МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологий

Специальность 1-40 01 01 «Программное обеспечение информационных технологий»

Специализация 1-40 01 01 10 «Программное обеспечение информационных технологий (программирование интернет-приложений)»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту на тему:**

Web-приложение «Расписание транспорта»

Выполнил студент Коржова Валерия Сергеевна

(Ф.И.О.)

Руководитель проекта ст. препод. Дубовик М.В.

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

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

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

Консультанты ст. препод. Дубовик М.В.

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

Нормоконтролер ст. препод. Дубовик М.В.

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

Курсовой проект защищен с оценкой

Минск 2023

**Содержание**

[Введение 3](#_Toc134465384)

[1 Постановка задачи 4](#_Toc134465385)

[1.1 Обзор аналогов 4](#_Toc134465386)

[2 Проектирование web-приложения 6](#_Toc134465387)

[2.1 Диаграмма использования 6](#_Toc134465388)

[2.2 Диаграмма развертывания 7](#_Toc134465389)

[2.3 Проектирование базы данных 8](#_Toc134465390)

[3 Разработка web-приложения 11](#_Toc134465391)

[3.1 Разработка серверной части 11](#_Toc134465392)

[3.2 Разработка клиентской части 14](#_Toc134465393)

[4 Тестирование web-приложения 17](#_Toc134465394)

[5 Руководство пользователя 22](#_Toc134465395)

[5.1 Регистрация пользователя 22](#_Toc134465396)

[5.2 Аутентификация и авторизация пользователя 23](#_Toc134465397)

[5.3 Просмотр постов 24](#_Toc134465398)

[5.4 Действия с постами 25](#_Toc134465399)

[5.5 Администрирование 27](#_Toc134465400)

[Заключение 29](#_Toc134465401)

[Список используемых источников 30](#_Toc134465402)

[Приложение А 31](#_Toc134465403)

[Приложение Б 41](#_Toc134465404)

[Приложение В 42](#_Toc134465405)

# Введение

Главная задачей курсового проекта является разработка Web-приложения, позволяющего пользователю просматривать расписание городского транспорта и его маршрут. Также осуществлена возможность покупки проездных билетов разных типов на различные типы транспорта. Программное средство должно быть выполнено с использованием асинхронного программирования, взаимодействовать с базой данных. Отображение, бизнес логика и хранилище данных должны быть максимально независимы друг от друга для возможности расширения.

Для создания клиентской части был использован язык JavaScript и библиотека React.js.

Для серверной части приложения использовалась платформа Node.js с фреймворком Express, для работы с базой данных использовалась ORM Mongoose. Express - это фреймворк для Node.js, который предоставляет простой и удобный способ создания веб-приложений и API. Он позволяет быстро создавать серверные приложения на Node.js и предоставляет множество полезных функций и инструментов для работы с запросами, ответами, маршрутизацией, обработкой ошибок и другими задачами, связанными с веб-разработкой. Mongoose - это ORM для работы с MongoDB. Она позволяет удобно и эффективно создавать модели данных, выполнять запросы к базе данных и управлять взаимодействием между приложением и базой данных MongoDB. Mongoose предоставляет множество инструментов для работы с данными, включая валидацию, преобразование типов данных, хуки жизненного цикла и другие возможности, которые значительно упрощают и ускоряют разработку приложений, использующих MongoDB. В данном приложения использовалась СУБД MongoDB. [1]

Записка состоит из введения, 5 глав, заключения и приложений.

В первой главе рассматривается постановка задачи.

Во второй главе представлена разработка архитектуры приложения и структура курсового проекта.

В третьей главе описана разработка функциональной модели и модели данных программного средства.

Четвёртая глава состоит из руководства пользователя.

В пятой главе описано тестирование приложения.

В заключении описывается результат работы над курсовым проектом, выполненные цели.

В приложении содержится листинг кода некоторых классов и моделей.

1 Постановка задачи

Главная задача курсового проекта является разработка web-приложения, которое позволяет пользователю просматривать расписание городского транспорта. В данном курсовом проекте требовалось реализовать следующие задачи:

* регистрация и авторизация пользователя;
* изменение информации о пользователе;
* просматривание расписания транспорта;
* обмен данными между клиентом и сервером должен производиться по протоколу HTTPS;
* возможность добавления нового транспорта администратором;
* возможность покупки проездных билетов пользователем;
* получение уведомлений пользователем об истечении проездного билета;
* просмотр остановок и маршрутов на карте пользователем.

## 1.1 Обзор аналогов

Современные города предлагают множество вариантов общественного транспорта для удобства жителей и гостей города. Однако, часто бывает сложно следить за расписанием движения транспорта и планировать свой маршрут. В таких случаях, приложения для просмотра расписания транспорта становятся незаменимым инструментом для многих людей.

На рисунке 1.1 представлен внешний вид web-приложения «Minsktrans». Это приложение предназначено для просмотра расписания прибытия транспортных средств на остановки.

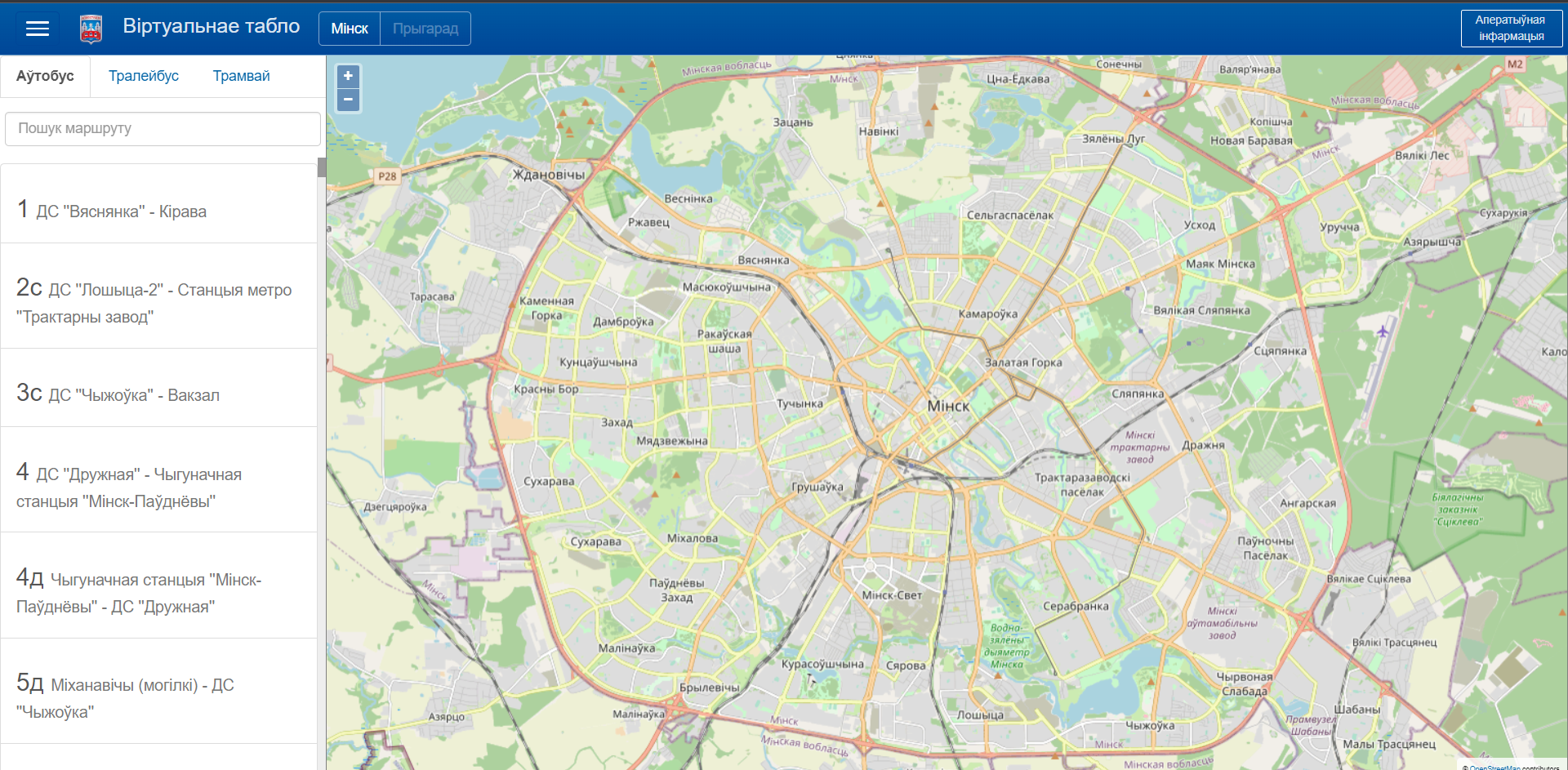


Рисунок 1.1 – Главная страница приложения «Minsktrans»

Главными преимуществами данного ресурса является актуальность расписания, хорошая визуализация маршрутов.

Из выявленных недостатков данного ресурса необходимо выделить то, что пользователь не может купить проездной билет.

Следующий аналог представлен на рисунке 1.2 – web-приложение «minsk.btrans».

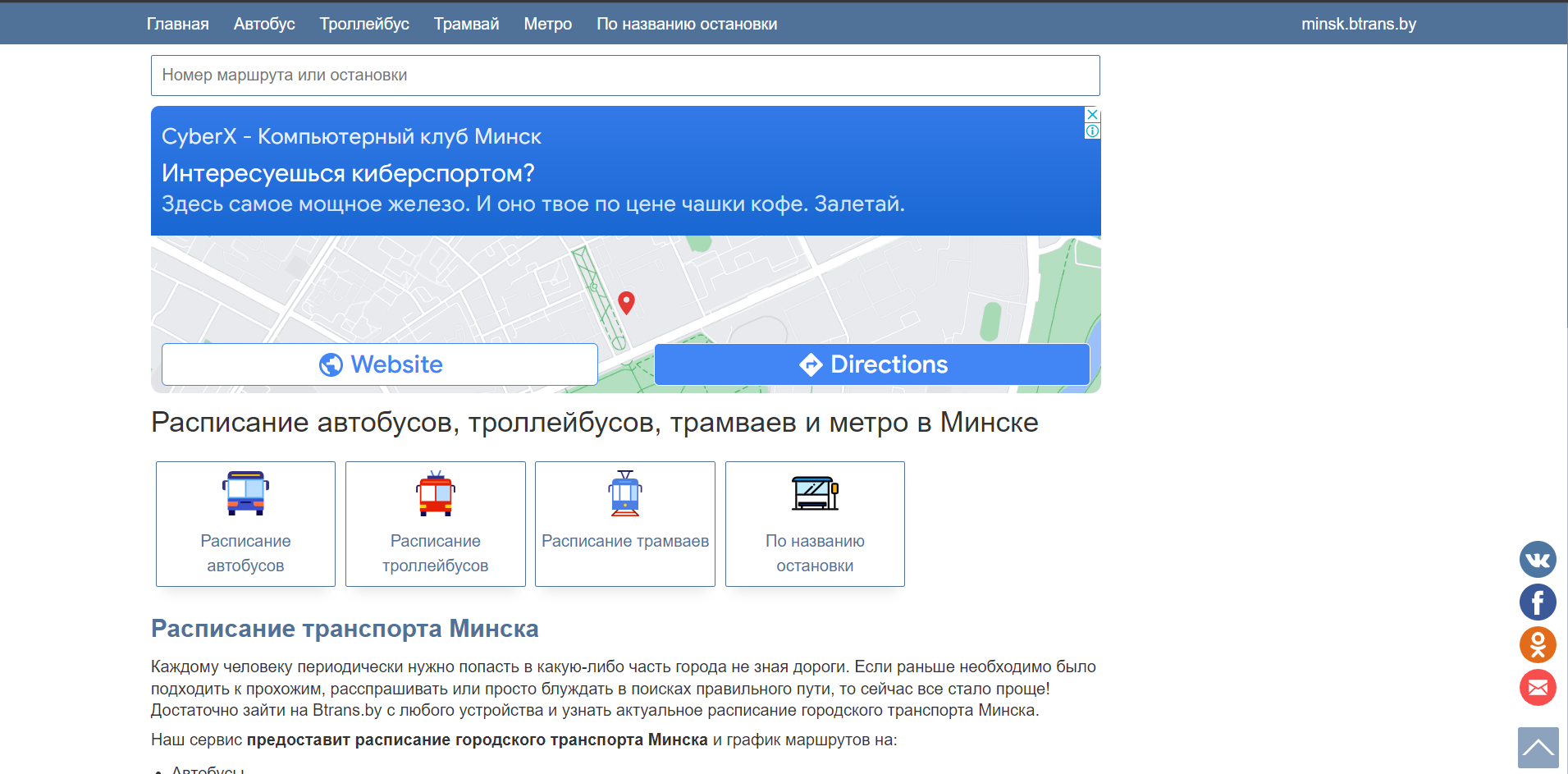


Рисунок 1.2 – Главная страница приложения «minsk.btrans»

Преимущества данного приложения заключается в просмотра расписания транспорта, наличие поиска остановок, фильтрация по типу транспорта.

Главным недостатком этого приложения является отсутствие карты с нанесенными на нее остановками.

2 Проектирование web-приложения

## 2.1 Диаграмма использования

Диаграмма использования в UML является графическим представлением набора элементов, связанных между собой в виде графа. Она используется для моделирования и визуализации вариантов использования (use cases) в системе. В диаграмме использования варианты использования представляются в виде овалов с их именами, а действующие лица (actors) изображаются в виде человечков, представляющих пользователей системы или внешние системы, взаимодействующие с системой.

Варианты использования могут быть связаны друг с другом различными типами связей. Обобщение (generalization) используется для представления отношения "является" между вариантами использования, где один вариант использования является более общим случаем другого. Расширение (extend relationship) используется, когда один вариант использования может быть расширен опциональными шагами другого варианта использования. Включение (include relationship) используется, когда один вариант использования включает другой вариант использования как обязательный шаг.

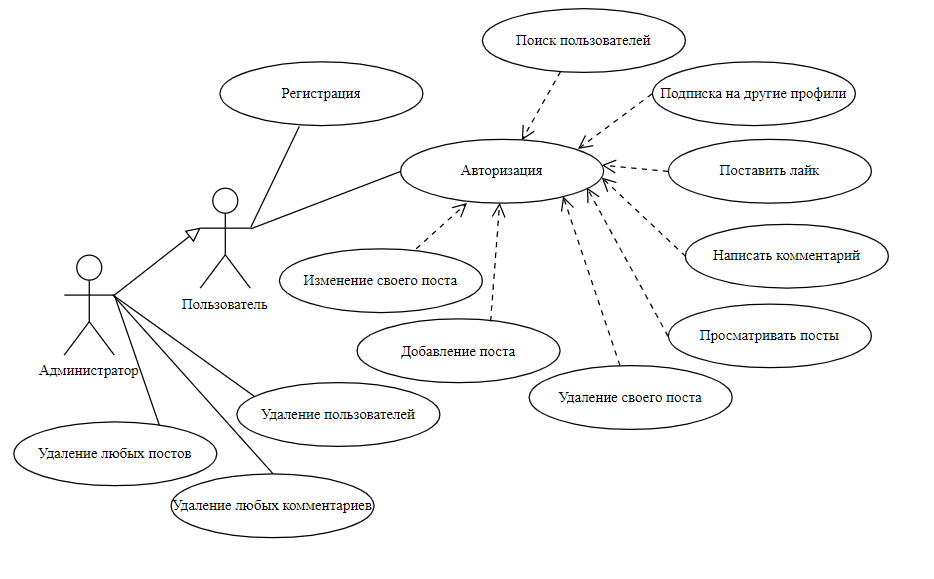
В целом, диаграмма использования UML предоставляет визуальное представление вариантов использования и их взаимосвязей, что помогает лучше понять и описать функциональные возможности системы и взаимодействие с ее пользователем.

Рисунок 2.1 – Диаграмма использования

Исходя из диаграммы использования можно выделить 2 основные роли в приложении: пользователь и администратор.

## 2.2 Диаграмма развертывания

Диаграмма развертывания, также известная как диаграмма размещения, используется для визуализации конфигурации и топологии распределенной программной системы. Она представляет собой графическое представление, которое показывает, как компоненты системы размещены на отдельных узлах и как физические соединения устанавливаются между аппаратными устройствами, используемыми в системе.

Диаграмма развертывания является инструментом для моделирования элементов и компонентов программной системы, которые существуют только во время выполнения (в runtime). Она фокусируется на представлении компонентов-экземпляров программы, таких как исполняемые файлы или динамические библиотеки, которые активно используются во время работы системы. На диаграмме развертывания не показываются компоненты, которые не используются во время выполнения системы.

В результате, диаграмма развертывания предоставляет информацию о размещении компонентов системы на физических узлах, а также о связях между ними через физические сетевые соединения. Это помогает визуализировать и лучше понять организацию и взаимодействие компонентов системы на различных узлах, что полезно для архитекторов и разработчиков при проектировании и развертывании распределенных программных систем.

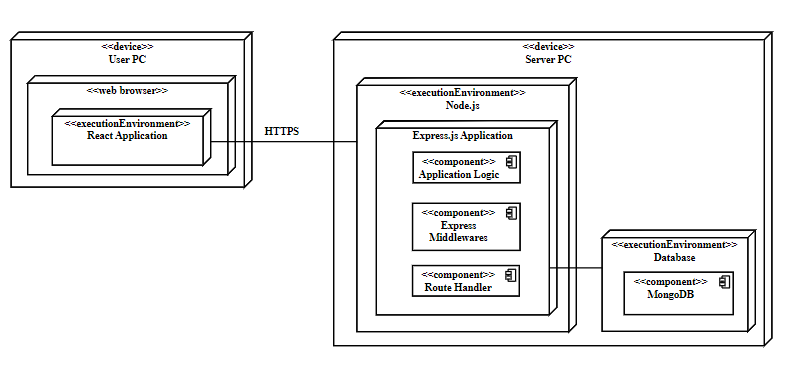


Рисунок 2.2 – Структурная схема приложения

## Приложение состоит из трех основных сервисов, которые обеспечивают его функциональность и взаимодействие с данными:

## 1. React.js - сервис, отвечающий за реализацию frontend-части приложения. Он обеспечивает создание пользовательского интерфейса, отображение данных и взаимодействие с пользователем на стороне клиента.

## 2. Node.js - сервис, реализующий backend-часть приложения. Он обрабатывает клиентские запросы, осуществляет бизнес-логику, взаимодействие с базой данных и другими сервисами. Node.js обеспечивает обработку запросов от клиента и генерацию ответов.

## 3. MongoDB - сервис, обеспечивающий доступ к данным. Это нереляционная база данных, которая хранит данные в формате JSON-подобных документов. MongoDB позволяет приложению хранить и извлекать данные, а также выполнять операции поиска, добавления, обновления и удаления данных.

## Каждый из этих сервисов в приложении работает независимо друг от друга и предоставляет определенный функционал. Они взаимодействуют между собой с помощью JSON, который используется для передачи данных между клиентской и серверной частями приложения. Клиентские запросы сначала обрабатываются Node.js сервером, затем React сервис генерирует соответствующее представление, которое передается обратно клиенту и отображается на странице веб-браузера.

## Такая архитектура приложения позволяет достичь высокой изолированности и гибкости, облегчая расширение и модификацию функционала приложения.

## 2.3 Проектирование базы данных

Важным аспектом при проектировании и создании базы данных является тщательный анализ предметной области приложения. Это позволяет разработать модель данных, которая точно отражает взаимодействие с данными в общем и соответствует требованиям приложения.

Корректный анализ предметной области, проектирование и разработка базы данных должны привести к созданию модели, которая предоставляет все необходимые функциональные возможности для пользователей. Это означает, что модель должна быть способна эффективно хранить, организовывать и обрабатывать данные, а также поддерживать необходимую бизнес-логику приложения.

Кроме того, грамотно разработанная модель данных должна быть гибкой и расширяемой, чтобы учитывать возможные изменения в предметной области и требования пользователя. Это позволит легко адаптировать базу данных к новым ситуациям и расширять ее функциональность, не нарушая целостность данных и эффективность работы системы.

Таким образом, правильный анализ, проектирование и разработка базы данных обеспечивают создание модели, которая соответствует потребностям пользователя и обеспечивает эффективное взаимодействие с данными. Модели базы данных для программного средства представлена на рисунке 2.3.

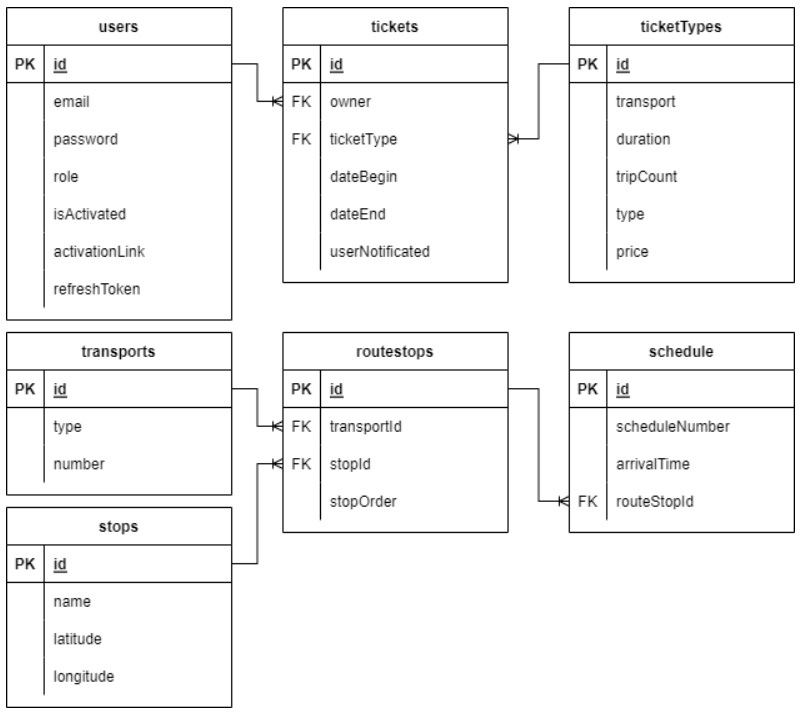


Рисунок 2.3 – Модель базы данных

Грамотно спроектированная модель базы данных является основой инфраструктуры базы данных и обеспечивает правильное взаимодействие пользовательских таблиц. Схема базы данных должна отображать связи между таблицами, чтобы корректно представлять их взаимосвязи.

Одна из основных таблиц в базе данных называется "users" и содержит информацию о пользователях. Эта таблица устанавливает связь с таблицей "tickets", где хранятся билеты, приобретенные пользователем. С таблицей "tickets" связана таблица "ticketTypes", где хранятся типы проездных билетов. Таблица "transports" содержит информацию о транспорте, а таблица "stops" – об остановках. Их связывает таблица "routeStops", где хранится информация об маршрутных остановках конкретного транспорта. Таблица "schedule" содержит информацию об расписании.

Это краткое описание основных таблиц базы данных, которые составляют инфраструктуру системы и обеспечивают правильное взаимодействие между данными. В таблице 2.1 представлена структура таблицы "users". Описание структуры таблицы «users» представлено в таблице 2.1.

Таблица 2.1 – Структура таблицы «users»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| id | ObjectId, primary key | Идентификатор пользователя |
| email | string, unique | Логин пользователя |
| password | string | Пароль пользователя |
| role | string | Роль пользователя |
| isActivated | boolean | Активность пользователя |
| activationLink | string | Ссылка активации |
| refreshToken | string | Токен |

В таблице «tickets» хранится информация о билетах пользователей. Описание структуры таблицы «tickets» представлено в таблице 2.2.

Таблица 2.2 – Структура таблицы «tickets»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| id | ObjectId, primary key | Идентификатор билета |
| owner | string, foreign key | Идентификатор пользователя |
| ticketType | string, foreign key | Идентификатор типа билета |
| dateBegin | Date | Дата начала действия билета |
| dateEnd | Date | Дата окончания действия билета |
| userNotificated | boolean | Оповещен ли пользователь об окончании действия билета |

Таблица «ticketTypes» содержит типы билетов. Её структура представлена в таблице 2.3.

Таблица 2.3 – Структура таблицы «ticketTypes»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| id | ObjectId, primary key | Идентификатор типа билета |
| transport | number | Идентификатор пользователя |
| duration | number | Идентификатор подписки |
| tripCount | number | Количество поездок |
| type | string | Тип проездного |
| price | number | Цена |

В таблице «transports» хранятся все наименования транспорта. Описание структуры таблицы «transports» представлено в таблице 2.4.

Таблица 2.4 – Структура таблицы «transports»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| id | ObjectId, primary key | Идентификатор транспорта |
| type | string | Тип транспорта |
| number | string | Номер транспорта |

В таблице 2.5 представлена структура таблицы "stops". Описание структуры таблицы «stops» представлено в таблице 2.5.

Таблица 2.5 – Структура таблицы «stops»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| id | ObjectId, primary key | Идентификатор остановки |
| name | string, unique | Название остановки |
| latitude | number | Широта остановки |
| longitude | number | Долгота остановки |

В таблице «routestops» хранится информация о остановках в маршруте. Описание структуры таблицы «routestops» представлено в таблице 2.6.

Таблица 2.6 – Структура таблицы «routestops»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| id | ObjectId, primary key | Идентификатор маршрутной остановки |
| transportId | ObjectId, foreign key | Идентификатор транспорта |
| stopId | ObjectId, foreign key | Идентификатор остановки |
| stopOrder | number | Номер остановки в маршруте |

Таблица «schedule» содержит расписание прибытия транспорта на остановку. Её структура представлена в таблице 2.7.

Таблица 2.7 – Структура таблицы «schedule»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| id | ObjectId, primary key | Идентификатор расписания |
| scheduleNumber | number | Номер расписания |
| arrivalTime | string | Время прибытия на остановку |
| routeStopId | ObjectId, foreign key | Идентификатор маршрутной остановки |

Таким образом при проектировании базы данных используется 7 таблиц и связь один-ко-многим.

3 Разработка web-приложения

## В рамках выполнения курсового проекта было разработано веб-приложение, которое применяет архитектурный стиль REST API.

## Серверная часть отвечает за обработку запросов от клиентов, выполнение бизнес-логики и взаимодействие с базой данных или другими внешними сервисами. Она предоставляет REST API, который клиенты могут использовать для отправки запросов и получения данных от сервера. Взаимодействие с базой данных или другими сервисами осуществляется через определенные эндпоинты API.

## Клиентское представление (React.js) отвечает за отображение данных и взаимодействие с пользователем. Клиентское представление получает данные от сервера, обычно через REST API, и отправляет запросы на сервер для получения или обновления информации.

## Такая архитектура разделяет ответственность между серверной и клиентской частями приложения, что облегчает разработку и обеспечивает модульность. REST API является стандартизированным способом взаимодействия между компонентами и позволяет создавать гибкие и масштабируемые веб-приложения.

## 3.1 Разработка серверной части

На рисунке 3.1 отображена структура серверной части разрабатываемого программного средства, включающая в себя директории, конфигурационные файлы и файлы скриптов.

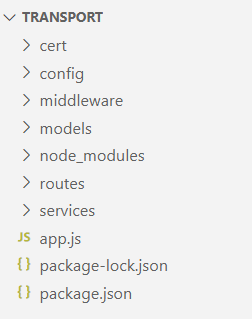


Рисунок 3.1 – Структура проекта сервера

Файл app.js является точкой входа в приложение. В нём происходит настройка прослушивания запросов, которые направляются к приложению на определенном порту. Веб-сервер подключает необходимые модули для работы сервера, выполняет начальную инициализацию приложения и регистрирует маршрутизаторы. Кроме того, в app.js осуществляется подключение к базе данных.

Директория «models» содержит файлы с классами-сущностями базы данных. Приложение использует ORM-библиотеку «Mongoose» для взаимодействия с базой данных. Данные, которые используются в «Mongoose», описываются определенной схемой по которой будет создана модель. Пример схемы пользователя приведён в листинге 3.1. Другие схемы находятся в Приложении А.

const { Schema, model, Types } = require('mongoose');

const schema = new Schema({

email: { type: String, required: true, unique: true },

password: { type: String, required: true },

role: { type: String, required: true },

isActivated: { type: Boolean, default: false },

activationLink: { type: String },

refreshToken: { type: String },

});

module.exports = model('User', schema);

Листинг 3.1 – Схема пользователя

В директории «routes» содержатся роуты. Маршрут (или роут) представляет собой определенный путь или URL, к которому можно отправить запрос веб-приложения. Он определяет, как приложение должно обрабатывать и отвечать на запросы, поступающие от клиентов. Роуты взаимодействуют с базой данных через модели. Пример одного из методов роута приведен в листинге 3.2. Другие примеры роутов находятся в Приложении Б.

router.post(

'/register',

async (req, res) => {

try {

const { email, password } = req.body;

const candidate = await User.findOne({ email: email.toLowerCase() });

if (candidate) {

return res.status(400).json({ message: 'Гэты карыстальнiк ужо зарэгiстраваны' });

}

const hashedPassword = await bcrypt.hash(password, 12);

const activationLink = uuid.v4();

await MailService.sendActivationMail(email, `${config.get('baseUrl')}/api/auth/activate/${activationLink}`);

const user = new User({ email: email.toLowerCase(), password: hashedPassword, role: "user", activationLink });

await user.save();

const tokens = TokenService.generateTokens({ id: user.\_id});

await TokenService.saveToken(user.\_id, tokens.refreshToken);

res.cookie('refreshToken', tokens.refreshToken, { maxAge: 30 \* 24 \* 60 \* 60 \* 1000, httpOnly: true, secure: true, sameSite: 'none' });

res.status(201).json({ ...tokens, user, message: 'Карыстальнiк створаны. Праверце пошту.' });

} catch (e) {

console.log(e);

res.status(500).json({ message: 'Что-то пошло не так, попробуйте снова' });

}

Листинг 3.2 – Метод из файла auth.routes.js

Директория «middleware» содержат файлы, реализующие функции проверки авторизации и роли администратора (листинг 3.3).

const auth const jwt = require('jsonwebtoken');

const config = require('config');

module.exports = (req, res, next) => {

if(req.method === 'OPTIONS') {

return next();

}

try {

const token = req.headers.authorization.split(' ')[1];

if(!token) {

return res.status(401).json({message: 'Карыстальнiк не аўтыразаваны'});

}

jwt.verify(token, config.get('jwtAccessSecret'));

next();

} catch (e) {

//console.log("middlware " + e);

if (e instanceof jwt.TokenExpiredError) {

return res.status(401).json({ message: e.message });

}

res.status(401).json({message: 'Карыстальнiк не аўтыразаваны'});

}

}

module.exports = (req, res, next) => {

if (req.method === 'OPTIONS') {

return next();

}

try {

const token = req.headers.authorization.split(' ')[1];

if (!token) {

return res.status(401).json({ message: 'Карыстальнiк не аўтыразаваны' });

}

const decoded = jwt.verify(token, config.get('jwtAccessSecret'));

if (decoded.role !== 'admin') {

return res.status(403).json({ message: 'Няма доступу' });

}

next();

} catch (e) {

//console.log("middlware " + e);

if (e instanceof jwt.TokenExpiredError) {

return res.status(401).json({ message: e.message });

}

res.status(401).json({ message: 'Карыстальнiк не аўтыразаваны' });

}

}

Листинг 3.3 – Функция проверки аунтификации и роли

Данная функция используется для извлечения токена из заголовка Authorization запроса, его декодирования с помощью jwt.verify() и получения идентификатора и роли пользователя, связанного с этим токеном. Если все шаги выполнены успешно, функция передает управление следующей middleware-функции в цепочке обработки запросов с помощью вызова next(). В случае возникновения ошибки отсутствия или невалидного токена функция отправляет ответ с кодом состояния 401 и сообщением об ошибке в формате JSON. При несоответствии роли администратору возвращается ошибка со статусом 403.

Директория «services» содержит файлы вспомогательных классов, такие как «MailService» и «TokenService». «MailService» предназначен для рассылки письма пользователям, которые прошли регистрацию. Код данного класса представлен в листинге 3.4.

const nodeMailer = require('nodemailer');

const config = require('config');

class MailService {

constructor() {

this.transporter = nodeMailer.createTransport({

host: config.get('smtpHost'),

port: config.get('smtpPort'),

secure: config.get('smtpSecure'),

auth: {

user: config.get('smtpUser'),

pass: config.get('smtpPassword')

}

});

}

async sendActivationMail(to, link) {

await this.transporter.sendMail({

from: config.get('smtpUser'),

to,

subject: `Активация аккаунта на ${config.get('baseUrl')}`,

text: '',

html:

`

<div>

<h1>Для активации перейдите по ссылке</h1>

<a href="${link}">${link}</a>

</div>

`

});

}

}

module.exports = new MailService();

Листинг 3.4 – Класс «MailService»

Класс «TokenService» предназначен для генерации токена доступа и токена обновления и сохранения токена обновления в базу данных в таблицу «users». Код данного класса приведен в листинге 3.5.

const jwt = require('jsonwebtoken');

const config = require('config');

const User = require('../models/User');

class TokenService {

generateTokens(payload) {

const accessToken = jwt.sign(payload, config.get('jwtAccessSecret'), { expiresIn: "1h" });

const refreshToken = jwt.sign(payload, config.get('jwtRefreshSecret'), { expiresIn: "6h" });

return {

accessToken,

refreshToken

}

}

async saveToken(id, refreshToken) {

const tokenData = await User.findOne({ \_id: id });

if (tokenData) {

tokenData.refreshToken = refreshToken;

return tokenData.save();

}

const token = await User.updateOne({ \_id: id }, { $set: { refreshToken } });

return token;

}

}

module.exports = new TokenService();

Листинг 3.5 – Класс «TokenService»

Файл «app.js» является точкой входа в приложение. В данном файле настраиваются https-сервер и websocket сервер (листинг 3.6).

const express = require('express');

const config = require('config');

const mongoose = require('mongoose');

const https = require("https");

const fs = require("fs");

const cors = require('cors');

const bodyParser = require('body-parser');

const cookieParser = require('cookie-parser');

const Ticket = require('./models/Ticket');

const app = express();

app.use(express.json({ extended: true }));

app.use(cors());

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({ extended: true }));

app.use(cookieParser());

app.use('/api/auth', require('./routes/auth.routes'));

app.use('/api/tickets/types', require('./routes/ticket.types.routes'));

app.use('/api/tickets', require('./routes/ticket.routes'));

app.use('/api/stops', require('./routes/stops.routes'));

app.use('/api/routes', require('./routes/routes.routes'));

app.use('/api/transports', require('./routes/transport.routes'));

app.use('/api/schedule', require('./routes/schedule.routes'));

app.use('/api/user', require('./routes/user.routes'))

const PORT = config.get('port') || 5000;

async function start() {

try {

await mongoose.connect(config.get('mongoUri'), {

useNewUrlParser: true,

useUnifiedTopology: true,

//useCreateIndex: true

});

const httpsServer = https.createServer(

{

key: fs.readFileSync("./cert/L.key"),

cert: fs.readFileSync("./cert/L.crt"),

},

app)

const io = require("socket.io")(httpsServer, {

cors: {

origin: "https://localhost:3000",

methods: ["GET", "POST"],

},

});

io.on("connection", (socket) => {

console.log(`⚡: ${socket.id} user just connected!`);

setInterval(async () => {

const tickets = await Ticket.find({

dateEnd: { $lte: new Date() }, userNotificated: false}).populate('ticketType');

if (tickets.length > 0) {

socket.emit('message', JSON.stringify(tickets));

}

}, 5000)

socket.on('disconnect', () => {

console.log('user disconnected');

})

});

httpsServer.listen(PORT, () => {

console.log(`Server has been started on port ${PORT}...`)

});

} catch (e) {

console.log('Server Error', e.message);

process.exit(1);

}

}

start()

Листинг 3.6 – Файл «app.js»

## В этом файле можно настроить и запустить сервер с использованием протокола HTTPS, который обеспечивает безопасное соединение между клиентом и сервером с помощью шифрования данных. Для настройки HTTPS-сервера в файле «app.js» нужно указать пути к сертификату и закрытому ключу, которые используются для шифрования и аутентификации соединения.

## Кроме того, в файле «app.js» можно настроить WebSocket-сервер с помощью пакета Socket.IO. WebSocket позволяет установить постоянное двустороннее соединение между клиентом и сервером, что позволяет обмениваться данными в режиме реального времени. Socket.IO облегчает создание WebSocket-сервера, предоставляя удобный API для управления событиями и обмена данными между клиентом и сервером.

## 3.2 Разработка клиентской части

На рисунке 3.2 отображена структура клиентской части разрабатываемого программного средства.

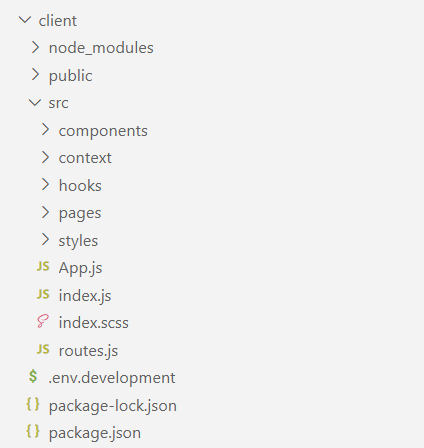


Рисунок 3.2 – Структура проекта клиента

React - это JavaScript-библиотека, разработанная Facebook, для создания пользовательских интерфейсов (UI) веб-приложений. Она позволяет разработчикам создавать многокомпонентные приложения, которые обновляются и отображают данные в режиме реального времени, не требуя перезагрузки страницы. React использует декларативный подход, позволяющий описывать состояние компонентов и их поведение, что делает его простым в использовании и позволяет разработчикам легко понимать и поддерживать свои приложения.Директория «src» содержит исходный код клиентской части приложения. Она содержит следующие файлы и папки:

* index.js: Это основной файл, который является точкой входа в приложение React. Он импортирует компонент App и отображает его в корневом элементе DOM с помощью метода ReactDOM.render().
* App.js: Это компонент, который является основным компонентом приложения и содержит все другие компоненты.
* index.scss: Это файл стилей для приложения. Он содержит общие стили для всех компонентов.
* routes.js: Файл, который содержит соответствие компонентов-страниц и адрес, по которому данная страница открывается.
* components: Это папка, которая содержит все компоненты приложения, которые используются в компонентах-страницах.
* context: Это директория, которая содержит файлы контекста react-приложения.
* hooks: Папка, которая содержит разработанные хуки.
* pages: Данная папка, содержит компоненты-страницы приложения, каждый из которых имеет свой маршрут и отображает определенный контент на странице. Компоненты-страницы используют компоненты из папки «components» для построения пользовательского интерфейса.
* styles: Это папка, которая содержит файлы шрифтов и картинок, использующиеся в данном приложении.

При разработке клиентской части на React, удобно разделять компоненты, стили и вспомогательные функции по отдельным папкам для лучшей организации кода и его поддержки.

4 Тестирование web-приложения

Тестирование программного обеспечения - это процесс, который заключается в проверке соответствия поведения программы ожидаемому результату на основе выборки тестов. Одной из целей тестирования является обнаружение и предотвращение возникновения исключительных ситуаций, которые могут привести к некорректной работе программы. Функциональное тестирование является одним из видов тестирования, которое проверяет, насколько точно и полно функции программного продукта реализованы. Это позволяет оценить, насколько программное средство способно выполнять необходимые задачи в заданных условиях. В моей разрабатываемой программе предусмотрены механизмы, которые направлены на предотвращение возникновения таких исключительных ситуаций и обеспечение корректной работы программного средства.

Рассмотрим регистрацию пользователя. Приложение проверяет корректно ли введены данные и выводит соответствующее сообщение, представлено на рисунке 4.1



Рисунок 4.1 – Ошибка некорректных данных

Все поля в окнах авторизации и регистрации являются обязательными к заполнению, пример ошибки заполнения представлен на рисунке 4.2.

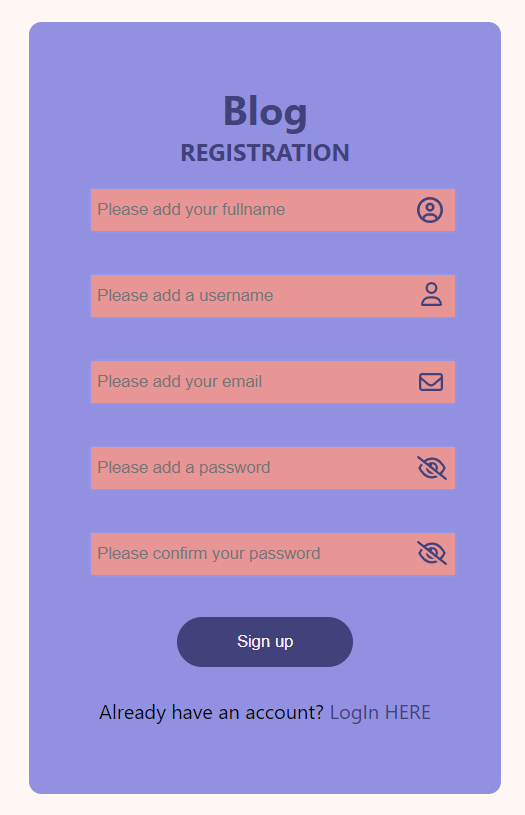


Рисунок 4.2 – Ошибка запылёнения полей

Кроме этого поля также проверяются на корректное заполнение, пример на рисунке 4.3. Примером такого валидируемого поля может являться поле email в окне регистрации.



Рисунок 4.3 – Ошибка заполнения поля email

Поле username является уникальным, следовательно, при регистрации уже существующего пользователя, сервер ответит ошибкой (рис. 4.4).

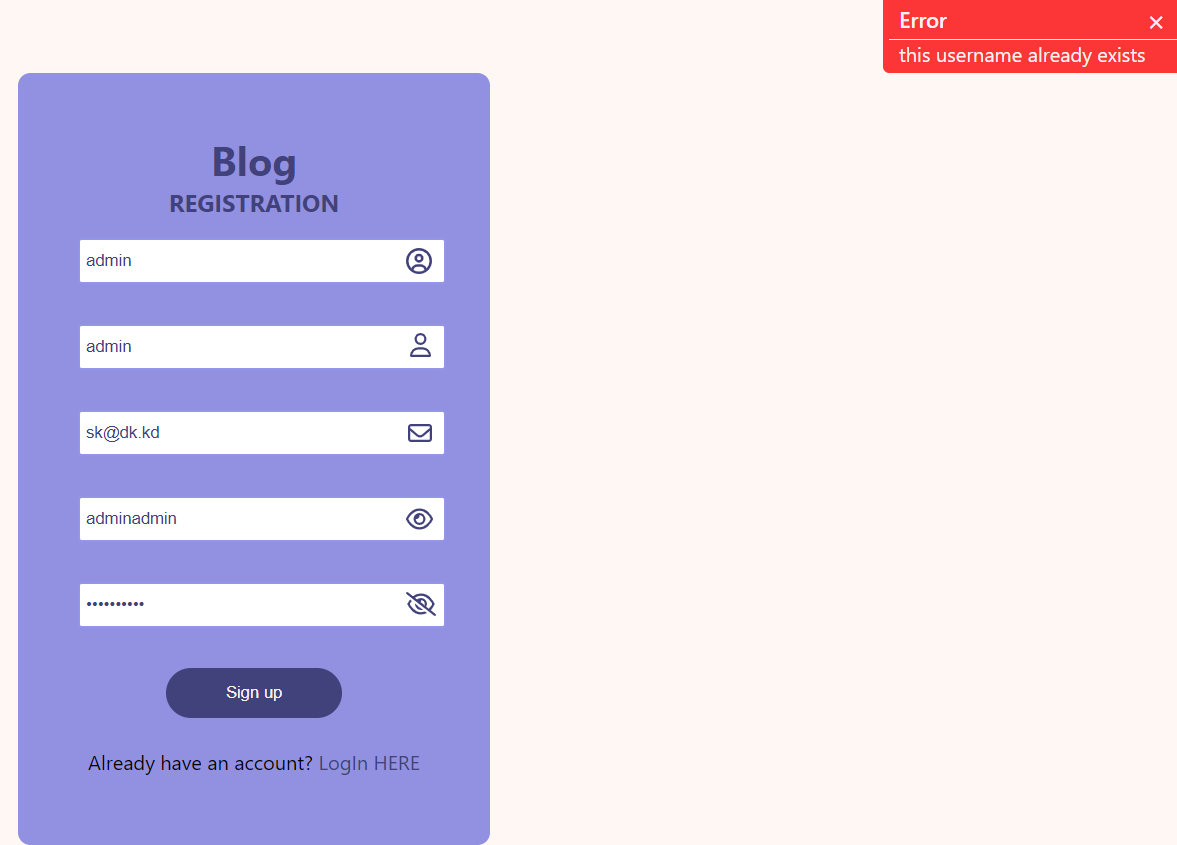


Рисунок 4.4 – Регистрация существующего пользователя

После успешной авторизации пользователь попадает на главную страницу, где ему доступна возможность создать новый пост. При создании поста, пользователь может прикрепить фотографию и добавить текстовое описание. Однако, при создании поста, обязательно должно быть заполнено только поле с фотографией. Если пользователь попытается создать пост без прикрепленной фотографии, он увидит соответствующее сообщение об ошибке (рис. 4.5).

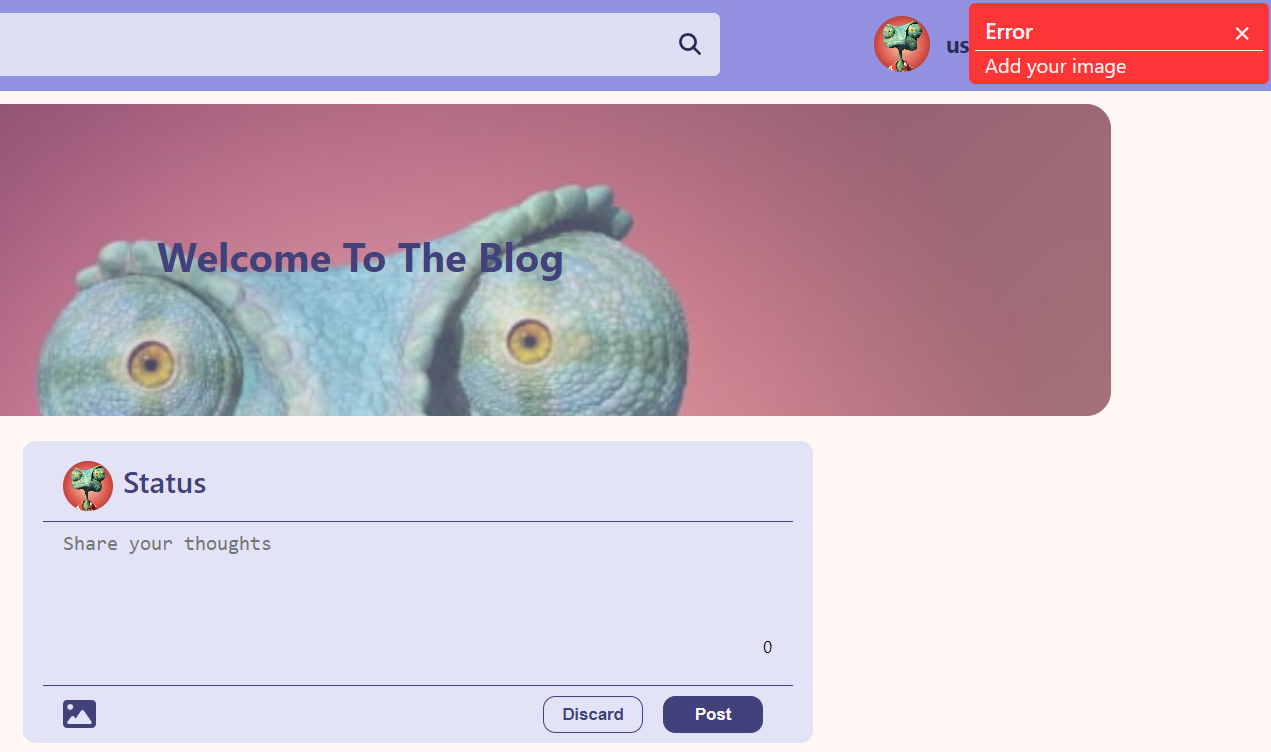


Рисунок 4.5 – Ошибка публикации поста

Пользователь имеет право изменять свои посты, однако если удалить все фотоиз поста выведется сообщение как на рисунке 4.6.

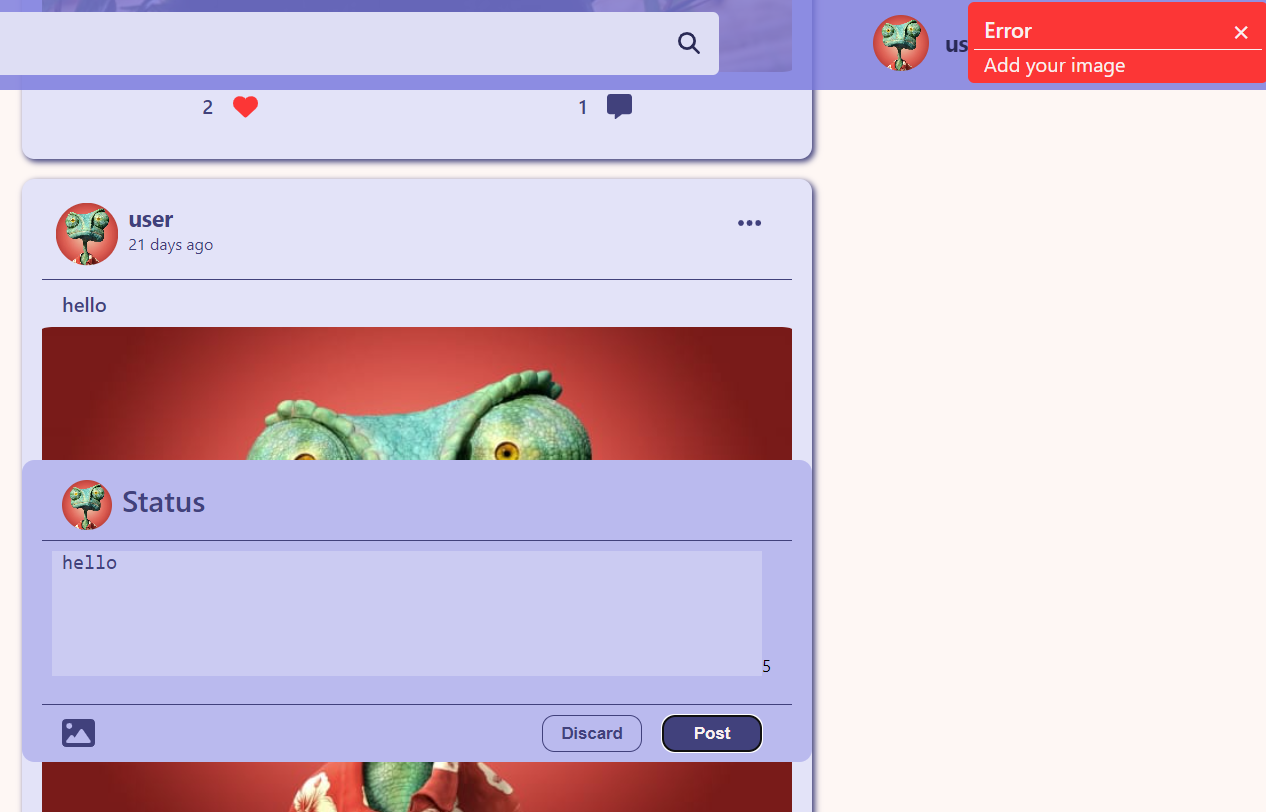


Рисунок 4.6 – Ошибка изменения поста

На странице профиль можно изменить информацию о себе и аватар, для аватара поддерживаются расширения jpeg и png, если фото не имеет данного расширения выведется ошибка представленная на рисунке 4.7.

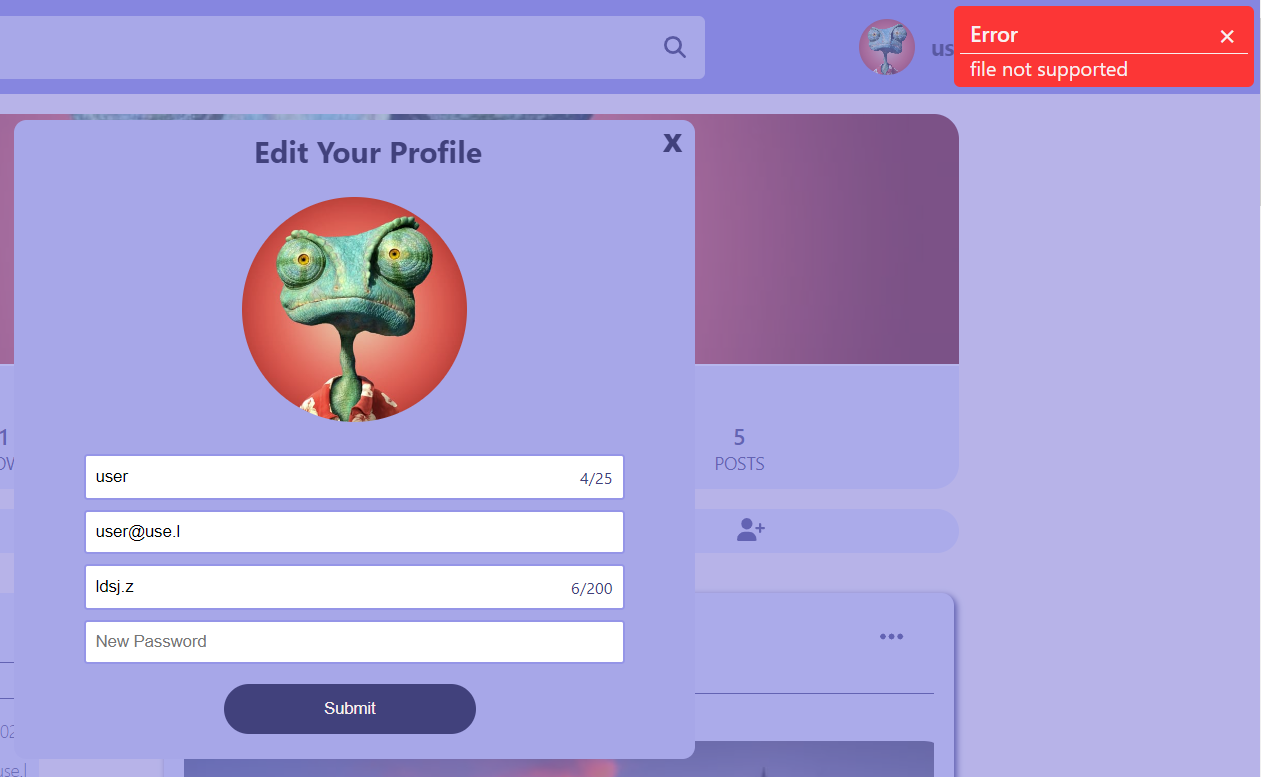


Рисунок 4.7 – Ошибка расширения

Администратор также может столкнуться с ошибками при использовании программного средства, как и любой другой пользователь. Например, при попытке добавить пост фото или при изменении собственного профиля.

5 Руководство пользователя

## 5.1 Регистрация пользователя

Для регистрации необходимо заполнить форму, которая продемонстрирована на рисунке 5.1.

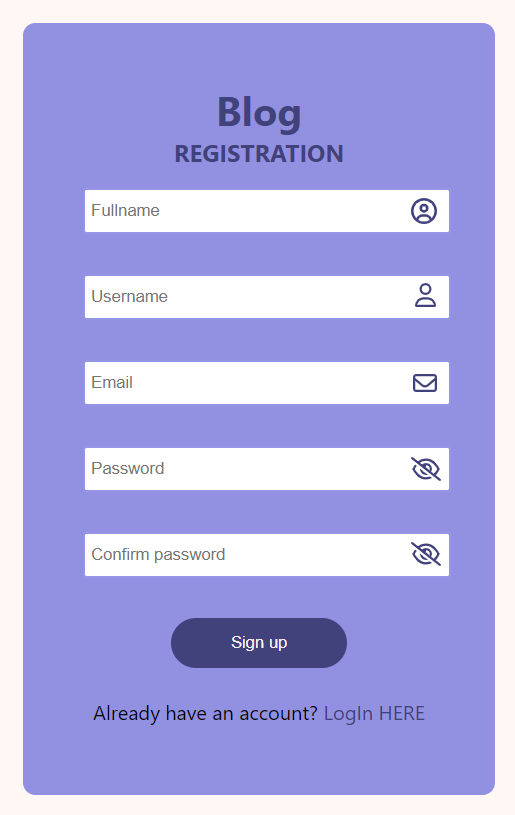


Рисунок 5.1 – Форма регистрации

После отправки пользователем запроса на регистрацию, происходит валидация всех полей формы. Если валидация будет неуспешной, пользователь увидит сообщения об ошибках в соответствующих полях для ввода информации. В случае успешной регистрации пользователь переадресовывается на главную страницу приложения.

## 5.2 Аутентификация и авторизация пользователя

Для аутентификации необходимо ввести пароль, логин и нажать на кнопку «Войти», что позволит проверить, существует ли пользователь с введенными данными в системе. Все поля являются обязательными для заполнения.

Если аутентификация прошла успешно и соблюдены все выше описанные правила, то происходит перенаправление на главную страницу, что означает, что пользователь был успешно авторизован. Скриншот формы авторизации представлен на рисунке 5.2.

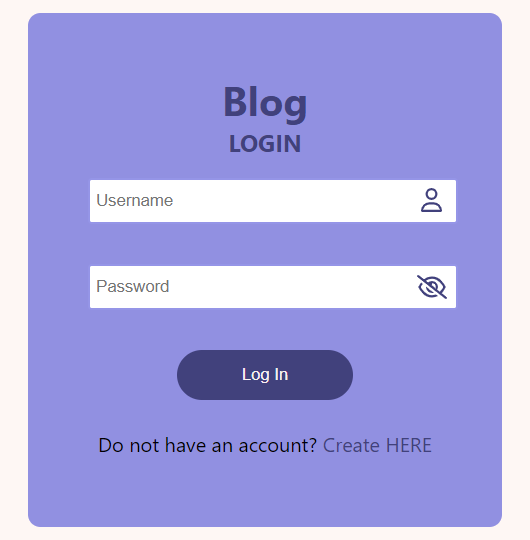


Рисунок 5.2 – Форма авторизации

При входе в систему авторизированному пользователю выдается токен, который записывается в cookies, тем самым позволяя разграничить возможности пользователей на сайте.

В данном курсовом проекте логика приложения задумана так, что пользователь, который не вошел в систему имеет доступ только к страницам регистрации и авторизации. Если пользователь авторизовался, то у него появляется доступ к возможностям приложения.

## 5.3 Просмотр постов

Для того, чтобы пользователь смог увидеть сообщения, ему достаточно зайти на сайт (рисунок 5.3).

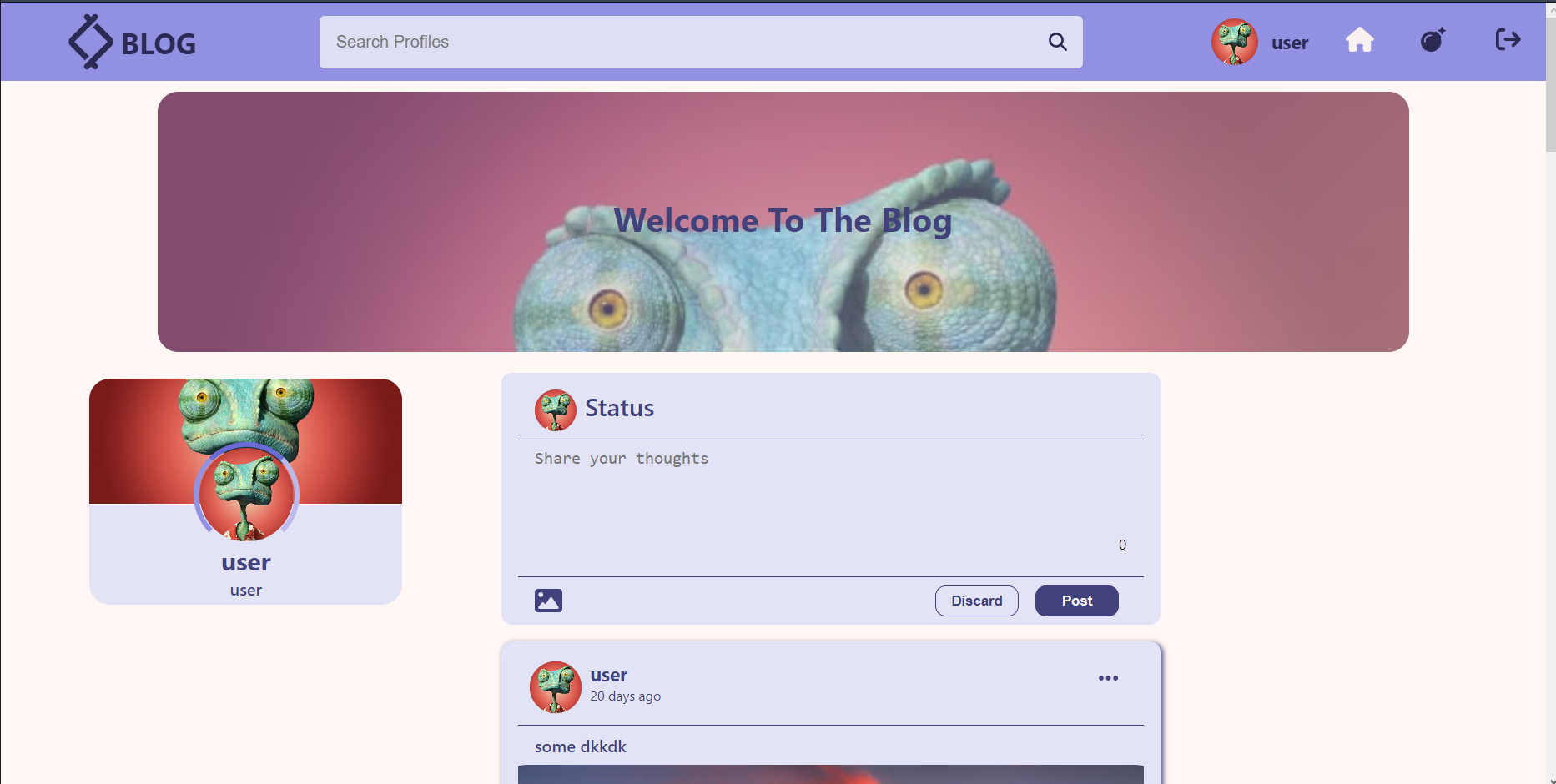


Рисунок 5.3 – Главная страница приложения

На данной странице также можно просматривать посты свои и своих друзей, добавлять посты.

Просматривать свои посты можно в собственном профиле (рис 5.4).

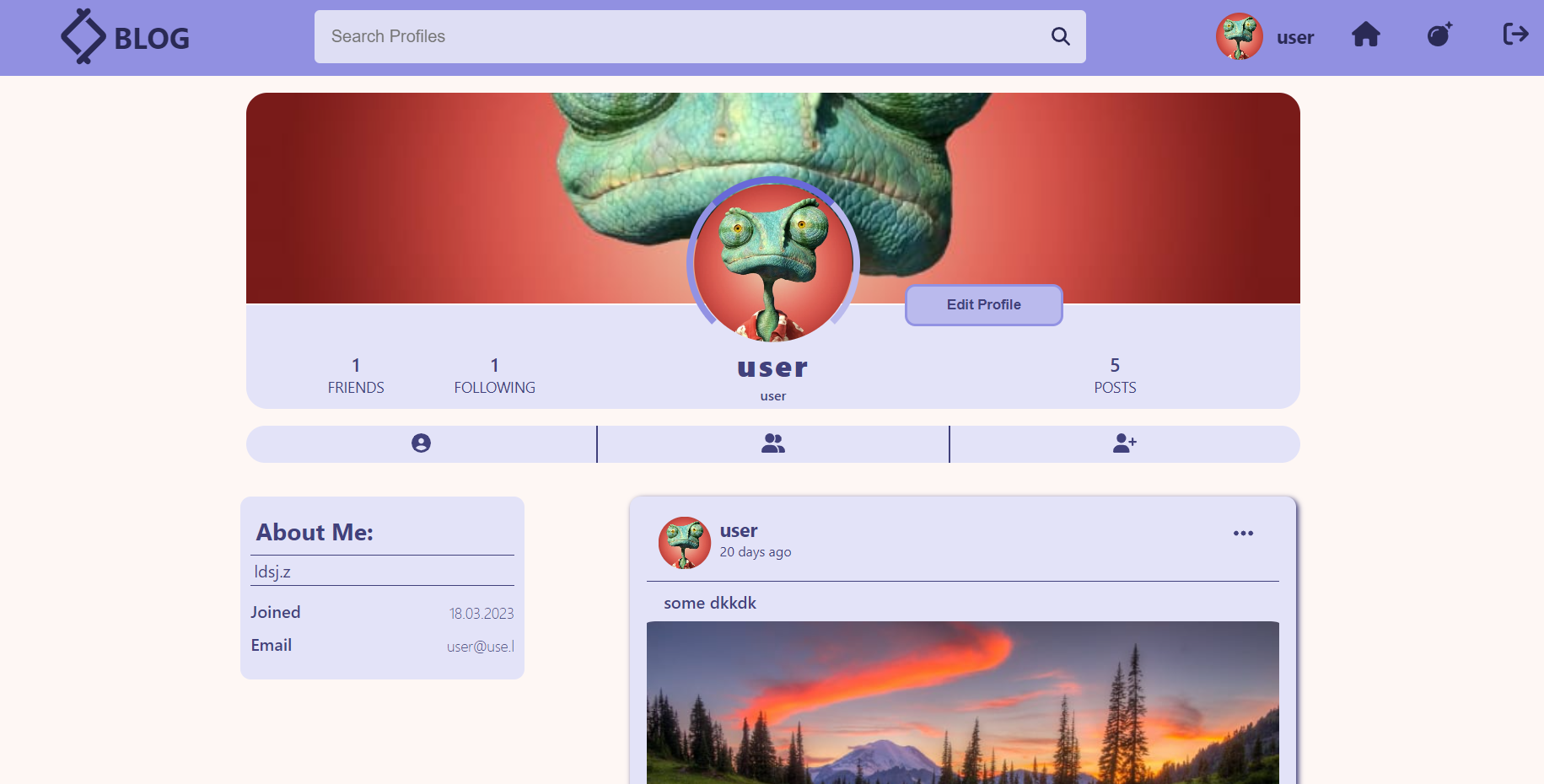


Рисунок 5.4 – Страница профиля

На данной странице также можно изменить профиль, нажав кнопку «Edit Profile», просмотреть своих друзей и подписчиков. Если перейти на страницу чужого профиля, то на месте изменения профиля появиться кнопка «Add Friend», которая позволяет подписаться на профиль или «Un Friend» чтобы отписаться от профиля. Скриншот демонстрирующий эту страницу представлен на рисунке 5.5.

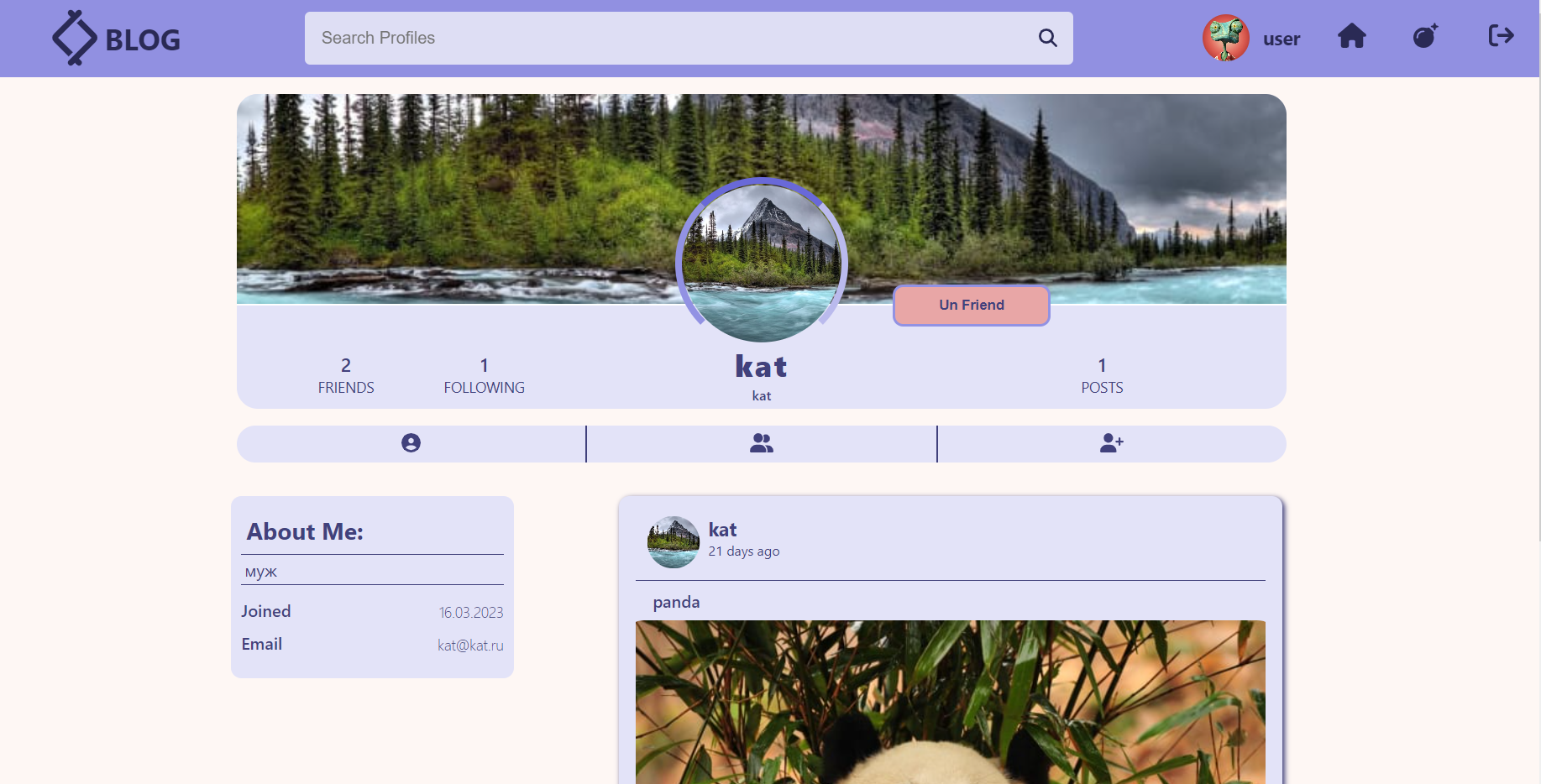


Рисунок 5.5 – Страница чужого профиля

Кроме кнопки эта страница ничем не отличается от профиля авторизованного пользователя.

## 5.4 Действия с постами

Добавить профиль можно на главной странице в форме, представленной на рисунке 5.6.

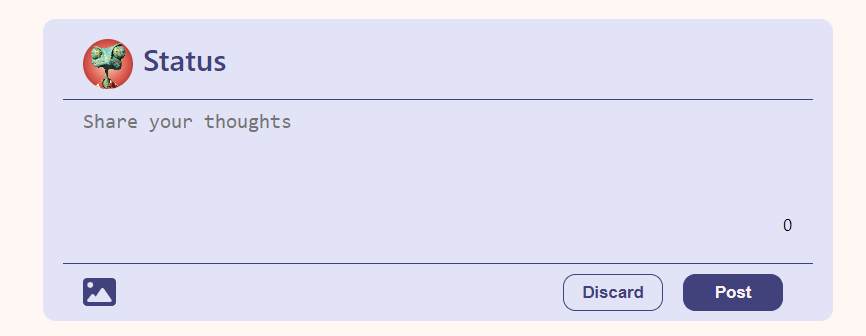


Рисунок 5.6 – Форма добавления поста

В текстовое поле можно добавить описание к посту, обязательно нужно нажать на иконку картинки, чтобы добавить фото к посту, для публикации поста следует нажать «Post».

Чтобы удалить или изменить пост нужно нажать 3 точки, как показано на рисунке 5.7 и выбрать нужное действие.



Рисунок 5.7 – Пост

Изменять и удалять можно только собственные посты, у чужих нет этих 3 точек.

Для того чтобы отметить понравившийся пост нужно нажать на сердечко, если вам оно разонравилось нужно нажать 2 раз на сердечко, и оно изменит цвет. Чтобы просмотреть все комментарии или оставить свой нужно нажать на иконку комментария в правом нижнем углу, как показано на рисунке 5.8.

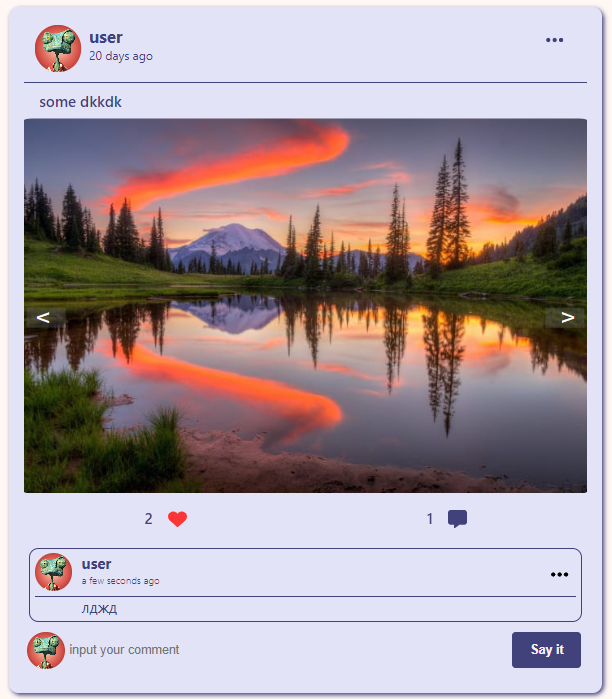


Рисунок 5.8 – Добавление комментария

Введя текст нажмите кнопку «Say it» чтобы опубликовать комментарии, изменять и удалять свои комментарии можно точно так же, как и посты, нажав три точки на нужном комментарии.

## 5.5 Администрирование

В данном курсовом проекте была разработана возможность администрирования. Администратор может удалять пользователей, посты, комментарии. Для того чтобы войти в админ панель, нужно авторизоваться как администратор и в хедере страницы нажать на иконку графика как показано на рисунке 5.9.

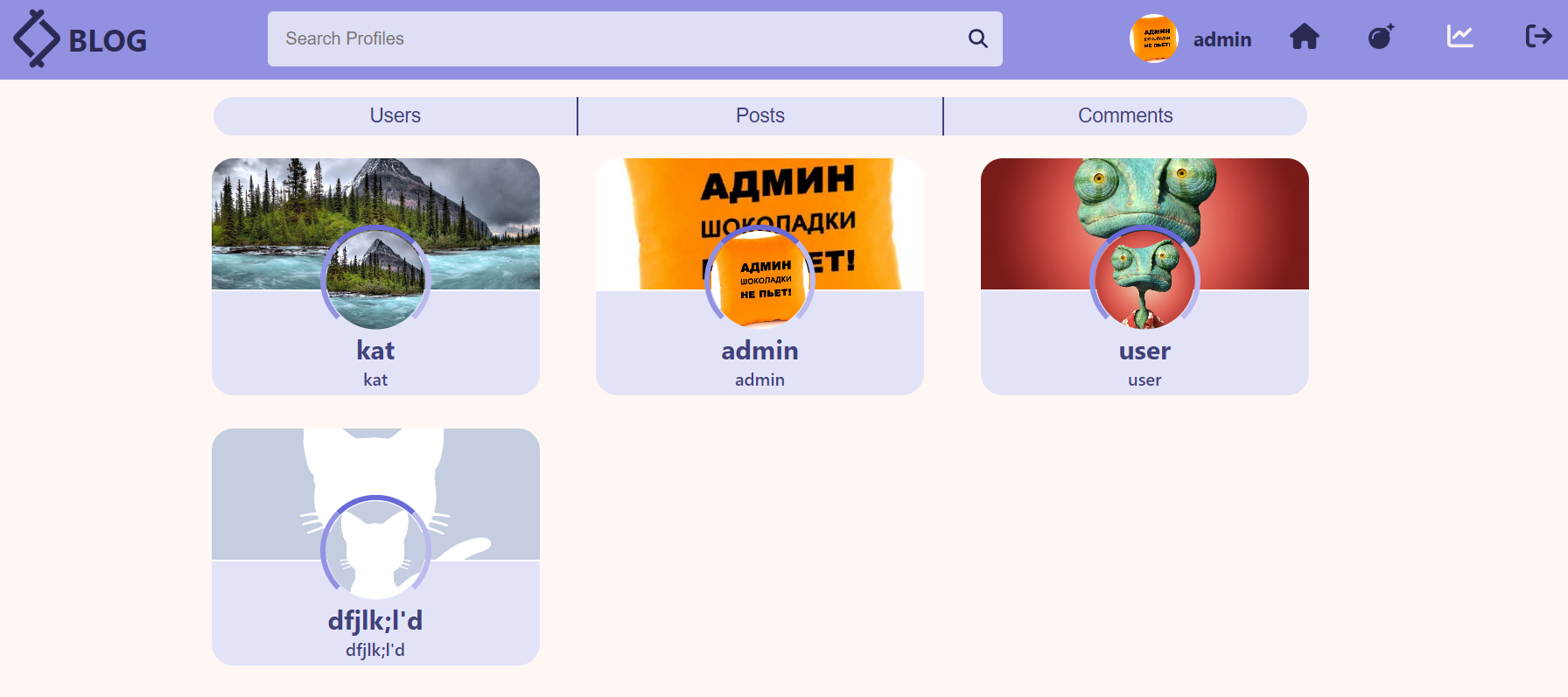


Рисунок 5.9 – Вход в панель администратора

Далее можно выбрать с чем именно вы хотите работать, например, чтобы удалить пользователя нужно нажать Users (рис. 5.10) и навести на пользователя, который хотите удалить, справа появиться крестик, нажав который пользователь удалиться.

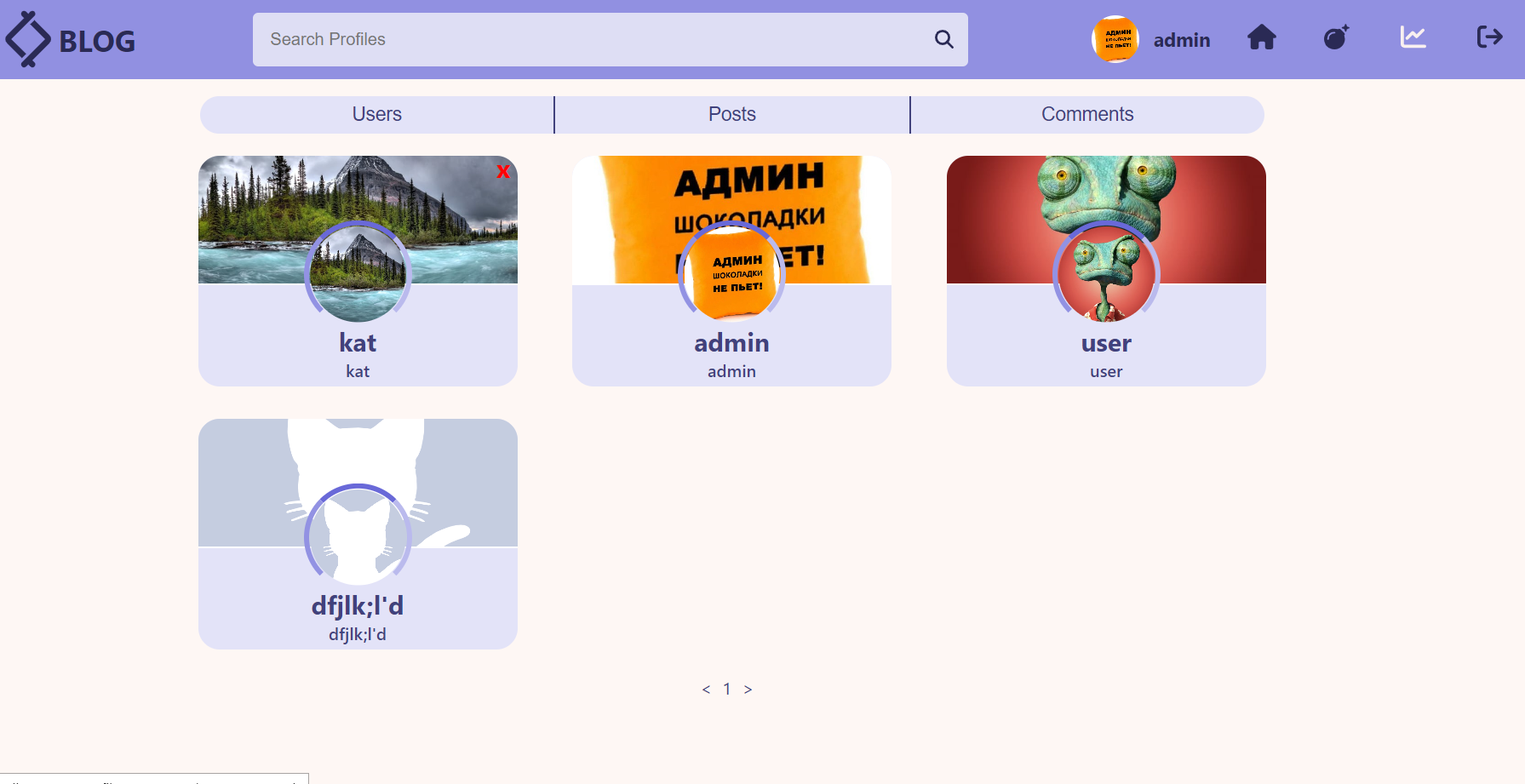


Рисунок 5.10 – Удаление пользователя

Удалить посты и комментарии можно аналогично перейдя по нужным страничкам.

# Заключение

В процессе выполнения курсовой работы было успешно достигнуто поставленное целью создание веб-приложения "Расписание транспорта". Перед началом разработки проведен анализ прототипов аналогичных приложений, а также определены функциональные возможности, которые будут реализованы в приложении.

В рамках работы была спроектирована база данных, которая использует систему управления базами данных MongoDB. Кроме того, были разработаны серверная и клиентская части приложения, включая пользовательский интерфейс. Было также проведено тестирование разработанного программного продукта.

Отдельно следует отметить, что для обеспечения безопасности и использования протокола HTTPS был создан и использован сертификат L.crt.

При разработке выполнены следующие пункты:

* регистрация и авторизация пользователя;
* изменение информации о пользователе;
* просматривание расписания транспорта;
* обмен данными между клиентом и сервером должен производиться по протоколу HTTPS;
* возможность добавления нового транспорта администратором;
* возможность покупки проездных билетов пользователем;
* получение уведомлений пользователем об истечении проездного билета;
* просмотр остановок и маршрутов на карте пользователем.

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

# Список используемых источников

1. HTTPS документация [Электронный ресурс]. – Режим доступа: <https://nodejs.org/dist/latest-v18.x/docs/api/https.html>. – Дата доступа: 08.05.2023.
2. Socket.io документация [Электронный ресурс]. – Режим доступа: <https://socket.io/docs/v4/>. – Дата доступа: 04.05.2023;
3. Документация Mongoose [Электронный ресурс].– Режим доступа: <https://mongoosejs.com/docs/guide.html> .– Дата доступа: 15.04.2023.
4. Документация React [Электронный ресурс]. – Режим доступа: <https://react.dev/learn/choosing-the-state-structure>– Дата доступа: 07.04.2023.

# Приложение А

const { Schema, model, Types } = require('mongoose');

const schema = new Schema({

dateBegin: { type: Date, required: true },

dateEnd: { type: Date, required: true },

ticketType: { type: Types.ObjectId, ref: 'TicketType' },

owner: {type: Types.ObjectId, ref: 'User'},

userNotificated: {type: Boolean}

});

module.exports = model('Ticket', schema);

const { Schema, model, Types } = require('mongoose');

const schema = new Schema({

type: { type: String, required: true },

transport: { type: String, required: true },

duration: { type: Number, required: true },

tripCount: { type: Number, required: true },

price: { type: Number, required: true }

});

module.exports = model('TicketType', schema);

const { Schema, model, Types } = require('mongoose');

const schema = new Schema({

type: { type: String, required: true },

number: { type: String, required: true },

routeStops: [{ type: Types.ObjectId, ref: 'RouteStop' }]

});

module.exports = model('Transport', schema);

const { Schema, model, Types } = require('mongoose');

const schema = new Schema({

name: { type: String, required: true },

latitude: { type: Number, required: true },

longitude: { type: Number, required: true }

});

module.exports = model('Stop', schema);

const { Schema, model, Types } = require('mongoose');

const schema = new Schema({

scheduleNumber: { type: Number, required: true },

arrivalTime: { type: String, required: true },

routeStopId: { type: Types.ObjectId, ref: 'RouteStop' }

});

module.exports = model('Schedule', schema);

const { Schema, model, Types } = require('mongoose');

const schema = new Schema({

stopOrder: { type: Number, required: true },

transportId: { type: Types.ObjectId, ref: 'Transport' },

stopId: {type: Types.ObjectId, ref: 'Stop'}

});

module.exports = model('RouteStop', schema);

# Приложение Б

const { Router } = require('express');

const bcrypt = require('bcryptjs');

const jwt = require('jsonwebtoken');

const uuid = require('uuid');

const User = require('../models/User');

const config = require('config');

const TokenService = require('../services/token.service');

const MailService = require('../services/mail.service');

const auth = require('../middleware/auth.middleware');

const admin = require('../middleware/admin.middleware');

const router = Router();

// /api/auth/login

router.post(

'/login',

async (req, res) => {

try {

//console.log(req.body);

const { email, password } = req.body;

const user = await User.findOne({ email: email.toLowerCase() });

if (!user) {

return res.status(400).json({ message: 'Карыстальнiк не знойдзены' });

}

if(!user.isActivated) {

return res.status(400).json({ message: 'Праверце пошту, каб актываваць акаунт' });

}

const isMatch = await bcrypt.compare(password, user.password);

if (!isMatch) {

return res.status(400).json({ message: 'Няправiльны пароль, паспрабуйце зноў' });

}

const tokens = TokenService.generateTokens({ id: user.\_id, role: user.role });

await TokenService.saveToken(user.\_id, tokens.refreshToken);

res.cookie('refreshToken', tokens.refreshToken, { maxAge: 30 \* 24 \* 60 \* 60 \* 1000, httpOnly: true, secure: true, sameSite: 'none' });

res.json({ token: tokens.accessToken, user: {id: user.id, role: user.role} });

} catch (e) {

console.log(e);

res.status(500).json({ message: 'Что-то пошло не так, попробуйте снова' });

}

});

// /api/auth/logout

router.post('/logout', async (req, res) => {

try {

const { refreshToken } = req.cookies;

const user = await User.findOne({ refreshToken });

user.refreshToken = '';

await user.save();

res.clearCookie('refreshToken');

return res.json({ message: 'Вы вышлі з акаунта' });

} catch (e) {

console.log(e);

res.status(500).json({ message: 'Что-то пошло не так' });

}

});

// /api/auth/refresh

router.post('/refresh', async (req, res) => {

try {

const { refreshToken } = req.cookies;

if (!refreshToken) {

return res.status(401).json({ message: '' });

}

const userData = jwt.verify(refreshToken, config.get('jwtRefreshSecret'));

console.log(userData)

const userFromDb = await User.findOne({ refreshToken });

const users = await User.find({});

if (!userData || !userFromDb) {

console.log('refresh' + userData + userFromDb)

return res.status(401).json({ message: 'Карыстальнiк не аўтыразаваны' });

}

const tokens = TokenService.generateTokens({ id: userFromDb.\_id });

res.json({ token: tokens.accessToken });

} catch (e) {

if(e instanceof jwt.TokenExpiredError) {

return res.status(401).json({ message: 'refresh jwt expired' });

}

console.log('refresh' + e);

res.status(401).json({ message: 'Карыстальнiк не аўтыразаваны' });

}

});

// /api/auth/activate/:link

router.get('/activate/:link', async (req, res) => {

try {

const activationLink = req.params.link;

const user = await User.findOne({ activationLink });

if (!user) {

return res.status(400).json({ message: 'Няправiльная спасылка актывацыi' });

}

user.isActivated = true;

await user.save();

res.redirect(`${config.get('clientUrl')}/login`);

} catch (e) {

console.log(e);

res.status(500).json({ message: 'Что-то пошло не так' });

}

});

// /api/auth/userrole

router.get('/userrole', auth, async (req, res) => {

try {

const token = req.headers.authorization.split(' ')[1];

const decoded = jwt.verify(token, config.get('jwtAccessSecret'));

const user = await User.findOne({ \_id: decoded.id });

res.json({ role: user.role, id: user.\_id });

} catch (e) {

//console.log(e);

res.status(500).json({ message: 'Что-то пошло не так' });

}

});

module.exports = router;

# Приложение В

const router =require('express').Router();

const authCtrl =require('../controllers/authCtrl')

router.post('/register', authCtrl.register);

router.post('/login', authCtrl.login);

router.post('/logout', authCtrl.logout);

router.post('/refresh\_token', authCtrl.generateAccessToken);

module.exports = router;

const router = require('express').Router()

const auth = require('../middlewares/auth')

const commentCtrl =require('../controllers/commentCtrl')

router.post('/comment',auth,commentCtrl.createComment)

router.patch('/comment/:id',auth,commentCtrl.updateComment)

router.delete('/comment/:id',auth,commentCtrl.deleteComment)

module.exports = router;

const router = require('express').Router()

const auth = require('../middlewares/auth')

const friendCtrl =require('../controllers/friendCtrl')

router.get('/fri/:id',auth,friendCtrl.getFriends)

router.get('/friends/:id',auth,friendCtrl.countOfFriends)

router.get('/followers/:id',auth,friendCtrl.countOfFollowers)

router.post('/user/:id/friend',auth, friendCtrl.friend)

router.post('/user/:id/unfriend',auth, friendCtrl.unfriend)

module.exports =router;

const router = require('express').Router()

const auth = require('../middlewares/auth')

const userCtrl =require('../controllers/userCtrl')

router.get('/search',auth,userCtrl.searchUser)

router.get('/user/all',auth,userCtrl.allUser)

router.get('/user/:id',auth,userCtrl.getUser)

router.patch('/user',auth, userCtrl.updateUser)

router.delete('/user/:id',auth, userCtrl.deleteUser)

module.exports =router;