

Содержание

Введение.....	5
1 Постановка задачи и обзор аналогичных решений	6
1.1 Постановка задач.....	6
1.2 Обзор аналогичных решений.....	6
1.2.1 «Спортивный вызов»	6
1.2.2 «Life +»	8
1.2.3 «SPOBI»	9
1.3 Выводы по разделу.....	10
2 Проектирование веб-приложения.....	11
2.1 Функциональные возможности веб-приложения	11
2.2 Проектирование базы данных.....	13
2.3 Архитектура веб-приложения.....	16
2.4 Выводы по разделу.....	17
3 Реализация веб-приложения	18
3.1 Обоснование выбора программной платформы	18
3.2 Система управления базами данных PostgreSQL	18
3.3 ApplicationDbContext	18
3.4 Программные библиотеки.....	21
3.5 Структура серверной части	22
3.7 Реализация функционала для пользователя с ролью «Гость».....	24
3.7.1 Регистрация.....	24
3.7.2 Авторизация.....	24
3.8 Реализация функционала для пользователя с ролью «Пользователь» ..	25
3.8.1 Добавление пользовательских данных	25
3.8.2 Выбор предпочитаемого вида активности	27
3.8.3 Получение сгенерированного задания.....	27
3.8.4 Ввод ежедневной активности	28
3.8.5 Возможность бросать вызов друзьям	29
3.8.6 Добавление в друзья	29
3.9 Реализация функционала для пользователя с ролью «Администратор»	31
3.9.1 Добавление активности	31
3.9.2 Ведение статистики	32
3.9.3 Сравнение активности пользователей	33
3.10 Структура клиентской части.....	34
3.11 Выводы по разделу.....	34
4 Тестирование веб-приложения	35
4.1 Функциональное тестирование.....	35
4.2 Выводы по разделу.....	38
5 Руководство пользователя.....	39
5.1 Руководство пользователя для роли «Гость»	39
5.1.1 Регистрация.....	39
5.1.2 Авторизация.....	39
5.2 Руководство пользователя для роли «Пользователь»	40

5.2.1 Добавление пользовательских данных	40
5.2.2 Выбор предпочитаемого вида активности	40
5.2.3 Получение сгенерированного задания	41
5.2.4 Ввод ежедневной активности	41
5.2.5 Возможность бросать вызов друзьям.....	42
5.2.6 Добавление в друзья	43
5.3 Руководство пользователя для роли «Администратор»	44
5.3.1 Добавление активности	44
5.3.2 Ведение статистики	45
5.3.3 Сравнение активности пользователей	45
5.4 Выводы по разделу.....	46
Заключение	47
Список используемых источников.....	48
ПРИЛОЖЕНИЕ А Скрипт создания базы данных	49
ПРИЛОЖЕНИЕ Б Middleware для проверки токена.....	51
ПРИЛОЖЕНИЕ В AuthContext и AuthProvider	53
ПРИЛОЖЕНИЕ Г Структурная схема веб-приложения.....	55

Введение

Веб-приложение – это клиент-серверное приложение, в котором клиент взаимодействует с сервером по протоколу HTTP.

Генератор спортивных вызовов представляет собой платформу для поощрения физической активности и мотивирования пользователей на достижение спортивных целей.

Целью данного проекта является реализация генерации предложений спортивных вызовов для пользователей в зависимости от их предпочтений и их состояния здоровья. Веб-приложение должно поддерживать несколько ролей: гость, пользователь, администратор.

Для достижения цели были поставлены следующие задачи:

1. Постановка задачи и обзор аналогичных решений (раздел 1);
2. Проектирование веб-приложения (раздел 2);
3. Реализация веб-приложения (раздел 3);
4. Тестирование веб-приложения (раздел 4);
5. Оформление пояснительной записки (раздел 5)

Целевая аудитория веб-приложения включает широкий круг пользователей, заинтересованных в улучшении своего здоровья, которые желают следить за своим прогрессом.

Серверная часть приложения реализована на ASP.NET [1], а клиентская часть — на React [2]. В качестве базы данных выбрана PostgreSQL [3], что обеспечивает высокую производительность и масштабируемость системы. Разработка ведется с учетом требований к безопасности, удобству эксплуатации и эффективному управлению данными, что подробно раскрыто в последующих главах.

Постановка задачи и обзор аналогичных решений

Постановка задач

Веб-приложение «Генератор спортивных вызовов» позволяет пользователям получать спортивные вызовы. Приложение поддерживает 3 основные роли: гость, пользователь и администратор, каждая из которых обладает определенным набором функций.

Гостя имеет возможность регистрации нового аккаунта, а также авторизации в системе. Пользователю может получать спортивные вызовы, просматривать список своих вызовов, вводить данные о себе, добавлять в друзья других пользователей. Администратор имеет возможность добавлять новые виды активности, просматривать статистику пользователей, просматривать списки пользователей и активностей.

1.2 Обзор аналогичных решений

«Спортивный вызов»

Приложение «Спортивный вызов» представляет собой мобильное приложение для участия в спортивных вызовах. Главная страница представлена на рисунке 1.1.

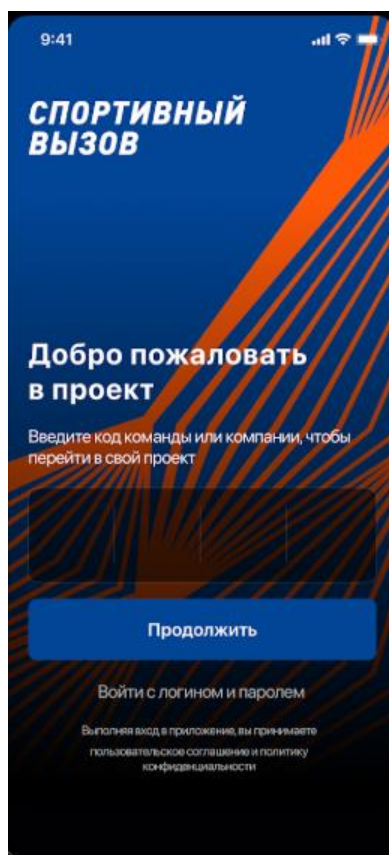


Рисунок 1.1 – Страница приложения «Спортивный вызов»

При входе сразу предлагается авторизоваться, чтобы иметь возможность использовать функции приложения. При успешной авторизации пользователю

открывается доступ к возможностям приложения, таким как участие в вызовах, просмотр ленты активностей и другое. На рисунке 1.2 и 1.3 представлены страницы с основным функционалом приложения.

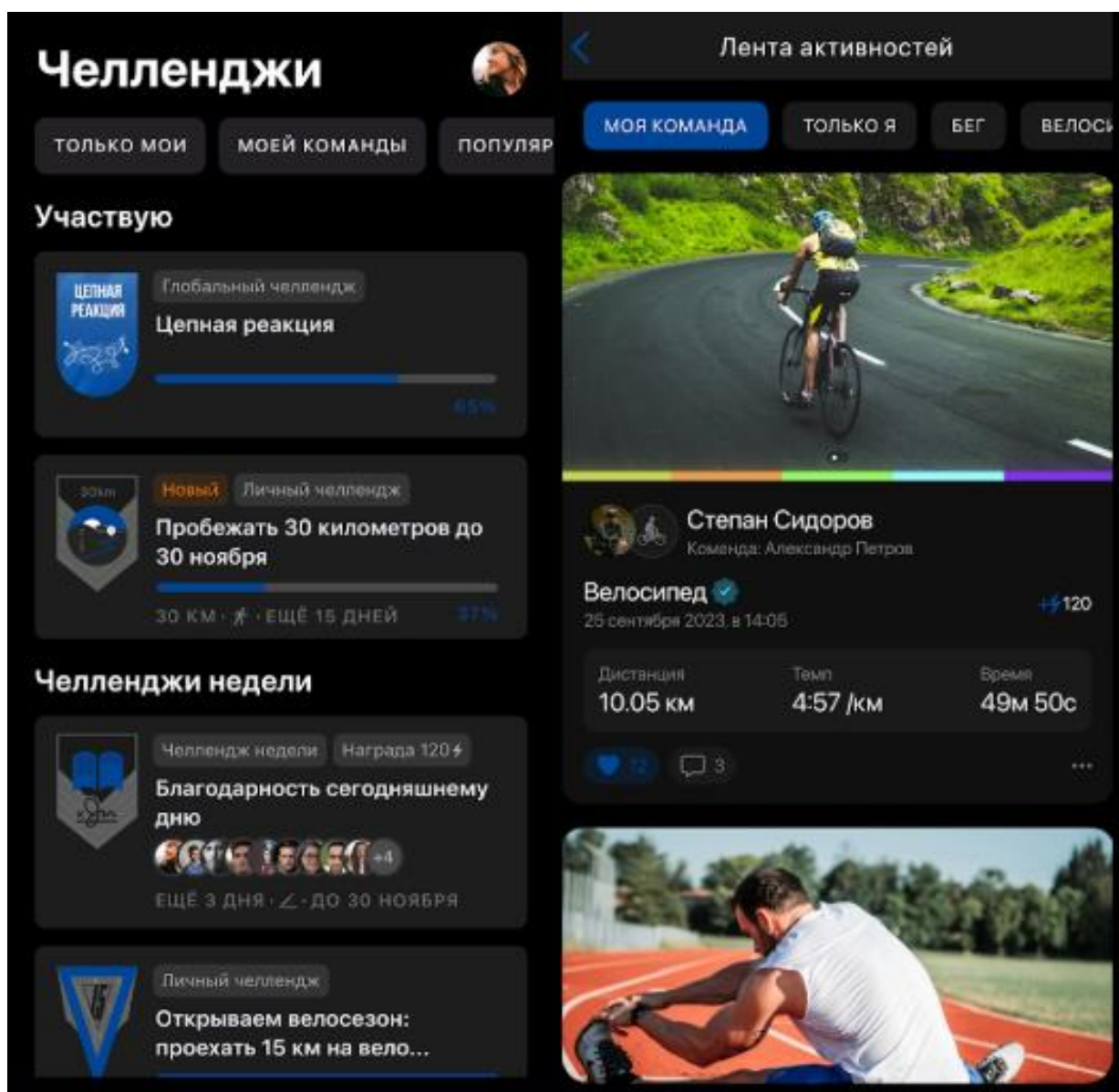


Рисунок 1.2 – Страница вызовов, рисунок 1.3 – Лента активностей

Достоинства:

- Удобная система просмотра вызовов;
- Простая и удобная навигация, позволяющая пользователям легко находить необходимые функции;
- Возможность делиться достижениями, приглашать друзей, создавать группы и соревноваться с другими пользователями;

Недостатки:

- Необходимость наличия современного смартфона или планшета, а также стабильного интернет-соединения для полноценного использования приложения.

Приложение для спорта «Life +» — это многофункциональная платформа, которая обеспечивает возможность участия в спортивных вызовах и позволяет отслеживать свою активность. Благодаря интуитивно понятному интерфейсу пользователи могут легко создавать и присоединяться к различным вызовам, соревноваться с друзьями или другими спортсменами, а также получать мотивацию через систему наград и достижений. Страница со спортивными вызовами представлена на рисунке 1.4.



- Возможность участвовать в соревнованиях, отслеживать результаты и прогресс;

- Возможность делиться достижениями.

- Возможные задержки, сбои или медленная работа приложения на менее мощных устройствах;

S P O B I

это мобильное приложение, разработанное специально для спортсменов и любителей активного образа жизни, доступное на платформах Android и iOS.

Приложение представляет собой современную социальную сеть, созданную для удобного общения, обмена опытом и поддержания мотивации среди пользователей, независимо от их уровня подготовки — от начинающих энтузиастов до профессиональных спортсменов. Страница приложения представлена на рисунке



Рисунок 1.5 – Страница статистики

Достоинства:

- Возможность загрузки фото в приложение;

Возможность оставлять отзывы.

Недостатки:

- Старомодный дизайн приложения.

1.3 Выводы по разделу

Поставленные задачи требуют разработки веб-приложения с поддержкой трёх ролей: гостя, пользователя и администратора, каждая из которых будет обладать своим набором функциональных возможностей. В результате разработки веб-приложения «Генератор спортивных вызовов» будет создана платформа, которая позволяет автоматически создавать вызовы на основе предпочтений пользователя.

Анализ аналогичных решений показал, что приложения предполагают базовый функционал, включая возможность выбора спортивных вызовов и отслеживания своего прогресса, но при этом имеющих свои недостатки, такие как устаревший дизайн, высокие требования.

2 Проектирование веб-приложения

2.1 Функциональные возможности веб-приложения

Функциональные возможности веб-приложения представлены в диаграмме вариантов использования, представленной на рисунке 2.1.

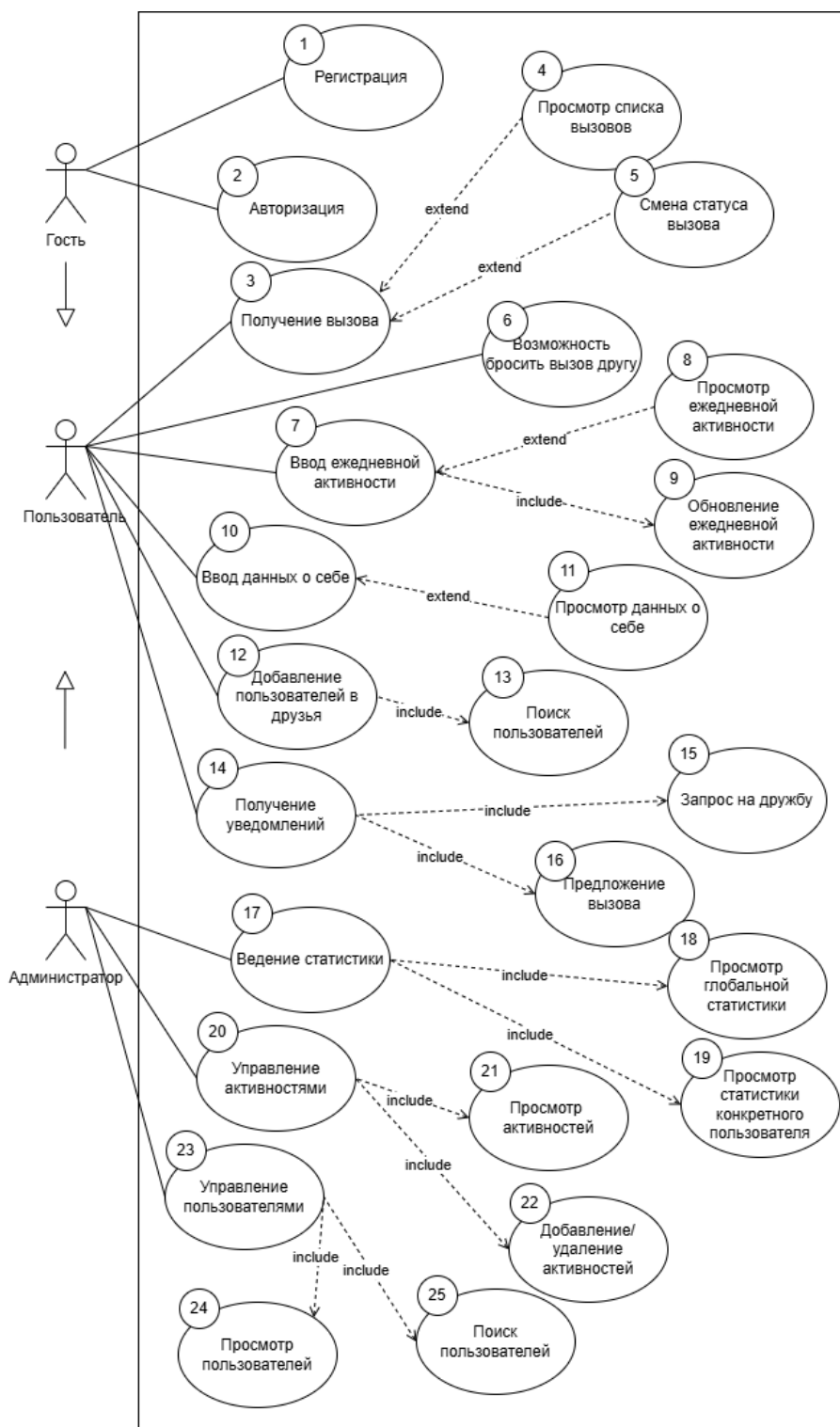


Рисунок 2.1 - Диаграмма вариантов использования веб-приложения

Описание ролей пользователей веб-приложения представлены в таблице 2.1.

Таблица 2.1 – Описание ролей пользователей веб-приложения

Роль	Описание
Гость	Пользователь, не прошедший регистрацию и авторизацию, не имеющий доступ к функциям приложения.
Пользователь	Пользователь, не прошедший регистрацию и аутентификацию, имеющий возможность получать вызовы, добавлять информацию о себе, добавлять друзей и заполнять ежедневную статистику.
Администратор	Уполномоченный пользователь, который может создавать виды активности, просматривать статистику и списки пользователей.

Роли в системе разделены таким образом, чтобы разграничить доступ к функционалу приложения и обеспечить безопасное использование платформы.

Функциональные возможности пользователя с ролью «Гость» приведены в таблице 2.2.

Таблица 2.2 – Функциональные возможности пользователя с ролью «Гость»

№	Вариант использования	Пояснение
	Регистрация	Возможность создания новой учетной записи в системе.
	Авторизация	Вход пользователя в систему с помощью данных учетной записи, таких как логин и пароль, получение прав доступа.

Функциональные возможности пользователя с ролью «Пользователь» представлены в таблице 2.3.

Таблица 2.3 – Функциональные возможности пользователя с ролью «Пользователь»

№	Вариант использования	Пояснение
	Добавление данных о себе. Выбор активности	Добавление информации о предпочитаемом виде активности, рост, вес.
	Добавление ежедневной активности	Ввод количества шагов и времени общей/другой активности, выполненных за день.
	Получение задания	Возможность получить ежедневное/еженедельное/ежемесячное задание.
	Добавление пользователя в друзья	Возможность найти и добавить выбранного пользователя в друзья для последующей возможности бросать вызовы друзьям.
	Бросание вызова	Возможность отправить свой вызов для выполнения друзьям.

Таблица 2.4 – Функциональные возможности пользователя с ролью «Администратор»

№	Вариант использования	Пояснение
	Добавлять виды активности	Возможность создавать новый вид активности в системе.
	Вести статистику активности пользователей	Ведение статистики каждого конкретного пользователя.
	Сравнивать активность пользователей	Ведение глобальную статистику активности пользователей.

Таким образом для каждой роли определен набор доступных действий и возможностей.

2.2 Проектирование базы данных

Согласно диаграммы вариантов использования была разработана база данных. PostgreSQL является реляционной системой управления базами данных (СУБД), которая использует концепцию таблиц, строк и столбцов.

Логическая схема базы данных приведена на рисунке 2.2.

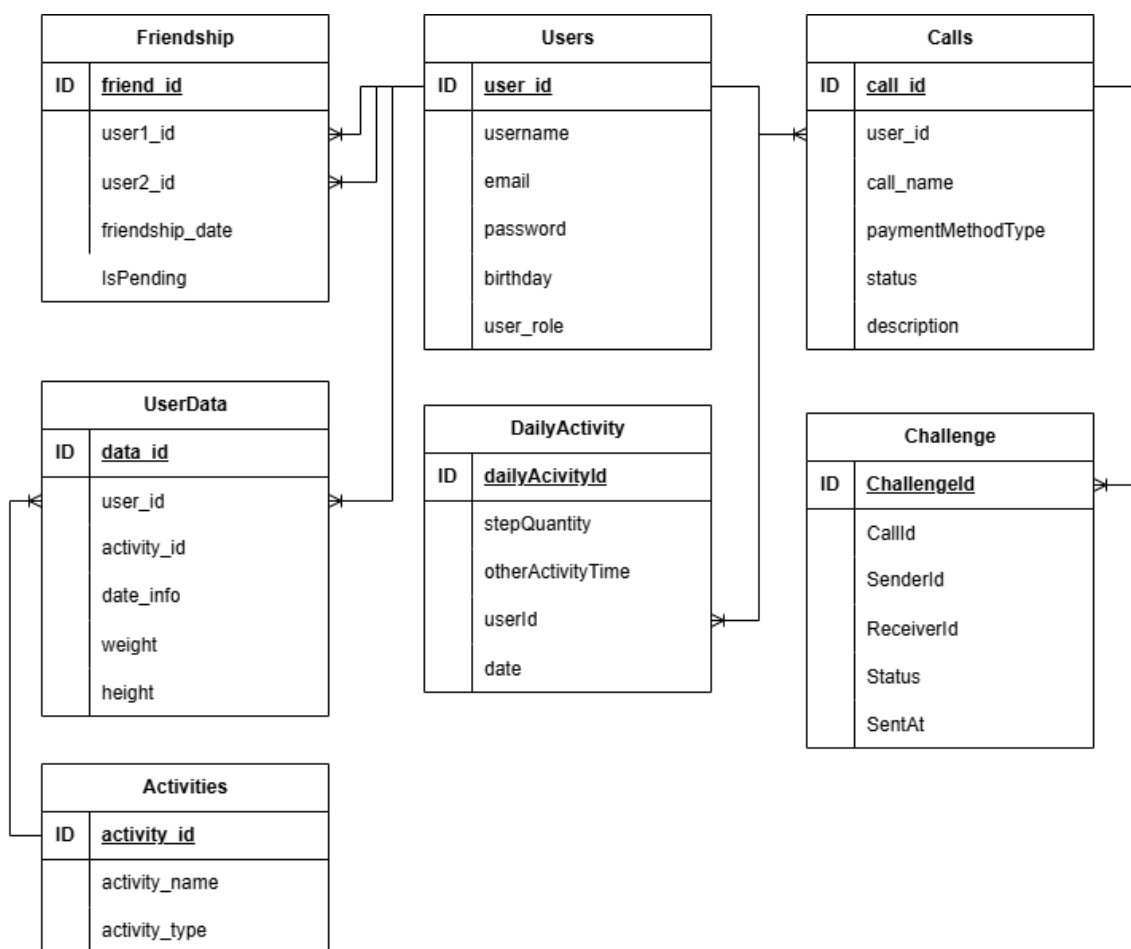


Рисунок 2.2 - Логическая схема базы данных

База данных содержит 7 коллекций, назначение каждой представлено в таблице 2.5.

Таблица 2.5 – Назначение коллекций базы данных

Коллекция	Назначение
Users	Хранение информации о пользователях
Calls	Хранение информации о сгенерированных вызовах
Challenges	Хранение информации о вызовах, полученных от других пользователей
UserData	Хранение данных о пользователе
Activities	Хранение видов активности
DailyActivity	Хранение записей о ежедневной активности
Friendship	Хранение данных о дружбе

Коллекция Users содержит информацию о пользователях веб-приложения. Структура данной коллекции приведена в таблице 2.6.

Таблица 2.6 – Структура коллекции Users

Название поля	Тип данных	Описание
User_id	INT	Идентификатор пользователя
username	STIRNG	Имя пользователя
password	STIRNG	Хешированный пароль пользователя
birthday	STIRNG	Дата рождения пользователя
user_role	STIRNG	Роль пользователя

Коллекция Calls содержит информацию о вызовах, сгенерированных в веб-приложении. Структура данной коллекции приведена в таблице 2.7.

Таблица 2.7 – Структура коллекции Calls

Название поля	Тип данных	Описание
call_id	INT	Идентификатор вызова
call_name	STIRNG	Название вызова
call_date	STIRNG	Дата создания вызова
status	STIRNG	Статус вызова
description	STIRNG	Описание вызова
user_id	INT	Идентификатор пользователя

Коллекция Challenges содержит информацию о вызовах, посланных между пользователями веб-приложения. Структура данной коллекции приведена в таблице

Таблица 2.8 – Структура коллекции Challenges

Название поля	Тип данных	Описание
ChallengeId	INT	Идентификатор отправленного вызова

Продолжение таблицы 2.8

SenderId	INT	Идентификатор отправителя
RecieverId	INT	Идентификатор получателя
CallId	INT	Идентификатор сгенерированного вызова
Status	STIRNG	Статус вызова
SentAt	DateTime	Время отправления вызова

Коллекция UserData содержит личные данные о пользователях веб-приложения. Структура данной коллекции приведена в таблице 2.9.

Таблица 2.9 – Структура коллекции UserData

Название поля	Тип данных	Описание
data_id	INT	Идентификатор записи данных
user_id	INT	Идентификатор пользователя
activity_id	INT	Идентификатор активности
date_info	Date	Дата записи данных
weight	FLOAT	Вес пользователя
height	FLOAT	Рост пользователя

Коллекция Activities содержит информацию о видах активностей веб-приложения. Структура данной коллекции приведена в таблице 2.10.

Таблица 2.10 – Структура коллекции Activities

Название поля	Тип данных	Описание
activity_id	INT	Идентификатор активности
activity_type	STIRNG	Тип активности
activity_name	STIRNG	Название активности

Коллекция DailyActivity содержит информацию о ежедневной активности веб-приложения. Структура данной коллекции приведена в таблице 2.11.

Таблица 2.11 – Структура коллекции DailyActivity

Название поля	Тип данных	Описание
dailyActivityId	INT	Идентификатор ежедневной активности
stepQuantity	INT	Количество шагов
otherActivityTime	FLOAT	Время общей активности
userId	INT	Идентификатор пользователя
date	Date	Дата внесения записи

Коллекция Friendship содержит информацию о дружбе между пользователями веб-приложения. Структура данной коллекции приведена в таблице 2.12.

Таблица 2.12 – Структура коллекции Friendship

Название поля	Тип данных	Описание
friend_id	INT	Идентификатор дружбы
user1_id	INT	Идентификатор пользователя, иницилирующего дружбу
user2_id	INT	Идентификатор пользователя, принявшего запрос на дружбу
friendship_date	Date	Дата заключения дружбы
isPending	Bool	Статус обработки запроса на дружбу

Каждая коллекция имеет четко определенные поля, отражающие определенные аспекты работы веб-приложения «Генератор спортивных вызовов».

2.3 Архитектура веб-приложения

Структурная схема веб-приложения представлена в приложении Г.

Пояснение назначения каждого элемента веб-приложения представлено в таблице 2.13.

Таблица 2.13 – Назначение элементов архитектурной схемы веб –приложения

Элемент	Назначение
R e s	Обрабатывает HTTP-запросы от клиентов, раздаёт статические файлы (HTML, CSS, JavaScript) и проксирует API-запросы к Backend Server через HTTPS.
Backend Server (ASP.NET Core)	Реализует бизнес-логику приложения, обрабатывает API-запросы, взаимодействует с базой данных, управляет аутентификацией пользователей и отправляет электронные уведомления через SMTP.
P o	Хранит структурированные данные, включая информацию о пользователях, стажировках, заявках, категориях и навыках.
Docker	Используется для контейнеризации приложений, что обеспечивает их изоляцию, портативность и упрощение развертывания.
Entity Framework	Используется для работы с базой данных через объектно-ориентированные модели, упрощая создание, обновление и запрос данных.
Сервисы	Используются для реализации бизнес-логики и предоставления функционала приложения через интерфейсы для взаимодействия с другими компонентами системы.

Описание протоколов, используемых при работе веб-приложений, представлено в таблице 2.14.

Таблица 2.13 – Назначение элементов архитектурной схемы веб –приложения

Протокол	Назначение
	Обеспечивает безопасное зашифрованное соединение между клиентами и сервером, защищая данные от перехвата.
	Обеспечивает передачу данных между клиентом и сервером
TCP	Обеспечивает надежную передачу данных между клиентом и сервером через стабильное соединение с гарантией доставки и правильного порядка сообщений.

2.4 Выводы по разделу

Таким образом, было спроектировано веб-приложение, обладающее следующими особенностями:

1. Поддержка трех ролей с четко разграниченными правами доступа и функциональными возможностями: гость, пользователь, администратор.
2. Спроектирована база данных для веб-приложения, которая состоит из семи коллекций. Эти коллекции охватывают все аспекты работы веб-приложения

Веб-приложение разработано на чистой архитектуре на ASP .NET, PostgreSQL для хранения данных, Docker Compose для запуска многоконтейнерных Docker-приложений.

3 Реализация веб-приложения

3.1 Обоснование выбора программной платформы

Для разработки серверной части приложения была выбрана платформа обеспечивает эффективную обработку запросов и поддержку масштабируемых решений, что позволяет приложению выдерживать большое количество пользователей и нагрузок. Платформа предлагает встроенные механизмы безопасности, включая аутентификацию и защиту от распространенных веб-угроз, что гарантирует защиту данных пользователей. Кроме того, ASP.NET обладает отличной интеграцией с различными базами данных, включая PostgreSQL, что обеспечивает гибкость в управлении данными и их обработке.

3.2 Система управления базами данных PostgreSQL

Для веб-приложения выбрана PostgreSQL — это реляционная система управления базами данных, которая использует таблицы для организации данных. Данные в PostgreSQL структурированы в виде строк и столбцов, где каждый столбец имеет строго определённый тип данных, что обеспечивает целостность и структурированность информации. PostgreSQL поддерживает сложные связи между таблицами через первичные и внешние ключи, что позволяет эффективно работать с данными и гарантировать их согласованность. Скрипт создания базы данных приведен в Приложении А.

ApplicationDbContext

Для генерации моделей в проекте использовался ApplicationDbContext. Модели представляются в виде классов C#. Коллекция DbSet описывает набор сущностей.

Код, описывающий модель Users, приведёт в листинге 3.1

```
public class Users
{
    public int user_id { get; set; }
    public string username { get; set; }
    public string password { get; set; }
    public string birthday { get; set; }
    public string user_role { get; set; } = "User"; }
```

Листинг 3.1 – Модель Users

Модель Users в C# описывает пользователя и включает несколько ключевых полей. Поле user_id представляет уникальный идентификатор пользователя, а username содержит имя пользователя. Пароль хранится в поле password, а поле

birthday предназначено для хранения даты рождения. Роль пользователя задается в поле user_role, которое по умолчанию имеет значение "User".

Код, описывающий модель Calls, приведёт в листинге 3.2

```
public class Calls
{
    public int call_id { get; set; }
    public string call_name { get; set; }
    public int? friend_id { get; set; }
    public string call_date { get; set; }
    public string status { get; set; }
    public string description { get; set; }
    public int user_id { get; set; }
}
```

Листинг 3.2 – Модель Calls

Модель Calls в C# описывает звонок и включает несколько ключевых полей. Поле call_id представляет уникальный идентификатор звонка, а call_name содержит название звонка. Поле friend_id указывает идентификатор друга, если он связан с данным звонком, а call_date содержит дату звонка. Статус звонка фиксируется в поле status, а дополнительная информация о звонке записывается в поле description. Поле user_id связывает звонок с конкретным пользователем.

Код, описывающий модель Challenges, приведёт в листинге 3.3

```
public class Challenge
{
    public int ChallengeId { get; set; }
    public int SenderId { get; set; }
    public int ReceiverId { get; set; }
    public int CallId { get; set; }
    public string Status { get; set; }
    public DateTime SentAt { get; set; }
    public DateTime? RespondedAt { get; set; }
    public Users Sender { get; set; }
    public Users Receiver { get; set; }
    public Calls Call { get; set; }
}
```

Листинг 3.3 – Модель Challenges

Модель Challenge в C# описывает вызов между пользователями и содержит основные поля для управления информацией о вызове. Поле ChallengeId представляет уникальный идентификатор вызова. Поля SenderId и ReceiverId фиксируют идентификаторы отправителя и получателя вызова соответственно, а CallId указывает на связанный с вызовом звонок. Статус вызова хранится в поле Status, дата отправки записывается в поле SentAt, а дата ответа фиксируется в поле RespondedAt, если она имеется.

Код, описывающий модель Activities, приведёт в листинге 3.4

```

public class Activities
{
    public int activity_id { get; set; }
    public string activity_name { get; set; }
    public string activity_type { get; set; }

    [JsonIgnore]
    public ICollection<UserData> UserData { get; set; }
}

```

Листинг 3.4 – Модель Activities

Модель Activities в C# описывает различные виды активности и содержит ключевые поля для хранения информации о них. Поле activity_id представляет уникальный идентификатор активности, поле activity_name хранит название активности, а activity_type фиксирует тип активности, например, спортивная или социальная.

Код, описывающий модель UserData, приведёт в листинге 3.5

```

public class UserData {
    public int data_id { get; set; }
    public int user_id { get; set; }
    public int activity_id { get; set; }
    public DateTime date_info { get; set; }
    public float weight { get; set; }
    public float height { get; set; } }

```

Листинг 3.5 – Модель UserData

Модель UserData в C# описывает данные пользователя, связанные с его активностями, и включает ключевые поля для хранения этой информации. Поле data_id представляет уникальный идентификатор записи, поле user_id связывает запись с конкретным пользователем, а activity_id указывает на связанную активность. Поле date_info хранит дату, на которую относятся данные, а поля weight и height содержат информацию о весе и росте пользователя на указанную дату.

Код, описывающий модель DailyActivity, приведёт в листинге 3.6

```

public class DailyActivity
{
    public int dailyActivityId { get; set; }
    public int stepQuantity { get; set; }
    public float? otherActivityTime { get; set; }
    public int userId { get; set; }
    public DateTime date { get; set; }

    [JsonIgnore]
    public Users User { get; set; }
}

```

Листинг 3.6 – Модель DailyActivity

Модель DailyActivity в C# описывает ежедневную активность пользователя и содержит ключевые поля для хранения информации. Поле dailyActivityId представляет уникальный идентификатор записи активности. Поле stepQuantity хранит количество шагов, выполненных пользователем за день, а поле otherActivityTime (с возможным значением null) фиксирует время, потраченное на другие виды активности. Поле userId связывает запись с конкретным пользователем, а поле date содержит дату, на которую относится эта активность.

Код, описывающий модель Friendship, приведёт в листинге 3.7

```
public class Friendship
{
    public int friend_id { get; set; }
    public int user1_id { get; set; }
    public int user2_id { get; set; }
    public string friendship_date { get; set; }
    public bool IsPending { get; set; } }
```

Листинг 3.7 – Модель Friendship

Модель Friendship в C# описывает отношения дружбы между пользователями и содержит ключевые поля для хранения информации. Поле friend_id представляет уникальный идентификатор дружбы, а поля user1_id и user2_id указывают на идентификаторы пользователей, участвующих в дружбе. Поле friendship_date хранит дату установления дружбы, а поле IsPending определяет, находится ли заявка на дружбу в ожидании подтверждения.

3.4 Программные библиотеки

В процессе разработки серверной части веб-приложения для обеспечения её функциональности и повышения эффективности работы системы были использованы программные библиотеки, представленные в таблице 3.2.

Таблица 3.2 – Программные библиотеки серверной части

Библиотека	Версия	Назначение
Npgsql [8]		Открытый провайдер данных для PostgreSQL, разработанный для платформы .NET
		Промежуточное программное обеспечение ASP.NET Core, которое позволяет приложению принимать маркер-носитель OpenID Connect.
		Объектно-реляционный маппер (ORM) для платформы .NET. Он поддерживает LINQ-запросы, отслеживание изменений, обновление данных и миграции схемы.

Продолжение таблицы 3.2		
[6]		Использует вариант схемы формирования ключей алгоритма шифрования Blowfish и вводит фактор нагрузки (work factor), который позволяет определить, насколько ресурсоемкой будет хеш-функция, обеспечивая "защищенность на будущее".

В процессе разработки клиентской части веб-приложения были задействованы программные библиотеки, представленные в таблице 3.3.

Таблица 3.3 – Программные библиотеки клиентской части

Библиотека	Версия	Назначение
		Основная библиотека для создания пользовательских интерфейсов на основе компонентов.
		Библиотека, используемая для рендеринга компонентов React в DOM, обеспечивая взаимодействие между React и веб-браузером.
[9]		Библиотека для маршрутизации в приложениях React.
		Библиотека компонентов пользовательского интерфейса от Material-UI.
		Библиотека для выполнения HTTP-запросов.
[10]		Библиотека для отображения уведомлений (тостов) в React-приложениях.
		Утилита для декодирования JSON Web Token

Программные библиотеки позволяют упростить реализацию веб-приложения.

3.5 Структура серверной части

Основные компоненты структуры серверной части включают в себя не несколько ключевых элементов, которые обеспечивают эффективную работу приложения:

1. Маршрутизаторы — управляют маршрутами и направляют запросы к соответствующим контроллерам.
2. Контроллеры — обрабатывают запросы от клиента, выполняют бизнес-логику через сервисы и возвращают ответы.
3. Middleware — промежуточные обработчики, используемые для валидации данных и обеспечения безопасности.

Серверная часть представляет из себя приложение, написанное на чистой архитектуре. В таблице 3.4 приведён состав приложения.

Таблица 3.4 – Состав серверной части

Проект	Назначение
Generator.API	Веб-приложение, содержащее к
Generator.Domain	Библиотека классов, содержащая модели.
Generator.Infrastructure	Библиотека классов, содержащая миграции, репозитории, интерфейсы репозитория и контекст БД.
Generator.Application	Библиотека классов, содержащая сервисы, интерфейсы сервисов и класс

Таблица соответствия маршрутов контроллерам и функциям в исходном коде представлена в таблице 3.5.

Таблица 3.6 – Контроллеры и функции маршрутов

Метод	Маршрут	Контроллер	Метод контроллера
POST	/account/register	AccountController	Register
POST	/account/login	AccountController	Login
POST	/account/logout	AccountController	Logout
POST	/activity/	ActivityController	AddActivity
DELETE	/activity/{id}	ActivityController	DeleteActivity
GET	/activity/activities	ActivityController	GetAllActivities
POST	/activity/add-daily-activity	ActivityController	AddDailyActivity
GET	/activity/daily-activity	ActivityController	GetDailyActivityById
PUT	/activity/update-daily-activity/{id}	ActivityController	UpdateDailyActivity
POST	/call/generate/daily	CallController	GenerateDailyCall
POST	/call/generate/weekly	CallController	GenerateWeeklyCall
POST	/call/generate/monthly	CallController	GenerateMonthlyCall
POST	/call/update-status	CallController	UpdateCallStatus
GET	/call/user-calls	CallController	GetUserCalls
POST	/challenge/send	ChallengeController	SendChallenge
GET	/challenge/received	ChallengeController	GetReceivedChallenges
POST	/friendship/respond	FriendshipController	AddFriend
POST	/userdata/	UserDataController	AddUserData
GET	/stats/admin	StatsController	GetGlobalStats
GET	/stats/user	StatsController	GetUserStats

3.7 Реализация функционала для пользователя с ролью «Гость»

Регистрация

Для гостя доступна регистрация, которая позволяет ему создать учетную запись в системе. Эта возможность реализована в контроллере AccountController. Реализация этого метода приведена в листинге 3.8.

```
[HttpPost("register")]
public IActionResult Register([FromBody] RegisterDto registerDto)
{
    if (_unitOfWork.Users.UserExists(registerDto.username))
    {
        return BadRequest("User already exists.");
    }
    var hashedPassword =
BCrypt.Net.BCrypt.HashPassword(registerDto.password);

    var user = new Users
    {
        username = registerDto.username,
        password = hashedPassword,
        birthday = registerDto.birthday
    };
    _unitOfWork.Users.Add(user);
    _unitOfWork.Commit();

    return Ok("User registered successfully.");
}
```

Листинг 3.8 – Реализация HTTP-метода для регистрации

В данном методе контроллера происходит регистрация нового пользователя. Сначала проверяется, существует ли пользователь с указанным именем в базе данных. Если пользователь уже существует, возвращается сообщение об ошибке. Если пользователя нет, пароль из входящих данных хешируется с использованием BCrypt, после чего создается новый объект пользователя с заданными именем, хешированным паролем и датой рождения. Этот объект добавляется в базу данных через единицу работы (Unit of Work), и изменения сохраняются. В завершение возвращается подтверждение успешной регистрации.

Авторизация

Для гостя доступна авторизация, которая позволяет ему войти в свою учетную запись в системе. Эта возможность реализована в контроллере AccountController. В реализации метода используется паттерн Unit Of Work. При успешной авторизации пользователю выдаётся токен.

Реализация этого метода приведена в листинге 3.9.

```
[HttpPost("login")]
```

```

public IActionResult Login([FromBody] LoginDto loginDto)
{
    var user = _unitOfWork.Users.GetByUsername(loginDto.username);

    if (user == null || !BCrypt.Net.BCrypt.Verify(loginDto.password,
user.password))
    {
        return Unauthorized("Invalid username or password.");
    }

    var token = _tokenService.GenerateToken(user);

    return Ok(new { Token = token });
}

```

Листинг 3.9 – Реализация HTTP-метода для аутентификации

В данном методе контроллера реализован процесс аутентификации пользователя. Сначала проверяется наличие пользователя с указанным именем в базе данных. Если пользователь не найден или введенный пароль не соответствует сохраненному хешу, возвращается сообщение о несанкционированном доступе. В случае успешной проверки для пользователя генерируется токен с помощью службы токенов, и этот токен отправляется в ответе.

Реализация функционала для пользователя с ролью «пользователь»

Добавление пользовательских данных

Пользователи веб-приложения «Генератор спортивных вызовов» имеют возможность заполнить форму со своими данными, на основании которых генерируется специализированный вызов.

В данном методе контроллера реализуется добавление данных пользователя. Метод защищен политикой авторизации "UserPolicy". Сначала проверяется, что переданные данные пользователя не равны null. Затем проверяется, авторизован ли пользователь, извлекая информацию из JWT-токена и проверяя наличие соответствующих утверждений (claims).

В этом методе контроллера реализуется добавление данных пользователя. Метод защищен политикой авторизации "UserPolicy". Сначала проверяется, что переданные данные пользователя не равны null. Затем проверяется, авторизован ли пользователь, извлекая информацию из JWT-токена и проверяя наличие соответствующих утверждений (claims).

Здесь реализуется добавление данных пользователя. Метод защищен политикой авторизации "UserPolicy". Сначала проверяется, что переданные данные пользователя не равны null. Затем проверяется, авторизован ли пользователь, извлекая информацию из JWT-токена и проверяя наличие соответствующих утверждений (claims).

Реализация метода добавления данных приведена в листинге 3.10.

```

[Authorize(Policy = "UserPolicy")]
[HttpPost]
public async Task<IActionResult> AddUserData([FromBody] UserDataDto
userDataDto)
{
    if (userDataDto == null)
        return BadRequest("User data cannot be null.");
    var token = Request.Headers["Authorization"].ToString();
    var userNameClaim = User.Claims.FirstOrDefault();
    if (userNameClaim == null)
        { return Unauthorized("User is not authorized."); }
    string userName = userNameClaim.Value;
    var user = await _context.Users
        .FirstOrDefaultAsync(u => u.username == userName);
    if (user == null)
        return BadRequest($"User with name '{userName}' not
found.");
    var activity = await _context.Activities
        .FirstOrDefaultAsync(a => a.activity_name ==
userDataDto.activity_name);
    if (activity == null)
        return BadRequest($"Activity with name
'{userDataDto.activity_name}' not found.");
    var userData = new UserData
    {
        user_id = user.user_id,
        activity_id = activity.activity_id,
weight = userDataDto.weight,
        height = userDataDto.height
    };
    try
    {
        _context.UserData.Add(userData);
        await _context.SaveChangesAsync();
        return CreatedAtAction(nameof(GetUserData), new { id =
userData.data_id }, userData);
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"Internal server error:
{ex.Message}");
    }
}

```

Листинг 3.10 – Реализация HTTP-метода добавления данных

В данном методе контроллера реализуется добавление данных пользователя. Метод защищен политикой авторизации "UserPolicy". Сначала проверяется, что переданные данные пользователя не равны null. Затем проверяется, авторизован ли пользователь, извлекая информацию из JWT-токена и проверяя наличие соответствующих утверждений (claims).

После извлечения имени пользователя из токена осуществляется поиск пользователя в базе данных. Если пользователь или указанный тип активности не найдены, возвращаются соответствующие сообщения об ошибке. Если данные корректны, создается новый объект `UserData`, который связывает пользователя с активностью, а также сохраняет вес и рост из переданных данных.

Попытка добавления нового объекта `UserData` осуществляется в блоке `try-catch`. При успешном добавлении данных возвращается статус 201 (Created) с информацией о добавленных данных, а в случае ошибки — сообщение об ошибке сервера с кодом 500.

3.8.2 Выбор предпочитаемого вида активности

Выбор предпочитаемого вида активности осуществляется при заполнении пользователем формы со своими данными. Пользователю предоставляется возможность выбрать тип активности, к примеру, командный спорт или лёгкая атлетика. На основе выбора типа отфильтровываются варианты активности для конкретного типа для упрощения поиска необходимого.

Получение сгенерированного вызова

Пользователи веб-приложения «Генератор спортивных вызовов» могут получить ежедневный, еженедельный и ежемесячный вызовы. Реализация метода генерации ежедневного вызова приведена в листинге 3.11.

```
[HttpPost("generate/daily")]
public async Task<IActionResult> GenerateDailyCall([FromQuery]
string username, [FromQuery] int? friendId = null) {
    if (string.IsNullOrEmpty(username))
        return BadRequest("Имя пользователя отсутствует.");
    var user = await _context.Users.Include(u => u.UserData)
        .FirstOrDefaultAsync(u => u.username == username);
    if (user == null)
        return NotFound("Пользователь не найден.");
    var call = _challengeGeneratorService.GenerateDailyCall(user,
friendId);
    _context.Calls.Add(call);
    await _context.SaveChangesAsync();
    return Ok(call); }
```

Листинг 3.11 – Реализация HTTP-метода для генерации ежедневного вызова

В данном методе контроллера реализуется генерация ежедневного вызова для пользователя. Метод защищен атрибутом `[Authorize]`, что требует авторизации для его вызова. Сначала проверяется, что имя пользователя передано и не пустое. Затем в базе данных осуществляется поиск пользователя

с указанным именем, включая связанные данные через навигационное свойство `UserData`. Если пользователь не найден, возвращается сообщение об ошибке.

После успешного поиска пользователя используется сервис `ChallengeGeneratorService` для генерации вызова, который может включать идентификатор друга, если он передан. Сгенерированный вызов добавляется в базу данных через контекст `Calls`, и изменения сохраняются. В случае успеха метод возвращает созданный вызов в ответе.

Аналогично выполнены методы для генерации еженедельного и ежемесячного вызовов.

3.8.4 Ввод ежедневной активности

```
[HttpPost("add-daily-activity")]
public async Task<IActionResult> AddDailyActivity([FromBody]
DailyActivityDto dailyActivityDto)
{
    try {
        var existingActivity = await _context.DailyActivities
.FirstOrDefaultAsync(a => a.userId == dailyActivityDto.UserId &&
a.date.Date == DateTime.UtcNow.Date);
        if (existingActivity != null)
            return Conflict("Активность для этого пользователя на
сегодня уже существует.");
        var user = await _context.Users.FirstOrDefaultAsync(u =>
u.user_id == dailyActivityDto.UserId);
        var dailyActivity = new DailyActivity
        {
            stepQuantity = dailyActivityDto.StepQuantity,
            otherActivityTime =
dailyActivityDto.OtherActivityTime.Value,
            userId = dailyActivityDto.UserId,
            date = DateTime.UtcNow,
            User = user
        };
        _context.DailyActivities.Add(dailyActivity);
        await _context.SaveChangesAsync();
        return Ok("Активность успешно добавлена.");
    }
    catch (Exception ex) {
        return StatusCode(500, "Произошла ошибка при добавлении
активности.");
    }
}
```

Листинг 3.12 - Реализация HTTP-метода для добавления ежедневной активности

Этот метод добавляет информацию о ежедневной активности пользователя. Он сначала проверяет, существует ли уже запись активности для указанного пользователя на текущую дату. Если такая запись есть,

возвращается ошибка с уведомлением о конфликте. Если активности нет, создается новая запись с указанными данными (например, количество шагов и время другой активности) и сохраняется в базе данных. Если операция успешна, возвращается сообщение о том, что активность успешно добавлена, в противном случае — сообщение об ошибке.

Возможность бросать вызов другим пользователям

```
[HttpPost("send")]
public async Task<IActionResult> SendChallenge([FromBody]
SendChallengeDto dto)
{
    if (dto == null)
        return BadRequest("Данные вызова отсутствуют.");

    var challenge = new Challenge
    {
        SenderId = dto.SenderId,
        ReceiverId = dto.ReceiverId,
        CallId = dto.CallId,
        Status = "Pending",
        SentAt = DateTime.UtcNow
    };

    _context.Challenges.Add(challenge);
    await _context.SaveChangesAsync();

    return Ok("Вызов успешно отправлен.");
}
```

Листинг 3.13 - Реализация HTTP-метода для вызова другу

Этот метод отправляет вызов между пользователями. Он сначала проверяет, были ли переданы данные для вызова, и, если данные отсутствуют, возвращает ошибку с соответствующим сообщением. Затем создается новый объект вызова с указанием отправителя, получателя, идентификатора вызова и статуса "Ожидает". Запись сохраняется в базе данных, и в случае успеха возвращается сообщение о том, что вызов успешно отправлен.

Добавление в друзья

В этом примере представлена реализация метода для отправки запроса на добавление дружбы между двумя пользователями. Метод выполняет проверку данных, существования пользователей и уже существующих запросов дружбы перед созданием нового запроса в базе данных.

```
[HttpPost("add")]
public async Task<IActionResult> AddFriend([FromBody] FriendshipDto
dto)
{
    Console.WriteLine($"Получены данные: user1_id={dto.user1_id},
user2_id={dto.user2_id}");
}
```

```

        if (dto == null)
        {
            return BadRequest("Friendship data is missing.");
        }
        var user1 = await _context.Users.FindAsync(dto.user1_id);
        var user2 = await _context.Users.FindAsync(dto.user2_id);

        if (user1 == null || user2 == null)
        {
            return NotFound("One or both users not found.");
        }
        var exists = await _context.Friendships.AnyAsync(f =>
            (f.user1_id == dto.user1_id && f.user2_id == dto.user2_id)
||
            (f.user1_id == dto.user2_id && f.user2_id == dto.user1_id));

        if (exists)
        {
            return BadRequest("Friendship already exists or is
pending.");
        }
        var friendship = new Friendship
        {
            user1_id = dto.user1_id,
            user2_id = dto.user2_id,
            friendship_date = DateTime.UtcNow.ToString(),
            IsPending = true
        };

        _context.Friendships.Add(friendship);

        try
        {
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateException ex)
        {
            return StatusCode(500, $"Database error: {ex.Message}");
        }

        return Ok("Friend request sent.");
    }
}

```

Листинг 3.13 - Реализация HTTP-метода для добавления в друзья

Этот метод отправляет запрос на добавление друга между двумя пользователями. Он сначала проверяет, были ли переданы данные о пользователях, и, если данные отсутствуют, возвращает ошибку. Далее происходит проверка существования обоих пользователей в базе данных. Если хотя бы одного пользователя не удастся найти, возвращается ошибка. Затем проверяется, существует ли уже дружба между этими пользователями или она

находится в ожидании. Если дружба уже есть, возвращается ошибка. Если все проверки пройдены, создается новый запрос на дружбу с установленным статусом "Ожидает". Запись сохраняется в базе данных, и при успешном выполнении операции возвращается сообщение о том, что запрос дружбы был отправлен. В случае ошибки в базе данных возвращается сообщение с подробностями об ошибке.

Реализация функционала для пользователя с ролью «Администратор»

Добавление активности

Администратор системы имеет возможность создавать виды активности. Реализация метода контроллера создания активностей представлена на листинге 3.14

```
[Authorize(Roles = "Admin")]
[HttpPost]
public IActionResult AddActivity([FromBody] ActivityDto activityDto)
{
    if (activityDto == null)
    {
        return BadRequest("Activity data is required.");
    }

    var activity = new Activities
    {
        activity_name = activityDto.activity_name,
        activity_type = activityDto.activity_type
    };

    _activityService.AddActivity(activity);

    return CreatedAtAction(nameof(GetActivityById), new { id =
activity.activity_id }, activity);
}
```

Листинг 3.14 – Создание активности

В данном методе реализована функция добавления новой активности. Метод защищен атрибутом `[Authorize(Roles = "Admin")]`, что ограничивает доступ только для пользователей с ролью администратора. Сначала проверяется, что переданные данные не равны `null`. Если данные отсутствуют, возвращается ошибка с соответствующим сообщением.

Далее создается объект `Activities` с использованием данных из `ActivityDto`, включая название активности и её тип. Новый объект передается в сервис `_activityService` для добавления в базу данных.

После успешного добавления метод возвращает ответ с кодом 201 (Created), включая ссылку на метод `GetActivityById` для получения добавленной активности и сам объект активности.

Ведение статистики

```
public async Task<IActionResult> GetGlobalStats() {
    var now = DateTime.UtcNow;
    var currentYear = now.Year;
    var currentMonth = now.Month;
    var topUsers = await _context.Calls
        .Where(c => c.status == "completed").GroupBy(c => c.user_id)
        .Select(g => new{
            g.Key, CompletedCount = g.Count(),
            Categories = g.GroupBy(c => c.call_name)
                .Select(cg => new{ cg.Key,
                    MonthlyCompleted = cg.Count(c =>
DateTime.TryParse(c.call_date, out var date) && date.Year ==
currentYear && date.Month == currentMonth),
                    YearlyCompleted = cg.Count(c =>
DateTime.TryParse(c.call_date, out var date) && date.Year ==
currentYear) }).ToList()}).OrderByDescending(x => x.CompletedCount)
        .Take(10).ToListAsync();
    var usersDict = await _context.Users
        .Where(u => topUsers.Select(u => u.Key).Contains(u.user_id))
        .ToDictionaryAsync(u => u.user_id, u => u.username);
    var result = new{
        totalMonthlyCompleted = topUsers.Sum(u => u.Categories.Sum(c
=> c.MonthlyCompleted)),
        totalYearlyCompleted = topUsers.Sum(u => u.Categories.Sum(c
=> c.YearlyCompleted)),
        topUsers = topUsers.Select(u => new{
            username = usersDict.GetValueOrDefault(u.Key,
$"User#{u.Key}"),
            completedCalls = u.CompletedCount,
            categories = u.Categories.Select(c => new { c.Key,
completedCalls = c.YearlyCompleted }).ToList()).ToList();
    }
    return Ok(result);}
```

Листинг 3.15 – Просмотр глобальной статистики

Администратор может просматривать как глобальную статистику, так и статистику каждого пользователя в отдельности.

Сравнение активности пользователей

```
public async Task<IActionResult> GetUserStats([FromQuery] string
username)
{
    if (string.IsNullOrEmpty(username)) {
        return BadRequest("Имя пользователя не указано.");
    }
    var user = await _context.Users.FirstOrDefaultAsync(u =>
u.username == username);
    if (user == null) {
        return NotFound($"Пользователь {username} не найден.");
    }
}
```

```

    }
    var now = DateTime.UtcNow;
    var currentYear = now.Year;
    var currentMonth = now.Month;
    var userCompletedCalls = await _context.Calls
        .Where(c => c.user_id == user.user_id && c.status ==
"completed")
        .ToListAsync();
    int monthlyCompleted = userCompletedCalls.Count(c =>
        DateTime.TryParse(c.call_date, out DateTime callDate) &&
        callDate.Year == currentYear &&
        callDate.Month == currentMonth);
    int yearlyCompleted = userCompletedCalls.Count(c =>
        DateTime.TryParse(c.call_date, out DateTime callDate) &&
        callDate.Year == currentYear);
    var categoriesStats = userCompletedCalls
        .Where(c => DateTime.TryParse(c.call_date, out DateTime
callDate) &&
            callDate.Year == currentYear)
        .GroupBy(c => c.call_name)
        .Select(g => new
    {
        category = g.Key,
        monthlyCompleted = g.Count(c =>
DateTime.TryParse(c.call_date, out DateTime callDate) &&
callDate.Month == currentMonth),
        yearlyCompleted = g.Count()
    })
        .ToList();
    var result = new
    {
        username = user.username,
        monthlyCompleted,
        yearlyCompleted,
        categoriesStats
    };
    return Ok(result);
}

```

Листинг 3.16 – Просмотр глобальной статистики

3.10 Структура клиентской части

Клиентская часть приложения реализована с использованием компонентного подхода. Основная логика и элементы пользовательского интерфейса размещены в директории `src`. Директории представлены в таблице 3.5.

Таблица 3.5 - Основные директории проекта в папке `src` и их назначение

Директория	Назначение
------------	------------

	Включает React-компоненты, предназначенные для создания элементов пользовательского интерфейса веб-приложения.
	Глобальное состояние для управления данными между компонентами React.
	Хранит маршруты веб-приложения для навигации между страницами.
services	Хранит сервисы регистрации и авторизации, а также путь к серверу.

3.11 Выводы по разделу

Таким образом, было реализовано веб-приложение «Генератор спортивных вызовов» со следующими особенностями:

1. Использована программная платформа ASP .NET. Для хранения данных использовалась реляционная СУБД PostgreSQL. Для упрощения взаимодействия с ней применялся ApplicationDbContext, который автоматизировал создание коллекций на основе моделей.

2. Разработана структура веб-приложения, которая базируется на модульном подходе с использованием современных библиотек для клиентской и серверной части. Реализованы все функции для все ролей: гостя, пользователя и администратора. Общее количество функций – 25.

4 Тестирование веб-приложения

4.1 Функциональное тестирование

Для проверки корректности работы всех функций разработанного веб-приложения было проведено ручное тестирование с использованием Postman [12] для отправки запросов на сервер. Описание, ожидаемый результат и итоги тестирования представлены в таблице 4.1.

Таблица 4.1 – Описание ручного тестирования веб-приложения

№	Функция веб-приложения	Описание тестирования	Ожидаемый результат	Полученный
	Регистрация	Отправить POST-запрос на /register с именем пользователя newUser, паролем password123 и д	Код 200 OK, с о о б	Совпадает с ожидаемым.
	Аутентификация	Отправить POST-запрос на /login с именем пользователя	Код 200 OK, возвращен токен авторизации в формате JSON.	Совпадает с ожидаемым.
	Добавление пользовательских данных	Отправить авторизованный POST-запрос на /AddUserData с корректным объектом содержащим существующее имя активности и данные пользователя (вес и рост).	Код 201 Created, объект UserData добавлен в базу данных, появилась запись в базе данных.	Совпадает с ожидаемым.
	Добавление в друзья	Отправить POST-запрос на /add с корректным объектом FriendshipDto, содержащим идентификаторы двух существующих пользователей	Код 200 OK, с о о б щ е	Совпадает с ожидаемым.

Продолжение таблицы 4.1

		и user2_id), которые еще не находятся в		
--	--	---	--	--

		отношениях дружбы.		
	Отправление заявки в друзья	Отправить GET-запрос на существующим есть ожидающие запросы на дружбу	Код 200 ОК, список уведомлений с информацией об отправителях запросов на дружбу. Если запросов нет, возвращается пустой список.	Совпадает с ожидаемым.
	Обработка заявки в друзья	Отправить POST-запрос на /respond с корректным объектом содержащим идентификаторы двух пользователей, и параметром ассерт, установленным в или false для отклонения запроса.	Код 200 ОК, если запрос принят — сообщение "Friend дружбы обновлен в базе. Если запрос отклонен — сообщение "Friend запись удалена из базы данных.	Совпадает с ожидаемым.
	Получение глобальной статистики активности пользователей	Отправить GET-запрос на /admin для получения глобальной статистики, включая общее количество выполненных вызовов за месяц и год, а также топ-10 пользователей с количеством выполненных вызовов и их распределением по категориям.	Код 200 ОК, возвращается объект с полями и topUsers. Поле список пользователей с их именами, количеством выполненных вызовов и категориями вызовов.	Совпадает с ожидаемым.

Продолжение таблицы 4.1

	Получение статистики	Отправить GET-запрос на /user, указав	Код 200 ОК, возвращается объект с полями username,	Совпадает с ожидаемым.
--	----------------------	---------------------------------------	--	------------------------

	конкретного пользователя	существующее имя пользователя в параметре username.	monthlyCompleted, yearlyCompleted и categoriesStats. Поле содержит список категорий с количеством выполненных вызовов за месяц и год.	
	Получение еженедельного вызова	Отправить авторизованный POST-запрос на указав существующее имя пользователя в параметре username и, при необходимости,	Код 200 ОК, возвращается объект вызова с данными о созданном еженедельном вызове, сохраненном в базе данных.	Совпадает с ожидаемым.
	Получение списка вызовов пользователя	Отправить авторизованный GET -	Отправить авторизованный GET -	Совпадает с ожидаемым.
	Установка нового статуса	Отправить авторизованный POST-запрос на существующий идентификатор вызова (CallId) и новый статус (Status) в теле запроса.	Отправить авторизованный POST-запрос на существующий идентификатор вызова (CallId) и новый статус (Status) в теле запроса.	Совпадает с ожидаемым.

Таким образом, были протестированы ключевые функции веб-приложения, включая основные аспекты его работы. В ходе тестирования проверялась корректность обработки запросов на регистрацию, авторизацию и управление пользовательскими данными. Особое внимание было уделено функциональности, связанной с вызовами, включая их создание, обновление статусов и получение статистики. Также были протестированы механизмы

взаимодействия между пользователями, такие как отправка и обработка запросов на дружбу.

Тестирование охватило как позитивные сценарии, когда действия выполнялись в рамках ожидаемого поведения, так и негативные сценарии, направленные на проверку обработки ошибок и исключительных ситуаций. Это позволило убедиться в надежности и устойчивости приложения к некорректным данным и нештатным ситуациям.

Результаты тестирования подтверждают, что приложение корректно обрабатывает запросы, возвращает ожидаемые ответы и успешно сохраняет изменения в базе данных. Проведенное тестирование обеспечивает уверенность в стабильной работе ключевых функций веб-приложения и его готовности к использованию пользователями.

4.2 Выводы по разделу

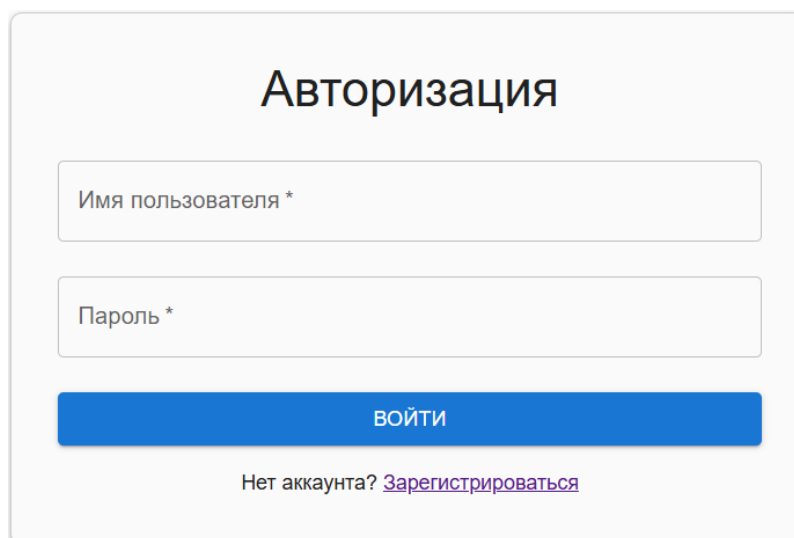
1. Проведено ручное тестирование всех основных функций веб-приложения.
2. Функциональность системы проверена на соответствие ожидаемым результатам, что подтверждает её корректную работу.
3. Количество выполненных тестов составило 11, что обеспечивает покрытие тестами на уровне 80%.

Руководство пользователя

5.1 Руководство пользователя для роли «Гость»

Авторизация

При запуске веб-приложения открывается страница авторизации, представленная на рисунке 5.1.

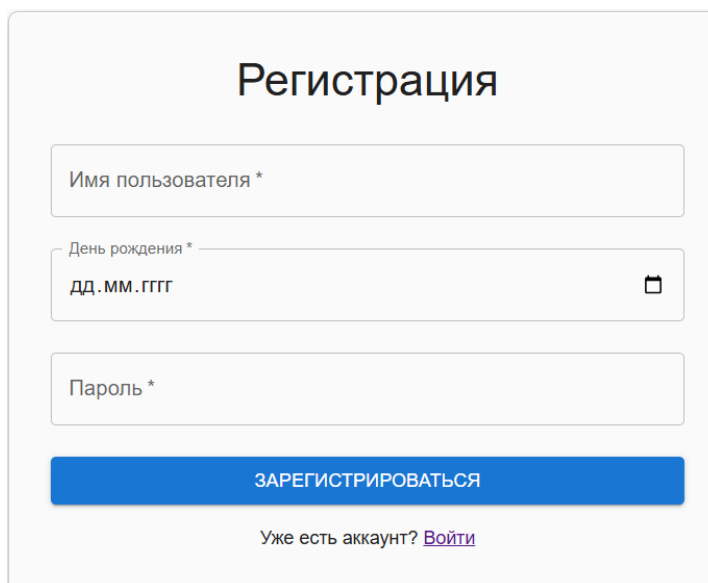


The screenshot shows a login form with the title 'Авторизация' at the top. Below the title are two input fields: 'Имя пользователя *' and 'Пароль *'. A blue button labeled 'ВОЙТИ' is positioned below the password field. At the bottom, there is a link that says 'Нет аккаунта? [Зарегистрироваться](#)'.

Рисунок 5.1 – Страница с формой авторизации

Регистрация

Если у пользователя ещё нет аккаунта, он может перейти на страницу регистрации, нажав на ссылку «Зарегистрироваться». При нажатии на ссылку, происходит переход на страницу регистрации, которая представлена на рисунке 5.2.



The screenshot shows a registration form with the title 'Регистрация' at the top. Below the title are three input fields: 'Имя пользователя *', 'День рождения *' (with a date format 'ДД.ММ.ГГГГ' and a calendar icon), and 'Пароль *'. A blue button labeled 'ЗАРЕГИСТРИРОВАТЬСЯ' is positioned below the password field. At the bottom, there is a link that says 'Уже есть аккаунт? [Войти](#)'.

Рисунок 5.2 – Страница регистрации

После успешной регистрации, происходит переадресация на страницу авторизации, где гость может ввести свои данные и войти в систему.

5.2 Руководство пользователя для роли «Пользователь»

5.2.1 Добавление пользовательских данных

После успешной авторизации, пользователь переходит на страницу пользователя, которая представлена на рисунке 5.3.

Добавить данные пользователя

[ВВЕСТИ АКТИВНОСТЬ ЗА СЕГОДНЯ](#) [ПРОСМОТРЕТЬ СВОЮ АКТИВНОСТЬ](#)

Тип активности
Team sport

Название активности
Football

Вес (кг)
74

Рост (см)
184

[ДОБАВИТЬ](#)

Ваши данные

Дата	Тип активности	Название активности	Вес (кг)	Рост (см)
04.01.2025	Gym	Powerlifting	75	184

Рисунок 5.3 – Добавление данных пользователя

Пользователь может ввести какие-либо данные о себе в специальную форму. При успешном заполнении, данные будут выведены на странице пользователя.

Выбор предпочитаемого вида активности

Тип активности
Командный спорт

Название активности
Выберите активность

Выберите активность
Хоккей

Рисунок 5.4 – Выбор активности

Пользователь может выбрать предпочитаемый вид активности из списка.

Получение сгенерированного задания

Основной функцией приложения является генерация спортивных вызовов. При наличии записей данных о пользователе, генерация становится более специализированной. Чтобы сгенерировать вызов, нужно нажать кнопку «Получить»

вызов» с соответствующем окне. В зависимости от того, какой тип вызова выбран: ежедневный, еженедельный или ежемесячный, на такой срок и выдаётся задание. Пример предложения задания представлен на рисунке 5.5.

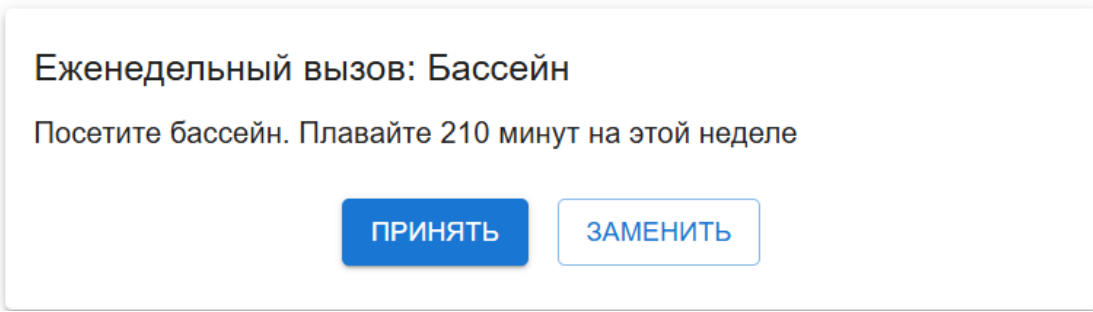


Рисунок 5.5 – Сгенерированный вызов

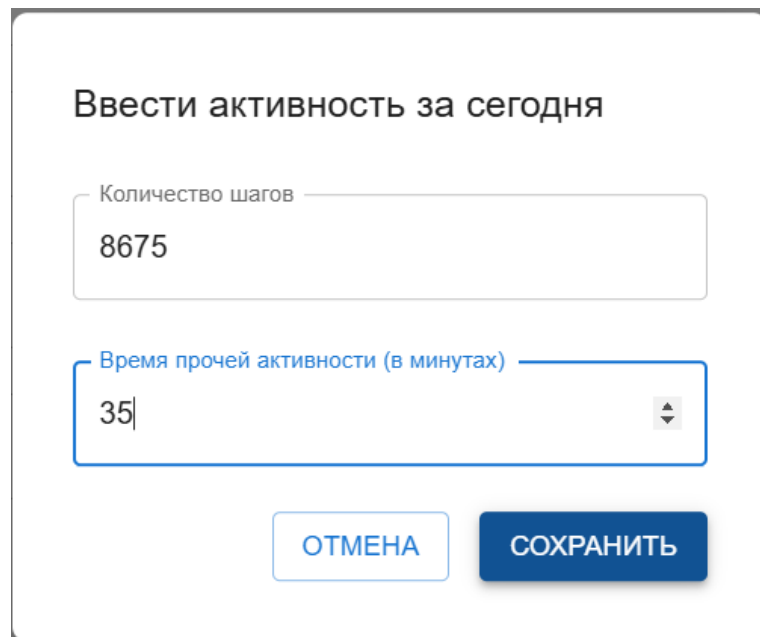
Вызов можно либо принять, либо отклонить. Принятый вызов заносится в список вызовов пользователя, где можно поменять статус или бросить вызов другу. Список вызовов пользователя представлен на рисунке 5.6.

Название	Описание	Дата	Статус	Действия
Еженедельный вызов: Занятие в тренажёрном зале	Посетите тренажёрный зал. Проведите тренировку длительностью 938 минут на этой неделе	2025-01-03	rejected	ВЫПОЛНЕНО
				НЕ ВЫПОЛНЕНО
				БРОСИТЬ ВЫЗОВ
Ежемесячный вызов: Беговые упражнения	Сделайте 1170 минут беговых упражнений в этом месяце	2025-01-03	rejected	ВЫПОЛНЕНО
				НЕ ВЫПОЛНЕНО
				БРОСИТЬ ВЫЗОВ
				ВЫПОЛНЕНО

Рисунок 5.6 – Список вызовов пользователя

Ввод ежедневной активности

Также пользователь может ввести свою активность за сегодня, нажав на кнопку «Ввести активность за сегодня». Появится модальное окно, где будет предложено ввести количество пройденных за сегодня шагов и время прочей активности. Модальное окно активности будет представлено на рисунке 5.7.



Ввести активность за сегодня

Количество шагов

8675

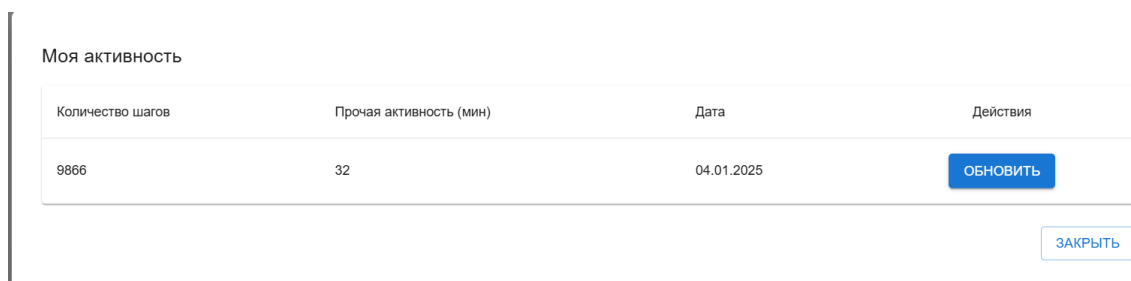
Время прочей активности (в минутах)

35

ОТМЕНА СОХРАНИТЬ

Рисунок 5.7 – Модальное окно активности

В случае, если пользователь ошибся при вводе данных в модальном окне, можно нажать кнопку «Просмотреть свою активность», где данные, внесённые пользователем, можно отредактировать. Модальное окно редактирования и просмотра активностей представлено на рисунке 5.8.



Моя активность

Количество шагов	Прочая активность (мин)	Дата	Действия
9866	32	04.01.2025	ОБНОВИТЬ

ЗАКРЫТЬ

Рисунок 5.8 – модальное окно редактирования и просмотра активностей

Возможность бросать вызов друзьям

Чтобы предложить пользователю из списка друзей вызов, нужно нажать на кнопку «Бросить вызов». Далее появится модальное окно, где вы можете выбрать друга из своего списка друзей и отправить ему уведомление. Модальное окно подготовки приглашения представлено на рисунке 5.9.

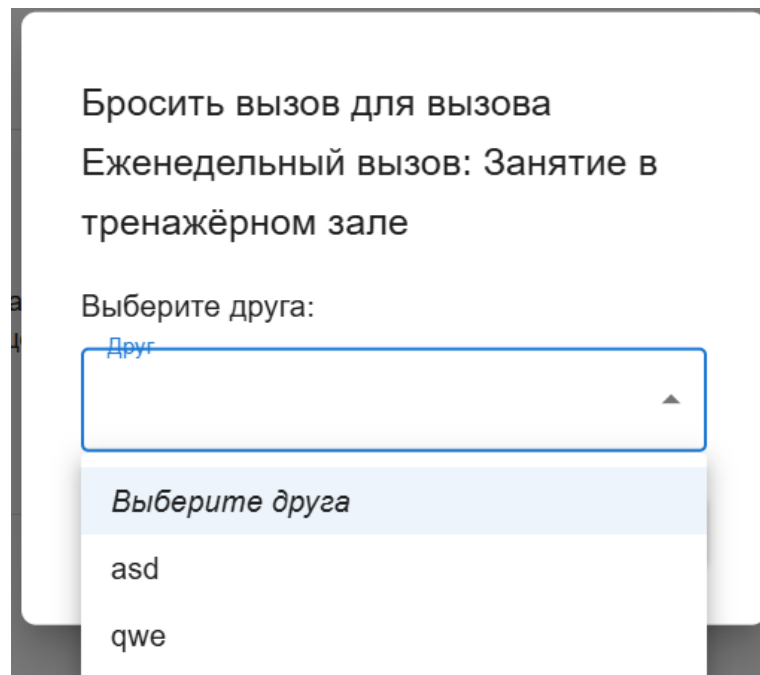


Рисунок 5.9 – Модальное окно для вызова

Добавление в друзья

Также, чтобы найти друга, можно воспользоваться поиском. Далее возле каждого пользователя есть кнопка «Добавить в друзья». При нажатии на кнопку пользователю отправляется запрос на дружбу. Как только заявка будет одобрена, пользователь будет добавлен в друзья. Поиск друзей представлен на рисунке 5.10.

Найти пользователя

Поиск пользователя

s

ИСКАТЬ

Имя пользователя	Действие
asd	ДОБАВИТЬ В ДРУЗЬЯ
sdf	ДОБАВИТЬ В ДРУЗЬЯ
user	ДОБАВИТЬ В ДРУЗЬЯ

Рисунок 5.10 – Поиск друзей

На рисунке 5.11 представлено уведомление заявки пользователю на дружбу, которая появляется при нажатии на кнопку «Добавить в друзья».

Уведомления

Имя пользователя	Тип уведомления	Описание	Действие
cvb	Запрос на дружбу	Не указано	ПРИНЯТЬ ОТКЛОНИТЬ

Рисунок 5.11 – Заявка на добавление в друзья

Теперь, когда пользователь добавлен в друзья, как видно из списка друзей, что изображён на рисунке 5.12, можно бросать другу вызов.

Ваши друзья

Имя пользователя

cvb

Рисунок 5.12 – Список друзей

На рисунке 5.13 изображено уведомление, которое приходит от пользователя из друзей. У пользователя есть возможность принять вызов или отклонить. Сам же вызов заносится в список вызовов.

Уведомления

Имя пользователя	Тип уведомления	Описание	Действие
cvb	Вызов	Посетите тренажерный зал. Проведите тренировку длительностью 938 минут на этой неделе	ПРИНЯТЬ ВЫЗОВ ОТКЛОНИТЬ ВЫЗОВ

Рисунок 5.13 – Вызов пользователю

5.3 Руководство пользователя для роли «Администратор»

Добавление активности

У администратора есть возможность добавлять новый вид активности. Форма добавления представлена на рисунке 5.14.

Добавить новую активность

Название активности

Lifting

Тип активности

Gym

ДОБАВИТЬ

Рисунок 5.14 – Форма добавления активности

Ведение статистики

Администратор может просматривать общую статистику за месяц и за год по категориям. Глобальная статистика представлена на рисунке 5.15

Глобальная статистика

Выполненных вызовов (этот месяц): 7		
Выполненных вызовов (этот год): 7		
Топ-10 пользователей по выполненным вызовам		
Пользователь	Всего выполнено	Категории
cvb	2	Еженедельный вызов: Ходьба: 1 вызов(ов) Ежедневный вызов: Бассейн: 1 вызов(ов)
qwe	2	Ежедневный вызов: Бассейн: 1 вызов(ов) Еженедельный вызов: Ходьба: 1 вызов(ов)
asd	2	Ежедневный вызов: Ходьба: 2 вызов(ов)
sdf	1	Ежедневный вызов: Занятие в тренажёрном зале: 1 вызов(ов)
ОБНОВИТЬ СТАТИСТИКУ		

Рисунок 5.15 – Глобальный поиск

Сравнение активности пользователей

Администратор имеет возможность просмотреть статистику каждого пользователя лично, нажав на кнопку «Статистика». Модальное окно со статистикой отдельного пользователя показано на рисунке 5.16.



Рисунок 5.16 – Модальное окно личной статистики

5.4 Выводы по разделу

1. Разработано руководство, описывающее действия пользователей системы: гостя, зарегистрированного пользователя и администратора, с учетом уникальных функций каждой роли, описанных в диаграмме вариантов использования.
2. Гости могут зарегистрироваться и аутентифицироваться. Зарегистрированные пользователи могут добавлять данные о себе, получать вызовы, принимать или отклонять их, добавлять в друзья других пользователей, бросать им вызовы. Администраторы могут просматривать глобальную статистику и статистику определённого пользователя, просматривать список пользователей и добавлять новые вызовы.

Заключение

В результате работы над проектом было разработано веб-приложение «Генератор спортивных вызовов», которое соответствует заявленным целям и требованиям:

1. Три ключевые роли: гость, зарегистрированный пользователь и администратор.

2. Веб-приложение использует клиент-серверную архитектуру. Серверная часть реализована на ASP.NET Core, клиентская – на React.js. Подключены сторонние сервисы: для управления пользователями используется JWT [5] для аутентификации и авторизации.

3. Веб-приложение включает 24 ключевых функций, охватывающих весь необходимый функционал: создание, обновление и удаление вызовов, управление дружескими запросами, просмотр конкретной и глобальной статистики.

4. Для хранения данных была создана реляционная база данных на PostgreSQL, общее количество таблиц - 7.

5. Общий объем программного кода веб-приложения составил 8000 строк авторского кода.

6. Общее количество тестов – 11, покрытие кода тестами составило 80%.

7. Количество маршрутов, реализованных в приложении – 20.

На основе полученных результатов работы веб-приложения можно сделать вывод, что цель проекта достигнута, а все требования технического задания были полностью выполнены.

Список используемых источников

- 1 ASP.NET [Электронный ресурс]. – Режим доступа: <https://dotnet.microsoft.com/apps/aspnet> – Дата доступа: 03.01.2025.
- 2 React [Электронный ресурс]. – Режим доступа: <https://react.dev/> – Дата доступа: 03.01.2025.
- 3 PostgreSQL [Электронный ресурс]. – Режим доступа: <https://www.postgresql.org/> – Дата доступа: 03.01.2025.
- 4 Entity Framework Core [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/ef/core/> – Дата доступа: 03.01.2025.
- 5 JWT (Json Web Tokens) [Электронный ресурс]. – Режим доступа: <https://jwt.io/> – Дата доступа: 03.01.2025.
- 6 BCrypt.NET [Электронный ресурс]. – Режим доступа: <https://www.nuget.org/packages/BCrypt.Net-Next/> – Дата доступа: 03.01.2025.
- 7 Swagger (OpenAPI) [Электронный ресурс]. – Режим доступа: <https://swagger.io/> – Дата доступа: 03.01.2025.
- 8 Npgsql (PostgreSQL для .NET) [Электронный ресурс]. – Режим доступа: <https://www.npgsql.org/> – Дата доступа: 03.01.2025.
- 9 React Router [Электронный ресурс]. – Режим доступа: <https://reactrouter.com/> – Дата доступа: 03.01.2025.
- 10 React Toastify [Электронный ресурс]. – Режим доступа: <https://fkhadra.github.io/react-toastify/> – Дата доступа: 03.01.2025.
- 11 Tailwind CSS [Электронный ресурс]. – Режим доступа: <https://tailwindcss.com/> – Дата доступа: 03.01.2025.
- 12 Postman [Электронный ресурс]. – Режим доступа: <https://www.postman.com/> – Дата доступа: 03.01.2025.

Приложение А

Листинг – Скрипт создания базы данных

```
using Generator.Domain;
using Microsoft.EntityFrameworkCore;

namespace Generator.Infrastructure;

public class ApplicationDbContext : DbContext
{
    public
    ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
        public DbSet<Domain.Users> Users { get; set; }
        public DbSet<Domain.Activities> Activities { get; set; }
        public DbSet<Domain.Friendship> Friendships { get; set; }
        public DbSet<Domain.UserData> UserData { get; set; }
        public DbSet<Domain.Calls> Calls { get; set; }
        public DbSet<Challenge> Challenges { get; set; }
        public DbSet<UserCall> UserCalls { get; set; }
        public DbSet<DailyActivity> DailyActivities { get; set; }

        protected override void OnModelCreating(ModelBuilder
modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            modelBuilder.Entity<Activities>()
                .HasKey(a => a.activity_id);
            modelBuilder.Entity<Calls>()
                .HasKey(a => a.call_id);
            modelBuilder.Entity<Friendship>()
                .HasKey(a => a.friend_id);
            modelBuilder.Entity<UserData>()
                .HasKey(a => a.data_id);
            modelBuilder.Entity<Users>()
                .HasKey(a => a.user_id);
            modelBuilder.Entity<DailyActivity>()
                .HasKey(a => a.dailyAcivityId);

            modelBuilder.Entity<UserData>()
                .HasOne(ud => ud.User)
                .WithMany(u => u.UserData)
                .HasForeignKey(ud => ud.user_id)
                .OnDelete(DeleteBehavior.Cascade);

            modelBuilder.Entity<UserData>()
```

```

        .HasOne(ud => ud.Activity)
        .WithMany(a => a.UserData)
        .HasForeignKey(ud => ud.activity_id)
        .OnDelete(DeleteBehavior.Cascade);

modelBuilder.Entity<Friendship>()
    .HasOne(f => f.User1)
    .WithMany(u => u.Friendships1)
    .HasForeignKey(f => f.user1_id)
    .OnDelete(DeleteBehavior.Cascade);

modelBuilder.Entity<Friendship>()
    .HasOne(f => f.User2)
    .WithMany(u => u.Friendships2)
    .HasForeignKey(f => f.user2_id)
    .OnDelete(DeleteBehavior.Cascade);

modelBuilder.Entity<Calls>()
    .HasOne(c => c.Friendship)
    .WithMany(f => f.Calls)
    .HasForeignKey(c => c.friend_id)
    .OnDelete(DeleteBehavior.Cascade);

modelBuilder.Entity<Friendship>()
    .HasIndex(f => new { f.user1_id, f.user2_id })
    .IsUnique();

modelBuilder.Entity<UserData>()
    .Property(ud => ud.date_info)
    .HasColumnType("date")
    .HasDefaultValueSql("CURRENT_DATE");

modelBuilder.Entity<DailyActivity>()
    .HasOne<Users>()
    .WithMany(u => u.DailyActivities)
    .HasForeignKey(d => d.userId)
    .OnDelete(DeleteBehavior.Cascade);
    }
}

```


Приложение Б

Листинг – Middleware для проверки токена

```
using Generator.Application.Services;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Text;

namespace Generator.API.Middleware;

public class JwtMiddleware
{
    private readonly RequestDelegate _next;
    private readonly TokenService _tokenService;

    public JwtMiddleware(RequestDelegate next, TokenService tokenService)
    {
        _next = next;
        _tokenService = tokenService;
    }

    public async Task InvokeAsync(HttpContext context)
    {
        var token =
context.Request.Headers["Authorization"].ToString().Replace("Bearer
", string.Empty);

        if (!string.IsNullOrEmpty(token))
        {
            if (_tokenService.IsTokenBlacklisted(token))
            {
                context.Response.StatusCode = 401;
                await context.Response.WriteAsync("Token is
blacklisted.");
                return;
            }

            try
            {
                var tokenHandler = new JwtSecurityTokenHandler();
                var key =
Encoding.UTF8.GetBytes(_tokenService.GetKey());

                var validationParameters = new
TokenValidationParameters
                {
                    ValidateIssuer = true,
                    ValidateAudience = true,
                    ValidIssuer = _tokenService.GetIssuer(),
                    ValidAudience = _tokenService.GetAudience(),
```

```

        IssuerSigningKey = new SymmetricSecurityKey(key),
        ClockSkew = TimeSpan.Zero
    };

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

    if (validatedToken is JwtSecurityToken jwtToken &&
jwtToken.Header.Alg.Equals(SecurityAlgorithms.HmacSha256,
StringComparison.InvariantCultureIgnoreCase))
    {
        context.User = principal;
    }
    catch (Exception)
    {
        context.Response.StatusCode = 401;
        await context.Response.WriteAsync("Invalid token.");
        return;
    }

    await _next(context);
}
}

```

Приложение В

Листинг – AuthContext и AuthProvider

```
import React, { createContext, useState, useEffect } from
'react';
import jwt_decode from 'jwt-decode';

export const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [authData, setAuthData] = useState(() => {
    const token = localStorage.getItem('token');
    console.log('Токен из localStorage:', token);
    if (token) {
      try {
        const decodedUser = jwt_decode(token);
        const now = Math.floor(Date.now() / 1000);
        console.log('Декодированный токен:', decodedUser);

        if (decodedUser.exp < now) {
          console.warn('Токен истёк.');
```

```
          localStorage.removeItem('token');
          localStorage.removeItem('currentUser');
          localStorage.removeItem('currentUserRole');
          return { token: null, user: null };
        }

        const role =
decodedUser['http://schemas.microsoft.com/ws/2008/06/identity/cl
aims/role'];
        if (!role) {
          console.error('Роль отсутствует в токене.');
```

```
          localStorage.removeItem('token');
          localStorage.removeItem('currentUser');
          localStorage.removeItem('currentUserRole');
          return { token: null, user: null };
        }

        localStorage.setItem('currentUser', decodedUser.sub);
        localStorage.setItem('currentUserRole', role);
        return { token, user: { ...decodedUser, role } };
      } catch (err) {
        console.error('Ошибка декодирования токена:', err);
        localStorage.removeItem('token');
```

```
        localStorage.removeItem('currentUser');
        localStorage.removeItem('currentUserRole');
        return { token: null, user: null };
      }
    }
  })
}
```

```

    return { token: null, user: null };
  });

  useEffect(() => {
    if (authData.token) {
      console.log('Сохранение токена в localStorage.');
```

 `localStorage.setItem('token', authData.token);`

```
    } else {
      console.log('Удаление токена и данных пользователя из
localStorage.');
```

 `localStorage.removeItem('token');`

```
      localStorage.removeItem('currentUser');
```

 `localStorage.removeItem('currentUserRole');`

```
    }
  }, [authData]);

  const logout = () => {
    console.log('Выход пользователя.');
```

 `setAuthData({ token: null, user: null });`

```
    localStorage.removeItem('token');
```

 `localStorage.removeItem('currentUser');`

```
    localStorage.removeItem('currentUserRole');
```

 `};

 return (
 <AuthContext.Provider value={{ ...authData, setAuthData,
logout }}>
 {children}
 </AuthContext.Provider>
);
};`

Приложение Г

Структурная схема веб-приложения

