МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПЕУБЛИКИ БЕЛАРУСЬ

Учреждение образования   
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1-40 05 01 Информационные системы и технологии

Направление специальности 1-40 01 02 03 Информационные системы

и технологии (издательско-полиграфический комплекс)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОЙ РАБОТЫ**

по дисциплине «Администрирование информационных систем и web-порталов»

Тема: «Администрирование веб-приложения форума ItHub и его развертывание при помощи Docker»

Исполнитель

Студент 4 курса 2 группы Гринцевич Ю.С.

(подпись)

Руководитель работы Сазонова Д.В. (подпись)

Курсовая работа защищена с оценкой

Минск 2023

**Оглавление**

[Введение 3](#_Toc122369195)

[1 Аналитический обзор технологий проекта 4](#_Toc122369196)

[2 Проектирование приложения 5](#_Toc122369197)

[2.1 Проектирование схемы базы данных 5](#_Toc122369198)

[2.2 Проектирование диаграммы вариантов использования 7](#_Toc122369199)

[3 Разработка приложения 8](#_Toc122369200)

[3.1 Разработка сервера 8](#_Toc122369201)

[3.2 Разработка клиента 10](#_Toc122369202)

[4 Руководство пользователя 14](#_Toc122369203)

[Заключение 20](#_Toc122369204)

[Список используемых источников 21](#_Toc122369205)

# Введение

В настоящее время Интернет стал неотъемлемой частью повседневной жизни, предоставляя людям широкие возможности для общения, обмена информацией и развлечений. С развитием социальных сетей, блогов и онлайн-платформ, возрос интерес пользователей к созданию своих собственных сообществ и форумов. Форумы позволяют людям обмениваться идеями, задавать вопросы, находить ответы и налаживать контакты с единомышленниками по различным интересам и профессиональным сферам.

В связи с растущим интересом к форумам и их значимостью в онлайн-сообществах, разработка собственного форума становится актуальной задачей для многих организаций, сообществ и предпринимателей. Форумы позволяют создать взаимодействие и обмен информацией между пользователями, способствуя росту сообщества, улучшению взаимодействия и обмену знаниями.

Для разработки форума будет использована современная комбинация веб-технологий, включающая Node.js, Express и React:

* Node.js – это среда выполнения JavaScript, построенная на движке V8 Chrome. Она позволяет разрабатывать серверные приложения с использованием JavaScript, что обеспечивает единый язык программирования и на стороне клиента, и сервера;
* Express – это минималистичный и гибкий веб-фреймворк для Node.js. Он предоставляет простой и интуитивно понятный способ создания серверных приложений на языке JavaScript;
* React – это JavaScript-библиотека для создания пользовательских интерфейсов. Она позволяет разрабатывать компоненты, которые могут быть масштабируемы в ходе последующей разработки.
* Целью данного курсового проекта является разработка форума, обеспечивающего эффективное и удобное взаимодействие между пользователями и обмен знаниями в выбранной тематической области.

На основе требований пользователей будет разработана архитектура форума. Это включает определение структуры, базы данных, модулей и функциональных компонентов, необходимых для обеспечения эффективной работы форума.

Следующим шагом будет разработка форума на основе разработанной архитектуры. Будут использованы современные веб-технологии и методы программирования для создания функционального и привлекательного интерфейса, способного обеспечить удобство использования и удовлетворение потребностей пользователей, включая авторизированных и неавторизированных.

После завершения разработки форума будет проведено тестирование его функциональности, производительности и безопасности. Будут выявлены возможные ошибки, недочеты и улучшения, которые будут внесены в процессе оптимизации форума, например валидация входящих данных.

# Аналитический обзор технологий проекта

Клиентская часть:

* Разработана с использованием React и Redux;
* React компоненты отображают пользовательский интерфейс и обрабатывают пользовательские действия;
* Redux хранит состояние приложения и управляет его изменениями;
* Когда пользователь взаимодействует с интерфейсом, React компоненты отправляют запросы к серверной части через API.

Серверная часть:

* Разработана на платформе Node.js с использованием фреймворка Express;
* Express обрабатывает входящие HTTP-запросы от клиента и маршрутизирует их к соответствующим обработчикам;
* При получении запросов, серверная часть взаимодействует с удаленной MongoDB для чтения или записи данных;
* Для взаимодействия с MongoDB используется библиотека Mongoose.

Удаленная MongoDB:

* MongoDB является базой данных, развернутой на удаленном сервере или хостинге;
* Серверная часть (Node.js и Express) устанавливает соединение с удаленной MongoDB с помощью URI-строки, указывающей на адрес и доступные учетные данные;
* Запросы серверной части отправляются к удаленной MongoDB для выполнения операций чтения и записи данных.

React –фреймворк для разработки пользовательского интерфейса. Он предоставляет разработчикам удобные и расширяемые инструменты для реализации различных задач, а также обширную экосистему библиотек, которые расширяют его функциональность.

Node.js – среда выполнения кода JavaScript, которая позволяет писать серверный код для веб-страниц и веб-приложений, а также для программ командной строки. С помощью Node.js реализуется парадигма «JavaScript для всего». Она предполагает использование одного языка программирования для разработки вэб-приложений вместо применения разных языков для работы над фронтендом и бэкендом.

Express.js – это фреймворк для Node.js, который реализовывает слой функций, необходимых для создания эффективных приложений и API. Его использование значительно сокращает написание кода, а, значит, уменьшается затрачиваемое на разработку время. Является стандартным каркасом для Node.js.

Docker – это программная платформа для быстрой разработки, тестирования и развертывания приложений. Docker упаковывает ПО в стандартизованные блоки, которые называются контейнерами. Каждый контейнер включает все необходимое для работы приложения: библиотеки, системные инструменты, код и среду исполнения. Благодаря Docker можно быстро развертывать и масштабировать приложения в любой среде и сохранять уверенность в том, что код будет работать.

# Проектирование приложения

2.1 Проектирование схемы базы данных

Важный этап разработки приложения, так как оно строится на основе структурированного хранения и доступа к данным. Правильное проектирование базы данных позволяет достичь оптимальной производительности, обеспечить безопасность данных и удобство использования приложения. Основной задачей проектирования базы данных является определение структуры, которая соответствует требованиям приложения и обеспечивает эффективное хранение и извлечение данных. Это включает определение сущностей и их атрибутов, установление связей между сущностями и создание таблиц для их представления.

При проектировании базы данных также необходимо учесть правила целостности данных, которые помогают поддерживать точность и надежность данных. Эти правила включают ограничения, проверки и уникальные индексы, которые предотвращают ошибки ввода данных и обеспечивают целостность данных в базе. Важно также учесть масштабируемость базы данных при проектировании, чтобы она могла эффективно обрабатывать растущие объемы данных. Необходимо предусмотреть возможность расширения базы данных в будущем, чтобы она могла легко адаптироваться к увеличивающимся потребностям приложения.

Таким образом, проектирование базы данных является критическим этапом, гарантирующим эффективность и надежность работы приложения, а также способствующим его дальнейшему развитию и масштабируемости. Для хранения данных в приложении используется база данных MongoDB.

На рисунке 2.3 изображена схема базы данных программного средства.

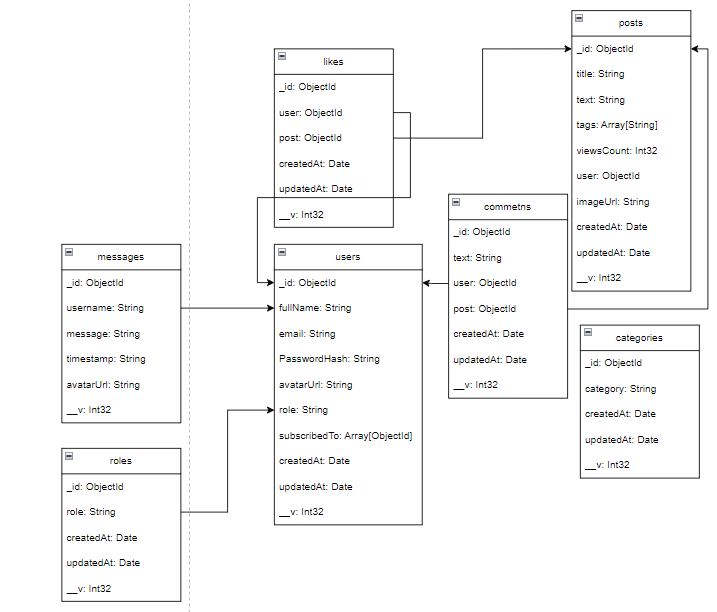


Рисунок 2.3 – Модель базы данных

Далее будет описана каждая таблица базы данных.

В таблице «categories» хранятся все основные разделы форума. Описание структуры таблицы «categories» представлено в таблице 2.1.

Таблица 2.1 – Структура таблицы «categories»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип | Описание столбца |
| \_id | ObjectId | Идентификатор категории |
| category | String | Названии категории |
| createdAt | Date | Дата создания документа |
| updatedAt | Date | Дата обновления документа |
| \_\_v | Int32 | Версия документа коллекции |

В таблице «comments» хранится информация о комментарияx. Описание структуры таблицы «comments» представлено в таблице 2.2.

Таблица 2.2 – Структура таблицы «comments»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип | Описание столбца |
| \_id | ObjectId | Идентификатор комментария |
| text | String | Текст комментария |
| user | ObjectId | Идентификатор пользователя |
| post | ObjectId | Идентификатор обсуждения |
| createdAt | Date | Дата создания документа |
| updatedAt | Date | Дата обновления документа |
| \_\_v | Int32 | Версия документа коллекции |

Таблица «likes» содержит информацию о лайках, которые ставили пользователи определённым обсуждениям. Описание структуры наглядно представлено в таблице 2.3.

Таблица 2.3 – Структура таблицы «likes»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип | Описание столбца |
| \_id | ObjectId | Идентификатор лайка |
| user | ObjectId | Идентификатор пользователя |
| post | ObjectId | Идентификатор обсуждения |
| createdAt | Date | Дата создания документа |
| updatedAt | Date | Дата обновления документа |
| \_\_v | Int32 | Версия документа коллекции |

В таблице «messages» содержится информация о последних десяти сообщениях в групповой чат. Описание структуры таблицы «messages» представлено в таблице 2.4.

Таблица 2.4 – Структура таблицы «messages»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип | Описание столбца |
| \_id | ObjectId | Идентификатор сообщения |
| username | String | Имя пользователя |
| message | String | Сообщение от пользователя |
| timestamp | String | Время отправки сообщения |
| avatarUrl | String | Ссылка на аватар пользователя |
| \_\_v | Int32 | Версия документа коллекции |

В таблице «posts» содержатся обсуждения, созданные пользователем. Описание структуры таблицы «posts» представлено в таблице 2.5.

Таблица 2.5 – Структура таблицы «posts»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип | Описание столбца |
| \_id | ObjectId | Идентификатор лайка |
| title | String | Заголовок обсуждения |
| text | String | Текст обсуждения |
| tags | Array[String] | Тэги обсуждения |
| viewsCount | Int32 | Количество просмотров обсуждения |
| user | ObjectId | Идентификатор пользователя |
| imageUrl | String | Путь до картинки обсуждения |
| createdAt | Date | Дата создания документа |
| updatedAt | Date | Дата обновления документа |
| \_\_v | Int32 | Версия документа коллекции |

Таблица «roles» содержит роли, поддерживаемые на форуме. Описание структуры таблицы «roles» представлено в таблице 2.6.

Таблица 2.6 – Структура таблицы «roles»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| \_id | ObjectId | Идентификатор роли |
| createdAt | Date | Дата создания документа |
| updatedAt | Date | Дата обновления документа |
| \_\_v | Int32 | Версия документа коллекции |

Таблица «users» содержит информацию о пользователях форума. Здесь можно увидеть полное имя, почту, хеш пароля, ссылку на картинку профиля и роль на форуме. Описание структуры таблицы «users» представлено в таблице 2.7.

Таблица 2.7 – Структура таблицы «users»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип | Описание столбца |
| \_id | ObjectId | Идентификатор пользователя |
| fullName | String | Полное имя пользователя |
| Email | String | Почта полльзователя |
| passwordHash | String | Хеш пароля |
| avatarUrl | String | Ссылка на картинку профиля |
| role | String | Роль на форуме |
| subscribedTo | Array[ObjectId] | На кого пользователь подписан |
| createdAt | Date | Дата создания документа |
| updatedAt | Date | Дата обновления документа |
| \_\_v | Int32 | Версия документа коллекции |

Общий вывод по представленным таблицам Mongoose (MongoDB) для форума:

* Каждая таблица представляет определенный аспект функциональности форума, такой как категории, комментарии, лайки, сообщения, обсуждения и пользователи;
* Использование ObjectId в качестве идентификаторов позволяет уникально идентифицировать каждую запись и обеспечивает эффективность при выполнении операций поиска и связывания данных;
* Все таблицы содержат общие столбцы, такие как идентификатор (\_id), даты создания (createdAt) и обновления (updatedAt) записи, а также версию документа коллекции (\_\_v). Эти столбцы позволяют отслеживать и управлять изменениями данных в базе данных;
* Каждая таблица имеет свои уникальные столбцы, соответствующие своей функциональности. Например, таблица «posts» содержит столбцы, такие как заголовок, текст, тэги, количество просмотров и путь к изображению;
* Связь между таблицами может быть установлена с использованием идентификаторов. Например, таблица «comments» имеет столбец «post», который ссылается на идентификатор обсуждения в таблице «posts»;
* Организация данных в структурированной форме позволяет эффективно хранить и извлекать информацию, связанную с форумом.

В целом, структура таблиц Mongoose обеспечивает удобный и эффективный способ хранения и управления данными для форума. Структура таблиц Mongoose в приложении форум представляет собой совокупность моделей данных, которые определяют схему и поведение каждой коллекции в базе данных MongoDB. Каждая модель соответствует определенному типу данных и содержит поля, описывающие структуру объектов.

Одной из ключевых преимуществ структуры таблиц Mongoose является возможность определения связей между коллекциями. Например, модель "Пользователь" может иметь поле, содержащее ссылку на модель "Обсуждение". Это позволяет связывать комментарии или ответы с конкретными обсуждениями и обеспечивает структурированность данных.

Также структура таблиц позволяет определить различные типы полей для хранения различных данных. Например, можно использовать поле типа "Строка" для хранения текстовых комментариев или поле типа "Дата" для сохранения даты и времени публикации комментария. Это способствует эффективному и точному хранению информации.

2.2 Проектирование диаграммы вариантов использования

Диаграмма использования для приложения форум представляет собой графическое изображение функциональности, которая доступна пользователям при использовании данного приложения. Она помогает визуализировать взаимодействие между пользователями и системой, а также показывает основные действия, которые могут быть выполнены в приложении форума.

Диаграмма использования для приложения представлена на рисунке 2.1.



Рисунок 2.1 – Диаграмма вариантов использования

Как видно из диаграммы использования, приложение поддерживает две основыные роли: пользователь и администратор. Авторизованный администратор, в отличие от пользователя, имеет доступ к функциям редактирования, удаления чужих обсуждений, а так же, удалять чужие комментарии.

# Разработка приложения

Архитектура приложения форум основана на клиент-серверной модели, где серверная часть отвечает за обработку запросов и хранение данных, а клиентская часть предоставляет пользовательский интерфейс для взаимодействия с сервером.

Взаимодействие между клиентом и сервером осуществляется посредством REST API (Representational State Transfer Application Programming Interface), что обеспечивает простоту и гибкость взаимодействия между различными компонентами приложения.

Для работы с базой данных MongoDB в приложении используется ODM (Object-Document Mapping) Mongoose. Он предоставляет удобный интерфейс для работы с данными MongoDB, позволяя определить модели данных и проводить операции чтения, записи, обновления и удаления объектов в базе данных.

Использование Mongoose позволяет сократить количество кода, необходимого для работы с базой данных, и упростить процесс разработки и поддержки приложения. Он также предоставляет возможности для валидации данных, определения связей между коллекциями и другие функциональности, упрощающие работу с данными.

Такая архитектура и выбранные инструменты позволяют эффективно разрабатывать и масштабировать приложение форум, обеспечивая удобство использования и надежность взаимодействия с данными.

3.1 Разработка сервера

Для разработки серверной части приложения использовался фреймворк Express.js. Основная концепция Express.js заключается в определении маршрутов и обработчиков запросов. Маршруты представляют собой URL-адреса, на которые сервер должен реагировать, а обработчики запросов определяют действия, которые должны быть выполнены при обращении к определенному маршруту.

Определение маршрутов в Express.js очень просто и понятно. Можно указать путь к маршруту, тип запроса (GET, POST, PUT, DELETE и т. д.) и соответствующий обработчик запроса. Обработчик представляет собой функцию, которая выполняет необходимые действия при обращении к данному маршруту, например, извлекает данные из базы данных, обрабатывает их и отправляет ответ клиенту.

Express.js также предоставляет множество дополнительных функций и middleware-компонентов, которые облегчают разработку серверной части приложения. Middleware позволяет выполнять определенные операции перед или после обработки запросов, такие как аутентификация, валидация данных, обработка ошибок и другие. Это способствует повышению безопасности и эффективности приложения. Кроме того, Express.js предлагает гибкую настройку сервера. Можно легко определить порт, на котором сервер будет прослушивать запросы, настроить маршруты статических файлов, задать параметры обработки запросов, включая обработку тела запроса, заголовков и других параметров.

### 3.1.1 Конфигурация

Для работы приложения строка подключения к MongoDB, которая располагается на сервере Mongo, предварительно в настройках базы данных нужно включить подключение из любого места на планете. Пример строки подключения и использованием ODM mongoose представлен на листинге 3.1.

mongoose .connect('mongodb+srv://admin:wwwwww@cluster0.klivmmt.

mongodb.net/blog?retryWrites=true&w=majority')

.then(() => console.log('DB ok'))

.catch((err) => console.log('DB error', err));

Листинг 3.1 – Пример строки подключения

Для запуска и настройки сервера определено создание экземпляра приложения Express, настройка хранилища для загружаемых файлов с использованием Multer, создание HTTPS-сервера с использованием предоставленных сертификатов. Ниже представлен листинг 3.2 с кодом файла index.js.

const app = express();import { AppModule } from './app.module';

const upload = multer({ storage });

app.use(express.*json*());  
 app.use(cors());  
 app.use('/uploads', express.*static*('uploads'));

const server = https.createServer(

{

key: fs.readFileSync('./cert/localhost.key', 'utf-8'),

cert: fs.readFileSync('./cert/localhost.crt', 'utf-8'),

},

app

);

Листинг 3.2 – Часть файла index.js

Общий смысл этого кода заключается в настройке сервера Express для обработки запросов, разбора JSON-данных, обработки загрузки файлов с использованием Multer, обеспечения поддержки CORS и предоставления статических файлов, а также создания HTTPS-соединения с использованием предоставленных сертификатов.

### 3.1.2 Реализация REST API. Контроллеры

Контроллеры в Express представляют собой обработчики, которые определяют функциональность и логику обработки каждого маршрута (роута) в вашем REST API. Они связываются с определенными URL-адресами и методами запроса, и выполняют необходимые действия, такие как обработка данных, взаимодействие с базой данных и отправка ответов клиенту. На листинге 3.3 приведен пример контроллера для регистрации и авторизации.

app.get('/auth/me', checkAuth, UserController.getMe);

app.post('/auth/login', *loginValidation*, handleValidationErrors, UserController.login);

app.post('/auth/register', *registerValidation*, handleValidationErrors, UserController.register);

Листинг 3.3 – Маршруты и методы для авторизации и регистрации

Первый маршрут предназначен для получения информации о текущем аутентифицированном пользователе. Перед вызовом обработчика UserController.getMe будет выполнена функция промежуточной обработки checkAuth, которая проверяет, аутентифицирован ли пользователь.

Второй маршрут предназначен для аутентификации пользователя путем входа в систему. Перед вызовом обработчика UserController.login будут выполнены функции промежуточной обработки loginValidation (для проверки входных данных) и handleValidationErrors (для обработки возможных ошибок валидации). Если данные прошли проверку, вызывается метод login из UserController, который обрабатывает запрос, проверяет учетные данные пользователя и возвращает результат аутентификации.

Третий маршрут предназначен для регистрации нового пользователя. Перед вызовом обработчика UserController.register будут выполнены функции промежуточной обработки registerValidation (для проверки входных данных) и handleValidationErrors (для обработки возможных ошибок валидации). Если данные прошли проверку, вызывается метод register из UserController, который обрабатывает запрос, создает нового пользователя и сохраняет его в базе данных. Ниже представлен листинг 3.4 с кодом проверки валидации.

export const loginValidation = [

body('email', 'Неверный формат почты').isEmail(),

body('password', 'Пароль должен быть минимум 5 символов').isLength({ min: 5 }),

];export const registerValidation = [body('email', 'Неверный формат почты').isEmail(), body('password', 'Пароль должен быть минимум 5 символов').isLength({ min: 5 }),

body('fullName', 'Укажите имя').isLength({ min: 3 }),

body('avatarUrl', 'Неверная ссылка на аватарку').optional().isURL(),];

Листинг 3.4 – Класс validations

Здесь используется библиотека express-validator, которая предоставляет удобные методы для проверки и валидации данных запроса.

loginValidation - это массив функций-обработчиков, которые выполняют проверку входных данных при попытке входа в систему. В данном случае, проверяется формат поля email (должен быть валидным email-адресом) и длина поля password (должна быть не менее 5 символов).

registerValidation - это массив функций-обработчиков, которые выполняют проверку входных данных при регистрации нового пользователя. В данном случае, проверяется формат поля email (должен быть валидным email-адресом), длина поля password (должна быть не менее 5 символов), длина поля fullName (должна быть не менее 3 символов) и опционально ссылка на аватарку avatarUrl (обязательно должна быть валидной URL).

Если входные данные не проходят одну из проверок, будет сгенерировано сообщение об ошибке, которое будет возвращено клиенту с соответствующим статусом ответа.

### 3.1.4 Взаимодействие с БД

Для того, чтобы взаимодействовать с базой данных, используется ODM Mongoose, предоставляющая удобные инструменты для работы с MongoDB.

При использовании Mongoose, вы можете легко определить модели данных, которые соответствуют коллекциям в вашей базе данных. Каждая модель содержит схему, которая определяет структуру документов в коллекции. Схема может включать поля, их типы, валидацию, уникальность и другие свойства.

После подключения и определения моделей, вы можете использовать Mongoose в ваших контроллерах для выполнения операций с базой данных. Например, вы можете создавать новых пользователей, получать список пользователей, обновлять и удалять пользователей с помощью соответствующих методов модели.

### 3.1.6 JWT. Аутентификация и авторизация

JWT (JSON Web Token) - это открытый стандарт для создания токенов доступа, которые могут быть переданы и использованы между двумя сторонами в формате JSON. Он чрезвычайно широко используется для аутентификации и авторизации во многих веб-приложениях.

JWT состоит из трех частей: заголовка (header), полезной нагрузки (payload) и подписи (signature). Заголовок содержит информацию о типе токена и используемом алгоритме шифрования. Полезная нагрузка содержит информацию, которую вы хотите передать в токене, например, идентификатор пользователя или его роли. Подпись используется для проверки целостности токена и подтверждения его подлинности.

При регистрации и входе в систему ваш код использовал библиотеку jsonwebtoken для создания и проверки JWT-токенов. В функции register после сохранения пользователя в базе данных с использованием bcrypt для хэширования пароля, токен генерируется с помощью метода jwt.sign(). В функции login также выполняется проверка пароля с использованием bcrypt, и при успешной аутентификации также генерируется и возвращает токен. JWT-токены имеют определенный срок действия, указанный в полезной нагрузке токена. По истечении срока действия токена требуется обновление или повторная аутентификация пользователя.

JWT-токены предоставляют ряд преимуществ, таких как простота использования, переносимость, возможность хранить в них информацию о пользователе и его правах, а также отсутствие необходимости хранить состояние сессии на сервере. Однако, для обеспечения безопасности, важно правильно настроить и защитить ключи подписи, а также учитывать потенциальные уязвимости, такие как утечка токенов или подделка токенов.

### 3.1.8 Обмен сообщениями с клиентом в реальном времени

Здесь мы рассмотрим, как обмениваться сообщениями с клиентом в режиме реального времени с помощью Socket.IO. Socket.IO является мощным инструментом, позволяющим устанавливать постоянное соединение между сервером и клиентом, чтобы обмениваться данными без необходимости постоянного обновления страницы.

Socket.IO - это библиотека, позволяющая реализовать веб-сокеты в Node.js и веб-браузерах. Веб-сокеты обеспечивают двустороннюю связь между сервером и клиентом, позволяя отправлять и получать данные в режиме реального времени. Прежде чем мы сможем начать обмениваться сообщениями с клиентом, нам нужно создать сервер Socket.IO на стороне сервера. Для этого мы можем использовать существующий сервер, созданный с помощью Express.js, или создать отдельный сервер. Когда клиент устанавливает соединение с сервером, событие connection срабатывает на сервере. Мы можем добавить обработчик этого события, чтобы выполнить действия при подключении нового клиента. Socket.IO позволяет обмениваться сообщениями между сервером и клиентом с помощью событий. События могут быть отправлены как с сервера на клиент, так и с клиента на сервер. На сервере мы принимаем сообщение от клиента и отправляем ответное сообщение обратно на клиент событием message. Таким образом, мы можем обмениваться сообщениями между клиентом и сервером в режиме реального времени.

3.2 Разработка клиента

В клиентской части приложения был использован фреймворк React с применением Redux для управления состоянием приложения. React является популярным и мощным инструментом для создания пользовательских интерфейсов, позволяющим разделить пользовательский интерфейс на компоненты и управлять ими с помощью виртуального DOM (Virtual DOM). Redux, в свою очередь, является библиотекой для управления состоянием приложения, обеспечивающей единообразное хранение данных и простую механику их изменения. Основные преимущества использования React включают:

* Компонентная архитектура: React позволяет разбить пользовательский интерфейс на небольшие и независимые компоненты, что упрощает разработку, тестирование и поддержку кода. Каждый компонент имеет свою внутреннюю логику и отображение, и может быть использован повторно в разных частях приложения.
* Виртуальный DOM: React использует виртуальный DOM, который является легковесным представлением реального DOM. Это позволяет React эффективно обновлять только необходимые части пользовательского интерфейса, минимизируя количество операций обновления и повышая производительность.
* Удобство работы с данными: React облегчает работу с данными и их передачу между компонентами. Он предлагает односторонний поток данных (от верхнего уровня к нижнему), что делает отслеживание изменений и управление состоянием проще и более предсказуемыми.

Redux, с другой стороны, предоставляет единое хранилище для состояния приложения и набор функций для его изменения. Он следует принципу однонаправленного потока данных, что упрощает отслеживание изменений и обновление пользовательского интерфейса. Основные преимущества использования Redux включают:

* Централизованное хранение данных: Redux хранит все данные приложения в едином хранилище. Это делает управление и синхронизацию состояния между компонентами более простыми и предсказуемыми.
* Предсказуемость и отладка: Изменения состояния происходят только через чистые функции, называемые "редюсерами". Это делает состояние предсказуемым и облегчает отладку, так как все изменения происходят в одном месте.
* Расширяемость: Redux обеспечивает удобный механизм для добавления промежуточного программного обеспечения (middleware), которое может обрабатывать дополнительные логику и позволяет легко расширять функциональность приложения.

Сочетание React и Redux обеспечивает мощный инструментарий для разработки сложных пользовательских интерфейсов с удобным управлением состоянием. React обеспечивает эффективное отображение пользовательского интерфейса, в то время как Redux предоставляет единое и предсказуемое хранилище данных. Это позволяет создавать масштабируемые и гибкие приложения, которые легко тестируются и поддерживаются.

### 3.2.1 Хранение состояния

В Redux состояние приложения хранится в едином объекте, называемом "хранилищем" (store). Хранилище содержит все данные, необходимые для работы приложения, и предоставляет механизм для их обновления и получения.

В Redux состояние является неизменяемым, что означает, что его нельзя изменить напрямую. Вместо этого, для изменения состояния, создается новая копия состояния с помощью чистых функций, называемых "редюсерами" (reducers). Редюсеры принимают текущее состояние и действие (action) в качестве аргументов и возвращают новую копию состояния с обновленными данными. Для работы с Redux в приложении необходимо выполнить следующие шаги:

* Определение действий (actions): Действия представляют собой объекты, которые описывают, что произошло в приложении. Они содержат информацию, необходимую для обновления состояния. Например, действие может быть "добавить пост" и содержать данные нового поста.
* Создание редюсеров (reducers): Редюсеры являются чистыми функциями, которые принимают текущее состояние и действие, и возвращают новую копию состояния с обновленными данными. Редюсеры определяют, как будет изменяться состояние в ответ на действия. Обычно каждый редюсер отвечает за обновление определенной части состояния.
* Создание хранилища (store): Хранилище создается с использованием функции createStore из пакета Redux. Оно принимает корневой редюсер, который объединяет все редюсеры приложения, и опциональные параметры. Хранилище содержит текущее состояние и методы для его обновления и получения.
* Использование состояния в компонентах: Для чтения состояния из хранилища и обновления состояния, компоненты могут использовать хуки useSelector и useDispatch из пакета react-redux. Хук useSelector позволяет получить доступ к определенной части состояния, а хук useDispatch предоставляет метод для отправки действий в хранилище.

На листинге 3.5 листинг полного кода «хранилища».

import { configureStore } from '@reduxjs/toolkit';

import { *postsReducer* } from './slices/posts';

import { *authReducer* } from './slices/auth';

const *store* = configureStore({reducer:{

posts: *postsReducer*, auth: *authReducer*, },});

export default *store*;

Листинг 3.5 – Хранилище Redux

Функция configureStore из пакета @reduxjs/toolkit принимает объект с настройками, в котором определен редюсер для каждой части состояния. В данном случае, состояние разделено на две части: posts и auth. Каждый редюсер обрабатывает соответствующую часть состояния и определяет, как будет происходить обновление состояния в ответ на действия.

После создания хранилища, оно экспортируется по умолчанию и может быть подключено к компонентам React с помощью Provider из react-redux. Компоненты могут использовать хуки useSelector и useDispatch для чтения состояния и отправки действий в хранилище, соответственно.

Таким образом, данный код создает и настраивает хранилище Redux, где postsReducer и authReducer определяют, как обновлять состояние для функциональности постов и аутентификации.

### 3.2.2 Маршрутизация

В приложении используется библиотека react-router-dom для реализации маршрутизации. Некоторые маршруты в приложении требуют авторизации пользователя, а также наличия определенной роли. При переходе на конкретный маршрут происходит проверка наличия токена в локальном хранилище. Если токен присутствует, пользователь считается авторизованным и получает доступ к определенному функционалу.

Часть реализации маршрутизации, представленная на листинге 3.6, позволяет по токену получить идентификатор пользователя (\_id) и использовать эту информацию для получения роли, полного имени и изображения профиля.

export const *fetchAuth* = createAsyncThunk('auth/fetchAuth', async (params)=>{ const { data } = await *axios*.post('/auth/login', params); return data;});

export const *fetchRegister* = createAsyncThunk('auth/fetchRegister', async (params) => { const { data } = await *axios*.post('/auth/register', params); return data;});

export const *fetchAuthMe* = createAsyncThunk('auth/fetchAuthMe', async ()=>{ const { data } = await *axios*.get('/auth/me');

return data;});

Листинг 3.6 – Подготовленные запросы проверяющие авторизацию

В данном коде определены три асинхронные функции, используемые для выполнения запросов к серверу, связанных с аутентификацией и регистрацией пользователей.

* fetchAuth ­– эта функция выполняет запрос к серверу для аутентификации пользователя. Она использует библиотеку Axios для отправки POST-запроса на путь '/auth/login' с переданными параметрами params. После получения ответа от сервера, извлекается поле data и возвращается как результат выполнения функции.
* fetchRegister – эта функция выполняет запрос к серверу для регистрации нового пользователя. Она также использует Axios для отправки POST-запроса на путь '/auth/register' с переданными параметрами params. После получения ответа от сервера, извлекается поле data и возвращается в качестве результата выполнения функции.
* fetchAuthMe – эта функция выполняет запрос к серверу для получения информации о текущем аутентифицированном пользователе. Она использует Axios для отправки GET-запроса на путь '/auth/me'. После получения ответа от сервера, извлекается поле data и возвращается в качестве результата выполнения функции.

Эти функции используются внутри Redux-редюсеров с помощью библиотеки Redux Toolkit. Они представляют асинхронные операции, которые могут быть диспетчеризованы из компонентов и могут обновлять глобальное состояние Redux-хранилища с помощью действий, сгенерированных Redux Toolkit.

# 3.2.3 Обмен сообщениями с сервером в реальном времени

Для реализации двунаправленного соединения с сервером в режиме реального времени используется протокол связи поверх TCP-соединения - Websocket. Для работы с протоколом была использована библиотека, упрощающая работу с сокетами socket-io-client. При загрузке страницы чата, мы подключаемся к серверу по протоколу WebSocket и при успешном подключении подписываемся на событие message при возникновении которого, отображаем новое сообщение в компоненте чата.  На листинге 3.7 клиентской части кода Websocket.

import *io* from "socket.io-client";

export const Chat = () => {

const [chatMessages, setChatMessages] = useState([]);

const [newMessage, setNewMessage] = useState("");

const [socket, setSocket] = useState(null);

const [userFullName, setUserFullName] = useState("");

const [userAvatarUrl, setUserAvatarUrl] = useState("");

useEffect(()=>{const newSocket = *io*("https://localhost:4444");  
 setSocket(newSocket);  
 newSocket.on("chatMessage",(message)=>{  
 setChatMessages((prevMessages) => [...prevMessages, message]);});

Листинг 3.7 – Клиенская часть Websocket

В данном коде определен компонент Chat, который использует библиотеку socket.io-client для создания сокетного подключения и обмена сообщениями в режиме реального времени.Компонент Chat содержит следующие состояния:

* chatMessages: массив сообщений чата.
* newMessage: текущее введенное пользователем сообщение.
* socket: объект сокета для установки соединения с сервером.
* userFullName: полное имя пользователя.
* userAvatarUrl: URL-адрес аватара пользователя.

При монтировании компонента (useEffect с пустым массивом зависимостей), устанавливается новый экземпляр сокета, который подключается к адресу "https://localhost:4444". Затем сокету присваивается новое значение с помощью setSocket. Затем устанавливается обработчик события "chatMessage".

# Руководство пользователя

После окончания разработки веб-приложения было принято решение развернуть данный проект на Docker, т.к. это существенно облегчает запуск данного приложения на других системах.

Docker является удобным способом развертывания приложений. Он позволяет создавать контейнеры, которые обеспечивают изолированное окружение для приложений, независимо от других приложений. Для удобного развертывания мы можем использовать Dockerfile – файл, содержащий инструкции по контейнеризации самого приложения. Необходимо добавить Dockerfile как для клиентской, так и для серверной части приложения.

Docker image представляет собой шаблон или образ для создания Docker-контейнеров. Контейнеры, в свою очередь, являются экземплярами Docker-образов, которые обеспечивают стандартизацию развертывания приложений и изолируют их от основной инфраструктуры.

Контейнеры позволяют упаковать приложение и все его зависимости (такие как библиотеки, системные утилиты и файлы настройки) в единый образ. Этот образ является портативным и может быть запущен на любой инфраструктуре, где установлен Docker. Контейнеры предоставляют изолированную среду выполнения для приложения, что означает, что они не влияют на основную операционную систему и другие контейнеры, работающие на том же хосте.

Благодаря контейнерам, приложение и его зависимости упаковываются вместе, что упрощает перенос приложения на другую инфраструктуру. Docker позволяет создавать, запускать и управлять контейнерами с помощью простых команд и инструментов. Это обеспечивает консистентное и повторяемое развертывание приложений в различных средах, включая локальные компьютеры разработчиков, серверы и облачные платформы.

Первое что нужно сделать – это установить Docker Desktop. Основная страница отображена на рисунке 4.1.

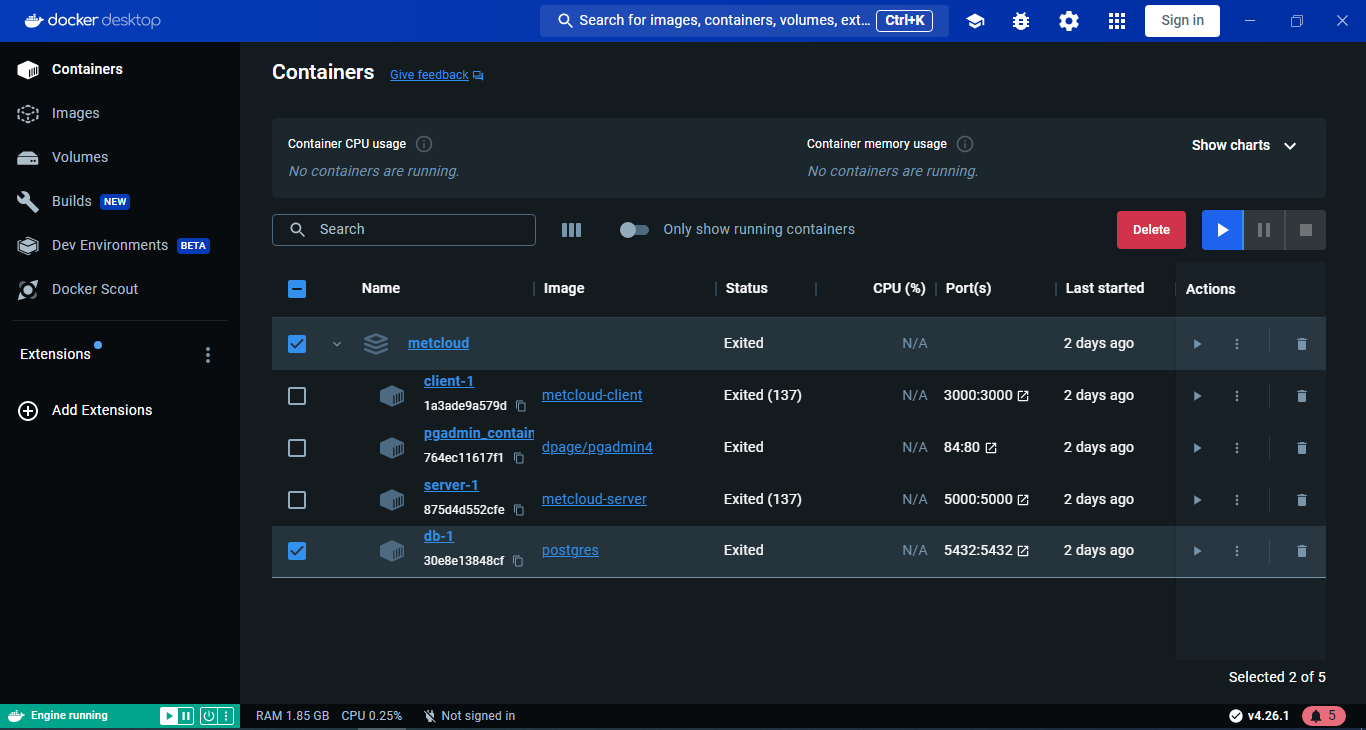


Рисунок 4.1 – Стартовая страница Docker Desktop

Как видно из рисунка, на данной системе уже установлены некоторые образы баз данных, а также имеются их рабочие контейнеры.

Docker Hub - это публичный репозиторий, предназначенный для хранения и обмена Docker-образами, расширениями и плагинами. На Docker Hub можно найти готовые образы контейнеров и установить их, следуя инструкциям.

Для создания собственного образа вашего разрабатываемого приложения нужно описать его с помощью Dockerfile. Dockerfile является инструкцией для сборки образа и содержит информацию о необходимых программных компонентах, зависимостях, базовых образах и командах, необходимых для развертывания образа.

Однако, если приложение состоит из нескольких компонентов, более удобным и целесообразным решением является использование docker-compose. Docker-compose позволяет развертывать и настраивать несколько контейнеров одновременно. Вместо того, чтобы настраивать каждый контейнер отдельно, можно описать все компоненты в файле docker-compose.yml и запустить их одной командой: «docker-compose up».

Использование docker-compose упрощает управление и развертывание многоконтейнерных приложений, обеспечивая простоту и консистентность в настройке и запуске нескольких контейнеров, связанных между собой.

Для того, чтобы использовать приведенную выше команду, Docker-compose должен быть установлен (в последних версиях устанавливается вместе с Docker Desktop). На рисунке 4.2 представлена общая структура проекта.

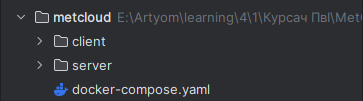


Рисунок 4.2 – Общая структура проекта

Папка с названием «client» является разработанным клиентом, а папка с названием «server» – разработанным сервером. Docker-compose – файл, описывающий и запускающий композицию нескольких контейнеров.

На серверной части можно выделить следующий порядок действий:

* FROM node:alpine: Определяет базовый образ, в данном случае, образ Node.js Alpine;
* WORKDIR /usr/app/back: Устанавливает рабочую директорию внутри контейнера как /usr/app/back.;
* EXPOSE 5000: Позволяет указать, что контейнер будет слушать порт 5000;
* COPY ./ ./: Копирует все файлы и директории из текущего контекста сборки внутрь контейнера в директорию /usr/app/back;
* RUN npm install: Устанавливает зависимости серверной части приложения с помощью npm install;
* CMD ["npm", "run", "dev"]: Задает команду, которая будет выполнена при запуске контейнера. В данном случае, выполняется команда npm run dev, которая запускает серверное приложение.

Dockerfile для серверной части приложения представлен в листинге 4.1.

FROM node:alpine  
WORKDIR /usr/app/back  
EXPOSE 5000  
COPY ./ ./  
RUN npm install  
CMD ["npm", "run", "dev"]

Листинг 4.1 – Dockerfile для серверной части приложения

Следующим шагом будет создание файла Dockerfile для развертки клиентской части приложения.

Порядок действий следующий:

* FROM node:alpine: Определяет базовый образ, в данном случае, образ Node.js Alpine;
* WORKDIR /usr/app/front: Устанавливает рабочую директорию внутри контейнера как /usr/app/front;
* EXPOSE 3000: Позволяет указать, что контейнер будет слушать порт 3000;
* COPY ./ ./: Копирует все файлы и директории из текущего контекста сборки внутрь контейнера в директорию /usr/app/front;
* RUN npm install --legacy-peer-deps: Устанавливает зависимости клиентской части приложения с помощью npm install --legacy-peer-deps. Флаг --legacy-peer-deps используется для обхода проверки совместимости зависимостей;
* CMD ["npm", "start"]: Задает команду, которая будет выполнена при запуске контейнера. В данном случае, выполняется команда npm start, которая запускает клиентское приложение.

Для каждого Dockerfile был создан dockerignore, чтобы указать исключение использования. Dockerfile для клиентской части приложения представлен в листинге 4.2.

FROM node:alpine  
WORKDIR /usr/app/front  
EXPOSE 3000  
COPY ./ ./  
RUN npm install --legacy-peer-deps  
CMD ["npm", "start"]

Листинг 4.2 – Dockerfile для клиентской части приложения

На данном этапе были созданы все необходимые docker-файлы, которые нужны для создания и запуска контейнеров Docker. Что бы не запускать все части проекта отдельно, был создан файл docker-compose.yml. В нем будет происходить процесс создания сервисов, настройка их окружения, конфигурационных файлов и так далее. Для развёртывания клиента и сервера используются сервисы client и server. Они выполняют содержимое их Dockerfile. Также в самом файле прописан скрипт для развёртывания базы данных и ресурса для визуального представления базы данных.

Настройка сервиса для базы данных:

* используется готовый образ postgres из Docker Hub;
* заданы переменные окружения для настройки базы данных, такие как имя пользователя, пароль и имя базы данных;
* порт 5432 контейнера отображается на порт 5432 хоста;
* сервис присоединен к сети mynetwork;
* создан том db\_data для сохранения данных PostgreSQL.

Настройка сервиса для pgadmin:

* используется готовый образ dpage/pgadmin4 из Docker Hub;
* заданы переменные окружения для настройки pgAdmin, такие как адрес электронной почты и пароль администратор;
* порт 80 контейнера отображается на порт 80 хоста;
* сервис присоединен к сети mynetwork;
* создан том pgadmin для сохранения данных pgAdmin;
* зависит от сервиса db, чтобы обеспечить правильную последовательность запуска контейнеров.

Для этого файла определены тома db\_data и pgadmin для сохранения данных базы данных PostgreSQL и pgAdmin соответственно. Создана сеть mynetwork с драйвером bridge, к которой присоединены все сервисы.

Содержимое docker-compose файла представлено в листинге 4.3.

version: '3'  
services:  
 server:  
 build:  
 context: ./server  
 dockerfile: Dockerfile  
 ports:  
 - 5000:5000  
 networks:  
 - mynetwork  
 depends\_on:  
 - db  
  
 client:  
 build:  
 context: ./client/metcloud  
 dockerfile: Dockerfile  
 ports:  
 - 3000:3000  
 networks:  
 - mynetwork  
 depends\_on:  
 - server  
  
 db:  
 image: postgres  
 environment:  
 - POSTGRES\_USER=postgres  
 - POSTGRES\_PASSWORD=postgres  
 - POSTGRES\_DB=metcloud\_db  
 ports:  
 - 5432:5432  
 networks:  
 - mynetwork  
 volumes:  
 - db\_data:/var/lib/postgresql/dat a  
  
 pgadmin:  
 container\_name: pgadmin\_container  
 image: dpage/pgadmin4  
 environment:  
 - PGADMIN\_DEFAULT\_EMAIL=postgres@gmail.com  
 - PGADMIN\_DEFAULT\_PASSWORD=postgres  
 - PGADMIN\_CONFIG\_SERVER\_MODE=False  
 volumes:  
 - pgadmin:/var/lib/pgadmin  
 depends\_on:  
 - db  
 ports:  
 - 80:80  
 networks:  
 - mynetwork  
 restart: unless-stopped  
  
volumes:  
 db\_data:  
 pgadmin:  
  
networks:  
 mynetwork:  
 driver: bridge

Листинг 4.3 – Docker-compose файл

Чтобы с помощью него запустить приложение, нужно в директории с этим файлом в консоли прописать команду «docker-compose up». Результаты команды в Docker Desktop продемонстрированы на рисунке 4.3.

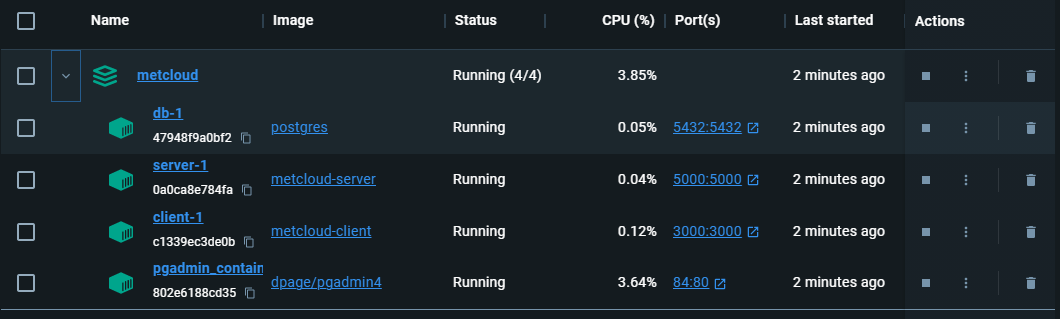


Рисунок 4.3 – Результаты команды docker-compose up

Таким образом, данный docker-compose файл определяет и связывает контейнеры для сервера, клиентского приложения, базы данных PostgreSQL и pgAdmin, позволяя запустить их вместе и взаимодействовать друг с другом.

## 4.1 Возможности авторизованного пользователя

При успешной авторизации пользователь попадает на главную страницу приложения, на которой находится основной функционал форума. При нажатии на кнопку «Create discission» происходит переход на страницу создания статьи, функционал создания статьи представлен на рисунке 4.4. Добавим заголовок, загрузим картинку, добавим теги по теме и введем текст обсуждения.

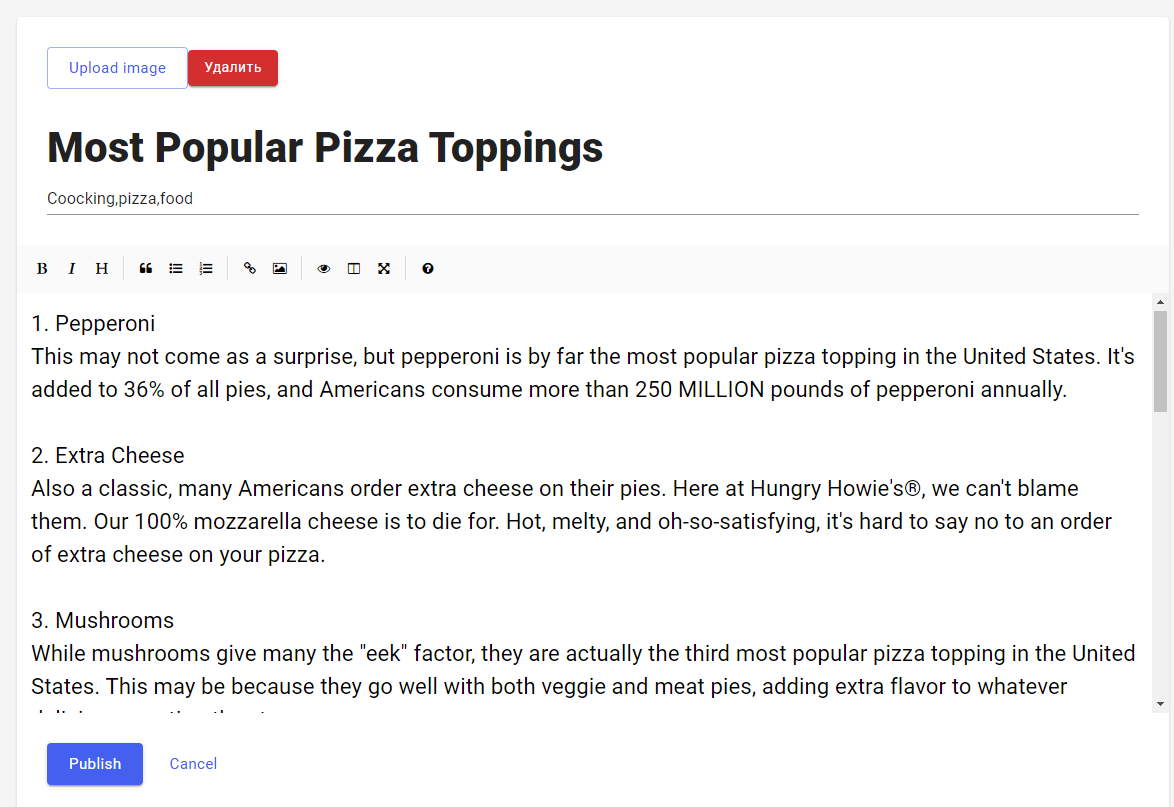


Рисунок 4.4 – Страница создание обсуждение

При нажатии на кнопку publish, обсуждение публикуется и его можно увидеть на главной странице приложения, представлена на рисунке 4.5.

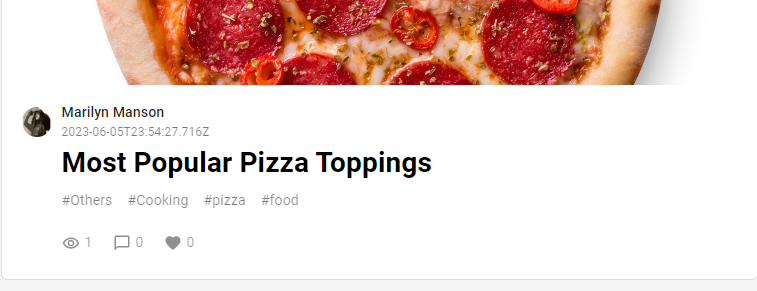


Рисунок 4.5 — Добавленное обсуждение

При наведении на обсуждение появятся карандаш и крестик, отвечающие за редактирование и удаление соответственно, представлено на рисунке 4.6.

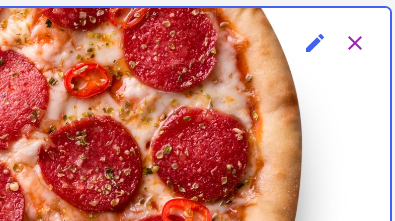


Рисунок 4.6 — Редактирование и удаление обсуждения

Для оставления комментария, нужно раскрыть обсуждение, ввести текст и нажать на кнопку send, представлено на рисунке 4.7.

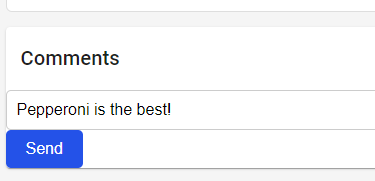


Рисунок 4.7 — Отправка комментария

Для удаления или редактирования нужно при наведении на свой комментарий нажать на появившийся значок карандаша или ведра для редактирования или удаления соответственно, наглядно представлено на рисунке 4.8.

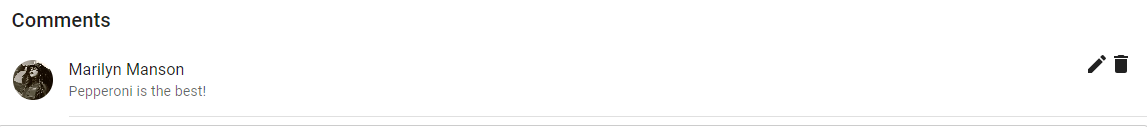


Рисунок 4.8 — Редактирование и удаление комментария

После авторизации появилось поле для отправки сообщения в групповой чат, представлено на рисунке 4.9.

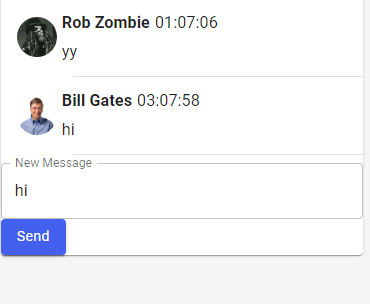


Рисунок 4.9 — Отправка сообщения в групповой чат

Отправленное сообщение сразу же отобразиться у всех подключённых пользователей, благодаря реализации на WebSocket.

## 4.2 Возможности Администратора

Роль администратора добавлена для модерации отправляемого контента, в его возможности входит удаление и редактирование чужих обсуждений, при наведении на него появляются для модерации, представлены на рисунке 4.10.

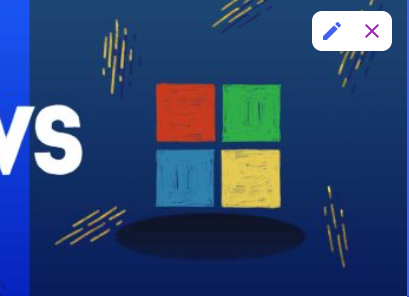


Рисунок 4.10 — Меню для редактирования и удаления чужого обсуждения

Таким же образом администратор может удалять или редактировать чужие комментарии, представлено на рисунке 4.11.

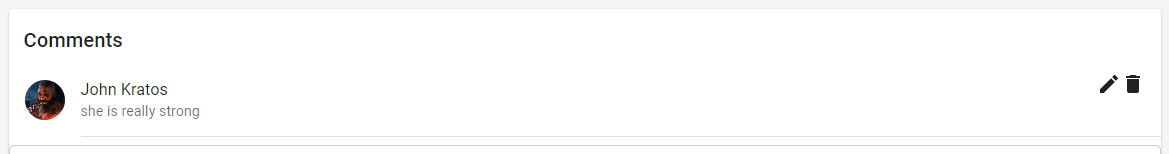


Рисунок 4.11 — Меню для редактирования и удаления чужого комментария

Роль администратора на форуме также включает в себя модерацию отправляемого контента, что позволяет ему эффективно контролировать и поддерживать качество обсуждений и комментариев. Администратор имеет возможность удаления и редактирования чужих обсуждений и комментариев для поддержания порядка и соответствия правилам форума.

При наведении на чужое обсуждение в интерфейсе администратора появляется меню для модерации, представленное на рисунке 4.10. Это меню обычно содержит опции, позволяющие администратору удалить или отредактировать обсуждение, в зависимости от конкретных требований и настроек форума.

Аналогично, для чужих комментариев также доступно меню для редактирования и удаления, как показано на рисунке 4.11. Администратор может выбрать соответствующую опцию в меню, чтобы удалить комментарий или внести необходимые изменения.

Такие функции модерации позволяют администратору быстро и эффективно реагировать на нарушения правил форума, не допуская распространения нежелательного или нарушающего контента. Кроме того, администратор может использовать свои полномочия для создания безопасной и приятной среды для всех пользователей форума, предотвращая спам и другие негативные проявления.

Заключение

В рамках данного курсового проекта было успешно разработано web-приложение форум «ItHub», которое отличается удобством использования и обеспечивает высокий уровень безопасности.

Перед началом разработки приложения был проведен аналитический обзор существующих прототипов приложений схожей тематики. Это помогло определить функциональные возможности, которые были реализованы в успешно разработанном приложении форум «ItHub».

Был разработан интуитивно понятный и простой в использовании интерфейс, который обеспечивает удобство взаимодействия пользователя с приложением.

В процессе разработки были использованы современные и популярные библиотеки, модули и фреймворки, такие как Express, Mongoose, Socket.io, React и Redux. Эти инструменты позволили полностью реализовать задуманный функционал приложения, согласно заданию.

Для обеспечения безопасности передачи данных приложение использует сгенерированные сертификаты, обеспечивая шифрованное HTTPS соединение между клиентом и сервером.

Важной составляющей разработки является взаимодействие приложения с базой данных MongoDB, которая размещена на сервере разработчиков. Это обеспечивает надежное и эффективное хранение данных приложения.

Особое внимание уделялось тестированию разработанного программного продукта. Тестирование позволило выявить и устранить возможные ошибки и недостатки в работе приложения, а также убедиться в правильности его функционирования и соответствии требованиям, указанным в техническом задании. Это позволило создать стабильное и надежное приложение, готовое к использованию.

В результате разработки были выполнены все поставленные задачи, в том числе приложение должно:

* обеспечивать возможность регистрации и авторизации;
* поддерживать роли администратора и пользователя;
* позволяет оставить комментарий под темой на определенную тематику;
* позволяет отвечать на комментарии под темами создавая тред;
* позволять пользователям создавать темы;
* реализовывать редактирование и удаление тем для их владельцев;
* предоставлять возможность поддерживать связь между пользователями форума при помощи чата;
* предоставлять возможность скрывать все темы для определенного подфорума;
* предоставлять возможность оценить тему лайком;
* предоставлять возможность подписаться на автора интересующих тем;
* предоставлять возможность поиска тематики по названию.

Список используемых источников

1. Справочник Node.js [Электронный ресурс] – Режим доступа: https://nodejsdev.ru/ – Дата доступа: 04.12.2023.
2. React – Getting Started [Электронный ресурс] – Режим доступа: https://reactjs.org/docs/getting-started.html – Дата доступа: 04.12.2023.
3. Axios [Электронный ресурс] – Режим доступа: https://axios-http.com/ru/docs/intro – Дата доступа: 05.12.2023.
4. React MobX [Электронный ресурс] – Режим доступа: https://mobx.js.org/about-this-documentation.html – Дата доступа: 05.12.2023.
5. Docker Desktop [Электронный ресурс] – Режим доступа: https://www.docker.com/products/docker-desktop/ – Дата доступа: 06.12.2023.
6. Docker Hub, PostgreSQL Image [Электронный ресурс] – Режим доступа: https://hub.docker.com/layers/library/postgres/ – Дата доступа: 06.12.2023.
7. Reddit [Электронный ресурс]. – Режим доступа: <https://www.reddit.com/> – Дата доступа: 06.02.2023.
8. 4PDA [Электронный ресурс]. – Режим доступа: <https://4pda.to/> – Дата доступа: 08.02.2023.
9. Express документация [Электронный ресурс]. – Режим доступа: <https://expressjs.com/>– Дата доступа: 11.02.2023.
10. MongoDB документация [Электронный ресурс]. – Режим доступа: <https://www.mongodb.com/docs/> – Дата доступа: 13.02.2023
11. Mongoose документация [Электронный ресурс]. – Режим доступа: https://mongoosejs.com/docs/ – Дата доступа: 22.02.2023.
12. React документация [Электронный ресурс]. – Режим доступа: <https://legacy.reactjs.org/docs/getting-started.html> Дата доступа: 24.04.2023.
13. Redux документация [Электронный ресурс]. – Режим доступа: <https://redux.js.org/introduction/getting-started> Дата доступа: 18.04.2023.
14. Уроки по React [Электронный ресурс]. – Режим доступа: <https://redux.js.org/introduction/getting-started> Дата доступа: 18.04.2023.
15. Уроки по React [Электронный ресурс]. – Режим доступа: <https://metanit.com/web/react/> Дата доступа: 18.04.2023.
16. Уроки по Redux [Электронный ресурс]. – Режим доступа: <https://redux.js.org/introduction/learning-resources> Дата доступа: 18.04.2023.
17. Уроки по Redux [Электронный ресурс]. – Режим доступа: <https://www.codecademy.com/learn/learn-redux> Дата доступа: 18.04.2023.
18. Код для новичков по Redux [Электронный ресурс]. – Режим доступа: <https://www.freecodecamp.org/news/redux-for-beginners/> Дата доступа: 18.04.2023.