

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ МОСКОВСКОЙ ОБЛАСТИ**

**Государственное автономное профессиональное образовательное учреждение Московской области**

**«Подмосковный колледж «Энергия»**

ДОПУЩЕН К ЗАЩИТЕ:

|  |  |  |
| --- | --- | --- |
| Заведующий СП ЦОПП «Энергия»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Н.Г. Близнякова  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2022г. | Заместитель директора по УР  \_\_\_\_\_\_\_\_\_\_\_\_А.В. Куликова  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2022г | Заместитель директора по УПР  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_К.С. Подоляк  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022г. |

**ДИПЛОМНАЯ РАБОТА**

**Тема работы:**

**«Создание макетов для многостраничного веб-сайта»**

Специальность 09.02.07 «Информационные системы и программирование»

Руководитель проекта /Архипкин Д.А./

Консультант по нормоконтролю /Васильченко Л.Г./

Рецензент /Харченко Г.И./

Обучающийся /Макаров К.К./

Работа защищена «\_\_\_» июня 2024 г. Оценка / /

Реутов

2024 г.

Содержание

[Содержание 1](#_Toc167059328)

[Введение 2](#_Toc167059329)

[Глава I. Исследование предметной области и составление требований к проекту. 4](#_Toc167059330)

[1.1 Анализ предметной области. Анализ основных конструкций и функций системы 4](#_Toc167059331)

[1.2 Планирование разработки информационной системы. 10](#_Toc167059332)

[1.2.1 База данных 10](#_Toc167059333)

[1.2.2 Карта сайта. 12](#_Toc167059334)

[1.2.3 Технологии 12](#_Toc167059335)

[1.2.4 Этапы разработки 18](#_Toc167059336)

[1.2.5 Тестирование 19](#_Toc167059337)

[1.2.6 Выбор конфигурации хостинга 19](#_Toc167059338)

[Глава II. Практическая часть. 22](#_Toc167059339)

[2.1. Разработка серверной части web-приложения. 22](#_Toc167059340)

[2.1.1 Разработка контроллеров 22](#_Toc167059341)

[2.1.2 Разработка модулей. 33](#_Toc167059342)

[2.1.3 Разработка роутеров. 43](#_Toc167059343)

[Заключение 49](#_Toc167059344)

[2.2. Разработка клиентской части web-приложения. 50](#_Toc167059345)

[2.2.1 Разработка верски. 50](#_Toc167059346)

Введение

**Актуальность темы**. В эпоху цифровизации все большее значение приобретает автоматизация процессов управления и планирования. Информационные системы становятся неотъемлемой частью организационной структуры любого предприятия, учебного заведения или общественной организации. Они позволяют сократить время на подготовку и проведение мероприятий, минимизировать риски ошибок и повысить общую эффективность работы.

**Цели и задачи исследования**. Основной целью данной дипломной работы является разработка информационной системы, которая будет способствовать оптимизации процессов планирования и проведения мероприятий. Для достижения этой цели были поставлены следующие задачи:

1. Изучение и анализ существующих информационных систем управления мероприятиями и расписанием.
2. Определение требований и функционала необходимого для создания эффективной системы.
3. Проектирование архитектуры системы и её компонентов.
4. Разработка прототипа системы с использованием современных программных решений.
5. Тестирование системы и оценка её функциональности.
6. Разработка рекомендаций по внедрению и использованию системы в реальных условиях.

**Методология исследования**. В работе будет использован комплексный подход, сочетающий теоретические исследования и практическое применение. Теоретическая часть включает в себя анализ литературы, изучение передовых практик и существующих технологий. Практическая часть предполагает разработку прототипа на основе современных программных платформ и языков программирования, а также его последующее тестирование и анализ полученных результатов.

**Научная новизна**. Работа будет содержать разработку уникальных алгоритмов и методик, которые могут быть применены для улучшения функциональности существующих информационных систем или создания новых. Особое внимание будет уделено удобству использования системы, её масштабируемости и безопасности.

**Практическая значимость**. Разработанная система будет иметь широкий спектр применения в различных организациях и учреждениях. Она позволит упростить процесс планирования мероприятий, сделать его более прозрачным и доступным для всех участников. Это, в свою очередь, приведет к повышению эффективности работы и удовлетворенности как организаторов, так и участников мероприятий.

Глава I. Исследование предметной области и составление требований к проекту.

* 1. Анализ предметной области. Анализ основных конструкций и функций системы

Информационные системы для управления расписанием - это специальные программные продукты, предназначенные для создания, редактирования и контроля за расписанием различных мероприятий, задач и событий. Они могут использоваться как для персонального планирования, так и для организации работы группы сотрудников или даже целой компании.

Такие системы обычно имеют удобный пользовательский интерфейс, который позволяет быстро и эффективно создавать и изменять расписание, устанавливать приоритеты для задач, устанавливать напоминания о предстоящих событиях и делах, а также делиться расписанием с другими пользователями.

Информационные системы для управления расписанием могут включать в себя функционал для синхронизации с другими календарными приложениями и устройствами, возможность создания групповых задач и общего календаря для коллективной работы, а также аналитику и статистику по выполненным задачам.

Такие системы очень удобны для организации рабочего времени, планирования встреч, деловых поездок, управления проектами и т.д. Они помогают повысить производительность и эффективность работы, уменьшить вероятность просрочек и забытых задач, а также улучшить коммуникацию и сотрудничество в коллективе.

Информационные системы для планирования мероприятий являются важным инструментом для организации и координации различных событий, таких как конференции, семинары, свадьбы и другие мероприятия. Эти системы помогают управлять всеми аспектами мероприятия, начиная от бюджета и графика до гостевого списка и логистики.

Информационные системы для планирования мероприятий помогают значительно упростить и ускорить процесс организации мероприятий, повысить их эффективность и качество. Поэтому они являются неотъемлемой частью профессиональной деятельности организаторов мероприятий.

Системы управления расписанием и проведением мероприятий имеют ряд преимуществ и недостатков.

Преимущества:

1. Эффективное планирование: системы управления расписанием позволяют точно определить даты, время и место проведения мероприятий, а также распределить ресурсы и обязанности между участниками.
2. Удобство для пользователей: платформы для организации мероприятий облегчают процесс регистрации и оплаты участия, обеспечивают доступ к информации о мероприятии и обновлениям.
3. Автоматизация процессов: с помощью систем управления можно автоматизировать ряд операций, такие как отправка приглашений, напоминания участникам, подача заявок на участие и др.
4. Оптимизация времени и ресурсов: системы управления расписанием позволяют эффективно распределить задачи и ресурсы, минимизируя время и затраты на организацию мероприятий.

Недостатки:

* 1. Необходимость обучения: для работы с системами управления мероприятиями требуется время на изучение функционала, что может быть сложно для непрофессионалов.
  2. Технические проблемы: некорректная работа платформы, сбои в системе или проблемы с доступом могут привести к задержкам и проблемам при организации мероприятий.
  3. Ограниченные возможности: не все системы управления мероприятиями могут предложить все необходимые функции и инструменты для организации сложных событий.
  4. Зависимость от интернета: большинство систем управления требуют постоянного подключения к интернету, что может создать проблемы в случае отсутствия доступа к сети.

Таким образом, системы управления расписанием и проведением мероприятий имеют свои преимущества и недостатки, и выбор конкретной платформы зависит от потребностей и целей организаторов мероприятий.

Понимание потребностей пользователей является ключевым аспектом при разработке любой информационной системы. Это особенно важно для систем, которые занимаются планированием расписания и мероприятий, поскольку они должны быть максимально удобными и функциональными для конечных пользователей. Исследование потребностей пользователей помогает определить, какие функции и характеристики системы будут наиболее ценными и какие проблемы могут возникнуть при её использовании.

При анализе потребностей пользователя с помощью анализа существубщих систем, были выявленны следующие ключевые потребности пользователей:

Интуитивно понятный интерфейс - пользователи предпочитают простой и понятный интерфейс, который позволяет быстро находить необходимую информацию и выполнять задачи без дополнительного обучения.

Гибкость планирования - система должна предоставлять возможность легко изменять расписание и адаптировать его под неожиданные изменения.

Мобильный доступ - доступ к системе с мобильных устройств является необходимым, так как многие пользователи планируют и координируют мероприятия на ходу.

Функции безопасности - защита личных данных и информации о мероприятиях имеет первостепенное значение для пользователей.

Определение требований и функционала необходимо для создания эффективной системы включает в себя описание основных задач и функций, которые должны быть реализованы в системе, чтобы она соответствовала поставленным целям и задачам.

Для создания такого определения проводится анализ бизнес-процессов и потребностей пользователей, выявляют ключевые требования к системе, определяют функциональные возможности, которые должны быть реализованы, а также нефункциональные требования, такие как производительность, надежность, безопасность и удобство использования.

На основе этого анализа разрабатывается спецификация требований и функционала, которая является основой для разработки и тестирования системы. В ней должны быть четко описаны все функции, модули и возможности системы, а также критерии и метрики, по которым будет оцениваться ее эффективность и соответствие требованиям заказчика.

Таким образом, определение требований и функционала является важным этапом в разработке любой системы, который позволяет направить усилия команды разработчиков в нужное русло и обеспечить успешную реализацию проекта.

Требования к системе:

* Надежность.
* Безопастность.
* Быстродействие.
* Удобность использования.
* Гибкость использования.
* Маштабируемость.

Функциональность системы:

* Авторизация пользователя
* Регистрация пользователя
* Удаление пользователя
* Изменения данных пользователя
* Подтвержения электронной почты пользователя
* Создание мероприятия
* Редактирование мероприятия
* Удаление мероприятия
* Создания заметки
* Обновления контента заметки
* Обновления данных заметки
* Удаление заметки
* Создание задачи в расписание
* Обновление данных задачи в расписании
* Удаление задачи в расписании

Надежность системы будет достигнута использованием горизонтальной маштабируемости, что позволит распределить нагрузку на систему междунесколькими серверами.

Безопасность системы будет достигнута путем хеширования авторизационных данных в базе данных и использованием временых токенов аунтефикации защещеными цифровой подписью.

Быстродействие системы будет достигнута путем распределения нагрузки, оптимизации запросов и кешированием запросов.

Удобность использования системы будет достигнута путем разработки адоптивного интуитивно понятного дизайна интерфейса.

Гибкость использования системы будет достигнута путем адоптивности дизайна, и легкодоступности внедрения обновлений системы.

Маштабируемость системы будет достигнкта путем использования контейнеризации и использованием гибких WEB-Серверов.

* 1. Планирование разработки информационной системы.

1.2.1 База данных

MongoDB — это документоориентированная система управления базами данных, которая не требует описания схемы таблиц. Это один из классических примеров NoSQL-систем, использующих JSON-подобные документы и схему базы данных.

MongoDB применяется в веб-разработке, в частности, в рамках JavaScript-ориентированного стека MEAN.Возможности MongoDB включают поддержку ad-hoc-запросов, поиск по регулярным выражениям, индексацию, работу с наборами реплик, горизонтальное масштабирование с помощью сегментирования объектов и использование в качестве файлового хранилища с балансировкой нагрузки и репликацией данных. MongoDB доступна по открытой лицензии AGPL, но также существует коммерческая версия с дополнительными функциями и поддержкой.

Структура базы данных — это ключевой элемент в проектировании системы, который определяет, как данные будут храниться, связаны и доступны для использования. В контексте системы планирования расписания и мероприятий, структура базы данных может включать следующие компоненты:

Коллекции

* Users:
* id (Уникальный идентификатор пользователя)
* login (Логин пользователя)
* name (Имя пользователя)
* email (Электронная почта пользователя)
* passwordHash (Захешированый пароль пользователя)
* avatar (Картинка аватара пользователя)
* verified (Статус верефикации пользователя)
* notes (Массив с идентификаторами заметок пользователя)
* events (Массив с идентификаторами мероприятий пользователя)
* schedules (Дерево расписания)

Дерево расписания это древовидная структура данных, каторая имеет структуру:



Notes:

* id (Уникальный идентификатор заметки)
* ownerId (Уникальный идентификатор владельца заметки)
* title (Название заметки)
* content (Содержание заметки)

Events:

* id (Уникальный идентификатор мероприятия)
* ownerId (Уникальный идентификатор владельца заметки)
* title (Название заметки)
* description (Описание мероприятия)
* date (Дата проведения мероприятия)
* time (Время проведения мероприятия)
* geoposition (Место проведения мероприятия)
* avatar (Картинка аватара мероприятия)

Verifications:

* id (Уникальный идентификатор сессии верефикации)
* userId (Уникальный идентификатор пользователя)
* createDataTime (Время создания сессии верефикации)
* code (Код подтверждения)

1.2.2 Карта сайта.

Страницы:

* Информационная/рекламная страница
* Страница расписания пользователя
* Страница мероприятий пользователя
* Страница заметок пользователя
* Страница заметки пользователя

Формы:

* Форма регистрации
* Форма авторизации
* Форма подтверждения почты
* Форма создания/изменения задачи расписания пользователя.
* Форма создания/изменения мероприятия пользователя.

1.2.3 Технологии

HTML (HyperText Markup Language) - это стандартный язык разметки для создания веб-страниц. Он используется для структурирования содержимого веб-страниц с помощью различных элементов и тегов.

Основные элементы HTML:

* Теги: HTML-документы состоят из набора тегов, которые определяют структуру и содержимое страницы. Теги обычно имеют открывающий и закрывающий элемент, например <p> для абзаца текста.
* Элементы: Элементы представляют различные части содержимого веб-страницы, такие как заголовки, абзацы, списки, изображения и другие.
* Атрибуты: Атрибуты используются для предоставления дополнительной информации о элементе. Например, атрибут src в элементе <img> указывает источник изображения.

Ключевые концепции HTML:

* Семантика: HTML предоставляет различные теги, которые описывают семантику контента, облегчая понимание структуры страницы поисковыми системами и разработчиками.
* Вложенность: Элементы HTML могут быть вложены друг в друга, создавая иерархию содержимого.
* Разное содержимое: HTML поддерживает различные типы содержимого, такие как текст, изображения, видео, аудио, формы и другие.

HTML является основой веб-разработки и позволяет создавать структурированные и доступные веб-страницы. Понимание основных элементов и принципов HTML поможет создавать качественные и удобные интерфейсы для пользователей.

CSS (Cascading Style Sheets) - это язык таблиц стилей, который используется для оформления и стилизации веб-страниц. Он позволяет разработчикам определять внешний вид элементов HTML, задавая им различные свойства и стили.

Основные принципы CSS:

* Селекторы: CSS использует селекторы для выбора элементов на странице, к которым будут применены стили. Селекторы могут быть элементарными (например, тег <p> ) или более сложными (например, классы или идентификаторы).
* Свойства: Свойства CSS определяют конкретные стили, такие как цвет текста, размер шрифта, отступы, границы и многое другое.
* Значения: Каждое свойство имеет значение, которое определяет конкретное действие или параметр стиля. Например, значение red для свойства color задает красный цвет текста.

Ключевые концепции CSS:

* Каскадность: CSS правила могут быть унаследованы от родительских элементов, но при этом могут быть переопределены более специфичными правилами.
* Локальность: CSS правила могут быть определены внутри тега <style> внутри HTML-документа или внешнем файле CSS, который подключается к странице.
* Адаптивность: С помощью CSS можно создавать адаптивные дизайны, которые корректно отображаются на различных устройствах и экранах.

CSS играет важную роль в создании привлекательного и функционального дизайна веб-страниц. Понимание принципов и возможностей CSS поможет разработчикам создавать стильные и современные интерфейсы для пользователей.

JavaScript — это мультипарадигменный язык программирования, который используется для создания веб-страниц, мобильных приложений и программного обеспечения для настольных компьютеров.

Он основан на скрипте, который описывает действия для обработки и выполнения. JavaScript является кроссплатформенным, так как работает во всех современных браузерах.

JavaScript имеет несколько стилей программирования:

* функциональный
* объектно-ориентированный
* императивный.

Он широко применяется в веб-разработке, фронтенд-разработке, создании мобильного ПО, бэкенд-программировании и написании программного обеспечения для десктопов.

Преимущества JavaScript включают его незаменимость в веб-разработке, интеграцию с версткой страниц и серверной частью, рациональность применения и простоту синтаксиса, скорость работы и производительность, комфортность использования пользовательских интерфейсов и наличие мощной экосистемы с множеством полезных фреймворков.

Однако у JavaScript есть и некоторые недостатки, такие как отсутствие возможности чтения и загрузки документов из соображений безопасности, отсутствие удалённого доступа и нестрогая типизация, которая может привести к ошибкам в коде.

TypeScript — это язык программирования, представленный Microsoft в 2012 году. Он расширяет возможности JavaScript и предназначен для разработки веб-приложений.

TypeScript обладает следующими особенностями:

* Статическая типизация: возможность явного назначения типов переменным и функциям, что облегчает поиск ошибок и улучшает читаемость кода.
* Поддержка классов: TypeScript позволяет использовать полноценные классы, как в традиционных объектно-ориентированных языках.
* Поддержка модулей: TypeScript включает поддержку модулей, что облегчает повторное использование кода, поиск ошибок и ускоряет выполнение программ.

TypeScript полностью обратно совместим с JavaScript и компилируется в него. После компиляции программу на TypeScript можно выполнять в любом современном браузере или использовать совместно с серверной платформой Node.js.

Node.js — это среда выполнения, которая позволяет запускать JavaScript вне веб-браузера, на стороне сервера. Она обеспечивает возможность использовать JavaScript для создания серверных приложений и для взаимодействия с базами данных, файлами и другими сервисами.

Node.js позволяет JavaScript взаимодействовать с устройствами ввода-вывода через API, подключать внешние библиотеки и выполнять неблокирующий ввод/вывод.

Node.js применяется преимущественно на сервере для создания веб-серверов, но также может использоваться для разработки десктопных и оконных приложений и программирования микроконтроллеров. В основе Node.js лежит событийно-ориентированное и асинхронное программирование.

Fastify.js — это высокопроизводительный и быстрый веб-фреймворк для Node.js, предназначенный для создания веб-приложений и API. Он фокусируется на скорости, поддержке схемы для проверки запросов и ответов, расширяемости, асинхронности и удобстве для разработчиков.

Fastify предлагает следующие преимущества:

* Высокая производительность благодаря оптимизации скорости и низкому уровню накладных расходов.
* Встроенная поддержка для проверки запросов и ответов на основе схемы.
* Расширяемость с помощью плагинов и middleware.
* Поддержка асинхронного программирования с использованием современных JavaScript-функций.
* Удобный и понятный API для разработчиков.

В процессе работы с Fastify вы можете создавать простые или сложные API, обрабатывать различные HTTP-методы (GET, POST, PUT, DELETE), отвечать на запросы JSON-данными и тестировать свои приложения с помощью инструментов, таких как Apidog.

NGINX — это свободный и открытый веб-сервер, работающий на операционной системе Linux, Unix и Windows. Он был создан Игорем Сысоевым в 2004 году и изначально предназначался для обслуживания сайтов с высокой посещаемостью.

NGINX способен обрабатывать большое количество одновременных подключений благодаря своей архитектуре и поддержке асинхронного ввода-вывода.

Основные функции NGINX:

* Кэш: NGINX может кэшировать статические файлы, такие как HTML, CSS, JavaScript, изображения и видео, чтобы уменьшить нагрузку на сервер и ускорить загрузку страниц.
* Балансировка нагрузки: NGINX может распределять входящие запросы между несколькими серверами, обеспечивая высокую доступность и стабильность работы системы.
* Обратный прокси: NGINX может выступать в роли обратного прокси-сервера, передавая запросы на другие серверы, такие как сервер приложений или база данных, и возвращая ответы обратно пользователям.
* Шифрование SSL/TLS: NGINX поддерживает шифрование SSL/TLS для защиты передачи данных между клиентами и серверами.
* Модульность: NGINX состоит из модулей, которые можно настраивать и расширять в зависимости от потребностей проекта.
* Гибкая конфигурация: NGINX предоставляет мощные инструменты для настройки и управления веб-сервером, включая возможность использования регулярных выражений и переменных.
* Мониторинг и логирование: NGINX может собирать информацию о работе сервера и предоставлять отчёты о состоянии системы, а также записывать логи для анализа и устранения неполадок.

NGINX широко используется в крупных и средних проектах, обеспечивая высокую производительность, стабильность и безопасность работы веб-сайтов и приложений.

1.2.4 Этапы разработки

Разработка логики серверной части web-приложения будет состоять из трех основных этапов:

* Разработка контрллеров (Отвечают за бизнес-логику)
* Разработка модулей (Отвечают за обработку запросов и проверку прав)
* Разработка роутеров (Отвечают за роуторизацию маршрутов)

Разработка клиентской части web-приложения будет состоять из 3 этапов:

* Разработка верстки
* Разработка контроллеров (Отвечают за бизнес-логику)
* Разработка моделей (Отвечают за ввод/вывод информации)

1.2.5 Тестирование

Тестирование API (программного интерфейса приложений) — это процесс проверки обмена данными между двумя модулями программы, разными приложениями, веб-сервисами и серверами. Оно помогает выявить ошибки и оценить общую работоспособность системы.

Существует несколько видов тестирования API:

* Тестирование методов — проверка ожидаемого от API действия с данными на входе, корректностью выполнения операций и выводом ожидаемых результатов.
* Тестирование взаимодействия — проверка корректной отправки, получения данных и обработки различных сценариев использования ПО.
* Тестирование авторизации и аутентификации — проверка механизма доступа к API и функциям систем.
* Тестирование обработки ошибок — оценка того, насколько корректно API обрабатывает исключения и передаёт верные коды ошибок в ПО.
* Тестирование производительности — оценка устойчивости сервиса при большом количестве запросов в систему или активных пользователей.
* Тестирование безопасности — оценка уязвимости API и предотвращение утечки данных или несанкционированного доступа.

1.2.6 Выбор конфигурации хостинга

Виды хостингов:

* Виртуальный хостинг (Shared) — на один сервер помещается множество сайтов, каждый с ограниченным количеством ресурсов. Подходит для небольших ресурсов с небольшой аудиторией.
* Виртуальный выделенный сервер (VPS) — похож на Shared, но владелец сайта может устанавливать своё ПО. Подходит для сайтов с аудиторией от 3 до 10 тысяч посетителей в сутки.
* Выделенный физический сервер (Dedicated) — весь сервер принадлежит одному пользователю, подходит для масштабных проектов.Облачный хостинг (Cloud hosting) — использует несколько серверов для равномерного распределения нагрузки и увеличения отказоустойчивости. Подходит для любых проектов.

Операционные системы для серверов отличаются от обычных ОС, используемых на домашних компьютерах. Они предназначены для работы с большим объёмом памяти, выполнения функций веб-сервера, сервера приложений и электронной почты, а также для подключения множества пользователей к локальной сети и Интернету.

Популярные серверные операционные системы включают:

* Microsoft Windows Server — корпоративная ОС с функциями управления хранением данных, приложениями и сетями.
* macOS Server — серверная ОС от Apple, включающая возможности программирования сервера и управления пользователями.
* Linux: Red Hat Enterprise Linux, Ubuntu Server и CentOS — альтернативные варианты на базе Linux.
* UNIX: различные серверные ОС, такие как FreeBSD и OpenBSD.Каждая из этих операционных систем имеет свои преимущества и недостатки, и выбор зависит от потребностей и бюджета компании.

Основные технические характеристики сервера:

* Колличество ядер процессора
* Объем оперативной памяти
* Объем внутреней памяти (SSD)
* Скорость интернета

Выбраная конфигурация сервера:

* Тип сервера: Виртуальный выделенный сервер (VPS).
* Операционная система: Ubuntu Server 22.04.
* Предустоновленное ПО: NGINX, MongoDB, Node.js 20.10.
* Технические характеристики:
  + Колличество ядер процессора: 1 ядро.
  + Объем оперативной памяти: 4ГБ.
  + Объем внутреней памяти (SSD): 25ГБ.
  + Скорость интернета: 200Мбит/c

Глава II. Практическая часть.

2.1. Разработка серверной части web-приложения.

2.1.1 Разработка контроллеров

Контроллеры отвечают за бизнес-логику приложения. Они являются классами со статическими методами.

Контроллер пользователя (UserController) имеет методы:

**Метод** **Register**

Метод отвечает за регистрацию пользователя.

Метод принимает в параметры данные регистрации:

* name – Имя пользователя.
* login – Логин пользователя.
* password – Пароль пользователя.
* email – Электронная почта пользователя.

После получения данных, выполняется запрос к базе данных, для проверки сущствования пользователя с полученными данными (логин, электронная почта), в случае если такой пользователь существует, метот возвращает ошибку с сообщением о том, что пользователь уже сужествует. В случае, если пользователя с такими данными не существует в базе данных, выполняется формирование классического объекта пользователя, с использованием полученных данных. После формирования объекта, выполняется его сохранение в базе данных. После этого выполняется формирование классического объекта верефикационной сессии, данная ссессия сохраняется в базе данных. После этого выполняется отправка кода верефикации пользователю на почту, и выполняется запуск асинхронной функции удаления данной сессии через определенный интервал времяни. Метод ничего не возвращает

**Метод** **Login**

Метод отвечает за авторизацию пользователя.

Метод принимает в параметры данные авторизации:

* login - Логин пользователя.
* Password - Пароль пользователя.

После получения данных, выполняется запрос к базе данных, для получения данных пользователя по логину, в случае если пользователя с таким логином не найден, метот возвращает ошибку с сообщением о том что пользователь не найден. В случае если пользователь с таким логином существует, выполняется проверка сапоставления хеша пароля пользователя в базе данных с полученным паролем, в случае если полученный пароль не совпадает с хешем пароля, метод возвращает ошибку что пользователь не найден. В случае если хеш пароля и полученный пароль совпадают, то выполняется генерация пары JWT-токенов. Метод возвращает пару JWT-токенов.

**Метод** **LoginForRefresh**

Метод отвечает за авторизацию пользователя с помощью refresh-токена.

Метод принимает в параметры refresh-токен.

После получения данных, выполняется проверка типа полученного токена, в случае если тип токена не является “refresh”, то метод возвращает ошибку с сообщением о том что токен не является валидным. В ином случае, выполняется запрос к базе данных для получения пользователя с тем же идентификатором что имеет JWT-токен. С помощью полученных данных, выполняется генерация пары JWT-токенов с полученными из базы даных данными. Метод возвращает пару JWT-токенов.

**Метод** **RemoveUser**

Метод отвечает за удаление пользователя.

Метот получает в параметрах идентификатор пользователя.

После получения данных, выполняется запрос к базе данных для получения данных пользователя оп полученному идентификатору. В случае если пользователь с таким иденьтфткатором не найде, метод позвращает ошибку с сообщением что пользователь не найден. В ином случае, метод выполняет запрос к базе данных, для удения пользователя. Метод ничего не возвращает.

**Метод** **VerifiedUser**

Метод отвечает за подтверждения почты пользователя.

Метод получает в параметрах идентификатор пользователя и код верефикации почты.

После получения данных, метод выполняет запрос к базе данных, для получения пользователя с таким идентификатором. В случае, если пользователь не найден, метод возвращает ошибку с сообщением что пользователь не найден. В ином случае метод выполняет запрос к базе данных для получения сессии верефикации пользователя. Далее выполняется проверка полученного кода верефикации с кодом в сессии. В случае, если коды не совпадают, метод возвращает ошибку с сообщением, о том что верефикация не пройдена. В ином случае, метод выполняет запрос к базе данных, для удаления сессии верефикации, и изменения статуса верефикации пользователя. Метод ничего не возвращает.

**Метод** **VerificationReload**

Метод отвечает за выполнение обновления ссессии верефикации пользователя.

В параметрах метод получает идентификатор пользователя.

После получения данных, метод выполняет запрос к базе данных для получения данных пользователя, с полученным идентификатором. В случае если пользователь не найден, метод возвращает ошибку с сообщением о том что пользователь не найден. В ином случае, метод выполняет запрос к баззе данных, для получения сессии верефикации пользователя. В случае если сессия существует, выполняется запрос к базе данных для обновления данных сессии с новым кодом верефикации. В случае если сессии не существует, выполняется генерация классического объекта сессии верефикации, и выполняется запрос к базе данных, для добовления новой сессиии. Метод ничего не возвращает

**Метод changeUserData**

Метод отвечает за обновление данных пользователя.

Метод получает в параметрах идентификатор пользователя и данные для обновления:

* name – Имя пользователя.
* avatar – Картинка аватара пользователя.
* password – Пароль пользователя.

После получения дданных, метод выполняет запросс к базе данных, для получения данных пользователя, с полученным идентификатором. В случае если пользователь не найден, метод возвращает ошибку с сообщенияе о том что пользователь не найден. В ином случае, метод выполняет проверку полученных данных для обновления на существование в нем “password”, в случае, если он существует, метод выполняет хеширование полученного пароля и дабовляет его в “passwordHash” объекта с данными для изменения. Далее метод выполняет запрос к базе данных, для обновления данных пользователя с полученными данными. Метод ничего не возвращает.

**Метод getData**

Метод отвечает за получения данных пользователя.

Метод принимает в параметрах идентификатор пользоваетля.

После получения данных, метод выполняет запрос к базе данных, для получения данных пользователя и полученным идентификатором. В случае если пользователь с таким идентификатором не найден, метод возвращает ошибку с сообщением о том, что пользователь не найден. В ином случае, формируется объект данных пользователя. Метод возвращет данные пользователя.

**Метод getList**

Метод отвечает за получения листа коллекции пользователя.

Метод принимает в параметрах идентификатор пользователя и название коллекции.

После получения данных, метод выполняет запрос к базе данных, для получения пользователя с полученным идентификатором. В случае если пользователь с таким идентификатором не найден, метод возвращает ошибку с сообщением о том, что пользователь не найден. В ином случае метод возвращет лист с идентификаторами необходимой коллекции.

Контроллер мероприятия (EventController) имеет методы:

**Метод create**

Метод отвечает за создание мероприятия.

Метод принимает в параметрах идентификатор владельца мероприятия, и данные меропрития:

* title – Название мероприятия.
* description – Описание мероприятия.
* date – Дата мероприятия.
* time – Время мероприятия.
* geoposition – Место проведения мероприятия.

После получения данных, метод выполняет генерацию классического объекта мероприятия с получеными данными. Далее метод выполняет запрос к базе данных, для сохранения мероприятия. Метод ничего не возвращает.

**Метод remove**

Метод отвечает за удалеия мероприятия.

Метод принимает в параметрах идентификатор владельца мероприятия и идентификатор мероприятия.

После получения данных, метод выполняет запрос к базе данных, для получения данных мероприятия по полученому идентификатору мероприятия. Далее выполняется проверка полученного идентификатора владельца мероприятия и индетификатора владельца мероприятия. В случаве если они не совпадают метод возвращает ошибку с сообщением о том, что мероприятие не доступно пользователю. В ином случае выполняется запрос к базе данных для удаления мероприятия. Метод ничего не возвращает.

**Метод changeData**

Метод отвечает за изменение данных мероприятия.

Метод в параметрах получает идентификатор владельца мероприятия и идентификатор мероприятия, и данные для изменения:

* title – Название мероприятия.
* description – Описание мероприятия.
* date – Дата мероприятия.
* time – Время мероприятия.
* geoposition – Место проведения мероприятия.

После получаения данных выполняется запросс к базе данных для получения данных мероприяти с полученным идентификатором мероприятия. В случае, если мероприятие не найдено, метод возвращает ошику с сообщением о том, что мероприятие не найдено. В ином случае, выполняется проверка полученного идентификатора владельца мероприятия с идентификатором владельца мероприятия. В случае если они не совпадают, метод возвращает ошибку с сообщением о том, что мероприятие для пользователя не найденно. В ином случае, метод выполняет запрос к базе данных, для изменения данных мероприятия. Метод ничего не возвращает.

**Метод getData**

Метод отвечает за получение данных мероприятия.

Метод принимает в параметрах идентификатор владельца мероприятия и идентификатор мероприятия.

После получаения данных выполняется запросс к базе данных для получения данных мероприяти с полученным идентификатором мероприятия. В случае, если мероприятие не найдено, метод возвращает ошику с сообщением о том, что мероприятие не найдено. В ином случае, выполняется проверка полученного идентификатора владельца мероприятия с идентификатором владельца мероприятия. В случае если они не совпадают, метод возвращает ошибку с сообщением о том, что мероприятие для пользователя не найденно. В ином случае, метод возвращает данные мероприятия.

Контроллер заметок (NoteController) имеет методы:

**Метод get**

Метод отвечает за получение заметки.

Метод принимает в параметрах идентификатор пользователя и идентификатор заметки.

После получения данных, метод выполняет запрос к базе данных, для получения заетки с полученным идентификатором заметки. В случае если заметка с таким идентификатором не найдена, метод возвращает ошибку с сообщением о том, что заметка не найдена. Далле выполняется проверка полученного идентификатора владельца заметки и идентификатора владельца заметки. В случае если они не совпадают, метод возвращает ошибку с сообщением о том, что заметки для пользователя не найдено. В иномслучае, метод возвращает данные заметки.

**Метод create**

Метод отвечает за создание заметки.

Метод принимает в параметры идентификатор владельца заметки, идентификатор заметки и данные заметки:

* title – Название заметки.
* content – Содержимое заметки.

После получения данных, метод выполняет генерацию классического объекта заметки с полученными данными. Далее метод выполняет запрос к базе данных для сохранения заметки. Метод ничего не возвращает.

**Метод getAll**

Метод отвечает за получение всех заметок пользователя.

Метод принимает в параметрах идентификатор владельца заметок.

После получения данных, метод выполняет запрос к базе данных, для получения всех заметок с идентификатором владельца. Далее метод возвращает полученный список.

**Метод removeById**

Метод отвечает за удаление заметки.

Метод принимает в параметрах идентификатор владельца заметки и идентификатор заметки.

После получения данных, метод выполняет запрос к базе данных, для получения данных заметки по ее идентификатору. В случае, если заметка не существует, метод возвращает ошибку с сообщением о том, что заметки не найдено. В ином случае, метод выполняет проверку полученного идентификатора владельца заметки с фактическим идентификатором владельца заметки. В случае, если идентификаторы не совпадают, метод возвращает ошибку с сообщением о том, что заметка для пользователя не найдена. В ином случае выполняется запросс к базе данных для удаления заметки. Метод ничего не возвращает.

**Метод changeContent**

Метод отвечает за изменение контента заметки.

Метод принимает в параметрах идентификатор владельца заметки, идентификатор заметки и новый контент заметки.

После получения данных, метод выполняет запрос к базе данных, для получения данных заметки по ее идентификатору. В случае, если заметка не существует, метод возвращает ошибку с сообщением о том, что заметки не найдено. В ином случае, метод выполняет проверку полученного идентификатора владельца заметки с фактическим идентификатором владельца заметки. В случае, если идентификаторы не совпадают, метод возвращает ошибку с сообщением о том, что заметка для пользователя не найдена. В ином случае, метод выполняет запрос к базе данных для обновления данных контента заметки.

Контролле файлов (FileController) имеет методы:

**Метод GetFile**

Метод отвечает за получение файла.

Метод принимает в параметрах название файла.

После получения данных, метод выполняет запрос к файловому серверу, для получения буффра файла. Метод возвращает буффер файла.

**Метод uploadFile**

Метод отвечает за загрузку файлов

Метод принимает в параметрах разширение файла и буффер файла.

После получения данных, метод выполняет запрос к файловуму серверу для сохранения файла и получения сгенерированного файловым сервером название файла. В случае, если файл не найден, метод возвращает ошибку с сообщением о том, что файл не найден. Метод возвращает название файла.

Контроллер расписания (ScheduleController) имеет методы:

**Метод get**

Метод отвечает за получение расписания пользователя.

Метод принимает в параметрах идентификатор пользователя и дату месаца.

После получения данных, метод проверяет, передан ли пораметр даты. В случае, если он не передан, то метод выполняет запрос к базе данных для получения объекта расписания пользователя по идентификатору пользователя. В ином случае, метод выполняет запрос к базе данных для получения объекта расписания за полученный месац. Метод возвращает объект расписания.

**Метод createTask**

Метод отвечает за создание задачи в расписании пользователя.

Метод принимает в параметрах идентификатор пользователя и данные задачи:

* date – Дата задачи.
* time – Время задачи.
* title – Название задачи.
* description – Описание задачи.

После получения данных, метод выполняет генерацию классического объекта задачи расписания с полученными данными. Далее, метод выполняет запрос к базе данных для получения расписания пользователя. Из полученных данных, формируется дерево расписания. После этого в данное дерево интергируется созданная задача. Далее метод выполняет запрос к базе данных, для сохранения изменений. Метод ничего не возвращает.

**Метод chandeTask**

Метод отвечает за изменение данных задачи в рассписании пользователя.

Метод принимает в параметрах идентификатор пользователя, идентификатор задачи и данные задачи для изменения:

* date – Дата задачи.
* time – Время задачи.
* title – Название задачи.
* description – Описание задачи.

После получения данных, метод выполняет запрос к базе данных для получения расписания пользователя. Из полученных данных генерируется дерево расписания. Далее в этом дере выполняется поиск необходимой задачи. Найденная задача изменяется полученными данными. Далее выполняется запрос к базе данных для сохранения расписания. Метод ничего не возвращает.

**Метод removeTask**

Метод отвечает за удаление задачи из расписания пользователя.

Метод принимает в параметрах идентификатор пользователя и идентификатор задачи.

После получения данных, метод выполняет запрос к базе данных для получения расписания пользователя. Из полученных данных генерируется дерево расписания. Далее в этом дере выполняется поиск необходимой задачи. Найденная задача удаляется из расписания. Далее выполняется запрос к базе данных для сохранения расписания пользователя. Метод ничего не возвращает.

2.1.2 Разработка модулей.

Модули отвечают за аунтефикацию пользователя, получения данных от клиента, выполнение бизнес-логипи приложения (выполнение методов контроллеров) и последующую отпраку данных клиенту.

Аунтефикация пользователя выполняется за счет получения заголовка Authorization у запросса, каторый несет в себе JWT токен. После его получения, происходит проверка подписи токена, и последующее получение данных пользователя из токена (идентификатора пользователя). В случае если подпись не является оригинальной, или в случае усторевания токена, клиенту возвращается ошибка, и дальнейшее выполнение бизнес-логики не происходит.

Далее будут рассписанны необходимые входные данные для модулей, и получаемые выходные данные.

Молуль пользователя (UserModule) имеет методы:

**Метод login**

Входные параметры:

* login - Логин пользователя
* password - Пароль пользователя

Выходные данные:

* statusCode – Код состояния запросса
* data – Пара JWT-токенов
* code – Код ошибки при ее возникновении
* description – Описание ошибки при ее возникновении

**Метод register**

Входные параметры:

* name - Имя пользователя.
* email - Электонная почта пользователя.
* login - Логин пользователя.
* password - Пароль пользователя.

Выходные данные:

* statusCode – Код состояния запросса
* code – Код ошибки при ее возникновении
* description – Описание ошибки при ее возникновении

**Метод loginForRefresh**

Входные параметры:

* Authorization – JWT токен.

Выходные данные:

* statusCode – Код состояния запросса
* data – Пара JWT-токенов
* code – Код ошибки при ее возникновении
* description – Описание ошибки при ее возникновении

**Метод getUserData**

Входные параметры:

* Authorization – JWT токен.

Выходные данные:

* statusCode – Код состояния запросса
* data – Данные пользователя
* code – Код ошибки при ее возникновении
* description – Описание ошибки при ее возникновении

**Метод removeUser**

Входные параметры:

* Authorization – JWT токен.

Выходные данные:

* statusCode – Код состояния запросса
* code – Код ошибки при ее возникновении
* description – Описание ошибки при ее возникновении

**Метод verifiedUser**

Входные параметры:

* Authorization – JWT токен.
* verificationCode – Код подтверждения полькователя

Выходные данные:

* statusCode – Код состояния запросса
* code – Код ошибки при ее возникновении
* description – Описание ошибки при ее возникновении

**Метод verificationReload**

Входные параметры:

* Authorization – JWT токен.

Выходные данные:

* statusCode – Код состояния запросса
* code – Код ошибки при ее возникновении
* description – Описание ошибки при ее возникновении

**Метод changeUserData**

Входные параметры:

* Authorization – JWT токен.
* name – имя пользователя
* avatar – Картинка аватара пользователя
* password – Пароль пользователя

Выходные данные:

* statusCode – Код состояния запросса
* code – Код ошибки при ее возникновении
* description – Описание ошибки при ее возникновении

**Метод getList**

Входные параметры:

* Authorization – JWT токен.
* listType – Название коллекции.

Выходные данные:

* statusCode – Код состояния запросса
* data – Список идентификаторов жлементов коллекции пользователя.
* code – Код ошибки при ее возникновении
* description – Описание ошибки при ее возникновении

Молуль мероприятий (EventModule) имеет методы:

**Метод get**

Входные параметры:

* Authorization – JWT токен.
* eventId – идентификатор мероприятия.

Выходные данные:

* statusCode – Код состояния запросса
* data – Данные мероприятия.
* code – Код ошибки при ее возникновении
* description – Описание ошибки при ее возникновении

**Метод create**

Входные параметры:

* Authorization – JWT токен.
* title – Название мероприятия.
* Description – Описание мероприятия.
* Date – Дата мероприятия
* Time – Время мероприятия
* Geoposition – Место проведения мероприятия

Выходные данные:

* statusCode – Код состояния запросса
* code – Код ошибки при ее возникновении
* description – Описание ошибки при ее возникновении

**Метод remove**

Входные параметры:

* Authorization – JWT токен.
* eventId – идентификатор мероприятия.

Выходные данные:

* statusCode – Код состояния запросса
* code – Код ошибки при ее возникновении
* description – Описание ошибки при ее возникновении

**Метод changeData**

Входные параметры:

* Authorization – JWT токен.
* eventId – идентификатор мероприятия.
* title – Название мероприятия.
* Description – Описание мероприятия.
* Date – Дата мероприятия
* Time – Время мероприятия
* Geoposition – Место проведения мероприятия

Выходные данные:

* statusCode – Код состояния запросса
* data – Данные мероприятия.
* code – Код ошибки при ее возникновении
* description – Описание ошибки при ее возникновении

Молуль заметок (NoteModule) имеет методы:

**Метод get**

Входные параметры:

* Authorization – JWT токен.
* noteId– идентификатор заметки.

Выходные данные:

* statusCode – Код состояния запросса
* data – Данные заметки.
* code – Код ошибки при ее возникновении
* description – Описание ошибки при ее возникновении

**Метод create**

Входные параметры:

* Authorization – JWT токен.
* title – Название заметки
* content – Контент заметки

Выходные данные:

* statusCode – Код состояния запросса
* code – Код ошибки при ее возникновении
* description – Описание ошибки при ее возникновении

**Метод getAll**

Входные параметры:

* Authorization – JWT токен.

Выходные данные:

* statusCode – Код состояния запросса.
* data - Список с данными заметок пользователя.
* code – Код ошибки при ее возникновении.
* description – Описание ошибки при ее возникновении.

**Метод remove**

Входные параметры:

* Authorization – JWT токен.
* noteId– идентификатор заметки.

Выходные данные:

* statusCode – Код состояния запросса.
* code – Код ошибки при ее возникновении.
* description – Описание ошибки при ее возникновении.

**Метод changeContent**

Входные параметры:

* Authorization – JWT токен.
* noteId– Идентификатор заметки.
* сontent - Контент заметки.

Выходные данные:

* statusCode – Код состояния запросса.
* code – Код ошибки при ее возникновении.
* description – Описание ошибки при ее возникновении.

Молуль расписания (SchedulesModule) имеет методы:

**Метод get**

Входные параметры:

* Authorization – JWT токен.
* date – Дата месаца

Выходные данные:

* statusCode – Код состояния запросса.
* data – объект расписания за месяц
* code – Код ошибки при ее возникновении.
* description – Описание ошибки при ее возникновении.

**Метод get**

Входные параметры:

* Authorization – JWT токен.
* date – Дата месаца

Выходные данные:

* statusCode – Код состояния запросса.
* data – объект расписания за месяц
* code – Код ошибки при ее возникновении.
* description – Описание ошибки при ее возникновении.

**Метод createTask**

Входные параметры:

* Authorization – JWT токен.
* date – Дата задачи расписания.
* time – Время задачи расписания.
* title – название задачи расписания.
* description – описание задачи расписания.

Выходные данные:

* statusCode – Код состояния запросса.
* code – Код ошибки при ее возникновении.
* description – Описание ошибки при ее возникновении.

**Метод changeTask**

Входные параметры:

* Authorization – JWT токен.
* taskId – Идентификатор задачи расписания.
* date – Дата задачи расписания.
* time – Время задачи расписания.
* title – название задачи расписания.
* description – описание задачи расписания.

Выходные данные:

* statusCode – Код состояния запросса.
* code – Код ошибки при ее возникновении.
* description – Описание ошибки при ее возникновении.

**Метод removeTask**

Входные параметры:

* Authorization – JWT токен.
* taskId – Идентификатор задачи расписания.

Выходные данные:

* statusCode – Код состояния запросса.
* code – Код ошибки при ее возникновении.
* description – Описание ошибки при ее возникновении.

Молуль файлов (FileModule) имеет методы:

**Метод get**

Входные параметры:

* filename – Название файла.

Выходные данные:

* Buffer – буфер файла.

**Метод upload**

Входные параметры:

* Authorization – JWT токен.
* Filename – Название файла.

Выходные данные:

* statusCode – Код состояния запросса.
* data – Название файла на файловом сервере.
* code – Код ошибки при ее возникновении.
* description – Описание ошибки при ее возникновении.

2.1.3 Разработка роутеров.

Роутеры отвечают за роуторизацию запросов к методам модулей. Они отвечают за валидацию получаемых данных, и отправку клиенту результата выполнения метода модуля.

Далее будет предоставоенна информация о всех роутах серверной части web-приложения.

**Авторизация пользователя по логину и паролю**

* Маршрут запроса - /api/user/login
* Метод запроса – GET
* Наличие аунтефикации – Нет
* Модуль – UserModule
* Метод модуля – login

**Авторизация пользователя по токену**

* Маршрут запроса - /api/user/loginForRefresh
* Метод запроса – GET
* Наличие аунтефикации – Да
* Модуль – UserModule
* Метод модуля – loginForRefresh

**Регистрация пользователя**

* Маршрут запроса - /api/user/register
* Метод запроса – POST
* Наличие аунтефикации – Нет
* Модуль – UserModule
* Метод модуля – register

**Получения данных пользователя**

* Маршрут запроса - /api/user
* Метод запроса – GET
* Наличие аунтефикации – Да
* Модуль – UserModule
* Метод модуля – getUserData

**Удаление данных пользователя**

* Маршрут запроса - /api/user
* Метод запроса – DELETE
* Наличие аунтефикации – Да
* Модуль – UserModule
* Метод модуля – removeUserData

**Верефикация пользователя**

* Маршрут запроса - /api/user/verification
* Метод запроса – GET
* Наличие аунтефикации – Да
* Модуль – UserModule
* Метод модуля – verifiedUser

**Обновление сессии верефикации пользователя**

* Маршрут запроса - /api/user/verificationReload
* Метод запроса – GET
* Наличие аунтефикации – Да
* Модуль – UserModule
* Метод модуля – verificationReload

**Обновление данных пользователя**

* Маршрут запроса - /api/user
* Метод запроса – PUT
* Наличие аунтефикации – Да
* Модуль – UserModule
* Метод модуля – changeUserData

**Получение списка коллекции пользователя**

* Маршрут запроса - /api/user/list/:listType
* Метод запроса – GET
* Наличие аунтефикации – Да
* Модуль – UserModule
* Метод модуля – getList

**Получение файла**

* Маршрут запроса - /api/file/:filename
* Метод запроса – GET
* Наличие аунтефикации – Нет
* Модуль – FileModule
* Метод модуля – get

**Загрузка файла на файловый сервер**

* Маршрут запроса - /api/file
* Метод запроса – POST
* Наличие аунтефикации – Да
* Модуль – FileModule
* Метод модуля – upload

**Получение данных мероприятия**

* Маршрут запроса - /api/event/:eventId
* Метод запроса – GET
* Наличие аунтефикации – Да
* Модуль – EventModule
* Метод модуля – get

**Создание мероприятия**

* Маршрут запроса - /api/event
* Метод запроса – POST
* Наличие аунтефикации – Да
* Модуль – EventModule
* Метод модуля - create

**Удаление данных мероприятия**

* Маршрут запроса - /api/event/:eventId
* Метод запроса – DELETE
* Наличие аунтефикации – Да
* Модуль – EventModule
* Метод модуля – remove

**Изменение данных мероприятия**

* Маршрут запроса - /api/event/:eventId
* Метод запроса – PUT
* Наличие аунтефикации – Да
* Модуль – EventModule
* Метод модуля – changeData

**Получение данных заметки**

* Маршрут запроса - /api/users/note/:noteId
* Метод запроса – GET
* Наличие аунтефикации – Да
* Модуль – NoteModule
* Метод модуля - get

**Получение данных заметки**

* Маршрут запроса - /api/users/note/:noteId
* Метод запроса – GET
* Наличие аунтефикации – Да
* Модуль – NoteModule
* Метод модуля – get

**Создание заметки**

* Маршрут запроса - /api/users/note
* Метод запроса – POST
* Наличие аунтефикации – Да
* Модуль – NoteModule
* Метод модуля – create

**Получение данных всех заметкок пользователя**

* Маршрут запроса - /api/users/notes
* Метод запроса – GET
* Наличие аунтефикации – Да
* Модуль – NoteModule
* Метод модуля – getAll

**Удаление данных заметки**

* Маршрут запроса - /api/users/notes/:noteId
* Метод запроса – DELETE
* Наличие аунтефикации – Да
* Модуль – NoteModule
* Метод модуля – remove

**Получение данных заметки**

* Маршрут запроса - /api/users/note/:noteId
* Метод запроса – GET
* Наличие аунтефикации – Да
* Модуль – NoteModule
* Метод модуля - get

**Получение данных расписания за месяц**

* Маршрут запроса - /api/schedule
* Метод запроса – GET
* Наличие аунтефикации – Да
* Модуль – SchedulesModule
* Метод модуля – get

**Получение данных расписания за месяц**

* Маршрут запроса - /api/schedule
* Метод запроса – GET
* Наличие аунтефикации – Да
* Модуль – SchedulesModule
* Метод модуля - get

**Создание задачи расписания пользователя**

* Маршрут запроса - /api/schedule/task
* Метод запроса – POST
* Наличие аунтефикации – Да
* Модуль – SchedulesModule
* Метод модуля – createTask

**Изменение задачи расписания пользователя**

* Маршрут запроса - /api/schedule/task/:taskId
* Метод запроса – PUT
* Наличие аунтефикации – Да
* Модуль – SchedulesModule
* Метод модуля – changeTask

**Удаление задачи расписания пользователя**

* Маршрут запроса - /api/schedule/task/:taskId
* Метод запроса – DELETE
* Наличие аунтефикации – Да
* Модуль – SchedulesModule
* Метод модуля - removeTask

Вывод

На этом, разработка серверной части системы законченна. Серверная часть системы соответствует всем требованиям и функционалу серверной части системы.

2.2. Разработка клиентской части web-приложения.

Разработка клиентской части web-приложения необходима для создания удобного и функционального пользовательского интерфейса (UI). Клиентские технологии, такие как HTML, CSS и JavaScript,отвечают за макет, дизайн, интерактивность и функциональность UI.

Разработка клиентской части позволяет:

* Обеспечить привлекательный и интуитивно понятный пользовательский опыт.
* Создать интерактивные элементы, такие как формы, меню и галереи изображений.
* Обеспечить быструю и плавную навигацию между страницами и экранами приложения.
* Снизить нагрузку на сервер, перенося вычислительные обязанности на устройство пользователя.
* Обеспечить экономию ресурсов и снижение задержки для больших объёмов одновременного трафика.
* Улучшить безопасность приложения, используя политики безопасности контента (CSP) и Secure Socket Layer (SSL) для шифрования и защиты передачи данных.
* Обеспечить кроссбраузерную совместимость и единообразное взаимодействие с приложением на разных устройствах и разрешениях экрана.

2.2.1 Разработка верстки.

После разработки дизайн-проекта (Приложение 1-6), приступаем к разработки верски.

2.2.2 Разработка контроллеров.

Контроллеры во Frontend отвечают за выполнение следующих задач:

* Обработка POST- и GET-запросов. Контроллеры обрабатывают запросы, поступающие от пользователей, и передают их соответствующим функциям для выполнения действий, таких как отправка форм, получение данных и обновление информации на странице.
* Получение и сохранение информации в базе данных. Контроллеры взаимодействуют с базой данных для сохранения и извлечения данных, необходимых для отображения информации на странице.
* Выполнение функций и обработка данных перед отображением. Контроллеры выполняют различные операции над данными, такими как фильтрация, сортировка и агрегирование, перед тем как передать их в шаблоны для отображения на экране.

Таким образом, контроллеры играют ключевую роль во Frontend-разработке, обеспечивая связь между пользователями, базой данных и шаблонами отображения информации.

Контроллер пользователя (UserController) имеет методы:

**Метод login**

Метод отвечает за выполнение авторизации на сервере.

Метод принимает в параметрах данные авторизации:

* Login – Логин пользователя
* Password – Пароль пользователя.

Метод выполняет GET запрос на сервер по маршруту “/api/user/login”, далее метод проверяет полученый статус код. В слуяае, если статус код не является “200”, то метод возвращает ошибку с сообщением с полученным описанием ошибки.

**Метод regiser**

Метод отвечает за регистрацию пользователя.

Метод принимает в параметрах данные регистрации:

* Name – Имя пользователя
* Login – логин пользователя
* Password – пароль пользователя
* Email – электронная почта пользователя

Метод выполняет GET запрос на сервер по маршруту “/api/user/login”, далее метод проверяет полученый статус код. В слуяае, если статус код не является “201”, то метод возвращает ошибку с сообщением с полученным описанием ошибки.

**Метод loginForRefresh**

Метод отвечает за авторизацию пользователя с помощью JWT-токена.

Метод принимает в параметрах JWT-токен.

Метод выполняет GET запрос на сервер по маршруту “/api/user/loginForRefresh”, далее метод проверяет полученый статус код. В слуяае, если статус код не является “200”, то метод возвращает ошибку с сообщением с полученным описанием ошибки.

**Метод changeUserData**

Метод отвечает за изменение данных пользователя.

Метод принимает в параметры jwt-токен и данные для обновления.

Метод выполняет PUT запрос на сервер по маршруту “/api/user ”, далее метод проверяет полученый статус код. В слуяае, если статус код не является “201”, то метод возвращает ошибку с сообщением с полученным описанием ошибки

**Метод get.**

Метод отвечает за получения данных пользователя.

Метод принимает в параметрах jwt-токен.

Метод выполняет GET запрос на сервер по маршруту “/api/user ”, далее метод проверяет полученый статус код. В слуяае, если статус код не является “200”, то метод возвращает ошибку с сообщением с полученным описанием ошибки.

**Метод getList**

Метод отвечает за получения списка элементов колекций пользователя.

Метод принимает в параметрах jwt-токен и название коллекции.

Метод выполняет GET запрос на сервер по маршруту “/api/user/:listName ”, далее метод проверяет полученый статус код. В слуяае, если статус код не является “201”, то метод возвращает ошибку с сообщением с полученным описанием ошибки.

**Метод removeUser**

Метод отвечает за удаления данных пользователя.

Метод принимает в параметрах jwt-токен.

Метод выполняет DELETE запрос на сервер по маршруту “/api/user ”, далее метод проверяет полученый статус код. В слуяае, если статус код не является “204”, то метод возвращает ошибку с сообщением с полученным описанием ошибки.

**Метод verefiUser**

Метод отвечает за подтверждение электронной почты пользователя.

Метод принимает в параметрах код верефикации и jwt-токен.

Метод выполняет GET запрос на сервер по маршруту “/api/user/verification”, далее метод проверяет полученый статус код. В слуяае, если статус код не является “201”, то метод возвращает ошибку с сообщением с полученным описанием ошибки.

**Метод vereficationReload**

Метод отвечает за обновление сессии подтверждение электронной почты пользователя.

Метод принимает в параметрах jwt-токен.

Метод выполняет GET запрос на сервер по маршруту “/api/user/verificationReload”, далее метод проверяет полученый статус код. В слуяае, если статус код не является “201”, то метод возвращает ошибку с сообщением с полученным описанием ошибки.

Контроллер мероприятия (EventController) имеет методы:

**Метод get**

Метод отвечает за получение данных мероприятия.

Метод принимает в параметрах jwt-токен и идентификатор мероприятия.

Метод выполняет GET запрос на сервер по маршруту “/api/event/:eventId ”, далее метод проверяет полученый статус код. В слуяае, если статус код не является “200”, то метод возвращает ошибку с сообщением с полученным описанием ошибки.

**Метод create**

Метод отвечает за создание мероприятия.

Метод принимает в параметрах jwt-токен и данные мероприятия:

* title – Название мероприятия.
* Description – Описание мероприятия.
* Date – Дата мероприятия
* Time – Время мероприятия
* Geoposition – Место проведения мероприятия

Метод выполняет POST запрос на сервер по маршруту “/api/event ”, далее метод проверяет полученый статус код. В слуяае, если статус код не является “201”, то метод возвращает ошибку с сообщением с полученным описанием ошибки.

**Метод changeData**

Метод отвечает за обновление данных мероприятия.

Метод принимает в параметрах jwt-токен, идентификатор мероприятя и новые данные мероприятия:

* title – Название мероприятия.
* Description – Описание мероприятия.
* Date – Дата мероприятия
* Time – Время мероприятия
* Geoposition – Место проведения мероприятия

Метод выполняет PUT запрос на сервер по маршруту “/api/event/:eventId ”, далее метод проверяет полученый статус код. В слуяае, если статус код не является “201”, то метод возвращает ошибку с сообщением с полученным описанием ошибки.

**Метод remove**

Метод отвечает за удаление мероприятия.

Метод принимает в параметрах идентификатор мероприятия и jwt-токен.

Метод выполняет DELETE запрос на сервер по маршруту “/api/event/:eventId ”, далее метод проверяет полученый статус код. В слуяае, если статус код не является “204”, то метод возвращает ошибку с сообщением с полученным описанием ошибки.

Контроллер заметок (NoteController) имеет методы:

**Метод get**

Метод отвечает за получение заметки.

Метод принимает в jwt-токен и идентификатор заметки.

Метод выполняет GET запрос на сервер по маршруту “/api/user/note/:noteId ”, далее метод проверяет полученый статус код. В слуяае, если статус код не является “200”, то метод возвращает ошибку с сообщением с полученным описанием ошибки.

**Метод create**

Метод отвечает за создание заметки.

Метод принимает в параметры jwt-токен, и данные заметки:

* title – Название заметки.
* content – Содержимое заметки.

Метод выполняет POST запрос на сервер по маршруту “/api/user/note ”, далее метод проверяет полученый статус код. В слуяае, если статус код не является “201”, то метод возвращает ошибку с сообщением с полученным описанием ошибки.

**Метод getAll**

Метод отвечает за получение всех заметок пользователя.

Метод принимает в параметрах jwt-токен.

Метод выполняет GET запрос на сервер по маршруту “/api/user/notes ”, далее метод проверяет полученый статус код. В слуяае, если статус код не является “200”, то метод возвращает ошибку с сообщением с полученным описанием ошибки.

**Метод removeById**

Метод отвечает за удаление заметки.

Метод принимает в jwt-токен и идентификатор заметки.

Метод выполняет DELETE запрос на сервер по маршруту “/api/user/note/:noteId”, далее метод проверяет полученый статус код. В слуяае, если статус код не является “204”, то метод возвращает ошибку с сообщением с полученным описанием ошибки.

**Метод changeContent**

Метод отвечает за изменение контента заметки.

Метод принимает в параметрах jwt-токен, идентификатор заметки и новый контент заметки.

Метод выполняет PUT запрос на сервер по маршруту “/api/user/note/:noteId”, далее метод проверяет полученый статус код. В слуяае, если статус код не является “201”, то метод возвращает ошибку с сообщением с полученным описанием ошибки.

Контролле файлов (FileController) имеет методы:

**Метод uploadFile**

Метод отвечает за загрузку файлов

Метод принимает в параметрах jwt-токен и FormData файла.

Метод выполняет POST запрос на сервер по маршруту “/api/file”, далее метод проверяет полученый статус код. В слуяае, если статус код не является “201”, то метод возвращает ошибку с сообщением с полученным описанием ошибки.

Контроллер расписания (ScheduleController) имеет методы:

**Метод get**

Метод отвечает за получение расписания пользователя.

Метод принимает в параметрах jwt-токен и дату месаца.

Метод выполняет GET запрос на сервер по маршруту “/api/schedules ”, далее метод проверяет полученый статус код. В слуяае, если статус код не является “200”, то метод возвращает ошибку с сообщением с полученным описанием ошибки.

**Метод createTask**

Метод отвечает за создание задачи в расписании пользователя.

Метод принимает в параметрах jwt-токен и данные задачи:

* date – Дата задачи.
* time – Время задачи.
* title – Название задачи.
* description – Описание задачи.

Метод выполняет POST запрос на сервер по маршруту “/api/schedules/task ”, далее метод проверяет полученый статус код. В слуяае, если статус код не является “201”, то метод возвращает ошибку с сообщением с полученным описанием ошибки.

.

**Метод chandeTask**

Метод отвечает за изменение данных задачи в рассписании пользователя.

Метод принимает в параметрах jwt-токен, идентификатор задачи и данные задачи для изменения:

* date – Дата задачи.
* time – Время задачи.
* title – Название задачи.
* description – Описание задачи.

Метод выполняет PUT запрос на сервер по маршруту “/api/schedules/task/:taskId”, далее метод проверяет полученый статус код. В слуяае, если статус код не является “201”, то метод возвращает ошибку с сообщением с полученным описанием ошибки.

**Метод removeTask**

Метод отвечает за удаление задачи из расписания пользователя.

Метод принимает в параметрах идентификатор пользователя и идентификатор задачи.

Метод выполняет DELETE запрос на сервер по маршруту “/api/schedules”, далее метод проверяет полученый статус код. В слуяае, если статус код не является “204”, то метод возвращает ошибку с сообщением с полученным описанием ошибки.

2.2.3 Разработка модулей

Модули отвечают за обработку событий. Изменение формы, отправка формы валидация формы, и изменение информации на странице.

Модуль пользователя (UserModule) отвечает:

Отработку форм:

* Форма регистрации
* Форма авторизации
* Форма подтверждения электронной почты
* Форма изменения данных пользователя

Оброботчики событий:

* Выход пользователя
* Удаление пользователя

Модуль мероприятия (EventModule) отвечает за:

Обработку форм:

* Форма нового мероприятия
* Форма редактирования мероприятия

Обработчики событий:

* Удаление мероприятия

Вывод данных:

* Данные пользователя

Модуль заметки (NoteModule) отвечает за:

Обработку форм:

* Форма новой заметки
* Форма редактирования данных заметки

Обработка событий:

* Удаление заметки

Вывод данных:

* Список заметок
* Данные заметки

Модуль расписания (SchedulesModule) отвечает за:

Обработку форм:

* Форма новой задачи
* Форма редактирования задачи

Обработка событий:

* Удаление задачи

Вывод данных:

* Список задач
* Данные задачи

Модуль файлов (FileModule) отвечает за:

Обработку форм:

* Форма загрузки файла

Вывод

На этом, разработкаклиентской части системы законченна. Клиентская часть системы соответствует всем требованиям удобства исползования интерфейся и функционалу серверной части системы.

2.3. Тестирование API.

Цель тестирования API заключается в проверке обмена данными между модулями программы, разными приложениями, веб-сервисами и серверами. Тестирование API помогает выявить ошибки, оценить общую работоспособность системы и определить уровень её стабильности и надёжности.

Далее будет предоставленна таблица отчета тестирования функций api:

|  |  |  |  |
| --- | --- | --- | --- |
| Сценарий | **Ожидаемый статус код** | **Фактический статус код** | **Статус** |
| GET /api/user/login | 200 | 200 | OK |
| POST /api/user/register | 201 | 201 | OK |
| GET /api/user/loginForRefresh | 200 | 200 | OK |
| GET /api/user | 200 | 200 | OK |
| DELETE /api/user | 204 | 204 | OK |
| GET /api/user/verification | 200 | 200 | OK |
| GET /api/user/verificationReload | 200 | 200 | OK |
| PUT /api/user | 201 | 201 | OK |
| GET /api/user/list/:listType | 200 | 200 | OK |
| GET /api/file/:filename | 200 | 200 | OK |
| POST /api/file | 201 | 201 | OK |
| GET /api/event/:eventId | 200 | 200 | OK |
| POST /api/event | 201 | 201 | OK |
| DELETE /api/event/:eventId | 204 | 204 | OK |
| PUT /api/event/:eventId | 201 | 201 | OK |
| GET /api/Users/note/:noteId | 200 | 200 | OK |
| POST /api/user/note | 201 | 201 | OK |
| GET /api/user/notes | 200 | 200 | OK |
| DELETE /api/user/notes/:noteId | 204 | 204 | OK |
| GET /api/schedule | 200 | 200 | OK |
| POST /api/schedule/task | 201 | 201 | OK |
| PUT /api/schedule/task | 201 | 201 | OK |
| DELETE /api/schedule/task/:taskId | 204 | 204 | OK |

2.4 Этап запуска системы на хостинге.