# Теоретические сведения для разработки

## 1.1 Обзор архитектуры веб-сервисов.

Архитектура веб-сервиса для управления пользователями в веб-приложениях с использованием Express и MongoDB представляет собой сложную систему компонентов, взаимодействующих между собой для обеспечения функциональности, безопасности и производительности приложения. Разработка такой системы требует внимания к каждой составляющей и их взаимодействию, чтобы обеспечить эффективную работу и удовлетворение потребностей пользователей.

На стороне клиента, фронтенд веб-приложения создается с использованием различных технологий, таких как HTML, CSS и JavaScript.

На серверной стороне приложения, Express.js является основным инструментом для создания веб-сервисов на Node.js. Express предоставляет множество функций, таких как обработка HTTP-запросов, маршрутизация и управление сессиями, что делает его идеальным выбором для создания мощных веб-приложений. Маршруты определяются для обработки различных видов запросов, таких как регистрация, вход в систему и управление профилем пользователя. Контроллеры обрабатывают эти запросы, взаимодействуя с базой данных MongoDB через Mongoose или другие библиотеки, и выполняют необходимые действия.

В качестве базы данных используется MongoDB, где модели данных определяют структуру и связи между данными. MongoDB предоставляет гибкую схему, что позволяет разработчикам создавать динамические и масштабируемые приложения, а также горизонтальное масштабирование для обеспечения высокой производительности и доступности данных.

Одним из важных аспектов архитектуры является безопасность. В этой системе, реализация механизмов аутентификации и авторизации пользователей обеспечивает защиту конфиденциальности данных и предотвращение несанкционированного доступа. Использование механизмов аутентификации, таких как JSON Web Tokens (JWT), и управление доступом к ресурсам в приложении помогает обеспечить безопасность данных и пользователей. Дополнительно, безопасное хранение паролей с использованием bcrypt и реализация механизма двухфакторной аутентификации (2FA) значительно повышают уровень безопасности.

Обработка ошибок и исключений является неотъемлемой частью архитектуры веб-сервиса. Это включает в себя обработку и логирование ошибок, а также предоставление информативных сообщений об ошибках для пользователей и администраторов. Такой подход помогает обеспечить надежную и стабильную работу сервиса, уменьшая возможные проблемы и повышая удовлетворенность пользователей.

Наконец, интеграция с внешними API и микросервисами может улучшить функциональность веб-приложения. Например, использование сервисов для отправки уведомлений по электронной почте или SMS, интеграция с платежными системами для обработки транзакций и использование аналитических сервисов для мониторинга и анализа данных о пользователях.

## 1.2 Основные принципы работы с MongoDB.

Работа с MongoDB в веб-приложениях для управления пользователями требует применения различных техник и стратегий для оптимизации процессов хранения и доступа к данным. Рассмотрим подробнее некоторые из этих методов, которые помогут улучшить производительность и надежность вашего приложения.

Кэширование является одной из ключевых техник, используемых для повышения производительности веб-приложений. Кэширование позволяет временно сохранять часто используемые данные в оперативной памяти или другом быстродействующем хранилище. Это значительно сокращает время доступа к данным и уменьшает нагрузку на базу данных. Например, если ваше приложение часто запрашивает информацию о профилях пользователей, вы можете использовать кэширование для хранения этой информации. Инструменты, такие как Redis или Memcached, предоставляют высокопроизводительные решения для кэширования данных. Redis, в частности, поддерживает различные типы данных, включая строки, хеши и списки, что делает его гибким инструментом для кэширования сложных данных. Memcached, с другой стороны, предлагает простой и быстрый кэш для ключ-значение пар, идеально подходящий для хранения результатов часто выполняемых запросов.

Управление версиями и миграции схемы данных является еще одним важным аспектом работы с MongoDB. В процессе развития приложения может возникнуть необходимость внести изменения в структуру данных или типы полей. MongoDB не требует строгой схемы (схема-агностик), что позволяет легко добавлять новые поля или изменять существующие. Однако для обеспечения согласованности данных и безопасности рекомендуется использовать инструменты управления версиями и миграций, такие как Mongeez или Flyway. Эти инструменты помогают автоматизировать процесс изменения структуры базы данных, обеспечивая контроль версий и возможность отката изменений в случае необходимости. Это особенно важно в больших проектах, где изменения в структуре данных могут затронуть множество частей системы.

Мониторинг и аналитика производительности базы данных играют важную роль в поддержании высокой производительности и надежности веб-приложений. Мониторинг позволяет отслеживать состояние базы данных и выявлять возможные проблемы производительности, такие как медленные запросы, блокировки или утечки ресурсов. Для мониторинга MongoDB можно использовать встроенные инструменты, такие как MongoDB Cloud Manager или сторонние решения, такие как Datadog или New Relic. Эти инструменты предоставляют визуализацию метрик, оповещения и детализированные отчеты, которые помогают администраторам баз данных и разработчикам своевременно реагировать на возникающие проблемы. Аналитика производительности помогает понять, какие запросы и операции являются наиболее ресурсоемкими, и оптимизировать их для повышения производительности приложения. Например, анализируя логи запросов, можно выявить "узкие места" и перестроить индексы или оптимизировать сам код запросов.

Резервное копирование данных является критически важным процессом для защиты информации о пользователях от потери в случае сбоев или аварий. Регулярное создание резервных копий данных позволяет быстро восстановить систему и минимизировать потери данных. MongoDB предоставляет инструменты для создания резервных копий данных и восстановления из них, такие как mongodump и mongorestore. Эти инструменты позволяют создавать полные или инкрементальные копии данных, которые могут храниться в различных местах, включая облачные хранилища. Кроме того, можно настроить автоматическое резервное копирование с помощью скриптов или использовать решения, такие как MongoDB Atlas, которые включают автоматическое резервное копирование и восстановление данных в облаке.

Оптимизация запросов и индексов является важной частью обеспечения высокой производительности MongoDB. Индексы позволяют ускорить выполнение запросов, но их создание и использование требуют тщательного планирования. Плохо спроектированные индексы могут не только не улучшить, но и ухудшить производительность базы данных. Поэтому важно анализировать запросы и создавать индексы, которые соответствуют паттернам использования данных в вашем приложении. Инструменты, такие как MongoDB Compass, предоставляют визуальный интерфейс для анализа и оптимизации запросов и индексов.

Шардирование (распределение данных по нескольким серверам) является стратегией, используемой для повышения масштабируемости базы данных. Шардирование позволяет распределить нагрузку и данные между несколькими серверами, что особенно полезно для приложений с большим объемом данных и высоким трафиком. MongoDB поддерживает горизонтальное шардирование, что позволяет эффективно масштабировать базу данных по мере роста приложения.

В заключение, использование этих дополнительных техник и стратегий в сочетании с основными принципами работы с MongoDB обеспечивает эффективное и надежное управление пользователями в веб-приложениях. Это позволяет создавать высокопроизводительные и масштабируемые приложения, способные эффективно обрабатывать как текущую, так и растущую нагрузку данных и запросов. Оптимизация кэширования, управление версиями данных, мониторинг производительности, регулярное резервное копирование, оптимизация запросов и использование шардирования совместно обеспечивают надежную и эффективную работу вашей системы.

## 1.3 Express и его основные компоненты.

Express.js является фреймворком для создания веб-приложений на платформе Node.js. Он предоставляет набор инструментов и функций, которые упрощают процесс создания серверной части веб-приложений. Основные компоненты Express в контексте управления пользователями в веб-приложениях включают:

Маршрутизация: Express позволяет определять маршруты для обработки HTTP-запросов. Это позволяет приложению реагировать на запросы, например, для регистрации новых пользователей, входа в систему, получения информации о пользователе и других действий. Промежуточное программное обеспечение (Middleware): Это функции, которые выполняются перед или после обработки запроса. Middleware может использоваться для выполнения различных задач, таких как аутентификация, авторизация, логирование, обработка ошибок и другие операции, что делает Express гибким и мощным инструментом для обработки запросов.

Шаблонизация: Express позволяет использовать различные шаблонизатоы для генерации HTML-страниц на сервере. Это позволяет создавать динамические веб-страницы, например, для отображения профиля пользователя или списка пользователей.

Управление состоянием: Express не предоставляет встроенного механизма управления состоянием приложения, однако разработчики могут использовать сторонние библиотеки, такие как Express Sessions или Passport.js, для реализации аутентификации и авторизации пользователей, управления сеансами и других функций управления состоянием.

Поддержка статических файлов: Express предоставляет возможность обслуживать статические файлы, такие как изображения, CSS-стили и JavaScript-файлы, что делает его удобным инструментом для разработки веб-приложений.

Express.js обладает простым и интуитивно понятным API, что делает его популярным выбором для создания веб-приложений. Он обеспечивает удобный способ организации серверной части приложений и обработки HTTP-запросов, что делает его идеальным инструментом для разработки веб-приложений для управления пользователями.

**1.4** **Расширенное описание HTTP методов.**

HTTP (HyperText Transfer Protocol) методы являются основополагающими компонентами веб-технологий, обеспечивающими взаимодействие между клиентом и сервером. Они определяют действия, которые могут быть выполнены над ресурсами, предоставляемыми сервером. Рассмотрим основные методы HTTP, их предназначение, использование, а также различия между ними.

GET метод используется для запроса данных с сервера. Он считается безопасным и идемпотентным, что означает, что выполнение одного и того же запроса несколько раз не изменяет состояние ресурса. GET запросы часто кешируются, могут передаваться в закладках браузеров и не изменяют серверные данные. Например, запрос на получение информации о пользователе с идентификатором 123 будет выглядеть как GET /users/123. Этот метод широко используется для получения веб-страниц, изображений, данных в формате JSON или XML и других ресурсов.

POST метод применяется для отправки данных на сервер с целью создания или обновления ресурса. В отличие от GET, POST запросы могут содержать тело запроса, что позволяет передавать большие объемы данных, такие как формы, файлы или JSON объекты. POST не является идемпотентным, то есть повторный вызов запроса может привести к созданию нескольких идентичных ресурсов. Например, создание нового пользователя может выполняться запросом POST /users с телом запроса, содержащим информацию о пользователе, такую как имя, email и пароль.

PUT метод используется для обновления ресурса на сервере. Он также является идемпотентным: повторный вызов одного и того же запроса PUT приводит к одному и тому же результату. PUT запросы содержат полные данные ресурса, которые необходимо обновить, и указывают на конкретный ресурс. Например, обновление информации о пользователе с идентификатором 123 может быть выполнено запросом PUT /users/123 с телом запроса, содержащим новые данные пользователя, такие как обновленный адрес или номер телефона.

DELETE метод применяется для удаления ресурса с сервера. Как и PUT, он является идемпотентным, что означает, что повторное выполнение DELETE запроса не изменяет результат: ресурс будет удален, если он существует, и последующие запросы не изменят это состояние. Удаление ресурса с идентификатором 123 может быть выполнено запросом DELETE /users/123. После успешного выполнения DELETE запроса ресурс больше не будет доступен по указанному URI, и сервер может вернуть статус 404 Not Found при попытке повторного доступа к удаленному ресурсу.

PATCH метод используется для частичного обновления ресурса. В отличие от PUT, который требует полную информацию о ресурсе, PATCH позволяет отправлять только измененные данные. Это делает PATCH более эффективным для обновления больших ресурсов, где изменяются лишь небольшие части данных. Например, частичное обновление информации о пользователе с идентификатором 123 может быть выполнено запросом PATCH /users/123 с телом запроса, содержащим только измененные поля, такие как обновленный email или добавленный номер телефона.

HEAD метод запрашивает заголовки ресурса, аналогично GET запросу, но без тела ответа. Он используется для проверки существования ресурса, получения метаданных или проверки актуальности данных перед загрузкой. Например, запрос HEAD /users/123 вернет заголовки ответа, содержащие информацию о дате последнего изменения ресурса, типе содержимого и размере, но не само содержимое. Это позволяет экономить трафик и ресурсы, если нужно только проверить статус ресурса без его загрузки.

OPTIONS метод позволяет клиенту узнать, какие методы поддерживаются сервером для конкретного ресурса. Он часто используется для настройки CORS (Cross-Origin Resource Sharing), определяя разрешенные методы и заголовки для междоменных запросов. Запрос OPTIONS /users/123 вернет список методов, доступных для этого ресурса, таких как GET, POST, PUT и DELETE, а также информацию о поддерживаемых заголовках и параметрах CORS. Этот метод полезен для понимания возможностей взаимодействия с сервером перед выполнением реальных запросов.

CONNECT метод используется для установления туннеля к серверу через прокси. Он часто применяется для создания защищенных соединений с использованием протокола HTTPS. Клиент отправляет запрос CONNECT, после чего прокси-сервер устанавливает туннель и передает данные между клиентом и сервером. Этот метод используется для обоснования и обеспечения безопасного соединения, особенно при работе с конфиденциальной информацией, такой как банковские данные или личные данные пользователей.

TRACE метод выполняет обратный запрос, позволяя клиенту увидеть, что сервер получает в запросе. Он используется для диагностики и отладки, возвращая клиенту запрос, каким его видит сервер. Однако использование TRACE может представлять риск для безопасности, так как он может привести к утечке данных, особенно если в запросе содержатся конфиденциальные данные. В связи с этим многие современные серверы отключают поддержку метода TRACE, чтобы предотвратить возможные атаки и защитить данные пользователей.

Эти методы формируют основу для взаимодействия между клиентом и сервером в HTTP, обеспечивая разнообразные способы манипуляции с ресурсами и поддержание эффективного и безопасного обмена данными. Понимание каждого из них важно для разработки веб-приложений, обеспечивающих надежное, безопасное и масштабируемое взаимодействие между различными компонентами системы.

**1.5 Ресурсы и URL.**

В веб-технологиях ресурсы и URL (Uniform Resource Locator) играют ключевую роль в обеспечении доступа к данным и взаимодействии между клиентом и сервером. Ресурсы представляют собой различные объекты, доступные через интернет, такие как веб-страницы, изображения, файлы и другие виды данных. URL, в свою очередь, является адресом, который позволяет уникально идентифицировать и получить доступ к этим ресурсам.

Ресурсы в интернете могут быть статическими и динамическими. Статические ресурсы включают в себя неизменяемые файлы, такие как HTML-страницы, изображения, стили CSS и скрипты JavaScript. Эти ресурсы остаются неизменными между запросами и часто кешируются для повышения производительности. Например, статическая веб-страница, которая представляет собой файл HTML, всегда будет отображаться одинаково для всех пользователей до тех пор, пока она не будет изменена на сервере.

Динамические ресурсы генерируются сервером на основе запросов пользователя и могут включать данные из баз данных, пользовательские данные и результаты выполнения скриптов. Примеры динамических ресурсов включают страницы с результатами поиска, профили пользователей и данные из различных онлайн-сервисов. Динамическая природа таких ресурсов позволяет им изменяться в зависимости от входных данных или состояния приложения. Например, при входе пользователя на веб-сайт, страница профиля этого пользователя может быть сгенерирована динамически с учетом его персональных данных.

URL состоит из нескольких компонентов, которые вместе определяют местоположение и способ доступа к ресурсу. Основные компоненты URL включают схему, доменное имя, порт, путь, параметры и фрагмент.

Схема (например, http или https) указывает на используемый протокол. HTTP используется для обычных запросов, тогда как HTTPS обеспечивает защищенное соединение.

Путь (например, /users/123) указывает на конкретное расположение ресурса на сервере. Путь может быть иерархическим, отражая структуру файлов и каталогов на сервере.

Параметры (например, ?name=John) передают дополнительные данные для запроса. Параметры используются для фильтрации, сортировки и передачи других данных, необходимых для обработки запроса.

Фрагмент (например, #section2) используется для навигации внутри ресурса. Фрагмент указывает на конкретную часть документа, позволяя браузеру прокручивать страницу до нужного места.

Использование URL играет важную роль в SEO (Search Engine Optimization), так как хорошо структурированные и описательные URL могут улучшить индексацию и рейтинг веб-сайта в поисковых системах. Краткие, понятные и логически организованные URL облегчают пользователям и поисковым системам понимание структуры сайта и нахождение нужной информации. Например, URL /products/books/harry-potter вместо /product?id=123 предоставляет более понятное и SEO-дружественное представление о содержимом страницы.

Кроме того, URL могут содержать параметры для передачи данных между клиентом и сервером. Эти параметры могут быть частью запроса (query string) и использоваться для фильтрации, сортировки и поиска данных. Например, URL /products?category=books&sort=price\_asc может использоваться для получения списка книг, отсортированных по цене. Важно правильно кодировать параметры URL, чтобы избежать проблем с интерпретацией данных.

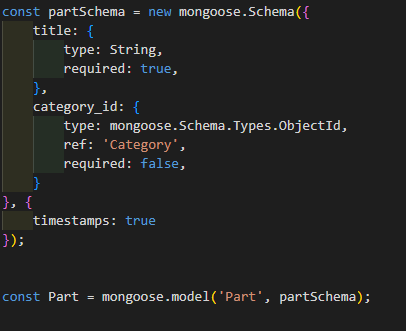
URL также поддерживают якоря или фрагменты, которые позволяют ссылаться на определенные части ресурса. Например, URL /about#team будет открывать страницу «about» и автоматически прокручивать ее до секции с якорем «team». Это полезно для навигации по длинным страницам и для создания ссылок на конкретные разделы контента.

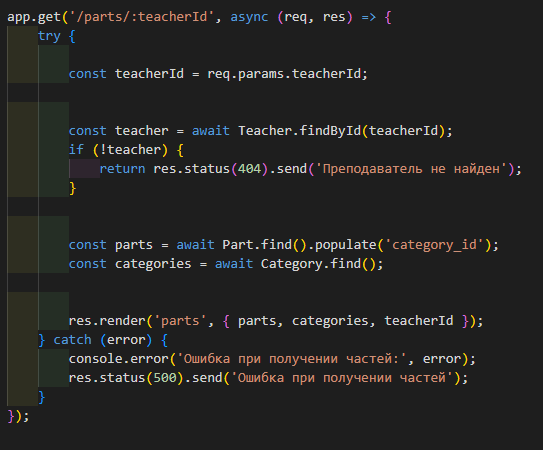
Веб-серверы и фреймворки часто предоставляют возможности для обработки и маршрутизации URL. Маршрутизация позволяет направлять запросы к соответствующим контроллерам и обработчикам на сервере, что обеспечивает гибкость и масштабируемость веб-приложений. Например, запрос к URL /users/123 может быть направлен к функции, которая извлекает информацию о пользователе с идентификатором 123 из базы данных и возвращает ее в ответе.

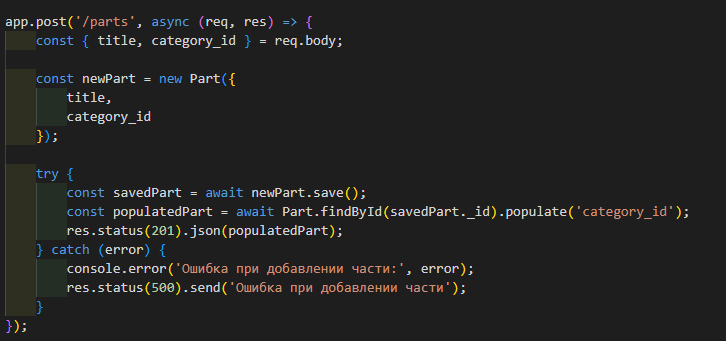
Современные веб-приложения часто используют дружественные URL, которые легко читаются и запоминаются пользователями. Вместо сложных и непонятных URL с множеством параметров, дружественные URL содержат ключевые слова и структуру, отражающую содержание и иерархию сайта. Например, URL /blog/2024/05/21/advantages-of-http-methods является более понятным и информативным, чем /article?id=12345.

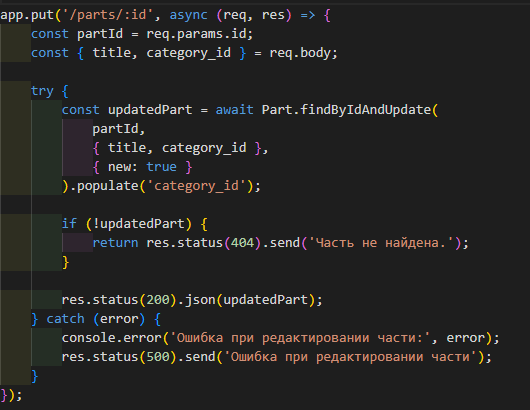
В заключение, ресурсы и URL являются основополагающими элементами веб-технологий, обеспечивающими доступ к данным и взаимодействие между клиентом и сервером. Понимание их структуры и принципов работы важно для разработки эффективных, масштабируемых и удобных веб-приложений.

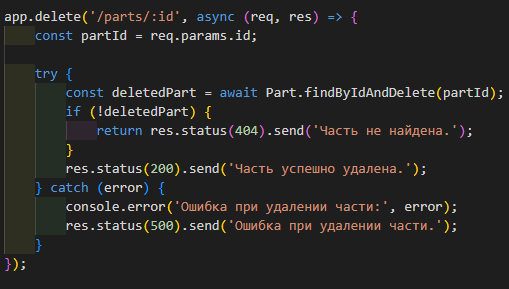
Пояснение функций:

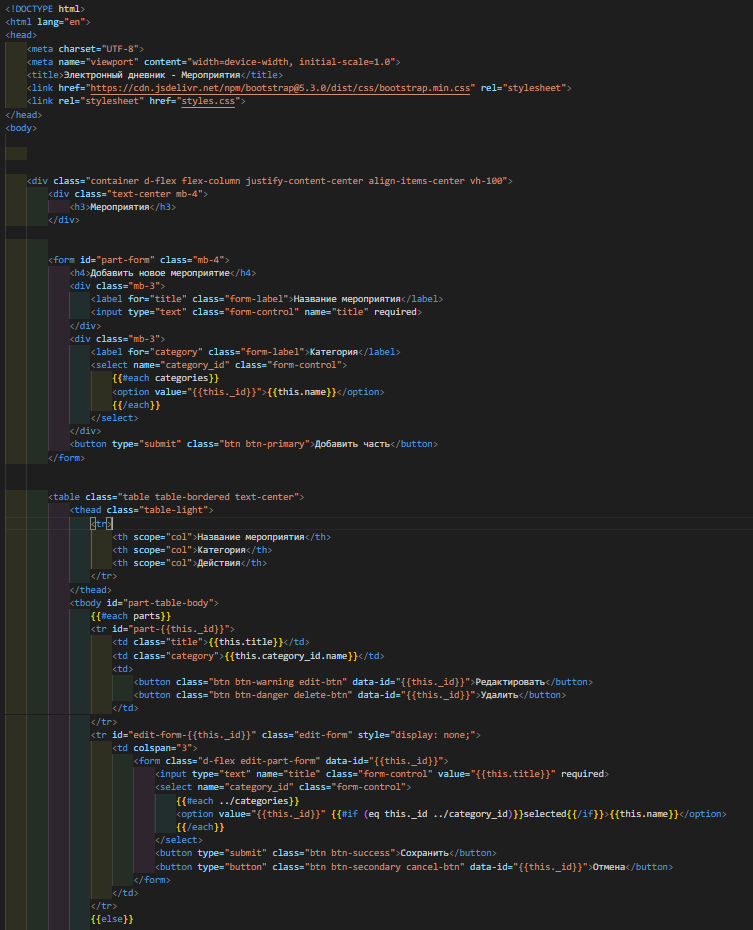
  
Рисунок 1 – Модель Part

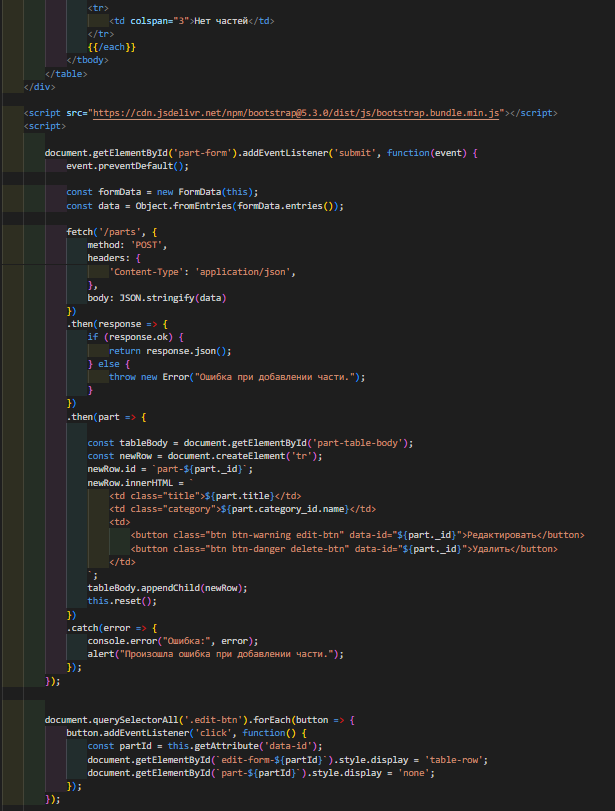
  
Рисунок 2 – функция GET

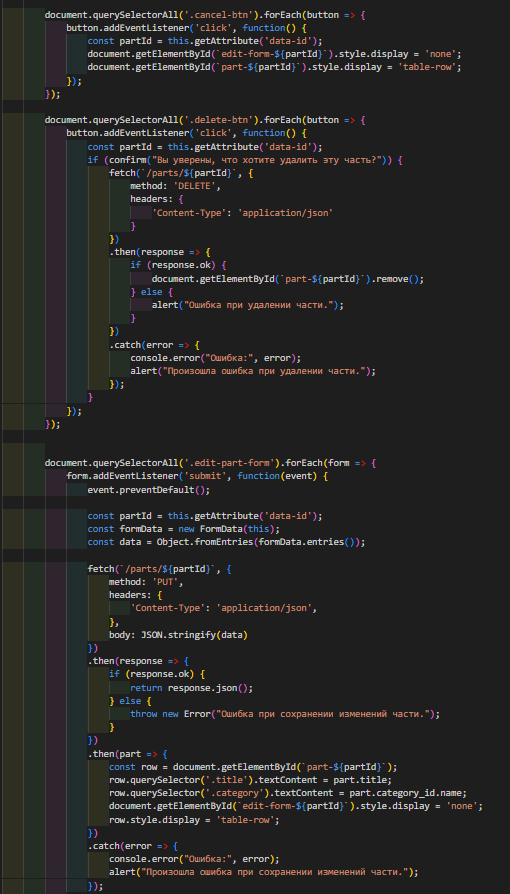
  
Рисунок 3 – функция Post

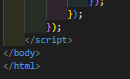
  
Рисунок 4 – функция PUT

  
Рисунок 5 – функция Delete

  
Рисунок 6

  
Рисунок 7 -

  
Рисунок 8

  
Рисунок 9

HTML: Структура страницы

HTML-код создает базовый интерфейс пользователя для добавления, редактирования и удаления мероприятий.

Шапка документа (<head>):

Подключает Bootstrap CSS для использования стандартных стилей и компонентов.

Подключает внешний файл styles.css, который может быть использован для пользовательских стилей.

Контейнер и центровка (<div class="container">):

Используется класс d-flex flex-column justify-content-center align-items-center vh-100, чтобы весь контент был выровнен по центру экрана.

Класс vh-100 устанавливает высоту контейнера на весь экран, что удобно для создания адаптивных интерфейсов.

Форма добавления нового мероприятия (<form id="part-form">):

Поле "Название мероприятия":

<input type="text" class="form-control" name="title" required> — текстовое поле для ввода названия мероприятия, обязательное для заполнения.

Категория мероприятия:

<select name="category\_id" class="form-control"> — выпадающий список, который заполняется с сервера. Это необходимо, чтобы выбрать категорию мероприятия.

{{#each categories}} позволяет пройти по массиву categories и сгенерировать <option> для каждой категории.

Таблица мероприятий:

Таблица показывает список добавленных мероприятий. Если мероприятий нет, отображается строка Нет частей.

Каждая строка содержит:

Название мероприятия.

Название категории, связанной с этим мероприятием (полученное через populate).

Кнопки "Редактировать" и "Удалить".

Форма редактирования мероприятия:

Вторая строка под каждым мероприятием содержит скрытую форму для редактирования, которая отображается при нажатии на кнопку "Редактировать".

Поля аналогичны форме добавления: поле для изменения названия и выбор категории.

JavaScript: Логика взаимодействия на клиенте

JavaScript код управляет интерфейсом и отправляет запросы к серверу с помощью fetch. Каждый запрос вызывает определенное действие (добавление, редактирование или удаление), а изменения отображаются в реальном времени на странице.

Добавление мероприятия:

Событие отправки формы (document.getElementById('part-form').addEventListener('submit', ...)):

event.preventDefault(); предотвращает перезагрузку страницы при отправке формы.

Содержимое формы собирается с помощью FormData и преобразуется в объект JavaScript для удобного использования.

POST-запрос отправляется на /parts, передавая данные мероприятия.

Если запрос успешен, возвращенное мероприятие добавляется в таблицу с использованием tableBody.appendChild(newRow).

Редактирование мероприятия:

Обработчик кнопки "Редактировать":

При нажатии на "Редактировать" для строки мероприятия скрывается стандартный вид строки и появляется форма редактирования (по идентификатору элемента edit-form-{{this.\_id}}).

Отмена изменений:

Кнопка "Отмена" возвращает вид таблицы в обычное состояние, скрывая форму редактирования.

Сохранение изменений:

Форма редактирования отправляет PUT-запрос на сервер по маршруту /parts/:id с измененными данными.

Обновленная строка возвращается от сервера и используется для обновления данных в таблице.

Удаление мероприятия:

Кнопка "Удалить":

При нажатии на кнопку "Удалить" у пользователя запрашивается подтверждение.

После подтверждения отправляется DELETE-запрос на /parts/:id для удаления элемента.

Если удаление выполнено успешно, строка мероприятия удаляется из таблицы на клиентской стороне.

Серверный код (Node.js и Express)

Серверная логика управляет CRUD-операциями с мероприятиями, используя маршруты в Express и Mongoose для взаимодействия с базой данных MongoDB.

Модель Part:

Поле title — строка, обязательное к заполнению, содержит название мероприятия.

Поле category\_id — идентификатор категории, позволяет присвоить мероприятию категорию через связь с другой коллекцией Category.

timestamps: true добавляет автоматически createdAt и updatedAt для записи времени создания и последнего обновления.

Маршруты

GET /parts/:teacherId:

Этот маршрут:

Ищет преподавателя по ID (teacherId).

Получает все мероприятия и категории из базы данных.

Рендерит страницу parts, передавая туда данные о мероприятиях, категориях и ID преподавателя.

POST /parts:

Создание нового мероприятия:

Принимает данные title и category\_id из req.body.

Создает и сохраняет новое мероприятие в базе данных.

После сохранения данные снова загружаются из базы с populate('category\_id'), чтобы заполнить поле category\_id именем категории.

Возвращает JSON-объект с данными нового мероприятия.

PUT /parts/:id:

Обновление мероприятия:

Находит мероприятие по идентификатору id, обновляет title и category\_id.

Если обновление выполнено успешно, возвращает JSON с обновленными данными.

Если мероприятие не найдено, возвращает ошибку 404.

DELETE /parts/:id:

Удаление мероприятия:

Находит мероприятие по id и удаляет его из базы данных.

Если мероприятие не найдено, возвращает ошибку 404. В случае успеха отправляет ответ об успешном удалении.

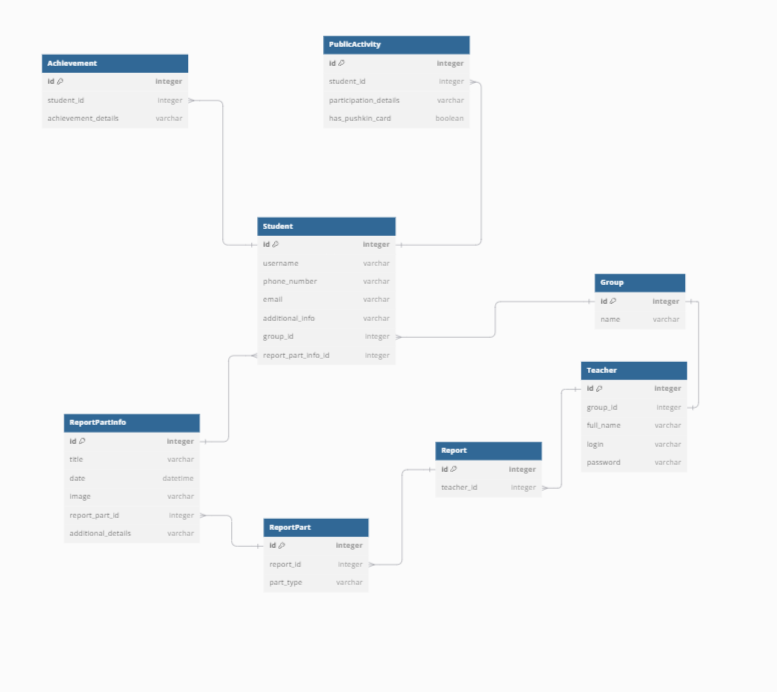
Поток данных

Пользователь заходит на страницу, и сервер отправляет список мероприятий и категорий.

При добавлении нового мероприятия клиент отправляет запрос на сервер, и оно добавляется в базу данных.

При редактировании клиент отправляет обновленные данные серверу, и они сохраняются в базе.

При удалении сервер удаляет мероприятие и уведомляет об успешном удалении.

  
Рисунок 10 – Схема БД

Описание БД :  
Group (Группа):

Хранит информацию о группах студентов.

Поля:

id — уникальный идентификатор группы (первичный ключ).

name — название группы.

Связи:

Один ко многим со студентами (Student) и учителями (Teacher).

Student (Студент):

Содержит информацию о студентах и их привязке к группе, а также данные для отчетов.

Поля:

id — уникальный идентификатор студента (первичный ключ).

username — имя пользователя.

phone\_number — номер телефона.

email — адрес электронной почты.

additional\_info — дополнительная информация о студенте.

group\_id — ссылка на группу, к которой принадлежит студент.

report\_part\_info\_id — ссылка на запись в ReportPartInfo, связанную с этим студентом.

Связи:

Связь с группой (Group) через group\_id.

Один ко многим с таблицами Achievement(Достижения) и PublicActivity(Публичная активность).

Teacher (Учитель):

Содержит данные об учителях и их привязке к группе.

Поля:

id — уникальный идентификатор учителя (первичный ключ).

group\_id — ссылка на группу, с которой связан учитель.

full\_name — полное имя учителя.

login — логин для доступа к системе.

password — пароль для входа.

Связи:

Связь с таблицей Group через group\_id.

Один ко многим с таблицей Report(Отчёт).

PublicActivity (Публичная Активность):

Хранит информацию об участии студента в публичных мероприятиях.

Поля:

id — уникальный идентификатор активности (первичный ключ).

student\_id — ссылка на студента, участвующего в активности.

participation\_details — детали участия.

has\_pushkin\_card — булевое значение, указывающее, есть ли у студента карта Пушкина.

Achievement (Достижение):

Содержит достижения, связанные со студентами.

Поля:

id — уникальный идентификатор достижения (первичный ключ).

student\_id — ссылка на студента, которому присвоено достижение.

achievement\_details — описание достижения.

Связи:

Один ко многим с таблицей ReportPart.

ReportPart (Часть Отчета):

Разделяет отчет на отдельные части с разными типами содержания.

Поля:

id — уникальный идентификатор части отчета (первичный ключ).

report\_id — ссылка на отчет, к которому относится часть.

part\_type — тип части отчета (например, type1, type2, type3).

ReportPartInfo (Информация о Части Отчета):

Содержит детальную информацию о конкретной части отчета.

Поля:

id — уникальный идентификатор записи (первичный ключ).

title — заголовок части отчета.

date — дата, к которой относится информация.

image — путь к изображению (например, для загрузки файлов).

report\_part\_id — ссылка на ReportPart, к которой относится данная информация.

additional\_details — дополнительные детали.

Связи:

Один ко многим с таблицей ReportPart.

Связи между таблицами

Group и Student: Один ко многим — каждая группа может содержать множество студентов.

Group и Teacher: Один ко многим — каждая группа может быть связана с несколькими учителями.

Student и PublicActivity / Achievement: Один ко многим — каждый студент может участвовать в множестве активностей и иметь несколько достижений.

Teacher и Report: Один ко многим — каждый учитель может создавать несколько отчетов.

ReportPart и ReportPartInfo: Один ко многим — каждая часть отчета может иметь связанную информацию о деталях