

Министерство цифрового развития, связи и массовых коммуникаций РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра ПМиК
Допустить к защите
зав. кафедрой:
_____ д.т.н. Нечта И.В.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА Календарь событий

Пояснительная записка

Студент	_____	/	/
Институт информатики и вычислительной техники			Группа
Руководитель	_____	/	/

Новосибирск 2023г.

**Министерство цифрового развития, связи и массовых коммуникаций РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)**

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

СТУДЕНТУ Домалега В.И.

ГРУППЫ ИП-915

УТВЕРЖДАЮ

«_____» _____

зав. кафедрой ПМиК

_____/_____/

Новосибирск 2023г.

1. Тема выпускной квалификационной работы бакалавра

утверждена приказом СибГУТИ от «___»___ 20___ г. № ___

2.Срок сдачи студентом законченной работы « ___ » _____ 20___ г.

3.Исходные данные к работе

~~1.Специальная литература~~

~~2.Материалы сети интернет~~

4.Содержание пояснительной записки (перечень подлежащих разработке вопросов)	Сроки выполнения по разделам

Дата выдачи задания «___» _____ 20___ г.

Руководитель _____
подпись

Задание принял к исполнению «___» _____ 20___ г.

Студент _____

Министерство цифрового развития, связи и массовых коммуникаций РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Форма утверждена научно-методическим
 советом университета
 протокол № 3 от 19.02.2015 г.

ОТЗЫВ

на выпускную квалификационную работу студента _____
 по теме « _____ »

Оценка уровней сформированности общекультурных и профессиональных компетенций обучающегося:

Компетенции		Уровень сформированности компетенции		
		Высокий	Средний	Низкий
Универсальные	УК-1			
	УК-3			
	УК-4			
Общепрофессиональные	ОПК-2			
	ОПК-4			
	ОПК-6			
	ОПК-7			
	ОПК-8			
Профессиональные	ПК-1			
	ПК-3			
	ПК-25			
Работа имеет практическую ценность			Тема предложена предприятием	
Работа внедрена			Тема предложена студентом	
Рекомендую работу к внедрению			Тема является фундаментальной	
Рекомендую работу к опубликованию			Рекомендую студента в магистратуру	
Работа выполнена с применением ЭВМ			Рекомендую студента в аспирантуру	

Руководитель бакалаврской работы _____
 (должность, уч. степень, подпись, фамилия, имя, отчество (полностью), дата)

АННОТАЦИЯ

Выпускной квалификационной работы студента _____
(Фамилия, И.О.)

по теме «_____»

Объём работы - _____ количество страниц до приложений _____ страниц, на которых
размещены _____ рисунков и _____ таблиц. При написании работы
использовалось _____ источников.

Ключевые слова: JavaScript, postgresSQL, React.js, Express.js.

Работа выполнена _____
(название предприятия, подразделения)

Руководитель _____
(должность, уч. степень, звание, Фамилия Имя Отчество)

Основные результаты

Содержание

Содержание.....	1
Введение.....	2
1 Теоретическая часть.....	3
1.1 Сравнение архитектур.....	3
1.2 Сравнительный анализ схожих проектов	7
1.3 Цели и задачи проекта.....	13
2 Инструменты, используемые в проекте.....	14
2.1 Базы данных	14
2.2 Серверная часть проекта.....	16
2.3 Клиентская часть проекта.....	22
2.3.1 Страница регистрации и авторизации	24
2.3.2 Основная страница.....	32
2.3.3 Модальное окно.....	35
2.3.4 Боковая панель	40
3 Описание библиотек	43
3.1 Общие библиотеки	43
3.2 Библиотеки серверной части.....	45
3.3 Библиотеки клиентской части.....	55
Заключение	60
Список использованной литературы.....	62

Введение

Мы живём в эпоху быстроразвивающихся технологий, легкодоступной информации огромного объема данных и именно поэтому многие люди забывают большое количество деталей из-за колоссального объема информации обрабатываемых нашим мозгом, от самых неважных до крайне весомых таких как, например, запись к врачу или день рождения родственников. Решение этой хоть и тривиальной, но очень важной задачи существует уже с очень древних времен, а именно календари. С тех пор прошло много времени и от бумажных календарей люди перешли к более удобной их версии, а именно электронным. А так как наши жизни тесно переплетены с различными устройствами, такими как планшеты, телефоны, персональные компьютеры и прочие, то было бы удобно использовать эти устройства и в качестве календарей. Большинство современных систем идёт с предустановленным приложениями календарями, но их недостаток состоит в их автономности, а то есть записав какую-либо информацию на телефоне, вы не сможете получить уведомление, например, на другой телефон или компьютер, что крайне критично в некоторых случаях. Поэтому все большую популярность набирают веб-приложения календарей, которые позволяют синхронизировать информацию между различными устройствами и иметь доступ к ней в любой момент и в любом месте, где есть интернет.

Цель данной дипломной работы заключается в создании веб-приложения календаря, которое позволит пользователям не только записывать и просматривать свои события и задачи, но и получать уведомления о них на различных устройствах.

В работе будут рассмотрены такие важные аспекты, как проектирование базы данных для хранения информации о событиях, разработка пользовательского интерфейса, а также реализация функционала, позволяющего создавать, редактировать, удалять и просматривать события, а также настраивать уведомления для них.

Ожидаемый результат работы - удобное, легкодоступное и многофункциональное веб-приложение, которое позволит пользователям эффективно управлять своим расписанием и задачами и не пропускать важные события благодаря уведомлениям на различных устройствах.

1 Теоретическая часть

Веб-приложение - это приложение, которое работает через интернет и доступно пользователю через браузер.

Основные принципы работы веб-приложения включают в себя:

- Клиент-серверная архитектура — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами;
- HTTP протокол — протокол прикладного уровня передачи данных, изначально — в виде гипертекстовых документов в формате HTML, в настоящее время используется для передачи произвольных данных;
- И др.

Существует несколько различных вариаций архитектур создания веб-приложений. Самые популярные на данный момент виды архитектура для клиент-серверного взаимодействия это микросервисная архитектура и монолитная архитектура.

1.1 Сравнение архитектур

Монолитная архитектура - это традиционная модель программного обеспечения, которая представляет собой единый модуль, работающий автономно и независимо от других приложений. Пример графического отображения такой архитектуры приведен на рисунке 1.1. Монолитом часто называют нечто большое и неповоротливое, и эти два слова хорошо описывают монолитную архитектуру для проектирования ПО.

Монолитная архитектура — это отдельная большая вычислительная сеть с единой базой кода, в которой объединены все бизнес-задачи. Чтобы внести изменения в такое приложение, необходимо обновить весь стек через базу кода, а также создать и развернуть обновленную версию интерфейса, находящегося на стороне службы. Это ограничивает работу с обновлениями и требует много времени.

Монолиты удобно использовать на начальных этапах проектов, чтобы облегчить развертывание и не тратить слишком много умственных усилий при управлении кодом. Это позволяет сразу выпускать все, что есть в монолитном приложении.

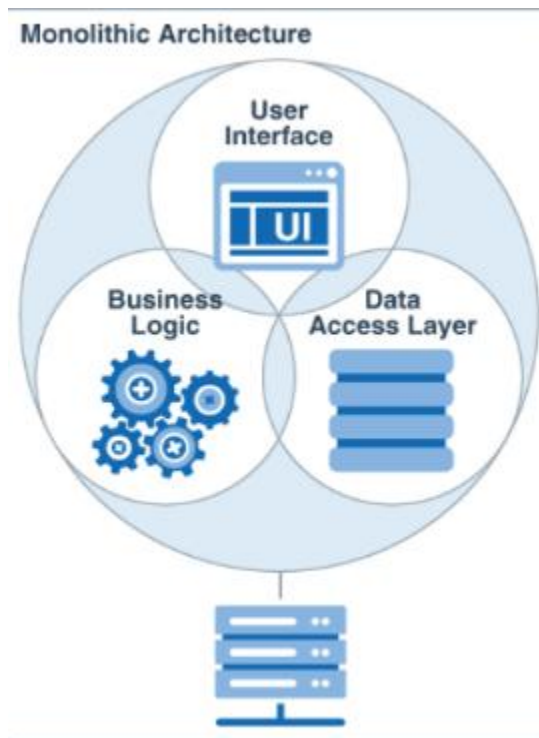


Рисунок 1.1. Графическое отображение монолитной архитектуры

Преимущества монолитной архитектуры:

- Простое развертывание приложения относительно другого подхода. Так как все необходимые файлы находятся вместе;
- Скорость разработки. Приложение с такой архитектурой разрабатываются быстрее, так как база кода едина;
- Производительность может быть гораздо выше из-за более конкретных функций реализованных в единственном API именно под нужные задачи;
- Упрощенное тестирование из-за централизации модулей приложения.

Недостатки монолитной архитектуры:

- Занижение скорости разработки при увеличении объема функционала. Из-за такой особенности подхода тяжело масштабировать проект, ведь каждый раз при каждом изменении придется перекомпилировать и перетестировать весь проект заново;
- Низкая масштабируемость больших проектов;
- Не самая высокая надежность, так как ошибка в одном из компонентов приложения может нарушить работу всего приложения;
- Ограничения по технологиям, выбрав стек технологий, например, для клиентского интерфейса в будущем его будет невозможно изменить, не переписав остальной код в клиентской части веб-приложения.
- При каждом изменении необходимо переразвертывать приложение.

Микросервисная архитектура (или просто «микросервисы») представляет собой метод организации архитектуры, основанный на ряде независимо развертываемых служб. У этих служб есть собственная бизнес-логика и база данных с конкретной целью. Пример графического отображение такого подхода отображен на Рисунке 1.2. Обновление, тестирование, развертывание и масштабирование выполняются внутри каждой службы. Микросервисы разбивают крупные задачи, характерные для конкретного бизнеса, на несколько независимых баз кода. Микросервисы не снижают сложность, но они делают любую сложность видимой и более управляемой, разделяя задачи на более мелкие процессы, которые функционируют независимо друг от друга и вносят вклад в общее целое.

Внедрение микросервисов зачастую тесно связано с DevOps, поскольку они лежат в основе методики непрерывной поставки, которая позволяет командам быстро адаптироваться к требованиям пользователей.

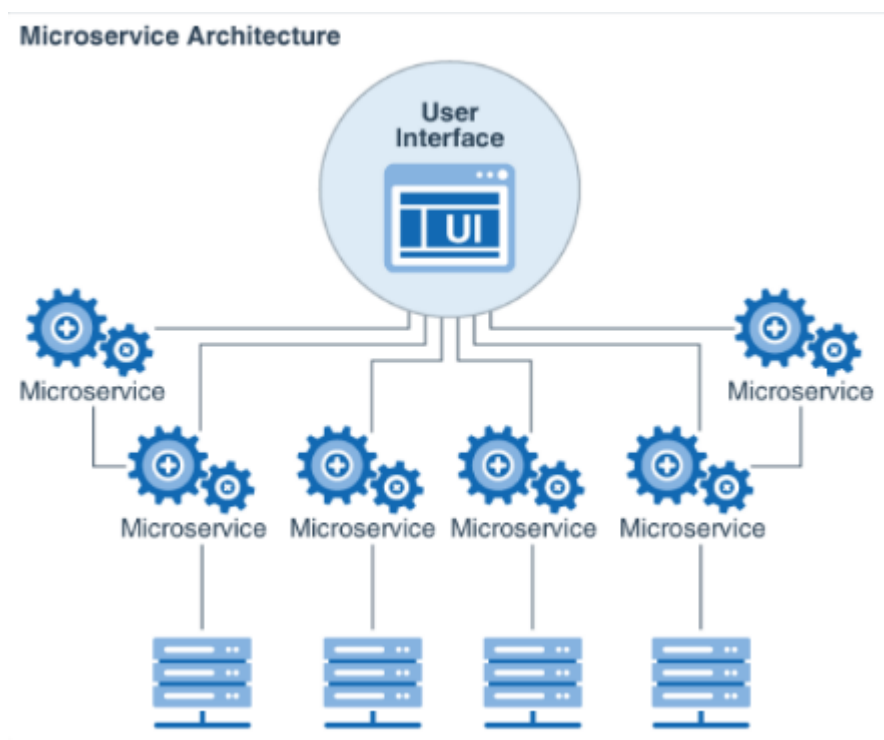


Рисунок 1.2. Графическое отображение микроархитектуры

Преимущества микросервисной архитектуры:

- Такая архитектура гораздо более гибкая, из-за того что каждый микросервис может выполнять отдельная команда разработчиков на своем стеке технологий;
- Такой подход имеет более гибкую масштабируемость;
- Непрерывное развертывание, так как каждый микросервис не связан с другим микросервисом напрямую;

- Гибкость технологий;
- Высокая надежность, так как при выходе из строя одного из сервисов, приложение не будет падать полностью.

Минусы микросервисной архитектуры:

- Большое количество микросервисов может привести к уменьшению темпа разработки из-за запутанности приложения, из-за этого стоит внимательнее следить за общим ходом разработки веб-приложения;
- В за частую такой подход разработки удобен для большого количества людей поделенных на маленькие команды, где для связи друг с другом иногда необходимы дополнительные уровни коммуникации и сотрудничества;

В соответствие со всеми этими данными, выбор пал на монолитную архитектуру разработки, так как проект будет выполнено не командой разработчиков и само приложение не может быть поделено на микросервисы.

Перед тем, как приступать к разработке собственного веб-приложения календаря, важно провести анализ уже существующих приложений данного типа на рынке. Это позволит оценить сильные и слабые стороны конкурентов, а также определить, какие функциональные возможности должны быть включены в ваше приложение, чтобы оно было конкурентоспособным.

В ходе анализа можно оценить такие параметры, как удобство использования интерфейса, функциональные возможности приложения, наличие возможности синхронизации данных между различными устройствами, степень персонализации календаря, наличие функционала уведомлений и напоминаний, а также прочие дополнительные функции, которые могут быть интересны пользователям.

Важно не только оценить уже существующие функциональные возможности приложений, но и выявить возможности для инноваций и улучшений, которые могут привлечь новых пользователей и увеличить удовлетворенность имеющихся.

Таким образом, проведение анализа существующих приложений календарей поможет определить требования к разрабатываемому приложению, а также дать представление о том, какие функции будут наиболее востребованы пользователями.

1.2 Сравнительный анализ схожих проектов

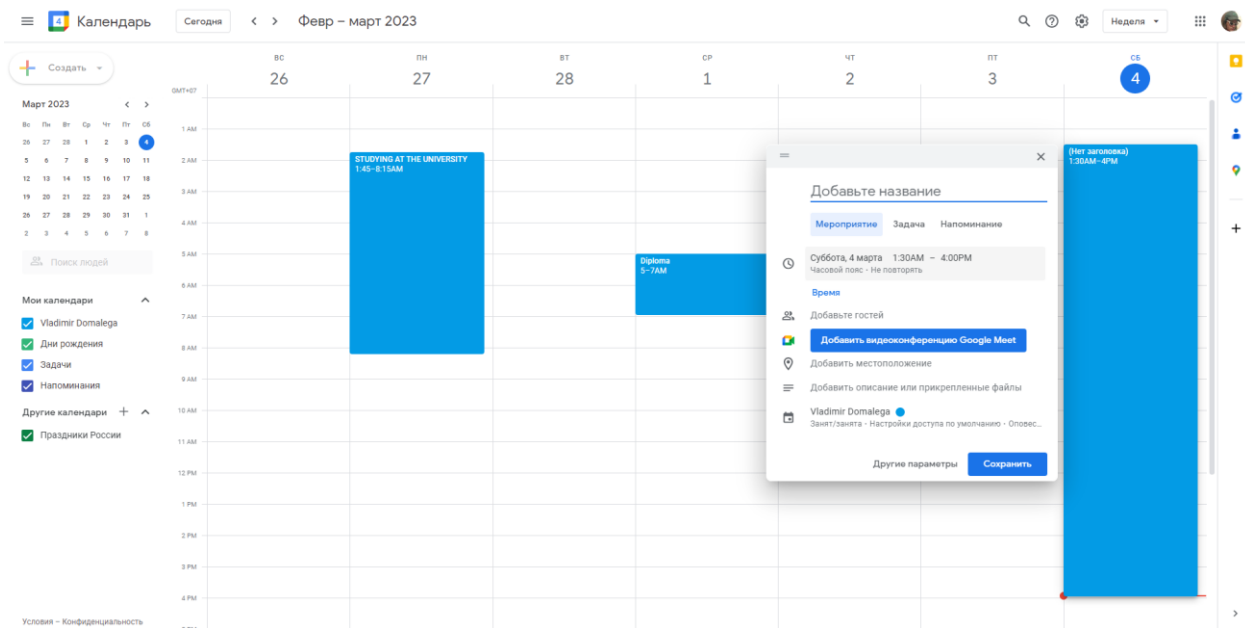
1. Google Календарь – сервис от компании Google для планирования встреч, событий и т.п. Приложение имеет как веб-версию, так и мобильные версии для android и iOS. Календарь от Google имеет обширный арсенал возможностей, например: имеется возможность задачи времени встреч, создавать повторяющиеся дат, приглашать других участников с помощью электронной почты. Пример внешнего вида приложения отображен на скриншоте 1.1;

Из особенных плюсов данной платформы можно отметить:

- Интуитивно понятный интерфейс;
- Использование почты Gmail от Google. Особенно удобно будет пользоваться приложением, тем, кто имеет почту от Google. Приложение календаря самостоятельно соберет информации о вас с других своих приложений компании и предоставит вам для более комфортного использования;
- Есть не только веб-версия, но и мобильные приложения;
- Встроенные сервисы по типу Google maps, Google meet и др.;
- Возможность использования расширений для увеличения возможностей календаря;
- Возможность создания различных календарей для разных целей;
- Обширные возможности кастомизации интерфейса и не только в настройках приложения;
- Имеется API для разработчиков под все популярные языки программирования.
- Различные варианты настройки уведомлений;
- Бесплатность;
- Имеется возможность делиться календарем с другими людьми;

Из минусов:

- Хотя и интуитивно понятный интерфейс, но переполненный разными деталями;
- Отсутствие темной, а также других цветовых тем в веб-версии;
- Зависимость от интернет соединения. Без доступа к интернету календарь не будет работать;



Скриншот 1.1. Пример интерфейса календаря от Google

2. Microsoft's outlook calendar - календарь от компании Microsoft полностью интегрированный с электронной почтой outlook. В данном приложении имеется возможность создавать встречи и расписания, делиться данными с другими пользователями с помощью приглашений по электронной почте или контактами, создавать новые календари и др. Особая черта этого календаря заключается в особенно тесной интеграции с другими приложениями компании Microsoft такими как веб-версией Office online(Word, Excel, PowerPoint и прочие), Skype, ToDo и др. Пример внешнего вида приложения отображен на скриншоте 1.2;

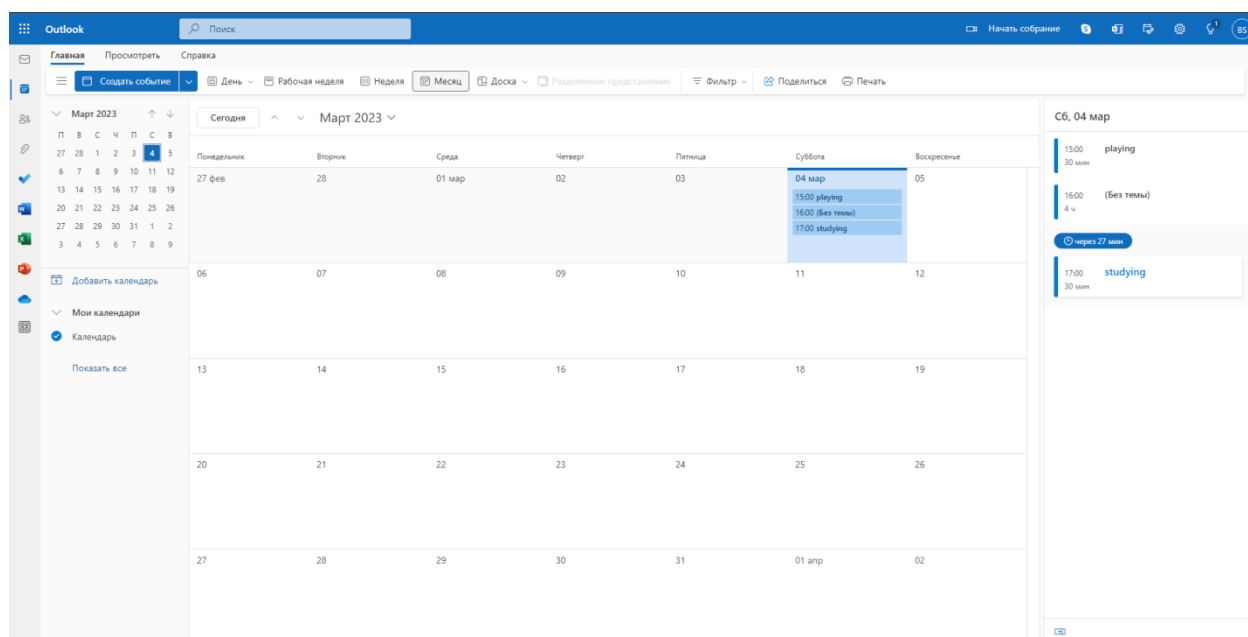
Плюсы данного календаря:

- Интуитивно понятный интерфейс;
- Интегрированность в среду Microsoft;
- Имеется API для разработчиков под все популярные языки программирования
- Различные варианты визуальных тем;
- Возможность создания разных календарей для разных целей;
- Имеется возможность делиться календарем с другими людьми;
- Бесплатность;
- Наличие функции уведомлений;

Минусы календаря Outlook:

- Хотя и интуитивно понятный интерфейс, но переполненный разными деталями;

- Зависимость от интернет соединения. Без доступа к интернету календарь не будет работать;



Скриншот 1.2. Пример интерфейса календаря от компании Microsoft

3. Яндекс Календарь - это онлайн-сервис от компании Яндекс, который позволяет пользователям создавать и редактировать расписания, задавать напоминания и делиться событиями с другими людьми. Пример внешнего вида приложения отображен на скриншоте 1.3;

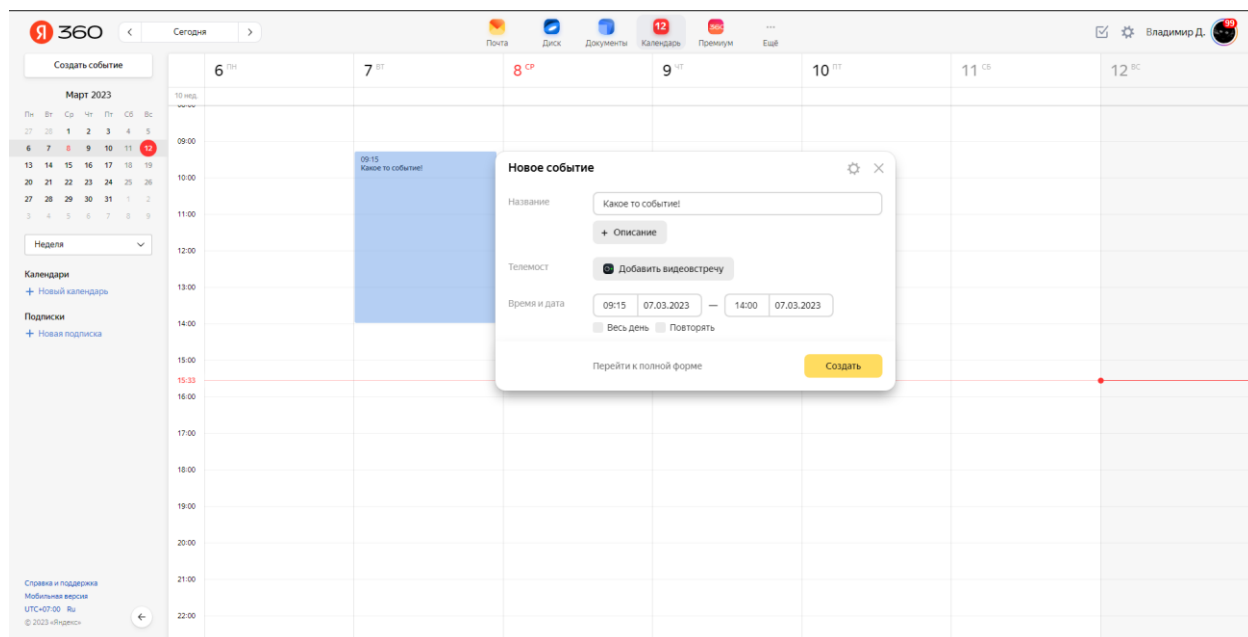
Плюсы:

- Удобный интерфейс: сервис имеет простой и интуитивно понятный интерфейс, что делает использование календаря максимально простым и удобным;
- Возможность совместного использования: с помощью сервиса можно создавать события и делиться ими с другими пользователями;
- Напоминания: Яндекс Календарь предоставляет широкие возможности настройки напоминаний для событий, что помогает не пропустить важные встречи и мероприятия;
- Интеграция с другими сервисами: календарь может быть интегрирован с другими сервисами Яндекса и сторонними сервисами, что делает его еще более удобным и гибким в использовании;

Минусы:

- Ограниченные возможности настройки: в некоторых случаях, возможности настройки Яндекс Календаря могут быть ограничены;

- Интеграция с другими сервисами: хотя возможность интеграции с другими сервисами является преимуществом, в некоторых случаях могут возникнуть проблемы с синхронизацией данных.

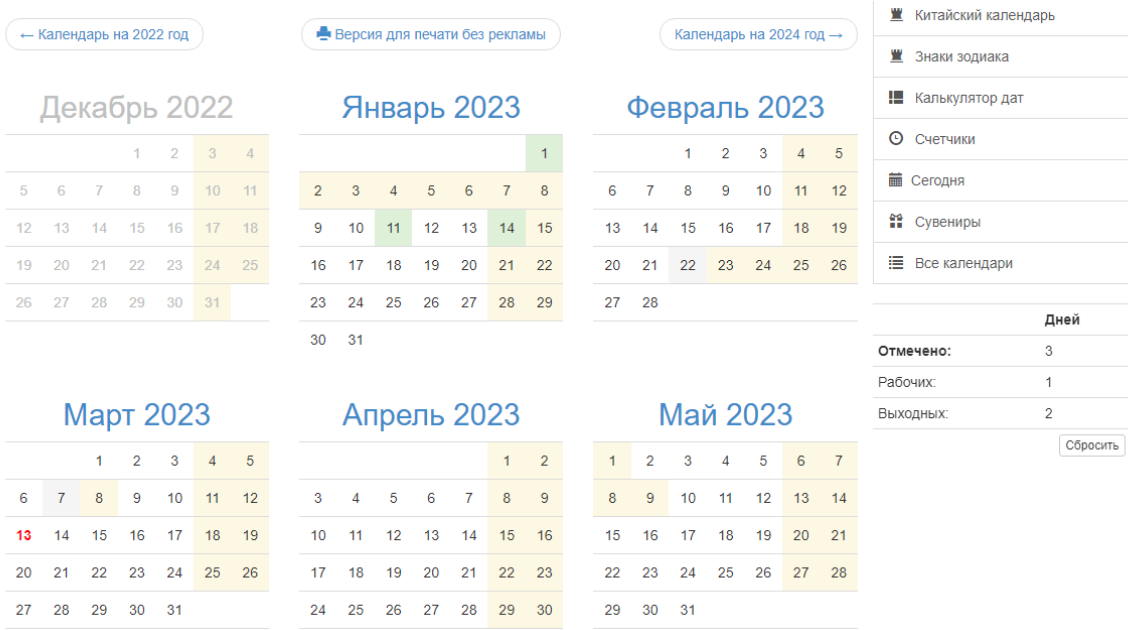


Скриншот 1.3. Пример интерфейса календаря от компании Яндекс

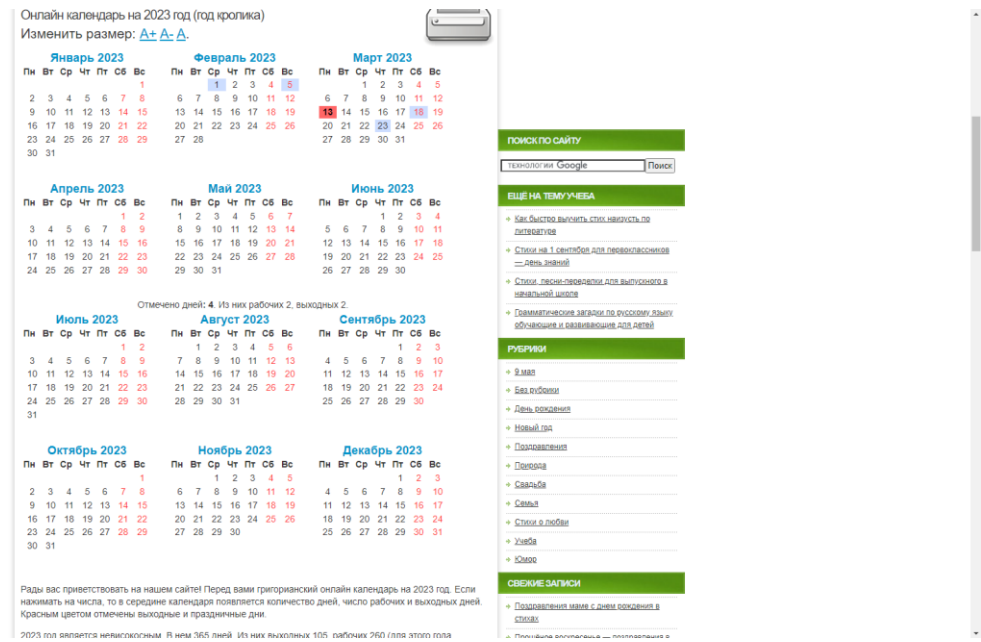
В соответствии с этими примерами можно сделать промежуточный вывод. Онлайн-календари - удобный инструмент для планирования и организации своего времени, который позволяет управлять расписанием, заданиями и напоминаниями в одном месте. Среди популярных календарей, Google Календарь отличается простотой использования, множеством функций и удобной синхронизацией с другими устройствами и сервисами. Недостатком может быть некоторая зависимость от Google и ограниченность возможностей интеграции. Outlook Календарь, в свою очередь, предлагает широкий функционал, включая управление заданиями, делами и контактами, а также поддержку бизнес-функций и интеграцию с Microsoft Office. Однако его интерфейс может показаться несколько запутанным, а синхронизация может быть не такой удобной, как у Google Календаря. Яндекс Календарь привлекает наличием умных функций и возможности создания совместных календарей. Недостатком может стать ограниченность функционала и менее удобная интеграция с другими сервисами. При выборе онлайн-календаря необходимо учитывать свои потребности и предпочтения в функционале и интерфейсе, а также возможность синхронизации с другими устройствами и сервисами. Так же хотелось бы отметить, что все календари имеют схожие интерфейсы и принципы работы, так что каждому человеку, который поработает с одним из них, сможет без проблем начать использовать и другие календари такого же типа. Как и схожий интерфейс, данные календари имеют и схожую проблему:

выбор календаря зачастую будет зависеть от того какую экосистему вы используете, а это означает тяжелый переход от одного календаря к другому.

Так же в интернете можно найти множество других решений, но большинство из них имеет огромное количество различных проблем. Вот пример нескольких таких календарей, интерфейсы которых отображены на скриншоте 1.4 и скриншоте 1.5.



Скриншот 1.4. Пример календаря с сайта «<https://calendar.yoip.ru/>»



Скриншот 1.5. Пример календаря с сайта «<https://otmetim.info/>»

Можно сразу же отметить такие проблемы как:

- Отсутствие авторизации и, следовательно, отсутствие возможности запоминать даты пользователей делает такие сайты календари сразу же непригодными для использования в качестве ежедневного инструмента;
- Низкокачественный UI/UX.

Вследствие анализа рынка можно сделать такой вывод, что существует лишь 3 качественных решения. Остальные абсолютно не могут конкурировать с ними. Некоторые другие имеют большие изъяны такие как:

- Некоторые календари имеют ежемесячную/ежегодную оплату;
- Отсутствие русского языка в интерфейсе;
- Долгое время регистрации;
- Некоторые не запоминают выбранные даты;
- Отсутствие возможности работать с другими людьми.

Так что можно сказать, что на рынке есть решения от крупных компаний, которые больше предназначены для работы с командой, но отсутствуют менее масштабные проекты для личного использования. Именно поэтому мой проект может решить эту проблему. Из описанных особенностей хотелось бы реализовать интуитивно понятный интерфейс для любого пользователя, а также возможность выбора темы интерфейса и возможность комфортного использования веб-приложения как на компьютере, так и на мобильном устройстве.

1.3 Цели и задачи проекта

Сделав предварительный анализ существующих уже онлайн-календарей можно сделать простой вывод о том, что существует два варианта календарей: первые это - многофункциональные календари, которые могут использоваться для решения большинства проблем, что однако делает их чересчур перегруженными для пользователей, которые не пользуются данными функциями. Вторые же это календари, которые не могут выполнять даже простейших действий: по типу записи даты в базы данных и тому подобное. Данный проект должен решить проблему нехватки вариантов календарей для личного использования. А так же дать мне опыт в создании собственного веб-приложения на основе языка JavaScript с использованием современных библиотек, фреймворков и стандартов языка.

Основные пункты проекта, которые должны быть реализованы в течении проекта:

- Создание серверной части приложения на языке программирования JavaScript с использованием современных подходов в программировании и инструментов для их реализации;
- Создание реляционной базы данных;
- Подключение базы данных к серверной части проекта;
- Создание веб-интерфейса с возможностью логиниться и авторизоваться;
- Возможность выбирать и записывать даты, отмеченные пользователем в базу данных, а так же отображения уже ранее выбранных дат.

Дополнительные задачи проекта:

- Подтверждение почты письмом, отправленным на почту;
- Шифрование паролей;
- Хранение данных об активном пользователе путем хранения токенов;
- Наличие, как темной, так и светлой темы;
- Помимо добавления возможность удаления и изменения сообщения, прикрепленное к дате, а также саму дату из списка отмеченных дат.

2 Инструменты, используемые в проекте

2.1 Базы данных

База данных (БД) - это структурированное хранилище данных, которое предназначено для эффективного хранения, организации и управления большим объемом информации. БД представляет собой совокупность данных, связанных между собой и организованных в соответствии с определенными правилами.

Базы данных используются во многих сферах жизни и деятельности, например, в банковском деле, ведении складского учета, интернет-магазинах, медицине и т.д. Они позволяют хранить большие объемы информации, обеспечивать быстрый доступ к данным и удобный поиск информации.

БД состоит из таблиц, каждая из которых имеет свою схему и структуру. Таблицы могут быть связаны друг с другом, что обеспечивает возможность получения данных из нескольких таблиц за один запрос. Каждая таблица состоит из строк и столбцов, где строки - это записи данных, а столбцы - это поля, которые хранят конкретную информацию.

С помощью языка запросов можно получать информацию из БД, добавлять новые данные, изменять или удалять существующие записи.

Существует несколько типов баз данных: реляционные, нереляционные, графовые и др. Реляционные базы данных - самый распространенный тип БД, в которых данные организованы в виде таблиц, связанных между собой по ключам. Примерами реляционных СУБД являются MySQL, PostgreSQL, Oracle и MS SQL Server.

В проекте я использую реляционная СУБД PostgreSQL. Рассмотрим подробнее эту базу данных.

PostgreSQL - это реляционная база данных с открытым исходным кодом, которая поддерживает множество типов данных, а также имеет возможность расширения с помощью пользовательских типов, функций и процедур.

В проекте используются 2 таблицы. Первая таблица «User», эта таблица описывает данные пользователя, введенные при регистрации на сайте. Вторая таблицы «UserData», эта таблица описывает дату и комментарий к этой дате, выбранную пользователем. В соответствии со скриншотом 1.7, таблицы имеют

связь «один ко многим» для того чтобы один пользователей мог выбирать любое количество дат.

Пример записей в базе данных PostgreSQL в приложении pgAdmin4 приведен на скриншоте 1.6.

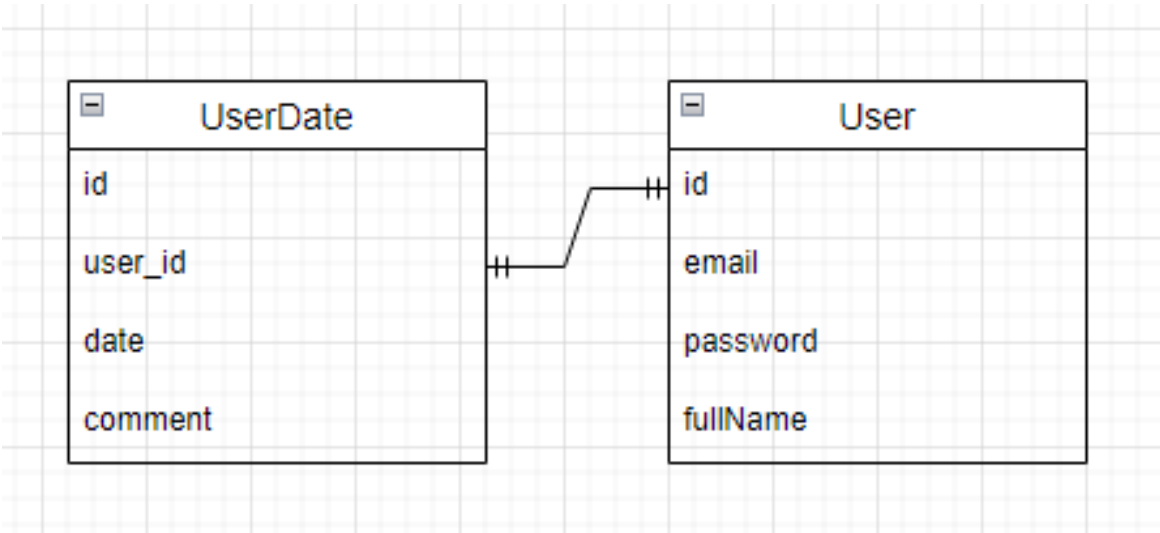
Query

1 SELECT * FROM public."userDates"
2 ORDER BY id ASC

Data Output Messages Query History Notifications

	id [PK] integer	date timestamp with time zone	comment character varying (255)	createdAt timestamp with time zone	updatedAt timestamp with time zone	userid integer
1	1	2001-11-11 00:00:00+06	test	2023-01-17 16:55:05.669+07	2023-01-17 16:55:05.669+07	11
2	3	2002-11-11 00:00:00+06	test	2023-01-17 17:36:18.097+07	2023-01-17 17:36:18.097+07	11
3	11	2003-11-11 00:00:00+06	test	2023-01-27 16:58:26.314+07	2023-01-27 16:58:26.314+07	11

Скриншот 1.6. Вид данных внутри таблица “userDate”



Скриншот 1.7. Диаграмма базы данных

2.2 Серверная часть проекта

Бэкэнд (backend) - это часть приложения, которая отвечает за обработку данных и логику работы с данными. Бэкэнд взаимодействует с базой данных и обеспечивает передачу данных между пользователем и сервером. Бэкэнд может быть написан на различных языках программирования, таких как Python, Java, Ruby, PHP, JavaScript и других.

Для бэкэнд части проекта я выбрал такой язык программирования как JavaScript. Основными причинами такого выбора стали: наличие некоторого опыта работы с данным языком, удобство написания как серверной, так и клиентской части приложения, а также большая поддержка языка как библиотеками и фреймворками, так и различного рода руководствами и документациями.

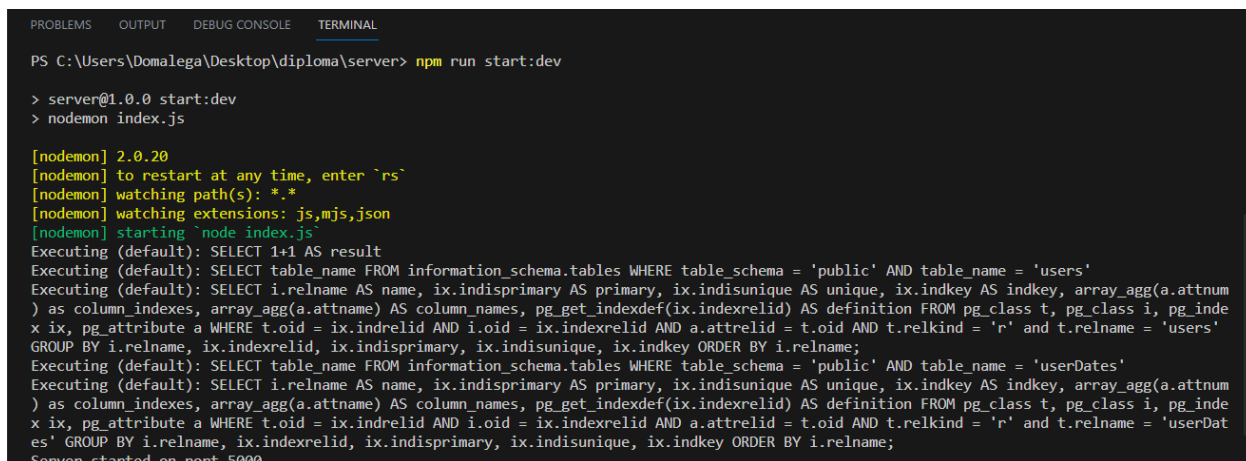
Для возможности работы с языком JavaScript вне браузера необходима программная платформа Node.js. Node.js - это среда выполнения JavaScript на сервере, которая позволяет запускать JavaScript-скрипты на стороне сервера и использовать его для создания высокопроизводительных, масштабируемых и быстрых приложений. Node.js базируется на движке V8, который используется в браузере Google Chrome для интерпретации и выполнения JavaScript-кода. Одним из главных преимуществ Node.js является его асинхронная модель выполнения, которая позволяет создавать высокоэффективные и отзывчивые приложения, обрабатывающие множество запросов одновременно.

Для создания RestAPI был выбран один из самых популярных и в тоже время простых фреймворков – express.js. В свою очередь Express.js - это минималистичный и гибкий веб-фреймворк для приложений Node.js, предоставляющий обширный набор функций для мобильных и веб-приложений, имеющий множества служебных методов HTTP и промежуточных обработчиков для создания API. На скриншоте 1.8 приведен пример запуска простейшего сервера на этой платформе, который выведет на экран пользователя “Hello World” при переходе по роутеру “/”, а так же выведется “Example app listening on port 3000!”.

```
1  const express = require('express')
2  const app = express()
3
4  app.get('/', (req, res) => res.send('Hello World!'))
5
6  app.listen(3000, () => console.log('Example app listening on port 3000!'))
```

Скриншот 1.8. Пример запуска сервера

В соответствии со скриншотом 1.9, с помощью скрипта “npm run start:dev” запускается локальный сервер «http://localhost:5000/».



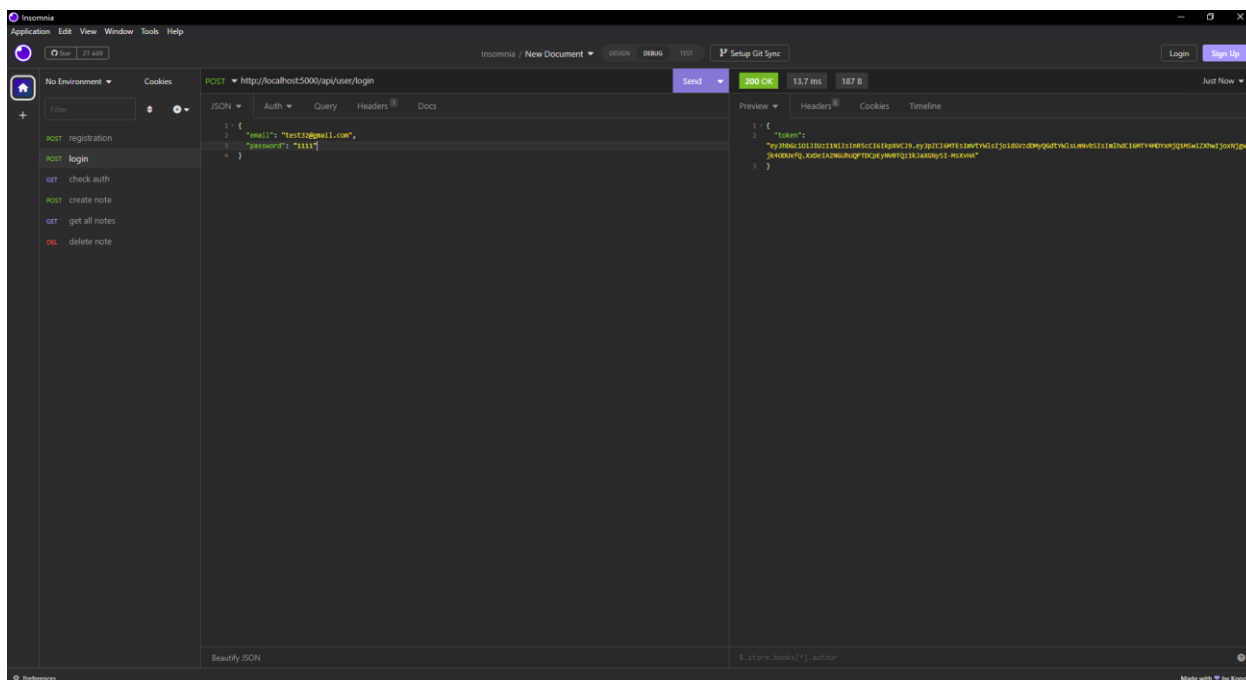
```
PS C:\Users\Domalega\Desktop\diploma\server> npm run start:dev

> server@1.0.0 start:dev
> nodemon index.js

[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
Executing (default): SELECT 1+1 AS result
Executing (default): SELECT table_name FROM information_schema.tables WHERE table_schema = 'public' AND table_name = 'users'
Executing (default): SELECT i.relname AS name, ix.indisprimary AS primary, ix.indisunique AS unique, ix.indkey AS indkey, array_agg(a.attnum
) as column_indexes, array_agg(a.attname) AS column_names, pg_get_indexdef(ix.indexrelid) AS definition FROM pg_class t, pg_class i, pg_inde
x ix, pg_attribute a WHERE t.oid = ix.indrelid AND i.oid = ix.indexrelid AND a.attrelid = t.oid AND t.relkind = 'r' and t.relname = 'users'
GROUP BY i.relname, ix.indexrelid, ix.indisprimary, ix.indisunique, ix.indkey ORDER BY i.relname;
Executing (default): SELECT table_name FROM information_schema.tables WHERE table_schema = 'public' AND table_name = 'userDates'
Executing (default): SELECT i.relname AS name, ix.indisprimary AS primary, ix.indisunique AS unique, ix.indkey AS indkey, array_agg(a.attnum
) as column_indexes, array_agg(a.attname) AS column_names, pg_get_indexdef(ix.indexrelid) AS definition FROM pg_class t, pg_class i, pg_inde
x ix, pg_attribute a WHERE t.oid = ix.indrelid AND i.oid = ix.indexrelid AND a.attrelid = t.oid AND t.relkind = 'r' and t.relname = 'userDat
es' GROUP BY i.relname, ix.indexrelid, ix.indisprimary, ix.indisunique, ix.indkey ORDER BY i.relname;
Server started on port 5000
```

Скриншот 1.9. Запуск серверной части приложения

Из-за того что изначально разрабатывается сервер без визуального интерфейса, для проверки работоспособности HTTP запросов используется приложение «Insomnia», пример которой отображен на скриншоте 1.10.



Скриншот 1.10. Внешний вид приложения «Insomnia»

Также важно отметить, то что каждое взаимодействие с сервером отображается в консоли, показывая вызванный запрос. Пример такого вызова виден на скриншоте 1.11, где происходит авторизация пользователя с почтой “test@gmail.com”.

```
Executing (default): SELECT "id", "email", "password", "fullName", "createdAt", "updatedAt" FROM "users" AS "user"
WHERE "user"."email" = 'test32@gmail.com';
```

Скриншот 1.11. Реакция севера на запрос с помощью приложения

Благодаря этому приложению появляется возможность проверки функций во время разработки приложения. На скриншоте 1.11 видно, пример эмуляции запроса на авторизацию пользователя, по URL <http://localhost:5000/api/user/login> приходит файл в формате json с данными введенными в форму, а именно логином и паролем. Сервер после получения данных запускает соответствующие роутеру функцию.

Листинг 1.1 Реализация логики на сервере при попытке авторизации пользователем

```
async login(req, res, next) {
  try {
    const { email, password } = req.body;
    const user = await User.findOne({ where: { email } });
    if (!user) return next(ApiError.badRequest("User not found"));

    let comparePassword = bcrypt.compareSync(password, user.password);
    if (!comparePassword)
      return next(ApiError.badRequest("Password or login is wrong"));

    const token = generateToken(user.id, user.email);
    console.log(req.body);
    return res.json({ token });
  } catch (error) {
    next(ApiError.badRequest(error.message));
  }
}
```

В примере кода, описанном в листинг 1.1, можно видеть, что функция получает на вход три аргумента, а именно req(request – запрос от клиента), res(response – ответ сервера на запрос клиентской части) и next, это функция, которая передается в качестве параметра в middleware функции Express. Она используется для передачи управления следующей middleware функции в цепочке обработки запросов. Если middleware функция не может обработать запрос и должна передать его следующей функции, она вызывает функцию next, передавая ей объект ошибки (если есть). Если объект middleware функции успешно обработали запрос, он будет отправлен клиенту.

Заходя в блок в try/catch, функция получает почту и пароль пользователя. После этого происходит поиск пользователя с этой почтой. Если пользователь не может быть найден, то вызывается функция next в параметр которой передается специально созданный класс ошибок с индикатором ошибки и текстом описывающий ошибку. После этого проверяется совпадения пароля введенного пользователем в форму и паролем из базы данных, Перед сравнением пароль пользователя шифруется.

Если же все предыдущие пункты выполнены успешно, то генерируется токен, содержащий id пользователя и его почту. Этот токен отправляется на клиентскую часть, говоря о том что авторизация пользователя прошла успешно. Если же на каком то из этапов произошла непредвиденная ошибка, то запускается блок catch, который так же вызывает ту же функцию next с передачей текста ошибки.

По схожей логике реализованы и другие функции, хотя они и имеют другую логику, они все имеют одни и те же аргументы, работу с базой данных и обработчик ошибок.

Так же поподробнее необходимо рассказать о том что такое middleware. Middleware (промежуточное программное обеспечение) - это программное обеспечение, которое находится между приложением и операционной системой или другими приложениями и позволяет им взаимодействовать друг с другом. Оно используется для решения различных задач, таких как обработка запросов, аутентификация пользователей, логирование действий пользователя и многое другое. Middleware может быть написано, как отдельная программа или интегрировано в код приложения. Обычно middleware представляет собой набор компонентов, которые работают в совокупности для выполнения определенной функциональности.

Листинг 1.2. Middleware авторизации

```
const ApiError = require("../errors/ApiError");
const jwt = require("jsonwebtoken");

module.exports = function (req, res, next) {
  if (req.method === "OPTIONS") next();

  try {
    const token = req.headers.authorization.split(" ")[1];
    if (!token) return res.status(401).json({ message: "unauthorized user" });

    const decode = jwt.verify(token, process.env.SECRET_KEY);
```



```

    req.user = decode;
    next();
  } catch (error) {
    console.log(error);
    res.status(401).json({ message: "unauthorized user" });
  }
};

```

Листинг 1.3 Middleware для ловли ошибок

```

const ApiError = require("../errors/ApiError");

module.exports = function (err, req, res, next) {
  if (err instanceof ApiError)
    return res.status(err.status).json({ message: err.message });

  return res.status(500).json({ message: "unannounced error!" });
};

```

В листинга 1.2 и 1.3 показаны middleware использованные в проекте. Это минимально необходимые middleware в каждом подобном приложении. Первый проверяет валидность токена на авторизацию и позволяет пользователю использовать возможности приложения при подтверждении подлинности токена. Второй же проверяет на наличие ошибки и возвращает ее статус и ее сообщение. Большинство ошибок описано в специальном классе `ApiError`. Данный класс содержит такие ошибки как 401, 403, 404 и 500, что покрывает ключевое большинство всех возможных проблем.

Так же на сервере существуют роутеры. Роутеры (или маршрутизаторы) в веб-разработке используются для определения путей (или маршрутов), которые приложение должно обрабатывать и как оно должно это делать.

Приложения, построенные на основе роутеров, могут принимать запросы от пользователей и выбирать нужный обработчик для каждого запроса, основываясь на определенных правилах. Это может быть полезно, например, при создании REST API, где каждый запрос имеет свой уникальный URL-адрес.

В общем, роутеры являются важным инструментом для разработки веб-приложений, поскольку они позволяют точно контролировать, как приложение обрабатывает запросы и отображает результаты в ответ на эти запросы.

Листинг 1.4. Роутеры на сервере для создания дат

```
const Router = require("express");
const userDateController = require("../controllers/userDateController");
const authMiddleware = require("../middleware/authMiddleware");
const router = new Router();

router.post("/create", authMiddleware, userDateController.create);
router.get("/get", authMiddleware, userDateController.getAll);
router.delete("/delete", authMiddleware, userDateController.delete);

module.exports = router;
```

Так в листинге 1.4 приведен пример роутера для работы с записями дат. Здесь создаются маршруты для разных методов работы данных с базой данных. Например, по URL <http://localhost/api/userDate/create> будет вызываться функция из класса userDateController create, а также будет вызываться middleware authMiddleware, для большей безопасности используется формат передачи данных post.

2.3 Клиентская часть проекта

Клиентская часть веб-приложения - это та часть приложения, которая выполняется на стороне клиента (то есть на стороне пользователя). Эта часть приложения обычно написана с использованием языков программирования, таких как HTML, CSS и JavaScript, и отвечает за отображение пользовательского интерфейса и взаимодействие с пользователем.

Клиентская часть веб-приложения может включать в себя следующие компоненты:

1. HTML - язык разметки, который определяет структуру и содержимое веб-страницы;
2. CSS - язык стилей, который определяет внешний вид веб-страницы, такой как цвет фона, шрифты, размеры и расположение элементов;
3. JavaScript - язык программирования, который позволяет создавать интерактивные элементы на странице, такие как кнопки, формы и анимации.

Клиентская часть веб-приложения может также взаимодействовать с серверной частью приложения, отправляя запросы на сервер и получая ответы. Например, клиентская часть может отправлять данные из формы на сервер для обработки или запрашивать информацию из базы данных на сервере.

Для разработки был выбран такой фреймворк как React. React - это библиотека JavaScript, которая используется для создания пользовательских интерфейсов. Она была разработана Facebook и является одним из самых популярных инструментов для разработки веб-приложений.

Основными преимуществами React являются:

1. Компонентный подход - приложение разбивается на небольшие компоненты, которые могут быть повторно использованы в разных частях приложения;
2. Виртуальный DOM - React использует виртуальный DOM, который позволяет минимизировать количество обращений к реальному DOM, что улучшает производительность приложения;
3. Однонаправленный поток данных - данные передаются от родительского компонента к дочернему, что упрощает отладку и управление состоянием приложения;
4. JSX - расширение языка JavaScript, которое позволяет описывать структуру пользовательского интерфейса в виде HTML-подобного кода.

React также имеет большое сообщество разработчиков, которые создают множество дополнительных библиотек и инструментов для упрощения разработки веб-приложений.

Так же для реализации некоторых компонентов использовалась такая библиотека как Bootstrap. Bootstrap - это бесплатный фреймворк для разработки веб-сайтов и приложений. Он был разработан Twitter и предоставляет набор готовых компонентов, которые можно использовать для создания адаптивных и кроссбраузерных интерфейсов.

Основными преимуществами Bootstrap являются:

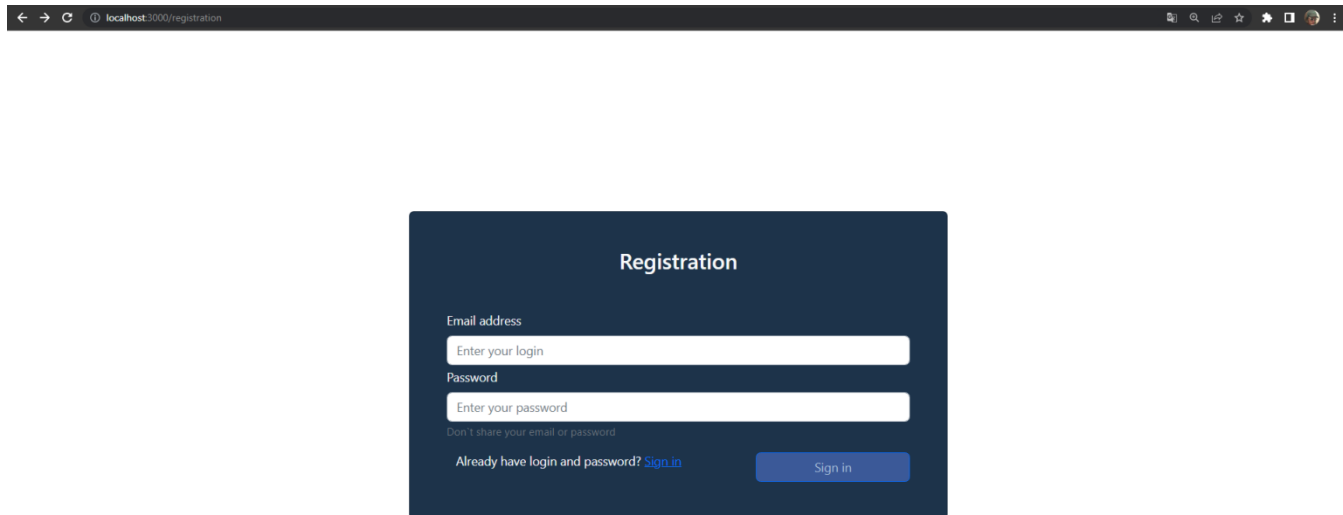
1. Адаптивность - Bootstrap предоставляет готовые компоненты, которые автоматически адаптируются к различным устройствам и экранам;
2. Кроссбраузерность - фреймворк обеспечивает совместимость с различными браузерами и платформами;
3. Готовые компоненты - Bootstrap содержит множество готовых компонентов, таких как кнопки, формы, таблицы, навигационные панели и т.д., которые можно использовать для быстрого создания интерфейса.

Bootstrap также имеет большое сообщество разработчиков, которые создают множество дополнительных компонентов и тем оформления для улучшения пользовательского интерфейса.

В приложении имеется 2 страницы. Первая это страница, выполняющая 2 разные роли: авторизации и регистрации. Они расположены по URL /login и /registration соответственно. Вторая это страница самого календаря по URL /calendar.

2.3.1 Страница регистрации и авторизации

Рассмотрим страницу авторизации и регистрации. В соответствии со скриншотом 1.12, пользователь может либо авторизоваться в данном окне либо, нажав на ссылку, перейти к регистрации нового аккаунта.



Скриншот 1.12. Внешний вид страницы с окном авторизации.

Листинг 2.1 Реализация логики окна авторизации/регистрации

```
import React, { useState } from "react";
import LoginForm from "../components/LoginForm";
import { login, register } from "../api/api";
import { Navigate, useLocation } from "react-router-dom";
import { CALENDAR_ROUTE } from "../utils/const";

const Auth = () => {
  const [loggedIn, setLoggedIn] = useState(false);
  const [registerIn, setRegisterIn] = useState(false);

  const location = useLocation();

  async function handleLogin(email, password) {
    if (location.pathname === "/login") {
      try {
        const response = await login(email, password);
```

```

    const data = await response.json();
    localStorage.setItem("token", data.token);
    console.log(data.token);
    if (response.ok) {
      setLoggedIn(true);
    } else alert(data.message);
  } catch (error) {
    console.error(error);
    alert("error response");
  }
} else if (location.pathname === "/registration") {
  try {
    const response = await register(email, password);
    const data = await response.json();
    localStorage.setItem("token", data.token);
    if (response.ok) {
      setRegisterIn(true);
    } else alert(data.message);
  } catch (error) {
    console.error(error);
    alert("error response");
  }
}
}

if (registerIn || loggedIn) return <Navigate to={CALENDAR_ROUTE} />;

return <LoginForm onSubmit={handleLogin} />;
};

export default Auth;

```

В участке кода, описанном в листинг 2.1, показана реализации логики работы страницы Auth, отвечающая за регистрацию и авторизацию.

Первоначально импортируются необходимые зависимости: `useState` для работы с состоянием, `LoginForm` для отображения формы авторизации, а также функции `login` и `register` из модуля `api` для отправки запросов на сервер. Импортируется также `Navigate` для перенаправления пользователя и константы `CALENDAR_ROUTE` и `LOGIN_ROUTE` из модуля `utils/const`.

Далее создается функциональный компонент `Auth`, который использует хук `useState` для отслеживания состояния `loggedIn` (авторизован ли

пользователь) и registerIn (зарегистрирован ли пользователь). Также используется хук useLocation для получения текущего пути (для определения на какой страницы сейчас находится пользователь).

Далее определяется функция handleLogin, которая принимает email и password и отправляет соответствующий запрос на сервер в зависимости от текущего пути. Если путь равен «/login», то отправляется запрос на авторизацию, если «/registration» – на регистрацию. В случае успешного ответа от сервера, клиент получает токен и сохраняет его в localStorage (веб-хранилище, которое хранит значение в виде пары ключ/значение и сохраняет это значение даже после перезапуска веб-браузера) и устанавливается соответствующее состояние (loggedIn или registerIn). В случае ошибки выводится сообщение об ошибке, как в консоль, так и пользователю через alert.

Затем проверяется, авторизован ли пользователь или зарегистрирован, и если да, то происходит перенаправление на страницу календаря.

В конце же, компонент отображает форму LoginForm, передавая в нее функцию handleLogin в качестве пропса onSubmit.

Так же стоит уделить внимание и для функций, которые напрямую общаются с сервером.

Листинг 2.2 Реализация логики окна авторизации/регистрации

```
Const BASE_URL = "http://localhost:5000";

async function login(email, password) {
  const response = await fetch(`${BASE_URL}/api/user/login`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({ email, password }),
  });
  return response;
}

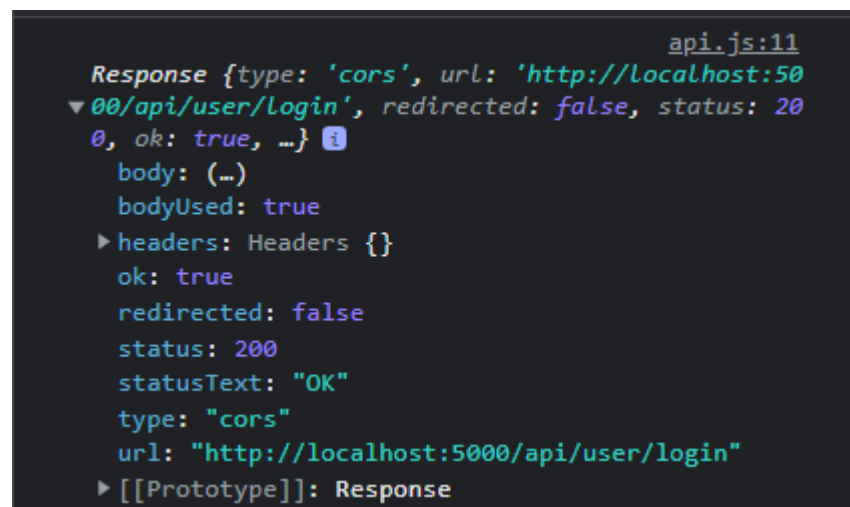
async function register(email, password) {
  const response = await fetch(`${BASE_URL}/api/user/registration`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
  },
```

```

    body: JSON.stringify({ email, password }),
  });
  return response;
}

```

Данные функции, описанные в листинг 2.2, работают с сервером, а значит, обязаны быть реализованы асинхронно и вследствие этого все функции является асинхронными (async), а основная функция запроса на сервер использует ключевое слово await. Для передачи данных через HTTP протоколы используется базовая функция языка JavaScript – fetch, она принимает в качестве аргумента URL сервера, метод запроса, заголовки, а также тело запроса. Например, функция login обращается к URL: “http://localhost:5000/api/user/login”, говорит о том, что метод передачи данных будет использоваться POST, будет передан заголовок Content-type: application/json, который говорит о том, что будет использоваться формат данных json (JavaScript Object Notation), а так же передает тело запроса с необходимыми параметрами. Функции возвращают ответ сервера, в котором имеется информация о том с каким статусом завершен запрос, и какие данные были получены, пример приведен на скриншоте 1.13.



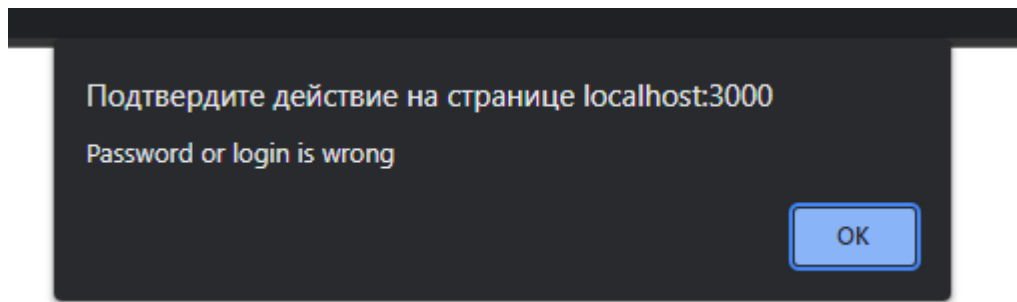
```

api.js:11
Response {type: 'cors', url: 'http://localhost:50
▼ 00/api/user/login', redirected: false, status: 20
0, ok: true, ...} ⓘ
  body: (...)
  bodyUsed: true
  ▶ headers: Headers {}
  ok: true
  redirected: false
  status: 200
  statusText: "OK"
  type: "cors"
  url: "http://localhost:5000/api/user/login"
  ▶ [[Prototype]]: Response

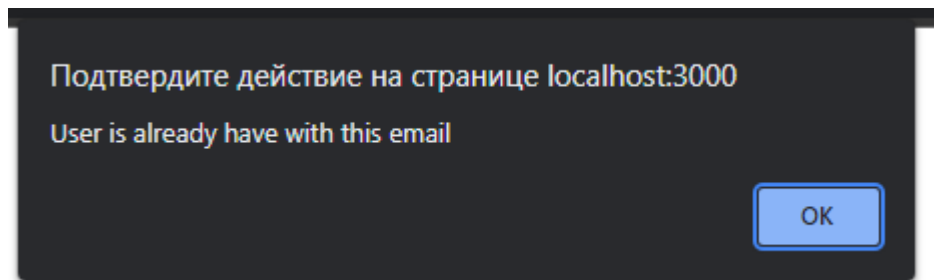
```

Скриншот 1.13. Пример формата ответа сервера

И в соответствии с этими данными на клиенте происходит действие: если статус ответа является в положении true, то пользователя перекидывает на страницу календаря, если же возвращается false, то показывается окошко с ошибкой при том не указывая на конкретику (скриншот 14.). Так же при регистрации пользователя по такому же логину вызывает ошибку, указывая на то, что такой пользователь уже есть в системе (скриншот 15).



Скриншот 1.14. Пример окна при неверных данных авторизации



Скриншот 1.15. Пример окна при использовании того же логина при регистрации, что уже существует

Так же поподробнее рассмотрим компонент окна регистрации и авторизации.

Листинг 2.3 Реализация окна авторизации и регистрации

```
import React, { useState } from "react";
import { Button, Container, Form, FormGroup, Row } from "react-bootstrap";
import Card from "react-bootstrap/esm/Card";
import { NavLink, useLocation } from "react-router-dom";
import colors from "../utils/colors";
import { LOGIN_ROUTE, REGISTRATION_ROUTE } from "../utils/const";

const LoginForm = (props) => {
  const [isHovered, setIsHovered] = useState(false);
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");

  const location = useLocation();

  const isLogin = location.pathname === LOGIN_ROUTE;

  const styles = {
    container: { height: window.innerHeight - 100 },
```

```

card: {
  width: 700,
  boxShadow: "5px black",
  background: colors.BgColorDark,
},
btnStyle: {
  background: isHovered ? colors.Success : colors.BtnColorDark,
  color: colors.TextColorDark,
},
};

function handleSubmit(event) {
  event.preventDefault();
  props.onSubmit(email, password);
}

return (
  <div>
    <Container
      className="d-flex justify-content-center align-items-center p-4"
      style={styles.container}
    >
      <Card style={styles.card} className="p-5 link-light">
        <h3 className="m-auto">
          {isLoggedIn ? "Authorization" : "Registration"}
        </h3>
        <Form onSubmit={handleSubmit}>
          <FormGroup>
            <Form.Label className="mt-5">Email address</Form.Label>
            <Form.Control
              placeholder="Enter your login"
              type="email"
              value={email}
              onChange={(e) => setEmail(e.target.value)}
            />
          </FormGroup>
          <FormGroup>
            <Form.Label className="mt-1">Password</Form.Label>
            <Form.Control
              placeholder="Enter your password"
              type="password"
              value={password}
              onChange={(e) => setPassword(e.target.value)}
            />
          </FormGroup>
        </Form>
      </Card>
    </div>
  );

```

```

    />
    <Form.Text className="text-muted">
      Don`t share your email or password
    </Form.Text>
  </FormGroup>

  <Row className="m-auto justify-content-between">
    {isLoggedIn ? (
      <div className="align-self-center w-50">
        Haven`t account?{" "}
        <NavLink to={REGISTRATION_ROUTE}>Create an
account</NavLink>
      </div>
    ) : (
      <div className="align-self-center w-75">
        Already have login and password?
        <NavLink to={LOGIN_ROUTE}>Sign in</NavLink>
      </div>
    )}

    <Button
      className="mt-3 w-25"
      type="submit"
      style={styles.btnStyle}
      onMouseEnter={() => setIsHovered(true)}
      onMouseLeave={() => setIsHovered(false)}
    >
      {isLoggedIn ? "Log in" : "Sign in"}
    </Button>
  </Row>
</Form>
</Card>
</Container>
</div>
);
};

export default LoginForm;

```

В данном коде, описанном в листинг 2.3, определяются необходимые зависимости для реализации окна, дальше объявляются хуки для записи почты и пароль, определения URL, а так же хук с переменной `isHovered` для стилизования будущей кнопки. Дальше по коду реализован CSS код в виде

объектов, что бы в дальнейшем можно было обратиться к этим объектам в необходимых тегах. Такой способ более удобен при малом количестве стилей без добавления лишних css файлов, а также позволяет использовать переменные имеющиеся внутри проекта, в отличие от обычного css. После же реализована функция, которая вызывается во время отправки формы (onSubmit). Внутри функции вызовется метод объекта event preventDefault(), который позволяет отменить стандартное поведение, а после в качестве пропсов передаем полученные из формы логин и пароль.

Листинг 2.4 Пример кода в котором с помощью тернарного оператора определяются текст вывода

```
<h3 className="m-auto">  
  {isLoggedIn ? "Authorization" : "Registration"}  
</h3>
```

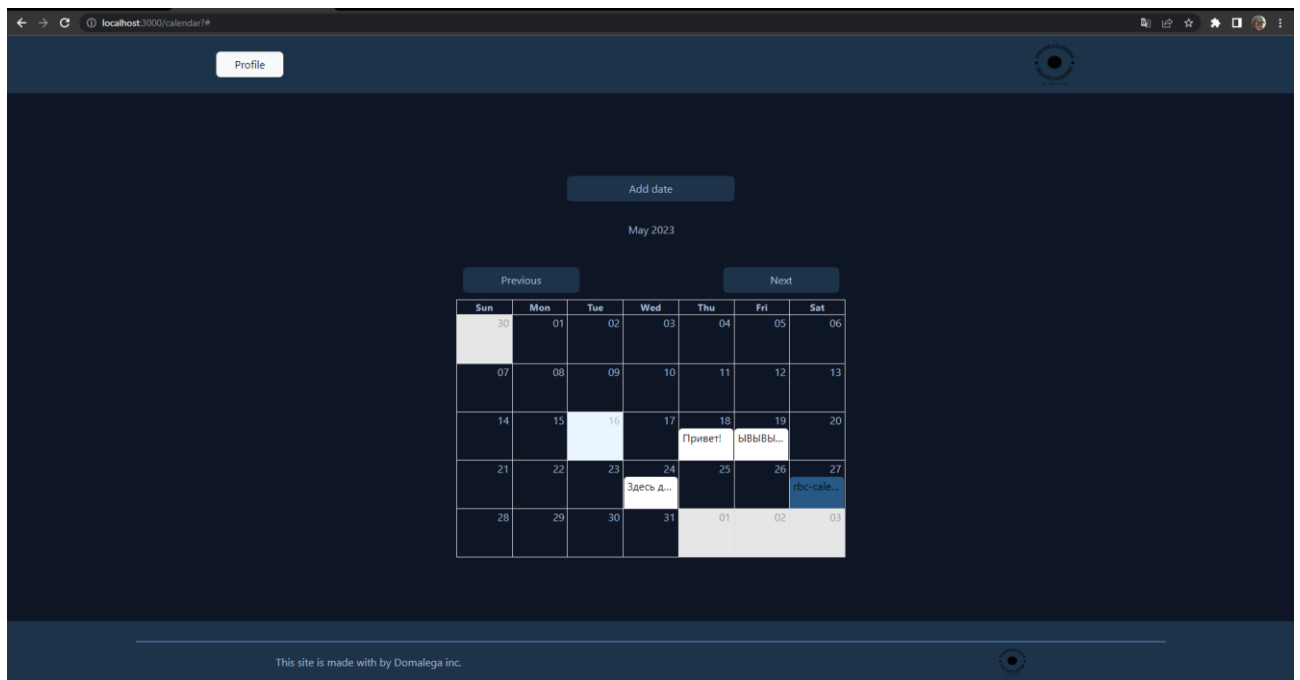
В конце возвращаем форму. В тегах используется информация в зависимости от того какой URL сейчас имеется, пример такого поведения приведен в листинге 2.4.

Так же для дополнительных стилей используются bootstrap классы.

2.3.2 Основная страница

После авторизации пользователь попадает в основной интерфейс веб-приложения, интерфейс которого показан на скриншоте 1.15.

Данное окно состоит из нескольких частей: Навигационная панель, расположенная в основном в семантическом теге <header>, основная часть, расположенная в семантическом теге <main> и подвал сайта, расположенный в семантическом теге <footer>.



Скриншот 1.15. Внешний вид страницы календаря

Листинг 2.5 Код основного окна приложения

```
import React, { useState } from "react";
import NavBar from "../components/NavBar";
import MainCalendar from "../components/MainCalendar";
import Footer from "../components/Footer";
import "../styles/Calendar.css";
import colors from "../utils/colors";

import ModalWindow from "../components/ModalChoseDate.jsx";

const Calendar = () => {
  const [modalIsOpen, setIsOpen] = useState(false);

  const style = {
```

```

    btnAddStyle: {
      background: colors.BgColorDark,
      color: colors.TextColorDark,
      margin: 10,
      width: 250,
    },
    mainWrapperStyle: { background: colors.WrapperColorDark },
    mainCalendarStyle: { color: colors.TextColorDark },
  };

  return (
    <div className="wrapper">
      <header className="header">
        <NavBar />
      </header>

      <main className="main" style={style.mainWrapperStyle}>
        <button
          onClick={() => setIsOpen(true)}
          className="btn containerBar__btn-add"
          style={style.btnAddStyle}
        >
          Add date
        </button>

        <div
          className={`child modal-${modalIsOpen}`}
          style={style.mainCalendarStyle}
        >
          <MainCalendar />
        </div>

        <ModalWindow modalIsOpen={modalIsOpen} setIsOpen={setIsOpen} />
      </main>

      <footer className="footer">
        <Footer />
      </footer>
    </div>
  );
};

export default Calendar;

```

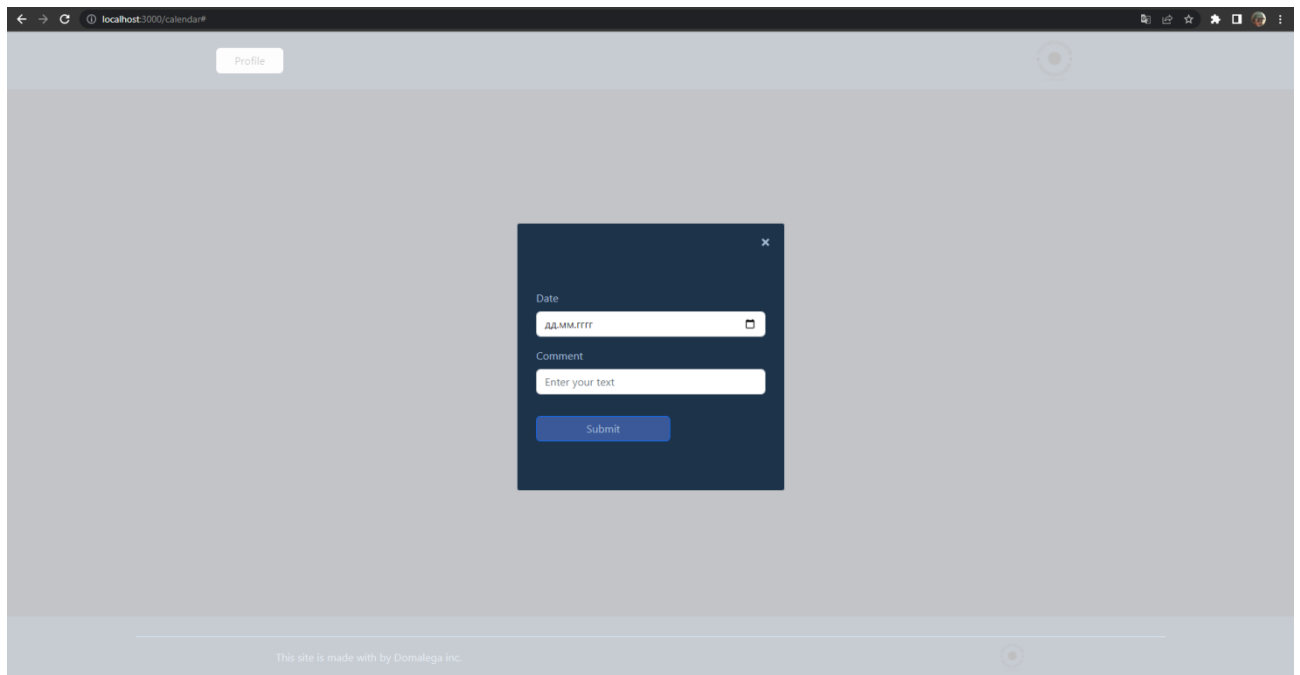
В коде, описанном в листинге 2.5, так же как и в окне авторизации в первую очередь подключаются зависимости для работоспособности дальнейшего кода. Затем объявляется хук с переменной `modalIsOpen` и функция `openModal` для работы с модальным окном при добавлении дат в базу данных, дальше идет объявление стилей, в которых используются цвета из массива находящегося внутри проекта. Дальше возвращается то, что будет отрендерено на экране.

В `css` файле же реализованы стили для корректного отображения семантических тегов, правильного расположения самого календаря, а так же для различного поведения компонента `<MainCalendar>` в зависимости от того открыто ли модальное окно или нет.

Также стоит отметить, что в данном окне также много и других компонентов с различной логикой, рассмотрим каждый поподробнее.

2.3.3 Модальное окно

При нажатии на кнопку «Add date» пользователь попадает на модальное окно, показанное на скриншоте 1.16. В данном окне пользователь может выбрать необходимую дату и ввести необходимый текст и записать данные в свой календарь.



Скриншот 1.16. Модальное окно

Листинг 2.6 Реализация модального окна

```
import React, { useState } from "react";
import { Form, FormControl, Button } from "react-bootstrap";
import Modal from "react-modal";
import colors from "../utils/colors";
import { createDate } from "../api/api";
import { FaTimes } from "react-icons/fa";

const ModalWindow = (props) => {
  const [date, setDate] = useState("");
  const [comment, setComment] = useState("");
  const [isHoverBtnClose, setIsHoverBtnClose] = useState(false);
  const [isHoverBtnSubmit, setIsHoverBtnSubmit] = useState(false);

  const styles = {
    content: {
      top: "50%",
```



```

left: "50%",
right: "auto",
bottom: "auto",
marginRight: "-50%",
transform: "translate(-50%, -50%)",
background: colors.BgColorDark,
color: colors.TextColorDark,
height: 400,
width: 400,
display: "grid",
gridTemplateColumns: "repeat(5, 1fr)",
gridTemplateRows: "repeat(5, 1fr)",
gridColumnGap: 0,
},
closeButton: {
  color: colors.TextColorDark,
  gridArea: "1 / 5 / 2 / 6",
  justifySelf: "end",
  cursor: "pointer",
  transform: isHoverBtnClose ? "scale(1.2)" : "scale(1)",
},
form: {
  gridArea: "2 / 1 / 6 / 6",
},
btnStyleSubmit: {
  background: isHoverBtnSubmit ? colors.Success : colors.BtnColorDark,
  color: colors.TextColorDark,
  width: 200,
},
};

async function handleSubmitDate(date, comment) {
  try {
    const token = localStorage.getItem("token");
    const response = await createDate(date, comment, token);
    const data = await response.json();
    if (response.ok && data.message === "ok") {
      props.setIsOpen(false);
      window.location.reload();
    } else {
      alert("Date is already created");
    }
  } catch (error) {
    alert(error);
  }
}

```

```

    }
  }

  return (
    <div>
      <Modal
        isOpen={props.modalIsOpen}
        onRequestClose={() => props.setIsOpen(false)}
        style={styles}
        contentLabel="Send data"
      >
        <FaTimes
          onClick={() => props.setIsOpen(false)}
          onMouseEnter={() => setIsHoverBtnClose(true)}
          onMouseLeave={() => setIsHoverBtnClose(false)}
          style={styles.closeButton}
        />
        <div style={styles.form}>
          <Form onSubmit={() => handleSubmitDate(date, comment)}>
            <Form.Group className="p-2">
              <Form.Label>Date</Form.Label>
              <Form.Control
                type="date"
                placeholder="Enter date"
                value={date}
                onChange={(e) => setDate(e.target.value)}
              />
            </Form.Group>

            <Form.Group className="p-2">
              <Form.Label>Comment</Form.Label>
              <Form.Control
                type="text"
                placeholder="Enter your text"
                value={comment}
                onChange={(e) => setComment(e.target.value)}
              />
            </Form.Group>

            <Button
              className="m-2 mt-4"
              style={styles.btnStyleSubmit}
              onMouseEnter={() => setIsHoverBtnSubmit(true)}
              onMouseLeave={() => setIsHoverBtnSubmit(false)}
            />
          </Form>
        </div>
      </Modal>
    </div>
  )
}

```

```

        type="submit"
      >
        Submit
      </Button>
    </Form>
  </div>
</Modal>
</div>
);
};

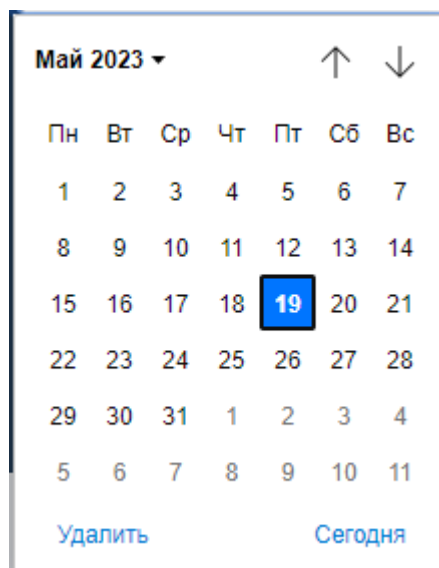
export default ModalWindow;

```

Данный компонент, описанный в листинге 2.6, является частью более глобального объекта, в который передаются параметры (пропсы), а именно булево значение `modalIsOpen` и функцию для изменения этого значения `SetIsOpen()`. Таким образом, используя, например, переменную `props.modalIsOpen` можно узнать, открыто ли модальное окно.

Так же стоит отметить, что использовались дополнительные стили для центрирования модального окна и корректно расположения элементов внутри самого окна с помощью `grid` сетки. Для реализации модального окна использовалась дополнительная `react` библиотека. `React-Modal` - это библиотека компонентов для создания модальных окон в `React` приложениях. Она предоставляет простой и удобный интерфейс для создания и настройки модальных окон, которые могут использоваться для отображения различных форм, сообщений, изображений и других элементов. Эта библиотека легко интегрируется с другими библиотеками и фреймворками `React`, такими как `Redux`, `MobX`, `React`, `Router` и другие. Она также поддерживает множество настроек и опций для управления поведением и внешним видом модальных окон. Этот инструмент прост в использовании, гибкий и имеет большие возможности настройки. Для верстки также используются элементы из `Bootstrap-react`, `bootstrap` классы, а так же стандартные `html` теги, с использованием `bootstrap` классов.

В данном модальном окне пользователь имеет два вида поля. Первое поле даты, имеющие тип данных `date`, что позволяет использовать удобный встроенный в язык `JavaScript` датапикер (скриншот 1.17). Второе поле - это комментарий к дате, оно ограничено 255 символами в базе данных, при нарушении этого условия, данные не будут отправлены на сервер, а пользователь получит уведомление об ошибке длинны ввода.

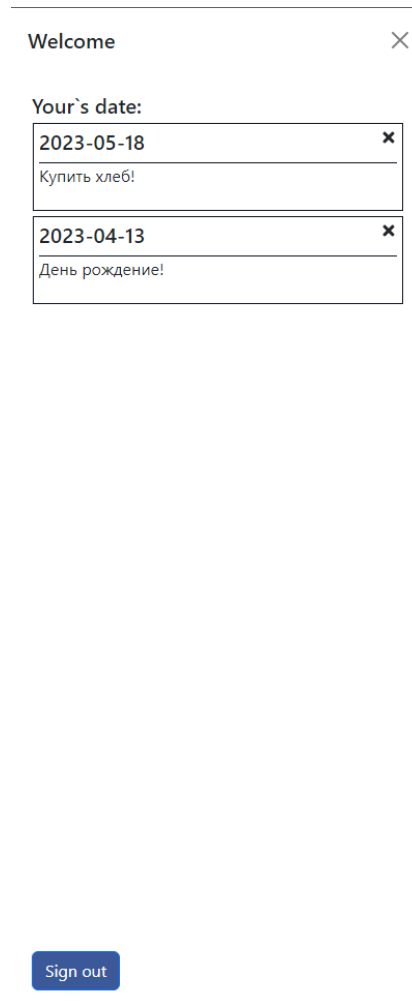


Скриншот 1.17. Датапиркер

Так же стоит отметить, что к дате подключена функция, которая будет обрабатывать информацию, когда пользователь нажмет кнопку подтверждения. В самой функции программа получает токен из внутреннего хранилища (localStorage) и отправляет запрос к функции из папки api createDate в которую передаются выбранные дату, комментарий и токен пользователя. Если же все проходит без ошибок, то модальное окно закрывается, а страница автоматически перезагружается что бы после загрузки страницы, данные автоматически подгрузились с базы данных и отобразились на экране пользователя.

2.3.4 Боковая панель

В левой части навигационной панели имеется кнопка «Profile», при нажатии на эту кнопку откроется боковая панель приложения. В данном окне выводятся все хранящиеся в базе данных даты в виде маленьких отдельных блоков, а так же имеется кнопка «Sign out», которая удаляет токен пользователя после чего пользователя выкидывает с основной страницы веб-приложения. Пример интерфейса, отображен на скриншоте 1.18.



Скриншот 1.18. Боковая панель

Листинг 2.7 Реализация боковой панели

```
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
import Button from "react-bootstrap/Button";
import Offcanvas from "react-bootstrap/Offcanvas";
import { getAllDates } from "../api/api";
import DateContainer from "../DateContainer";
import colors from "../utils/colors";
```

```

const SideBtn = () => {
  const [show, setShow] = useState(false);
  const [dates, setGetDates] = useState();

  const navigate = useNavigate();

  const handleShow = async () => {
    try {
      const token = localStorage.getItem("token");
      const response = await getAllDates(token);
      const data = await response.json();
      setGetDates(data);
      setShow(true);
    } catch (error) {
      console.log(error);
    }
  };

  const handleOut = () => {
    localStorage.removeItem("token");
    navigate("/login");
  };

  const styles = {
    btnOpen: { width: 100 },
    btnOut: {
      position: "absolute",
      bottom: 0,
      left: 0,
      margin: 20,
      backgroundColor: colors.BtnColorDark,
    },
  };

  return (
    <div>
      <Button variant="light" onClick={handleShow} style={styles.btnOpen}>
        Profile
      </Button>

      <Offcanvas show={show} onHide={() => setShow(false)}>
        <Offcanvas.Header closeButton>
          <Offcanvas.Title>Welcome</Offcanvas.Title>

```

```

    </Offcanvas.Header>
    <Offcanvas.Body>
      <p className="m-1 h5">Your`s date:</p>
      {dates &&
        dates.map((data) => <DateContainer key={data.id} date={data} />)}
      <Button style={styles.btnOut} onClick={handleOut}>
        Sign out
      </Button>
    </Offcanvas.Body>
  </Offcanvas>
</div>
);
};

export default SideBtn;

```

В коде, описанном в листинге 2.7, мы подключаем необходимые зависимости и стили для некоторых объектов разметки. Далее возвращаем объект, который будет рендериться. Для разметки используются Bootstrap-react теги, стандартные html, а так же компонент, реализованный в ручную <DateContainer>. Это объекты, который получает в качестве аргумента при создании id, так как каждый повторяющийся объект обязан иметь свой собственный и уникальный id, по правилам react, а так же объект дата, хранящая в себе информацию о дате и комментарии. Так же при нажатии на кнопку с именем Profile, вызывается функция handleShow(), которая делает запрос к базе данных для получения всех данных о датах данного пользователя и записать их в массив dates.

3 Описание библиотек

3.1 Общие библиотеки

Npm(Node Packet Manager)

Это менеджер пакетов для языка программирования JavaScript, который позволяет устанавливать, обновлять и удалять зависимости в проектах. Он предоставляет доступ к множеству библиотек и модулей, которые могут быть использованы для разработки веб-приложений, серверных приложений, игр и других проектов на JavaScript. Npm также позволяет создавать и публиковать свои собственные пакеты, которые могут быть использованы другими разработчиками. Кроме того, npm предоставляет инструменты для управления версиями пакетов, поиска зависимостей, установки глобальных модулей и т.д. Он является неотъемлемой частью экосистемы Node.js и широко используется в различных проектах на JavaScript.

Таким образом, можно сказать, что npm выполняет следующие функции:

- Позволяет устанавливать дополнительные зависимости к своему проекту, реализованные другими пользователями.
- А так же это инструмент командой строки, позволяющий расширить ее функционал. Например, запуск сервера происходит командой - `npm run start:dev`.

Так же npm создает 2 файла внутри проекта : `package-lock.json` и `package.json`, пример которого приведен на скриншоте 2.1. В них находятся метаданные: название проекта, версия, автор, лицензия и зависимости, подключенные к проекту с указанием их версии. Так же здесь описаны скрипты, которые можно запускать с консоли. В другом файле так же более подробно описываются все зависимости с описанием репозитория откуда берутся изначальные данные и с их зависимостями.

Таким образом, можно сказать, что npm это основополагающий инструмент разработки любого приложения реализованного на платформе node.js.


```
1 {
2   "name": "server",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "start:dev": "nodemon index.js"
8   },
9   "author": "",
10  "license": "ISC",
11  "dependencies": {
12    "bcrypt": "^5.1.0",
13    "cors": "^2.8.5",
14    "dotenv": "^16.0.3",
15    "express": "^4.18.2",
16    "express-validator": "^6.14.3",
17    "jsonwebtoken": "^9.0.0",
18    "pg": "^8.8.0",
19    "pg-hstore": "^2.3.4",
20    "sequelize": "^6.28.0"
21  },
22  "devDependencies": {
23    "nodemon": "^2.0.20"
24  }
25 }
26
```

Скриншот 2.1. Данные файла package.json

3.2 Библиотеки серверной части

Express.js

Express.js, или просто Express, фреймворк web-приложений для Node.js, реализованный как свободное и открытое программное обеспечение под лицензией MIT. Он спроектирован для создания веб-приложений и API. Де-факто является стандартным каркасом для Node.js. Автор фреймворка, TJ Holowaychuk, описывает его как созданный на основе написанного на языке Ruby каркаса Sinatra, подразумевая, что он минималистичен и включает большое число подключаемых плагинов. Express может являться backend'ом для большого количества программных стеков, например для такого как – стек MEAN (аббревиатура от MongoDB, Express.js, Angular.js, Node.js).

Pg

PG в Node.js - это модуль, который предоставляет возможность подключения к базе данных PostgreSQL из приложений на Node.js. Он предоставляет удобный интерфейс для выполнения запросов к базе данных и получения результатов. PG использует протоколы работы с базой данных PostgreSQL, что обеспечивает быстрое и эффективное взаимодействие с базой данных. Модуль PG также поддерживает многопоточность и асинхронную работу, что позволяет выполнять запросы к базе данных параллельно и ускорить работу приложения. PG поддерживает множество функций для работы с данными, включая транзакции, индексы, хранимые процедуры и многое другое. Он также имеет хорошую документацию и широкую поддержку сообщества, что делает его одним из наиболее популярных модулей для работы с базой данных PostgreSQL в Node.js.

Sequelize

Sequelize - это ORM-библиотека (Object-Relational Mapping — объектно-реляционное отображение или преобразование) для приложений на Node.js, которая осуществляет сопоставление таблиц в бд и отношений между ними с классами. При использовании Sequelize мы можем не писать SQL-запросы, а работать с данными как с обычными объектами. Причем Sequelize может работать с рядом СУБД - MySQL, Postgres, MariaDB, SQLite, MS SQL Server.

Для создания базы создаются модели будущих таблиц с их типом данных, а так же атрибутами по типу уникальности, первичного ключа, возможности быть поля со значением NULL и другие.

Листинг 1.5 Файл с моделями пользователей и дат

```

const sequelize = require("../bd");
const { DataTypes, Model } = require("sequelize");

const User = sequelize.define("user", {
  id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    autoIncrement: true,
  },

  email: {
    type: DataTypes.STRING,
    unique: true,
    allowNull: false,
  },

  password: {
    type: DataTypes.STRING,
    allowNull: false,
  },

  fullName: {
    type: DataTypes.STRING,
    allowNull: true,
  },
});

const UserData = sequelize.define("userData", {
  id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    autoIncrement: true,
  },

  date: {
    type: DataTypes.DATEONLY,
    allowNull: false,
  },

  comment: {
    type: DataTypes.STRING,
    allowNull: true,
  },
});

```

```

userId: {
  type: DataTypes.INTEGER,
  allowNull: false,
},
});

User.hasMany(UserDate);
UserDate.belongsTo(User);

module.exports = {
  User,
  UserDate,
};

```

В примере, приведенном в листинг 1.5, видно, что с помощью библиотеки `sequelize` объявляются модели, по которой в дальнейшем будет, проходит создание всех объектов такого типа. Например, таблица “userDate” обладает следующими полями: `id`, `date`, `comment` и `userId` для связи таблицы дат и пользователя. Так же существует множество различных атрибутов для описания полей, например:

- `type` – этот атрибут указывает на тип данных, хранящихся в данном поле. Подключить больше типов можно импортирую модуль `DataTypes` из библиотеки `sequelize`;
- `allowNull` – этот атрибут указывает на то может ли быть поле пустым или обязано иметь данные;
- `primaryKey` – показывает, что данное поле является первичным ключом, что гарантирует, то что значение является уникальным и не пустым;
- `autoIncrement` – означает автоматическое увеличение числового значения первичного ключа при добавлении новой записи в таблицу, что обеспечивает уникальность каждого значения первичного ключа;
- `foreignKey` – вторичный ключ для связи таблиц с использованием первичного ключа. Неявным образом `sequelize` автоматически создает вторичный ключ из-за указания на отношения таблиц “`hasMany`” и “`belongsTo`”;
- и др.

В конце файла объявляется, то какие связи друг с другом будут иметь модели. Так можно понять, что каждый пользователь приложения может иметь много различных дат.

Листинг 1.6 Подключения к базе данных с помощью ORM

```
const { Sequelize } = require("sequelize");

module.exports = new Sequelize(
  process.env.DB_NAME,
  process.env.DB_USER,
  process.env.DB_PASSWORD,
  {
    dialect: "postgres",
    host: process.env.DB_HOST,
    port: process.env.DB_PORT,
  }
);
```

Так же в приложении реализован отдельный файл с подключением к базе данных, показанный в Листинге 1.3. В нем указывается имя пользователя, имя базы данных, пароль базы данных, а так же вид базы данных (в данном случае PostgreSQL), порт и хост.

В дальнейшем при запуске при запуске сервера будет происходить авторизация в базе данных и создание моделей, если они не существуют.

CORS

CORS (Cross-Origin Resource Sharing) - это стандарт, позволяющий предоставлять веб-страницам доступ к объектам сторонних интернет-ресурсов. Сторонним считается любой интернет-ресурс, который отличается от запрашиваемого протоколом, доменом или портом.

Структура Cross-Origin Resource Sharing

Методы CORS предназначены для управления доступом к дескрипторам (тегам) на веб-страницах в сети. Управляемые типы доступа подразделяются на три основных категории по работе с информацией сторонних ресурсов:

1. Доступ на запись — это доступ к ссылкам, заполнению веб-форм и переадресации на сторонние веб-страницы, т.е. на передачу информации в сторонний источник (веб-ресурс).
2. Доступ на вставку относится к категории доступа на считывание информации из стороннего источника. К этому типу принадлежат вставки в код дескрипторов audio, video, img, embed, object, link, script, iframe и другие элементы оформления веб-страниц. Структура подобных дескрипторов подразумевает самостоятельную инициацию перекрестных

(cross-origin) запросов из сторонних источников. Все дескрипторы этой категории представляют низкий уровень угрозы безопасности, поэтому разрешены в веб-браузере по умолчанию.

3. Доступ на считывание — это дескрипторы, загружаемые с использованием фоновых методов вызова, таких как `fetch()`, технологии обмена данными Ajax и пр. Поскольку подобные дескрипторы могут содержать в теле любые участки кода (в том числе вредоносного), они запрещены в веб-браузерах по умолчанию.

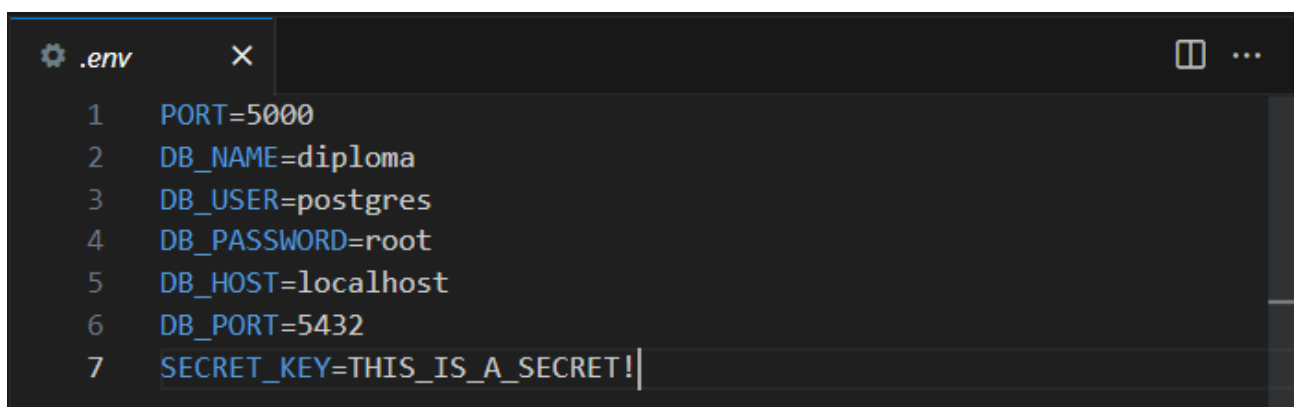
Dotenv.js

Библиотека, позволяющая создавать и использовать переменные окружения в приложении, написанном на JavaScript.

Переменные окружения или переменные среды (environment variables) — это некие глобальные значения, расположенные на уровне операционной системы, доступные программам, например настройки системы.

Самой известной переменной, можно считать `PATH`. Операционная система использует значение этой переменной для того, чтобы найти нужные исполняемые файлы в командной строке или окне терминала. Например, когда мы выполняем в терминале команду `node`, `npm` или другую, именно переменная `PATH` подсказывает где искать исполняемые файлы.

Файл, хранящий переменные окружения принято называть `.env` и хранит в себе пары в виде ключ/значение.



```
.env
1  PORT=5000
2  DB_NAME=diploma
3  DB_USER=postgres
4  DB_PASSWORD=root
5  DB_HOST=localhost
6  DB_PORT=5432
7  SECRET_KEY=THIS_IS_A_SECRET!
```

Скриншот 2.2. Данные в файл `.env`.

Из-за того, что переменные глобальные их можно использовать в любой части приложения, как например, при подключении базы данных использовались данные именно из файла `.env`. Из-за механизма работы переменных окружения, такие данные будут идеальным местом хранения таких данных, которые не должны отображаться, к примеру, на `github`. И благодаря глобальному объекту **process** (доступный из любого места программы), хранящему информацию о текущем процессе можно получить доступ к этим переменным внутри свойства `env`.

Nodemon

В Node.js для вступления изменений в силу необходимо перезапустить процесс. Это добавляет в рабочий процесс дополнительный шаг, необходимый для внесения изменений. Вы можете устранить этот дополнительный шаг, используя `nodemon` для автоматического перезапуска процесса.

`nodemon` — это утилита командной строки. Она заключает в оболочку ваше приложение Node, наблюдает за файловой системой и автоматически перезапускает процесс. Для корректной работоспособности утилиты необходимо так же предварительно ее настроить, указав ей точку входа.

Jsonwebtoken (JWT)

Jsonwebtoken — это открытый стандарт для создания токенов доступа, основанный на формате JSON. Как правило, используется для передачи данных для аутентификации в клиент-серверных приложениях. Токены создаются сервером, подписываются секретным ключом и передаются клиенту, который в дальнейшем использует данный токен для подтверждения подлинности аккаунта.

Структура JWT состоит из трех частей: заголовок `header`, полезные данные `payload` и подпись `signature`.

Header - Заголовок (`header`) в JSON Web Token (JWT) - это первая часть токена и содержит информацию о типе токена и используемом алгоритме подписи.

Обычно заголовок представляет собой JSON-объект, который содержит два поля: `"alg"` и `"typ"`. Поле `"alg"` указывает на алгоритм подписи, используемый для создания подписи токена. Значение этого поля может быть, например `"HS256"`, что означает, что используется алгоритм HMAC-SHA256. Поле `"typ"` определяет тип токена, который обычно устанавливается в значение `"JWT"`. Если необходимо передать дополнительные данные в заголовке, можно добавить новые поля в JSON-объект заголовка.

Payload - Полезная нагрузка (payload) в JSON Web Token (JWT) - это вторая часть токена и содержит информацию о пользователе или объекте, для которого был создан токен.

Обычно полезная нагрузка представляет собой JSON-объект, который может содержать любую информацию, которую отправитель хочет закодировать в JWT. Например, это может быть идентификатор пользователя, его имя, email, права доступа или другие метаданные. Важно понимать, что эти данные не должны быть конфиденциальными, так как они могут быть декодированы без ключа.

Также в полезную нагрузку можно добавить зарезервированные поля, такие как "iss" (issuer), "exp" (expiration time), "sub" (subject), "aud" (audience) и другие. Эти поля помогают определить, кто создал токен, когда он истекает, кому адресован и т.д.

Signature - Подпись (signature) в JSON Web Token (JWT) - это третья часть токена, которая гарантирует целостность и подлинность токена. Подпись создается путем применения криптографического алгоритма к заголовку, полезной нагрузке и секретному ключу.

Принимающая сторона может проверить подпись, используя общий с отправляющей стороной секретный ключ. Если подпись верна, то получатель может доверять токenu и использовать данные, содержащиеся в его полезной нагрузке.

Важно отметить, что подпись не является шифрованием. Токен не защищен от чтения, поэтому конфиденциальные данные должны быть помещены только в зашифрованную часть токена, если это необходимо. Использование в проекте:

Листинг 1.7 создание JWT токена для пользователя

```
const generateToken = (id, email) => {  
  return jwt.sign({ id: id, email: email }, process.env.SECRET_KEY, {  
    expiresIn: "24h",  
  });  
};  
  
const token = generateToken(user.id, user.email);
```

В проекте не было большой необходимости в особом виде шифрования, поэтому для создания токена в заголовки не была передана дополнительная информация. В качестве стандартного метода шифрования в JWT используются

HS256, что означает, что данные будут подписываться и проверяться с помощью HMAC алгоритма с использованием ключа, указанного в качестве параметра `process.env.SECRET_KEY`.

HMAC (Hash-based Message Authentication Code) - это алгоритм, который используется для создания имитовставки. Он позволяет подписать сообщение с помощью секретного ключа, чтобы обеспечить его целостность и аутентичность. При верификации HMAC создается для полученных данных на стороне сервера и сравнивается с подписью, которая была отправлена клиентом. Если значения совпадают, то можно считать, что данные не были изменены и переданы легитимным пользователем.

Таким образом, в соответствии с листингом 1.7 можно сказать, что функция `generateToken()`, принимает 2 параметра: айди и почту пользователя, секретное значение и так же объект в котором указывается на какое время выдается токен. Эта функция возвращает уже подписанный и готов к использованию токен, который возвращает сервер клиенту для дальнейшего клиент/серверного взаимодействия.

Листинг 1.8 Обработка данных пользователя и JWT токена.

```
const { date, comment } = req.body;
const token = req.headers.authorization;
const tokenSplitted = token.split(" ")[1];
const decode = jwt.verify(tokenSplitted, process.env.SECRET_KEY);
```

После авторизации пользователя на сайте и проверки подключения, он имеет право взаимодействовать с датами, а именно добавлять, получать и удалять их. Для этого пользователь делает запрос на сервер и передает в качестве аргументов помимо необходимых данных, так же и токен из авторизованного заголовка HTTP-запроса. А так же важно уточнить, что токен передается с припиской `Bearer`, который говорит о том, что данный токен является токеном на авторизацию.

Таким образом, исходя из листинга 1.8, функция `verify` принимает значение токена и секретное значение для получения `id` пользователя и дальнейшей работа с базой данных, что повышает безопасность системы в разы.

Bcrypt

Библиотека `Bcrypt.js` - это JavaScript библиотека, которая предоставляет простой интерфейс для хеширования паролей с использованием алгоритма `Bcrypt`.

Bcrypt - один из наиболее безопасных алгоритмов хеширования паролей, который использует медленную функцию хеширования, чтобы затруднить атаки перебором и словарными атаками.

Он также использует соль, что делает его еще более устойчивым к атакам. Bcrypt.js предоставляет методы для создания и проверки хешей паролей. Для создания хеша пароля можно использовать метод `bcrypt.hashSync()` или `bcrypt.hash()`. Эти методы принимают пароль и количество раундов хеширования в качестве параметров и возвращают хеш пароля. Для проверки хеша пароля можно использовать метод `bcrypt.compareSync()` или `bcrypt.compare()`. Эти методы принимают пароль и хеш пароля в качестве параметров и возвращают `true`, если хеш соответствует паролю, и `false` в противном случае.

Листинг 1.9 Создание захэшированного пароля

```
const hashPassword = await bcrypt.hash(password, 5);
```

Листинг 1.10 Сравнение пароля и захэшированного пароля

```
let comparePassword = await bcrypt.compare(password, user.password);
```

Исходя из примеров листинга 1.9 и листинга 1.10, видно использования данной библиотеки в проекте. Для создания захэшированного пароля использовалась функция асинхронная функция `hash()`, которая принимает в качестве аргументов сам пароль и соль. Соль - это случайный набор данных, который добавляется к паролю до его хеширования. Использование соли делает хеш уникальным для каждого конкретного пароля и предотвращает возможность использования "общих" хешей для подбора пароля методом перебора или словарной атаки. Для сравнения пароль используется асинхронная функция `compare()`, которая принимает в качестве аргументов пароль введенный пользователем и пароль полученный из базы данных.

Express-validator

Express-validator - это библиотека для проверки и валидации данных, получаемых в запросах веб-приложений на Node.js с использованием фреймворка Express.

С ее помощью можно легко определять правила валидации для различных типов данных, таких как строки, числа, даты и другие. Библиотека обеспечивает удобный интерфейс для задания правил и сообщений об ошибках,

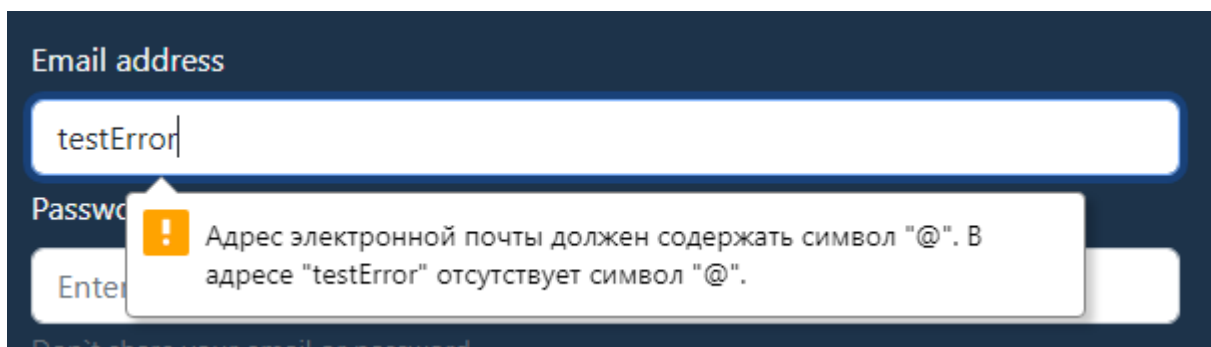
а также позволяет выполнять кастомную валидацию с помощью пользовательских функций.

Эта библиотека является очень гибкой и популярной среди разработчиков Node.js при создании веб-приложений, особенно в случаях, когда важна надежность и безопасность вводимых пользователем данных.

Листинг 1.11 Валидация почты

```
router.post("/registration", body("email").isEmail(), userController.registration);  
router.post("/login", body("email").isEmail(), userController.login);
```

В листинге 1.11 продемонстрировано использование express-validator в дипломной работе. В данном коде в качестве одного из аргументов передается функция валидации isEmail(), которая получает значение “email” из объекта “body”. На скриншоте 2.3 приведен пример вывода при некорректном вводе данных.



Скриншот 2.3 Ошибка ввода логина

3.3 Библиотеки клиентской части

React.js

Как у бэкэнд части `express.js` основополагающая библиотека, так у клиентской части это `React.js`. Этот инструмент позволяет создавать и обрисовывать компоненты, которые могут быть переиспользованы в различных частях проекта, а так же позволяет автоматически обновлять данные при их изменению.

В основе `React.js` лежит концепция "однонаправленного потока данных" (One Way Data Flow), когда данные передаются сверху вниз по дереву компонентов. Это позволяет управлять состоянием приложения и обновлять только те компоненты, которые действительно нуждаются в обновлении без перерисовки всего интерфейса.

`React.js` также предоставляет средства для работы с JSX-синтаксисом, который позволяет описывать структуру пользовательского интерфейса внутри JavaScript-кода. Также, `React.js` можно использовать вместе с другими библиотеками и фреймворками для создания полноценных SPA-приложений (Single Page Applications).

React-router-dom

React Router DOM - это библиотека маршрутизации для приложений React, которая позволяет управлять навигацией между страницами и компонентами приложения.

Она предоставляет компоненты, которые позволяют определять маршруты и связывать их с соответствующими компонентами вашего приложения. Когда пользователь переходит по URL-адресу, соответствующий компонент отображается на экране.

React Router DOM также обеспечивает возможность использования дополнительных функций, таких как вложенные маршруты, защищенные маршруты (требующие авторизации), передача параметров через URL и т.д.

Листинг 2.8 Файл управляющий отображающим файлы в зависимости от URL

```
import React, { useState, useEffect } from "react";
import { Routes, Route, Navigate } from "react-router-dom";
import Auth from "../pages/Auth";
```

```

import Calendar from "../pages/Calendar";
import { check } from "../api/api";
import {
  LOGIN_ROUTE,
  REGISTRATION_ROUTE,
  CALENDAR_ROUTE,
} from "../utils/const";

async function CheckAuth(token) {
  try {
    const response = await check(token);
    return response.ok;
  } catch (error) {
    console.log(error);
  }
}

function PrivateRoute(props) {
  const [auth, setAuth] = useState(null);

  useEffect(() => {
    const token = localStorage.getItem("token");
    CheckAuth(token).then((auth) => {
      setAuth(auth);
    });
  }, []);

  if (auth === null)
    return (
      <div class="d-flex justify-content-center align-items-center">
        <p>Loading...</p>
      </div>
    );

  const ComponentToRender = auth
    ? props.component
    : () => <Navigate to={LOGIN_ROUTE} replace />;

  return <ComponentToRender />;
}

const AppRouter = () => {
  return (
    <Routes>

```

```

<Route path={LOGIN_ROUTE} element={<Auth />} />
<Route path={REGISTRATION_ROUTE} element={<Auth />} />
<Route
  path={CALENDAR_ROUTE}
  element={<PrivateRoute component={Calendar} />}
/>
<Route path="*" element={<Navigate to={LOGIN_ROUTE} replace />} />
</Routes>
);
};

export default AppRouter;

```

В листинге 2.8. показан файл в котором основным инструментом разработки являются компоненты из библиотеки `react-router-dom`. Таким образом, компонент `AppRouter` содержит все возможные маршруты. Тег `<Routes>` является контейнером для всех остальных маршрутов с тегом `<Route>`. Эти компоненты хранят путь URL и компонент, который должен отобразиться при переходе по данному URL. Например, первый тег `<Route>` содержит атрибут `“path”` со значением `“LOGIN_ROUTE”` и атрибут `“element”` со значением `<Auth>`. В следствии информации со скриншота 2.4, можно понять что при переходе по URL хосту + URL из `“LOGIN_ROUTE”`, а то есть целый URL это - <http://localhost:3000/login> будет отображаться страница авторизации.

```

1  export const LOGIN_ROUTE = "/login";
2  export const REGISTRATION_ROUTE = "/registration";
3  export const CALENDAR_ROUTE = "/calendar";
4  export const DATE_ROUTE = "/date";
5

```

Скриншот 2.4 Константы с маршрутами из файла `const.js`

Так же частым компонентом из этой библиотеки является тег `<Navigate>`, который обновляет URL и рендерит новый компонент по этому пути. В листинге 2.8 есть сразу 2 использования этой функции. Первое отображено в компоненте `PrivateRoute`, в этом элементе если пользователь не подтверждает авторизованность, то его перенаправляет с помощью тега `<Navigate>` по роутеру `“LOGIN_ROUTE”`. Второй же такой тег встроен внутри тега `<Route>` и говорит о том, что если пользователь вводит в URL строку такие пути, которых нет на сайте, то необходимо их перенаправить на страницу с роутом `“LOGIN_ROUTE”`.

React-bootstrap

React-bootstrap - это библиотека компонентов React, которая предоставляет набор готовых компонентов пользовательского интерфейса, основанных на Bootstrap. Bootstrap - это популярный фреймворк CSS, который предоставляет различные стили и компоненты для создания современных веб-страниц.

React-bootstrap позволяет использовать компоненты Bootstrap в приложениях React без необходимости написания значительного количества кода. Библиотека содержит множество компонентов, таких как кнопки, модальные окна, формы, таблицы и многое другое.

В листинга 2.3, например, из данной библиотеки используется больше количество различных компонентов. Для создания формы используются соответствующие теги `<Form>` и `<FormGroup>`, а также кнопка `<Button>` для отправки данных и семантические теги `<Container>` и `<Card>`.

React-big-calendar

React-Big-Calendar это библиотека компонентов React, которая предоставляет гибкую и настраиваемую функциональность для создания календарных приложений. Библиотека поставляется с несколькими различными компонентами, которые можно использовать для отображения календарей, планировщиков, расписаний, и т.д.

Эта библиотека позволяет изменять цветовые схемы, формирования дат., формат отображения событий, представления дат, например, отображение, только месяцев. А также поддерживает технологию Drag-and-Drop: пользователи могут перемещать события в календаре с помощью простого перетаскивания мышью.

React-modal

React Modal - это компонент React, который используется для создания модальных диалоговых окон веб-приложений. Модальное окно - это окно, которое появляется поверх основного содержимого страницы и блокирует доступ к нему до тех пор, пока пользователь не закроет модальное окно.

React Modal предоставляет гибкий способ создания пользовательских модальных окон веб-приложений. Он может быть настроен для отображения различных типов контента, таких как текст, изображения или формы, и обеспечивает удобный API для управления состоянием модального окна, таким как его видимость и содержимое.

React Modal имеет множество опций конфигурации, позволяющих настраивать внешний вид и поведение модального окна в соответствии с потребностями приложения. Кроме того, он поддерживает анимации и прозрачность, что делает модальное окно более привлекательным и интерактивным для пользователей. Использование в проекте описано в разделе 2.3.3.

React-icons

React-icons - это библиотека React-компонентов, которая содержит множество популярных значков и иконок, таких как флажки, стрелки, иконки социальных сетей и т.д.

React-icons предоставляет удобный способ использования значков в React-приложениях без необходимости создания своих собственных компонентов для каждой иконки. Каждый компонент иконки React-icons представляет собой простую функциональную компоненту, которая может быть легко импортирована и использована в любом месте кода приложения.

Эта библиотека содержит большое количество иконок, написанных на SVG, Font Awesome, Ionicons и других популярных иконных наборах, что позволяет выбрать подходящий вид иконок для вашего приложения. Кроме того, React-icons легко настраивается, что позволяет изменять размеры и цвета иконок, а также добавлять анимации и другие эффекты для достижения нужного дизайна и интерактивности.

В Листинге 2.6. видно применение данной библиотеки для создание иконки в виде крестиков для выхода из модального окна.

Заключение

Целью проекта было реализовать простой веб-календарь с понятным каждому интерфейсом. Минимальные требования были выполнены все: реализовано клиент-серверное взаимодействие, которое работает с базой данных, реализована система регистрации и авторизации, а так же возможность записи и удаления данных и их отображение на интерфейсе пользователя. Так же были реализованы некоторые и дополнительные цели. Каждый пароль шифруется и хранится в таком виде в базе данных, что обеспечивает безопасность каждому пользователю. Так же данные об активной сессии хранятся в токене и при каждом взаимодействии клиента с приложением происходит проверка корректности этого токена, что также повышает безопасность использования, так как при исходе двадцати четырех часов токен будет уничтожен. Так же важно отметить, что проект реализован таким образом, что он легко масштабируется, а значит, добавление новых функций не будет сверхтяжелой задачей, таким образом, в данном проекте можно дорабатывать еще большое количество разнообразных функций. Кроме того в коде приложения уже начаты реализации некоторых функций. Например, в массиве цветов из файла `colors.js`, так же описаны цвета не только в темном виде, но и в светлом, а так как описание цветов реализовано константами из массива, то реализовать темную тему не так уж и трудно. Вдобавок можно реализовать и мобильный интерфейс, так как в проекте уже есть специально написанный для этой задачи хук – `resizeHook.js`, который определяет изменения ширины и высоты экрана, что может помочь для реализации разных стилей для разных форматов экрана. Так же можно еще более расширить функционал приложения, добавив настройки календаря, что бы каждый пользователь мог настроить его под себя, например, изменив окно вывода с месяцев на года и тому подобное. Кроме того можно реализовать полноценное мобильное приложение, которое использовало бы ту же самую базу данных, что увеличит удобство пользования в разы.

Но в проекте так же существуют и недочеты, которые появились во время разработки. Самая главная проблема, на мой взгляд, это некорректное формирование базы данных из-за нехватки опыта. Существующая база на данный момент не соответствует всем нормам проектирования. Так в соответствии со скриншотом 1.7 база данных, по-моему мнению, находится не в третьей нормальной форме и это может помешать при масштабировании проекта.

Таким образом, в ходе создания веб-приложения «Календарь событий» я смог изучить и главное опробовать на личном опыте огромное количество технологий. От таких глобальных как фреймворки `bootstrap` и `express.js`, до маленьких по типу `express-validator`. Помимо прочего смог узнать больше о проектировании и настройке баз данных и подключении сервера к ней. Так же

более подробно узнал о работе HTTP запросов, которыми общаются клиент и сервер. И смог узнать, как выглядит веб разработка в целом.

Список использованной литературы

1. Документация Bootstrap:[электронный ресурс]. //URL: <https://getbootstrap.com/docs/4.1/getting-started/introduction/> (Дата обращения: 17.03.2023)
2. Русскоязычная документация по React:[электронный ресурс]. //URL: <https://ru.reactjs.org/tutorial/tutorial.html> (Дата обращения 25.03.2023)
3. Уроки по React JS:[электронный ресурс]. //URL: <https://itproger.com/course/react-js> (Дата обращения 10.02.2023)
4. Web-технологии : учебно-методическое пособие / А. В. Моргунов :[электронный ресурс]. //URL: http://ellib.sibsutis.ru/ellib/2022/1059_Morgunov_A.V._Veb-tekhnologii.pdf (Дата обращения 20.12.2022)
5. Что такое NPM:[электронный ресурс]. // URL: <https://wiki.merionet.ru/servernye-resheniya/116/chto-takoe-npm-rukovodstvo-po-node-package-manager-dlya-nachinayushchih/> (Дата обращения 19.05.2023)
6. Что такое CORS:[электронный ресурс]. // URL: <https://developer.mozilla.org/ru/docs/Web/HTTP/CORS>(Дата обращения 19.05.2023)
7. Сравнение микросервисной и монолитной архитектуры:[электронный ресурс]. // URL: <https://www.atlassian.com/ru/microservices/microservices-architecture/microservices-vs-monolith>(Дата обращения 22.04.2023)
8. Как работает Middleware в Express:[электронный ресурс]. //URL: <https://habr.com/ru/companies/otus/articles/562914/>(Дата обращения 20.05.2023)
9. HTTP протокол в js:[электронный ресурс]. //URL: <https://learn.javascript.ru/fetch>(Дата обращения 19.02.202)