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 для организации ремонтов в ВУЗе

Студент гр. 1384 _____ Феопентов А.Ю.

Студент гр. 1384 _____ Галенко А. С.

Студент гр. 1384 _____ Сочков И.С.

Преподаватель _____ Заславский М.М.

Санкт-Петербург

2024

ЗАДАНИЕ

Студент Феопентов А.Ю.
Студент Галенко А.С.
Студент Сочков И.С.
Группа 1384

Тема: Сервис для организации ремонтов в ВУЗе

Исходные данные:
Необходимо реализовать веб-приложение для организации ремонтов в ВУЗе с использование СУБД MongoDB.

Содержание пояснительной записки:
«Содержание»
«Введение»
«Сценарии использования»
«Модель данных»
«Разработанное приложение»
«Выводы»
«Приложения»
«Список использованных источников»

Предполагаемый объем пояснительной записки:
Не менее 72 страниц.

Дата выдачи задания: 05.09.2024

Дата сдачи реферата: 22.12.2024

Дата защиты реферата: 22.12.2024

Студент	_____	Феопентов А.Ю.
Студент	_____	Галенко А. С.
Студент	_____	Сочков И.С.
Преподаватель	_____	Заславский М.М.

АННОТАЦИЯ

В рамках ИДЗ разработано веб-приложение, представляющее собой сервис для организации ремонтов в ВУЗе. Приложение предоставляет возможность управлять проектами ремонта, включая планирование этапов, задач и сроков их выполнения. Пользователи могут учитывать потребности в материалах, трудозатратах и переселении сотрудников, а также оценивать и минимизировать риски. Функционал приложения охватывает создание, редактирование и мониторинг всех аспектов ремонтных работ. Реализована система фильтрации и поиска проектов по различным критериям.

Для разработки использованы технологии Vue.js, JavaScript, СУБД MongoDB, Python (FastAPI), Docker.

Найти исходный код можно по ссылке: repair

SUMMARY

As part of the IDZ, a web application has been developed, which is a service for organizing repairs at the university. The application provides the ability to manage repair projects, including planning stages, tasks and deadlines. Users can take into account the needs for materials, labor costs and relocation of employees, as well as assess and minimize risks. The functionality of the application covers the creation, editing and monitoring of all aspects of repair work. A system for filtering and searching projects by various criteria has been implemented.

Vue.js, JavaScript, СУБД MongoDB, Python (FastAPI), Docker technologies were used for development.

The source code can be found at the link: repair

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

1.1. Актуальность проблемы

Организация ремонтных работ в образовательных учреждениях требует тщательного планирования, учета ресурсов и соблюдения сроков. Текущие подходы часто основаны на разрозненных таблицах и документах, что приводит к увеличению времени на координацию, рискам ошибок и отсутствию прозрачности в процессах. Это особенно критично в вузах, где параллельно с ремонтами необходимо учитывать график учебной деятельности и переселение сотрудников.

1.2. Постановка задачи

Разработать цифровой сервис, позволяющий систематизировать и автоматизировать процессы управления ремонтными работами. Сервис должен включать инструменты:

- для создания и планирования проектов,
- управления этапами, задачами, ресурсами и рисками,

а также предоставлять прозрачный доступ к информации для всех участников.

1.3. Предлагаемое решение

Для реализации создается веб-приложение с использованием Vue.js, JavaScript, СУБД MongoDB, Python (FastAPI), Docker. Приложение поддерживает фильтрацию, поиск и редактирование.

1.4. Качественные требования к решению

Решение должно быть удобным, производительным, надёжным и легко расширяемым.

2. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

2.1. Макет UI

Ниже представлены макеты страниц приложения.

Рисунок 1- Страница авторизации и регистрации

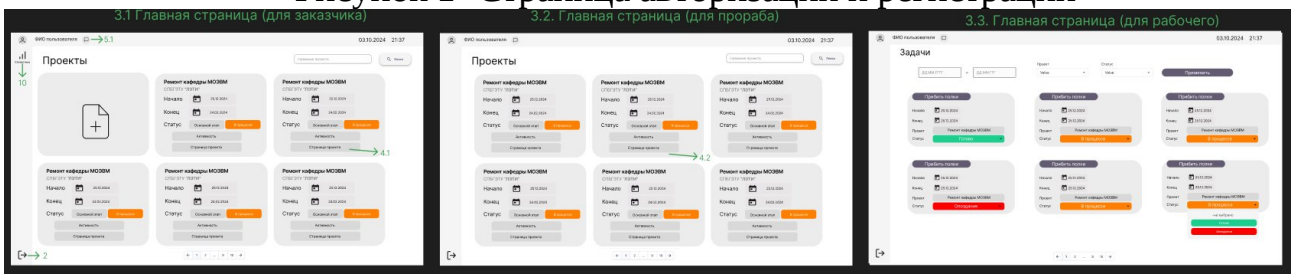

Рисунок 2 - Главная страница

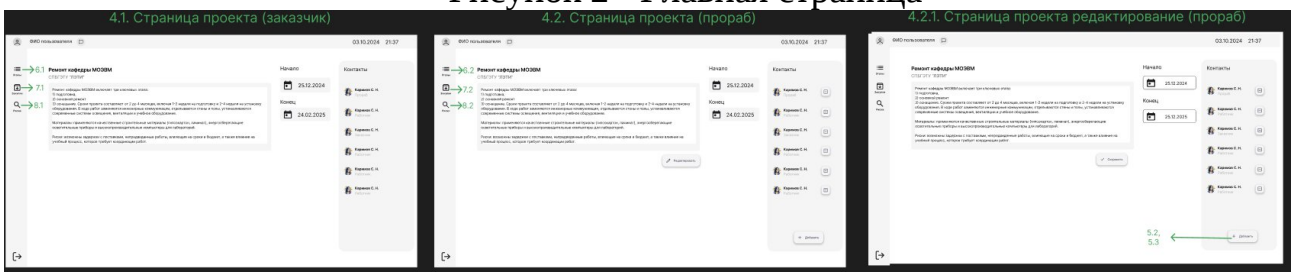

Рисунок 3 - Страница проекта

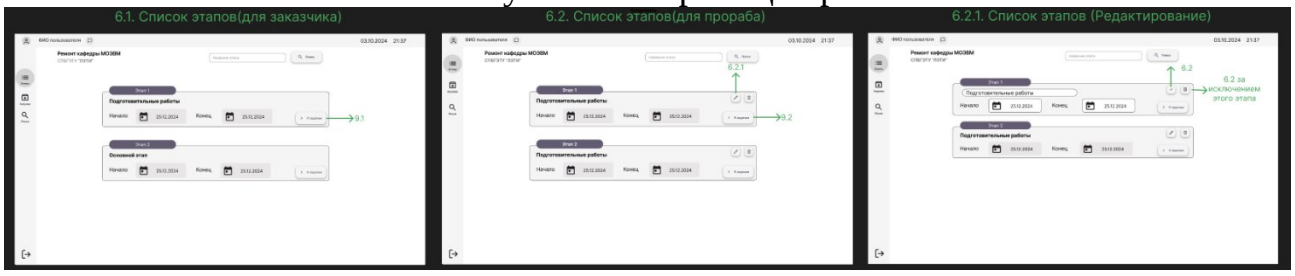

Рисунок 4 - Страница этапов

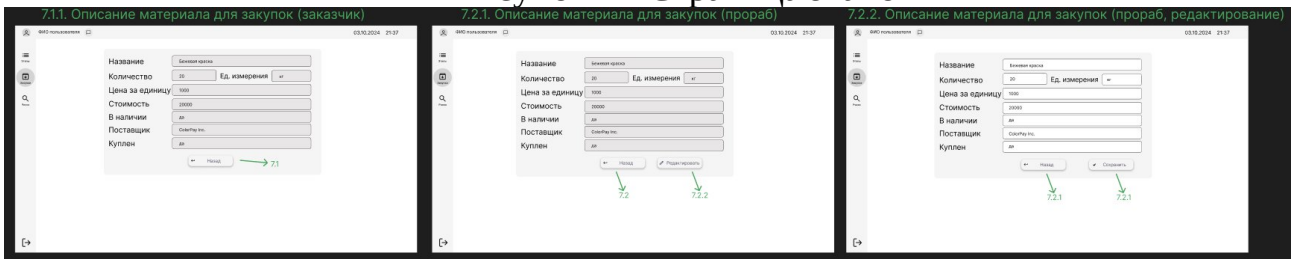

Рисунок 5 - Страница закупок

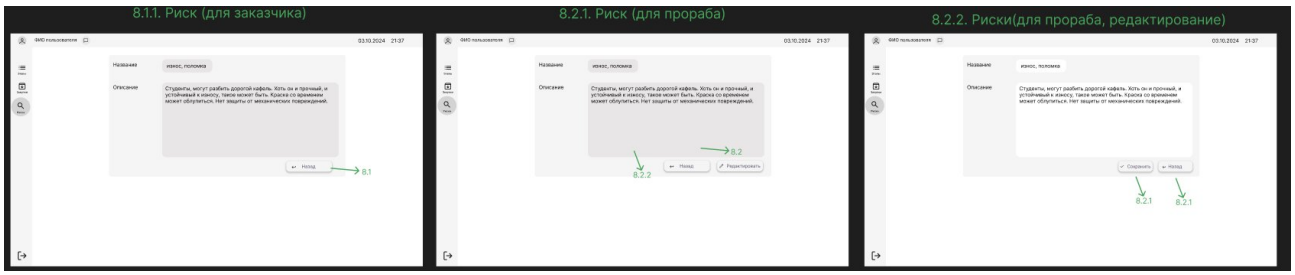

Рисунок 6 - Страница рисков

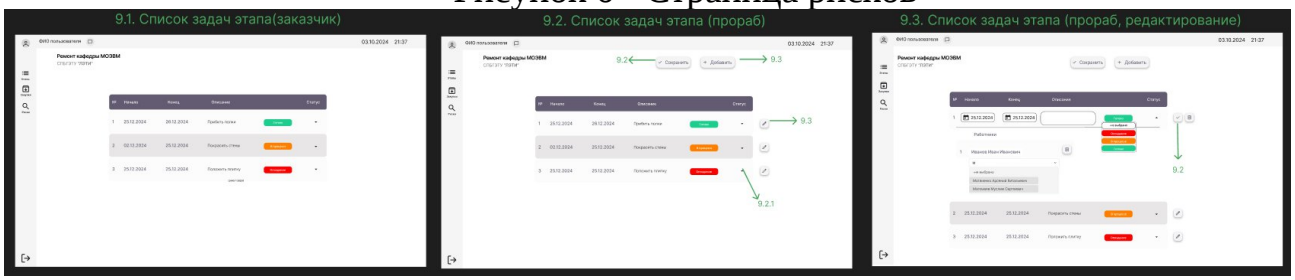

Рисунок 7 - Страница списка задач

Рисунок 8 - Страница задачи

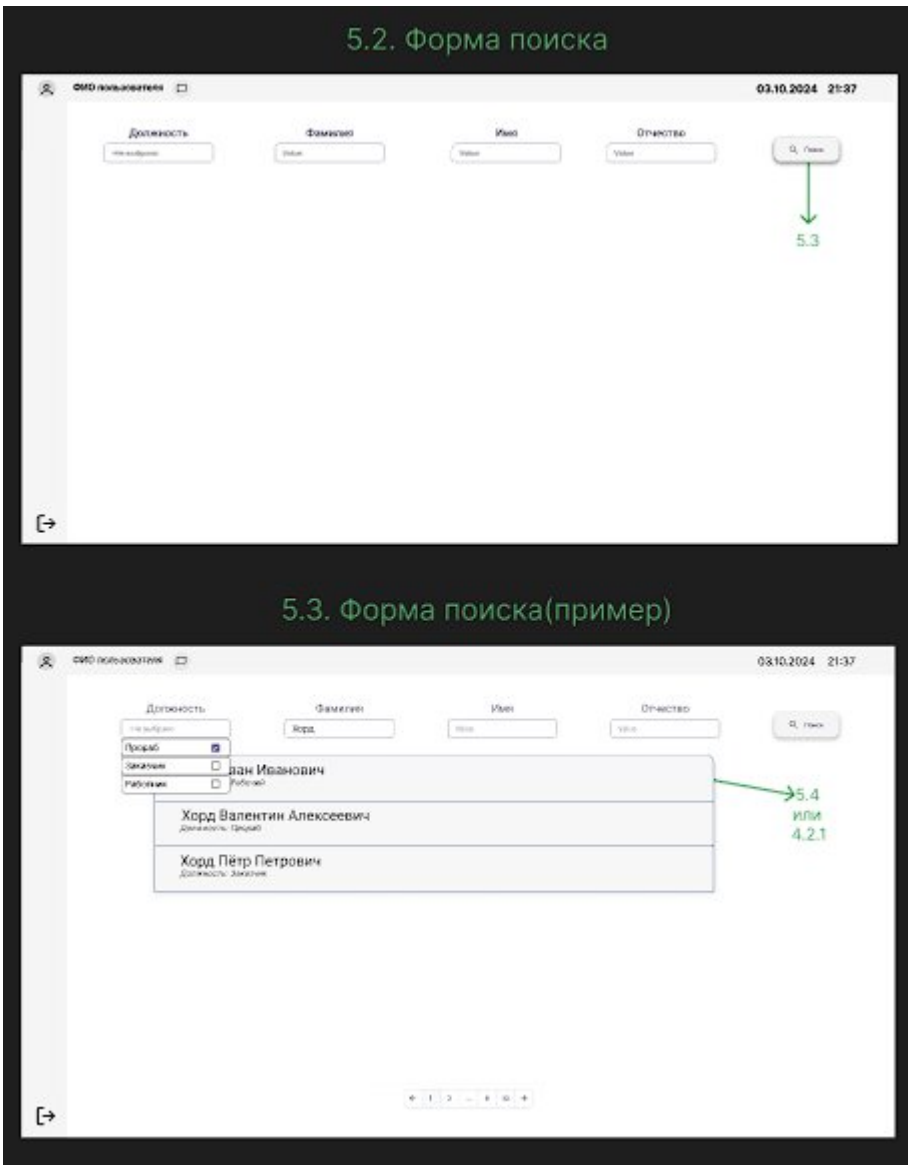

Рисунок 9 - Страница поиска

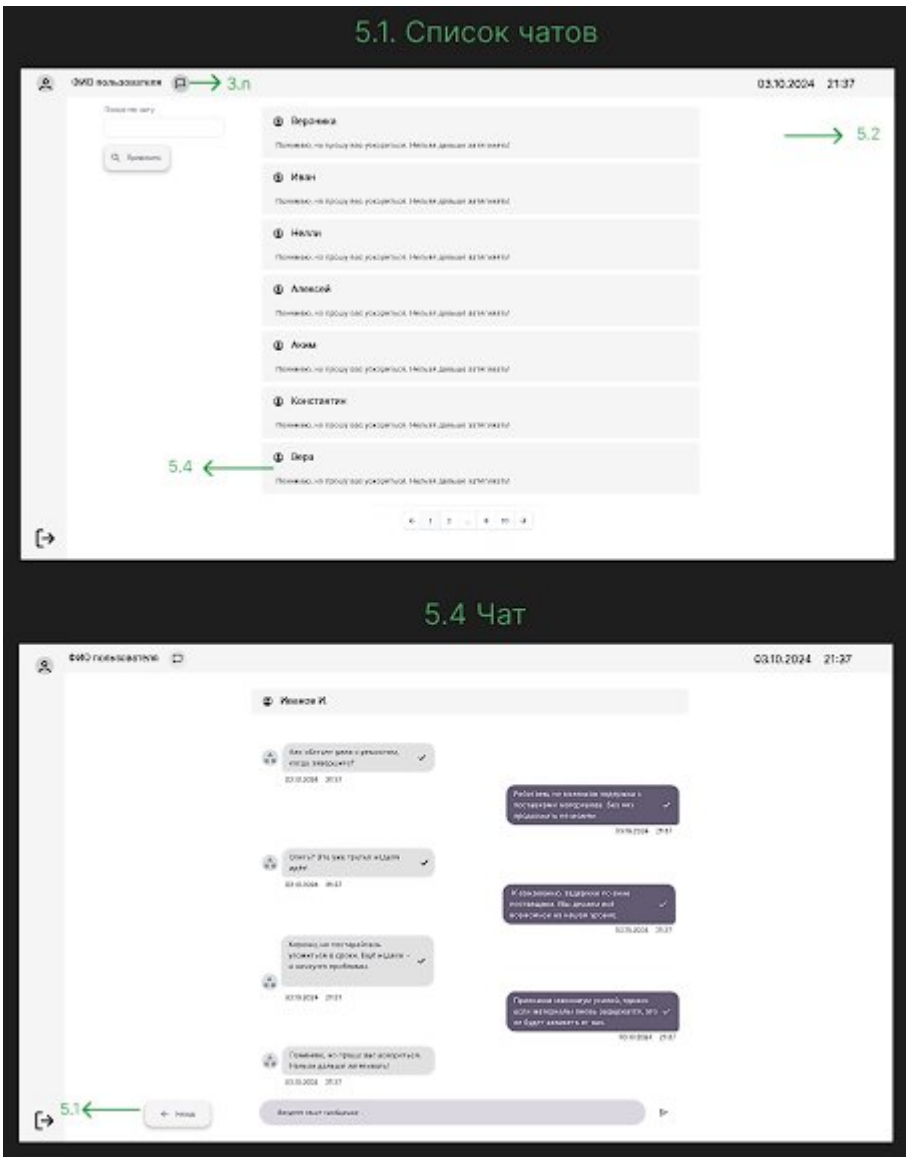

Рисунок 10 - Страница чата

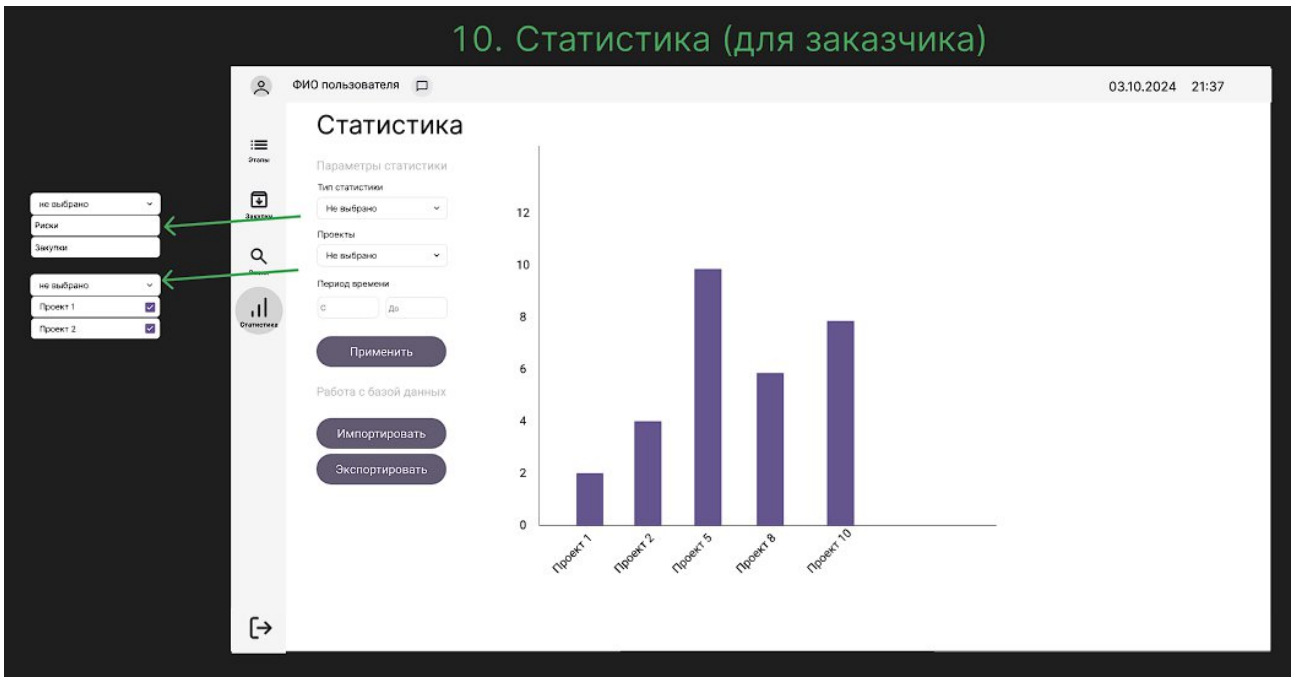

Рисунок 11 - Страница статистики

2.2. Сценарий использования для импорта данных

А) **Сценарий использования: Регистрация пользователя**

Цель:

- Зарегистрировать пользователя и выдать ему роль.

Действующие лица:

- Администратор, пользователь.

Основной сценарий:

1. Администратор получает данные пользователя, которого нужно зарегистрировать.
2. Администратор открывает страницу с регистрацией и регистрирует пользователя, назначает ему роль (страница 1).
3. Администратор высылает логин и пароль пользователю.

Альтернативный сценарий:

1. Администратор получает сообщение от пользователя, что тот забыл пароль.

2. Администратор восстанавливает пароль, или создает новы логин с паролем.

3. Администратор высылает пароль пользователь.

Результат:

- Пользователь зарегистрирован, роль выдана.

Б) Сценарий использования: Написание сообщения.

Цель:

- Написать сообщение любому посетителю сайта.

Действующие лица:

- Администратор, прораб, заказчик, рабочий.

Основной сценарий:

1. Посетитель сайта авторизовывается (страница 2).
2. На главной странице (страница 3.1, 3.2, 3.3.) нажимает на колокольчик (колокольчик может сигнализировать о непрочитанном сообщении) и переходит в список чатов (страница 5.1).
3. В правом верху списка чатов нажимает на кнопку найти и переходит на страницу поиска сообщения (страница 5.2).
4. На странице поиска вводит фамилию, имя, отчество, выбирает должность. Можно ввести что-то одно. Например, фамилию. Далее нажимает кнопку найти. И ниже выбирает из списка пользователей. В низу под каждым пользователем указана его должность (страница 5.3). Далее нажимает на пользователя и переходит в чат (страница 5.4).
5. Или на страницу списка чатов (страница 5.1) выбирает нужного ему пользователя. Тогда он нажимает на этот чат с пользователем и переходит на страницу 5.4.

6. На странице чата отображается вся переписка с пользователем. Поле ввода сообщения и кнопка отправить. Также предусмотрен выход назад на страницу чатов (5.1).

Альтернативные сценарии:

- Первые два пункта как в основном сценарии.

1. Пользователь не нашел нужного адресата. Тогда ищет администратора и пишет ему.
2. Администратор получает сообщение и решает проблему.

Результат:

- Посетитель сайта отправил сообщение, или получил информацию почему оно не может быть отправлено.

В) Сценарий использования: Создание проекта.

Цель:

- Создать проект и распланировать его.

Действующие лица:

- Администратор, прораб, заказчик.

Основной сценарий:

1. Заказчик сайта авторизовывается (страница 2).
2. На главной странице (3.1) нажимает на новый проект.
3. Пишет описание нового проекта (страница 4.1.1)
4. Администратор видит запрос на создание нового проекта и назначает на него прораба.
5. Прораб авторизуется (2), нажимает на главной странице нужный проект (3.2) переходит в новый проект (4.2). Формирует план этапы (6.2), закупки (7.2), риски (8.2), пишет описание и ставит дату начала и конца внизу кнопка редактировать, добавляет на

правой боковой панели контакты (5.2-5.3) и отправляет на согласование заказчику.

6. Заказчик просматривает проект, и ставит статус согласован, либо пишет недочеты в чате и отправляет обратно прорабу.

7. Прораб исправляет и все повторяется.

Альтернативные сценарии:

1. Заказчик и прораб не смогли согласовать проект.

2. Разбирается администратор, возможно, назначает другого прораба.

3. Ошибка при загрузке страницы.

4. Система выводит сообщение об ошибке и предлагает повторить попытку, в случае неудачи система сама уведомит администратора.

Результат:

- Проект сформирован со всеми пожеланиями клиента.

Г) Сценарий использования: Оформление закупки.

Цель:

- Провести закупку необходимых материалов.

Действующие лица:

- Администратор, прораб.

Основной сценарий:

1. Прораб сайта авторизовывается (страница 2).

2. Переходит в нужный проект (страница 3.2).

3. Производит расчеты каких материалов закупить и сколько.

4. Переходит в закупки, расположенные в левой боковой панели (страница 7.2).

5. Открывается страничка. Тут он может просматривать закупки. При нажатии на кнопку редактировать страница (7.2) превращается в страницу (7.2.1).

6. На странице (7.2.1) можно редактировать закупки и добавлять новые.

7. При нажатии на кнопку добавить в таблице появляется строчка, где прораб вносит название товара, его количество и цена за одну единицу.

8. При нажатии на кнопку удалить удаляется вся строчка целиком.

9. При нажатии на кнопку редактировать, появляется возможность редактировать столбцы этой строки прямо в таблице.

Альтернативные сценарии:

1. Ошибка при загрузке страницы.

2. Система выводит сообщение об ошибке и предлагает повторить попытку, в случае неудачи система сама уведомит администратора.

Результат:

- Прораб провел редактирование закупок.

Д) Сценарий использования: Создание рисков.

Цель:

- Провести прорабу аналитику проекта, посчитать все риски и записать их.

Действующие лица:

- Администратор, прораб.

Основной сценарий:

1. Прораб сайта авторизовывается (страница 2).
2. Переходит в нужный проект (страница 3.2).

3. Рассчитывает риски самостоятельно.

4. Переходит в риски, расположенные в левой боковой панели (страница 8.2).

5. Открывается страничка. Тут пользователь может просматривать риски. При нажатии на кнопку редактировать страница (8.2) превращается в страницу (8.2.1).

6. На странице (8.2.1) можно редактировать риски и добавлять новые.

7. При нажатии на кнопку добавить, появляется строка с низу таблицы с двумя столбцами: название риска и описание.

8. При нажатии на кнопку удалить удаляется вся строчка целиком.

9. При нажатии на кнопку редактировать, появляется возможность редактировать столбцы этой строки прямо в таблице.

Альтернативные сценарии:

1. Ошибка при загрузке страницы.

2. Система выводит сообщение об ошибке и предлагает повторить попытку, в случае неудачи система сама уведомит администратора.

Результат:

- Прораб произвел расчет рисков и добавил их в проект.

Е) Сценарий использования: Создание контактов.

Цель:

- Возможность прораба добавить людей, которые относятся к этому проекту.

Действующие лица:

- Администратор, прораб.

Основной сценарий:

1. Прораб сайта авторизовывается (страница 2).
2. Переходит в нужный проект (страница 3.2).
3. Переходит на главную страницу проекта (4.2).
4. Переходит к контактам в правой боковой панели. С низу кнопка добавить. Добавляет себя, заказчика и рабочих. Из этого списка рабочие будут распределяться по задачам. И сообщение об уведомлении графиков, изменение работ, внесение правок будет приходить всем участником в чате, кроме прораба, который сам эти изменения будет делать. А при написании письма прорабу, заказчик будет выбирать прорабов только из своих проектов, что облегчит поиск.
5. В контактах прораб может добавлять или удалять участников проекта.

Альтернативные сценарии:

1. Ошибка при загрузке страницы.
2. Система выводит сообщение об ошибке и предлагает повторить попытку, в случае неудачи система сама уведомит администратора.

Результат:

- Прораб сформировал список участников проекта с возможностью его отредактировать.

Ж) Сценарий использования: Создание и редактирование этапов.

Цель:

- Возможность прораба возможность прораба создавать этапы и их редактировать.

Действующие лица:

- Администратор, прораб.

Основной сценарий:

1. Прораб сайта авторизовывается (страница 2).
2. Переходит в нужный проект (страница 3.2).
3. Переходит на главную страницу проекта (4.2).
4. Переходит в этапы (6.2).
5. На этой странице ему открывается возможность добавить этап, удалить или редактировать.
6. Также есть дата начала и дата конца, которую заполняет прораб.
7. К каждому этапу будут прикрепляться задачи.

Альтернативные сценарии:

1. Ошибка при загрузке страницы.
2. Система выводит сообщение об ошибке и предлагает повторить попытку, в случае неудачи система сама уведомит администратора.

Результат:

- Прораб сформировал этапы проекта с возможностью их отредактировать.

3) Сценарий использования: Создание и редактирование задач.

Цель:

- Возможность прораба возможность прораба создавать задачи и их редактировать.

Действующие лица:

- Администратор, прораб.

Основной сценарий:

1. Прораб сайта авторизовывается (страница 2).
2. Переходит в нужный проект (страница 3.2).

3. Переходит на главную страницу проекта (4.2).

4. Переходит в этапы (6.2).

5. В нужном этапе нажимает кнопку к задачам и переходит на страницу (9.2).

6. На этой странице есть таблица с задачами. С возможностями добавить, удалить или отредактировать задачу.

7. При нажатии на кнопку добавить, появляется строка с низу таблицы со столбцами: название задачи, сроки на её выполнение, место и работники, которых выбирает прораб из списка контактов (страница 9.3).

8. Возможность поставить статус задачи (страница 9.3).

9. При нажатии на кнопку удалить удаляется вся строчка целиком (страница 9.3).

- 10.При нажатии на кнопку редактировать, появляется возможность редактировать столбцы этой задачи прямо в таблице (страница 9.3).

11. Вид задачи в развернутом виде (страница 9.2.1).

12. К каждому этапу будут прикрепляться задачи.

Альтернативные сценарии:

1. Ошибка при загрузке страницы.

2. Система выводит сообщение об ошибке и предлагает повторить попытку, в случае неудачи система сама уведомит администратора.

Результат:

- Прораб сформировал задачи проекта с возможностью их отредактировать.

2.3. Сценарий использования для представления данных

А) Сценарий использования: Просмотр графика работы.

Цель:

- Рабочий просматривает свой график и обсуждает его с прорабом.

Действующие лица:

- Рабочий, прораб.

Основной сценарий:

1. Рабочий авторизовывается (страница 2).
2. На главной странице задачи (страница 3.3).
3. Возможность отфильтровать задачи по проекту, по статусу, по времени (фильтры вверху панели).
4. На задаче, есть информация (тип задачи, где, часы работы и т.п).
5. После выполнения меняет её статус.
6. Если что-то не устраивает, пишет прорабу.

Альтернативные сценарии:

1. Ошибка при загрузке страницы.
2. Система выводит сообщение об ошибке и предлагает повторить попытку, в случае неудачи система сама уведомит администратора.

Результат:

- Рабочий осведомлен о графике работы и высказал все недовольства и пожелания прорабу.

Б) Сценарий использования: Просмотр проекта, внесение правок.

Цель:

- Мониторить информацию о проекте.

Действующие лица:

- Администратор, прораб, заказчик.

Основной сценарий:

1. Заказчик или прораб сайта авторизовывается (страница 2).
2. Выбирает проект (страница 3.2 или 3.1).
3. Заказчик может только смотреть на информацию о проекте (страница 4.1). А прораб, редактировать описание, добавлять заказы, добавлять контакты, менять графики и риски (страница 4.2).

Альтернативные сценарии:

1. Заказчику приходит уведомлении о изменение уже сформированного проекта в чате от прораба автоматически.
2. Ошибка при загрузке страницы.
3. Система выводит сообщение об ошибке и предлагает повторить попытку, в случае неудачи система сама уведомит администратора.

Результат:

- Мониторинг осуществлен.

2.4. Сценарий использования для анализа данных

Сценарий использования: Подсчет статистики в системе

Цель:

Посмотреть статистику данных в системе.

Действующее лицо:

Заказчик, администратор

Основной сценарий:

1. Пользователь попадает на главную страницу.
2. Пользователь нажимает кнопку «статистика».

3. Пользователь выбирает временной промежуток для построения статистики.

4. Пользователь выбирает критерий, по которым построить статистику (например, риски или закупки), а также проекты.

5. Система формирует отчет со статистикой (график, диаграмму и т.д.).

6. Система отображает отчет пользователю на экране.

Альтернативные сценарии:

1. Импорт/Экспорт графиков, для сравнения их с текущими.

2. Отсутствуют данные для построения статистики.

Результат:

Пользователь смотрит статистику, построенную на основе выбранных им данных.

2.5. Сценарий использования для экспорта данных

Сценарий использования: Массовый Импорт/экспорт графика

Цель:

- Помогать анализировать заказчику информацию о проекте.

Действующие лица:

- Заказчик, администратор.

Основной сценарий:

1. Заказчик сайта авторизовывается (страница 2).
2. Переходит на главную страницу (страница 3.1).
3. Переходит в статистику (10).
4. На этой странице есть таблица панель слева, где заказчик может выбрать проекты и вывести к ним график по рискам, по закупкам и т.п. С возможностями добавить или удалить проекты.
5. При нажатии на кнопку применить строится график по вводимым параметрам.

6. Возможность этот график импортировать, а потом спустя время экспортировать и сравнивать их между собой

Альтернативные сценарии:

1. Ошибка при загрузке страницы. 2 Система выводит сообщение об ошибке и предлагает повторить попытку, в случае неудачи система сама уведомит администратора.

2. Не удаётся экспортировать/импортировать файл. Тогда пользователь читает ошибку и пишет администратору

Результат:

- Заказчик получил информацию о своем проекте или проекта и произвел аналитику.

2.6. Вывод

Разработанное решение сбалансировано в плане выполнения операций записи и чтения данных, поскольку пользователи активно взаимодействуют с системой:

Операции чтения:

Пользователи регулярно просматривают данные, такие как проект (Этапы, задачи, риски, закупки, контакты), сообщения, статистика или данные других участников системы, в зависимости от роли пользователя. Это обеспечивает актуальность информации для всех ролей.

Операции записи:

Добавление, редактирование и удаление данных являются неотъемлемой частью взаимодействия с системой. Пользователи имеют возможность изменять, удалять, добавлять данные, такие как проект (Этапы, задачи, риски, закупки, контакты), сообщения, статистика, что повышает гибкость и динамичность работы приложения.

Таким образом, система обеспечивает равномерное распределение нагрузки между операциями чтения и записи, что позволяет сохранять высокую производительность и удобство использования при увеличении числа пользователей.

3. МОДЕЛЬ ДАННЫХ

3.1. Нереляционная модель данных

Графическое представление модели

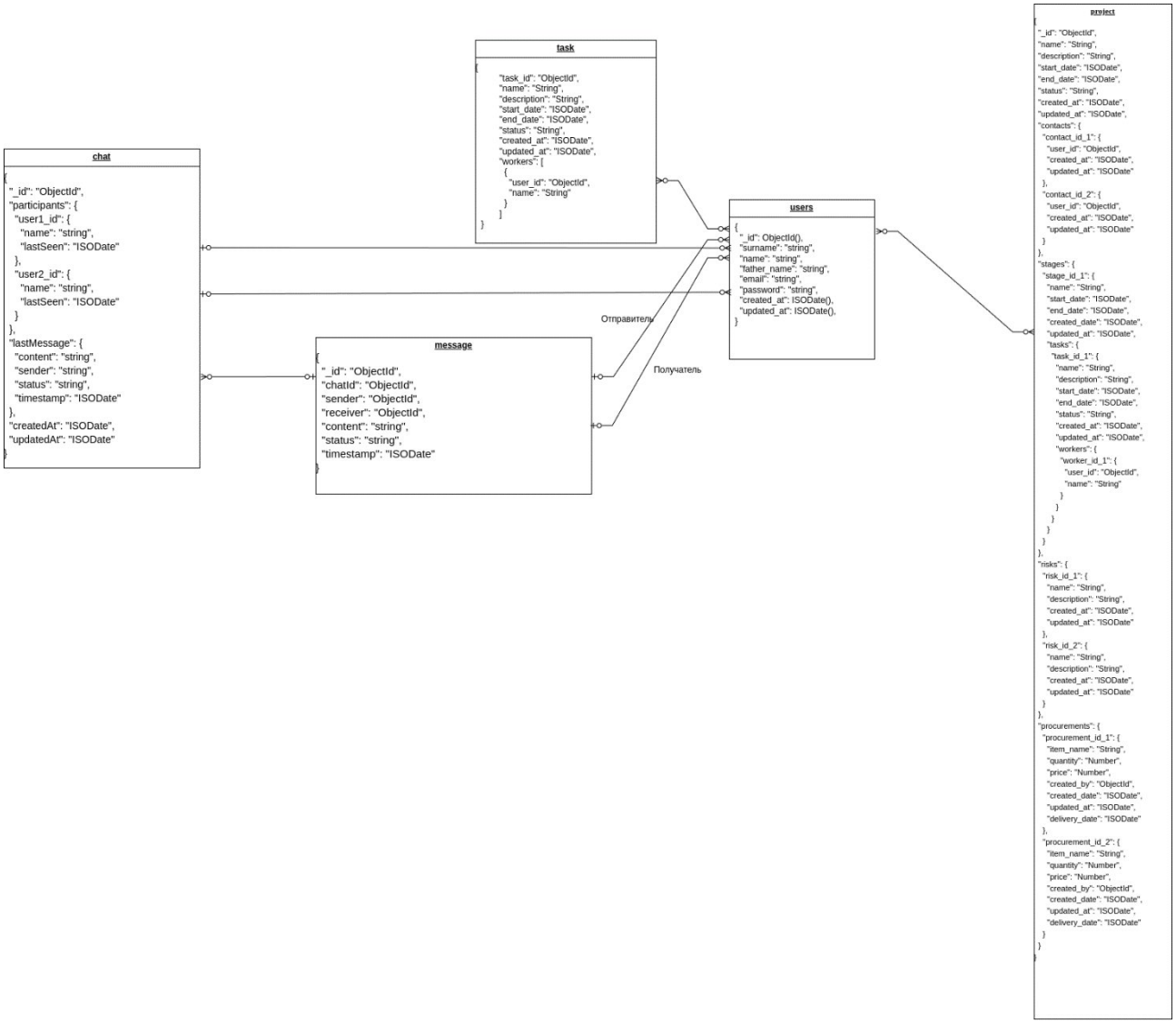

Рисунок 12 - модель данных MongoDB

Описание назначений коллекций, типов данных и сущностей

1. Пользователь (User)

```
{
  "_id": "ObjectID",                                    //
Уникальный идентификатор
  "surname": "String",                          // Фамилия
пользователя
  "name": "String",                             // Имя пользователя
```

```
  "father_name": "String",                // Отчество
пользователя
  "username": "String",                   // Логин
пользователя
  "password": "String",                   // Хеш пароля
  "role": "String",                       // Роль пользователя
(можно также использовать ENUM-подобную логику с набором
ролей)
  "created_at": "ISODate",                // Дата создания
  "updated_at": "ISODate",                // Дата последнего
обновления
}
```

2. Проект (Project)

```
{
  "_id": "ObjectId",                      // Уникальный
идентификатор проекта
  "name": "String",                       // Название проекта
  "description": "String",                // Описание проекта
  "start_date": "ISODate",                // Дата начала
проекта
  "end_date": "ISODate",                  // Дата окончания
проекта
  "status": "String",                     // Статус проекта (в
работе/завершён)
  "created_at": "ISODate",                // Дата создания
  "updated_at": "ISODate",                // Дата последнего
обновления

  "contacts": {                           // Контакты,
связанные с проектом
    "contact_id_1": {
      "user_id": "ObjectId",              // Идентификатор
пользователя
      "created_at": "ISODate",            // Дата создания
контакта
      "updated_at": "ISODate"             // Дата обновления
контакта
    },
    "contact_id_2": {
      "user_id": "ObjectId",
      "created_at": "ISODate",
      "updated_at": "ISODate"
    }
  },

  "stages": {                             // Этапы проекта
    "stage_id_1": {
      "name": "String",                   // Название этапа
      "start_date": "ISODate",            // Дата начала этапа
      "end_date": "ISODate",              // Дата окончания
этапа
```

```
      "created_date": "ISODate",          // Дата создания
этапа
      "updated_at": "ISODate",            // Дата обновления
этапа
      "tasks": {                          // Задачи этапа
        "task_id_1": {
          "name": "String",               // Название задачи
          "description": "String",        // Описание задачи
          "start_date": "ISODate",        // Дата начала
задачи
          "end_date": "ISODate",          // Дата окончания
задачи
          "status": "String",             // Статус задачи
          "created_at": "ISODate",
          "updated_at": "ISODate",
          "workers": {
            "worker_id_1": {              // Рабочий,
назначенный на задачу
              "user_id": "ObjectId",      // Идентификатор
пользователя
              "name": "String"            // Имя рабочего
            }
          }
        }
      }
    }
  },

  "risks": {                              // Риски, связанные
с проектом
    "risk_id_1": {
      "name": "String",                   // Название риска
      "description": "String",            // Описание риска
      "created_at": "ISODate",            // Дата создания
риска
      "updated_at": "ISODate"             // Дата обновления
риска
    },
    "risk_id_2": {
      "name": "String",
      "description": "String",
      "created_at": "ISODate",
      "updated_at": "ISODate"
    }
  },

  "procurements": {                       // Закупки,
связанные с проектом
    "procurement_id_1": {
      "item_name": "String",              // Наименование
товара
      "quantity": "Number",               // Количество товара
```

```
      "price": "Number",                  // Цена товара
      "created_by": "ObjectId",           // Идентификатор
пользователя, создавшего закупку
      "created_date": "ISODate",          // Дата создания
закупки
      "updated_at": "ISODate",            // Дата обновления
закупки
      "delivery_date": "ISODate"          // Дата поставки
товара
    },
    "procurement_id_2": {
      "item_name": "String",
      "quantity": "Number",
      "price": "Number",
      "created_by": "ObjectId",
      "created_date": "ISODate",
      "updated_at": "ISODate",
      "delivery_date": "ISODate"
    }
  }
}
```

3. Сообщение (Message)

```
{
  "_id": "ObjectId",                      // Уникальный
идентификатор сообщения
  "chatId": "ObjectId",                   // Идентификатор
чата, к которому относится сообщение
  "sender": "ObjectId",                    // Идентификатор
отправителя
  "receiver": "ObjectId",                   // Идентификатор
получателя
  "content": "string",                    // Содержание
сообщения
  "status": "string",                     // Статус сообщения
(например, "unread", "read")
  "timestamp": "ISODate"                 // Временная метка
сообщения
}
```

4. Этап (Stage)

```
{
  "_id": "ObjectId",                      // Уникальный
идентификатор этапа
  "name": "String",                       // Название этапа
  "start_date": "ISODate",                // Дата начала этапа
  "end_date": "ISODate",                  // Дата окончания
этапа
  "created_date": "ISODate",              // Дата создания
этапа
  "updated_at": "ISODate",                // Дата последнего
обновления
```

```
  "tasks": {                              // Список задач
этапа
    "task_id_1": {
      "name": "String",                   // Название задачи
      "description": "String",            // Описание задачи
      "start_date": "ISODate",            // Дата начала
задачи
      "end_date": "ISODate",              // Дата окончания
задачи
      "status": "String",                 // Статус задачи
(запланировано/в работе/завершено)
      "created_at": "ISODate",            // Дата создания
задачи
      "updated_at": "ISODate",            // Дата обновления
задачи
      "workers": {                        // Список рабочих,
ответственных за задачу
        "worker_id_1": {
          "user_id": "ObjectId",          // Идентификатор
рабочего
          "name": "String"                // Имя рабочего
        }
      }
    }
  }
}
```

5. Закупка (Procurement)

```
{
  "_id": "ObjectId",                      // Уникальный
идентификатор закупки
  "item_name": "String",                  // Наименование
товара
  "quantity": "Number",                   // Количество товара
  "price": "Number",                      // Цена товара
  "created_by": "ObjectId",               // Идентификатор
пользователя, создавшего закупку
  "created_date": "ISODate",              // Дата создания
закупки
  "updated_at": "ISODate",                // Дата последнего
обновления
  "delivery_date": "ISODate",             // Дата поставки
товара
  "project_id": "ObjectId"                // Идентификатор
связанного проекта
}
```

6. Риск (Risk)

```
{
  "_id": "ObjectId",                      // Уникальный
идентификатор риска
  "name": "String",                       // Название риска
```

```
  "description": "String",                // Описание риска
  "created_at": "ISODate",                   // Дата создания
  "updated_at": "ISODate"                    // Дата
обновления
}
```

7. Контакт (Contact)

```
{
  "_id": "ObjectId",                       // Уникальный
идентификатор контакта
  "user_id": "ObjectId",                   // Идентификатор
пользователя
  "project_id": "ObjectId",                // Идентификатор
проекта
  "created_at": "ISODate",                   // Дата создания
контакта
  "updated_at": "ISODate"                    // Дата
обновления
}
```

8. Задачи (Task)

```
{
  "tasks": {                               // Словарь задач в
рамках этапа
    "task_id_1": {                         // Идентификатор
задачи
      "name": "String",                    // Название задачи
      "description": "String",             // Описание задачи
      "start_date": "ISODate",             // Дата начала задачи
      "end_date": "ISODate",               // Дата окончания
задачи
      "status": "String",                  // Статус задачи
(запланировано/в работе/завершено)
      "created_at": "ISODate",             // Дата создания
задачи
      "updated_at": "ISODate",             // Дата обновления
задачи
      "workers": {                         // Список рабочих,
ответственных за задачу
        "worker_id_1": {                   // Идентификатор
рабочего
          "user_id": "ObjectId",           // Идентификатор
пользователя
          "name": "String"                 // Имя рабочего
        },
        "worker_id_2": {
          "user_id": "ObjectId",
          "name": "String"
        }
      }
    },
    "task_id_2": {
```

```
      "name": "String",
      "description": "String",
      "start_date": "ISODate",
      "end_date": "ISODate",
      "status": "String",
      "created_at": "ISODate",
      "updated_at": "ISODate",
      "workers": {
        "worker_id_1": {
          "user_id": "ObjectId",
          "name": "String"
        }
      }
    }
  }
}
```

9. Чат (Chat)

```
{
  "_id": "ObjectId",                    // Уникальный
идентификатор чата
  "participants": {                     // Словарь
участников чата
    "user1_id": {                       // Идентификатор
пользователя
      "name": "string",                  // Имя пользователя
      "lastSeen": "ISODate"              // Временная метка
последнего просмотра
    },
    "user2_id": {                       // Идентификатор
пользователя
      "name": "string",                  // Имя пользователя
      "lastSeen": "ISODate"              // Временная метка
последнего просмотра
    }
  },
  "lastMessage": {                      // Последнее
сообщение в чате
    "content": "string",                // Содержание
сообщения
    "sender": "string",                 // Идентификатор
отправителя
    "status": "string",                 // Статус сообщения
(например, "unread", "read")
    "timestamp": "ISODate"              // Временная метка
сообщения
  },
  "createdAt": "ISODate",               // Временная метка
создания чата
  "updatedAt": "ISODate"                // Временная метка
последнего обновления чата
}
```

Оценка объема информации, хранимой в модели

Средний размер одного документа коллекции:

Предположительно пользователь будет создавать:

- заказчиков в среднем u
- в среднем прорабов u
- в среднем работников 8*u
- в среднем 1*u проектов
- в среднем 3 этапа на проект
- в среднем 5 задач на этап
- в среднем 105u рассылок + 630u сообщений между пользователями
- в среднем 100 закупок на проект
- в среднем 8 рабочих + 1 прораб+ 1 заказчик на проект
- в среднем 5 рисков на проект

User:

```
{
  "_id": "ObjectId": 12 байт,
  "surname": "String": 500 байт,
  "name": "String": 500 байт,
  "father_name": "String": 500 байт,
  "username": "String": 500 байт,
  "password": "String": 500 байт,
  "role": "String": 50 байт,
  "created_at": "ISODate": 8 байт,
  "updated_at": "ISODate": 8 байт
}
```

Общий размер: приблизительно 2578 байт.

Project:

```
{
  "_id": {
    "type": "ObjectId",
    "size_bytes": 12
  },
  "name": {
    "type": "String",
    "size_bytes": 500
  },
```

```
"description": {
  "type": "String",
  "size_bytes": 1000
},
"start_date": {
  "type": "ISODate",
  "size_bytes": 8
},
"end_date": {
  "type": "ISODate",
  "size_bytes": 8
},
"status": {
  "type": "String",
  "size_bytes": 50
},
"created_at": {
  "type": "ISODate",
  "size_bytes": 8
},
"updated_at": {
  "type": "ISODate",
  "size_bytes": 8
},
"contacts": {
  "contact_id_1": {
    "user_id": {
      "type": "ObjectId",
      "size_bytes": 12
    },
    "created_at": {
      "type": "ISODate",
      "size_bytes": 8
    },
    "updated_at": {
      "type": "ISODate",
      "size_bytes": 8
    }
  },
  "contact_id_2": {
    "user_id": {
      "type": "ObjectId",
      "size_bytes": 12
    },
    "created_at": {
      "type": "ISODate",
      "size_bytes": 8
    },
    "updated_at": {
      "type": "ISODate",
      "size_bytes": 8
    }
```

```
  },
  "total_size_bytes": 56
},
"stages": {
  "stage_id_1": {
    "name": {
      "type": "String",
      "size_bytes": 500
    },
    "start_date": {
      "type": "ISODate",
      "size_bytes": 8
    },
    "end_date": {
      "type": "ISODate",
      "size_bytes": 8
    },
    "created_date": {
      "type": "ISODate",
      "size_bytes": 8
    },
    "updated_at": {
      "type": "ISODate",
      "size_bytes": 8
    },
    "tasks": {
      "task_id_1": {
        "name": {
          "type": "String",
          "size_bytes": 500
        },
        "description": {
          "type": "String",
          "size_bytes": 1000
        },
        "start_date": {
          "type": "ISODate",
          "size_bytes": 8
        },
        "end_date": {
          "type": "ISODate",
          "size_bytes": 8
        },
        "status": {
          "type": "String",
          "size_bytes": 50
        },
        "created_at": {
          "type": "ISODate",
          "size_bytes": 8
        },
        "updated_at": {
```

```
            "type": "ISODate",
            "size_bytes": 8
          },
          "workers": {
            "worker_id_1": {
              "user_id": {
                "type": "ObjectId",
                "size_bytes": 12
              },
              "name": {
                "type": "String",
                "size_bytes": 50
              }
            },
            "total_size_bytes": 62
          },
          "total_size_bytes": 1644
        },
        "total_size_bytes": 1644
      },
      "total_size_bytes": 536
    },
    "total_size_bytes": 536
  },
  "risks": {
    "risk_id_1": {
      "name": {
        "type": "String",
        "size_bytes": 500
      },
      "description": {
        "type": "String",
        "size_bytes": 1000
      },
      "created_at": {
        "type": "ISODate",
        "size_bytes": 8
      },
      "updated_at": {
        "type": "ISODate",
        "size_bytes": 8
      }
    },
    "total_size_bytes": 1516
  },
  "procurements": {
    "procurement_id_1": {
      "item_name": {
        "type": "String",
        "size_bytes": 500
      },
      "quantity": {
```

```
        "type": "Number",
        "size_bytes": 8
      },
      "price": {
        "type": "Number",
        "size_bytes": 8
      },
      "created_by": {
        "type": "ObjectId",
        "size_bytes": 12
      },
      "created_date": {
        "type": "ISODate",
        "size_bytes": 8
      },
      "updated_at": {
        "type": "ISODate",
        "size_bytes": 8
      },
      "delivery_date": {
        "type": "ISODate",
        "size_bytes": 8
      }
    },
    "total_size_bytes": 540
  },
  "total_document_size_bytes":
}
```

Общий размер: приблизительно 12114 байт

Task:

```
{
  "_id": "ObjectId": 12 байт,
  "name": "String": 500 байт,
  "description": "String": 1000 байт,
  "start_date": "ISODate": 8 байт,
  "end_date": "ISODate": 8 байт,
  "status": "String": 50 байт,
  "created_at": "ISODate": 8 байт,
  "updated_at": "ISODate": 8 байт,
  "workers": [{ // Предположим 3 работника
    "user_id": "ObjectId": 12 байт,
    "name": "String": 500 байт
  }] * 3 = (12 * 3 + 500 * 3) = 1536 байт
}
```

Общий размер: приблизительно 3130 байт

Procurement:

```
{
  "_id": "ObjectId": 12 байт,
  "item_name": "String": 500 байт,
  "quantity": "Number": 8 байт,
  "price": "Number": 8 байт,
  "created_date": "ISODate": 8 байт,
  "updated_at": "ISODate": 8 байт,
  "project_id": "ObjectId": 12 байт
}
```

Общий размер: приблизительно 556 байт

Чат:

```
{
  "_id": "12 байт",
  "participants": {
    "user1_id": {
      "name": "50 байт",
      "lastSeen": "8 байт"
    },
    "user2_id": {
      "name": "50 байт",
      "lastSeen": "8 байт"
    }
  },
  "lastMessage": {
    "content": "200 байт",
    "sender": "50 байт",
    "status": "20 байт",
    "timestamp": "8 байт"
  },
  "createdAt": "8 байт",
  "updatedAt": "8 байт",
}
```

Общий размер: приблизительно 422 байт

При количестве заказчиков равным u:

V(u) = (2578×10 + 422×8+ 12114+735×570+15×3130+556×100) × u= 562770×u

Избыточность данных

Предположительно пользователь будет создавать:

- заказчиков в среднем u
- в среднем прорабов u
- в среднем работников 8*u
- в среднем 1*u проектов
- в среднем 3 этапа на проект
- в среднем 5 задач на этап
- в среднем 105×u рассылок + 630×u сообщений между пользователями
- в среднем 100 закупок на проект
- в среднем 8 рабочих + 1 прораб+ 1 заказчик на проект
- в среднем 5 рисков на проект

User:

```
{
  "surname": "String": 500 байт,
  "name": "String": 500 байт,
  "father_name": "String": 500 байт,
  "username": "String": 500 байт,
  "password": "String": 500 байт,
  "role": "String": 50 байт,
}
```

Общий размер: приблизительно 2550 байт.

Project:

```
  "name": {
    "type": "String",
    "size_bytes": 500
  },
  "description": {
    "type": "String",
    "size_bytes": 1000
  },
  "start_date": {
    "type": "ISODate",
    "size_bytes": 8
  },
  "end_date": {
    "type": "ISODate",
```

```
  "size_bytes": 8
},
"status": {
  "type": "String",
  "size_bytes": 50
},
"stages": {
  "stage_id_1": {
    "name": {
      "type": "String",
      "size_bytes": 500
    },
    "start_date": {
      "type": "ISODate",
      "size_bytes": 8
    },
    "end_date": {
      "type": "ISODate",
      "size_bytes": 8
    },
    "tasks": {
      "task_id_1": {
        "name": {
          "type": "String",
          "size_bytes": 500
        },
        "description": {
          "type": "String",
          "size_bytes": 1000
        },
        "start_date": {
          "type": "ISODate",
          "size_bytes": 8
        },
        "end_date": {
          "type": "ISODate",
          "size_bytes": 8
        },
        "status": {
          "type": "String",
          "size_bytes": 50
        },
        "workers": {
          "worker_id_1": {
            "name": {
              "type": "String",
              "size_bytes": 50
            }
          },
          "total_size_bytes": 50
        },
        "total_size_bytes": 1564
```

```
        },
      },
      "total_size_bytes": 516
    },
  },
  "risks": {
    "risk_id_1": {
      "name": {
        "type": "String",
        "size_bytes": 500
      },
      "description": {
        "type": "String",
        "size_bytes": 1000
      },
    },
    "total_size_bytes": 1500
  },
  "procurements": {
    "procurement_id_1": {
      "item_name": {
        "type": "String",
        "size_bytes": 500
      },
      "quantity": {
        "type": "Number",
        "size_bytes": 8
      },
      "price": {
        "type": "Number",
        "size_bytes": 8
      },
      "created_by": {
        "type": "ObjectId",
        "size_bytes": 12
      },
      "created_date": {
        "type": "ISODate",
        "size_bytes": 8
      },
    },
    "total_size_bytes": 536
  },
  "total_document_size_bytes":
```

Общий размер: приблизительно 10344 байт

Message:

```
{
  "text": "String": 500 байт,
```

```
  "sent_date": "ISODate": 8 байт,
  "sender_id": "ObjectId": 12 байт,
  "receiver_id": "ObjectId": 12 байт,
  "status": "String": 10 байт
}
```

Общий размер: приблизительно 542 байт

Task:

```
{
  "name": "String": 500 байт,
  "description": "String": 1000 байт,
  "start_date": "ISODate": 8 байт,
  "end_date": "ISODate": 8 байт,
  "status": "String": 50 байт,
  "workers": [{ // Предположим 3 работника
    "name": "String": 500 байт
  }] * 3 = (500 * 3) = 1500 байт
}
```

Общий размер: приблизительно 3066 байт

Procurement:

```
{
  "item_name": "String": 500 байт,
  "quantity": "Number": 8 байт,
  "price": "Number": 8 байт,
}
```

Общий размер: приблизительно 516 байт

Чат:

```
{
  "participants": {
    "user1_id": {
      "name": "50 байт",
      "lastSeen": "8 байт"
    },
    "user2_id": {
      "name": "50 байт",
      "lastSeen": "8 байт"
    }
  },
  "lastMessage": {
    "content": "200 байт",
    "sender": "50 байт",
    "status": "20 байт",
    "timestamp": "8 байт"
  },
}
```

Общий размер: приблизительно 394 байт

Vclean(u) = (2550×10 + 394×8 + 10344+735×542+15×3066+516×100) × u =534956×u

Тогда рассчитаем избыточность как отношение объема модели к "чистому" объему:

V(u)/Vclean(u)=562770∗u/534956 ∗u≈1,052

Примеры запросов для совершения сценариев использования

Авторизация пользователя

```
MONGO_DETAILS = "mongodb://localhost:27017"
client = AsyncIOMotorClient(MONGO_DETAILS)
database = client['repair-db']

# ...

async def authenticate_user(username: str, password:
str):
    user_collection = database.get_collection("users")
    user = await user_collection.find_one({"username":
username})

    if user and verify_password(password,
user["password"]):
        return user
    else:
    return None
```

Добавление пользователя

```
MONGO_DETAILS = "mongodb://localhost:27017"
client = AsyncIOMotorClient(MONGO_DETAILS)
database = client['repair-db']

# ...

async def create_user(user: User):
    user_collection = database.get_collection("users")
    user.password = hash_password(user.password)
    user.created_at = datetime.utcnow()
    user.updated_at = datetime.utcnow()
    result = await
user_collection.insert_one(user.dict(by_alias=True))
    return result.inserted_id
```

Создание проекта

```
MONGO_DETAILS = "mongodb://localhost:27017"
client = AsyncIOMotorClient(MONGO_DETAILS)
database = client['repair-db']

# ...
class Project(BaseModel):
    _id: ObjectId = Field(default_factory=ObjectId,
alias="_id")
    name: str
    description: str
    start_date: datetime
    end_date: datetime
    status: str
    created_at: datetime
    updated_at: datetime
    contacts: Dict[str, dict] = {}
    stages: Dict[str, dict] = {}
    risks: Dict[str, dict] = {}
    procurements: Dict[str, dict] = {}

# ...

async def create_project(project: Project):
    project_collection =
database.get_collection("projects")
    project.created_at = datetime.utcnow()
    project.updated_at = datetime.utcnow()
    result = await
project_collection.insert_one(project.dict(by_alias=True))
    return result.inserted_id
```

Создание задачи

```
MONGO_DETAILS = "mongodb://localhost:27017"
client = AsyncIOMotorClient(MONGO_DETAILS)
database = client['repair-db']

# ...

class Task(BaseModel):
    _id: ObjectId = Field(default_factory=ObjectId,
alias="_id")
    name: str
    description: str
    start_date: datetime
    end_date: datetime
    status: str
    created_at: datetime
    updated_at: datetime
    workers: Dict[str, Worker] = {}
```

```
    # ...

    async def create_task(task: Task):
        task_collection = database.get_collection("tasks")
        task.created_at = datetime.utcnow()
        task.updated_at = datetime.utcnow()
        result = await
task_collection.insert_one(task.dict(by_alias=True))
        return result.inserted_id
```

Добавление пользователя в задачу

```
    MONGO_DETAILS = "mongodb://localhost:27017"
    client = AsyncIOMotorClient(MONGO_DETAILS)
    database = client['repair-db']

    # ...
    async def add_worker_to_task(task_id: str, worker:
Worker):
        task_collection = database.get_collection("tasks")
        result = await task_collection.update_one(
            {"_id": ObjectId(task_id)},
            {"$set": {f"workers.{str(worker.user_id)}":
worker.dict(by_alias=True)}}
        )
        return result.modified_count
```

Создание чата и сообщения

```
    MONGO_DETAILS = "mongodb://localhost:27017"
    client = AsyncIOMotorClient(MONGO_DETAILS)
    database = client['repair-db']

    # ...

    class LastMessage(BaseModel):
        content: str
        sender: str
        status: str
        timestamp: datetime

    class Participant(BaseModel):
        name: str
        lastSeen: datetime

    class Chat(BaseModel):
        _id: ObjectId = Field(default_factory=ObjectId,
alias="_id")
        participants: Dict[ObjectId, Participant]
        lastMessage: LastMessage
        createdAt: datetime
        updatedAt: datetime
```

```
    # ...

    class Message(BaseModel):
        _id: ObjectId = Field(default_factory=ObjectId,
alias="_id")
        chatId: ObjectId
        sender: str
        receiver: str
        content: str
        status: str
        timestamp: datetime

    # ...

    async def create_chat(chat: Chat):
        chat_collection = database.get_collection("chats")
        chat.createdAt = datetime.utcnow()
        chat.updatedAt = datetime.utcnow()
        result = await
chat_collection.insert_one(chat.dict(by_alias=True))
        return result.inserted_id

    async def create_message(message: Message):
        message_collection =
database.get_collection("messages")
        result = await
message_collection.insert_one(message.dict(by_alias=True))
        return result.inserted_id
```

Обновление чата (появление нового сообщения)

```
    MONGO_DETAILS = "mongodb://localhost:27017"
    client = AsyncIOMotorClient(MONGO_DETAILS)
    database = client['repair-db']

    # ...

    async def update_last_message(chat_id: str, message:
Message):
        chat_collection = database.get_collection("chats")
        last_message = LastMessage(
            content=message.content,
            sender=message.sender,
            status=message.status,
            timestamp=message.timestamp
        )
        result = await chat_collection.update_one(
            {"_id": ObjectId(chat_id)},
            {"$set": {"lastMessage":
last_message.dict(by_alias=True), "updatedAt":
datetime.utcnow()}}
        )
```

```
        return result


    async def create_message_route(message_create: Message):
        message = Message(**message_create.dict())
        message_id = await create_message(message)
        await update_last_message(message.chatId, message)
        return {"_id": message_id, **message.dict()}
```

Поиск чатов

```
    MONGO_DETAILS = "mongodb://localhost:27017"
    client = AsyncIOMotorClient(MONGO_DETAILS)
    database = client['repair-db']

    # ...

    async def get_chats_for_user(user_id: str):
        chat_collection = database.get_collection("chats")
        chats = await
chat_collection.find({"participants.{}".format(user_id):
{"$exists": True}}).to_list(None)
        return chats
```

Получение всех сообщений всех пользователей, на которые не ответили в чате

```
    async def get_unread_messages():
        message_collection =
database.get_collection("messages")
        messages = await message_collection.find({"status":
"unread"}).to_list(None)
        return messages
```

Создание риска

```
    client = MongoClient('mongodb://localhost:27017/')
    db = client['repair-db']

    # ...

    def create_risk(name, description):
        risks_collection = db['risks']
        risk = {
            "_id": ObjectId(),
            "name": name,
            "description": description,
            "created_at": datetime.utcnow(),
            "updated_at": datetime.utcnow()
        }
        risks_collection.insert_one(risk)
```

```
create_risk("Risk 1", "Description of Risk 1")
```

Создание закупки

```
client = MongoClient('mongodb://localhost:27017/')
db = client['repair-db']

# ...

def create_procurement(item_name, quantity, price,
project_id):
    procurements_collection = db['procurements']
    procurement = {
        "_id": ObjectId(),
        "item_name": item_name,
        "quantity": quantity,
        "price": price,
        "created_date": datetime.utcnow(),
        "updated_at": datetime.utcnow(),
        "project_id": project_id
    }
    procurements_collection.insert_one(procurement)

create_procurement("Item 1", 10, 100.0, ObjectId())
```

Получение суммарных сумм закупок по каждой стадии проекта

```
async def get_total_procurement_sums():
    procurement_collection =
database.get_collection("procurements")
    pipeline = [
        {
            "$group": {
                "_id": None,
                "total_sum": {"$sum": {"$multiply":
["$quantity", "$price"]}}
            }
        },
        {
            "$project": {
                "_id": 0,
                "total_sum": 1
            }
        }
    ]
    result = await
procurement_collection.aggregate(pipeline).to_list(None)
    return result
```

Получение статистики

```
client = MongoClient('mongodb://localhost:27017/')
db = client['repair-db']
```

```
    # ...

    def get_statistics(stat_type, project_ids, start_date,
end_date):
        if stat_type not in ['risks', 'procurements']:
            raise ValueError("Неподдерживаемый тип
статистики. Поддерживаемые типы: 'risks', 'procurements'.")

        query = {
            'project_id': {'$in': [ObjectId(pid) for pid in
project_ids]},
            'created_date': {
                '$gte': start_date,
                '$lte': end_date
            }
        }

        if stat_type == 'risks':
            collection = db['risks']
        elif stat_type == 'procurements':
            collection = db['procurements']

        statistics = collection.find(query)
        return list(statistics)

    roject_ids = ['60f1b8d4e138234068a9099a',
'60f1b8d4e138234068a9099b']
    start_date = datetime(2023, 1, 1)
    end_date = datetime(2023, 12, 31)

    # Получение статистики по рискам
    risk_statistics = get_statistics('risks', project_ids,
start_date, end_date)

    # Получение статистики по закупкам
    procurement_statistics = get_statistics('procurements',
project_ids, start_date, end_date)
```

Импорт на примере коллекции projects

```
    import json
    from pymongo import MongoClient
    from datetime import datetime
    from bson import ObjectId


    client = MongoClient('mongodb://localhost:27017/')
    db = client['repair-db']
    projects_collection = db['projects']

    def convert_date_strings(data):
        for item in data:
```

```
            for key in ['start_date', 'end_date',
'created_at', 'updated_at']:
                if key in item:
                    try:
                        item[key] =
datetime.strptime(item[key], '%Y-%m-%dT%H:%M:%SZ')
                    except ValueError:
                        raise ValueError(f"Некорректный
формат даты для поля {key}: {item[key]}")
            if '_id' in item:
                item['_id'] = ObjectId(item['_id'])
        return data

    def validate_data(data):
        required_fields = ['_id', 'name', 'description',
'start_date', 'end_date', 'status', 'created_at',
'updated_at']
        for item in data:
            for field in required_fields:
                if field not in item:
                    raise ValueError(f"Отсутствует
обязательное поле {field} в элементе: {item}")
        return data


    with open('data.json', 'r') as file:
        data = json.load(file)


    data = validate_data(data)
    data = convert_date_strings(data)
    projects_collection.insert_many(data)
```

Экспорт на примере коллекции projects

```
    import json
    from pymongo import MongoClient
    from datetime import datetime
    from bson import ObjectId

    client = MongoClient('mongodb://localhost:27017/')
    db = client['repair-db']
    projects_collection = db['projects']

    def convert_datetime_to_string(data):
        for item in data:
            for key in ['start_date', 'end_date',
'created_at', 'updated_at']:
                if key in item and isinstance(item[key],
datetime):
                    item[key] =
item[key].strftime('%Y-%m-%dT%H:%M:%SZ')
```

```
            if '_id' in item:
                item['_id'] = str(item['_id'])
        return data


    data = list(projects_collection.find())
    data = convert_datetime_to_string(data)


    with open('exported_projects.json', 'w') as file:
        json.dump(data, file, indent=4)
```

3.2. Аналог модели данных для SQL СУБД

Графическое представление модели

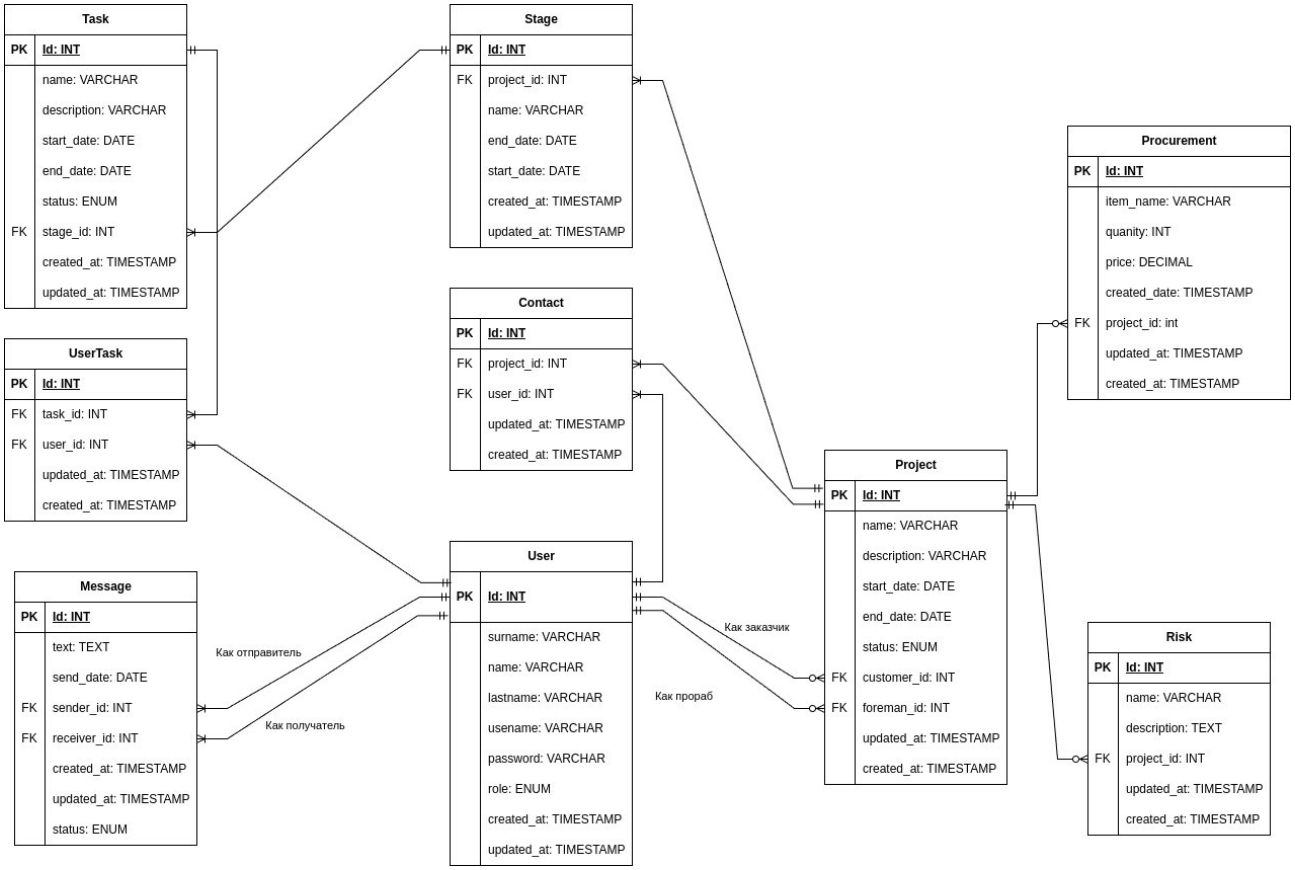

Рисунок 13 - Модель SQL

Описание назначений коллекций, типов данных и сущностей

n − максимально необходимый размер строки для текста. Оптимальное n = 500 символов.

Пользователь (User)

Назначение: Хранение информации о пользователях системы, их идентификацию и роли.

Атрибуты:

- id (PK) — уникальный идентификатор пользователя, тип: INT (4 байта)
- surname — фамилия пользователя, тип: VARCHAR (до n+2 байт (n символов + служебная информация))
- name — имя пользователя, тип: VARCHAR (до n+2 байт (n символов + служебная информация))
- lastname — отчество пользователя, тип: VARCHAR (до n+2 байт (n символов + служебная информация))
- username — уникальный логин для входа, тип: VARCHAR (до n+2 байт (n символов + служебная информация))
- password — зашифрованный пароль пользователя, тип: VARCHAR (до n+2 байт (n символов + служебная информация))
- role — роль пользователя в системе (Администратор, Прораб, Рабочий, Заказчик), тип: ENUM (1 байт (считаем 4 возможных значения))
- created_at — Время создания записи пользователя. Это поле автоматически фиксирует дату и время, когда учетная запись пользователя была создана в системе, тип TIMESTAMP (8 байт)
- updated_at — Время последнего обновления записи пользователя. Это поле автоматически обновляется при каждом изменении данных пользователя, тип TIMESTAMP (8 байт)

Общий размер: приблизительно 5n+31 байт на пользователя

Связи:

- Один пользователь может участвовать в нескольких проектах.
- Один пользователь может быть отправителем или получателем нескольких сообщений.

Проект (Project)

Назначение: Хранение информации о проектах по организации ремонта, включая этапы и задачи.

Атрибуты:

- id (PK) — уникальный идентификатор проекта, тип: INT (4 байта)
- name — название проекта, тип: VARCHAR (до n+2 байт (n символов + служебная информация))
- description — описание проекта, тип: TEXT (до n+2 байт (n символов + служебная информация))
- start_date — дата начала проекта, тип: DATE (3 байта)
- end_date — дата окончания проекта, тип: DATE (3 байта)
- status — статус проекта (в работе, завершён), тип: ENUM (1 байт (считаем 4 возможных значения))
- contact_id (FK) — идентификатор контактов (ссылка на контакты), тип: INT (4 байта)
- created_at — Время создания записи проекта. Это поле автоматически фиксирует дату и время, когда учетная запись проекта была создана в системе, тип TIMESTAMP (8 байт)
- updated_at — Время последнего обновления записи проекта. Это поле автоматически обновляется при каждом изменении данных проекта, тип TIMESTAMP (8 байт)

Общий размер: приблизительно 2n+41 байт на пользователя

Связи:

- Один проект может включать несколько этапов.
- Проект содержит этапы, риски и закупки.

Этап (Stage)

Назначение: Хранение информации об этапах проекта.

Атрибуты:

- id (PK) — уникальный идентификатор этапа, тип: INT (4 байта)
- name — название этапа, тип: VARCHAR (до n+2 байт (n символов + служебная информация))
- start_date — дата начала этапа, тип: DATE (3 байта)
- end_date — дата окончания этапа, тип: DATE (3 байта)
- project_id (FK) — идентификатор проекта, к которому относится этап, тип: INT (4 байта)
- created_at — Время создания записи этапа. Это поле автоматически фиксирует дату и время, когда учетная запись этапа была создана в системе, тип TIMESTAMP (8 байт)
- updated_at — Время последнего обновления записи этапа. Это поле автоматически обновляется при каждом изменении данных этапа, тип TIMESTAMP (8 байт)

Общий размер: приблизительно n+32 байт на этап.

Связи:

- Этап включает несколько задач.
- Этап связан с одним проектом.

Задача (Task)

Назначение: Хранение информации о задачах, связанных с этапами проекта.

Атрибуты:

- id (PK) — уникальный идентификатор задачи, тип: INT (4 байта)
- name — название задачи, тип: VARCHAR (до n+2 байт (n символов + служебная информация))
- description — описание задачи, тип: TEXT (до n+2 байт (n символов + служебная информация))
- start_date — дата начала задачи, тип: DATE (3 байта)
- end_date — дата окончания задачи, тип: DATE (3 байта)

- status — статус задачи (запланировано, в работе, завершено), тип: ENUM (1 байт)
- stage_id (FK) — идентификатор этапа, к которому относится задача, тип: INT (4 байта)
- contact_id (FK) — идентификатор рабочего, ответственного за задачу, тип: INT (4 байта)
- created_at — Время создания записи задачи. Это поле автоматически фиксирует дату и время, когда учетная запись задачи была создана в системе, тип TIMESTAMP (8 байт)
- updated_at — Время последнего обновления записи задачи. Это поле автоматически обновляется при каждом изменении данных задачи, тип TIMESTAMP (8 байт)

Общий размер: приблизительно 2n+39 байт на задачу.

Связи:

- Задача может быть связана с несколькими рабочими через связь многие ко многим.
- Задача может быть связана с этапом через связь один ко многим.

Сообщение (Message)

Назначение: Хранит информацию о сообщениях между пользователями.

Атрибуты:

- id (PK) — уникальный идентификатор сообщения, тип: INT (4 байта)
- text — текст сообщения, тип: TEXT (до n+2 байт (n символов + служебная информация))
- sent_date — дата отправки сообщения, тип: DATETIME (8 байт)
- sender_id (FK) — идентификатор отправителя (ссылка на пользователя), тип: INT (4 байта)
- receiver_id (FK) — идентификатор получателя (ссылка на пользователя), тип: INT (4 байта)

- status — статус сообщения (прочитано или нет), тип: ENUM (1 байт)
- created_at — Время создания записи сообщения. Это поле автоматически фиксирует дату и время, когда учетная запись сообщения была создана в системе, тип TIMESTAMP (8 байт)
- updated_at — Время последнего обновления записи сообщения. Это поле автоматически обновляется при каждом изменении данных сообщения, тип TIMESTAMP (8 байт)

Общий размер: приблизительно n+39 байт на сообщение.

Связи:

- Одно сообщение может быть отправлено одним пользователем и направлено другому пользователю.

Закупка (Procurement)
Назначение: Хранение информации о закупках материалов для проектов.

Атрибуты:

- id (PK) — уникальный идентификатор закупки, тип: INT (4 байта)
- item_name — наименование закупаемого товара, тип: VARCHAR (до n+2 байт (n символов + служебная информация))
- quantity — количество товара, тип: INT (4 байта)
- price — цена товара, тип: DECIMAL (8 байт)
- created_date — дата создания закупки, тип: TIMESTAMP (8 байт)
- project_id (FK) — идентификатор проекта, связанного с закупкой, тип: INT (4 байта)
- created_at — Время создания записи закупки. Это поле автоматически фиксирует дату и время, когда учетная запись закупки была создана в системе, тип TIMESTAMP (8 байт)
- updated_at — Время последнего обновления записи закупки. Это поле автоматически обновляется при каждом изменении данных закупки, тип TIMESTAMP (8 байт)

- created_by (FK) — идентификатор пользователя, который создал закупку, тип: INT (4 байта)
- delivery_date — дата поставки, тип: DATE (3 байта)

Общий размер: приблизительно n+46 байт на сообщение.

Связи:

- Одна закупка привязана к одному проекту.

Риск (Risk)
Назначение: Хранит информацию о рисках, связанных с проектами.

Атрибуты:

- id (PK) — уникальный идентификатор риска, тип: INT (4 байта)
- name — название риска, тип: VARCHAR (до n+2 байт (n символов + служебная информация))
- description — описание риска, тип: TEXT (до n+2 байт (n символов + служебная информация))
- project_id (FK) — идентификатор проекта, к которому относится риск, тип: INT (4 байта)
- created_at — Время создания записи риски. Это поле автоматически фиксирует дату и время, когда учетная запись риска была создана в системе, тип TIMESTAMP (8 байт)
- updated_at — Время последнего обновления записи риска. Это поле автоматически обновляется при каждом изменении данных риска, тип TIMESTAMP (8 байта)

Общий размер: приблизительно 2n+ 28 байт на риск

Связи:

- Один проект может иметь несколько рисков.

Контакт (Contact)

Назначение: Хранение информации о пользователях и проектах, в которых они участвуют.

Атрибуты:

- id (PK) — уникальный идентификатор контакта, тип: INT (4 байта)
- user_id (FK) — идентификатор пользователя, тип: INT (4 байта)
- project_id (FK) — идентификатор проекта, тип: INT (4 байта)
- created_at — Время создания записи контакта. Это поле автоматически фиксирует дату и время, когда учетная запись контакта была создана в системе, тип TIMESTAMP (8 байт)
- updated_at — Время последнего обновления записи контакта. Это поле автоматически обновляется при каждом изменении данных контакта, тип TIMESTAMP (8 байт)

Общий размер: приблизительно 28 байт на контакт

Связи:

- Один контакт связывает пользователя с проектом.

Работник_Задача (User_Task)

Назначение: Хранение информации о пользователях и проектах, в которых они участвуют.

Атрибуты:

- id (PK) — уникальный идентификатор контакта, тип: INT (4 байта)
- user_id (FK) — идентификатор пользователя, тип: INT (4 байта)
- task_id (FK) — идентификатор задачи, тип: INT (4 байта)
- created_at — Время создания записи таблицы. Это поле автоматически фиксирует дату и время, когда учетная запись в таблицу была создана в системе, тип TIMESTAMP (8 байт)

- updated_at — Время последнего обновления записи в таблицу. Это поле автоматически обновляется при каждом изменении данных таблицы, тип TIMESTAMP (8 байт)

Общий размер: приблизительно 28 байт на таблицу Работник_Задача

Связи:

- Много задач связаны с многими работниками.

Оценка объема информации, хранимой в модели

Средний размер одного документа коллекции:

- User: 5n+3 + 24 (оверхед строки) = 2527 байт
- Project: 2n+41 +24 (оверхед строки) = 1065 байт
- Stage: n+32 = 532 байт
- Task: 2n+39 + 24 (оверхед строки) = 1063 байт
- Message: n+39+ 24 (оверхед строки) = 563 байт
- Procurement: n+46 = 546 байт
- Risk: 2n+ 28 + 24 (оверхед строки) = 1052 байт
- Contact: 28 байт

Предположительно пользователь будет создавать:

- заказчиков в среднем u
- в среднем прорабов u
- в среднем работников 8*u
- в среднем 1*u проектов
- в среднем 3 этапа на проект
- в среднем 5 задач на этап
- в среднем 105*u рассылок + 630*u сообщений между пользователями
- в среднем 100 закупок на проект
- в среднем 8 рабочих + 1 прораб+ 1 заказчик на проект
- в среднем 5 рисков на проект

При количестве заказчиков равным u:

V(u)=(252*710 +735*563+1065+3*532+106*315+100*546+5*1052+10*28) ∗u= 516221∗u

Избыточность данных

- В таблице User: избыточные данные id, created_at, updated_at (20 байт)
- В таблице Project: избыточные данные id, created_at, updated_at, project_id (24 байт)
- В таблице Stage: избыточные данные id, created_at, updated_at, project_id (24 байт)
- В таблице Task: избыточные данные id, created_at, updated_at, stage_id, worker_id (28 байт)
- В таблице Message: избыточные данные id, created_at, updated_at, receiver_id, sender_id (28 байт)
- В таблице Procurement: избыточные данные id, created_at, updated_at, project_id (24 байт)
- В таблице Risk: избыточные данные id, created_at, updated_at, project_id (24 байт)
- В таблице Contact: избыточные данные вся таблица.

При количестве заказчиков равным u:

Vclean(u)=(250*710 +1041+3*508+1035*15+735*535+100*522+5*1028) ∗u=493625∗u

Тогда рассчитаем избыточность как отношение объема модели к "чистому" объему:

V(u)/Vclean(u)=516221∗u/493625∗u≈1,0457

Примеры запросов

1. Регистрация пользователя

- Запрос на регистрацию пользователя:

```
INSERT INTO users (username, password_hash, email,
role_id)
VALUES (:username, :password_hash, :email, :role_id);
```

- Запрос на получение роли пользователя:

```
SELECT role_id FROM roles WHERE role_name = :role_name;
```

- Запрос на создание логина и пароля:

```
UPDATE users SET password_hash = :password_hash WHERE
email = :email;
```

2. Написание сообщения

- Запрос на поиск пользователя по фамилии, имени или должности:

```
SELECT * FROM users WHERE last_name = :last_name OR
first_name = :first_name OR position = :position;
```

- Запрос на отправку сообщения:

```
INSERT INTO messages (chat_id, sender_id, content,
timestamp)
VALUES (:chat_id, :sender_id, :content, NOW());
```

- Запрос получение сообщений от пользователя:

```
INSERT INTO messages (chat_id, sender_id, content,
timestamp)
VALUES (:chat_id, :sender_id, :content, NOW());
```

3. Просмотр графика работы

- Запрос на получение задач рабочего:

```
SELECT * FROM tasks WHERE assigned_worker_id = :worker_id
AND (status = 'active' OR status = 'in_progress');
```

- Запрос на фильтрацию задач:

```
SELECT * FROM tasks WHERE project_id = :project_id AND
status = :status AND task_date BETWEEN :start_date
AND :end_date;
```

4. Создание проекта

- Запрос на создание проекта:

```
INSERT INTO projects (name, description, client_id,
start_date, end_date, status)
VALUES
(:name, :description, :client_id, :start_date, :end_date,
'pending');
```

- Запрос на назначение прораба на проект:

```
UPDATE projects SET foreman_id = :foreman_id WHERE
project_id = :project_id;
```

5. Просмотр и правка проекта

- Запрос на просмотр информации о проекте:

```
SELECT * FROM projects WHERE project_id = :project_id;
```

- Запрос на редактирование проекта:

```
UPDATE projects SET description = :description,
start_date = :start_date, end_date = :end_date WHERE
project_id = :project_id;
```

6. Оформление закупки

- Запрос на добавление материалов в закупки:

```
INSERT INTO purchases (project_id, item_name, quantity,
unit_price)
VALUES (:project_id, :item_name, :quantity, :unit_price);
```

- Запрос на редактирование закупки:

```
UPDATE purchases SET item_name = :item_name, quantity
= :quantity, unit_price = :unit_price WHERE purchase_id
= :purchase_id;
```

7. Создание рисков

- Запрос на добавление рисков проекта:

```
INSERT INTO risks (project_id, risk_name, description)
VALUES (:project_id, :risk_name, :description);
```

- Запрос на редактирование риска:

```
UPDATE risks SET risk_name = :risk_name, description
= :description WHERE risk_id = :risk_id;
```

8. Создание контактов

- Запрос на добавление контакта в проект:

```
INSERT INTO project_contacts (project_id, user_id)
VALUES (:project_id, :user_id);
```

- Запрос на удаление контакта:

```
DELETE FROM project_contacts WHERE project_id
= :project_id AND user_id = :user_id;
```

9. Создание и редактирование этапов

- Запрос на добавление этапа проекта:

```
    INSERT INTO stages (project_id, stage_name, start_date,
end_date)
    VALUES
(:project_id, :stage_name, :start_date, :end_date);
```

- Запрос на редактирование этапа:

```
    UPDATE stages SET stage_name = :stage_name, start_date
= :start_date, end_date = :end_date WHERE stage_id
= :stage_id;
```

10. Создание и редактирование задач

- Запрос на создание задачи:

```
    INSERT INTO tasks (stage_id, task_name, start_date,
end_date, worker_id, status)
    VALUES
(:stage_id, :task_name, :start_date, :end_date, :worker_id,
'pending');
```

- Запрос на редактирование задачи:

```
    UPDATE tasks SET task_name = :task_name, start_date
= :start_date, end_date = :end_date, status = :status WHERE
task_id = :task_id;
```

11. Массовый Импорт/Экспорт графика

- Запрос на экспорт данных проекта для анализа:

```
    COPY projects TO '/path/projects.csv' WITH (FORMAT csv,
HEADER);
```

- Запрос на импорт данных в проект:

```
    COPY projects FROM '/path/projects.csv'  WITH (FORMAT
csv, HEADER);
```

12. Получение статистики

- Получение статистики по рискам

```
    SELECT *
    FROM risks
    WHERE project_id IN ('60f1b8d4e138234068a9099a',
'60f1b8d4e138234068a9099b')
      AND created_date BETWEEN '2023-01-01' AND '2023-12-31';
```

- Получение статистики по закупкам

```
    SELECT *
    FROM procurements
    WHERE project_id IN ('60f1b8d4e138234068a9099a',
'60f1b8d4e138234068a9099b')
    AND created_date BETWEEN '2023-01-01' AND '2023-12-31';
```

13. Получение всех сообщений всех пользователей, на которые не ответили в чате

```
SELECT
    m.id,
    m.text,
    m.sent_date,
    m.sender_id,
    m.receiver_id,
    m.status,
    m.created_at,
    m.updated_at
FROM
    messages m
WHERE
    m.status = 'непрочитано';
```

14. Получение суммы закупок по каждой стадии проекта

```
SELECT
    s.id AS stage_id,
    s.name AS stage_name,
    SUM(p.price * p.quantity) AS total_procurement_amount
FROM
    procurements p
JOIN
    projects pr ON pr.id = p.project_id
JOIN
    stages s ON s.project_id = pr.id
GROUP BY
    s.id, s.name
ORDER BY
s.id;
```

3.3. Сравнение моделей

Удельный объем информации

- Реляционная модель данных обычно имеет меньший удельный объем информации по сравнению с нереляционной. Это связано с тем, что в нереляционных системах данные часто дублируются для уменьшения числа обращений к базе данных, что приводит к избыточности. В реляционной модели используются разные таблицы для хранения связанных данных, что требует больше времени на объединение (JOIN) при сложных запросах, но обеспечивает более компактное хранение и

минимизацию дублирования. Выбор между реляционной и нереляционной моделями зависит от требований системы, таких как скорость доступа или целостность данных.

Сравнения для моделей

- для модели User: 2578 байт(NoSQL) | 2527 байт(SQL)
- для модели Project: 12062 байт(NoSQL) | 11253 байт(SQL)
- для модели Task: 3130 байт(NoSQL) | 1063 байт(SQL)
- для модели Message: 570 байт(NoSQL) | 563 байт(SQL)
- для модели Procurement: 556 байт(NoSQL) | 546 байт(SQL)

Запросы по отдельным юзкейсам

- В нереляционной модели количество запросов для выполнения юзкейсов остаётся неизменным, независимо от количества объектов в базе данных. В реляционной модели с увеличением числа объектов количество запросов может возрастать.

Сравнения количества запросов для отдельных юзкейсов

- Регистрация/Авторизация: 1 запрос в (SQL) vs 1 запрос (NoSQL)
- Создание проекта: 1 запрос в (SQL) vs 1 запрос (NoSQL)
- Создание риска: 1 запрос в (SQL) vs 1 запрос (NoSQL)
- Поиск сообщений: 1 запрос в (SQL) vs 1 запрос (NoSQL)
- Получение статистики: 1 запрос в (SQL) vs 1 запрос (NoSQL)

Количество задействованных коллекций

- Регистрация/Авторизация: 1 (SQL) vs 1(NoSQL)
- Создание проекта: 1 (SQL) vs 1 (NoSQL)
- Создание риска: 1 (SQL) vs 1 (NoSQL)
- Поиск сообщений: 1 (SQL) vs 1 (NoSQL)
- Получение статистики: 1 (SQL) vs 1 (NoSQL)

4. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

4.1 Краткое описание

Приложение разделено на две основные части: серверную (back-end) и клиентскую (front-end).

Back-end разработан с использованием фреймворка FastAPI и отвечает за корректное взаимодействие с базой данных (БД), предоставляя данные для отображения на клиентской стороне. Серверная часть обрабатывает API-запросы, извлекает необходимые записи из БД и передает их на фронтенд.

Front-end взаимодействует с API серверной части и отображает полученные данные в удобном для пользователя формате. При создании клиентской части применялась структура, напоминающая FSD-архитектуру, что упрощает процесс разработки и последующего сопровождения проекта.

В проекте выделены несколько ключевых поддиректорий папки src:

- components — содержит компоненты для проекта;
- router – содержит скрипт, определяющий основную навигацию;

Папка assets включает пользовательские текстовые формы, селекторы, чекбоксы, иконки, а также элементы для выбора даты (в поддиректории icons).

4.2 Использованные технологии

БД: MongoDB.

Back-end: FastAPI, Python.

Front-end: JavaScript, Vue3.

4.3 Схема экранов приложения

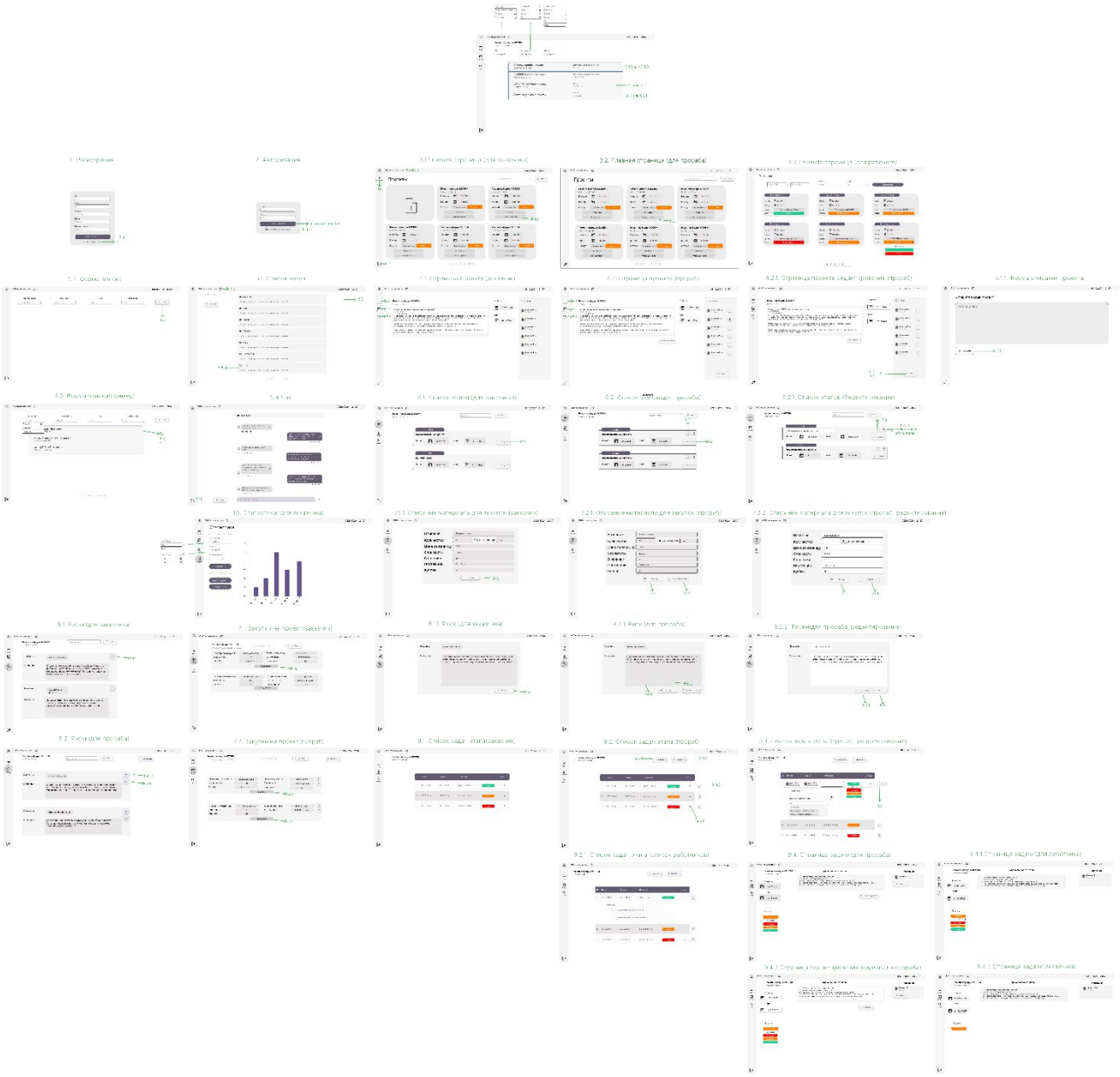

Рисунок 7. Схема экранов приложения

5. ВЫВОДЫ

5.1 Достигнутые результаты

В ходе работы было разработано приложение, представляющее собой сервис по ремонту вузов, которое позволяет пользователям создавать проекты, редактировать основные параметры проекта (этапы, задачи, закупки, риски). Сервис ориентирован на заказчиков и подрядчиков, предоставляя удобный инструмент для взаимодействия.

Реализованы функции для обеспечения коммуникации между рабочими и заказчиками, включая встроенный чат с возможностью поиска пользователей и обмена сообщениями. Приложение также предоставляет статистические данные, функции импорта и экспорта информации, что облегчает перенос данных между проектами, а также создание резервных копий для надежного хранения и восстановления.

5.2 Недостатки и пути для улучшения

Уведомления: В текущей версии отсутствует система уведомлений о внесенных изменениях в проекты, что снижает оперативность информирования пользователей. Реализация этой функции значительно повысит удобство работы.

Роли пользователей: В приложении не предусмотрено деление рабочих на специализации (например, маляр, столяр, укладчик плитки). Добавление иерархии и расширение перечня ролей позволит точнее распределять задачи и учитывать специфику работ.

Многоязычность: Приложение доступно только на русском языке, что ограничивает его использование. Внедрение многоязычной поддержки расширит аудиторию.

Мобильная версия: На данный момент приложение доступно только в формате веб-приложения, что затрудняет работу на мобильных устройствах. Разработка мобильной версии для платформ iOS и Android обеспечит пользователям доступ в любом месте.

Адаптивность: Веб-приложение недостаточно хорошо адаптировано для различных размеров экранов. Улучшение адаптивного дизайна позволит комфортно использовать сервис на смартфонах, планшетах и компьютерах.

5.3 Будущее развитие решения

Планы по развитию включают:

- Добавление системы уведомлений о новых сообщениях и изменениях в проекте;
- Расширение функционала ролей пользователей для учета профессиональных специализаций;
- Реализацию многоязычной поддержки для привлечения более широкой аудитории;
- Разработку мобильной версии для iOS и Android;
- Повышение адаптивности интерфейса для корректного отображения на устройствах с различными размерами экранов.

Эти шаги направлены на повышение удобства и доступности приложения для пользователей, а также расширение его возможностей.

6. ПРИЛОЖЕНИЯ

6.1 Документация по сборке и развертыванию приложения.

1. Склонировать репозиторий с проектом (ссылка указана в списке литературы) и перейти в директорию проекта.
2. Настроить переменные окружения для сервера приложения в файле backend/.env:
   - *MONGO_INITDB_ROOT_USERNAME*: имя пользователя для инициализации базы данных MongoDB.
   - *MONGO_INITDB_ROOT_PASSWORD*: пароль для пользователя, указанного в *MONGO_INITDB_ROOT_USERNAME*.
   - *MONGO_INITDB_DATABASE*: имя базы данных, которая будет создана при инициализации.
   - *DATABASE_URL*: URL подключения к базе данных MongoDB. Формат: mongodb://<username>:<password>@<host>:<port>/?authSource=admin.
   - *ACCESS_TOKEN_EXPIRES_IN*: время жизни токена доступа (в минутах). Например, 60.
   - *JWT_ALGORITHM*: алгоритм, используемый для подписи JWT. Например, HS256.
   - *JWT_SECRET_KEY*: секретный ключ для подписи JWT. Этот параметр должен быть уникальным и безопасным.
   - *DEFAULT_DATA_FILE*: путь к файлу с начальными данными, которые загружаются при старте приложения, если база данных пуста.
   - *JWT_PRIVATE_KEY*: приватный ключ RSA для подписи JWT. Значение должно быть в формате PEM.
   - *JWT_PUBLIC_KEY*: публичный ключ RSA для проверки JWT. Значение должно быть в формате PEM.
3. Собрать контейнеры приложения командой: *docker-compose build --no-cache*.
4. Запустить контейнеры командой: *docker-compose up*.

5. Открыть приложение в браузере по адресу 127.0.0.1:8080 или нажав на порт контейнера frontend в приложении Docker Desktop.

6.2 Инструкция для пользователя.

1. Массовый импорт-экспорт данных.

Доступна админу. Необходимо нажать на кнопку “Пользователи”, в левой боковой панели. Откроется страница, в которой будет доступны 2 кнопки. Экспортировать и импортировать базу данных в формате json. Если попытаться импортировать базу данных в другом формате или вообще не базу данных просто json, на клиенте высветятся соответствующие ошибки.

2. Создание нового проекта и всё что с ним связано (этапы, задачи, закупки, риски, добавление в контакты).

Для того, чтобы создать новый проект, необходимо нажать на кнопку “Создать проект” на главное странице. Эта кнопка доступна только пользователям с ролями администратора и заказчика. После этого осуществится переход на страницу с формой, в которую необходимо внести название проекта, его даты (начало, конец) и описание. Далее создастся проект, или высветятся ошибки при создании. После создания проекта, редактировать, добавлять, удалять всё что в нем находится может прораб и администратор.

3. Написание сообщения.

Пользователь (Все) нажимает на иконку сообщения в левом верхнем углу и переходит в список чатов. Далее если в чате не найти нужного человека, то при нажатии кнопки “Поиск” пользователя переносит на страницу поиска. На этой странице ему доступен поиск по фильтрам, роль, и по отдельности имя, фамилия, отчество. После введенных данных пользователь нажимает на кнопку “Найти” и если существует пользователь в базе данных подходящий по данным из фильтров, то он высвечивается на главном экране. При нажатии на него, пользователь переносится в чат, в пустой, если не было до этого диалога, не в пустой, если был.

4. Страничка рабочего.

В отличии от других пользователей. Рабочему не доступен сам проект. Его главная и единственная страничка, это страничка с его задачами. На ней есть

фильтрация по названию задачи, проекта, по дате и по статусу задачи. Рабочий может менять статус задачи, которая принадлежит ему и писать сообщения как было в пункте выше.

5. Статистика

Статистику могут смотреть только заказчик и админ. Статистика представляется в виде количества рисков или закупок, представленных в графике, в определенное время в определенном проекте. Также график можно экспортировать и импортировать в формате json с проверками на корректность файла.

7. ЛИТЕРАТУРА

1. Ссылка на GitHub. - [Электронный ресурс]. - URL: https://github.com/moevm/nosql2h24-yoga-cat.
2. Vue.js. – [Электронный ресурс]. – URL: https://vuejs.org/ (дата обращения: 26.11.2024).
3. MongoDB. – [Электронный ресурс]. – URL: https://www.mongodb.com/ (дата обращения: 26.11.2024).
4. Chart.js: Bar Chart. – [Электронный ресурс]. – URL: https://www.chartjs.org/docs/latest/charts/bar.html (дата обращения: 26.11.2024).
5. Vue Datepicker. – [Электронный ресурс]. – URL: https://vue3datepicker.com/ (дата обращения: 26.11.2024).
6. VueUse: onClickOutside. – [Электронный ресурс]. – URL: https://vueuse.org/core/onClickOutside/ (дата обращения: 26.11.2024).
7. Pinia. – [Электронный ресурс]. – URL: https://pinia.vuejs.org/ (дата обращения: 26.11.2024).
8. NestJS. – [Электронный ресурс]. – URL: https://docs.nestjs.com/ (дата обращения: 26.11.2024).
9. Lodash. – [Электронный ресурс]. – URL: https://lodash.com/docs/4.17.15 (дата обращения: 26.11.2024).
10. ESLint. – [Электронный ресурс]. – URL: https://eslint.org/ (дата обращения: 26.11.2024).
11. Prettier. – [Электронный ресурс]. – URL: https://prettier.io/ (дата обращения: 26.11.2024).
12. Chaturanga Yoga. – [Электронный ресурс]. – URL: https://chaturanga.yoga/asanas/ (дата обращения: 26.11.2024).