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

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

по дисциплине «Введение в нереляционные базы данных»

Тема: Сервис управления задачами

Студент гр. 0303 _____ Смирнов А.В.

Студентка гр. 0382 _____ Кривенцова Л.С.

Студентка гр. 0382 _____ Здобнова К.Д.

Преподаватель _____ Заславский М.М.

Санкт-Петербург

2023

ЗАДАНИЕ

Студенты

Смирнов А.В. группа 0303

Кривенцова Л.С. группа 0382

Здобнова К.Д. группа 0382

Тема проекта: Разработка приложения для управления задачами.

Исходные данные:

Необходимо реализовать приложение для управления задачи с помощью СУБД MongoDB.

Содержание пояснительной записки:

«Содержание», «Введение», «Сценарий использования», «Модель данных», «Разработанное приложение», «Выводы», «Приложения», «Литература»

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 20.09.2023

Дата сдачи реферата: 24.12.2023

Дата защиты реферата: 24.12.2023

Студент гр. 0303		Смирнов А.В.
Студентка гр. 0382		Кривенцова Л.С.
Студентка гр. 0382		Здобнова К.Д.
Преподаватель		Заславский М.М.

АННОТАЦИЯ

В рамках данного курса предполагалось разработать какое-либо приложение в команде на одну из поставленных тем. Была выбрана тема создания приложения для управления задачами с помощью СУБД MongoDB, так как это отличная возможность понять принцип работы документно-ориентированных баз данных для данной предметной области. Во внимание будут приниматься такие аспекты как производительность и удобство разработки. Найти исходный код и всю дополнительную информацию можно по ссылке: https://github.com/moevm/nosql2h23- todolist

ANNOTATION

This course was intended to develop some application in a team on one of the assigned topics. The topic of creating a task management application using MongoDB was chosen, as it is a great opportunity to understand how document-oriented databases work in this subject area. Aspects such as performance and usability of development will be taken into consideration. Find the source code and all additional information at: https://github.com/moevm/nosql2h23- todolist.

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

1.1 Актуальность решаемой проблемы

Цель работы - создать быстрое и удобное веб-приложения по созданию, хранению и обработке задач.

1.2 Постановка задачи

Сервис управления задачами должен решать следующие задачи:

- Позволять пользователям просматривать, создавать и редактировать задачи.

- Предоставлять пользователям инструменты для сбора и хранения информации о своих задачах.

1.3 Предлагаемое решение

Для решения поставленной задачи необходимо разработать web-приложение по управлению задачами.

1.4 Качественные требования к решению

Требуется разработать веб-приложение с использованием СУБД MongoDB, фреймворков Vue.js и Spring Boot.

2. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

2.1. Макеты UI

1. Экран профиля (Рис. 1)

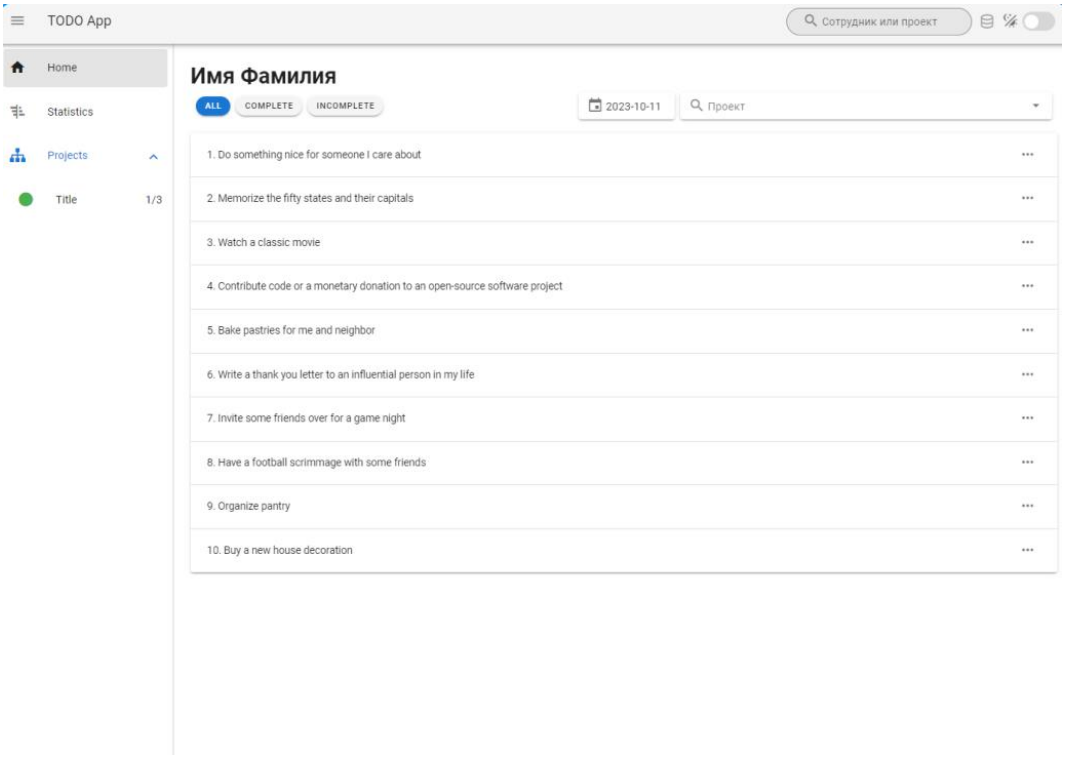

Рисунок 1 - Экран профиля пользователя

2. Экран проекта (Рис. 2)

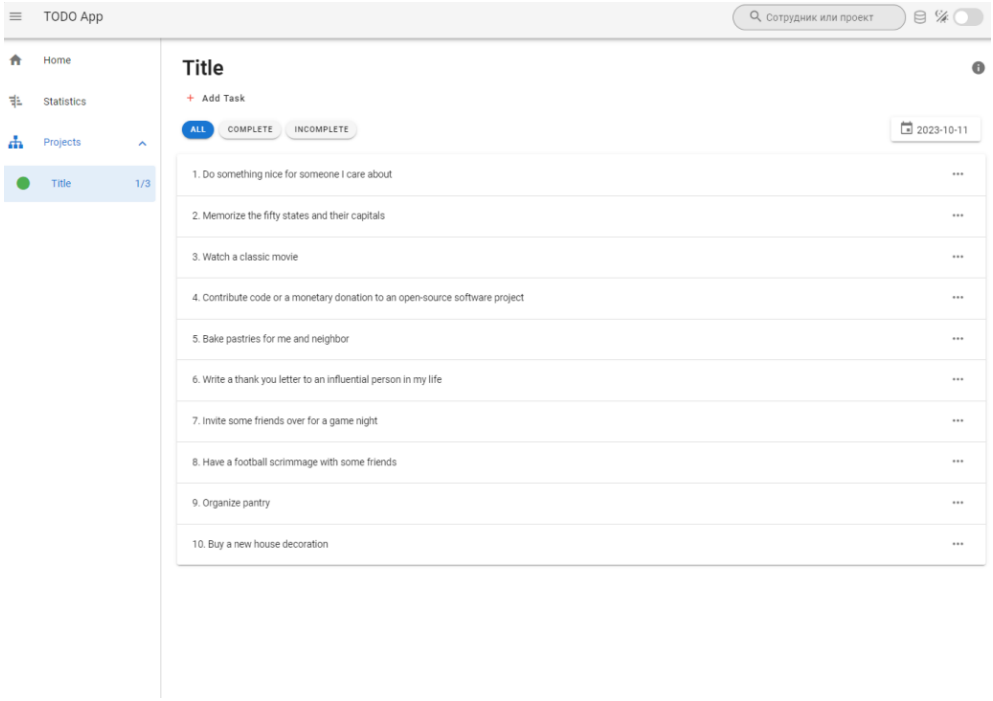

Рисунок 2 - Экран проекта

3. Экран просмотра статистики (Рис. 3)

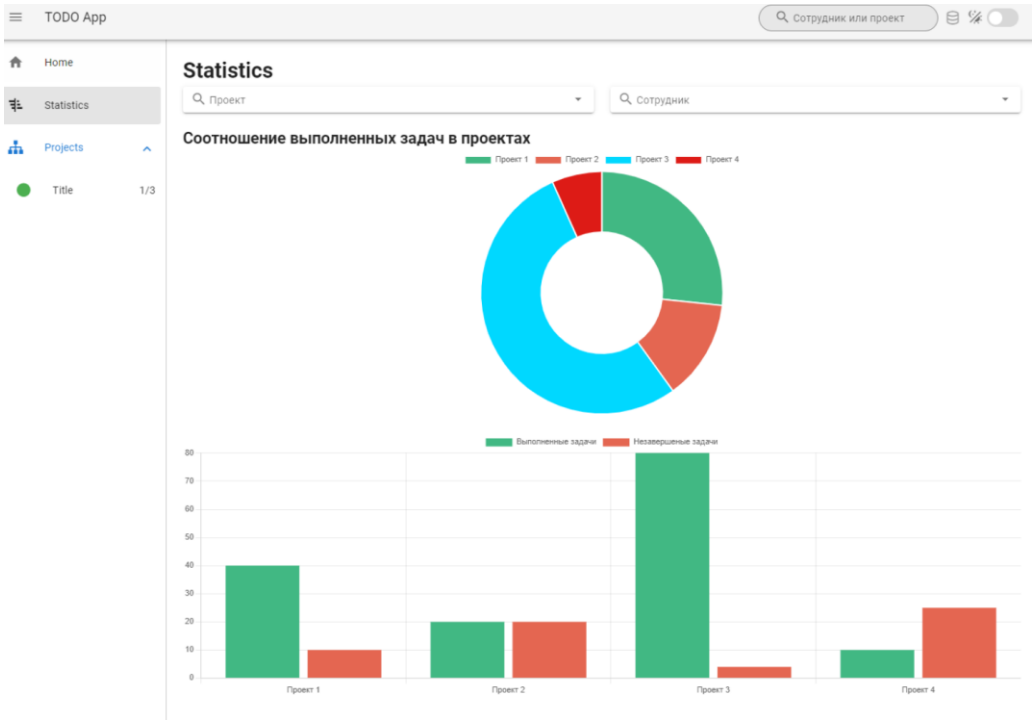

Рисунок 3 - Экран просмотра статистики

4. Экран входа в приложение (Рис. 4)

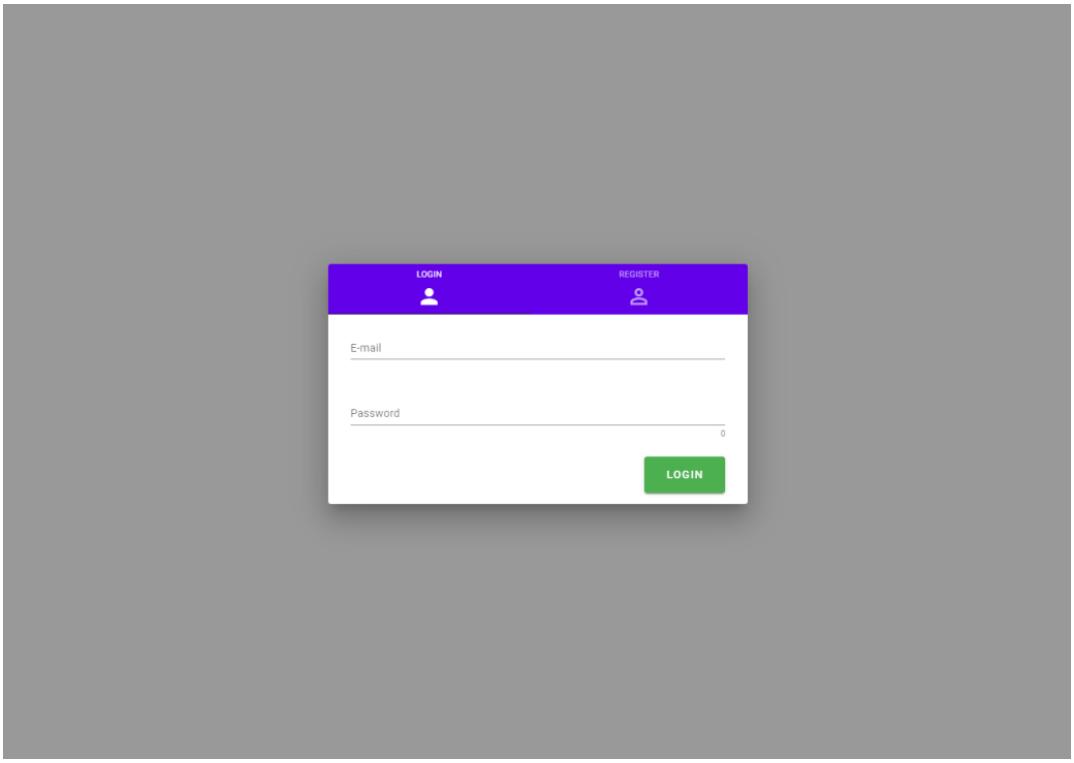

Рисунок 4 - Экран входа в приложение

5. Экран добавления задачи (Рис. 5)

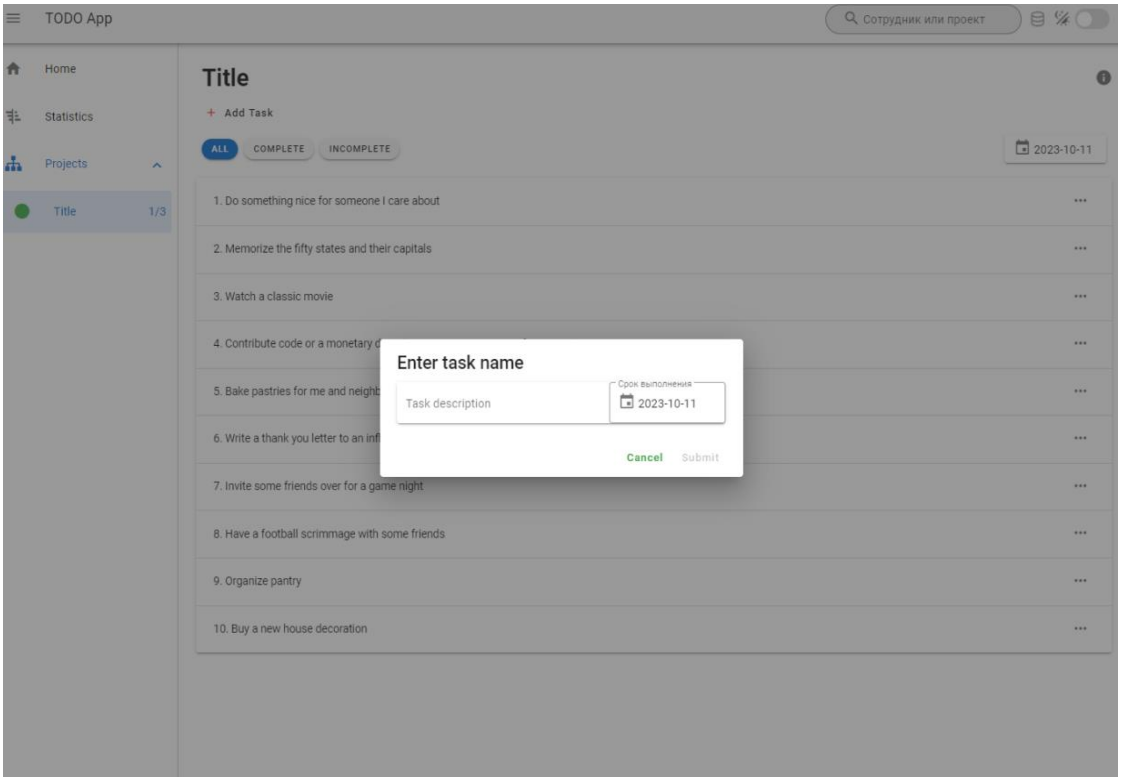

Рисунок 5 - Экран добавления задачи

6. Экран подтверждения добавления задачи (Рис. 6)

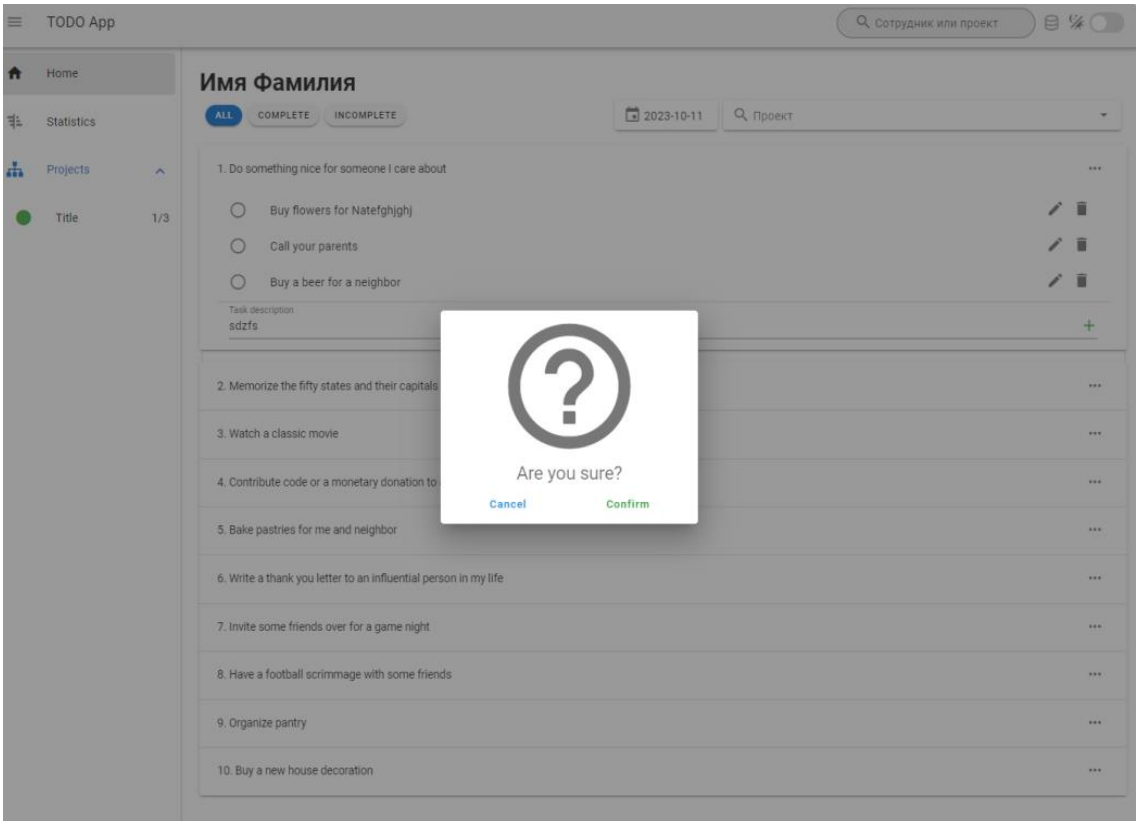

Рисунок 6 – Экран подтверждения добавления задачи

7. Экран редактирования задачи (Рис. 7)

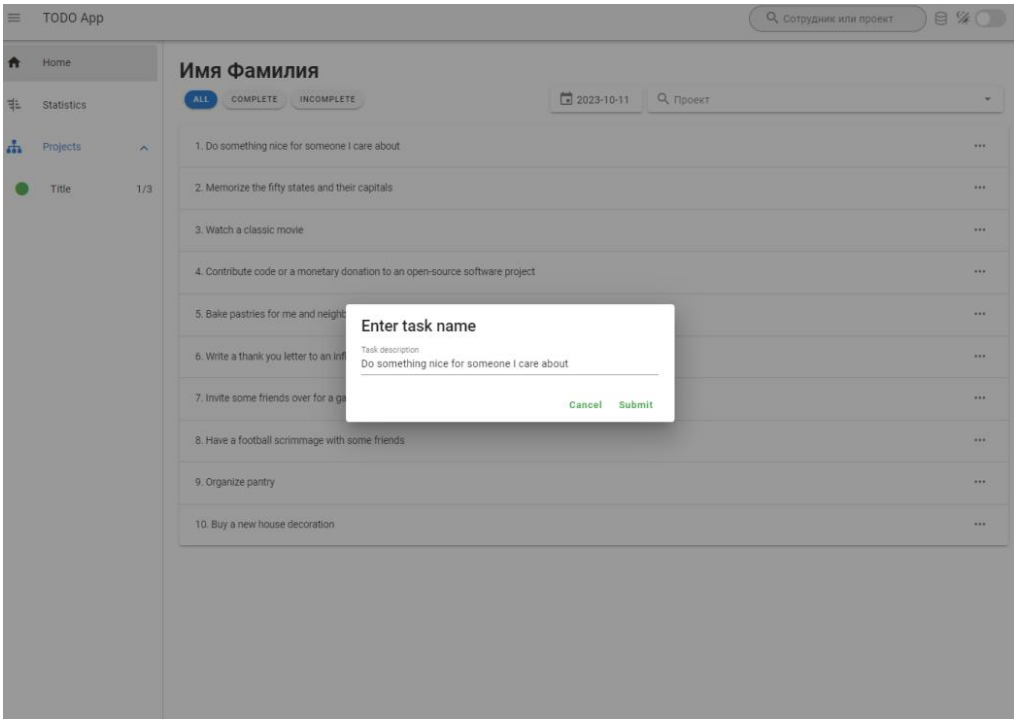

Рисунок 7 - Экран редактирования задачи

2.2. Сценарии использования для задачи

2.2.1. Импорт данных

2.2.1.1. Сценарий использования - “массовый импорт данных”

Действующее лицо – пользователь (администратор)

Предусловие:

- Пользователь авторизовался в системе.

Основной сценарий:

1. Пользователь видит кнопку “импорт”
2. Пользователь нажимает на иконку “импорт”
3. Появляется окно с выбором файла для импорта данных
4. Пользователь выбирает файл
5. Данные отправляются на серверную часть и успешно импортируются

Альтернативный сценарий:

1. Пользователь выбрал некорректный файл

2. Выводится сообщение о некорректном файле

2.2.1.2. Сценарий использования - “ручной импорт данных”

Действующее лицо – пользователь (администратор)

Предусловие:

- Пользователь авторизовался в системе.

Основной сценарий:

1. Пользователь нажимает на кнопку “Создать задачу”
2. Пользователь вводит данные в поля поля для названия, срока исполнения и исполнителя задачи.
3. Пользователь заполняет необходимые данные.
4. Пользователь нажимает на кнопку “Сохранить”
5. Задача успешно добавлена.

Альтернативный сценарий № 1:

1. Пользователь вводит не все необходимые данные/вводит их некорректно.
2. Кнопка “Сохранить” становится неактивной.
3. Пользователь заполняет данные правильно.

Альтернативный сценарий № 2:

1. Пользователь нажимает кнопку “Отмена”
2. Пользователь не добавляет задачу.

2.2.2. Представление данных

Действующее лицо - пользователь

Предусловие:

- Пользователь авторизовался в системе.

Основной сценарий:

1. Пользователь видит список, в котором представлены все задачи, доступные для него.

2.2.3. Анализ данных

Действующее лицо - пользователь

Предусловие:

- Пользователь авторизовался в системе.

Основной сценарий:

1. Пользователь нажимает на кнопку навигации “Анализ”
2. Пользователь успешно переходит на страницу статистики, на которой отображена информация о проектах и соотношениях задач в них.

2.2.4. Экспорт данных

Действующее лицо – пользователь (администратор).

Предусловие:

- Пользователь авторизовался в системе.

Основной сценарий:

1. Пользователь видит кнопку “экспорт”.
2. Пользователь нажимает на кнопку.
3. Система формирует соответствующий файл в формате JSON с информацией о проектах и отправляет на клиентскую сторону.
4. Файл скачивается на устройство пользователя.

2.3. Вывод относительно нашего функционала

В результате анализа сценариев использования и самой задачи приложения был сделан вывод, что в задаче преобладают операции чтения, поскольку импорт представляет собой запись информации в базу данных, а все остальные операции направлены на чтение данной информации.

3. МОДЕЛЬ ДАННЫХ

3.1. Нереляционная модель данных - MongoDB

3.1.1. Графическое представление данных

Графическое представление нереляционной базы данных представлено на рис. 9

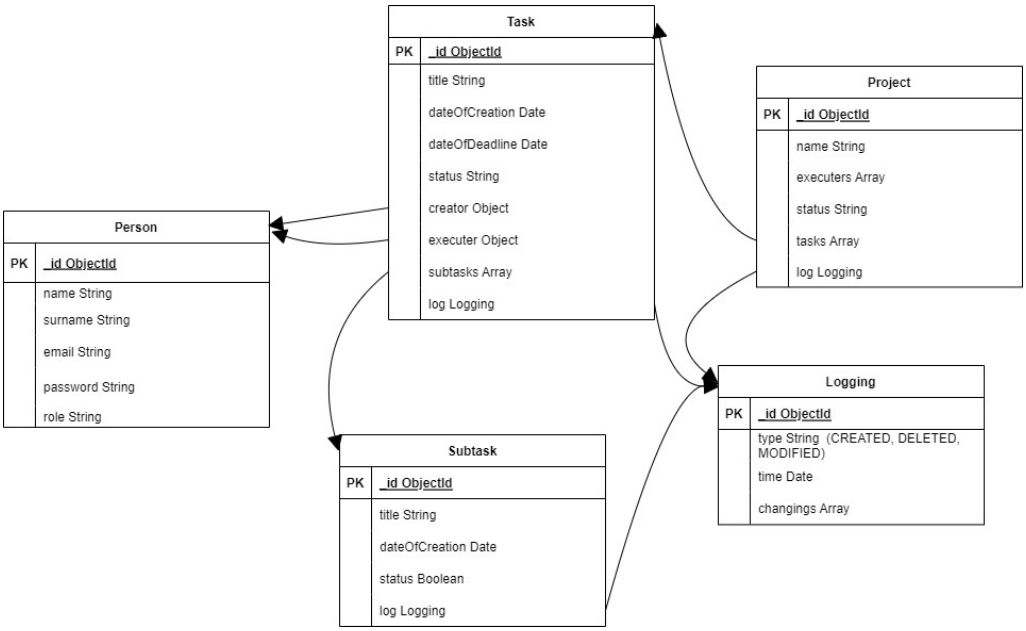

Рисунок 9 - Графическое представление нереляционной базы данных

3.1.2. Описание назначений коллекций, типов данных и сущностей

БД содержит 4 коллекции:

Person

- _id - уникальный идентификатор, 8b
- name - Имя, 50b
- surname - Фамилия, 50b
- email - почта пользователя, 50b
- password - пароль пользователя, 50b
- role - роль пользователя, 15b

Один элемент занимает 223b

Project

- _id - уникальный идентификатор, 8b
- name - название проекта, 100b
- tasks - список задач 127b
    - _id - уникальный идентификатор, 8b
    - title - заголовок задачи, 100b
    - dateOfDeadline - дата дедлайна, 4b
    - status - статус задачи, 15b
- executers - исполнители, 123b
    - _id - уникальный идентификатор, 8b
    - name - Имя, 50b
    - surname - Фамилия, 50b
    - status - статус проекта ("В процессе"/"завершен"), 15b

Один элемент занимает 373b

Task

- _id - уникальный идентификатор, 8b
- title - заголовок задачи, 100b
- dateOfCreation - дата создания задачи, 4b
- dateOfDeadline - дата дедлайна, 4b
- creater - создатель задачи, 108b
    - _id - уникальный идентификатор, 8b
    - name - Имя, 50b
    - surname - Фамилия, 50b
- executer - исполнитель задачи, 108b
    - _id - уникальный идентификатор 8b
    - name - Имя, 50b
    - surname - Фамилия, 50b

- status - статус задачи, 15b
- subTasks - список подзадач 73b
    - _id - уникальный идентификатор, 8b
    - title - заголовок подзадачи, 50b
    - status - статус, 15b

Один элемент занимает 420b

Subtask

- _id - уникальный идентификатор, 8b
- title - заголовок подзадачи, 50b
- dateOfCreation - дата создания подзадачи, 4b
- status - статус, 15b

Logging

- _id - уникальный идентификатор, 8b
- type - тип изменения, 50b
- date - дата внесения изменений, 4b
- changes - внесённые изменения, 50b

Один элемент занимает 112b

3.1.3. Оценка удельного объема информации, хранимой в модели

Моделирование системы с 20 пользователями, 10 проектами, 100 задач, 150 подзадач и 400 логов - 106540b = 104 Kb - это фактический объем информации.

Посчитаем чистый объем: 20 * 267 + 10 * 420 + 100 * 460 + 150 * 90 + 400 * 150 = 129040b = 126 Kb

Выразим объем модели через количество задач, на каждый проект приходится по 30 задач, на каждую задачу - 2 подзадчи на каждый проект - по 7 человек и 100 логов. Тогда получим линейную зависимость, равную

$$V_d = 373 + p/30(420) + p/60(70) + p/100(112) = 390p$$

Избыточность модели равна:

$$V_d = \frac{373 + p/30(420) + p/60(70) + p/100(112)}{123 + p/30(420) + p/60(70) + p/100(112)} = \frac{390}{140} = 2.8$$

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

Поиск пользователя при авторизации:

```
db.user.findOne({email:email, password: password})
```

Поиск всех актуальных проектов:

```
db.project.find({$where: function(){
  return obj.status == "ACTIVE"
}})
```

Поиск задач с близким дедлайном:

```
db.task.find({
  $expr: { $lt: [
      { $subtract: ["$dateOfDeadline", new Date()] },]}})
```

Поиск всех выполненных задач:

```
db.task.find({$where: function(){
  return obj.status == "COMPLETED"
}})
```

Поиск всех задач, назначенных определенному работнику:

```
var personId = ObjectId("yourPersonId");
db.Person.aggregate([
  {$match: {_id: personId}},
  {$lookup: {from: "Task", localField: "_id", foreignField:
"executer", as: "tasks"}},
  {$project: {_id: 1,name: 1, surname: 1, email: 1, tasks:
"$tasks"}}]);
```

Удалить задачу по её названию:

```
db.task.delete({name: name})
```

3.2. Реляционная модель данных

3.2.1. Графическое представление данных

Графическое представление реляционной базы данных представлено на рис. 10

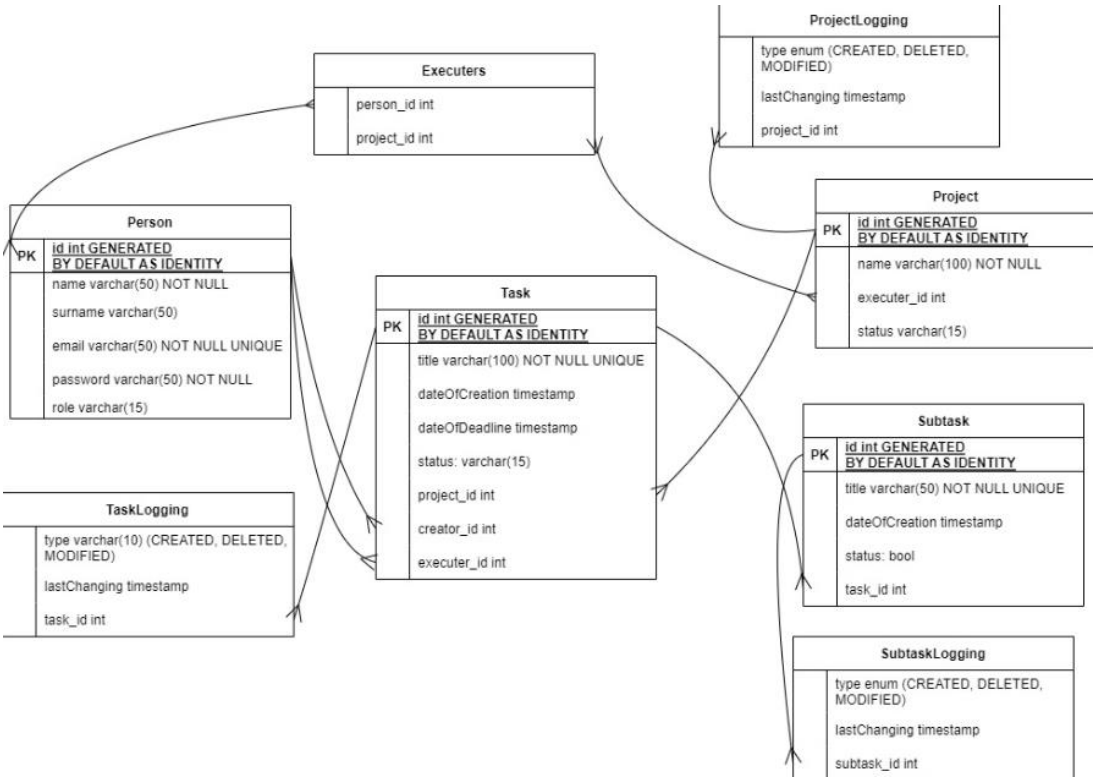

Рисунок 10 - графическое представление реляционной базы данных

3.2.2. Описание назначений коллекций, типов данных и сущностей

Person:

- id: уникальный идентификатор int64 - 8b
- name: имя varchar (50 символов) - 50b
- surname: фамилия varchar (50 символов) - 50b
- email: электронная почта varchar (50 символов) - 50b
- password: пароль varchar (50 символов) - 50b
- role: роль varchar (15 символов) - 15b Один элемент занимает 223b

Project:

- id: уникальный идентификатор int64 - 8b
- name: название varchar (100 символов) - 100b
- status: статус varchar (15 символов) - 15b Один элемент занимает 123b

ProjectLogging:

- type: тип изменения varchar(10 символов) (CREATED/DELETED/MODIFIED) - 10b
- lastChanging: время изменения timestamp - 4b
- subtask_id: уникальный идентификатор проекта int64 - 8b Один элемент занимает 22

Task:

- id: уникальный идентификатор int64 - 8b
- title: заголовок varchar (100 символов) - 100b
- dateOfCreation: дата создания timestamp - 4b
- dateOfDeadline: дата дедлайна timestamp - 4b
- project_id: уникальный идентификатор проекта int64 - 8b
- creater_id: уникальный идентификатор создателя int64 - 8b
- executer_id: уникальный идентификатор исполнителя int64 - 8b
- status: статус varchar (15 символов) - 15b Один элемент занимает 155b

TaskLogging:

- type: тип изменения varchar(10 символов) (CREATED/DELETED/MODIFIED) - 10b
- lastChanging: время изменения timestamp - 4b
- task_id: уникальный идентификатор задачи int64 - 8b Один элемент занимает 22b

Subtask:

- id: уникальный идентификатор int64 - 8b
- title: заголовок varchar (50 символов) - 50b
- dateOfCreation: дата создания timestamp - 4b
- task_id: уникальный идентификатор задачи int64 - 8b
- status: статус bool - 1b Один элемент занимает 71b

SubtaskLogging:

- type: тип изменения varchar(10 символов) (CREATED/DELETED/MODIFIED) - 10b
- lastChanging: время изменения timestamp - 4b
- subtask_id: уникальный идентификатор подзадачи int64 - 8b Один элемент занимает 22b

Executers

- person_id: уникальный идентификатор исполнителя int64 - 8b
- project_id: уникальный идентификатор іпроекта nt64 - 8b Один элемент занимает 16b

3.2.3. Оценка удельного объема информации, хранимой в модели

Моделирование системы с 20 пользователями, 10 проектами, 100 задач и 150 подзадач с 400 записей логов - 35176 = 34,35 Kb - это фактический объем информации.

Посчитаем чистый объем: 20 * 223 + 10 * 123 + 100 * 155 + 150 * 71 + 400 * 22 + 30 * 16 = 41120b = 40,1 Kb.

Избыточность модели равна:

$$V_d = \frac{123 + p/30(155) + p/60(71) + p/100(22)}{115 + p/30(155) + p/60(71) + p/100(22)} = \frac{129}{121} = 1.07$$

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

Поиск пользователя при авторизации:

```
SELECT * FROM Person WHERE email = email AND password = password;
```

Поиск всех актуальных проектов:

```
SELECT * FROM Project WHERE status = "IN PROGRESS";
```

Поиск задач с близким дедлайном:

```
SELECT * FROM Task WHERE dateOfDeadline > NOW() AND dateOfDeadline < DATE_ADD(NOW(), INTERVAL 24 HOUR);
```

Поиск всех выполненных задач:

```
SELECT * FROM Task WHERE status = "COMPLETED"
```

Поиск всех задач, назначенных определенному работнику:

```
SELECT Task.*
FROM Task
JOIN Person ON Task.executer = Person.id
WHERE Person.email = 'specific_email@example.com'';
```

Удалить задачу по её названию:

```
DELETE FROM Task WHERE title = 'specific_title';
```

3.3. Сравнение моделей

3.3.1. Удельный объем информации

Чистый объем для SQL-модели больше чем в два раза меньше чистого объема нереляционной модели, что происходит из-за дублирования данных. Однако в нашем случае, где ключевыми факторами являются скорость выполнения и эффективная обработка запросов от пользователей, более предпочтительным вариантом будет использование нереляционной модели.

3.3.2. Запросы по отдельным юзкейсам

В нескольких запросах у SQL использовано несколько таблиц, так из-за особенности хранения данных (также в реляционной модели есть дополнительная таблица, реализующая связь многие-ко-

многим), а в MongoDB - 1 коллекция, так как данные вложены друг в друга и не требуется объединение таблиц.

3.3.3. Вывод

В данном случае лучше использовать MongoDB, что показало сравнение двух моделей.

4. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

4.1. Краткое описание

Back-end представляет из себя REST API web-приложение, реализована авторизация, аутентификация пользователя, реализована две роли – ROLE_ADMIN и ROLE_USER, добавлены тестовые данные в базу данных (пользователи, проекты, задачи и подзадачи), написаны контроллеры для получения данных из базы данных, работой с ними, их фильтрацией, вывода статистик и массового импорта/экспорта.

Front-end представляет собой web-приложение, которое использует API сервера и отображает данные для пользователя. Реализованы формы регистрации и аутентификации, главная страницы для отображения проектов, задач для каждого пользователя, вывода статистики.

Клиент и сервер общаются по средством обмена JSON-объектами, настроена авторизация через JWT токены, настроены CORS.

Серверная часть собирается в докере на порте 8080, клиентская локально – на 8081.

Front-end основан на фреймворке Vue.js с использованием UI библиотеки компонентов Vuetify.

4.2. Использованные технологии

БД: MongoDB.

Back-end: Java, Spring Boot 2.7.24, Spring Security JWT, Docker.

Front-end: JavaScript, CSS, Vue.js, Vuetify, Docker.

4.3. Снимки экрана приложения

Снимки экрана приложения представлены на рис. 11-19.

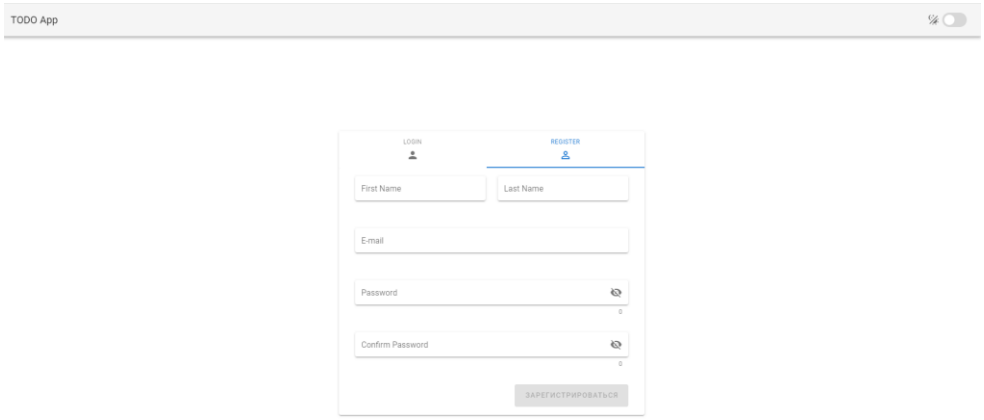

Рисунок 11 – Экран регистрации

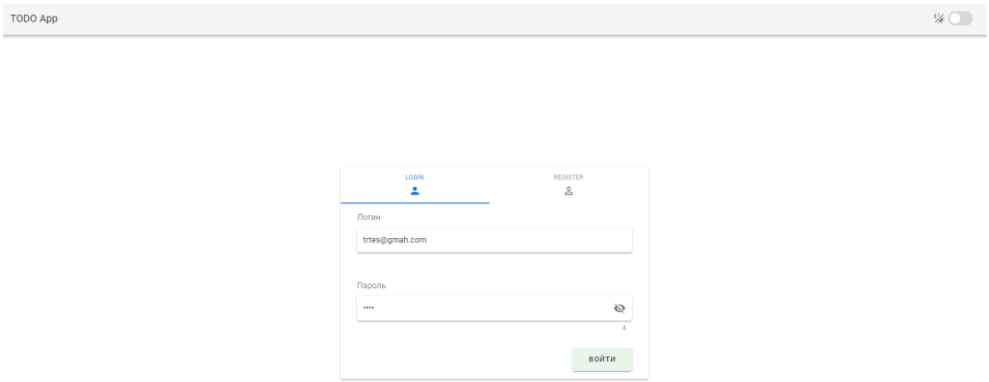

Рисунок 12 – Экран авторизации

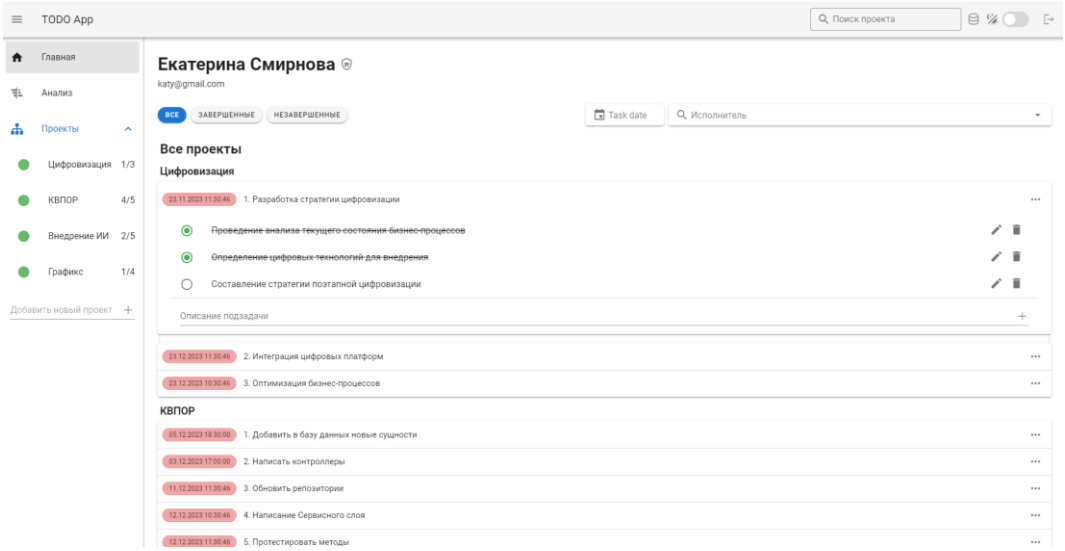

Рисунок 13 – Экран профиля пользователя

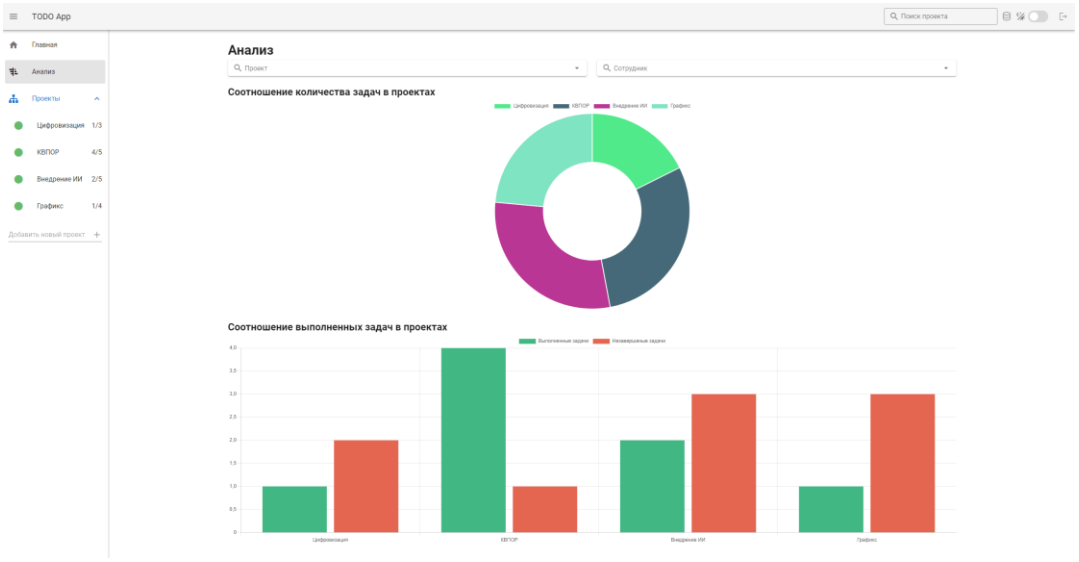

Рисунок 14 – Экран анализа

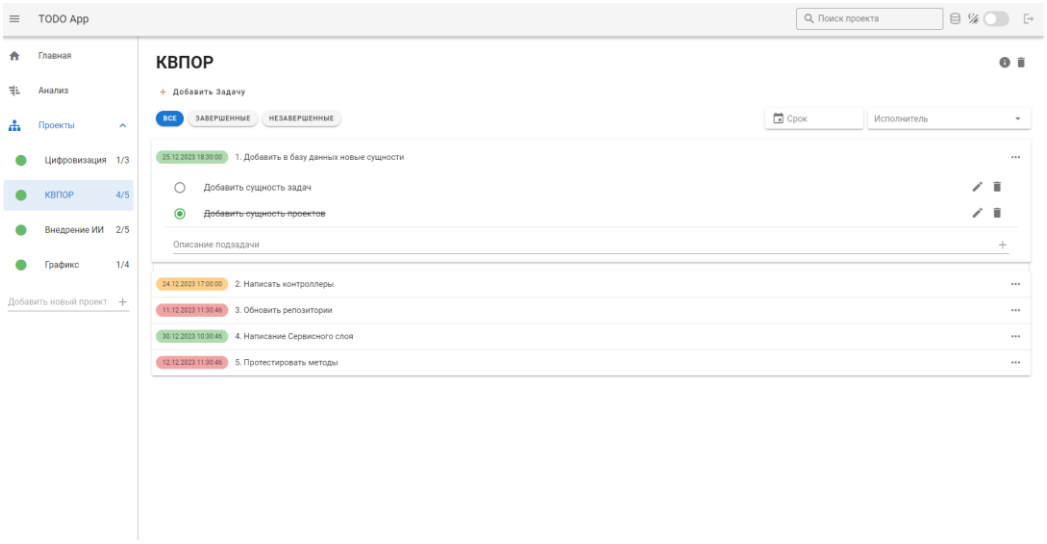

Рисунок 15 – Экран проекта

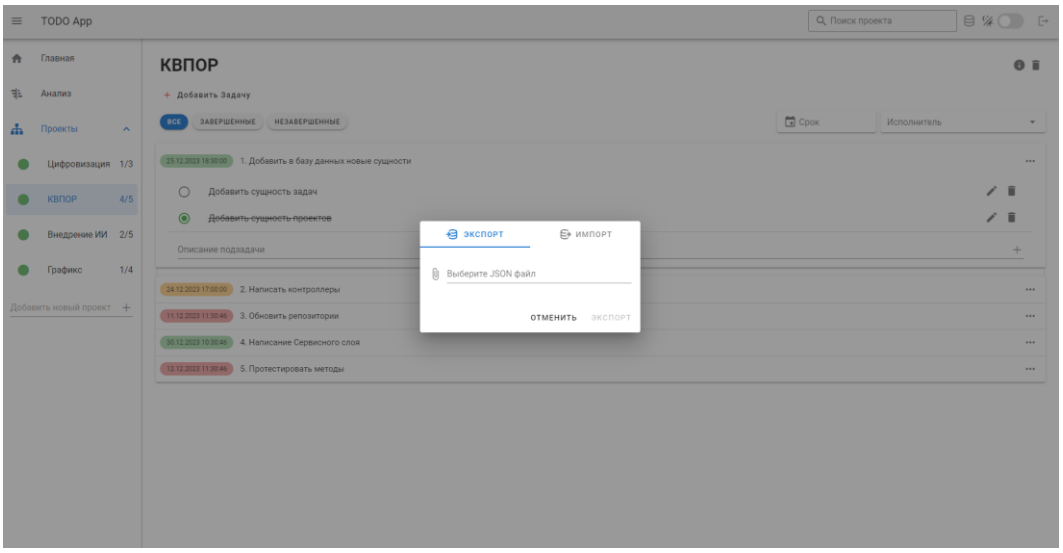

Рисунок 16 – Экран импорта\экспорта

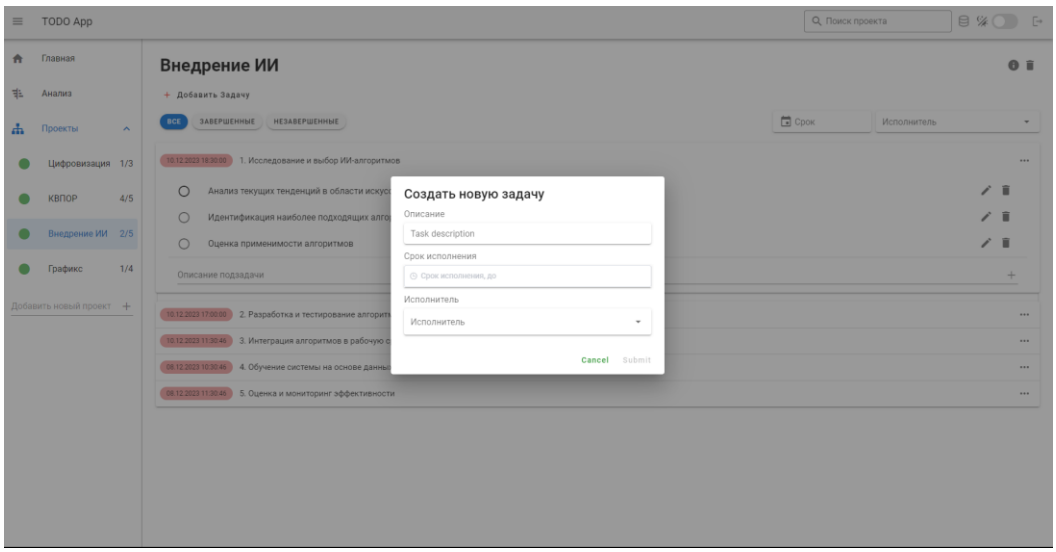

Рисунок 17 – Экран добавления задачи

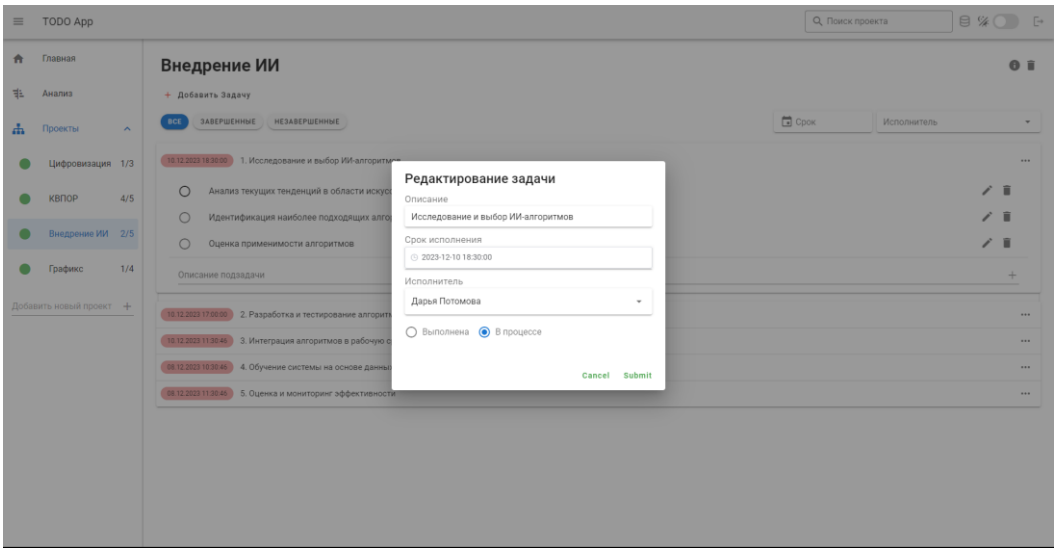

Рисунок 18 – Экран редактирования задачи

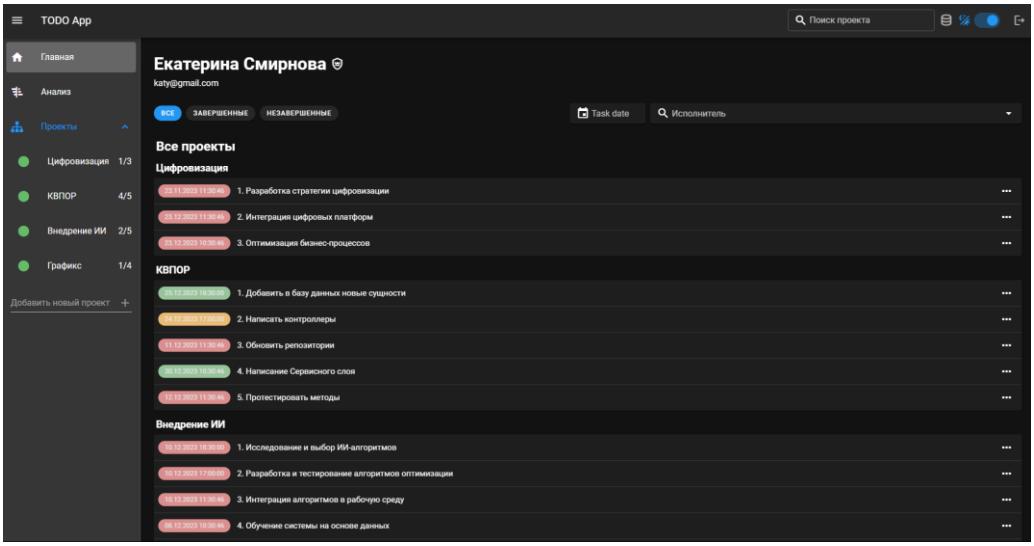

Рисунок 19 – Экран просмотра задач к выполнению

5. ВЫВОДЫ

5.1. Достигнутые результаты

В ходе работы было разработано приложение сервиса управления задачами с использованием нереляционной базы данных MongoDB. Была реализована вся заявленная функциональность: корректная работа с MongoDB, управление проектами, задачами, их подзадачами, вывод статистики по проектам и массовый импорт\экспорт. Добавлена возможностью локального разворота приложения с помощью docker-compose.

5.2. Недостатки и пути для улучшения полученного решения

В MongoDB нет встроенной механики для автоматического обновления вложенных документов. При изменении поля в Subtask, приходилось вручную обновить соответствующий Subtask внутри Task, а также внутри Project, а затем сохранить изменения в базе данных. Вложенные обходы можно было бы решить хранением в сущности id документа, а не полностью весь объект. Это бы упростило процесс управления обновлениями.

5.3. Будущее развитие решения

Добавление возможности установки приоритетов для каждой задачи, а также функционал для оставления комментариев к задачам для обсуждения деталей или предоставления обратной связи. Расширить приложение можно добавлением автоматической отправкой уведомлений о новых задач на почту.

6. ПРИЛОЖЕНИЯ

6.1. Документация по сборке и развертыванию приложения

1. Склонировать репозиторий (указан в списке литературы)

2. Зайти в папку ./ nosql2h23-todolist/

3. Установить docker и docker-compose.

4. Запустить локальный разворот приложения через docker-compose с помощью следующих команд:

- sudo docker-compose build --no-cache
- sudo docker-compose up –d

5. Перейти в любом удобном браузере по адресу: localhost:8081

6.2. Инструкция для пользователя

1. Используя свой логин и пароль войти в систему на странице авторизации ИЛИ зарегистрироваться на странице регистрации, после чего авторизоваться.

2. Обратиться к администратору для назначения задач.

7. ЛИТЕРАТУРА

1. Ссылка на github проекта - https://github.com/moevm/nosql2h23-todolist
2. Документация MongoDB - https://www.mongodb.com/
3. Документация Vue.js - https://v2.vuejs.org/
4. Документация Spring-Boot - https://spring.io/projects/spring-boot/
5. Документация Docker - https://www.docker.com/