МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 01 01 Программное обеспечение информационных технологий

Специализация 1–40 01 01 Программное обеспечение информационных технологий (программирование интернет – изданий)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Сервис для обучения работы с SQL»

Выполнил студент Мозолевский Александр Дмитриевич

(Ф.И.О.)

Руководитель проекта асс. Дубовик М.В.

(учен. степень, звание, должность, Ф.И.О., подпись)

Заведующий кафедрой к.т.н., доц. Смелов В.В.

(учен. степень, звание, должность, Ф.И.О., подпись)

Консультант:  асс. Дубовик М.В.

(учен. степень, звание, должность, Ф.И.О., подпись)

Нормоконтролер: асс. Дубовик М.В.

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовой проект защищен с оценкой

Минск 2023

Реферат

Пояснительная записка содержит 41 страницы, 29 иллюстраций, 7 источников литературы, 3 приложения.

NODEJS, NEST, ANGULAR, POSGRESQL, SOCKET.IO, CROSSPLATFORM APPLICATION, ANGULAR MATERIAL, REST API, ARGON2

Основная цель курсового проекта: проектирование базы данных и разработка приложения для решения задач SQL.

Пояснительная записка состоит из введения, пяти разделов, заключения и списка используемой литературы.

В введении формулирована цель и задачи проекта.

В первом разделе проводится постановка задачи, обзор аналогичных решений.

Второй раздел посвящен проектировании базы данных и веб-приложения.

В третьем разделе описаны детали реализации серверной и клиентской частей приложения.

В четвертом разделе описаны основные шаги тестирования проекта для разных компонентов.

В пятом разделе описано руководство пользователя, позволяющее понять принцип взаимодействия с интерфейсом клиентского приложения.

В заключении представлены итоги курсового проекта и задачи, которые были решены в ходе разработки проекта.

**Содержание**

[Введение 4](#_Toc134908903)

[1 Аналитический обзор литературы 5](#_Toc134908904)

[1.1 Аналитический обзор источников 5](#_Toc134908905)

[1.2 Обзор аналогов 5](#_Toc134908906)

[2 Проектирование программного средства 7](#_Toc134908907)

[2.1 Проектирование структуры базы данных 7](#_Toc134908908)

[2.2 Определение ролей приложения 9](#_Toc134908909)

[2.3 Проектирование структуры приложения 10](#_Toc134908910)

[3 Разработка программного средства 11](#_Toc134908911)

[3.1 Разработка модели базы данных 11](#_Toc134908912)

[3.2 Разработка серверной части 12](#_Toc134908913)

[3.2.1 Проектирование архитектуры сервера 13](#_Toc134908914)

[3.2.2 Конечные точки HTTP API 14](#_Toc134908915)

[3.3 Разработка клиентской части 15](#_Toc134908916)

[3.3.1 Разработка интерцепторов 15](#_Toc134908917)

[3.3.2 Разработка сервисов 17](#_Toc134908918)

[3.3.3 Разработка компонентов 19](#_Toc134908919)

[3.3.4 Маршрутизация приложения 20](#_Toc134908920)

[4 Тестирование приложения 21](#_Toc134908921)

[5 Руководство пользователя 24](#_Toc134908922)

[Заключение 32](#_Toc134908923)

[Список использованных источников 33](#_Toc134908924)

[Приложение А 34](#_Toc134908925)

[Приложение Б 35](#_Toc134908926)

[Приложение В 38](#_Toc134908927)

Введение

Целью данной работы является разработка кроссплатформенного веб-приложения на тему «Сервис для обучения работы с SQL». Приложение представляет из себя сервис, на котором предоставляется удобный функционал для ознакомления с основами написания запросов на языке программирования SQL.

Для написания веб-приложения был выбран язык программирования TypeScript. TypeScript – язык программирования, представленный Microsoft в 2012 году и позиционируемый как средство разработки веб-приложений [1].

Приложение строится на клиент-серверной архитектуре. Для серверной части приложения был выбран фреймворк Nest. Nest – это платформа для создания эффективных масштабируемых программ Node.js на стороне сервера. Он использует JavaScript, создан и полностью поддерживает TypeScript и объединяет элементы ООП, ФП и ФРП.

Для клиентской части приложения был выбран фреймворк Angular. Angular – открытая и свободная платформа для разработки веб-приложений, написанная на языке TypeScript, разрабатываемая командой из компании Google, а также сообществом разработчиков из различных компаний.

Для хранения информации необходимо использовать базы данных, в данном курсовом проекте было решено использовать PostgreSQL. PostgreSQL – свободная объектно-реляционная система управления базами данных.

Данный курсовой проект должен выполнять следующие требования:

* обеспечивать возможность регистрации и авторизации;
* поддерживать роли администратора и пользователя;
* реализовать создание задач и их решение;
* обеспечивать комментирования задач пользователями;
* позволять просматривать задачи;
* обеспечивать возможность сортировки задач по времени создания и сложности;
* поддерживать статистику пользователей для построения таблицы лидеров;
* обеспечивать возможность участвовать в обсуждении между пользователями.

Пояснительная записка содержит краткую информацию о похожих продуктах, архитектуре, реализации проекта, руководстве пользователя, а также информацию о тестировании приложения.

1 Аналитический обзор литературы

1.1 Аналитический обзор источников

В ходе разработки приложения была изучена специальная техническая, учебно-методическая и справочная литература, статьи и материалы, опубликованные в сети интернет.

Основная информация о работе с фреймворками была получения из их официальных документаций.

Также были использованы интернет-источники и статьи для детального изучения некоторых аспектов работы с технологиями.

1.2 Обзор аналогов

Для создания принципиально нового решения в виде программного продукта, используемого в качестве сервиса для обучения работы с SQL, необходимо проанализировать уже существующие программные средства в данной сфере и выделить их недостатки, которые следует исправить в готовом продукте.

В результате поиска были рассмотрены следующие ресурсы: PostgreSQL Exercises и SQLBolt.

PostgreSQL Exercises представляет собой сервис, который позволяет изучать написание SQL-запросов. Интерфейс данного сайта представлен на рисунке 1.1.

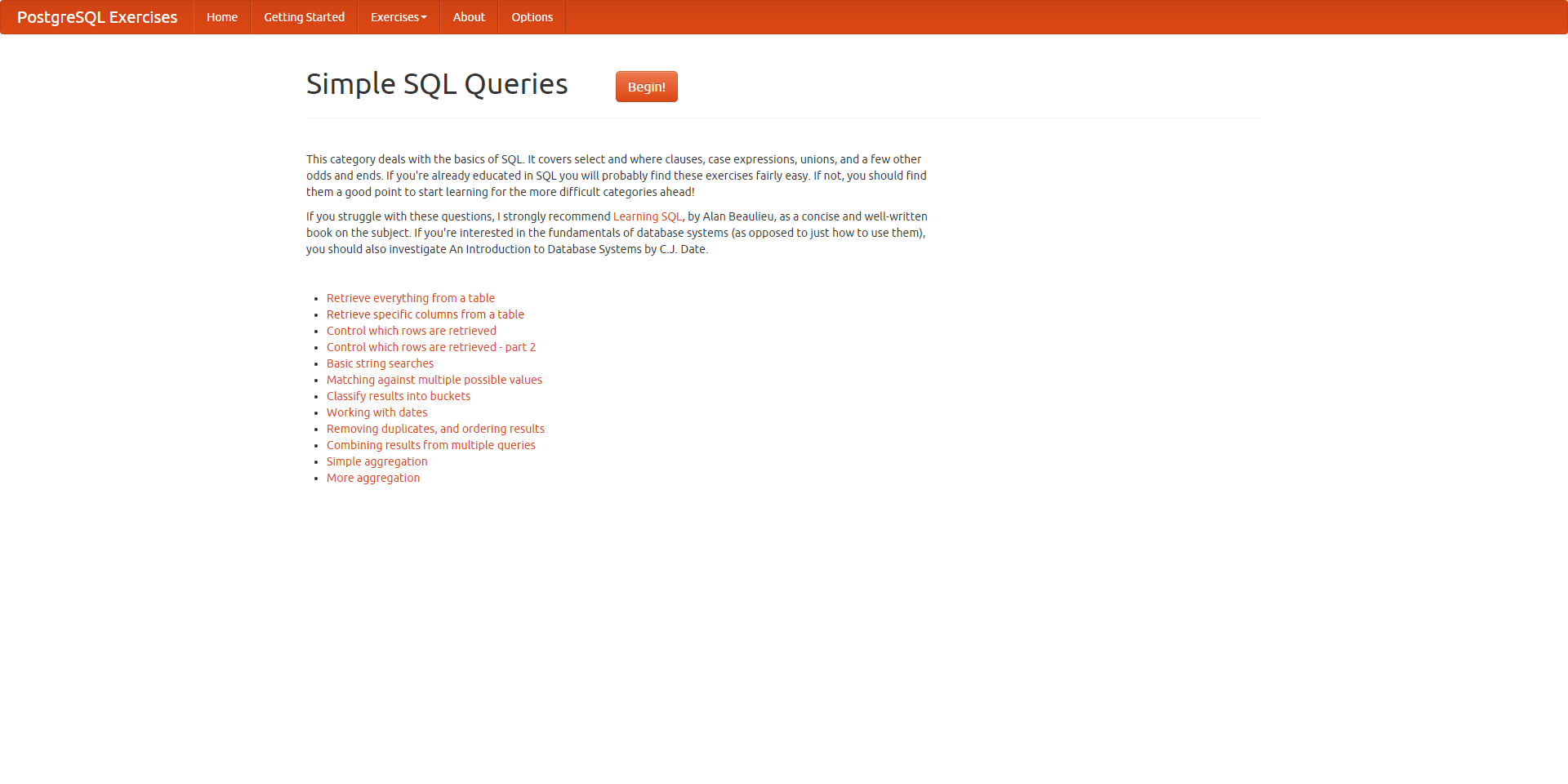


Рисунок 1.1 – Главная страница сайта «PostgreSQL Exercises»

Он предоставляет возможность поиска задач по определённым темам и проверки их решения. Из недостатков стоит отметить, что данный сайт не разделяет задачи по сложностям и не содержит их обсуждений, что может затруднить обучение.

SQLBolt является сервисом интерактивных уроков и упражнений для обучения работы с SQL. Интерфейс данного сайта представлен на рисунке 1.2.

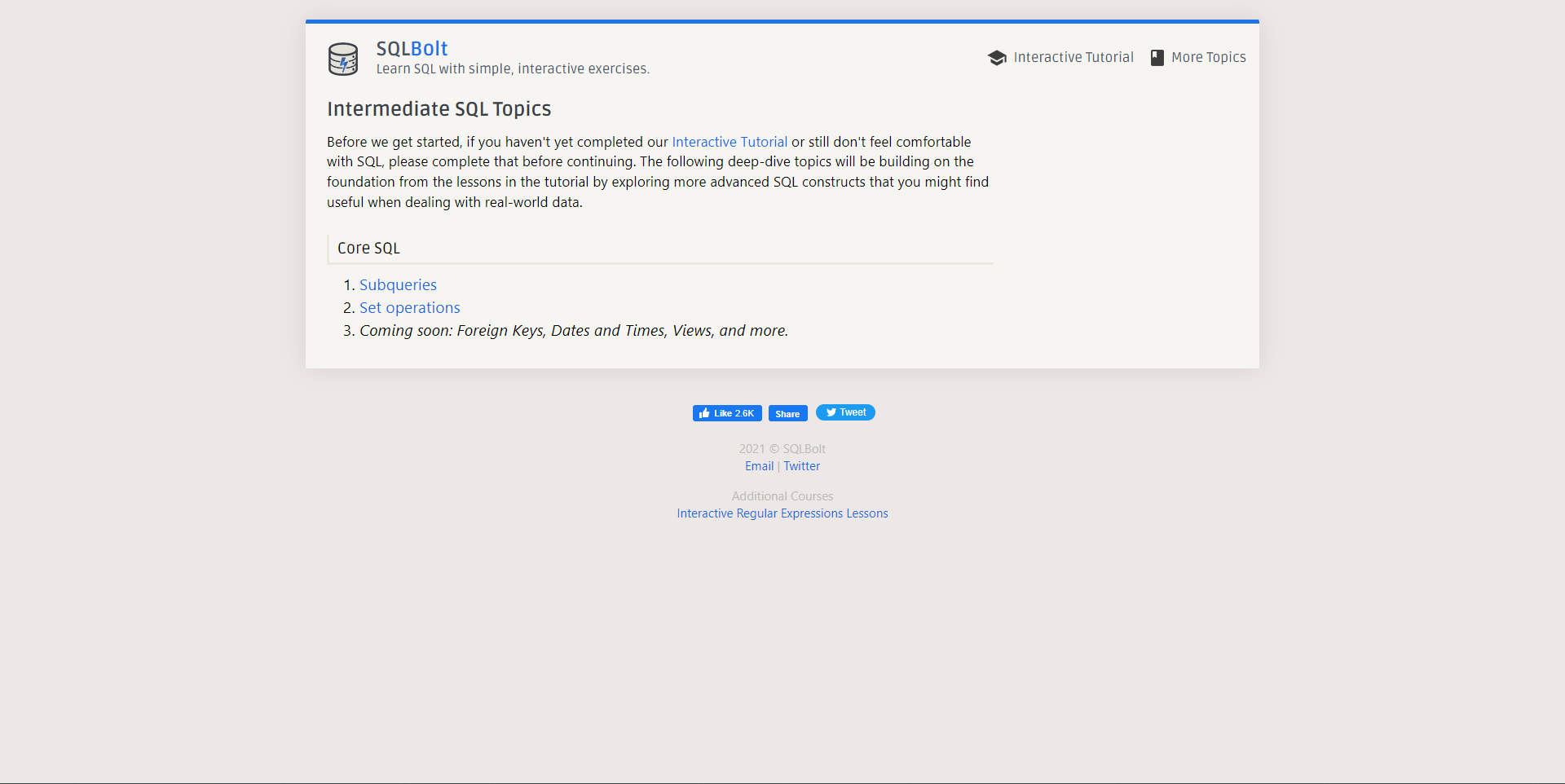


Рисунок 1.2 – Главная страница сайта «SQLBolt»

Данный сайт позволяет изучать написание SQL-запросов поэтапно, задачи затрагивают только основные операторы языка программирования SQL, без использования дополнительных встроенных функций. Это означает, что данный сайт способен лишь обучить основам написания SQL-запросов, и не подходит для продвинутых пользователей.

Оба рассмотренных сайта не предоставляют возможности создания своих задач, что снижает расширяемость приложения.

Отсутствие таблицы лидеров и личного профиля также может привести к тому, что пользователи не будут чувствовать себя мотивированными к достижению больших результатов и улучшению своих навыков. Эти функции могут помочь пользователям отслеживать свой прогресс, устанавливать цели и сравнивать свои результаты с результатами других пользователей.

Отсутствие возможности обсуждения задач может привести к тому, что пользователи не будут иметь возможности общаться и обмениваться мнениями с другими участниками. Общение может помочь пользователям получить новые знания и перспективы, а также научиться решать проблемы более эффективно.

Все отмеченные достоинства и недостатки будут учитываться в дальнейшем написании приложения.

2 Проектирование программного средства

Проектирование программного средства делится на три этапа:

* проектирование структуры базы данных;
* определение ролей приложения;
* проектирование структуры приложения.

2.1 Проектирование структуры базы данных

В данном курсовом проекте используется две базы данных: основная база данных для хранения данных приложения и база данных, которая хранит таблицы, над которыми пользователь сможет выполнять задачи. В данной главе будет рассмотрена основная база данных.

В базе данных находится 10 таблиц. Таблица Users используется для хранения данных, идентифицирующих пользователей, зарегистрированных в приложении, User\_roles – роли пользователей, Discussions – обсуждения, в которых может принимать участие пользователь, Messages – список сообщений внутри обсуждений, Statistic – статистика каждого пользователя, Comments – список комментариев, оставляемых пользователем под задачами, Tasks – список задач, Task\_complexities – уровни сложности задач, Solutions – список решений задач пользователями, Databases – информация о используемых схемах для задач.

Таблица Users состоит из следующих столбцов:

* id – тип данных integer;
* username – тип данных varchar;
* email – тип данных varchar;
* password – тип данных varchar;
* activation\_code – тип данных text;
* role\_id – тип данных integer.

Столбец username содержит имя пользователя, email – почта пользователя, которая используется как логин в процессе авторизации, password – хешированный пароль алгоритмом Argon2, activation\_code – код активации, который отправляется пользователю на указанных email, role\_id – идентификатор роли пользователя.

Таблица User\_roles состоит из следующих столбцов:

* id – тип данных integer;
* name – тип данных varchar.

Столбец name хранит название роли, которая может быть присвоена пользователю.

Таблица Discussions состоит из следующих столбцов:

* id – тип данных integer;
* topic – тип данных varchar.

Столбец topic хранит название обсуждения, создаваемого пользователями.

Таблица Task\_complexities состоит из следующих столбцов:

* id – тип данных integer;
* name – тип данных varchar.

Столбец name хранит название уровня сложности, которая может быть присвоена отдельной задаче, значения задаются как константы и содержат следующие строки: easy, medium, hard.

Таблица Messages состоит из следующих столбцов:

* id – тип данных integer;
* content – тип данных varchar;
* creation\_time – тип данных timestamp;
* discussion\_id – тип данных integer;
* user\_id – тип данных integer.

Столбец content хранит в себе содержание сообщения, creation\_time – время создания сообщения, discussion\_id – идентификатор обсуждения, в которое отправлено сообщение, user\_id – идентификатор пользователя, который отправил сообщение.

Таблица Statistic состоит из следующих столбцов:

* id – тип данных integer;
* tasks\_completed – тип данных integer;
* score – тип данных integer;
* user\_id – тип данных integer.

Столбец tasks\_completed хранит в себе количество задач, решенных пользователем, score – текущий счёт пользователя, который увеличивается при решении задач различной сложности, user\_id – идентификатор пользователя, к которому относится данная статистика.

Таблица Comments состоит из следующих столбцов:

* id – тип данных integer;
* content – тип данных varchar;
* creation\_time – тип данных timestamp;
* user\_id – тип данных integer;
* task\_id – тип данных integer;
* reported – тип данных boolean.

Столбец content хранит в себе содержание комментария, отправленного пользователем, creation\_time – время создания комментария, user\_id – идентификатор пользователя, который отправил комментарий, task\_id – идентификатор задачи, к которой был отправлен комментарий, reported – отметка, которая сообщает о том, содержит ли данный комментарий жалобу от какого-либо пользователя приложения.

Таблица Tasks состоит из следующих столбцов:

* id – тип данных integer;
* title – тип данных varchar;
* description – тип данных varchar;
* creation\_time– тип данных timestamp;
* solved\_times – тип данных integer;
* solution – тип данных json;
* accepted – тип данных boolean;
* complexity\_id – тип данных integer;
* database\_id – тип данных integer.

Столбец title хранит в себе название задачи, description – описание задачи, creation\_time – время создания задачи, solved\_times – счётчик количества решений задачи, solution – решение задачи, accepted – отметка о том, принята ли задача администратором, complexity\_id – идентификатор уровня сложности задачи, database\_id – идентификатор схемы таблиц, над которыми строится задача.

Таблица Solutions состоит из следующих столбцов:

* id – тип данных integer;
* query – тип данных text;
* solution\_time – тип данных timestamp;
* task\_id – тип данных integer;
* user\_id – тип данных integer.

Столбец query хранит в себе select-запрос, которым пользователь решил задачу, solution\_time – время, когда задача была решена, task\_id – идентификатор задачи, которую решил пользователь, user\_id – идентификатор пользователя, который решил задачу.

Таблица Databases состоит из следующих столбцов:

* id – тип данных integer;
* name – тип данных varchar;
* image\_path – тип данных varchar.

Столбец name хранит в себе имя схемы, которая используется для проектирования задач, image\_path – путь к картинке схемы.

Это формальное описание таблиц основной базы данных приложения, её логическая схема представлена в приложении А.

2.2 Определение ролей приложения

Роли в приложении – это набор правил, определяющих действия, которые могут выполнять пользователи в системе. Каждый пользователь может иметь разные уровни доступа и разные возможности в зависимости от его роли в системе. В многопользовательской системе, такой как веб-приложение, роли играют важную роль в обеспечении безопасности и правильной работы приложения.

Два основных типа ролей, используемых во многих приложениях, – это роли "Пользователь" и "Администратор". Роль "Пользователь" обычно предоставляет ограниченный доступ к приложению, позволяя пользователю просматривать и редактировать только его собственные данные и информацию. Роль "Администратор" предоставляет полный доступ к приложению, позволяя пользователю просматривать и редактировать все данные и информацию в приложении, а также управлять другими пользователями и их правами.

Правильное определение ролей в приложении помогает обеспечить безопасность, защитить данные и предотвратить нежелательные действия в системе. Определение ролей также помогает лучше организовать код приложения, разделяя функциональность по разным ролям и позволяя более эффективно управлять приложением в целом.

Для выполнения поставленных задач необходимо реализовать две роли: пользователь и администратор.

Диаграмма возможностей для авторизированного пользователя, неавторизированного пользователя и администратора представлена на рисунке 2.1.

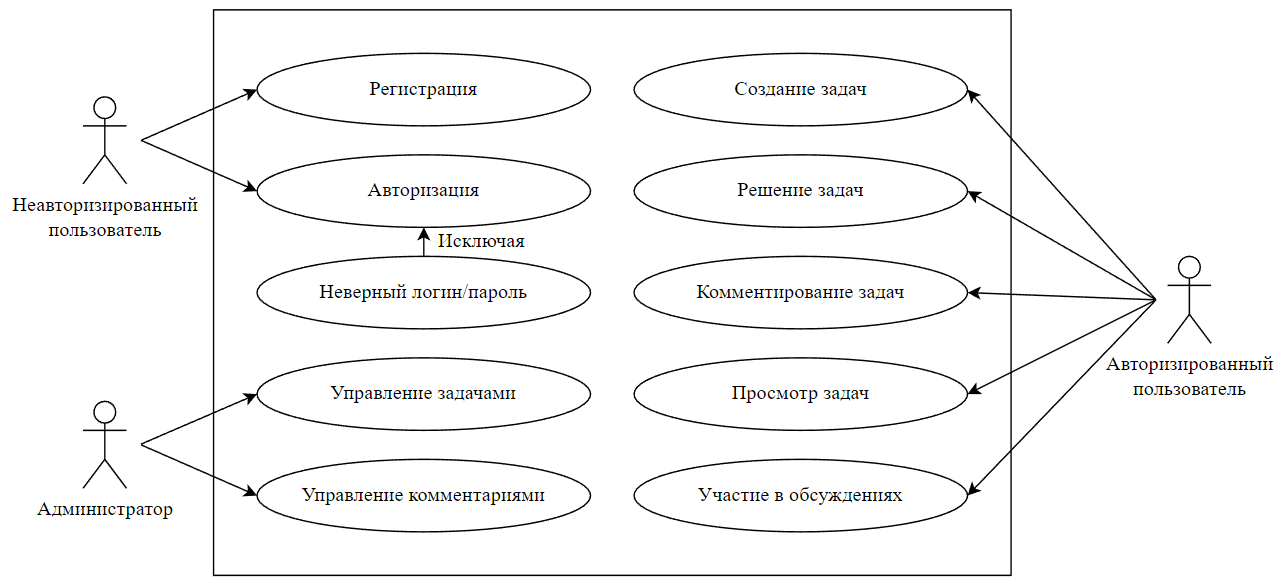


Рисунок 2.1 – Диаграмма возможностей

2.3 Проектирование структуры приложения

Для разрабатываемого проекта была разработана диаграмма развертывания, представленная на рисунке 3.1.

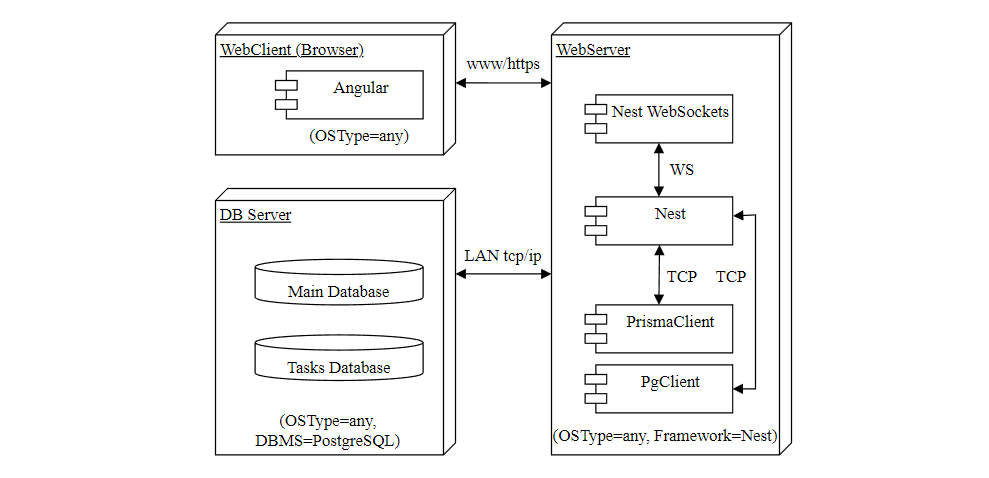


Рисунок 2.2 – Диаграмма развёртывания

Само приложение будет работать по принципу клиент-серверного взаимодействия. Серверная часть приложения будет работать на базе фреймворка Nest, клиентская часть на базе фреймворка Angular, база данных – PostgreSQL.

3 Разработка программного средства

При разработке курсового проекта весь процесс был разбит на следующие три этапа:

* разработка модели базы данных;
* разработка серверной части приложения;
* разработка клиентской части приложения.

3.1 Разработка модели базы данных

Проектирование модели базы данных – это важный этап в разработке информационной системы. Одним из подходов к проектированию модели базы данных является объектно-реляционное отображение (ORM), который используется для связывания объектной модели приложения с реляционной моделью базы данных.

ORM позволяет разработчикам использовать объектно-ориентированный подход при работе с базой данных. Это позволяет создавать более гибкие и удобные приложения, поскольку объектно-ориентированный подход более естественен для программистов, чем работа с реляционными таблицами и SQL-запросами.

Для реализации ORM были рассмотрены различные инструменты, и выбор был сделан в пользу Prisma. Prisma – это ORM для Node.js и TypeScript, который обеспечивает быстрый и безопасный доступ к базе данных [2]. Он предоставляет мощные инструменты для создания и управления базами данных, а также облегчает разработку приложений, использующих базы данных.

Одним из преимуществ использования Prisma является его автоматическая генерация кода на основе модели базы данных. Это позволяет быстро и легко создавать новые таблицы и поля в базе данных, а также выполнять различные операции с базой данных. Сгенерированный код на основе модели базы данных называется схемой, который представлен в виде листинга приложении Б.

Для работы с Prisma в данном проекте используется класс PrismaService, который расширяет функционал базового класса PrismaClient. Код данного класса представлен в листинге 3.1.

|  |
| --- |
| @Injectable()  export class PrismaService extends PrismaClient implements OnModuleInit {  async onModuleInit() {  await this.$connect();  }  async enableShutdownHooks(app: INestApplication) {  this.$on('beforeExit', async () => {  await app.close();  });  }  } |

Листинг 3.1 – Класс для работы с Prisma

В этом классе реализуется интерфейс OnModuleInit, что позволяет использовать метод onModuleInit, который вызывается после инициализации модуля приложения. Внутри метода onModuleInit происходит соединение с базой данных через вызов метода $connect(), который предоставляет PrismaClient.

Таким образом, класс PrismaService является ключевым компонентом для работы с Prisma и предоставляет функционал для инициализации и завершения работы с базой данных. Реализация метода enableShutdownHooks гарантирует корректное завершение работы приложения и сохранение данных в базе данных.

Поскольку серверная часть не изменяет вторую базу данных для задач, а использует её только для выполнения запросов пользователя, достаточно использовать примитивное соединение для работы с ней. Чтобы обеспечить примитивное соединение с базой данных для задач был написан модуль, код которого представлен в листинге 3.2.

|  |
| --- |
| @Module({  imports: [ConfigModule],  providers: [  {  provide: 'TASKS\_DB\_CONNECTION',  useFactory: async (configService: ConfigService) => {  const client = new Client({  user: configService.get<string>('DB\_USER'),  host: configService.get<string>('DB\_HOST'),  database: configService.get<string>('DB\_NAME'),  password: configService.get<string>('DB\_PASSWORD'),  port: configService.get<number>('DB\_PORT')  });  await client.connect();  return client;  },  inject: [ConfigService]  },  ],  exports: ['TASKS\_DB\_CONNECTION'],  })  export class ExercisesModule { } |

Листинг 3.2 – Модуль для обеспечения примитивного соединения с базой данных

3.2 Разработка серверной части

Для разработки серверной части приложения был выбран фреймворк Nest. Nest – это фреймворк для разработки масштабируемых и модульных веб-приложений на языке TypeScript и JavaScript [3]. Nest предоставляет разработчикам множество инструментов для создания современных веб-приложений, включая удобный механизм инъекции зависимостей, модульную архитектуру, возможность создания RESTful API, поддержку GraphQL, WebSocket и многие другие.

3.2.1 Проектирование архитектуры сервера

Архитектура приложения Nest, созданного с использованием генератора ресурсов (resource), представляет собой модульную структуру, которая позволяет разделять приложение на небольшие, изолированные компоненты.

Каждый компонент представляет собой отдельный модуль приложения, который содержит в себе необходимую функциональность и зависимости, а также экспортирует API для взаимодействия с другими компонентами.

Генерация ресурсов позволяет создавать быстро и удобно новые модули приложения, используя принципы RESTful API. При использовании генератора ресурсов создаются следующие файлы:

* контроллер (controller): файл, содержащий логику обработки запросов и взаимодействия с сервисом;
* сервис (service): файл, содержащий логику работы с данными и выполнения бизнес-логики приложения;
* модуль (module): файл, содержащий определение модуля приложения для данного ресурса.

Каждый модуль содержит в себе контроллер, сервис и модель, которые связаны между собой через инъекцию зависимостей. Таким образом, созданный модуль предоставляет полноценный функционал для работы с данными и взаимодействия с клиентской частью приложения.

В серверной части данного проекта было разработано 13 модулей, название и назначение которых представлено в таблице 3.1.

Таблица 3.1 – Модули серверной части приложения

|  |  |
| --- | --- |
| Название | Назначение |
| App | Основной модуль приложения, который определяет все необходимые компоненты и провайдеры для запуска приложения на базе Nest, а также связывает их вместе, чтобы создать рабочее веб-приложение. |
| Auth | Модуль, который обеспечивание аутентификацию и авторизацию пользователей в веб-приложении. |
| Chat | Модуль, который предоставляет пользователям веб-приложения общение между собой через WebSocket. |
| Comment | Модуль для работы с комментариями. |
| Discussion | Модуль для работы с обсуждениями. |
| Exercises | Модуль для работы с второй базой данных, который позволяет отправлять select-запросы пользователей и получать их результат. |
| Message | Модуль для работы с сообщениями. |
| Pagination | Модуль для пагинации, который позволяет ограничить ответ от сервера разделив его на страницы. |
| Schema | Модуль для работы со схемами таблиц. |
| Solution | Модуль для работы с решениями задач. |
| Statistic | Модуль для работы со статистикой пользователей. |
| Task | Модуль для работы с задачами. |
| User | Модуль для работы с аккаунтами пользователей. |

3.2.2 Конечные точки HTTP API

Для упрощения взаимодействия сторонних приложений с сервисом было разработано API, содержащее необходимые методы для взаимодействия с частями приложения. Данное API используется в клиентской части приложения.

Всего разработано 9 контроллеров, каждый из которых содержит методы, обрабатывающие входящие запросы. Для взаимодействия с частью методов необходима авторизация.

Всего в проекте 27 REST API методов, которые можно разделить на 9 областей соответственно ответственности. Описание доступных методов REST API приведено в таблице 3.2.

Таблица 3.2 – Описание методов REST API

|  |  |  |
| --- | --- | --- |
| Адрес | Метод | Описание |
| /api/comment/:id | GET | Получение комментариев задачи |
| /api/comment/:id | PUT | Отметка о жалобе на комментарий |
| /api/comment/:id | DELETE | Удаление комментария |
| /api/comment/reported | GET | Получение комментариев с жалобой |
| /api/comment | POST | Создание комментария |
| /api/statistic | GET | Получение статистики |
| /api/task/query | POST | Выполнение select-запроса пользователя |
| /api/task | POST | Создание задачи |
| /api/task/check | POST | Проверка решения пользователя |
| /api/task | GET | Получение списка задач |
| /api/task/solved | GET | Получение списка решённых задач |
| /api/task/:id | GET | Получение информации о задаче |
| /api/task/unconfirmed | GET | Получение списка неподтверждённых задач |
| /api/task/:id | PUT | Принятие задачи |
| /api/task/:id | DELETE | Удаление задачи |
| /api/profile | GET | Получение профилей пользователей |
| /api/profile/:username | GET | Получение профиля пользователя |
| /api/register | POST | Регистрация нового пользователя |
| /api/login | POST | Вход в аккаунт |
| /api/login/access-token | POST | Обновление access токена |
| /api/schema | GET | Получение списка схем таблиц |
| /api/schema/:id | GET | Получение схемы таблиц задачи |
| /api/discussion/:id/message | GET | Получение сообщений в обсуждении |
| /api/discussion/:id/message | POST | Создание сообщения в обсуждении |
| /api/discussion/ | GET | Получение списка обсуждений |
| /api/discussion/ | POST | Создание обсуждения |
| /api/discussion/:id | DELETE | Удаление обсуждения |

Наглядно таблица REST API методов сервера может быть представлена с помощью библиотеки Swagger.

3.3 Разработка клиентской части

Для разработки клиентской части приложения был выбран фреймворк Angular. Angular – это один из наиболее популярных фреймворков для разработки веб-приложений на базе JavaScript и TypeScript. Он был создан и поддерживается компанией Google и используется для создания масштабируемых, высокопроизводительных веб-приложений [4].

Основное назначение Angular – это упрощение и ускорение разработки сложных веб-приложений путем предоставления разработчикам удобных инструментов и библиотек для создания, тестирования и развертывания приложений.

Одной из главных особенностей Angular является использование компонентной архитектуры, которая позволяет разбивать приложение на отдельные компоненты, каждый из которых отвечает за определенную часть интерфейса и логику взаимодействия с пользователем.

Также Angular предоставляет разработчикам множество инструментов для создания динамических интерфейсов, таких как двустороннее связывание данных, директивы, пайпы, роутинг и многие другие.

Использование Angular обосновано в первую очередь тем, что этот фреймворк позволяет ускорить разработку сложных веб-приложений, упростить их поддержку и обеспечить высокую производительность и стабильность работы.

3.3.1 Разработка интерцепторов

Интерцепторы – это специальные классы, которые позволяют перехватывать и обрабатывать запросы и ответы. Они позволяют модифицировать данные, добавлять заголовки, обрабатывать ошибки и выполнять другие операции до и после отправки запроса.

Интерцепторы в Angular представлены в виде сервисов, которые могут быть добавлены в любой модуль или компонент приложения. Они могут использоваться для обработки любых HTTP-запросов, которые выполняются в приложении, а также для обработки запросов к другим службам или API.

Для создания интерсепторов в Angular необходимо реализовать HttpInterceptor интерфейс и определить метод intercept(). Метод intercept() выполняется для каждого HTTP-запроса, и в нем можно выполнять любые операции, например, добавлять заголовки, изменять URL-адрес, обрабатывать ошибки или изменять ответ.

Использование интерсепторов в Angular обеспечивает более гибкий и удобный способ обработки HTTP-запросов, чем использование других методов, таких как обработчики событий или фильтры. Интерсепторы позволяют создавать модульные и переиспользуемые компоненты, которые могут быть использованы в различных частях приложения, что делает их удобными и эффективными инструментами для разработки веб-приложений.

В данном курсовом проекте было разработано 3 интерцептора для разных задач.

Первым будет рассмотрен интерцептор для изменения URL. ApiInterceptor – это интерцептор, который применяется для обработки HTTP-запросов к API серверу. Этот интерцептор выполняет операцию перехвата и модификации запроса, добавляя к его URL-адресу префикс API сервера. Код данного интерцептора представлен в листинге 3.3.

|  |
| --- |
| @Injectable({ providedIn: 'root' })  export class ApiInterceptor implements HttpInterceptor {  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {  const apiReq = req.clone({ url: `http://localhost:3000/api${req.url}` });  return next.handle(apiReq);  }  } |

Листинг 3.3 – Интерцептор для изменения URL

Второй интерцептор используется для добавления токена авторизации к запросу на сервер. Это происходит путём добавления токена в заголовки запроса, где ключом является строка Authorization. Код данного интерцептора представлен в листинге 3.4.

|  |
| --- |
| @Injectable({  providedIn: 'root'  })  export class AuthInterceptor implements HttpInterceptor {  constructor(  private tokenStorage: TokenStorageService  ) { }  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {  let authRequest = req;  const token = this.tokenStorage.getToken();  if (token != null) {  authRequest = req.clone({ headers: req.headers.set(TOKEN\_HEADER\_KEY, `Bearer ${token}`) });  }  return next.handle(authRequest);  }  } |

Листинг 3.4 – Интерцептор для добавления токена авторизации

Последний интерцептор используется для обработки ошибок. Если с сервера приходит ответ с кодом 401, то этот интерцептор удаляет текущий токен авторизации из хранилища и пользователь выходит из системы.

При других ошибочных кодах, данный интерцептор выбрасывает исключение для дальнейшей обработки. Код данного интерцептора представлен в листинге 3.5.

|  |
| --- |
| @Injectable()  export class ErrorInterceptor implements HttpInterceptor {  constructor(  private tokenStorage: TokenStorageService  ) { }  intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {  return next.handle(request).pipe(catchError(err => {  if (err.status === 401) {  this.tokenStorage.logOut();  }  const error = err.error.message || err.statusText;  return throwError(() => new Error(error));  }));  }  } |

Листинг 3.5 – Интерцептор для обработки ошибок

Разработанные интерцепторы позволяют приложению безопасно взаимодействовать с сервером и снижают повторяемость кода.

3.3.2 Разработка сервисов

Сервисы Angular – это классы, которые выполняют определенные задачи, не относящиеся к представлению данных, но необходимые для функционирования приложения. Это может быть любой тип функциональности, например, доступ к данным, аутентификация пользователя, логирование или обработка ошибок.

Сервисы в Angular предназначены для обеспечения повторного использования кода и обеспечения чистоты кода компонентов, которые являются основными строительными блоками приложения. Сервисы могут использоваться компонентами для выполнения необходимых операций и получения необходимых данных, а также другими сервисами для совместной работы.

Сервисы в Angular предоставляются в виде инъекций зависимостей, что означает, что их можно использовать в других классах и компонентах приложения. Это позволяет легко расширять функциональность приложения, не создавая дублированный код.

Создание сервиса в Angular начинается с создания класса, который реализует необходимую функциональность. Затем класс регистрируется в качестве сервиса с помощью механизма инъекции зависимостей. Классы сервисов могут иметь зависимости, которые также могут быть зарегистрированы в качестве сервисов и инъецироваться в основной сервис.

Сервисы Angular являются важной частью разработки приложений на этой платформе, так как позволяют создавать масштабируемые, модульные и легко поддерживаемые приложения.

В данном курсовом проекте было разработано 10 сервисов, название и назначение которых представлено в таблице 3.3.

Таблица 3.3 – Сервисы клиентской части приложения

|  |  |
| --- | --- |
| Название | Назначение |
| AuthGuard | Сервис для проверки пользователя на аутентификацию, если пользователь не авторизирован, он не сможет получить доступ к некоторым компонентам. |
| Auth | Сервис для аутентификации пользователя. |
| Comment | Сервис для комментирования. |
| Discussion | Сервис для обсуждений. |
| Notification | Сервис для уведомлений пользователя. |
| Schema | Сервис для схем таблиц. |
| Statistic | Сервис для статистики. |
| Task | Сервис для задач. |
| TokenStorage | Сервис для хранения токена авторизации пользователя. |
| User | Сервис для работы с профилем пользователя. |

Для примера работы с сервисами будет рассмотрен AuthService, код которого представлен в листинге 3.6.

|  |
| --- |
| export class AuthService {  constructor(  private http: HttpClient  ) { }  public login(email: string, password: string): Observable<any> {  return this.http.post('/auth/login', {  email,  password  });  }  public register(username: string, email: string, password: string): Observable<any> {  return this.http.post('/auth/register', {  username,  email,  password  });  }  } |

Листинг 3.6 – Сервис для авторизации

Данный код представляет собой класс AuthService, который предназначен для выполнения операций аутентификации пользователя в приложении.

Класс содержит два публичных метода login и register, которые позволяют пользователю войти в систему и зарегистрироваться.

Класс AuthService имеет конструктор, который принимает экземпляр сервиса HttpClient, который используется для выполнения HTTP-запросов к серверу.

Метод login принимает параметры email и password в качестве аргументов и выполняет POST-запрос на конечную точку сервера /auth/login с передачей данных в теле запроса в виде объекта. Метод возвращает объект типа Observable<any>, который представляет поток данных, получаемых в ответ на запрос.

Метод register также принимает параметры username, email и password, выполняет POST-запрос на /auth/register с передачей данных в теле запроса в виде объекта. Метод также возвращает объект типа Observable<any>.

3.3.3 Разработка компонентов

Одним из ключевых элементов приложения являются компоненты. Компоненты Angular представляют собой классы TypeScript, которые содержат данные, связанные с пользовательским интерфейсом, а также логику, связанную с управлением этим интерфейсом. Каждый компонент может быть использован как самостоятельный элемент, который может быть добавлен в другие компоненты или шаблоны приложения. Пример рабочего компонента представлен в приложении В.

Данный курсовой проект состоит из 13 компонентов, название и описание которых представлено в таблице 3.4.

Таблица 3.4 – Компоненты клиентской части приложения

|  |  |
| --- | --- |
| Название | Описание |
| Auth | Компонент для авторизации. |
| Challenge | Компонент для работы с задачей. |
| Challenges | Компонент для отображения списка задач. |
| Chat | Компонент для взаимодействия с чатом. |
| Comments | Компонент для отображения списка комментариев. |
| DiscussionDialog | Компонент для модального окна создания обсуждения. |
| Discussions | Компонент для отображения списка обсуждений. |
| Homepage | Компонент для отображения домашней страницы приложения. |
| Navigation | Компонент для взаимодействия с навигацией. |
| NotFound | Компонент для оповещения пользователя об ошибке 404 (Not Found, ресурс не найден) |
| Profile | Компонент для отображения профиля пользователя. |
| Sandbox | Компонент для создания задачи. |
| Statistic | Компонент для отображения статистики пользователей. |

Каждый компонент Angular состоит из трех основных частей: шаблон, класс и метаданные. Шаблон представляет собой HTML-код, который определяет структуру и внешний вид компонента. Класс представляет собой TypeScript-код, который определяет логику и данные компонента. Метаданные определяют конфигурацию компонента, такую как его селектор, стили и другие параметры.

3.3.4 Маршрутизация приложения

Маршрутизация Angular представляет собой механизм, который позволяет определить маршруты для переходов между компонентами приложения в зависимости от текущего URL-адреса. Он используется для организации навигации между страницами веб-приложения и позволяет отображать различные компоненты приложения на основе текущего маршрута.

Маршрутизация Angular обеспечивается модулем RouterModule, который предоставляет необходимые сервисы и директивы для настройки маршрутов и обработки навигации в приложении. Маршруты определяются с помощью объекта конфигурации, который состоит из путей и соответствующих компонентов. Когда пользователь переходит по URL-адресу, маршрутизатор ищет соответствующий маршрут и загружает соответствующий компонент, который затем отображается на экране. Код для организации маршрутов в приложении представлен в листинге 3.7.

|  |
| --- |
| const routes: Routes = [  { path: '', component: HomepageComponent },  { path: 'auth', component: AuthComponent },  { path: 'profile/:username', component: ProfileComponent, canActivate: [AuthGuardService] },  { path: 'challenges', component: ChallengesComponent, canActivate: [AuthGuardService] },  { path: 'challenges/:id', component: ChallengeComponent, canActivate: [AuthGuardService] },  { path: 'discussions', component: DiscussionsComponent, canActivate: [AuthGuardService] },  { path: 'discussions/:id', component: ChatComponent, canActivate: [AuthGuardService] },  { path: 'leaderboard', component: StatisticComponent, canActivate: [AuthGuardService] },  { path: 'sandbox', component: SandboxComponent, canActivate: [AuthGuardService] },  { path: 'comments', component: CommentsComponent, canActivate: [AuthGuardService] }  ];  @NgModule({  imports: [RouterModule.forRoot(routes)],  exports: [RouterModule]  })  export class AppRoutingModule { } |

Листинг 3.7 – Код для организации маршрутизации

4 Тестирование приложения

Тестирование является неотъемлемой частью разработки любого программного продукта. Оно позволяет проверить работоспособность и соответствие приложения заданным требованиям. В данном приложении было проведено ручное тестирование.

Ручное тестирование заключается в том, чтобы проверить компоненты приложения в соответствии с их спецификацией и дизайном [7].

На формах входа и регистрации предусмотрены правила валидации, такие как ввод неверной пары логин-пароль и обязательные к заполнению поля.

При нарушениях данных правил пользователь увидит следующее уведомление, которое представлено на рисунке 4.1.

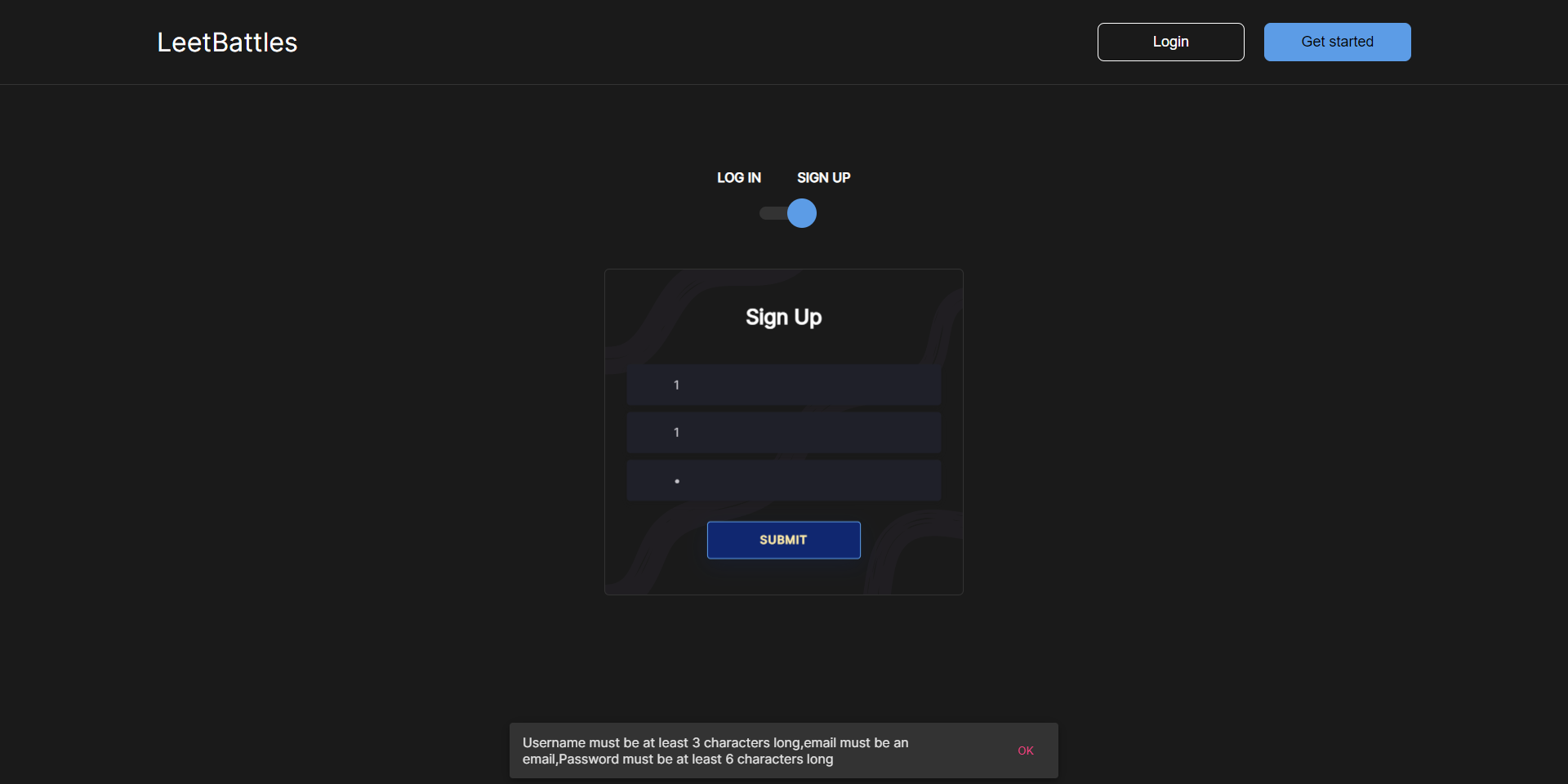


Рисунок 4.1 – Ошибочные данные в форме регистрации

Имя пользователя должно быть не менее трёх символов, электронная почта должна иметь формат электронной почты, а пароль должен быть не менее шести символов.

Также была совершена попытка зарегистрировать пользователя с уже имеющимся логином, результат этого теста представлен на рисунке 4.2.

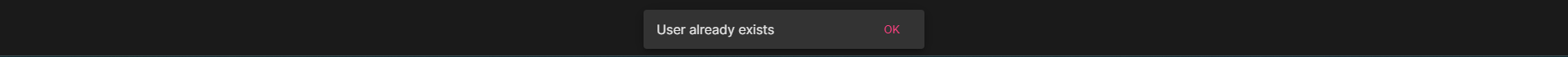


Рисунок 4.2 – Попытка зарегистрировать существующего пользователя

В данном случае нарушилось правило уникальности имени пользователя и его почты, что не позволило совершить регистрацию.

Для формы логина применяются схожие правила валидации, но попытка входа будет считать успешной, если пользователь прошел аутентификацию, если нет такого пользователя, будет отображено оповещение об этом.

Результат попытки входа под несуществующей электронной почты представлен на рисунке 4.3.



Рисунок 4.3 – Попытка входа под несуществующей электронной почтой

Результат попытки входа с правильной электронной почтой, но неправильным паролем представлен на рисунке 4.4.



Рисунок 4.4 – Попытка входа с неправильным паролем

Также важной частью является проверка формы отправки решения задачи. В компоненте задач предусмотрена обработка запроса на замену символов перевода каретки, перехода на новую строку и символа табуляции на пробел.

Проверка корректного запроса к базе данных представлена на рисунке 4.5.

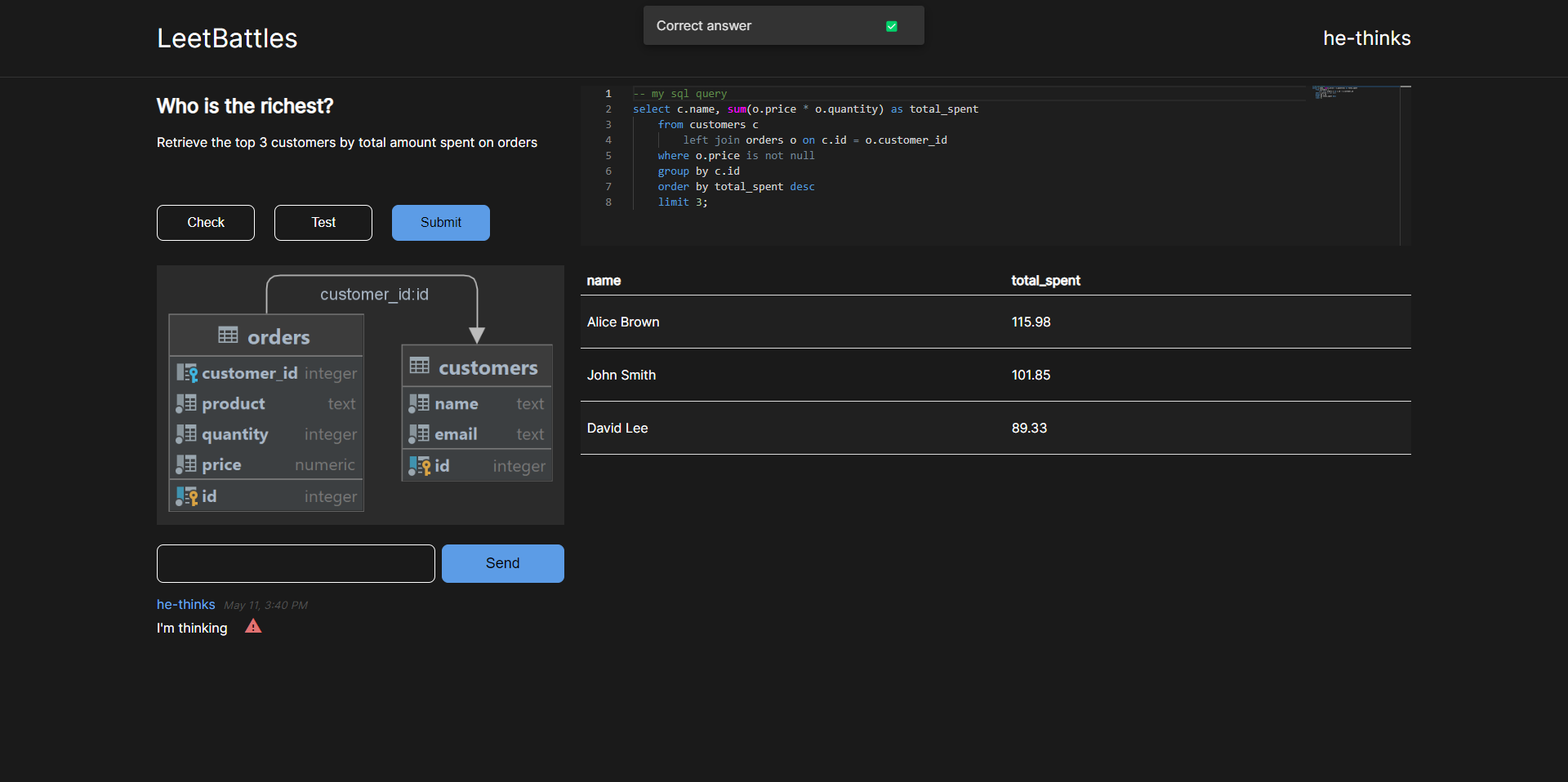


Рисунок 4.5 – Проверка корректного запроса к базе данных

В данном запросе используются специальные символы и комментарии, которые можно использовать в запросе, в результате чего выводится корректный результат. Результат проверки sql-запроса с ошибкой представлен на рисунке 4.6.

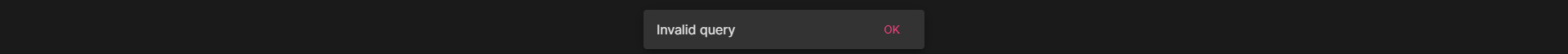


Рисунок 4.6 – Проверка sql-запроса с ошибкой

Также правила валидации присутствуют на форме для создания нового обсуждения.

Результат тестирования создания нового обсуждения представлен на рисунке 4.7.

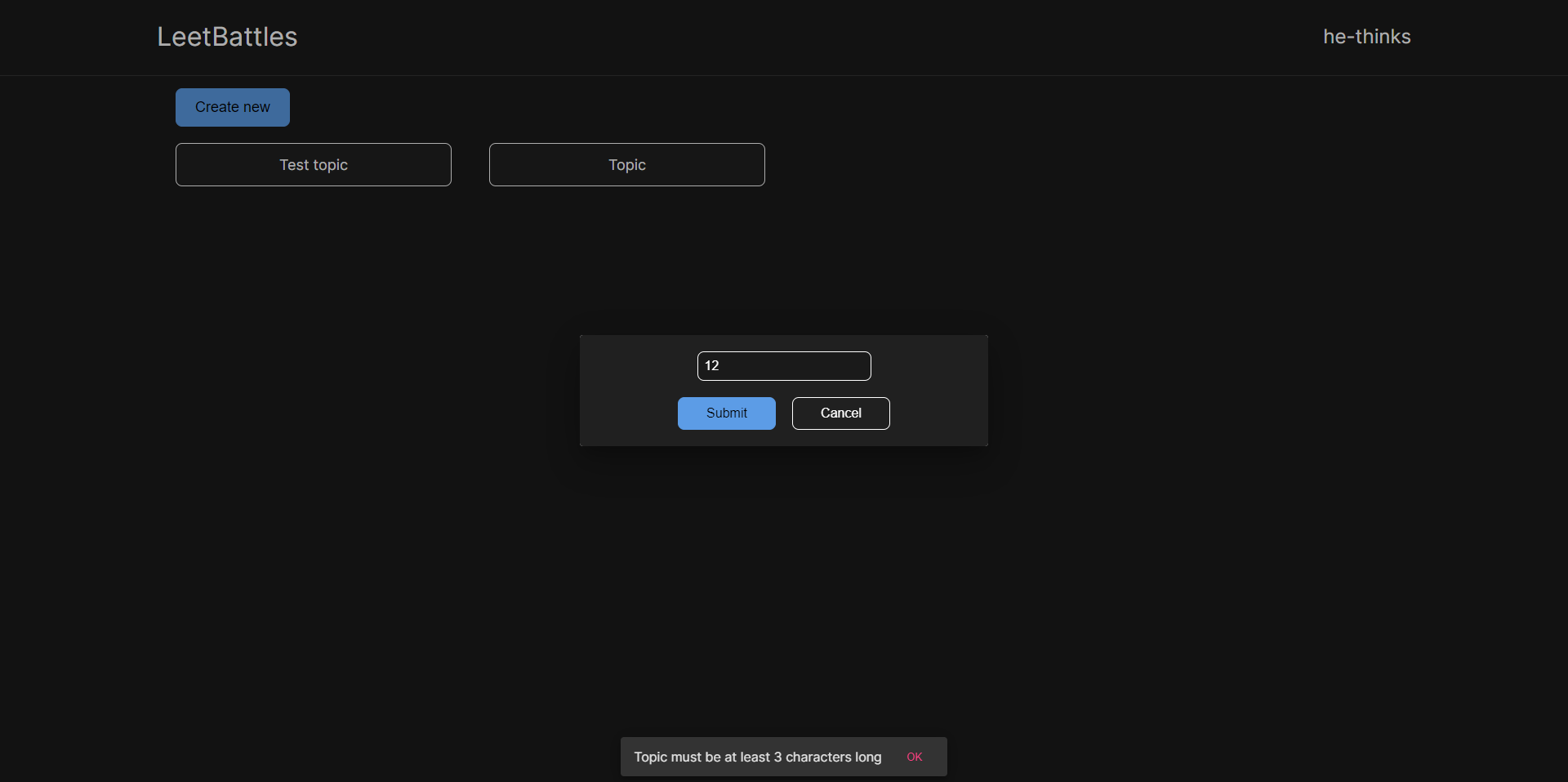


Рисунок 4.7 – Попытка создать обсуждение с коротким названием

Название обсуждения не может быть меньше трёх символов и не должно превышать количество в 20 символов. Проверка этого ограничения представлена на рисунке 4.8.



Рисунок 4.8 – Попытка превысить длину названия обсуждения

В данном разделе предоставлены результаты тестирования веб-приложение, разработанного для решения задач по sql.

Использовался ручной метод тестирования, были осуществлены проверки на ввод некорректных и неверных данных.

Полученные результаты при ручном тестировании показали, что программное средство работает правильно и в соответствии с требованиями.

5 Руководство пользователя

При входе пользователя на главную страницу, он может авторизоваться на сайте или зарегистрироваться. Главная страница приложения представлена на рисунке 5.1.

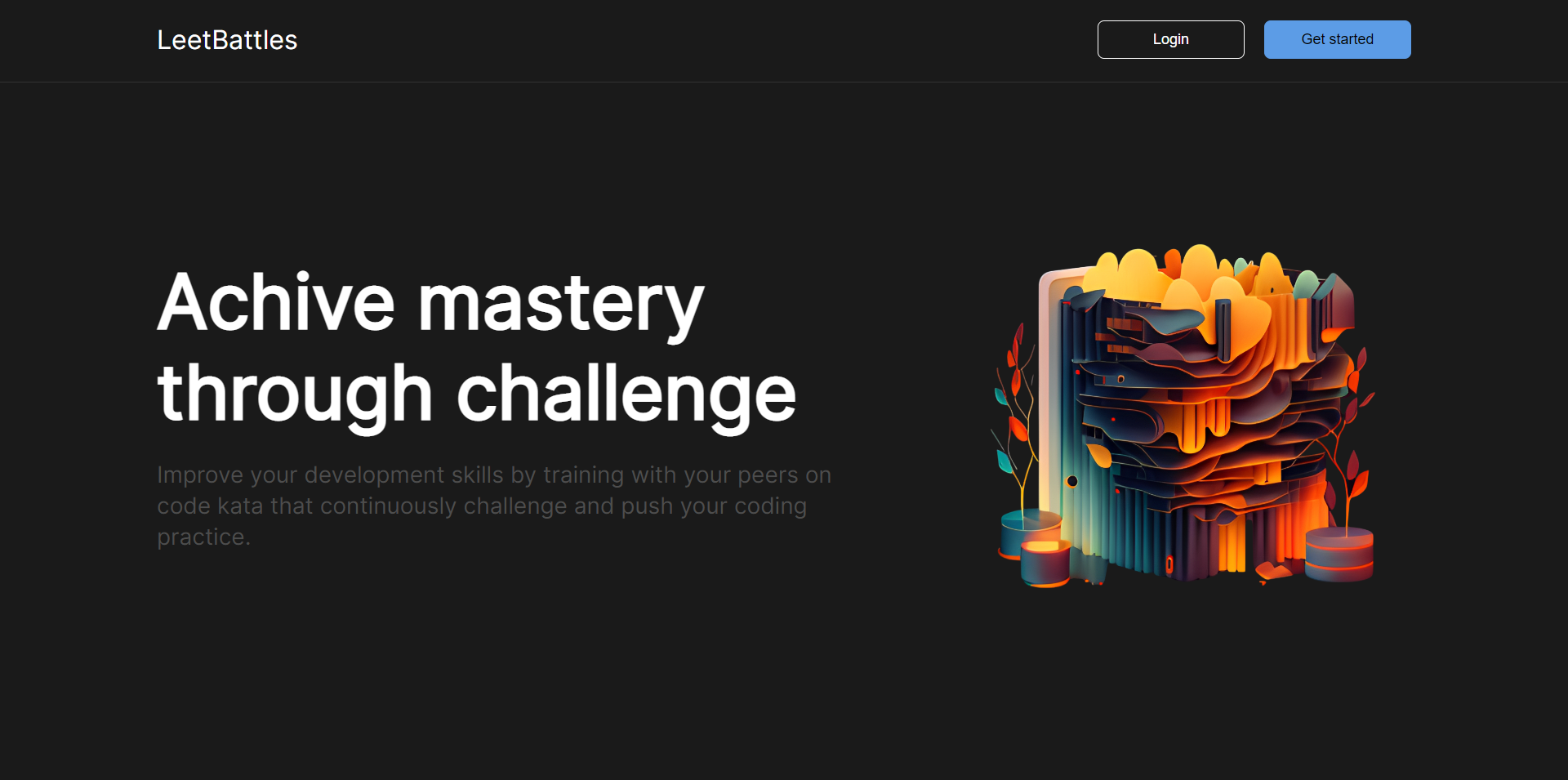


Рисунок 5.1 – Главная страница приложения

Чтобы перейти к формам авторизации или регистрации, необходимо нажать на соответствующие кнопки вверху экрана. Страница с авторизацией и регистрацией представлена на рисунке 5.2.

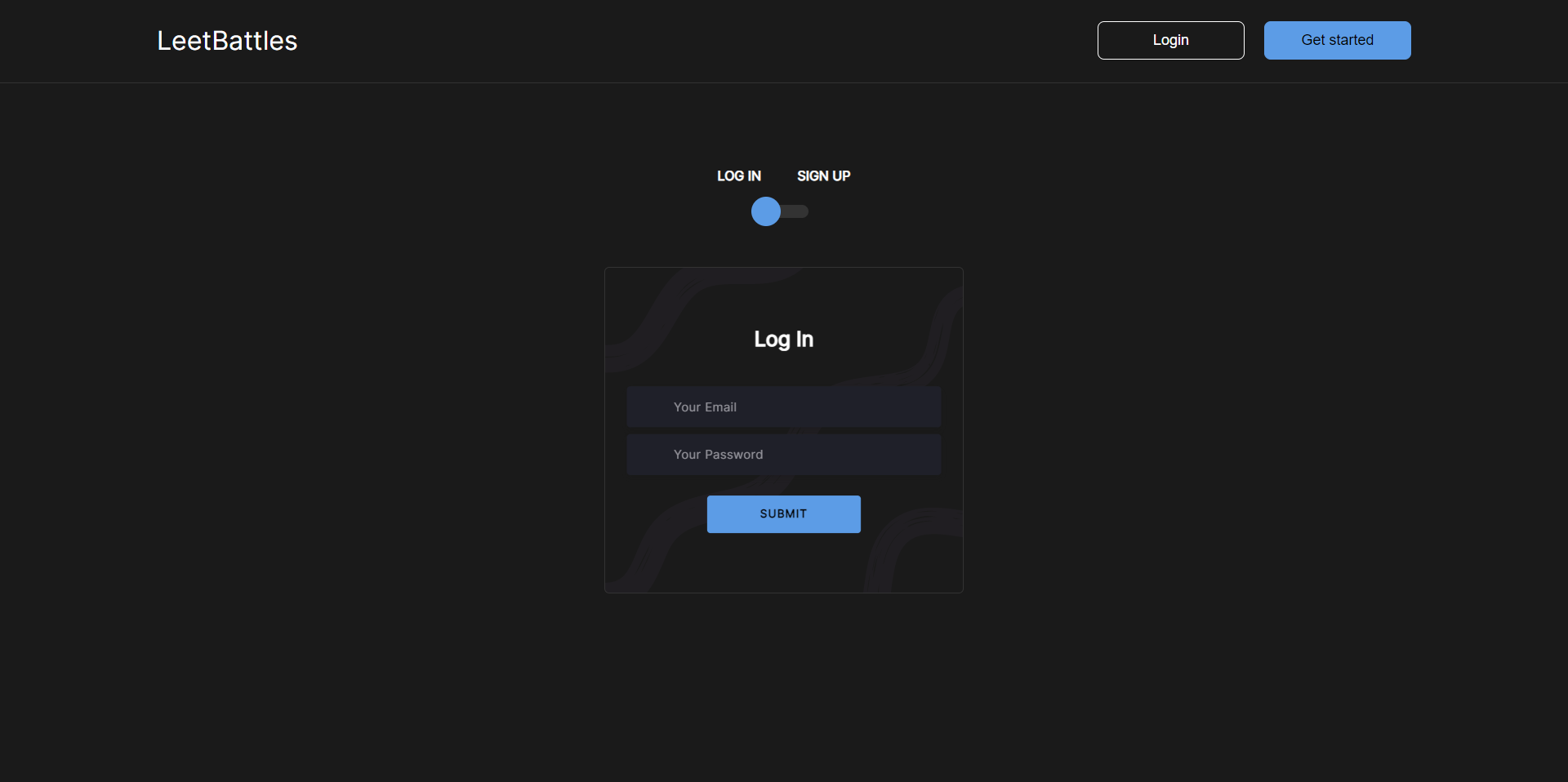


Рисунок 5.2 – Страница авторизации и регистрации

После входа в приложение, пользователь обладает одной из ролей, которая ему выдается при авторизации: пользователь или администратор. При входе пользователя отображается страница с его профилем, которая представлена на рисунке 5.3.

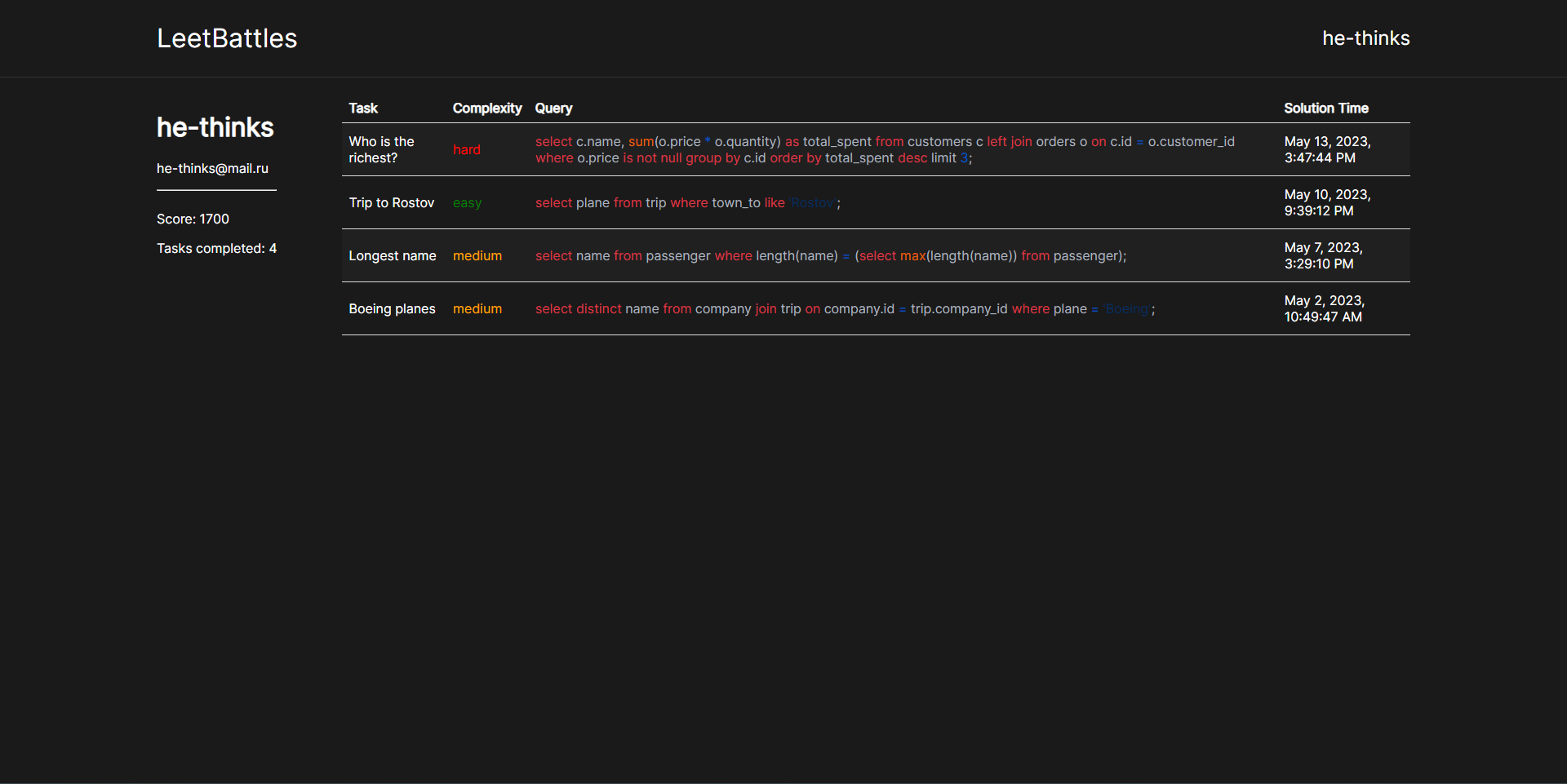


Рисунок 5.3 – Страница профиля пользователя

На странице профиля, пользователю доступен просмотр личной статистики и его решений задач. При наведении на имя пользователя отображается основная навигация приложения, представленная на рисунке 5.4.

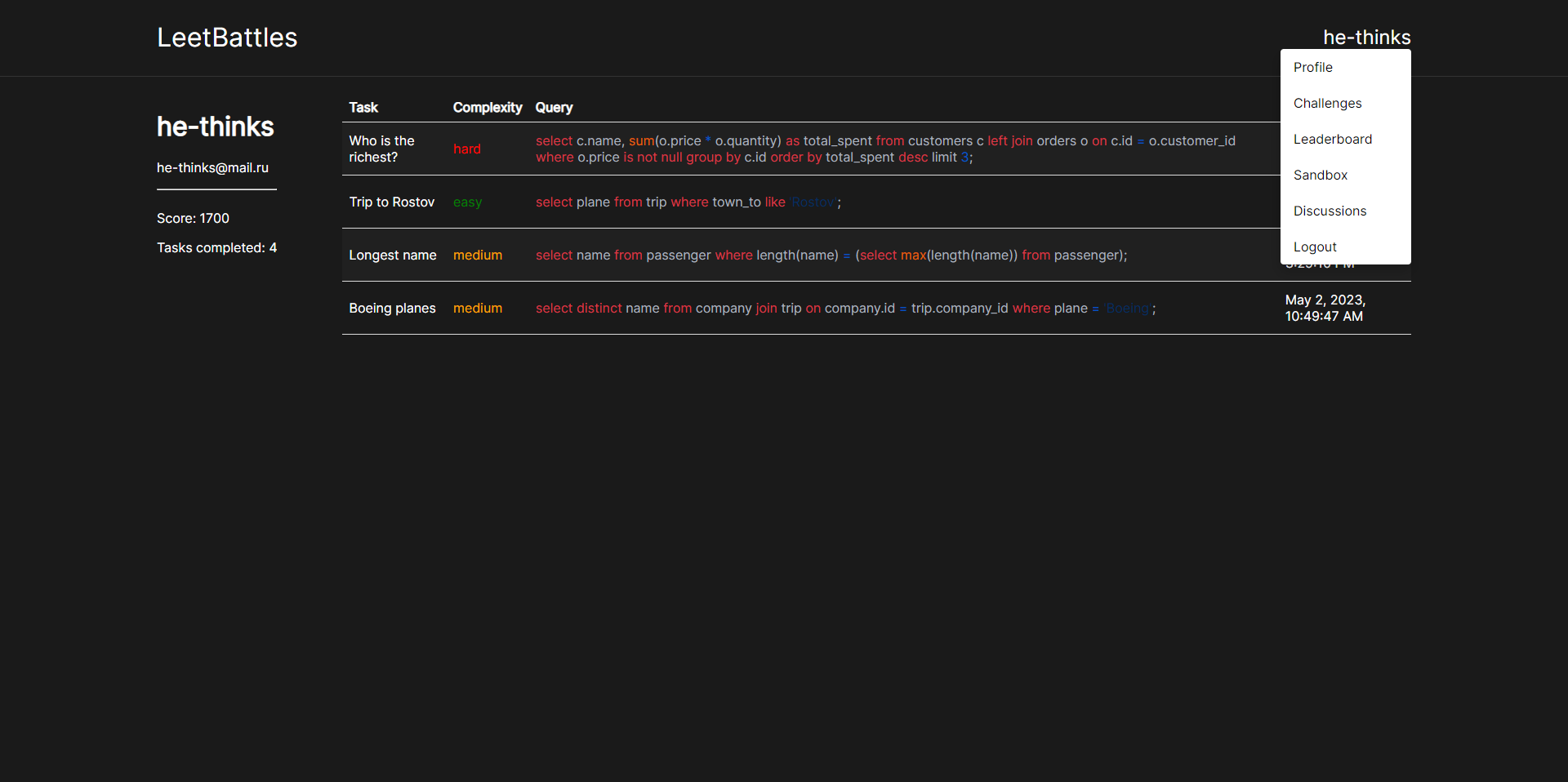


Рисунок 5.4 – Навигация приложения

В навигации пользователю доступны переходы на страницы профиля, списка задач, таблицы лидеров, создание задачи и обсуждений. При переходе на страницу с задачами пользователь видит список задач и возможности их сортировки. Данная страница представлена на рисунке 5.5.

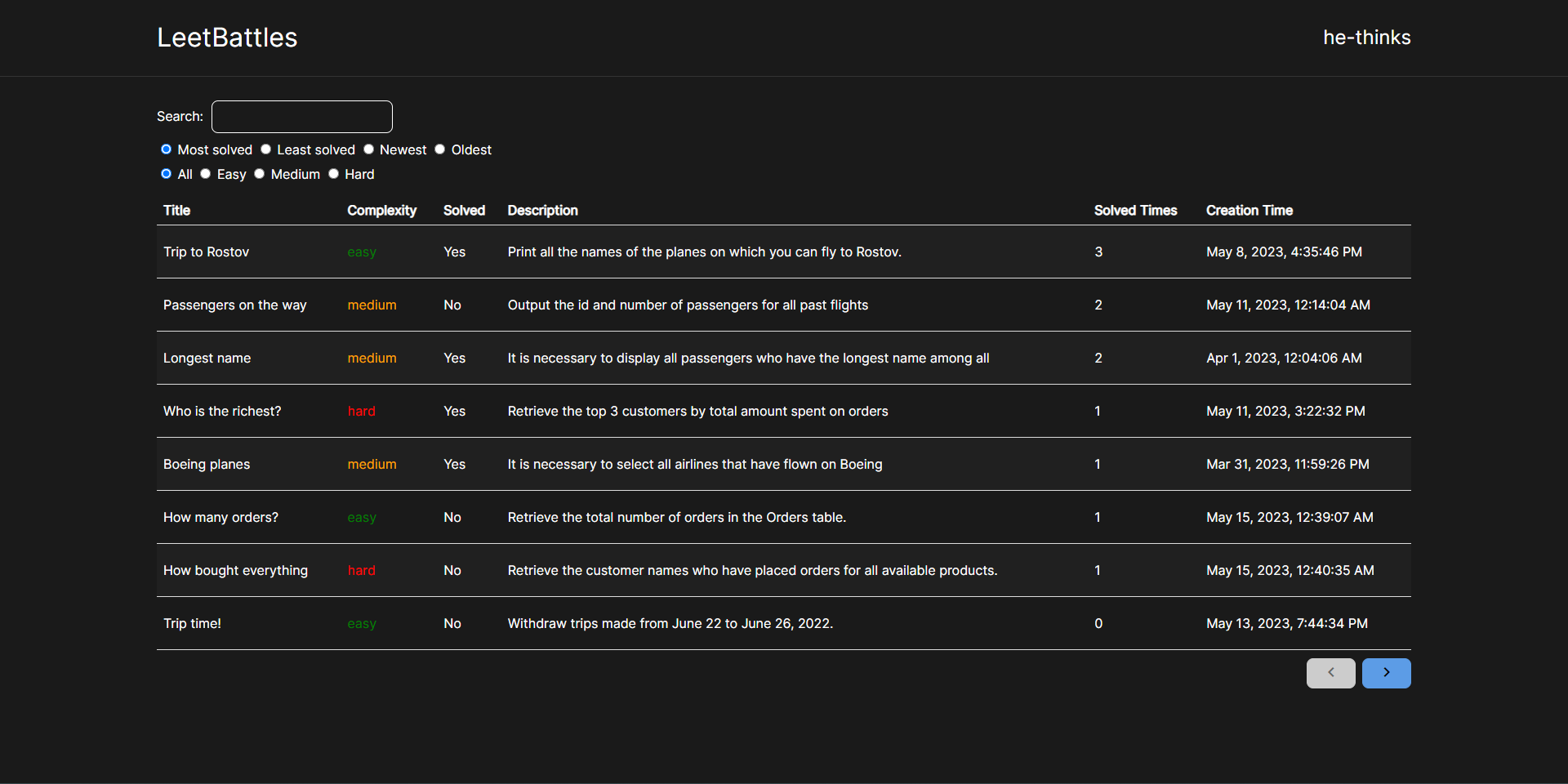


Рисунок 5.5 – Страница списка задач

При нажатии на любую задачу выполнится переход на страницу, содержащую название, описание, схему, комментарии, редактор кода и таблица с решением задачи или результатом запроса. Данная страница представлена на рисунке 5.6.

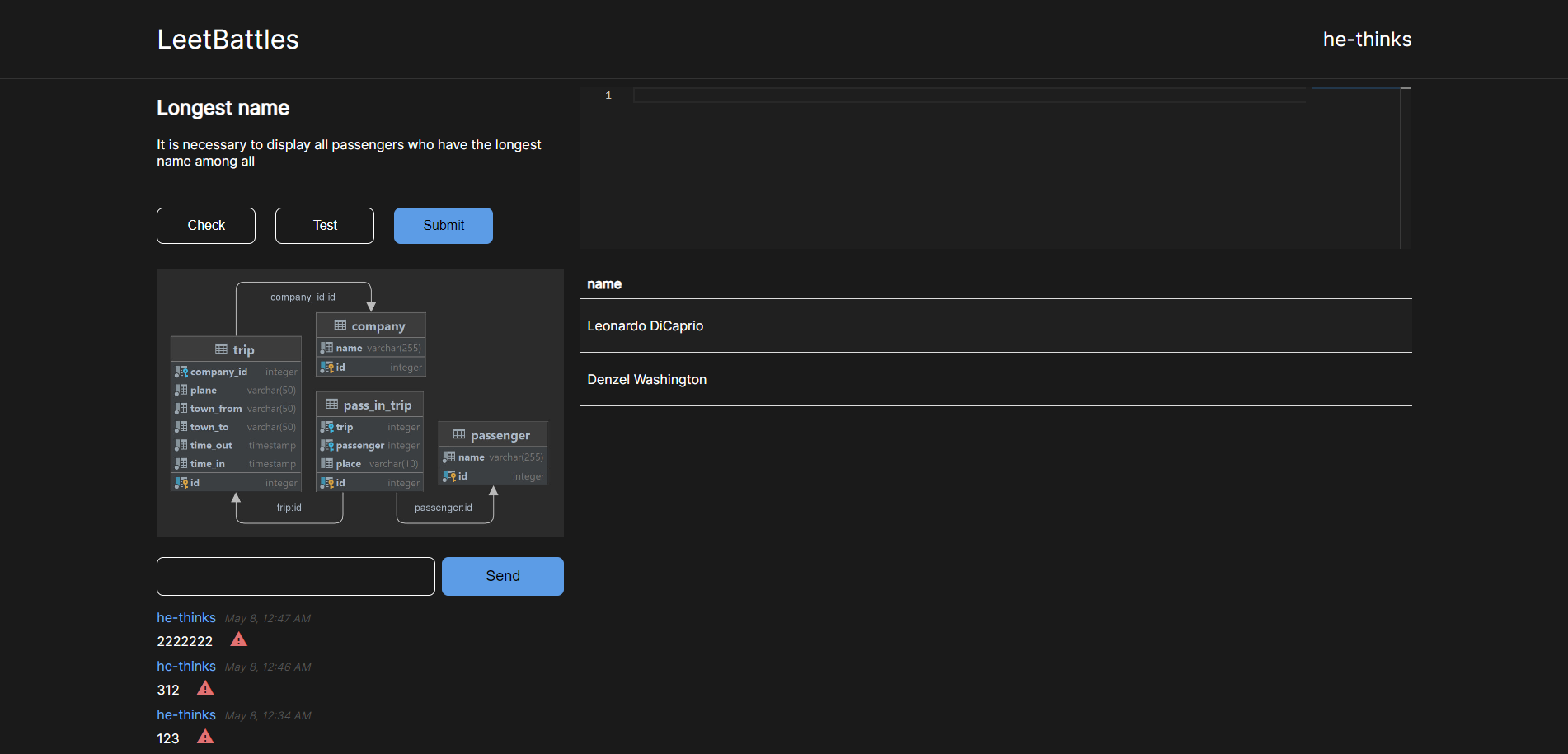


Рисунок 5.6 – Страница задачи

На странице присутствуют кнопки, которые отвечают за различные действия.

Кнопка check отображает в таблице результирующий набор, который представляет из себя решение задачи. Кнопка test отправляет запрос на сервер и возвращает его результирующий набор, который выводится в таблице. Кнопка submit отправляет получившийся результирующий набор запроса и сравнивается с ответом к задаче. Если ответ правильный, то отобразится соответствующее уведомление, которое представлено на рисунке 5.7.

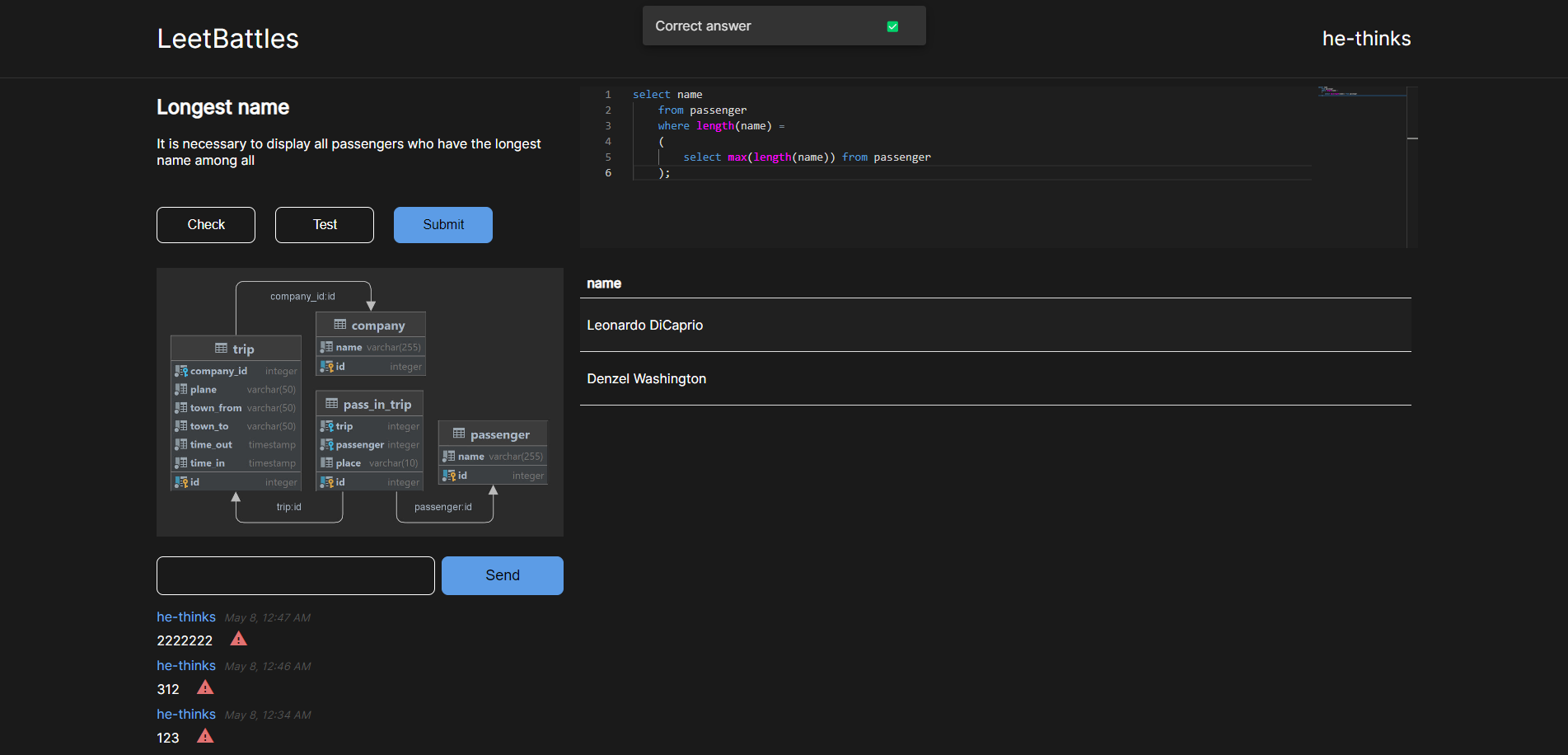


Рисунок 5.7 – Корректное решение задачи

При переходе на страницу лидеров, пользователь увидит таблицу, содержащую первые 10 пользователей с максимальным количеством очков, данная страница представлена на рисунке 5.8.

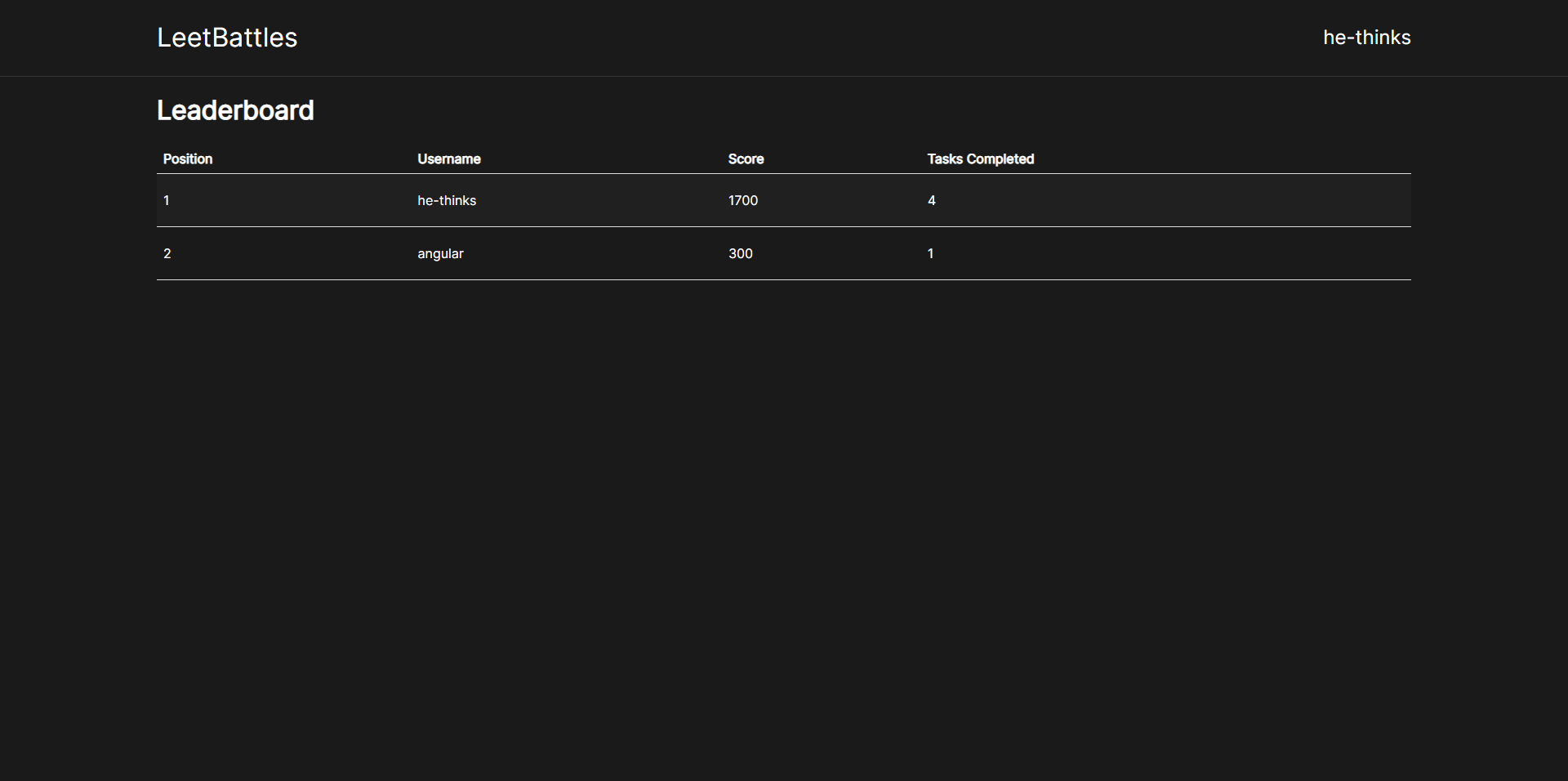


Рисунок 5.8 – Страница лидеров

При переходе на создание задачи, пользователю отобразится страница, представленная на рисунке 5.9.

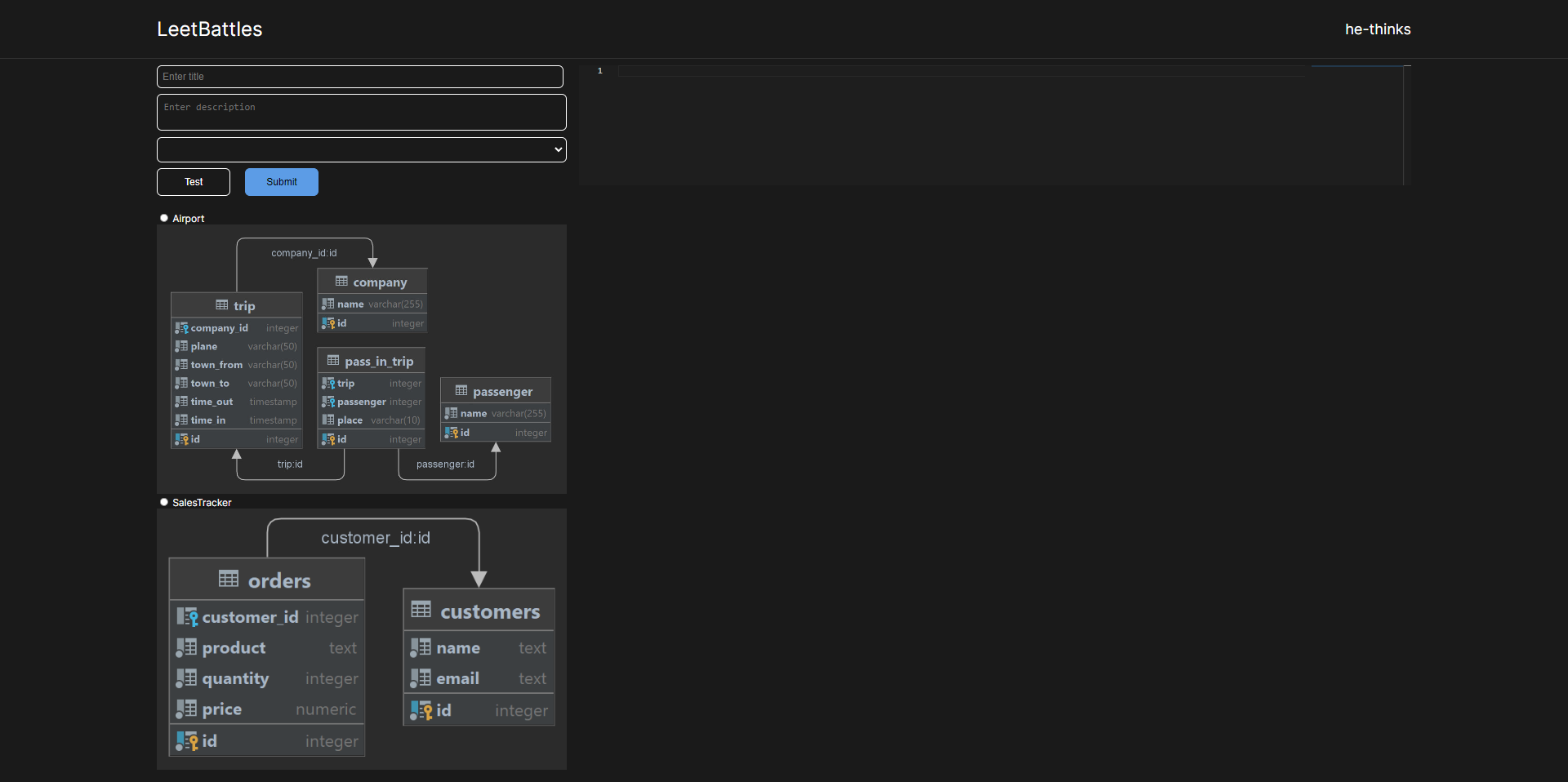


Рисунок 5.9 – Страница создания задачи

На этой странице пользователь выбирает название, описание, схему таблиц, сложность и результирующий набор запроса, определяющий её ответ. Заполненная форма создании задачи представлена на рисунке 5.10.

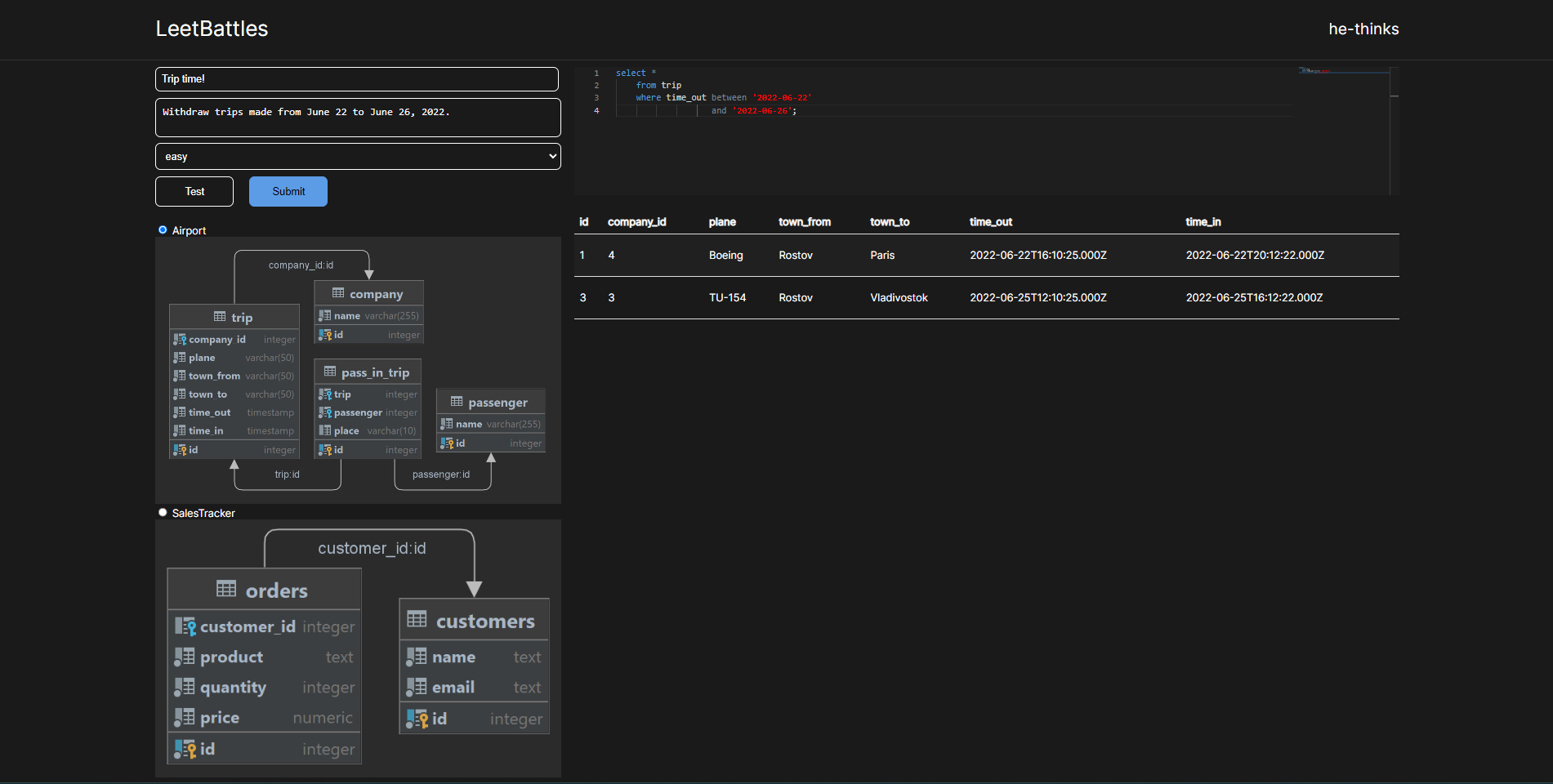


Рисунок 5.10 – Заполненная форма создания задачи

Если все поля заполнены верно, то при нажатии кнопки submit задача отправится на рассмотрение администратору.

При переходе на страницу обсуждений, пользователю будет доступно создание нового обсуждения, или участие в уже существующем. Страница с обсуждениями представлена на рисунке 5.11.

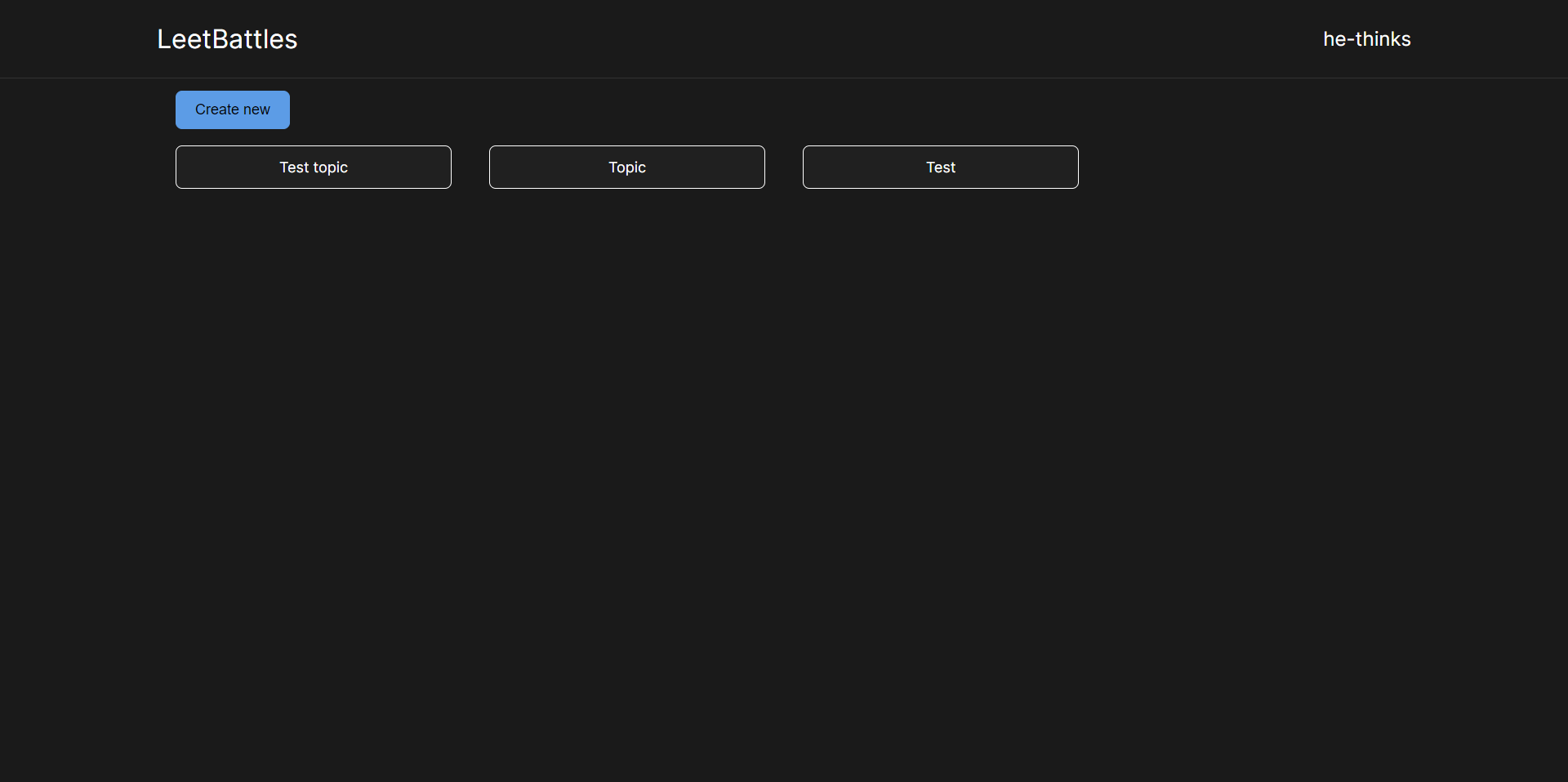


Рисунок 5.11 – Страница со списком обсуждений

При нажатии на обсуждение, пользователь попадает в чат, в котором он будет отправлять и получать сообщения в режиме реального времени. Страница чата представлена на рисунке 5.12.

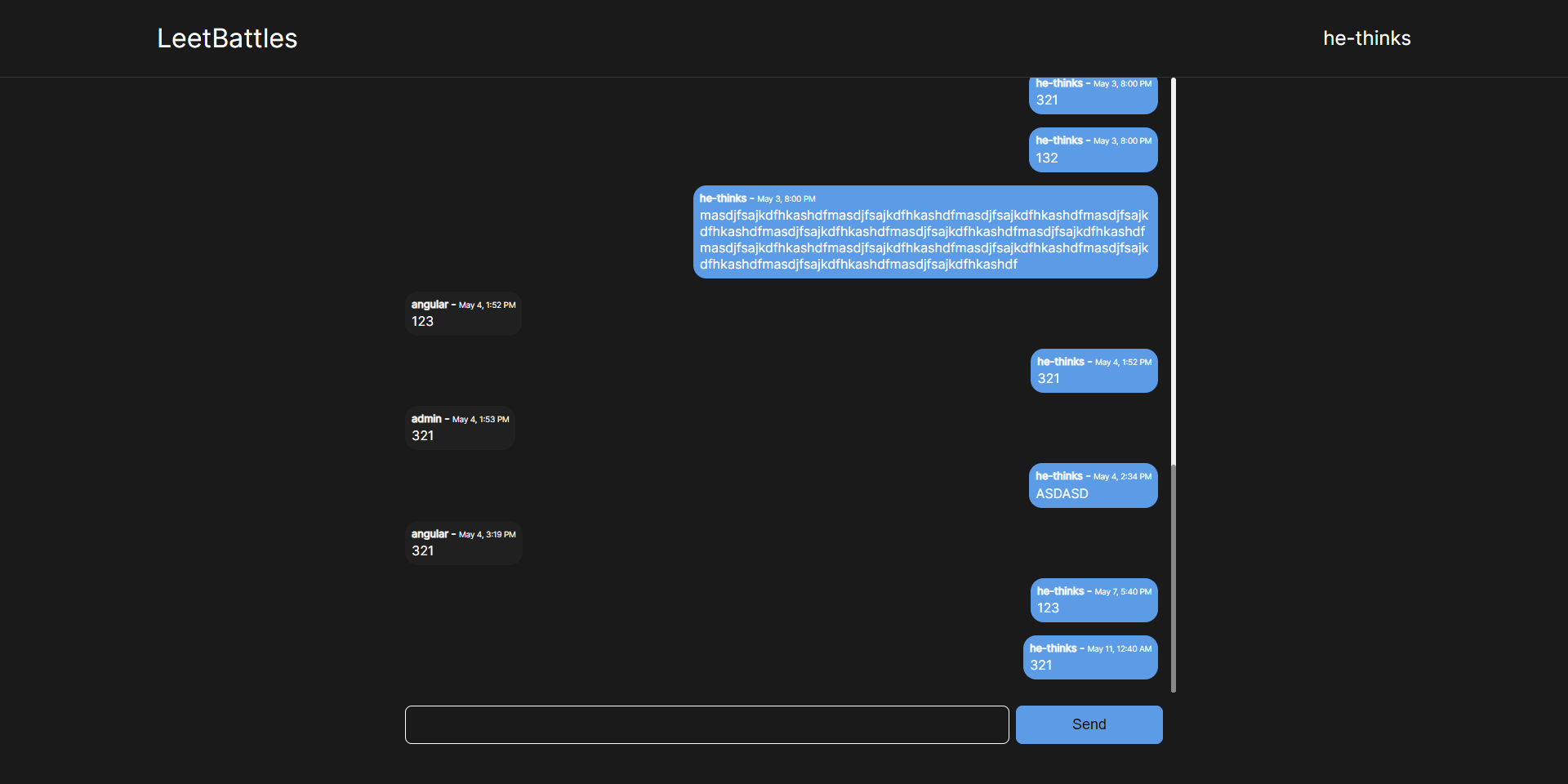


Рисунок 5.12 – Страница чата приложения

При входе под учётной записью с ролью администратор, выводится список предложенных задач пользователями.

Страница с предложенными задачами представлена на рисунке 5.13.

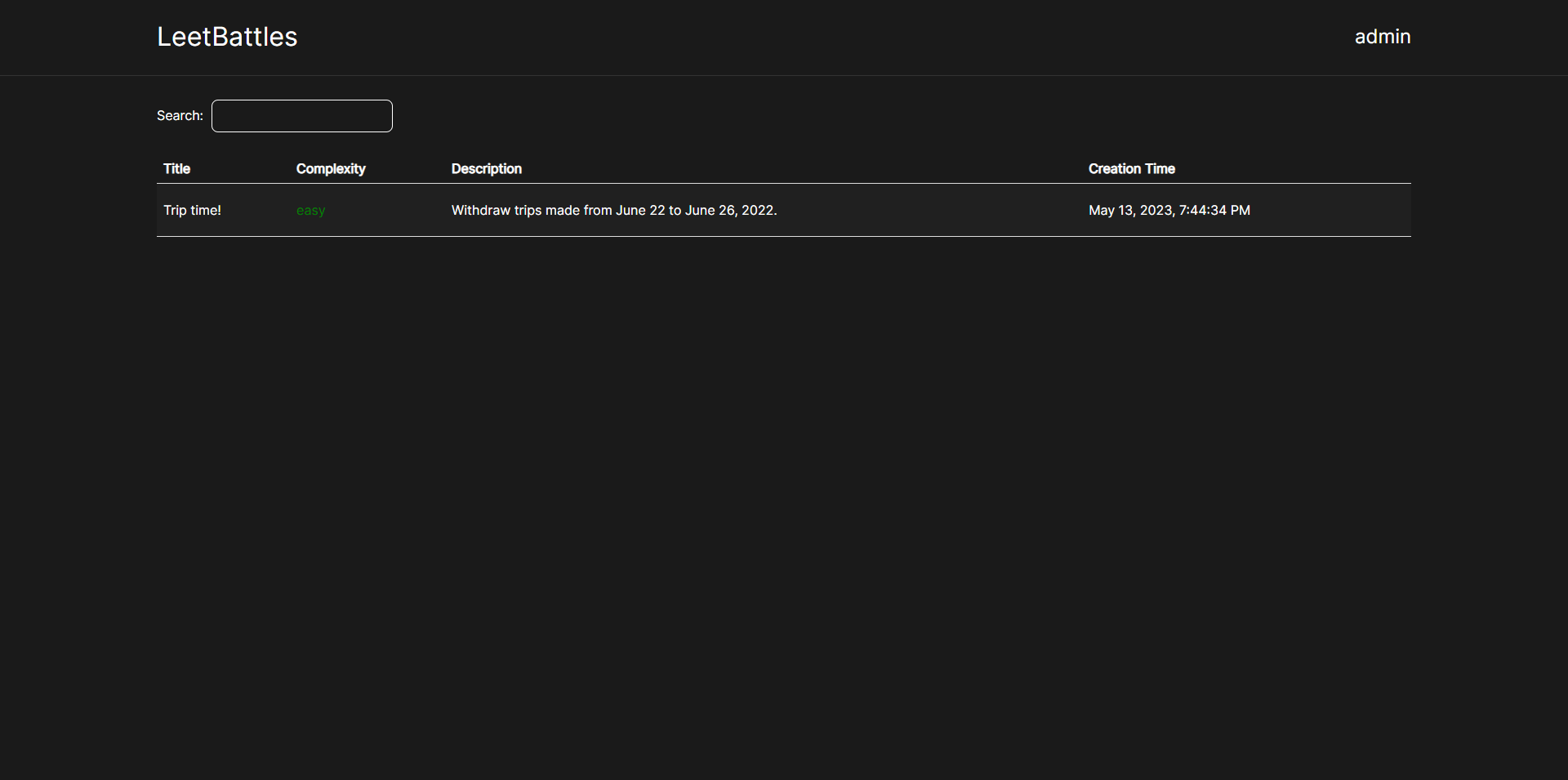


Рисунок 5.13 – Страница с предложенными задачами

При нажатии на задачу, администратор перейдет на страницу с принятием или отклонением задачи, которая представлена на рисунке 5.14.

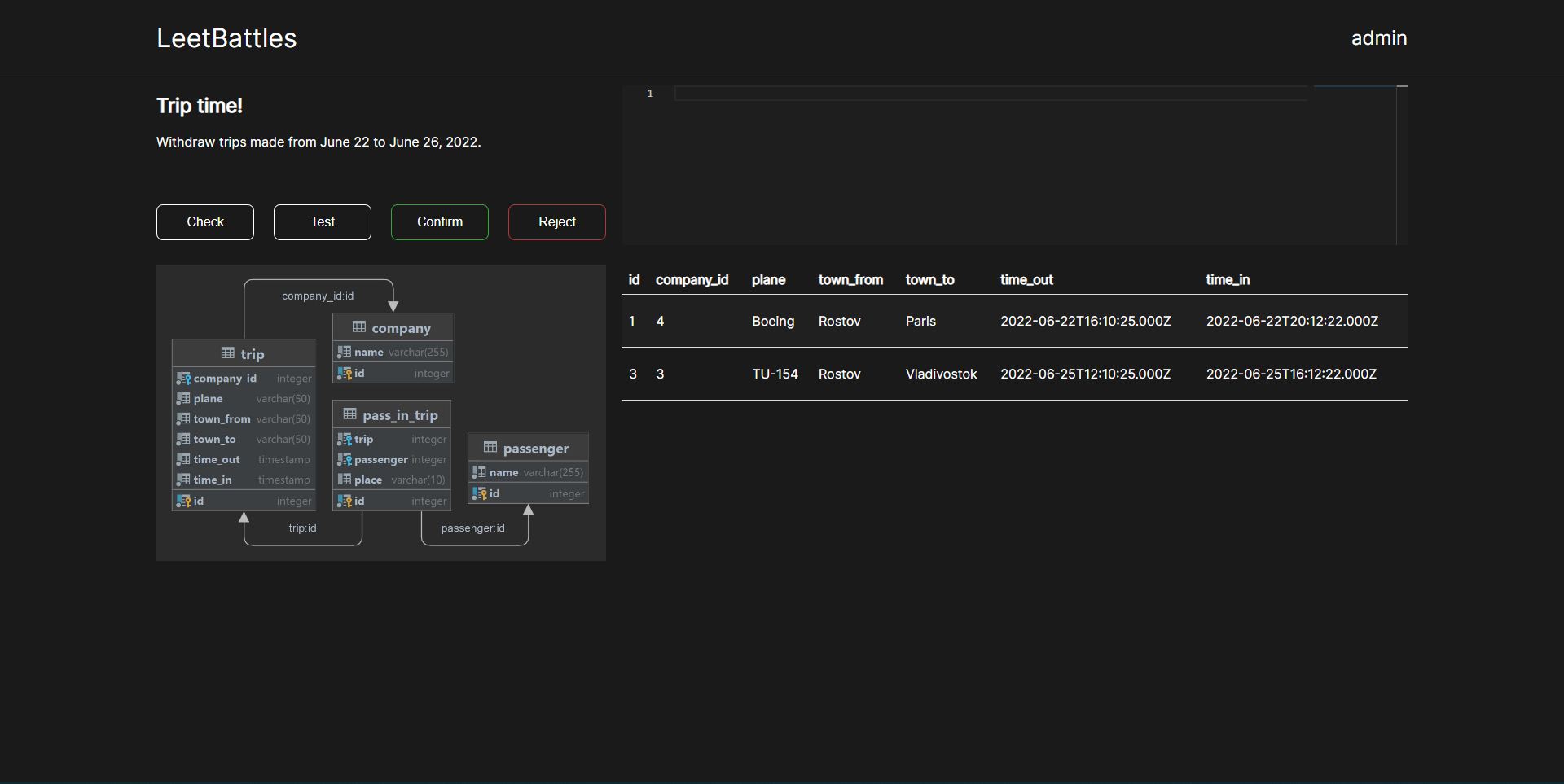


Рисунок 5.14 – Страница с принятием или отклонением задачи

На данной странице выводится информация о задаче и возможность её решения. Также доступны дополнительные кнопки, которые принимают или отклоняют задачу. После нажатия одной из них, администратор переходит на предыдущую страницу.

Также администратор может просматривать комментарии, которые были отмечены жалобами. Данная страница представлена на рисунке 5.15.

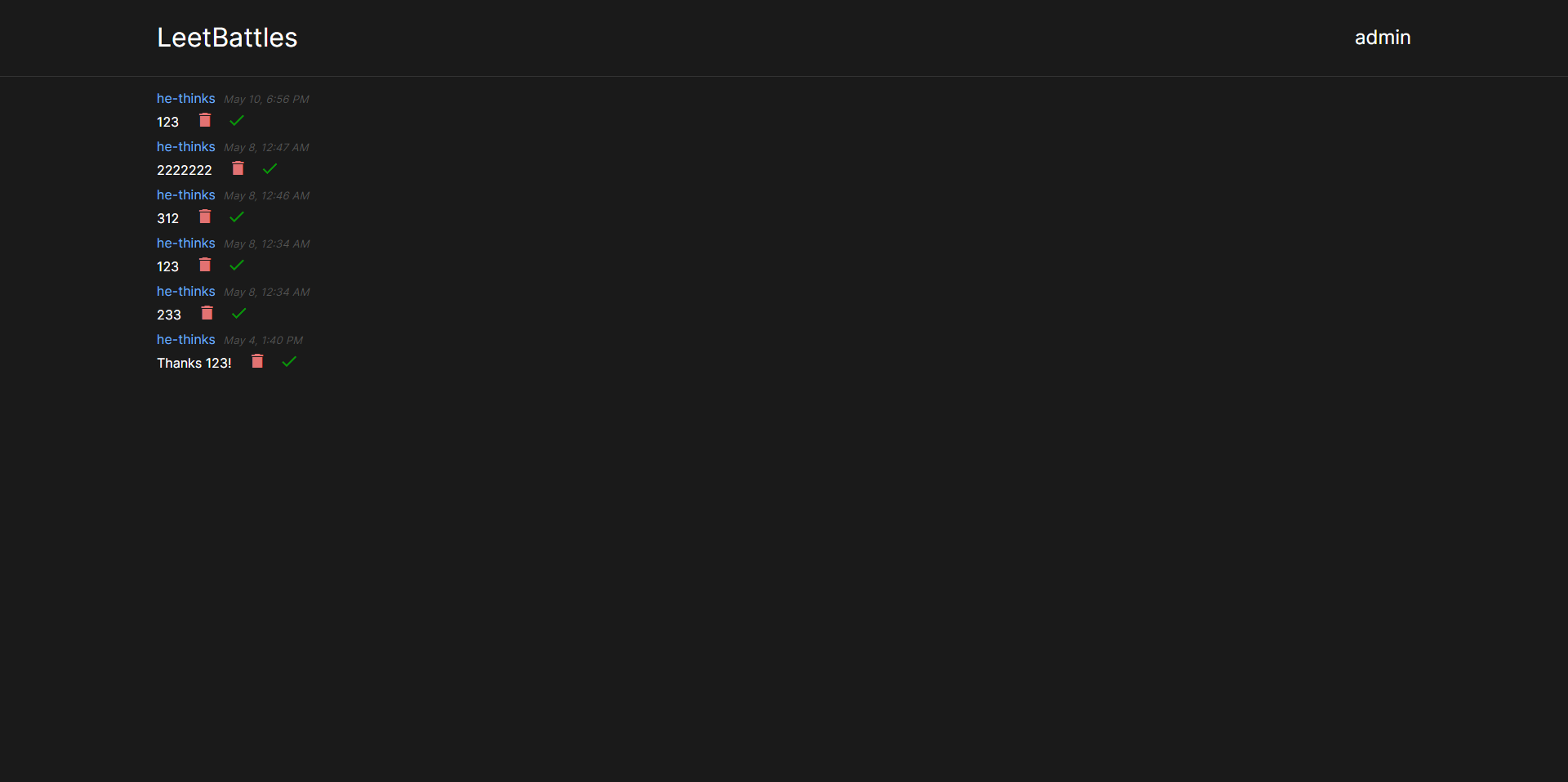


Рисунок 5.15 – Страница комментариями с жалобой

На данной странице администратор может удалить или сохранить комментарий, после чего отметка о жалобе пропадёт.

Также администратору добавляется кнопка удаления на странице обсуждений, которая представлена на рисунке 5.16.

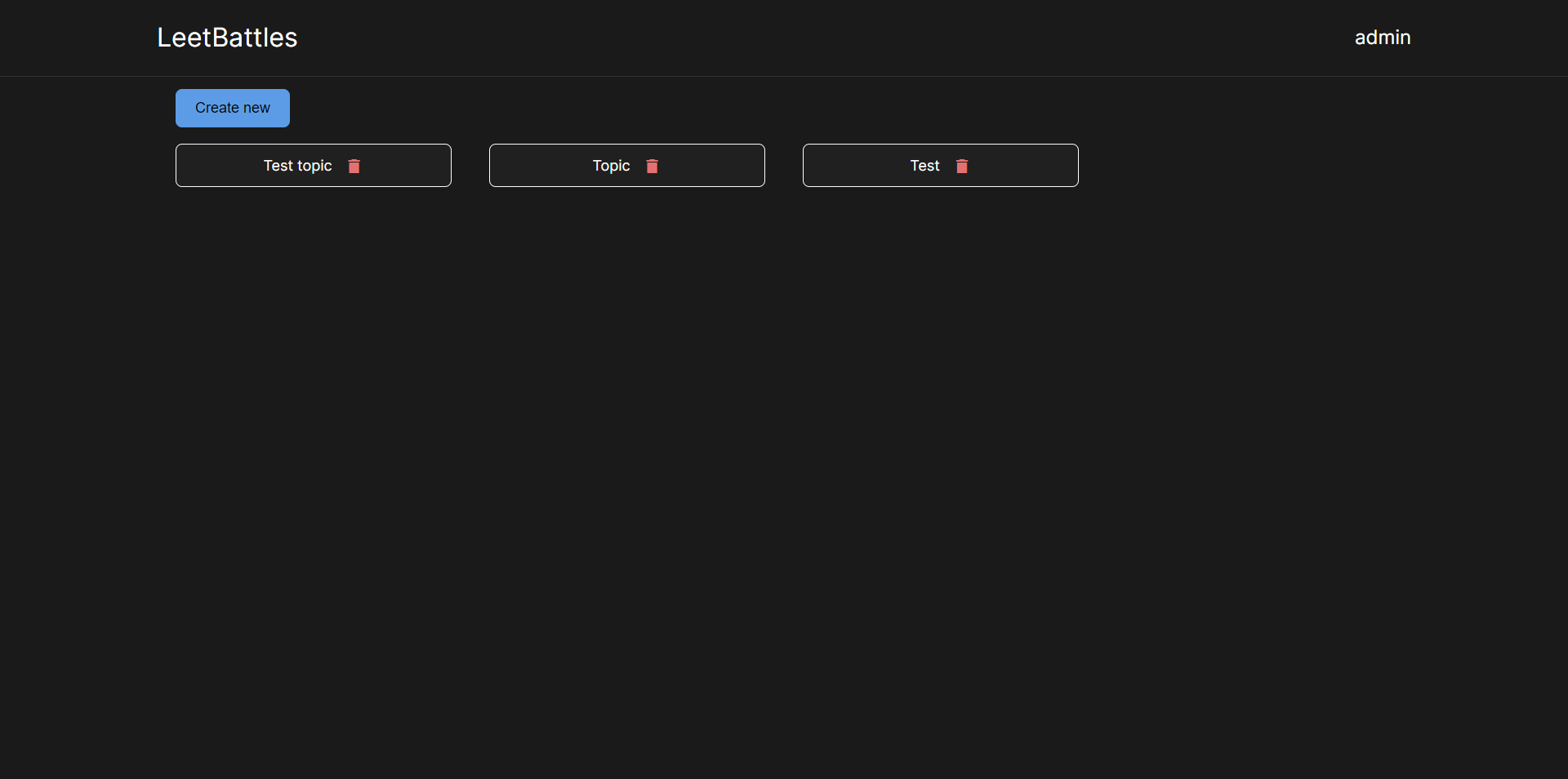


Рисунок 5.16 – Страница списка обсуждений

При нажатии на корзину обсуждение удалится из базы данных.

Заключение

В результате выполнения курсового проекта было разработано веб-приложение для решения задач по SQL. В рамках работы над проектом был проведён обзор аналогичных решений, выбрана платформа для разработки серверной и клиентской частей приложения, спроектирована архитектура и структуры баз данных, проведено тестирование приложения.

В пояснительной записке приведено руководство пользователя для веб-приложения. Веб-приложение позволяет всем пользователям регистрироваться и авторизоваться в системе. Для пользователей предусмотрен функционал для обучения использования select-запросов, общения с другими пользователями, просмотра таблицы лидеров и оставления комментариев под задачами. Администратор имеет возможности подтверждения расширения списка задач и контроля за нежелательным контентом от пользователей.

В качестве основы для разработки серверной части был выбран фреймворк Nest, который обладает возможностью повторного использования модулей для разработки других проектов. Это позволит использовать полученный опыт в будущих работах и сократить время на разработку новых приложений.

В качестве основы для разработки клиентской части был выбран фреймворк Angular. Он был выбран из-за своей высокой производительности, расширяемости и большого сообщества разработчиков, что обеспечивает быстрое решение возникающих проблем и возможность получения дополнительной поддержки в разработке.

Разработка клиентской части приложения включала в себя создание пользовательского интерфейса, взаимодействие с сервером и обработку данных. С помощью Angular было создано удобное и интуитивно понятное пользовательское взаимодействие, что позволило значительно упростить работу с приложением и повысить его функциональность.

Программа является законченным программным продуктом, хотя возможна доработка. Пользование данной программой не составит труда для пользователя любого возраста, так как программное средство имеет удобный и понятный интерфейс. Данное приложение соответствует поставленной задаче и отвечает всем требованиям, необходимым для пользования данным программным средством.

Список использованных источников

1. TypeScript [Электронный ресурс] – Режим доступа: https://ru.wikipedia.org/wiki/TypeScript – Дата доступа: 09.04.2023.
2. Prisma Get Started [Электронный ресурс] – Режим доступа: https://www.prisma.io/docs/getting-started – Дата доступа: 11.05.2023.
3. NestJS Documentation [Электронный ресурс] – Режим доступа: https://docs.nestjs.com/ – Дата доступа: 12.05.2023.
4. Angular Documentation [Электронный ресурс] – Режим доступа: https://angular.io/docs – Дата доступа: 12.05.2023.
5. Запросы в PostgreSQL [Электронный ресурс] – Режим доступа: https://habr.com/ru/companies/postgrespro/articles/574702/ – Дата доступа: 11.05.2023.
6. NestJS Gateways [Электронный ресурс] – Режим доступа: https://docs.nestjs.com/websockets/gateways – Дата доступа: 09.11.2023.
7. Ручное тестирование программного обеспечения [Электронный ресурс] – Режим доступа: http://getbug.ru/ruchnoe-testirovanie-programmnogo-obespecheniya/ – Дата доступа: 13.05.2023.

Приложение А

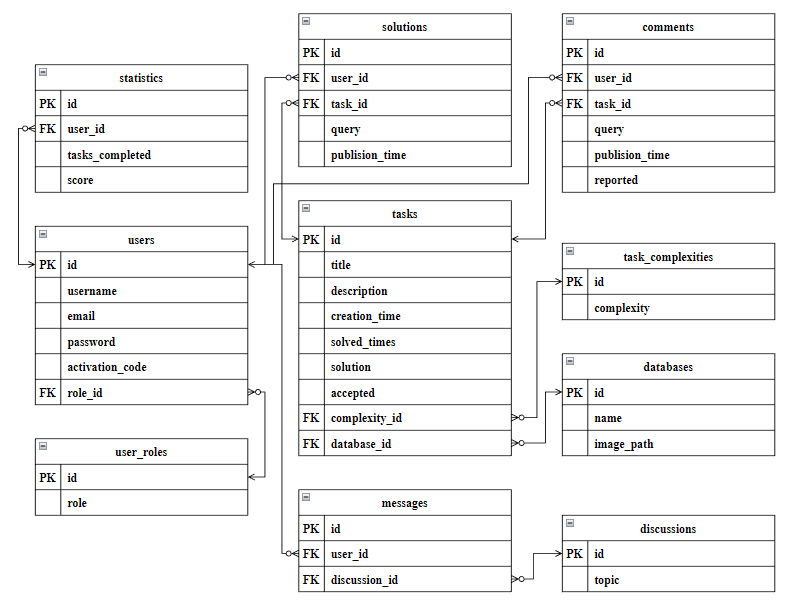


Рисунок – Логическая схема базы данных

Приложение Б

|  |
| --- |
| generator client {  provider = "prisma-client-js"  }  datasource db {  provider = "postgresql"  url = env("DATABASE\_URL")  }  model comments {  id Int @id @default(autoincrement())  content String  creation\_time DateTime @default(dbgenerated("timezone('utc-3'::text, CURRENT\_TIMESTAMP(0))")) @db.Timestamp(6)  user\_id Int?  task\_id Int?  reported Boolean @default(false)  tasks tasks? @relation(fields: [task\_id], references: [id], onDelete: NoAction, onUpdate: NoAction)  users users? @relation(fields: [user\_id], references: [id], onDelete: NoAction, onUpdate: NoAction)  }  model discussions {  id Int @id @default(autoincrement())  topic String @db.VarChar(50)  messages messages[]  }  model messages {  id Int @id @default(autoincrement())  content String  creation\_time DateTime @default(dbgenerated("timezone('utc-3'::text, CURRENT\_TIMESTAMP(0))")) @db.Timestamp(6)  discussion\_id Int?  user\_id Int?  discussions discussions? @relation(fields: [discussion\_id], references: [id], onDelete: NoAction, onUpdate: NoAction)  users users? @relation(fields: [user\_id], references: [id], onDelete: NoAction, onUpdate: NoAction)  }  model solutions {  id Int @id @default(autoincrement())  query String  solution\_time DateTime @default(dbgenerated("timezone('utc-3'::text, CURRENT\_TIMESTAMP(0))")) @db.Timestamp(6)  task\_id Int?  user\_id Int?  tasks tasks? @relation(fields: [task\_id], references: [id], onDelete: NoAction, onUpdate: NoAction)  users users? @relation(fields: [user\_id], references: [id], onDelete: NoAction, onUpdate: NoAction)  }  model statistics {  id Int @id @default(autoincrement())  tasks\_completed Int @default(0)  score Int @default(0)  user\_id Int?  users users? @relation(fields: [user\_id], references: [id], onDelete: NoAction, onUpdate: NoAction)  }  model task\_complexities {  id Int @id @default(autoincrement())  name String @db.VarChar(50)  tasks tasks[]  }  model tasks {  id Int @id @default(autoincrement())  title String @db.VarChar(50)  description String  creation\_time DateTime @default(dbgenerated("timezone('utc-3'::text, CURRENT\_TIMESTAMP(0))")) @db.Timestamp(6)  solved\_times Int @default(0)  solution Json @db.Json  accepted Boolean @default(false)  complexity\_id Int?  database\_id Int?  comments comments[]  solutions solutions[]  task\_complexities task\_complexities? @relation(fields: [complexity\_id], references: [id], onDelete: NoAction, onUpdate: NoAction)  databases databases? @relation(fields: [database\_id], references: [id], onDelete: NoAction, onUpdate: NoAction)  }  model user\_roles {  id Int @id @default(autoincrement())  name String @db.VarChar(50)  users users[]  }  model users {  id Int @id @default(autoincrement())  username String @unique @db.VarChar(50)  email String @unique @db.VarChar(50)  password String @db.VarChar(300)  activation\_code String?  role\_id Int? @default(2)  comments comments[]  messages messages[]  solutions solutions[]  statistics statistics[]  user\_roles user\_roles? @relation(fields: [role\_id], references: [id], onDelete: NoAction, onUpdate: NoAction)  }  model databases {  id Int @id @default(autoincrement())  name String @db.VarChar(50)  image\_path String @db.VarChar(100)  tasks tasks[]  } |

Листинг – Схема Prisma

Приложение В

|  |
| --- |
| import { AfterViewInit, Component, ElementRef, OnInit, ViewChild } from '@angular/core';  import { ActivatedRoute, Router } from '@angular/router';  import { Comment } from 'src/app/models/comment.model';  import { Task } from 'src/app/models/task.model';  import { CommentService } from 'src/app/services/comment.service';  import { NotificationService } from 'src/app/services/notification.service';  import { SchemaService } from 'src/app/services/schema.service';  import { TaskService } from 'src/app/services/task.service';  import { TokenStorageService } from 'src/app/services/token-storage.service';  @Component({  selector: 'app-challenge',  templateUrl: './challenge.component.html',  styleUrls: ['./challenge.component.css']  })  export class ChallengeComponent implements OnInit {  editorOptions = { theme: 'vs-dark', language: 'pgsql' };  public task: Task | undefined;  public comments: Comment[] = [];  public isTaskLoaded: boolean = false;  public sqlCode: string = '';  public columns: string[] = [];  public data: any[] = [];  public imageURL: string = '';  public content: string = '';  public isAdmin: boolean = false;  constructor(  private taskService: TaskService,  private route: ActivatedRoute,  private notificationService: NotificationService,  private commentService: CommentService,  private schemaService: SchemaService,  private tokenStorage: TokenStorageService,  private router: Router  ) { }  ngOnInit(): void {  this.isAdmin = this.tokenStorage.isAdmin();  this.route.params.subscribe(params => {  this.taskService.getTask(params['id']).subscribe({  next: data => {  this.task = data;  if (this.task?.solution) {  this.columns = Object.keys(this.task.solution[0]);  this.data = this.task.solution;  }  this.isTaskLoaded = true;  },  error: err => {  console.log(err);  }  });  this.schemaService.getSchema(params['id']).subscribe({  next: data => {  this.imageURL = URL.createObjectURL(data);  }  });  if (!this.isAdmin) {  this.commentService.getCommentsByTaskId(params['id']).subscribe({  next: data => {  this.comments = data;  },  error: err => {  console.log(err);  }  });  }  });  }  public check(): void {  if (this.task?.solution) {  this.columns = Object.keys(this.task.solution[0]);  this.data = this.task.solution;  }  }  public query(): void {  const queryToRun = this.sqlCode.replace(/--.\*|\/\\*[\s\S]\*?\\*\//gm, '')  .replace(/(\r\n|\n|\r|\t)/gm, ' ');  this.taskService.query(queryToRun).subscribe({  next: (data: any) => {  if (data.length > 0) {  this.columns = Object.keys(data[0]);  this.data = data;  } else {  this.notificationService.error('Invalid query');  }  },  error: err => {  this.notificationService.error('Error during execution of query');  console.log(err);  }  });  }  public submit(): void {  const queryToRun = this.sqlCode.replace(/--.\*|\/\\*[\s\S]\*?\\*\//gm, '')  .replace(/(\r\n|\n|\r|\t)/gm, ' ');  this.taskService.check(this.task!.id, queryToRun).subscribe({  next: (data: any) => {  if (data.success) {  this.notificationService.success(data.message);  } else {  this.notificationService.error(data.message);  }  }  });  }  public createComment(): void {  if (this.content.trim()) {  this.commentService.createComment(this.content, this.tokenStorage.getUserId(), this.task!.id).subscribe({  next: (comment: Comment) => {  this.comments.unshift(comment);  this.content = '';  },  error: err => {  this.notificationService.error(err.message);  console.log(err);  }  })  }  }  public reportComment(id: number): void {  this.commentService.reportComment(id).subscribe({  next: () => {  this.notificationService.success('Comment reported');  },  error: err => {  this.notificationService.error(err.message);  console.log(err);  }  });  }  public confirm(): void {  this.taskService.confirm(this.task!.id).subscribe({  next: () => {  this.router.navigate(['/challenges']);  },  error: err => {  this.notificationService.error(err.message);  console.log(err);  }  });  }  public reject(): void {  this.taskService.reject(this.task!.id).subscribe({  next: () => {  this.router.navigate(['/challenges']);  },  error: err => {  this.notificationService.error(err.message);  console.log(err);  }  });  }  } |

Листинг – Компонент для работы с задачей