

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

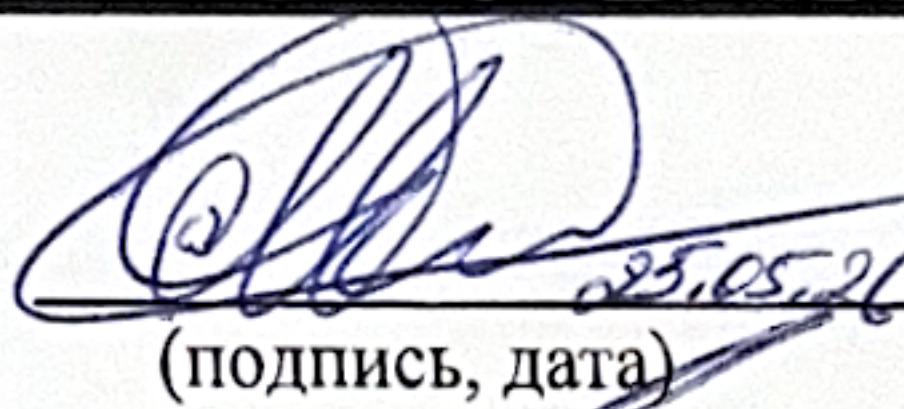
Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

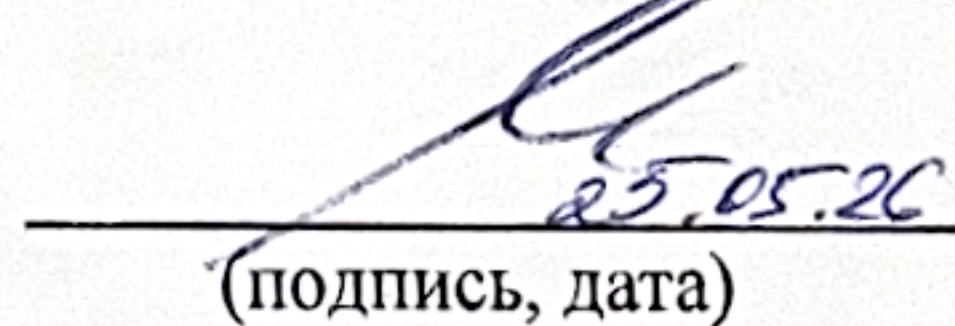
МОБИЛЬНОЕ ПРИЛОЖЕНИЕ
«ПРОФИТНЕС»

Студент


(подпись, дата)

Серобаба Денис Дмитриевич
(ФИО полностью)

Руководитель


(подпись, дата)

Митрохин М.А.
(фамилия, инициалы)

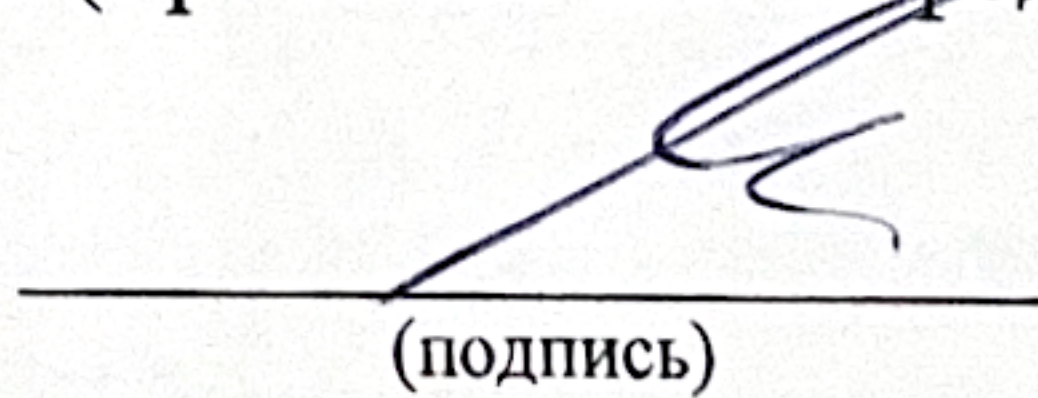
Нормоконтролёр


(подпись, дата)

Бычков А.С.
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.2026г. № 14)

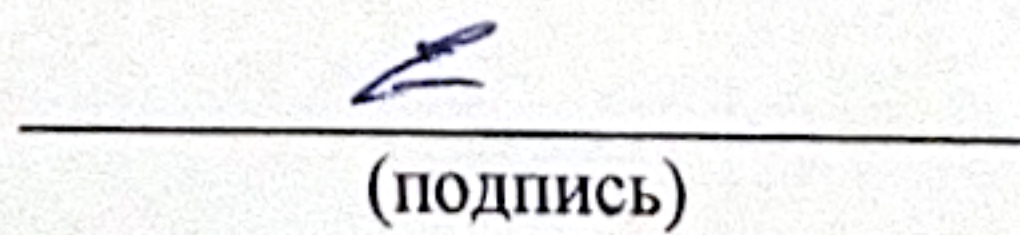
Заведующий кафедрой


(подпись)

Митрохин М.А.
(фамилия, инициалы)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от
02 июня 2026г № 5)

Секретарь ГЭК


(подпись)

Гурин Е.И.
(фамилия, инициалы)

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

«Утверждаю»

Заведующий кафедрой

М.А. Митрохин

« 07 » 04 2026 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Серобаба Денис Дмитриевич
гр. 22ВВП2 факультета ФВТ

2. Тема работы Мобильное приложение
«Профитнес»

Тема утверждена приказом ПГУ № 347/о от " 07 " апреля 2026г.

3. Руководитель бакалаврской работы Митрохин М.А.

4. Задание на работу (назначение разработки, исходные данные и т.п.) Разработать
мобильное приложение «ПроФитнес», позволяющее клиентам фитнес-студии
осуществлять запись на тренировки и контролировать прогресс. Язык программирования
Go для серверной части и JavaScript для клиентского мобильного приложения. В качестве
среды разработки использовался Visual Studio Code. Приложение должно обладать
интуитивно понятным пользовательским интерфейсом и использовать следующие
технологии:

- React Native;

- Gin Web Framework;

- PostgreSQL;

- Redis

- Ассинхронное программирование;

- Rest API;

- Интеграция с CRM-системой Impulse

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметной области: изучить существующие аналоги, изучить способы проектирования и разработки мобильного приложения. Выбрать язык программирования, среду разработки и базу данных.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	45	08.04.2026	
2	Анализ требований	35	13.04.2026	
3	Выбор технологий, применяемых при проектировании и разработке	55	16.04.2026	
4	Создание графического интерфейса	65	24.04.2026	
5	Разработка мобильного приложения	84	05.05.2026	
6	Оформление пояснительной записки	40	15.05.2026	

Дата выдачи задания " 07 " апреля 2026г.

Руководитель бакалаврской работы

(подпись, дата)

Задание к исполнению принял студент

(подпись, дата)

Работу к защите допустить
Декан факультета

(подпись, дата)

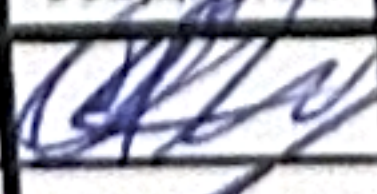
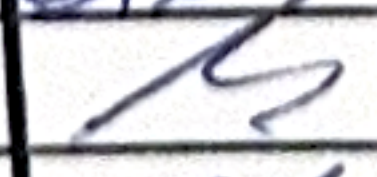
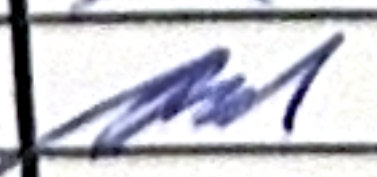
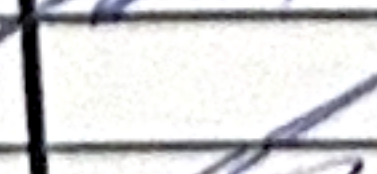
28.05.2026

Реферат

Бакалаврская работа содержит 122 листа, 31 рисунок, 4 таблицы, 7 источников, 3 приложения

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, REACT NATIVE, GO, GIN, POSTGRESQL, REDIS, ГЕЙМИФИКАЦИЯ, ТРЕКИНГ ШАГОВ, REST API, CRM-ИНТЕГРАЦИЯ.

Цель работы – разработать мобильное приложение «ПроФитнес», предоставляющее пользователям запись на занятия, отслеживание прогресса и участие в мотивационных активностях. Мобильное приложение должно позволять пользователям просматривать расписание, записываться на тренировки, просматривать профиль с информацией об абонементе, баллах и замерах тела. Помимо этого должны присутствовать система геймификации, трекинг шагов и push-уведомления. Для синхронизации данных требуется реализовать регистрацию и авторизацию по номеру телефона. В конечном итоге было разработано мобильное приложение на фреймворке React Native с серверной частью на Go, базами PostgreSQL и Redis, которое удовлетворяет всем поставленным задачам. Была реализована авторизация по SMS-коду и интеграция с CRM-системой Impulse через REST API. Приложение работает на ОС Android и iOS, разрабатывалось в Visual Studio Code с использованием Expo.

					ПГУ1.090301.46.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Мобильное приложение «Профитнес» Пояснительная записка	Лит.	Лист	Листов	
Разраб.		Серобаба Д.Д.		25.05.20					
Провер.		Митрохин М.А.		25.05.20			2	66	
Н. Контр		Бычков А.С.		25.05.20		ФВТ, каф. ВТ гр. 22ВВП2			
Утверд.		Митрохин М.А.		25.05.20					

Содержание

Введение	7
1 Приложение «Профитнес»	10
1.1 Анализ актуальности темы исследования	10
1.2 Анализ аналогов	11
1.2.1 Мобильное приложение «ClassPass»	11
1.2.2 Мобильное приложение «Mindbody»	13
1.2.3 Мобильное приложение «MyFitnessPal»	14
1.2.4 Вывод по результатам анализа аналогов.....	16
1.3 Постановка задачи на разработку.....	17
1.3.1 Цель разработки	17
1.3.2 Общие требования к системе.....	18
1.3.3 Функциональные требования	18
1.3.4 Нефункциональные требования	23
1.3.5 Требования к технологическому стеку.....	23
1.3.6 Требования к вычислительным ресурсам	24
1.3.7 Ограничения и допущения.....	24
2 Обоснование выбора технологического стека и настройка инструментов разработки.....	25
2.1 Общие принципы выбора технологий	26
2.2 Обоснование выбора серверного языка программирования Go	26
2.3 Обоснование выбора веб-фреймворка Gin.....	27
2.4 Обоснование выбора баз данных PostgreSQL и Redis.....	27
2.5 Обоснование выбора фреймворка React Native	28
2.6 Выводы по разделу.....	29
3 Архитектура приложения.....	29
3.1 Общая архитектура системы.....	30
3.2 Архитектура серверной части.....	30
3.3 Архитектура клиентской части.....	33

3.4 Взаимодействие компонентов	34
4 Реализация приложения	37
4.1 Описание функционала	37
4.2 Описание работы приложения	38
4.3 Реализация серверной части	39
4.5 Взаимодействие с сервером	43
5 Тестирование	45
5.1 Тестирование авторизации.....	46
5.2 Тестирование расписания и записи на занятия.....	51
5.3 Тестирование профиля и геймификации	55
5.4 Тестирование трекинга шагов.....	59
5.5 Выводы по тестированию.....	60
Заключение	61
Список использованных источников	65
Приложение А - UML-диаграммы	
Приложение Б - Листинг программы	
Приложение В - Презентация	

Перечень сокращений и обозначений

В данной работе применяются следующие сокращения и обозначения

БД – база данных

СМС – служба коротких сообщений (SMS)

API – Application Programming Interface (программный интерфейс приложения)

ACID – Atomicity, Consistency, Isolation, Durability (атомарность, согласованность, изоляция, долговечность)

CLI – Command Line Interface (интерфейс командной строки)

CORS – Cross-Origin Resource Sharing (совместное использование ресурсов между разными источниками)

CRM – Customer Relationship Management (управление взаимоотношениями с клиентами)

CPU – Central Processing Unit (центральное процессорное устройство)

CSS – Cascading Style Sheets (каскадные таблицы стилей)

Go – язык программирования Go (Golang)

HTTP – HyperText Transfer Protocol (протокол передачи гипертекста)

HTTPS – HyperText Transfer Protocol Secure (защищённый протокол передачи гипертекста)

iOS – iPhone Operating System (операционная система для устройств Apple)

JSON – JavaScript Object Notation (текстовый формат обмена данными)

npm – Node Package Manager (менеджер пакетов для Node.js)

ОС – Operating System (операционная система)

REST – Representational State Transfer (архитектурный стиль передачи состояния представления)

SMS– Short Message Service (служба коротких сообщений)

SQL – Structured Query Language (язык структурированных запросов)

TTL – Time To Live (время жизни)

UML – Unified Modeling Language (унифицированный язык моделирования)

URL – Uniform Resource Locator (единообразный локатор ресурса)

					ПГУ1.090301.46.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

YAML – YAML Ain't Markup Language (язык сериализации данных, используется для конфигурационных файлов)

					ПГУ1.090301.46.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

В современном мире фитнес-индустрия переживает период активной цифровой трансформации. Фитнес-центры все чаще сталкиваются с необходимостью предоставления клиентам удобных цифровых сервисов для записи на занятия, отслеживания прогресса и управления абонементом. При этом большинство фитнес-клубов уже используют специализированные CRM-системы для управления расписанием и клиентской базой, однако эти системы, как правило, не предоставляют клиентам качественных мобильных интерфейсов. Клиенты вынуждены записываться на занятия через администраторов по телефону или использовать неадаптированные веб-версии, что создает дополнительную нагрузку на персонал и снижает уровень удовлетворенности обслуживанием.

Существующие на рынке мобильные приложения для фитнеса представляют собой либо универсальные трекеры тренировок, не привязанные к конкретному клубу, либо дорогостоящие кастомизированные решения, стоимость разработки которых недоступна для средних и малых фитнес-центров. При этом потребность в качественном клиентском портале, интегрированном с уже используемой CRM, остается высокой. Кроме того, современные клиенты ожидают от цифровых сервисов не только базовой функциональности записи, но и элементов геймификации, мотивации и персонализации.

Целью выпускной квалификационной работы является разработка мобильного приложения «Профитнес», функционирующего в качестве клиентского портала для фитнес-центра, интегрированного с внешней CRM-системой и предоставляющего пользователям функционал для записи на занятия, отслеживания личного прогресса и участия в мотивационных активностях.

Для достижения поставленной цели необходимо решить следующие задачи.

					ПГУ1.090301.46.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		7

Провести анализ предметной области, включающий изучение существующих аналогов мобильных приложений для фитнес-центров и трекеров привычек, выявление их достоинств и недостатков, а также обоснование актуальности разработки собственного решения.

Сформулировать функциональные и нефункциональные требования к разрабатываемой системе, определить перечень необходимых функций и ограничений.

Обосновать выбор технологического стека для реализации серверной и клиентской частей приложения, включая выбор языка программирования, веб-фреймворка, систем управления базами данных и средств разработки мобильного приложения.

Разработать архитектуру системы, определить взаимодействие между компонентами, спроектировать структуру серверной части с модулями маршрутизации, бизнес-логики и хранения данных, а также структуру клиентской части с экранами, компонентами и сервисами.

Реализовать серверную часть приложения на языке Go с использованием фреймворка Gin, обеспечивающую интеграцию с внешней CRM-системой через публичное API, управление листом ожидания на базе Redis, аутентификацию пользователей и хранение данных в PostgreSQL.

Реализовать мобильное приложение на фреймворке React Native, обеспечивающее удобный интерфейс для просмотра расписания, записи на занятия, управления профилем, отслеживания баллов и достижений, а также интеграцию с датчиком шагов устройства.

Провести тестирование разработанного приложения, включающее проверку сценариев авторизации, просмотра расписания, записи на занятия, работы листа ожидания, управления профилем и системой геймификации.

Разработать проект для коммерциализации приложения, включающий анализ целевой аудитории, определение ключевых процессов, каналов

					ПГУ1.090301.46.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

взаимодействия с клиентами, источников дохода, ключевых ресурсов и оценку рисков.

Объектом исследования в рамках данной работы являются методы и средства интеграции мобильных приложений с внешними CRM-системами, а также подходы к реализации геймификации в фитнес-приложениях.

Предметом исследования является процесс разработки мобильного приложения «Профитнес», включая архитектурные решения, выбор технологического стека, реализацию ключевых функций и тестирование программного продукта.

Практическая значимость работы заключается в создании готового к внедрению программного продукта, который может быть использован фитнес-центрами для повышения качества обслуживания клиентов, снижения нагрузки на административный персонал и увеличения клиентской лояльности за счет элементов геймификации.

Работа состоит из введения, семи разделов, заключения, списка использованных источников и приложений. В первом разделе проводится анализ актуальности темы, обзор существующих аналогов и постановка задачи на разработку. Во втором разделе обосновывается выбор технологического стека и описывается настройка инструментов разработки. В третьем разделе рассматривается архитектура разрабатываемого приложения. В четвертом разделе описывается реализация серверной и клиентской частей. В пятом разделе приводятся результаты тестирования. Шестой раздел посвящен безопасности жизнедеятельности при работе с компьютерной техникой и мобильными устройствами. В седьмом разделе представлен стартап-проект по коммерциализации разработанного приложения.

Результатом выполнения выпускной квалификационной работы является разработанное мобильное приложение «Профитнес», соответствующее всем поставленным требованиям и готовое к внедрению в практическую деятельность фитнес-центров.

					ПГУ1.090301.46.001 ПЗ	Лист
						9
Изм.	Лист	№ докум.	Подпись	Дата		

1 Приложение «Профитнес»

1.1 Анализ актуальности темы исследования

В современном мире фитнес-индустрия переживает период активной цифровой трансформации. Согласно данным международных исследований, более 70 процентов клиентов фитнес-клубов предпочитают использовать мобильные приложения для управления своими тренировками, записями на занятия и отслеживания прогресса. При этом ключевым требованием современного пользователя является не просто наличие цифрового инструмента, а его бесшовная интеграция с существующей инфраструктурой фитнес-центра.

Актуальность разработки мобильного приложения "Профитнес" обусловлена следующими факторами.

Во-первых, большинство фитнес-клубов на сегодняшний день используют специализированные CRM-системы для управления расписанием, клиентской базой и финансовыми операциями. Однако эти системы, как правило, имеют закрытую архитектуру и не предоставляют клиентам удобных мобильных интерфейсов. Клиенты вынуждены записываться на занятия через администраторов по телефону или использовать неадаптированные веб-версии, что создает дополнительную нагрузку на персонал и снижает качество обслуживания.

Во-вторых, существующие на рынке мобильные приложения для фитнеса часто представляют собой либо универсальные трекеры тренировок, не привязанные к конкретному клубу, либо кастомизированные решения, стоимость разработки и внедрения которых недоступна для средних и малых фитнес-центров. При этом потребность в качественном клиентском портале, интегрированном с уже используемой CRM, остается высокой.

В-третьих, современные клиенты фитнес-центров ожидают от цифровых сервисов не только базовой функциональности записи, но и элементов геймификации, мотивации и персонализации. Возможность отслеживать

					ПГУ1.090301.46.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		10

собственную прогрессию, получать баллы за посещения, видеть историю замеров и достижения — становится неотъемлемой частью клиентского опыта, формирующего лояльность и удерживающего пользователя.

В-четвертых, внедрение мобильного приложения-зеркала позволяет фитнес-центру снизить нагрузку на административный персонал, автоматизировать процессы записи и отмены занятий, управления листом ожидания, а также получить дополнительный канал коммуникации с клиентами.

Таким образом, разработка приложения "Профитнес", выступающего в роли удобного мобильного интерфейса к существующей CRM-системе и дополненного мотивационными механиками, является актуальной задачей, имеющей как практическую ценность для фитнес-центров, так и научно-методическую значимость с точки зрения проектирования интеграционных решений.

1.2 Анализ аналогов

На данный момент на рынке представлено множество мобильных приложений, предоставляющих функционал для управления тренировками, записи на занятия и отслеживания прогресса. В данном пункте рассмотрены такие популярные и глобально известные решения как ClassPass, Mindbody и MyFitnessPal. Проведен анализ и выявлены слабые и сильные стороны каждого из приложений.

1.2.1 Мобильное приложение «ClassPass»

Интерфейс приложения представлен на рисунке 1.

ClassPass [3] является одной из крупнейших глобальных платформ для бронирования занятий в фитнес-студиях и оздоровительных центрах. Приложение позволяет пользователям находить и записываться на тренировки в тысячах заведений по всему миру по модели подписки.

Достоинства:

					ПГУ1.090301.46.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		11

- обширная сеть заведений. Приложение объединяет тысячи фитнес-студий, тренажерных залов и оздоровительных центров по всему миру;
- гибкая система подписки. Пользователи могут выбирать различные тарифные планы в зависимости от желаемого количества занятий в месяц;
- удобный поиск и фильтрация. Приложение позволяет фильтровать занятия по типу тренировки, времени, местоположению и уровню сложности;
- отзывы и рейтинги. Пользователи могут оставлять отзывы о студиях и тренерах, что помогает другим клиентам делать осознанный выбор.

Недостатки:

- отсутствует интеграция с внешними CRM-системами фитнес-центров. Приложение работает только в рамках экосистемы ClassPass и не может быть интегрировано с уже используемой фитнес-центром CRM;
- модель работы требует от фитнес-центра подключения к платформе ClassPass как к посреднику, что влечет дополнительные комиссионные отчисления;
- отсутствует геймификация. В приложении нет системы баллов, достижений и мотивационных механик для удержания пользователей;
- ограниченные возможности кастомизации для конкретного фитнес-центра. Приложение выглядит одинаково для всех заведений и не позволяет адаптировать интерфейс под бренд конкретного клуба.

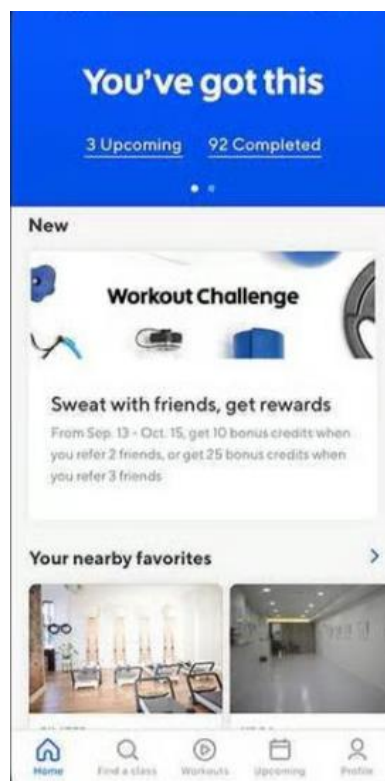


Рисунок 1 – Интерфейс ClassPass

1.2.2 Мобильное приложение «Mindbody»

Интерфейс приложения представлен на рисунке 2.

Mindbody представляет собой комплексную платформу для управления фитнес-бизнесом, включающую как CRM-систему для администраторов, так и мобильное приложение для клиентов. Приложение позволяет пользователям записываться на занятия, оплачивать услуги и отслеживать прогресс.

Достоинства:

- запись на занятия через расписание. Приложение предоставляет удобный календарь для просмотра расписания и бронирования мест на групповые тренировки;
- интеграция с собственной CRM. Приложение работает в связке с CRM-системой Mindbody, обеспечивая синхронизацию данных о клиентах, записях и платежах;
- онлайн-оплата. Пользователи могут оплачивать абонементы и отдельные занятия непосредственно через приложение;

- оповещения. Mindbody отправляет напоминания о предстоящих тренировках и уведомления об изменениях в расписании.

Недостатки:

- приложение интегрируется только с собственной CRM-системой Mindbody, что делает невозможным его использование с уже существующими CRM фитнес-центров;
- отсутствует геймификация. В приложении не реализованы механики баллов, достижений и мотивации пользователей;
- отсутствует трекинг шагов и активности. Приложение не использует датчики мобильного устройства для отслеживания физической активности;
- ежемесячная абонентская плата. Фитнес-центры вынуждены платить регулярную плату за использование платформы.

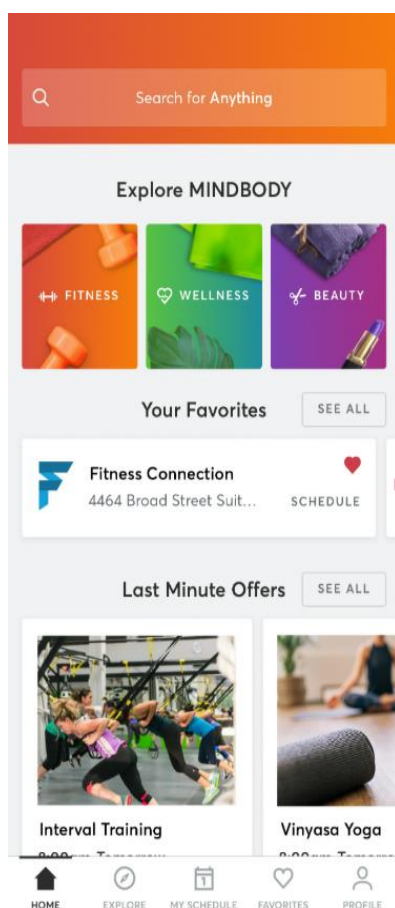


Рисунок 2 – Экран расписания приложения Mindbody

1.2.3 Мобильное приложение «MyFitnessPal»

					ПГУ1.090301.46.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		14

Интерфейс приложения представлен на рисунке 3.

MyFitnessPal является одним из самых популярных универсальных фитнес-трекеров в мире, принадлежащим компании Under Armour. Приложение позволяет пользователям отслеживать питание, физическую активность и прогресс в достижении целей. Однако оно не привязано к конкретным фитнес-клубам и не предоставляет возможности записи на групповые занятия.

Достоинства:

- трекинг активности. Приложение интегрируется с датчиками устройства и позволяет отслеживать количество шагов, пройденную дистанцию и сожженные калории;
- отслеживание питания. MyFitnessPal содержит обширную базу продуктов и позволяет вести дневник питания с подсчетом калорий и макронутриентов;
- статистика и аналитика. Приложение предоставляет детальную аналитику прогресса пользователя в виде графиков и диаграмм;
- интеграция с другими приложениями. MyFitnessPal синхронизируется со многими популярными фитнес-приложениями и носимыми устройствами.

Недостатки:

- отсутствует интеграция с фитнес-центрами. Пользователь не может записаться на занятие или увидеть расписание реально существующей студии;
- отсутствует запись на занятия через расписание. Приложение не предоставляет функционал бронирования мест на групповые тренировки;
- отсутствует лист ожидания. Механизм управления очередью при отсутствии свободных мест не реализован;
- отсутствует привязка к абонементу конкретного клуба. Приложение не может отображать информацию об остатке занятий или дней по абонементу фитнес-центра.

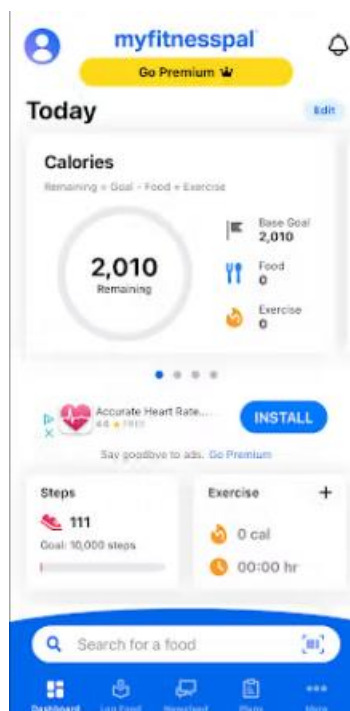


Рисунок 3 – Экран статистики приложения MyFitnessPal

1.2.4 Вывод по результатам анализа аналогов

Проведенный анализ существующих мобильных приложений для фитнеса позволяет сделать следующие выводы, указанные в таблице 1.

Таблица 1 – Сравнительный анализ функциональных возможностей аналогов

Функциональные возможности	ClassPass	Mindbody	MyFitnessPal	Профитнес
Запись на занятия	+	+	-	+
Интеграции с произвольными CRM	-	-	-	+
Трекинг активности	-	-	+	+
Геймификация	-	-	+	+
История замеров тела	+	+	+	+
Русификация	+	+	+	+

Функциональные возможности	ClassPass	Mindbody	MyFitnessPal	Профитнес
Низкий порог входа	-	-	+	+

Проанализировав существующие приложения, можно отметить, что базовый функционал трекера у аналогов имеет существенные ограничения. ClassPass и Mindbody предоставляют возможность записи на занятия, но жестко привязаны к собственным экосистемам и не могут интегрироваться с уже используемыми фитнес-центрами CRM-системами. MyFitnessPal обладает развитыми механиками трекинга и статистики, но полностью отсутствует функционал записи на реальные занятия и управления абонементом.

Ключевые различия между аналогами заключаются в том, что ни одно из рассмотренных решений не сочетает в себе три ключевых свойства: гибкую интеграцию с произвольной внешней CRM-системой через публичное API, продвинутую систему геймификации для мотивации пользователей и доступную стоимость внедрения для средних и малых фитнес-центров.

Таким образом, разработка собственного решения «Профитнес», объединяющего возможность интеграции с любой CRM-системой через публичное API, полноценные механики геймификации (баллы, достижения, серии, трекинг шагов) и низкий порог входа для фитнес-центров, является обоснованной и целесообразной.

1.3 Постановка задачи на разработку

На основе проведенного анализа актуальности и существующих аналогов сформулирована следующая постановка задачи на разработку программного продукта — мобильного приложения "Профитнес".

1.3.1 Цель разработки

Целью выпускной квалификационной работы является разработка мобильного приложения "Профитнес", функционирующего в качестве клиентского портала для фитнес-центра, интегрированного с внешней CRM-системой и предоставляющего пользователям функционал для записи на занятия, отслеживания личного прогресса и участия в мотивационных активностях.

1.3.2 Общие требования к системе

Система должна представлять собой клиент-серверное приложение, состоящее из серверной части на языке Go[1] и мобильного клиента на платформе React Native с поддержкой операционных систем iOS и Android.

Серверная часть должна обеспечивать взаимодействие с внешней CRM-системой через публичное API, управление листом ожидания с использованием технологии Redis, аутентификацию пользователей по номеру телефона и хранение дополнительных пользовательских данных в реляционной базе данных PostgreSQL.

Мобильное приложение должно предоставлять интуитивно понятный интерфейс для просмотра расписания занятий с фильтрацией по филиалам и датам, записи на занятия, автоматического добавления в лист ожидания при отсутствии свободных мест, а также для просмотра профиля пользователя с информацией об абонементе, баллах, замерах тела и истории посещений.

1.3.3 Функциональные требования

Система должна реализовывать следующий набор функций.

В части аутентификации и управления профилем: регистрация и вход пользователя по номеру телефона с подтверждением через одноразовый код; возможность загрузки и смены аватара; возможность редактирования личных данных, включая возраст, рост, весовые параметры и замеры тела; просмотр информации о типе абонемента, количестве оставшихся занятий и дней до окончания.

В части расписания и записи: отображение списка филиалов фитнес-центра; просмотр расписания на выбранную дату с группировкой по дням

недели; фильтрация занятий по выбранному филиалу; отображение информации о занятии: название, время, длительность, тренер, зал, количество свободных мест; запись на занятие при наличии свободных мест.

В части геймификации и мотивации: отображение текущего баланса баллов пользователя; начисление баллов за посещение тренировок и заполнение данных о замерах; система достижений с отслеживанием прогресса выполнения условий; отображение истории начислений и списаний баллов. Диаграмма на рисунке 4 демонстрирует полный набор возможностей системы геймификации.

В части трекинга активности: интеграция с датчиком шагов устройства; отображение количества шагов за текущий день; отслеживание рекордного дня, среднего количества шагов за неделю и месяц; ведение истории шагов.

В части истории: отображение истории посещенных тренировок с указанием даты, типа занятия и тренера; отображение истории замеров тела с возможностью просмотра динамики изменений.



Рисунок 4 – UML диаграмма вариантов использования системы геймификации

Данная диаграмма отражает взаимодействие пользователя мобильного приложения «Профитнес» с подсистемой геймификации и отслеживания активности. Основным действующим лицом диаграммы является пользователь мобильного приложения, который использует различные функции системы для контроля собственной физической активности и мотивации.

Пользователь может просматривать достижения, которые выдаются системой за выполнение определенных условий, например за количество посещенных тренировок, достижение серии тренировок без пропусков или

выполнение целей по шагам. Каждое достижение содержит описание, информацию о награде и текущем прогрессе выполнения.

Также пользователь может просматривать статистику шагов. Для этого приложение взаимодействует со встроенным шагомером мобильного устройства. Система получает данные о количестве шагов за текущий день, а также рассчитывает статистические показатели за неделю и месяц. Полученные данные отображаются в интерфейсе приложения в удобном для пользователя виде.

Диаграмма также показывает возможность получения баллов за активность внутри приложения. Баллы начисляются за выполнение различных действий: посещение тренировок, заполнение информации о замерах тела, выполнение достижений и поддержание ежедневной активности.

Кроме того, пользователь может просматривать историю собственной активности, включая информацию о ранее полученных достижениях, изменениях количества баллов и истории шагов. Отдельной функцией является редактирование замеров тела, позволяющее пользователю сохранять данные о весе, росте и других параметрах организма.

Диаграмма деятельности, изображенная на рисунке 5, описывает последовательность действий, выполняемых в процессе авторизации пользователя в мобильном приложении «Профитнес». Диаграмма отражает логику взаимодействия между пользователем, мобильным приложением и серверной частью системы.

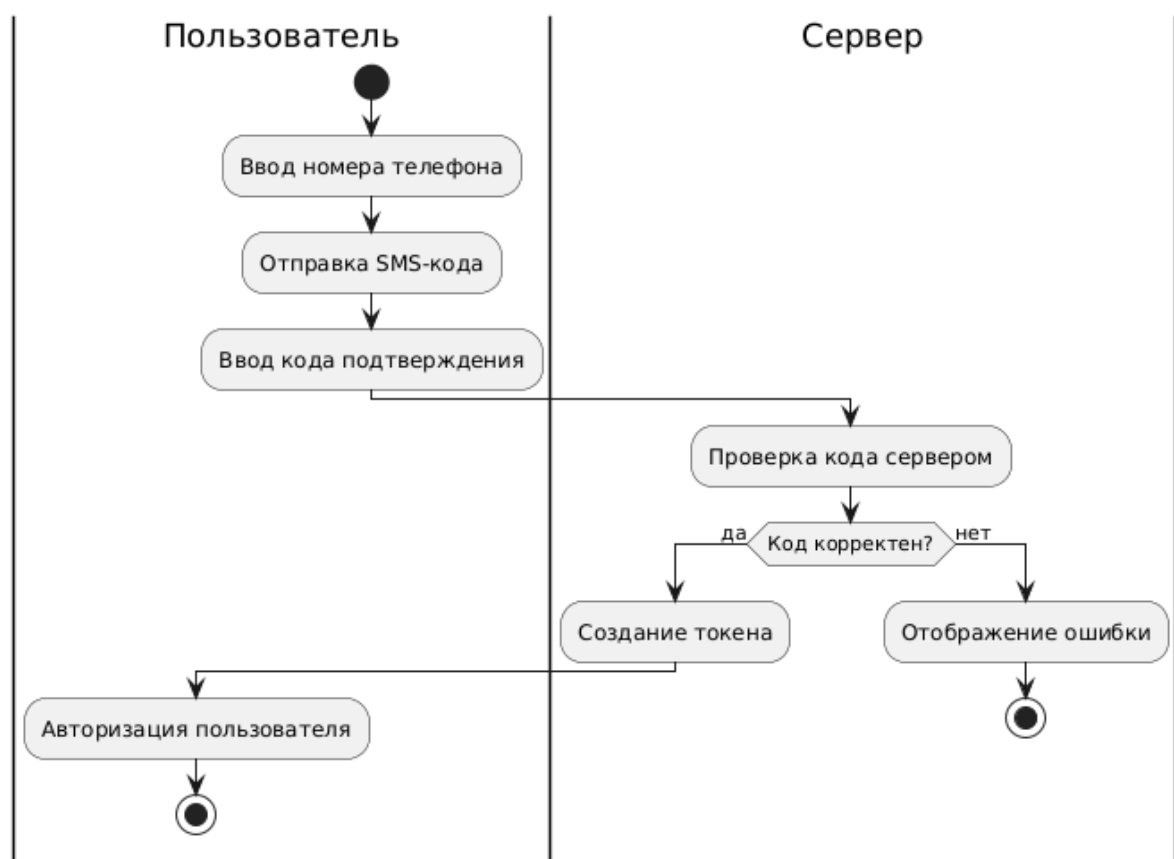


Рисунок 5 – UML диаграмма деятельности процесса авторизации

Процесс начинается с запуска сценария авторизации, после чего пользователь вводит номер телефона в специальное поле мобильного приложения. После ввода номера приложение отправляет запрос на сервер для генерации и отправки SMS-кода подтверждения.

Пользователь получает SMS-сообщение с кодом и вводит его в приложение. Далее введенный код отправляется на сервер для проверки корректности.

На этапе проверки сервер сравнивает введенный код с ранее сгенерированным значением. Если код является корректным, сервер создает токен доступа пользователя. Токен используется для дальнейшей авторизации всех запросов внутри системы. После успешного создания токена пользователь считается авторизованным и получает доступ к функционалу приложения.

Если код введен неверно, сервер возвращает ошибку, а приложение отображает пользователю сообщение о неправильном коде подтверждения. После этого пользователь может повторно ввести код без необходимости повторной отправки номера телефона.

Диаграмма отражает как позитивный сценарий успешной авторизации, так и негативный сценарий обработки ошибок.

1.3.4 Нефункциональные требования

Система должна соответствовать следующим нефункциональным требованиям.

Требования к производительности: время ответа сервера на API-запрос не должно превышать 500 миллисекунд при штатной нагрузке; мобильное приложение должно запускаться не дольше 10 секунд на устройствах среднего класса.

Требования к надежности: система должна сохранять работоспособность при временных сбоях в работе внешней CRM-системы; данные пользователя не должны теряться при некорректном завершении работы приложения.

Требования к безопасности: аутентификация пользователей должна осуществляться по номеру телефона с использованием одноразового кода; все взаимодействия между мобильным приложением и сервером должны осуществляться по протоколу HTTPS; ключи доступа к API CRM должны храниться на серверной стороне и не передаваться на клиент.

Требования к удобству использования: интерфейс приложения должен быть адаптивным и корректно отображаться на устройствах с разными размерами экранов; все ключевые действия (запись, отмена) должны сопровождаться понятными подтверждениями и сообщениями об ошибках; навигация между экранами должна быть интуитивно понятной и соответствовать стандартам платформ iOS и Android.

1.3.5 Требования к технологическому стеку

					ПГУ1.090301.46.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		23

Разработка серверной части должна быть выполнена на языке программирования Go с использованием веб-фреймворка Gin. В качестве системы управления базами данных для хранения пользовательских данных должна использоваться PostgreSQL. Для реализации механизма листа ожидания и управления временными данными должен использоваться Redis. Коммуникация с внешней CRM-системой должна осуществляться через HTTP-запросы к публичному API.

Разработка мобильного приложения должна быть выполнена с использованием фреймворка React Native. Для маршрутизации между экранами должен использоваться React Navigation. Для хранения локальных данных на устройстве должна использоваться библиотека AsyncStorage. Для работы с датчиком шагов должна использоваться библиотека Expo Sensors.

1.3.6 Требования к вычислительным ресурсам

Для работы серверной части необходим процессор с тактовой частотой от 2 ГГц и оперативная память от 2 ГБ. На сервере требуется установленная операционная система семейства Linux или Windows. Для хранения данных пользователей и кеширования необходимо от 10 ГБ свободного дискового пространства.

Мобильное приложение работает на устройствах под управлением iOS 13 и выше, а также Android 8 и выше. На смартфоне требуется наличие датчика шагов (педагометра) для работы функции трекинга активности. Рекомендуемый объем оперативной памяти на мобильном устройстве — от 2 ГБ.

Для корректной работы приложения требуется активное интернет-соединение. На устройстве должно быть установлено приложение Expo Go (на этапе разработки) или собранная production-версия приложения

1.3.7 Ограничения и допущения

Разрабатываемая система функционирует в предположении, что внешняя CRM-система предоставляет стабильное публичное API со следующими методами: получение списка филиалов, получение расписания занятий за период,

					ПГУ1.090301.46.001 ПЗ	Лист
						24
Изм.	Лист	№ докум.	Подпись	Дата		

получение списка бронирований, создание бронирования, отмена бронирования, получение информации о клиенте, отправка SMS-кода подтверждения.

Также предполагается, что администратор фитнес-центра имеет возможность настроить необходимые параметры подключения к CRM (базовый URL и API-ключ) в конфигурационном файле серверной части.

Разрабатываемое приложение не решает задач управления персоналом, финансового учета или администрирования — эти функции остаются в зоне ответственности CRM-системы.

					ПГУ1.090301.46.001 ПЗ	Лист
						25
Изм.	Лист	№ докум.	Подпись	Дата		

2 Обоснование выбора технологического стека и настройка инструментов разработки

2.1 Общие принципы выбора технологий

При выборе технологического стека для разработки приложения "Профитнес" учитывались следующие критерии: соответствие архитектурным требованиям проекта, производительность и масштабируемость, доступность и качество документации, размер сообщества разработчиков, стоимость лицензирования (предпочтение отдавалось открытому и бесплатному программному обеспечению), а также возможность кроссплатформенной разработки для мобильной части.

На основе анализа указанных критериев был сформирован следующий технологический стек: серверная часть на языке Go с использованием фреймворка Gin; база данных PostgreSQL для хранения пользовательских данных; Redis для управления листом ожидания и временными данными; мобильное приложение на фреймворке React Native.

2.2 Обоснование выбора серверного языка программирования Go

Язык программирования Go был выбран по следующим причинам.

Высокая производительность. Go компилируется в машинный код и по скорости выполнения сопоставим с C++ и Java, что критически важно для серверной части, обрабатывающей множество одновременных запросов от мобильных клиентов и выполняющей интеграцию с внешней CRM-системой.

Простота конкурентного программирования. Встроенная поддержка горутин (goroutines) и каналов (channels) позволяет эффективно обрабатывать тысячи одновременных соединений без значительного потребления оперативной памяти, что особенно важно для системы, работающей в режиме реального времени с листом ожидания. [2]

Минимализм и читаемость кода. Go не перегружен синтаксическими конструкциями, что упрощает поддержку и развитие проекта. Стандартная библиотека Go предоставляет все необходимые инструменты для создания HTTP-сервера, работы с JSON и взаимодействия с базами данных.

Альтернативные варианты — Node.js и Python — были отклонены. Node.js, несмотря на высокую скорость выполнения асинхронных операций, менее эффективен для CPU-интенсивных задач и требует дополнительных усилий для обеспечения типобезопасности. Python, хотя и обладает богатой экосистемой, значительно уступает Go в производительности и эффективности использования памяти.

2.3 Обоснование выбора веб-фреймворка Gin

Среди доступных веб-фреймворков для Go рассматривались стандартная библиотека net/http, фреймворки Echo, Fiber и Gin. Стандартная библиотека хороша для простых проектов, но требует ручной реализации маршрутизации, обработки middleware и валидации запросов. Echo и Fiber также являются высокопроизводительными решениями, однако Gin был выбран благодаря оптимальному сочетанию производительности, удобства использования и распространенности в сообществе.

Gin предоставляет следующие преимущества. Во-первых, высокая производительность маршрутизации благодаря реализации на основе радикс-дерева. Во-вторых, встроенная поддержка middleware, включая логирование, восстановление после паники и CORS. В-третьих, удобная валидация входящих JSON-запросов с использованием тегов структур. В-четвертых, минимальный порог входа для разработчиков, знакомых с другими веб-фреймворками.

2.4 Обоснование выбора баз данных PostgreSQL и Redis

					ПГУ1.090301.46.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		27

В проекте используются две системы управления базами данных: PostgreSQL для постоянного хранения и Redis для кеширования и управления временными данными.

PostgreSQL выбрана по следующим причинам. Это реляционная база данных с открытым исходным кодом, обладающая высокой надежностью и поддержкой ACID-транзакций. Для проекта важны гарантии целостности данных пользователей — информации о профилях, замерах, истории операций. PostgreSQL поддерживает сложные типы данных и мощный язык запросов, что позволяет гибко извлекать статистическую информацию. Также PostgreSQL имеет обширную документацию и активное сообщество.

Redis выбран для реализации механизма листа ожидания. Основная задача — хранить временные записи о пользователях, ожидающих освобождения места на занятии, с возможностью автоматического истечения срока подтверждения (TTL). Redis идеально подходит для этой задачи благодаря встроенному механизму Expire, который автоматически удаляет ключи по истечении заданного времени. Кроме того, Redis обеспечивает микросекундные задержки доступа к данным и поддерживает структуры данных типа Set, удобные для управления очередями.

2.5 Обоснование выбора фреймворка React Native

Для разработки мобильного приложения был выбран фреймворк React Native. Альтернативами рассматривались нативная разработка под iOS (Swift) и Android (Kotlin), а также кроссплатформенный фреймворк Flutter.

Нативная разработка была отклонена в связи с необходимостью поддержки двух платформ, что привело бы к значительному увеличению трудозатрат — разработке двух независимых приложений или найму двух специализированных разработчиков. Это нецелесообразно для дипломного проекта с ограниченными ресурсами.

Flutter также является кроссплатформенным решением и обладает высокой производительностью, однако требует изучения языка Dart и новой экосистемы. Разработчик проекта уже имел опыт работы с JavaScript и React, что сделало React Native более предпочтительным выбором с точки зрения скорости разработки.

React Native предоставляет следующие преимущества. Кроссплатформенность — до 90 процентов кода может быть переиспользовано между iOS и Android. Hot Reload — мгновенное отображение изменений в коде без перекомпиляции всего приложения. Большое сообщество и экосистема — доступны тысячи готовых библиотек для решения типовых задач. Возможность использования нативных модулей при необходимости доступа к специфичным функциям устройства.

2.6 Выводы по разделу

Обоснованный технологический стек полностью соответствует требованиям, предъявляемым к разрабатываемой системе. Выбор Go и Gin обеспечивает высокую производительность серверной части и простоту реализации интеграции с внешним API. Использование PostgreSQL гарантирует надежное хранение пользовательских данных, а Redis — эффективную работу механизма листа ожидания. Применение React Native позволяет создать кроссплатформенное мобильное приложение с минимальными затратами на разработку и поддержку. Все выбранные инструменты являются открытыми и бесплатными, что исключает дополнительные лицензионные расходы.

Настройка инструментов разработки выполнена в полном объеме, среда разработки готова к реализации функциональных требований, описанных в постановке задачи.

3 Архитектура приложения

В данном разделе рассматриваются архитектурные решения, принятые при разработке приложения «Профитнес». Описывается общая архитектура системы как клиент-серверного приложения, а также внутреннее устройство серверной и клиентской частей.

3.1 Общая архитектура системы

Приложение «Профитнес» представляет собой клиент-серверную систему, состоящую из трех основных компонентов: мобильное приложение (клиент), серверное приложение (бэкенд) и внешняя CRM-система фитнес-центра.

Мобильное приложение, разработанное на фреймворке React Native, обеспечивает пользовательский интерфейс и взаимодействие с конечным пользователем. Серверное приложение, написанное на языке Go, выступает в роли промежуточного слоя (прокси) между мобильным приложением и внешней CRM-системой. Такое архитектурное решение позволяет абстрагировать мобильное приложение от особенностей реализации конкретной CRM, обеспечивает единую точку входа для всех клиентских запросов и предоставляет возможность расширения функционала без модификации CRM.

Внешняя CRM-система (Impulse CRM) является источником данных о расписании занятий, тренерах, филиалах, бронированиях и клиентах. Серверное приложение взаимодействует с ней через публичное API, выполняя HTTP-запросы с использованием API-ключа для аутентификации.

3.2 Архитектура серверной части

Серверная часть приложения реализована на языке программирования Go[4] с использованием веб-фреймворка Gin. Архитектура сервера организована по модульному принципу и включает следующие основные компоненты.

Слой маршрутизации представлен пакетом `api`, в котором определены все HTTP-обработчики (handlers). В файле `handlers.go` происходит регистрация

					ПГУ1.090301.46.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		30

маршрутов и группировка эндпоинтов по функциональному признаку. Публичные эндпоинты (доступные без аутентификации) объединены в группу ``/api/public`` и включают получение списка филиалов, расписания занятий и списка тренеров. Защищенные эндпоинты объединены в группу ``/api/v1`` и требуют наличия валидного токена доступа, который проверяется middleware-функцией ``ValidateTokenMiddleware``. К защищенным эндпоинтам относятся операции бронирования занятий, получения текущих записей пользователя и отмены записей.

Слой бизнес-логики реализован в пакете ``services``. В нем выделены следующие подсистемы.

Сервис ``crm`` отвечает за взаимодействие с внешней CRM-системой. Он инкапсулирует логику формирования HTTP-запросов, обработку ответов и преобразование данных из формата CRM во внутренние структуры приложения. Клиент CRM инициализируется при запуске сервера как глобальный синглтон и используется всеми обработчиками.

Сервис ``bookings`` реализует логику управления бронированиями занятий. Он принимает запрос от пользователя, проверяет наличие свободных мест через CRM и при успешной проверке создает бронирование. В случае отсутствия свободных мест сервис делегирует запрос сервису листа ожидания.

Слой хранения данных представлен пакетом ``storage``. Для постоянного хранения информации о пользователях, истории начислений баллов, замерах тела и других данных используется реляционная база данных PostgreSQL. Модуль ``postgres`` содержит реализацию функций для работы с таблицами базы данных. Для управления временными данными, в частности для реализации листа ожидания, используется Redis. Модуль ``redis`` обеспечивает подключение и выполнение операций с Redis.

На уровне конфигурации используется файл ``config.yaml``, в котором хранятся настройки подключения к базам данных, CRM и другие параметры. Приоритет переменных окружения над значениями в YAML-файле позволяет

безопасно настраивать систему в различных средах. На рисунке 6 представлена диаграмма компонентов серверной части приложения.



Рисунок 6 – UML диаграмма компонентов серверной части

Диаграмма компонентов демонстрирует архитектурную структуру серверной части приложения «Профитнес» и взаимодействие между основными модулями системы.

Основным внешним компонентом является мобильное приложение, разработанное на React Native. Оно отправляет HTTP-запросы к серверной части приложения.

Запросы поступают в API Layer, реализованный на базе веб-фреймворка Gin. Данный слой отвечает за обработку входящих HTTP-запросов, маршрутизацию и передачу управления соответствующим сервисам бизнес-логики.

Внутри серверной части располагается сервис бронирований. Он реализует основную бизнес-логику приложения, связанную с получением расписания, обработкой данных и взаимодействием с CRM-системой.

Для получения информации о расписании, тренерах, филиалах и бронированиях сервис использует CRM Client. Этот модуль инкапсулирует всю логику взаимодействия с внешней CRM-системой Impulse CRM через HTTP API.

Для хранения постоянных пользовательских данных используется база данных PostgreSQL. В ней хранятся данные профилей пользователей, история баллов, замеры тела и другие сущности системы.

Redis используется для хранения временных данных и ускорения доступа к часто используемой информации. Благодаря высокой скорости работы Redis применяется для кэширования и обработки временных операций.

Внешняя CRM-система выступает основным источником данных о тренировках, филиалах и расписании занятий. Серверная часть приложения выполняет роль промежуточного слоя между мобильным приложением и CRM.

Диаграмма показывает разделение ответственности между компонентами системы и демонстрирует модульную архитектуру серверной части приложения. Также более подробно архитектура серверной части описана с помощью UML-диаграммы классов в приложении А.3

3.3 Архитектура клиентской части

Мобильное приложение разработано с использованием фреймворка React Native [5], который позволяет создавать кроссплатформенные приложения для iOS и Android с использованием JavaScript и React. Архитектура клиентской части организована по функционально-компонентному принципу.

Слой навигации отвечает за перемещение пользователя между экранами приложения. В приложении используется библиотека React Navigation, настроенная на использование стековой навигации (Stack Navigator). Начальным экраном является экран авторизации. После успешного входа пользователю становится доступен основной стек экранов, включающий профиль, расписание, преподавателей и достижения.

Слой экранов (screens) содержит основные страницы приложения и изображен на рисунке 7. Экран 'AuthScreen' отвечает за аутентификацию пользователя по номеру телефона. Экран 'ScheduleScreen' отображает расписание занятий, позволяет выбирать дату и филиал, а также записываться на занятия или добавляться в лист ожидания. Экран 'ProfileScreen' предоставляет пользователю информацию об абонементе, баллах, замерах тела и истории посещений. Экран 'TeachersScreen' отображает список тренеров с возможностью

просмотра подробной информации о каждом. Экран 'RecordsScreen' показывает достижения пользователя, статистику шагов и информацию о сериях посещений.

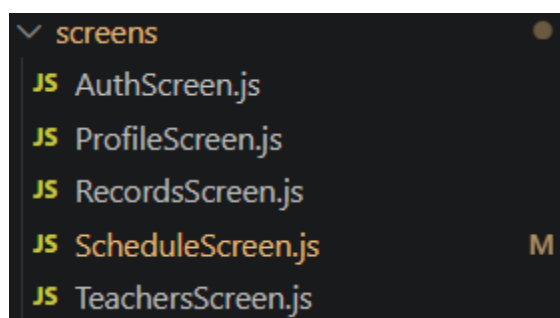


Рисунок 7 – Список экранов

Слой компонентов (components) содержит переиспользуемые элементы интерфейса, такие как счетчик серий, отображение баллов, индикатор загрузки и карточка занятия. Такая организация позволяет избежать дублирования кода и упрощает поддержку приложения.

Слой сервисов (services) отвечает за взаимодействие с серверным API. Модуль 'api.js' содержит настроенный экземпляр Axios для выполнения HTTP-запросов, модуль 'auth.js' управляет хранением и обновлением токенов доступа в AsyncStorage, модуль 'notifications.js' отвечает за регистрацию устройства для получения push-уведомлений.

Слой хранения на устройстве реализован с использованием AsyncStorage, который позволяет сохранять локальные данные, такие как токен доступа и идентификатор клиента. Это обеспечивает сохранение сессии пользователя между запусками приложения.

3.4 Взаимодействие компонентов

Взаимодействие между компонентами системы осуществляется следующим образом. Пользователь запускает мобильное приложение, которое через сервисный слой отправляет HTTP-запрос к серверной части. Сервер, получив запрос, проверяет валидность токена (для защищенных эндпоинтов) и передает управление соответствующему обработчику. Обработчик

взаимодействует с необходимыми сервисами, которые, в свою очередь, обращаются к CRM или базам данных. Полученный ответ преобразуется в JSON и отправляется обратно мобильному приложению.

В случае записи на занятие при свободных местах сервер создает бронирование через CRM и возвращает подтверждение. Если места отсутствуют, сервер добавляет пользователя в лист ожидания в Redis и возвращает соответствующий статус. После освобождения места сервис листа ожидания через фоновый listener отправляет push-уведомление пользователю.

Изолирование серверной части от прямого доступа к CRM позволяет гибко обрабатывать ошибки, кэшировать данные при необходимости и реализовывать дополнительную логику, такую как система баллов и геймификация. Графическую схему можно изучить на рисунке 8.

Таким образом, выбранная архитектура обеспечивает надежное, масштабируемое и поддерживаемое решение, полностью соответствующее функциональным требованиям, предъявленным к приложению «Профитнес».

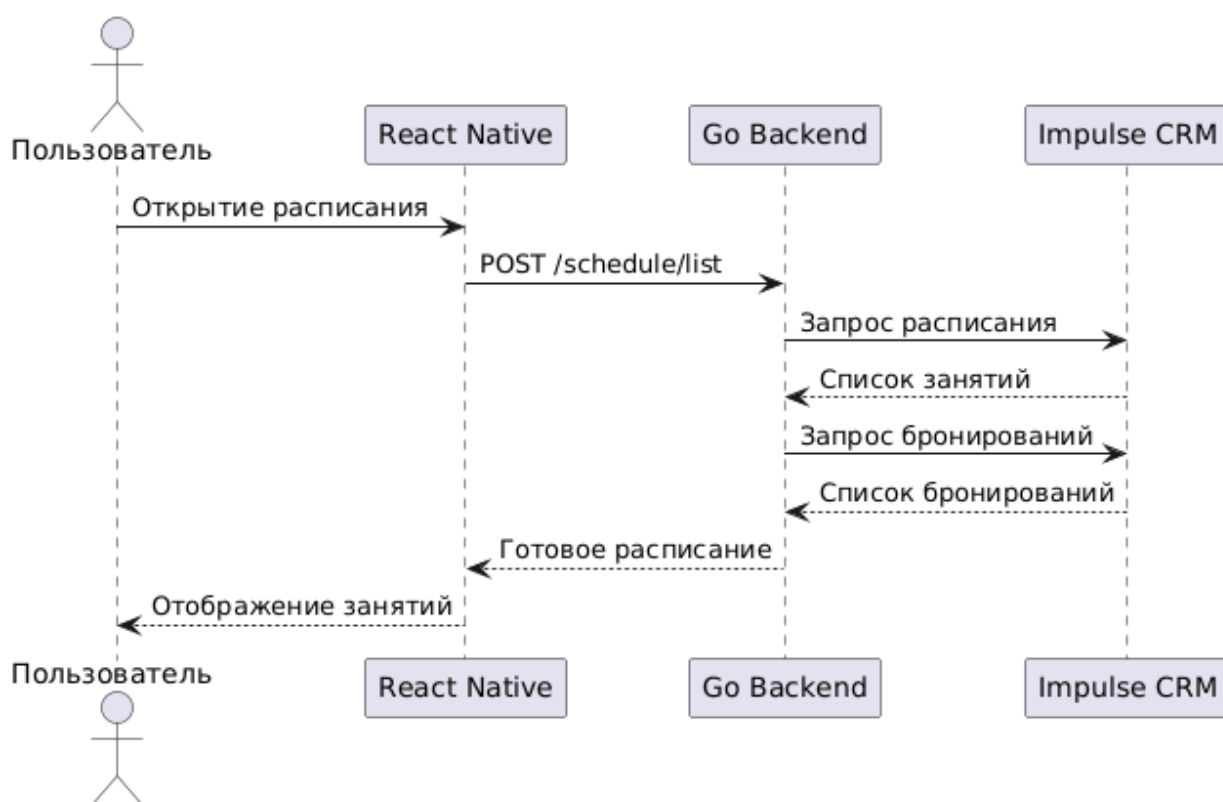


Рисунок 8 – UML диаграмма последовательности загрузки расписания

Диаграмма последовательности описывает процесс загрузки расписания занятий и взаимодействие между всеми компонентами системы при выполнении данного сценария.

Процесс начинается с того, что пользователь открывает экран расписания в мобильном приложении. После открытия экрана клиентское приложение React Native автоматически отправляет HTTP-запрос на сервер по маршруту /schedule/list. В приложении А.1 изображена UML-диаграмма отправки этого HTTP-запроса.

Серверное приложение принимает запрос и инициирует обращение к внешней CRM-системе для получения информации о расписании занятий. CRM возвращает список тренировок за выбранный период времени.

После получения расписания сервер дополнительно выполняет второй запрос к CRM для получения информации о текущих бронированиях пользователей. Это необходимо для вычисления количества свободных мест на каждой тренировке.

Полученные данные обрабатываются сервером: вычисляется количество свободных мест, выполняется сортировка занятий по времени, а также формируется итоговая структура ответа.

После завершения обработки сервер отправляет готовое расписание обратно мобильному приложению. Клиентское приложение получает данные и отображает пользователю список доступных занятий.

Диаграмма демонстрирует последовательность вызовов между клиентом, сервером и CRM-системой, а также показывает распределение ответственности между компонентами.

Подробное описание взаимодействия компонентов графически описано в приложении А.2 с помощью UML-диаграммы развертывания.

4 Реализация приложения

В данном разделе рассматриваются особенности реализации мобильного приложения «Профитнес». Описывается функционал разработанного приложения, сценарии работы пользователя, а также ключевые аспекты программной реализации серверной и клиентской частей.

4.1 Описание функционала

Разработанное приложение «Профитнес» предоставляет пользователям следующие функциональные возможности.

В части аутентификации и управления профилем реализована регистрация и вход в приложение по номеру телефона с подтверждением через SMS-код. После успешной аутентификации сервер возвращает токен доступа, который сохраняется на устройстве и используется для авторизации последующих запросов. Пользователь может просматривать и редактировать информацию о себе, включая возраст, рост, вес и замеры тела. Заполнение данных о замерах поощряется начислением баллов.

В части расписания и записи на занятия пользователь может просматривать список филиалов фитнес-центра, выбирать дату для записи, просматривать расписание занятий на выбранную дату. Для каждого занятия отображается следующая информация: название, время начала, длительность, имя тренера, название зала, максимальное количество мест и количество свободных мест. При наличии свободных мест пользователь может записаться на занятие. При отсутствии свободных мест пользователь может добавиться в лист ожидания.

В части геймификации и мотивации пользователь получает баллы за посещение тренировок и заполнение данных о замерах. В приложении реализована система достижений, отслеживающая прогресс пользователя по различным критериям, таким как количество посещенных тренировок, серии

посещений без пропусков, разнообразие типов занятий. История начислений и списаний баллов доступна для просмотра.

В части трекинга активности приложение интегрируется с датчиком шагов устройства, отображает количество шагов за текущий день, рекордный день, среднее количество шагов за неделю и месяц.

В части истории пользователь может просматривать историю своих посещений с указанием даты, типа занятия и тренера, а также историю замеров тела с возможностью отслеживания динамики изменений.

4.2 Описание работы приложения

Работа приложения «Профитнес» может быть описана через основные сценарии использования.

Сценарий авторизации. Пользователь запускает приложение и попадает на экран авторизации. При отсутствии учетной записи пользователь вводит номер телефона, после чего на указанный номер отправляется SMS с кодом подтверждения. После ввода кода сервер проверяет его корректность и при успешной проверке возвращает токен доступа и идентификатор клиента из CRM. Токен сохраняется в локальном хранилище устройства, и пользователь переходит на главный экран приложения. При повторном запуске приложение проверяет наличие сохраненного токена и при его валидности выполняет автоматический вход.

Сценарий просмотра расписания и записи на занятие. После авторизации пользователь попадает на экран расписания. Приложение загружает список филиалов из CRM через серверную часть и отображает их для выбора. Пользователь выбирает филиал и интересующую дату. Приложение отправляет запрос на сервер для получения расписания на выбранную дату. Сервер, в свою очередь, обращается к CRM, получает список занятий и количество бронирований, вычисляет количество свободных мест и возвращает обработанные данные клиенту. Пользователь видит список занятий с

					ПГУ1.090301.46.001 ПЗ	Лист
						38
Изм.	Лист	№ докум.	Подпись	Дата		

информацией о времени, тренере, зале и наличии свободных мест. При нажатии на кнопку «Записаться» приложение отправляет запрос на бронирование. Если места есть, сервер создает бронирование через CRM и возвращает успешный ответ. Если мест нет, сервер добавляет пользователя в Redis-очередь листа ожидания. Пользователь получает соответствующее уведомление.

Сценарий просмотра профиля и геймификации. Пользователь переходит на экран профиля, где видит информацию о своем абонементе: тип абонемента, количество оставшихся занятий, количество оставшихся дней. Также отображается текущий баланс баллов и количество баллов, которые будут начислены после посещения запланированных тренировок. Пользователь может заполнить или отредактировать данные о замерах тела, за что получает дополнительные баллы. В разделе достижений отображается список доступных и полученных достижений с прогрессом выполнения.

4.3 Реализация серверной части

Серверная часть приложения реализована на языке Go. Точкой входа является файл `main.go`, в котором выполняются следующие основные действия: загрузка конфигурации, инициализация глобального клиента CRM, подключение к базе данных PostgreSQL, подключение к Redis, инициализация сервиса листа ожидания с запуском фонового listener, инициализация сервиса бронирований, настройка маршрутизации и запуск HTTP-сервера на порту 8080.[6]

Листинг 1 демонстрирует фрагмент инициализации сервера.

```
func main() {
    cfg, err := configs.Load()
    if err != nil {
        log.Fatalf("Ошибка загрузки конфигурации: %v", err)
    }
    if err := crm.InitGlobalClient(cfg.CRM.APIKey,
        cfg.CRM.BaseURL); err != nil {
        log.Fatalf("Ошибка инициализации CRM клиента: %v", err)
    }
}
```



```

db, err := postgres.NewPostgres()
if err != nil {
    log.Fatalf("Ошибка подключения к БД: %v", err)
}
rdb, err := redis.NewRedis()
if err != nil {
    log.Fatalf("Ошибка подключения к Redis: %v", err)
}
wl := waitlist.NewService(rdb, waitlist.Config{
    CRMBaseURL: cfg.CRM.BaseURL,
    CRMAPIKey:  cfg.CRM.APIKey,
    OfferTTL:   20 * time.Minute,
})
if err := wl.StartExpirationListener(context.Background());
err != nil {
    log.Fatalf("Ошибка запуска expiration listener: %v",
err)
}
bsvc := bookings.NewService(cfg.CRM.BaseURL, cfg.CRM.APIKey,
wl)
r := gin.Default()
api.RegisterRoutes(r, db, rdb, wl, bsvc)
r.Run(":8080")
}

```

Обработчик получения расписания (файл `schedule.go`) выполняет следующие шаги: парсинг входных параметров (диапазон дат, идентификатор филиала), формирование запроса к CRM для получения расписания, формирование запроса к CRM для получения количества бронирований, вычисление количества свободных мест как разницы между максимальным количеством мест и количеством бронирований, сортировку занятий по дате и времени, возврат результата клиенту.

Листинг 2 демонстрирует фрагмент обработчика расписания.

```

func GetSchedulePublic(c *gin.Context) {

```

					ПГУ1.090301.46.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		40


```

var req scheduleListReq
if err := c.ShouldBindJSON(&req); err != nil || req.DateFrom
== "" {
    c.JSON(http.StatusBadRequest, gin.H{"success": false,
"error": "date_from required"})
    return
}
scheduleBody, err := crmClient.Post("schedule/list",
map[string]interface{}{
    "range": map[string]interface{}{
        "from": fromUnix,
        "to":    toUnix,
    },
    "fields": map[string]interface{}{
        "branch_id": []int{req.BranchID},
    },
})
resBody, err := crmClient.Post("reservation/list",
map[string]interface{}{
    "range":    map[string]interface{}{"from": fromUnix,
"to": toUnix},
    "filters": map[string]interface{}{"branch_id":
[]int{req.BranchID}},
})
for i := range schedules {
    booked := bookedMap[schedules[i].ID]
    free := schedules[i].MaxPlaces - booked
    if free < 0 {
        free = 0
    }
    schedules[i].FreePlaces = free
}
c.JSON(http.StatusOK, gin.H{"success": true, "items":
schedules})

```

					ПГУ1.090301.46.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		41


```
}
```

4.4 Реализация клиентской части

Мобильное приложение реализовано на фреймворке React Native. Точкой входа является файл `App.js`, в котором настраивается навигация между экранами. Используется стековый навигатор, начальным экраном является экран авторизации. После успешной авторизации пользователю становится доступен основной стек экранов.

Листинг 3 демонстрирует настройку навигации.

```
const Stack = createNativeStackNavigator();  
function App() {  
  return (  
    <NavigationContainer>  
      <Stack.Navigator initialRouteName="Auth" screenOptions={{  
headerShown: false }}>  
        <Stack.Screen name="Auth" component={AuthScreen} />  
        <Stack.Screen name="Profile" component={ProfileScreen}  
/>  
        <Stack.Screen name="Teachers" component={TeachersScreen}  
/>  
        <Stack.Screen name="Schedule" component={ScheduleScreen}  
/>  
        <Stack.Screen name="Records" component={RecordsScreen}  
/>  
      </Stack.Navigator>  
    </NavigationContainer>  
  );  
}
```

Экран расписания (`ScheduleScreen.js`) реализует выбор филиала, выбор даты с помощью календаря, отображение списка занятий. При загрузке экрана приложение запрашивает список филиалов через API и сохраняет его в состоянии. Пользователь выбирает филиал, после чего загружается расписание

для выбранного филиала и даты. Каждое занятие отображается в виде карточки, содержащей время, название, тренера, зал и информацию о свободных местах.

Листинг 4 демонстрирует загрузку расписания.

```
const loadScheduleData = async (date) => {
  if (!selectedBranch?.id || !date) return;
  setLoading(true);
  const response = await getSchedule(date, date,
selectedBranch.id);
  let items = response?.data?.items || [];
  const sorted = items.sort((a, b) => {
    const ta = String(a.time).substring(0, 5);
    const tb = String(b.time).substring(0, 5);
    return ta.localeCompare(tb);
  });
  setSchedule(sorted);
  setLoading(false);
};
```

Экран профиля (`ProfileScreen.js`) реализует отображение информации об абонементе, баллах, управление аватаром, редактирование замеров тела, просмотр истории посещений и истории баллов. Также в профиле реализован функционал выхода из аккаунта и заморозки абонемента. Для хранения аватара используется AsyncStorage.

Система достижений в экране `RecordsScreen.js` отслеживает прогресс пользователя по различным критериям. Для отслеживания шагов используется библиотека `expo-sensors`, которая предоставляет доступ к Pedometer API. Приложение запрашивает разрешение на доступ к датчику шагов и при положительном ответе получает количество шагов за текущий день, а также генерирует историю шагов за последние 30 дней.

4.5 Взаимодействие с сервером

Для взаимодействия с сервером в приложении используется библиотека Axios. В файле `api.js` создается настроенный экземпляр Axios с базовым URL сервера. Все запросы к API инкапсулированы в отдельные функции, которые вызываются из компонентов.

Листинг 5 демонстрирует пример API-вызовов.

```
const api = axios.create({
  baseURL: 'http://localhost:8080/api',
});

api.interceptors.request.use(async (config) => {
  const token = await AsyncStorage.getItem('token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

export const getSchedule = (dateFrom, dateTo, branchId) => {
  return api.post('/public/schedule/list', {
    date_from: dateFrom,
    date_to: dateTo,
    branch_id: branchId,
  });
};

export const createBooking = (scheduleId) => {
  return api.post('/v1/bookings', { schedule_id: scheduleId });
};

export const joinWaitlist = (scheduleId, clientId) => {
  return api.post('/v1/waitlist/join', { schedule_id:
scheduleId, client_id: clientId });
};
```

Таким образом, реализация приложения полностью соответствует поставленным требованиям и обеспечивает все заявленные функциональные возможности. Подробнее изучить весь реализованный для пользователей

функционал возможно с помощью диаграммы прецедентов, изображенной в приложении А.4.

5 Тестирование

После завершения разработки приложения «Профитнес»[7] был проведен комплекс мероприятий по тестированию программного продукта. Целью тестирования являлось выявление ошибок и дефектов в реализации, проверка соответствия функциональным требованиям, а также оценка удобства пользовательского интерфейса.

5.1 Тестирование авторизации

В первую очередь был составлен тест-кейс для авторизации. Он изображен в таблице 2.

Таблица 2 – Тест-кейс авторизации пользователя

Действие	№ шага	Описание шага	Ожидаемый результат	Фактический результат
Пользователь вводит номер телефона	1	Открыть приложение, попасть на экран авторизации	Экран авторизации отобразился	Тест пройден
	2	Ввести номер телефона	Номер вводится	Тест пройден
	3	Нажать кнопку «Получить код»	На указанный номер приходит код авторизации	Тест пройден
Пользователь вводит некорректный номер телефона	1	Открыть экран авторизации	Экран авторизации отобразился	Тест пройден
	2	Ввести номер телефона, не соответствующий ожидаемому формату	Приложение отображает сообщение об ошибке	Тест пройден
	3	Проверить доступность кнопки «Получить код»	Кнопка недоступна до ввода корректного номера	Тест пройден
Пользователь вводит неверный код	1	После получения SMS ввести четырёхзначный код в соответствующее поле	Код вводится	Тест пройден

Действие	№ шага	Описание шага	Ожидаемый результат	Фактический результат
	2	нажать кнопку подтверждения	приложение отображает сообщение об ошибке	Тест пройден
	3	проверить возможность повторного ввода	пользователь может повторить попытку без повторной отправки номера	Тест пройден

На данном экране пользователю предлагается ввести номер телефона для получения SMS-кода подтверждения. В рамках текущего этапа разработки проекта код для авторизации является моковым.

Был протестирован сценарий ввода корректного номера телефона. Пользователь вводит номер телефона в международном формате и нажимает кнопку «Получить код». На указанный номер приходит SMS с кодом. После ввода кода выполняется автоматический вход в приложение, и пользователь переходит на главный экран. Токен доступа сохраняется в AsyncStorage. При повторном запуске приложения проверяется наличие сохраненного токена, и при его валидности выполняется автоматический вход без повторного ввода номера.

Был протестирован сценарий ввода некорректного номера телефона. При вводе номера, не соответствующего ожидаемому формату, приложение отображает сообщение об ошибке. Кнопка получения кода становится недоступной до тех пор, пока пользователь не введет корректный номер.

Был протестирован сценарий ввода неверного кода подтверждения. При вводе кода, не совпадающего с отправленным, приложение отображает сообщение об ошибке и предлагает повторить попытку. Количество попыток ввода кода ограничено для предотвращения несанкционированного доступа.

На рисунке 10 представлен экран ввода номера телефона. На рисунке 11 представлен ввод некорректного кода. На рисунке 12 представлено оповещение о неправильном вводе четырехзначного кода. На

рисунке 13 введен правильный код. На рисунке 14 изображен главный экран после успешной авторизации.

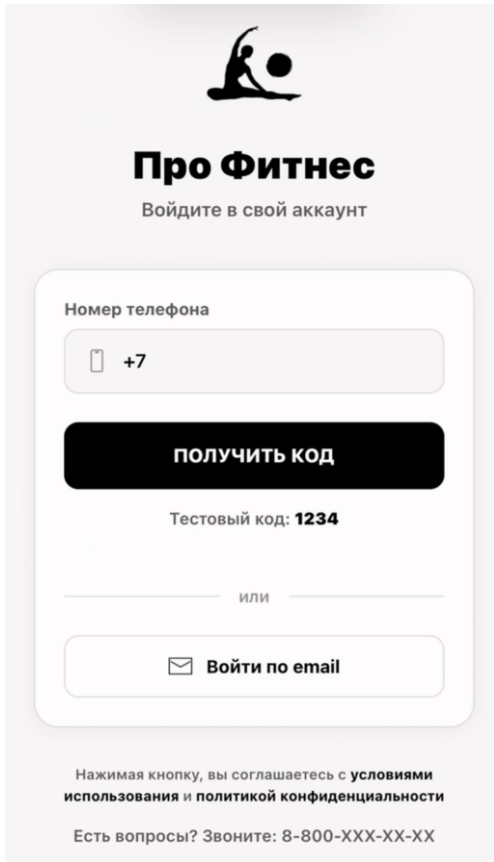


Рисунок 10 – Экран авторизации



Рисунок 11 – ввод неправильного четырехзначного кода

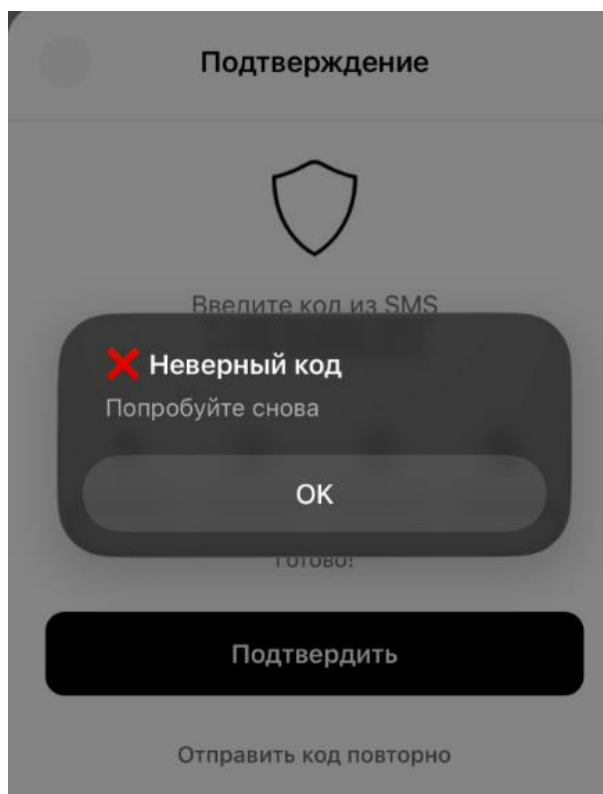


Рисунок 12 – Проверка верности кода

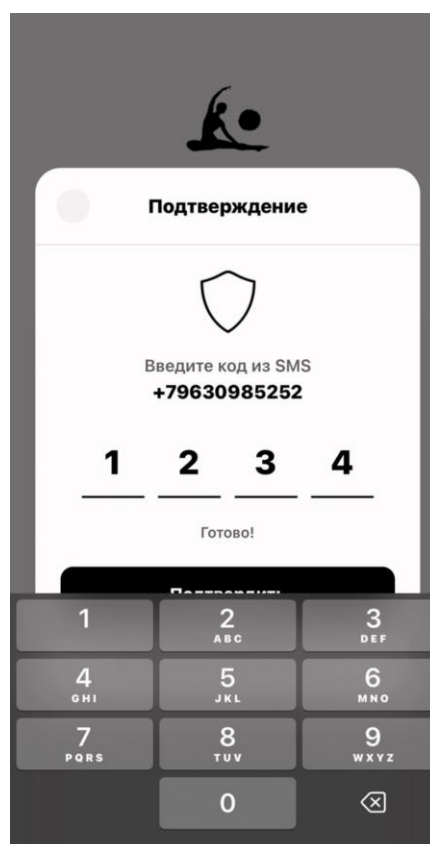


Рисунок 13 – Экран ввода четырехзначного кода



Рисунок 14 – Главный экран после успешной авторизации

Начальным состоянием является состояние неавторизованного пользователя. В этом состоянии пользователь еще не прошел процедуру входа и не имеет доступа к функционалу приложения.

После ввода номера телефона пользователь переходит в состояние ввода кода подтверждения. На данном этапе система ожидает корректный SMS-код для завершения авторизации.

Если код подтверждается сервером, пользователь переходит в авторизованное состояние. Далее создается и сохраняется токен доступа, который используется для поддержания пользовательской сессии между перезапусками приложения.

После сохранения токена пользователь переходит в состояние активной сессии. В этом состоянии ему доступны все функции мобильного приложения.

При выполнении выхода из аккаунта токен удаляется из локального хранилища устройства, а пользователь возвращается в неавторизованное состояние.

Диаграмма позволяет наглядно представить жизненный цикл пользовательской сессии и используется при тестировании механизмов авторизации и хранения токенов.

5.2 Тестирование расписания и записи на занятия

В первую очередь был составлен тест-кейс для отображения расписания и записи на занятия. Он изображен в таблице 3.

Таблица 3 – тест-кейс отображения расписания и записи на занятие

Действие	№ шага	Описание шага	Ожидаемый результат	Фактический результат
Пользователь загружает список филиалов	1	Перейти на экран расписания	Приложение отправляет запрос к серверу	Тест пройден
	2	Дождаться ответа от CRM	Получен список филиалов с корректными названиями	Тест пройден
	3	Проверить отображение списка	Филиалы отображаются в виде выпадающего списка	Тест пройден
Пользователь выбирает филиал	1	Нажать на выпадающий список филиалов	Открылся список доступных филиалов	Тест пройден
	2	Выбрать один из филиалов	Приложение автоматически перезагружает расписание	Тест пройден
	3	Проверить обновление данных	Расписание обновилось для выбранного филиала и текущей даты	Тест пройден
Пользователь выбирает дату	1	На экране расписания открыть календарь	Отображается неделя с возможностью переключения	Тест пройден

Действие	№ шага	Описание шага	Ожидаемый результат	Фактический результат
	2	Нажать на стрелку «вперед» или «назад»	Неделя переключилась на предыдущую или следующую	Тест пройден
	3	Выбрать любую дату в пределах доступного диапазона	Расписание перезагрузилось для выбранной даты	Тест пройден
Пользователь просматривает карточку занятия	1	На экране расписания выбрать фильм и дату	Отобразился список занятий	Тест пройден
	2	Проверить информацию в карточке занятия	Отображаются: время, название, тренер, зал, свободные места	Тест пройден
	3	Проверить сортировку занятий	Занятия отсортированы по времени начала по порядку	Тест пройден
Пользователь записывается на занятие	1	В карточке занятия нажать кнопку «Записаться»	Приложение отправляет запрос на бронирование	Тест пройден
	2	Дождаться ответа сервера	Сервер создает бронирование через CRM	Тест пройден
	3	Проверить получение подтверждения	Пользователь получает сообщение об успешной записи	Тест пройден

Был протестирован сценарий загрузки списка филиалов, что изображено на рисунке 15. При входе на экран расписания приложение отправляет запрос к серверу, который, в свою очередь, обращается к CRM. Полученный список филиалов отображается в виде выпадающего списка. Тестирование подтвердило, что отображаются все филиалы, доступные в CRM, с корректными названиями и адресами.

```

[CRM] Найдено 6 элементов в items
[CRM DEBUG] Элемент 0: map[id:1 name:Хабаровск phone:<nil>]
[CRM DEBUG] Элемент 1: map[id:11 name:Застава Пенза phone:+7 (993) 377 1393]
[CRM DEBUG] Элемент 2: map[id:15 name:Симферополь phone:<nil>]
[CRM DEBUG] Элемент 3: map[id:17 name:Спутник Пенза phone:+7 (995) 491 5985]
[CRM DEBUG] Элемент 4: map[id:18 name:Новые локации phone:<nil>]
[CRM DEBUG] Элемент 5: map[id:19 name:Массаж спутник phone:+7 (995) 491 5985]
[CRM] Успешно сконвертировано 6 филиалов
[API] Успешно получено 6 филиалов
[GIN] 2026/05/13 - 12:24:24 | 200 | 224.4916ms | 192.168.0.101 | POST | "/api/public/branch/list"

```

Рисунок 15 – Логирование загрузки филиалов из CRM

Был протестирован сценарий выбора филиала (рисунок 16). При смене выбранного филиала приложение автоматически перезагружает расписание для нового филиала и текущей даты. Тестирование подтвердило корректное обновление данных.

Рисунок 16 – Меню выбора филиала

Был протестирован сценарий выбора даты. Пользователь может выбирать дату из календаря, отображающего неделю с возможностью переключения на предыдущую или следующую неделю. При выборе даты приложение перезагружает расписание. Тестирование подтвердило корректное отображение расписания для любой даты в пределах доступного диапазона. Эти действия изображены на рисунках 17, 18, 19.

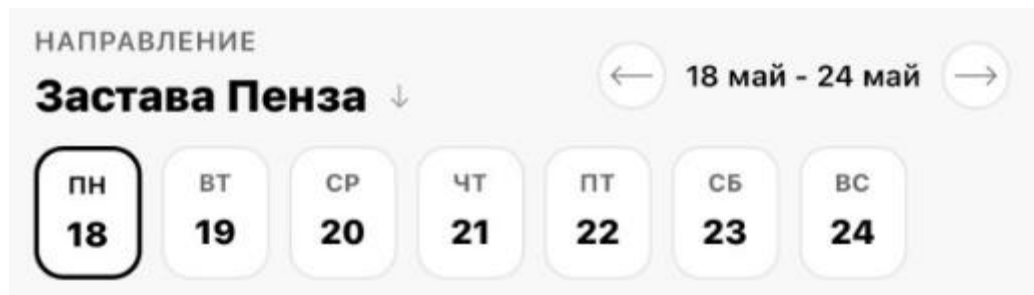


Рисунок 17 – Первоначальный филиал и дата

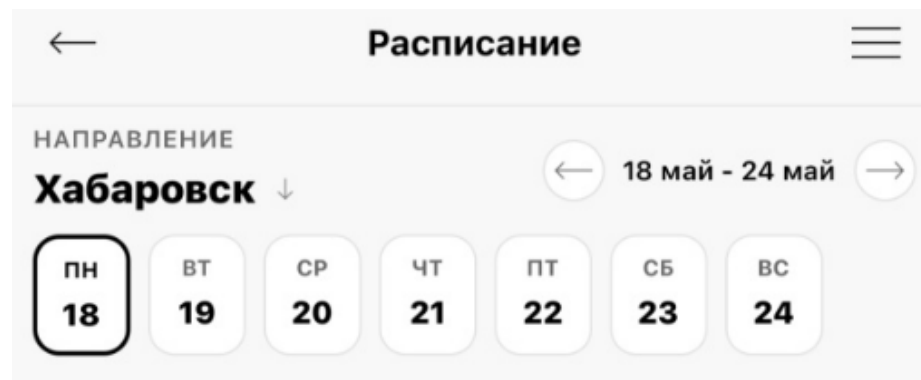


Рисунок 18 – смена филиала

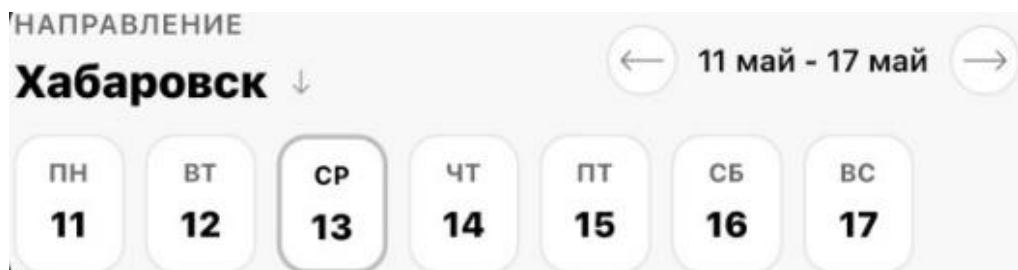


Рисунок 19 – Смена дат

Был протестирован сценарий просмотра расписания на день. На рисунке 20 и 21 можно заметить, что для каждого занятия корректно отображаются время начала, длительность, название занятия, имя тренера, название зала, максимальное количество мест и количество свободных мест. Занятия отсортированы по времени начала в хронологическом порядке.

[GIN] 2026/05/13 - 12:24:27 | 200 | 315.5673ms | 192.168.0.101 | POST | "/api/public/schedule/list"

Рисунок 20 – Логирующее получение данных о расписании из CRM



Рисунок 21 – Отображение расписания в приложении

5.3 Тестирование профиля и геймификации

Был протестирован сценарий загрузки информации об абонементе. На экране профиля корректно отображаются тип абонемента, количество оставшихся занятий, количество оставшихся дней, а также информация о доступных днях заморозки. Это можно увидеть на рисунках 22 и 23.



Рисунок 22 – Информация об абонементе

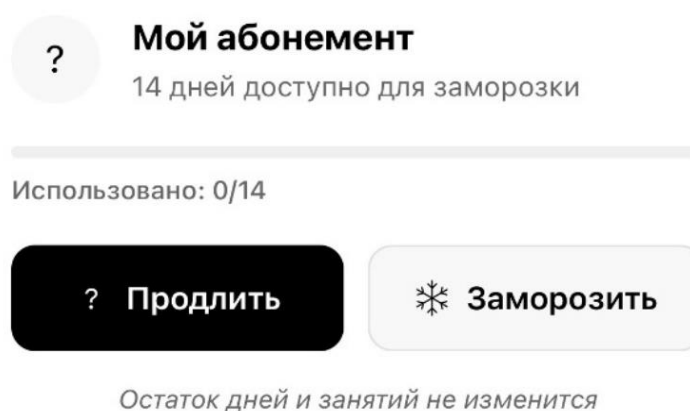


Рисунок 23 – Информация о заморозке абонемента

Был протестирован сценарий отображения текущего баланса баллов. На экране профиля отображается актуальное количество баллов, накопленных пользователем. На рисунке 24 это можно увидеть.



Рисунок 24 – Информация о количестве баллов

Был протестирован сценарий отображения достижений, результаты изображены на рисунках 25 и 26. На экране достижений отображается список

					ПГУ1.090301.46.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		56

всех доступных достижений с указанием названия, описания, награды в баллах и прогресса выполнения. Выполненные достижения отмечаются визуально и содержат дату получения.

Начало положено

Получено 2024-01-10

+50

Не откладывая

Получено 2024-01-10

+100

Неделя под контролем

Получено 2024-01-12

+75

Неделя началась правильно

Получено 2024-01-11

+80

Первая тренировка

Получено 2024-01-11

+100

Рисунок 24 – Общий список достижений с количеством баллов за их выполнение

Неделя под контролем

Планирование 2+ тренировок на неделю вперёд

Награда

+75 баллов

Получено

2024-01-12

Закреть

Рисунок 25 – Подробная информация о конкретном достижении

Был протестирован сценарий выхода из аккаунта. При нажатии на соответствующий пункт меню пользователю показывается диалог подтверждения. Пользователь переходит на экран авторизации. Повторный вход требует повторного ввода номера телефона и кода подтверждения. На рисунках 26 и 27 изображен процесс выхода.

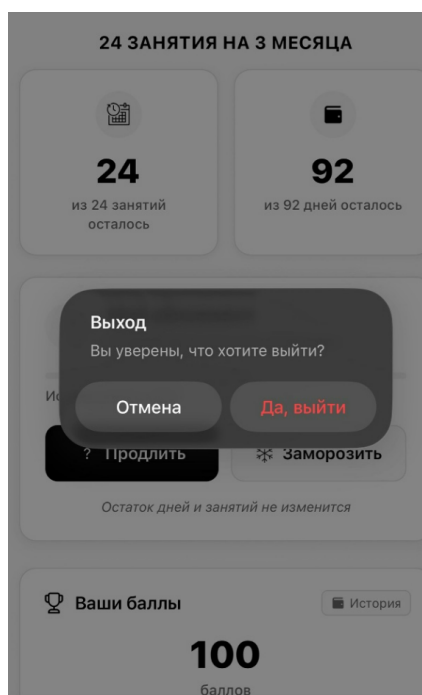


Рисунок 26 – Выход из аккаунта

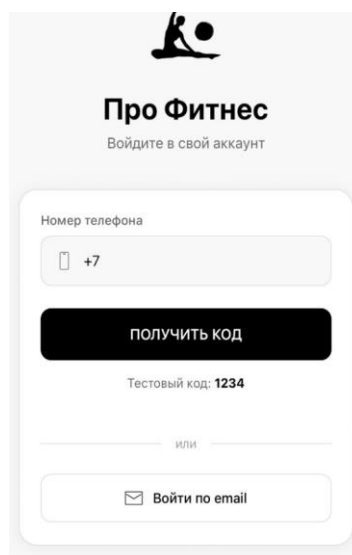


Рисунок 27 – Возврат на экран авторизации

					ПГУ1.090301.46.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		58

5.4 Тестирование трекинга шагов

Был протестирован сценарий получения количества шагов за день. При наличии необходимых разрешений приложение получает от системы количество шагов, сделанных пользователем с начала текущего дня. Значение отображается на экране достижений вместе с целевым значением (по умолчанию 10000 шагов). На рисунке 28 и 29 изображено сравнение двух систем измерения количества шагов.



Рисунок 28 – Шагомер внутри приложения «Профитнес»

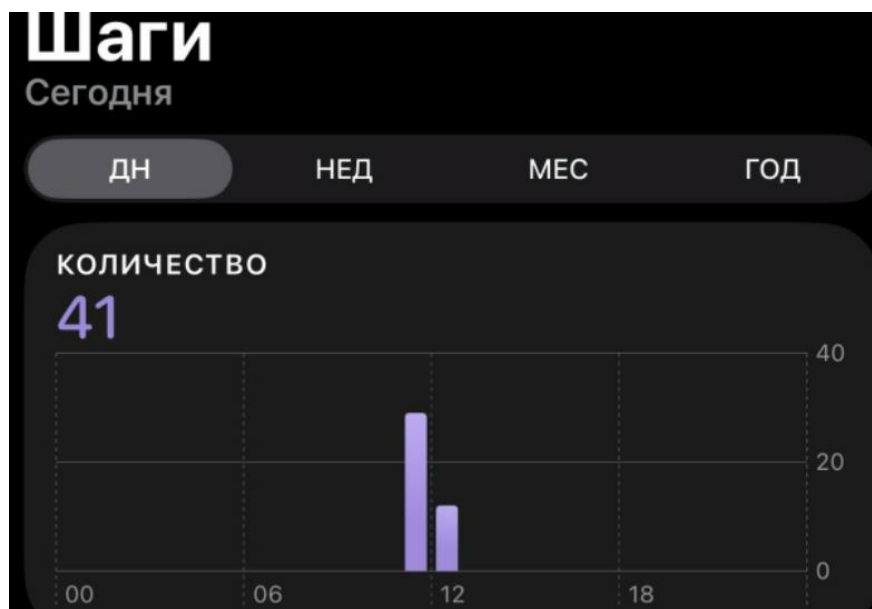


Рисунок 29 – Шагомер внутри IOS, приложение «Фитнес»

Был протестирован сценарий отображения среднего за неделю и за месяц. Приложение рассчитывает среднее арифметическое количество шагов за

последние 7 дней и за последние 30 дней. Рассчитанные значения отображаются на экране достижений.



Рисунок 30 – Отображение среднего количества шагов за период и рекордного количества шагов за 30 дней

Был протестирован сценарий просмотра истории шагов. Пользователь может открыть модальное окно, в котором отображается список количества шагов по дням за последние 14 дней. Для каждого дня указывается дата и количество шагов (рисунок 31).

Чт, 30 апр.	9 940 шагов
Пт, 1 мая	7 710 шагов
Сб, 2 мая	7 336 шагов
Вс, 3 мая	9 540 шагов
Пн, 4 мая	5 314 шагов
Вт, 5 мая	9 506 шагов
Ср, 6 мая	6 097 шагов
Чт, 7 мая	7 152 шагов
Пт, 8 мая	6 543 шагов
Сб, 9 мая	8 687 шагов

Рисунок 31 – История шагов за каждый из последних 14 дней

5.5 Выводы по тестированию

В процессе создания приложения было выявлено достаточное количество ошибок на этапах разработки и интеграции. К числу наиболее значимых проблем

относились некорректная обработка ответов CRM-системы при отсутствии данных, ошибки при вычислении свободных мест из-за несоответствия форматов данных, проблемы с TTL-механизмом в Redis.

Протестировав программное обеспечение, удалось исправить и обработать большую часть выявленных ошибок. Были добавлены дополнительные проверки входящих данных, реализована более надежная обработка ошибок при взаимодействии с CRM, оптимизирован механизм листа ожидания и улучшена обработка состояний загрузки на клиентской стороне.

Проведенное тестирование позволило сделать приложение более надежным и отказоустойчивым. На момент завершения тестирования все критические и основные функциональные требования выполняются корректно. Приложение готово к развертыванию в тестовой эксплуатации.

					ПГУ1.090301.46.001 ПЗ	Лист
						61
Изм.	Лист	№ докум.	Подпись	Дата		

Безопасность жизнедеятельности

Разработанная СА является программным решением, взаимодействие с которым происходит с помощью компьютера или мобильного телефона. Пользователь работает с приложением через веб-интерфейс.

При работе с такой системой отсутствуют многие опасные факторы, такие как: работа с механическим оборудованием, вредными веществами, физически тяжелый труд, повышенная или пониженная температура, высокий уровень шума и тд. Но при этом работа за компьютером, а в частности с данной системой, несет за собой ряд других вредных факторов:

- нагрузка на мышцы шеи, верхнего плечевого пояса и спины;
- зрительная нагрузка;
- усталость кистей рук;
- чрезмерное или недостаточное освещение;
- неправильная организация рабочего места;
- риск, связанный с использованием электрического оборудования.

В таблице 4 приведены основные опасные факторы и меры предосторожности для их минимизации.

Таблица 4 – Опасные факторы при работе с системой

Фактор	Возможное последствие	Меры снижения риска
Длительная работа за экраном	Утомление глаз, снижение концентрации	Перерывы, правильная яркость экрана, достаточное освещение
Неправильная посадка	Боль в спине, шее, плечах	Регулировка положения на стуле, высоты стола и расположения монитора
Длительная работа с мышью и клавиатурой	Усталость кистей рук	Удобное расположение устройств ввода, кратковременные разминки
Недостаточное или чрезмерное освещение	Повышенная зрительная нагрузка	Использование равномерного искусственного или естественного освещения

Фактор	Возможное последствие	Меры снижения риска
Электрический ток	Риск поражения электрическим током или возгорания	Использование исправной техники без видимых повреждений и качественных сетевых фильтров

Чтобы снизить зрительную нагрузку, экран монитора должен находиться примерно на расстоянии вытянутой руки. Лучше расположить монитор на уровне глаз или чуть ниже. Это позволяет также снизить нагрузку на шею при длительной работе.

Важную роль играет равномерное освещение. Монитор стоит разместить таким образом, чтобы на него не попадали прямые солнечные лучи. Также не стоит работать в помещении без освещения, так как это сильно увеличивает зрительную нагрузку.

Также при длительной работе с таблицами периодически стоит делать короткие перерывы. Нужно отвлечься от экрана, переключить внимание на какой-либо объект вдали, размять кисти рук, шею и спину.

Важно следить за своим рабочим местом. У стула или рабочего кресла должна быть поддержка спины, а также верно отрегулирована высота, чтобы руки находились в естественном положении, а ноги были в устойчивом положении. Мышь и клавиатура должны быть расположены таким образом, чтобы при работе с ними не напрягались запястья, руки и плечи. Несоблюдение этих простых правил при длительной работе приводит к дополнительной утомляемости и снижению работоспособности.

Учитывая меры электробезопасности, перед началом работы нужно убедиться, что все оборудование исправно, включая розетки, сетевые фильтры, кабели питания и тд. Недопустимо использовать поврежденные провода, блоки питания, розетки. Подключение компьютера лучше производить с использованием сетевого фильтра или с источником бесперебойного напряжения.

Рядом с любым из перечисленных электрических устройств запрещено размещать любые жидкости, которые при попадании на приборы может привести к короткому замыканию, поломке техники или даже к поражению электрическим током.

Также не стоит подключать много устройств к одному удлинителю или же оставлять включенные устройства без присмотра. Помещения, в которых есть компьютеры, должны быть оборудованы средствами пожаротушения, а также должен быть обеспечен свободный доступ к ним. Пользователь должен быть ознакомлен с тем, как необходимо поступать при появлении запаха гари, задымлении или возгорании электрических приборов. При появлении любых признаков неисправности или поломки оборудования, нужно немедленно остановить работу и отключить устройство от сети.

Таким образом, при соблюдении этих правил, можно минимизировать опасные факторы и сделать взаимодействие с системой более удобным и продуктивным.

					ПГУ1.090301.46.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		64

Заключение

Результатом выполнения выпускной квалификационной работы является успешно разработанное мобильное приложение «Профитнес», полностью соответствующее заданию. Выполнены следующие задачи:

- анализ предметной области и аналогов;
- выбор технологического стека;
- проектирование архитектуры;
- разработка серверной части на Go с использованием Gin, PostgreSQL и Redis;
- разработка клиентской части на React Native;
- интеграция с CRM-системой Impulse;
- тестирование ПО.

В результате создано полностью протестированное клиент-серверное приложение. В ходе разработки получены навыки работы с Gin, PostgreSQL, Redis и интеграцией мобильного приложения со сторонними CRM-системами. Приложение соответствует поставленным задачам, имеет интуитивно понятный интерфейс в стиле Material Design 3 и работает корректно.

В перспективе приложение можно улучшить, добавив онлайн-оплату абонементов, чат с тренером и видеотренировки. Приложение готово к использованию в фитнес-центрах и поможет повысить качество обслуживания клиентов и их лояльность за счет системы геймификации.

Список использованных источников

1. Галеев, И. Х. Введение в язык Go: учебное пособие / И. Х. Галеев, П. В. Иванов. – Казань: Казанский университет, 2020. – 112 с.
2. Строукер, Д. Go на практике / Д. Строукер, М. Бейли, К. Дьюри. – Москва: ДМК Пресс, 2019. – 478 с.
3. ClassPass [Электронный ресурс]: официальный сайт. – URL: <https://classpass.com>.
4. Donovan, A. A. The Go Programming Language / A. A. Donovan, B. W. Kernighan. – New York: Addison-Wesley, 2016. – 380 p.
5. Eisenman, B. Learning React Native: Building Native Mobile Apps with JavaScript / B. Eisenman. – 2nd ed. – Sebastopol: O'Reilly Media, 2017. – 241 p.
6. Labouardy, M. Building Distributed Applications in Gin : A Hands-On Guide for Go Developers to Build and Deploy Distributed Web Apps with the Gin Framework / M. Labouardy. – Birmingham: Packt Publishing, 2021. – 456 p.
7. Tigor, H. React Native in Action: Developing iOS and Android Apps with JavaScript / H. Tigor. – Shelter Island: Manning Publications, 2019. – 352 p.

Приложение А

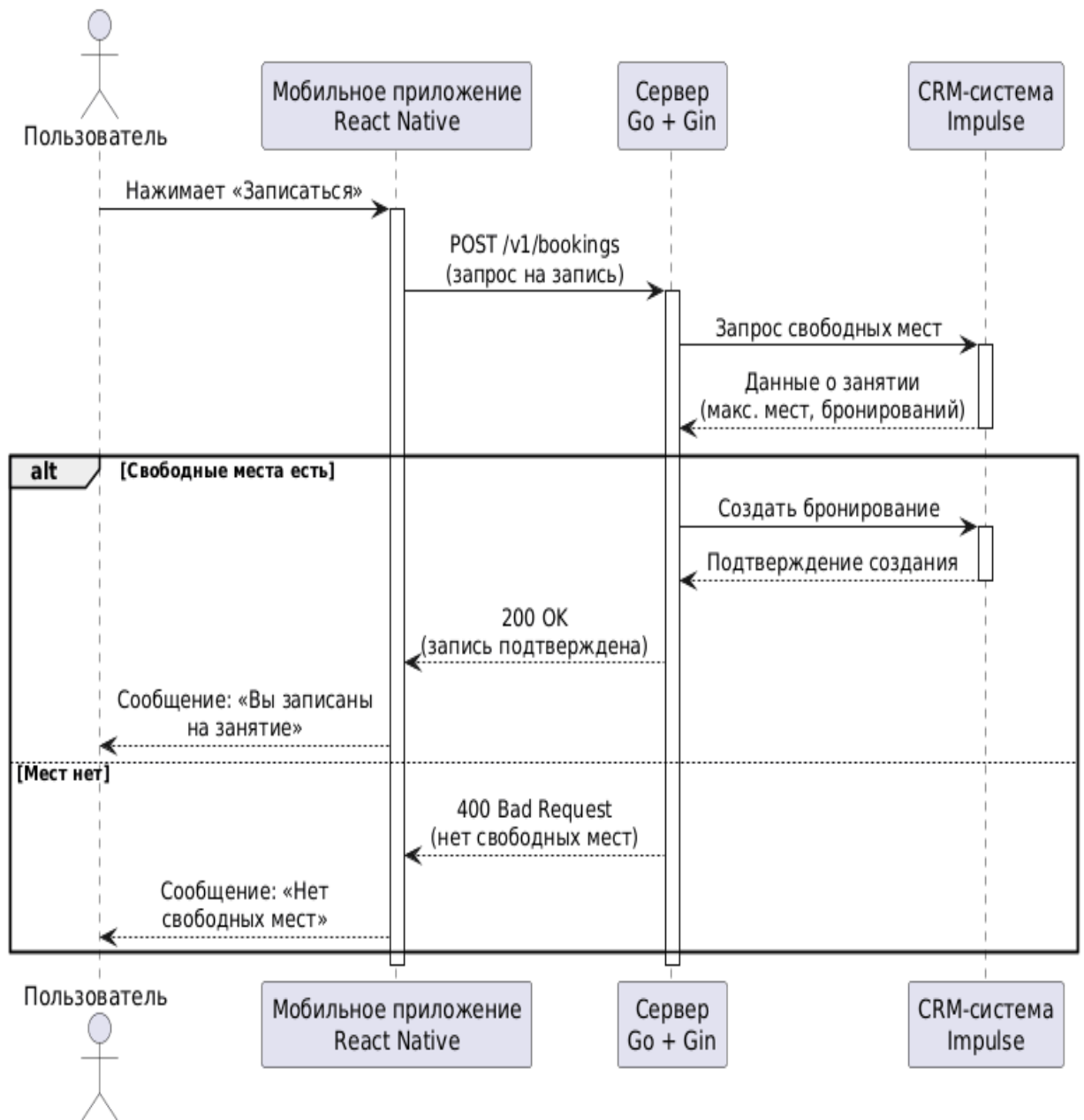
(обязательное)

Мобильное приложение «Профитнес»

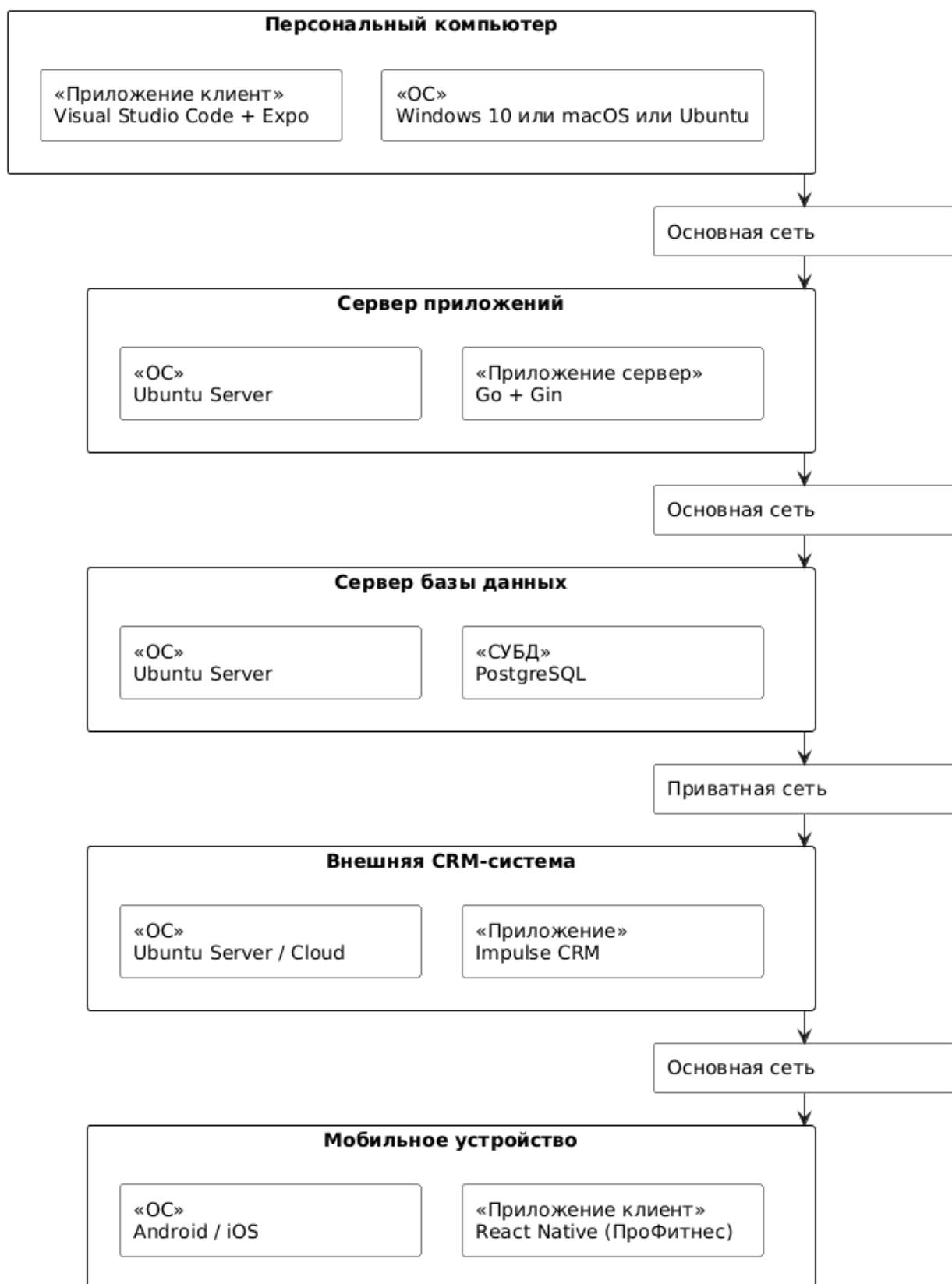
UML-диаграммы

					ПГУ1.090301.46.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		67

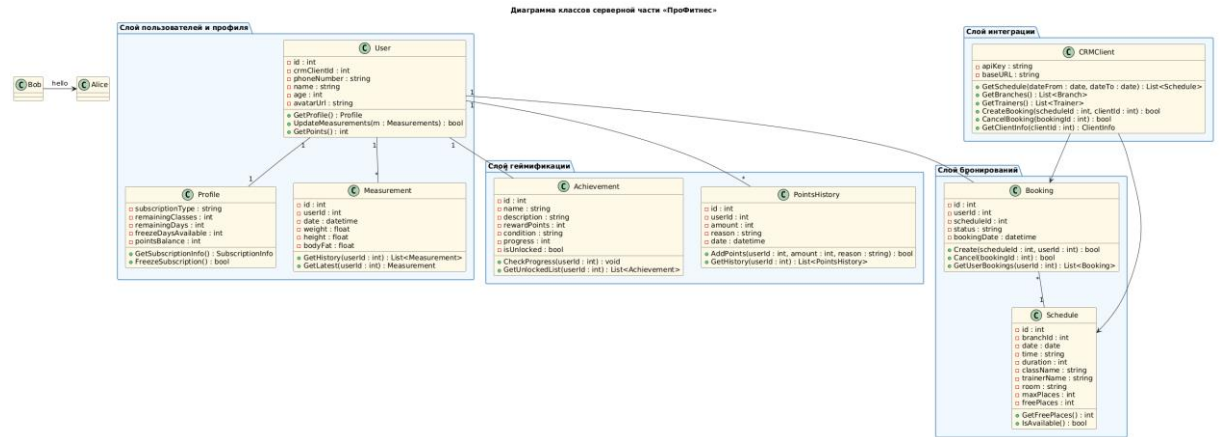
UML – диаграмма последовательности



UML – диаграмма развертывания



UML – диаграмма классов



Изм.	Лист	№ докум.	Подпись	Дата

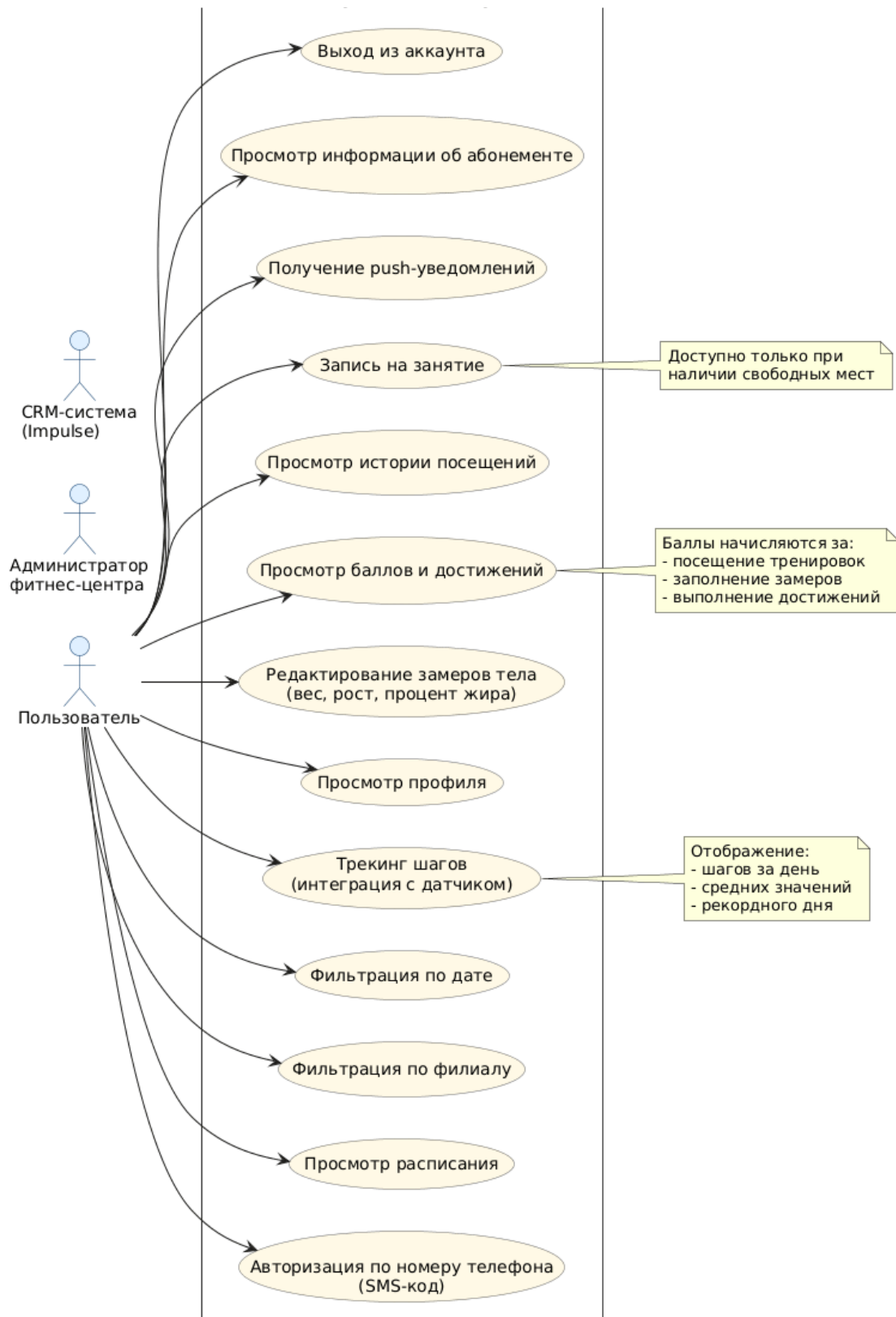
ПГУ1.090301.46.001 ПЗ

Лист

70

UML – диаграмма прецедентов

Диаграмма вариантов использования системы «ПроФитнес»



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.46.001 ПЗ

Лист

71

Приложение Б

(обязательное)

Мобильное приложение «Профитнес»

Листинг программы

					ПГУ1.090301.46.001 ПЗ	Лист
						72
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «main.go»

```
package main

import (
    "context"
    "fmt"
    "log"
    "time"
    "backend/configs"
    "backend/internal/api"
    "backend/internal/services/bookings"
    "backend/internal/services/crm"
    "backend/internal/services/waitlist"
    "backend/internal/storage/postgres"
    "backend/internal/storage/redis"
    "github.com/gin-contrib/cors"
    "github.com/gin-gonic/gin"
)

func main() {
    cfg, err := configs.Load()
    if err != nil {
        log.Fatalf("Ошибка загрузки конфигурации: %v", err)
    }
    if err := crm.InitGlobalClient(cfg.CRM.APIKey,
cfg.CRM.BaseURL); err != nil {
        log.Fatalf("Ошибка инициализации CRM клиента: %v", err)
    }
    fmt.Println(" CRM клиент инициализирован")
    db, err := postgres.NewPostgres()
    if err != nil {
        log.Fatalf("Ошибка подключения к БД: %v", err)
    }
    rdb, err := redis.NewRedis()
    if err != nil {
        log.Fatalf("Ошибка подключения к Redis: %v", err)
    }
    wl := waitlist.NewService(rdb, waitlist.Config{
```



```

        CRMBaseURL: cfg.CRM.BaseURL,
        CRMAPIKey:  cfg.CRM.APIKey,
        OfferTTL:   20 * time.Minute,
    })

    if err := wl.StartExpirationListener(context.Background());
err != nil {

        log.Fatalf("✗ Ошибка запуска expiration listener:
%v", err)

    }

    bsvc := bookings.NewService(cfg.CRM.BaseURL,
cfg.CRM.APIKey, wl)

    r := gin.Default()
    r.Use(cors.New(cors.Config{
        AllowOrigins:      []string{"http://localhost:5173",
"http://localhost:3000"},
        AllowMethods:       []string{"GET", "POST", "PUT",
"DELETE", "OPTIONS"},
        AllowHeaders:        []string{"Origin", "Content-Type",
"Authorization"},
        AllowCredentials: true,
    })))

    api.RegisterRoutes(r, db, rdb, wl, bsvc)
    port := ":8080"

    fmt.Printf("🚀 Сервер запущен на http://localhost%s\n",
port)

    if err := r.Run(port); err != nil {

        log.Fatal("✗ Ошибка запуска сервера:", err)

    }

}

```

Файл «config.yaml»

```
server:
  port: "8080"
  env: "development"
database:
  host: "localhost"
  port: 5432
  user: "fitness_pro"
  password: "qwe1234!)"
  dbname: "fitness_app"
crm:
  base_url:
    "https://profitnes31gmailcom.impulsecrm.ru/api/public/"
  api_key: "0cfefe8ddcb57f74c351bc1791372c23"
redis:
  host: "localhost"
  port: 6379
  password: ""
sms:
  provider: "twilio"
  sid: ""
  token: ""
  from_number: ""
notifications:
  telegram_bot_token: ""
  firebase_credentials_path: "./firebase-credentials.json"
features:
  enable_streaks: true
  enable_points: true
  enable_measurements: true
```

					ПГУ1.090301.46.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		75

Файл «auth.go»

```
func GetClientInfo(c *gin.Context) {
    clientID, _ := c.Get("client_id")
    phone, _ := c.Get("phone")
    name, _ := c.Get("name")

    c.JSON(http.StatusOK, gin.H{
        "success": true,
        "client": gin.H{
            "id":    clientID,
            "phone": phone,
            "name":   name,
        },
    })
}

func findClientInCRM(phone string) (map[string]interface{},
error) {
    crmClient := crm.GetClient()
    if crmClient == nil {
        return nil, fmt.Errorf("CRM клиент не инициализирован")
    }

    requestBody := map[string]interface{}{
        "limit": 1,
        "columns": map[string]interface{}{
            "phone": phone,
        },
    }

    body, err := crmClient.Post("client/list", requestBody)
    if err != nil {
        return nil, err
    }

    var result map[string]interface{}
    if err := json.Unmarshal(body, &result); err != nil {
        return nil, err
    }
}
```

```

        if items, ok := result["items"].([]interface{}); ok &&
len(items) > 0 {
            if clientData, ok := items[0].(map[string]interface{});
ok {
                return clientData, nil
            }
        }
        return nil, nil
    }
}

func getSubscriptionFromCRM(clientID int) (*time.Time, error) {
    crmClient := crm.GetClient()
    requestBody := map[string]interface{}{
        "limit": 1,
        "columns": map[string]interface{}{
            "client": clientID,
            "active": true,
        },
    }

    body, err :=
crmClient.Post("api/public/individual_account/list",
requestBody)

    if err != nil {
        return nil, err
    }

    var result map[string]interface{}
    if err := json.Unmarshal(body, &result); err != nil {
        return nil, err
    }

    if items, ok := result["items"].([]interface{}); ok &&
len(items) > 0 {
        account := items[0].(map[string]interface{})
        if endDate, ok := account["endDate"].(float64); ok {
            t := time.Unix(int64(endDate), 0)
            return &t, nil
        }
    }

    return nil, nil
}

```

					ПГУ1.090301.46.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		77

}

					ПГУ1.090301.46.001 ПЗ	Лист
						78
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «bookings.go»

```
func GetMyBookings(c *gin.Context) {
    clientIDAny, ok := c.Get("client_id")
    if !ok {
        c.JSON(http.StatusUnauthorized, gin.H{"error":
"unauthorized"})
        return
    }
    clientID, _ := clientIDAny.(int)
    if clientID <= 0 {
        switch v := clientIDAny.(type) {
        case float64:
            clientID = int(v)
        case string:
            clientID, _ = strconv.Atoi(v)
        }
    }
    if clientID <= 0 {
        c.JSON(http.StatusUnauthorized, gin.H{"error":
"unauthorized"})
        return
    }
    crmClient := crm.GetClient()
    if crmClient == nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error":
"crm client not initialized"})
        return
    }
    body, err := crmClient.Post("reservation/list",
map[string]interface{}{
        "limit": 200,
        "page": 1,
        "columns": map[string]interface{}{
            "client": clientID,
        },
    })
    if err != nil {
```



```

        c.JSON(http.StatusBadGateway, gin.H{"error": "crm
request failed", "details": err.Error()})

        return
    }

    var parsed struct {
        Total int `json:"total"`
        Items []struct {
            ID      int `json:"id"`
            Schedule struct {
                ID      int `json:"id"`
                Date      int64 `json:"date"`
                MinutesBegin int `json:"minutesBegin"`
                Hall      *struct {
                    Name string `json:"name"`
                } `json:"hall"`
                Group *struct {
                    PlaceCount int `json:"placeCount"`
                    Style      *struct {
                        Name string `json:"name"`
                    } `json:"style"`
                    Teacher1 *struct {
                        LastName string `json:"lastName"`
                        Name      string `json:"name"`
                        MiddleName string `json:"middleName"`
                    } `json:"teacher1"`
                } `json:"group"`
            } `json:"schedule"`
        } `json:"items"`
    }

    if err := json.Unmarshal(body, &parsed); err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error":
"crm parse failed", "details": err.Error()})
        return
    }

    out := make([]BookingItem, 0, len(parsed.Items))
    for _, it := range parsed.Items {
        t := time.Unix(it.Schedule.Date, 0)

```

```

        h := it.Schedule.MinutesBegin / 60
        m := it.Schedule.MinutesBegin % 60
        className := ""
        if it.Schedule.Group != nil && it.Schedule.Group.Style
!= nil {
            className = it.Schedule.Group.Style.Name
        }
func BookSchedule(c *gin.Context) {
    clientIDAny, ok := c.Get("client_id")
    if !ok {
        c.JSON(http.StatusUnauthorized, gin.H{"error":
"unauthorized"})
        return
    }
    clientID, _ := clientIDAny.(int)
    if clientID <= 0 {
        switch v := clientIDAny.(type) {
        case float64:
            clientID = int(v)
        case string:
            clientID, _ = strconv.Atoi(v)
        }
    }
    if clientID <= 0 {
        c.JSON(http.StatusUnauthorized, gin.H{"error":
"unauthorized"})
        return
    }
var req struct {
    ScheduleID int `json:"schedule_id" binding:"required"`
}
    if err := c.ShouldBindJSON(&req); err != nil ||
req.ScheduleID <= 0 {
        c.JSON(http.StatusBadRequest, gin.H{"error": "invalid
request"})
        return
    }

```



```

    crmClient := crm.GetClient()
    if crmClient == nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error":
"crm client not initialized"})
        return
    }

    _, err := crmClient.Post("reservation/update",
map[string]interface{}{
        "schedule": req.ScheduleID,
        "client":   clientID,
    })
    if err != nil {
        c.JSON(http.StatusBadGateway, gin.H{"error": "crm
request failed", "details": err.Error()})
        return
    }
    c.JSON(http.StatusOK, gin.H{
        "success": true,
        "message": "Запись создана",
    })
}

func CancelBooking(c *gin.Context) {
    bookingIDStr := c.Param("id")
    bookingID, _ := strconv.Atoi(bookingIDStr)
    if bookingID <= 0 {
        c.JSON(http.StatusBadRequest, gin.H{"error": "invalid
booking id"})
        return
    }

    crmClient := crm.GetClient()
    if crmClient == nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error":
"crm client not initialized"})
        return
    }

```

```

    scheduleID, err :=
loadReservationScheduleID(c.Request.Context(), crmClient,
bookingID)

    if err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": "crm load
failed", "details": err.Error()})
        return
    }

_, err = crmClient.Post("reservation/delete",
map[string]interface{}{
    "id": bookingID,
})
if err != nil {
    c.JSON(http.StatusBadRequest, gin.H{"error": "crm
delete failed", "details": err.Error()})
    return
}

if waitlistSvc != nil && scheduleID > 0 {
    _ = waitlistSvc.Process(c.Request.Context(),
scheduleID)
}

c.JSON(http.StatusOK, gin.H{
    "success": true,
    "message": "Бронирование отменено",
})
}

func loadReservationScheduleID(ctx context.Context, crmClient
*crm.Client, reservationID int) (int, error) {
    body, err :=
crmClient.Get(fmt.Sprintf("reservation/load?id=%d",
reservationID))
    if err != nil {
        return 0, err
    }

    var parsed struct {
        ID      int `json:"id"`
        Schedule struct {
            ID int `json:"id"`
        } `json:"schedule"`
    }

```

```

    }

    if err := json.Unmarshal(body, &parsed); err != nil {
        return 0, err
    }

    if parsed.Schedule.ID <= 0 {
        return 0, fmt.Errorf("schedule.id not found for
reservation %d", reservationID)
    }

    return parsed.Schedule.ID, nil
}

```

					ПГУ1.090301.46.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		84

Файл «branches.go»

```
func GetBranches(c *gin.Context) {
    fmt.Println("[API] GET /api/public/branch - Запрос филиалов")
    branches, err := fetchBranchesFromCRM()
    if err != nil {
        fmt.Printf("[API] Ошибка получения филиалов: %v\n", err)
        c.JSON(http.StatusInternalServerError, gin.H{
            "success": false,
            "error":   "Ошибка получения филиалов из CRM",
            "details": err.Error(),
        })
        return
    }
    fmt.Printf("[API] Успешно получено %d филиалов\n", len(branches))
    c.JSON(http.StatusOK, gin.H{
        "success": true,
        "data":    branches,
        "meta": gin.H{
            "total":    len(branches),
            "timestamp": time.Now().Unix(),
            "source":    "impulseCRM",
        },
    })
}

func fetchBranchesFromCRM() ([]BranchResponse, error) {
    crmClient := crm.GetClient()
    if crmClient == nil {
        return nil, fmt.Errorf("CRM клиент не инициализирован")
    }
    requestBody := map[string]interface{}{
        "fields": []string{"id", "name", "phone"},
        "limit":  20,
        "page":   1,
        "sort":   map[string]string{"id": "asc"},
    }
```

```

    }

    fmt.Printf("[CRM] Запрашиваем филиалы...\n")
    body, err := crmClient.Post("branch/list", requestBody)
    if err != nil {
        fmt.Printf("[CRM ERROR] Ошибка при запросе к CRM:
%v\n", err)
        return nil, fmt.Errorf("ошибка запроса к CRM: %v", err)
    }

    fmt.Printf("[CRM] Найдено %d элементов в items\n",
len(items))
    for i, item := range items {
        if branchData, ok := item.(map[string]interface{}); ok
{
            fmt.Printf("[CRM DEBUG] Элемент %d: %+v\n", i,
branchData)

            branch := convertCRMBranch(branchData)
            if branch.ID > 0 {
                branches = append(branches, branch)
            }
        }
    }

    fmt.Printf("[CRM] Успешно сконвертировано %d филиалов\n",
len(branches))
    return branches, nil
}

func convertCRMBranch(crmData map[string]interface{})
BranchResponse {
    branch := BranchResponse{
        IsActive: true,

        if id, ok := crmData["id"].(float64); ok {
            branch.ID = int(id)
        }

        if name, ok := crmData["name"].(string); ok {

```

```

        branch.Name = name
    } else {
        branch.Name = "Филиал"
    }
    if phone, ok := crmData["phone"].(string); ok && phone != "" {
        branch.Phone = phone
    } else {
        branch.Phone = ""
    }
    return branch
}

func GetBranchesPublic(c *gin.Context) {
    fmt.Println("[API] POST /api/public/branch/list - Запрос филиалов")
    var request struct {
        Limit    int    `json:"limit,omitempty"`
        Page     int    `json:"page,omitempty"`
        Active   bool   `json:"active_only,omitempty"`
    }
    if err := c.ShouldBindJSON(&request); err != nil {
        request.Limit = 100
        request.Page = 1
        request.Active = true
    }
    branches, err := fetchBranchesFromCRM()
    if err != nil {
        fmt.Printf("[API ERROR] Ошибка получения филиалов: %v\n", err)
        c.JSON(http.StatusInternalServerError, gin.H{
            "success": false,
            "error":    "Ошибка получения филиалов из CRM",
            "details":  err.Error(),
        })
        return
    }
}

```



```

    fmt.Printf("[API] Успешно получено %d филиалов\n",
len(branches))
    c.JSON(http.StatusOK, gin.H{
        "success": true,
        "items":    branches,
        "meta": gin.H{
            "total": len(branches),
        },
    })
}

```

					ПГУ1.090301.46.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		88

Файл «handlers.go»

```
package api
import (
    "database/sql"
    "backend/internal/services/bookings"
    "backend/internal/services/waitlist"
    "github.com/gin-gonic/gin"
    "github.com/redis/go-redis/v9"
)
var (
    waitlistSvc *waitlist.Service
    bookingsSvc *bookings.Service
)
func RegisterRoutes(r *gin.Engine, db *sql.DB, rdb
*redis.Client, wl *waitlist.Service, bk *bookings.Service) {
    authHandler := &AuthHandler{DB: db}
    waitlistHandler := NewWaitlistHandler(wl)
    r.GET("/health", func(c *gin.Context) {
        c.JSON(200, gin.H{"status": "ok"})
    })
    public := r.Group("/api/public")
    {
        public.POST("/branch/list", GetBranchesPublic)
        public.POST("/schedule/list", GetSchedulePublic)
        public.POST("/teacher/list", GetTeachersPublic)
    }
    v1 := r.Group("/api/v1")
    {
        v1.POST("/auth/phone", authHandler.AuthByPhone)
        v1.POST("/auth/send-code", SendCode)
        v1.POST("/auth/verify", VerifyCode)

        v1.POST("/waitlist/join", waitlistHandler.JoinWaitlist)
        v1.POST("/waitlist/confirm",
waitlistHandler.ConfirmSpot)
```

```

protected := v1.Group("/")
protected.Use(ValidateTokenMiddleware())
{
    protected.GET("/bookings", GetMyBookings)
    protected.POST("/bookings", BookSchedule)
    protected.DELETE("/bookings/:id", CancelBooking)
}
}
}

```

					ПГУ1.090301.46.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		90

Файл «schedule.go»

```
func parseSchedule(body []byte) ([]ScheduleItem, error) {
    var wrapper struct {
        Data struct {
            Total int           `json:"total"`
            Items []map[string]any `json:"items"`
        } `json:"data"`
    }

    if err := json.Unmarshal(body, &wrapper); err == nil &&
        wrapper.Data.Total > 0 {
        return parseScheduleItems(wrapper.Data.Items)
    }

    var direct struct {
        Total int           `json:"total"`
        Items []map[string]any `json:"items"`
    }

    if err := json.Unmarshal(body, &direct); err == nil &&
        direct.Total > 0 {
        return parseScheduleItems(direct.Items)
    }

    var arr []map[string]any
    if err := json.Unmarshal(body, &arr); err == nil {
        return parseScheduleItems(arr)
    }

    return []ScheduleItem{}, nil
}

func parseScheduleItems(items []map[string]any)
([]ScheduleItem, error) {
    out := make([]ScheduleItem, 0, len(items))
    for _, it := range items {
        // id
        id := asInt(it["id"])
        if id <= 0 {
            continue
        }

        dateUnix := asInt64(it["date"])
        dateStr := time.Unix(dateUnix, 0).Format("2006-01-02")
    }
}
```

```

minBegin := asInt(it["minutesBegin"])
h := minBegin / 60
m := minBegin % 60
timeStr := fmt.Sprintf("%02d:%02d", h, m)
minEnd := asInt(it["minutesEnd"])
dur := 60
if minEnd > minBegin {
    dur = minEnd - minBegin
}
item := ScheduleItem{
    ID:      id,
    Date:    dateStr,
    Time:    timeStr,
    Duration: dur,
    MaxPlaces: 1,
}

func parseReservationCounts(body []byte) (map[int]int, error) {
    counts := make(map[int]int)

    var wrapper struct {
        Data struct {
            Total int          `json:"total"`
            Items []map[string]any `json:"items"`
        } `json:"data"`
    }

    if err := json.Unmarshal(body, &wrapper); err == nil {
        for _, it := range wrapper.Data.Items {
            sid := extractScheduleID(it)
            if sid > 0 {
                counts[sid]++
            }
        }
        return counts, nil
    }

    var direct struct {

```

```

    Total int                `json:"total"`
    Items []map[string]any `json:"items"`
}

if err := json.Unmarshal(body, &direct); err == nil {
    for _, it := range direct.Items {
        sid := extractScheduleID(it)
        if sid > 0 {
            counts[sid]++
        }
    }
    return counts, nil
}

var arr []map[string]any
if err := json.Unmarshal(body, &arr); err == nil {
    for _, it := range arr {
        sid := extractScheduleID(it)
        if sid > 0 {
            counts[sid]++
        }
    }
}

return counts, nil
}

func extractScheduleID(item map[string]any) int {
    if sch, ok := item["schedule"].(map[string]any); ok {
        if id := asInt(sch["id"]); id > 0 {
            return id
        }
    }

    if id := asInt(item["schedule_id"]); id > 0 {
        return id
    }

    if id := asInt(item["id"]); id > 0 {
        return id
    }

    return 0
}

```


}

					ПГУ1.090301.46.001 ПЗ	Лист
						94
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «teachers.go»

```
func fetchTeachersFromCRM(limit, page int, activeOnly bool)
([]TeacherResponse, error) {
    crmClient := crm.GetClient()
    if crmClient == nil {
        return nil, fmt.Errorf("CRM клиент не инициализирован")
    }
    requestBody := map[string]interface{}{
        "fields": []string{"id", "firstName", "lastName",
"middleName", "phone", "email", "isActive"},
        "limit":  limit,
        "page":    page,
        "sort":    map[string]string{"lastName": "asc"},
    }

    if activeOnly {
        requestBody["columns"] = map[string]interface{}{
            "isActive": true,
        }
    }

    fmt.Printf("[CRM] Запрашиваем тренеров: limit=%d, page=%d,
active=%v\n",
        limit, page, activeOnly)
    body, err := crmClient.Post("teacher/list", requestBody)
    if err != nil {
        return nil, err
    }
    return parseCRMTeachersResponse(body)
}

func parseCRMTeachersResponse(body []byte) ([]TeacherResponse,
error) {
    var result map[string]interface{}
    if err := json.Unmarshal(body, &result); err != nil {
        return nil, fmt.Errorf("ошибка парсинга JSON: %v", err)
    }
    var teachers []TeacherResponse
```

```

var items []interface{}
var found bool
if data, ok := result["items"].([]interface{}); ok {
    items = data
    found = true
    fmt.Printf("[CRM] Найдено %d тренеров в 'items'\n",
len(items))
}
if !found {
    if data, ok := result["data"].([]interface{}); ok {
        items = data
        found = true
        fmt.Printf("[CRM] Найдено %d тренеров в 'data'\n",
len(items))
    }
}
if !found {
    fmt.Println("[CRM] В ответе нет ни items, ни data")
    return teachers, nil
}
for _, item := range items {
    if teacherData, ok := item.(map[string]interface{}); ok
{
        teacher := convertCRMTeacherToResponse(teacherData)
        // Добавляем только если есть ID
        if teacher.ID > 0 {
            teachers = append(teachers, teacher)
        }
    }
}
fmt.Printf("[CRM] Успешно сконвертировано %d тренеров\n",
len(teachers))
return teachers, nil
}

func convertCRMTeacherToResponse(crmTeacher
map[string]interface{}) TeacherResponse {
    teacher := TeacherResponse{

```



```

        HallID: 1,
        IsActive: true,
    }
    if id, ok := crmTeacher["id"].(float64); ok {
        teacher.ID = int(id)
    }
    lastName, _ := crmTeacher["lastName"].(string)
    firstName, _ := crmTeacher["name"].(string) // ВАЖНО: в CRM
имя в поле "name"
    middleName, _ := crmTeacher["middleName"].(string)

```

					ПГУ1.090301.46.001 ПЗ	Лист
						97
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «init.sql»

```

CREATE TABLE IF NOT EXISTS app_users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    crm_client_id INTEGER NOT NULL UNIQUE,
    phone VARCHAR(20) NOT NULL UNIQUE,
    notification_preference VARCHAR(10) DEFAULT 'push', --
    'push' или 'telegram'
    telegram_chat_id BIGINT,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE IF NOT EXISTS streaks (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES app_users(id) ON DELETE CASCADE,
    current_streak INTEGER DEFAULT 0,
    best_streak INTEGER DEFAULT 0,
    last_visit_date DATE,
    check_date DATE DEFAULT CURRENT_DATE,
    UNIQUE(user_id, check_date)
);

CREATE TABLE IF NOT EXISTS points (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES app_users(id) ON DELETE CASCADE,
    amount INTEGER NOT NULL,
    reason VARCHAR(50) NOT NULL, -- 'attended',
    'future_booking', 'measurement'
    metadata JSONB,
    created_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_points_user_id ON points(user_id);
CREATE INDEX idx_points_created_at ON points(created_at DESC);

```

Файл «db.go»

```
package postgres

import (
    "database/sql"
    "fmt"
    _ "github.com/lib/pq"
)

func NewPostgres() (*sql.DB, error) {
    dsn :=
    "postgres://fitness_user:your_password@172.21.124.213:5432/fitn
    ess_app?sslmode=disable"
    db, err := sql.Open("postgres", dsn)
    if err != nil {
        return nil, fmt.Errorf("ошибка открытия БД: %w", err)
    }
    if err = db.Ping(); err != nil {
        return nil, fmt.Errorf("ошибка подключения к БД: %w",
err)
    }
    fmt.Println("✅ PostgreSQL подключен")
    return db, nil
}
```


Файл «redis.go»

```
package redis

import (
    "context"
    "fmt"
    goredis "github.com/redis/go-redis/v9"
)

var Ctx = context.Background()

func NewRedis() (*goredis.Client, error) {
    rdb := goredis.NewClient(&goredis.Options{
        Addr: "localhost:6379",
    })

    _, err := rdb.Ping(Ctx).Result()
    if err != nil {
        return nil, err
    }

    fmt.Println("✅ Redis подключен")
    return rdb, nil
}
```

Файл «.env»

```
CRM_API_KEY=0cfefe8ddcb57f74c351bc1791372c23
CRM_BASE_URL=https://profitnes31gmailcom.impulsecrm.ru/api/v1/
```

					ПГУ1.090301.46.001 ПЗ	Лист
						100
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «RecordsScreen.js»

```
import { SafeAreaView } from 'react-native-safe-area-context';
import { Pedometer } from 'expo-sensors';

const { width } = Dimensions.get('window');
const isSmallDevice = width < 375;
const Icon = ({ name, size, color }) => {
  const [iconComponent, setIconComponent] = useState(null);
  useEffect(() => {
    import('../assets/icons/icon').then(module => {
      setIconComponent(() => module.default);
    }).catch(err => {
      console.log('Icon loading error:', err);
    });
  }, []);
  if (!iconComponent) {
    return <View style={{ width: size, height: size }} />;
  }
  const IconComponent = iconComponent;
  return <IconComponent name={name} size={size} color={color} />;
};

export default function RecordsScreen({ navigation }) {
  const [stepsToday, setStepsToday] = useState(0);
  const [stepGoal, setStepGoal] = useState(10000);
  const [stepStreak, setStepStreak] = useState(7);
  const [stepHistory, setStepHistory] = useState([]);
  const [bestDay, setBestDay] = useState(0);
  const [avgWeek, setAvgWeek] = useState(0);
  const [avgMonth, setAvgMonth] = useState(0);
  const [showStepHistory, setShowStepHistory] =
    useState(false);
  const [attendanceStreak, setAttendanceStreak] =
    useState(5);
  const [weeklyGoal, setWeeklyGoal] = useState(2);
```

```

    const [completedThisWeek, setCompletedThisWeek] =
useState(1);

    const [selectedAchievement, setSelectedAchievement] =
useState(null);

    const [achievementModalVisible, setAchievementModalVisible]
= useState(false);

    const [infoModalVisible, setInfoModalVisible] =
useState(false);

    const [selectedInfoAchievement, setSelectedInfoAchievement]
= useState(null);

    const [achievements, setAchievements] = useState([
        {
            id: 1,
            title: 'Начало положено',
            completed: true,
            date: '2024-01-10',
            points: 50,
            icon: 'user_check',
            total: 1,
            current: 1,
            description: 'Заполнение профиля (имя, дата
рождения, цель и т.д.)'
        },

        useEffect(() => {
            const initPedometer = async () => {
                try {
                    const isAvailable = await
Pedometer.isAvailableAsync();
                    if (isAvailable) {
                        const { granted } = await
Pedometer.requestPermissionsAsync();
                        if (granted) {
                            const end = new Date();
                            const start = new Date();
                            start.setHours(0, 0, 0, 0);
                            const result = await
Pedometer.getStepCountAsync(start, end);

```



```

        if (result && result.steps !==
undefined) {
            setStepsToday(result.steps);
        }
        const mockStepHistory = [];
        const stepValues = [];
        for (let i = 29; i >= 0; i--) {
            const date = new Date();
            date.setDate(date.getDate() - i);
            const steps =
Math.floor(Math.random() * 5000) + 5000;
            mockStepHistory.push({
                date:
date.toISOString().split('T')[0],
                steps
            });
            stepValues.push(steps);
        }
        setStepHistory(mockStepHistory);

        const best = Math.max(...stepValues,
result?.steps || 0);
        setBestDay(best);
        const lastWeekSteps =
mockStepHistory.slice(-7).map(item => item.steps);
        const avgWeekCalc =
Math.floor(lastWeekSteps.reduce((a, b) => a + b, 0) / 7);
        setAvgWeek(avgWeekCalc);
        const avgMonthCalc =
Math.floor(stepValues.reduce((a, b) => a + b, 0) / 30);
        setAvgMonth(avgMonthCalc);
    } else {
        console.log('Разрешение на шагомер не
предоставлено');
        generateMockData();
    }
} else {

```

```

        console.log('Шагомер недоступен на этом
устройстве');
        generateMockData();
    }
} catch (error) {
    console.error('Ошибка инициализации шагомера:',
error);
    generateMockData();
}
};
const generateMockData = () => {
    const mockStepHistory = [];
    const stepValues = [];
    for (let i = 29; i >= 0; i--) {
        const date = new Date();
        date.setDate(date.getDate() - i);
        const steps = Math.floor(Math.random() * 5000)
+ 5000;
        mockStepHistory.push({
            date: date.toISOString().split('T')[0],
            steps
        });
        stepValues.push(steps);
    }
    setStepHistory(mockStepHistory);
    setStepsToday(7532);
    const allSteps = [...stepValues, 7532];
    setBestDay(Math.max(...allSteps));
    const lastWeekSteps = mockStepHistory.slice(-
7).map(item => item.steps);
    setAvgWeek(Math.floor(lastWeekSteps.reduce((a, b)
=> a + b, 0) / 7));
    setAvgMonth(Math.floor(stepValues.reduce((a, b) =>
a + b, 0) / 30));
};
initPedometer();
}, []);
const goToProfile = () => navigation.navigate('Profile');

```

```

const goToTeachers = () => navigation.navigate('Teachers');
const goToSchedule = () => navigation.navigate('Schedule');
const calculateProgress = (achievement) => {
    return achievement.total > 0 ? (achievement.current /
achievement.total) * 100 : 0;
};

const openAchievementInfo = (achievement) => {
    setSelectedInfoAchievement(achievement);
    setInfoModalVisible(true);
};

const renderAchievementItem = ({ item }) => (
    <View style={[styles.achievementCard, item.completed &&
styles.achievementCardCompleted]}>
        <View style={[styles.achievementIconContainer,
item.completed && styles.achievementIconCompleted]}>
            <Icon name={item.icon} size={24}
color={item.completed ? "#FFF" : "#000"}

```


Файл «ScheduleScreen.js»

```
const initialSelected = dates.find((d) => d.isToday) ||
dates[0];

dates.forEach((d) => (d.isSelected = d.date ===
initialSelected.date));

setWeekDates(dates);

setSelectedDate(initialSelected.date);

setCurrentWeekStart(monday);

return initialSelected.date;
};

const nextWeek = () => {
  if (!currentWeekStart) return;
  const next = new Date(currentWeekStart);
  next.setDate(currentWeekStart.getDate() + 7);
  generateWeekDates(next);
};

const prevWeek = () => {
  if (!currentWeekStart) return;
  const prev = new Date(currentWeekStart);
  prev.setDate(currentWeekStart.getDate() - 7);
  generateWeekDates(prev);
};

const formatTime = (timeStr) => {
  if (!timeStr) return '--:--';
  return String(timeStr).substring(0, 5);
};

const formatWeekRange = () => {
  if (weekDates.length !== 7) return '';
  const first = weekDates[0];
  const last = weekDates[6];
  return `${first.day} ${first.month} - ${last.day}
${last.month}`;
};

const loadBranches = async () => {
  try {
    const response = await getBranches();
```

```

    let items = [];
    if (response?.data?.items) items = response.data.items;
    else if (response?.data?.data) items =
response.data.data;
    else if (Array.isArray(response?.data)) items =
response.data;
    const formatted = (items || []).map((b) => ({
      id: Number(b.id ?? b.ID ?? 0),
      name: String(b.name ?? b.Name ?? 'Филиал'),
      address: String(b.address ?? b.Address ?? ''),
    })).filter((b) => b.id > 0);
    setBranches(formatted);
    if (!selectedBranch && formatted.length > 0) {
      setSelectedBranch(formatted[0]);
    }
  } catch (e) {
    console.error('Ошибка загрузки филиалов:', e?.message ||
e);
    Alert.alert('Ошибка', 'Не удалось загрузить филиалы');
  }
};

const loadScheduleData = async (date) => {
  if (!selectedBranch?.id || !date) return;
  try {
    setRefreshing(true);
    const response = await getSchedule(date, date,
selectedBranch.id);
    let items = [];
    if (response?.data?.items) items = response.data.items;
    else if (response?.data?.data) items =
response.data.data;
    const sorted = (items || []).slice().sort((a, b) => {
      const ta = String(a.time || '00:00').substring(0, 5);
      const tb = String(b.time || '00:00').substring(0, 5);
      return ta.localeCompare(tb);
    });
    setSchedule(sorted);
  }

```

```

    } catch (e) {
        console.error('Ошибка загрузки расписания:', e?.message
|| e);
        setSchedule([]);
        Alert.alert('Ошибка', 'Не удалось загрузить расписание');
    } finally {
        setLoading(false);
        setRefreshing(false);
    }
};

```

					ПГУ1.090301.46.001 ПЗ	Лист
						108
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «api.js»

```
import axios from 'axios';
import AsyncStorage from '@react-native-async-storage/async-storage';
const API_URL = 'http://192.168.0.100:8080';
const api = axios.create({
  baseURL: API_URL,
  timeout: 20000,
  headers: { 'Content-Type': 'application/json' },
});
api.interceptors.request.use(async (config) => {
  const url = config?.url || '';
  if (url.startsWith('/api/v1/')) {
    const token = await AsyncStorage.getItem('token');
    if (token) config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});
export const getBranches = () => {
  return api.post('/api/public/branch/list', {
    limit: 50,
    page: 1,
  });
};
export const getSchedule = (date_from, date_to, branch_id) => {
  const payload = {
    date_from,
    date_to: date_to || date_from,
    branch_id,
    limit: 200,
    page: 1,
  };
  return api.post('/api/public/schedule/list', payload);
};
export const getTeachers = () => {
```

```

    return api.post('/api/public/teacher/list', {
      limit: 200,
      page: 1,
      columns: { isActive: true },
    });
  };

export const getHalls = () => {
  return api.post('/api/public/hall/list', {
    limit: 50,
    page: 1,
    columns: { isActive: true },
  });
};

export const sendSMSCode = (phone) => {
  return api.post('/api/v1/auth/phone', { phone });
};

export const verifyCode = (sessionId, code) => {
  return api.post('/api/v1/auth/verify', {
    session_id: sessionId,
    code,
  });
};

export const getMyBookings = () => {
  return api.get('/api/v1/bookings');
};

export const createBooking = (scheduleId) => {
  return api.post('/api/v1/bookings', { schedule_id: scheduleId });
};

export const cancelBooking = (bookingId) => {
  return api.delete(`/api/v1/bookings/${bookingId}`);
};

export const joinWaitlist = (scheduleId, userId) => {
  return api.post('/api/v1/waitlist/join', {
    schedule_id: scheduleId,
    user_id: userId,
  });
};

```

```

    });
};
export const confirmWaitlist = (scheduleId, userId) => {
    return api.post('/api/v1/waitlist/confirm', {
        schedule_id: scheduleId,
        user_id: userId,
    });
};
};

```

					ПГУ1.090301.46.001 ПЗ	Лист
						111
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «App.js»

```
import { registerRootComponent } from 'expo';
import React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import AuthScreen from './src/screens/AuthScreen';
import ScheduleScreen from './src/screens/ScheduleScreen';
import ProfileScreen from './src/screens/ProfileScreen';
import TeachersScreen from './src/screens/TeachersScreen';
import RecordsScreen from './src/screens/RecordsScreen';
const Stack = createNativeStackNavigator();
function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator
        initialRouteName="Auth"
        screenOptions={{
          headerShown: false,
          cardStyle: { backgroundColor: '#0A0A0A' },
        }}
      >
        <Stack.Screen name="Auth" component={AuthScreen} />
        <Stack.Screen name="Profile" component={ProfileScreen} />
        <Stack.Screen name="Teachers"
component={TeachersScreen} />
        <Stack.Screen name="Schedule"
component={ScheduleScreen} />
        <Stack.Screen name="Records" component={RecordsScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
export default registerRootComponent(App);
```

Файл «app.json»

```
{
  "expo": {
    "name": "Fitness Studio",
    "slug": "fitness-studio-app",
    "version": "1.0.0",
    "orientation": "portrait",
    "icon": "./assets/icon.png",
    "userInterfaceStyle": "light",
    "splash": {
      "image": "./assets/splash.png",
      "resizeMode": "contain",
      "backgroundColor": "#ffffff"
    },
    "assetBundlePatterns": [
      "**/*"
    ],
    "ios": {
      "supportsTablet": true,
      "bundleIdentifier": "com.yourapp.fitness",
      "infoPlist": {
        "NSPhotoLibraryUsageDescription": "Приложение  
использует галерею для выбора фото профиля",
        "NSCameraUsageDescription": "Приложение использует  
камеру для создания фото профиля"
      }
    },
    "android": {
      "adaptiveIcon": {
        "foregroundImage": "./assets/adaptive-icon.png",
        "backgroundColor": "#ffffff"
      },
      "package": "com.yourapp.fitness",
      "permissions": [
        "android.permission.READ_EXTERNAL_STORAGE",
        "android.permission.WRITE_EXTERNAL_STORAGE",

```

```

        "android.permission.CAMERA"
    ]
},
"web": {
    "favicon": "../assets/favicon.png",
    "bundler": "metro"
},
"scheme": "profitnes",
"extra": {
    "eas": {
        "projectId": "60858507-3e2e-480f-8519-8e5414fc6f86"
    }
},
"plugins": [
    [
        "expo-image-picker",
        {
            "photosPermission": "Приложение использует галерею
для выбора фото профиля",
            "cameraPermission": "Приложение использует камеру для
создания фото профиля"
        }
    ]
]
}
}

```

Файл «docker-compose.yml»

```

services:
  postgres:
    image: postgres:15-alpine
    environment:
      POSTGRES_DB: fitness_app
      POSTGRES_USER: fitness_user
      POSTGRES_PASSWORD: your_password
    ports:
      - "5432:5432"

```

```

    volumes:
      - postgres_data:/var/lib/postgresql/data
  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    command: [ "redis-server", "--notify-keyspace-events", "Ex"
]

```

					ПГУ1.090301.46.001 ПЗ	Лист
						115
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение В
(обязательное)

Мобильное приложение «Профитнес»
Презентация

					<i>ПГУ1.090301.46.001 ПЗ</i>	<i>Лист</i>
						<i>116</i>
<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>		

Мобильное приложение «Профитнес»

Автор работы: студент группы 22ВВП2 Серобаба Д.Д.

Научный руководитель: д.т.н, профессор, зав. каф. ВТ Митрохин М.А.

Пенза, 2026

Актуальность проекта

Фитнес-индустрия активно цифровизируется, однако большинство CRM-систем, используемых фитнес-центрами, не предоставляют клиентам удобных мобильных интерфейсов. Клиенты вынуждены записываться на занятия через администраторов по телефону или использовать неадаптированные веб-версии, что создает дополнительную нагрузку на персонал. Существующие на рынке мобильные приложения либо жестко привязаны к собственным экосистемам, либо не интегрируются с произвольными CRM, либо имеют высокую стоимость внедрения, недоступную для средних и малых фитнес-центров.

Современные клиенты ожидают не только базовой функциональности записи, но и элементов геймификации, мотивации и персонализации. Разработка мобильного приложения «ПроФитнес», интегрируемого с уже используемой CRM через публичное API и дополненного системой баллов и достижений, является актуальной задачей. Такое решение позволит фитнес-центру повысить качество обслуживания клиентов и снизить нагрузку на административный персонал.

					ПГУ1.090301.46.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		117

Цели работы

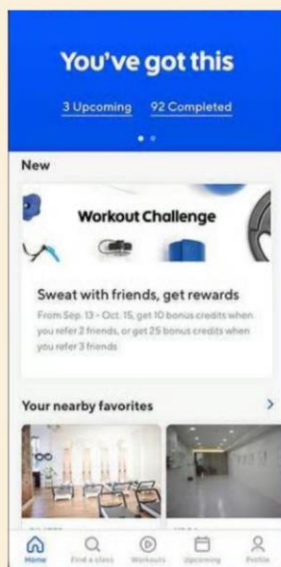
Целью выпускной квалификационной работы является разработка мобильного приложения «ПроФитнес», функционирующего в качестве клиентского портала для фитнес-центра.

Для достижения поставленной цели необходимо решить следующие задачи:

- разработать систему аутентификации по номеру телефона;
- реализовать просмотр расписания и запись на занятия;
- внедрить механизмы геймификации (баллы и достижения);
- обеспечить трекинг активности и ведение истории замеров тела пользователя.

В результате должна быть создана система, предоставляющая пользователям удобный инструмент для взаимодействия с фитнес-центром, повышающая их вовлеченность через мотивационные механики и снижающая нагрузку на административный персонал.

ClassPass

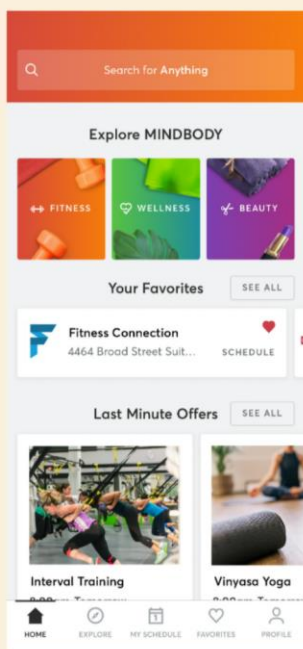


Достоинства:

- Обширная сеть заведений по всему миру
- Гибкая система подписки с разными тарифными планами
- Удобный поиск и фильтрация по типу тренировки и времени
- Отзывы и рейтинги студий и тренеров

Недостатки:

- Отсутствует интеграция с внешними CRM-системами фитнес-центров
- Отсутствует геймификация (баллы, достижения)
- Высокие комиссионные отчисления для фитнес-центра



Mindbody

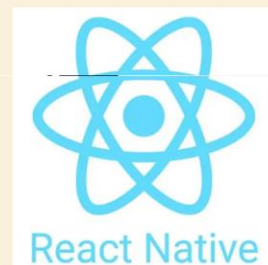
Достоинства:

- Интеграция с собственной CRM-системой Mindbody
- Удобный календарь для просмотра расписания
- Онлайн-оплата абонементов
- Push-уведомления о предстоящих тренировках

Недостатки:

- Отсутствует интеграция с произвольными CRM-системами
- Отсутствует геймификация (баллы, достижения)
- Отсутствует трекинг шагов и активности

Стек технологий



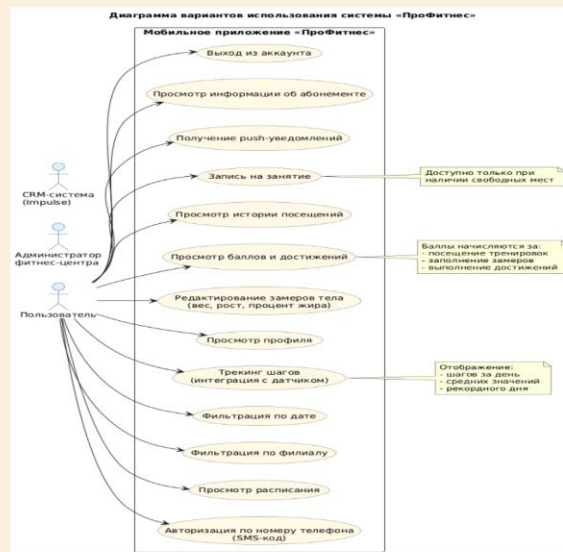
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.46.001 ПЗ

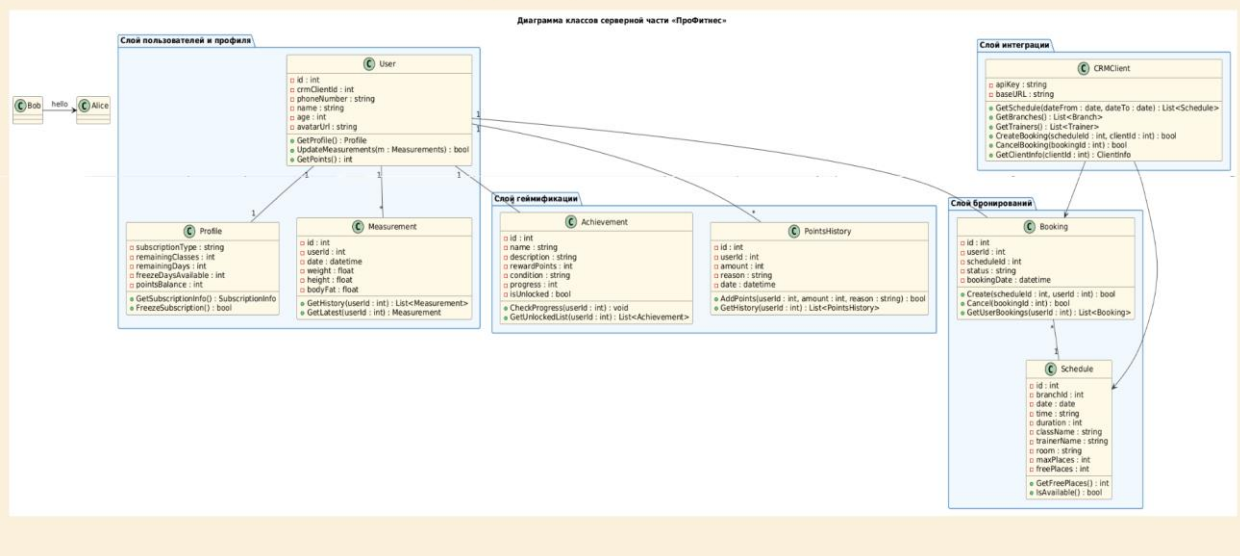
Лист

119

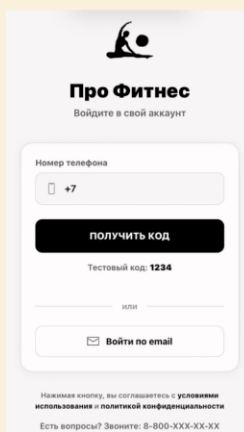
UML-диаграмма прецедентов



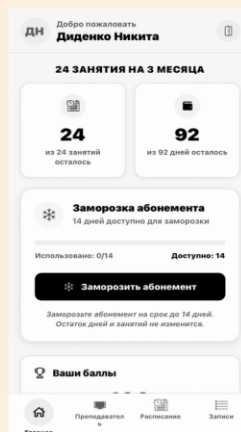
UML-диаграмма классов



Приложение «Профитнес»



экран авторизации



главный экран



экран расписания

Область применения

Мобильное приложение «Профитнес» предназначено для использования в фитнес-центрах, студиях групповых тренировок и оздоровительных клубах различных масштабов.

Приложение может быть интегрировано с любой CRM-системой, предоставляющей публичное REST API, что расширяет круг потенциальных заказчиков.

Система позволяет клиентам самостоятельно записываться на занятия, отслеживать список занятий по абонементу и просматривать историю посещений.

Геймификационный модуль может быть применен в качестве дополнительного мотивационного инструмента для повышения удержания клиентов.

Трекинг шагов и замеры тела дают возможность пользователям контролировать собственный прогресс без использования сторонних приложений.

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.46.001 ПЗ

Лист

121

Спасибо за внимание!

					ПГУ1.090301.46.001 ПЗ	Лист
						122
Изм.	Лист	№ докум.	Подпись	Дата		