**Министерство образования и науки Кыргызской Республики**

**Кыргызский Государственный Технический Университет**

**им. И. Раззакова**

**Институт электроники и телекоммуникаций**

**кафедра «*Информационные системы и технологии***

***им. акад. А. Жайнакова»***

**направления: 710200 Информационные системы и технологии**

**КУРСОВОЙ ПРОЕКТ**

**по дисциплине**

**«Проектирование и разработка веб-приложений BACKEND»**

**на тему «Веб-приложение для ведения дневника»**

**Разработал: студент гр. ИСТ(б)-1-23 \_\_\_\_\_\_\_\_\_\_\_\_ А.К.Аккыярова**

**(подпись)**

**Проверила: к.ф.-м.н., доцент \_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.Н.Аманкулова**

**(подпись)**

**Бишкек-2025**

**ЗАДАНИЯ**

на выполнение курсового проекта по дисциплине

**«Проектирование и разработка веб-приложений (Backend)»**

**Студент:** Аккыярова Кымбат Аккыяровна

**Группа:** ИСТ(б)-1-23

**Тема курсового проекта:** **«Веб-приложение для планирования поездок»**

Исходные данные для проектирования: Разработка веб-приложения, позволяющего пользователям планировать и организовать поездки, управлять маршрутами.

- Требования к функциональности: регистрация и аутентификация пользователей, создание и удаление маршрутов, сохранение информации о местах, просмотр истории поездок.

- Технические условия: приложение должно быть доступно через веб-браузер, обеспечивать быстрый отклик и хранить данные в базе данных.

- Ограничения по технологиям и инструментам: Frontend реализуется в одном файле index.html с использованием чистого JavaScript.

Backend реализуется на Node.js с использованием Express.js. В качестве базы данных используется MongoDB.

**Цель курсового проекта:** Разработка веб-приложения для планирования поездок с использованием HTML/JavaScript на клиенте и Node.js + MongoDB на сервере.

**Задачи проектирования:**

-Проектирование архитектуры веб-приложения;

-Разработка клиентской части на HTML, CSS, JavaScript;

-Разработка серверной части на Node.js и Express;

-Организация хранения данных в MongoDB;

-Обеспечение регистрации и аутентификации пользователей.

Требования к разрабатываемому приложению:

- Регистрация и вход пользователя;

-Просмотр, создание, удаление данных;

-Отправка запросов к серверу через API;

-Корректная обработка ошибок.

**Используемые технологии:**

-Frontend: HTML, CSS, JavaScript.

-Backend: Node.js, Express.js;

-База данных: MongoDB.

**График выполнения проекта:**

| Этап | Содержание работ | Срок выполнения |
| --- | --- | --- |
| 1 | Аналитический этап (исследование, постановка задачи) | 04/03/2025 |
| 2 | Проектировочный этап (архитектура, выбор технологий) | 16/03/2025 |
| 3 | Разработка приложения | 23/04/2025 |
| 4 | Тестирование и откладка | 17/05/2025 |
| 5 | Подготовка пояснительной записки | 26/02025 |
| 6 | Защита проекта | 28/05/2025 |

**Руководитель курсового проекта:**

Фамилия, инициалы: Аманкулова Н.А

Должность: Заведующая кафедрой «Информационные системы и технологии им.академика А.Жайнакова»

**Подписи:**

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /ФИО/ \_\_\_\_\_«\_\_\_\_» 20\_\_\_г.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /ФИО/ \_\_\_\_\_ «\_\_\_\_\_» 20\_\_ г.

**Содержание**

[Введение 4](#_Toc200297829)

[ГЛАВА 1. ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 5](#_Toc200297830)

[**1.1.Анализ предметной области** 5](#_Toc200297831)

[**1.2.Требования к функциональности и дизайну веб-приложения** 6](#_Toc200297832)

[**1.3.Проектирование структуры и интерфейса веб-сайта** 7](#_Toc200297833)

[**1.4.Выбор технологий и инструментов разработки** 9](#_Toc200297834)

[ГЛАВА 2. РАЗРАБОТКА И ТЕСТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ ПЛАНИРОВАНИЯ ПОЕЗДОК 10](#_Toc200297835)

[**2.1.Реализация frontend-части проекта** 10](#_Toc200297836)

[**2.1.1.Интерфейс пользователя** 14](#_Toc200297837)

[**2.1.2.Обработка формы и отображение списка поездок** 17](#_Toc200297838)

[**2.2.Реализация backend-части проекта** 18](#_Toc200297839)

[**2.2.1.Модели базы данных** 20](#_Toc200297840)

[**2.2.2.Реализация API** 21](#_Toc200297841)

[**2.2.3.Тестирование** 24](#_Toc200297842)

[Заключение 26](#_Toc200297843)

[Список использованных источников 27](#_Toc200297844)

# **Введение**

В условиях стремительного развития информационных технологий и цифровизации различных сфер жизни всё большую популярность приобретают веб-приложения, обеспечивающие пользователям возможность взаимодействовать с данными в режиме реального времени, независимо от их местоположения. Особенно востребованными становятся приложения, помогающие в планировании и организации повседневной деятельности, включая путешествия. Веб-приложение для планирования поездок представляет собой современный инструмент, позволяющий пользователю создавать маршруты, добавлять достопримечательности, указывать даты поездок, просматривать информацию на карте, а также сохранять детали путешествий. Такое решение упрощает процесс подготовки к поездке и делает его более структурированным и персонализированным. Целью данного курсового проекта является разработка и реализация веб-приложения «Планировщик поездок», которое предоставляет пользователю возможность создавать и редактировать маршруты, взаимодействовать с интерфейсом для визуализации на карте, а также сохранять данные с помощью удалённой базы данных. Проект реализован с использованием современных веб-технологий: Node.js и Express.js для разработки серверной части, MongoDB для хранения и управления данными, HTML и CSS для создания пользовательского интерфейса. Карты и геолокационные функции реализуются с помощью библиотеки Leaflet.js. Взаимодействие между клиентом и сервером осуществляется по REST API, обеспечивая чёткую архитектуру и расширяемость приложения. В рамках проекта были поставлены следующие задачи: проанализировать требования к функциональности приложения; спроектировать архитектуру веб-приложения с разделением на клиентскую и серверную части; реализовать пользовательский интерфейс с отображением маршрутов и формами для ввода данных; разработать серверную часть с поддержкой REST API и маршрутов для обработки CRUD-операций; обеспечить подключение и взаимодействие с базой данных MongoDB.

# **ГЛАВА 1. ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

## **1.1.Анализ предметной области**

Планирование поездок представляет собой важный этап подготовки к путешествию, позволяющий заранее определить маршрут, продумать логистику, рассчитать время и ресурсы, а также выделить интересные места для посещения. С развитием цифровых технологий традиционные способы планирования с использованием бумажных карт и записных книжек постепенно теряют актуальность, уступая место удобным веб-приложениям. Такие цифровые инструменты обладают рядом значительных преимуществ. Во-первых, они обеспечивают доступ к информации с любого устройства — компьютера, планшета или смартфона, что особенно удобно при изменении планов в пути или в условиях ограниченного времени. Во-вторых, веб-приложения позволяют централизованно хранить все данные о маршруте, достопримечательностях, времени отправления, бронированиях и заметках, что делает организацию путешествия более эффективной [6]. Дополнительные функции, такие как отображение маршрута на карте, интеграция с календарём, использование геолокации и возможность редактирования маршрута в реальном времени, позволяют адаптировать план под изменяющиеся условия. Важным преимуществом также является удобный пользовательский интерфейс, поддержка нескольких поездок и сохранение маршрутов в базе данных. Приложение может быть полезно как для индивидуальных путешественников, так и для туристических агентств, организующих групповые туры. Таким образом, разработка «Планировщика поездок» представляет собой актуальную задачу, направленную на создание эффективного и надёжного инструмента для цифровой организации путешествий.

## **1.2.Требования к функциональности и дизайну веб-приложения**

Веб-приложение «Планировщик поездок» предназначено для создания, визуализации и управления маршрутами путешествий пользователя. Приложение должно обеспечивать интуитивно понятный и адаптивный интерфейс, надёжную работу с данными и высокий уровень безопасности. Основные функциональные возможности включают регистрацию и вход пользователя, создание новой учётной записи, авторизацию по логину и паролю, а также использование JWT-токенов для защиты пользовательских сессий [2].

Работа с маршрутами предусматривает возможность создания новой поездки с указанием названия маршрута, дат, списка достопримечательностей и краткого описания. Каждый маршрут сохраняется в базе данных MongoDB [6] и отображается в виде карточки с краткой информацией. Пользователь может открывать полное описание поездки, добавлять или удалять маршруты. Отображение маршрутов на карте реализуется с помощью библиотеки Leaflet.js, что позволяет визуализировать каждый пункт путешествия с учётом географических координат [4].

Особое внимание уделяется безопасности данных. Все введённые пользователем данные проходят валидацию на стороне клиента и сервера, используется шифрование паролей при регистрации и входе. Каждому пользователю предоставляется доступ только к своим маршрутам, что реализуется за счёт аутентификации и привязки данных к конкретному ID пользователя. Также предусмотрена возможность выхода из системы с удалением активной сессии. Вся серверная логика реализована на платформе Node.js с использованием Express.js для маршрутизации и обработки API-запросов [5].

Реализация данного проекта способствует повышению удобства планирования поездок, обеспечивая пользователям централизованный и безопасный инструмент для организации своих путешествий.

## **1.3.Проектирование структуры и интерфейса веб-сайта**

Проектирование структуры и интерфейса веб-приложения «Планировщик поездок» является важным этапом разработки, направленным на обеспечение удобства, логичности и визуальной привлекательности системы. Основная задача этого этапа — создать интуитивно понятный пользовательский интерфейс и удобную навигацию между страницами приложения.

Веб-приложение реализовано как одностраничное приложение, что позволяет загружать и отображать контент без полной перезагрузки страницы. Это повышает скорость работы приложения и улучшает пользовательский опыт[3]. Для маршрутизации между разделами используется клиентская навигация.

Основные страницы приложения включают: страницу авторизации, где пользователь вводит логин и пароль, а также имеет возможность перейти к регистрации нового аккаунта; страницу регистрации с полями для ввода имени пользователя, пароля и его подтверждения; главную страницу, на которой отображаются все поездки текущего пользователя в виде карточек с названием маршрута, датами и кратким описанием. Также на главной странице присутствуют кнопки для добавления новой поездки и удаления существующих. Для создания новой поездки предусмотрена отдельная страница с формой, включающей поля для названия маршрута, описания, дат и карты для выбора точек маршрута. Визуализация маршрутов на карте реализована с помощью библиотеки Leaflet.js, которая позволяет отображать географические координаты каждого пункта путешествия [4]. Особое внимание уделяется безопасности данных. Все введённые пользователем данные проходят валидацию на стороне клиента и сервера, обеспечивается шифрование паролей при регистрации и входе, а также использование JWT-токенов для защиты пользовательских сессий [2]. Каждому пользователю предоставляется доступ только к своим маршрутам, что достигается за счёт аутентификации и привязки данных к конкретному ID пользователя. Также реализована функция выхода из системы с удалением активной сессии. Серверная часть приложения построена на платформе Node.js с использованием Express.js, который отвечает за маршрутизацию и обработку API-запросов [5]. Для хранения данных используется база MongoDB, что обеспечивает гибкость и масштабируемость хранилища [6]. Интерфейс выполнен в современном минималистичном стиле, с использованием стандартных HTML-элементов и стилизацией через CSS. В дальнейшем планируется возможность подключения CSS-фреймворков для улучшения адаптивности интерфейса. Для шрифтов выбраны современные и читаемые варианты, такие как Roboto и Open Sans, которые обеспечивают комфортное восприятие текста [1]. Интерфейс адаптивен и корректно отображается на различных устройствах — от мобильных до десктопных.

## **1.4.Выбор технологий и инструментов разработки**

При создании веб-приложения для планирования поездок был осуществлён обоснованный выбор современных технологий и инструментов, обеспечивающих удобную разработку, масштабируемость, безопасность и высокую производительность. Выбор технологий основывался на актуальных практиках веб-разработки, а также на целях и особенностях проекта. Важным критерием стало соотношение простоты использования и функциональности, что позволило сосредоточиться на реализации ключевых возможностей приложения без излишних затрат времени на изучение сложных или малораспространённых решений [1]. Для серверной части использован Node.js платформа, обеспечивающая эффективную работу с одновременными подключениями и высокую скорость обработки запросов [2]. В связке с Node.js применён фреймворк Express.js, позволяющий удобно организовать маршрутизацию, обработку HTTP-запросов и реализацию REST API [5]. Это обеспечивает гибкую и расширяемую серверную логику, позволяющую обрабатывать действия пользователей, управлять сессиями и защищать приватные маршруты. Для отображения интерактивных карт и маршрутов использована библиотека Leaflet.js, которая предоставляет простой и функциональный API для работы с картографическими данными [4].

В качестве базы данных выбрана MongoDB — документно-ориентированная NoSQL база, которая хорошо подходит для хранения разнообразной информации о поездках, маршрутах и пользователях [6]. MongoDB хранит данные в формате JSON-подобных документов, что упрощает работу с данными в JavaScript-приложении и обеспечивает гибкость структуры. Для взаимодействия с базой данных используется нативный драйвер MongoDB, что позволяет реализовать все необходимые CRUD-операции напрямую, без дополнительных библиотек. Для обеспечения безопасности реализована аутентификация с использованием JSON Web Token (JWT), что защищает маршруты от несанкционированного доступа и позволяет надёжно идентифицировать пользователей. Для тестирования API и отладки запросов активно используется Postman — инструмент, позволяющий отправлять HTTP-запросы к серверу, просматривать ответы и анализировать структуру данных. Это ускоряет выявление ошибок и упрощает разработку. Архитектура проекта разделена на логические части: backend — серверная логика и API, база данных — хранение информации, что соответствует современным архитектурным принципам и облегчает сопровождение и расширение приложения. Также соблюдаются стандарты REST, что делает API понятным, удобным для использования и масштабируемым [1].

# **ГЛАВА 2. РАЗРАБОТКА И ТЕСТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ ПЛАНИРОВАНИЯ ПОЕЗДОК**

## **2.1.Реализация frontend-части проекта**

Frontend-часть веб-приложения *«*Планировщик поездок*»* разработана с использованием HTML\CSS представляет собой одностраничное приложение (SPA — Single Page Application), обеспечивающее быструю и интуитивно понятную навигацию без перезагрузки страницы. Все компоненты интерфейса разработаны с нуля, без использования CSS-фреймворков, с акцентом на чистый дизайн и удобство пользователя [1].

**Структура проекта**

Frontend проекта расположен в директории frontend и имеет следующую структуру:

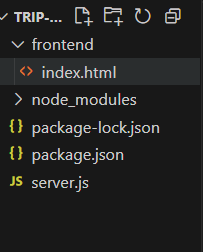


Рис.2.1.1. Структура папок frontend части.

CSS:

1. Общие стили и фон страницы:

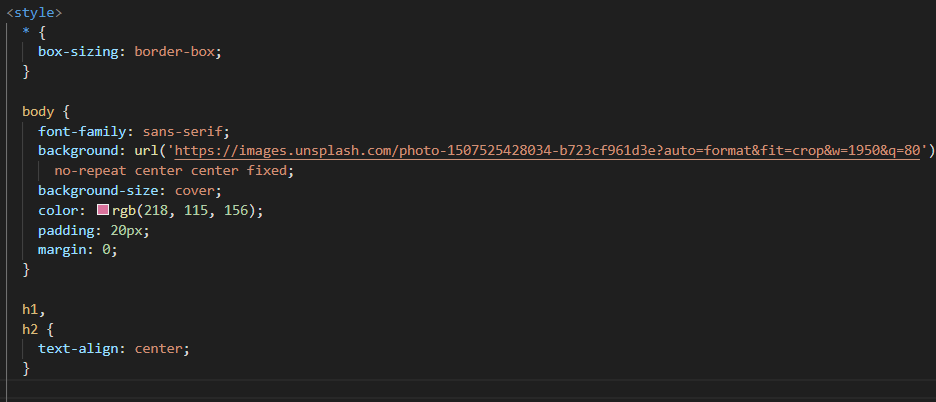


Рис.2.1.2.Стили в index.html.

-Задаёт шрифт, отступы, фон сайта.

-box-sizing: border-box позволяет проще управлять размерами элементов.

-Картинка задаётся фоном на всю страницу.

-Заголовки центрированы.

2. Оформление блоков, полей и кнопок:

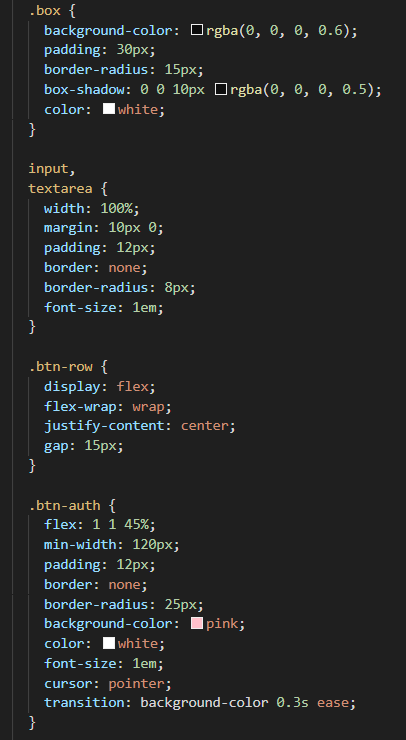


Рис.2.1.3.Стили в index.html. Рис.2.1.4.Стили в index.html.

-Все формы, списки, поля и кнопки красиво оформлены: скругления, тени, фоновый цвет.

-transition делает плавную анимацию при наведении. Элементы формы-адаптивные и удобные для ввода.

3. Макет страницы: форма, список и карта:

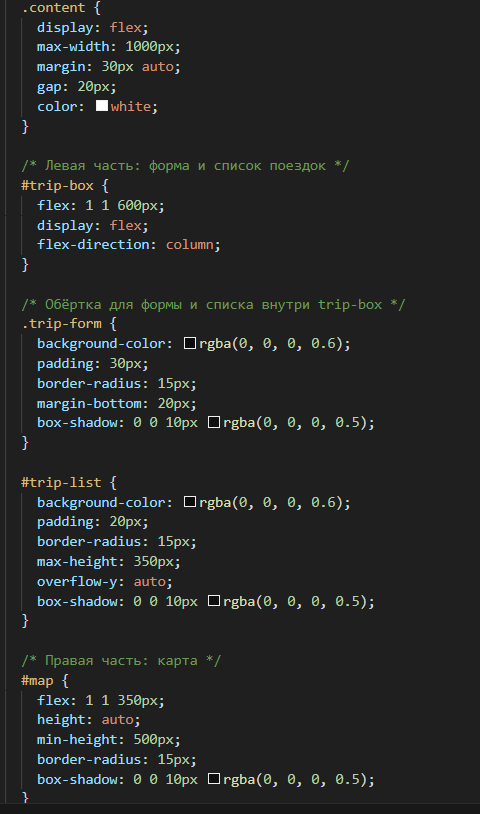


Рис.2.1.5.Стили в index.html.

-Страница делится на две части: слева форма и список поездок, справа — карта.

-Используется Flexbox для адаптивного и гибкого размещения элементов. Карта и формы имеют одинаковый визуальный стиль.

4. Адаптивность для мобильных устройств

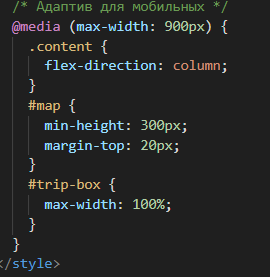


Рис.2.1.6.Стили в index.html.

-При маленькой ширине экрана (<900px) блоки перестраиваются вертикально.

-Обеспечивает удобство на смартфонах и планшетах. Карта опускается ниже формы и уменьшается по высоте.

## **2.1.1.Интерфейс пользователя**

Интерфейс веб-приложения «Планировщик поездок» разработан с учётом принципов удобства, минимализма и адаптивности. Он состоит из нескольких ключевых элементов, обеспечивающих простую и логичную навигацию, а также быстрый доступ к основным функциям приложения.

**Страница входа**

Первым экраном, который видит пользователь, является страница входа. Она содержит форму с полем для ввода имени пользователя, пароля и кнопками «Войти» и «Зарегистрироваться». При вводе имени («Имя пользователя») приложение проверяет, существует ли такой пользователь. Если имя вводится впервые выходит ошибка, чтобы пользователь зарегистрировался. В таком случае, надо ввести имя, пароль и нажать на «Зарегистрироваться». Если пользователь уже зарегистрирован, он пишет свои данные и нажимает на «Войти».

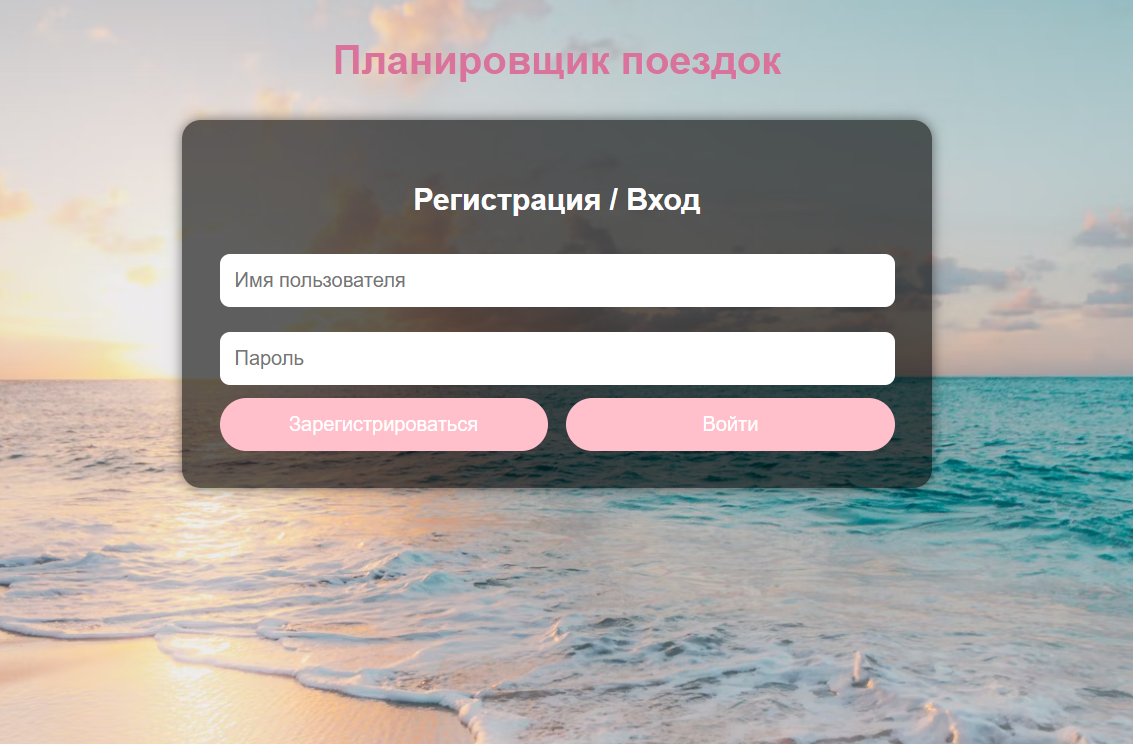


Рис.2.1.2. Страница «Планировщик поездок».

**Главная страница приложения**

После запуска приложения открывается основная страница с формой планирования поездки. Интерфейс оформлен в светлых тонах с фоновой фотографией пляжа, что создаёт приятную атмосферу.

На странице расположены следующие элементы:

- Поле “Куда” — ввод названия населённого пункта (например, “Ташкент”);

- Дата начала и окончания поездки — выбор через встроенные календарные поля;

- Поле “Описание” — дополнительная информация о поездке (цель, детали и т.п.);

- Кнопка “Создать” — добавляет поездку в список.

Под формой отображается интерактивная карта, на которой автоматически отображается выбранный город. Это помогает визуализировать маршрут и точку назначения.

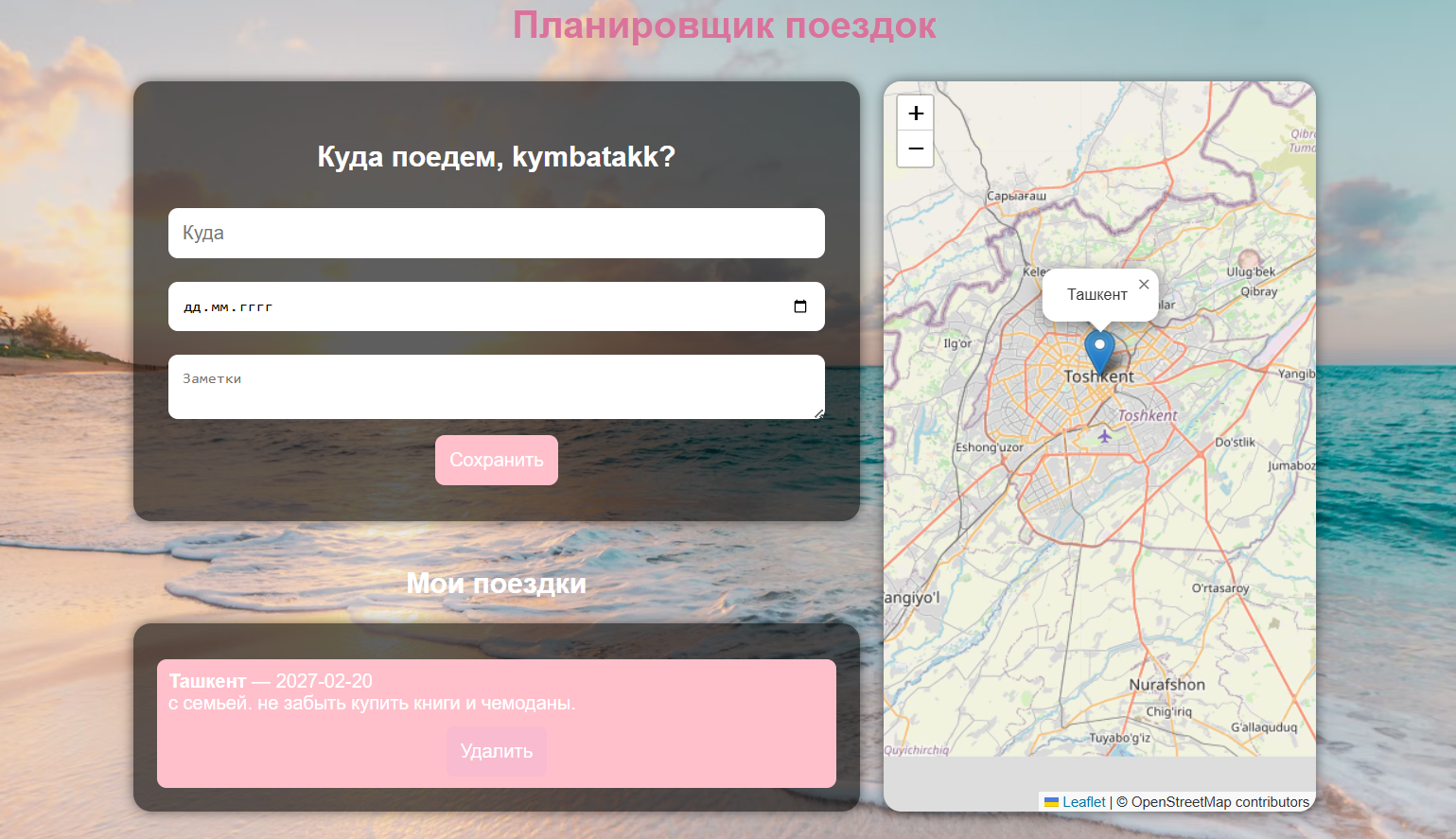


Рис.2.1.3. Интерфейс создания поездки с картой

Карта реализована через API и обновляется при вводе нового города. Это удобно для наглядного планирования.

Список поездок

Ниже формы отображаются все созданные поездки в виде карточек. Каждая карточка включает:

- Название направления (город);

- Даты;

- Краткое описание;

- Кнопка «Удалить» — для удаления поездки из списка.

Записи отображаются в виде блочных элементов с аккуратным оформлением и цветовой консистентностью с остальной частью интерфейса.

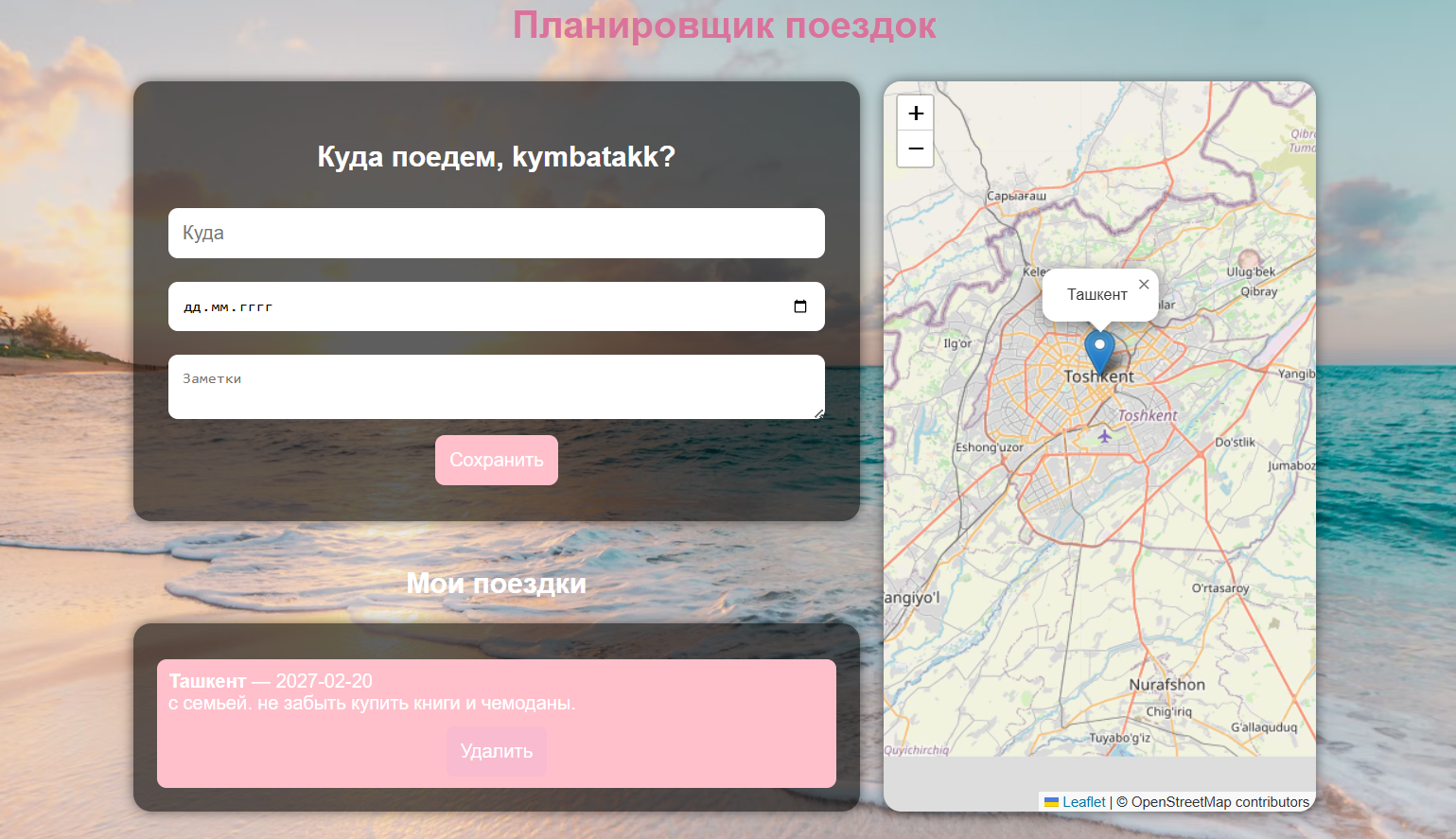


Рис. 2.1.4. Список сохранённых поездок

Это позволяет пользователю быстро просматривать все свои планы и управлять ими на одной странице.

Стилизация и адаптивность

Приложение реализовано с использованием чистого CSS. Дизайн адаптивный корректно отображается на различных устройствах и экранах. Основные особенности: удобная и логичная структура, приятные цвета и визуальные эффекты, все элементы интерфейса оформлены в едином стиле.

## **2.1.2.Обработка формы и отображение списка поездок**

Во веб-приложении для планирования поездок навигация между отдельными страницами не используется, поскольку весь интерфейс реализован в формате одностраничного приложения (SPA) на основе стандартных технологий HTML [1], CSS и JavaScript. Вся основная логика взаимодействия с пользователем сосредоточена на странице index.html, где размещена форма для ввода информации о планируемой поездке, включая пункт назначения, дату и описание, а также визуальный блок с отображением карты. Ни переходов между страницами, ни перезагрузки не требуется — все действия выполняются динамически с помощью JavaScript. После заполнения полей и отправки формы информация о поездке передаётся на сервер, где обрабатывается и сохраняется в базу данных. Сервер написан с использованием Node.js [2] и фреймворка Express [5], а взаимодействие с ним осуществляется с помощью асинхронных HTTP-запросов (fetch API). Список запланированных поездок отображается на той же странице под формой в виде структурированных блоков, и обновляется без перезагрузки страницы. Это позволяет пользователю в реальном времени видеть все свои записи и добавлять новые. Для краткосрочного хранения данных на стороне клиента может использоваться объект localStorage, который позволяет сохранять введённые данные даже при перезагрузке страницы, повышая удобство использования. Однако основное хранение данных осуществляется на сервере, что обеспечивает долговременное сохранение информации и возможность расширения функционала в будущем. Такой подход делает интерфейс простым и удобным, снижая сложность навигации и устраняя необходимость в сложных маршрутах или системе переходов. При этом архитектура приложения остаётся гибкой: при необходимости можно в дальнейшем добавить отдельные страницы, например, для авторизации, статистики или фильтрации поездок. Таким образом, навигация и управление состоянием в приложении реализованы с использованием простых и надёжных решений, обеспечивая удобную работу пользователю и стабильную передачу данных между клиентом и сервером.

## **2.2.Реализация backend-части проекта**

Backend-часть проекта отвечает за обработку запросов, хранение и управление данными пользователей и их списками поездок. Она реализована с использованием **Node.js** [2], а база данных — **MongoDB** [6] с использованием ODM-библиотеки **Mongoose**. Это обеспечивает гибкую работу с документами, удобное создание REST API и выполнение всех CRUD-операций (создание, чтение, обновление, удаление).

**Цели backend-части**

Основными задачами реализации серверной части являются:

-предоставление REST API для взаимодействия с frontend;

-реализация CRUD-функционала для дневниковых записей;

-защита маршрутов с помощью JWT-токенов;

-валидация и обработка ошибок во входящих данных;

-логическая привязка записей к конкретному пользователю.

**Используемые технологии и библиотеки**

В проекте использовались следующие технологии и библиотеки. Node.js — это среда выполнения JavaScript на сервере, которая позволяет создавать серверную часть приложения и обрабатывать запросы от пользователей. В качестве базы данных используется MongoDB — документо-ориентированная NoSQL база данных, которая хранит данные о пользователях и их поездках в удобном формате. Для удобного взаимодействия с MongoDB применяется библиотека Mongoose, которая позволяет описывать структуру данных с помощью моделей и схем, а также выполнять различные операции с базой данных. Для обеспечения безопасности и авторизации пользователей используется JWT (jsonwebtoken) — технология создания и проверки токенов, которые подтверждают личность пользователя без необходимости постоянного ввода пароля. Для тестирования работы API и проверки правильности запросов и ответов применяется инструмент Postman, который предоставляет удобный интерфейс для отправки различных HTTP-запросов и анализа ответов сервера. Все эти технологии и библиотеки вместе обеспечивают надёжную, безопасную и удобную работу веб-приложения для планирования поездок.

Использование MongoDB[6] позволяет хранить данные в формате JSON-документов, что хорошо сочетается с архитектурой REST-приложения и делает структуру гибкой и масштабируемой. Благодаря Mongoose обеспечивается валидация схем, связи между коллекциями и простое взаимодействие с базой данных через JavaScript.

Серверная и клиентская части проекта чётко разделены, что способствует лучшей читаемости кода и облегчает его сопровождение. Визуальная часть приложения расположена в папке frontend/, а логика обработки данных реализована в отдельном файле server.js.

**Папка frontend:** Содержит файл index.html, который является основным интерфейсом пользователя. В этом файле размещена форма для ввода информации о поездках, отображение списка маршрутов и другие элементы. Страница может содержать подключённые стили и JavaScript [1], которые обрабатывают действия пользователя и отправляют запросы на сервер (например, через fetch()).

**Файл server.js:** Это основной серверный файл, написанный на Node.js [1] с использованием фреймворков и библиотек. В нём настраиваются маршруты, подключается база данных MongoDB [6] через Mongoose, реализуются функции регистрации, входа в систему, добавления, редактирования и удаления поездок. Сервер принимает HTTP-запросы от клиента и отправляет соответствующие ответы — в виде JSON.

**Файл package.json**: package.json хранит информацию о проекте и список зависимостей:

-"express" — фреймворк для создания веб-сервера.

-"mongoose" — библиотека для взаимодействия с MongoDB [6].

-"cors" — модуль для настройки политики доступа между доменами.

Также указан основной файл запуска (index.js, хотя по факту ты, скорее всего, запускаешь server.js) и базовый тестовый скрипт. Пример команды запуска: node index.js

## **2.2.1.Модели базы данных**

В проекте используются две основные модели — **User** (пользователь) и **Trip** (поездка). Они реализованы через mongoose и хранятся в базе MongoDB под названием trip-planner.

**Модель пользователя (User):**

username — имя пользователя

password — пароль

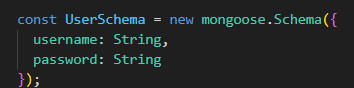


Рис. 2.2.1. Схема UserSchema

**Модель поездок (Trips):**

Модель описывает информацию о конкретной поездке, привязанной к пользователю по имени.

Поля:

-user-имя пользователя, которому принадлежит поездка;

-destination-место назначения;

-date-дата поездки;

-notes-дополнительные заметки.

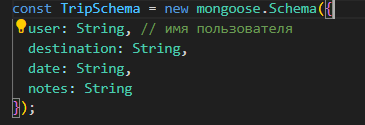


Рис. 2.2.2. Схема TripSchema

## **2.2.2.Реализация API**

REST API реализован с использованием Express.js. Все маршруты обрабатываются на стороне сервера и используют mongoose для работы с базой данных MongoDB [6].

Основные маршруты:

-/register- маршрут для регистрации нового пользователя. Принимает в теле запроса username и password. Если пользователь с таким именем уже существует, возвращается сообщение об ошибке. Иначе создаётся новый пользователь.

-/login- маршрут для авторизации пользователя. Принимает username и password. Если пара логин/пароль совпадает с существующими данными, возвращается успешный ответ. В противном случае — сообщение о неправильных данных.

-/trips- маршрут для добавления новой поездки. В теле запроса указывается имя пользователя (user), место назначения (destination), дата (date) и дополнительные заметки (notes). Создаётся новая запись в базе данных.

-/trips/:username- маршрут для получения всех поездок конкретного пользователя. Имя пользователя передаётся как параметр в URL. Возвращает массив всех поездок, связанных с данным пользователем.

-/trips/:id- маршрут для удаления поездки по её уникальному идентификатору. После удаления возвращается сообщение об успешном выполнении операции.

Пример логики:

**Регистрация:**

При получении username и password проверяется, существует ли пользователь. Если нет — создаётся новая запись:

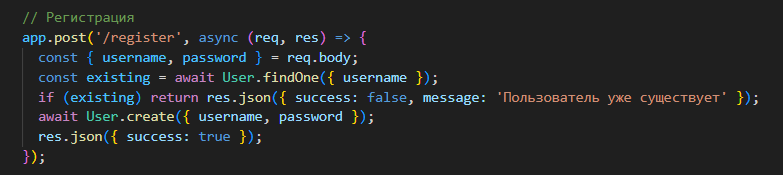


Рис. 2.2.3. Логика маршрута /register

**Авторизация:**

При входе (/login) проверяется соответствие имени пользователя и пароля. Если всё верно — возвращается успешный ответ и имя пользователя.

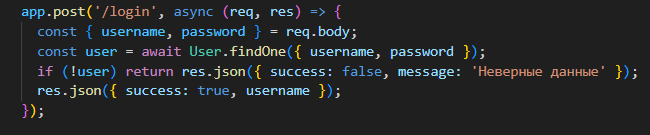


Рис. 2.2.4. Логика маршрута /login

**Добавление поездки:**

Пользователь может добавить поездку через POST-запрос на /trips. Создаётся новая запись в базе данных:

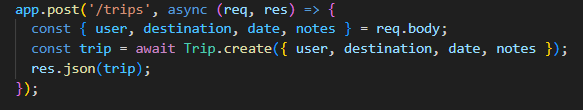


Рис. 2.2.5. Логика маршрута /trips

**Получение списка поездок:**

Для получения всех поездок пользователя отправляется GET-запрос на /trips/:username. Сервер возвращает все записи, соответствующие переданному имени.

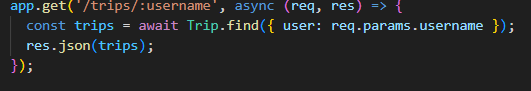


Рис. 2.2.6. Логика маршрута /trips/:username

**Удаление поездки:**

Для удаления поездки отправляется DELETE-запрос на /trips/:id, где :id — это идентификатор поездки в MongoDB. Поездка удаляется, и клиенту возвращается статус успеха.

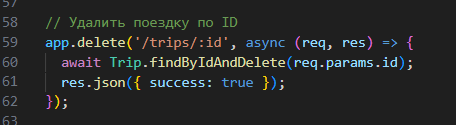


Рис. 2.2.6. Логика маршрута /trips/:id

## **2.2.3.Тестирование**

Тестирование веб-приложения для планирования поездок проводилось на локальном сервере. Бэкенд-приложение работало по адресу http://localhost:3000/, где через Postman и браузер проверялись основные функции API.

В процессе тестирования проверялись ключевые маршруты: регистрация нового пользователя, вход по имени и паролю, создание поездки, получение списка поездок по имени пользователя, а также удаление поездки по её идентификатору.

Особое внимание уделялось:

-корректности обработки данных;

-стабильности при множественных запросах;

-точности фильтрации поездок по конкретному пользователю;

-проверке удаления без сбоев;

-контролю ошибок (например, при повторной регистрации).

Тестирование проводилось вручную с использованием Postman для API-запросов. Были выполнены следующие действия:

**Выполняемое действие:** Регистрация пользователя с именем Anna

**Ожидаемый результат:** Успешная регистрация, сообщение о создании нового пользователя

**Фактический результат:** Пользователь создан, получен ответ{ success: true }

**Статус:** Успешно

**Выполняемое действие:** Повторная регистрация с тем же именем Anna

**Ожидаемый результат:** Ошибка: пользователь уже существует

**Фактический результат:** { success: false, message: 'Пользователь уже существует' }

**Статус:** Успешно

**Выполняемое действие:** Вход пользователя Anna с правильным паролем

**Ожидаемый результат:** Успешный вход, получение имени пользователя

**Фактический результат:** Получен ответ { success: true, username: 'Anna' }

**Статус:** Успешно

**Выполняемое действие:** Добавление поездки пользователем Anna

**Ожидаемый результат:** Создание новой поездки, отображение данных о маршруте

**Фактический результат:** Поездка сохранена и возвращена в ответе

**Статус:** Успешно

**Выполняемое действие:** Получение всех поездок пользователя Anna

**Ожидаемый результат:** Список поездок, связанных с пользователем

**Фактический результат:** Корректный JSON-массив с сохранёнными поездками

**Статус:** Успешно

**Выполняемое действие:** Удаление поездки по её ID

**Ожидаемый результат:** Поездка удалена, больше не отображается в списке

**Фактический результат:** { success: true }, данные не возвращаются при повторном запросе

**Статус:** Успешно

Проведённые тесты подтвердили, что API работает корректно. Запросы обрабатываются без критических ошибок, данные сохраняются и удаляются по назначению. Обработка ошибок реализована на базовом уровне: при повторной регистрации или неправильном входе пользователь получает понятное сообщение. Текущая версия не использует токены авторизации, поэтому защита данных реализована только на уровне пользовательского фильтра (по имени). Для повышения безопасности в будущем планируется внедрение JWT-аутентификации и привязка поездок к конкретным ID пользователей, а не просто к строковому имени. Тестирование показало, что приложение работает стабильно и готово к дальнейшему развитию и улучшению функционала.

# **Заключение**

В ходе выполнения курсового проекта была разработана полноценная веб-система для планирования поездок. Приложение предоставляет пользователю простой и понятный интерфейс для создания маршрутов, их просмотра и управления ими. Также реализованы регистрация и вход в систему, что позволяет каждому пользователю работать со своими личными данными.

Серверная часть разработана с использованием платформы Node.js и фреймворка Express, а для хранения данных применена база данных MongoDB. Через REST API реализованы основные функции: регистрация, вход, добавление, получение и удаление поездок. Обеспечена валидация данных, базовая проверка авторизации и фильтрация по имени пользователя. Хотя текущая реализация не использует JWT, архитектура приложения позволяет легко добавить полноценную систему аутентификации в будущем. Клиентская часть представлена в виде простой HTML-страницы, отправляющей асинхронные запросы к серверу с помощью JavaScript. Интерфейс включает форму для добавления маршрута и таблицу для отображения списка запланированных поездок. Разработка данного проекта позволила закрепить на практике знания в области: проектирования и архитектуры веб-приложений, создания и тестирования REST API, взаимодействия между клиентом и сервером, работы с базой данных MongoDB. Проект легко масштабируется: в дальнейшем можно реализовать фильтрацию по датам и направлениям, визуализацию маршрутов на карте, авторизацию по JWT, подключение внешних API. Таким образом, приложение представляет собой не только учебную, но и потенциально полезную систему, которую можно адаптировать под реальные задачи по планированию путешествий.

# **Список использованных источников**

1. Франклин С. Основы разработки веб-приложений: от HTML до JavaScript. — М.: ДМК Пресс, 2020. — 256 с.

2. Фримен Э. Программирование на Node.js: создание масштабируемых приложений. — М.: Вильямс, 2018. — 408 с.

3. Райс М. Разработка современных веб-приложений с использованием React.js. — М.: Вильямс, 2019. — 312 с.

4. Leaflet.js — библиотека для работы с картами. URL: https://leafletjs.com

5. Express.js — фреймворк для Node.js. URL: https://expressjs.com

6. MongoDB — документация по работе с базами данных. URL: https://www.mongodb.com/docs/