Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные системы и сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

**ВЭБ – СЛУЖБА УПРАВЛЕНИЯ СОБЫТИЯМИ КАЛЕНДАРЯ**

БГУИР КП 6-05-0612-01-001 ПЗ

Студент Бельский М.Д.

Руководитель Болтак С.В.

Минск 2025

СОДЕРЖАНИЕ

[Введение 3](#_Toc85934173)

[1 Анализ предметной области 4](#_Toc85934174)

[1.1 Обзор аналогов 4](#_Toc85934175)

[1.2 Постановка задачи 6](#_Toc85934176)

[2 Проектирование программного средства 7](#_Toc85934177)

[2.1 Структура программы 7](#_Toc85934178)

[2.2 Проектирование интерфейса программного средства 7](#_Toc85934179)

[2.3 Проектирование функционала программного средства 10](#_Toc85934180)

[3 Разработка программного средства 13](#_Toc85934181)

[3.1 Выбор технологий и инструментов](#_Toc85934182) 13

[3.2 Реализация клиентской части 14](#_Toc85934183)

[3.3 Работа с базой данных 18](#_Toc85934184)

[3.4 Аутентификация и авторизация 19](#_Toc85934185)

[4 Тестирование программного средства 21](#_Toc85934187)

[5 Руководство пользователя 22](#_Toc85934188)

[5.1 Интерфейс программного средства 22](#_Toc85934189)

[5.2 Управление программным средством 25](#_Toc85934190)

[Заключение 25](#_Toc85934191)

[Список использованных источников 27](#_Toc85934192)

[Приложение А. Исходный код программы 28](#_Toc85934196)

# ВВЕДЕНИЕ

Современные информационные технологии активно проникают во все сферы жизни человека, облегчая выполнение повседневных задач. Одной из таких задач является планирование времени — управление личным и рабочим расписанием, контроль за предстоящими событиями и напоминания о них. Для этого активно используются электронные календари, интегрированные с различными веб-сервисами и приложениями.

С ростом популярности облачных решений и мобильных устройств появилась потребность в разработке универсальных веб-служб, обеспечивающих доступ к календарям из любой точки мира и с любого устройства. Такие веб-службы позволяют пользователю создавать, редактировать и удалять события, получать уведомления, а также синхронизировать данные с другими сервисами и приложениями.

В рамках данного проекта рассматривается создание веб-службы для работы с мероприятиями в календаре. Эта служба предоставляет программный интерфейс (API), с помощью которого можно централизованно управлять событиями: от добавления новых встреч до просмотра расписания на любой период. Основное внимание уделяется реализации REST-архитектуры, обеспечению безопасности, масштабируемости и удобству взаимодействия с клиентскими приложениями.

Для реализации проекта используются современные технологии и инструменты:

ASP.NET Core — кроссплатформенный веб-фреймворк от Microsoft, предназначенный для создания высокопроизводительных и масштабируемых веб-приложений и API;

REST API — архитектурный стиль взаимодействия между клиентом и сервером, основанный на использовании стандартных HTTP-методов;

JWT (JSON Web Token) — стандарт безопасной передачи информации между участниками, применяемый для аутентификации и авторизации пользователей;

Куки (Cookies) — используются для безопасного хранения refresh-токенов на стороне клиента, что обеспечивает удобство и защиту при повторной авторизации;

Entity Framework Core — ORM-технология, позволяющая работать с базой данных на уровне объектов;

PostgreSQL — мощная объектно-реляционная система управления базами данных, применяемая для хранения информации о мероприятиях, пользователях и связанных данных.

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## Обзор аналогов

На сегодняшний день существует множество веб-сервисов и API, предназначенных для работы с календарями и событиями. Каждый из них предоставляет различные возможности по созданию, хранению, отображению и уведомлению о запланированных мероприятиях. Рассмотрим наиболее популярные решения, представляющие интерес как для конечных пользователей, так и для разработчиков интеграционных приложений.

Одним из наиболее известных и широко используемых решений является Google Calendar API. Этот сервис предоставляет мощный и гибкий интерфейс для взаимодействия с календарями пользователей. С помощью API можно создавать, редактировать и удалять события, устанавливать напоминания, запрашивать списки мероприятий по временным промежуткам, выполнять поиск и сортировку. Большое внимание уделяется безопасности — доступ к данным осуществляется через авторизацию OAuth 2.0. Также стоит отметить широкую поддержку SDK для разных языков программирования и богатую документацию. Внешний вид данного приложения представлен на рисунке 1.1

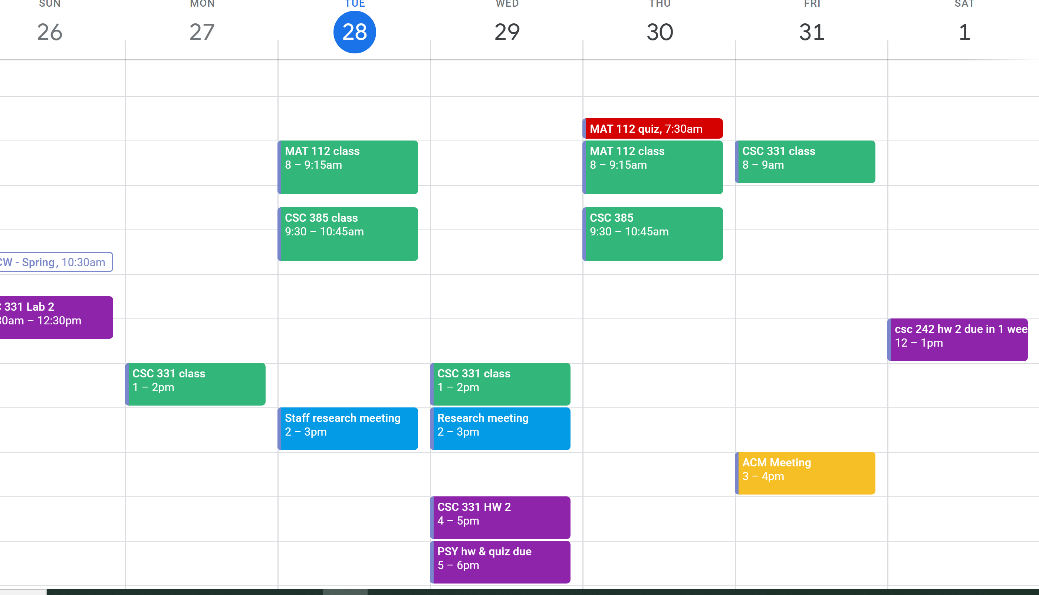


Рисунок 1.1 – Google Calendar

Ещё одним заметным представителем является Microsoft Graph API, предоставляющий доступ к календарям пользователей Microsoft Outlook и других сервисов Office 365. С помощью Graph API можно не только управлять событиями, но и организовывать встречи, отслеживать статусы участников, работать с временными зонами и учитывать рабочие календари. Microsoft делает ставку на корпоративную интеграцию, позволяя выстраивать сложные сценарии взаимодействия с другими компонентами экосистемы Microsoft.

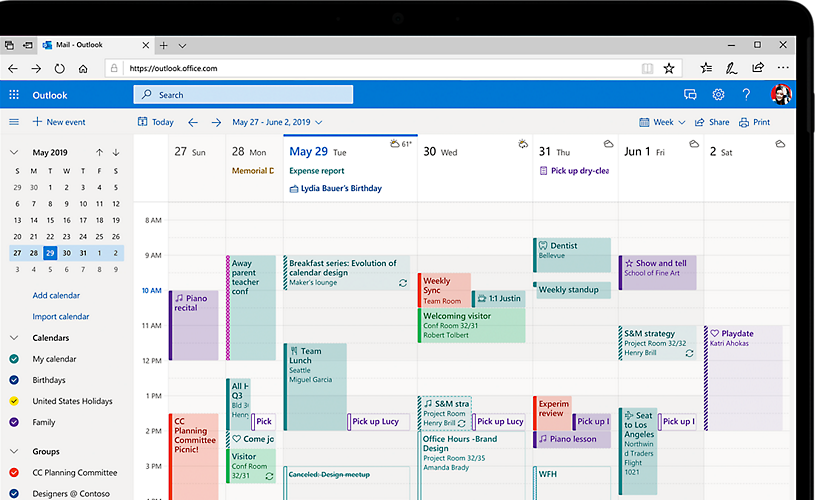


Рисунок 1.2 – Microsoft Outlook

Также стоит упомянуть Apple EventKit, который используется в устройствах Apple для взаимодействия с локальными и облачными календарями. Хотя EventKit ограничен платформами iOS и macOS, он предоставляет полноценный набор инструментов для работы с календарными данными, включая напоминания, события, повторяющиеся задачи и синхронизацию с iCloud.

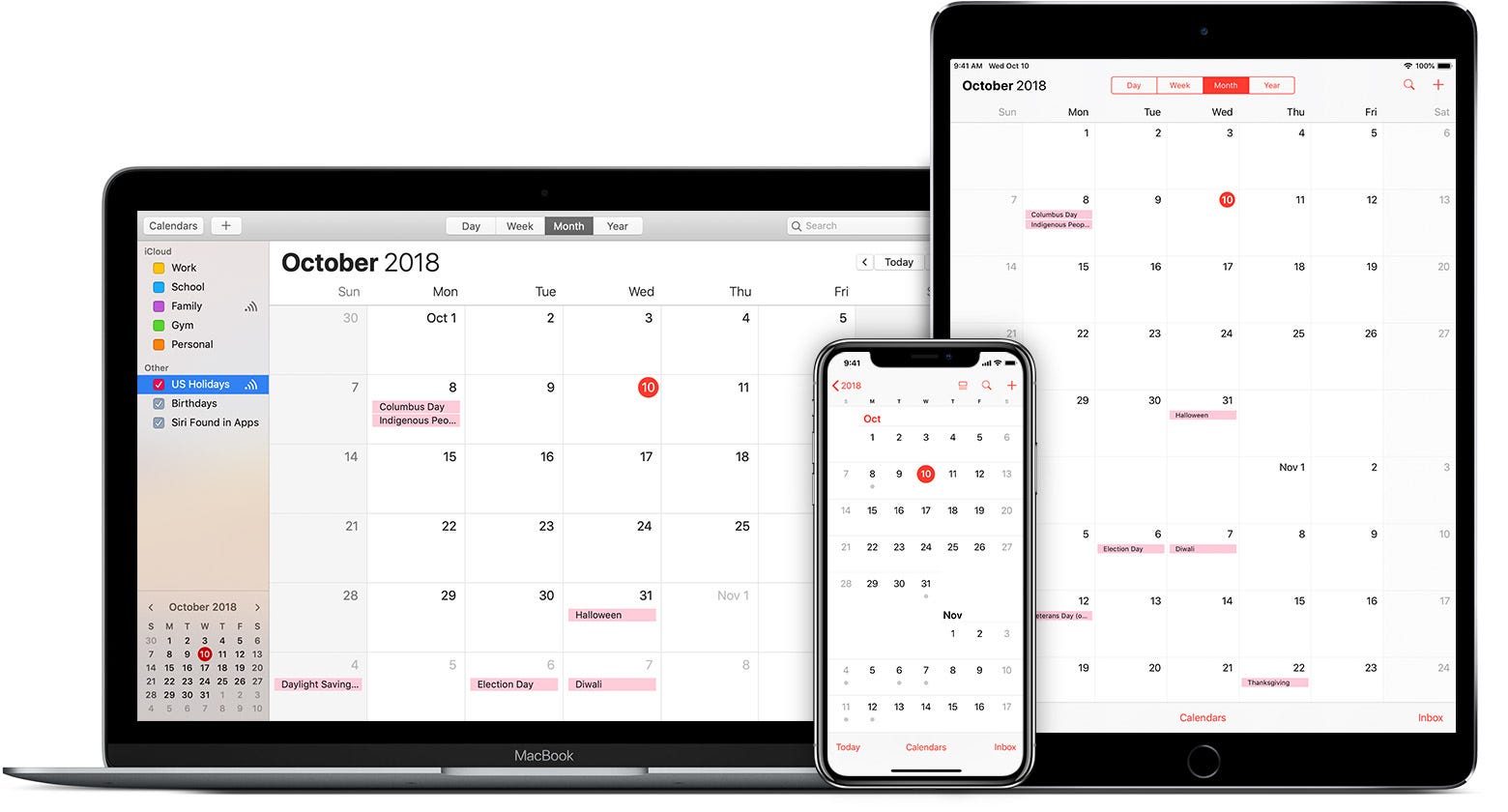


Рисунок 1.3 – Apple EventKit

В отличие от упомянутых решений, разрабатываемая веб-служба предлагает простой, удобный и не перегруженный большим набором функционала REST-интерфейс для управления событиями.

Она позволяет:

* добавлять, удалять и изменять события;
* выполнять выборку событий с сортировкой по дате;
* фильтровать события по заданным временным промежуткам;
* осуществлять постраничную загрузку данных для удобства отображения в интерфейсе;
* использовать API-ключ для передачи информации о событиях всех пользователей внешним приложениям с целью своевременной рассылки уведомлений о наступлении указанных событий.

Дополнительно в службе реализована система регистрации и авторизации пользователей с использованием JWT-токенов, хранящихся в HTTP-only cookies. Такой подход обеспечивает безопасную и удобную аутентификацию, защищённую от XSS-атак, и даёт возможность автоматически обновлять токен доступа при его истечении с помощью refresh-токена.

Несмотря на наличие более масштабных и универсальных решений, предлагаемая веб-служба ориентирована на конкретные прикладные сценарии, упрощая интеграцию с внешними системами и предоставляя ключевой функционал в легковесной и настраиваемой форме.

## Постановка задачи

В рамках данной курсовой работы планируется разработать веб-службу для управления событиями в пользовательском календаре. Сервис предоставляет REST API и позволяет создавать, редактировать, удалять и получать события с возможностью фильтрации и сортировки. Основной целью является создание надёжного и расширяемого решения, пригодного для интеграции с внешними системами, например, для уведомления пользователей о приближении запланированных мероприятий.

Основные функции, подлежащие реализации:

* добавление новых событий в календарь;
* удаление и изменение существующих событий;
* получение списка событий с возможностью:
  + сортировки по дате,
  + фильтрации по временным промежуткам,
  + постраничного вывода (пагинации);
* предоставление возможности внешним клиентским приложениям при помощи API-ключа запрашивать события пользователей за указанный период времени для последующей отправки уведомлений;
* авторизация и аутентификация пользователей с использованием JWT-токенов, передаваемых через HTTP-only cookies;
* защита доступа к данным пользователей с использованием уникальных API-ключей, выданных каждому зарегистрированному клиенту.

Веб-служба будет реализована с использованием следующих технологий:

* язык программирования C#;
* платформа ASP.NET Core с архитектурой Web API;
* реляционная база данных PostgreSQL для хранения информации о пользователях и событиях;
* Entity Framework Core в качестве ORM для удобства работы с базой данных.

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

## 2.1 Структура программы

Разрабатываемая веб-служба построена на модульной архитектуре, обеспечивающей разделение логики по ответственностям и упрощающей поддержку и расширение проекта. В рамках реализации предусмотрено использование следующих ключевых модулей:

* Program.cs – модуль начальной конфигурации приложения, регистрация сервисов, middleware и маршрутизации HTTP-запросов;
* Controllers – модуль, содержащий контроллеры ASP.NET Core Web API. Отвечает за обработку входящих HTTP-запросов, валидацию данных и возврат ответов клиенту;
* Repositories – модуль доступа к данным, реализующий шаблон Repository:
* EventRepository – доступ к событиям в базе данных;
* UserRepository – доступ к данным о пользователях;
* Models – модуль, содержащий описания сущностей базы данных (User, Event), а также DTO-классы для входных и выходных данных API;
* Database – модуль настройки подключения к базе данных PostgreSQL, миграций, инициализации схемы;
* Middleware – модуль, содержащий промежуточные компоненты, такие как обработка исключений, логирование или JWT-авторизация;
* Configuration – модуль, содержащий настройки приложения (параметры подключения к БД, JWT-настройки), получаемые из appsettings.json;

## 2.2 Проектирование интерфейса программного средства

В качестве основы интерфейса календаря будет использован минималистичный и современный стиль, направленный на простоту взаимодействия и визуальное удобство. Интерфейс включает календарную сетку, панель управления событиями, кнопки управления, а также кнопку выхода из аккаунта.

2.2.1 Главное окно

Главное окно приложения разделено на две основные части: левая часть включает календарную сетку текущего месяца, где пользователь может выбирать нужные даты, выделяя их визуально. В правой части отображается список событий, запланированных на выбранную дату, с возможностью добавления, редактирования или удаления этих событий. Также в верхней центральной части расположена кнопка в виде закрывающейся двери, предназначенная для выхода и завершения пользовательской сессии. Макет главного окна представлен на рисунке 2.1.

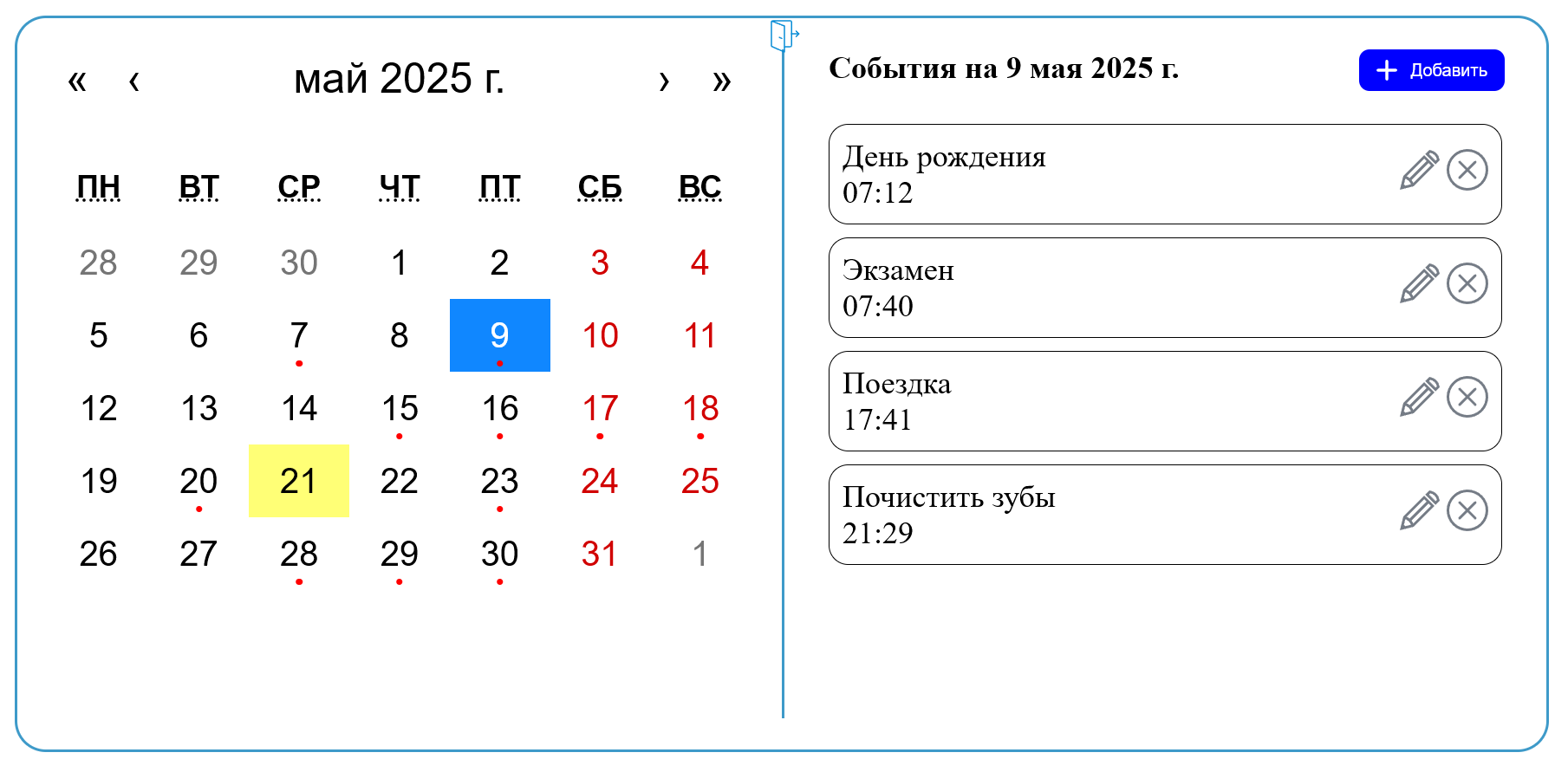


Рисунок 2.1 – Главное окно приложения

2.2.2 Элементы управления

Управление событиями осуществляется через несколько интерфейсных элементов. В верхней части главного окна находится календарная сетка, где представлены навигационные стрелочки: две позволяют переходить между месяцами, а две другие — между годами. Кнопка «Добавить событие» открывает окно для создания нового события с полями для ввода названия и времени начала. Рядом с каждым событием находятся кнопки редактирования и удаления: иконка карандаша предназначена для редактирования, а иконка крестика в кружке — для удаления события. Для завершения пользовательской сессии предусмотрена кнопка выхода, оформленная в виде закрывающейся двери, которая возвращает пользователя на экран авторизации. Внешний вид элементов управления веб-браузером представлен на рисунке 2.2.

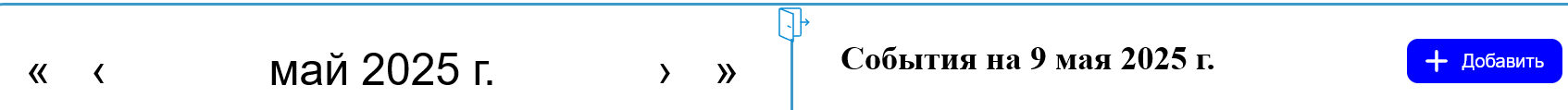




Рисунок 2.2 – Элементы управления приложением

2.2.5 Окно добавления события

Интерфейс окна для добавления нового события включает в себя две кнопки: одну для подтверждения добавления, другую — для отмены операции. Также в нем предусмотрены поля для ввода названия события, а также для выбора даты и времени. Значения по умолчанию для дат и времени будут установлены в текущие — что обеспечивает удобство и быстроту оформления. Визуальное оформление макета окна добавления закладок подробно показано на рисунке 2.3.

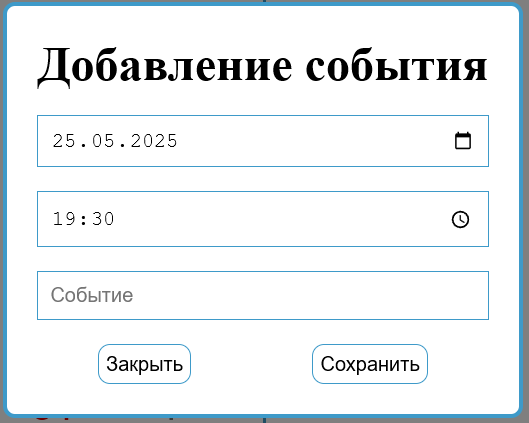


Рисунок 2.3 – Внешний макета окна добавления события

2.2.6 Окно для регистрации и входа

Окно регистрации пользователя включает в себя поля для ввода имени, почты и пароля, а также две кнопки — для регистрации и перехода ко входу. При нажатии на кнопку «Войти» пользователь перенаправляется на окно входа, где предусмотрены поля для ввода имени и почты, а также две кнопки — для входа и регистрации. Внешний вид макетов окон регистрации и входа представлен на рисунке 2.4.

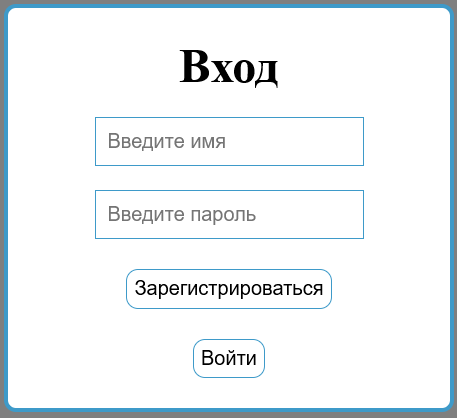
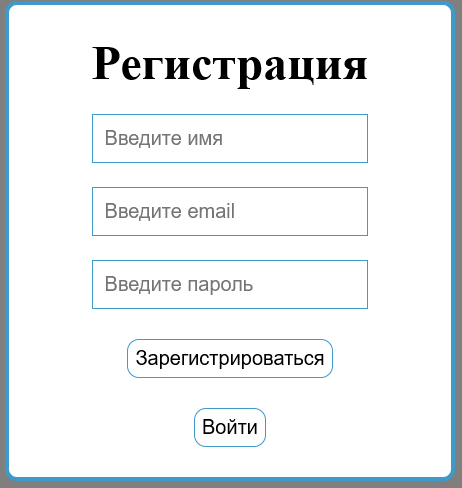


Рисунок 2.4 – Внешний макета окна регистрации и входа

## 2.3 Проектирование функционала программного средства

Грамотно поставленная задача и четко определенные алгоритмы — ключевые этапы в проектировании программного средства.

Приложение должно предоставлять пользователю базовый и интуитивно понятный функционал, необходимый для планирования и управления событиями. Минимальный набор функций включает:

* вход в аккаунт;
* просмотр событий по выбранной дате;
* добавление нового события;
* редактирование существующего события;
* удаление события;
* выход из аккаунта.

2.3.1 Добавление события

Добавление события будет происходит по нажатию кнопки «Добавить». После нажатия данной кнопки должно будет открыться окно, в котором пользователь сможет задать название нового события, определить дату и время, и добавить событие. При добавлении события будет вызываться метод AddEvent(). Он будет забирать новое событие из тела запроса. После добавления в список событий, он возвращает созданное событие клиенту. Блок-схема метода AddEvent() представлена на рисунке 2.5.



Рисунок 2.5 – Блок-схема метода AddEvent ()

2.3.2 Удаление закладки

Удаление события осуществляется по нажатию на иконку крестика в кружочке. При этом вызывается метод DeleteEvent(), который извлекает id события из строки запроса. Затем он находит соответствующее событие в общем списке по этому идентификатору и удаляет его. После успешного удаления метод отправляет уведомление об этом. Блок-схема работы метода DeleteEvent() представлена на рисунке 2.6.



Рисунок 2.6 – Блок-схема метода DeleteEvent ()

# РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

3.1 Выбор технологий и инструментов

Для реализации программного средства были выбраны современные и широко применяемые технологии, обеспечивающие надёжность, масштабируемость и удобство поддержки проекта. Архитектура приложения построена по клиент-серверной модели с использованием REST API, что позволяет легко разделять фронтенд и бэкенд и обеспечивает гибкость при взаимодействии между ними.

Frontend (пользовательский интерфейс)

Для клиентской части приложения был выбран фреймворк React. Он обеспечивает:

* Компонентный подход к построению интерфейса;
* Быструю и отзывчивую работу за счёт виртуального DOM;
* Возможность реализации динамического интерфейса (например, календаря, интерактивных форм и списков событий);
* Хорошую совместимость с библиотеками для маршрутизации и работы с формами.

Backend (серверная часть)

В качестве серверной платформы выбран ASP.NET Core на языке C#. Причины выбора:

* Высокая производительность и безопасность;
* Поддержка асинхронного программирования;
* Удобная реализация REST API;
* Хорошая интеграция с системами авторизации и базами данных.

Сервер реализует все основные функции:

* Обработка запросов от клиента (получение, добавление, редактирование, удаление событий);
* Управление пользователями и сессиями;
* Проверка прав доступа через JWT и Cookie.

База данных

Для хранения информации о событиях и пользователях используется PostgreSQL — надёжная реляционная СУБД, обладающая следующими преимуществами:

* Поддержка сложных SQL-запросов и индексов;
* Высокая производительность при работе с большими объёмами данных;
* Гибкость масштабирования и резервного копирования.

В базе данных хранятся:

* Данные пользователей (логин, хэш пароля);
* События, связанные с определённой датой и пользователем;
* Метаданные (дата создания, дата изменения и пр.).

Технологии взаимодействия

Обмен данными между клиентом и сервером реализован через REST API. Все запросы и ответы передаются в формате JSON, что обеспечивает простоту обработки на обоих концах.

Для аутентификации и авторизации используется связка JWT (JSON Web Token) и Cookie:

* При успешной авторизации сервер выдаёт access token и refresh token;
* Токены хранятся в cookie с флагами безопасности (HttpOnly, Secure);
* При истечении срока действия access-токена происходит автоматическое обновление с помощью refresh-токена.

Такой подход обеспечивает безопасную и масштабируемую работу приложения с учётом аутентификации.

**3.2 Реализация клиентской части**

Клиентская часть программного средства разработана с использованием библиотеки React. Интерфейс построен по компонентному принципу, что обеспечивает удобство масштабирования, повторного использования и тестирования компонентов.

**3.2.1** Структура интерфейса

Пользовательский интерфейс включает следующие основные компоненты:

* Календарь – отображает текущий месяц с возможностью навигации по месяцам и годам. При нажатии на дату отображаются все события, связанные с этой датой.
* Список событий – отображается сбоку или внизу календаря; содержит краткую информацию о каждом событии (время, название, описание).
* Кнопки управления:
  + Добавить событие – открывает модальное окно для создания нового события;
  + Редактировать событие – позволяет изменить существующее событие;
  + Удалить событие – удаляет выбранное событие;
  + Выйти – завершает текущую сессию пользователя.
  + Форма авторизации и регистрации – отдельные компоненты, доступные до входа в систему.

**3.2.2** Работа с сервером

Взаимодействие с сервером осуществляется с помощью REST API. Все запросы выполняются через библиотеку. Реализованы следующие основные вызовы API:



Рисунок 3.1 – Получение списка событий по дате



Рисунок 3.2 – Добавление нового события и изменение события

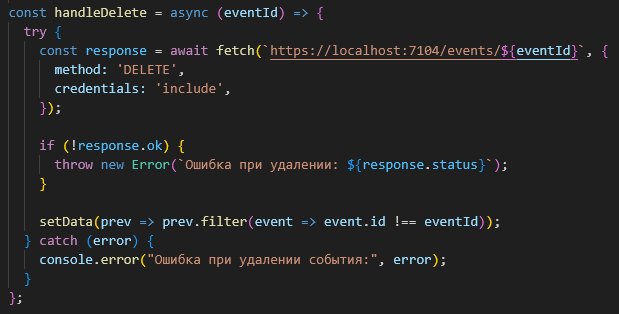


Рисунок 3.3 – Удаление события

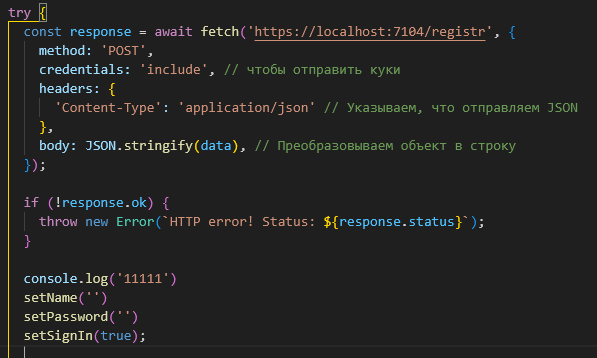


Рисунок 3.4 – Регистрация пользователя

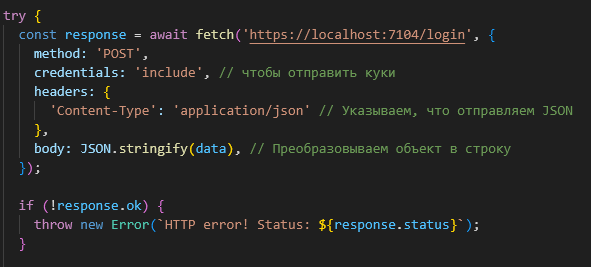


Рисунок 3.5 – Вход пользователя



Рисунок 3.6 – Обновление access token с использованием refresh token

**3.2.3** Обработка токенов и авторизации

После входа пользователя access token сохраняется в cookie, и он автоматически подставляется в заголовки всех последующих запросов. При истечении срока действия токена клиент выполняет запрос на обновление токена с использованием refresh token.

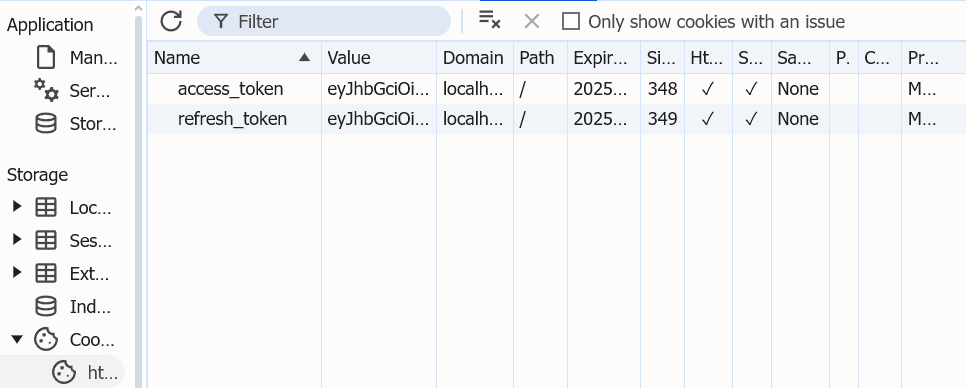


Рисунок 3.7 – Access token и refresh token в Cookie

## 3.3 Работа с базой данных

Для хранения и управления данными в программном средстве используется реляционная база данных PostgreSQL. С её помощью осуществляется долговременное хранение информации о пользователях, событиях календаря и связанных данных.

**3.3.1** Структура базы данных

База данных спроектирована с учётом принципов нормализации, что позволяет избежать избыточности данных и упростить работу с ними. Основные таблицы:

* + Users — содержит информацию о пользователях (ID, логин, хеш пароля, email и др.).
  + Events — таблица для хранения событий (ID, заголовок, описание, дата, время начала и окончания, ID пользователя).

**3.3.2** Взаимодействие с базой данных

Во взаимодействии backend-приложения на ASP.NET Core с PostgreSQL использовалась технология Entity Framework Core:

* Модели данных (C# классы) отражают структуру таблиц.
* Для доступа к базе используется контекст данных DbContext.
* Операции чтения, добавления, изменения и удаления выполняются через LINQ-запросы.
* Миграции применяются для управления версией схемы базы данных.



Рисунок 3.8 – Модель Event

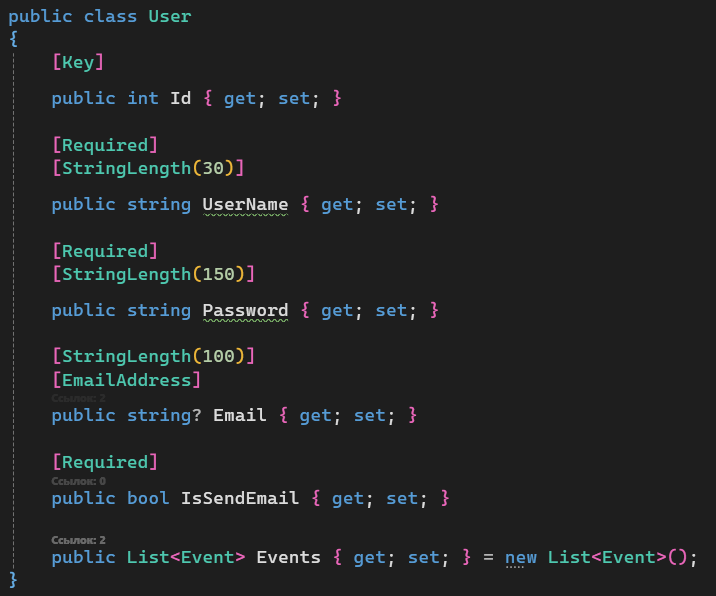


Рисунок 3.9 – Модель User

**3.3.3** Безопасность работы с БД

Для обеспечения безопасности при работе с базой данных реализованы следующие меры:

* Использование параметризованных запросов (через EF Core), что исключает SQL-инъекции.
* Доступ к данным строго разграничен: пользователь может видеть и изменять только свои события.
* Все операции с базой выполняются на стороне сервера, клиент не имеет прямого доступа к БД.

## 3.4 Аутентификация и авторизация

Для обеспечения защищённого доступа к функционалу приложения реализованы механизмы аутентификации (подтверждение личности пользователя) и авторизации (предоставление доступа к определённым данным и действиям в зависимости от прав пользователя).

**3.4.1** Используемые технологии

В проекте используется JWT (JSON Web Token) в связке с HTTP-only cookie:

* JWT — основной механизм передачи информации о пользователе между клиентом и сервером после входа в систему.
* Refresh Token — используется для безопасного обновления access token без необходимости повторного входа.
* Cookie — токены хранятся в защищённых HTTP-only cookie, что исключает возможность доступа к ним через JavaScript и снижает риск XSS-атак.

**3.4.2** Процесс аутентификации

Пользователь вводит свои учётные данные на форме входа. Эти данные отправляются на backend через REST API. Backend проверяет подлинность данных, сравнивая хеш пароля (с использованием безопасного алгоритма, например, HMACSHA256 или bcrypt).

В случае успеха:

* Генерируется access token (короткоживущий, 5–15 минут).
* Генерируется refresh token (долгоживущий, до 7 дней).
* Токены отправляются в cookie.

3.4.3 Обновление токенов

Когда срок действия access token истекает, клиент автоматически отправляет refresh token на endpoint /refresh. Если refresh token действителен, сервер выдаёт новый access token.

3.4.4 Авторизация

Для защиты API применяется политика авторизации:

* Доступ к защищённым ресурсам разрешён только авторизованным пользователям.
* Backend проверяет access token, извлекает из него идентификатор пользователя и выполняет запросы от его имени.
* Пользователи имеют доступ только к своим событиям, что реализуется через фильтрацию по UserId.

3.4.5 Безопасность

Для обеспечения безопасности реализованы следующие меры:

* Хеширование паролей перед хранением в базе данных.
* Хранение токенов только в HTTP-only cookie.
* Проверка подлинности токенов с использованием секретного ключа (симметричная подпись JWT).
* Ограничение срока жизни токенов.

3.4.6 Logout (выход из системы)

* При выходе пользователя:
* Cookie с токенами удаляются на клиенте.
* Следующие запросы от пользователя не будут авторизованы, пока он не войдёт снова.

# ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

На этапе тестирования программного средства проводилась проверка корректности функционирования всех основных компонентов веб-приложения: интерфейса пользователя (frontend), серверной логики (backend), системы аутентификации и взаимодействия с базой данных.

В результате тестирования были выявлены и устранены следующие недочёты:

1. Ошибка отображения событий после добавления

Во время тестирования приложения была обнаружена ошибка, при которой добавленное событие не отображалось в календаре до принудительного обновления страницы. Это было связано с тем, что после успешного POST-запроса на добавление нового события данные в компоненте календаря оставались прежними и не обновлялись автоматически.

Решение:

Для устранения этой проблемы после выполнения fetch-запроса на добавление события было реализовано обновление состояния с помощью хука useState. После получения успешного ответа от API вызывалась функция, изменяющая значение состояния, отвечающего за перезагрузку данных. Это изменение инициировало повторный вызов функции для вывода событий и обеспечивало отображение нового события в календаре без необходимости ручного обновления страницы.

1. Проблема с истечением срока действия токена

В случае истечения срока действия access token запросы к API начинали возвращать статус 401 Unauthorized, и пользователь терял доступ к своим данным без очевидной причины.

Решение:

Реализован механизм автоматического обновления access token с помощью refresh token, хранящегося в cookie. При получении 401 от API frontend отправляет запрос на обновление токена (/refresh) и повторяет исходный запрос после успешного обновления.

1. Оптимизация загрузки событий по месяцам

Изначально при запуске приложения происходила загрузка всех событий за весь период, что приводило к длительному ожиданию и ухудшению пользовательского опыта. Это было особенно ощутимо при наличии большого количества данных в базе.

Решение:

Была реализована оптимизация, заключающаяся в загрузке событий только за текущий месяц. При каждом изменении месяца в компоненте календаря формируется диапазон дат — первый и последний день выбранного месяца. Эти даты передаются на сервер как параметры, и API возвращает только те события, которые попадают в указанный диапазон.

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## 5.1 Интерфейс программного средства

5.1.1 Окно авторизации и аунтификации

При запуске приложения пользователю сначала отображается окно регистрации. В этом окне необходимо ввести логин и пароль для создания нового аккаунта. После успешной регистрации пользователь автоматически перенаправляется на окно входа. В окне входа осуществляется проверка учетных данных. Если введённые данные корректны, пользователь успешно авторизуется и попадает на главное окно календарного приложения, где отображается календарь и события текущей даты. Внешний вид главного окна веб-браузера представлен на рисунке 5.1.

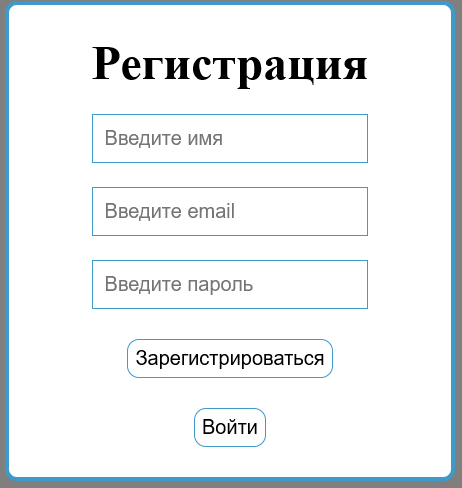


Рисунок 5.1 – Окно регистрации

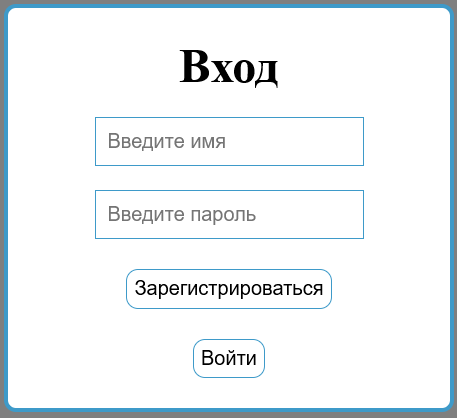


Рисунок 5.2 – Окно входа

5.1.2 Главное окно

Главное окно приложения разделено на две главные зоны. В левой части располагается календарная сетка с навигацией по месяцам и годам, а в правой части — панель выбранной даты, где отображаются события и расположены кнопки «Добавить событие», «Редактировать», «Удалить» и «Выйти».

Для повышения удобства использования кнопки оформлены в виде иконок:

* Редактировать — иконка в виде карандаша;
* Удалить — иконка крестика в кружочке;
* Выйти — иконка двери.

Такая компоновка позволяет пользователю одновременно видеть календарь и список задач на выбранный день, а также интуитивно управлять событиями. Внешний вид главного окна календаря показан на рисунке 5.3

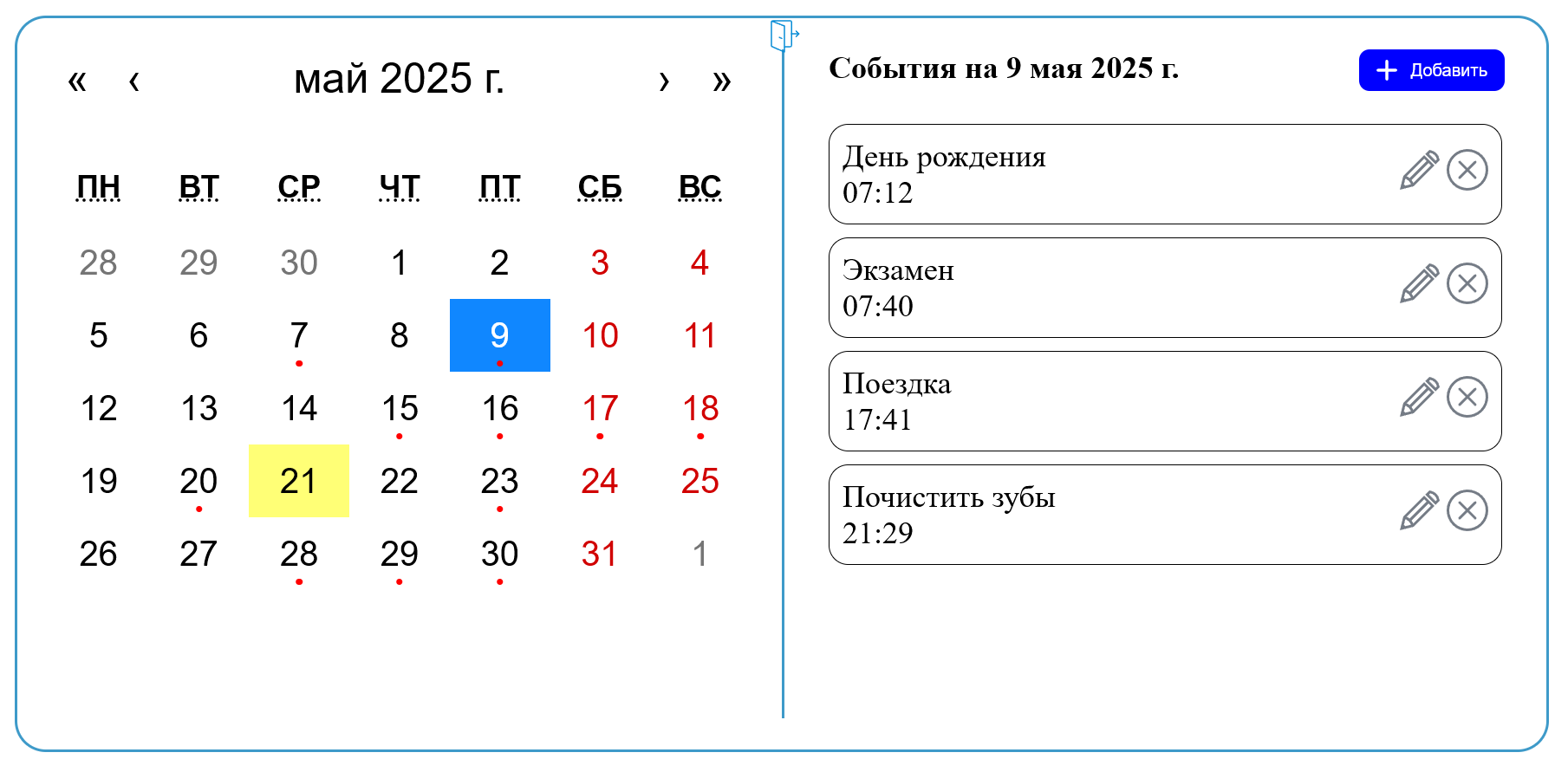


Рисунок 5.3 – Главное окно

5.1.3 Окно добавления события

Окно добавления события содержит две кнопки для добавления и отмены добавления события и поля ввода названия события, даты и времени. Внешний вид окна добавления закладок представлен на рисунке 5.3.

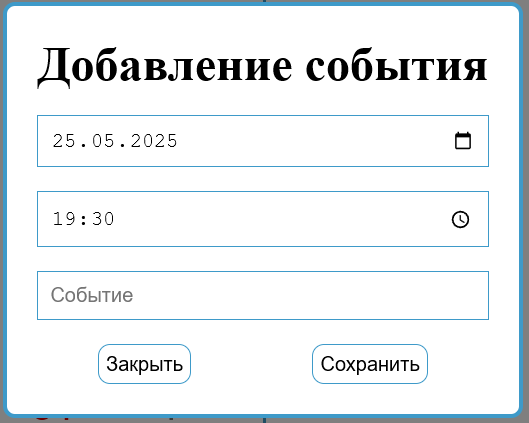


Рисунок 5.3 – Окно добавления события

5.1.4 Окно изменения события

Окно изменения события открывается при нажатии на иконку карандаша, расположенную рядом с соответствующим событием в списке. Оно используется для редактирования уже существующего события.

Интерфейс окна аналогичен окну добавления события и включает в себя:

* поле ввода названия события;
* поле выбора даты;
* поле выбора времени начала и окончания события;
* кнопки «Сохранить изменения» и «Отмена».

После внесения правок пользователь может сохранить изменения или отказаться от редактирования. Внешний вид окна изменения события представлен на рисунке 5.4.

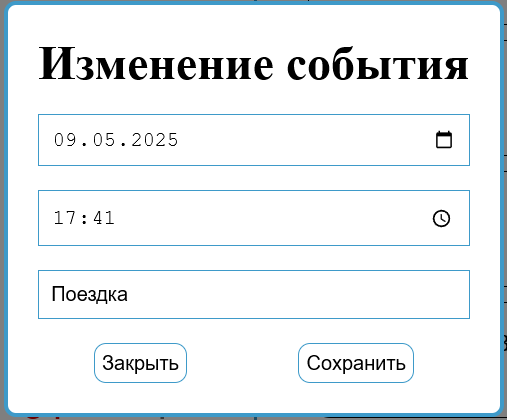


Рисунок 5.4 – Окно изменения события

## 5.2 Управление программным средством

5.2.1 Элементы управления приложением

Элементы управления приложения включают в себя кнопки, иконки и поля ввода, предназначенные для взаимодействия пользователя с календарём и событиями. Все интерактивные элементы снабжены всплывающими подсказками, описывающими их назначение.

Пользователю доступны следующие функции:

* Переключение между месяцами и годами с помощью стрелок навигации;
* Просмотр списка событий на выбранную дату;
* Добавление события через кнопку «Добавить событие»;
* Редактирование события (иконка карандаша рядом с событием);
* Удаление события (иконка крестика в кружочке);
* Выход из аккаунта (иконка двери);
* Автоматическая подгрузка событий при смене месяца;
* Обработка повторного входа и регистрации с переходом между соответствующими окнами.

Такая организация интерфейса обеспечивает удобную и интуитивно понятную работу с календарём.

# ЗАКЛЮЧЕНИЕ

В современном мире цифровые инструменты планирования и организации времени приобретают всё большее значение. Календарные приложения становятся неотъемлемой частью повседневной жизни, помогая пользователям эффективно управлять своими задачами и событиями.

В рамках данного курсового проекта было разработано программное средство — веб-календарь, предназначенное для создания, просмотра и управления пользовательскими событиями. Приложение реализовано с использованием современных технологий: React для frontend-части, ASP.NET Core (C#) для backend-а, PostgreSQL в качестве СУБД, и REST API для взаимодействия между клиентом и сервером. Также в проекте были успешно внедрены механизмы аутентификации и авторизации на базе JWT и Cookie.

В процессе разработки были успешно реализованы все поставленные задачи. Функционал календаря включает:

* Регистрацию и вход пользователя в систему;
* Отображение календарной сетки с возможностью навигации по месяцам и годам;
* Добавление событий на выбранную дату;
* Редактирование и удаление событий (с помощью иконок карандаша и мусорной корзины);
* Выход из аккаунта (иконка двери);
* Загрузка событий с сервера только за выбранный месяц (для повышения производительности);
* Автоматическое обновление интерфейса после добавления и изменения событий.

Приложение прошло этап тестирования, в ходе которого были выявлены и успешно устранены ошибки, включая проблему с отображением новых событий без перезагрузки страницы и начальную загрузку всех данных за всё время.

Перспективы развития приложения включают:

* Реализацию напоминаний и уведомлений о предстоящих событиях;
* Добавление категории или цвета для событий;
* Расширение локализации интерфейса;
* Возможность интеграции с внешними календарями (например, Google Calendar).

Созданное программное средство обеспечивает удобный и интуитивно понятный интерфейс для управления личным расписанием, что делает его полезным инструментом как для повседневной жизни, так и для профессиональной деятельности.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] React. React – A JavaScript library for building user interfaces [Электронный ресурс]. – Режим доступа: https://react.dev/, свободный.

[2] Microsoft. ASP.NET Core Documentation [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/aspnet/core/, свободный.

[3] PostgreSQL. The world’s most advanced open source relational database [Электронный ресурс]. – Режим доступа: https://www.postgresql.org/docs/, свободный.

[4] JWT.IO. JSON Web Token Introduction [Электронный ресурс]. – Режим доступа: https://jwt.io/introduction, свободный.

[5] Mozilla. Mozilla Developer Network – MDN Web Docs [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/, свободный.

[6] Фленаган Д. JavaScript. Подробное руководство. — СПб.: Символ-Плюс, 2020. — 960 с.

[7] Мартин Р. Чистый код: создание, анализ и рефакторинг. — СПб.: Питер, 2018. — 464 с.

# ПРИЛОЖЕНИЕ А. Исходный код программы

UsersControllers

[ApiController]

[Route("/users")]

public class UsersControllers(UserRepositories repositories) : ControllerBase

{

static string secretKey = "super\_super\_secret\_key\_with\_256\_bits\_!!!"; // 256+ бит

static string issuer = "MyApp";

static string audience = "MyUsers";

static SymmetricSecurityKey signingKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secretKey));

string GenerateToken(string username, int id, int minutes)

{

var claims = new[]

{

new Claim(ClaimTypes.Name, username),

new Claim(ClaimTypes.NameIdentifier, id.ToString())

};

var token = new JwtSecurityToken(

issuer,

audience,

claims,

expires: DateTime.UtcNow.AddMinutes(minutes),

signingCredentials: new SigningCredentials(signingKey, SecurityAlgorithms.HmacSha256)

);

return new JwtSecurityTokenHandler().WriteToken(token);

}

[HttpPost("/registr")]

public async Task<IActionResult> Registration([FromBody] CreateUserRequest User)

{

bool result = await repositories.Add(User.UserName, User.Password, User.Email);

return result ? Ok("Регистрация прошла успешно") : Unauthorized("Пользователь с таким именем существует");

}

[HttpPost("/login")]

public async Task<IActionResult> Login([FromBody] LoginRequest request)

{

User user = await repositories.GetUser(request.UserName, request.Password);

if(user == null)

return Unauthorized();

var accessToken = GenerateToken(user.UserName, user.Id, 10); // 1 минута

var refreshToken = GenerateToken(user.UserName, user.Id, 60); // 60 минут

Response.Cookies.Append("access\_token", accessToken, new CookieOptions

{

Path = "/",

HttpOnly = true,

SameSite = SameSiteMode.None,

Secure = true,

Expires = DateTime.UtcNow.AddMinutes(1)

});

Response.Cookies.Append("refresh\_token", refreshToken, new CookieOptions

{

Path = "/",

HttpOnly = true,

SameSite = SameSiteMode.None,

Secure = true,

Expires = DateTime.UtcNow.AddMinutes(60)

});

return Ok("Авторизация успешна");

}

[HttpGet("/logout")]

[Authorize]

public async Task<IActionResult> Logout()

{

var options = new CookieOptions

{

Path = "/",

HttpOnly = true,

SameSite = SameSiteMode.None,

Secure = true

};

Response.Cookies.Delete("access\_token", options);

Response.Cookies.Delete("refresh\_token", options);

return Ok("Вы вышли");

}

[HttpPost("/refresh")]

public IActionResult Refresh()

{

// Получаем refresh\_token из cookie

var refreshToken = Request.Cookies["refresh\_token"];

if (string.IsNullOrEmpty(refreshToken))

return Unauthorized("Нет refresh\_token");

try

{

var tokenHandler = new JwtSecurityTokenHandler();

var validationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidIssuer = issuer,

ValidateAudience = true,

ValidAudience = audience,

ValidateIssuerSigningKey = true,

IssuerSigningKey = signingKey,

ValidateLifetime = true,

ClockSkew = TimeSpan.Zero // чтобы не было "запаса" времени

};

var principal = tokenHandler.ValidateToken(refreshToken, validationParameters, out SecurityToken validatedToken);

var username = principal.Identity.Name;

var userIdClaim = principal.FindFirst(ClaimTypes.NameIdentifier);

if (username == null || userIdClaim == null || !int.TryParse(userIdClaim.Value, out int userId))

return Unauthorized("Недопустимый refresh\_token");

// Генерируем новые токены

var newAccessToken = GenerateToken(username, userId, 1); // 1 минута

var newRefreshToken = GenerateToken(username, userId, 60); // 60 минут

// Устанавливаем куки

Response.Cookies.Append("access\_token", newAccessToken, new CookieOptions

{

Path = "/",

HttpOnly = true,

SameSite = SameSiteMode.None,

Secure = true,

Expires = DateTime.UtcNow.AddMinutes(1)

});

Response.Cookies.Append("refresh\_token", newRefreshToken, new CookieOptions

{

Path = "/",

HttpOnly = true,

SameSite = SameSiteMode.None,

Secure = true,

Expires = DateTime.UtcNow.AddMinutes(60)

});

return Ok(new { message = "Токены обновлены" });

}

catch (SecurityTokenExpiredException)

{

return Unauthorized("Refresh token просрочен");

}

catch (Exception ex)

{

return Unauthorized($"Ошибка проверки refresh token: {ex.Message}");

}

}

}

public class LoginRequest

{

public string UserName { get; set; }

public string Password { get; set; }

}

public class CreateUserRequest

{

public string UserName { get; set; }

public string Password { get; set; }

public string Email { get; set; }

}

EventsControllers

[ApiController]

[Route("/events")]

public class EventsControllers(EventsRepositories repositories) : ControllerBase

{

[HttpGet]

[Authorize]

public async Task<ActionResult<List<EventDemonstration>>> GetListEvents(

[FromQuery] int? page = null,

[FromQuery] int? pageSize = null,

[FromQuery] DateTimeOffset? from = null,

[FromQuery] DateTimeOffset? to = null,

[FromQuery] string? sort = null

)

{

var idClaim = User.FindFirst(ClaimTypes.NameIdentifier);

var userId = idClaim != null ? int.Parse(idClaim.Value) : (int?)null;

if (userId == null) return Unauthorized();

var result = await repositories.GetList(userId.Value, page, pageSize, from, to, sort);

return Ok(result);

}

[HttpPost]

[Authorize]

public async Task<ActionResult<Event>> AddEvent([FromBody] CreateEventRequest newEvent)

{

var idClaim = User.FindFirst(ClaimTypes.NameIdentifier);

var userId = idClaim != null ? int.Parse(idClaim.Value) : (int?)null;

var ev = await repositories.Add(userId.Value, newEvent.Description, newEvent.Time);

return Ok(ev);

}

[HttpPut("{id:int}")]

[Authorize]

public async Task<ActionResult> UpdateEvent([FromBody] CreateEventRequest eventRequest, int id)

{

var ev = await repositories.UpdateAll(id, eventRequest);

return Ok(ev);

}

[HttpPatch("{id:int}")]

[Authorize]

public async Task<ActionResult> UpdatePartially(int id, [FromBody] PatchEventRequest patchEventRequest)

{

await repositories.UpdatePatch(id, patchEventRequest);

return Ok("Обновлено успешно");

}

[HttpDelete("{id:int}")]

[Authorize]

public async Task<ActionResult> DeleteEvent(int id)

{

await repositories.Delete(id);

return Ok("Удаление прошло успешно");

}

[HttpGet("test")]

public async Task<ActionResult<List<EventDemonstration>>> GetTest()

{

Console.WriteLine("GET /test was called");

var result = await repositories.GetList(5, 1, 10, null, null, "desc");

return Ok(result);

}

}

public class CreateEventRequest

{

public string Description { get; set; }

public DateTimeOffset Time { get; set; }

}

public class PatchEventRequest

{

public string? Description { get; set; } = null;

public DateTimeOffset? Time { get; set; }

}

EventsRepositories

public class EventsRepositories

{

private readonly ApplicationDbContext \_dbContext;

public EventsRepositories(ApplicationDbContext dbContext)

{

\_dbContext = dbContext;

}

public async Task<List<EventDemonstration>> GetList(

int userId,

int? page,

int? pageSize,

DateTimeOffset? from,

DateTimeOffset? to,

string? sort

)

{

var query = \_dbContext.Events.Where(x => x.UserId == userId);

Console.WriteLine(query.ToString());

if (from.HasValue)

{

var fromDate = from.Value.ToUniversalTime();

query = query.Where(x => x.Time >= fromDate);

}

if (to.HasValue)

{

var toDate = to.Value.ToUniversalTime();

query = query.Where(x => x.Time <= toDate);

}

// Применяем сортировку только если явно указана

if (!string.IsNullOrWhiteSpace(sort))

{

query = sort.ToLower() switch

{

"asc" => query.OrderBy(e => e.Time),

"desc" => query.OrderByDescending(e => e.Time),

\_ => query // игнорируем сортировку, если значение некорректное

};

}

// Применяем пагинацию только если указаны оба параметра

if (page.HasValue && pageSize.HasValue)

{

query = query

.Skip((page.Value - 1) \* pageSize.Value)

.Take(pageSize.Value);

}

var items = await query.ToListAsync();

return items.Select(x => new EventDemonstration

{

Id = x.Id,

Description = x.Description,

Time = x.Time

}).ToList();

}

public async Task<Event> Add(int id, string description, DateTimeOffset date)

{

Event newEvent = new Event

{

UserId = id,

Description = description,

Time = date.ToUniversalTime(),

};

\_dbContext.Events.Add(newEvent);

await \_dbContext.SaveChangesAsync(); // Сохраняем изменения в базе данных

return newEvent;

}

public async Task<Event> UpdateAll(int eventId, CreateEventRequest eventRequest)

{

Event updateEvent = await \_dbContext.Events.FirstOrDefaultAsync(x => x.Id == eventId);

if (updateEvent == null)

throw new Exception("Событие не найдено");

updateEvent.Description = eventRequest.Description;

updateEvent.Time = eventRequest.Time;

await \_dbContext.SaveChangesAsync();

return updateEvent;

}

public async Task UpdatePatch(int eventId, PatchEventRequest patchEventRequest)

{

Event updateEvent = await \_dbContext.Events.FirstOrDefaultAsync(x => x.Id == eventId);

if (updateEvent == null)

throw new Exception("Событие не найдено");

if (patchEventRequest.Description != null)

{

updateEvent.Description = patchEventRequest.Description;

}

if (patchEventRequest.Time.HasValue)

{

updateEvent.Time = patchEventRequest.Time.Value;

}

await \_dbContext.SaveChangesAsync();

}

public async Task Delete(int evId)

{

var eventToRemove = await \_dbContext.Events

.FirstOrDefaultAsync(x => x.Id == evId);

if (eventToRemove != null)

{

\_dbContext.Events.Remove(eventToRemove);

await \_dbContext.SaveChangesAsync();

}

else

{

throw new InvalidOperationException("Событие не найдено.");

}

}

}

public class EventDemonstration

{

public int Id { get; set; }

public string Description { get; set; }

ApplyFormatInEditMode = true)]

public DateTimeOffset Time { get; set; }

}

UserRepositories

public class UserRepositories

{

private readonly ApplicationDbContext \_dbContext;

static string secretKey = "super\_super\_secret\_key\_with\_256\_bits\_!!!"; // 256+ бит

static string issuer = "MyApp";

static string audience = "MyUsers";

static SymmetricSecurityKey signingKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secretKey));

public UserRepositories(ApplicationDbContext dbContext)

{

\_dbContext = dbContext;

}

public async Task<bool> Add(string username, string password, string email)

{

// Создайте экземпляр PasswordHasher

var passwordHasher = new PasswordHasher<User>();

if (\_dbContext.Users.Select(x => x.UserName).ToList().Contains(username)) return false;

// Хэшируйте пароль

var hashedPassword = passwordHasher.HashPassword(null, password);

var user = new User

{

UserName = username,

// Сохраните только хэшированный пароль

Password = hashedPassword,

Email = email

};

await \_dbContext.Users.AddAsync(user);

await \_dbContext.SaveChangesAsync();

return true;

}

public async Task<User> GetUser(string Name, string Password)

{

var passwordHasher = new PasswordHasher<User>();

// Хэшируйте пароль

//var hashedPassword = passwordHasher.HashPassword(null, Password);

User user = await \_dbContext.Users.FirstOrDefaultAsync(x => x.UserName == Name);

var verificationResult = passwordHasher.VerifyHashedPassword(user, user.Password, Password);

//bool a = user.Password == hashedPassword;

return verificationResult == PasswordVerificationResult.Success ? user : null;

}

}

Program.cs

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Options;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using Swashbuckle.AspNetCore.Filters;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using System.Text.Encodings.Web;

using System.Text.Json;

using System.Text.Json.Serialization;

using Курсач\_1;

using Курсач\_1.Repositories;

//AppContext.SetSwitch("Npgsql.EnableLegacyTimestampBehavior", true);

var builder = WebApplication.CreateBuilder(args);

var secretKey = "super\_super\_secret\_key\_with\_256\_bits\_!!!"; // 256+ бит

var issuer = "MyApp";

var audience = "MyUsers";

var signingKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secretKey));

// Add services to the container.

builder.Services.AddControllers();

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo { Title = "Auth API", Version = "v1" });

c.AddSecurityDefinition("cookieAuth", new OpenApiSecurityScheme

{

Name = "access\_token",

Type = SecuritySchemeType.ApiKey,

In = ParameterLocation.Cookie,

Description = "JWT stored in cookie"

});

c.MapType<DateTime>(() => new OpenApiSchema

{

Type = "string",

Format = "date-time"

});

c.AddSecurityRequirement(new OpenApiSecurityRequirement

{

{

new OpenApiSecurityScheme

{

Reference = new OpenApiReference

{

Type = ReferenceType.SecurityScheme,

Id = "cookieAuth"

}

},

Array.Empty<string>()

}

});

});

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(JwtBearerDefaults.AuthenticationScheme, options =>

{

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidIssuer = issuer,

ValidateAudience = true,

ValidAudience = audience,

ValidateIssuerSigningKey = true,

IssuerSigningKey = signingKey,

ValidateLifetime = false,

//ClockSkew = TimeSpan.Zero

};

// Берём токен из куки

options.Events = new JwtBearerEvents

{

OnMessageReceived = ctx =>

{

var token = ctx.Request.Cookies["access\_token"];

if (!string.IsNullOrEmpty(token))

{

ctx.Token = token;

}

return Task.CompletedTask;

},

OnChallenge = ctx =>

{

// Прерываем редирект

ctx.HandleResponse();

ctx.Response.StatusCode = 401;

ctx.Response.ContentType = "application/json";

return ctx.Response.WriteAsync("{\"error\":\"Unauthorized\"}");

},

OnForbidden = ctx =>

{

ctx.Response.StatusCode = 403;

ctx.Response.ContentType = "application/json";

return ctx.Response.WriteAsync("{\"error\":\"Forbidden\"}");

}

};

});

builder.Services.AddAuthorization();

var configuration = builder.Configuration.GetConnectionString("ApplicationDbContext");

builder.Services.AddDbContext<ApplicationDbContext>(context => context.UseNpgsql(configuration));

builder.Services.AddTransient<UserRepositories>();

builder.Services.AddTransient<EventsRepositories>();

builder.Services.AddCors(options =>

{

options.AddPolicy("ReactPolicy", policy =>

{

policy.WithOrigins("http://localhost:5173", "http://localhost:5174") // адрес фронтенда

.AllowAnyHeader()

.AllowAnyMethod()

.AllowCredentials(); // если используешь куки

});

});

var app = builder.Build();

app.UseDeveloperExceptionPage();

app.UseRouting();

app.UseCors("ReactPolicy");

app.Use(async (context, next) =>

{

var handler = new JwtSecurityTokenHandler();

var accessToken = context.Request.Cookies["access\_token"];

var refreshToken = context.Request.Cookies["refresh\_token"];

ClaimsPrincipal? principal = null;

if (!string.IsNullOrEmpty(accessToken))

{

try

{

principal = handler.ValidateToken(accessToken, new TokenValidationParameters

{

ValidateIssuer = true,

ValidIssuer = issuer,

ValidateAudience = true,

ValidAudience = audience,

ValidateIssuerSigningKey = true,

IssuerSigningKey = signingKey,

ValidateLifetime = true,

ClockSkew = TimeSpan.Zero

}, out \_);

}

catch (SecurityTokenExpiredException)

{

// Токен просрочен — попробуем обновить

}

catch

{

// access\_token невалиден — удалим

context.Response.Cookies.Delete("access\_token");

}

}

if (principal == null && !string.IsNullOrEmpty(refreshToken))

{

try

{

var refreshPrincipal = handler.ValidateToken(refreshToken, new TokenValidationParameters

{

ValidateIssuer = true,

ValidIssuer = issuer,

ValidateAudience = true,

ValidAudience = audience,

ValidateIssuerSigningKey = true,

IssuerSigningKey = signingKey,

ValidateLifetime = true,

ClockSkew = TimeSpan.Zero

}, out \_);

var username = refreshPrincipal.Identity?.Name;

var idClaim = refreshPrincipal.FindFirst(ClaimTypes.NameIdentifier);

var userId = idClaim != null ? int.Parse(idClaim.Value) : (int?)null;

var newAccessToken = GenerateToken(username, userId.Value, 1);

context.Response.Cookies.Append("access\_token", newAccessToken, new CookieOptions

{

HttpOnly = true,

SameSite = SameSiteMode.Lax,

Expires = DateTime.UtcNow.AddMinutes(1)

});

// Добавим в заголовок, чтобы JwtBearer его увидел

context.Request.Headers["Authorization"] = $"Bearer {newAccessToken}";

}

catch

{

context.Response.Cookies.Delete("refresh\_token");

}

}

// Устанавливаем principal в контексте, если он валиден

if (principal != null)

{

context.User = principal;

}

await next();

});

string GenerateToken(string username, int id, int minutes)

{

var claims = new[]

{

new Claim(ClaimTypes.Name, username),

new Claim(ClaimTypes.NameIdentifier, id.ToString())

};

var token = new JwtSecurityToken(

issuer,

audience,

claims,

expires: DateTime.UtcNow.AddMinutes(minutes),

signingCredentials: new SigningCredentials(signingKey, SecurityAlgorithms.HmacSha256)

);

return new JwtSecurityTokenHandler().WriteToken(token);

}

app.UseAuthentication();

app.UseAuthorization();

app.UseSwagger();

app.UseSwaggerUI();

// Configure the HTTP request pipeline.

app.MapControllers();

app.Run();

// dotnet ef migrations add InitialCreate --project "Курсач\_1" --startup-project "Курсач\_1"

// dotnet ef database update --project "Курсач\_1" --startup-project "Курсач\_1"

// dotnet ef migrations add UpdateEventTable

// dotnet ef database update

EventConfiguration

public class EventConfiguration : IEntityTypeConfiguration<Event>

{

public void Configure(EntityTypeBuilder<Event> builder)

{

builder.HasKey(e => e.Id);

builder.HasOne(e => e.User).WithMany(e => e.Events).HasForeignKey(e => e.UserId).OnDelete(DeleteBehavior.Cascade);

}

}

UserConfiguration

public class UserConfiguration : IEntityTypeConfiguration<User>

{

public void Configure(Microsoft.EntityFrameworkCore.Metadata.Builders.EntityTypeBuilder<User> builder)

{

builder.HasKey(t => t.Id);

builder.HasMany(e => e.Events).WithOne(e => e.User).HasForeignKey(e => e.UserId);

}

}