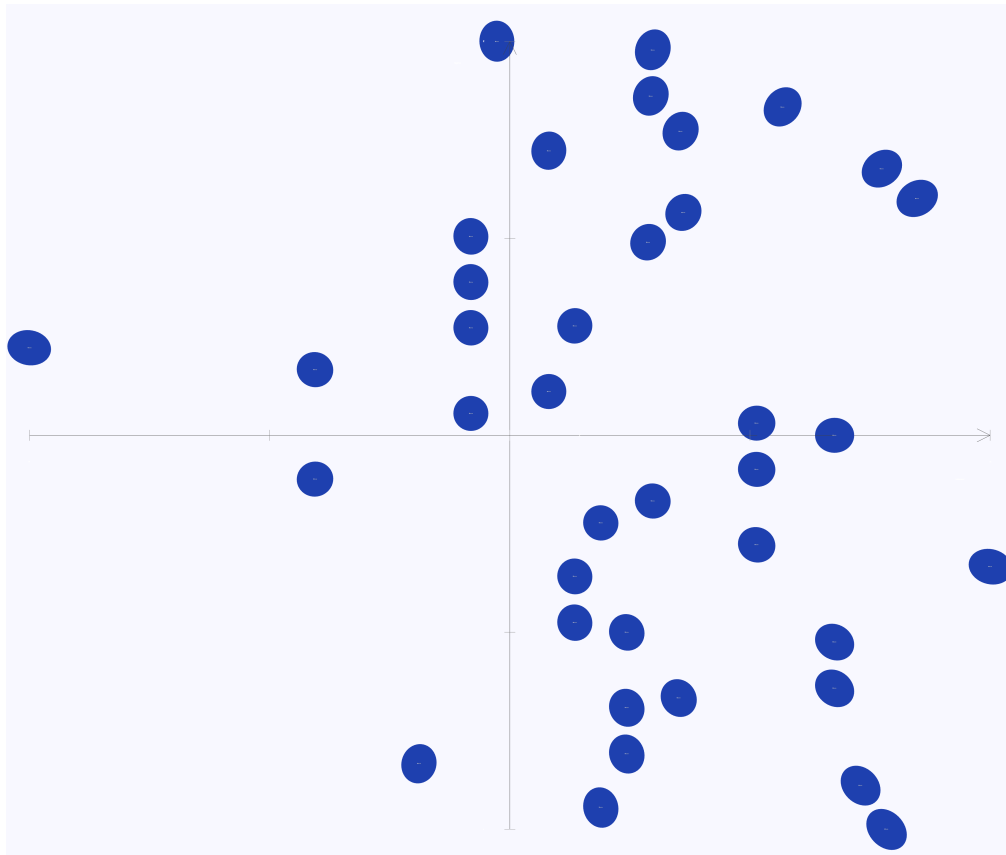


Obrázok 42: Jeden filtrovaný atribút objektu Movie

Na obrázku 42 je zobrazený príklad jednorozmerného filtrovaného rozloženia, kde je ako filter použitý kľúč *rok vydania*. Každému vrcholu sa priradí pozícia na osi X na základe hodnoty tohto kľúča, čím vznikne usporiadanie zľava doprava podľa času. Ľavá časť osi reprezentuje staršie hodnoty (približne 1900 a menej), zatiaľ čo pravá časť predstavuje novšie hodnoty (2000 a viac), takže prechod smerom doprava zodpovedá rastúcemu roku vydania. Zobrazenie zároveň umožňuje rýchlo identifikovať zhluky objektov s podobným rokom vydania, keďže vrcholy s blízkymi hodnotami sa prirodzene zobrazujú v tesnejšej vzdialenosti na osi.

V prípade, že viacero objektov nadobúda rovnakú hodnotu filtrovaného kľúča (t.j. rovnaký rok vydania), všetky by sa teoreticky premietli na tú istú súradnicu na osi X . Aby sa predišlo ich úplnému prekrytiu, aplikácia v takomto prípade rozkladá uzly s rovnakou hodnotou kolmo na os X , takže sa zobrazia „nad sebou“ (s malým posunom v smere Y). Tento mechanizmus zabezpečuje, že aj pri zhodných rokoch zostanú uzly vizuálne rozlíšiteľné a používateľ ich môže samostatne identifikovať a vyberať.

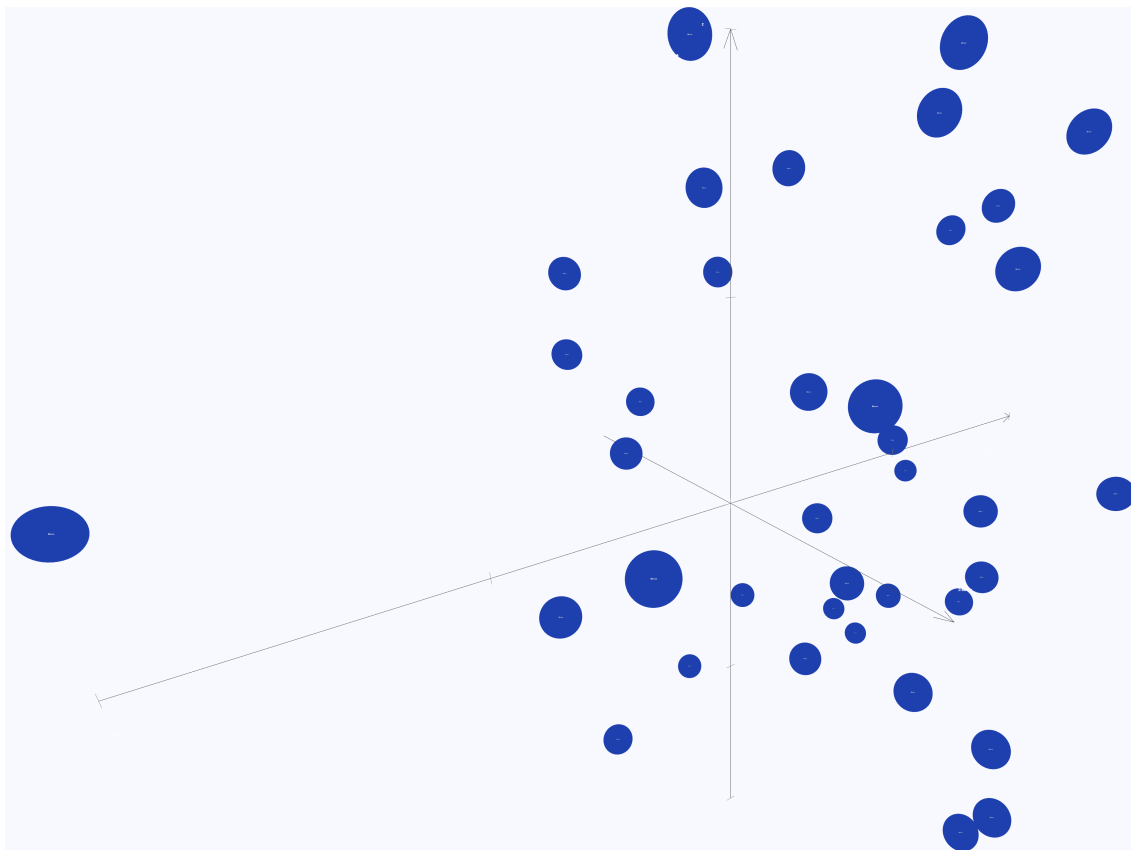
3.4.2 Filtrovanie s dvoma kľúčmi



Obrázok 43: Dva filtrované kľúče objektu Movie

Na obrázku 43 je zobrazené dvojfiltrové rozloženie v rovine XY , kde je každý vrchol umiestnený podľa dvojice zvolených kľúčov. Hodnota prvého filtra určuje pozíciu na osi X a hodnota druhého filtra pozíciu na osi Y , takže výsledné zobrazenie má charakter bodového grafu, v ktorom poloha vrchola vyjadruje kombináciu oboch metrík. Takýto pohľad umožňuje rýchlo porovnávať objekty podľa dvoch kritérií súčasne a identifikovať zhluky (vrcholy s podobnými hodnotami) alebo odlahlé hodnoty, ktoré sa od väčšiny dát výrazne odlišujú. V prípade zhodných hodnôt v jednej z osí sa vrcholy môžu ukladať do stĺpcov alebo radov, čo zvyšuje prehľadnosť a zároveň zachováva rozlíšiteľnosť jednotlivých entít. V prípade že dva kľúče sú identické a tým pádom sú ich súradnice rovnaké sa jeden z objektov posunie o pár pixelov po X osi.

3.4.3 Filtrovanie s tromi kľúčmi



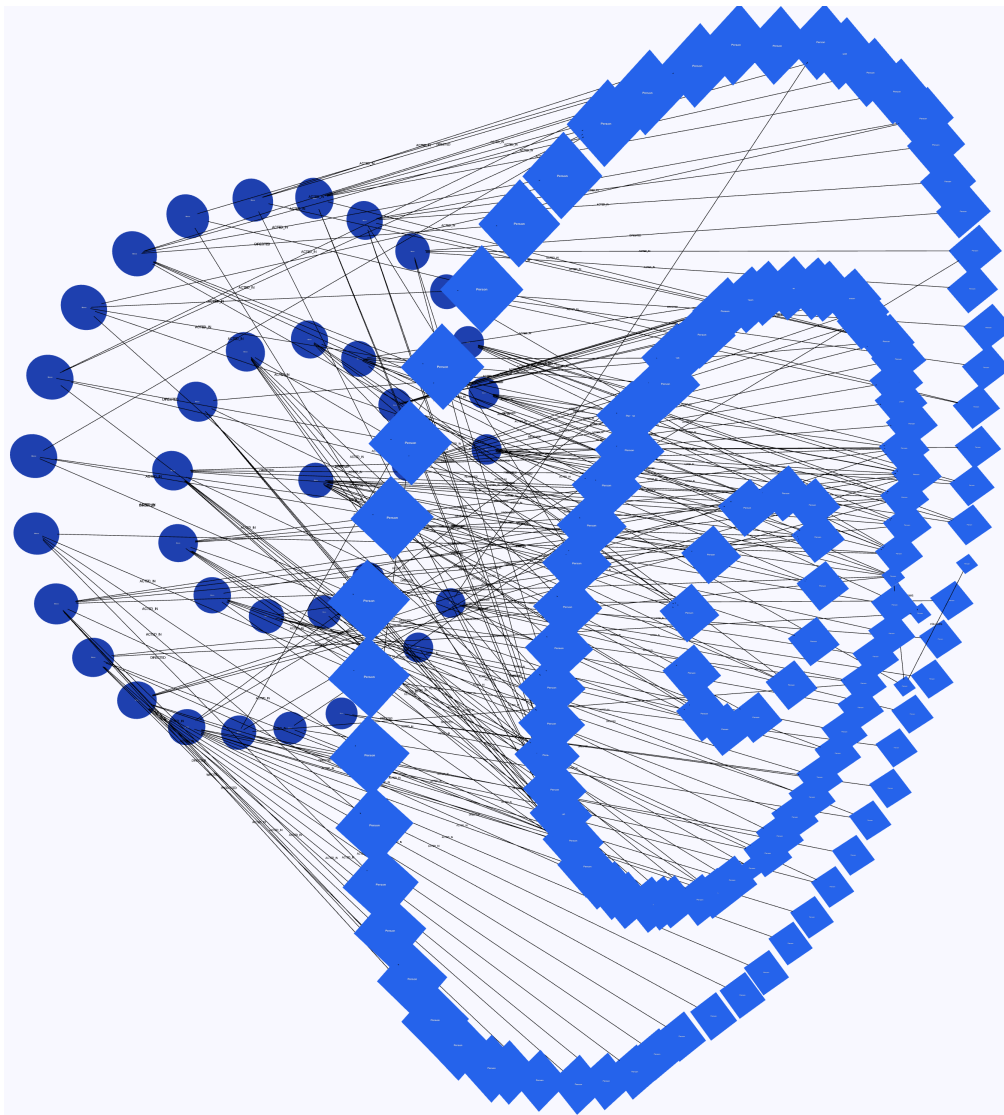
Obrázok 44: Tri filtrované kľúče objektu Movie

Na obrázku 44 je zobrazené filtrovanie v troch dimenziách, kde sú vrcholy umiestnené v priestore XYZ podľa trojice zvolených kľúčov. Každý vrchol má priradené súradnice (x,y,z) , pričom os X reprezentuje prvý filter, os Y druhý filter a os Z tretí filter, takže poloha objektu priamo vyjadruje kombináciu troch metrík. Vďaka perspektívnemu zobrazeniu je možné pri rotácii a priblížení odhaliť priestorové vzory, ktoré by v čisto 2D pohľade neboli zreteľné, napríklad rozdelenie dát do viacerých vrstiev alebo súčasnú závislosť medzi tromi kľúčmi.

Z analytického hľadiska má takýto spôsob zobrazenia viacero praktických využití. Umožňuje identifikovať zhľady objektov s podobnými charakteristikami v troch kľúčoch naraz, nájsť odľahlé objekty (outliers) a rozpoznať trendy, napríklad situácie, keď zmena jednej metriky systematicky súvisí so zmenou ďalších dvoch. V praxi sa tento režim hodí aj na porovnávanie skupín podľa viacerých kritérií a na rýchlu vizuálnu kontrolu kvality dát, keďže neštandardné alebo extrémne hodnoty sa často prejavujú ako izolované body mimo hlavných zhľadov. Takéto 3D filtrovanie preto slúži ako doplnok ku klasickému dopytovaniu: namiesto vyhľadania jednej konkrétnej množiny vrcholov umožňuje vykonať priestorovú exploráciu dát a následne sa zamerať na vybranú časť scény, ktorá je z pohľadu analýzy najzaujímavejšia.

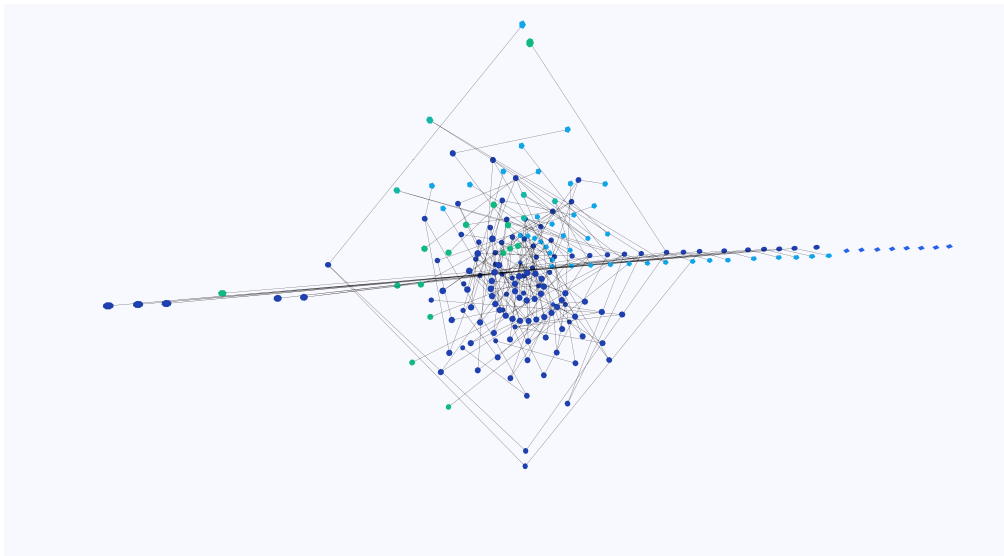
3.5 Sub-zhlukované trojdimenzionálne rozloženie

V ďalšej časti experimentálneho overenia budú na konkrétnych obrázkoch demonštrované vlastnosti sub-zhlukovaného rozloženia: vznik prstencov, správanie pri viacerých komponentoch, a porovnanie čitateľnosti s topologickým silovým rozložením najmä pri hustejších podgrafoch.



Obrázok 45: Databáza movies v sub-zhlukovom rozložení

Na obrázku 45 je zobrazená databáza *Movies* v radiálnom BFS rozložení. Uzly sú rozmiestnené do sústredných prstencov, pričom poloha vrchola vzhľadom na stred scény zodpovedá jeho BFS úrovni, teda vzdialenosti od zvoleného koreňového uzla. Vzhľadom na menší rozsah podgrafu a nižšiu hustotu prepojení je výsledné rozloženie prehľadné, prstence sú stabilné a prekrývanie uzlov je len minimálne. Takéto zobrazenie preto poskytuje jednoznačný pohľad na vrstvenie grafu a umožňuje vizuálne rozlíšiť jednotlivé úrovne okolia bez potreby ďalších úprav.



Obrázok 46: Databáza messages v sub-zhlukovom rozložení

V prípade databázy *Messages* viditeľného na obrázku 46 sa už ukazujú limity tohto rozloženia pri väčšom počte vrcholov a pri vyššej hustote prepojení. Hoci BFS stále vytvorí vrstvy, niektoré prstence sú v projekcii výraznejšie natiahnuté a časť grafu sa ťahá do strán, čo znižuje symetriu radiálneho rozloženia a môže zhoršiť čitateľnosť. Ide o prirodzenú imperfekciu danú tým, že uzly v jednej BFS vrstve sú rozmiestnené rovnomerne po kružnici bez ďalšieho optimalizačného kroku, takže pri nerovnomernom počte uzlov v jednotlivých vrstvách alebo pri väzbách naprieč viacerými vrstvami vznikajú vizuálne deformácie. Napriek tomu zobrazenie stále poskytuje užitočný pohľad na vzdialenosti a štruktúru okolia v grafe, najmä pri analýze dosahu vybraného uzla.

3.6 Porovnanie implementovaného nástroja

Pre porovnanie so štandardným prístupom k vizualizácii grafových databáz uvádzame porovnanie implementovaného nástroja s prostredím *Neo4j Desktop*, ktoré predstavuje bežne používané riešenie pri práci s databázou *Neo4j*. Natívne prostredie typicky pozostáva z viacerých častí rozhrania, medzi ktoré patrí vizualizácia grafu, editor aktívneho dopytu, prehľad výsledkov a panel s informáciami o vybranom objekte. Implementovaný nástroj zachováva základné prvky práce s grafovou databázou (dopytovanie a interakciu s vrcholmi), avšak rozširuje možnosti analýzy grafovej štruktúry a mení spôsob prezentácie výsledkov priamo v 3D priestore.

V porovnaní s *Neo4j Desktop* sa rozdiely prejavujú najmä v spôsobe vizualizácie a podpore analýzy. Natívna vizualizácia v *Neo4j Desktop* využíva predovšetkým štandardné topologické rozloženie v 2D priestore, zatiaľ čo implementovaný nástroj pracuje s viacerými režimami rozloženia a využíva trojdimenzionálne zobrazenie vrátane rotácie, priblíženia a priestorového posunu. Funkcionalita dopytovania v jazyku *Cypher* je

v oboch prípadoch zachovaná, avšak náš nástroj ju dopĺňa o asistované formy tvorby dopytov prostredníctvom *Query Builder* a dopytovania v prirodzenom jazyku. Zobrazenie detailov objektov rieši *Neo4j Desktop* cez samostatný informačný panel, kým v našom riešení sa detail zobrazuje priamo v scéne, čím zostáva v kontexte vizualizácie. Implementované riešenie taktiež poskytuje pokročilé analytické zobrazenie ktoré nám umožňuje vrcholy z databázy zobrazit' na osi (x , y , z podľa počtu zvolených atribútov). Toto umožňuje širšiu a efektívnejšiu prácu s databázou a pomáha pri interaktivite a používateľskom zážitku [21].

Vlastnosť	Neo4j	Navrhnutý nástroj
Vizualizácia		
Počet rozložení	1	4
Trojdimenzionálne zobrazenie	×	✓
Interaktivita		
Priblíženie (zoom)	✓	✓
Rotácia priestoru	×	✓
Interakcia s vrcholmi	✓	✓
Dopytovanie		
<i>Cypher</i>	✓	✓
Formuláre (no-code)	×	✓
Prirodzený jazyk	×	✓
Ostatné vlastnosti		
Natívny nástroj	✓	×
Prehľad výsledkov (legendy/súhrn)	✓	×
Podpora štruktúrálnej analýzy	×	✓

Tabuľka 4: Porovnanie vybraných vlastností medzi prostredím *Neo4j* a navrhnutým nástrojom.

V tabuľke 4 je uvedené porovnanie natívneho prostredia *Neo4j* a navrhnutého nástroja z hľadiska vizualizácie, interaktivity a možností dopytovania. Z porovnania je zrejmé, že navrhnuté riešenie rozširuje štandardný prístup najmä o viacero režimov rozloženia, trojdimenzionálne zobrazenie s rotáciou a o doplnkové spôsoby tvorby dopytov (formuláre a prirodzený jazyk). Vďaka tomu poskytuje používateľovi širšie možnosti exploračie a štruktúrálnej analýzy grafu v jednom prostredí. Natívny nástroj má naopak výhodu integrovanej podpory prehľadu výsledkov a legendy, ktoré sú v navrhnutom riešení nahradené odlišným prístupom.

Všetky režimy zobrazenia spoločne umožňujú analyzovať databázu z viacerých uhlov a získať štruktúrlnu spätnú väzbu bez potreby externých analytických nástrojov.

Záver

Cieľom práce bolo navrhnúť, implementovať a experimentálne overiť nástroj na trojdimenzionálnu vizualizáciu grafov nad dátami uloženými v grafovej databáze. V rámci riešenia sme sa zamerali na praktické spracovanie grafových dopytov a ich následnú vizualizáciu v 3D priestore, pričom databázovým systémom použitým v práci bol *Neo4j*. Výsledkom je aplikácia, ktorá dokáže prevziať dáta z databázy, transformovať ich na grafovú štruktúru a zobraziť ich v interaktívnej scéne s podporou navigácie, výberu prvkov a exportu vizualizácie.

V rámci práce bol vytvorený program založený na aplikačnom rámci Angular, renderovaní pomocou *Three.js* a výpočte rozloženia pomocou knižnice *D3 (d3-force-3d)*. Implementácia zahŕňa viacero vizualizačných režimov, ktoré sú vhodné pre rozdielne typy analýz. Predvolený topologický režim využíva silové rozloženie s kolíziami, čím zlepšuje čitateľnosť výsledných podgrafov a znižuje prekryvanie uzlov. Pre komplexnejšie grafy bola navrhnutá a implementovaná agregovaná vrstva zhlukového zobrazenia, ktorá znižuje vizuálnu náročnosť tým, že sumarizuje vrcholy do spoločných reprezentácií. Súčasťou riešenia je aj sub-zhlukové zobrazenie, ktoré zachováva všetky uzly a hrany, avšak zlepšuje prehľadnosť priestorovým zoskupovaním prvkov do logických celkov. Analytický režim následne umožňuje mapovať vrcholy podľa kľúčov na osi *X*, *Y* a *Z*, čím poskytuje kľúčovo orientovaný pohľad na dáta a podporuje porovnávanie objektov podľa viacerých kritérií naraz.

Dôležitou súčasťou nástroja je aj podpora tvorby dopytov. Aplikácia umožňuje ručné zadávanie *Cypher* dopytov, generovanie dopytov cez formulárový *Query Builder* a doplnkovo aj generovanie dopytov z prirodzeného jazyka v režime *LLM Queries*. V kombinácii s interaktívnym 3D zobrazením tak nástroj poskytuje ucelené prostredie, v ktorom používateľ dokáže databázu prehľadávať, analyzovať a vizuálne interpretovať výsledky bez nutnosti prepínania medzi viacerými externými nástrojmi. Experimentálne overenie na vybraných scenároch ukázalo, že implementované rozloženia sú použiteľné a poskytujú zmysluplnú vizuálnu spätnú väzbu pri práci s reálnymi dátami, pričom voľba režimu závisí najmä od veľkosti grafu a cieľa analýzy.

V rámci ďalšieho smerovania práce je možné nástroj rozšíriť najmä v oblastiach použiteľnosti a škálovania. Z analytického hľadiska je možné rozšíriť filtrovaný režim o viac kombinovaných podmienok, interaktívne selekcie a doplnkové metriky, ktoré by zjednotili vyhľadávanie zaujímavých podgrafov. Zároveň je pri prirodzenom jazyku vhodné doplniť pokročilejšiu validáciu a korekciu generovaných dopytov, aby sa zvýšila spoľahlivosť výstupu a znížila potreba manuálnych úprav.

Referencie

- [1] Besta M. *Demystifying Graph Databases: Analysis and Taxonomy of Data Organization, System Designs, and Graph Queries*, volume 56. ACM Computing Surveys, 2023. doi: 10.1145/3604932.
- [2] A. Kleinedler. A. Dudáš. *Effective Visualization of Data Structures in Graph Databases*. *Journal of Image and Graphics*. 2024. doi: 10.18178/JOIG.12.3.283-291.
- [3] Visualizing graph data in amazon neptune notebooks. <https://docs.aws.amazon.com/neptune/latest/userguide/notebooks-visualization.html>, 2026.
- [4] Neptune dokumentácia. <https://docs.aws.amazon.com/neptune/latest/userguide/intro.html>, 2.3.2026.
- [5] Tigergraph dokumentácia. <https://docs.tigergraph.com/home/>, 2.3.2026.
- [6] Neo4j dokumentácia. <https://neo4j.com/docs/>, 2.3.2026.
- [7] Neo4j cloudové riešenie aura. <https://neo4j.com/docs/aura/explore/explore-visual-tour/explore-overview/>, 2.3.2026.
- [8] Neo4j aura dokumentácia. <https://neo4j.com/docs/aura/>, 2.3.2026.
- [9] How to install the neo4j desktop app and connect it to a remote server. <https://www.techrepublic.com/article/install-and-use-neo4j-desktop-app/>, 2.3.2026.
- [10] Dátová vizualizácia v rámci neo4j. <https://neo4j.com/docs/visualize/>, 2.3.2026.
- [11] yworks, data explorer for neo4j. <https://www.yworks.com/products/data-explorer-for-neo4j>, 2026.
- [12] yworks, automatic layout. https://www.yworks.com/assets/images/data-explorer-for-neo4j/feature2_automatic_layout.e7c6fdeca0.png, 2026.
- [13] yworks, heatmap. https://www.yworks.com/assets/images/data-explorer-for-neo4j/feature8_heatmap.adbc007328.png, 2026.
- [14] Gephi. <https://gephi.org/quickstart/>, 2026.
- [15] Visualizing a network dataset using gephi. <https://mdl.library.utoronto.ca/technology/tutorials/visualizing-network-dataset-using-gephi>, 2026.

- [16] Björn Somme Karsten Klein Ulrich Engelke Daniel Keim Falk Schreiber Matthias Kraus, Johannes Fuchs. Immersive analytics with abstract 3d visualizations: A survey. *<https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14430>*, 2021.
- [17] A. Kleinedler. *Vizualizácia štruktúr a údajov v kontexte grafových databáz. Bakalárska práca.* 2024.
- [18] D3-force: Collide force. *<https://d3js.org/d3-force/collide>*, 2026.
- [19] Api overview (authentication). *<https://developers.openai.com/api/reference/overview/>*, 2026.
- [20] Example datasets. *<https://neo4j.com/docs/getting-started/appendix/example-data/>*, 2026.
- [21] Emil Kršák A. Kleinedler, A. Dudáš. Analytical visualization of graph databases in three-dimensional spaces. *International Journal on Informatics Visualization*, 2026.

UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI

FAKULTA PRÍRODNÝCH VIED

EFEKTÍVNA VIZUALIZÁCIA GRAFOVÝCH DATABÁZ V TROJROZMERNOM
PRIESTORE
SYSTÉMOVÁ DOKUMENTÁCIA

3DGraphDatabaseViewer vyžaduje na svoje správne fungovanie nasledujúce systémove požiadavky:

- 8GB pamäti
- Aplikáciu 3DGraphDatabaseViewer
- *Neo4j*(Konzolovú alebo desktopovú verziu)
- Java 17(Kvôli *Neo4J*)

Potrebný softvér na spustenie aplikácie je možné nainštalovať z nasledujúcich odkazov:

- *Neo4j*: <https://neo4j.com/deployment-center/?desktop-gdb> (Odporúčam spustiť cez docker nakoľko *Neo4j Desktop* vyžaduje vyplniť formulár)
- 3DGraphDatabaseViewer: <https://github.com/Eldam804/GraphDatabaseViewer3D>, nakoľko ide o Angular aplikáciu je potrebné daný kód spustiť cez: npm run start.
- Java 17: <https://www.oracle.com/java/technologies/downloads/> (Vyžadované *Neo4j Desktopom*)

UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI

FAKULTA PRÍRODNÝCH VIED

EFEKTÍVNA VIZUALIZÁCIA GRAFOVÝCH DATABÁZ V TROJROZMERNOM
PRIESTORE
POUŽÍVATEĽSKÁ PRÍRUČKA

Príprava dát a spustenie databázy. Pred prvým použitím je potrebné mať spustený *Neo4j* server a pripravená databáza. Na jednoduché otestovanie je možné použiť ukážkové dáta dostupné priamo v prostredí *Neo4j Browser* cez príkaz `:play movies`, ktorý obsahuje krátku interaktívnu ukážku a dopyty na vytvorenie testovacích dát.

Aplikácia je distribuovaná vo forme zdrojového kódu. Po rozbalení balíčka je potrebné najprv nainštalovať závislosti príkazom `npm install`. Následne sa aplikácia spustí príkazom `npm run start`. Po úspešnom spustení sa zobrazí hlavné okno s 3D vizualizáciou a hornou navigačnou lištou, ktorá obsahuje dostupné nástroje a režimy práce.

Pripojenie k databáze. Pripojenie sa realizuje cez tlačidlo *Login to DB* (resp. *Database Connection*). Používateľ najprv overí dostupnosť databázy zadaním URL v tvare `bolt://localhost:7687` a kliknutím na *Check*. Následne zadá prihlasovacie údaje (meno a heslo) a potvrdí prihlásenie. Po úspešnom pripojení aplikácia načíta dáta z databázy a zobrazí ich vo vizualizéri.

Režimy zobrazovania. V aplikácii sú dostupné viaceré režimy vizualizácie, ktoré umožňujú zvoliť vhodný spôsob interpretácie grafu podľa cieľa analýzy:

- **Topologické zobrazovanie** (*Force*). Predvolené rozloženie založené na silovej simulácii, ktoré sa snaží minimalizovať prekrytie uzlov a zachovať topológiu vzťahov.
- **Zhlukované zobrazovanie** (*Clustering*). Zjednodušené zobrazovanie, v ktorom sa uzly zoskupia podľa typu do agregovaných vrcholov; hrany medzi nimi reprezentujú súhrnné počty vzťahov. Kliknutím na zhuk sa zobrazí podgraf patriaci k zvolenému typu.
- **Sub-zhlukové zobrazovanie** (*Radial BFS*). Rozloženie založené na BFS, kde sú uzly usporiadané do sústredných vrstiev podľa vzdialenosti.
- **Filtrované (analytické) zobrazovanie** (*Data analytics*). Zobrazovanie určené na kľúčovú analýzu, kde sa uzly mapujú na osi X, Y, Z podľa zvolených atribútov (1D/2D/3D) a umožňujú vizuálne porovnávanie objektov podľa hodnôt filtrov.

Dopytovanie a práca s výsledkami. Dopyty sa zadávajú cez tlačidlo *Queries*, ktoré otvorí modálne okno pre písanie a odosielanie *Cypher* príkazov. Pri syntakticky korektnom dopyte sa okno uzavrie a výsledok sa zobrazí v 3D vizualizéri. V prípade chyby zostáva okno otvorené a používateľ môže dopyt upraviť. Aplikácia zároveň podporuje asistované vytváranie dopytov cez *Query Builder* a generovanie dopytov z prirodzeného jazyka v režime *LLM Queries*, čo predstavuje rozšírenie určené na zjednodušenie práce s jazykom *Cypher*.

Interakcia so scénou a detail uzlov. Pohyb v scéne je možné realizovať myšou (rotácia, priblíženie, posun) a klávesnicou pomocou WASD. V klasickom zobrazovaní kliknutie na uzol zobrazí jeho detail priamo v scéne vo forme informačného panelu s kľúčmi.