МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики и информационных систем

Факультет автоматики и вычислительной техники

Кафедра систем автоматизации управления

Дата сдачи на проверку:

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2024 г.

Проверено:

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2024 г.

**Разработка информационной системы задачника**

Отчет по практическим работам  
по дисциплине

«Проектирование информационных систем»

Разработали студенты гр. ИТб-3301-01-00 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Олюнин В. К./

(Подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Кузнецов Д. Д./

(Подпись)

Руководитель ст. преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Кузьминых М. М./

(Подпись)

Работа защищена «\_\_\_\_\_\_\_\_\_\_\_» «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2024 г.

Киров 2024

Цель работы: закрепление изученного теоретического материала и практических навыков, полученных в ходе освоения курса «Проектирование информационных систем». Для достижения указанной цели, в ходе выполнения работы необходимо пройти все этапы одного из процессов проектирования и решить задачи, стоящие на каждом из этапов.

1. **Бизнес-моделирование**

Целью разработки задачника, является создание системы управления задачами, которая позволит пользователям организовывать и контролировать рабочие процессы. Данная система включает в себя следующие ключевые элементы:

* доски: основные контейнеры для проектов, где организуются задачи;
* типы задач: категории задач, которые можно настраивать под конкретные нужды проекта;
* задачи: конкретные задания, которые помещаются в соответствующие типы задач.
  1. **Роли и пользователи**

Система задачника предусматривает несколько уровней доступа и ролей пользователей, каждая из которых обладает определенным набором прав и обязанностей. Создатель доски является основной ролью, имеющей полный доступ ко всем функциям системы. Этот пользователь может редактировать типы задач и сами задачи, изменять параметры доски, закрывать доски, перемещать задачи между различными типами задач, назначать задачи другим пользователям, а также управлять ролями участников.

Редактор представляет собой роль, предназначенную для пользователей, которым необходимо вносить изменения в задачи, но не требуется полный доступ ко всем функциям системы. Редактор может редактировать задачи и типы задач, назначать задачи другим пользователям, а также перемещать задачи между типами задач.

Исполнитель – это роль для пользователей, которые выполняют задачи, назначенные им редакторами или создателями досок. Исполнители могут перемещать только те задачи, которые были им назначены, между различными типами задач.

* 1. **Процессы и взаимодействия**

Основные процессы, происходящие в системе задачника, включают:

* создание и настройка доски: создатель доски организует новый проект, создавая доску и определяя основные типы задач;
* создание и редактирование типов задач: пользователи с соответствующими правами могут добавлять новые типы задач и изменять существующие;
* создание и управление задачами: в рамках каждой доски редактор или создатель создают задачи, назначают их исполнителям и следят за их выполнением;
* перенос задач: задачи могут перемещаться между типами в зависимости от их статуса и прогресса;
* назначение задач исполнителям: задачи распределяются между участниками проекта, что позволяет четко определить ответственность за выполнение каждой задачи;
* управление ролями и правами пользователей: создатель доски управляет ролями участников, назначая редакторов и исполнителей.
  1. **Нормативные документы и стандарты**

Проектирование и разработка задачника ведется в соответствии с современными стандартами разработки программного обеспечения, такими как:

* Unified Modeling Language (UML): используется для построения объектно-ориентированных моделей системы;
* Rational Unified Process (RUP): итеративный процесс разработки, который обеспечивает четкую структуризацию проекта и этапов разработки.

В данном разделе были разобраны ключевые элементы системы задачника, такие как доски, типы задач и сами задачи. Также были описаны роли пользователей и основные процессы, обеспечивающие организацию и контроль рабочих процессов. Проектирование системы осуществляется в соответствии с современными стандартами UML и RUP.

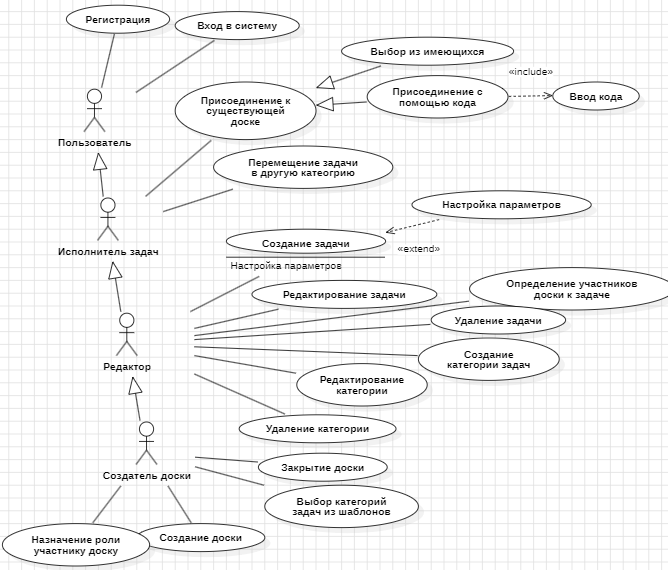
1. **Управление требованиями**

В этом разделе будут выявлены как функциональные, так и нефункциональные требования к проектируемой системе задачника.

* 1. **Функциональные требования**

Поскольку основой является RUP и UML, функциональные требования будут оформляться в виде диаграммы вариантов использования (Use-Case Diagram), а также диаграммы деятельности (Activity Diagram).

На рисунке 1 представлена диаграмма вариантов использования.

  
Рисунок 1 – Диаграмма вариантов использования (Use-Case Diagram)

Данная диаграмма иллюстрирует взаимодействие пользователей с задачником. Диаграмма включает несколько основных ролей пользователей: пользователь, исполнитель задач, редактор и создатель доски. Для каждой роли определены действия (прецеденты), которые они могут выполнять в системе.

Пользователь может зарегистрироваться в системе и выполнить авторизацию для доступа к функционалу задачника.

Исполнитель задач, наследуя функции пользователя, может уже присоединиться к существующей доске, выбрав необходимую доску из списка имеющихся или присоединиться к другой доске с помощью ввода кода. В доске исполнитель может перемещать назначенные ему задачи между различными категориями.

Редактор, помимо всех возможностей исполнителя задач, может создавать и редактировать задачи и категории, а также настраивать параметры задач. Редактор также может назначать участников доски на выполнение конкретных задач.

Создатель доски, обладая всеми функциями редактора, может создавать новые доски для организации проектов, выбирая готовые категории задач из существующих шаблонов. Также создатель управляет ролями участников доски.

Диаграмма вариантов использования показывает основные действия, которые могут выполнять пользователи системы в зависимости от их роли.

Диаграмма деятельности (Activity Diagram) иллюстрирует последовательность действий пользователей в системе задачника. Данная диаграмма представлена на рисунке 2.

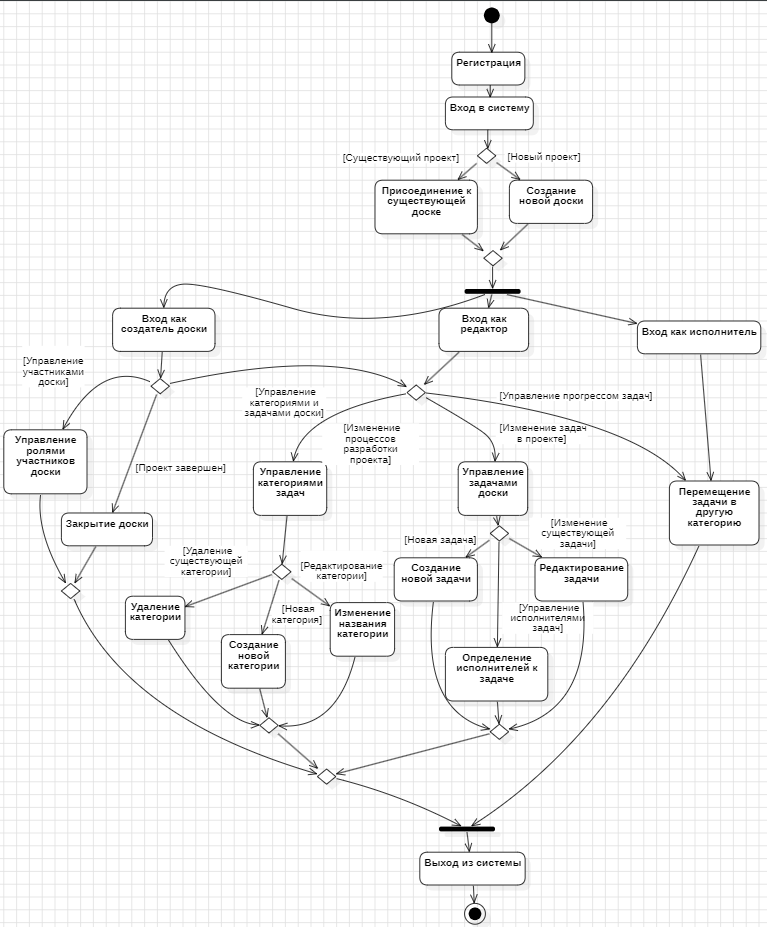
  
Рисунок 2 – Диаграмма деятельности (Activity Diagram)

Диаграмма деятельности описывает поток выполнения различных задач в зависимости от ролей пользователей: пользователь, исполнитель задач, редактор и создатель доски. Диаграмма включает основные действия и решения, с которыми сталкиваются пользователи при работе с задачником.

Процесс начинается с регистрации нового пользователя, после чего пользователь входит в систему. Далее пользователь может выбрать между присоединением к существующей доске или созданием новой доски. Если выбран существующий проект, пользователь присоединяется к существующей доске. Если создается новый проект, происходит создание новой доски.

После этого пользователь может войти в систему в роли исполнителя задач, управляющего задачами или создателя доски. Если пользователь входит как исполнитель задач, он может перемещать задачи в другую категорию в зависимости от их прогресса.

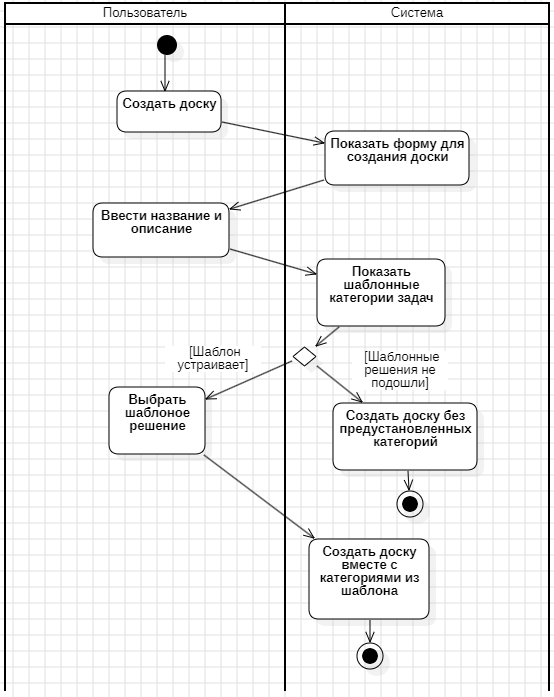
Если пользователь входит как редактор, он может управлять категориями и задачами доски, что включает изменение и создание задач и категорий в проекте, управление исполнителями задач.

Создатель доски, помимо всех функций редактора, может управлять ролями участников доски. Также в случае, если проект завершен, он может закрыть доску.

Основной поток действий завершается выходом пользователя из системы.

Диаграмма активности наглядно показывает основные действия и решения, которые пользователи могут выполнять в системе задачника в зависимости от их роли.

Диаграмма деятельности, представленная на рисунке 3, иллюстрирует процесс создания доски пользователем в системе задачника. Она включает шаги, которые выполняются как пользователем, так и системой.

  
Рисунок 3 – Диаграмма деятельности, рассматривающая процесс создания доски (Activity Diagram)

Процесс начинается с того, что пользователь инициирует создание доски. Система в ответ на это показывает форму для создания доски, где пользователь вводит название и описание будущей доски.

После этого система предлагает пользователю выбрать категории из шаблонных решений. Пользователь может принять одно из предложенных шаблонных решений или отказаться от них.

Если пользователь выбирает шаблонное решение, система создает доску вместе с категориями задач, указанными в выбранном шаблоне. В случае, если шаблонные решения не подошли, система создает доску без предустановленных категорий, предоставляя пользователю возможность добавить категории самостоятельно позже.

Текстовое описание одного из вариантов использования (создание задачи) приведено в Приложении А в формате, предложенном Алистером Коберном. Этот вариант использования описывает процесс, в котором пользователь создает новую задачу в уже существующей доске.

* 1. **Нефункциональные требования**

Исходя из общих соображений о функционировании предметной области, выделяются следующие нефункциональные требования:

1. объемы хранимых и обрабатываемых данных:
   * система должна поддерживать хранение данных о задачах, досках и пользователях, а также историю изменений;
   * объем данных может включать тысячи задач, сотни досок и десятки тысяч пользователей;
2. аппаратные ресурсы:
   * постоянная память: не менее 100 ГБ для хранения данных;
   * оперативная память: не менее 16 ГБ для обеспечения быстрого доступа к данным и выполнения операций;
3. быстродействие:
   * система должна обеспечивать минимальное время отклика при выполнении основных операций;
   * интерфейс пользователя должен загружаться быстро и без задержек, обеспечивая комфортную работу;
4. надежность:
   * система должна быть устойчивой и надежной, обеспечивая высокую доступность и минимальное время простоя;
   * данные должны быть защищены от потери и повреждения.

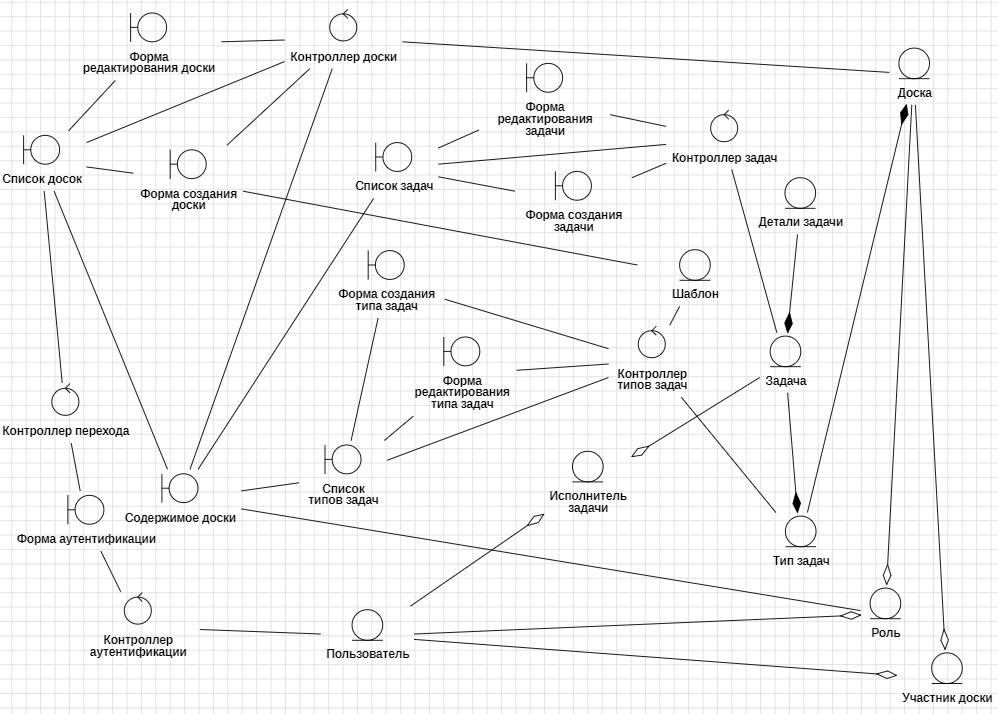
Таким образом, функциональные и нефункциональные требования, выделенные для проектируемой системы задачника, обеспечивают реализацию всех необходимых функций для управления задачами.

1. **Анализ предметной области**

В данном разделе будут проанализированы классы, существующие в предметной области, а также описано их взаимодействие в рамках реализации вариантов использования. Также будет рассмотрена архитектура будущей системы.

* 1. **Классы и их взаимодействие**

На рисунке 4 представлена диаграмма классов этапа анализа

Рисунок 4 – Диаграмма классов этапа анализа (Class Diagram)

На данной диаграмме представлены четыре ключевые сущности: «Пользователь», «Доска», «Тип задач» и «Задача».

Доска содержит определенный набор типов задач, поэтому между сущностями «Тип задач» и «Доска» установлена композиционная связь, поскольку тип задач не может существовать без доски. Аналогичная композиционная связь присутствует между сущностями «Задача» и «Тип задач», так как задача не может существовать вне типа задач. Композиционная связь также имеется между сущностями «Задача» и «Детали задачи».

У каждой задачи могут быть свои исполнители, для чего предусмотрена сущность «Исполнитель задачи», связанная агрегацией с сущностями «Пользователь» и «Задача».

Диаграмма включает контроллер перехода, который направляет на форму аутентификации, связанную с контроллером аутентификации, либо на список досок.

Из граничного класса «Список досок» можно управлять досками с помощью форм редактирования и создания. Также из списка досок можно перейти в конкретную доску через граничный класс «Содержимое доски», который содержит набор типов задач и задач, относящихся к выбранной доске. Управление задачами и типами задач осуществляется через формы создания и редактирования.

Редактирование и создание задач доступны не всем пользователям доски, поэтому добавлена связь между сущностью «Роль» и граничным классом «Содержимое доски».

Для упорядочения выявленных классов был применен анализ по категориям, результат которого представлен в виде таблицы 1 в приложении Б.

Для отображения взаимодействия между выявленными классами была построена диаграмма коммуникации, представленная на рисунке 5.

Рисунок 5 – Диаграмма коммуникации (Communication Diagram).

Диаграмма коммуникации была построена для процесса создания доски. Процесс начинается с инициации пользователем создания доски, он обращается к форме создания доски. Пользователь заполняет название доски, и затем доска создается с помощью контроллера доски.

Далее пользователю предлагается выбрать шаблон, содержащий набор типов задач. Контроллер типов задач передает шаблоны на форму создания доски. Если пользователь выбирает шаблон, он передается в контроллер доски для создания типов задач из этого шаблона для данной доски.

Если шаблоны не подходят пользователю, он может создать свои собственные типы задач.

Таким образом, диаграмма классов этапа анализа и диаграмма коммуникации описывают ключевые сущности системы, их взаимодействие и процесс создания доски, обеспечивая представление логики функционирования задачника.

* 1. **Описание архитектуры системы**

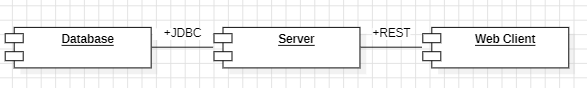
Проектируемая система будет реализована с использованием архитектурного паттерна "клиент-сервер". Она будет состоять из трех основных слоев:

1. клиентское веб-приложение;
2. серверное приложение;
3. база данных.

В качестве СУБД для базы данных выбрана PostgreSQL. Серверное приложение будет построено с помощью Spring Boot на языке программирования Java. Для этого слоя будет применен архитектурный шаблон MVC (Model-View-Controller), который разделяет данные приложения и управляющую логику на три отдельных компонента: модель, представление и контроллер. Также будут использованы такие архитектурные паттерны, как Repository для управления доступом к базе данных и DTO (Data Transfer Object) для передачи данных между слоями приложения. Доступ к данным будет осуществляться через JDBC API.

Клиентское веб-приложение будет построено на React. В этом слое будет использован архитектурный шаблон Router для организации маршрутизации на клиенте с помощью react-router-dom. Для стилизации компонентов будет применен CSS-фреймворк Tailwind CSS. Для создания модальных окон будет использоваться библиотека React Modal, а для реализации функции drag and drop будет применена библиотека React Beautiful DnD. Для работы с HTTP-запросами будет использоваться Axios.

Результат разработки архитектуры системы представлен на рисунке 6 в виде диаграммы компонентов.

  
Рисунок 6 – Диаграмма компонентов (Component diagram)

Представленная диаграмма иллюстрирует архитектуру проектируемой системы, основанной на клиент-серверной модели. Она включает три ключевых компонента: базу данных, сервер и веб-клиент. Взаимодействие между сервером и базой данных осуществляется с использованием JDBC (Java Database Connectivity), который представляет собой API для подключения и выполнения запросов к базам данных на языке Java. Взаимодействие между сервером и веб-клиентом происходит через REST (Representational State Transfer), архитектурный стиль, использующий стандартные HTTP методы для обеспечения взаимодействия между различными компонентами системы.

На рисунке 7 представлена диаграмма развертывания.

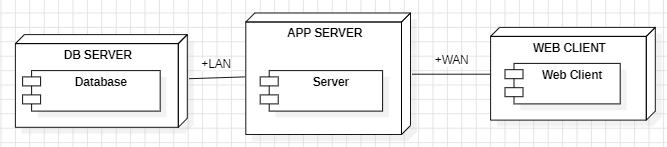
Рисунок 7 – Диаграмма развертывания (Deployment Diagram)

Диаграмма развертывания (Deployment Diagram) иллюстрирует физическое распределение компонентов системы на узлах. На сервере базы данных (DB Server) размещена база данных (Database), которая взаимодействует с сервером приложений (App Server) через локальную сеть (LAN).

Сервер приложений (App Server) содержит серверное приложение (Server), обрабатывающее бизнес-логику и управление данными. Веб-клиент (Web Client) подключается к серверу приложений через глобальную сеть (WAN) и обеспечивает взаимодействие пользователя с системой через веб-интерфейс.

1. **Проектирование системы**

В данном разделе будет представлено проектирование системы, включающее описание классов и их свойств, определение поведения, а также дополнение модели классами, необходимыми для обеспечения работы системы. Кроме того, будут построены диаграммы классов, последовательности и состояний, иллюстрирующие структуру и динамику взаимодействий внутри системы.

* 1. **Описание классов и их свойств**

В ходе проектирования построены диаграммы классов (Class diagram) для серверной части (рисунок 8) и клиентской части (рисунок 9).

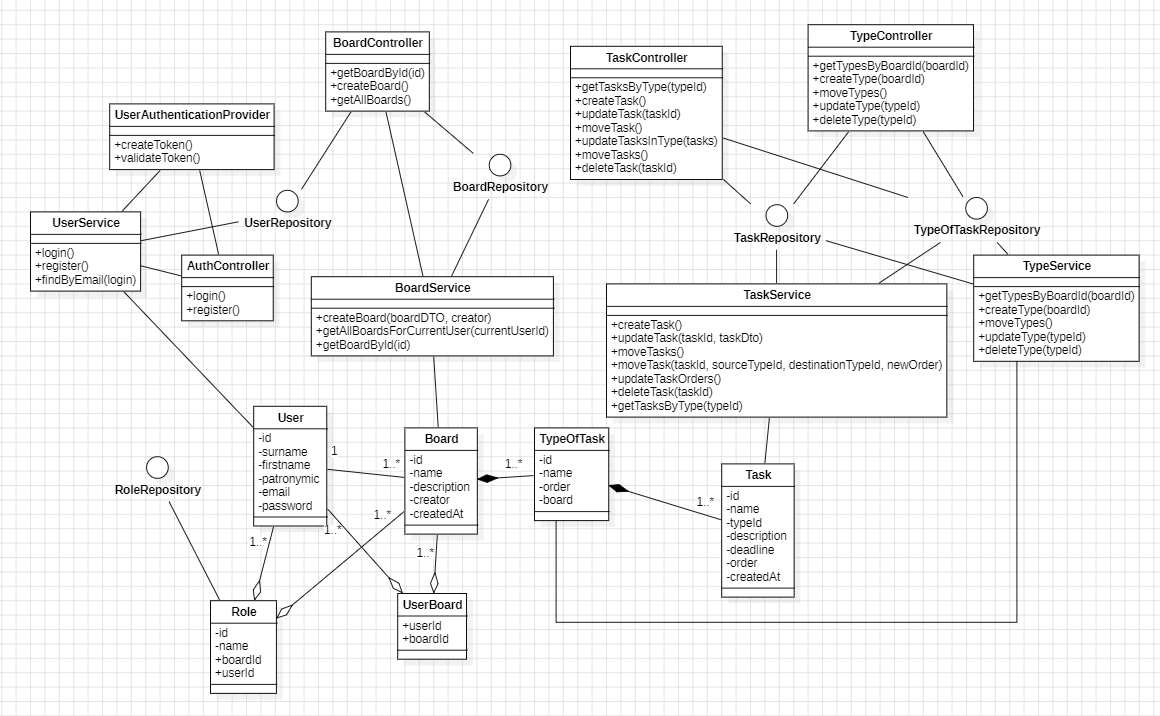


Рисунок 8 – Диаграмма классов серверной части

В таблице 2 приведены описания ключевых классов для серверной части системы, которые обеспечивают ее функционирование.

Таблица 2 – Основные классы серверной части

|  |  |
| --- | --- |
| Название | Описание |
| BoardController | Содержит логику для получения, редактирования и удаления досок. |
| TypeController | Управляет типами задач, включая их создание, получение, изменение и удаление. |
| TaskController | Отвечает за управление задачами, обеспечивая их создание, получение, обновление и удаление. |
| AuthController | Управляет процессами аутентификации и авторизации пользователей. |
| Board, User, TypeOfTask, Task, UserBoard, Role | Сущности, взаимодействующие друг с другом, образуют основу для работы системы управления досками, типами, задачами. |
| UserRepository, BoardRepository, TaskRepository, TypeOfTaskRepository, RoleRepository | Отвечают за доступ к базе данных и выполнение операций CRUD (create, read, update, delete) над данными. |
| BoardService, TaskService, TypeService, UserService | Методы этих классов используются в контроллерах для обработки запросов и выполнения операций с данными. |

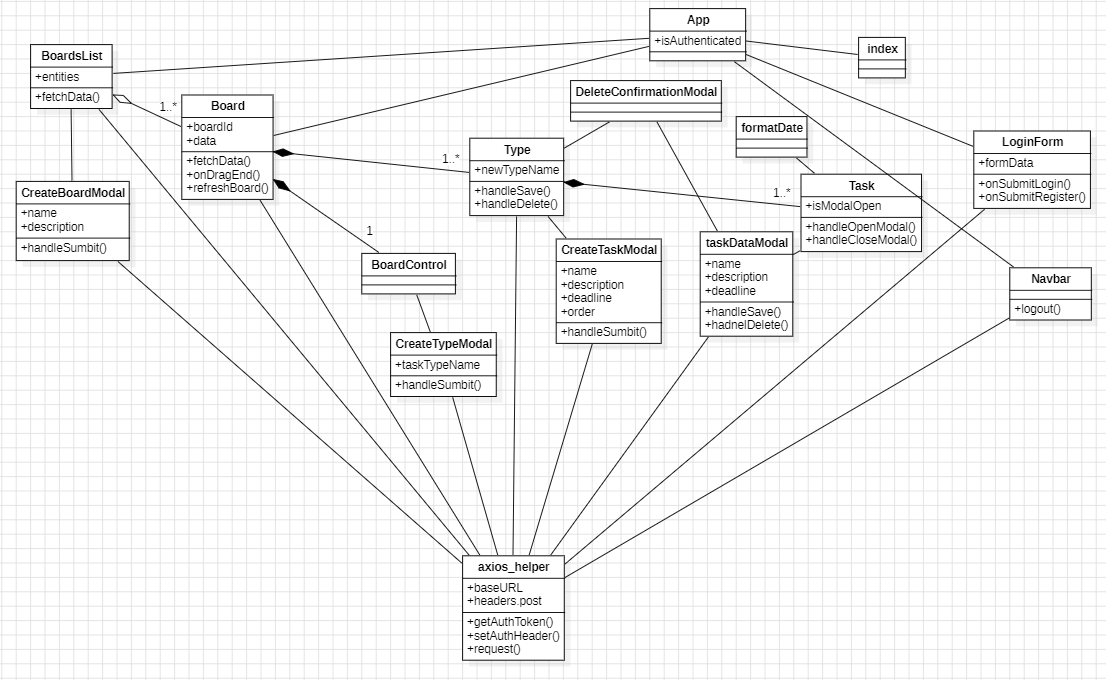
****

Рисунок 9 – Диаграмма классов клиентской части

В таблице 3 приведены описания ключевых классов для клиентской части системы, которые обеспечивают ее взаимодействие с пользователем.

Таблица 3 – Основные классы клиентской части

|  |  |
| --- | --- |
| Название | Описание |
| Board | Отвечает за представление и логику доски на клиентской стороне. Включает в себя методы для получения данных с сервера, обновления доски и обработки операций перетаскивания задач с помощью drag and drop. |
| Task | Отвечает за представление отдельной задачи. |
| Type | Отвечает за представление типа задач. Включает в себя методы сохранения и удаления типа. |

Продолжение таблицы 3

|  |  |
| --- | --- |
| CreateBoardModal, CreateTaskModal, CreateTypeModal | Предназначены для создания доски, задачи и типа в виде модального окна. Предоставляют пользователю форму для ввода данных и обрабатывают отправку данных на сервер. |
| Axios\_helper | Предназначен для настройки и выполнения HTTP-запросов с использованием библиотеки axios, включая установку базового URL, заголовков по умолчанию и управление токеном аутентификации, хранящимся в localStorage. |
| Navbar | Навигационная панель. |
| LoginForm | Представляет собой форму для входа и регистрации пользователей |
| BoardsList | Отображает список доступных досок относительно конкретного пользователя. |

Таким образом были приведены диаграммы классов серверной и клиентской частей системы, а также описания ключевых классов и их свойств, которые обеспечивают функционирование и взаимодействие различных компонентов системы.

* 1. **Диаграмма состояний**

Диаграмма состояний предназначена для отображения жизненного цикла объекта. Представлена на рисунке 10.

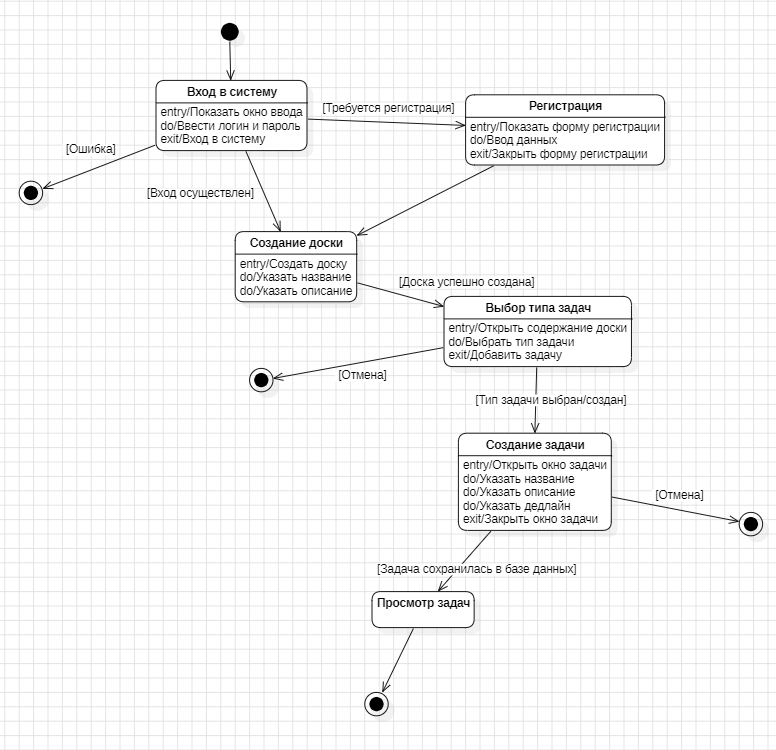


Рисунок 10 – Диаграмма состояний

Данная диаграмма описывает различные состояния:

– «Вход в систему»: Пользователь находится в состоянии входа в систему, где вводит свои данные. Пользователь может отменить вход, либо сделать ошибку в введенных данных;

– «Регистрация»: Если пользователь не имеет учетной записи, он переходит в состояние регистрации, где вводит необходимую информацию для создания новой учетной записи;

– «Создание доски»: После успешного входа в систему или регистрации пользователь переходит в состояние создания новой доски, где указывает название и описание для новой доски;

– «Выбор типа задач»: После успешного создания доски пользователь выбирает тип задачи. Пользователь может отменить это действие;

– «Создание задачи»: После выбора типа задачи пользователь переходит в состояние создания новой задачи на выбранной доске. Он вводит название, описание и дедлайн задачи. Пользователь может отменить это действие;

– «Просмотр задач»: Пользователь просматривает задачи на доске, отображенные в системе.

* 1. **Диаграмма последовательности**

Диаграмма последовательности служит для подробного описания взаимодействия объектов между собой в ходе выполнения функций системы. На рисунке 11 представлена диаграмма последовательности для изменения данных задачи.

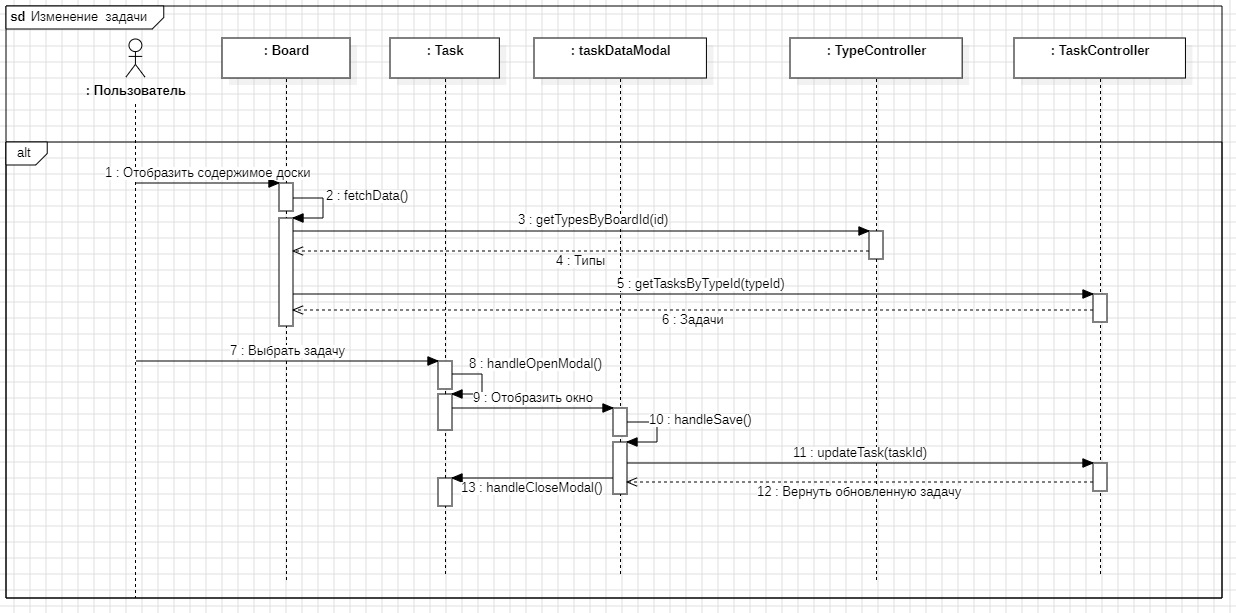


Рисунок 11 – Изменение задачи

Эта диаграмма демонстрирует процесс изменения данных задачи в системе.

Описание шагов:

1) Пользователь инициирует процесс получения данных о доске.

2) Board взаимодействует с BoardService для получения данных и возвращает их пользователю.

3) Board инициирует процесс получения типов задач через TypeController.

4) TypeController взаимодействует с TypeService для получения типов задач и возвращает их в Board.

5) Board инициирует процесс получения задач через TaskController.

6) TaskController взаимодействует с TaskService для получения задач и возвращает их в Board.

7) Пользователь выбирает конкретную задачу для редактирования.

8) Task вызывает метод handleOpenModal для открытия модального окна с данными задачи.

9) В TaskDataModal отображается модальное окно с текущими данными задачи.

10) Пользователь вносит изменения в данные задачи и сохраняет их через метод handleSave.

11) Task отправляет запрос на сохранение обновленных данных задачи в TaskController.

12) TaskController обновляет задачу через TaskService и возвращает обновленную информацию в taskDataModal.

13) Task вызывает метод handleCloseModal для закрытия модального окна после успешного сохранения изменений.

На рисунке 12 представлена диаграмма последовательности для изменения типа задачи.

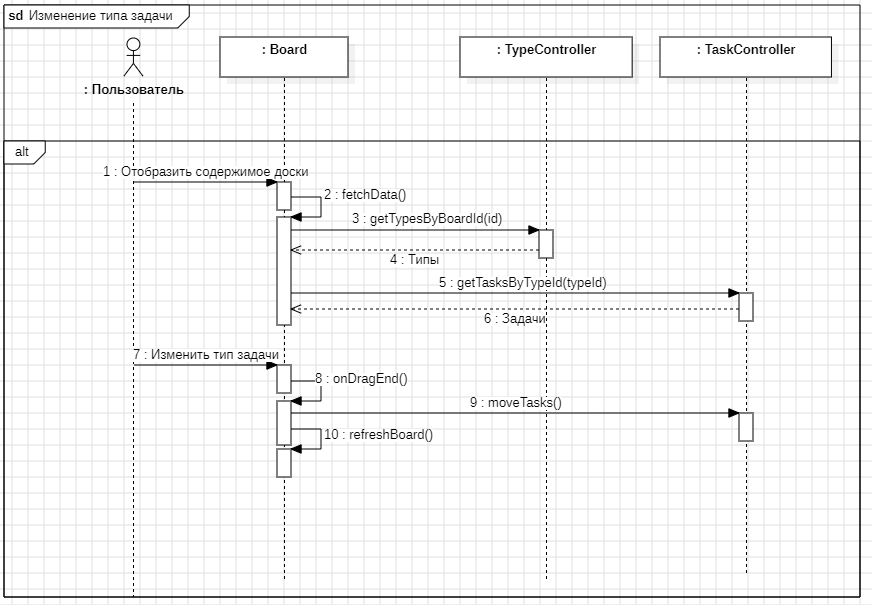


Рисунок 12 – Изменение типа задачи

Данная диаграмма иллюстрирует процесс изменения типа задачи.

Описание шагов:

1) Пользователь инициирует процесс получения данных о доске.

2) Board взаимодействует с BoardService для получения данных и возвращает их пользователю.

3) Board инициирует процесс получения типов задач через TypeController.

4) TypeController взаимодействует с TypeService для получения типов задач и возвращает их в Board.

5) Board инициирует процесс получения задач через TaskController.

6) TaskController взаимодействует с TaskService для получения задач и возвращает их в Board.

7) Пользователь изменяет тип задачи, перетаскивая задачу на другой тип.

8) В Board происходит событие onDragEnd(), которое фиксирует завершение перетаскивания задачи.

9) Board отправляет запрос на обновление типа задачи в TaskController с вызовом метода moveTasks(). TaskController обновляет тип задачи через TaskService.

10) В Board вызывается метод refreshBoard(), который обновляет содержимое доски, получая обновленные данные из контроллеров.

1. **Реализация системы**

В данном разделе рассматривается физическая структура системы, а также приводятся примеры программного кода разработанных модулей.

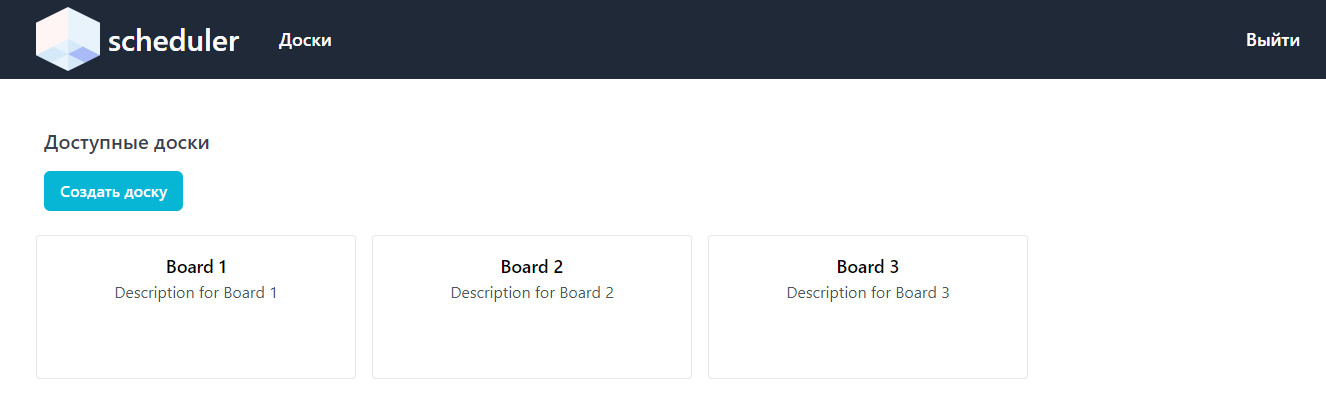
Серверная часть приложения разбита на модули, описание которых приведено в таблице 4.

Таблица 4 – Описание разделов серверной части системы

|  |  |
| --- | --- |
| Раздел | Описание |
| Config | Содержит конфигурационные файлы и классы, необходимые для настройки приложения. |
| Controllers | Включает контроллеры, отвечающие за обработку HTTP-запросов и маршрутизацию |
| Dtos | Содержит объекты передачи данных (DTO), используемые для обмена информацией между клиентом и сервером. |
| Entities | Включает сущности, представляющие объекты базы данных и их отношения. |
| Exceptions | Содержит классы для обработки исключений, возникающих в приложении. |
| Mappers | Включает мапперы для преобразования сущностей в DTO и обратно. |
| Repositories | Содержит репозитории, предоставляющие методы для взаимодействия с базой данных. |
| Services | Включает сервисы, реализующие бизнес-логику приложения. |

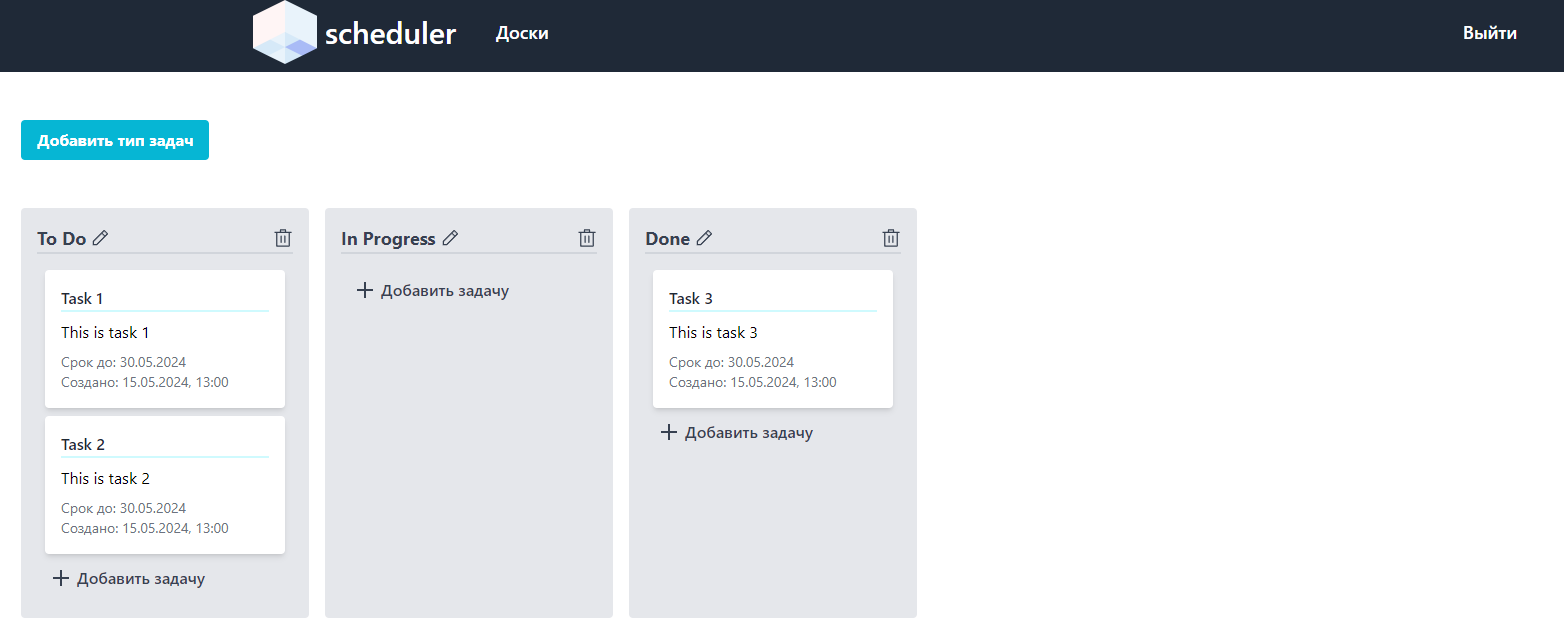
Примеры программного кода разработанных модулей серверной части приведены в приложении В.

Клиентская часть системы состоит из различных компонентов пользовательского интерфейса. Одним из таких компонентов является список задач, доступных пользователю. На рисунке 13 представлена данная страница.

Рисунок 13 – Страница списка досок

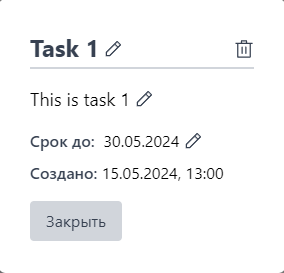
Этот компонент отображает список доступных досок для конкретного пользователя, в зависимости от того, участником каких досок он является.

Рисунок 14 отображает страницу, которая отображает содержимое доски.

Рисунок 14 – Содержимое доски

Эта страница отображает содержимое выбранной доски, состоящей из набора типов задач, которые являются родительскими объектами для задач. В компоненте реализована функция drag and drop для управления задачами. С помощью drag and drop можно менять порядок задач и типов задач, а также изменять тип задач, перетаскивая их из одного типа в другой. Для сохранения изменений отправляются HTTP-запросы на сервер, после чего доска обновляется.

Кроме того, пользователь может выбрать определенную задачу, кликнув на нее. Это откроет модальное окно, которое позволяет управлять задачей и удобно просматривать ее содержимое. Данная страница представлена на рисунке 15.

  
Рисунок 15 – Окно для управление задачей

Пример программного кода компонентов клиентской части размещен в приложении Г.

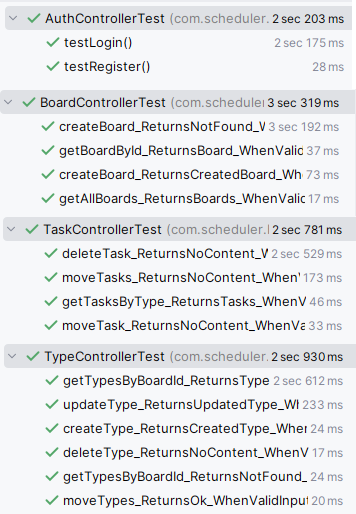
В данном разделе была рассмотрена физическая структура системы, включая описание модулей серверной и клиентской частей.

1. **Тестирования системы**

Для разработанной системы было применено модульное тестирование. В качестве фреймворка для модульного тестирования был выбран MockMvc.

MockMvc – это тестовый фреймворк на стороне сервера, который позволяет проверять большинство функциональных возможностей приложения Spring MVC с помощью облегченных и целевых тестов.

Для всех контроллеров серверной части системы были разработаны модульные тесты, результаты прохождения которых представлены на рисунке 16.

  
Рисунок 16 – Результаты прохождения модульных тестов

Как показывают результаты все тесты были пройдены успешно. Программный код модульных тестов представлен в приложении Д.

**Приложение А**  
(обязательное)  
**Текстовое описания варианта использования**

Создать задачу

**Контекст использования:** пользователь создает новую задачу в уже существующей доске.

**Область действия:** система управления задачами.

**Уровень:** подфункция.

**Основное действующее лицо:** редактор или создатель доски.

**Участники и интересы:**

1. **пользователь**: хочет создать задачу для отслеживания рабочего процесса;
2. с**истема:** должна предоставить интерфейс для создания задачи и сохранить данные в базе данных.

**Предусловие:** пользователь прошел аутентификацию; создана доска; имеется хотя бы один тип задач; пользователь имеет роль редактора или создателя доски.

**Минимальные гарантии:** система должна обеспечить сохранение данных о созданной задаче, даже в случае частичных ошибок.

**Гарантии успеха:** новая задача создана и видна пользователю в соответствующем типе задач.

**Триггер:** пользователь инициирует создание новой задачи.

**Основной сценарий:**

1. **пользователь открывает доску;**
2. **пользователь выбирает тип задач, в который нужно добавить новую задачу;**
3. **пользователь нажимает на кнопку "Создать задачу";**
4. **система отображает форму для создания задачи;**
5. **пользователь заполняет поля: название, описание, дата дедлайна (выбирает исполнителей по желанию);**
6. п**ользователь нажимает кнопку "Сохранить";**
7. **система сохраняет задачу в базе данных;**
8. **система подтверждает создание задачи и отображает её в списке задач выбранного типа.**

**Расширения:**

1. пользователь не заполнил обязательные поля (описание): система отображает сообщение об ошибке и просит заполнить обязательные поля.
2. ошибка при сохранении данных: система показывает сообщение об ошибке и предлагает повторить попытку.

**Список изменений в технологии и данных:**

1. создание нового объекта задачи в базе данных: таблица задач (добавляется новая запись).

**Вспомогательная информация:** необходимо наличие подключенной базы данных для хранения задач.

**Приложение Б**  
(обязательное)  
**Анализ по категориям**

Таблица 1 – упорядочение выявленных классов по категориям

|  |  |  |
| --- | --- | --- |
| № | Категория | Классы |
| 1 | Физические или материальные объекты | Board, Task |
| 2 | Спецификации, описания объектов | BoardDto, CredentialsDto, ErrorDto, MoveTaskRequest, RoleDto, SignUpDto, TaskDto, TaskUserDto, TypeOfTaskDto |
| 3 | Места |  |
| 4 | Транзакции |  |
| 5 | Элементы транзакций |  |
| 6 | Роли людей | Пользователь, роль, исполнитель задачи. |
| 7 | Контейнеры других объектов | Список досок, список типов задачи, список задач |
| 8 | Содержимое контейнеров | Доска, тип задач, задача, детали задачи |
| 9 | Другие компьютеры и прочие системы, внешние по отношению к создаваемой |  |
| 10 | Организации |  |
| 11 | События |  |
| 12 | Процессы |  |
| 13 | Каталоги |  |
| 14 | Записи о деятельности |  |

**Приложение В**  
(обязательное)  
**Программный код разработанных модулей серверной части**

package com.scheduler.backend.config;  
  
import lombok.RequiredArgsConstructor;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.http.HttpMethod;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  
import org.springframework.security.config.http.SessionCreationPolicy;  
import org.springframework.security.web.SecurityFilterChain;  
import org.springframework.security.web.authentication.www.BasicAuthenticationFilter;  
  
  
@RequiredArgsConstructor  
@Configuration  
@EnableWebSecurity  
public class SecurityConfig{  
  
 private final UserAuthenticationEntryPoint userAuthenticationEntryPoint;  
 private final UserAuthenticationProvider userAuthenticationProvider;  
  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 http  
 .exceptionHandling().authenticationEntryPoint(userAuthenticationEntryPoint)  
 .and()  
 .addFilterBefore(new JwtAuthFilter(userAuthenticationProvider), BasicAuthenticationFilter.class)  
 .csrf().disable()  
 .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.*STATELESS*)  
 .and()  
 .authorizeHttpRequests((requests) -> requests  
 .requestMatchers(HttpMethod.*POST*, "/login", "/register").permitAll()  
 .anyRequest().authenticated())  
 ;  
 return http.build();  
 }  
  
}

package com.scheduler.backend.controllers;  
  
import com.scheduler.backend.config.UserAuthenticationProvider;  
import com.scheduler.backend.dtos.CredentialsDto;  
import com.scheduler.backend.dtos.SignUpDto;  
import com.scheduler.backend.dtos.UserDto;  
import com.scheduler.backend.services.UserService;  
import jakarta.validation.Valid;  
import lombok.RequiredArgsConstructor;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RestController;  
  
import java.net.URI;  
  
@RequiredArgsConstructor  
@RestController  
public class AuthController {  
  
 private final UserService userService;  
 private final UserAuthenticationProvider userAuthenticationProvider;  
  
 @PostMapping("/login")  
 public ResponseEntity<UserDto> login(@RequestBody @Valid CredentialsDto credentialsDto) {  
 UserDto userDto = userService.login(credentialsDto);  
 userDto.setToken(userAuthenticationProvider.createToken(userDto.getEmail()));  
 return ResponseEntity.*ok*(userDto);  
 }  
  
 @PostMapping("/register")  
 public ResponseEntity<UserDto> register(@RequestBody @Valid SignUpDto user) {  
 UserDto createdUser = userService.register(user);  
 createdUser.setToken(userAuthenticationProvider.createToken(user.getEmail()));  
 return ResponseEntity.*created*(URI.*create*("/users/" + createdUser.getId())).body(createdUser);  
 }  
  
}

package com.scheduler.backend.dtos;  
  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
@Builder  
public class UserDto {  
 private Long id;  
 private String firstname;  
 private String surname;  
 private String email;  
 private String token;  
}

package com.scheduler.backend.entities;  
  
import jakarta.persistence.\*;  
import jakarta.validation.constraints.NotNull;  
import jakarta.validation.constraints.Size;  
import lombok.\*;  
  
import java.time.LocalDateTime;  
  
@Entity  
@Table(name = "board")  
@AllArgsConstructor  
@NoArgsConstructor  
@Builder  
@Data  
public class Board {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 @Getter  
 @Setter  
 @NotNull  
 @Size(max = 255)  
 @Column(name = "name", nullable = false, length = 255)  
 private String name;  
  
 @Getter  
 @Setter  
 @NotNull  
 @Size(max = 255)  
 @Column(name = "description", nullable = false, length = 255)  
 private String description;  
  
 @ManyToOne  
 @JoinColumn(name = "creator\_id")  
 private User creator;  
  
 @Column(name = "created\_at")  
 private LocalDateTime createdAt;  
  
 public Board(String name, User creator, LocalDateTime createdAt) {  
 this.name = name;  
 this.creator = creator;  
 this.createdAt = createdAt;  
 }  
  
}

package com.scheduler.backend.exceptions;  
  
import org.springframework.http.HttpStatus;  
  
public class AppException extends RuntimeException {  
  
 private final HttpStatus status;  
  
 public AppException(String message, HttpStatus status) {  
 super(message);  
 this.status = status;  
 }  
  
 public HttpStatus getStatus() {  
 return status;  
 }  
}

package com.scheduler.backend.mappers;  
  
import com.scheduler.backend.dtos.SignUpDto;  
import com.scheduler.backend.dtos.UserDto;  
import com.scheduler.backend.entities.User;  
import org.mapstruct.Mapper;  
import org.mapstruct.Mapping;  
  
@Mapper(componentModel = "spring")  
public interface UserMapper {  
 UserDto toUserDto(User user);  
  
 @Mapping(target = "password", ignore = true)  
 User signUpToUser(SignUpDto signUpDto);  
}

package com.scheduler.backend.repositories;  
  
import com.scheduler.backend.entities.Task;  
import com.scheduler.backend.entities.TypeOfTask;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.data.repository.query.Param;  
import org.springframework.stereotype.Repository;  
  
import java.util.List;  
  
@Repository  
public interface TaskRepository extends JpaRepository<Task, String> {  
 @Query("SELECT t FROM Task t WHERE t.typeId.id = :typeId")  
 List<Task> findByTaskTypeId(@Param("typeId") Long typeId);  
  
 @Query("SELECT MAX(t.order) FROM Task t WHERE t.typeId.id = :typeId")  
 Integer findMaxOrderByTypeId(@Param("typeId") Long typeId);  
}

package com.scheduler.backend.services;  
  
import com.scheduler.backend.dtos.CredentialsDto;  
import com.scheduler.backend.dtos.SignUpDto;  
import com.scheduler.backend.dtos.UserDto;  
import com.scheduler.backend.entities.User;  
import com.scheduler.backend.exceptions.AppException;  
import com.scheduler.backend.mappers.UserMapper;  
import com.scheduler.backend.repositories.UserRepository;  
import lombok.RequiredArgsConstructor;  
import org.springframework.http.HttpStatus;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.stereotype.Service;  
  
import java.nio.CharBuffer;  
import java.util.Optional;  
  
@RequiredArgsConstructor  
@Service  
public class UserService {  
 private final UserRepository userRepository;  
 private final PasswordEncoder passwordEncoder;  
 private final UserMapper userMapper;  
  
 public UserDto login(CredentialsDto credentialsDto) {  
 User user = userRepository.findByEmail(credentialsDto.getEmail())  
 .orElseThrow(() -> new AppException("Unknown user", HttpStatus.*NOT\_FOUND*));  
  
 if (passwordEncoder.matches(CharBuffer.*wrap*(credentialsDto.getPassword()), user.getPassword())) {  
 return userMapper.toUserDto(user);  
 }  
 throw new AppException("Invalid password", HttpStatus.*BAD\_REQUEST*);  
 }  
  
 public UserDto register(SignUpDto userDto) {  
 Optional<User> optionalUser = userRepository.findByEmail(userDto.getEmail());  
  
 if (optionalUser.isPresent()) {  
 throw new AppException("Login already exists", HttpStatus.*BAD\_REQUEST*);  
 }  
 User user = userMapper.signUpToUser(userDto);  
 user.setPassword(passwordEncoder.encode(CharBuffer.*wrap*(userDto.getPassword())));  
 User savedUser = userRepository.save(user);  
 return userMapper.toUserDto(savedUser);  
 }  
  
 public UserDto findByEmail(String login) {  
 User user = userRepository.findByEmail(login)  
 .orElseThrow(() -> new AppException("Unknown user", HttpStatus.*NOT\_FOUND*));  
 return userMapper.toUserDto(user);  
 }  
}

**Приложение Г**  
(обязательное)  
**Программный код разработанных модулей клиентской части**

import React, { useState } from 'react';

import { Draggable } from 'react-beautiful-dnd';

import { formatDate, formatDateDead } from '../../helpers/formatDate';

import TaskData from '../TaskDataModal';

const Task = ({ task, index }) => {

    const [isModalOpen, setIsModalOpen] = useState(false);

    const handleOpenModal = () => {

        setIsModalOpen(true);

    };

    const handleCloseModal = () => {

        setIsModalOpen(false);

    };

    return (

        <>

        <Draggable draggableId={task.id.toString()} index={index}>

            {(provided) => (

                <div

                    className="p-4 mb-2 bg-white rounded shadow-md hover:shadow-xl"

                    {...provided.draggableProps}

                    {...provided.dragHandleProps}

                    ref={provided.innerRef}

                    onClick={handleOpenModal}

                >

                    <h3 className="font-semibold text-gray-800 border-b-2 border-cyan-100 mb-2">{task.name}</h3>

                    <p className="mb-2">{task.description}</p>

                    <p className="text-sm text-gray-500">Срок до: {formatDateDead(task.deadline)}</p>

                    <p className="text-sm text-gray-500">Создано: {formatDate(task.createdAt)}</p>

                </div>

            )}

        </Draggable>

        <TaskData isOpen={isModalOpen} onClose={handleCloseModal} task={task} />

        </>

    );

};

export default Task;

import React, { useState } from 'react';

import { Draggable, Droppable } from 'react-beautiful-dnd';

import Task from './Task';

import { AddIcon, CancelIcon, DeleteIcon, EditIcon } from '../../icons/iconsList';

import { request } from '../../helpers/axios\_helper';

import DeleteConfirmationModal from '../DeleteConfirmationModal';

import CreateTaskModal from '../CreateTaskModal';

const Type = ({ type, tasks, index, refreshBoard }) => {

  const [isEditing, setIsEditing] = useState(false);

  const [newTypeName, setNewTypeName] = useState(type.name);

  const [isDeleteModalOpen, setIsDeleteModalOpen] = useState(false);

  const [isTaskAddModalOpen, setIsTaskAddModalOpen] = useState(false);

  const handleEditToggle = () => setIsEditing(!isEditing);

  const handleTypeNameChange = (e) => setNewTypeName(e.target.value);

  const handleSave = async () => {

    try {

      await request('PUT', `/board/type/${type.id}`, { id: type.id, name: newTypeName });

      handleEditToggle();

      refreshBoard();

    } catch (error) {

      console.error('Failed to update type name', error);

    }

  };

  const handleDelete = async () => {

    try {

      await request('DELETE', `/board/type/${type.id}`);

      refreshBoard();

    } catch (error) {

      console.error('Failed to delete type', error);

    } finally {

      setIsDeleteModalOpen(false);

    }

  };

  return (

    <Draggable draggableId={type.id.toString()} index={index}>

      {(provided) => (

        <div

          className="bg-gray-200 rounded p-4 w-72 hover:shadow-xl"

          {...provided.draggableProps}

          ref={provided.innerRef}

        >

          <div className="flex items-center justify-between border-b-2 border-gray-300 mb-2">

            <div className="flex flex-grow items-center">

              {isEditing ? (

                <div className="flex items-center w-full">

                  <input

                    type="text"

                    value={newTypeName}

                    onChange={handleTypeNameChange}

                    className="flex-grow px-2 py-1 border rounded w-32 text-lg font-bold text-gray-700 bg-gray-200 outline-none"

                  />

                  <button

                    onClick={handleSave}

                    className="ml-2 px-2 py-1 bg-cyan-500 hover:bg-cyan-400 text-white rounded"

                  >

                    Сохранить

                  </button>

                  <div onClick={handleEditToggle}>

                    <CancelIcon />

                  </div>

                </div>

              ) : (

                <h3 className="text-lg font-bold text-gray-700" {...provided.dragHandleProps}>

                  {type.name}

                </h3>

              )}

              {!isEditing &&

              <div onClick={handleEditToggle}>

                <EditIcon />

              </div>}

            </div>

            {!isEditing &&

            <div onClick={() => setIsDeleteModalOpen(true)}>

              <DeleteIcon />

            </div>}

          </div>

          <Droppable droppableId={type.id.toString()} type="task">

            {(provided) => (

              <div

                className="rounded p-2 h-full"

                {...provided.droppableProps}

                ref={provided.innerRef}

              >

                {tasks

                  .sort((a, b) => a.order - b.order)

                  .map((task, index) => (

                    <Task key={task.id} task={task} index={index} />

                  ))}

                {provided.placeholder}

                <button

                  className="flex items-center mt-1"

                  onClick={() => setIsTaskAddModalOpen(true)}

                >

                  <AddIcon />

                  <span className="font-medium text-gray-700">Добавить задачу</span>

                </button>

              </div>

            )}

          </Droppable>

          <DeleteConfirmationModal

            isOpen={isDeleteModalOpen}

            onClose={() => setIsDeleteModalOpen(false)}

            onConfirm={handleDelete}

            entityName="Удалить тип задачи?"

          />

          <CreateTaskModal

            isOpen={isTaskAddModalOpen}

            onClose={() => setIsTaskAddModalOpen(false)}

            typeId={type.id}

            refreshBoard={refreshBoard}

          />

        </div>

      )}

    </Draggable>

  );

};

export default Type;

import React, { useState, useEffect } from 'react';

import { useParams } from 'react-router-dom';

import { DragDropContext, Droppable } from 'react-beautiful-dnd';

import Type from './Type';

import { request } from '../../helpers/axios\_helper';

import { initialData } from '../testData';

import BoardControl from '../BoardControl';

import { formatDate, formatDateDead } from '../../helpers/formatDate';

const Board = () => {

  const params = useParams();

  const boardId = params.boardId;

  // const [data, setData] = useState({ types: {}, tasks: {} });

  const [data, setData] = useState(initialData);

  const [loading, setLoading] = useState(true);

  const fetchData = async () => {

    try {

      const typesResponse = await request('GET', `/board/types/${boardId}`);

      const typesData = typesResponse.data;

      const tasksData = {};

      for (const type of typesData) {

        const tasksResponse = await request('GET', `/board/type/tasks/${type.id}`);

        tasksResponse.data.forEach(task => {

          tasksData[task.id] = task;

        });

      }

      const formattedTypes = typesData.reduce((acc, type) => {

        acc[type.id] = type;

        return acc;

      }, {});

      setData({ types: formattedTypes, tasks: tasksData });

    } catch (error) {

      console.error("Failed to fetch data", error);

    } finally {

      setLoading(false);

    }

  };

  useEffect(() => {

    fetchData();

  }, [boardId]);

  const refreshBoard = () => {

    fetchData();

  };

  const onDragEnd = async (result) => {

    const { destination, source, draggableId, type } = result;

  if (!destination) return;

  if (type === 'type') {

    const newTypeOrder = Array.from(Object.values(data.types)).sort((a, b) => a.order - b.order);

    const [moved] = newTypeOrder.splice(source.index, 1);

    newTypeOrder.splice(destination.index, 0, moved);

    const updatedTypes = newTypeOrder.map((type, index) => ({

      ...type,

      order: index

    }));

    const newState = {

      ...data,

      types: updatedTypes.reduce((acc, type) => {

        acc[type.id] = type;

        return acc;

      }, {}),

    };

    setData(newState);

    try {

      console.log("Sending types:", JSON.stringify(updatedTypes));

      await request('PUT', `/board/types/move`, updatedTypes);

      refreshBoard();

    } catch (error) {

      console.error("Failed to update type order", error);

    }

    return;

  }

    const startType = data.types[source.droppableId];

    const finishType = data.types[destination.droppableId];

    if (startType === finishType) {

      const newTaskOrder = Array.from(Object.values(data.tasks)

        .filter(task => task.typeId === startType.id)

        .sort((a, b) => a.order - b.order));

      const [movedTask] = newTaskOrder.splice(source.index, 1);

      newTaskOrder.splice(destination.index, 0, movedTask);

      const updatedTasks = newTaskOrder.map((task, index) => ({

        ...task,

        order: index

      }));

      const newState = {

        ...data,

        tasks: updatedTasks.reduce((acc, task) => {

          acc[task.id] = task;

          return acc;

        }, {}),

      };

      setData(newState);

      try {

        await request('PUT', `/board/tasks/move`, {

          moveTasks: updatedTasks.map(task => ({

            taskId: task.id,

            sourceTypeId: startType.id,

            destinationTypeId: finishType.id,

            newOrder: task.order,

          })),

        });

        refreshBoard();

      } catch (error) {

        console.error("Failed to update task order within type", error);

      }

    } else {

      const startTaskOrder = Array.from(Object.values(data.tasks)

        .filter(task => task.typeId === startType.id)

        .sort((a, b) => a.order - b.order));

      const finishTaskOrder = Array.from(Object.values(data.tasks)

        .filter(task => task.typeId === finishType.id)

        .sort((a, b) => a.order - b.order));

      const [movedTask] = startTaskOrder.splice(source.index, 1);

      movedTask.typeId = finishType.id;

      finishTaskOrder.splice(destination.index, 0, movedTask);

      const updatedStartTasks = startTaskOrder.map((task, index) => ({

        ...task,

        order: index

      }));

      const updatedFinishTasks = finishTaskOrder.map((task, index) => ({

        ...task,

        order: index

      }));

      const newState = {

        ...data,

        tasks: {

          ...data.tasks,

          ...updatedStartTasks.reduce((acc, task) => {

            acc[task.id] = task;

            return acc;

          }, {}),

          ...updatedFinishTasks.reduce((acc, task) => {

            acc[task.id] = task;

            return acc;

          }, {}),

          [movedTask.id]: movedTask,

        },

      };

      setData(newState);

      const moveTasks = [

        ...updatedStartTasks.map(task => ({

          taskId: task.id,

          sourceTypeId: startType.id,

          destinationTypeId: startType.id,

          newOrder: task.order,

        })),

        ...updatedFinishTasks.map(task => ({

          taskId: task.id,

          sourceTypeId: startType.id,

          destinationTypeId: finishType.id,

          newOrder: task.order,

        })),

      ];

      try {

        console.log("Sending moveTasks:", JSON.stringify(moveTasks, null, 2));

        await request('PUT', `/board/tasks/move`, { moveTasks: moveTasks });

        refreshBoard();

      } catch (error) {

        console.error("Failed to update task order between types", error);

      }

    }

  };

  if (loading) {

    return <div>Loading...</div>;

  }

  return (

    <div className="mx-auto w-11/12">

      <BoardControl boardId={boardId} refreshBoard={refreshBoard} />

      <DragDropContext onDragEnd={onDragEnd}>

        <Droppable droppableId="all-types" direction="horizontal" type="type">

          {(provided) => (

            <div

              className="flex space-x-4 p-4"

              {...provided.droppableProps}

              ref={provided.innerRef}

            >

              {Object.values(data.types)

                .sort((a, b) => a.order - b.order)

                .map((type, index) => {

                  const tasks = Object.values(data.tasks)

                    .filter(task => task.typeId === type.id) // Ensure correct mapping

                    .sort((a, b) => a.order - b.order);

                  return <Type key={type.id} type={type} tasks={tasks} index={index} refreshBoard={refreshBoard} />;

                })}

              {provided.placeholder}

            </div>

          )}

        </Droppable>

      </DragDropContext>

    </div>

  );

};

export default Board;

import React, { useState, useEffect } from 'react';

import { Link } from 'react-router-dom';

import { request } from '../helpers/axios\_helper';

import CreateBoardModal from './CreateBoardModal';

import { testEntities } from './testData';

const EntityCard = ({ entity }) => {

  const boardId = entity.id;

  return (

    <Link to={`/boards/${boardId}`} className="border rounded p-4 m-2 hover:shadow-md transition duration-300 ease-in-out h-36 w-80 flex flex-col justify-start items-center">

      <h2 className="text-lg font-semibold">{entity.name}</h2>

      <p className="text-gray-600">{entity.description}</p>

    </Link>

  );

};

const MemoizedEntityCard = React.memo(EntityCard); // Оптимизация рендеринга

const BoardsList = () => {

  const [entities, setEntities] = useState(testEntities);

  // const [entities, setEntities] = useState([]);

  const [isModalOpen, setIsModalOpen] = useState(false);

  const fetchData = async () => {

    try {

      const response = await request('GET', '/boards');

      setEntities(response.data);

    } catch (error) {

      console.error('Error fetching data:', error);

    }

  };

  useEffect(() => {

    fetchData();

  }, []);

  const handleCreateBoard = () => {

    setIsModalOpen(true);

  };

  const handleCloseModal = () => {

    setIsModalOpen(false);

  };

  return (

    <div className="max-w-screen-xl items-start mx-auto">

      <p className="text-gray-700 font-semibold text-xl ml-4 mb-4">Доступные доски</p>

      <button className="bg-cyan-500 hover:bg-cyan-400 text-white font-semibold px-4 py-2 rounded-md mb-4 ml-4 hover:shadow-md" onClick={handleCreateBoard}>

        Создать доску

      </button>

      <div className="flex flex-wrap">

        {entities.map(entity => (

          <MemoizedEntityCard key={entity.id} entity={entity} />

        ))}

        <CreateBoardModal isOpen={isModalOpen} onClose={handleCloseModal} onBoardCreated={fetchData} />

      </div>

    </div>

  );

};

export default BoardsList;

**Приложение Д**  
(обязательное)  
**Программный код модульных тестов системы**

package com.scheduler.backend.tests;  
  
import com.fasterxml.jackson.databind.ObjectMapper;  
import com.scheduler.backend.controllers.AuthController;  
import com.scheduler.backend.dtos.CredentialsDto;  
import com.scheduler.backend.dtos.SignUpDto;  
import com.scheduler.backend.dtos.UserDto;  
import com.scheduler.backend.services.UserService;  
import com.scheduler.backend.config.UserAuthenticationProvider;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.extension.ExtendWith;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.Mockito;  
import org.mockito.junit.jupiter.MockitoExtension;  
import org.springframework.http.MediaType;  
import org.springframework.test.web.servlet.MockMvc;  
import org.springframework.test.web.servlet.setup.MockMvcBuilders;  
  
import static org.mockito.ArgumentMatchers.*any*;  
import static org.mockito.ArgumentMatchers.*eq*;  
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*post*;  
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*jsonPath*;  
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*status*;  
  
@ExtendWith(MockitoExtension.class)  
public class AuthControllerTest {  
  
 @Mock  
 private UserService userService;  
  
 @Mock  
 private UserAuthenticationProvider userAuthenticationProvider;  
  
 @InjectMocks  
 private AuthController authController;  
  
 private MockMvc mockMvc;  
  
 private ObjectMapper objectMapper = new ObjectMapper();  
  
 private UserDto userDto;  
 private CredentialsDto credentialsDto;  
 private SignUpDto signUpDto;  
  
 @BeforeEach  
 void setUp() {  
 mockMvc = MockMvcBuilders.*standaloneSetup*(authController).build();  
  
 userDto = UserDto.*builder*()  
 .id(1L)  
 .firstname("John")  
 .surname("Doe")  
 .email("john.doe@example.com")  
 .token("dummyToken")  
 .build();  
  
 credentialsDto = CredentialsDto.*builder*()  
 .email("john.doe@example.com")  
 .password("password".toCharArray())  
 .build();  
  
 signUpDto = SignUpDto.*builder*()  
 .firstname("John")  
 .surname("Doe")  
 .patronymic("M")  
 .email("john.doe@example.com")  
 .password("password".toCharArray())  
 .build();  
 }  
  
 @Test  
 void testLogin() throws Exception {  
 Mockito.*when*(userService.login(*any*(CredentialsDto.class))).thenReturn(userDto);  
 Mockito.*when*(userAuthenticationProvider.createToken(*eq*(userDto.getEmail()))).thenReturn("dummyToken");  
  
 mockMvc.perform(*post*("/login")  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(objectMapper.writeValueAsString(credentialsDto)))  
 .andExpect(*status*().isOk())  
 .andExpect(*jsonPath*("$.email").value(userDto.getEmail()))  
 .andExpect(*jsonPath*("$.token").value("dummyToken"));  
 }  
  
 @Test  
 void testRegister() throws Exception {  
 Mockito.*when*(userService.register(*any*(SignUpDto.class))).thenReturn(userDto);  
 Mockito.*when*(userAuthenticationProvider.createToken(*eq*(userDto.getEmail()))).thenReturn("dummyToken");  
  
 mockMvc.perform(*post*("/register")  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(objectMapper.writeValueAsString(signUpDto)))  
 .andExpect(*status*().isCreated())  
 .andExpect(*jsonPath*("$.email").value(userDto.getEmail()))  
 .andExpect(*jsonPath*("$.token").value("dummyToken"));  
 }  
}

package com.scheduler.backend.tests;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import com.fasterxml.jackson.datatype.jsr310.JavaTimeModule;  
import com.scheduler.backend.config.UserAuthenticationProvider;  
import com.scheduler.backend.controllers.BoardController;  
import com.scheduler.backend.dtos.BoardDto;  
import com.scheduler.backend.dtos.UserDto;  
import com.scheduler.backend.entities.User;  
import com.scheduler.backend.repositories.UserRepository;  
import com.scheduler.backend.services.BoardService;  
import jakarta.persistence.EntityNotFoundException;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.extension.ExtendWith;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.junit.jupiter.MockitoExtension;  
import org.springframework.http.MediaType;  
import org.springframework.security.core.Authentication;  
import org.springframework.test.web.servlet.MockMvc;  
import org.springframework.test.web.servlet.setup.MockMvcBuilders;  
  
import java.util.Collections;  
import java.util.List;  
  
import static org.mockito.ArgumentMatchers.\*;  
import static org.mockito.Mockito.*mock*;  
import static org.mockito.Mockito.*when*;  
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.\*;  
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;  
  
@ExtendWith(MockitoExtension.class)  
public class BoardControllerTest {  
  
 @Mock  
 private BoardService boardService;  
  
 @Mock  
 private UserAuthenticationProvider userAuthenticationProvider;  
  
 @Mock  
 private UserRepository userRepository;  
  
 @InjectMocks  
 private BoardController boardController;  
  
 private MockMvc mockMvc;  
  
 @BeforeEach  
 void setUp() {  
 mockMvc = MockMvcBuilders.*standaloneSetup*(boardController).build();  
 }  
  
 @Test  
 void getBoardById\_ReturnsBoard\_WhenValidId() throws Exception {  
 *// Arrange* Long boardId = 1L;  
 BoardDto boardDto = new BoardDto();  
 *when*(boardService.getBoardById(boardId)).thenReturn(boardDto);  
  
 *// Act & Assert* mockMvc.perform(*get*("/boards/{id}", boardId))  
 .andExpect(*status*().isOk())  
 .andExpect(*content*().contentType(MediaType.*APPLICATION\_JSON*));  
 }  
  
 @Test  
 void createBoard\_ReturnsCreatedBoard\_WhenValidInput() throws Exception {  
 *// Arrange* BoardDto inputDto = new BoardDto();  
 BoardDto createdDto = new BoardDto();  
 createdDto.setId(1L);  
  
 String jwt = "token";  
 UserDto userDto = new UserDto();  
 userDto.setId(1L);  
 User user = new User();  
  
 *when*(userAuthenticationProvider.validateToken(*anyString*())).thenReturn(*mock*(Authentication.class));  
 *when*(((Authentication) userAuthenticationProvider.validateToken(*anyString*())).getPrincipal()).thenReturn(userDto);  
 *when*(userRepository.findById(userDto.getId())).thenReturn(java.util.Optional.*of*(user));  
 *when*(boardService.createBoard(*any*(BoardDto.class), *any*(User.class))).thenReturn(createdDto);  
  
 *// Act & Assert* mockMvc.perform(*post*("/boards/create")  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(asJsonString(inputDto))  
 .header("Authorization", "Bearer " + jwt))  
 .andExpect(*status*().isCreated())  
 .andExpect(*content*().contentType(MediaType.*APPLICATION\_JSON*))  
 .andExpect(*jsonPath*("$.id").value(1));  
 }  
  
 @Test  
 void createBoard\_ReturnsNotFound\_WhenUserNotFound() throws Exception {  
 *// Arrange* BoardDto inputDto = new BoardDto();  
  
 String jwt = "some.valid.token";  
 UserDto userDto = new UserDto();  
 userDto.setId(1L);  
  
 *when*(userAuthenticationProvider.validateToken(*anyString*())).thenReturn(*mock*(Authentication.class));  
 *when*(((Authentication) userAuthenticationProvider.validateToken(*anyString*())).getPrincipal()).thenReturn(userDto);  
 *when*(userRepository.findById(userDto.getId())).thenThrow(new EntityNotFoundException("User not found"));  
  
 *// Act & Assert* mockMvc.perform(*post*("/boards/create")  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(asJsonString(inputDto))  
 .header("Authorization", "Bearer " + jwt))  
 .andExpect(*status*().isNotFound());  
 }  
  
 @Test  
 void getAllBoards\_ReturnsBoards\_WhenValidToken() throws Exception {  
 *// Arrange* String jwt = "token";  
 UserDto userDto = new UserDto();  
 userDto.setId(1L);  
 List<BoardDto> boards = Collections.*emptyList*();  
  
 *when*(userAuthenticationProvider.validateToken(*anyString*())).thenReturn(*mock*(Authentication.class));  
 *when*(((Authentication) userAuthenticationProvider.validateToken(*anyString*())).getPrincipal()).thenReturn(userDto);  
 *when*(boardService.getAllBoardsForCurrentUser(userDto.getId())).thenReturn(boards);  
  
 *// Act & Assert* mockMvc.perform(*get*("/boards")  
 .header("Authorization", "Bearer " + jwt))  
 .andExpect(*status*().isOk())  
 .andExpect(*content*().contentType(MediaType.*APPLICATION\_JSON*));  
 }  
  
 private String asJsonString(final Object obj) {  
 try {  
 ObjectMapper mapper = new ObjectMapper();  
 mapper.registerModule(new JavaTimeModule());  
 return mapper.writeValueAsString(obj);  
 } catch (Exception e) {  
 throw new RuntimeException(e);  
 }  
 }  
}

package com.scheduler.backend.tests;  
  
import com.fasterxml.jackson.databind.ObjectMapper;  
import com.scheduler.backend.config.UserAuthenticationProvider;  
import com.scheduler.backend.controllers.TaskController;  
import com.scheduler.backend.dtos.MoveTaskRequest;  
import com.scheduler.backend.dtos.TypeOfTaskDto;  
import com.scheduler.backend.services.TaskService;  
import com.scheduler.backend.services.TypeService;  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.extension.ExtendWith;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.junit.jupiter.MockitoExtension;  
import org.springframework.http.MediaType;  
import org.springframework.test.web.servlet.MockMvc;  
import org.springframework.test.web.servlet.setup.MockMvcBuilders;  
  
import com.fasterxml.jackson.datatype.jsr310.JavaTimeModule;  
  
import java.util.Collections;  
import java.util.List;  
import java.util.Map;  
  
import static org.mockito.ArgumentMatchers.\*;  
import static org.mockito.Mockito.*doNothing*;  
import static org.mockito.Mockito.*when*;  
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.\*;  
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;  
  
@ExtendWith(MockitoExtension.class)  
public class TaskControllerTest {  
  
 @Mock  
 private TaskService taskService;  
  
 @Mock  
 private UserAuthenticationProvider userAuthenticationProvider;  
  
 @InjectMocks  
 private TaskController taskController;  
  
 private MockMvc mockMvc;  
  
 @Test  
 void getTasksByType\_ReturnsTasks\_WhenValidTypeId() throws Exception {  
 *// Arrange  
 when*(taskService.getTasksByType(*anyLong*())).thenReturn(Collections.*emptyList*());  
 mockMvc = MockMvcBuilders.*standaloneSetup*(taskController).build();  
  
 *// Act & Assert* mockMvc.perform(*get*("/board/type/tasks/{typeId}", 1)  
 .header("Authorization", "Bearer auth\_token"))  
 .andExpect(*status*().isOk())  
 .andExpect(*content*().contentType(MediaType.*APPLICATION\_JSON*));  
 }  
  
 @Test  
 void moveTask\_ReturnsNoContent\_WhenValidInput() throws Exception {  
 *// Arrange* MoveTaskRequest moveTaskRequest = new MoveTaskRequest("1", 1L, 2L, 1);  
 *doNothing*().when(taskService).moveTask(*anyString*(), *anyLong*(), *anyLong*(), *anyInt*());  
  
 mockMvc = MockMvcBuilders.*standaloneSetup*(taskController).build();  
  
 *// Act & Assert* mockMvc.perform(*put*("/board/type/task/move")  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(asJsonString(moveTaskRequest))  
 .header("Authorization", "Bearer token"))  
 .andExpect(*status*().isNoContent());  
 }  
  
 @Test  
 void moveTasks\_ReturnsNoContent\_WhenValidInput() throws Exception {  
 *// Arrange* Map<String, List<MoveTaskRequest>> moveTasks = Collections.*singletonMap*("moveTasks", Collections.*emptyList*());  
 *doNothing*().when(taskService).moveTasks(*anyList*());  
  
 mockMvc = MockMvcBuilders.*standaloneSetup*(taskController).build();  
  
 *// Act & Assert* mockMvc.perform(*put*("/board/tasks/move")  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(asJsonString(moveTasks))  
 .header("Authorization", "Bearer token"))  
 .andExpect(*status*().isNoContent());  
 }  
  
 @Test  
 void deleteTask\_ReturnsNoContent\_WhenValidTaskId() throws Exception {  
 *// Arrange  
 doNothing*().when(taskService).deleteTask(*anyString*());  
  
 mockMvc = MockMvcBuilders.*standaloneSetup*(taskController).build();  
  
 *// Act & Assert* mockMvc.perform(*delete*("/board/type/task/{taskId}", "1")  
 .header("Authorization", "Bearer token"))  
 .andExpect(*status*().isNoContent());  
 }  
  
 private String asJsonString(final Object obj) {  
 try {  
 ObjectMapper mapper = new ObjectMapper();  
 mapper.registerModule(new JavaTimeModule());  
 return mapper.writeValueAsString(obj);  
 } catch (Exception e) {  
 throw new RuntimeException(e);  
 }  
 }

package com.scheduler.backend.tests;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import com.fasterxml.jackson.datatype.jsr310.JavaTimeModule;  
import com.scheduler.backend.config.UserAuthenticationProvider;  
import com.scheduler.backend.controllers.TypeController;  
import com.scheduler.backend.dtos.TypeOfTaskDto;  
import com.scheduler.backend.services.TypeService;  
import jakarta.persistence.EntityNotFoundException;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.extension.ExtendWith;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.junit.jupiter.MockitoExtension;  
import org.springframework.http.MediaType;  
import org.springframework.security.core.Authentication;  
import org.springframework.test.web.servlet.MockMvc;  
import org.springframework.test.web.servlet.setup.MockMvcBuilders;  
  
import java.util.Collections;  
import java.util.List;  
  
import static org.mockito.ArgumentMatchers.\*;  
import static org.mockito.Mockito.\*;  
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.\*;  
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;  
  
@ExtendWith(MockitoExtension.class)  
public class TypeControllerTest {  
  
 @Mock  
 private TypeService typeService;  
  
 @Mock  
 private UserAuthenticationProvider userAuthenticationProvider;  
  
 @InjectMocks  
 private TypeController typeController;  
  
 private MockMvc mockMvc;  
  
 @BeforeEach  
 void setUp() {  
 mockMvc = MockMvcBuilders.*standaloneSetup*(typeController).build();  
 }  
  
 @Test  
 void getTypesByBoardId\_ReturnsTypes\_WhenValidBoardId() throws Exception {  
 *// Arrange* Long boardId = 1L;  
 List<TypeOfTaskDto> taskTypes = Collections.*emptyList*();  
 String jwt = "token";  
  
 *when*(userAuthenticationProvider.validateToken(*anyString*())).thenReturn(*mock*(Authentication.class));  
 *when*(typeService.getTypesByBoardId(boardId)).thenReturn(taskTypes);  
  
 *// Act & Assert* mockMvc.perform(*get*("/board/types/{boardId}", boardId)  
 .header("Authorization", "Bearer " + jwt))  
 .andExpect(*status*().isOk())  
 .andExpect(*content*().contentType(MediaType.*APPLICATION\_JSON*));  
 }  
  
 @Test  
 void getTypesByBoardId\_ReturnsNotFound\_WhenEntityNotFound() throws Exception {  
 *// Arrange* Long boardId = 1L;  
 String jwt = "token";  
  
 *when*(userAuthenticationProvider.validateToken(*anyString*())).thenReturn(*mock*(Authentication.class));  
 *when*(typeService.getTypesByBoardId(boardId)).thenThrow(EntityNotFoundException.class);  
  
 *// Act & Assert* mockMvc.perform(*get*("/board/types/{boardId}", boardId)  
 .header("Authorization", "Bearer " + jwt))  
 .andExpect(*status*().isNotFound());  
 }  
  
 @Test  
 void createType\_ReturnsCreatedType\_WhenValidInput() throws Exception {  
 *// Arrange* Long boardId = 1L;  
 TypeOfTaskDto inputDto = new TypeOfTaskDto();  
 TypeOfTaskDto createdDto = new TypeOfTaskDto();  
 createdDto.setId(1L);  
 createdDto.setName("Test Type");  
  
 String jwt = "token";  
  
 *when*(userAuthenticationProvider.validateToken(*anyString*())).thenReturn(*mock*(Authentication.class));  
 *when*(typeService.createType(*eq*(boardId), *any*(TypeOfTaskDto.class))).thenReturn(createdDto);  
  
 *// Act & Assert* mockMvc.perform(*post*("/board/{boardId}/type", boardId)  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(asJsonString(inputDto))  
 .header("Authorization", "Bearer " + jwt))  
 .andExpect(*status*().isCreated())  
 .andExpect(*content*().contentType(MediaType.*APPLICATION\_JSON*))  
 .andExpect(*jsonPath*("$.id").value(1))  
 .andExpect(*jsonPath*("$.name").value("Test Type"));  
 }  
  
 @Test  
 void moveTypes\_ReturnsOk\_WhenValidInput() throws Exception {  
 *// Arrange* List<TypeOfTaskDto> updatedTypes = Collections.*emptyList*();  
 *doNothing*().when(typeService).moveTypes(*anyList*());  
  
 *// Act & Assert* mockMvc.perform(*put*("/board/types/move")  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(asJsonString(updatedTypes))  
 .header("Authorization", "Bearer token"))  
 .andExpect(*status*().isOk());  
 }  
  
 @Test  
 void updateType\_ReturnsUpdatedType\_WhenValidInput() throws Exception {  
 *// Arrange* Long typeId = 1L;  
 TypeOfTaskDto inputDto = new TypeOfTaskDto();  
 TypeOfTaskDto updatedDto = new TypeOfTaskDto();  
 updatedDto.setId(typeId);  
 updatedDto.setName("Updated Type");  
  
 String jwt = "token";  
  
 *when*(userAuthenticationProvider.validateToken(*anyString*())).thenReturn(*mock*(Authentication.class));  
 *when*(typeService.updateType(*eq*(typeId), *any*(TypeOfTaskDto.class))).thenReturn(updatedDto);  
  
 *// Act & Assert* mockMvc.perform(*put*("/board/type/{typeId}", typeId)  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(asJsonString(inputDto))  
 .header("Authorization", "Bearer " + jwt))  
 .andExpect(*status*().isOk())  
 .andExpect(*content*().contentType(MediaType.*APPLICATION\_JSON*))  
 .andExpect(*jsonPath*("$.id").value(typeId))  
 .andExpect(*jsonPath*("$.name").value("Updated Type"));  
 }  
  
 @Test  
 void deleteType\_ReturnsNoContent\_WhenValidTypeId() throws Exception {  
 *// Arrange* Long typeId = 1L;  
 String jwt = "token";  
  
 *doNothing*().when(typeService).deleteType(typeId);  
 *when*(userAuthenticationProvider.validateToken(*anyString*())).thenReturn(*mock*(Authentication.class));  
  
 *// Act & Assert* mockMvc.perform(*delete*("/board/type/{typeId}", typeId)  
 .header("Authorization", "Bearer " + jwt))  
 .andExpect(*status*().isNoContent());  
 }  
  
 private String asJsonString(final Object obj) {  
 try {  
 ObjectMapper mapper = new ObjectMapper();  
 mapper.registerModule(new JavaTimeModule());  
 return mapper.writeValueAsString(obj);  
 } catch (Exception e) {  
 throw new RuntimeException(e);  
 }  
 }  
}