МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«СМОЛЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ФИЗИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ

Кафедра прикладной математики и информатики

**КУРСОВАЯ РАБОТА**

на тему: «Создание одностраничного приложения средствами React»

Выполнил:  
студент 2 подгруппы  
3 курса ИС21 ФМФ  
Павлюк Денис Сергеевич

Научный руководитель:  
доцент Самойлова Татьяна Аркадьевна

Сдана на проверку:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты и оценка:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Смоленск, 2024**

**Оглавление**

[**Введение** 3](#_Toc168413310)

[**Глава 1. Исследование решений и теоретические основы** 4](#_Toc168413311)

[1.1 Анализ существующих решений 4](#_Toc168413312)

[*1.2 React* 5](#_Toc168413313)

[*1.3 Одностраничные приложения SPA* 7](#_Toc168413314)

[*1.4 Firebase* 8](#_Toc168413315)

[**Глава 2. Разработка приложения** 9](#_Toc168413316)

[*2.1 Постановка задачи* 9](#_Toc168413317)

[2.1.2 Требования к функционалу 10](#_Toc168413318)

[2.1.3 Технические требования 10](#_Toc168413319)

[*2.2 Начало разработки* 11](#_Toc168413320)

[*2.3 Создание одностраничной навигации* 11](#_Toc168413321)

[*2.4 Подключение Firebase Authentication* 11](#_Toc168413322)

[*2.5 Ограничение навигации для неавторизованных пользователей* 13](#_Toc168413323)

[*2.6 Создание компонентов приложения* 13](#_Toc168413324)

[*2.7 Подключение Firebase Firestore для управления данными задач* 14](#_Toc168413325)

[*2.8 Функционал приложения* 15](#_Toc168413326)

[2.8.1 Функционал “TodoList.jsx” 15](#_Toc168413327)

[2.8.2 Функционал “TodoCalendar” 15](#_Toc168413328)

[2.8.3 Функционал “Home” 16](#_Toc168413329)

[*2.11 Стилизация компонентов Tailwind CSS* 18](#_Toc168413330)

[*2.10 Структура приложения и реализация интерфейса* 20](#_Toc168413331)

[2.10.1 Реализация функционала для входа 21](#_Toc168413332)

[2.10.2 Реализация функционала для регистрации 21](#_Toc168413333)

[2.10.3 Реализация функционала для навигации 22](#_Toc168413334)

[2.10.4 Реализация функционала для домашней страницы 24](#_Toc168413335)

[2.10.5 Реализация функционала для Todo List 25](#_Toc168413336)

[2.10.6 Реализация функционала для Calendar 26](#_Toc168413337)

[**Заключение** 29](#_Toc168413338)

[**Список использованных источников** 30](#_Toc168413339)

# **Введение**

Объектом исследования является процесс разработки одностраничного веб-приложения для управления задачами. Одностраничные приложения (SPA) весьма актуальные благодаря своей высокой производительности, интуитивно понятному пользовательскому интерфейсу и удобству использования. Использование React, одного из самых популярных JavaScript-фреймворков для разработки пользовательских интерфейсов, позволяет создавать динамичные и отзывчивые веб-приложения. Система управления задачами, как пример такого приложения, представляет собой инструмент, способный повысить личную продуктивность и организованность пользователей, обеспечивая удобный интерфейс.

Предметом исследования выступают методы и инструменты разработки, а также технологии, используемые для создания эффективных и удобных в использовании приложений.

Цель данной курсовой работы – разработка одностраничного веб-приложения для управления задачами с использованием React.

Для достижения этой цели был поставлен ряд задач:

1. Проанализировать существующие решения и определить основные требования к системе управления задачами.
2. Разработать клиентскую часть на React JS с использованием Firebase для управления базой данных и Tailwind CSS для стилизации элементов интерфейса
3. Реализовать основные функциональные возможности задачника: создание, изменение, удаление задач и фильтрация задач по различным критериям.
4. Оценить экономическую эффективность разработки и использования задачника.

# **Глава 1. Исследование решений и теоретические основы**

* 1. Анализ существующих решений

Задачники представляют собой программное обеспечение, предназначенное для управления задачами и проектами. Они позволяют пользователям организовывать свою работу, устанавливать приоритеты, отслеживать прогресс и достигать поставленных целей. В современных условиях задачники часто используют как в личной продуктивности, так и в эффективной работе команд и организаций.

Выделяется в основном три вида:

Персональные задачники: они предназначены для индивидуального использования. Они фокусируются на простоте и удобстве, предоставляя базовые функции для управления личными задачами.

Пример: Todoist - популярное приложение для управления личными задачами. Оно позволяет пользователям создавать списки задач, устанавливать приоритеты и дедлайны, а также отслеживать свой прогресс.

Командные задачники: предназначены для совместной работы над проектами и задачами. Они помогают командам координировать усилия, распределять обязанности и отслеживать выполнение задач.

Пример: Trello - визуальная система управления проектами, позволяющая командам организовывать задачи с помощью карточек и досок. Каждая карточка представляет собой задачу, которую можно перемещать между колонками, такими как "To Do", "In Progress" и "Done".

Корпоративные задачники: предназначены для использования в больших организациях и компаниях. Они обеспечивают масштабируемость, безопасность и интеграцию с корпоративными системами.

Пример: Asana - мощный инструмент для управления проектами и задачами, который помогает командам планировать, организовывать и отслеживать работу.

Основные функции задачников:

1. Создание задач: Пользователи могут создавать задачи, устанавливать сроки выполнения и приоритеты.
2. Редактирование и удаление задач: Возможность изменять информацию о задаче или удалять её при необходимости.
3. Установка напоминаний и уведомлений: Задачники часто предлагают напоминания о дедлайнах и важных событиях.
4. Фильтрация и сортировка: Функции для фильтрации задач по различным критериям, таким как дата, приоритет или статус.

## *1.2 React*

React – это библиотека JavaScript для разработки пользовательских интерфейсов. React используется для разработки одностраничных приложений, а также мобильных приложений при использовании таких технологий, как React Native. В основе React лежит компонентный подход, который позволяет разрабатывать интерфейсы из независимых компонентов. Это говорит о том, что компонент представляется самостоятельным модулем, который может содержать логику и разметку для части пользовательского интерфейса.

Компоненты могут быть двух видов:

* Функциональные: простые и не имеют состояния. Они определяются как функции, которые принимают свойство props и возвращают элементы React.
* Классовые: используются для более сложных логик и имеют внутреннее состояние. Они определяются как классы, наследующиеся от React.Component.

React позволяет лучше визуализировать разработку разметки и логики интерфейса. Использование JSX даёт возможность разработчикам создавать компоненты, в которых логика рендеринга тесно связана с остальной логикой интерфейса, делая код более читаемым и управляемым. Использование JSX является индивидуальным предпочтением - «React не требует использования JSX, но большинство людей считают его полезным в качестве наглядного пособия при работе с пользовательским интерфейсом внутри кода JavaScript. Это также позволяет React отображать более полезные сообщения об ошибках и предупреждения.». [1]

Также плюсом фреймворка является большое количество библиотек, которые позволяют расширять возможности React и решать различные задачи. Основным принципом фреймворка является модульность и разделение задач. Поэтому вместо создания одной большой библиотеки создаются специализированные, каждая из которых решает конкретную задачу. Пользуясь необходимыми библиотеками, проект не перегружается лишним функционалом. Наличие такого количество библиотек, объясняется - «за счет крупного комьюнити есть множество энтузиастов, которые создают свои собственные решения.» [2]

Библиотеки в React хорошо совместимы друг с другом и легко интегрируются в проект. Это позволяет разработчикам быстро добавлять новые функции и улучшать приложение без необходимости значительных изменений в коде.

## *1.3 Одностраничные приложения SPA*

Одностраничные приложения (SPA, Single Page Applications) представляют собой современный подход к разработке веб-приложений, который обеспечивает улучшенный пользовательский опыт за счет быстрого и интерактивного интерфейса. SPA позволяют загружать и отображать данные без необходимости перезагрузки всей страницы, что делает взаимодействие с приложением более плавным и отзывчивым.

В SPA все приложение загружается на единственную HTML-страницу. При первой загрузке пользователю предоставляется полный интерфейс, который динамически обновляется по мере взаимодействия с приложением. Это отличается от традиционных многостраничных приложений, где каждая новая страница требует полной перезагрузки браузера.

Также этот тип разработки активно используют асинхронные запросы “AJAX” для взаимодействия с сервером. Это позволяет загружать данные в фоновом режиме и обновлять интерфейс без перезагрузки страницы. Библиотеки, такие как Axios или встроенные функции fetch в JavaScript, часто используются для выполнения таких запросов.

В SPA маршрутизация осуществляется на стороне клиента, что позволяет изменять URL без полной перезагрузки страницы. Библиотеки, такие как React Router, обеспечивают управление маршрутами, позволяя динамически отображать различные компоненты на основе текущего URL.

SPA хранят и управляют состоянием на стороне клиента. Для управления состоянием могут использоваться встроенные хуки React, такие как useState и useReducer или сторонние библиотеки, такие как Redux.

Преимущества SPA:

* SPA обеспечивает более быстрое взаимодействие с пользователем, так как большинство обновлений происходит без перезагрузки страницы. Это приводит к более плавному и интуитивному пользовательскому опыту.
* Благодаря асинхронной загрузке данных и динамическому обновлению интерфейса, SPA создают впечатление более высокой производительности по сравнению с традиционными веб-приложениями.
* SPA может использовать технологии кэширования и хранить данные на стороне клиента, что позволяет приложению работать даже в условиях ограниченного или отсутствующего интернет-соединения.
* Благодаря компонентному подходу в React, разработка SPA становится более модульной. Компоненты можно легко переиспользовать и тестировать отдельно, что упрощает разработку и поддержку кода.

Недостатки SPA:

* Одним из основных недостатков SPA является сложность поисковой оптимизации. Поскольку контент загружается динамически, поисковым системам может быть сложно индексировать страницы. Для решения этой проблемы используются такие техники, как серверный рендеринг (SSR) и предрендеринг.
* Поскольку все приложение загружается сразу, начальная загрузка может занять больше времени по сравнению с традиционными многостраничными приложениями. Однако это можно смягчить с помощью оптимизации загрузки и кэширования.
* Разработка SPA может быть сложнее из-за необходимости управления состоянием на клиенте и обеспечения синхронности данных. Это требует более сложной архитектуры и использования дополнительных библиотек.

## *1.4 Firebase*

Firebase – это платформа для разработки мобильных и веб-приложений, созданная и поддерживаемая компанией Google. Она предоставляет множество инструментов и сервисов, которые помогают разработчикам создавать, управлять и развертывать приложения. Firebase особенно популярен среди разработчиков благодаря своей простоте использования и интеграции с другими продуктами Google.

Основные возможности Firebase:

* Firebase Authentication: поддерживает различные способы аутентификации, включая email и пароль, телефон, Google, Facebook, Twitter и другие провайдеры OAuth.
* Firebase Realtime Database и Firestore: две облачные базы данных Realtime Database и Firestore. Обе базы данных синхронизируют данные в реальном времени и позволяют хранить и получать данные с минимальной задержкой. Realtime Database – иерархическая база данных JSON, подходящая для простых приложений и задач, требующих низкой задержки. Firestore – более мощная и гибкая база данных с поддержкой коллекций и документов, обладающая улучшенной производительностью и возможностями масштабирования.
* Firebase Hosting Firebase: хостинг предоставляет быстрый и безопасный хостинг для веб-приложений. Он поддерживает автоматическое развертывание, SSL, глобальное распределение контента и многое другое. Firebase Hosting идеально подходит для SPA и других веб-приложений, требующих высокой производительности и безопасности.
* Firebase Cloud Messaging (FCM): позволяет отправлять push-уведомления на устройства пользователей. Это инструмент для взаимодействия с пользователями.

Исходя из всех возможностей Firebase, было принято решение использовать эту платформу. На данном этапе разработки используется: Firebase Authentication, Firebase Firestore [4]

# **Глава 2. Разработка приложения**

# *2.1 Постановка задачи*

Разрабатываемое приложение представляет собой одностраничное приложение (SPA), созданное с использованием React, предназначенное для управления задачами. Основные функции приложения включают создание, редактирование, удаление задач, а также фильтрацию задач по различным критериям, таким как текущие задачи, задачи на неделю, просроченные задачи, предстоящие задачи и выполненные задачи. Приложение предназначено для индивидуальных пользователей, которым необходимо эффективно управлять своими делами и задачами.

2.1.2 Требования к функционалу

Создание задач: Пользователь должен иметь возможность создавать новые задачи с указанием названия, описания, даты выполнения и статуса.

Редактирование задач: Пользователь должен иметь возможность редактировать ранее созданные задачи.

Удаление задач: Пользователь должен иметь возможность удалять задачи.

Фильтрация задач: Пользователь должен иметь возможность фильтровать задачи по различным критериям:

* Текущие задачи (задачи, которые должны быть выполнены сегодня).
* Задачи на неделю (задачи, которые должны быть выполнены в течение текущей недели).
* Просроченные задачи (задачи с истекшим сроком выполнения).
* Предстоящие задачи (задачи, которые должны быть выполнены в будущем).
* Выполненные задачи (задачи, которые были отмечены как выполненные).

2.1.3 Технические требования

Приложение должно быть одностраничным (SPA) и быстро реагировать на действия пользователя без перезагрузки страницы.

Приложение должно быть кроссбраузерным и корректно работать в современных веб-браузерах.

Приложение должно использовать Firebase для аутентификации пользователей и хранения данных о задачах.

Интерфейс приложения должен быть интуитивно понятным и удобным для пользователей.

Приложение должно быть масштабируемым и легко расширяемым для добавления новых функций в будущем.

# *2.2 Начало разработки*

Первым шагом в разработке является настройка окружения и установка всех необходимых зависимостей. Для этого мы используем инструмент Create React App. Воспользуемся этой командой (рис 1):

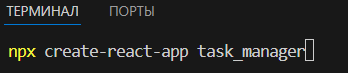


Рисунок 1 – npx команда create-react-app

# *2.3 Создание одностраничной навигации*

На следующем этапе мы настроили одностраничную навигацию с помощью React Router. Это позволило создать структуру маршрутов для компонентов “Home”, “TodoList”, “TodoCalendar” и “UserProfile”.

На данном этапе удобнее сделать одностраничную навигацию с пустыми компонентами, так как это упрощает разработку и тестирование приложения, позволяя сосредоточиться на логике маршрутизации без необходимости заполнять каждый экран содержимым сразу.

# *2.4 Подключение Firebase Authentication*

После настройки навигации можно приступать к подключению Firebase Authentication для реализации функциональности авторизации и регистрации пользователей. Для начала работы с Firebase и интеграции его в React нужно создать проект в Firebase и добавить продукт Authentication. (рис 2)

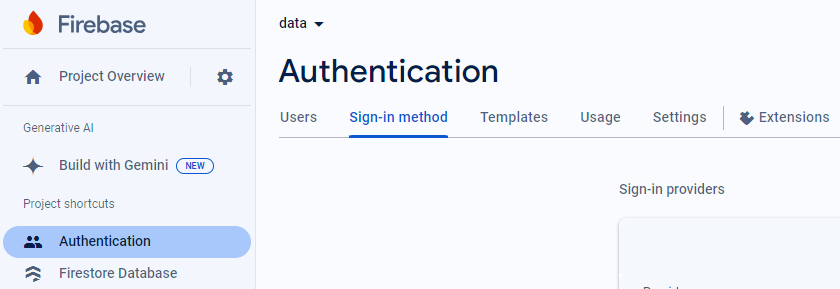


Рисунок 2 – Страница с добавленным продуктом Authentication

Теперь, когда проект создан, нам нужно интегрировать его в наше React-приложение. В React для этого используется библиотека firebase, поэтому выполним команду (рис 3).

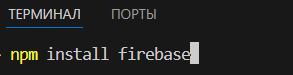


Рисунок 3 – npm команда install firebase

Так же понадобится создать файл конфигурации Firebase. Файл “firebase.jsx” создается в корневой директории проекта.

В конфигурации код инициализирует Firebase в приложении, используя конфигурацию, определённую в объекте firebaseConfig. Конфигурация включает в себя ключ API, домен авторизации, идентификатор проекта, хранилище данных, отправителя сообщений и идентификатор приложения, которые предоставляются после создания проекта Firebase.

Инициализация происходит через вызов функции initializeApp, передавая ей объект конфигурации. Эта функция создаёт экземпляр приложения Firebase, который затем используется для получения экземпляров других сервисов Firebase, таких как аутентификация (getAuth) и база данных Firestore (getFirestore).

В конце кода экспортируются переменные auth и db, чтобы они могли быть использованы в других частях приложения для работы с аутентификацией и базой данных Firestore.

Пример конфигурации смотрите в приложении 1.

# *2.5 Ограничение навигации для неавторизованных пользователей*

Основная причина такого ограничения заключается в том, что большинство действий, выполняемых на этих страницах, предполагают изменение состояния системы или сохранение данных в базе данных. Без аутентификации и идентификации пользователя невозможно гарантировать, что любые данные, введенные или изменения, внесенные таким пользователем, могут быть надежно сохранены или связаны с его персональным профилем в системе. В результате, все действия останутся незарегистрированными и не будут иметь никакого смысла в системе.

Для этого был создан компонент “ProtectedRoute.jsx”, который проверял, вошел ли пользователь в систему, и перенаправлял его на страницу входа, если он не был авторизован.

# *2.6 Создание компонентов приложения*

На этом этапе мы приступили к разработке основных компонентов приложения: Home, TodoList, TodoCalendar и UserProfile. Эти компоненты обеспечивали основной функционал нашего приложения.

* “Home“: Страница демонстрирует динамичный обзор текущего состояния задач пользователя, включая количество задач на сегодня и на эту неделю, а также разделение выполненных, просроченных и предстоящих задач
* “TodoList“: Компонент для отображения списка задач, позволяющий пользователям добавлять, редактировать, удалять задачи и отмечать их выполнение.
* “TodoCalendar“: Компонент для отображения задач в календарном формате, для более легкой визуализации своих планов.
* “UserProfile“: Компонент для редактирования профиля пользователя.

# *2.7 Подключение Firebase Firestore для управления данными задач*

Для хранения и управления данными задач пришло время Firebase Firestore. Для подключения Firebase Firestore и управления данными задач в нашем React-приложении необходимо его добавить в проект Firebase и настроить.

Процесс настройки представляет собой набор правил безопасности для облачного хранилища Firestore, определяющий условия доступа к данным в базе данных. Он состоит из двух основных блоков правил, применяемых к категориям документов в базе данных (рис 4).

Первый блок правил применяется ко всем документам в базе данных и позволяет всем авторизованным пользователям читать эти документы. Это сделано для того, чтобы только те пользователи, которые прошли процесс аутентификации и имеют идентификатор сессии, получили доступ к чтению данных.

Второй блок правил для документов в коллекции tasks. Он разрешает как чтение, так и запись (создание, обновление, удаление) для всех авторизованных пользователей. Также, он вводит дополнительное условие для создания новых задач: только авторизованный пользователь, чей идентификатор совпадает с полем “userId” в данных новой задачи, может создать её. Это обеспечивает, создание задачи исключительно пользователем.

  
Рисунок 4 – правила безопасности Firestore.

# *2.8 Функционал приложения*

2.8.1 Функционал “TodoList.jsx”

Реализация основных функциональных возможностей задачника является ключевой задачей приложения. Этот процесс включал в себя несколько этапов, каждый из которых был направлен на обеспечение удобства использования.

Для начала необходимо было реализовать создание и удаление задач. Эти функции стали основой компонента “TodoList”, где пользователь может легко добавить новую задачу, указав название, дату и время, а также удалить те, которые больше не актуальны. Реализация этих функций включает в себя работу с базой данных Firestore для сохранения и удаления задач. Функция “addTodo” позволяет добавлять новые задачи, сохраняя их в коллекции “tasks” в базе данных Firestore. При успешном добавлении задачи, она автоматически добавляется в локальное состояние приложения, обновляя список задач. Функция “removeTodo” используется для удаления задач из коллекции “tasks” в базе данных Firestore и из локального состояния приложения. Функции “addTodo” и “removeTodo” предоставлены в приложении 2.

2.8.2 Функционал “TodoCalendar”

В функционале TodoCalendar предусмотрены возможности редактирования и удаления задач, которые позволяют пользователям адаптировать свой план задач под текущие потребности и убирать выполненные или ненужные элементы списка.

Функция “handleEditTask” предназначена для изменения названия задачи. Она принимает объект задачи в качестве параметра, который содержит идентификатор и новое название задачи. Сначала функция обращается к документу задачи в базе данных Firestore, используя идентификатор задачи, и обновляет его название с помощью функции “updateDoc”. Затем, чтобы синхронизировать изменения в пользовательском интерфейсе, функция обновляет локальный массив задач, заменяя старое название задачи на новое в соответствующем объекте задачи.

Функция handleDeleteTask отвечает за удаление задач из списка. Она принимает идентификатор задачи, которую нужно удалить. Аналогично, функция обращается к документу задачи в базе данных Firestore и удаляет его с помощью функции deleteDoc. После удаления задачи из базы данных, функция также обновляет локальный массив задач, удаляя из него объект задачи с соответствующим идентификатором, чтобы пользовательский интерфейс отражал актуальное состояние задач. Посмотреть функции “handleEditTask” и “handleDeleteTask” можно в приложении 3.

2.8.3 Функционал “Home”

Всю функциональность фильтрации задач на себя взял компонент “Home”. Фильтры включают:

* Задачи на сегодня: за этот фильтр отвечает “tasksToday”, который показывает задачи, срок выполнения которых приходится на текущий день. Для этого используется сравнение даты задачи с форматированной строкой текущей даты, представленной в виде ISO строки и разделенной на части, где учитывается только дата без времени (рис 5).



Рисунок 5 - фильтр “tasksToday”

* Задачи на эту неделю: Фильтр “tasksThisWeek” отображает задачи, срок выполнения которых составляет менее или равно семи дней от текущей даты. Для каждого задания сравнивается абсолютное значение разницы между датой задачи и сегодняшней датой, преобразованной в количество дней. Задачи, срок выполнения которых попадает в этот диапазон и которые еще не выполнены, отображаются в списке (рис 6).

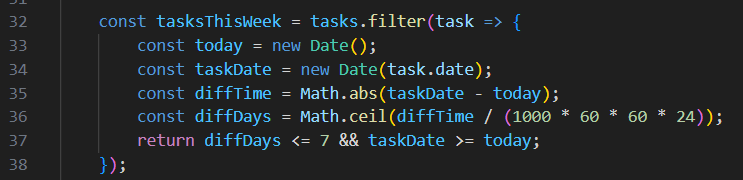


Рисунок 6 - фильтр “tasksThisWeek”

* Выполненные задачи на сегодня: “copletedTasksToday” отображает задачи, выполненные в течение текущего дня, что позволяет пользователю видеть результаты своей работы за день. Она работает аналогично фильтру “tasksToday”, но добавляет условие, что задача должна быть помечена как выполненная (isCompleted = true) (рис 7).



Рисунок 7 - фильтр “copletedTasksToday”

* Выполненные задачи: “completedTasks“ показывает все задачи, отмеченные как выполненные, независимо от даты их выполнения (рис 8).



Рисунок 8 - фильтр “completedTasks“

* Предстоящие задачи: “upcomingTasks” содержит задачи, которые еще не выполнены и срок выполнения которых не попадает в текущую неделю или сегодняшний день. Исключает задачи, которые уже были отфильтрованы как задачи на эту неделю или на сегодня, и которые не выполнены. Таким образом, в список попадают только задачи с будущими датами выполнения (рис 9).

 Рисунок 9 - фильтр “upcomingTasks”

* Просроченные задачи: “overdueTasks” отображает задачи, срок выполнения которых уже истек. Сравнивает дату каждой задачи с текущей датой. Если дата задачи уже прошла текущую дату и задача не выполнена, она считается просроченной и добавляется в список (рис 10).

 Рисунок 10 - фильтр “overdueTasks”

Для отображения задач используются две функции: “renderTasks” и “renderTasksWithDates”. Первая функция отображает только заголовки задач, в то время как второй добавляет к каждой задаче информацию о дате выполнения. Пример их реализации продемонстрирован в приложении 4.

# *2.11 Стилизация компонентов Tailwind CSS*

Для стилизации проекта была выбрана библиотека Tailwind CSS. Это позволило быстро и эффективно стилизовать элементы интерфейса без написания большого количества CSS-стилей.

Сначала необходимо установить tailwindcss через npm и создайте свой tailwind.config.js файл (рис 11).



Рисунок 11 – npm команда install -D tailwindcss и tailwindcss init

После выполнения этих команд в проекте появляется два файла: “tailwind.config.js” и “postcss.config.js”. В файл tailwind.config.js необходимо настроить пути к вашим файлам компонентов (рис 12).

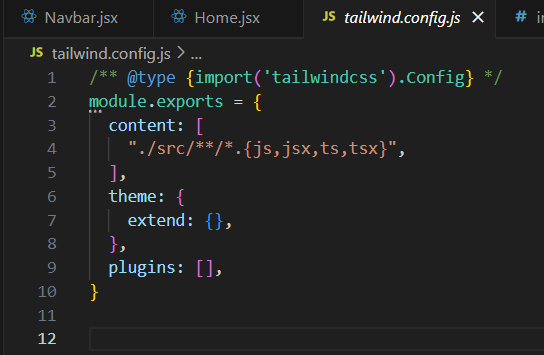


Рисунок 12 – Настройка пути к шаблонам

Также необходимо добавить директивы Tailwind в стили “index.css” (рис 13).

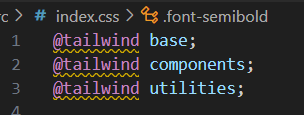


Рисунок 13 – Добавление директив

После настройки Tailwind CSS можно приступать к стилизации компонентов. Стили прописываются в JSX-коде компонентов, что позволяет избежать написания отдельных CSS-файлов. [5]

# *2.10 Структура приложения и реализация интерфейса*

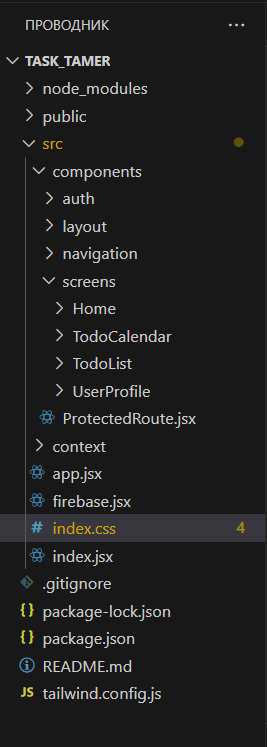
****

Рисунок 14 – Структура проекта

Папка “components” содержит все основные компоненты приложения, разделенные по их функциональности:

* “auth”: Эта папка содержит компоненты, связанные с аутентификацией пользователя.  
  “layout”: который предназначен для использования как общий контейнер.
* “navigation”: в этой папке находятся компоненты, связанные с навигацией.
* “screens”: содержит компоненты, представляющие собой отдельные компоненты приложения. Каждый компонент отвечает за отображение определенной функциональности.
* “context”: предназначен для хранения файлов, связанных с контекстами React, которые используются для управления глобальным состоянием в приложении.

2.10.1 Реализация функционала для входа

На странице войти (см. Приложение 5) пользователи заходят в свою учетную запись. Для этого используется компонент “Login”, который отвечает за отображение формы входа и обработку процесса аутентификации пользователя. За отправку формы отвечает функция handleSubmit, которая вызывается при нажатии кнопки “Войти”.

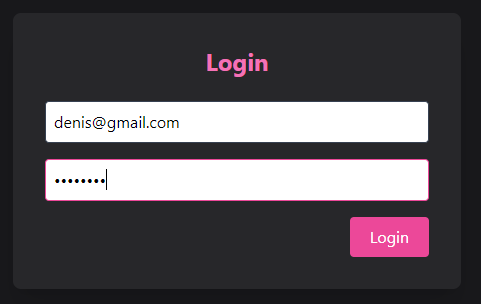


Рисунок 15 – Форма входа в браузере

Функция использует signInWithEmailAndPassword для попытки входа и при успешном входе перенаправляет на главную страницу, а в случае ошибки отображает сообщение об ошибке.

2.10.2 Реализация функционала для регистрации

Компонент Register (см. Приложение 6) включает форму регистрации с полями для имени, электронной почты и пароля, а также обработчик отправки формы, который выполняет регистрацию нового пользователя.

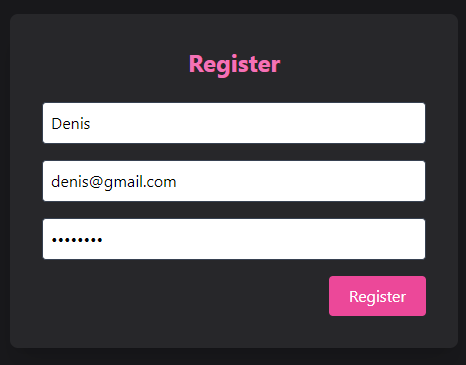


Рисунок 16 – Форма регистрации в браузере

При отправке формы вызывается функция “handleSubmit”. После чего происходит попытка создать нового пользователя с помощью createUserWithEmailAndPassword из Firebase. Если создание проходит успешно, профиль пользователя обновляется с помощью updateProfile, после чего пользователь перенаправляется на главную страницу приложения с помощью navigate('/'). В случае возникновения ошибки, сообщение об ошибке отображается с помощью toast.error.

2.10.3 Реализация функционала для навигации

Компонент “Navbar” (см. Приложение 7) реализует навигационную панель для приложения, которая адаптируется в зависимости от состояния аутентификации пользователя. Он использует react-router-dom для навигации между различными частями приложения и Firebase для управления состоянием аутентификации пользователя.

В навигационной панели “Navbar”, логика отображения элементов меню зависит от того, авторизован ли пользователь или нет. Это достигается за счет использования состояния user, которое обновляется при изменении состояния аутентификации пользователя благодаря onAuthStateChanged из Firebase.

Если пользователь не авторизован, в меню доступны следующие опции:

* Войти: Переход на страницу входа в систему.
* Регистрация: Переход на страницу регистрации нового пользователя.

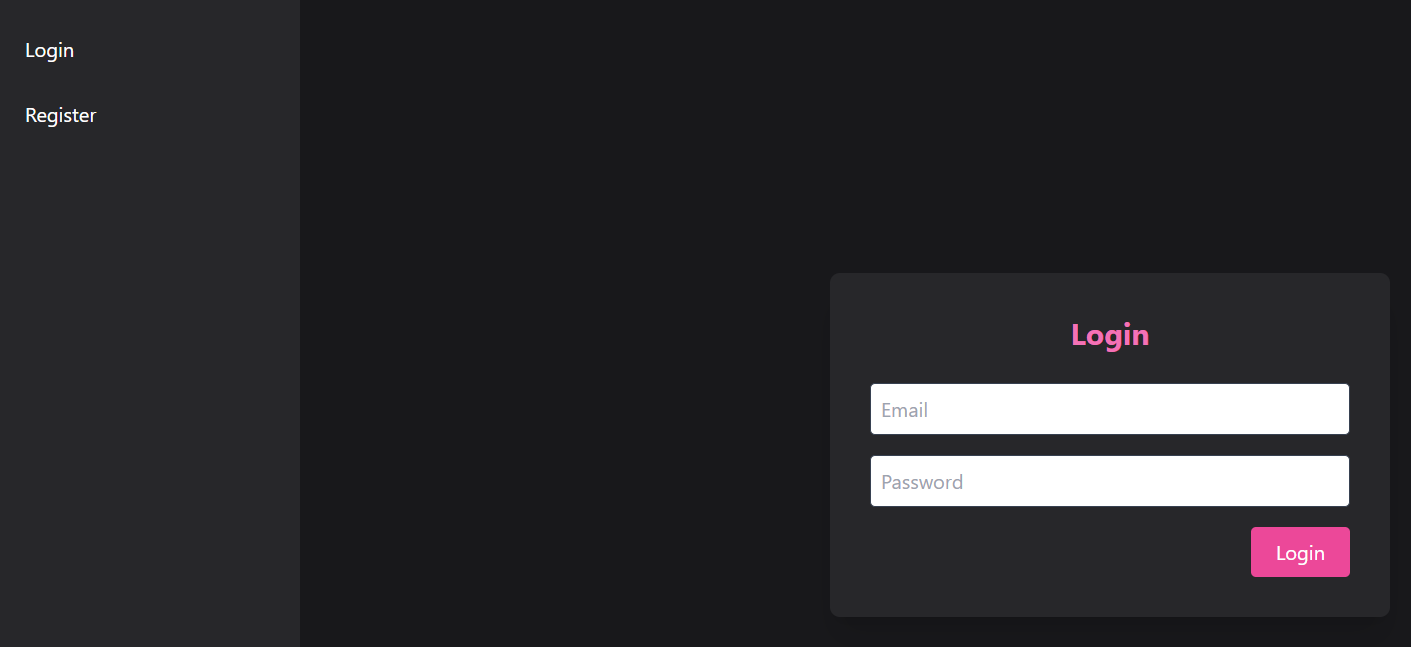


Рисунок 17 – Navbar, когда пользователь не авторизирован

Эти опции предоставляются для тех, кто еще не вошел в систему, позволяя им начать процесс регистрации или входа в приложение.

Если пользователь авторизован, в меню отображается приветствие с именем пользователя, сопровождаемое иконкой стрелки вниз “GoChevronDown” из библиотеки React-icons. При клике на эту иконку появляется выпадающее меню с опциями: редактировать и выйти.

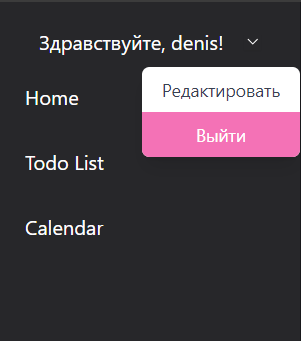


Рисунок 18 – Navbar, когда пользователь авторизован.

Кроме того, для авторизованных пользователей доступны следующие ссылки:

* Home: Переход на главную страницу приложения.
* Todo List: Переход на страницу списка задач.
* Calendar: Переход на страницу календаря.

2.10.4 Реализация функционала для домашней страницы

Компонент Home (см. Приложение 8) представляет собой домашнюю страницу приложения, которая отображает информации о задачах пользователя, полученных из базы данных Firestore. Эта страница отображает задачи в удобном виде с использованием графиков для наглядности.



Рисунок 19 – Домашняя страница

Задачи отображаются в виде списка с возможностью увидеть дату выполнения для некоторых категорий задач. Так же на странице отображаются два графика: круговой график, показывающий прогресс выполнения задач на сегодня, и столбчатый график, отображающий распределение задач по дням недели за последнюю неделю. Для отображения графиков используется библиотека react-chartjs-2.

2.10.5 Реализация функционала для Todo List

Компонент TodoList представляет собой экран списка задач в приложении, который позволяет пользователям добавлять, удалять и отмечать задачи как выполненные.

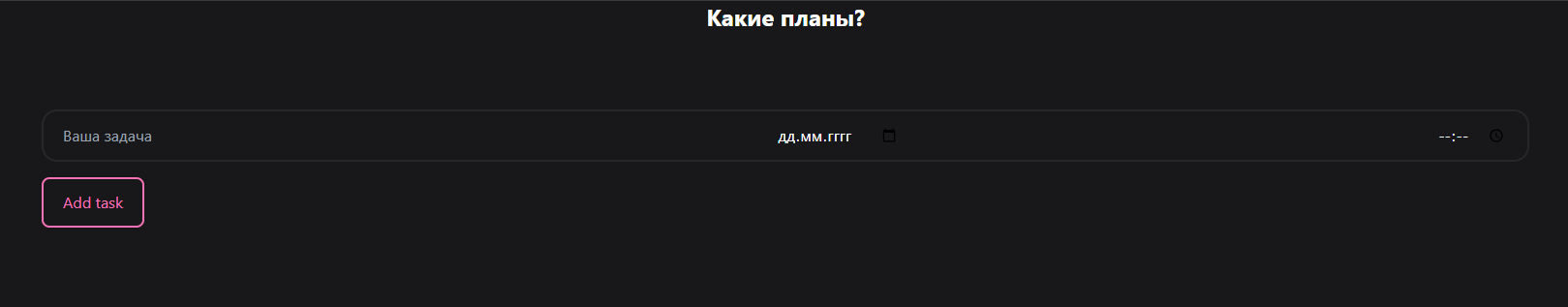


Рисунок 20 – Страница Todo List

При загрузке страницы выполняется запрос к Firestore для получения всех задач, связанных с текущим пользователем. Эти данные затем сохраняются в состоянии компонента.

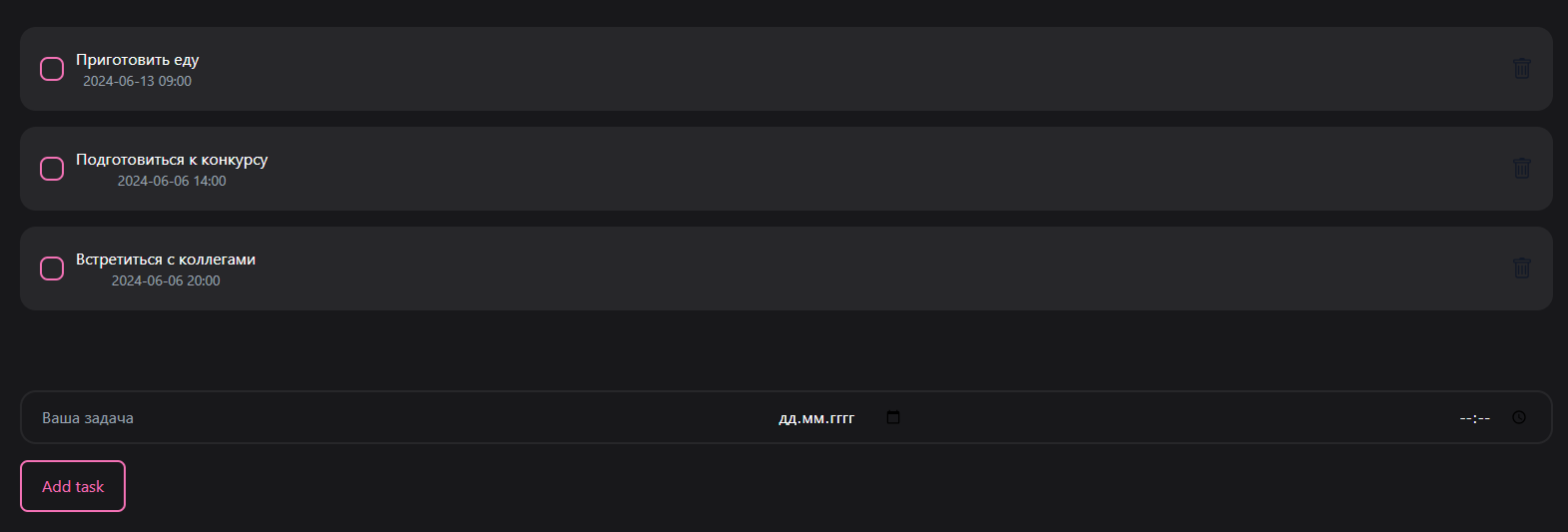


Рисунок 21 – Загруженные задачи пользователя

Пользователи могут добавить новую задачу с помощью компонента “CreateTodoField” (см. Приложение 9), который передает функцию “addTodo” для создания новой задачи в базу данных и обновления локального состояния.

Для добавления задачу необходимо ввести название задачи, дату и время.

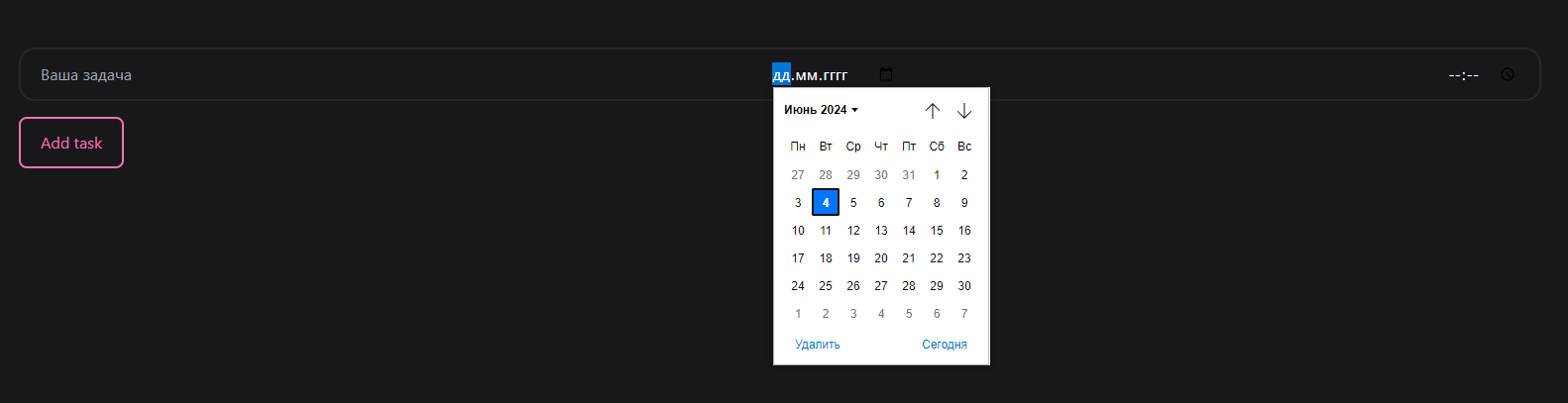


Рисунок 22 – Добавление новой задачи

Пользователи могут отметить задачу как выполненную с помощью функции changeTodo, которая обновляет состояние задачи “isCompleted” в базе данных и в локальном состоянии. Выглядит это следующим образом:

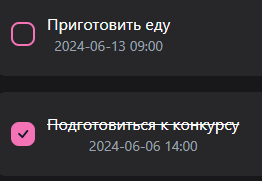


Рисунок 23 – Отметка выполненной задачи

Так же есть возможность удалить задачу, нажав на иконку удаления, которая вызывает функцию “removeTodo” для удаления задачи из базы данных и обновляет локальное состояние.

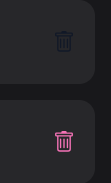


Рисунок 24 – Кнопка удаления задачи

2.10.6 Реализация функционала для Calendar

Компонент “TodoCalendar” (см. Приложение 10) является календарной страницей, где пользователи могут просматривать свои задачи, распределённые по датам, и редактировать или удалять их, за счет модального окна, появляющегося при нажатии на дату. Для работы с календарем используется библиотеки react-calendar для его отображения и date-fns для работы с датами.

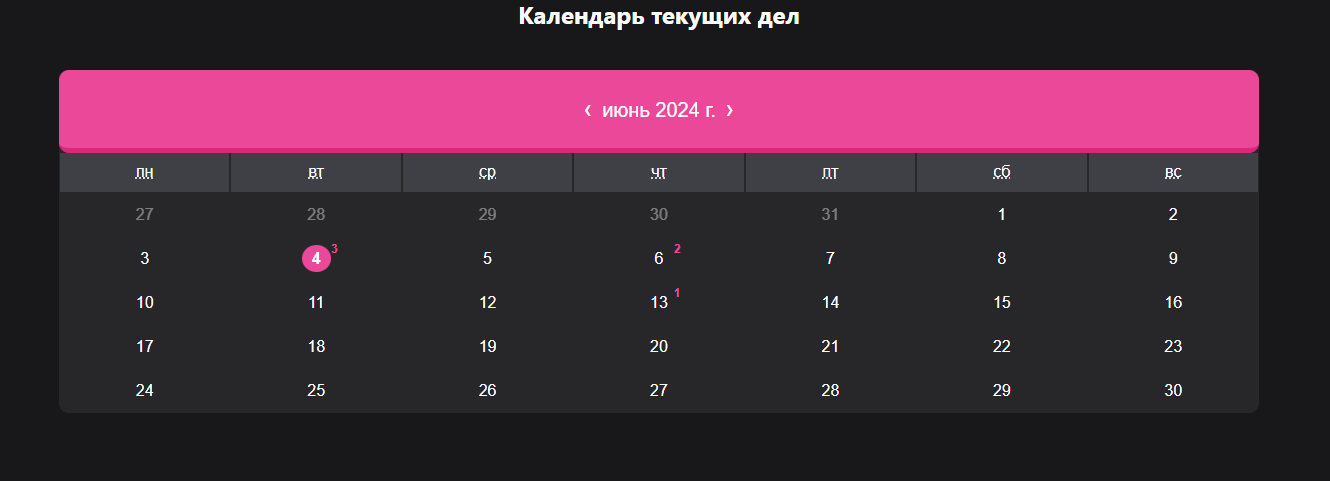


Рисунок 25 – Страница календаря задач

“TodoCalendar” интегрирует дочерним компонентом “CalendarPage“, который отображает задачи пользователя в виде интерактивного календаря. (см. Приложение 11). В компоненте есть функция “getTileContent” отвечающая за отображение содержимого внутри каждой ячейки календаря. Если для даты существуют задачи, в ячейке отображается индикатор с количеством задач. Индикатор отображается только для будущих дат и сегодняшнего дня.

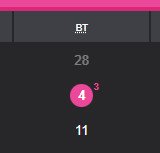


Рисунок 26 - Индикатор с количеством задач

Функция “handleDayClick” срабатывает при клике на день в календаре. Она отфильтровывает задачи, относящиеся к выбранному дню, и открывает модальное окно с этими задачами, если они существуют.

За модальное окно отвечает дочерний компонент “TaskModal” (см. Приложение 12). Он принимает несколько пропсов: isOpen для управления видимостью модального окна, onRequestClose для закрытия модального окна, tasks для передачи списка задач, относящихся к выбранному дню, onEdit для обработки изменений задач и onDelete для обработки удаления задач.

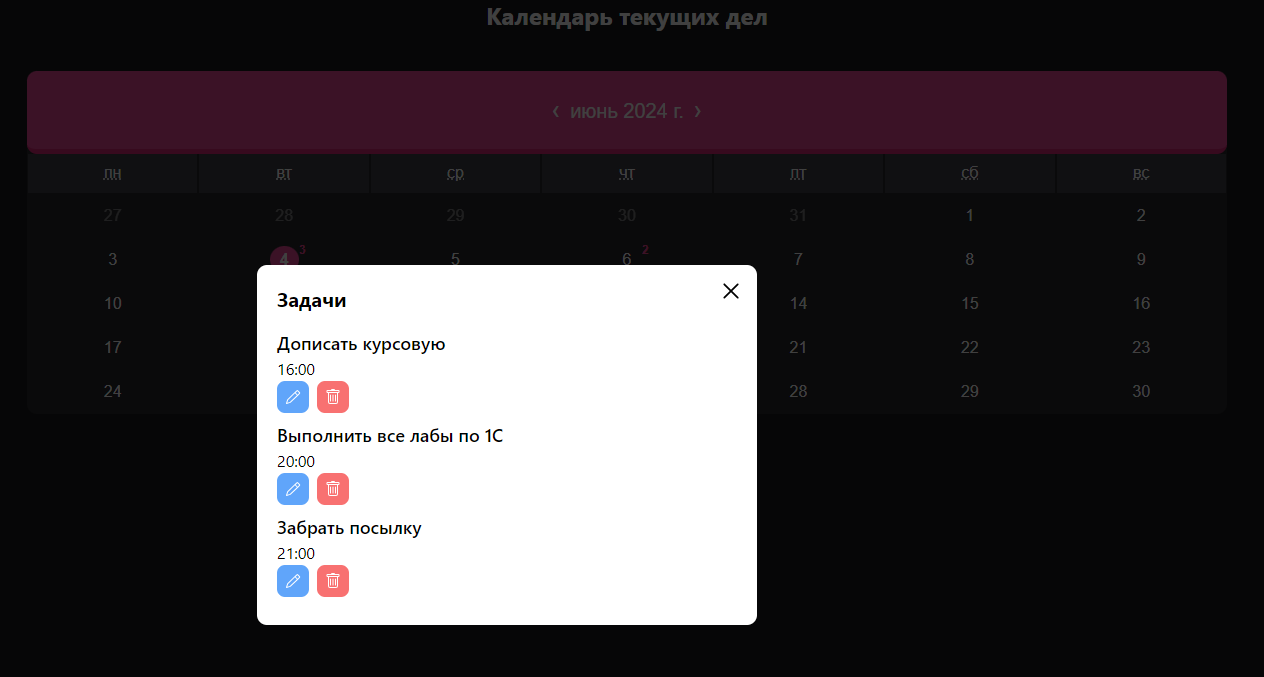


Рисунок 27 – Модальное окно

В модальном окне помимо просмотра задач, можно ещё изменить название задачи и удалить. При редактировании задачи через модальное окно, функция onEdit обновляет состояние selectedTasks, чтобы отразить изменения.

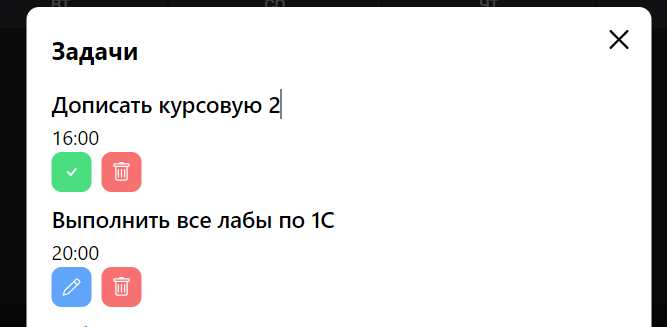


Рисунок 28 – редактирование задачи

При удалении задачи через модальное окно, функция onDelete обновляет состояние selectedTasks, удаляя соответствующую задачу.

На данном этапе разработка приложения завершилась. Хотя текущий проект предоставляет базовый функционал для управления задачами, существует множество возможностей для его дальнейшего улучшения и расширения.

# **Заключение**

Целью курсовой работы стало разработка системы управления задачами, используя современные веб-технологий, такие как React JS для клиентской части, Firebase для управления базой данных и Tailwind CSS для стилизации элементов интерфейса.

Для достижения этой цели был поставлен ряд задач:

1. Проанализировать существующие решения и определить основные требования к системе управления задачами.
2. Разработать клиентскую часть на React JS с использованием Firebase для управления базой данных и Tailwind CSS для стилизации элементов интерфейса
3. Реализовать основные функциональные возможности задачника: создание, изменение, удаление задач и фильтрация задач по различным критериям.
4. Оценить экономическую эффективность разработки и использования задачника.

В первой главе было выяснено, какие наиболее востребованные функции понадобятся в разработке и поставлены основные требования к системе управления задачами. Из чего следует, что удобный и интуитивно понятный интерфейс с интеграцией с календарем и гибкой системой фильтрации являются ключевыми аспектами в разработке и проектировании современной системы для управления задачами.

Во второй главе подробно описаны этапы создания задачника, включая разработку клиентскую часть и управления базой данных. А также была дана оценка экономической эффективности разработки и использования задачника.

В процессе выполнения курсовой работы были достигнуты все поставленные цели и решены основные задачи.

# **Список использованных источников**

1. docs/introducing-jsx [электронный ресурс] — Режим доступа. - URL: https://legacy.reactjs.org/docs/introducing-jsx.html/ (дата обращения 23.05.2024)
2. Преимущества и недостатки React [электронный ресурс] — Режим доступа. - URL: https://kz.hexlet.io/blog/posts/biblioteka-react-review-article/ (дата обращения 29.02.2024)
3. Плюсы и минусы SPA перед многостраничным сайтом [электронный ресурс] — Режим доступа. - URL: https://sibdev.pro/blog/articles/chto-takoe-spa-prilozhenie-i-nuzhno-li-ono-vashemu-proektu/ (дата обращения 25.01.2024)
4. Преимущества Firebase [электронный ресурс] — Режим доступа. - URL: http://surl.li/ufexr (дата обращения 14.05.2024)
5. Get started with Tailwind CSS [электронный ресурс] — Режим доступа. - URL: https://tailwindcss.com/docs/installation (дата обращения 03.06.2024)

**Приложения**

**Приложение 1**

import { initializeApp } from 'firebase/app'

import { getFirestore } from 'firebase/firestore';

import { getAuth } from 'firebase/auth'

import { getStorage } from 'firebase/storage';

const firebaseConfig = {

    apiKey: 'AIzaSyDLlzV\_k2rXQdpYda5MpttmYPlSsNJnM3s',

    authDomain: 'auth-example-e2aee.firebaseapp.com',

    projectId: 'auth-example-e2aee',

    storageBucket: 'auth-example-e2aee.appspot.com',

    messagingSenderId: '418951460460',

    appId: '1:418951460460:web:954f0d151c5fdc326eca49'

}

const app = initializeApp(firebaseConfig);

const db = getFirestore(app);

const auth = getAuth(app);

const storage = getStorage(app);

export { db, auth, storage };

**Приложение 2**

const removeTodo = async (id) => {

    try {

      await deleteDoc(doc(db, 'tasks', id));

      setTodos(todos.filter((todo) => todo.id !== id));

    } catch (error) {

      console.error('Error removing document: ', error);

    }

};

const addTodo = async (newTask) => {

    try {

       const docRef = await addDoc(collection(db, 'tasks'), newTask);

       const updatedTodos = [...todos, { ...newTask, id: docRef.id }];

       setTodos(updatedTodos);

    } catch (error) {

       console.error('Error adding document: ', error);

    }

};

**Приложение 3**

const handleEditTask = async (task) => {

    const taskDoc = doc(db, 'tasks', task.id);

    await updateDoc(taskDoc, { title: task.title });

    setTodos((prevTodos) =>

      prevTodos.map((t) => (t.id === task.id ? { ...t, title: task.title } : t))

    );

};

const handleDeleteTask = async (taskId) => {

    const taskDoc = doc(db, 'tasks', taskId);

    await deleteDoc(taskDoc);

    setTodos((prevTodos) => prevTodos.filter((t) => t.id !== taskId));

};

**Приложение 4**

const tasksThisWeek = tasks.filter(task => {

     const today = new Date();

     const taskDate = new Date(task.date);

     const diffTime = Math.abs(taskDate - today);

     const diffDays = Math.ceil(diffTime / (1000 \* 60 \* 60 \* 24));

     return diffDays <= 7 && taskDate >= today;

});

const tasksToday = tasks.filter(task => task.date === new Date().toISOString().split('T')[0]);

const completedTasksToday = tasks.filter(task => task.isCompleted && task.date === new Date().toISOString().split('T')[0]);

const completedTasks = tasks.filter(task => task.isCompleted);

const upcomingTasks = tasks.filter(task => !task.isCompleted && !tasksThisWeek.includes(task) && !tasksToday.includes(task));

const overdueTasks = tasks.filter(task => new Date(task.date) < new Date() && !task.isCompleted);

**Приложение 5**

import React, { useState } from 'react';

import { signInWithEmailAndPassword } from 'firebase/auth';

import { auth } from '../../firebase';

import { toast } from 'react-toastify';

import { useNavigate } from 'react-router-dom';

const Login = () => {

  const [email, setEmail] = useState('');

  const [password, setPassword] = useState('');

  const [error, setError] = useState(null);

  const navigate = useNavigate();

  const handleSubmit = async (e) => {

    e.preventDefault();

    try {

      await signInWithEmailAndPassword(auth, email, password);

      toast.success('User logged in successfully');

      navigate('/');

    } catch (err) {

      setError(err.message);

      toast.error(err.message);

    }

  };

  return (

    <div className='mx-auto my-auto'>

      <div className="w-full max-w-md p-8 bg-zinc-800 rounded-lg shadow-xl">

        <h2 className="text-2xl font-bold mb-6 text-pink-400 text-center">Login</h2>

        {error && <p className="mb-4">{error}</p>}

        <form onSubmit={handleSubmit} className="space-y-4">

          <input

            type="email"

            value={email}

            onChange={(e) => setEmail(e.target.value)}

            placeholder="Email"

            className="w-full p-2 border border-gray-700 rounded focus:outline-none focus:border-pink-500"

          />

          <input

            type="password"

            value={password}

            onChange={(e) => setPassword(e.target.value)}

            placeholder="Password"

            className="w-full p-2 border border-gray-700 rounded focus:outline-none focus:border-pink-500"

          />

          <div className="flex justify-end mt-4">

            <button type="submit" className="py-2 px-5 bg-pink-500 text-white rounded hover:bg-pink-400 focus:outline-none focus:ring-2 focus:ring-pink-400 focus:ring-opacity-50">

              Login

            </button>

          </div>

        </form>

      </div>

    </div>

  );

};

export default Login;

**Приложение 6**

import React, { useState } from 'react';

import { createUserWithEmailAndPassword, updateProfile } from 'firebase/auth';

import { auth } from '../../firebase';

import { toast } from 'react-toastify';

import { useNavigate } from 'react-router-dom';

const Register = () => {

  const [email, setEmail] = useState('');

  const [password, setPassword] = useState('');

  const [name, setName] = useState('');

  const [error, setError] = useState(null);

  const navigate = useNavigate();

  const handleSubmit = async (e) => {

    e.preventDefault();

    try {

      const userCredential = await createUserWithEmailAndPassword(auth, email, password);

      const user = userCredential.user;

      await updateProfile(user, { displayName: name });

      toast.success('User registered successfully');

      navigate('/');

    } catch (err) {

      setError(err.message);

      toast.error(err.message);

    }

  };

  return (

    <div className='mx-auto my-auto'>

      <div className="w-full max-w-md p-8 bg-zinc-800 rounded-lg shadow-xl ">

        <h2 className="text-2xl font-bold mb-6 text-pink-400 text-center">Register</h2>

        {error && <p>{error}</p>}

        <form onSubmit={handleSubmit} className="space-y-4">

          <input

            type="text"

            value={name}

            onChange={(e) => setName(e.target.value)}

            placeholder="Name"

            className="w-full p-2 border border-gray-700 rounded focus:outline-none focus:border-pink-500"

          />

          <input

            type="email"

            value={email}

            onChange={(e) => setEmail(e.target.value)}

            placeholder="Email"

            className="w-full p-2 border border-gray-700 rounded focus:outline-none focus:border-pink-500"

          />

          <input

            type="password"

            value={password}

            onChange={(e) => setPassword(e.target.value)}

            placeholder="Password"

            className="w-full p-2 border border-gray-700 rounded focus:outline-none focus:border-pink-500"

          />

          <div className="flex justify-end mt-4">

            <button type="submit" className="py-2 px-5 bg-pink-500 text-white rounded hover:bg-pink-400 focus:outline-none focus:ring-2 focus:ring-pink-400 focus:ring-opacity-50">

              Register

            </button>

          </div>

        </form>

      </div>

    </div>

  );

};

export default Register;

**Приложение 7**

import React, { useEffect, useState } from 'react';

import { Link, useNavigate } from 'react-router-dom';

import { onAuthStateChanged, signOut } from 'firebase/auth';

import { auth } from '../../firebase';

import { GoChevronDown } from "react-icons/go";

const Navbar = () => {

    const [user, setUser] = useState(null);

    const [showMenu, setShowMenu] = useState(false);

    const navigate = useNavigate();

    useEffect(() => {

      const unsubscribe = onAuthStateChanged(auth, (currentUser) => {

        setUser(currentUser);

      });

      return () => unsubscribe();

    }, []);

    const handleLogout = async () => {

      await signOut(auth);

      setUser(null);

      navigate('/login');

    };

    const toggleMenu = () => {

      setShowMenu(!showMenu);

    };

    const closeMenu = () => {

      setShowMenu(false);

    };

    return (

      <nav className="bg-zinc-800 pt-5 w-60 overflow-auto h-screen">

        <ul className="space-y-3 text-white">

          {!user ? (

            <>

              <li>

                <Link to="/login" className="block py-2 px-5 rounded hover:bg-zinc-700 transition duration-300">Login</Link>

              </li>

              <li>

                <Link to="/register" className="block py-2 px-5 rounded hover:bg-zinc-700 transition duration-300">Register</Link>

              </li>

            </>

          ) : (

            <>

              <li className="text-center relative">

                <span>Здравствуйте, {user.displayName}!</span>

                <button

                  onClick={toggleMenu}

                  className="ml-4 shadow-sm text-sm font-medium text-white"

                >

                  <GoChevronDown />

                </button>

                {showMenu && (

                  <div className="absolute right-0 mt-2 w-35 bg-white shadow-lg rounded-md overflow-hidden z-10">

                    <Link to="/edit-profile" onClick={(e) => { e.preventDefault(); navigate('/edit-profile'); closeMenu(); }} className="block px-4 py-2 text-sm text-gray-700 hover:bg-pink-400 hover:text-white">

                      Редактировать

                    </Link>

                    <Link to="/" onClick={(e) => { e.preventDefault(); handleLogout(); closeMenu(); }} className="block px-4 py-2 text-sm text-gray-700 hover:bg-pink-400 hover:text-white">

                      Выйти

                    </Link>

                  </div>

                )}

              </li>

              <li>

                <Link to="/" className="block py-2 px-5 rounded hover:bg-zinc-700 transition duration-300">Home</Link>

              </li>

              <li>

                <Link to="/list" className="block py-2 px-5 rounded hover:bg-zinc-700 transition duration-300">Todo List</Link>

              </li>

              <li>

                <Link to="/grap" className="block py-2 px-5 rounded hover:bg-zinc-700 transition duration-300">Calendar</Link>

              </li>

            </>

          )}

        </ul>

      </nav>

    );

};

export default Navbar;

**Приложение 8**

import React, { useState, useEffect } from 'react';

import { db, auth } from '../../../firebase';

import { collection, query, where, onSnapshot } from 'firebase/firestore';

import { Pie } from 'react-chartjs-2';

import { Bar } from 'react-chartjs-2';

import 'chart.js/auto';

const Home = () => {

    const [tasks, setTasks] = useState([]);

    useEffect(() => {

        const fetchData = async () => {

            const user = auth.currentUser;

            if (user) {

                const q = query(collection(db, 'tasks'), where('userId', '==', user.uid));

                const unsubscribe = onSnapshot(q, (querySnapshot) => {

                    const tasksData = [];

                    querySnapshot.forEach((doc) => {

                        tasksData.push({ ...doc.data(), id: doc.id });

                    });

                    setTasks(tasksData);

                });

                return () => unsubscribe();

            }

        };

        fetchData();

    }, []);

    const tasksThisWeek = tasks.filter(task => {

        const today = new Date();

        const taskDate = new Date(task.date);

        const diffTime = Math.abs(taskDate - today);

        const diffDays = Math.ceil(diffTime / (1000 \* 60 \* 60 \* 24));

        return diffDays <= 7 && taskDate >= today;

    });

    const tasksToday = tasks.filter(task => task.date === new Date().toISOString().split('T')[0]);

    console.log(new Date().toISOString().split('T')[0]);

    console.log(new Date().toISOString());

    const completedTasksToday = tasks.filter(task => task.isCompleted && task.date === new Date().toISOString().split('T')[0]);

    const completedTasks = tasks.filter(task => task.isCompleted);

    const upcomingTasks = tasks.filter(task => !task.isCompleted && !tasksThisWeek.includes(task) && !tasksToday.includes(task));

    const overdueTasks = tasks.filter(task => new Date(task.date) < new Date() && !task.isCompleted);

    const renderTasks = (tasks) => (

        <ul className="text-left mt-2">

            {tasks.map(task => (

                <li key={task.id} className="text-lg">

                    {task.title}

                </li>

            ))}

        </ul>

    );

    const renderTasksWithDates = (tasks) => (

        <ul className="text-left mt-2">

            {tasks.map(task => (

                <li key={task.id} className="text-lg flex justify-around">

                    <span className='w-2/3'>{task.title}</span>

                    <span className="text-pink-400">

                        {new Date(task.date).toLocaleDateString('ru-RU', { month: 'long', day: 'numeric' })}

                    </span>

                </li>

            ))}

        </ul>

    );

    const progressData = {

        labels: ['Выполненные', 'Осталось'],

        datasets: [{

            data: [completedTasksToday.length, tasksToday.length - completedTasksToday.length],

            backgroundColor: ['#4CAF50', '#FF6384'],

            hoverBackgroundColor: ['#66BB6A', '#FF6384']

        }]

    };

    const labels = [];

    const today = new Date();

    for (let i = 0; i < 7; i++) {

        const date = new Date(today);

        date.setDate(today.getDate() + i);

        labels.push(date.getDate());

    }

    const weeklyTaskData = {

        labels: labels,

        datasets: [

            {

                label: 'Количество задач на неделю по дням',

                backgroundColor: '#ec4899',

                borderColor: '#9d174d',

                borderWidth: 1,

                hoverBackgroundColor: '#f472b6',

                hoverBorderColor: '#f472b6',

                data: []

            }

        ]

    };

    labels.forEach(day => {

        const dayTasks = tasksThisWeek.filter(task => new Date(task.date).getDate() === day);

        weeklyTaskData.datasets[0].data.push(dayTasks.length);

    });

    return (

        <div className="dashboard text-white w-4/5 mx-auto">

            <h2 className="text-2xl font-bold mb-10 text-center">Домашняя страница</h2>

            <div className="flex">

                <div className="bg-zinc-800 p-4 rounded-lg shadow-lg w-1/2 text-center">

                    <h3 className="text-xl mb-2">Задачи на сегодняшний день</h3>

                    <p className="text-3xl text-pink-400">{tasksToday.length}</p>

                    <div className='flex justify-around'>

                        {renderTasks(tasksToday)}

                        {/\* <p className="text-3xl">{completedTasksToday.length}</p> \*/}

                        <div className="w-48 h-48">

                            <Pie data={progressData} />

                        </div>

                    </div>

                </div>

                <div className="bg-zinc-800 ml-10 p-4 rounded-lg shadow-lg w-1/3 text-center">

                    <h3 className="text-xl mb-2">Задачи на неделю</h3>

                    {renderTasksWithDates(tasksThisWeek)}

                </div>

            </div>

            <div className='flex'>

                <div className='mt-10 flex w-1/2'>

                    <div className='w-1/2'>

                        <div className="bg-zinc-800 p-4 rounded-lg shadow-lg text-center">

                            <h3 className="text-xl mb-2">Выполненные задачи</h3>

                            <p className="text-3xl text-pink-400">{completedTasks.length}</p>

                            {renderTasks(completedTasks)}

                        </div>

                        <div className="bg-zinc-800 p-4 rounded-lg shadow-lg text-center mt-10">

                            <h3 className="text-xl mb-2">Просроченные задачи</h3>

                            <p className="text-3xl">{overdueTasks.length}</p>

                            {renderTasks(overdueTasks)}

                        </div>

                    </div>

                    <div className="bg-zinc-800 p-4 rounded-lg shadow-lg w-1/2 text-center ml-10">

                        <h3 className="text-xl mb-2">Предстоящие задачи</h3>

                        <p className="text-3xl">{upcomingTasks.length}</p>

                        {renderTasksWithDates(upcomingTasks)}

                    </div>

                </div>

                <div className='w-1/3 ml-10 mt-24'>

                <Bar data={weeklyTaskData} />

                </div>

            </div>

        </div>

    );

};

export default Home;

**Приложение 9**

import { useState } from 'react';

import { db, auth } from '../../../../firebase';

import { collection, addDoc } from 'firebase/firestore';

import { v4 as uuidv4 } from 'uuid';

const CreateTodoField = ({ addTodo }) => {

  const [title, setTitle] = useState('');

  const [date, setDate] = useState('');

  const [time, setTime] = useState('');

  const [isAddingTask, setIsAddingTask] = useState(false);

  const handleSubmit = async (e) => {

    e.preventDefault();

    const user = auth.currentUser;

    if (user) {

      try {

        const newTask = {

          title,

          date,

          time,

          isCompleted: false,

          userId: user.uid,

        };

        await addTodo(newTask);

        setTitle('');

        setDate('');

        setTime('');

      } catch (error) {

        console.error('Error adding document: ', error);

      }

    } else {

      console.error('User is not authenticated');

    }

  };

  return (

    <form onSubmit={handleSubmit}>

      <div className='flex items-center justify-between mb-4 rounded-2xl border-zinc-800 border-2 px-5 py-3 w-full mt-20'>

        <input className='bg-transparent border-none outline-none' type="text" value={title} onChange={(e) => setTitle(e.target.value)} placeholder="Ваша задача" />

        <input className='bg-transparent border-none outline-none' type="date" value={date} onChange={(e) => setDate(e.target.value)} />

        <input className='bg-transparent border-none outline-none' type="time" value={time} onChange={(e) => setTime(e.target.value)} />

      </div>

      <button className='border-2 rounded-lg border-pink-400 text-pink-400 pl-5 pr-5 pt-3 pb-3' type="submit">Add task</button>

    </form>

  );

};

export default CreateTodoField;

**Приложение 10**

import React, { useState, useEffect } from 'react';

import { db, auth } from '../../../firebase';

import { collection, query, where, onSnapshot, updateDoc, doc, deleteDoc } from 'firebase/firestore';

import CalendarPage from './item/CalendarPage';

const TodoCalendar = () => {

    const [todos, setTodos] = useState([]);

    useEffect(() => {

        const fetchData = async () => {

            const user = auth.currentUser;

            if (user) {

                const q = query(collection(db, 'tasks'), where('userId', '==', user.uid));

                const unsubscribe = onSnapshot(q, (querySnapshot) => {

                    const tasksData = [];

                    querySnapshot.forEach((doc) => {

                        tasksData.push({ ...doc.data(), id: doc.id });

                    });

                    setTodos(tasksData);

                });

                return () => unsubscribe();

            }

        };

        fetchData();

    }, []);

    const handleEditTask = async (task) => {

        const taskDoc = doc(db, 'tasks', task.id);

        await updateDoc(taskDoc, { title: task.title });

        setTodos((prevTodos) =>

            prevTodos.map((t) => (t.id === task.id ? { ...t, title: task.title } : t))

        );

    };

    const handleDeleteTask = async (taskId) => {

        const taskDoc = doc(db, 'tasks', taskId);

        await deleteDoc(taskDoc);

        setTodos((prevTodos) => prevTodos.filter((t) => t.id !== taskId));

    };

    return (

        <div className="mx-auto">

            <h1 className="text-white text-2xl font-bold text-center mb-10">Календарь текущих дел</h1>

            <CalendarPage

                todos={todos}

                onEditTask={handleEditTask}

                onDeleteTask={handleDeleteTask}

            />

        </div>

    );

};

export default TodoCalendar;

**Приложение 11**

import React, { useState } from 'react';

import Calendar from 'react-calendar';

import { format, isBefore, startOfDay } from 'date-fns';

import TaskModal from './TaskModal';

import '../../../../index.css';

const CalendarPage = ({ todos, onEditTask, onDeleteTask }) => {

    const [selectedTasks, setSelectedTasks] = useState([]);

    const [isModalOpen, setIsModalOpen] = useState(false);

    const handleDayClick = (date) => {

        const formattedDate = format(date, 'yyyy-MM-dd');

        const dayTasks = todos.filter(todo => todo.date === formattedDate);

        if (dayTasks.length > 0) {

            setSelectedTasks(dayTasks);

            setIsModalOpen(true);

        }

    };

    const getTileContent = ({ date, view }) => {

        const todayStart = startOfDay(new Date());

        if (view === 'month') {

            const formattedDate = format(date, 'yyyy-MM-dd');

            const dayTasks = todos.filter(todo => todo.date === formattedDate);

            if (!isBefore(date, todayStart)) {

                if (dayTasks.length > 0) {

                    return (

                        <div className="relative">

                            <div className="dot">

                                {dayTasks.length}

                            </div>

                        </div>

                    );

                }

            }

        }

        return null;

    };

    return (

        <div className="calendar-container text-white">

            <Calendar

                tileContent={getTileContent}

                onClickDay={handleDayClick}

            />

            <TaskModal

                isOpen={isModalOpen}

                onRequestClose={() => setIsModalOpen(false)}

                tasks={selectedTasks}

                onEdit={(task) => {

                    onEditTask(task);

                    setSelectedTasks((prevTasks) =>

                        prevTasks.map((t) => (t.id === task.id ? { ...t, title: task.title } : t))

                    );

                }}

                onDelete={(taskId) => {

                    onDeleteTask(taskId);

                    setSelectedTasks((prevTasks) => prevTasks.filter((t) => t.id !== taskId));

                }}

            />

        </div>

    );

};

export default CalendarPage;

**Приложение 12**

import React, { useState } from 'react';

import Modal from 'react-modal';

import '../../../../index.css';

import { GoX } from "react-icons/go";

import { GoPencil } from "react-icons/go";

import {BsTrash} from 'react-icons/bs'

import {BsCheck} from 'react-icons/bs'

const TaskModal = ({ isOpen, onRequestClose, tasks, onEdit, onDelete }) => {

    const [editTaskId, setEditTaskId] = useState(null);

    const [editTaskTitle, setEditTaskTitle] = useState('');

    const handleEditClick = (task) => {

        setEditTaskId(task.id);

        setEditTaskTitle(task.title);

    };

    const handleSaveClick = (task) => {

        onEdit({ ...task, title: editTaskTitle });

        setEditTaskId(null);

        setEditTaskTitle('');

    };

    return (

        <Modal

            isOpen={isOpen}

            onRequestClose={onRequestClose}

            contentLabel="Task Details"

            className="modal"

            overlayClassName="modal-overlay"

        >

            <h2 className="text-xl font-bold mb-4">Задачи</h2>

            <ul>

                {tasks.map(task => (

                    <li key={task.id} className="mb-2">

                        {editTaskId === task.id ? (

                            <input

                                type="text"

                                value={editTaskTitle}

                                onChange={(e) => setEditTaskTitle(e.target.value)}

                                className="text-lg font-semibold rounded"

                            />

                        ) : (

                            <h3 className="text-lg font-semibold">{task.title}</h3>

                        )}

                        <p>{task.time}</p>

                        {editTaskId === task.id ? (

                            <button

                                onClick={() => handleSaveClick(task)}

                                className="mr-2 bg-green-400 text-white p-2 rounded-lg"

                            >

                                <BsCheck />

                            </button>

                        ) : (

                            <button

                                onClick={() => handleEditClick(task)}

                                className="mr-2 bg-blue-400 text-white p-2 rounded-lg"

                            >

                                <GoPencil />

                            </button>

                        )}

                        <button

                            onClick={() => onDelete(task.id)}

                            className="bg-red-400 text-white p-2 rounded-lg"

                        >

                            <BsTrash />

                        </button>

                    </li>

                ))}

            </ul>

            <button

                onClick={onRequestClose}

                className="absolute top-3 right-3"

            >

                <GoX className="h-7 w-7"/>

            </button>

        </Modal>

    );

};

export default TaskModal;