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 Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

WEB-приложение «Форум»

Выполнил студент Истомин Илья Олегович

(Ф.И.О.)

Руководитель проекта асс. Бурмакова А.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. Бурмакова А.В.

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

Нормоконтролер: асс. Бурмакова А.В.

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

Курсовой проект защищен с оценкой

Минск 2021

# Реферат

ФИО

Подпись

Дата

Лист

1

Курсовой проект

Разраб.

Истомин И.О.

Мингазов А. И

Провер.

Бурмакова А.В.

Н. контр.

Утверд.

Реферат

Лит.

Листов

1

БГТУ

Пояснительная записка курсового проекта содержит 42 страницы пояснительной записки, 40 иллюстраций, 6 источников литературы, 5 приложений.

NODE.JS, MONGO DB, EXPRESS.JS, MONGOOSE, jwt authentication, JAVASCRIPT, ajax, REACT, controller/components, css, html, SOCKET.IO

Целью курсового проекта является создание веб-приложения, которое предназначено для выполнения основных функций интернет форума, а именно создание постов, просмотр постов, тем, создание комментариев, редактирование собственных постов и комментариев, общение с другими пользователями в чате, регистрация и авторизация, просмотр собственного профиля и его редактирование.

В первой главе проводится аналитический обзор прототипов по тематике курсового проекта и содержится описание технологий, использованных во время выполнения проекта.

Вторая глава посвящена процессу разработки архитектуры проекта.

В третьей главе описывается процесс разработки проекта, реализации поставленных задач.

Четвёртая глава посвящена тестированию приложения.

Содержание

ФИО

Подпись

Дата

Лист

1

Курсовой проект

Разраб.

Истомин И.О.

Мингазов А. И

Провер.

Бурмакова А.В.

Н. контр.

Утверд.

Содержание

Лит.

Листов

1

БГТУ

[Реферат 2](#_Toc72794201)

[Введение 4](#_Toc72794202)

[1 Постановка задачи 5](#_Toc72794203)

[1.1 Аналитический обзор аналогов 5](#_Toc72794204)

[1.2 Используемые технологии 7](#_Toc72794205)

[2 Разработка архитектуры проекта 9](#_Toc72794206)

[2.1 Взаимосвязь всех компонентов 9](#_Toc72794207)

[2.2 Спецификация функциональных требований 10](#_Toc72794208)

[2.3 Модель базы данных 11](#_Toc72794209)

[3 Разработка программного средства 15](#_Toc72794210)

[3.1 Проектирование серверной части приложения 15](#_Toc72794211)

[3.2 Проектирование клиентской части приложения 19](#_Toc72794212)

[3.2.1 React JS 19](#_Toc72794213)

[3.2.2 Реализация технологии AJAX 20](#_Toc72794214)

[3.2.2 Реализация технологии Socket.IO 22](#_Toc72794215)

[4 Тестирование программного средства 24](#_Toc72794216)

[Заключение 29](#_Toc72794217)

[Список литературных источников 30](#_Toc72794218)

[ПРИЛОЖЕНИЕ А 31](#_Toc72794219)

[ПРИЛОЖЕНИЕ Б 34](#_Toc72794220)

[ПРИЛОЖЕНИЕ В 35](#_Toc72794221)

[ПРИЛОЖЕНИЕ Г 36](#_Toc72794222)

[ПРИЛОЖЕНИЕ Д 38](#_Toc72794223)

# Введение

ФИО

Подпись

Дата

Лист

1

Курсовой проект

Разраб.

Истомин И.О.

Мингазов А. И

Провер.

Бурмакова А.В.

Н. контр.

Утверд.

Введение

Лит.

Листов

1

БГТУ

Интернет форум Topic Hub – это онлайн-платформа, предназначенная для обсуждения определенных тем, а также нахождения решений для определенных постов пользователей.

Создание полноценного работающего интернет форума Topic Hub – это объемный и трудоемкий процесс, который требует от разработчика огромного вложения сил, концентрации и проявления творчества. Для реализации полноценного продукта могут уйти месяцы непрерывной работы, в виду отсутствия этого ресурса, я решил создать свой интернет форум Topic Hub, который позволит мне с легкостью оставлять посты и находить ответы на поставленные вопросы.

Целью моего курсового проекта является разработка веб-приложения «Topic Hub», интернет форум, который позволит создавать посты, комментарии, а также общаться с пользователями в чате.

Основными задачами курсовой работы являются:

* провести аналитический обзор литературы;
* разработать модель базы данных;
* научиться разрабатывать приложение с использованием многоуровневой архитектуры;
* провести тестирование;
* написать руководство пользователя.

Node.js – программная платформа, основанная на движке V8 (транслирующем JavaScript в машинный код), превращающая JavaScript из узкоспециализированного языка в язык общего назначения.

В соответствии с заданием курсового проекта для создания модели базы данных используется документно-ориентированная база данных.

MongoDB — документно-ориентированная система управления базами данных, не требующая описания схемы таблиц. Считается одним из классических примеров NoSQL-систем, использует JSON-подобные документы и схему базы данных.

Сильными сторонами MongoDB считаются:

* кроссплатформенность;
* документы вместо строк;
* отсутствие жесткой схемы базы данных и в связи с этим потребности при малейшем изменении концепции хранения данных пересоздавать эту схему значительно облегчают работу с базами данных MongoDB и дальнейшим их масштабированием.

# 1 Постановка задачи

ФИО

ПодписьПодпись

ДатаДата

Лист

1

Курсовой проект

Разраб. Разраб.

Истомин И.О.

Пуцик Н. Ю.

Провер. Провер.

Бурмакова А.В.

Нистюк О. А.

Н. контр. контр.

Утверд. Утверд.

Постановка задачи

Лит.

Листов

1

БГТУ

В соответствии с заданием курсового проекта следует разработать web-приложение, которое должно реализовывать свои функции, а именно позволять пользователю регистрироваться и авторизоваться, добавлять, удалять, редактировать посты, комментарии, добавлять, изменять, удалять изображения. Для того, чтобы сформировать окончательные требования к проектируемому программному средству, сначала рассмотрим прототипы программных средств того же направления.

## 1.1 Аналитический обзор аналогов

На сегодняшний день существует множество подобных приложений, однако функционал их шире. Связано это, в первую очередь, с тем, что каждое приложение индивидуальное и заточено под конкретный функционал. На сегодняшний день, самым крупным и популярным считается Ответы *Mail.Ru* [1]. Ответы *Mail.Ru* – это пространство, в котором вы можете задавать вопросы и получать настоящие ответы от живых людей, это мгновенная помощь по любому вопросу от тысяч пользователей онлайн. Благодаря Ответам вы можете узнать мнение миллионов пользователей проекта Ответы *Mail.ru* по любому насущному вопросу, публиковать ответы на вопросы других пользователей, комментировать их, выбирать лучшие ответы на чужие вопросы, а также получать моментальные уведомления об ответах на свои вопросы и других действиях (например, оценках ответов).

Основными функциями является добавление, редактирование, удаление вопросов, комментариев, поиск вопросов. На рисунке 1.1 изображен главный экран веб-приложения.

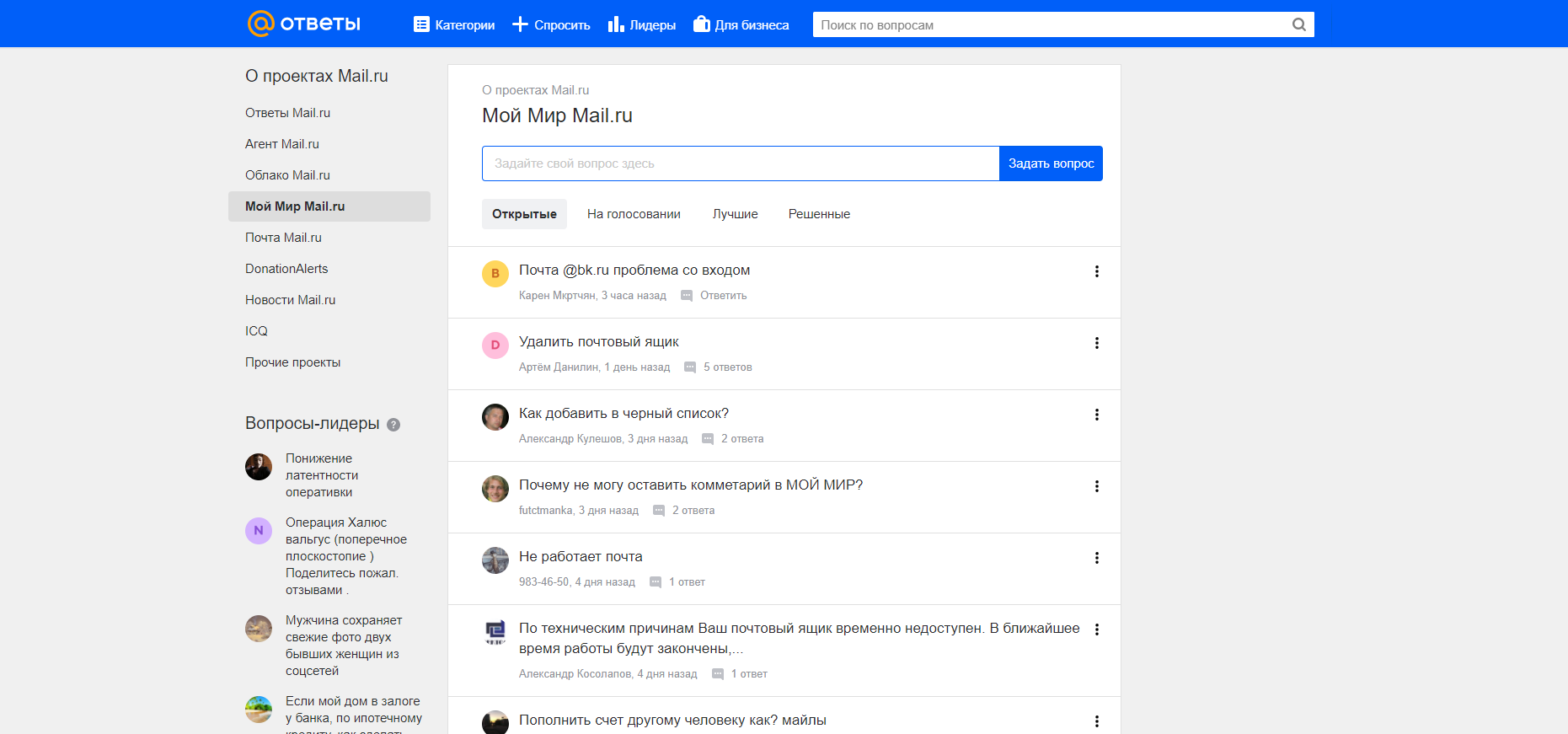


Рисунок 1.1 – Главная страница веб-приложения Ответы *Mail.ru*

На начальной странице присутствуют все необходимые элементы. Отображена и интуитивно понятна иерархия веб-сайта. Меню организовано горизонтально в шапке сайта. Навигация работает одинаково на всех страницах

Также есть возможность найти интересуемый вопрос (рисунок 1.2).

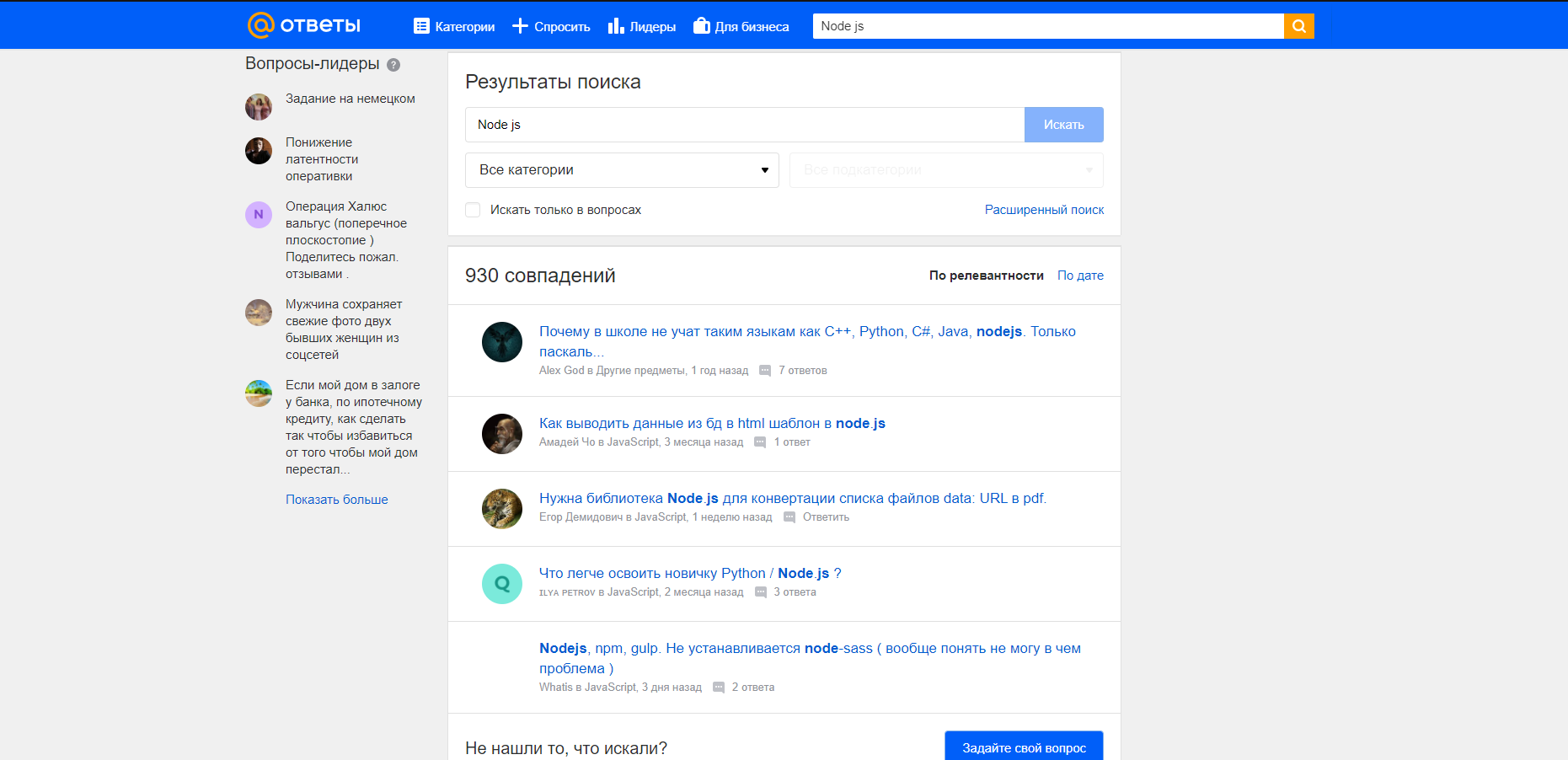


Рисунок 1.2 – Раздел «Результаты поиска» веб-приложения Ответы *Mail.ru*

Просмотреть интересующий вопрос возможно при клике на иконку данного вопроса (рисунок 1.3).

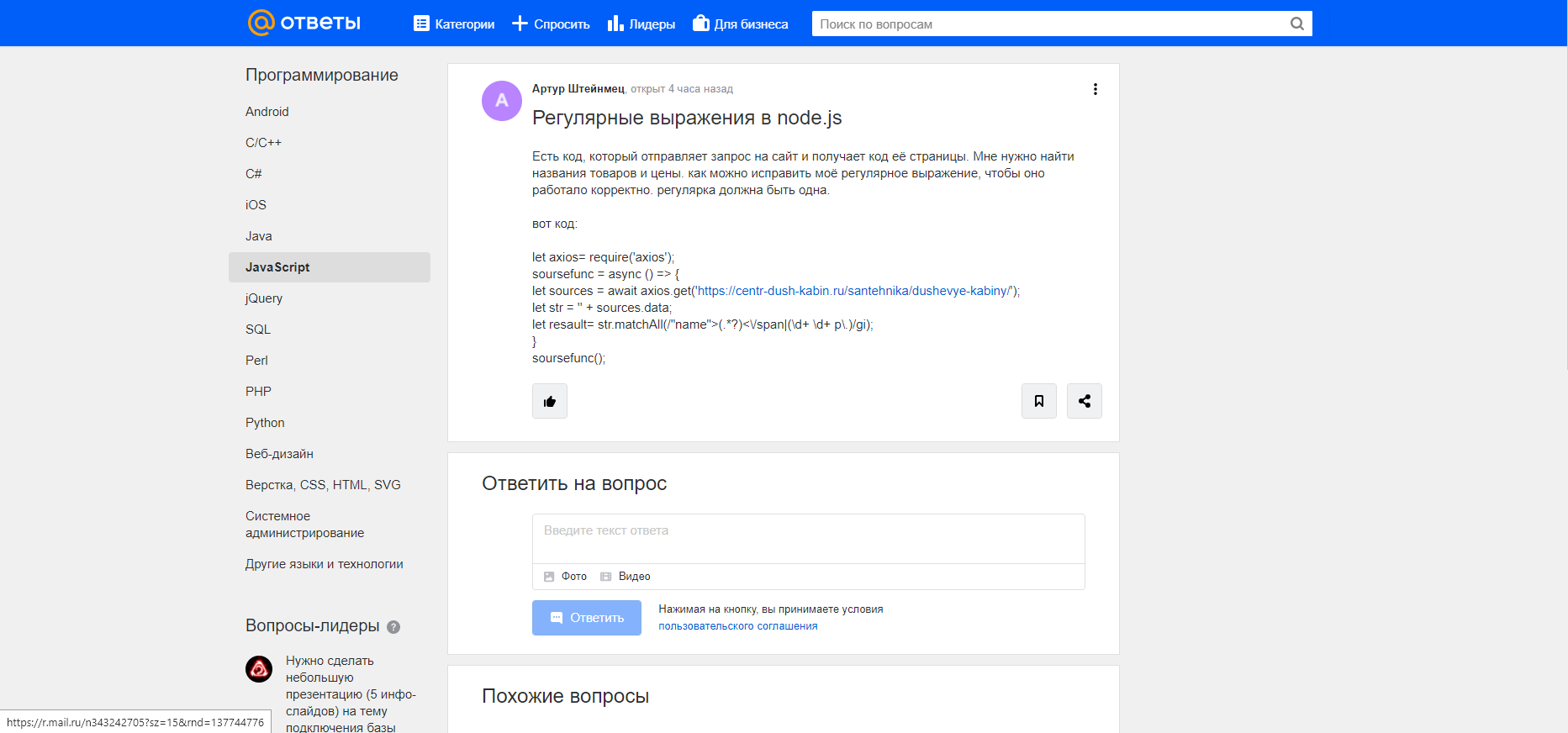


Рисунок 1.3 – Просмотр вопроса Ответы *Mail.ru*

Достоинствами данного сайта является интуитивно понятный интерфейс, адаптивность под различные устройства и разрешения, удобная навигация по сайта. Ясна его цель и содержание с первой страницы.

В качестве недостатка можно выделить отсутствие чата, так как бывают ситуации, когда удобнее будет задать вопрос с онлайн чате, а не создавать отдельный пост-вопрос на платформе.

Проанализировав аналог, можно отметить ожидаемую схожесть в их функциональности и интерфейсе.

Исходя из полученных результатов, можно сформулировать основные функциональные требования курсовой работы:

* регистрация и авторизация пользователей;
* добавление, удаление, редактирование постов, комментариев;
* работа с изображениями.

**1.2 Используемые технологии**

*Node.js* – программная платформа для разработки серверных web-приложений на основе движка *JavaScript Chrome V8*. Является средой исполнения приложений на JavaScript. Также ориентирована на события, поддерживает асинхронность и является однопоточной. Также *node.js* является открытым проектом [2].

*Express* – это минималистичный и гибкий веб-фреймворк для приложений *Node.js*, предоставляющий обширный набор функций для мобильных и веб-приложений. Он использует модуль *http*, а также в него включены готовые абстракции для упрощения создания сервера и серверной логики. Express не мешает общей производительности приложения, т.к. представляет собой тонкий слой основных функций веб-приложений. Также в нём удобно реализована маршрутизация.

Для проектирования базы данных будет использоваться система управления базами данных *MongoDB*. *MongoDB* это кросс-платформенная, документно-ориентированная база данных, которая обеспечивает высокую производительность и лёгкую масштабируемость. В основе данной БД лежит  концепция коллекций и документов. База данных представленна в виде физического хранилища коллекций. Каждая БД имеет свой собственный набор файлов в файловой системе. Обычно, один *MongoDB* сервер имеет несколько БД. Коллекция – это группа документов *MongoDB*. Является эквивалентом простой таблицы в реляционной базе данных. Коллекция помещена внутри одной БД. Документ в коллекции моет иметь различные поля. Чаще всего, все документы в коллекции созданы для одной, либо относящихся друг ко другу целей. Документ – это набор пар “ключ – значение”. Документ имеет динамическую схему. Это означает, что документ в одной и той же коллекции не обязан иметь один одинаковый набор полей или структуру, а общие поля в коллекции могут иметь различные типы данных. [3].

*Mongoose*  – специальная ODM-библиотека (*Object Data Modelling*) для работы с *MongoDB*, которая позволяет сопоставлять объекты классов и документы коллекций из базы данных. Грубо говоря, *Mongoose* работает подобно инструментам *ORM*.  [4].

*React* – это библиотека *JavaScript*, которая используется для создания пользовательского интерфейса. *React* может использоваться для разработки одностраничных и мобильных приложений. Его цель — предоставить высокую скорость, простоту и масштабируемость. В качестве библиотеки для разработки пользовательских интерфейсов *React* часто используется с другими библиотеками, такими как *MobX*, *Redux* и *GraphQL*. В *React* все компоненты пользовательского интерфейса могут быть выражены в виде наборов чистых функций. [5].

Также в данном проекте использовался *Socket.IO* – это *JavaScript*-библиотека для веб-приложений и обмена данными в реальном времени. *Socket.IO* главным образом использует протокол *WebSocket*, но если нужно, использует другие технологии, например *Flash Socket, AJAX Long Polling* и т. д. Помимо того, что Socket.IO может быть использован как оболочка для *WebSocket*, он содержит много других функций, включая вещание на несколько сокетов, хранение данных, связанных с каждым клиентом, и асинхронный ввод/вывод.

Для просмотра состояний объектов использовался контейнер *Redux*. [*Redux*](https://redux.js.org/) – это контейнер с предсказуемым состоянием для приложений *JavaScript* и очень ценный инструмент для управления состоянием приложения. Также это популярная библиотека для управления состоянием в приложениях *React*, но ее можно использовать и с *Angular*, *Vue.js* и со старым добрым *JavaScript*.

Главная сложность *Redux* заключается в том, что многие разработчики не понимают, когда его использовать. Чем больше и сложнее становится приложение, тем выше вероятность того, что *Redux* будет вам полезен. Если вы начинаете работать над приложением и ожидаете, что в скором будущем оно существенно вырастет, вы можете сразу использовать *Redux*: так по мере изменения и масштабирования приложения вы сможете легко внедрять все новые функции, избегая рефакторинга большого количества готового кода.

Иногда объём работы, за выполнение которой отвечает отдельный компонент, оказывается слишком большим, компоненту приходится решать слишком много задач. Использование паттерна *Container*/*Component* позволяет отделить логику функционирования приложения от логики формирования его визуального представления. Это позволяет улучшить структуру приложения, разделить ответственность за выполнение различных задач между разными компонентами.

# Разработка архитектуры проекта

ФИО

ПодписьПодпись

ДатаДата

Лист

1

Курсовой проект

Разраб. Разраб.

Истомин И.О.

Пуцик Н. Ю.

Провер. Провер.

Бурмакова А.В.

Нистюк О. А.

Н. контр. контр.

Утверд. Утверд.

Разработка архитектуры проекта

Лит..

Листов

1

БГТУ

Разработка архитектуры проекта – важная задач в процессе работы над приложением, потому что в зависимости от неё определяется уровень зависимости компонентами приложения, и насколько легко можно будет его расширить.

## Взаимосвязь всех компонентов

При разработке курсового проекта весь процесс был разбит на 2 следующие составляющие:

– сервер;

– клиент.

Структура программного средства представлена на рисунке 2.1. Как мы можем увидеть в проекте имеется четыре главные директории: *services*, *models*, *controllers* и *routes*.

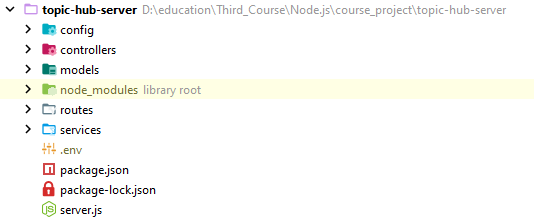


Рисунок 2.1 – Структура программного средства

Директория *models* содержит в себе файлы, описывающие структуру данных, которые мы использовали в приложении. Директория *routes* содержит файлы, которые отвечают за обработку логики запроса пользователя. Директория *controllers* содержит в себе файлы, обрабатывающие входящие *http*-запросы и вызывающие определенные сервисы. Директория *services* содержит в себе файлы, которые используют для обработки *http*-методов модели и представления и отправляют в ответ клиенту некоторый результат.

## Спецификация функциональных требований

Рассмотрим возможности, которые предоставляет разработанное программное средство. Для наглядности демонстрации использовались *UML*-диаграммы.

*UML* — уникальный язык моделирования (*Unified Modeling Language*) – это система обозначений, которую можно применять для объектно-ориентированного анализа и проектирования. Его можно использовать для визуализации, спецификации, конструирования и документирования программных систем. [6]

На рисунке 2.2 показана *UML*-диаграмма, отображающая возможности приложения с точки зрения неавторизованного пользователя.

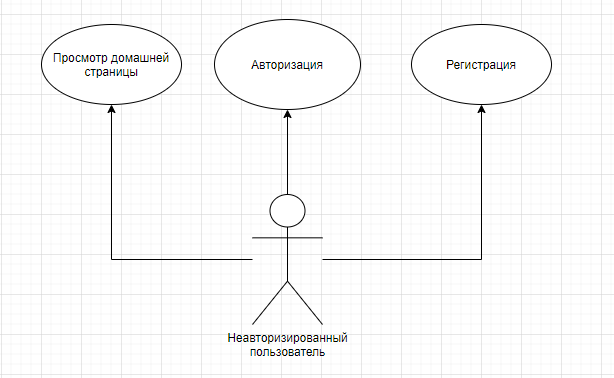


Рисунок 2.2 – *UML* диаграмма вариантов использования приложения для неавторизованного пользователя

Неавторизованный пользователь имеет массу возможностей представлены на рисунке 2.2.

Если пользователь авторизуется, то будет иметь возможности, представленные на рисунке 2.3.

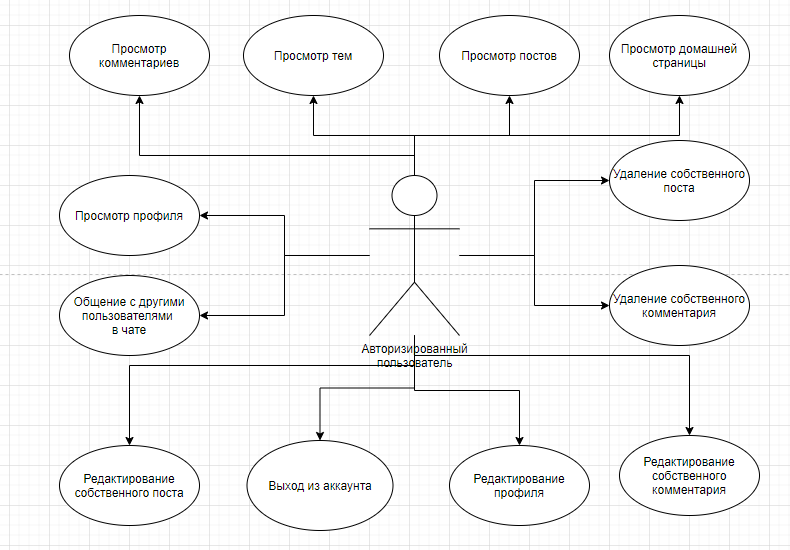


Рисунок 2.3 – *UML* диаграмма вариантов использования приложения для авторизованного пользователя

После авторизации пользователю открывается доступ к профилю, темам, постам и комментариям, так же пользователь может общаться в другими пользователями в чате.

## Модель базы данных

Для работы с базой данных было принято решение использовать *MongoDB*. База данных, разработанная для этого курсового проекта, состоит из 4 сущностей:

– users – сущность, в которой описана модель пользователя;

– topics – сущность, в которой описана модель темы;

– posts – сущность, в которой описана модель поста. Также данная модель связана с моделью topics;

– comments – сущность, в которой описана модель комментария. Также данная модель связана с моделью posts.

Подробное описание сущностей располагается под рисунком 2.4.

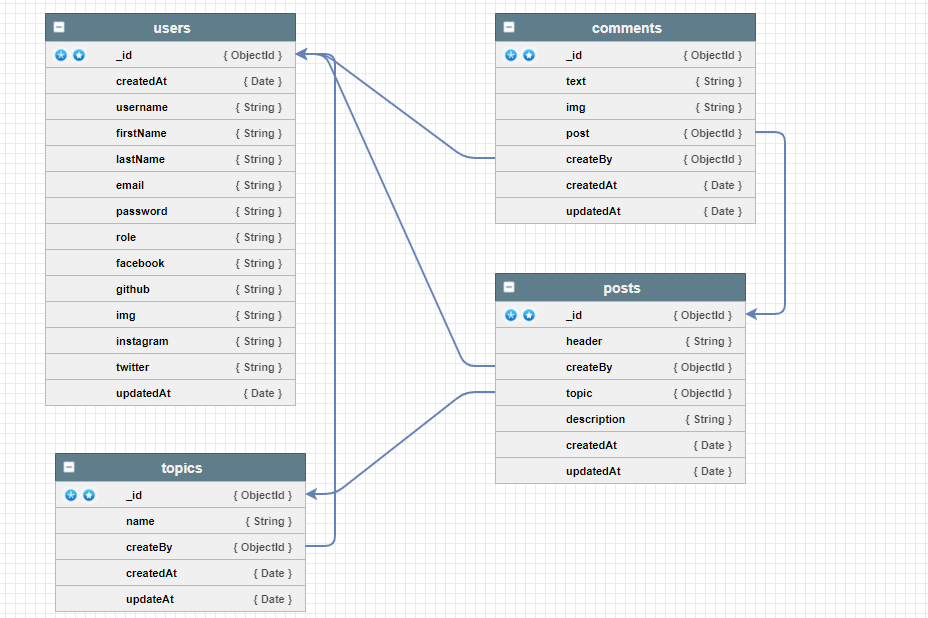


Рисунок 2.4 – Набор коллекций базы данных проекта

Сущность users включает в себя следующие поля:

– \_*id* – идентификатор пользователя;

– *createdAt* – время создания документа;

– *username* – логин пользователя;

– *firstName* – имя пользователя;

– *lastName* – фамилия пользователя;

– *email* – электронная почта пользователя;

– *password* – пароль пользователя;

– *role* – роль пользователя;

– *facebook* – логин от аккаунта *facebook* пользователя;

– *github* – логин от аккаунта *github* пользователя;

– *img* – изображение пользователя;

– *instagram* – логин от аккаунта *instagram* пользователя;

– *twitter* – логин от аккаунта *twitter* пользователя;

– *updatedAt* – время обновления документа.

Схема сущности users представлена на рисунке 2.5.

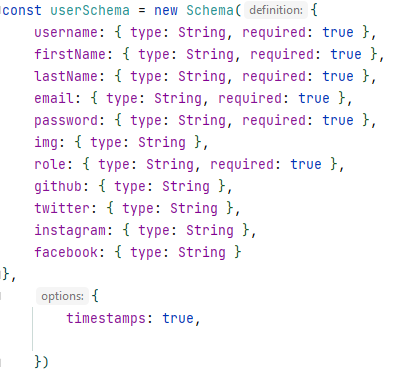


Рисунок 2.5 – Схема сущности *users*

Сущность posts включает в себя следующие поля:

– \_*id* – идентификатор поста;

– *header* – заголовок поста;

– *createBy*– идентификатор пользователя;

– *createdAt* – время создания документа;

– *updatedAt* – время обновления документа.

Схема коллекции posts представлена на рисунке 2.6.

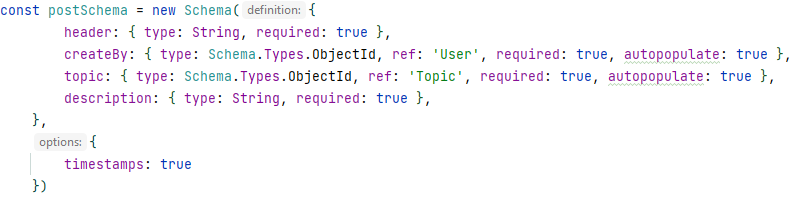


Рисунок 2.6 – Схема сущности *posts*

Сущность *topics* включает в себя следующие поля:

– \_*id* – идентификатор темы;

– *name* – название темы;

– *createBy* – идентификатор пользователя;

– *createdAt* – время создания документа;

– *updatedAt* – время обновления документа.

Схема сущности *topics* представлена на рисунке 2.7.

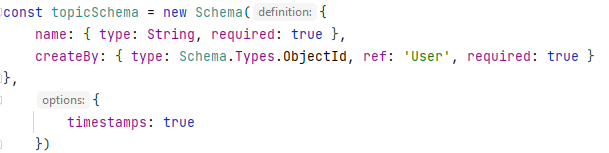


Рисунок 2.7 – Схема сущности *topics*

Сущность comments включает в себя следующие поля:

– \_*id* – идентификатор комментария;

– *text* – текст комментария;

– *img* – изображения для комментария;

– *post* – идентификатор поста;

– *createBy* – идентификатор пользователя;

– *createdAt* – время создания документа;

– *updatedAt* – время обновления документа.

Схема сущности comments представлена на рисунке 2.8.

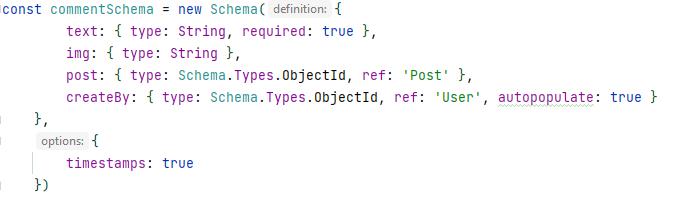


Рисунок 2.8 – Схема сущности *comments*

# Разработка программного средства

ФИО

Подпись

Дата

Лист

1

Курсовой проект

Разраб.

Истомин И.О.

Провер.

Бурмакова А.В.

Н. контр.

Утверд.

Разработка программного средства

Лит.

Листов

1

БГТУ

При проектировании программного средства в качестве программной платформы для сервера был выбран *Node.js.* Основной веб-фреймворк для приложения – *Express*.*js*.

## Проектирование серверной части приложения

Маршрутизация в приложении организована с помощью роутеров. Основными являются *AuthRouter*, *TopicRouter*, *CommentRouter*, *UserRouter*, *PostRouter*. Для этого используется *Express*. Ниже на рисунке 3.1 представлено добавление роутеров с учётом их расположения в директориях проекта.

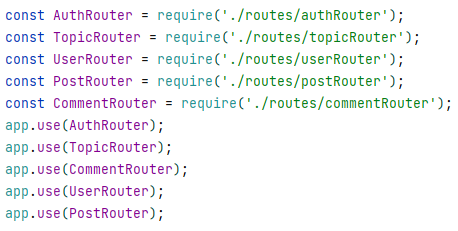


Рисунок 3.1 – Список роутеров в проекте

В некоторых маршрутах подключена *middleware* *passportJS*, использована *JWTStrategy* и *LocalStrategy*, представленная на рисунке 3.2. Она предназначена для проверки аутентифицированного пользователя. Главной является стратегия *JWTStrategy*, которая проверяет и генерирует валидный *JWT* *Token*. *LocalStrategy* позволяет проверить наличие и идентифицировать пользователя, принимая его логин и пароль.

Благодаря данным стратегиям можно рассчитывать, что авторизация пользователя пройдёт без ошибок





Рисунок 3.2 – *Middleware* функция *passportJS*

Реализация модуля *postRouter* представлена на рисунке 3.3. Данные роуты вызывают сервисы, которые выполняют свою логику и проверяют *JWT* *Token*.

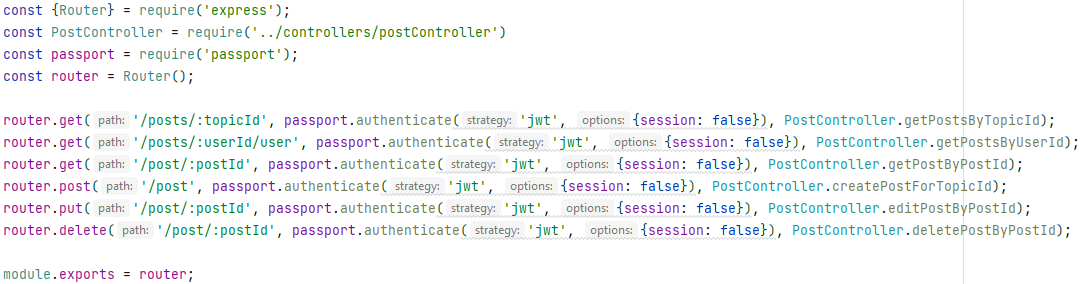


Рисунок 3.3 – Модуль *postRouter*

Ниже приведён код, обрабатывающий авторизацию пользователей. Реализовано это в роутере *authRouter*.*js*, код обрабатывает метод *post*, представленный на рисунке 3.4.



Рисунок 3.4 – Реализация авторизации пользователя

Авторизация происходит путем проверки введенной логина и пароля, которые передаются в теле запроса.

Сервер производит проверку на наличие такого логина в базе данных и сравнивает захешированные с помощью библиотеки *bcrypt* пароли на совпадение между собой. В случае, если такой пользователь существует и пароль совпадает с паролем, находящемся в базе данных, то формируется *JWT* *токен*, с помощью которого будет происходить дальнейшая авторизация.

После успешно выполненной авторизации, нас перенаправляет на страницу профиля.

Рассмотрим сервис *getUserByIdService*, данный сервис принимает входящие данные пользователя и производит операции с базой данных, извлекается вся необходимая информация о пользователе, если такого пользователя нет, то на сторону клиента будет отправлено сообщение об ошибке, код представлен на рисунке 3.5.



Рисунок 3.5 – Сервис *getUserByIdService*

Когда пользователь хочет получить данные о себе, посылается запрос на сервер и вызывается метод *findById*, который извлекает всю необходимую информацию о пользователе.

Необходимо проверить, что данный пользователь существует. Если все в порядке, то сервис вернет информацию о данном пользователе. Код данной функции находится в приложении Б.

Для редактирования поста использовался сервис *editPostByPostIdService*. Данный сервис получает *postId* и обновляет определенный пост, рисунок 3.6



Рисунок 3.6 – Сервис *editPostByPostIdService*

## Проектирование клиентской части приложения

Клиентская часть приложения была написана с использованием библиотеки *React JS*, а также *Socket IO* для общения пользователей в реальном времени.

## 3.2.1 React JS

*React* ­– это библиотека *JavaScript*, которая используется для создания пользовательского интерфейса. Вся клиентская часть представлена в двух директориях, директория *public*, содержащая файлы стилей, *JavaScript* файлы и картинки и *src*, где хранятся все ключевые компоненты клиентской части.

*React*-разработка заключается в описании того, что нужно вывести на страницу (а не в составлении инструкций для браузера, посвящённых тому, как это делать). Это, кроме прочего, означает значительное сокращение объёмов шаблонного кода. В составе *Angular*, с другой стороны, есть средства командной строки, которые генерируют шаблонный код компонентов. Не кажется ли это немного не тем, чего можно ждать от современных инструментов разработки интерфейсов? Фактически, речь идёт о том, что в *Angular* так много шаблонного кода, что для того, чтобы его генерировать, даже создано специальное средство.

В *React*, приступая к разработке, просто начинают писать код. Тут нет шаблонного кода компонентов, который нужно как-то генерировать. Конечно, перед разработкой нужна некоторая подготовка, но, когда дело доходит до компонентов, их можно описывать в виде чистых функций. На рисунке 3.7 приведено содержимое папки *src*.

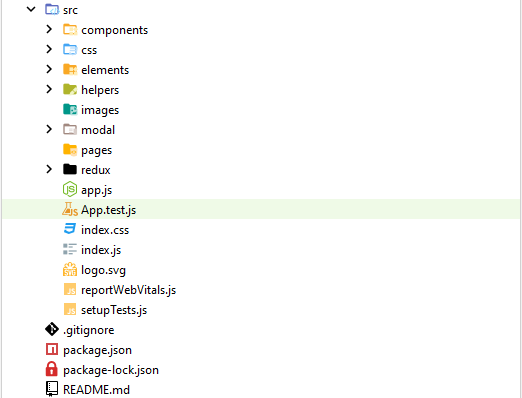


Рисунок 3.7 – Содержимое директории *src*

На рисунке 3.8 приведено содержимое директории *components*, где расположены все компоненты веб-приложения.

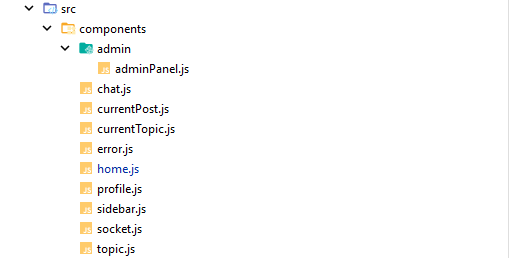


Рисунок 3.8 – Содержимое директории *components*

На рисунке 3.9 представлен пример компонента страницы тем. В случае успешной проверки токена, пользователю будет доступна страница тем, иначе будет отображен компонент *Error* с ошибкой. В данном компоненте производится извлечение тем и их отображение на странице. Также реализация страниц *Error*, *CurrentTopic*, *CurrentPost* будет отображена в приложениях В, Г, Д соответственно.



Рисунок 3.9 – Пример компонента страницы тем

Во всех остальных представлениях, способ отображения схож с примером, но кроме, логики реализованной в контролерах, добавляется *JavaScript* код реализующий дополнительный функционал.

## 3.2.2 Реализация технологии AJAX

*AJAX* – технология обращения к серверу без перезагрузки страницы. За счёт этого уменьшается время отклика веб-приложения.

Несмотря на то, что в названии технологии присутствует буква X (от слова *XML*), использовать *XML* вовсе не обязательно. Под *AJAX* подразумевают любое общение с сервером без перезагрузки страницы, организованное при помощи *JavaScript*.

*Fetch* – это новый API, основанный на промисах. Он позволяет выполнять запросы *Ajax* и при этом избежать всей работы, связанной с *XMLHttpRequest*.

Условно, все *JavaScript* файлы можно поделить на две группы, первая группа – это файлы, относящиеся к конкретным представлениям, вторая группа – это файлы, содержащие общую логику, на которую ссылаются другие компоненты, к таким файлам относятся *common*.*js* и *clientSocket*.*js*.

На рисунках 3.10 представлен пример запроса поиска пользователя при помощи технологии *AJAX fetch API.*



Рисунок 3.10 – Пример содержимого *JavaScript* запроса поиска пользователя

После отправки запроса, клиент получит данные без перезагрузки страницы. Исходя из этого можно делать несколько запросов на одной странице. Это позволит получать некоторые данные быстрее, чтобы обеспечить постепенное отображение страницы, а не ждать, пока сформируется один большой ответ от сервера.

Функции авторизации пользователя представлена на рисунке 3.11.



Рисунок 3.11 – Функции авторизации пользователя

После того, как клиент получил ответ, отрабатывает функция авторизации, которая в случае некорректных данных вернет ошибку пользователю. Если данные верны, вызывается функция *then*(), при получении положительного ответа, данная функция заносит *JWT* токен в локальное хранилище пользователя, чтобы клиенту не потребовалась новая авторизация и он мог посылать дальнейшие запросы с данным токеном авторизации.

## 3.2.2 Реализация технологии Socket.IO

*Socket.IO* относится и к клиентской части и к серверной, поэтому эта технология вынесена в отдельный раздел. *Socket.IO* – *JavaScript*-библиотека для веб-приложений и обмена данными в реальном времени. Состоит из двух частей: клиентской, которая запускается в браузере (рисунок 3.12) и серверной для *Node.js.* Оба компонента имеют похожее *API*. Подобно *Node.js,* *Socket.IO* событийно-ориентированный.

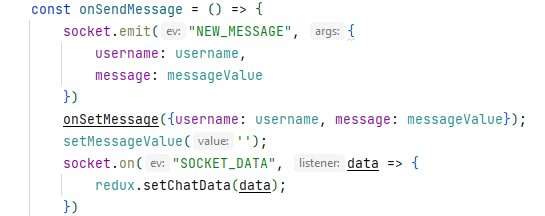


Рисунок 3.12 – Клиентская часть *Socket.IO*

Серверная часть *Socket.IO* для *Node.js* представлена на рисунке 3.13.



Рисунок 3.13 – Серверная часть *Socket.IO*

С помощью *Socket.IO* можно реализовать обсуждение определенных тем с иными пользователями в реальном времени. Изображения страниц отображены в приложении А.

ФИО

Подпись

Дата

Лист

1

Курсовой проект

Разраб.

Истомин И.О.

Провер.

Бурмакова А.В.

Н. контр.

Утверд.

Тестирование

Лит.

Листов

8

БГТУ

# Тестирование программного средства

Тестирование программного обеспечения – проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом. Для предотвращения возникновения исключительных ситуаций, которые препятствуют нормальной работе пользователя, их необходимо обработать. Функциональное тестирование проводится для проверки качества программного продукта, а точнее того, какие функции реализованы и насколько верно это сделано. Иными словами, это способ узнать, сможет ли разработанное ПО при определённых условиях выполнять необходимые задачи. В реализуемом мной программном средстве предусмотрены механизмы, которые направлены на их предотвращение.

Рассмотрим вход пользователя в систему. Приложение проверяет корректно ли введены поля и выводит соответствующее сообщение, представлено на рисунке 4.1

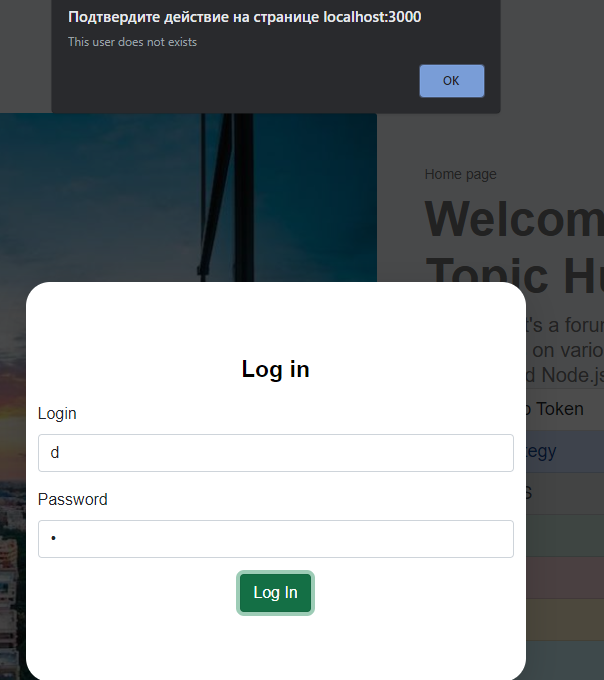


Рисунок 4.1 – Ошибка некорректно введенных полей

Все поля в окнах авторизации и регистрации являются обязательными к заполнению. Кроме этого поля также проверяются на корректное заполнение, пример на рисунке 4.2. Примером такого валидируемого поля может являться поле *password* в окне регистрации.

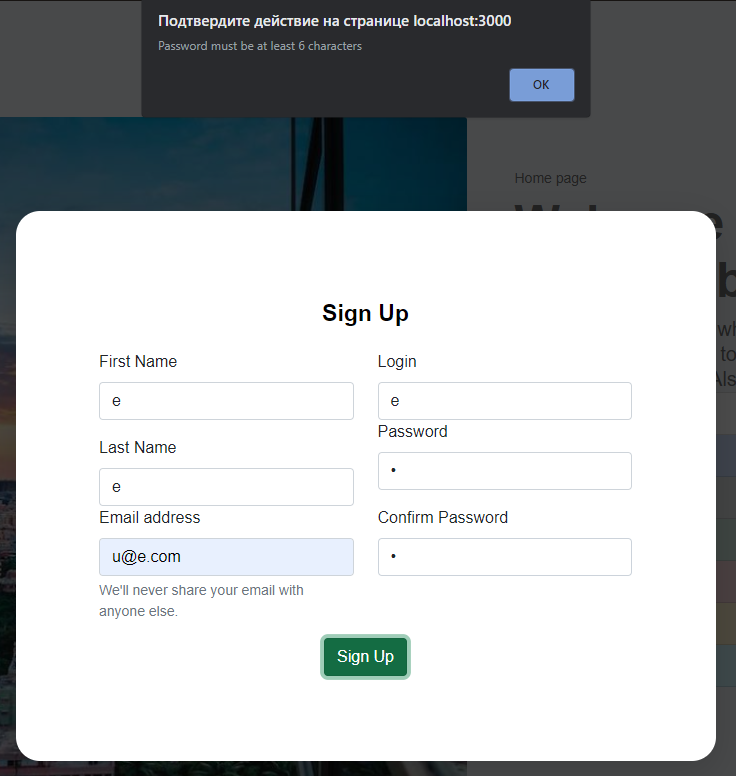


Рисунок 4.2 – Проверка корректности пароля

Для проверки существующего пользователя или верного пароля, клиент отправляет запрос на сервер. В данном запросе содержится информация, что ввел пользователь. В случае неверного пароля, сервер ответит ошибкой.

После того, как пользователь авторизовался, открывается профиль пользователя, показанный на рисунке 4.3, со всеми данными о пользователе.

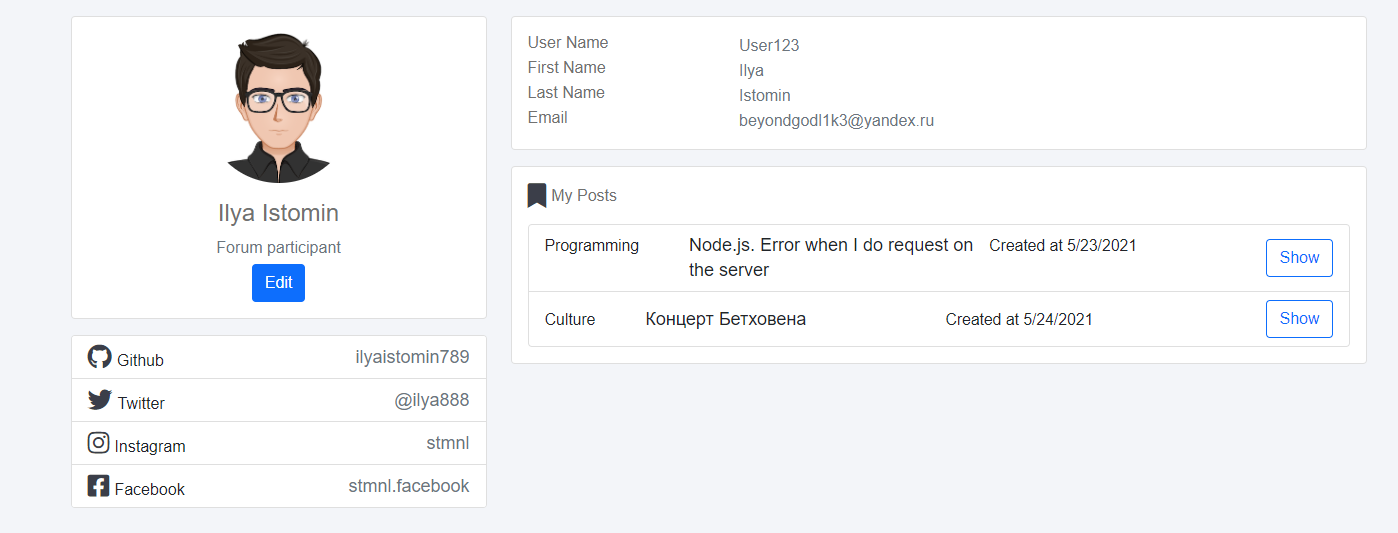


Рисунок 4.3 – Профиль пользователя

Пользователю доступна возможность просмотра своих постов, а также редактирование своего профиля

Редактирование профиля отображено на рисунке 4.4

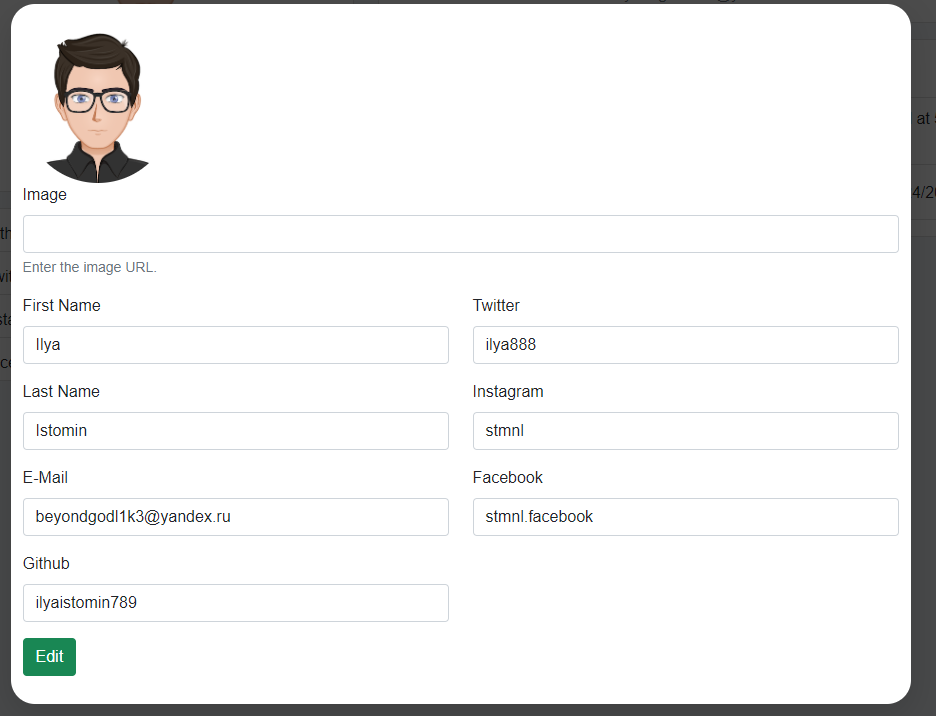


Рисунок 4.4 – Редактирование профиля

Просмотр постов изображен на рисунке 4.5

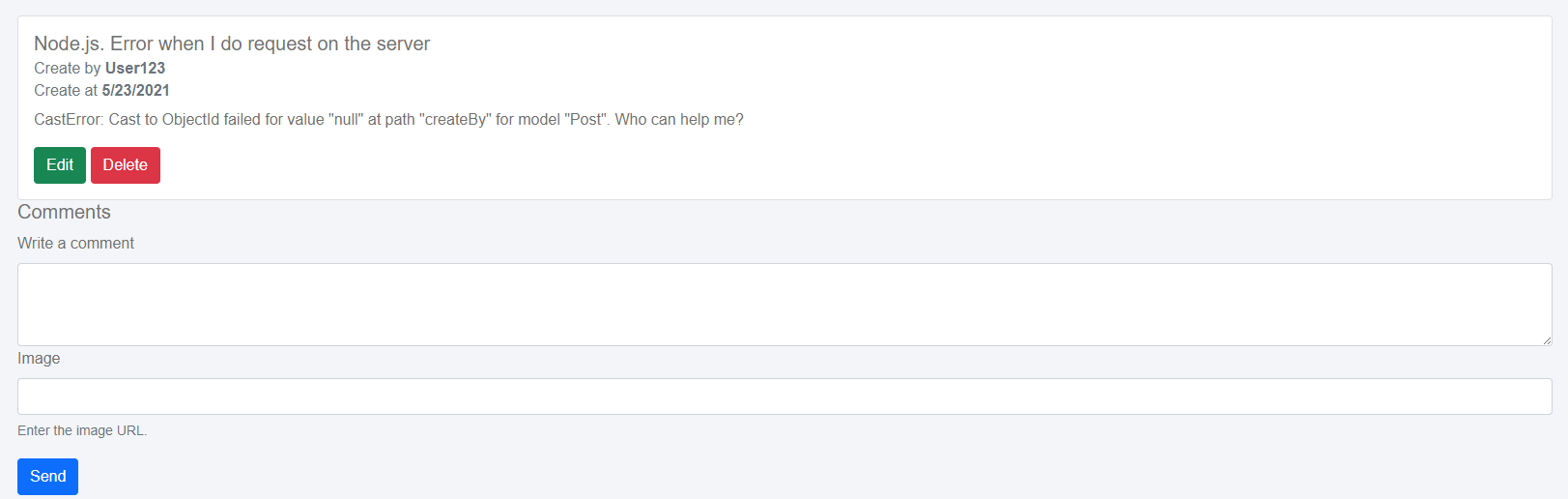


Рисунок 4.5 – Просмотр постов

На рисунке 4.6 отображена главная страница приложения. Пользователь может наблюдать информацию о проекте, а также может перемещаться по страницам с помощью вертикального навигационного бара.

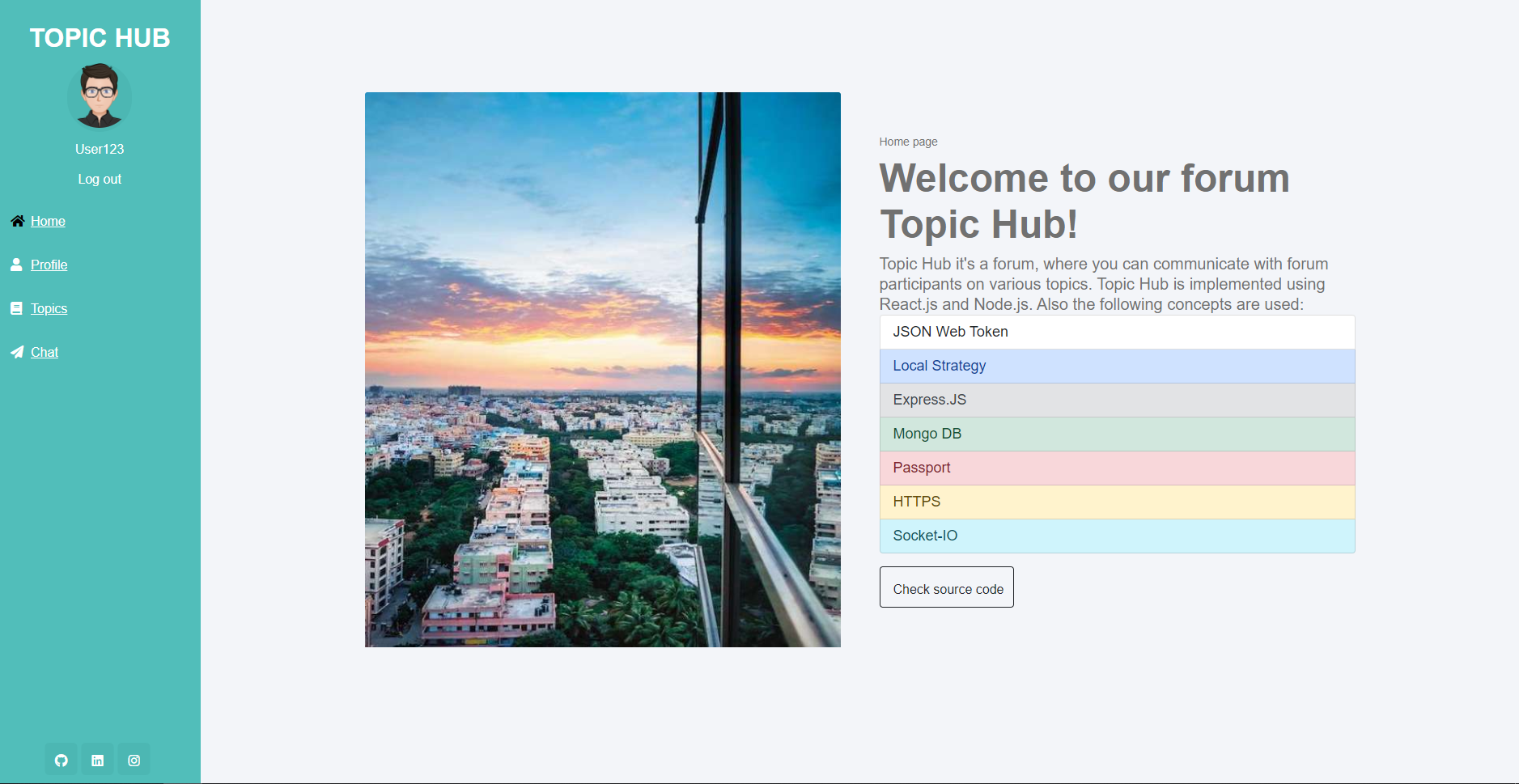


Рисунок 4.6 – Главная страница приложения

На рисунке 4.7 отображена страница тем, которые можно обсудить с другими пользователями.

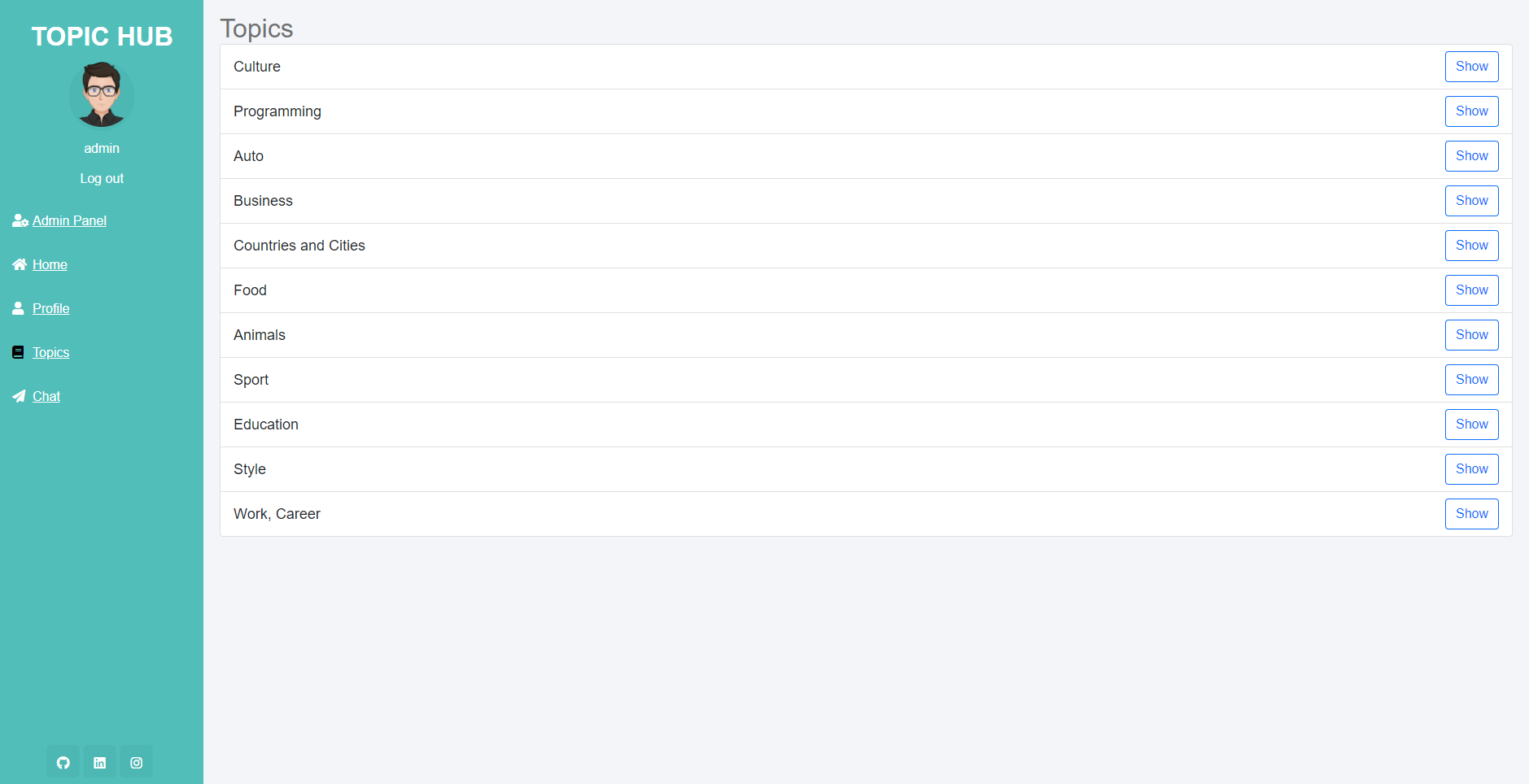


Рисунок 4.7 – Страница тем

На рисунке 4.8 отображена страница определенного поста, в которой можно оставлять, редактировать, удалять комментарии, а также редактировать, удалять пост, если он принадлежит данному пользователю.

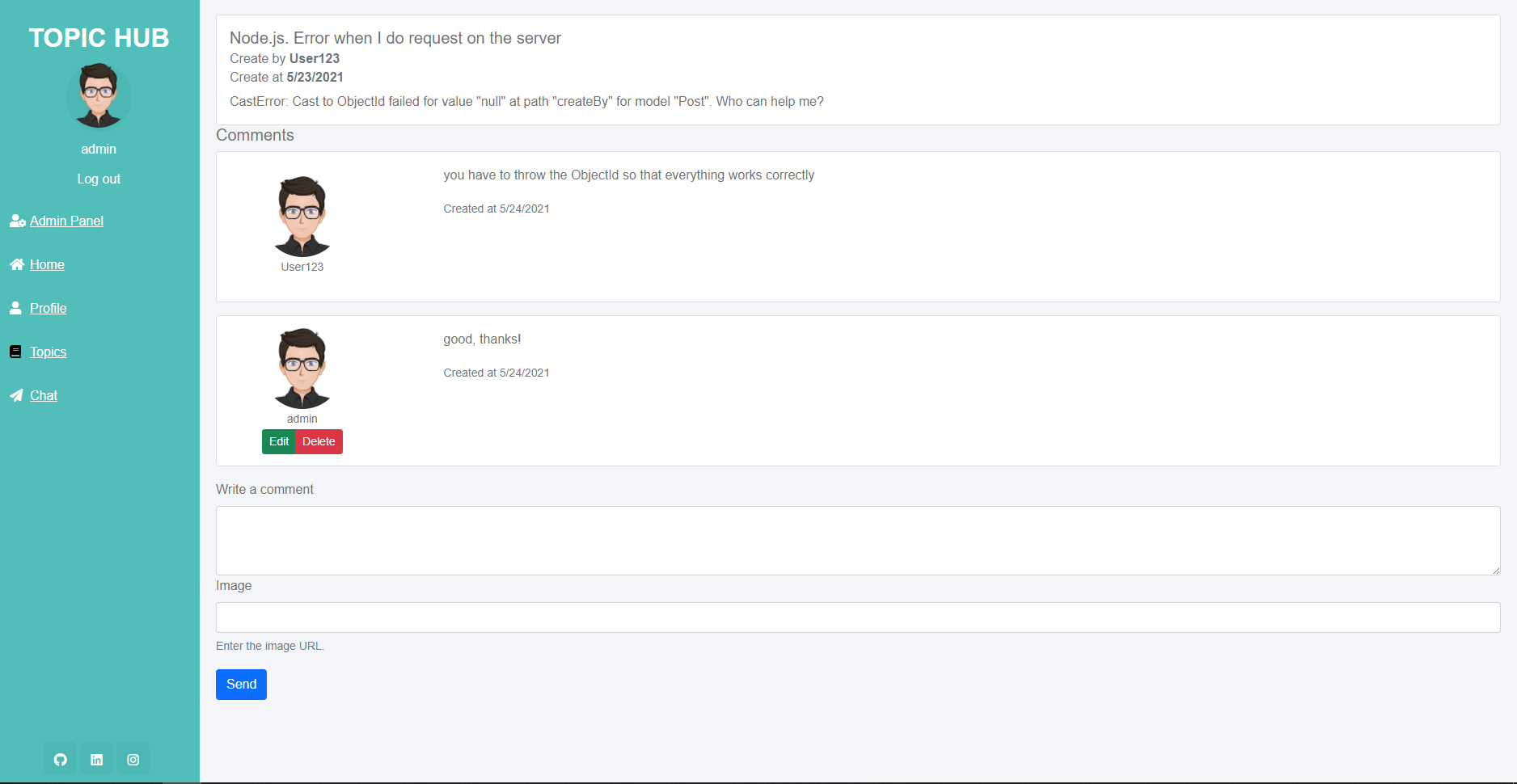


Рисунок 4.8 – Страница определенного поста

На рисунке 4.9 отображена страница *Chat*, где пользователи могут общаться между собой и обсуждать интересующиеся их вопросы в реальном времени.

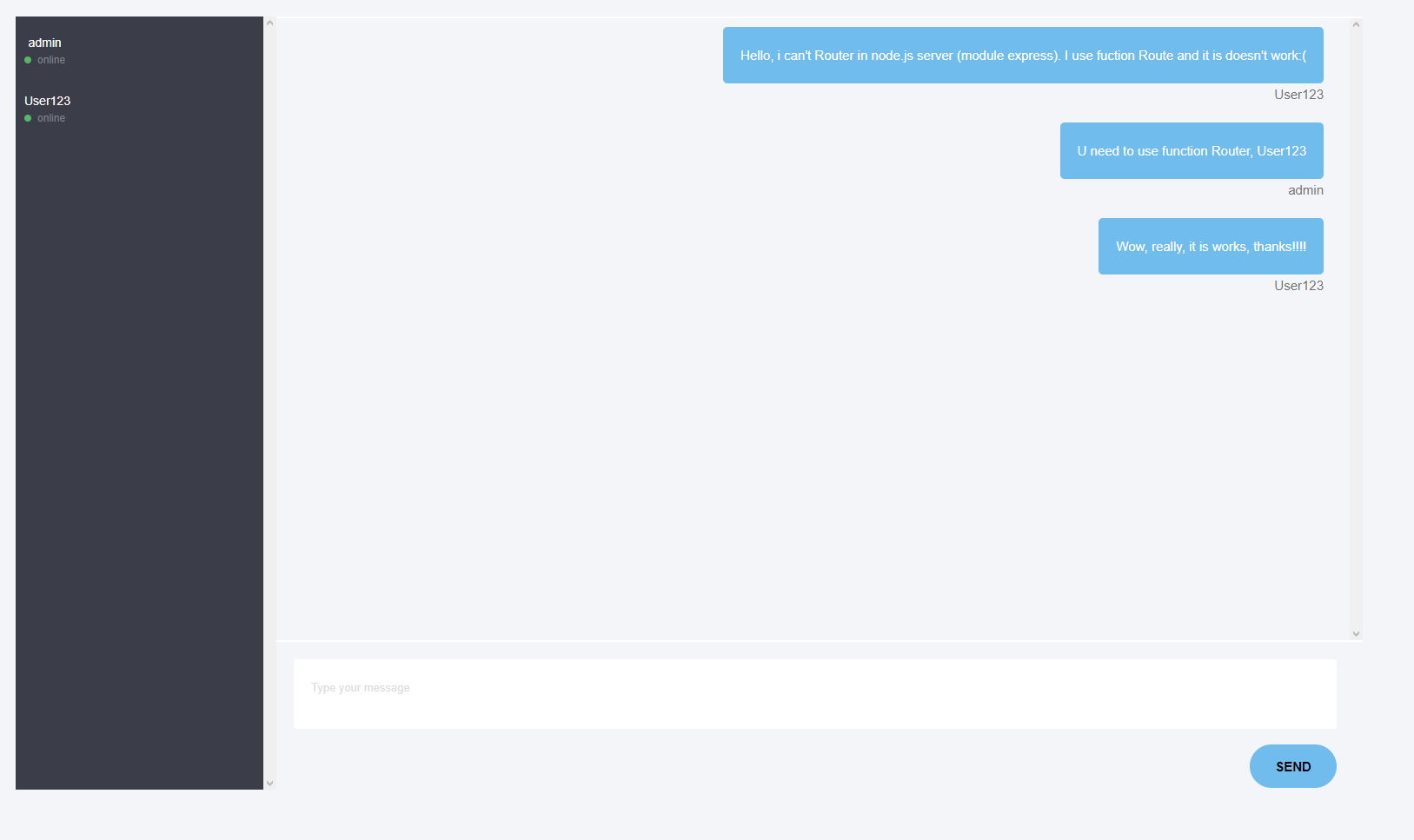


Рисунок 4.9 – Страница чата

На рисунке 4.9 можем увидеть, что два пользователя обмениваются сообщениями в реальном времени и все работает корректно.

# Заключение

ФИО

Подпись

Дата

Лист

1

Курсовой проект

Разраб.

Истомин И.О.

Провер.

Бурмакова А.В.

Н. контр.

Утверд.

Заключение

Лит.

Листов

1

БГТУ

В процессе решения поставленной задачи была достигнута поставленная цель по созданию веб-приложения «Форум».

Перед началом разработки был произведен аналитический обзор прототипов приложений подобной тематики и определение функциональных возможностей разрабатываемого приложения.

В процессе выполнения курсовой работы была спроектирована база данных для хранения в ней информации. База данных была разработана с помощью системы управления базами данных «MongoDB». Также был создан сервер и клиент с пользовательским интерфейсом. В том числе было реализовано тестирование программного продукта.

Также следует отметить, что для данного проекта был реализован сертификат stmnl.crt для использования HTTPS протокола.

В приложении была реализована технология AJAX и Socket.IO.

При разработке выполнены следующие пункты:

* регистрация и авторизация пользователей;
* разделение ролей;
* добавление, удаление, редактирование постов, комментариев;
* работа с изображениями.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объёме.

**Список литературных источников**

ФИО

Подпись

Дата

Лист

1

Курсовой проект

Разраб.

Истомин И.О.

Провер.

Бурмакова А.В.

Н. контр.

Утверд.

Список литературных источников

Лит.

Листов

1

БГТУ

1. Главная страница сайта Ответы Mail.Ru [Электронный ресурс] / Сайт otvet.mail.ru – Режим доступа: https://otvet.mail.ru. – Дата доступа: 15.05.2021.
2. Главная страница сайта Node Js [Электронный ресурс] / Сайт nodejs.org. – Режим доступа: https://nodejs.org/en/. – Дата доступа: 15.05.2021.
3. Главная страница сайта MongoDB [Электронный ресурс] / Сайт mongodb.com. – Режим доступа: <https://mongodb.com>. – Дата доступа: 15.05.2021.
4. Главная страница сайта Mongoose [Электронный ресурс] / Сайт [mongoosejs.com](file:///C:\Users\ilyai\Downloads\mongoosejs.com). – Режим доступа: https://mongoosejs.com/ – Дата доступа: 15.05.2021.
5. Главная страница сайта React [Электронный ресурс] / Сайт reactjs.org. – Режим доступа: https://reactjs.org/ – Дата доступа: 15.05.2021.
6. Главная страница сайта интернет ресурс для UML-диаграмм [Электронный ресурс] / Сайт app.diagrams.net. – Режим доступа: https://app.diagrams.net/ – Дата доступа: 15.05.2021.

ПРИЛОЖЕНИЕ А

**Изображения страниц**

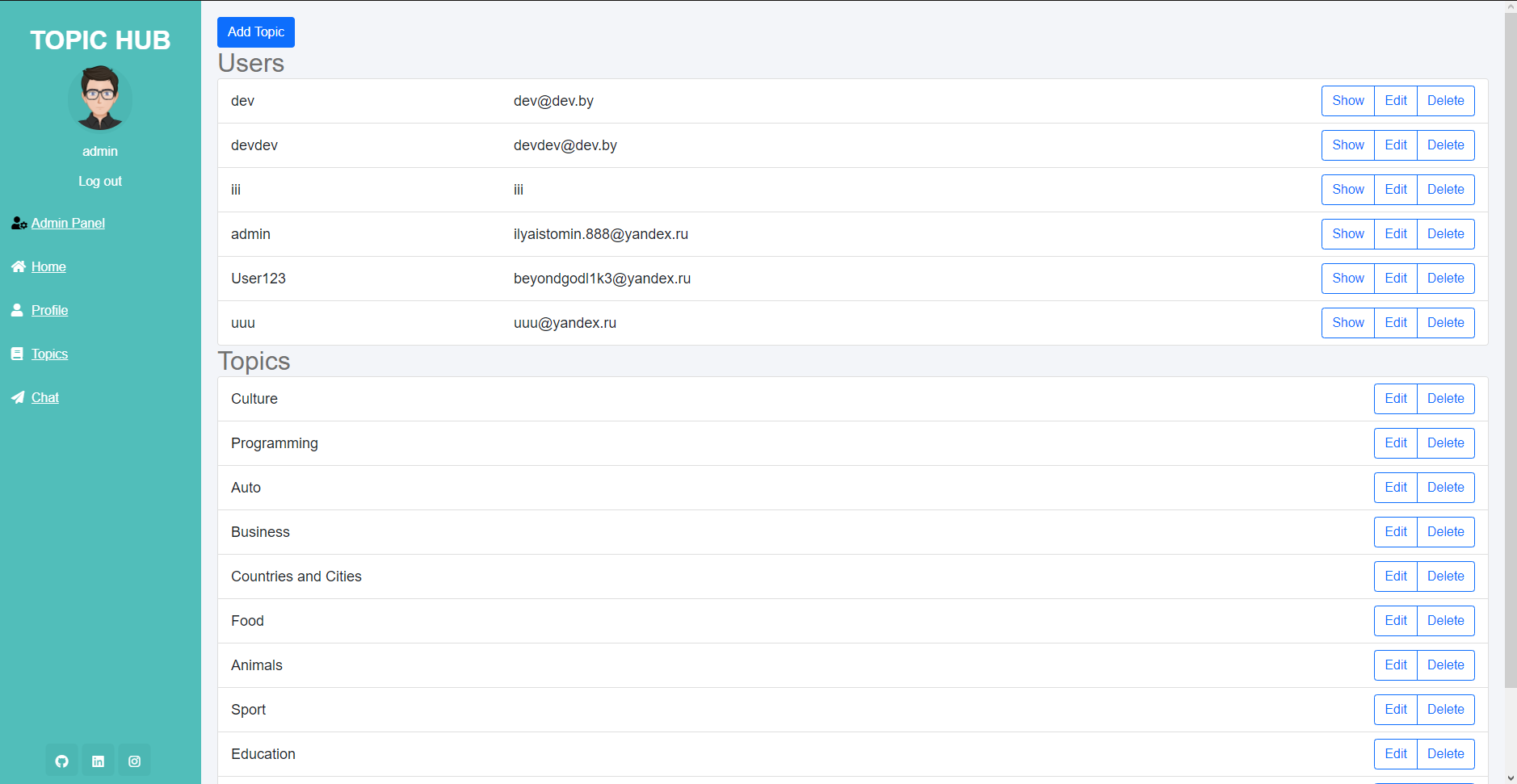


Рисунок А.1 – Изображение страницы «*Admin Panel*»

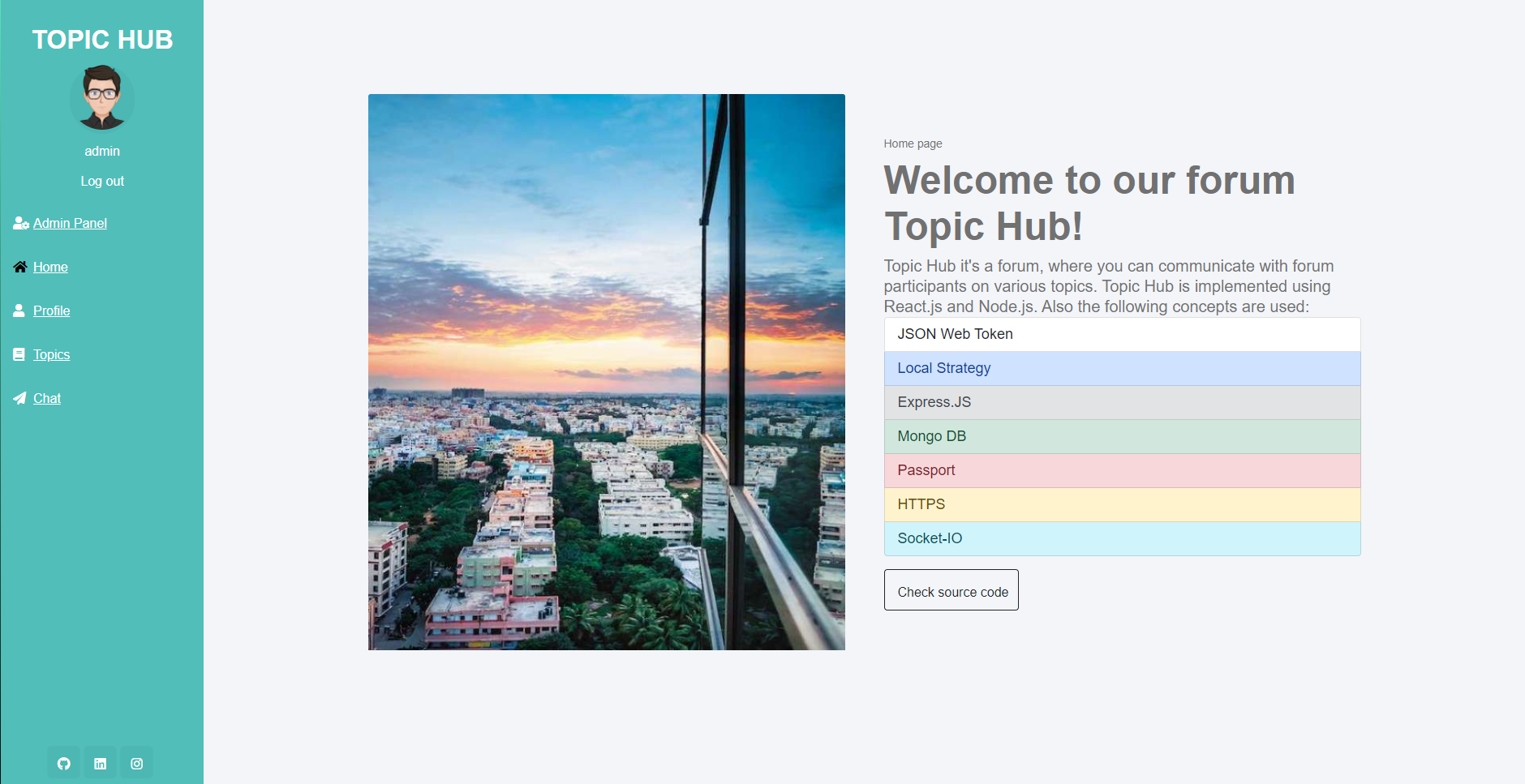


Рисунок А.2 – Изображение страницы «*Home*»

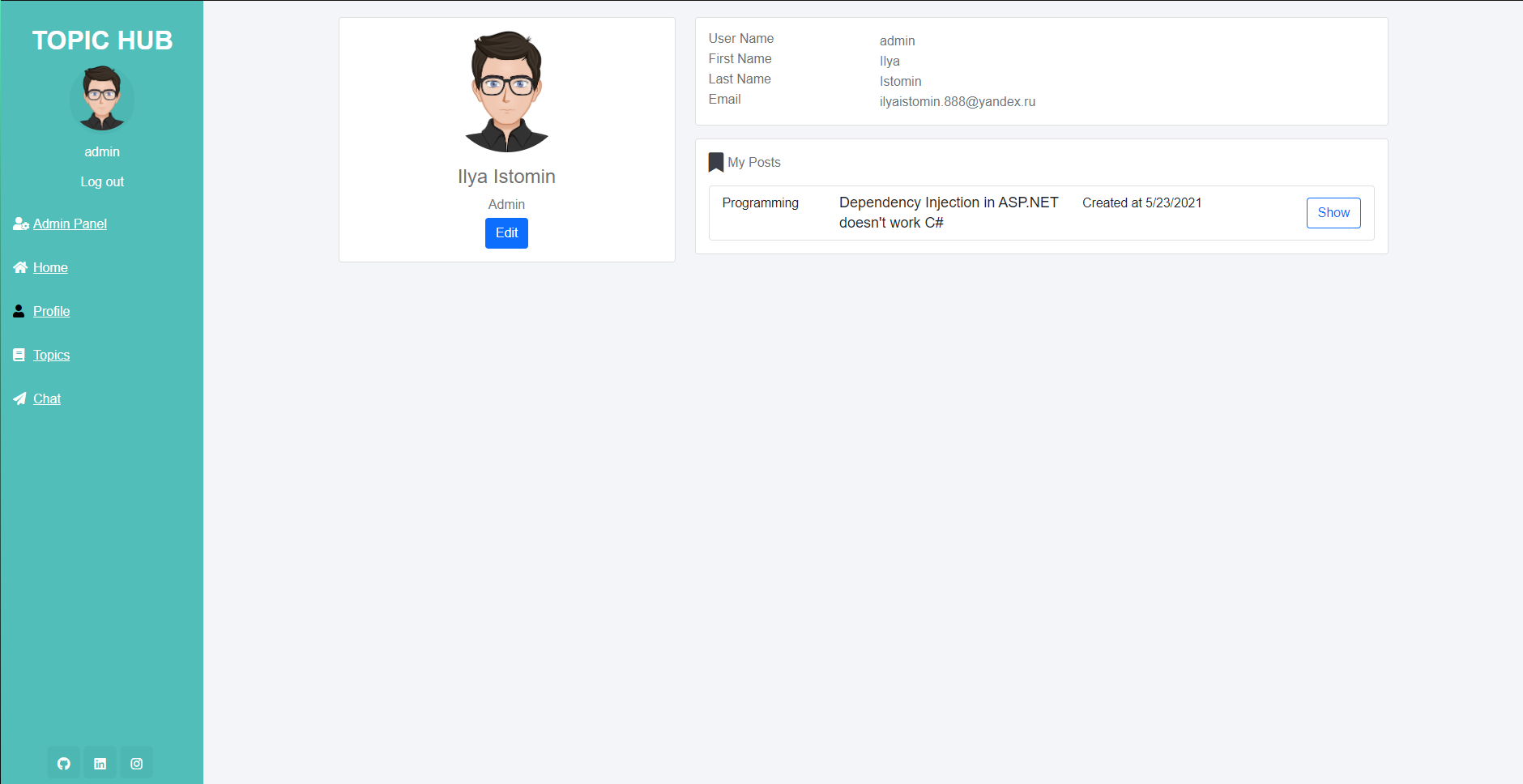


Рисунок А.3 – Изображение страницы «*Profile*»

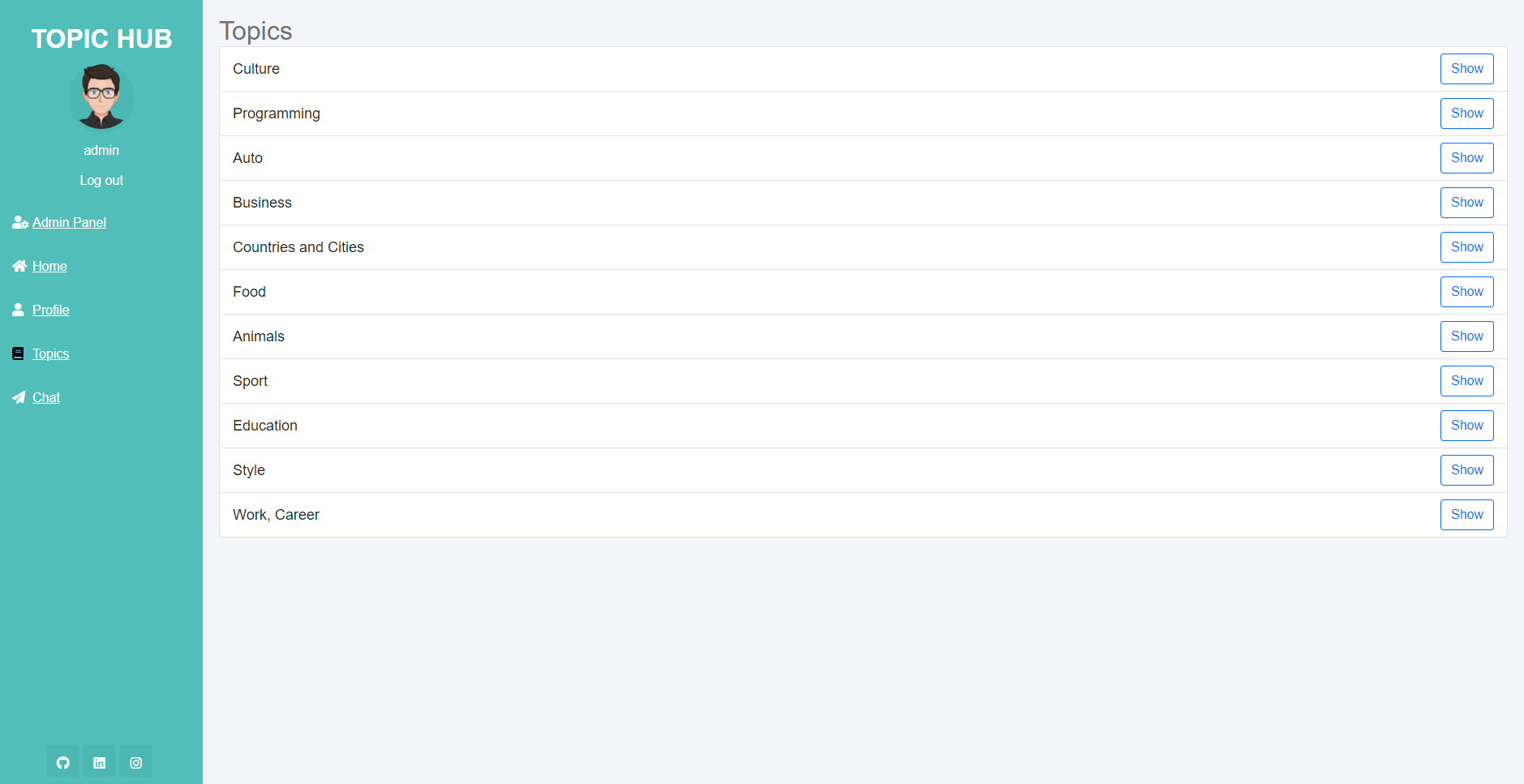


Рисунок А.4 – Изображение страницы «*Topics*»

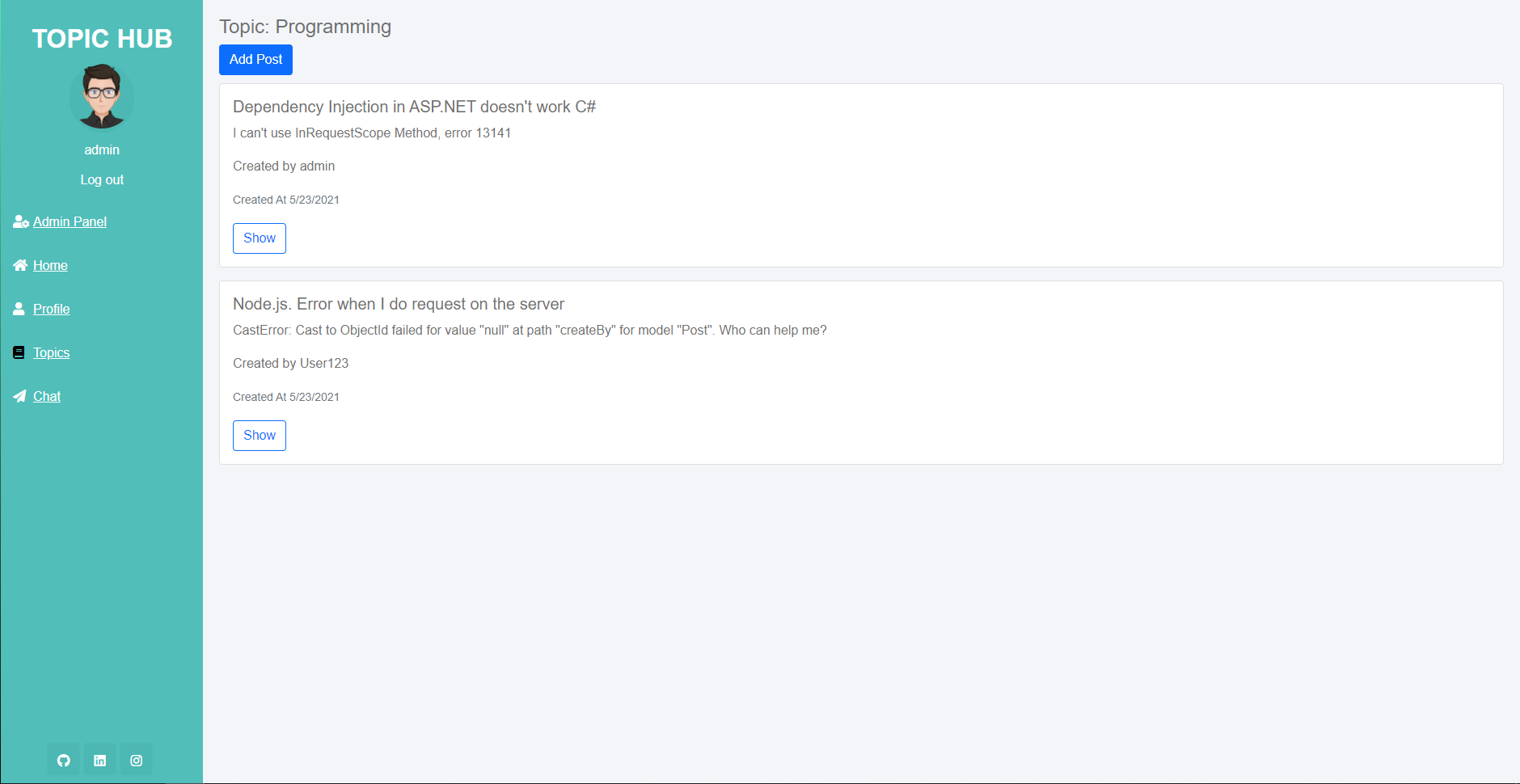


Рисунок А.5 – Изображение страницы «*Current Topic*»

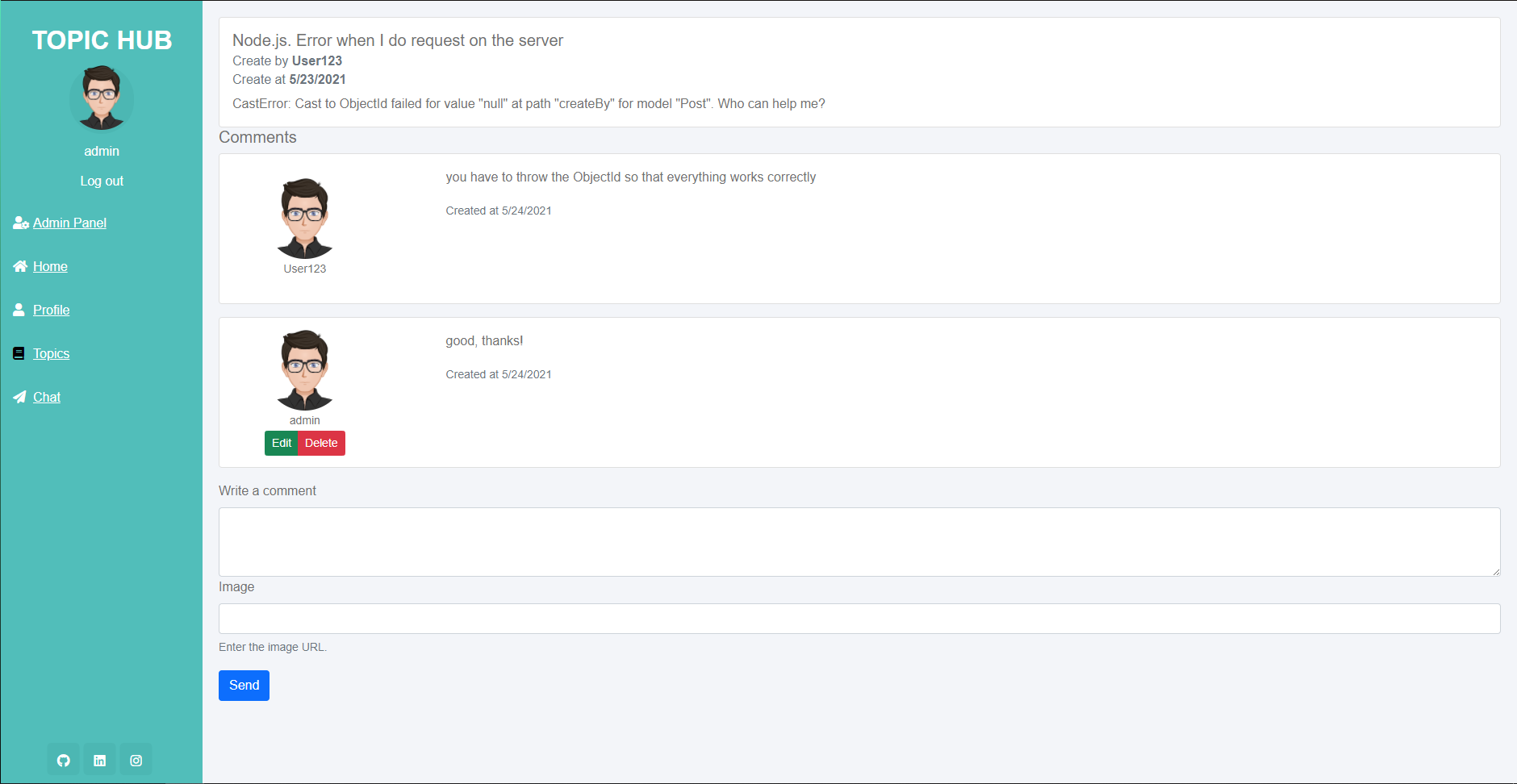


Рисунок А.6 – Изображение страницы «*Current Post*»

ПРИЛОЖЕНИЕ Б

const User = require('../../models/user.model');  
  
module.exports = async (req, res) => {  
 const {id} = req.params;  
 const user = await User.findById(id);  
 if (user !== null) {  
 const {firstName, lastName, email, github, twitter, instagram, facebook} = user;  
 const data = {  
 firstName,  
 lastName,  
 email,  
 github,  
 twitter,  
 instagram,  
 facebook  
 }  
 return res.json(data);  
 } else {  
 return res.json({message: 'This user does not exist'});  
 }  
  
}

ПРИЛОЖЕНИЕ В

import '../css/error.css';  
  
const Error = ({statusCode, statusMessage, message}) => {  
 return (  
 <div className="main\_content">  
 <div className="info">  
 <div id="notfound">  
 <div className="notfound">  
 <div className="notfound-404">  
 <h3>Oops! {statusMessage}</h3>  
 <h1><span>{statusCode}</span></h1>  
 </div>  
 <h2>{message}</h2>  
 </div>  
 </div>  
 </div>  
 </div>  
 )  
}  
  
export default Error;

ПРИЛОЖЕНИЕ Г

import {NavLink, useParams} from "react-router-dom";  
import React, {useEffect, useState} from "react";  
import useActions from "../helpers/hooks/useActions";  
import {useSelector} from "react-redux";  
import AddPostModal from "../modal/addPostModal";  
  
const CurrentTopic = () => {  
 const {topicName} = useParams();  
 const redux = useActions();  
 const [showAddPostModal, toggleAddPostModal] = useState(false);  
 const {posts} = useSelector(state => state.post);  
 useEffect(() => {  
 (async () => {  
 await fetch(`/topic/${topicName}`, {  
 method: 'GET',  
 headers: {  
 'Authorization': `Bearer ${localStorage.getItem('token')}`  
 }  
 })  
 .then(data => data.json())  
 .then(({\_id, name, createBy, createdAt, updatedAt}) => {  
 redux.getCurrentTopic(\_id, name, createBy, createdAt, updatedAt);  
 fetch(`/posts/${\_id}`, {  
 method: 'GET',  
 headers: {  
 'Authorization': `Bearer ${localStorage.getItem('token')}`  
 }  
 }).then(data => data.json())  
 .then((posts) => {  
 redux.getPosts(posts);  
 })  
 })  
 }  
 )()  
 }, [])  
 return (  
 <>  
 {showAddPostModal ? <AddPostModal closeCallback={() => toggleAddPostModal(false)}/> : null}  
 <div className="main\_content">  
 <div className="info">  
 <h4>Topic: {topicName}</h4>  
 <div style={{marginBottom: '10px'}}>  
 <button type="button" className="btn btn-primary" onClick={() => toggleAddPostModal(true)}>Add Post</button>  
 </div>  
 {!!posts ? (  
 <>  
 {posts.map(({\_id, header, description, createBy, createdAt}, index) => (  
 <div className="card mb-3" key={index}>  
 <div className="card-body">  
 <h5 className="card-title">{header}</h5>  
 <p className="card-text">{description}</p>  
 <p className="card-text">Created by {createBy.username}</p>  
 <p className="card-text"><small className="text-muted">Created  
 At {new Date(createdAt).toLocaleDateString('en-US')}</small></p>  
 <div className="btn-group" role="group" aria-label="Basic outlined example">  
 <NavLink to={`${topicName}/${\_id}`} exact  
 className="btn btn-outline-primary">Show</NavLink>  
 </div>  
 </div>  
 </div>  
 ))}  
  
 </>  
 ) : null}  
 </div>  
 </div>  
 </>  
 )  
}  
export default CurrentTopic;

ПРИЛОЖЕНИЕ Д

import {useParams} from "react-router-dom";  
import React, {useEffect, useState} from "react";  
import useActions from "../helpers/hooks/useActions";  
import {useSelector} from "react-redux";  
import EditCommentModal from "../modal/editCommentModal";  
import DeleteCommentModal from "../modal/deleteCommentModal";  
import EditPostModal from "../modal/editPostModal";  
import DeletePostModal from "../modal/deletePostModal";  
  
const CurrentPost = () => {  
 const redux = useActions();  
 const [showEditCommentModal, toggleEditCommentModal] = useState(false);  
 const [showEditPostModal, toggleEditPostModal] = useState(false);  
 const [showDeletePostModal, toggleDeletePostModal] = useState(false);  
 const {postId} = useParams();  
 const {header, createBy, description, createdAt} = useSelector(state => state.post);  
 const {id} = useSelector(state => state.user);  
 const {comments} = useSelector(state => state.comment);  
 const [commentValue, setCommentValue] = useState('');  
 const [showDeleteCommentModal, toggleDeleteCommentModal] = useState(false);  
 const [imgValue, setImgValue] = useState('');  
 const [commentIdValue, setCommentIdValue] = useState('');  
 const sendCommentSubmit = async (e) => {  
 e.preventDefault();  
 await fetch('/comment', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${localStorage.getItem('token')}`  
 },  
 body: JSON.stringify({  
 text: commentValue,  
 img: imgValue,  
 post: postId,  
 createBy: id  
 })  
 })  
 window.location.reload();  
 }  
 useEffect(() => {  
 (async () => {  
 await fetch(`/post/${postId}`, {  
 method: 'GET',  
 headers: {  
 'Authorization': `Bearer ${localStorage.getItem('token')}`  
 }  
 })  
 .then(data => data.json())  
 .then(({\_id, header, createBy, topic, description, createdAt, updatedAt}) => {  
 redux.getCurrentPost(\_id, header, createBy, topic, description, createdAt, updatedAt);  
 fetch(`/comments/${\_id}`, {  
 method: 'GET',  
 headers: {  
 'Authorization': `Bearer ${localStorage.getItem('token')}`  
 }  
 })  
 .then(data => data.json())  
 .then(comments => {  
 redux.getComments(comments);  
 })  
 })  
 })()  
 }, [])  
  
 const editCommentClick = (commendId) => {  
 setCommentIdValue(commendId);  
 toggleEditCommentModal(true);  
 }  
 const deleteCommentClick = (commendId) => {  
 setCommentIdValue(commendId);  
 toggleDeleteCommentModal(true);  
 }  
 return (  
 <>  
 {showEditPostModal ? (  
 <EditPostModal closeCallback={() => toggleEditPostModal(false)} postId={postId}/>) : null}  
 {showDeleteCommentModal ? (  
 <DeleteCommentModal commentId={commentIdValue} showDeleteCommentModal={showDeleteCommentModal}  
 closeCallback={() => toggleDeleteCommentModal(false)}/>) : null}  
 {showEditCommentModal ?  
 (<EditCommentModal closeCallback={() => toggleEditCommentModal(false)}  
 commentId={commentIdValue}/>) : null}  
 {showDeletePostModal ? (  
 <DeletePostModal closeCallback={() => toggleDeletePostModal(false)} postId={postId}  
 showDeletePostModal={showDeletePostModal}/>) : null}  
 <div className="main\_content">  
 <DeleteCommentModal/>  
 <div className="info">  
 <div className="card">  
 <div className="card-body">  
 <h5 className="card-title">{header}</h5>  
 <h6 className="card-subtitle mb-2 text-muted">Create by <b>{createBy.username}</b></h6>  
 <h6 className="card-subtitle mb-2 text-muted">Create  
 at <b>{new Date(createdAt).toLocaleDateString('en-US')}</b></h6>  
 <p className="card-text">{description}</p>  
 {createBy.\_id === id ? (<>  
 <button type="button" className="btn btn-success" style={{marginRight: '5px'}}  
 onClick={() => toggleEditPostModal(true)}>Edit  
 </button>  
 <button type="button" className="btn btn-danger"  
 onClick={() => toggleDeletePostModal(true)}>Delete  
 </button>  
 </>) : null}  
 </div>  
 </div>  
 <h5>Comments</h5>  
 {!!comments ? (  
 <>{comments.map(({\_id, text, createdAt, createBy, img}, index) => (  
 <div className="card mb-3" key={index}>  
 <div className="row g-0">  
 <div className="col-md-2">  
 <div style={{  
 display: 'flex',  
 justifyContent: 'center',  
 alignItems: 'center',  
 flexDirection: 'column',  
 height: '185px',  
 width: '80%'  
 }}>  
 {!!createBy.img ?  
 <img src={createBy.img} alt="Admin" className="rounded-circle"  
 width="100"/> :  
 <img src="https://bootdey.com/img/Content/avatar/avatar7.png"  
 alt="Admin"  
 className="rounded-circle" width="100"/>}  
 <small>{createBy.username}</small>  
 {createBy.\_id === id ? (  
 <div className="btn-group btn-group-sm" role="group" aria-label="...">  
 <button type="button" className="btn btn-success"  
 onClick={() => editCommentClick(\_id)}>Edit  
 </button>  
 <button type="button" className="btn btn-danger"  
 onClick={() => deleteCommentClick(\_id)}>Delete  
 </button>  
 </div>) : null}  
 </div>  
 </div>  
 <div className="col-md-8">  
 <div className="card-body">  
 <p className="card-text">{text}</p>  
 {!!img ? (<a href={img} target='\_blank'>Image</a>) : null}  
 <p className="card-text"><small className="text-muted">Created  
 at {new Date(createdAt).toLocaleDateString('en-US')}</small>  
 </p>  
 </div>  
 </div>  
 </div>  
 </div>  
 ))}</>  
 ) : null}  
 <form onSubmit={sendCommentSubmit}>  
 <div className="mb-3">  
 <label htmlFor="comment-textArea" className="form-label">Write a comment</label>  
 <textarea className="form-control" id="comment-textArea" rows="3" value={commentValue}  
 onChange={e => setCommentValue(e.target.value)}/>  
 <div className="mb-3">  
 <label htmlFor="imgInput" className="form-label">Image</label>  
 <input type="text" className="form-control" id="imgInput"  
 aria-describedby="imgHelp" value={imgValue}  
 onChange={e => setImgValue(e.target.value)}/>  
 <div id="imgHelp" className="form-text">Enter the image URL.  
 </div>  
 </div>  
 </div>  
 <input type="submit" className='btn btn-primary' value="Send"/>  
 </form>  
 </div>  
 </div>  
 </>  
 );  
}  
  
export default CurrentPost;