

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА
по дисциплине «Введение в нереляционные базы данных»
Тема: Межвузовская ИС данных абитуриентов

Студент гр. 8383

Колмыков В.Д.

Студент гр. 8383

Степанов В.Д.

Студент гр. 8383

Шишкин И. В.

Преподаватель

Заславский М.М.

Санкт-Петербург

2021

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студенты Колмыков В.Д., Степанов В.Д., Шишкин И.В.

Группа 8383

Тема работы: Межвузовская ИС данных абитуриентов

Исходные данные:

Веб-интерфейс для сбора данных абитуриентов, возможность подачи документов в несколько ВУЗов. Раздельный доступ для приемных комиссий и абитуриентов. В работе необходимо использовать нереляционную базу данных MongoDB.

Содержание пояснительной записки:

«Содержание», «Введение», «Качественные требования к решению»,
«Сценарии использования», «Модель данных», «Разработанное приложение»,
«Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания:

Дата сдачи реферата:

Дата защиты реферата:

Студент		Колмыков В.Д.
Студент		Степанов В.Д.
Студент		Шишкин И.В.
Преподаватель		Заславский М.М.

АННОТАЦИЯ

В данной работе реализуется приложение для взаимодействия абитуриентов и ВУЗов. Абитуриент имеет возможность просматривать интересующие ВУЗы, отфильтровать их при поиске, подать заявление в понравившиеся. Работник ВУЗа, состоящий в приемной комиссии, может просматривать всех абитуриентов, а также абитуриентов, которые подали заявление на поступление в ВУЗ работника. Помимо этого, работник может просматривать заявления.

Front-end написан на React, Back-end - на Java с использованием Spring. В качестве базы данных используется MongoDB.

SUMMARY

In this work, an application is implemented for the interaction of applicants and universities. The applicant has the opportunity to view the universities of interest, filter them when searching, apply to the ones they like. An employee of the university, who is on the admissions committee, can view all applicants, as well as applicants who have applied for admission to an employee's university. In addition, the employee can view the statements.

Front-end is written in React, Back-end is written in Java using Spring. MongoDB is used as a database.

СОДЕРЖАНИЕ

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ	2
АННОТАЦИЯ	5
ВВЕДЕНИЕ	8
КАЧЕСТВЕННЫЕ ТРЕБОВАНИЯ К РЕШЕНИЮ	9
СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ	10
2.1. Макет UI	10
2.2. Сценарии использования для задачи	10
2.2.1. Сценарий использования - Авторизация	10
2.2.2. Сценарий использования - Регистрация	10
2.2.3. Сценарий использования - Поиск ВУЗов	12
2.2.4. Сценарий использования - Просмотр выбранного ВУЗа	12
2.2.5. Сценарий использования - Подача заявления в ВУЗ	12
2.2.6. Сценарий использования - Настройка профиля абитуриента	13
2.2.7. Сценарий использования: Поиск абитуриентов	13
2.2.8. Сценарий использования: Просмотр абитуриента	13
2.2.9. Сценарий использования - Настройка профиля работника ВУЗа	14
2.2.10. Сценарий использования - Настройка описания ВУЗа	14
2.2.11. Сценарий использования: Экспорт данных	14
2.2.12. Сценарий использования: Импорт данных	15
2.2.13. Сценарий использования: Просмотр статистики	15
2.2.14. Сценарий использования: Поиск абитуриентов, подавших заявление в ВУЗ работника	15
2.2.15. Сценарий использования: Просмотр абитуриента, подавших заявление в ВУЗ работника	16
МОДЕЛЬ ДАННЫХ	17
3.1. Нереляционная модель данных	17
3.1.1. Графическое представление	17
3.1.2. Описание назначений коллекций, типов данных и сущностей	17
3.1.3. Оценка удельного объема информации, хранимой в модели	19
3.1.4. Запросы к модели, с помощью которых реализуются сценарии использования	22
3.2. Аналог модели данных для SQL СУБД	24

3.2.1. Графическое представление.	24
3.2.2. Описание назначений коллекций, типов данных и сущностей	24
3.2.3. Оценка удельного объема информации, хранимой в модели	26
3.2.4. Запросы к модели, с помощью которых реализуются сценарии использования	28
3.3. Сравнение моделей	29
РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ	31
4.1. Краткое описание	31
4.2. Схема экранов приложения	31
4.3. Используемые технологии	31
4.4. Ссылки на Приложение	31

ВВЕДЕНИЕ

Цель работы - реализовать приложение для сбора данных абитуриентов, возможность подачи документов в несколько ВУЗов. Раздельный доступ для приемных комиссий и абитуриентов. Для решения этой проблемы, был реализован веб-интерфейс, позволяющий двум группам пользователей просматривать и искать информацию по ВУЗам и абитуриентам.

1. КАЧЕСТВЕННЫЕ ТРЕБОВАНИЯ К РЕШЕНИЮ

Разработать веб-интерфейс, который в качестве базы данных использует MongoDB. С помощью созданного интерфейса абитуриенты могут просматривать ВУЗы и подавать в них заявления, а работники ВУЗа - изучать информацию об абитуриентах, а также смотреть связанную с ними статистику.

2. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

2.1. Макет UI

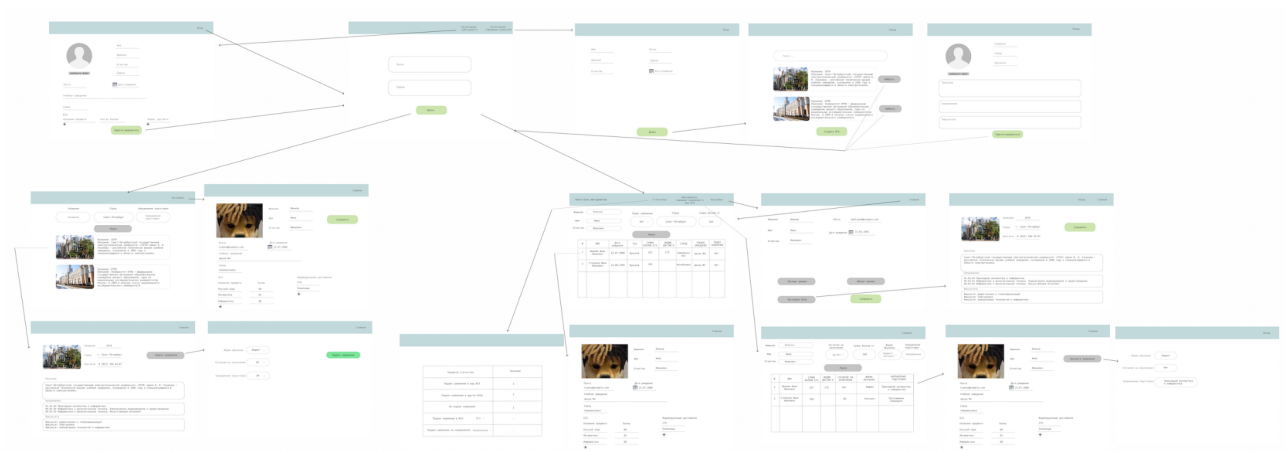


Рисунок 2.1.1 – Макет приложения

Полная версия изображения доступна по ссылке:

https://github.com/moevm/nosql2h21-university/blob/main/reports/NoSQL_UseCase.png

2.2. Сценарии использования для задачи

2.2.1. Сценарий использования - Авторизация

Действующие лицо: Абитуриент, Работник ВУЗа

Основной сценарий:

1. Пользователь попадает на страницу входа
2. Вводит почту и пароль
3. Нажимает кнопку "Войти"
4. Пользователь переходит на главную страницу, соответствующую его роли

Альтернативный сценарий:

- Введена неверная почта и/или пароль

2.2.2. Сценарий использования - Регистрация

Действующие лицо: Абитуриент

Основной сценарий:

1. Абитуриент попадает на страницу входа
2. Нажимает кнопку "Регистрация (абитуриент)"
3. Пользователь перенаправляется на страницу регистрации
4. Вводит запрашиваемые данные (такие, как ФИО, дата рождения, почта и др.)
5. Пользователь нажимает на кнопку "Зарегистрироваться"
6. После пользователь перенаправляется на страницу авторизации

Альтернативный сценарий:

- Введены не все данные
- Введенная почта уже занята

Действующие лицо: Работник ВУЗа

Основной сценарий:

1. Работник ВУЗа попадает на страницу входа
2. Нажимает кнопку "Регистрация (приемная комиссия)"
3. Пользователь перенаправляется на страницу регистрации
4. Вводит запрашиваемые данные (такие, как ФИО, дата рождения, почта и дата рождения)
5. Пользователь нажимает на кнопку "Далее"
6. После пользователь перенаправляется на страницу выбора ВУЗа
7. Работник выбирает ВУЗ и переходит к шагу 11
8. Работник нажимает кнопку "Создать ВУЗ" и попадает на страницу создания ВУЗа
9. Далее вводит данные вуза (такие, как название, город, контакты и др.)
10. Пользователь нажимает на кнопку "Зарегистрироваться"
11. После пользователь перенаправляется на страницу авторизации

Альтернативный сценарий:

- Введены не все данные

- Введенная почта уже занята

2.2.3. Сценарий использования - Поиск ВУЗов

Действующие лицо: Абитуриент

Основной сценарий:

1. Пользователь после логина попадает на главную страницу абитуриента
2. Вводит интересующие пользователя параметры для поиска (такие, как название ВУЗа, город, в котором расположен ВУЗ, направление подготовки)
3. Нажимает кнопку "Поиск"
4. Пользователю показываются подходящие под фильтр ВУЗы

Альтернативный сценарий:

- ВУЗы по выбранным параметрам не найдены

2.2.4. Сценарий использования - Просмотр выбранного ВУЗа

Действующие лицо: Абитуриент

Основной сценарий:

1. После поиска ВУЗов пользователь выбирает интересующий и нажимает на иконку ВУЗа
2. Пользователь перенаправляется на страницу описания выбранного ВУЗа
3. На странице показывается информация о ВУЗа

2.2.5. Сценарий использования - Подача заявления в ВУЗ

Действующие лицо: Абитуриент

Основной сценарий:

1. На странице ВУЗа пользователь нажимает кнопку "Подать заявление"
2. Пользователь перенаправляется на страницу подачи заявления

3. На странице пользователь заполняется все необходимые поля и загружает файл с заявлением
4. Пользователь нажимает кнопку отправить
5. Пользователь перенаправляется на страницу ВУЗа

2.2.6. Сценарий использования - Настройка профиля абитуриента

Действующие лицо: Абитуриент

Основной сценарий:

1. На главной странице пользователь нажимает кнопку "Настройки"
2. Пользователь перенаправляется на страницу настроек профиля абитуриента
3. Пользователь изменяет интересующие его поля
4. Пользователь нажимает кнопку "Сохранить"

2.2.7. Сценарий использования: Поиск абитуриентов

Действующие лицо: Работник ВУЗа

Основной сценарий:

1. Пользователь после логина попадает на главную страницу работника ВУЗа
2. Вводит интересующие пользователя параметры для поиска абитуриентов (такие, как ФИО, город, сумма баллов ЕГЭ, превосходящих заданное значение, статус заявления)
3. Нажимает кнопку "Поиск"
4. Пользователю показываются подходящие под фильтр абитуриенты

2.2.8. Сценарий использования: Просмотр абитуриента

Действующие лицо: Работник ВУЗа

Основной сценарий:

1. После поиска абитуриентов пользователь выбирает интересующего и нажимает на него
2. Пользователь перенаправляется на страницу профиля выбранного абитуриента
3. На странице показывается информация об абитуриенте

2.2.9. Сценарий использования - Настройка профиля работника ВУЗа

Действующие лицо: Работник ВУЗа

Основной сценарий:

1. На главной странице пользователь нажимает кнопку "Настройки"
2. Пользователь перенаправляется на страницу настроек профиля
3. Пользователь изменяет интересующие его поля
4. Пользователь нажимает кнопку "Сохранить"

2.2.10. Сценарий использования - Настройка описания ВУЗа

Действующие лицо: Работник ВУЗа

Основной сценарий:

1. На странице настроек профиля работника ВУЗа пользователь нажимает кнопку "Настройки ВУЗа"
2. Пользователь перенаправляется на страницу описания ВУЗа
3. Пользователь изменяет интересующие его поля
4. Пользователь нажимает кнопку "Сохранить"

2.2.11. Сценарий использования: Экспорт данных

Действующие лицо: Работник ВУЗа

Основной сценарий:

1. На странице настроек профиля работника ВУЗа пользователь нажимает кнопку "Экспорт данных"

2. Пользователю предоставляется возможность выбрать, куда сохранить файл с данными
3. Пользователь сохраняет файл

2.2.12. Сценарий использования: Импорт данных

Действующие лицо: Работник ВУЗа

Основной сценарий:

1. На странице настроек профиля работника ВУЗа пользователь нажимает кнопку "Импорт данных"
2. Пользователю предоставляется возможность выбрать файл с данными
3. Пользователь выбирает файл

2.2.13. Сценарий использования: Просмотр статистики

Действующие лицо: Работник ВУЗа

Основной сценарий:

1. На главной странице работника ВУЗа пользователь нажимает кнопку "Статистика"
2. Пользователь перенаправляется на страницу со статистикой
3. На странице показана таблица со статистикой по ВУЗам, абитуриентам, направлениям
4. Пользователь выбирает просмотр статистики в определенном ВУЗе и/или по определенному направлению
5. Таблица обновляется и пользователю предоставляются выбранные им данные

2.2.14. Сценарий использования: Поиск абитуриентов, подавших заявление в ВУЗ работника

Действующие лицо: Работник ВУЗа

Основной сценарий:

1. На главной странице работника ВУЗа пользователь нажимает кнопку "Абитуриенты, подавшие заявление в ваш ВУЗ"
2. Вводит интересующие пользователя параметры для поиска абитуриентов (такие, как ФИО, сумма баллов ЕГЭ, превосходящих заданное значение, согласие на зачисление, форма обучения)
3. Нажимает кнопку "Поиск"
4. Пользователю показываются подходящие под фильтр абитуриенты

2.2.15. Сценарий использования: Просмотр абитуриента, подавших заявление в ВУЗ работника

Действующие лицо: Работник ВУЗа

Основной сценарий:

1. После поиска абитуриентов пользователь выбирает интересующего и нажимает на него
2. Пользователь перенаправляется на страницу профиля выбранного абитуриента
3. На странице показывается информация об абитуриенте
4. Пользователь нажимает кнопку "Просмотр заявления"
5. Пользователю перенаправляется на страницу заявления, выбранного абитуриента, где отображены все необходимые данные

3. МОДЕЛЬ ДАННЫХ

3.1. Нереляционная модель данных

3.1.1. Графическое представление

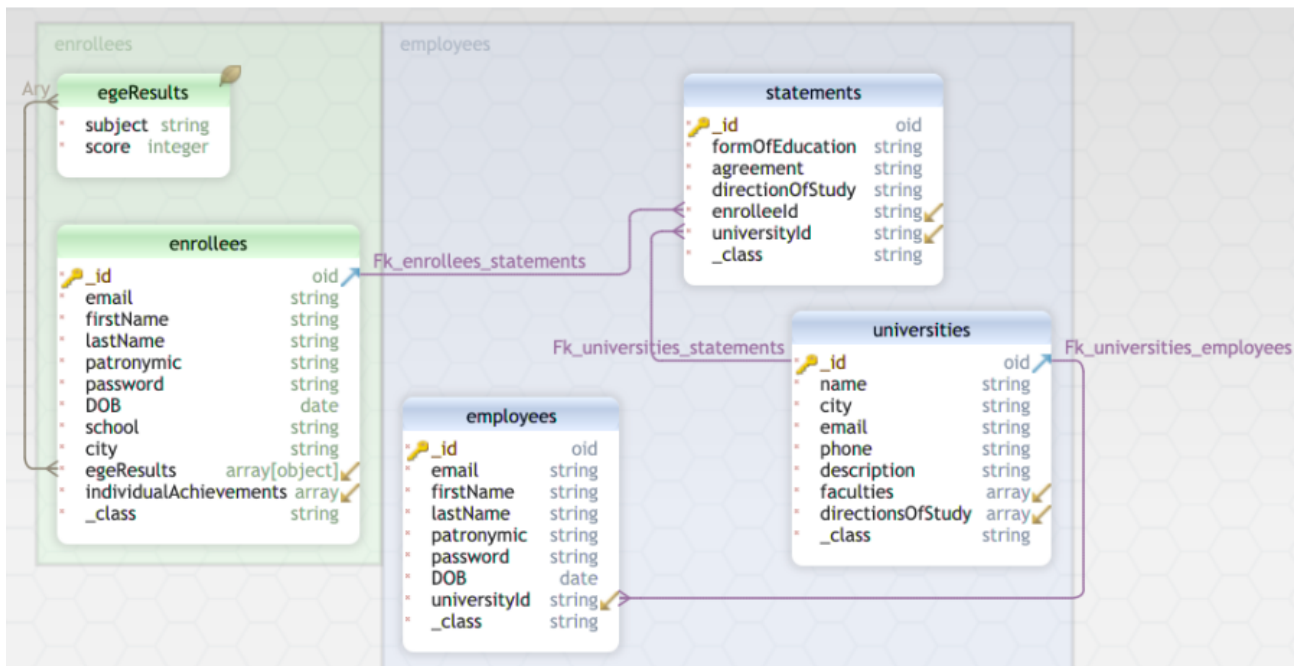


Рисунок 3.1.1.1 - Графическое представление нереляционной модели данных

3.1.2. Описание назначений коллекций, типов данных и сущностей

Сущность "абитуриент" (enrollee)

- **_id (ObjectID)** – уникальный идентификатор
- **Email (String)** (ключ) – почта абитуриента
- **Firstname (String)** – имя абитуриента
- **Lastname (String)** – фамилия абитуриента
- **Patronymic (String)** – отчество абитуриента
- **Password (String)** – пароль
- **DOB (Date)** – Дата рождения
- **School (String)** – школа
- **City (String)** – город

- EGResults (Array[Object]) – результаты ЕГЭ (предмет и количество баллов)
- IndividualAchievements (Array) – индивидуальные достижения (Золотая медаль, ГТО и др.)

Пусть среднее количество "Результатов ЕГЭ" S_{av} , среднее количество "Индивидуальных достижений" I_{av} .

Тогда объем сущности "абитуриент" $E = (8 \text{ байт} + 25 \text{ байт} + 7 \text{ байт} + 9 \text{ байт} + 10 \text{ байт} + 10 \text{ байт} + 8 \text{ байт} + 15 \text{ байт} + 10 \text{ байт} + S_{av} * (8 \text{ байт} + 8 \text{ байт}) + I_{av} * (10 \text{ байт})) = 102 + 16 * S_{av} + 10 * I_{av}$.

Сущность "работник ВУЗа" (employees)

- _id (ObjectID) – уникальный идентификатор
- Email (String) – почта работника
- Firstname (String) – имя работника
- LastName (String) – фамилия работника
- Patronymic (String) – отчество работника
- Password (String) – пароль
- DOB (Data) – дата рождения
- UniversityId (String) – id прикрепленного университета

Сущность "университет" (university)

- _id (ObjectID) – уникальный идентификатор
- Name (String) – название ВУЗа
- City (String) – город
- Email (String) – почта
- Phone (String) – телефон
- Description (String) – описание
- Faculties (Array) – имеющиеся факультеты
- DirectionsOfStudy (Array) – имеющиеся направления обучения

Пусть среднее количество факультетов F_{av} и среднее количество направлений D_{av} .

Тогда объем сущности "университет" $U = (8 \text{ байт} + 30 \text{ байт} + 10 \text{ байт} + 400 \text{ байт} + 25 \text{ байт} + 12 \text{ байт} + F_{av} * 20 + D_{av} * 30) = 485 + 20 * F_{av} + 30 * D_{av}$.

Сущность "Заявление"(Statement)

- $_id$ (String) – уникальный идентификатор
- FormOfEducation (String) – форма обучения
- Agreement (String) – Согласие на зачисление
- DirectionOfStudy (String) – Направление обучения
- EnrolleeId (String) – id абитуриента
- UniversityId (String) – id университета

3.1.3. Оценка удельного объема информации, хранимой в модели

Пусть у нас совокупность полей ($_id$, email, firstname, lastname, patronymic, password, DOB, universityId) сущности "Работник ВУЗа" занимает N памяти. Имеем среднее количество работников N_{av} , количество ВУЗов U_{co} , среднее количество факультетов F_{av} и среднее количество направлений D_{av} .

Тогда хранение данных о ВУЗе и работнике в MongoDB будет занимать $U_{co} * (U(F_{av}, D_{av}) + N * N_{av})$.

Тогда при условии $U(F_{av}, D_{av}) = 485 + 20 * F_{av} + 30 * D_{av}$, $N = (8 \text{ байт} + 25 \text{ байт} + 7 \text{ байт} + 9 \text{ байт} + 10 \text{ байт} + 10 \text{ байт} + 8 \text{ байт} + 15 \text{ байт} = 92 \text{ байт})$. В среднем имеем $N_{av} = 3$ работника и имеем $U_{co} = 300$ вузов, в каждом ВУЗе примерно $F_{av} = 10$ и $D_{av} = 50$.

Для хранения данных о ВУЗе и работнике нам понадобится $300 * (485 + 20 * 10 + 30 * 50 + 92 * 3) = 738\,300$ байт.

Пусть у нас совокупность полей ($_id$, FormOfEducation, Agreement, DirectionOfStudy, EnrolleeId, UniversityId) сущности "Заявление" занимает T . Так же имеем, что среднее количество "Индивидуальных достижений" I_{av} , среднее

количество "Результатов ЕГЭ" S_{av} , количество "Абитуриентов" E_{co} и среднее количество заявления T_{av} .

Тогда хранение данных об абитуриентах в MongoDB будет занимать $E_{co} * (E(S_{av}, I_{av}) + T * T_{av})$.

Тогда при условии $E(S_{av}, I_{av}) = 102 + 16 * S_{av} + 10 * I_{av}$, $T = (8 \text{ байт} + 10 \text{ байт} + 3 \text{ байта} + 30 \text{ байт} + 8 \text{ байт} + 8 \text{ байт} = 67 \text{ байт})$. В среднем имеем $I_{av} = 2$, $S_{av} = 3$, количество абитуриентов $E_{co} = 6\,000$ и среднее количество заявлений $T_{av} = 3$.

Для хранения данных об абитуриентах нам понадобится $6\,000 * (102 + 16 * 3 + 10 * 2 + 67 * 3) = 2\,226\,000$ байт.

Всего хранение модели будет занимать $V = U_{co} * (U(F_{av}, D_{av}) + N * N_{av}) + E_{co} * (E(S_{av}, I_{av}) + T * T_{av}) = U_{co} * 2\,461 + E_{co} * 371$.

Чистый объем данных.

У сущности "университет" убираются факультеты и направления: $U_{new} = (8 \text{ байт} + 30 \text{ байт} + 10 \text{ байт} + 400 \text{ байт} + 25 \text{ байт} + 12 \text{ байт}) = 485 \text{ байт}$.

У сущности "работник ВУЗа" убирается universityId: $N_{new} = (8 \text{ байт} + 25 \text{ байт} + 7 \text{ байт} + 9 \text{ байт} + 10 \text{ байт} + 10 \text{ байт} + 8 \text{ байт}) = 77 \text{ байт}$.

У сущности "абитуриент" убираются результаты ЕГЭ и индивидуальные достижения: $E_{new} = (8 \text{ байт} + 25 \text{ байт} + 7 \text{ байт} + 9 \text{ байт} + 10 \text{ байт} + 10 \text{ байт} + 8 \text{ байт} + 15 \text{ байт} + 10 \text{ байт}) = 102 \text{ байт}$.

У сущности "заявление" убираются enrolleeId, universityId: $T_{new} = (8 \text{ байт} + 10 \text{ байт} + 3 \text{ байта} + 30 \text{ байт}) = 51 \text{ байт}$.

$V_c = U_{co} * (U_{new} + N_{new} * N_{av}) + E_{co} * (E_{new} + T_{new} * T_{av}) = U_{co} * 716 + E_{co} * 255$.

Допустим, на каждый вуз приходится по 1000 абитуриентов, тогда:

$$V_c = U_{co} * (U_{new} + N_{new} * N_{av} + E_{co} * (E_{new} + T_{new} * T_{av})) = \\ U_{co} * (716 + 1\,000 * 255) = U_{co} * 255\,716 \text{ байт.}$$

$$V = U_{co} * (U(F_{av}, D_{av}) + N * N_{av} + E_{co} * (E(S_{av}, I_{av}) + T * T_{av})) \\ = U_{co} * (2\,461 + 1\,000 * 371) = U_{co} * 373\,461 \text{ байт.}$$

Тогда отношение фактического объема к чистому: $V / V_c = (U_{co} * 373\,461) / (U_{co} * 255\,716) = 1.46$

Избыточность модели.

В сущности "Абитуриент" в поле "Результаты ЕГЭ", для удобства, мы храним полное название предмета. Получается, что у каждого абитуриента хранится полное название предмета, по которому он сдавал экзамен (так как ЕГЭ по Русскому и Математике являются обязательными, то у каждого абитуриента они точно будут). В данной ситуации можно сделать отдельную сущность "Предметы ЕГЭ", где будут перечислены все возможные предметы ЕГЭ. Примерно такая же ситуация наблюдается и у "Индивидуальных достижений". Абитуриенты могут вообще не иметь достижений или иметь одинаковые для всех (ГТО, Золотая медаль и т.д.). В сущности "ВУЗ" имеются поля "Факультеты" и "Направления подготовки". Данные поля будут повторяться у некоторых университетов, что будет являться избыточностью.

Данные избыточности допущены в качестве повышения скорости поиска и удобства. Другие избыточности отсутствуют.

Направление роста модели.

При добавлении новых абитуриентов не создаются записи в других сущностях. При этом, при добавлении работника ВУЗа мы можем выбрать имеющийся ВУЗ или добавить новый.

3.1.4. Запросы к модели, с помощью которых реализуются сценарии использования

- Добавление абитуриента (используется одна коллекция)

```
db.enrolles.insertOne({
  email: "someemail@mail.ma",
  firstName: "Ivan",
  lastName: "Ivanov",
  patronymic: "Ivanovich",
  password: "some_pass",
  DOB: "2000-01-01",
  school: "some school 1",
  city: "SPb",
  egeResults: [{subject: "rus", score: 43}],
  individualAchievements: ["GTO"]
})
```

- Добавление ВУЗа (используется одна коллекция)

```
db.universities.insertOne({
  name: "LETI",
  city: "SPb",
  email: "someEmail@mail.ma",
  phone: "+7-983-321-23-43",
  description: "We are the best! (no)",
  faculties: ["First fac", "Second fac"],
  directionsOfStudy: ["1.1.1 The best direction in life" , "6.6.6 What the
F*ck"]
})
```

- Добавление работника (используется одна коллекция)

```
db.employees.insertOne({
  email: "someEmail@mail.ma",
  firstName: "Ivan",
  lastName: "Ivanov",
  patronymic: "Ivanovich",
  password: "some_pas",
```

```
DOB: "1980-01-01", universityId:"6173d3fbc85dd2433f25f039"})
```

- Поиск ВУЗа

```
db.universities.find({
  _id: ObjectId("6173cf0c985faf33df8ea21b")
})
```

- Удаление ВУЗа

```
db.universities.remove({
  _id: ObjectId("6173cf0c985faf33df8ea21b")
})
```

- Поиск ВУЗов по городу и по направлению

```
db.universities.find({
  city: "SPb",
  directionsOfStudy: "1.1.1 The best direction in life"
})
```

- Добавление заявления:

```
db.statements.insertOne({
  formOfEducation: "form",
  agreement: "yes",
  directionOfStudy: "dir",
  enrolleeId: "id"
})
```

- Поиск заявления по id абитуриента:

```
db.statements.find({
  enrolleeId: "id"
})
```

- Статистика количества абитуриентов, которые подали заявление:

```
db.statements.distinct("enrolleeId").length
```

- Статистика количества абитуриентов, которые не подали заявление:

```
db.enrollees.count() - db.statements.distinct("enrolleeId").length
```

- Статистика поданных заявлений в ВУЗ

```
db.statements.find({
  universityId: "id"
}).count()
```

3.2. Аналог модели данных для SQL СУБД

3.2.1. Графическое представление.



Рисунок 3.2.1.1 - Графическое представление модели данных для SQL СУБД

3.2.2. Описание назначений коллекций, типов данных и сущностей

Сущность "абитуриент" (Enrollee)

- id (text) (ключ) - идентификатор
- Email (text) – почта абитуриента
- Firstname (text) – имя абитуриента
- Lastname (text) – фамилия абитуриента
- Patronymic (text) – отчество абитуриента
- Password (text) – пароль
- DOB (date) – Дата рождения

- School (text) – школа
- City (text) – город

Сущность "результаты ЕГЭ" (EGEResults)

- id (text) (ключ) - идентификатор
- subject (text) - предмет
- score (integer) - количество баллов
- enrolleId (text) - id абитуриента (связь между Enrollee - EGEResults = 1 - n)

Сущность "индивидуальные достижения" (IndividualAchievements)

- id (text) (ключ) - идентификатор
- name (text) - название достижения
- enrolleId (text) - id абитуриента (связь между Enrollee - IndividualAchievements = 1 - n)

Сущность "заявления" (Statements)

- id (text) (ключ) - идентификатор
- formOfEducation (text) - форма обучения
- agreement (text) - согласие на зачисление
- directionOfStudy (text) - направление обучения
- enrolleId (text) - id абитуриента (связь между Enrollee - Statements = 1 - n)
- universityId (text) – id университета (связь между University - Statements = 1 - n)

Сущность "университет" (University)

- id (text) (ключ) - идентификатор
- Name (text) – название ВУЗа
- City (text) – город, в котором расположен ВУЗ
- Email (text) – почта ВУЗа
- Phone (text) - номер телефона ВУЗа
- Description (text) - описание ВУЗа

Сущность "факультеты" (Faculties)

- id (text) (ключ) - идентификатор
- name (text) - название факультета
- UniversityId (text) – id университета (связь между University - Faculties = 1 - n)

Сущность "направления подготовки" (Directions)

- id (text) (ключ) - идентификатор
- name (text) - название направления
- UniversityId (text) – id университета (связь между University - Directions = 1 - n)

Сущность "работник ВУЗа" (Employee)

- id (text) (ключ) - идентификатор
- Email (text) – почта работника
- Firstname (text) – имя работника
- LastName (text) – фамилия работника
- Patronymic (text) – отчество работника
- Password (text) – пароль
- DOB (date) – дата рождения
- UniversityId (text) – id прикрепленного университета (связь между University - Employee = 1 - n)

3.2.3. Оценка удельного объема информации, хранимой в модели

Пусть у нас совокупность полей (id, email, firstname, lastname, patronymic, password, DOB, universityId) сущности "Работник ВУЗа" занимает $W = (8 + 25 + 7 + 9 + 10 + 10 + 8 + 15 = 92$ байт) памяти, совокупность полей (id, name, city, description, email, phone) сущности "ВУЗ" занимает $U = (8 + 30 + 10 + 400 + 25 + 12 = 485$ байт), совокупность полей (id, name, universityId) сущности "Факультеты" занимает $F = (8 + 20 = 28$ байт) памяти, совокупность полей (id,

name, universityId) сущности "Направления подготовки" занимает $N = (8 + 30 = 38$ байт) памяти.

Помимо этого, среднее количество работников $W_{av} = 3$, количество ВУЗов $U_{av} = 300$, количество факультетов $F_{av} = 10$, количество направлений $N_{av} = 50$.

Тогда хранение данных о ВУЗе, факультетах, направлениях и работниках в реляционной БД будет занимать $U_{av} * (U + F * F_{av} + N * N_{av} + W_{av} * W) = 300 * (485 + 28 * 10 + 38 * 50 + 3 * 92) = 882\,300$ байт.

Пусть у нас совокупность полей (id, email, firstname, lastname, patronymic, password, DOB, school, city) сущности "Абитуриент" занимает $A = (8 + 25 + 7 + 9 + 10 + 10 + 8 + 15 + 10 = 102$ байт) памяти, совокупность полей (id, subject, score, enrolleId) сущности "Результаты ЕГЭ" занимает $R = (8 + 8 + 8 + 8 = 32$ байт), совокупность полей (id, name, enrolleId) сущности "Индивидуальные достижения" занимает $I = (8 + 10 + 8 = 36$ байт) памяти, совокупность полей (id, formOfEducation, agreement, directionOfStudy, enrolleId, universityId) сущности "Заявления" занимает $S = (8 + 10 + 3 + 30 + 8 + 8 = 67$ байт) памяти.

Помимо этого, среднее количество индивидуальных достижений $I_{av} = 2$, среднее количество результатов ЕГЭ $R_{av} = 3$, среднее количество абитуриентов $A_{av} = 6000$, среднее количество заявлений $S_{av} = 3$.

Тогда хранение данных об абитуриентах в реляционной БД будет занимать $A_{av} * (A + R_{av} * R + I_{av} * I + S_{av} * S) = 6000 * (102 + 3 * 32 + 2 * 36 + 3 * 67) = 2\,826\,000$ байт.

Всего хранение модели будет занимать $882\,300 + 2\,826\,000 = 3\,708\,300$ байт = 3.54 Мбайт.

Избыточность модели.

В таблицах, которые хранят факультеты и направления подготовки ВУЗов, будут храниться повторяющиеся данные. То же самое с результатами ЕГЭ, с

индивидуальными достижениями абитуриента и с заявлениями. Данную проблему можно исправить, создав таблицы, в которых будут храниться все возможные факультеты, направления, предметами в ЕГЭ.

Эти избыточности допущены для повышения скорости поиска и удобства.

Направление роста модели.

При создании нового абитуриента, помимо создания записи в таблице с абитуриентами, будут добавляться записи в "результаты ЕГЭ" и "индивидуальные достижения" (если абитуриент их указал). Также, когда создается новый работник, может создаваться еще одна запись в таблице с университетами.

3.2.4. Запросы к модели, с помощью которых реализуются сценарии использования

- Добавление абитуриента (Задействованы 3 таблицы и 3 запроса):

```
INSERT INTO Enrollee VALUES ('someemail@mail.ma', 'Ivan', 'Ivanov', 'Ivanovich',  
'pass', DATE '2000-12-17', 'school', 'spb'); INSERT INTO EGResults VALUES ('rus',  
89, enrolleeId); INSERT INTO IndividualAchievements VALUES ('GTO', enrolleeId),  
('GOLD MEDAL', enrolleeId);
```

- Добавление ВУЗа (Задействована 1 таблица и 1 запрос):

```
INSERT INTO University VALUES ('LETI', 'SPb', 'email@email.email',  
' +7-983-321-23-43');
```

- Добавление работника (Задействована 1 таблица и 1 запрос):

```
INSERT INTO Employee VALUES ('email@m.m', 'Ivan', 'Ivanov', 'Ivanovich', 'pass',  
DATE '1970-06-13', universityId);
```

- Поиск ВУЗа (Задействована 1 таблица и 1 запрос):

```
SELECT * FROM University WHERE University.id = id;
```

- Удаление ВУЗа (Задействована 1 таблица и 1 запрос):

```
DELETE * FROM University WHERE University.id = id;
```

- Поиск ВУЗов по городу и по направлению (Задействованы 2 таблицы и 1 запрос):

```
SELECT University.id FROM University, Directions WHERE University.city = city AND  
Directions.name = directionName;
```

- Добавление заявления (Задействована 1 таблица и 1 запрос):

```
INSERT INTO Statements VALUES ('formOfEducation', 'agreement', 'directionOfStudy',  
'enrolleeId');
```

- Поиск заявления по id абитуриента (Задействована 1 таблица и 1 запрос):

```
SELECT * FROM Students WHERE Students.id = id;
```

- Статистика количества абитуриентов, которые подали заявление (Задействована 1 таблица и 1 запрос):

```
SELECT COUNT( DISTINCT Statements.enrolleeId) FROM Statements;
```

- Статистика количества абитуриентов, которые не подали заявление (Задействована 1 таблица и 3 запрос):

```
SELECT (SELECT COUNT(*) FROM Statements) - (SELECT COUNT(DISTINCT  
Statements.enrolleeId) FROM Statements);
```

- Статистика поданных заявлений в ВУЗ (Задействована 1 таблица и 1 запрос):

```
SELECT COUNT(*) FROM Statements WHERE Statements.universityId = id;
```

3.3. Сравнение моделей

SQL модель занимает 3.54 Мбайт, а NoSQL – 2.96 Мбайт. Нереляционная база данных имеет меньший объем, для операции вставки некоторых сущностей необходимо использовать меньше запросов в сравнении с SQL. Также, например, для поиска результатов ЕГЭ определенного абитуриента по ФИО в SQL нам будет необходимо найти сначала ID абитуриента, а после найти его результаты по ID, в NoSQL мы можем сразу получить результаты, найдя

абитуриента. При этом, в нереляционной БД, к примеру, при удалении ВУЗа, сначала нужно удалить все связанные с ним записи, а в SQL это можно задать каскадное удаление связанных данных. Также проверка данных в нереляционной базе данных осуществляется только на стороне сервиса.

4. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

4.1. Краткое описание

Разработанное приложение хранит данные об университетах, абитуриентах, работниках приемной комиссии. Был реализован REST API, позволяющий производить регистрацию и аутентификацию студентов, получение информации об абитуриентах и ВУЗах, подачу заявлений в ВУЗы. Также был разработан UI дизайн приложения, а также частично реализован прототип интерфейса.

4.2. Схема экранов приложения

Экраны приложения см. на рис. 4.2.1-4.2.5.

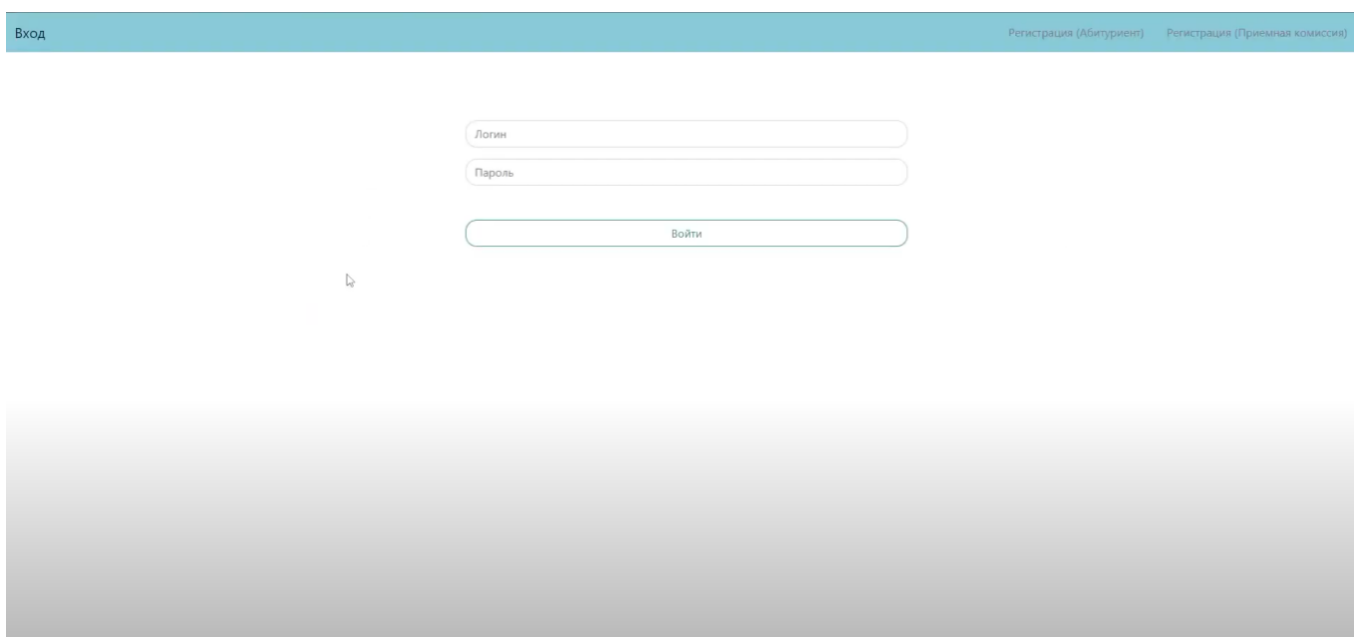



Рисунок 4.2.1 - Начальная страница

Регистрация абитуриента

Вход



Выберите файл

Файл не выбран

ЕГЭ +

Индивидуальные достижения +

Зарегистрироваться

Рисунок 4.2.2 - Регистрация абитуриента

ВУЗы

Профиль

Название

Город

Направление подготовки

Название

Город

Направление

Поиск

Рисунок 4.2.3 - Страница с ВУЗами

Поиск всех абитуриентов

Статистика

Абитуриенты, подавшие заявление в ваш вуз

Настройки

Фамилия

Имя

Отчество

Подал заявление

Нет

Город

Сумма баллов от

Поиск

№	ФИО	Дата рождения	Баллы ЕГЭ	Индив. достиж-я	Город	Учебное заведение	Подал заявление
1	Shishkin Ivan Viktorovich	01.01.1970	rus: 55 math: 90 physics: 80	some	spb	school	NET

Рисунок 4.2.4 - Страница с абитуриентами

Экспорт данных

Импорт данных

Рисунок 4.2.5 - Страница с кнопками для экспорта и импорта данных

4.3. Используемые технологии

СУБД: MongoDB

Сервер: Java Spring

Клиент: React

4.4. Ссылки на Приложение

GitHub: <https://github.com/moevm/nosql2h21-university>

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы был реализован веб-интерфейс для сбора данных абитуриентов и возможности подачи документов в ВУЗы. Для каждой из сущностей был реализован собственный интерфейс использования. В качестве базы данных использовалась MongoDB. Помимо этого, было произведено сравнение нереляционной модели данных и реляционной для этой задачи.

Для реализации данной задачи лучше подходит SQL модель, так как мы имеем структурированные данные, которые удобнее хранить в таблицах. Проверка данных не происходит на стороне базы данных, что увеличивает шанс хранения некорректных данных. Так как для создания записей в NOSQL требуется меньше операций.

В будущем будет необходимо задуматься о масштабировании базы данных. Также стоит добавить подсчет разных статистик для работников ВУЗа и абитуриентов (средний балл поступающего, количество подавших документы из определенного региона и т.д.).

ПРИЛОЖЕНИЯ

Документация по сборке и развертыванию приложения

С помощью Docker:

```
docker-compose down && docker-compose build --no-cache && docker-compose up
```

Либо просто:

```
docker-compose up
```

Инструкция для пользователя

Пользователь попадает на страницу с логином. Новому пользователю нужно зарегистрироваться, а старому - ввести данные и нажать на кнопку с входом. По умолчанию в базу данных занесены следующие пользователи:

Работник ВУЗа:

- email: employee1@email.com
- пароль: pass

Абитуриент:

- email: enrollee1@email.com
- пароль: pass

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация MongoDB // MongoDB URL: <https://docs.mongodb.com/>
(Дата обращения: 30.09.2021)
2. Документация React // React URL:
<https://ru.reactjs.org/docs/getting-started.html> (Дата обращения: 30.09.2021)
3. Документация Java // Java URL: <https://docs.oracle.com/en/java/> (Дата
обращения: 30.09.2021)
4. Документация Spring // Spring URL:
<https://docs.spring.io/spring-framework/docs/current/reference/html/> (Дата
обращения: 30.09.2021)