|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Теоретическая информатика и компьютерные технологии

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ***

***НА ТЕМУ:***

Веб-ориентированное образовательное приложение для изучения графовой системы управления базами данных

Студент \_\_\_ИУ9-82Б\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_**Белякова С.С.**\_\_\_\_**

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель ВКР **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Консультант **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Консультант **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Нормоконтролер **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

*2024 г.*

# АННОТАЦИЯ

Темой выпускной квалификационной работы является «Веб-ориентированное образовательное приложение для изучения графовой системы управления базами данных». Объем дипломной работы составляет 56 страниц.

Целью настоящей работы является проектирование и реализация веб-ориентированного образовательного приложения для изучения графовой системы управления базами данных с реализацией 3D-визуализации данных, представленных в виде графов.

Для реализации поставленной цели сформулированы следующие задачи:

1. Обзор предметной области и анализ существующих решений. Выполнить выбор системы управления базами данных, далее рассмотреть выбранную графовую СУБД, её язык запросов и существующие средства для обучения и визуализации.
2. Спроектировать приложение и выбрать для его реализации подходящий стек технологий. Для этого необходимо сформулировать функциональные и нефункциональные требования к приложению, продумать архитектуру приложения и проанализировать подходящие программные средства для его реализации.
3. Для создания визуализации графовых данных необходимо выбрать целесообразный алгоритм укладки графов.
4. Реализовать веб-ориентированную информационную систему изучения и управления графовой базой данных. Необходимо продемонстрировать работу программы и протестировать её функциональность. Разработать пользовательскую документацию.

# СОДЕРЖАНИЕ

[АННОТАЦИЯ 2](#_Toc169504382)

[СОДЕРЖАНИЕ 3](#_Toc169504383)

[ВВЕДЕНИЕ 4](#_Toc169504384)

[1. Обзор предметной области 6](#_Toc169504385)

[1.1 NoSQL и графовые базы данных 6](#_Toc169504386)

[1.2 Система управления базами данных Neo4j и язык Cypher 8](#_Toc169504387)

[2. Анализ существующих решений 11](#_Toc169504388)

[2.1 Анализ существующих обучающих Neo4j источников 11](#_Toc169504389)

[2.2 Анализ существующих решений визуализации данных 12](#_Toc169504390)

[3. Стек технологий 14](#_Toc169504391)

[3.1 Язык программирования Java, фреймворки и библиотеки Java 14](#_Toc169504392)

[3.2 Библиотеки для языка программирования JavaScript 16](#_Toc169504393)

[4. Проектирование приложения 18](#_Toc169504394)

[4.1 Функциональные и нефункциональные требования к приложению 18](#_Toc169504395)

[4.2 Архитектура приложения 19](#_Toc169504396)

[5. Выбор алгоритма укладки графов 21](#_Toc169504397)

[5.1 Алгоритм Идеса 21](#_Toc169504398)

[5.2 Алгоритм Фрюхтермана-Рейнгольда 22](#_Toc169504399)

[5.3 Алгоритм GEM 24](#_Toc169504400)

[5.4 Магнитно-пружинный алгоритм 27](#_Toc169504401)

[5.5 Алгоритм Камады-Кавай 29](#_Toc169504402)

[5.6 Выбор подходящего алгоритма 31](#_Toc169504403)

[6. Реализация приложения 32](#_Toc169504404)

[6.1 Серверная часть приложения 32](#_Toc169504405)

[6.2 Клиентская часть приложения 34](#_Toc169504406)

[7. Работа программы 36](#_Toc169504407)

[8. Тестирование приложения 41](#_Toc169504408)

[9. Инструкция пользователя 42](#_Toc169504409)

[9.1 Описание программных средств, необходимых для установки и запуска программы 42](#_Toc169504410)

[9.2 Руководство пользователя по использованию обучающей программы 43](#_Toc169504411)

[ЗАКЛЮЧЕНИЕ 48](#_Toc169504412)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 49](#_Toc169504413)

[ПРИЛОЖЕНИЕ А 51](#_Toc169504414)

# ВВЕДЕНИЕ

В настоящее время для хранения, обработки и использования данных широко используются различные базы данных. Современные системы управления базами данных постоянно совершенствуются и развиваются. Разработчикам стало не хватать для реализации своих целей стандартных реляционных баз данных, поэтому начали появляться различные виды нереляционных баз данных, каждая из которых представляла некую модель данных и подходила для решения узкого класса задач. Примером таких являются графовые базы данных. По мере использования и развития графовых баз данных, они получают все большее применение.

К сожалению, из-за относительной молодости данной области, существует мало практических примеров реализации задач с применением графовых баз данных, предназначенных для более эффективного изучения графовых систем управления базами данных. Так как современное образование становится все более ориентированным на веб-технологии, было принято решение спроектировать и реализовать веб-ориентированное образовательное приложение для изучения графовой системы управления базами данных. Для удобства восприятия информации, получаемой в ответ на запросы пользователей, актуальна также реализация 3D-визуализации данных, представленных в виде графов, которая позволит пользователю с различных ракурсов изучить полученный им ответ. Данное приложение может предоставить возможность студентам, профессионалам и просто интересующимся освоить указанные системы управления базами данных в удобной и интерактивной форме.

Целью настоящей работы является проектирование и реализация веб-ориентированного образовательного приложения для изучения графовой системы управления базами данных с реализацией 3D-визуализации данных, представленных в виде графов.

Задачами данной работы являются:

1. Обзор предметной области и анализ существующих решений. Выполнить выбор системы управления базами данных, далее рассмотреть выбранную графовую СУБД, её язык запросов и существующие средства для обучения и визуализации.
2. Спроектировать приложение и выбрать для его реализации подходящий стек технологий. Для этого необходимо сформулировать функциональные и нефункциональные требования к приложению, продумать архитектуру приложения и проанализировать подходящие программные средства для его реализации.
3. Для создания визуализации графовых данных необходимо выбрать целесообразный алгоритм укладки графов.
4. Реализовать веб-ориентированную информационную систему изучения и управления графовой базой данных. Необходимо продемонстрировать работу программы и протестировать её функциональность. Разработать пользовательскую документацию.

## 1. Обзор предметной области

1.1 NoSQL и графовые базы данных

База данных – это структурированное хранилище информации, которое обычно управляется специализированным программным обеспечением, известным как система управления базами данных (далее – СУБД). Базы данных делятся на реляционные и нереляционные в зависимости от способа хранения информации.

Реляционные базы данных основаны на реляционной модели – табличном способе представления данных. Строками таблиц являются сущности, а столбцами – атрибуты сущностей. Для взаимодействия между пользователями и реляционной базой данных СУБД используют язык структурированных запросов SQL.

Нереляционные базы данных, например, NoSQL (Not only SQL) базы данных – семейство баз данных, не являющихся реляционными. NoSQL базы данных обычно разработаны для работы с распределенными, масштабируемыми и неструктурированными данными, что делает их особенно полезными для современных приложений, таких как социальные сети, анализ и хранение больших данных, интернета вещей (IoT) и др. Основными видами NoSQL баз данных являются:

1. Документоориентированные базы данных, такие как MongoDB, Couchbase, Firebase. Документоориентированные базы данных состоят из коллекций, которые, в свою очередь состоят из документов. Документы состоят из полей, а также могут содержать другие документы.
2. Базы данных «ключ-значение», например, Redis, Memcached, Etcd. Особенностью является отсутствие структур и связей. Ключом является название поля документа, а значением – информация, носимая данным полем.
3. Графовые базы данных – Neo4j, OrientDB, Dgraph, ArangoDB. В базах данных такого типа совокупность данных представлена в виде ориентированных графов, узлами являются экземпляры объектов, а ребрами – связи между ними.
4. Колоночные базы данных, например, Cassandra, Snowflake, Redshift. Данные в этих базах данных группируются не по строкам, а по столбцам. Каждая строка состоит из одного или нескольких столбцов, содержащих данные, а каждый столбец из разных строк.

Для дальнейшей работы и изучения были выбраны графовые базы данных. Графовые базы данных – NoSQL базы данных, специально реализованы для представления данных в виде графовых структур. Узлами являются экземпляры или сущности данных, которые являются отслеживаемым объектом. Для отображения взаимосвязи между узлами используются направленные рёбра. Описательную информацию, связанную с ребрами и узлами, называют свойствами.

В то время, как реляционные базы данных используют структурированный подход к данным, графовые базы более гибкие и благодаря своему устройству легко изменяемые. Они уделяют больше внимания отношениям между данными, делая моделирование сложных связей проще.

Безусловно, графовые базы данных не создавались для замены реляционных баз данных. Графовые базы данных в текущий момент используются для решения разных задач. Далее приводятся некоторые примеры областей, в которых графовые базы данных превосходят прочие методы моделирования данных:

* социальные сети;
* рекомендательные сервисы;
* контекстно-зависимые сервисы;
* выявление мошенничества и др.

1.2 Система управления базами данных Neo4j и язык Cypher

Среди всех графовых СУБД более всего выделяется Neo4j, так как она имеет ряд значительных преимуществ перед остальными:

* полная функциональность, соответствие требованиям ACID и встроенный собственный язык запросов Cypher;
* является лидером рынка (имеет наибольшую базу пользователей);
* проста для изучения начинающим разработчикам, благодаря подробной документации;
* существуют различные версии для различных операционных систем;
* имеет высокую производительность, даже при значительном увеличении объема данных.

В силу вышеизложенных фактов в качестве СУБД для реализации цифрового образовательного приложения была выбрана графовая база данных Neo4j.

Для написания запросов в Neo4j используется язык Cypher. Cypher – декларативный графовый язык запросов, позволяющий писать выразительные и эффективные запросы к графовым данным. Данный язык сфокусирован на том, чтобы при написании запросов пользователь представлял какую информацию ему необходимо получить из базы данных, а не на том, как это сделать. Благодаря этому оптимизация запросов является деталью выполнения, а не обременяет пользователя. Cypher вдохновлен языками SQL, SPARQL, Haskell и построен на устоявшейся практике выразительных запросов. Cypher – самый простой графовый язык для изучения из-за его сходства с другими языками и интуитивно понятном синтаксисе. Рассмотрим особенности и синтаксис языка Cypher подробнее.

Язык Cypher обеспечивает поддержку различных типов данных:

* простые типы, такие как числа, строки, логический тип, дата и время;
* cтруктурные типы, например, идентификаторы узлов и связей, метки;
* cоставные типы – списки (коллекции значений любого типа), карты (коллекции пар «ключ-значение»);

Программа, написанная на языке Cypher, состоит из одного или нескольких запросов. Запросы содержат в себе секции, объединённые в цепочку и представляющие собой смесь из ключевых слов и пользовательских переменных. Каждый запрос, как и каждая секция начинаются с ключевого слова. После них следует описание шаблона данных, состоящего из определений узлов и отношений. Узел описывается так, как показано на рисунке 1. Отношение описывается похожим образом, за исключением того, что находится в квадратных скобках. Узел представляет собой объект базы данных, хранящий информацию и характеризующийся меткой (типом) и атрибутами. Отношения (рёбра), как и узлы, характеризуются метками и атрибутами, при этом они имеют направление. Метки обозначают некую совокупность узлов, обычно обозначающих принадлежность узла к некоторому объекту реального мира. Для обращения к узлам и рёбрам, заданным в запросе шаблоном, используются переменные, которые пользователь может в дальнейшем использовать в том же запросе.



Рисунок 1 – Структура описания узла в шаблоне

Чаще всего запросы начинаются со слов MATCH, RETURN, CREATE, DELETE, MERGE, рассмотрим их подробнее.

1. MATCH – ключевое слово языка Cypher, используемое для поиска данных в графе. Благодаря нему можно найти определённый узел, закономерности среди узлов, все узлы с определённой связью и т.д. Обычно данное ключевое слово ставится в начале запроса.
2. RETURN используется для вывода результата запроса к которому относится, это может быть как часть графа, так и системная информация. Обычно это ключевое слово присутствует в конце запроса.
3. CREATE – оператор создания новых узлов, отношений и узлов с отношениями, то есть добавляет в базу данных все, что описывает следующий после ключевого слова шаблон (при добавлении отношения до секции CREATE идёт секция MATCH для поиска узлов, между которыми необходимо создать связь).
4. DELETE – ключевое слово, используемое для удаления информации из базы данных. Перед данной секцией всегда присутствует секция MATCH, для предоставления информации Neo4j о том, какую часть графа необходимо стереть. DELETE имеет ограничение – не может удалить узел, имеющий хотя бы одно отношение.
5. MERGE действует как комбинация операций MATCH и CREATE, сначала проверяет, что такие данные не присутствуют в графе, а потом создает их. Если вместо MERGE использовать CREATE, в базе данных могут возникнуть дубликаты.

Как и в SQL, в Cypher возможно множество действий, как например следующие:

* создание ограничений;
* наложение условий на запросы;
* создание ключей и индексов;
* обновление и сортировка данных;
* встроенные функции (скалярные и агрегатные);
* объединение результатов нескольких запросов.

Кроме этого, Neo4j использует библиотеку графовых алгоритмов Graph Data Science Library для решения задач из теории графов. Данная библиотека содержит реализованные версии методов теории графов, доступных в виде процедур Cypher. Среди них присутствуют:

1. Методы центральности и ранжирования (методы промежуточной и гармоничной центральности, методы ранжирования PageRank и ArticleRank).
2. Методы кластеризации (метод помечивания, метод подсчета треугольников и др.).
3. Методы нахождения пути (метод нахождения кратчайшего пути, метод определения минимального остовного дерева, метод нахождения случайного пути, метод нахождения кратчайшего пути для всех пар и др.).

## 2. Анализ существующих решений

2.1 Анализ существующих обучающих Neo4j источников

В контексте данного приложения происходит столкновение с проблемой отсутствия на рынке приложений для практического изучения СУБД Neo4j. Однако, существуют информационные ресурсы, которые частично исполняют функции (реализующие либо только теоретический аспект обучения, либо предоставляющие практические примеры, но недоступные пользователям) информационные ресурсы:

1. Официальная документация доступна на официальном сайте Neo4j [1], подробно описывает все теоретические аспекты самой СУБД и языка Cypher. Минусами данного ресурса являются отсутствие практических задач, среды выполнения кода, а также отсутствие перевода на русский язык.
2. Учебных пособий как в электронном, так и в бумажном виде, существует значительное количество, в том числе и от разработчиков Neo4j. Недостатками электронных бесплатных вариантов являются те же, что и у официальной документации, но в отличие от которой присутствуют учебные пособия на русском, например, учебное пособие «Системы графовых баз данных. Neo4j» [2]. Среди платных пособий уже присутствует книга с практическими задачами - «Practical Neo4j», автора Грега Джордана [3], но её главными минусами является отсутствие перевода на русский язык и невозможность ее приобретения в России.

2.2 Анализ существующих решений визуализации данных

Важным аспектом любого учебного процесса является наглядность излагаемого материала, возможность пользователя самому что-то попробовать и посмотреть работоспособность. Поэтому, при проектировании данного приложения было принято решение визуализировать полученные пользователем ответы от базы данных. Рассмотрим существующие решения, предназначенные для визуализаций графов:

1. Для отображения графов в 2D существует множество программных решений, в том числе и персонально для визуализации графов баз данных из Neo4j. Приведём пару самых известных примеров, которые возможно интегрировать в проектируемое приложение:

* D3.js [4] – библиотека языка программирования JavaScript, предназначенная для динамической интерактивной визуализации информации в браузерах. Данная библиотека прекрасно подходит для 2D-визуализации ориентированных графов, предоставляемых Neo4j, в ней реализовано несколько алгоритмов укладки для отображения графов.
* Cytoscape.js [5] – библиотека языка JavaScript, вмещающая в себя множество методов теории графов. Предназначена для анализа и визуализации графов. Библиотека предлагает различные алгоритмы компоновки графов и широкий набор инструментов стилизации визуализации.
* Graphviz [6] – программное обеспечение для автоматической визуализации графов, заданных в виде описания на языке DOT. Достаточно распространенное средство для визуализации различных блок-схем и графовых данных для учебных целей.

1. Решения 3D-визуализации графов значительно менее популярны и реже встречаются, чем 2D-визуализация по многим причинам, поэтому готовых решений, тем более таких, которые возможно использовать в стороннем приложении значительно меньше:

* Cy3D [7] – приложение для визуализации различных сетей, основанное на Cytoscape.js. Данное средство визуализации расширяет возможности взаимодействия с графом, позволяя пользователям «двигаться» вокруг графа, изучая его узлы и связи.

Рассмотрев и проанализировав существующие средства визуализации графов было принято решение самостоятельно реализовать алгоритм 3D-визуализации, подходящий для решения задач данной дипломной работы. В результате была выбрана именно 3D-визуализация, так как в Neo4j данные представлены в виде ориентированного графа, каждый узел и ребро которого имеет информацию, которую для наглядности лучше отображать в 3D-режиме. Однако при среднем размере графа (более 20 узлов), данные графовых составляющих просто сольются друг с другом и будет трудно воспринимать визуализируемую информацию. В 3D-варианте данный минус исчезает, так как при корректном выборе алгоритма укладки графа и реализованном «движении» вокруг графа, пользователь сможет рассмотреть всю необходимую для себя информацию. К сожалению, приложение Cy3D более не поддерживается создателями, работает не на всех платформах и предоставляет избыточный функционал для проектируемого приложения. Поэтому было принято решение о самостоятельной реализации 3D-визуализации графов.

## 3. Стек технологий

Для реализации веб-ориентированного образовательного приложения для изучения графовой СУБД Neo4j был выбран следующий стек технологий:

1. Для написания базы данных, предназначенной для получения запросов пользователя и ответа на них, был выбран язык Cypher и СУБД Neo4j по уже рассмотренным выше причинам.
2. Серверная часть приложения будет написана на языке программирования Java [8]. Для интеграции с базой данных будет использован Neo4j JDBC драйвер [9]. Взаимодействие клиента с сервером приложения будет обеспечиваться с помощью фреймворка Spring [10]. Также на сервере будет производиться вычисление укладки графа, предоставляемого в ответ пользователю от базы данных.
3. Клиентская часть приложения будет написана с использованием языка программирования JavaScript [11], а также языка разметки гипертекста HTML и каскадных таблиц стилей CSS. Вдобавок, в данной секции приложения будет использоваться графическая библиотека для отрисовки поступившего от сервера размещения графа, выбор которой будет произведён ниже.

В дальнейших секциях указанные выше средства будут разобраны подробнее.

3.1 Язык программирования Java, фреймворки и библиотеки Java

Java – универсальный строго типизированный объектно-ориентированный язык программирования. Механизм работы языка следующий – исходный текст программы транслируется в байт-код, обрабатываемый виртуальной машиной Java (JVM). Для данного процесса не важно, какая операционная система установлена на устройстве, так как реализован принцип WORA, от английского: write once, run anywhere, то есть язык обладает кроссплатформенностью. Благодаря объектно-ориентированности код на языке программирования Java способствует повышению повторного использования кода из-за возможности легкого разбиения программы на модули.

В настоящее время язык программирования Java наиболее востребован в следующих сферах:

* веб-разработка;
* десктопное программное обеспечение;
* приложения для мобильных устройств;
* распределённые системы;

Кроме вышесказанного не менее важными преимуществами Java являются безопасность, надежность и огромное количество дополнительных фреймворков и библиотек.

В данной работе будут использованы следующие фреймворки Neo4j JDBC Driver и Spring Web MVC.

Neo4j JDBC Driver обеспечивает возможность подключения к графовой СУБД Neo4j с использованием Java Database Connectivity (JDBC). Данный драйвер позволяет Java-приложениям осуществлять операции чтения и записи данных в графовую базу данных. Взаимодействие с Neo4j через JDBC Driver позволяет использовать стандартные JDBC-методы для выполнения запросов к базе данных, создания транзакций, управления соединениями и других стандартных операций, которые обычно выполняются при работе с реляционными базами данных.

Spring – набор взаимосвязанных фреймворков, созданных для работы над различными частями приложения. В частности, Spring Web MVC [12] – полноценный веб-фреймворк обеспечивающий архитектуру шаблона Model – View – Controller (модель – вид – контроллер). Данный подход разделяет аспекты приложения, обеспечивая при этом свободную связь между ними, модель объединяет данные приложения, вид отвечает за отображение данных модели, генерируя HTML, видимый пользователю в браузере, контроллер обрабатывает запросы пользователя и передает модель для отображения в вид. Таким образом данный фреймворк позволяет эффективно создавать веб-приложения.

3.2 Библиотеки для языка программирования JavaScript

JavaScript – мультипарадигменный кроссплатформенный язык с динамической типизацией, одновременно поддерживающий несколько стилей: функциональный, императивный и объектно-ориентированный. Программы, реализованные на данном языке программирования, называют скриптами, так как они представляют из себя текст, в котором описаны действия, позволяющие обрабатывать и выполнять команды. В браузере для JavaScript доступно всё, что связано с манипулированием веб-страницами, взаимодействием с пользователем и веб-сервером. JavaScript – единственная браузерная технология, сочетающая в себе полную интеграцию с HTML и CSS, поддержку всеми основными браузерами (включён по умолчанию) и относительную простоту. Кроме этого, JavaScript обладает высокой скоростью, производительностью и мощной инфраструктурой. Таким образом, JavaScript прекрасно подходит для написания веб-клиента приложения.

Для 3D-визуализации данных, получаемых с сервера приложения необходимо рассмотреть и выбрать JavaScript библиотеку, наиболее подходящую под данную задачу.

Рассмотрим наиболее популярные библиотеки для 3D-графики, написанные на языке программирования Javascript:

* Three.js [13]. Библиотека предоставляет простой способ создания 3D-графики веб-приложений. Она основана на WebGL и упрощает работу с ней, предоставляя API для создания и управления 3D-сценами. Используется в масштабных проектах, не требующих проработанной физики.
* Babylon.js [14]. Мощная библиотека, также основанная на WebGL. Предоставляет разработчикам инструменты для создания интерактивных и высокопроизводительных 3D-сцен в браузере. Широко используется для создания игр, виртуальной реальности и различных визуализаций данных.
* A-Frame [15]. Простая и мощная библиотека для создания интерактивных 3D-сцен и виртуальной реальности. Основана на Three.js, имеет HTML-подобный синтаксис, что упрощает работу с ней.
* CesiumJS [16]. Также основана на WebGL, но специализируется на создании 3D-географических браузерных приложениях. Позволяет отображать различные типы геоданных и является интерактивной.

В таблице 1 представлено сравнение приведённых выше библиотек по критериям кроссплатформенности, простоты использования и области применения для выбора наиболее подходящей для задач нашего приложения библиотеки и последующего ее применения.

Таблица 1 – Сравнение JavaScript библиотек 3D визуализации.

|  |  |  |  |
| --- | --- | --- | --- |
| Библиотека\Критерий сравнения | Кроссплат-форменность | Простота использования | Область применения |
| Three.js | + | Имеет средний порог входа из-за широких возможностей создания 3D-графики. | Широко используется для создания интерактивной 3D-графики. |
| Babylon.js | + | Имеет средний порог входа по тем же причинам, что и Three.js. | Преимущественно используется для создания игр и приложений, требующих высокой проработки графики. |

Продолжение таблицы 1

|  |  |  |  |
| --- | --- | --- | --- |
| Библиотека\Критерий сравнения | Кроссплат-форменность | Простота использования | Область применения |
| A-Frame | + | Имеет малый порог входа из-за удобного HTML-подобного синтаксиса. | Используется для создания интерактивной 3D-графики и виртуальной реальности. |
| CesiumJS | + | Имеет высокий порог входа из-за специфики работы с геоданными. | используется для создания географических информационных систем, визуализации геоданных. |

Как видно из таблицы, наиболее подходящими для использования являются библиотеки Three.js и A-Frame. Но, из-за наличия опыта работы с библиотекой A-Frame и перспективы использования в будущем виртуальной реальности в приложении будет использована A-Frame.

## 4. Проектирование приложения

4.1 Функциональные и нефункциональные требования к приложению

Определим функциональные и нефункциональные требования к разрабатываемому приложению.

Функциональные требования:

1. Приложение должно предоставлять практические задания пользователю для обучения работе с системой управления базами данных Neo4j.
2. Приложение должно обеспечивать пользователя необходимым для решения практических задач теоретическим материалом.
3. Должна быть обеспечена возможность передачи запросов к базе данных.
4. Приложение должно корректно обрабатывать любые полученные от базы данных результаты пользовательского запроса.
5. Клиентская часть приложения должна корректно визуализировать различные полученные от серверной части данные в 3D-виде.
6. После запуска приложение не должно требовать никаких дополнительных мер для корректного функционирования.
7. Пользователь должен иметь возможность приводить базу данных в исходное состояние.

Нефункциональные требования:

1. Приложение должно реализовывать клиент-серверную архитектуру.
2. Серверная часть приложения должна быть написана на языке программирования Java.
3. Клиентская часть приложения должна использовать библиотеку A-Frame.
4. Для установки приложения не должно быть необходимости в установке локальной базы данных.

4.2 Архитектура приложения

Рассмотрим каждый компонент спроектированной схемы приложения, представленной на рисунке 2, схема была построена с помощью инструмента построения диаграмм [17].

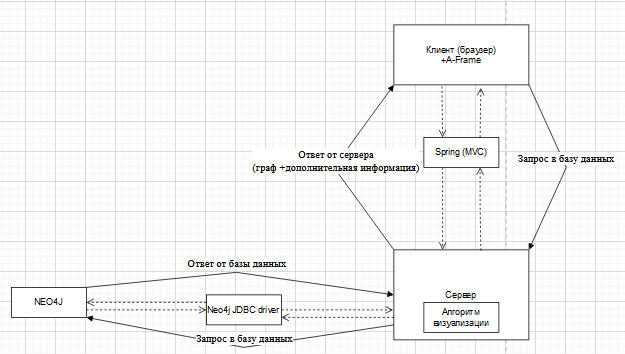


Рисунок 2 – Архитектура приложения

1. Neo4j. Данный компонент представляет собой базу данных для обучения, расположенную на удалённом сервере.
2. Neo4j JDBC driver. Данный компонент необходим для соединения и обмена сообщениями между базой данных и серверной частью приложения.
3. Сервер. Серверная часть приложения отвечает за отправку запросов к базе данных и получение от неё ответа в виде сообщения об ошибке, успешности запроса или графовых данных в JSON-формате. Далее серверу необходимо обработать ответ от базы данных, преобразовать JSON-данные в граф и применить к нему выбранный алгоритм визуализации. После чего, обработанные данные должны быть отправлены клиентской стороне приложения. От клиентской части сервер получает запросы пользователя к базе данных.
4. Spring MVC – веб-фреймворк обеспечивающий архитектуру шаблона Model – View – Controller (Модель – Вид – Контроллер). Данный фреймворк позволяет эффективно соединить клиентскую часть приложения и серверную.
5. Клиент. Клиентская сторона приложения принимает запросы от пользователя, отправляет полученные запросы на серверную часть приложения и получает ответные данные для их отображения и визуализации. Для визуализации используется библиотека A-Frame, принимающая информацию об узлах и ребрах (в том числе и координаты) и отрисовывающая полученные данные в 3D.

## 5. Выбор алгоритма укладки графов

5.1 Алгоритм Идеса

Алгоритм размещения графов, разработанный Питером Идесом и представленный в 1984 году, основан на пружинной модели, где вершины графа заменяются стальными кольцами, а рёбра пружинами. Согласно Идесу [18] эстетичность размещения определяется равной длиной пружин, наиболее возможной симметрией размещения и минимальным количеством пересечений ребер.

На вершины действуют силы притяжения и отталкивания. Сила отталкивания действует между каждой парой не смежных вершин и вычисляется по формуле:

(1)

где и – это координаты вершин u и v;

– коэффициент отталкивания (у Идеса );

– Евклидово расстояние между вершинами, вычисляемое по формуле:

(2)

– единичный вектор направления между вершинами u и v, вычисляемый по формуле:

(3)

Сила притяжения действует между каждой парой смежных вершин и вычисляется по формуле:

ё (4)

где – это коэффициент притяжения (у Идеса );

– «идеальная» длина пружины.

Исходными данными является связный неориентированный граф G = (V, E), где V – множество вершин графа, E – множество рёбер графа. Псевдокод алгоритма представлен в листинге 1, k – количество итераций алгоритма.

Листинг 1 – Псевдокод алгоритма Идеса

for i = 1 to k do:

for v V do:

(i) = -

for v V do:

(i)

Пружинная модель вдохновила на создание ряда модифицированных и расширенных алгоритмов размещения графов, некоторые из которых представлены ниже. Идеи Идеса стали основой для целого класса алгоритмов укладки графов.

5.2 Алгоритм Фрюхтермана-Рейнгольда

Данный алгоритм укладки графов был разработан Томасом Фрюхтерманом и Эдвардом Рейнгольдом и был представлен в 1991 году [19], данный алгоритм основан на предыдущем алгоритме. Критерием эстетичности полученной укладки графа кроме равной длины пружин, наиболее возможной симметрии размещения и минимального количества пересечений ребер является равномерное распределение вершин в кадре.

Одним из отличий от алгоритма Идеса является немного изменённое вычисление сил притяжения и отталкивания. Силы отталкивания в данном алгоритме действуют между каждой парой вершин:

(5)

где – «идеальная» длина пружины, вычисляемая по формуле:

(6)

где W и H – ширина и высота кадра;

|V| - количество вершин в исходном графе.

Силы притяжения все еще действуют только между смежными вершинами:

(7)

Для ускорения сходимости к решению используется упрощенный метод имитации отжига, откуда Фрюхтерман и Рейнгольд взяли идею, что для ускорения минимизации энергии системы (достижения наиболее хорошей укладки графа) необходима высокая начальная температура и длительное время охлаждения.

Псевдокод алгоритма представлен в листинге 2 (объект каждой вершины содержит два вектора – pos (текущую позицию вершины) и disp (смещение данной вершины под действием сил), init\_t – начальное значение температуры, cooling\_coef – коэффициент охлаждения).

Листинг 2 – Псевдокод алгоритма Фрюхтермана-Рейнгольда

t = init\_t

for i = 1 to k do:

for v V do:

v.disp = 0

for u V do:

if (u ≠ v) then

v.disp +=

for e E do:

e.v.disp -=

e.u.disp +=

for v V do:

v.pos +=

v.pos.x = min(max(-, v.pos.x))

v.pos.y = min(max(-, v.pos.y))

t = cooling\_coef\*t

Фрюхтерман и Рейнгольд также предложили способ ускорения своего алгоритма – вычислять силы отталкивания только между близкими друг к другу вершинами, близкими называются вершины, находящиеся друг от друга в радиусе 2. Другими словами, формула для силы отталкивания между двумя вершинами будет выглядеть следующим образом:

(8)

где .

5.3 Алгоритм GEM

Арне Фрик, Андрэ Людвиг и Хейко Мехлдау представили в 1995 году алгоритм GEM (graph embedder) [20]. Кроме уже привычных сил притяжения и отталкивания они ввели силу тяжести, тянущую вершины к общему центру тяжести и ускоряющую скорость сходимости алгоритма.

На вход алгоритм принимает неориентированный граф G = (V, E), где E – множество рёбер графа, а V – множество структур данных, каждая из которых соответствует вершине графа и содержит:

* pos – текущая позиция вершины;
* p – последний сохранённый импульс;
* t – локальная температура данной вершины;
* d – датчик перекоса.

В основном цикле алгоритма, который останавливается либо по достижении желаемой минимальной температуры системы, либо по завершению изначально заданного числа раундов, где раунд – группа итераций происходят следующие шаги:

1. Выбор вершины для обновления. На каждой итерации обновляется только одна вершина, выбранная случайно. Создатели данного алгоритма вычисляли в начале каждого раунда случайную перестановку вершин с помощью алгоритм тасования Фишера-Йетса.
2. Вычисление импульса, который представляет собой последнее движение вершины. На данном этапе происходит вычисление сил притяжения, отталкивания и тяжести, действующих на вершины и обновление импульса вершины. В листинге 3 представлен псевдокод алгоритма нахождения импульса вершины, где B – барицентр системы, Φ(v) – сила тяжести вершины v, deg(v) – степень вершины v, γ – гравитационный фактор, δ – случайный малый вектор:

Листинг 3 – Часть алгоритма GEM, отвечающая за вычисление импульса вершины

с =

B =

Φ(v) =

p = (B – v.pos)γΦ(v)+ δ

for u ∈ V\{v} do:

p +=

for (u, v) ∈ E do:

p -=

1. Обновление позиции вершины и температуры. На данном этапе происходит вычисление новой позиции вершины и угла между старым импульсом вершины и новым. В зависимости от этого угла меняется датчик перекоса и температура вершины. Если угол небольшой, то температура повышается, иначе имеем два варианта – либо вершина вращается на месте (вращением считается случай, когда полученный угол оказывается в секторе вращения, как показано на рисунке 3), либо колеблется (колебанием считается случай, когда предыдущий и новый импульсы направлены в противоположные стороны).

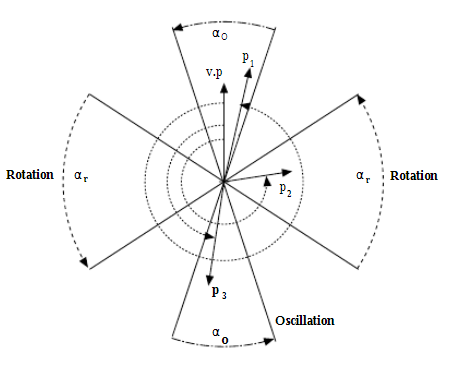


Рисунок 3 – Обнаружение вращений и колебаний

Подробнее, как происходит обновление температуры и позиции вершины описано в псевдокоде обновления позиции вершины и температуры в листинге 4 ( и – углы секторов вращения и колебания, и – чувствительность соответственно к вращению и колебанию).

Листинг 4 – Часть алгоритма GEM, отвечающая за обновление позиции вершины и температуры

p = v.p

if p ≠ 0 then

p =

v.pos += p

c += p

β = ∠(p, v.p)

if sin(β) > sin() then

v.d +=

if |cos(β)| cos() then

v.t = cos(β)

v.t = (1 - |v.d|)

v.t = min(v.t, )

v.p = p

5.4 Магнитно-пружинный алгоритм

Козо Сигуяма и Казуо Мисуе представили в 1995 году алгоритм укладки графов, основанный на магнитно-пружинной модели [21]. В данной модели вершины являются металлическими кольцами, а ребра – магнитными пружинами, также были введены различные типы магнитных полей. К эстетическим критериям, рассмотренным до этого в алгоритме Фрюхтермана-Рейнгольда, они прибавили условие, что ребра полученного размещения графа должны соответствовать заданной ориентации.

В своей работе, Козо и Казуо рассмотрели три типа магнитных полей – параллельное, когда все силы действуют в одном направлении; концентрическое, когда сила действует по концентрическим окружностям и радиальное, когда сила действует радиально из некоторой точки, примеры таких полей представлены на рисунке 4.

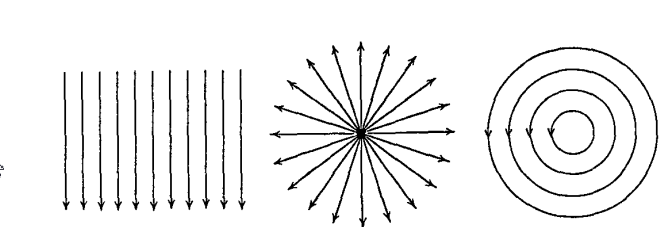


Рисунок 4 – Примеры магнитных полей

Для каждого магнитного поля задаются сила действия магнитного поля, далее коэффициент b, и функция, описывающая ориентацию поля в любой точке, далее m(x,y). Рассмотрим определение m(x,y) для разных типов полей:

1. Параллельное поле представлено формулой 9, где (0, 1) северное направление, (-1, 0) – западное направление, (0, -1) – южное направление, (1, 0) – восточное направление:

m(x,y) = (0, 1)|(-1, 0)|(0, -1)|(1, 0) (9)

1. Радиальное поле представлено формулой:

m(x,y) = (10)

1. Концентрическое поле представлено формулами 11 и 12, где формула 11 отвечает за вращение по часовой стрелке, а формула 12 – против часовой стрелки:

m(x, y) = (11)

m(x, y) = (12)

В данном алгоритме присутствуют три типа сил:

1. Силы притяжения между вершинами, соединёнными ребром. Вычисляются, как в алгоритме Идеса.
2. Силы отталкивания между всеми парами вершин, не соединённых ребрами. Вычисляются, как в алгоритме Идеса.
3. Вращающие силы, действующие на вершины, соединённые ребром согласно направлению магнитного поля. Вычисляются по формуле:

(13)

где , α, β – параметры настройки системы;

θ (-π < θ < π) – угол, между ориентацией поля и направлением ребра (магнитной пружины).

Псевдокод магнитно-пружинного алгоритма отличается от псевдокода алгоритма Идеса следующими положениями:

1. Перед началом алгоритма вершины в случайном порядке распределяются равномерно по кругу радиуса .
2. К сумме сил притяжения и отталкивания прибавляются вращающие магнитные силы.

Таким образом, с помощью данного алгоритма можно не только получить размещение графа, соответствующее общепринятым эстетическим критериям, но и управлять геометрической ориентацией ребер с помощью магнитных вращающих сил. Магнитно-пружинный алгоритм позволяет простым и единым образом представлять не только неориентированные графы, но и деревья, ориентированные и смешанные графы.

5.5 Алгоритм Камады-Кавай

В 1989 году Томихиса Камада и Сатору Кавай представили свой алгоритм размещения графов [22], все так же основанный на пружинной модели, но сильно отличающийся от остальных алгоритмов, описанных ранее. Во-первых, в данной модели используется потенциальная энергия пружины и нет раздельных сил притяжения и отталкивания. Во-вторых, критериями эстетичности получаемой укладки графа являются минимальное количество пересечений ребер и равномерное распределение вершин и ребер. Также, оптимальное расстояние между любыми вершинами (не обязательно соединенными ребром) определяется как кратчайший путь между этими вершинами в графе. Заметим, что поскольку такая цель не всегда достижима, оптимальное размещение находится путем уменьшения энергии системы.

Общая энергия системы определяется по формуле:

(14)

где – сила пружины между вершинами, которая вычислется по формуле:

(15)

где K – заданная константа;

– длина наикратчайшего пути между вершинами.

– желаемая длина пружины между вершинами, где L (длина ребра в квадратном кадре) вычисляется по формуле:

(16)

где – длина стороны квадрата.

Для нахождения наикратчайшего пути между каждой парой вершин можно воспользоваться например алгоритмом Беллмана-Форда(O()). В двухмерном евклидовом пространстве формула энергии системы приобретет вид:

(17)

Цель алгоритма – минимизировать функцию энергии, но вместо поиска минимума, авторы ищут локальный минимум из-за сложностей вычисления. Метод поиска локального минимума, использованный в данном алгоритме был основан на методе Ньютона-Рафсона. Необходимыми условиями существования локального минимума являются равенство нулю всех частных производных, это соответствует решению системы из 2n нелинейных уравнений. Было принято решение за одну итерацию двигать лишь одну вершину, замораживая остальные. Выбиралась вершина с максимальной действующей на неё силой, то есть та, для которой было максимально следующее выражение:

(18)

Далее происходит итерационный процесс, нацеленный на минимизацию , описанный формулами 19, 20:

(19)

(20)

где

Неизвестные и можно найти из следующей системы уравнений:

Изначально вершины располагают в узлах правильного n-образного многоугольника, описанного окружностью диаметром . В листинге 5 представлен псевдокод алгоритма Камады-Кавай.

Листинг 5 – Псевдокод алгоритма Камады-Кавай

for 1 ≤ i ≠ j ≤ |V| do:

compute and

for 1 ≤ i ≤ |V| do:

init

while ( < ε) do:

while ( > ε) do:

compute

5.6 Выбор подходящего алгоритма

Для отображения ответов базы данных на запросы пользователя необходим алгоритм, способный выдать визуально-понятную и привлекательную укладку ориентированного графа. Если граф не связный, его компоненты не должны быть далеко друг от друга. Также важна эффективность и быстрота алгоритма, под быстротой понимается скорость сходимости алгоритма к удовлетворительному размещению графа.

С точки зрения эффективности наименее предпочтительные алгоритмы – алгоритм Камады-Кавай и GEM, так как временная сложность у обоих алгоритмов O().

Если рассматривать приведённые алгоритмы по критерию быстроты сходимости к оптимальной укладке для разных типов графов, наиболее подходящими будут магнитно-пружинный алгоритм и алгоритм GEM, так как в статьях [20] и [21] приводилось сравнение данных алгоритмов с алгоритмами Камады-Кавай и Фрюхтермана-Рейнгольда.

Со стороны «привлекательности» размещения, наиболее к поставленным условиям подходят так же алгоритмы GEM и магнитно-пружинный, но магнитно-пружинный выделяется тем, что учитывает ориентированность графа, а также имеет возможность гибкой настройки с помощью изменения направления и вида магнитного поля.

Исходя из всего вышеперечисленного для дальнейшей реализации был выбран магнитно-пружинный алгоритм.

## 6. Реализация приложения

6.1 Серверная часть приложения

Данный компонент приложения состоит из нескольких классов. Рассмотрим назначение каждого класса:

1. Main. Данный класс отвечает за запуск всего приложения.
2. Service. Следующий класс отвечает за установку соединения с базой данных и обменом сообщениями с ней с помощью Neo4j JDBC driver. В конструкторе класса создаётся объект данного драйвера, используемый в методах класса. Важным методом является sol\_query (String q), представленный в листинге 1 приложения А, принимающий на вход строку запроса, выполняющий этот запрос к базе данных с помощью драйвера и преобразующий полученные данные в сообщение клиентской стороне и граф. После получения данных в виде графа для них вызываются дополнительные вспомогательные методы преобразования информации в том числе и алгоритм укладки графа. Также в данном классе присутствует метод, отвечающий за перезагрузку базы данных, выполняющий все необходимые для этого запросы.
3. Node. Специальный класс, описывающий структуру узлов базы данных (в базе данных для обучения узлами являются абстракции сообществ и людей). Данный класс необходим для корректного преобразования строк ответа базы данных в узлы.
4. Relationship. Специальный класс, описывающий структуру связей базы данных (в базе данных для обучения связями являются абстракции понятий дружбы, ненависти и подписки). Данный класс, как и предыдущий, необходим для корректного преобразования строк ответа базы данных в объекты (в данном случае в ребра).
5. Vertex и Eadge. Данные классы предназначены для представления данных в виде узлов и ребер графа. В Vertex кроме информации о содержании узла базы данных и его текущей позиции, вершина содержит список связанных с ней вершин.
6. Graph. Данный класс предназначен для хранения и обработки данных в виде графа. Кроме вспомогательных методов, таких как, задание цветов у вершин и ребер, получения их атрибутов и добавления новых узлов и ребер к графу (с проверкой на присутствие таких объектов в графе), в данном классе реализован магнитно-пружинный алгоритм. Реализация данного алгоритма и его вспомогательных функций представлена в листингах 2-7 приложения A. Рассмотрим подробнее, как была произведена реализация данного алгоритма:

В данной реализации было использовано концентрическое поле. Для магнитного поля была задана сила действия магнитного поля (далее коэффициент b принимаем равным 8) и функция, описывающая ориентацию поля в любой точке (далее m(x,y) будет вычисляться по формуле:

m(x,y) = ) (21)

Начальным размещением является расположение вершин в случайном порядке равномерно по кругу радиуса . Для вычисления такого размещения был выбран алгоритм, известный как решётка Фибоначчи (или сфера Фибоначчи [23]).

Псевдокод алгоритма представлен в листинге 6 (fibonachi\_disp(G) – начальное размещение графа, δ – корректирующий коэффициент).

Листинг 6 – Псевдокод магнитно-пружинного алгоритма

fibonachi\_disp(G)

for i = 1 to k do:

for v V do:

(i) = + –

for v V do:

+= (i)

1. AppController. Данный класс предназначен для общения клиентской стороны приложения с серверной. В нем обрабатываются различные запросы от клиента и передаётся информация необходимая для отображения в браузере.

6.2 Клиентская часть приложения

Следующий компонент приложения состоит из двух типов web-страниц. Рассмотрим каждую из них поподробнее.

1. Страница с теорией. В таких HTML-файлах представлен, кроме шапки приложения, теоретический материал для пользовательского изучения. Подробнее содержание шапки приложения описано руководстве пользователя по использованию обучающей программы.
2. Страница с задачей. Следующий HTML-шаблон предназначен для отображения любой задачи для пользовательского решения. Данная страница предоставляет пользователю возможность отправлять запросы к обучающей базе данных и получать обработанные серверной стороной приложения ответы, в том числе и визуализированный с помощью библиотеки A-Frame граф. В листинге 7 представлена функция для создания узла в необходимых координатах, с нужным цветом и прикрепленным к нему текстом.

Листинг 7 – Функция создания узла

function make\_node(p, id\_num, hex, text){

const scene = document.querySelector('a-scene');

const sphere = document.createElement('a-sphere');

sphere.setAttribute('id', id\_num);

sphere.setAttribute('radius', 0.15);

sphere.setAttribute('color', hex);

scene.appendChild(sphere); document.getElementById(id\_num).object3D.position.set(p.x, p.y, p.z);

make\_text({x:p.x -0.15, y:p.y, z:p.z}, text, {x: 0, y: -90, z: 0}, false);

make\_text({x:p.x +0.15, y:p.y, z:p.z}, text, {x: 180, y: -90, z: 180}, false);

}

Данная функция создаёт в главном объекте A-Frame (сцене) новую сферу с заданными атрибутами, а также вызывает аналогичную функцию make\_text, создающую текст около сферы. Аналогично работает функция создания направленного ребра, создающая стрелку определённого цвета, соединяющую две сферы и добавляющая к ней подпись, используя ту же функцию make\_text.

## 7. Работа программы

В данном разделе рассмотрим работу приложения, в особенности визуализацию графовых данных, полученных в ответ на различные пользовательские запросы.

На рисунках 5 и 6 представлены примеры теоретических сводок к задачам. В них описывается вся необходимая информация для решения последующих задач.

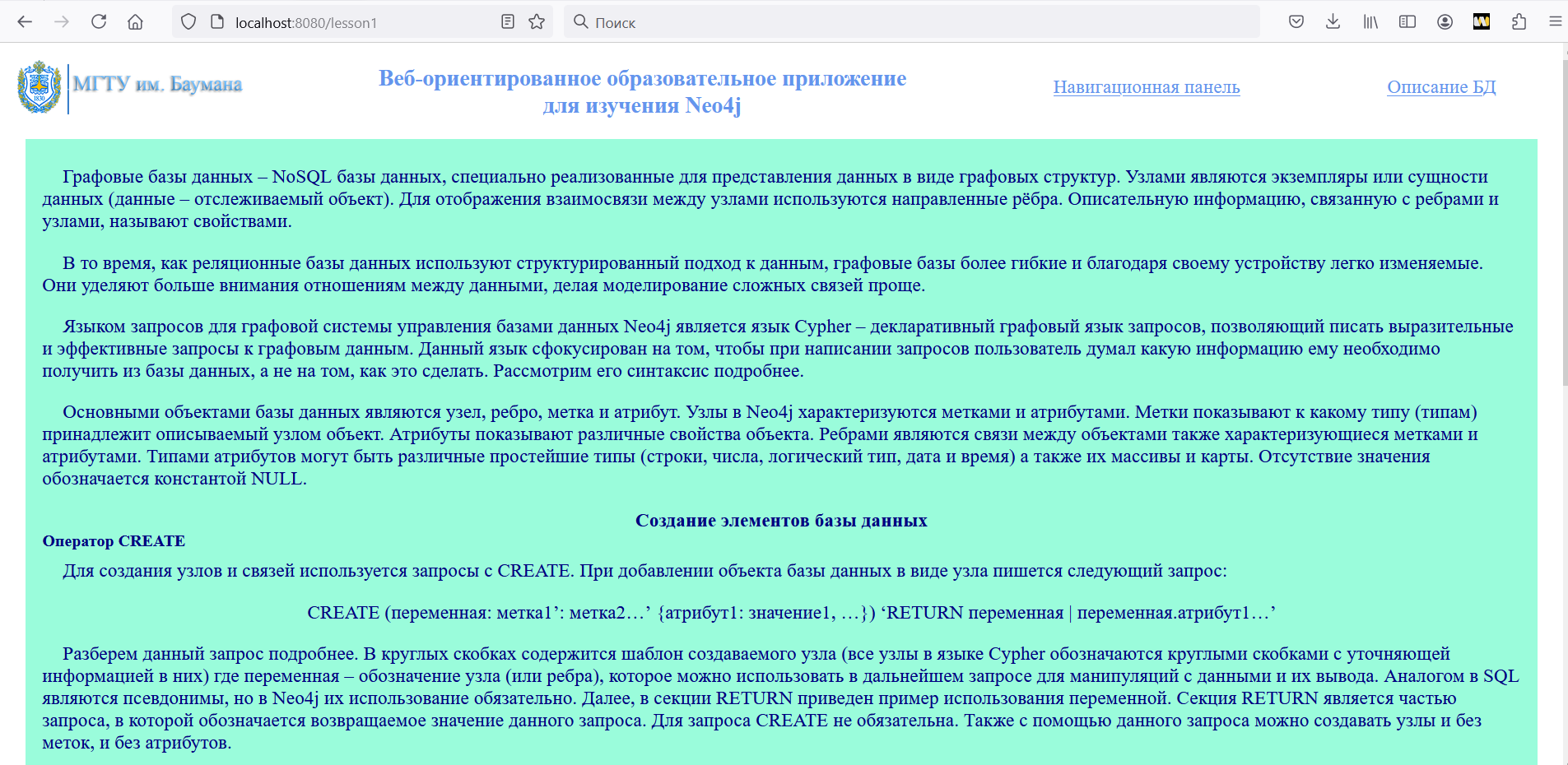


Рисунок 5 – Теоретическая сводка к первому уроку

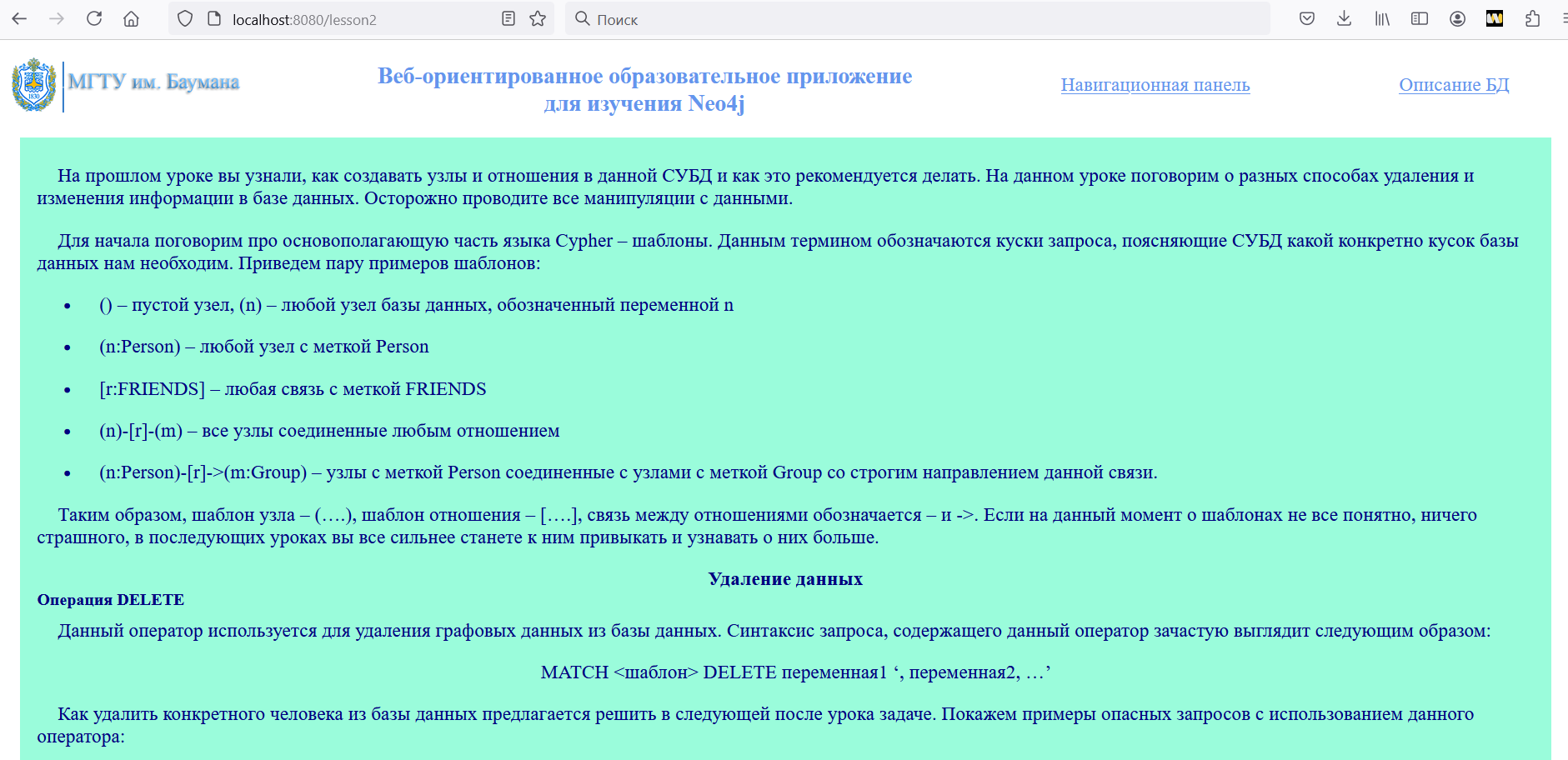


Рисунок 6 – Теоретическая сводка ко второму уроку

После ознакомления с теорией пользователь может перейти с помощью, представленной на рисунке 7 панели навигации на страницу с задачей.

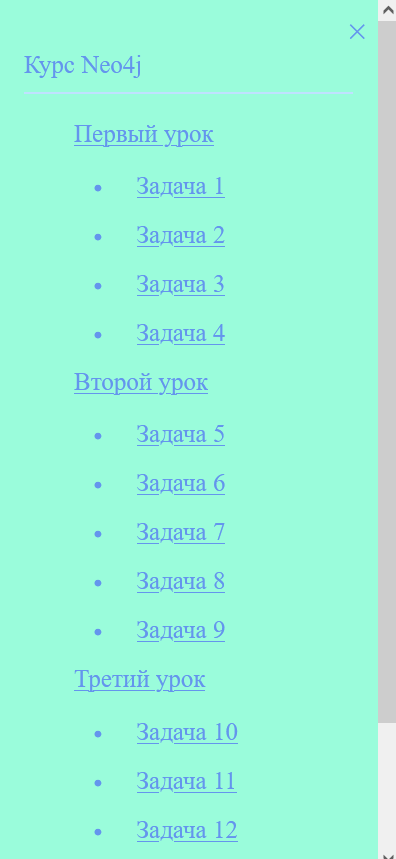


Рисунок 7 – Панель навигации

Пример страницы задач представлен на рисунках 8 и 9. На первом рисунке показана верхняя часть страницы, а на втором нижняя.

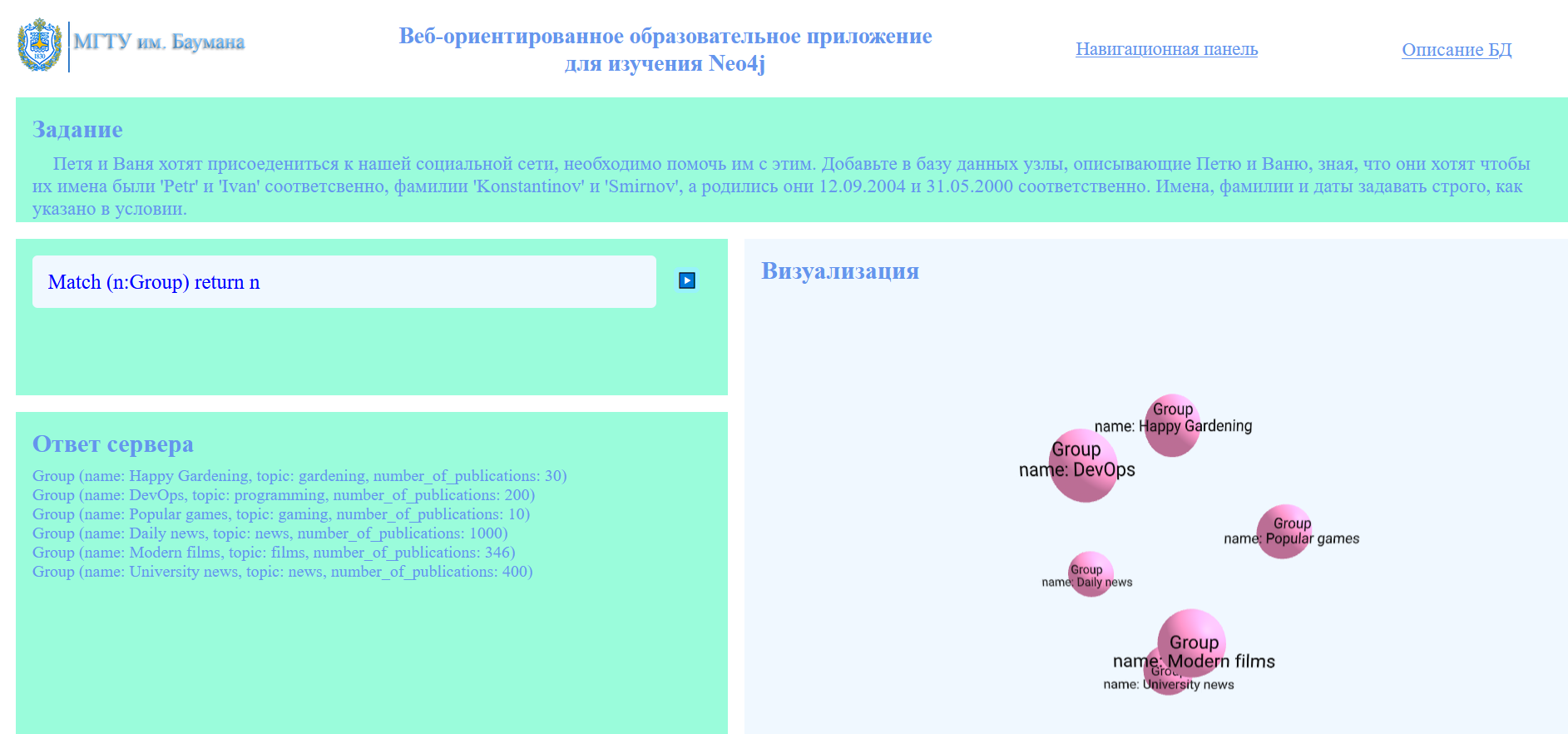


Рисунок 8 – Верхняя часть страницы задач

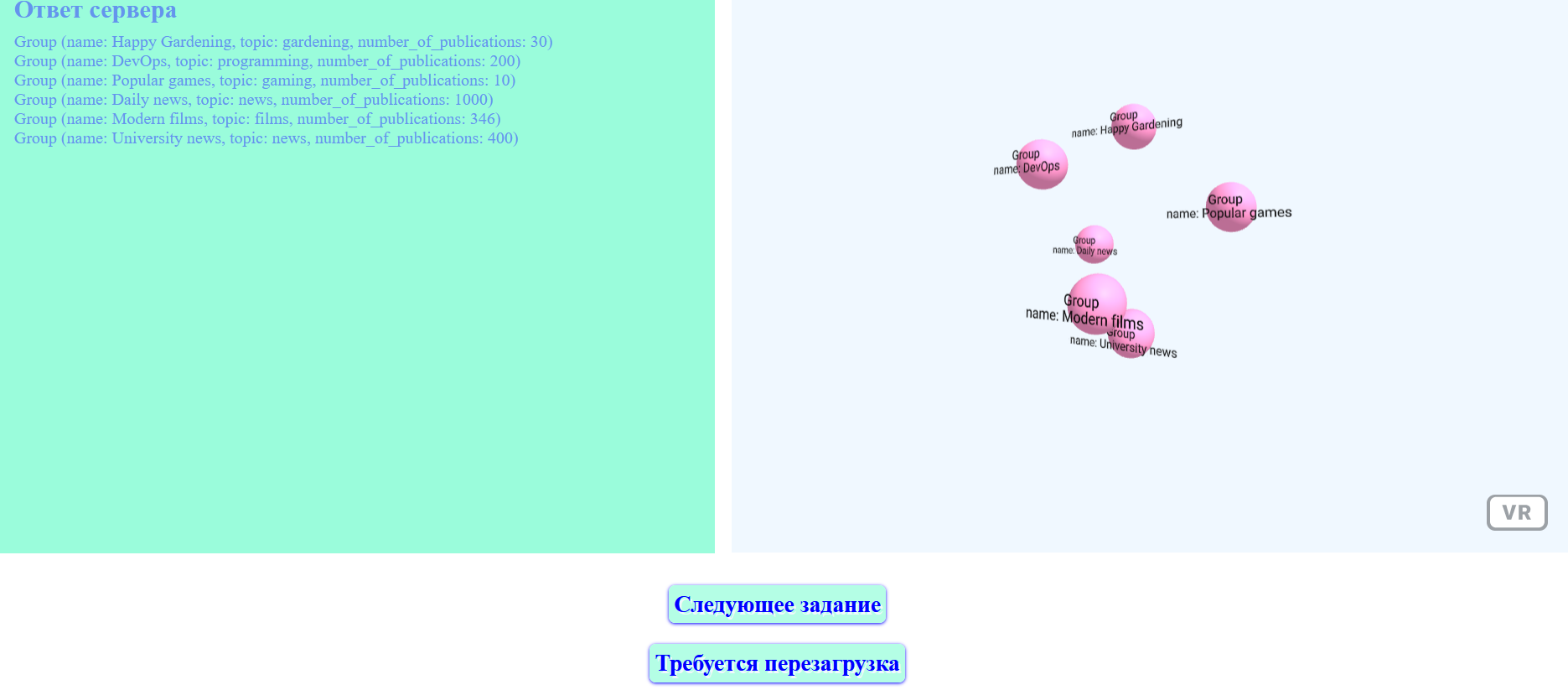


Рисунок 9 – Нижняя часть страницы задач

Также в шапке страниц можно заметить надпись «Описание БД», при наведении на которую возникнет текст, представленный на рисунке 10. В нем, для удобства пользователей есть описание учебной базы данных.

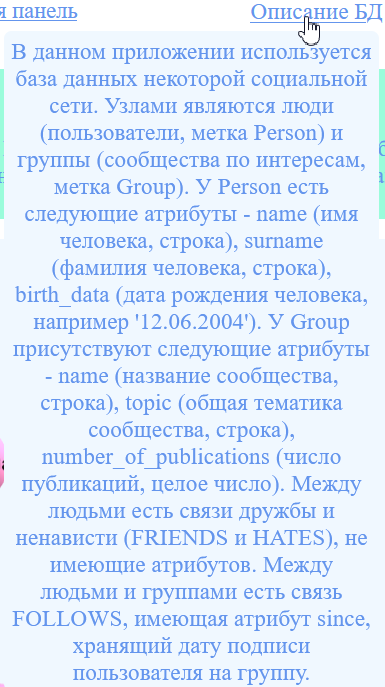


Рисунок 10 – Описание учебной базы данных

Рассмотрим различные примеры визуализаций графов, которые может получить пользователь.

На рисунке 11 показано, как программа изображает граф, состоящий лишь из узлов.

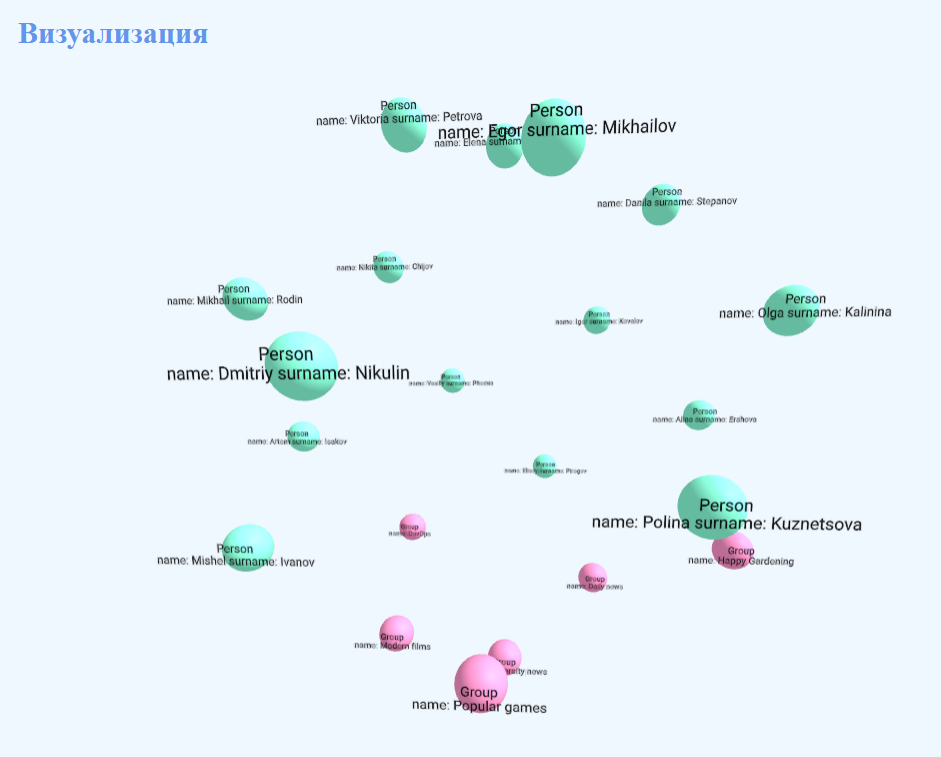


Рисунок 11 – Визуализация узлов

Визуализация всех связанных с заданной вершиной узлов представлена на рисунке 12.

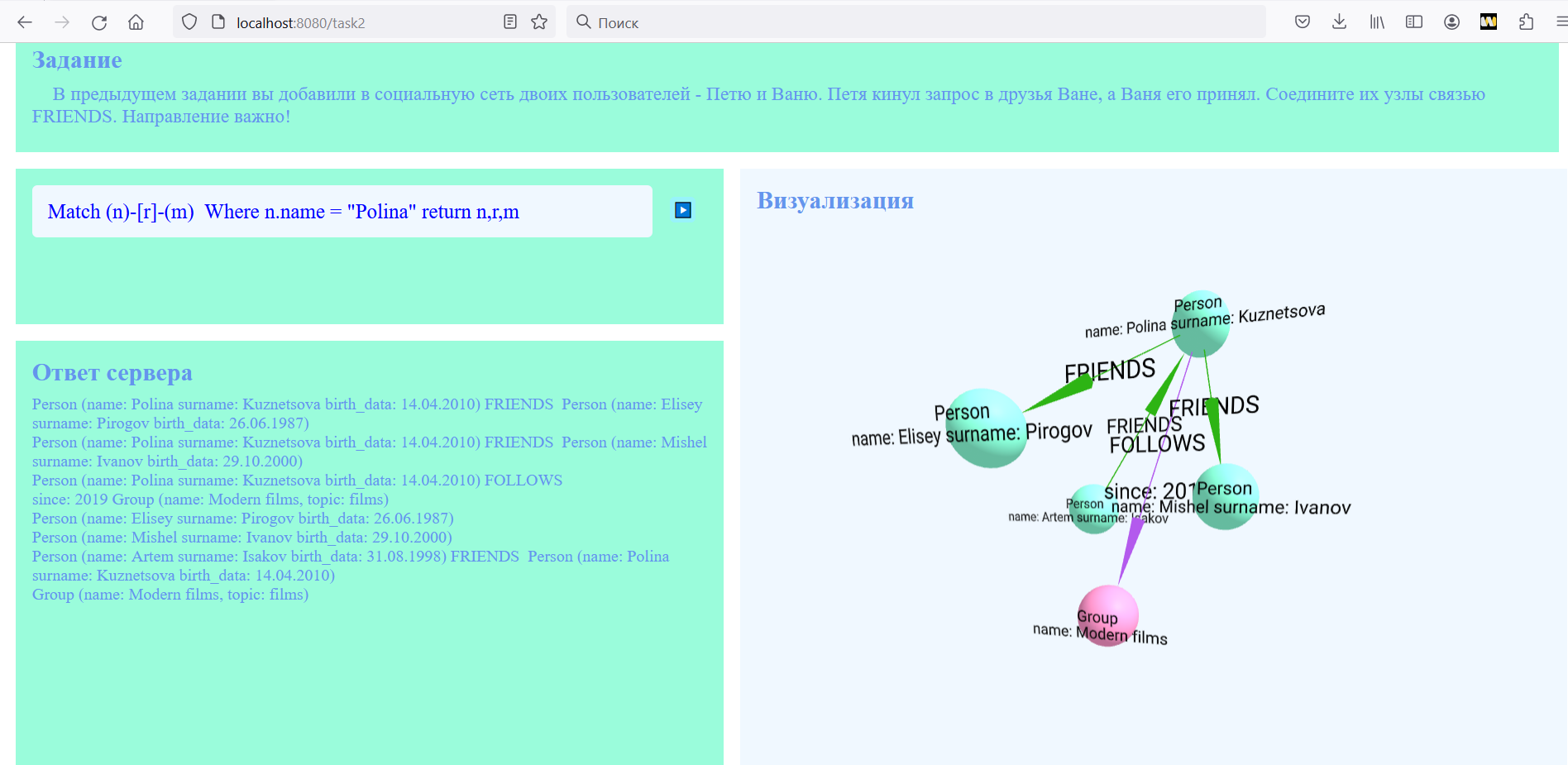


Рисунок 12 – Визуализация связанных с данной вершин

Рисунок 13 демонстрирует способность программы к визуализации несвязных графов.

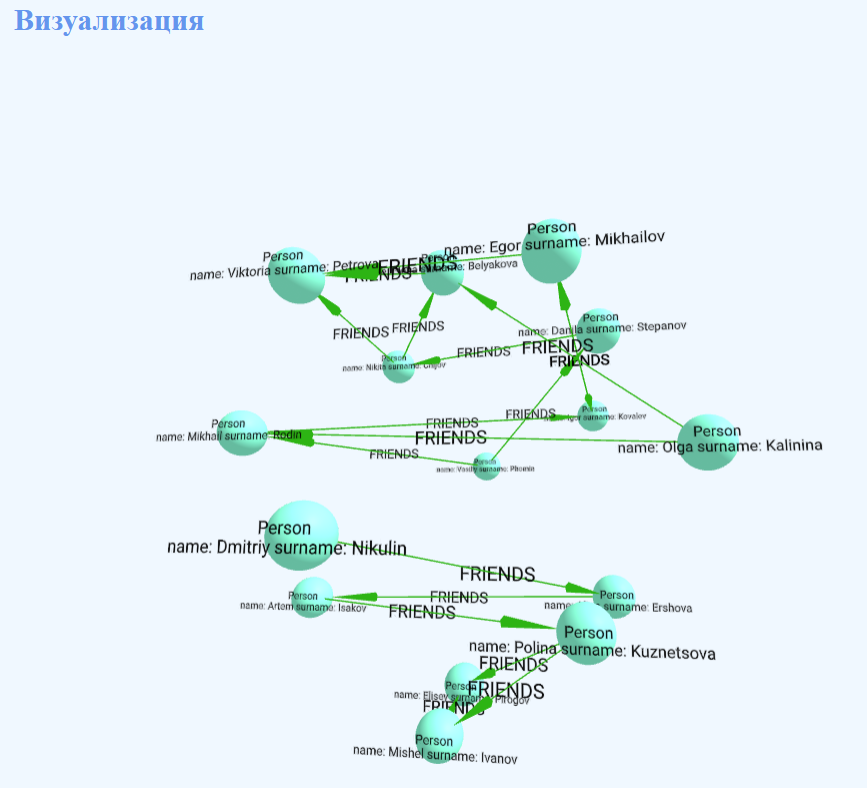


Рисунок 13 – Визуализация графа из двух компонент

Визуализация всего графа базы данных представлена на рисунке 14. Такое изображение возникает не только при запросе, требующем такой визуализации, но и в случаях, когда база данных по каким-либо причинам не возвращает графовых данных.

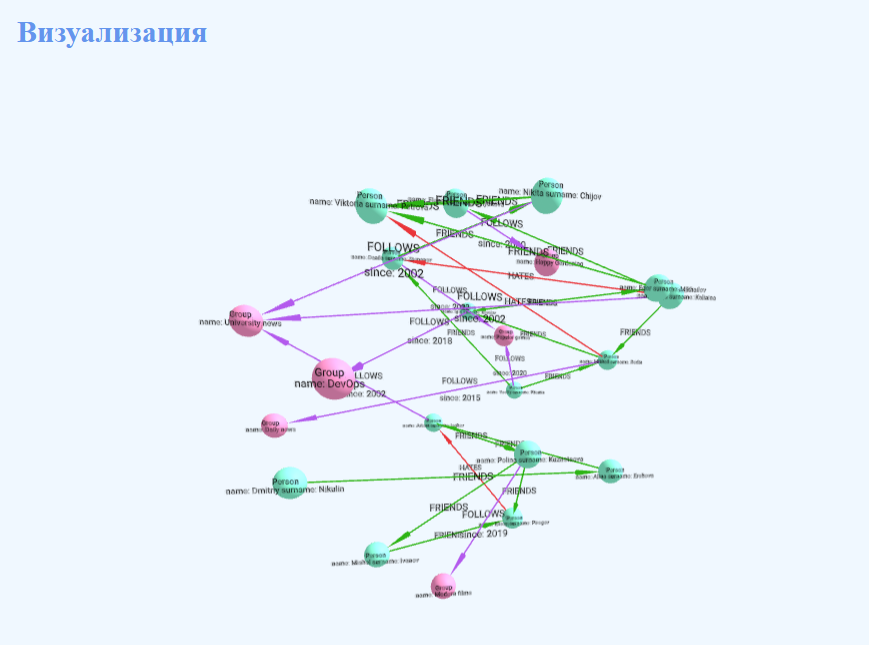


Рисунок 14 – Визуализация всего графа базы данных

## 8. Тестирование приложения

На протяжении всей разработки производилось последовательное тестирование каждого компонента приложения. Результаты данного функционального тестирования представлены в таблице 2.

Таблица 2 – Функциональное тестирование приложения.

|  |  |  |
| --- | --- | --- |
| Действие | Полученный результат | Соответствие полученного результата ожидаемому |
| Предоставление практических заданий пользователю для обучения работе с системой управления базами данных Neo4j | Приложение предоставляет пользователю набор практических заданий. | + |
| Предоставление теоретических справок, необходимых для возможности решения задач. | Приложение предоставляет всю необходимую теоретическую информацию для решения представленных задач. | + |
| Передача запросов пользователя к обучающей базе данных. | Приложение осуществляет корректную передачу запросов пользователя к базе данных через серверную часть приложения. | + |
| Обработка полученной от базы данных ответной информации. | Приложение производит корректную обработку любых полученных данных, перед их отправкой на клиентскую сторону. | + |

Продолжение таблицы 2

|  |  |  |
| --- | --- | --- |
| Действие | Полученный результат | Соответствие полученного результата ожидаемому |
| Корректная визуализация различных полученных от серверной части данных в 3D виде. | Приложение корректно визуализирует любые полученные от серверной части приложения данные в 3D виде. | + |
| Отсутствие необходимости дополнительных действий для функционирования приложения после его запуска | Приложение не требует никаких дополнительных действий после запуска. | + |
| Предоставление возможности пользователю восстановления изначальной учебной базы данных | Пользователь приложения может восстановить учебную базу данных с помощью специальной кнопки. | + |

Таким образом, полученное приложение соответствует всем заявленным функциональным требованиям.

## 9. Инструкция пользователя

9.1 Описание программных средств, необходимых для установки и запуска программы

Для корректной работы приложения, необходимо распаковать архив с приложением и установить следующие программные средства:

1. Java development kit [24] (JDK версии 18 и выше) – набор инструментов для разработки на языке Java. В него входят среда выполнения, компилятор и программный интерфейс Java. JDK необходимо для запуска и корректной работы приложения.
2. Apache maven [25] – фреймворк для автоматизации сборки проектов на основе описания их структуры в файлах на языке POM (Project Object Model). Maven необходим для создания структуры проекта и подключением всех необходимых библиотек.

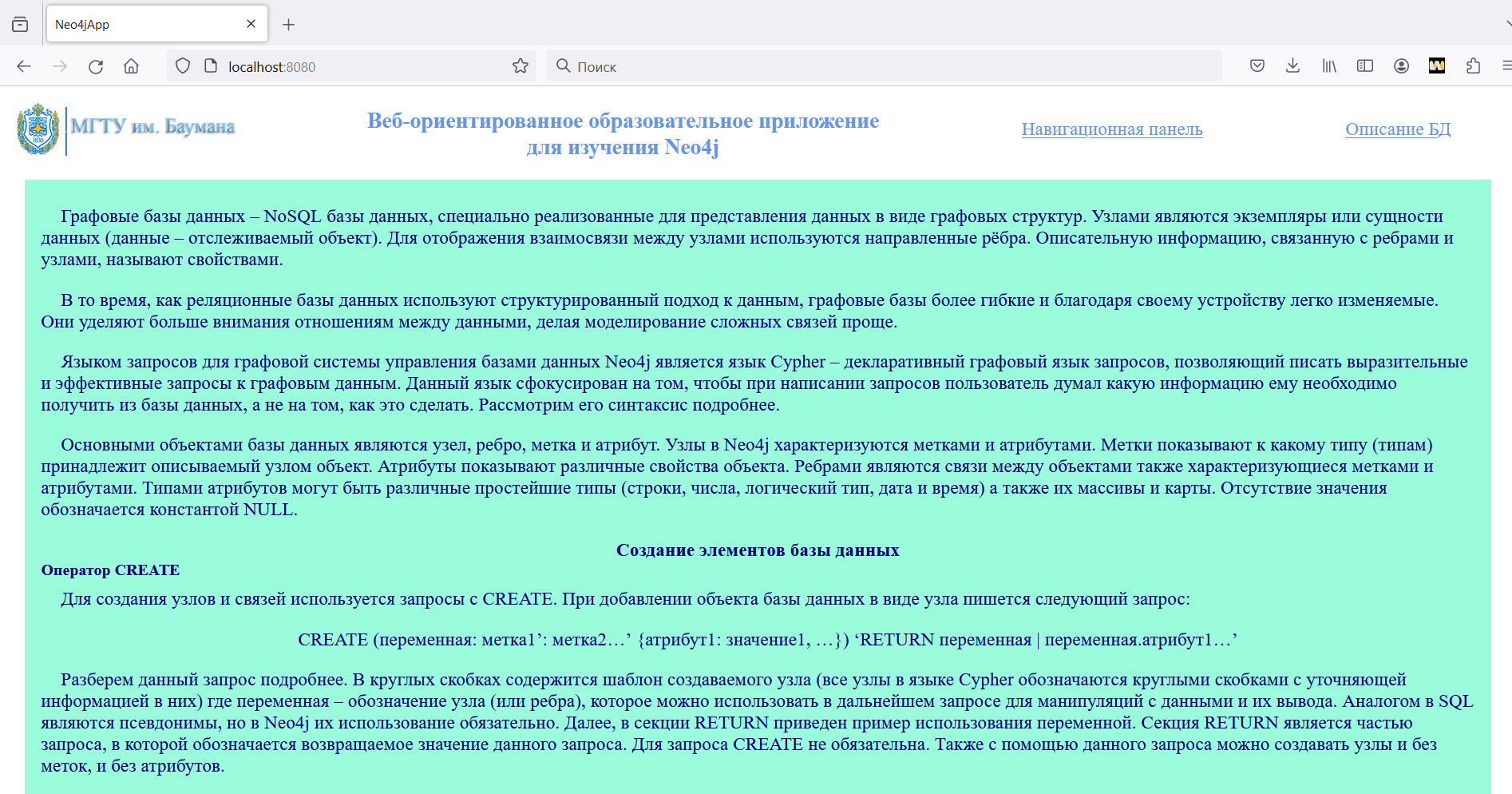
После установки, в корневом каталоге приложения (…/Neo4j\_learning\_app/) необходимо открыть терминал и в нём выполнить следующие команды:

* mvn clean package
* mvn exec:java -Dexec.mainClass=app.spring.data.neo4j.Main

Первая команда отвечает за очистку каталога target – это гарантирует, что при следующей сборке проекта будут использоваться самые последние исходные файлы, а также за компиляцию Java кода и его упаковку в файл со всеми зависимостями. Вторая команда с помощью Maven плагина exec запускает приложение.

9.2 Руководство пользователя по использованию обучающей программы

Для начала использования приложения, пользователю необходимо открыть браузер и перейти по ссылке <http://localhost:8080> после чего он увидит страницу, представленную на рисунке 15. На данной странице пользователь может увидеть теоретический материал к первому уроку, с которым пользователю, запустившему приложение, предлагается ознакомиться.

Рисунок 15 – Страница с теорией

В шапке веб-приложения расположены надписи «Описание БД» и «Навигационная панель». При нажатии на первую высвечивается общее описание структуры учебной базы данных как показано на рисунке 16. Данный текст описывает пользователю какие узлы, ребра и с какими атрибутами пользователю необходимо работать. Из-за нахождения в шапке страницы пользователь имеет доступ к данной информации в любой момент времени работы с приложением.

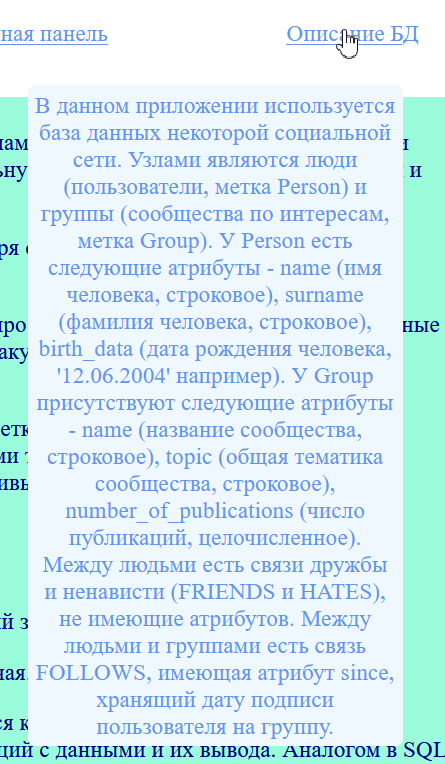


Рисунок 16 – Описание структуры учебной базы данных

При нажатии на надпись «Навигационная панель» слева открывается панель, представленная на рисунке 17. На данной панели присутствуют ссылки на все добавленные на данный момент теоретические уроки и задачи, присутствующие в приложении. Таким образом, навигационная панель позволяет пользователю в любой момент удобно перемещаться по всему приложению.

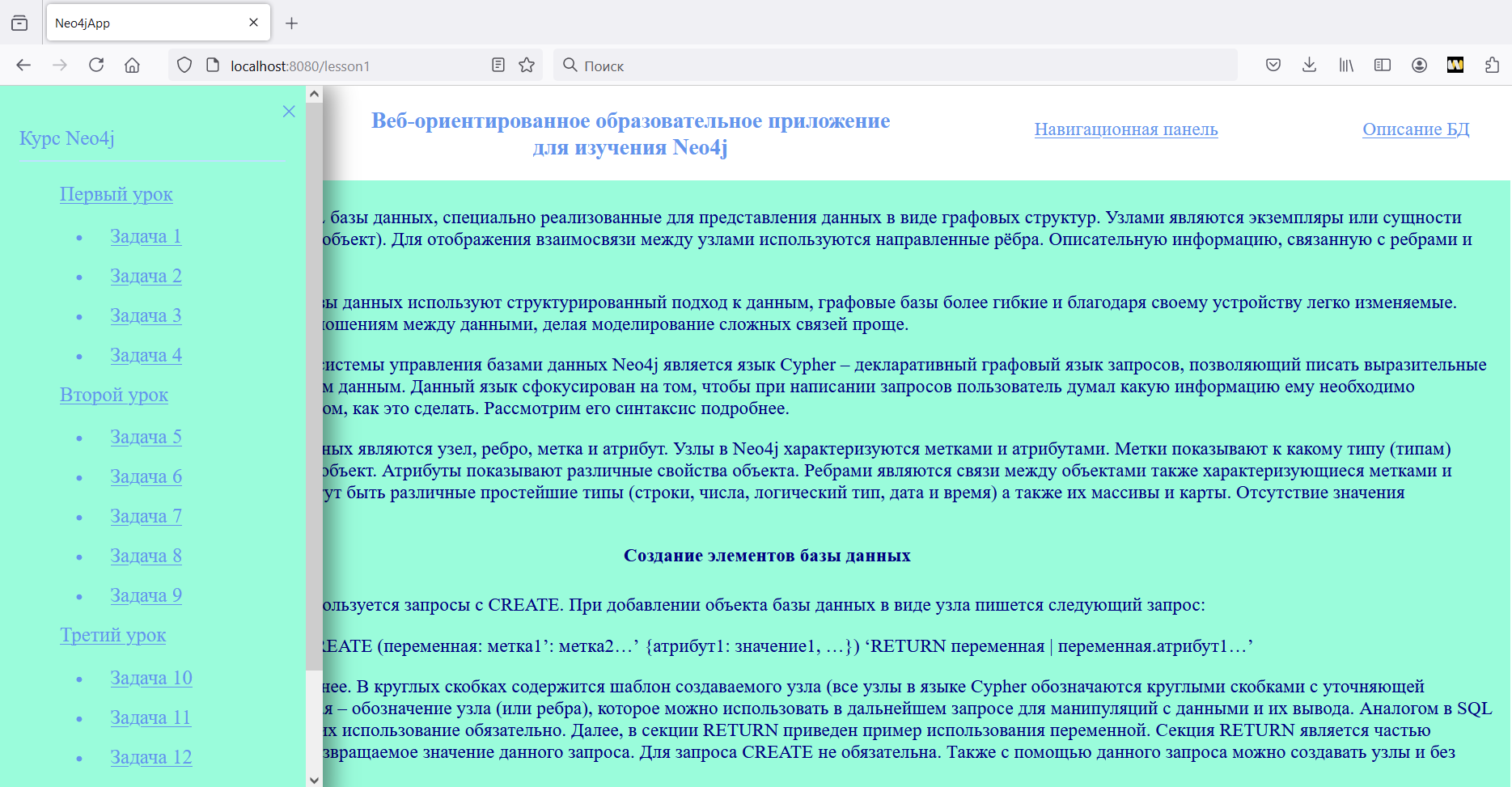


Рисунок 17 – Навигационная панель

При переходе на уроки, выводится соответствующая теория. После каждого урока располагаются задачи на его тему. При нажатии на любую из задач на панели навигации открывается страница задачи как показано на рисунке 18.

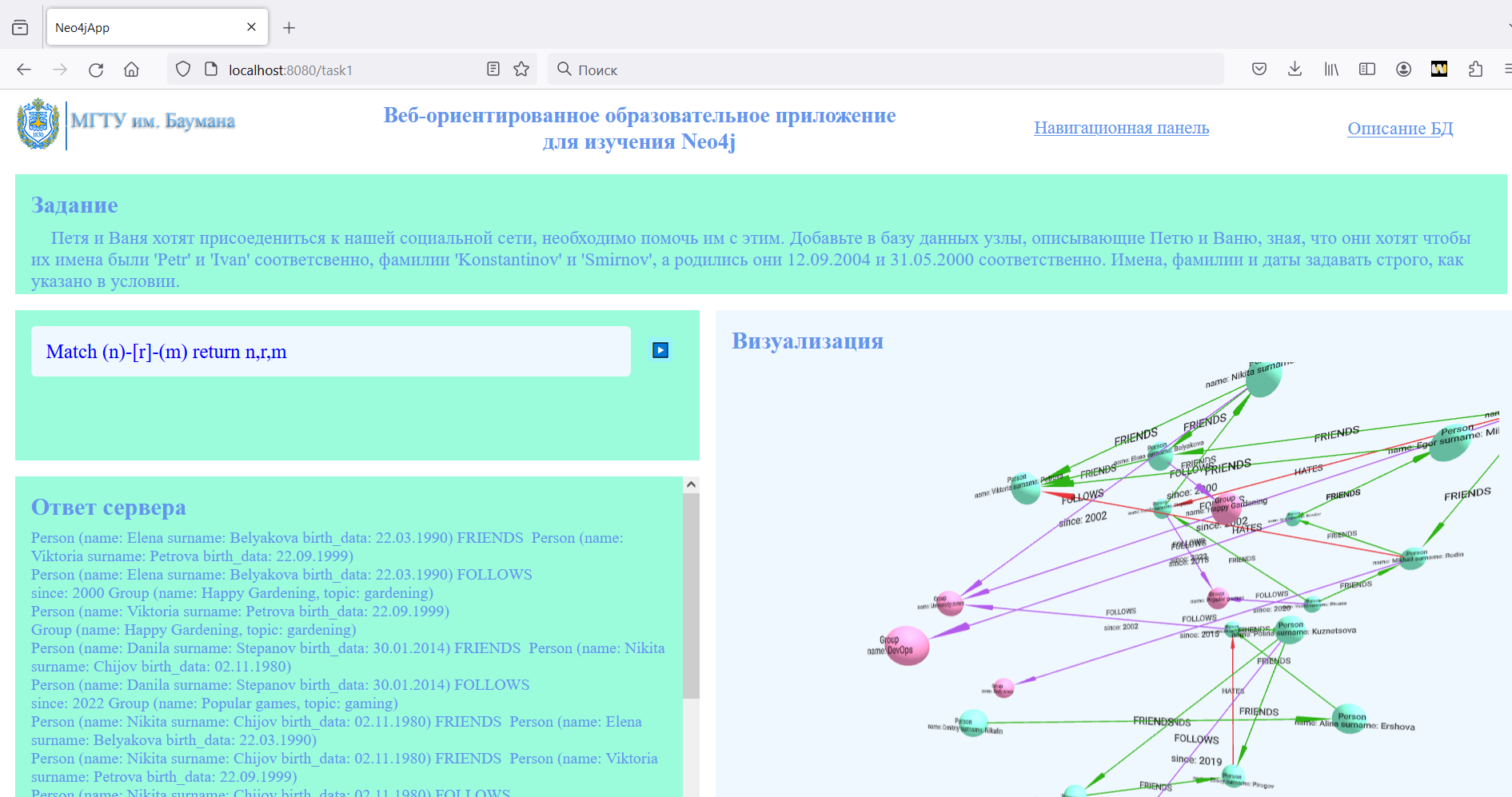


Рисунок 18 – Страница задачи

На странице задачи расположены следующие компоненты:

1. Условие задачи. Сразу после шапки идёт условие соответствующей задачи, в котором описываются действия, необходимые сделать пользователю.
2. Строка ввода запроса. В данной части страницы пользователь может вводить запросы к базе данных, при нажатии на кнопку слева от поля ввода запрос будет отправлен к базе данных.
3. Ответ сервера. Ниже строки ввода запроса располагается часть страницы, где выводится преобразованный ответ от базы данных. При некорректном запросе выводится текст ошибки, при корректном запросе, не возвращающем графовых данных выводится сообщение «Done successfully!». Если ответ на пользовательский запрос содержит графовые данные, в данное поле выводится вся информация о полученных узлах и рёбрах.
4. Визуализация. В данной компоненте происходит отображение полученного от базы данных графа. В случаях, когда на пользовательский запрос база данных не возвращает графовых данных или возвращает ошибку, в данном поле визуализируется вся база данных. Для перемещения в трёхмерном пространстве данного поля есть следующие комбинации клавиш:

* нажатие на клавишу W отвечает за перемещение вперёд;
* нажатие на клавишу S отвечает за перемещение назад;
* нажатие на клавишу D отвечает за перемещение вправо;
* нажатие на клавишу A отвечает за перемещение влево;
* комбинация клавиш Shift + PgUp отвечает за перемещение вверх;
* комбинация клавиш Shift + PgDn отвечает за перемещение вниз;
* для поворота камеры используется перемещение зажатой левой клавиши мыши.

1. Следующее задание. Внизу страницы расположена кнопка «Следующее задание», при нажатии на которую пользователя переносит на следующее задание. Если следующего задания нет, пользователя переносит на ту же страницу.
2. Требуется перезагрузка. Ниже кнопки «Следующее задание» расположена кнопка «Требуется перезагрузка», при нажатии на которую происходит восстановление обучающей базы данных.

# ЗАКЛЮЧЕНИЕ

В результате выполнения данной работы было спроектировано, реализовано и протестировано веб-ориентированное приложение для изучения Neo4j. В частности, было выполнено:

1. Произведён обзор предметной области и выбрана СУБД для реализации.
2. Произведён анализ существующих решений как в обучающем аспекте, так и в аспекте визуализации.
3. Спроектирована и схематично изображена структура приложения. Выбран подходящий стек технологий.
4. Выбран алгоритм размещения графов.
5. Спроектирована и реализована обучающая база данных.
6. Реализована клиент-серверная структура приложения.
7. Реализован магнитно-пружинный алгоритм укладки графов и подобраны коэффициенты для его корректной работы.
8. Реализована 3D визуализация графовых данных, получаемых от серверной части приложения.
9. Реализована форма ввода запроса и вывода обработанного ответа от базы данных в виде сообщения и визуализированного графа.
10. Разработаны несколько уроков, написаны теоретические материалы и задачи к ним.
11. Протестирована работа программы и написана инструкция пользователя.

Впоследствии, на основе данной работы планируется дальнейшее улучшение веб-приложения как, например, добавление задач и теоретического материала к ним для лучшего обучения пользователя, а также перенос серверной части приложения на удалённый сервер.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Neo4j Graph Database & Analytics | Graph Database Management System URL: https://neo4j.com/ (дата обращения: 15.03.2024).
2. А. В. Маркин Системы графовых баз данных. Neo4j. - Москва: Издательство Юрайт, 2021. - 304 с.
3. G. Jordan. Practical Neo4j. - New-York: Apress, 2014. - 426 с.
4. D3 by Observable | The JavaScript library for bespoke data visualization URL: https://d3js.org/ (дата обращения: 17.03.2024).
5. Cytoscape.js URL: https://js.cytoscape.org/ (дата обращения: 17.03.2024).
6. Graphviz URL: https://graphviz.org/ (дата обращения: 17.03.2024).
7. Cytoscape App Store - Cy3D URL: https://apps.cytoscape.org/apps/cy3d (дата обращения: 17.03.2024).
8. Java | Oracle URL: https://www.java.com/ru/ (дата обращения: 15.04.2024).
9. Neo4j JDBC Driver // GitHub URL: https://github.com/neo4j/neo4j-jdbc (дата обращения: 16.04.2024).
10. Spring // Spring | Home URL: https://spring.io/ (дата обращения: 20.04.2024).
11. Home - Ecma International URL: https://ecma-international.org/ (дата обращения: 20.04.2024).
12. Spring Web MVC:: Spring Framework URL: https://docs.spring.io/spring-framework/reference/web/webmvc.html (дата обращения: 20.04.2024).
13. Three.js - JavaScript 3D Library URL: https://threejs.org/ (дата обращения: 30.04.2024).
14. Babylon.js: Powerful, Beautiful, Simple, Open - Web-Based 3D At Its Best URL: https://www.babylonjs.com/ (дата обращения: 30.04.2024).
15. A-Frame - Make WebVR URL: https://aframe.io/ (дата обращения: 30.04.2024).
16. CesiumJS - Cesium URL: https://cesium.com/platform/cesiumjs/ (дата обращения: 30.04.2024).
17. draw.io URL: https://app.diagrams.net/ (дата обращения: 01.05.2024).
18. P. Eades. A heuristic for graph drawing // Congressus Numerantium 42. - Utilitas Mathematica Pub. Incorporated, 1984. - С. 149-160.
19. T.M.J. Fruchterman, E.M. Reingold Graph drawing by force-directed placemen // Software–Practice and Experience, 21. - Wiley Interscience, 1991. - С. 1129-1164.
20. Frick, A. Ludwig, H. Mehldau. A Fast Adaptive Layout Algorithm for Undirected Graphs (Extended Abstract and System Demonstration) // Graph Drawing: Symposium on Graph Drawing . - Springer, 1995. - С. 388-403.
21. Sugiyama K., Misue K. A Simple and Unified Method for Drawing Graphs: Magnetic-Spring Algorithm // International Symposium Graph Drawing and Network Visualization. - 1994
22. T. Kamada, S. Kawai. An algorithm for drawing general undirected graphs // Information Processing Letters, 31. - ScienceDirect, 1989. - С. 7-15.
23. Evenly distributing points on a sphere URL: https://extremelearning.com.au/evenly-distributing-points-on-a-sphere/ (дата обращения: 05.05.2024).
24. Java Downloads | Oracle СНГ URL: https://www.oracle.com/cis/java/technologies/downloads/ (дата обращения: 28.05.2024).
25. Maven – Download Apache Maven URL: https://maven.apache.org/download.cgi (дата обращения: 29.05.2024).

# ПРИЛОЖЕНИЕ А

Листинг 1 – метод класса Service sol\_query()

public Graph sol\_query(final String q) {

String str = "";

StringBuilder stringBuilder = new StringBuilder();

Graph gr = new Graph();

Graph gr1 = new Graph();

try (var session = driver.session()) {

var greeting = session.executeWrite(tx -> {

var query = new Query(q);

var result = tx.run(query);

while ( result.hasNext() ) {

Record record = result.next();

List<String> keys = record.keys();

for (int i = 0; i < keys.size(); i++){

String key = keys.get(i);

String skey = "";

if (gson.toJson(record.get(key)).contains("labels")){

skey = gson.toJson(record.get(key).asNode());

Node n = this.get\_node(skey);

Vertex v = new Vertex(n.get\_elementId(), n.get\_label(), n.make\_Properites(), n.make\_FullProperties());

gr.add\_vertex(v);

}else{

skey = gson.toJson(record.get(key).asRelationship());

Relationship r = this.get\_rel(skey);

Eadge e = new Eadge(r.get\_elementId(), r.get\_type(), r.get\_startElementId(), r.get\_endElementId(), r.make\_Properites());

gr.add\_eadge(e);

}

}

}

gr.preinitGraph(5);

gr.set\_info();

return gr;

});

gr1 = greeting;

}catch (Exception e) {

str = e.toString();

}

if (str.equals("")) {

str = stringBuilder.toString();

if (str.equals("")){

str = "Done Successfully!";

}

//gr1.info = str;

}

if (gr1.eadges.size() == 0 && gr1.vertexes.size() ==0){

gr1 = this.get\_all\_data();

gr1.info = str;

}

return gr1;

}

Листинг 2 – метод magnetic\_spring\_alg()

public void magnetic\_spring\_alg(double l, int k){

fibonacci\_disp(l);

//rand\_disp();

for (int i = 0; i < k; i++){

for (int j = 0; j < this.vertexes.size(); j++){

Vertex u = this.vertexes.get(j);

for (int a = 0; a < u.connected\_vertexes.size(); a++){

if (a != j) {

if (u.connected\_vertexes.get(a)) {

Vertex v = this.vertexes.get(a);

ArrayList<Double> new\_f= f\_attr(u.pos, v.pos, l);

//System.out.println("attr " + new\_f);

ArrayList<Double> new\_magn\_f = f\_magn(u.pos, v.pos);

//System.out.println("magn " + new\_magn\_f);

for (int y = 0; y < new\_f.size();y++){

u.F.set(y, u.F.get(y) + new\_f.get(y)+ new\_magn\_f.get(y));

}

//u.F.get(0) += new\_f.get(0);

} else{

Vertex v = this.vertexes.get(a);

ArrayList<Double> new\_f= f\_rep(u.pos, v.pos);

//System.out.println("rep " + new\_f);

for (int y = 0; y < new\_f.size();y++){

u.F.set(y, u.F.get(y) - new\_f.get(y));

}

}

}

}

this.vertexes.set(j, u);

}

for (int j = 0; j < this.vertexes.size(); j++){

double sigm = 0.001;

//System.out.println(this.vertexes.get(j).F);

for (int a = 0; a < this.vertexes.get(j).pos.size(); a++){

this.vertexes.get(j).pos.set(a, this.vertexes.get(j).pos.get(a) + sigm\*this.vertexes.get(j).F.get(a));

//this.vertexes.get(j).pos.set(a, Math.min(AA, Math.max(-AA,this.vertexes.get(j).pos.get(a))));

}

//System.out.println(this.vertexes.get(j).pos);

}

}

}

Листинг 3 – метод для вычисления направления магнитного поля m()

public ArrayList<Double> m(ArrayList<Double> xyz){

double l = vec\_len(xyz);

ArrayList<Double> res = new ArrayList<>();

res.add(0, xyz.get(1)/l);

res.add(1, -xyz.get(0)/l);

res.add(2, xyz.get(2)/l);

return res;

}

Листинг 4 – метод для вычисления притягивающей силы f\_attr()

public ArrayList<Double> f\_attr(ArrayList<Double> pu, ArrayList<Double> pv, double l){

ArrayList<Double> res = new ArrayList<>();

double c\_attr = 2.0;

ArrayList<Double> pvpu = new ArrayList<>();

for(int i = 0; i < pu.size(); i++){

pvpu.add(i, pv.get(i)-pu.get(i));

}

double pvpv\_norm = vec\_len(pvpu);

for(int i = 0; i < pu.size(); i++){

res.add(i, (c\_attr\*Math.log(pvpv\_norm/l)) \* (pvpu.get(i)/pvpv\_norm));

}

return res;

}

Листинг 5 – метод для вычисления отталкивающей силы f\_rep()

public ArrayList<Double> f\_rep(ArrayList<Double> pu, ArrayList<Double> pv){

ArrayList<Double> res = new ArrayList<>();

double c\_rep = 1.0;

ArrayList<Double> pvpu = new ArrayList<>();

for(int i = 0; i < pu.size(); i++){

pvpu.add(i, pv.get(i)-pu.get(i));

}

double pvpu\_norm = vec\_len(pvpu);

//System.out.println(pvpu\_norm);

for(int i = 0; i < pu.size(); i++){

res.add(i, (c\_rep/(pvpu\_norm\*pvpu\_norm)) \* (pvpu.get(i)/pvpu\_norm));

}

return res;

}

Листинг 6 – метод для вычисления силы магнитного поля f\_magn()

public ArrayList<Double> f\_magn(ArrayList<Double> pu, ArrayList<Double> pv){

ArrayList<Double> res = new ArrayList<>();

double c\_magn = 1.0;

double alpha = 1.0;

double beta = 1.0;

double b = 8.0;

ArrayList<Double> pvpu = new ArrayList<>();

for(int i = 0; i < pu.size(); i++){

pvpu.add(i, pv.get(i)-pu.get(i));

}

double d = vec\_len(pvpu);

ArrayList<Double> m\_vec = m(pvpu);

double theta = Math.acos((m\_vec.get(0)\*pvpu.get(0) + m\_vec.get(1)\*pvpu.get(1) + m\_vec.get(2)\*pvpu.get(2))/(vec\_len(m\_vec)\*vec\_len(pvpu)));

for(int i = 0; i < pu.size(); i++){

res.add(i, c\_magn\*b\*Math.pow(d, alpha)\*Math.pow(theta, beta));

}

return res;}

Листинг 7 – метод для начальной укладки графа fibonacci\_disp()

public void fibonacci\_disp(double l){

double phi = Math.PI \* (Math.sqrt(5.0) - 1.0);

for (int i = 0; i < this.vertexes.size(); i++){

double y = 1.0 - ((double)i/(double)(this.vertexes.size()-1)) \* 2;

//System.out.println(y);

double r = Math.sqrt(1.0-y\*y);

double theta = phi\*i;

double x = Math.cos(theta)\*r;

double z = Math.sin(theta)\*r;

this.vertexes.get(i).pos.add(0, x\*l);

this.vertexes.get(i).pos.add(1, y\*l);

this.vertexes.get(i).pos.add(2, z\*l);

//System.out.println(this.vertexes.get(i).pos);

}

}