

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

«ВЕБ-ПРИЛОЖЕНИЕ ФИТНЕС-ЦЕНТРА»

Студент Тарасов Никита Артемович
(подпись, дата) 28.05.2026 (ФИО полностью)

Руководитель Сорокин А.А.
(подпись, дата) 28.05.2026 (фамилия, инициалы)

Нормоконтролёр Бычков А.С.
(подпись, дата) 28.05.2026 (фамилия, инициалы)

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

Заведующий кафедрой М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой удовлетворительно (протокол заседания ГЭК от
15 июня 2026 г. № 30)

Секретарь ГЭК Е.И. Гурин
(подпись) (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» апреля 2026 г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Тарасов Никита Артемович гр. 22ВВП2 факультета ФВТ

2. Тема работы Веб приложение фитнес центра

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

3. Руководитель бакалаврской работы Сорокин А.А.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

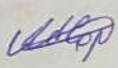
Разработать веб-приложение для автоматизации работы фитнес-центра. Тема является актуальной в условиях роста популярности здорового образа жизни и увеличения количества фитнес-центров, что приводит к повышению потока клиентов и усложнению процессов управления расписанием и записи на занятия. Для того чтобы достичь максимальной загрузки залов и оборудования, а также снизить риск ошибок при ручной обработке заявок, часто применяют системы автоматизации, которые позволяют собирать отчетность по всем тренировкам и клиентам.

Разработанное веб-приложение для автоматизации работы фитнес-центра позволяет уменьшить временные затраты персонала, избавиться от лишних ошибок и сконцентрироваться на качестве обслуживания клиентов, отслеживать всю необходимую информацию для анализа популярности услуг и загрузки тренеров, а также сокращения времени на выполнение рутинной задачи по обработке заявок.

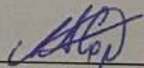
5. Перечень подлежащих разработке вопросов

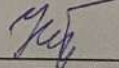
Требуется произвести анализ предметных областей: изучить существующие информационные системы, выделить их достоинства и недостатки. Выявить потребности пользователей. Выбрать язык программирования, среду разработки, изучить вопрос электротехники и схемотехники. Определить архитектуру приложения и вспомогательные средства разработки. Изучить функционал Visual Studio для разработки веб-приложения фитнес-центра.

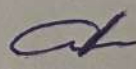
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	32	07.04.2026 – 21.04.2026	
2	Анализ требований	32	22.04.2026 – 28.04.2026	
3	Создание макета графического интерфейса	30	29.04.2026 – 02.05.2026	
4	Выбор технологий, применяемых при проектировании и разработке	40	03.05.2026 – 08.05.2026	
5	Реализация графического интерфейса	66	09.05.2026 – 16.05.2026	
6	Разработка основной части программы	70	17.05.2026 – 25.05.2026	
7	Оформление пояснительной записки	54	26.05.2026 – 30.05.2026	

Дата выдачи задания "07" апреля 2026 г.

Руководитель бакалаврской работы  Сорокин А.А.
(подпись, дата)

Задание к исполнению принял студент  Тарасов Н.А.
(подпись, дата)

Работу к защите допустить
Декан факультета  10.06.2026 Фионова Л.Р.
(подпись, дата)

Реферат

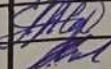
Бакалаврская работа содержит 167 листов, 41 рисунок, 1 таблица, 14 листингов, 7 источников, 3 приложения.

АВТОМАТИЗАЦИЯ, ВЕБ-ПРИЛОЖЕНИЕ, ФИТНЕС-ЦЕНТР, РАСПИСАНИЕ, ЗАПИСЬ КЛИЕНТОВ, CRM, АДМИНИСТРИРОВАНИЕ, БАЗА ДАННЫХ, АНАЛИЗ НАГРУЗКИ, ЭФФЕКТИВНОСТЬ

Цель работы – разработать веб-приложение фитнес-центра.

Разработанное приложение обеспечивает две категории пользователей. Клиенты могут просматривать расписание, записываться на занятия, отслеживать статус заявок в личном кабинете и отправлять обращения в поддержку. Администраторы получают расширенный функционал: управление тренерами, услугами, расписанием, обработку заявок и ответы на обращения. Система автоматически проверяет наличие свободных мест и предотвращает двойную запись.

Итогом работы является готовое к эксплуатации веб-приложение для операционной системы Windows, позволяющее фитнес-центру сократить временные затраты на обработку заявок, исключить ошибки, связанные с человеческим фактором, и предоставить руководству достоверную аналитическую информацию. Программа протестирована, соответствует поставленной цели и может быть расширена новыми модулями, такими как онлайн-оплата или SMS-уведомления.

					ПГУ1.090301.47.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Веб-приложение фитнес-центра Пояснительная записка	Лит.	Лист	Листов	
Разраб.		Тарасов Н.А.		28.05.20					
Провер.		Сорокин А. А.		28.05.20			2	167	
Н. Контр		Бычков А.С.		28.05.20		ФВТ, каф. ВТ гр. 22ВВП2			
Утверд.		Митрохин М.А.		28.05.20					

Содержание

Перечень сокращений и обозначений.....	5
Введение.....	7
1 Веб–приложение фитнес центра	10
1.1 Анализ предметной области	10
1.2 Обзор аналогов	14
1.3 Анализ требований к разработке.....	20
1.4 Выбор архитектуры системы.....	25
1.5 Выбор языка программирования.....	30
1.6 Выбор среды разработки	31
1.7 Проектирование и разработка.....	40
1.8 Разработка управляющей программы.....	44
1.8.1 Инициализация клиентского приложения.....	44
1.8.2 Инициализация серверного приложения.....	45
1.8.3 API-клиент для взаимодействия с сервером	45
1.8.4 Аутентификация и авторизация	46
1.8.5 Отображение расписания и запись на занятия.....	48
1.8.6 Административная панель	49
1.8.7 База данных и ORM	51
1.8.8 Вспомогательные функции и утилиты	52
1.8.9 Архитектура развертывания	53
1.9 Графический интерфейс клиентского приложения.....	53
1.9.1 Сценарий «Регистрация нового пользователя»	54
1.9.2 Сценарий «Авторизация в системе»	56
1.9.3 Сценарий «Просмотр расписания и запись на занятие»	57

1.9.4	Сценарий «Просмотр заявок в личном кабинете».....	58
1.9.5	Сценарий «Отправка обращения в поддержку»	59
1.9.6	Сценарий «Управление тренерами»	60
1.9.7	Сценарий «Управление расписанием»	62
1.9.8	Сценарий «Обработка заявок клиентов»	63
1.9.9	Сценарий «Просмотр услуг».....	64
1.9.10	Сценарий «Просмотр контактов».....	66
1.9.11	Сценарий «Управление услугами».....	66
1.9.12	Сценарий «Управление обращениями в поддержку»	68
2	Требования безопасности по охране труда при работе с ПК.	70
2.1	Организация рабочего места.....	70
2.2	Профилактика зрительного утомления и требования к освещению	71
2.3	Пожарная безопасность	71
2.4	Санитарно-гигиенические требования	72
	Заключение	74
	Список использованных источников	75
	Приложение А — UML диаграммы	76
	Приложение Б — Листинг программы	85
	Приложение В — Презентация.....	166

Перечень сокращений и обозначений

В данной работе приняты следующие сокращения и обозначения.

ОС (операционная система) — комплекс программ, обеспечивающих управление аппаратными ресурсами компьютера и взаимодействие с пользователем

ПК (персональный компьютер) — термин, используемый в разделе требований безопасности для обозначения персонального компьютера

СКУД (система контроля и управления доступом) — комплекс технических средств, используемый для ограничения и регистрации входа/выхода посетителей

API (Application Programming Interface) — программный интерфейс приложения, набор правил и инструментов для взаимодействия между клиентской и серверной частями системы

CRM (Customer Relationship Management) — система управления взаимоотношениями с клиентами, используемая для автоматизации взаимодействия с клиентами, учёта продаж и маркетинга

DOM (Document Object Model) — объектная модель документа, представляющая HTML-страницу в виде дерева объектов, доступных для управления через JavaScript

HTTP (HyperText Transfer Protocol) — протокол передачи гипертекста, используемый для обмена данными между клиентом и сервером в веб-приложениях

HTTPS (HyperText Transfer Protocol Secure) — расширение протокола HTTP, поддерживающее шифрование для безопасной передачи данных

IDE (Integrated Development Environment) — интегрированная среда разработки, программное обеспечение для написания, тестирования и отладки кода

JSX (JavaScript XML) — синтаксическое расширение JavaScript, позволяющее писать HTML-подобную разметку внутри кода React

					ПГУ1.090301.47.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

JWT (JSON Web Token) — компактный URL-безопасный способ представления утверждений (токенов) для передачи информации между сторонами в виде JSON-объекта

ORM (Object-Relational Mapping) — технология объектно-реляционного отображения, позволяющая работать с базой данных через объекты языка программирования

					ПГУ1.090301.47.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

В современном мире автоматизация процессов становится не просто преимуществом, а необходимостью для любой сферы деятельности, особенно когда речь идет о сфере услуг. Одной из таких сфер является фитнес-индустрия, которая в последние годы переживает активный рост. Количество фитнес-центров увеличивается, а вместе с ним растет и поток клиентов, желающих посещать тренировки, следить за своим здоровьем и поддерживать физическую форму. В таких условиях администрация фитнес-центров сталкивается с рядом организационных и управленческих проблем, связанных с ведением записи клиентов, составлением расписания занятий, контролем загруженности тренеров и учетом предоставляемых услуг. Традиционные способы организации работы, основанные на телефонных звонках, бумажных журналах или электронных таблицах, перестают быть эффективными при большом количестве клиентов и высоком темпе работы.

Проблема ручного сбора и оформления отчетности особенно остро проявляется в процессе управления расписанием и обработки заявок на занятия. Когда клиент хочет записаться на тренировку, он вынужден звонить в фитнес-центр, ожидать ответа администратора, уточнять наличие свободных мест и только после этого подтверждать свое намерение. Такой подход требует значительных временных затрат как со стороны клиента, так и со стороны сотрудников центра. Администратор, в свою очередь, тратит драгоценное время на прием звонков, заполнение журналов, проверку статуса записи и уведомление клиентов. Кроме того, при ручной обработке данных неизбежно возникают ошибки: двойная запись на одно и то же время, потеря информации о клиенте, неверно указанная дата или время занятия. Все это создает дополнительные сложности для всех участников процесса и негативно сказывается на репутации фитнес-центра.

Не менее важной является и задача анализа работы фитнес-центра. Руководству необходимо понимать, какие услуги пользуются наибольшей

					ПГУ1.090301.47.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		7

популярностью, какие тренеры наиболее востребованы, в какое время суток наблюдается максимальная загрузка, а когда зал простаивает. Без этих данных невозможно эффективно планировать расписание, нанимать дополнительный персонал или, наоборот, оптимизировать расходы. Традиционное решение — сбор информации вручную с последующим переносом в электронные таблицы — является крайне трудоемким и подверженным ошибкам. Кроме того, такой подход не позволяет получать оперативную информацию в реальном времени, что особенно важно для принятия управленческих решений.

Таким образом, назревает объективная необходимость в создании автоматизированной системы, которая позволила бы централизованно управлять расписанием, обрабатывать заявки клиентов в автоматическом режиме, контролировать загруженность тренеров и оборудования, а также формировать наглядную отчетность для руководства. Внедрение такого программного решения позволит фитнес-центру выйти на новый уровень обслуживания клиентов, повысить эффективность работы персонала и получить конкурентные преимущества на рынке фитнес-услуг.

Целью данной работы является проектирование и разработка современного клиент-серверного веб-приложения для автоматизации работы фитнес-центра. Создаваемая система должна обеспечить удобный и интуитивно понятный интерфейс для двух категорий пользователей: клиентов и администраторов. Для клиентов необходимо реализовать возможность просмотра актуального расписания занятий, получения информации об услугах и тренерах, самостоятельной записи на выбранные тренировки, а также отслеживания статуса своих заявок в личном кабинете. Для администраторов же требуется предоставить расширенный функционал: управление списком услуг, добавление и редактирование информации о тренерах, формирование расписания занятий на несколько дней вперед, обработка поступающих заявок на запись, администрирование обращений в

					ПГУ1.090301.47.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		8

службу поддержки и формирование отчетности по загруженности залов и популярности различных направлений.

Особое внимание при разработке уделяется вопросам надежности и безопасности. Система должна гарантировать целостность данных, предотвращать двойную запись на одно и то же занятие, обеспечивать разграничение доступа между обычными пользователями и администраторами, а также сохранять историю всех операций для последующего анализа. Кроме того, предполагается возможность масштабирования разработанного решения — при необходимости функционал может быть расширен за счет интеграции с системами онлайн-оплаты, смс-уведомлениями или мобильными приложениями.

Практическая значимость работы заключается в том, что разработанное веб-приложение позволит фитнес-центру значительно сократить временные затраты на обработку заявок клиентов, полностью исключить ошибки, связанные с человеческим фактором, предоставить клиентам возможность удобной и быстрой записи на занятия в любое время суток, а также обеспечить руководство центра достоверной и оперативной аналитической информацией для принятия стратегических решений. Внедрение подобной системы способствует повышению качества обслуживания, росту лояльности клиентов и увеличению эффективности работы фитнес-центра в целом.

					ПГУ1.090301.47.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		9

1 Веб–приложение фитнес центра

1.1 Анализ предметной области

Предметной областью данного исследования является процесс управления фитнес-центром, включающий в себя организацию и контроль всех аспектов деятельности: от ведения расписания занятий и управления тренерским составом до записи клиентов на тренировки и формирования отчетности. Фитнес-центр представляет собой сложную социально-техническую систему, в которой взаимодействуют три основные группы участников: администрация, представляющая управленческий персонал, тренерский состав, непосредственно проводящий занятия, и клиенты, то есть посетители, желающие получить услуги фитнес-центра. Каждая из этих групп имеет свои потребности, цели и задачи, которые должны быть учтены при автоматизации деятельности. Современный фитнес-центр — это не просто помещение с тренажерами, а полноценный сервисный бизнес, предоставляющий широкий спектр услуг: групповые тренировки, такие как йога, пилатес, аэробика, кросс-тренинг, функциональный тренинг, персональные занятия с тренером, а также различные оздоровительные программы. Успешное функционирование такого центра требует четкой координации множества процессов: составление и актуализация расписания, учет загруженности залов, контроль количества записавшихся на каждое занятие, управление информацией о тренерах и их специализации, обработка отмен и переносов, ведение базы клиентов и многое другое.

В процессе анализа предметной области были выделены следующие ключевые сущности. Клиент, то есть пользователь системы, представляет собой физическое лицо, желающее посещать занятия в фитнес-центре. Клиент обладает такими характеристиками, как уникальный идентификатор, которым выступает адрес электронной почты, контактные данные, включая имя и номер телефона, а также история посещений и записей. Клиент может

					ПГУ1.090301.47.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		10

быть как авторизованным пользователем системы, имеющим учетную запись, так и гостем, который впервые обращается в фитнес-центр. Администратор — это сотрудник фитнес-центра, наделенный правами на управление всей системой. Администратор отвечает за актуальность расписания, добавление и редактирование информации о тренерах и услугах, обработку заявок на запись, рассмотрение обращений в службу поддержки, формирование аналитических отчетов. Тренер — это специалист, проводящий занятия, важными характеристиками которого являются его квалификация, специализация, указывающая, какие виды тренировок он ведет, график работы, контактная информация и краткая биография для ознакомления клиентов. От квалификации и личных качеств тренера напрямую зависит популярность тех или иных занятий. Услуга представляет собой вид тренировки или программы, предлагаемой фитнес-центром, и характеризуется названием, кратким описанием, длительностью стандартного занятия, рекомендуемым количеством участников, необходимостью специального оборудования. Примерами услуг являются йога, пилатес, функциональная тренировка, кросс-тренинг, стретчинг, силовая тренировка, TRX и другие. Занятие, или тренировка, — это конкретный сеанс в расписании, привязанный к определенной дате и времени, месту проведения, такому как зал А, зал В или студия, закрепленный за конкретным тренером и относящийся к определенной услуге. Каждое занятие имеет лимит по количеству участников, определяемый техническими возможностями зала и требованиями безопасности, а также длительность в минутах. Заявка на запись — это документ, фиксирующий намерение клиента посетить конкретное занятие, содержащий информацию о клиенте, выбранном занятии, статусе обработки, который может быть ожидающим, одобренным, отклоненным или отмененным, дате и времени создания, а также дополнительных пожеланиях или комментариях клиента. Обращение в поддержку — это сообщение от

клиента или гостя, содержащее вопрос, предложение или жалобу, которое может быть отправлено как авторизованным пользователем, автоматически привязываясь к его учетной записи, так и гостем с указанием контактных данных для обратной связи.

Рассмотрим основные процессы, протекающие в предметной области. Процесс записи клиента на занятие в традиционном неавтоматизированном фитнес-центре выглядит следующим образом: клиент звонит администратору, узнает о наличии свободных мест, договаривается о времени, после чего администратор вручную вносит запись в журнал. Такой подход характеризуется низкой скоростью обработки, высокими временными затратами для обеих сторон и неизбежными ошибками, такими как запись одного клиента на одно время с другим, потеря информации или неправильно указанное время. Кроме того, клиент не имеет возможности узнать о статусе своей записи без дополнительного звонка, а администратор тратит до нескольких минут на каждую заявку, что при большом потоке звонков создает очереди и недовольство посетителей. В автоматизированной системе клиент самостоятельно просматривает расписание через веб-интерфейс, видит актуальную загруженность каждого занятия, выбирает подходящее время и отправляет заявку на запись одним нажатием кнопки. Система автоматически проверяет, не записан ли уже клиент на это занятие, не достигнут ли лимит участников, активно ли занятие, после чего создает заявку и направляет ее администратору на рассмотрение. Таким образом, автоматизация сокращает временные затраты на обработку одной заявки с нескольких минут до нескольких секунд, полностью исключает дублирование записи и предоставляет клиенту максимальную прозрачность процесса.

Процесс управления расписанием в традиционном подходе требует от администратора ручного составления расписания, например в электронной таблице, с последующим размещением на доске объявлений или в

социальных сетях. При любых изменениях, таких как отмена занятия, замена тренера или перенос времени, необходимо заново оповещать всех заинтересованных лиц и вносить правки во все источники информации, что крайне неудобно и приводит к путанице. В автоматизированной системе администратор через удобный веб-интерфейс создает новые занятия, указывая дату, время, длительность, локацию, выбирая тренера из списка и услугу из каталога. Система сама проверяет корректность введенных данных, например, не пересекается ли новое занятие по времени с уже существующими в том же зале, и мгновенно отражает все изменения в расписании, которое видят клиенты в режиме реального времени.

Процесс обработки заявок администратором включает просмотр списка всех поступивших заявок, сгруппированных по статусу и дате создания. По каждой заявке администратор может принять решение: одобрить, то есть подтвердить запись, отклонить, например, если клиент не подходит по уровню подготовки, или отменить, если клиент передумал или занятие отменяется. При одобрении заявки система автоматически проверяет, не превышен ли лимит участников с учетом уже одобренных заявок, и при необходимости может сама предложить альтернативные занятия. Информация о решении администратора автоматически становится видна клиенту в его личном кабинете. Процесс управления тренерами и услугами позволяет администратору добавлять новых тренеров, редактировать информацию о существующих, изменять статус активности и удалять тренеров, которые больше не работают в центре. Аналогичные операции доступны для услуг: добавление новой тренировки, редактирование описания, временное отключение или удаление. Важно, что удаление услуги или тренера не приводит к потере исторических данных, а записи о прошлых занятиях и заявках сохраняются. Процесс обратной связи и поддержки дает возможность клиентам и гостям обращаться в службу поддержки с любыми вопросами, причем авторизованные пользователи отправляют обращение с

					ПГУ1.090301.47.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		13

привязкой к своему аккаунту, а гости указывают свои контактные данные для обратной связи.

На основе анализа предметной области были сформулированы требования к разрабатываемому веб-приложению. Система должна обеспечивать регистрацию новых пользователей и аутентификацию существующих, разграничивать права доступа на две роли — клиент и администратор. Неавторизованные пользователи должны иметь возможность просматривать список услуг, информацию о тренерах и расписание занятий. Авторизованные клиенты должны иметь возможность записываться на занятия, просматривать и отслеживать статус своих заявок в личном кабинете, отправлять обращения в службу поддержки. Администратор должен иметь возможность управлять списком услуг, тренеров, создавать, редактировать и удалять занятия в расписании, просматривать все заявки клиентов и изменять их статус, а также просматривать обращения в поддержку и управлять их статусом. Система должна предотвращать двойную запись одного клиента на одно и то же занятие и запись на занятие при достижении лимита вместимости. К нефункциональным требованиям относятся работа в среде веб-браузера без установки дополнительного программного обеспечения, время отклика не более двух секунд, обеспечение целостности и сохранности данных, интуитивно понятный интерфейс, кроссплатформенность с адаптивным дизайном и безопасность передачи данных с хешированием паролей. Проведенный анализ показал, что разрабатываемое веб-приложение призвано решить выявленные проблемы за счет автоматизации ключевых процессов, позволяя фитнес-центру сократить временные затраты, повысить качество обслуживания клиентов и предоставить администрации инструменты для принятия обоснованных управленческих решений на основе актуальных данных.

1.2 Обзор аналогов

					ПГУ1.090301.47.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		14

Обзор аналогов является важной частью разработки информационной системы, так как позволяет изучить подобные решения и определить их преимущества и недостатки. В рамках данного исследования был проведен анализ рынка информационных систем для автоматизации фитнес-центров с целью выявления существующих аналогов и определения их сильных и слабых сторон. Следует отметить, что специализированных программных решений для малых и средних фитнес-центров на российском рынке представлено не так много, а многие существующие разработки ориентированы преимущественно на крупные сети клубов и имеют высокую стоимость внедрения. Тем не менее, можно выделить несколько категорий аналогов, каждая из которых имеет свои особенности.

Первым аналогом является облачная CRM-система «Fitbase». Данный сервис представлен на рисунке 1. Fitbase представляет собой многофункциональную платформу, предназначенную для автоматизации работы фитнес-клубов, спортивных центров и студий. Система объединяет учет клиентов, финансов и сотрудников в единой CRM-системе, предоставляя инструменты для управления заявками, контроля продаж и продлений, а также аналитики по клиентам, менеджерам и тренерам. Fitbase включает в себя онлайн-расписание и виджет записи для сайта и социальных сетей, автоматическое списание услуг при записи на занятия, лист ожидания, а также синхронизацию с системами контроля доступа и мобильными приложениями.

Сервис обеспечивает автоматическую рассылку уведомлений и акций через мессенджеры и push-уведомления, а также интеграцию с онлайн-кассами [1]. Fitbase ориентирован на коммерческие фитнес-центры с полным штатом сотрудников и большим потоком клиентов. Основными преимуществами являются широкая функциональность, автоматизация маркетинга и возможность интеграции с различными системами. К недостаткам можно отнести высокую стоимость, ежемесячную плату за

использование, сложность настройки и избыточность функционала для небольших студий или частных тренеров, а также привязанность к облачной инфраструктуре без возможности развертывания на собственном оборудовании.

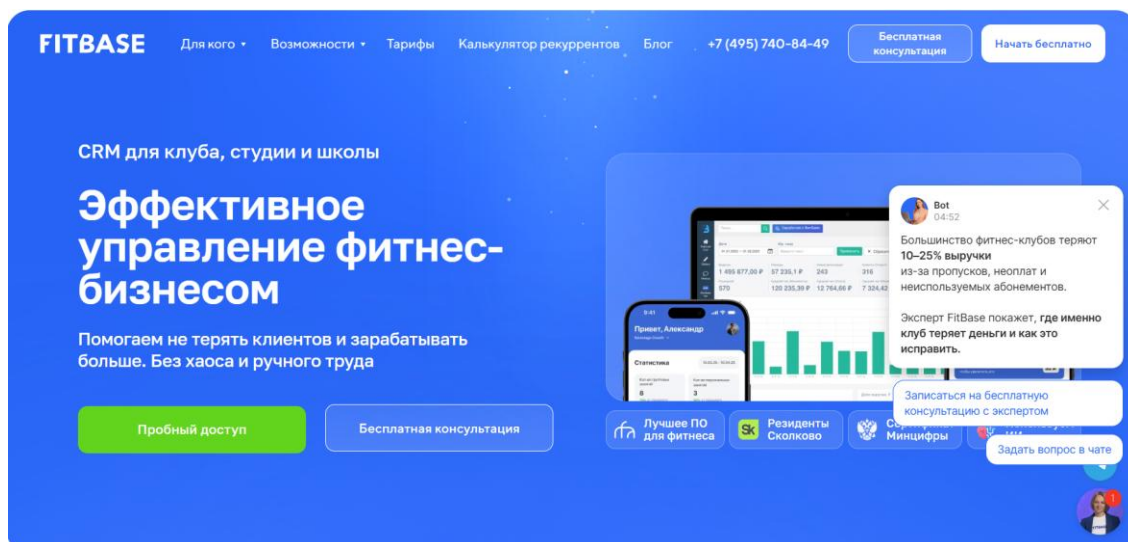


Рисунок 1 — «Fitbase»

Вторым аналогом является сеть фитнес-клубов «XFIT». Данный сервис представлен на рисунке 2. XFIT представляет собой бренд спортивных клубов премиального класса, предлагающий широкий спектр услуг: тренажерные залы, плавательные бассейны, теннисные корты, групповые программы и СПА-салоны. Особенностью является наличие автоматизированных клубов нового формата XFIT Point, в которых отсутствуют администраторы и очереди, а доступ к услугам осуществляется через современные технологические решения. XFIT предлагает различные виды клубных карт с разными сроками действия и набором подарков, включая старт-тест и гостевой визит. В клубах работают профессиональные тренеры, о чем свидетельствуют многочисленные положительные отзывы клиентов. Основным преимуществом является высокий уровень сервиса и развитая инфраструктура премиального фитнес-клуба. Однако данное решение ориентировано исключительно на клиентов собственной сети XFIT

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.47.001 ПЗ

Лист

16

и недоступно для сторонних фитнес-центров [2]. Отсутствует функционал управления расписанием, записи и учета посещений для независимых организаций, нет возможности формирования отчетности для администрации. XFIT является продуктом конкретной фитнес-сети, а не программным решением для автоматизации, что делает его неприменимым для использования другими фитнес-центрами.

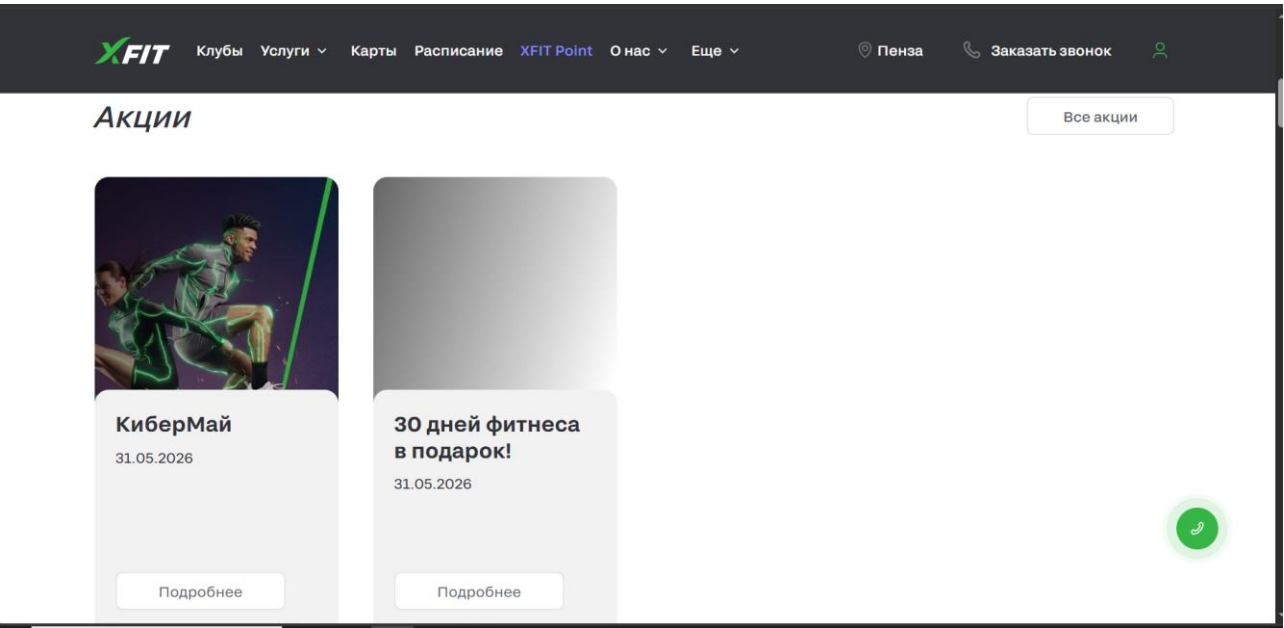


Рисунок 2 — «XFIT»

Третьим аналогом является мобильное приложение фитнес-сети «World Class». Данный сервис представлен на рисунке 3. Приложение разработано крупнейшей российской фитнес-сетью для собственных клиентов и предоставляет возможность просматривать расписание клубов, записываться на групповые и персональные тренировки, отслеживать историю посещений, получать персональные рекомендации от тренеров, а также оплачивать услуги и абонементы. Приложение ориентировано на клиентов, уже являющихся членами сети World Class, и не предназначено для использования независимыми фитнес-центрами. Основными преимуществами являются высокое качество пользовательского интерфейса, интеграция с фитнес-трекерами и системами лояльности, а также наличие

пуш-уведомлений о предстоящих тренировках [3]. К недостаткам можно отнести невозможность использования для управления другими фитнес-центрами, привязанность к конкретной сети, а также необходимость установки мобильного приложения на смартфон при отсутствии веб-версии для записи через компьютер.

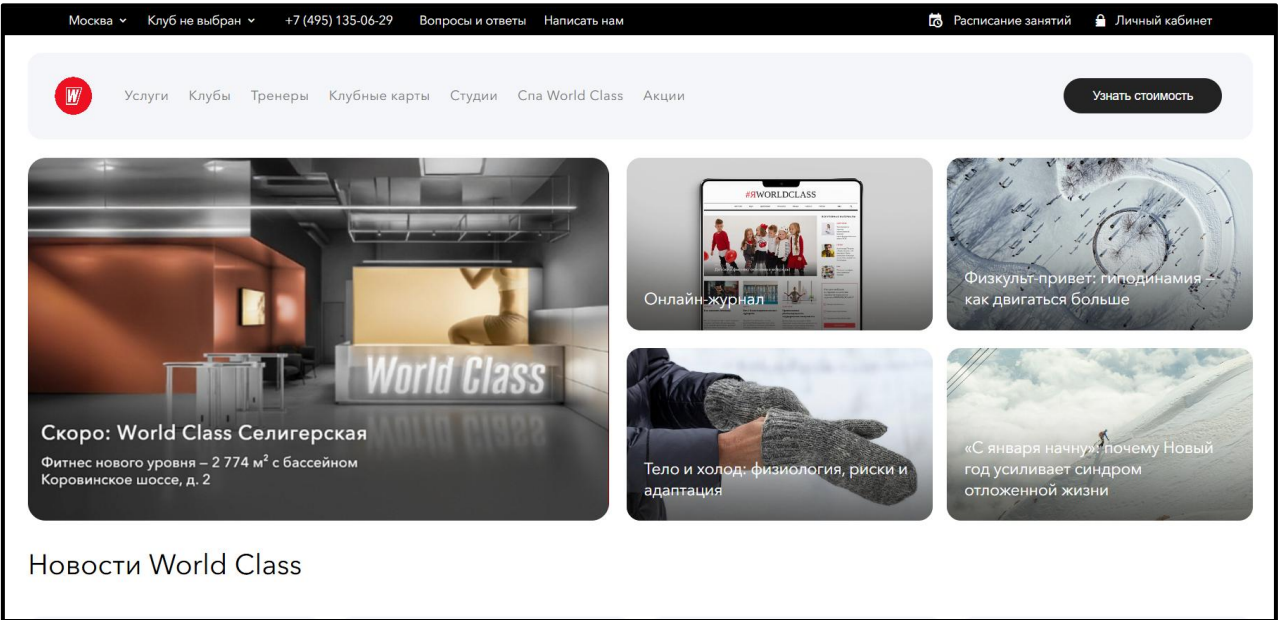


Рисунок 3 — «World Class»

Сравнение преимуществ и недостатков рассмотренных аналогов представлено в таблице 1.

Таблица 1 — Сравнительная таблица аналогов

Аналог	Преимущества	Недостатки
--------	--------------	------------

Fitbase (https://fitbase.io/)	Комплексная CRM-система, объединяющая учет клиентов, финансов и сотрудников в единой платформе. Онлайн-расписание и запись для клиентов и тренеров. Интеграция с системами контроля доступа (СКУД), онлайн-кассами и платежными сервисами	Высокая стоимость, ориентированность на коммерческие фитнес-центры с полным штатом сотрудников. Ежемесячная плата за использование
Аналог	Преимущества	Недостатки
XFIT (https://www.xfit.ru/)	Широкая сеть фитнес-клубов премиального класса с разнообразной инфраструктурой. Наличие автоматизированных клубов нового формата XFIT Point без администраторов и очередей.	Отсутствует функционал управления расписанием, записью и учетом посещений для независимых организаций

World Class (https://worldclass.ru/)	Высокое качество пользовательского интерфейса, интеграция с фитнес-трекерами и системами лояльности, наличие пуш – уведомлений о предстоящих тренировках, персональные рекомендации от тренеров, удобная запись на групповые и персональные занятия	Приложение разработано исключительно для клиентов сети World Class и недоступно для сторонних фитнес-центров
--	---	--

На основании проведенного анализа был сделан вывод, что на рынке веб приложений фитнес – центра имеется потенциал для разработки новых решений.

1.3 Анализ требований к разработке

В рамках проекта разрабатывается клиент-серверное веб-приложение для автоматизации работы фитнес-центра, предназначенное для сбора, обработки и анализа данных о записях клиентов, управлении расписанием и контроля загруженности тренеров и оборудования. Программа ориентирована на малые и средние фитнес-центры, которые нуждаются в автоматизации процесса записи клиентов, ведения расписания и формирования отчетности, что критически важно для эффективного управления деятельностью фитнес-центра и повышения качества обслуживания посетителей. Разрабатываемая система призвана заменить устаревшие ручные методы ведения учета, основанные на телефонных звонках, бумажных журналах или электронных таблицах, которые

характеризуются низкой скоростью обработки заявок, высокими временными затратами для сотрудников и неизбежными ошибками при внесении данных.

Помимо функциональных требований, в связи с тем, что разрабатываемое приложение имеет клиент-серверную архитектуру, целесообразно разделить описание требований на клиентскую и серверную части. Клиентская часть представляет собой веб-интерфейс, работающий в браузере пользователя, и отвечает за отображение информации, взаимодействие с пользователем и отправку запросов на сервер. Серверная часть обрабатывает запросы от клиента, взаимодействует с базой данных, реализует бизнес-логику приложения и возвращает обработанные данные клиенту.

Для клиентской части разрабатываемой системы выдвигаются следующие требования:

- интуитивно понятный интерфейс, не требующий специального обучения для конечных пользователей, с единым стилем оформления всех страниц и адаптивной версткой для корректного отображения на различных устройствах;

- главная страница с приветствием и краткой информацией о фитнес-центре, а также страница контактов с информацией о местонахождении и способах связи;

- страница услуг с перечнем всех доступных тренировок и их описанием;

- страница расписания с отображением всех занятий на ближайшие дни, где авторизованный клиент может записаться на выбранную тренировку;

- страница входа и регистрации для аутентификации пользователей;

- личный кабинет клиента с отображением его заявок и статусов их обработки в табличном виде;

					ПГУ1.090301.47.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		21

– административная панель с возможностью переключения между вкладками для управления тренерами, услугами, расписанием, заявками и обращениями в поддержку, включающая формы для добавления данных и таблицы для просмотра существующих записей;

– страница поддержки с формой для отправки обращений, адаптированная под авторизованного пользователя и гостя.

Для серверной части разрабатываемой системы выдвигаются следующие требования:

– обработка входящих запросов от клиентской части, реализация бизнес-логики приложения и взаимодействие с базой данных;

– предоставление взаимодействий для регистрации и аутентификации пользователей с проверкой уникальности email и хешированием паролей;

– предоставление публичных эндпоинтов для получения списка услуг, тренеров и расписания занятий;

– предоставление точек входа для авторизованных пользователей: создание заявок на запись с автоматической проверкой наличия свободных мест и отсутствия дублирования, получение списка своих заявок, отправка обращений в поддержку;

– предоставление административных взаимодействий для управления услугами, тренерами, расписанием, а также для обработки заявок клиентов и управления обращениями в поддержку;

– проверка прав доступа при каждом запросе: запрет доступа к защищенным эндпоинтам для неавторизованных пользователей и запрет административных действий для обычных клиентов;

– валидация всех входящих данных для предотвращения некорректных или вредоносных запросов;

– автоматическая проверка при создании заявки: не записан ли уже клиент на это занятие, не достигнут ли лимит участников, активно ли занятие;

					ПГУ1.090301.47.001 ПЗ	Лист
						22
Изм.	Лист	№ докум.	Подпись	Дата		

– повторная проверка наличия свободных мест при одобрении заявки администратором.

Для базы данных разрабатываемой системы выдвигаются следующие требования:

– хранение информации о пользователях: идентификатор, электронная почта, хеш пароля, имя, телефон, роль, дата регистрации;

– хранение информации о тренерах: идентификатор, имя, биография, статус активности;

– хранение информации об услугах: идентификатор, название, описание, статус активности;

– хранение информации о занятиях: идентификатор, дата и время начала, длительность, вместимость, локация, статус активности, идентификатор тренера, идентификатор услуги;

– хранение информации о заявках на запись: идентификатор, статус, заметки, дата создания, идентификатор пользователя, идентификатор занятия;

– хранение информации об обращениях в поддержку: идентификатор, тема, текст, статус, дата создания, идентификатор пользователя, а также опциональные поля для гостей: электронная почта, имя, телефон;

– обеспечение целостности ссылок с помощью внешних ключей.

Исходя из целевого назначения проекта, были сформированы диаграммы прецедентов для неавторизованного, авторизованного пользователей и администратора, изображённые на рисунках 4 – 6.



Рисунок 4 — Диаграмма прецедентов неавторизованного пользователя



Рисунок 5 — Диаграмма прецедентов авторизованного пользователя

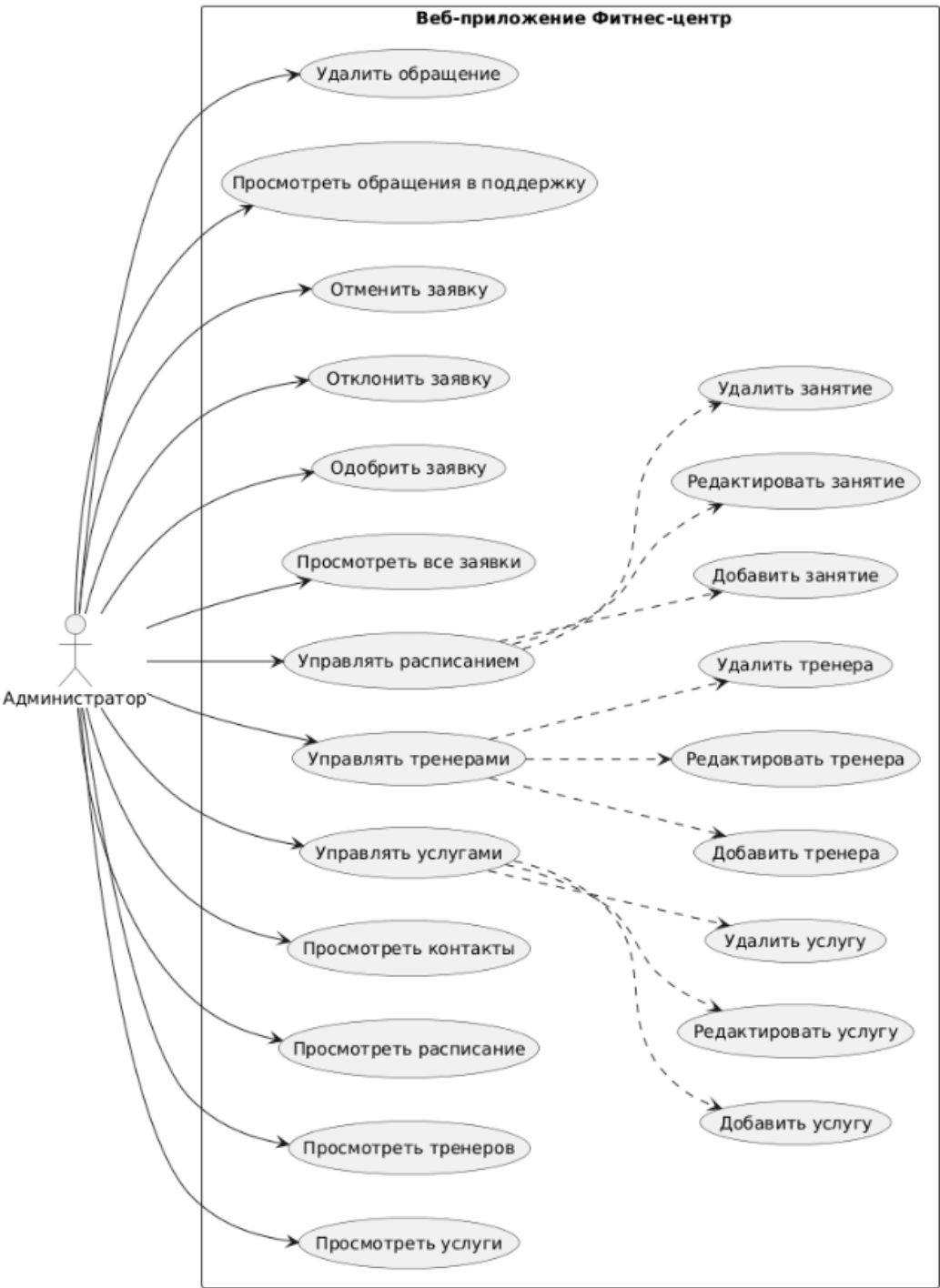


Рисунок 6 — Диаграмма прецедентов администратора

1.4 Выбор архитектуры системы

Для организации корректного и отказоустойчивого взаимодействия между пользователем, клиентской частью и серверной частью, работа

приложения построена на архитектуре клиент-сервер с асинхронными операциями. Основной рабочий цикл программы можно разбить на следующие этапы:

- ожидание и обработка пользовательского взаимодействия с элементами графического интерфейса веб-приложения. Пользователь нажимает на кнопки, переходит по ссылкам или заполняет формы, что генерирует событие, которое перехватывается соответствующим обработчиком на стороне клиента;

- в обработчике события инициируется асинхронный HTTP-запрос к серверной части приложения. Это позволяет не блокировать поток пользовательского интерфейса на время выполнения длительных операций, таких как авторизация, загрузка расписания или отправка заявки на запись;

- серверная часть принимает запрос, выполняет валидацию входящих данных, проверяет права доступа пользователя и инициирует соответствующую операцию с базой данных. При необходимости сервер обращается к системе управления базами данных для чтения или записи информации о пользователях, тренерах, услугах, занятиях, заявках и обращениях в поддержку;

- извлеченные или сохраненные данные преобразуются в формат JSON и возвращаются клиентской части в качестве ответа на HTTP-запрос. Структурированные данные содержат всю необходимую информацию для обновления пользовательского интерфейса: список услуг, расписание занятий, статусы заявок, информацию о тренерах;

- клиентская часть получает ответ от сервера, обрабатывает полученные данные и обновляет соответствующие компоненты интерфейса. Например, после успешной авторизации пользователь перенаправляется в личный кабинет, после записи на занятие обновляется таблица заявок, а при загрузке расписания отображается актуальный список занятий с количеством свободных мест;

- после завершения всех операций пользователь видит обновленную информацию на экране, а при необходимости отображаются всплывающие уведомления, информирующие о результате выполненного действия — успешной записи, ошибке валидации или проблемах с соединением;
- приложение вновь переходит в режим ожидания следующего пользовательского события, готовая к выполнению нового цикла. Время выполнения цикла зависит от сложности операции, скорости интернет-соединения, загруженности сервера и производительности базы данных.

Функциональная схема выполнения программы показана на рисунке 7.

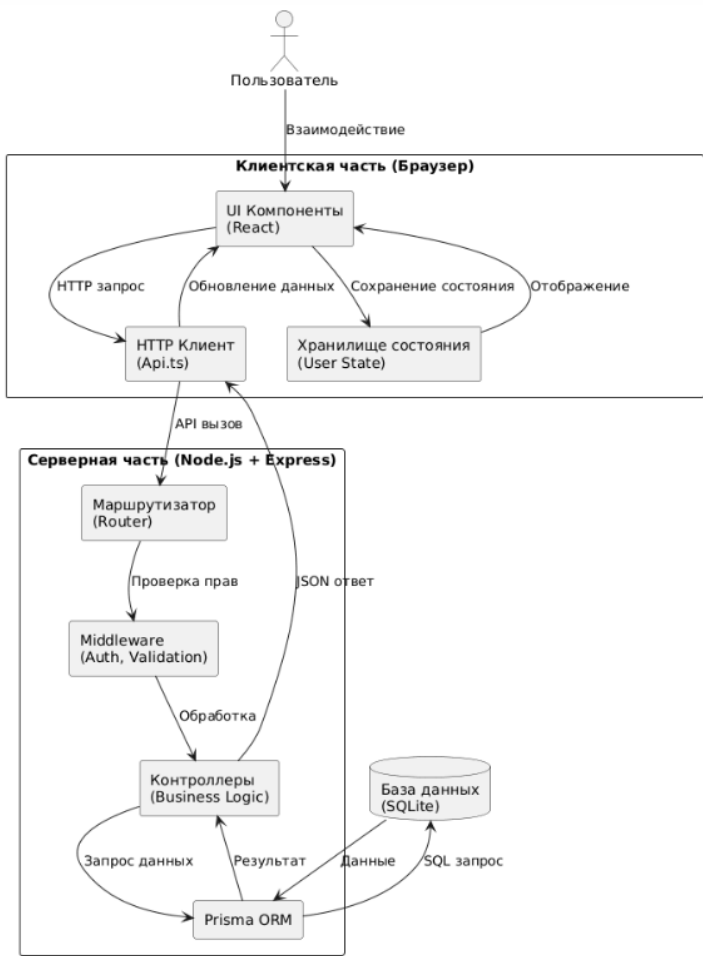


Рисунок 7 — Функциональная схема проекта

Детально процесс авторизации и последующего сбора данных показан на диаграммах последовательностей (рисунки 8 и 9).

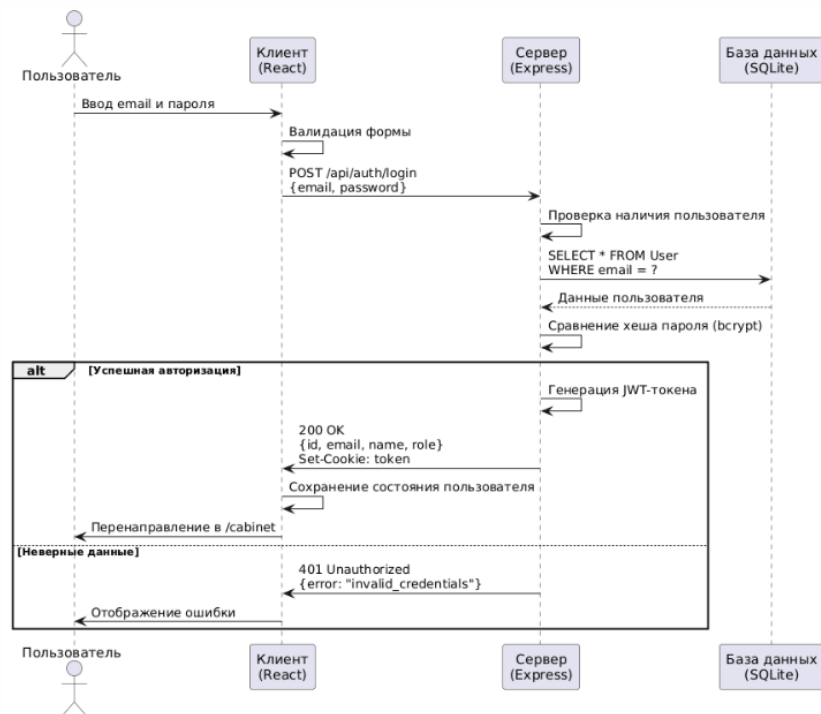


Рисунок 8 — Диаграмма последовательностей «Авторизация»

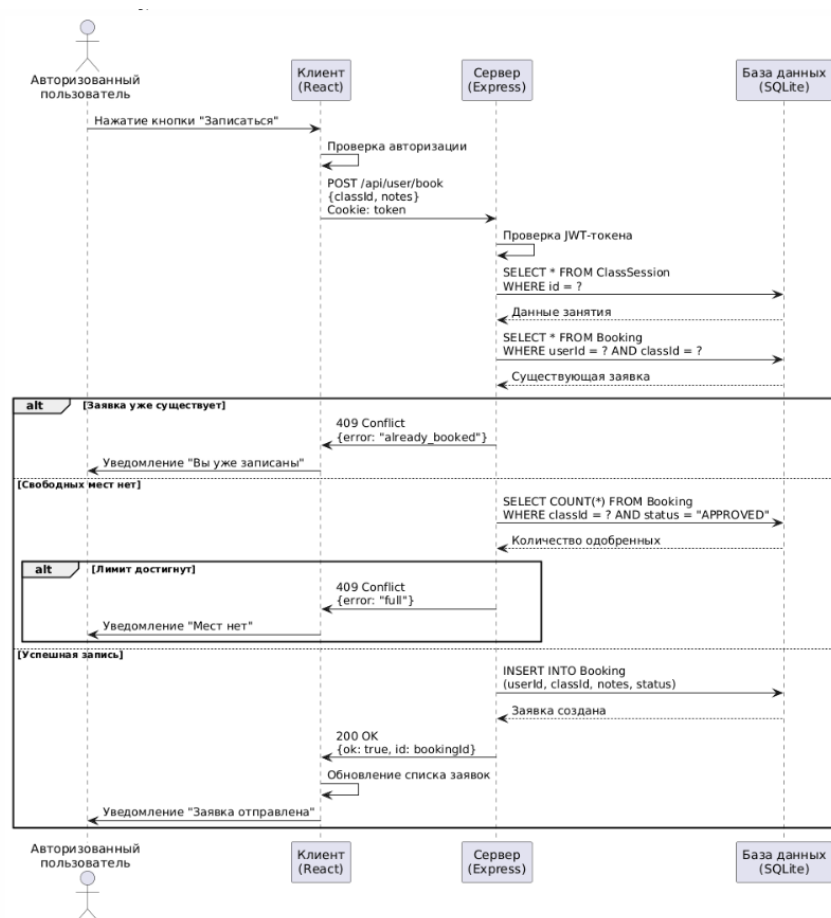


Рисунок 9 — Диаграмма последовательностей «Запись на занятие»

Развертывание приложения предполагает его установку на серверное оборудование фитнес-центра или использование облачного хостинга. Для корректной работы необходим сервер с установленной платформой Node.js и системой управления базами данных (SQLite не требует отдельного сервера). Клиентская часть представляет собой статические файлы, которые могут быть развернуты на любом веб-сервере или отдаваться тем же Node.js-сервером. Пользователи получают доступ к приложению через любой современный веб-браузер (Google Chrome, Mozilla Firefox, Safari, Edge) по протоколу HTTPS. Приложение взаимодействует с сервером по защищенному протоколу, а все данные сохраняются в базе данных на серверной стороне. Архитектура развертывания системы отражена на рисунке 10.

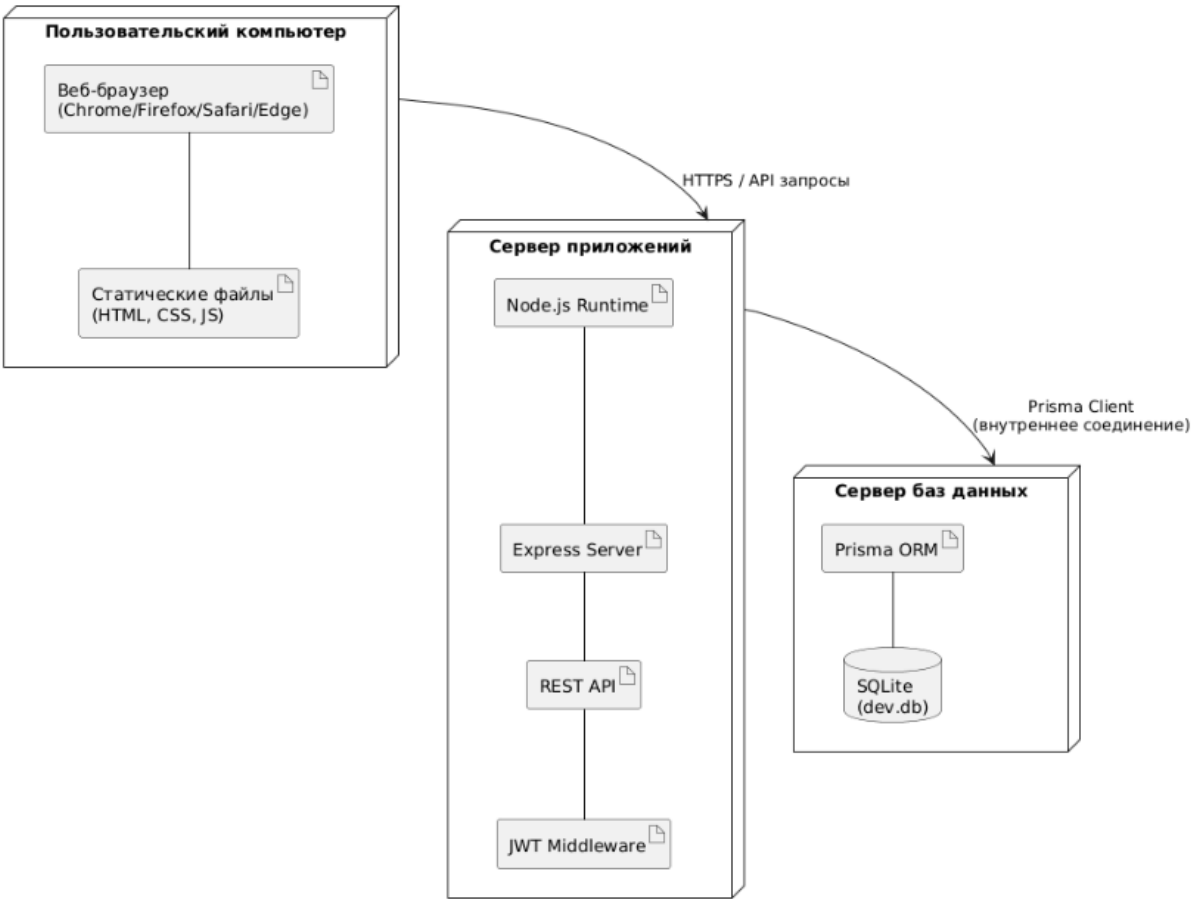


Рисунок 10 — Диаграмма развертывания приложения

Исходя из требуемого функционала, планируемой архитектуры проекта и выбранной целевой платформы, был определен следующий набор инструментов разработки: для клиентской части — язык программирования TypeScript, библиотека React, сборщик Vite, среда разработки Visual Studio Code; для серверной части — язык программирования TypeScript, платформа Node.js, фреймворк Express, ORM Prisma, база данных SQLite; для системы контроля версий — Git; для управления зависимостями — npm. Операционная система разработки не имеет принципиального значения, так как все используемые инструменты являются кроссплатформенными и могут работать как на Windows, так и на macOS или Linux.

1.5 Выбор языка программирования

TypeScript — статически типизированный язык программирования, разработанный компанией Microsoft как надстройка над JavaScript. Он предназначен для разработки крупных и сложных веб-приложений, включая клиентскую и серверную части, благодаря возможности выявлять ошибки на этапе компиляции. TypeScript добавляет в JavaScript такие возможности, как статическая типизация, классы, интерфейсы, дженерики, перечисления и модули [4]. Программа на TypeScript представляет собой набор классов, функций и интерфейсов, которые транспонируются в чистый JavaScript для последующего выполнения в браузере или на платформе Node.js. Классы могут наследоваться от базовых классов и реализовывать интерфейсы, что позволяет создавать гибкую и расширяемую архитектуру, удобную для поддержки и масштабирования проекта [4].

При программировании используются встроенные типы и наборы библиотечных модулей из экосистемы Node.js и React, а также собственные классы и компоненты, написанные на TypeScript. Класс — это шаблон для создания объектов, объединяющий данные и поведение. Встроенные библиотеки предоставляют широкий набор готовых компонентов: классы для

					ПГУ1.090301.47.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		30

работы с HTTP-запросами, файловой системой, базами данных через ORM, коллекциями и датами. В клиентской части React-компоненты представляют собой функции или классы, возвращающие JSX-разметку для отображения пользовательского интерфейса. В серверной части Express-маршруты обрабатывают входящие HTTP-запросы и возвращают соответствующие ответы.

Каждый отдельный метод представляет собой последовательность инструкций, выполняемых строго в порядке их написания. Методы могут вызывать другие методы, создавая иерархию вызовов. Внутри метода инструкции выполняются последовательно сверху вниз, за исключением случаев использования управляющих конструкций, условные операторы, циклы, обработка исключений. Результат выполнения метода возвращается в вызывающий код либо сохраняется в полях класса для дальнейшего использования. Асинхронные методы, помеченные ключевым словом `async`, позволяют выполнять длительные операции без блокировки основного потока выполнения [4].

Ключевой особенностью TypeScript является наличие строгой системы типов, позволяющей выявлять ошибки на этапе разработки до выполнения программы. В сочетании с событийно-ориентированной моделью React, где действия пользователя генерируют события, которые обрабатываются специальными функциями-обработчиками, это позволяет создавать интерактивные веб-приложения, реагирующие на действия пользователя в реальном времени без перезагрузки страницы [4].

1.6 Выбор среды разработки

В качестве среды разработки, исходя из разнородности целевых платформ проекта, клиентская часть выполняется в браузере, серверная — на платформе Node.js, была выбрана среда разработки Visual Studio Code. Данный выбор обусловлен тем, что VS Code идеально подходит для

разработки на TypeScript, поддерживает работу с React, Node.js и Express, а также предоставляет богатую экосистему расширений, универсальных как для клиентской, так и для серверной разработки.

Visual Studio Code — это легковесная интегрированная среда разработки (IDE) от компании Microsoft, предназначенная для разработки современных веб-приложений на различных языках программирования. VS Code работает на всех основных операционных системах: Windows, macOS и Linux, что обеспечивает кроссплатформенность разработки и не привязывает проект к конкретной ОС [5].

Возможности Visual Studio Code, используемые в проекте:

- встроенная поддержка TypeScript обеспечивает подсветку синтаксиса, автодополнение кода IntelliSense, проверку типов в реальном времени и быструю навигацию по проекту. Это единое окружение работает как для клиентского кода, так и для серверного кода, что существенно ускоряет разработку и позволяет выявлять ошибки на ранних этапах [5];

- редактор кода с подсветкой синтаксиса обеспечивает удобное написание программного кода на TypeScript и TSX с автоматическим дополнением, подсказками по параметрам методов и функций, а также возможностью быстрого рефакторинга (переименование символов, выделение в отдельную функцию, импорт модулей) [5];

- встроенный терминал позволяет выполнять команды npm, запускать скрипты сборки, управлять зависимостями, запускать сервер разработки для клиента и сервера без переключения на отдельное окно терминала. Возможность открытия нескольких вкладок терминала позволяет одновременно запускать и клиент, и сервер в рамках одного окна VS Code[5];

- отладчик позволяет выполнять программу по шагам, устанавливать точки останова как в клиентском коде, так и в серверном коде Node.js, отслеживать значения переменных в реальном времени, что необходимо для поиска и устранения ошибок в обеих частях приложения;

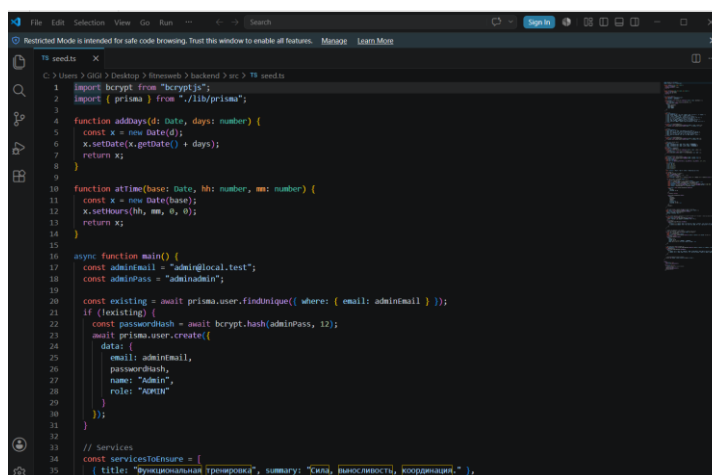
– менеджер пакетов npm интегрирован через встроенный терминал и позволяет легко подключать сторонние библиотеки, такие как React, Express, Prisma, jsonwebtoken, bcryptjs, Vite, и другие, без необходимости ручного скачивания и настройки;

– расширения предоставляют возможность добавлять дополнительный функционал, улучшающий процесс разработки клиентской и серверной частей. Ключевые расширения для проекта: Prisma – подсветка синтаксиса для schema.prisma и автодополнение запросов, ESLint – статический анализ кода для выявления потенциальных ошибок, Prettier – автоматическое форматирование кода для единообразия стиля, Thunder Client – тестирование API-эндпоинтов непосредственно из VS Code;

– встроенная система контроля версий Git позволяет отслеживать изменения кода, создавать коммиты, ветки и синхронизироваться с удаленными репозиториями без выхода из среды разработки. Это удобно для ведения проекта и командной работы;

– интеграция с терминалом и задачи позволяет автоматизировать рутинные операции: запуск клиента и сервера одной командой, выполнение миграций базы данных, запуск seed-файлов, сборку проекта для продакшена.

Пример фрагмента программы в среде Visual Studio Code представлен на рисунке 11:



```
1 import bcrypt from "bcryptjs";
2 import { prisma } from "../lib/prisma";
3
4 function addDays(d: Date, days: number) {
5   const x = new Date(d);
6   x.setDate(x.getDate() + days);
7   return x;
8 }
9
10 function aTime(base: Date, hh: number, mm: number) {
11   const x = new Date(base);
12   x.setHours(hh, mm, 0, 0);
13   return x;
14 }
15
16 async function main() {
17   const adminmail = "adminlocaltest";
18   const adminPass = "adminadmin";
19
20   const existing = await prisma.user.findUnique({ where: { email: adminmail } });
21   if (existing) {
22     const passwordHash = await bcrypt.hash(adminPass, 12);
23     await prisma.user.create({
24       data: {
25         email: adminmail,
26         passwordHash,
27         name: "Admin",
28         role: "Admin"
29       }
30     });
31   }
32 }
33
34 // Services
35 const servicesToInsure = [
36   { title: "Функциональный прототип", summary: "Свод, известность, координация." },
37   { title: "Исследование требований", summary: "Исследование требований." }
38 ]
```

Рисунок 11 — пример фрагмента алгоритма

Visual Studio Code довольно прост в освоении для начинающих благодаря минималистичному и понятному интерфейсу, а также огромному количеству обучающих материалов и встроенной подсказке по командам. Подключение нужных расширений позволяет быстро настраивать среду под свой язык и сразу приступать к созданию приложений. Инструменты отладки, точки остановки, просмотр переменных, консоль отладки помогают эффективно искать и исправлять ошибки. Редактор наглядно показывает связь между файлами проекта и кодом, обрабатывающим пользовательские действия через систему импортов и подсветку ссылок. При условии грамотно спроектированной архитектуры приложения с чётким разделением логики и интерфейса, разработка в VS Code становится гораздо более продуктивной, чем использование обычных текстовых блокнотов. Данная среда программирования абсолютно бесплатна. Сайт Visual Studio Code представлен на рисунке 12.

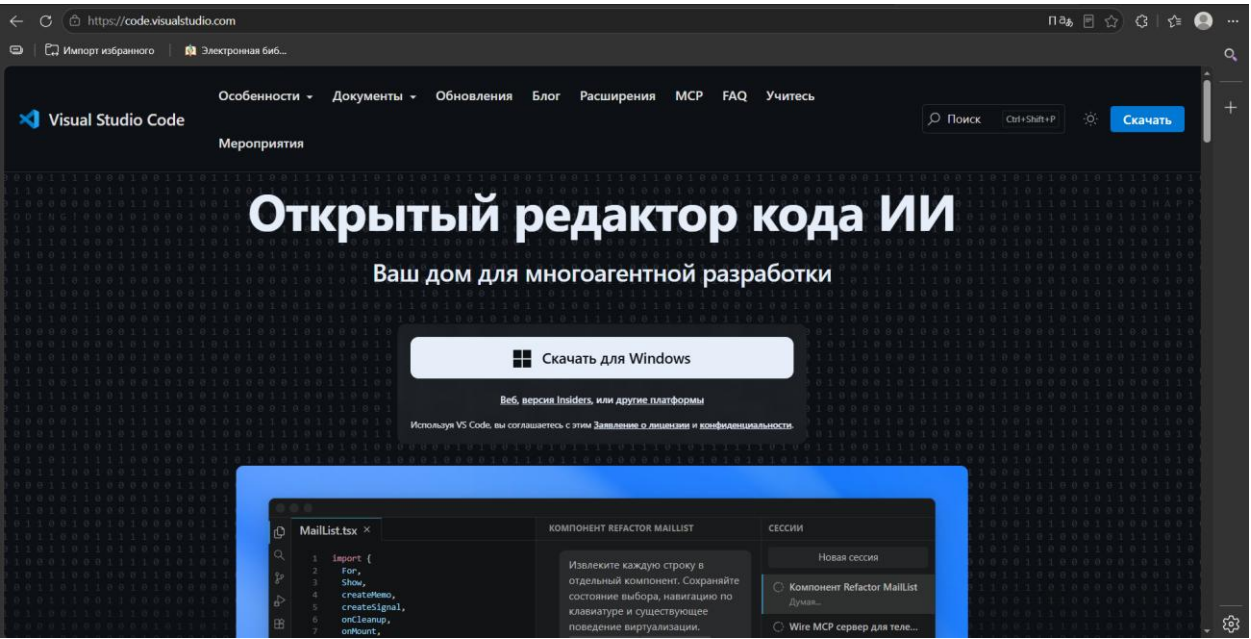


Рисунок 12 — Сайт Visual Studio

Процесс установки и настройки выполнялся в следующей последовательности.

На первом этапе необходимо загрузить установочный файл Visual Studio Code с официального сайта. На главной странице ресурса расположена область загрузки, соответствующая операционной системе пользователя. Данный этап проиллюстрирован на рисунке 10.

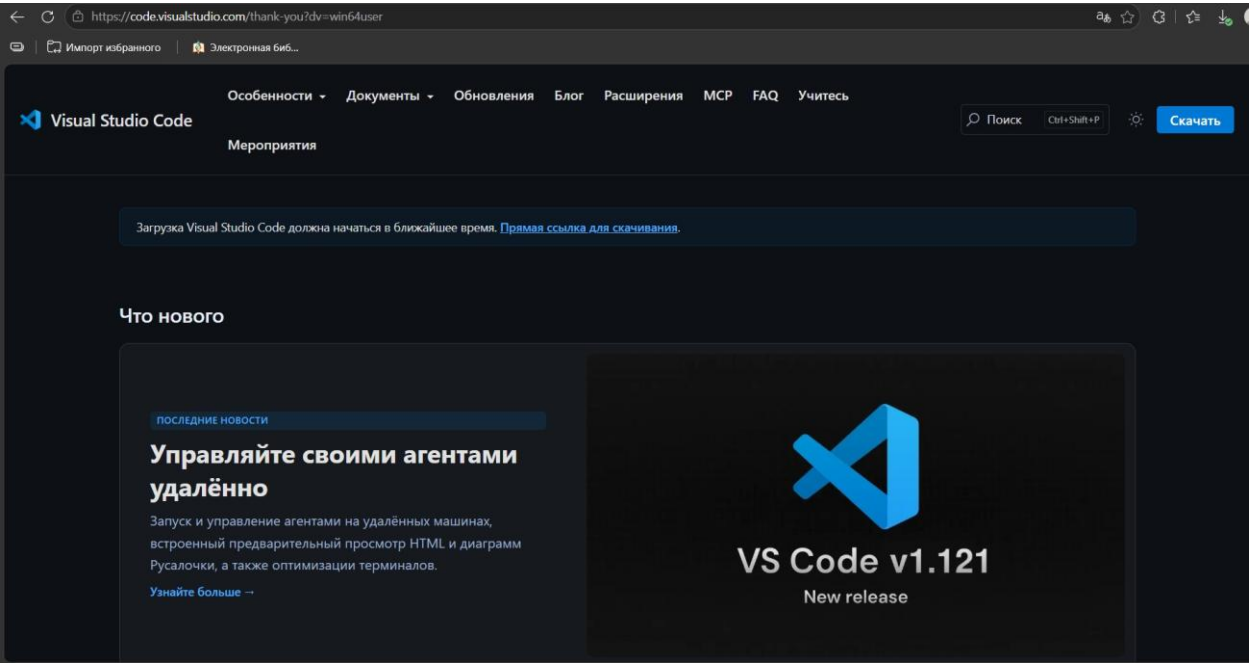


Рисунок 10 — Область загрузки Visual Studio Code с официального сайта

После нажатия кнопки загрузки запускается получение исполняемого файла установщика. Ход загрузки отображается в интерфейсе браузера. По завершении загрузки получен файл VSCodeSetup.exe для ОС Windows.

Далее был запущен установочный файл. На начальном этапе установки открывается окно с лицензионным соглашением, после принятия которого предлагается выбрать путь установки и компоненты. Для работы проекта были оставлены параметры по умолчанию. Внешний вид окна установки показан на рисунке 11.

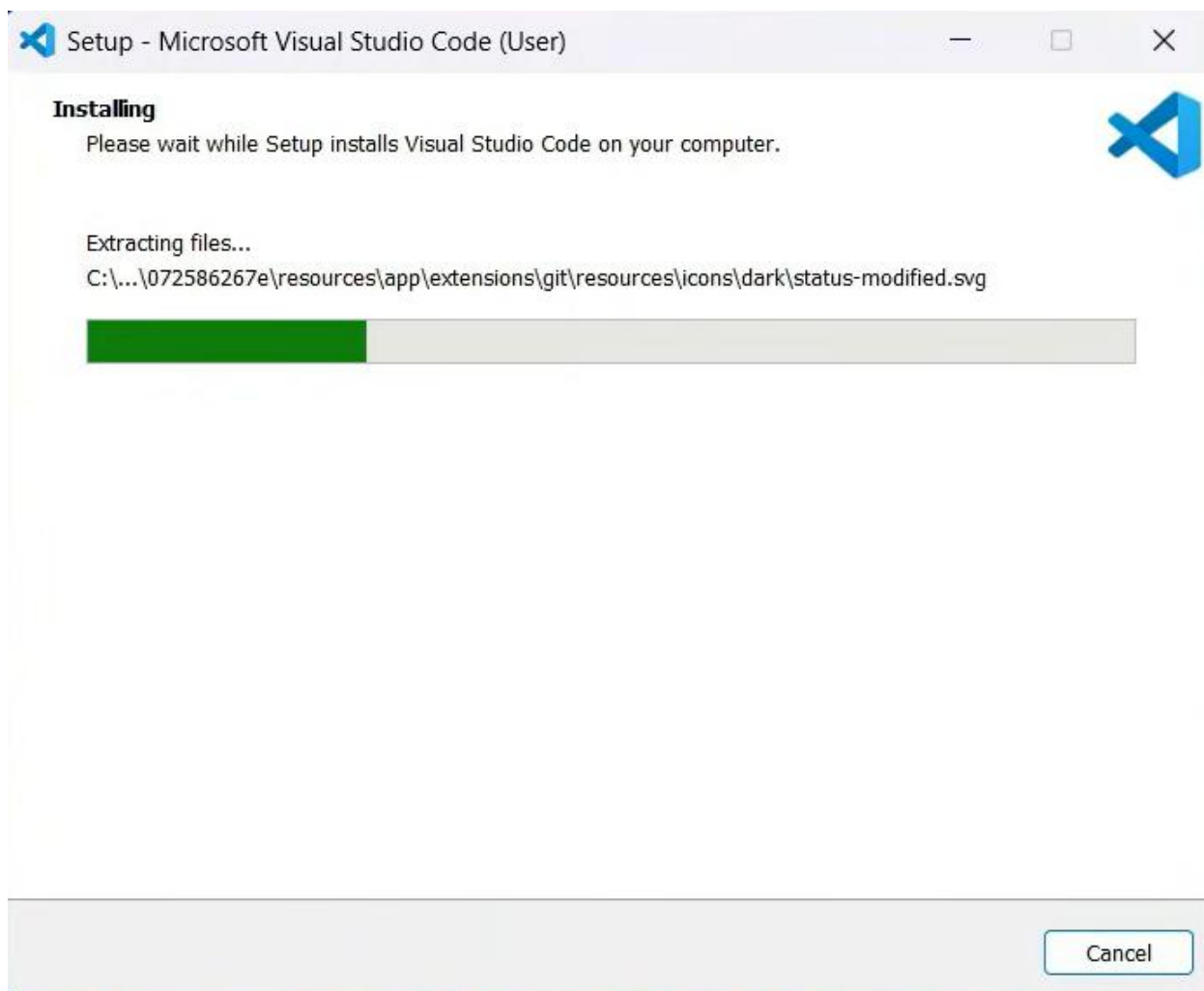


Рисунок 11 — Процесс установки Visual Studio Code

После завершения инсталляции среда становится доступной для запуска через главное меню или рабочий стол.

Затем был выполнен запуск Visual Studio Code. При первом открытии интерфейс имеет минималистичный вид, что удобно для дальнейшей настройки под нужды проекта. Главное окно VS Code изображено на рисунке 12.

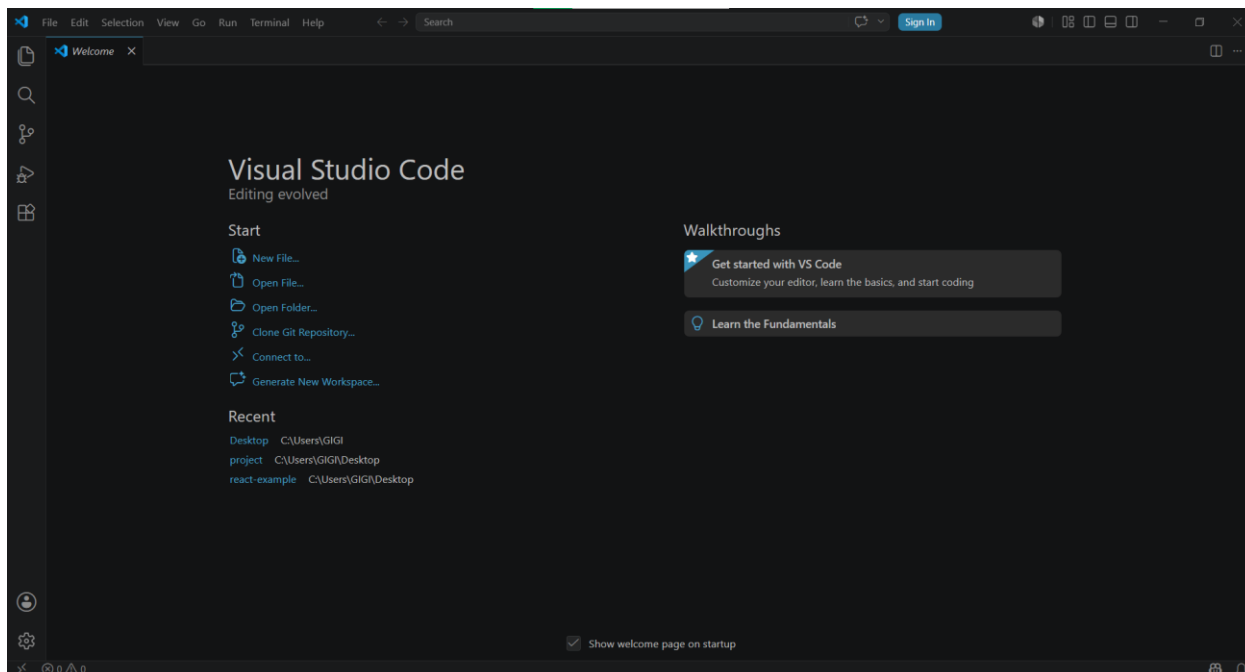


Рисунок 12 — Главное окно Visual Studio Code после установки

Для выполнения серверной части дипломного проекта требуется среда выполнения Node.js. Загрузка дистрибутива Node.js также выполнена с официального сайта. Выбрана версия с длительной поддержкой LTS. На рисунке 13 представлено окно выбора версии для загрузки.

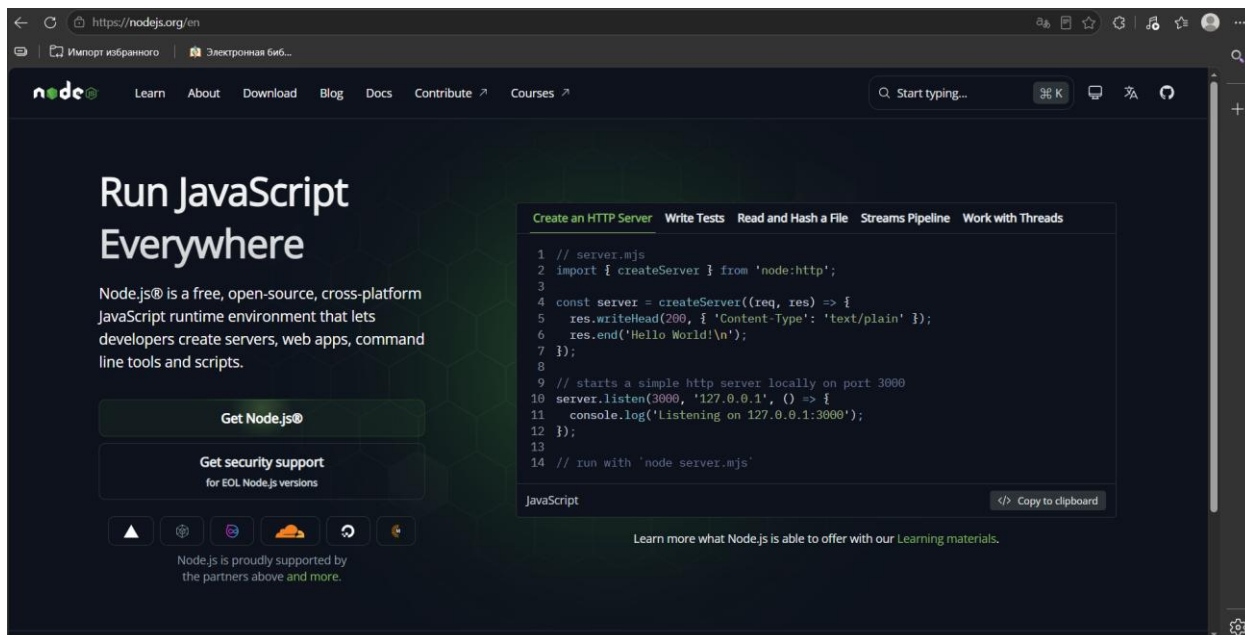


Рисунок 13 — Выбор версии Node.js для загрузки

					ПГУ1.090301.47.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		37

Установка Node.js производилась стандартным установщиком. В процессе установки автоматически добавляются в системную переменную PATH команды node и npm, что позволяет использовать их из любого терминала. Процесс установки Node.js показан на рисунке 14.

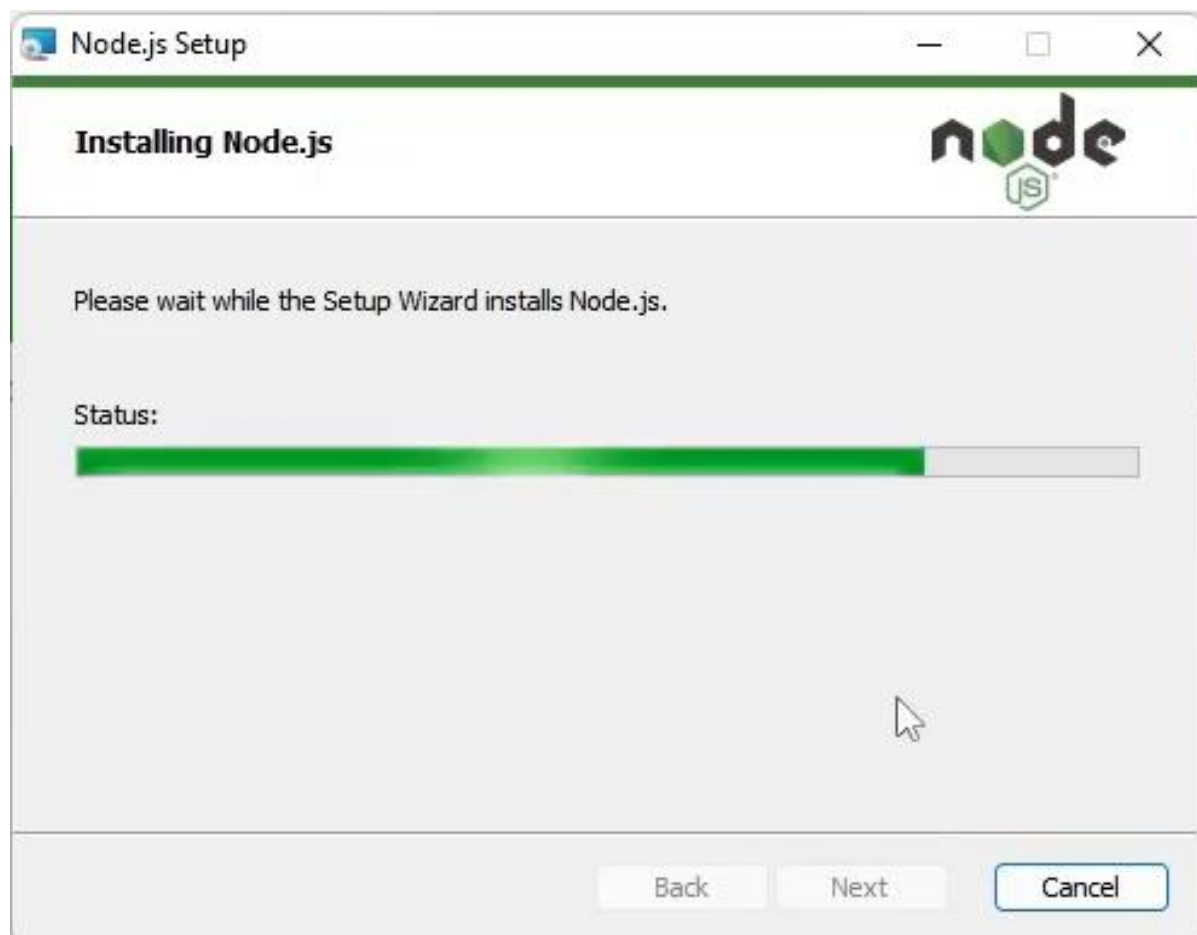


Рисунок 14 — Процесс установки Node.js

После завершения установки была выполнена проверка корректности работы. Для этого в терминале Visual Studio Code были последовательно введены команды. Для разработки дипломного проекта в Visual Studio Code были дополнительно установлены следующие расширения через встроенный магазин. Процесс установки расширений в VS Code показан на рисунке 15.

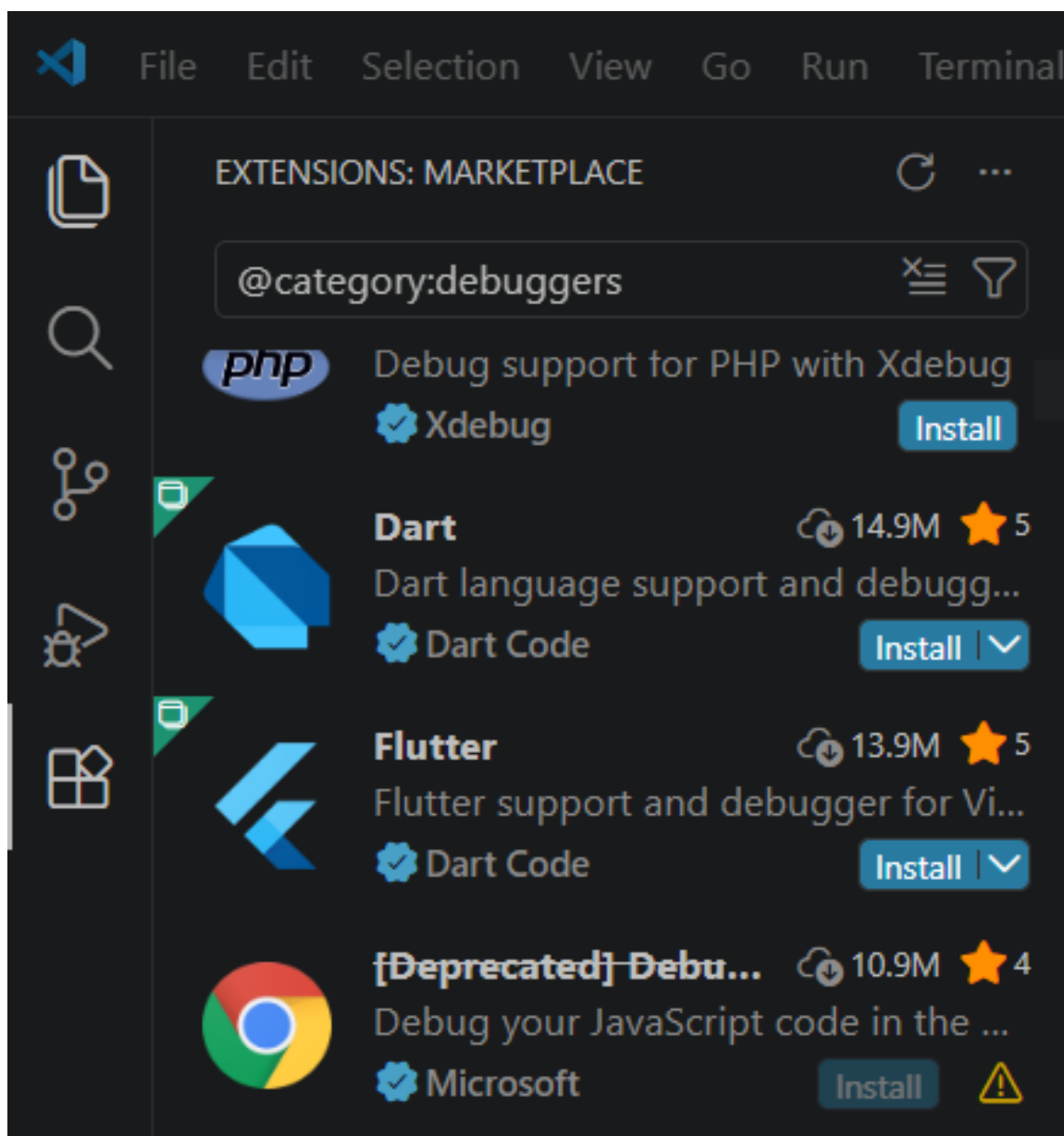
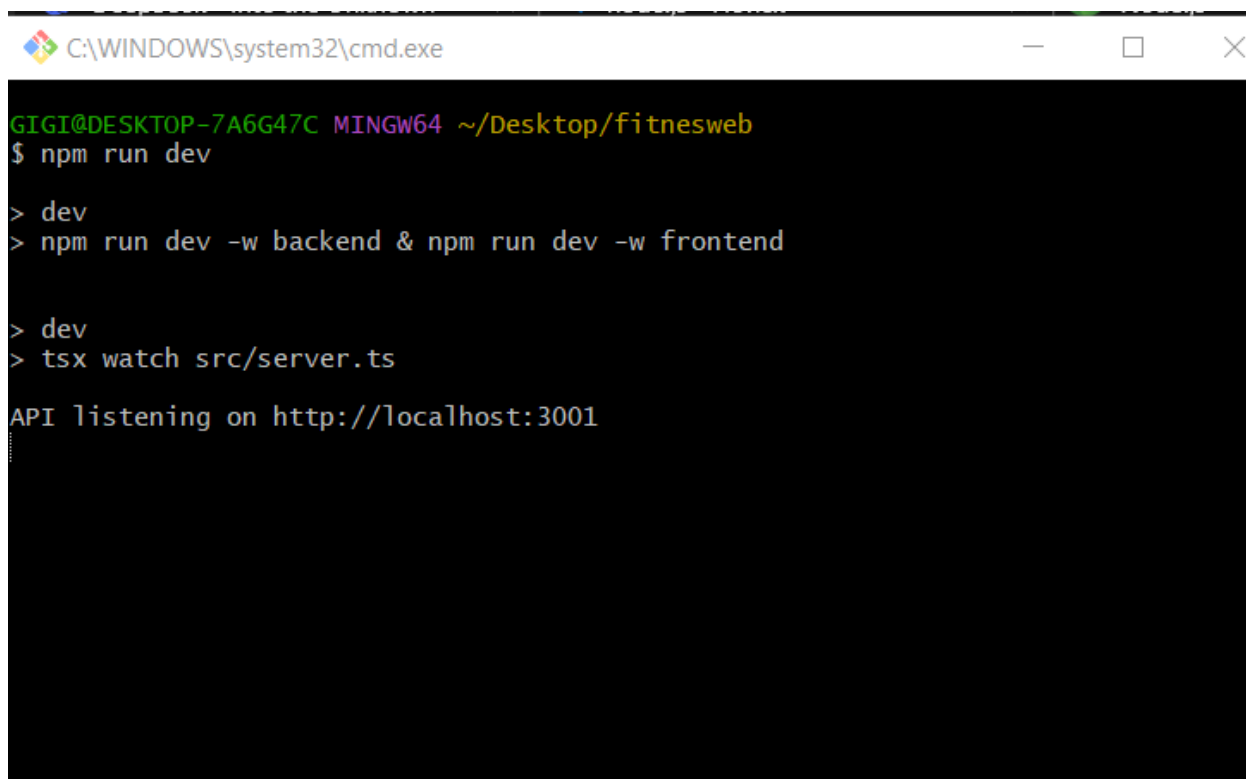


Рисунок 15 — Управление расширениями в Visual Studio Code

Далее в среде был создан новый проект: через меню выбран пункт «Open Folder», после чего инициализирован файл `package.json` с помощью команды `npm init` в интегрированном терминале. Этот этап позволил управлять зависимостями проекта непосредственно из среды разработки.

На рисунке 16 показан процесс инициализации Node.js-проекта в терминале Visual Studio Code.



```
C:\WINDOWS\system32\cmd.exe
GIGI@DESKTOP-7A6G47C MINGW64 ~/Desktop/fitnesweb
$ npm run dev
> dev
> npm run dev -w backend & npm run dev -w frontend
> dev
> tsx watch src/server.ts
API listening on http://localhost:3001
```

Рисунок 16 — Создание нового проекта Node.js в VS Code

1.7 Проектирование и разработка

Проект представляет собой веб-приложение Fitness Center Web, разработанное для автоматизации работы фитнес-центра, включая управление расписанием занятий, запись клиентов, контроль загруженности тренеров и оборудования, а также формирование отчетности. Приложение создано с использованием клиент-серверной архитектуры: клиентская часть реализована на React с TypeScript, серверная часть — на Node.js с Express и Prisma ORM, что обеспечивает современный веб-интерфейс, доступный через браузер, и надежную работу под управлением любой операционной системы.

Приложение состоит из двух основных программных модулей: клиентского и серверного, объединяющих пользовательский интерфейс, бизнес-логику и логику обработки данных. Структурная организация программы отражена на диаграмме классов, представленной на рисунке 17.

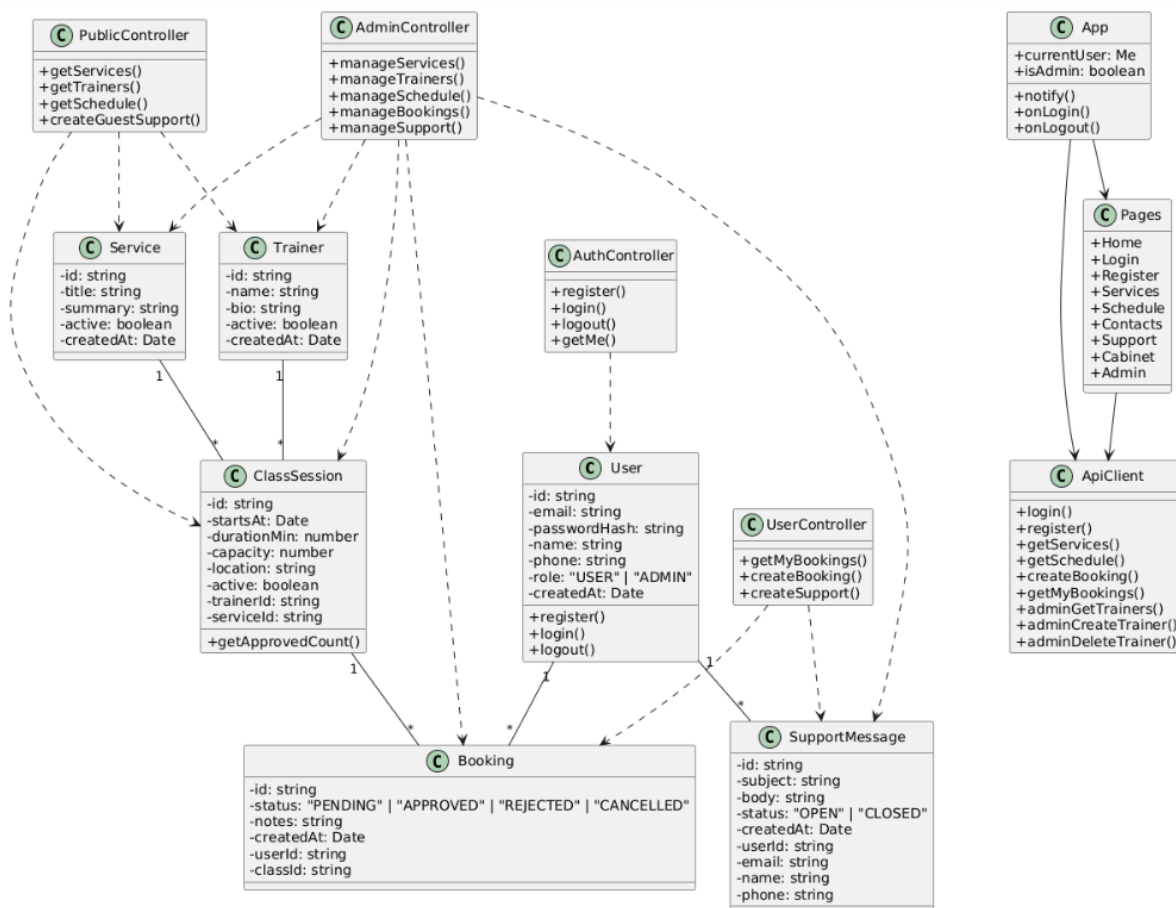


Рисунок 17 — Диаграмма классов

Графический интерфейс приложения представляет собой веб-страницы, доступные через браузер. Интерфейс включает следующие основные разделы: главная страница с приветствием и краткой информацией о фитнес-центре; страница услуг с перечнем всех доступных тренировок и их описанием; страница расписания с отображением занятий на ближайшие дни; страница контактов с информацией о местонахождении и способах связи; страница поддержки с формой для отправки обращений; страница входа и регистрации для аутентификации пользователей; личный кабинет клиента с отображением его заявок и статусов их обработки в табличном виде; административная панель с возможностью переключения между вкладками для управления тренерами, услугами, расписанием, заявками и обращениями в поддержку.

В таблицах приложения выводятся следующие данные:

- для расписания: дата и время занятия, услуга, тренер, место проведения, количество свободных мест и кнопка записи;
- для личного кабинета: дата создания заявки, время занятия, услуга, тренер, статус заявки с цветовой индикацией;
- для административной панели: списки тренеров, услуг, занятий, заявок и обращений с соответствующими атрибутами и действиями.

Пользователь взаимодействует с программой через набор управляющих элементов веб-интерфейса, инициируя такие операции, как регистрация и вход в систему, просмотр услуг и расписания, запись на занятия, просмотр и отслеживание статуса своих заявок, отправка обращений в поддержку, а также для администратора — добавление, редактирование и удаление тренеров, услуг и занятий, обработка заявок клиентов, просмотр и удаление обращений в поддержку.

Автоматизация сбора и обработки данных реализована с применением серверной логики на Node.js и Express, а также базы данных SQLite с ORM Prisma. Сервер обрабатывает входящие запросы от клиента, выполняет валидацию данных, проверку прав доступа и взаимодействие с базой данных. После получения данных с сервера клиентское приложение выполняет их обработку и представляет пользователю в табличном виде. Предусмотрена цветовая индикация статусов заявок и активных элементов интерфейса, что упрощает визуальный анализ состояния записей.

Структура базы данных приложения отражена на диаграмме, представленной на рисунке 18.

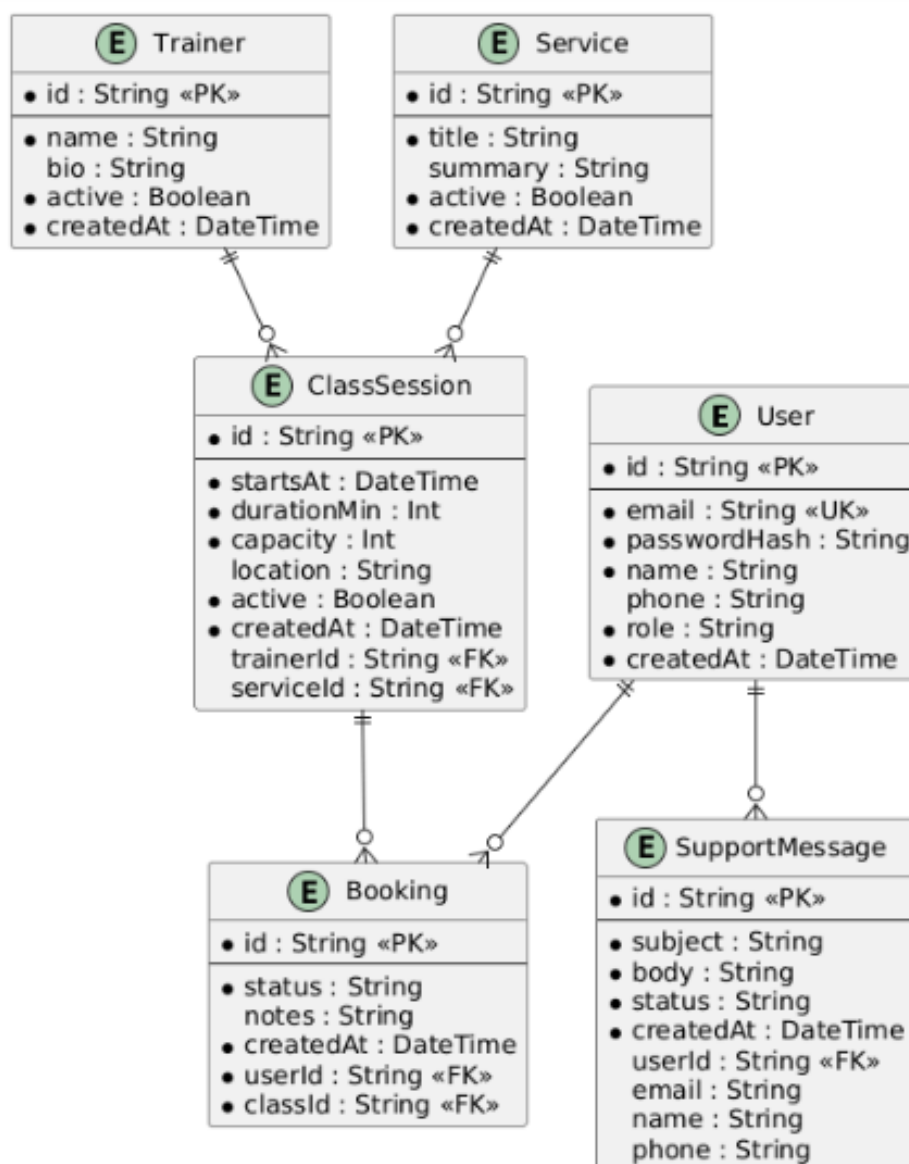


Рисунок 18 — Диаграмма базы данных

Пользователь имеет возможность просмотреть детальную информацию по любой заявке или обращению, а также по любому занятию, тренеру или услуге в административной панели.

Дипломный проект автоматизирует процессы записи клиентов на занятия, управления расписанием и контроля загруженности фитнес-центра, что способствует повышению эффективности администрирования, сокращению временных затрат персонала и улучшению качества обслуживания посетителей.

1.8 Разработка управляющей программы

Проект представляет собой веб-приложение Fitness Center Web, разработанное для автоматизации работы фитнес-центра, включая управление расписанием занятий, запись клиентов и контроль загруженности оборудования. Приложение создано с использованием клиент-серверной архитектуры: клиентская часть реализована на React с TypeScript, серверная часть — на Node.js с Express и Prisma ORM. Центральными элементами архитектуры являются клиентское приложение React и серверное приложение Express, которые взаимодействуют между собой посредством HTTP-запросов. Модель данных представлена класса-сущностями User, Trainer, Service, ClassSession, Booking и SupportMessage, которые транслируются в таблицы базы данных через Prisma ORM.

1.8.1 Инициализация клиентского приложения

При запуске клиентского приложения вызывается функция `main()` в файле `main.tsx`, которая выполняет инициализацию React-приложения и монтирование корневого компонента `App` в DOM-элемент с идентификатором `root`. В данном методе создается корневой узел React и оборачивается в `BrowserRouter` для обеспечения маршрутизации. Фрагмент кода инициализации представлен в листинге 1.

Листинг 1 — Инициализация React-приложения

```
ReactDOM.createRoot(document.getElementById("root")!).render(
  (
    <React.StrictMode>
      <BrowserRouter>
        <App />
      </BrowserRouter>
    </React.StrictMode>
  );
```

Корневой компонент `App` управляет состоянием авторизованного пользователя и глобальными уведомлениями. При монтировании компонента

вызывается `useEffect`, который выполняет запрос к серверу для восстановления сессии через метод `api.me()`. В листинге 2 представлен фрагмент кода, отвечающий за загрузку текущего пользователя.

Листинг 2 — Восстановление сессии пользователя

```
useEffect(() => {  
    api.me().then(setMe).catch(() => setMe(null));  
}, []);
```

1.8.2 Инициализация серверного приложения

При запуске серверного приложения создается экземпляр `Express`, настраиваются `middleware` и подключаются маршруты. В листинге 3 представлен фрагмент кода, отвечающий за настройку сервера.

Листинг 3 — Настройка Express-сервера

```
const app = express();  
app.use(cors({ origin: env.APP_ORIGIN, credentials: true  
}));  
app.use(express.json({ limit: "1mb" }));  
app.use(cookieParser());  
app.use("/api/auth", authRouter);  
app.use("/api/public", publicRouter);  
app.use("/api/user", userRouter);  
app.use("/api/admin", adminRouter);  
app.listen(env.PORT, () => console.log(`API listening on  
port ${env.PORT}`));
```

Структура серверного приложения включает четыре группы маршрутов: публичные – доступны без авторизации, аутентификации, пользовательские – требуют авторизации, и административные – требуют прав администратора. Каждая группа маршрутов вынесена в отдельный файл с использованием `Router` из `Express`.

1.8.3 API-клиент для взаимодействия с сервером

Для централизованного взаимодействия клиента с сервером разработан модуль `api.ts`, который содержит класс `ApiError` и набор методов для

выполнения HTTP-запросов. Базовый метод `req()` автоматически подставляет учетные данные и обрабатывает ответы сервера. В листинге 4 представлен фрагмент кода API-клиента.

Листинг 4 — API-клиент

```
async function req<T>(path: string, init?: RequestInit):
Promise<T> {
    const res = await fetch(`${API_BASE}${path}`, {
        credentials: "include",
        headers: { "content-type": "application/json",
... (init?.headers || {}) },
        ...init
    });
    if (!res.ok) throw new ApiError(res.status, message,
data);
    return await res.json();
}
export const api = {
    login: (email, password) => req<Me>("/api/auth/login", {
        method: "POST",
        body: JSON.stringify({ email, password })
    }),
    schedule: () => req<any[]>("/api/public/schedule"),
    book: (classId, notes) => req("/api/user/book", {
        method: "POST",
        body: JSON.stringify({ classId, notes })
    })
};
```

1.8.4 Аутентификация и авторизация

Процесс авторизации реализован в асинхронном методе `Login` компонента `Login.tsx` и использует API-клиент для отправки учетных данных на сервер. При нажатии кнопки «Войти» выполняется валидация введенных данных и отправка POST-запроса на эндпоинт `/api/auth/login`. Фрагмент кода авторизации представлен в листинге 5.

					ПГУ1.090301.47.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		46

Листинг 5 — Авторизация пользователя на клиенте

```
async function submit(e: React.FormEvent) {
  e.preventDefault();
  setErr("");
  try {
    const me = await api.login(email, password);
    onLogin(me);
  } catch {
    setErr("Неверные данные");
  }
}
```

На серверной стороне обработка запроса авторизации выполняется в `authRouter`. В листинге 6 представлен фрагмент кода эндпоинта `login`.

Листинг 6 — Эндпоинт авторизации на сервере

```
authRouter.post("/login", async (req, res) => {
  const { email, password } = req.body;
  const user = await prisma.user.findUnique({ where: { email
  } });
  if (!user) return res.status(401).json({ error:
  "invalid_credentials" });
  const ok = await bcrypt.compare(password,
  user.passwordHash);
  if (!ok) return res.status(401).json({ error:
  "invalid_credentials" });
  const token = signSessionToken({ sub: user.id, role:
  user.role });
  res.cookie(env.COOKIE_NAME, token, { httpOnly: true,
  sameSite: "lax" });
  return res.json({ id: user.id, role: user.role, email:
  user.email, name: user.name });
});
```

При успешной авторизации сервер генерирует JWT-токен, сохраняет его в `httpOnly cookie` и возвращает клиенту информацию о пользователе.

Клиент сохраняет состояние авторизации и перенаправляет пользователя в личный кабинет.

1.8.5 Отображение расписания и запись на занятия

Основной функционал приложения по отображению расписания реализован в компоненте Schedule.tsx. При загрузке компонента выполняется GET-запрос на эндпоинт /api/public/schedule для получения списка занятий. В листинге 7 представлен фрагмент кода компонента Schedule.

Листинг 7 — Загрузка расписания

```
typescript
useEffect(() => {
  api.schedule().then(setItems).catch(() => setItems([]));
}, []);
const rows = useMemo(() => items ?? [], [items]);
```

На серверной стороне обработка запроса расписания включает получение списка занятий из базы данных с привязкой данных о тренерах и услугах, а также вычисление количества одобренных заявок для каждого занятия. В листинге 8 представлен фрагмент кода эндпоинта schedule.

Листинг 8 — Эндпоинт получения расписания

```
publicRouter.get("/schedule", async (req, res) => {
  const classes = await prisma.classSession.findMany({
    where: { active: true },
    orderBy: { startsAt: "asc" },
    include: { trainer: true, service: true }
  });
  const ids = classes.map((c) => c.id);
  const approvedCounts = await prisma.booking.groupBy({
    by: ["classId"],
    where: { classId: { in: ids }, status: "APPROVED" },
    _count: { _all: true }
  });
  const m = new Map(approvedCounts.map((r) => [r.classId, r._count._all]));
```

```

        return res.json(classes.map((c) => ({ ...c, approvedCount:
m.get(c.id) ?? 0 })));
    });

```

Запись на занятие выполняется при нажатии кнопки «Записаться» в строке таблицы расписания. Метод `book()` отправляет POST-запрос на эндпоинт `/api/user/book`. В листинге 9 представлен фрагмент кода записи на занятие.

Листинг 9 — Запись на занятие

```

async function book(classId: string) {
    if (!me) return;
    try {
        await api.book(classId);
        notify("Заявка на запись отправлена");
    } catch (e: any) {
        if (e instanceof ApiError && e.data?.error === "full")
            return notify("Мест нет");
        if (e.data?.error === "already_booked")
            return notify("Вы уже записаны");
    }
}

```

1.8.6 Административная панель

Управление данными фитнес-центра реализовано в компоненте `Admin.tsx`, который содержит вкладки для управления тренерами, услугами, расписанием, заявками и обращениями в поддержку. При переключении между вкладками выполняются соответствующие запросы к API для получения данных. В листинге 10 представлен фрагмент кода загрузки списка тренеров.

Листинг 10 — Загрузка списка тренеров в админ-панели

```

async function openTab(next: typeof tab) {
    setTab(next);
    try {

```

```

        if (next === "trainers")      setTrainers(await
api.adminTrainers());
        if (next === "services")      setServices(await
api.adminServices());
        if (next === "classes")      setClasses(await
api.adminClasses());
        if (next === "bookings")      setBookings(await
api.adminBookings());
        if (next === "support")      setSupport(await
api.adminSupport());
    } catch {
        notify("Ошибка загрузки");
    }
}

```

Добавление нового тренера реализовано в методе `addTrainer`, который отправляет POST-запрос на эндпоинт `/api/admin/trainers`. В листинге 11 представлен фрагмент кода добавления тренера на серверной стороне.

Листинг 11 — Создание тренера на сервере

```

adminRouter.post("/trainers", async (req, res) => {
    const schema = z.object({ name: z.string().min(1), bio:
z.string().optional() });
    const parsed = schema.safeParse(req.body);
    if (!parsed.success) return res.status(400).json({ error:
"invalid_input" });
    const trainer = await prisma.trainer.create({
        data: { ...parsed.data, active: true }
    });
    return res.json(trainer);
});

```

Для обеспечения безопасности административные маршруты защищены middleware `requireAdmin`, который проверяет наличие и валидность JWT-токена в cookie, а также наличие у пользователя роли ADMIN. В листинге 12 представлен фрагмент кода middleware.

Листинг 12 — Middleware проверки прав администратора

```
export function requireAdmin(req: AuthedRequest, res:
Response, next: NextFunction) {
  requireAuth(req, res, () => {
    if (req.user?.role !== "ADMIN")
      return res.status(403).json({ error: "forbidden" });
    next();
  });
}
```

1.8.7 База данных и ORM

Для взаимодействия с базой данных используется Prisma ORM. Схема базы данных описана в файле `schema.prisma`, фрагмент которой представлен в листинге 13.

Листинг 13 — Схема базы данных Prisma

```
model User {
  id          String    @id @default(cuid())
  email       String    @unique
  passwordHash String
  name        String
  phone       String?
  role        UserRole  @default(USER)
  bookings   Booking[]
  messages    SupportMessage[]
}

model ClassSession {
  id          String    @id @default(cuid())
  startsAt    DateTime
  durationMin Int
  capacity    Int       @default(12)
  location    String?
  trainerId   String?
  trainer     Trainer?  @relation(fields: [trainerId],
references: [id])
}
```

```

        serviceId    String?
        service      Service? @relation(fields: [serviceId],
references: [id])
        bookings     Booking[]
    }
    model Booking {
        id            String    @id @default(cuid())
        status        String    @default("PENDING")
        userId        String
        user          User      @relation(fields: [userId], references:
[id])
        classId       String
        class          ClassSession @relation(fields: [classId],
references: [id])
    }

```

1.8.8 Вспомогательные функции и утилиты

Приложение включает ряд вспомогательных функций, обеспечивающих корректную работу интерфейса и обработку данных. Функция `formatDT()` форматирует дату и время в российский формат. Функция `statusView()` возвращает локализованное название статуса заявки и соответствующий CSS-класс для цветовой индикации. В листинге 14 представлен фрагмент кода этих утилит.

Листинг 14 — Вспомогательные функции

```

function fmt(iso: string) {
    return new Date(iso).toLocaleString("ru-RU", {
        day: "2-digit",
        month: "2-digit",
        hour: "2-digit",
        minute: "2-digit"
    });
}

function statusView(status: string): { label: string; cls:
string } {

```

```

switch (status) {
    case "PENDING": return { label: "Ожидает", cls: "status-
pending" };
    case "APPROVED": return { label: "Одобрено", cls:
"status-approved" };
    case "REJECTED": return { label: "Отклонено", cls:
"status-rejected" };
    case "CANCELLED": return { label: "Отменено", cls:
"status-cancelled" };
    default: return { label: status, cls: "" };
}
}

```

1.8.9 Архитектура развертывания

Развертывание приложения предполагает его установку на серверное оборудование фитнес-центра или использование облачного хостинга. Для корректной работы необходим сервер с установленной платформой Node.js и системой управления базами данных SQLite. Клиентская часть представляет собой статические файлы, которые могут быть развернуты на любом веб-сервере или отдаваться тем же Node.js-сервером. Пользователи получают доступ к приложению через любой современный веб-браузер по протоколу HTTPS.

1.9 Графический интерфейс клиентского приложения

В данном разделе будут описаны сценарии взаимодействия с веб-интерфейсом программы. Для общего понимания функциональности разработанной программы рассмотрим главный экран приложения на рисунке 19. Интерфейс программы выполнен в современном минималистичном стиле с использованием светлой цветовой схемы и округлых элементов. Верхняя часть экрана занята навигационной панелью, которая остается видимой при перемещении между страницами. В левой части навигационной панели расположен логотип Fitness Center с кратким

описанием «Онлайн-запись, кабинеты, админ-панель». Центральную часть занимает навигационное меню со ссылками на основные разделы: «Главная», «Услуги», «Расписание», «Контакты», «Поддержка». Правая часть панели отведена под блок авторизации, который в зависимости от состояния пользователя отображает либо кнопки «Войти» и «Регистрация», либо имя пользователя и кнопку «Выйти».

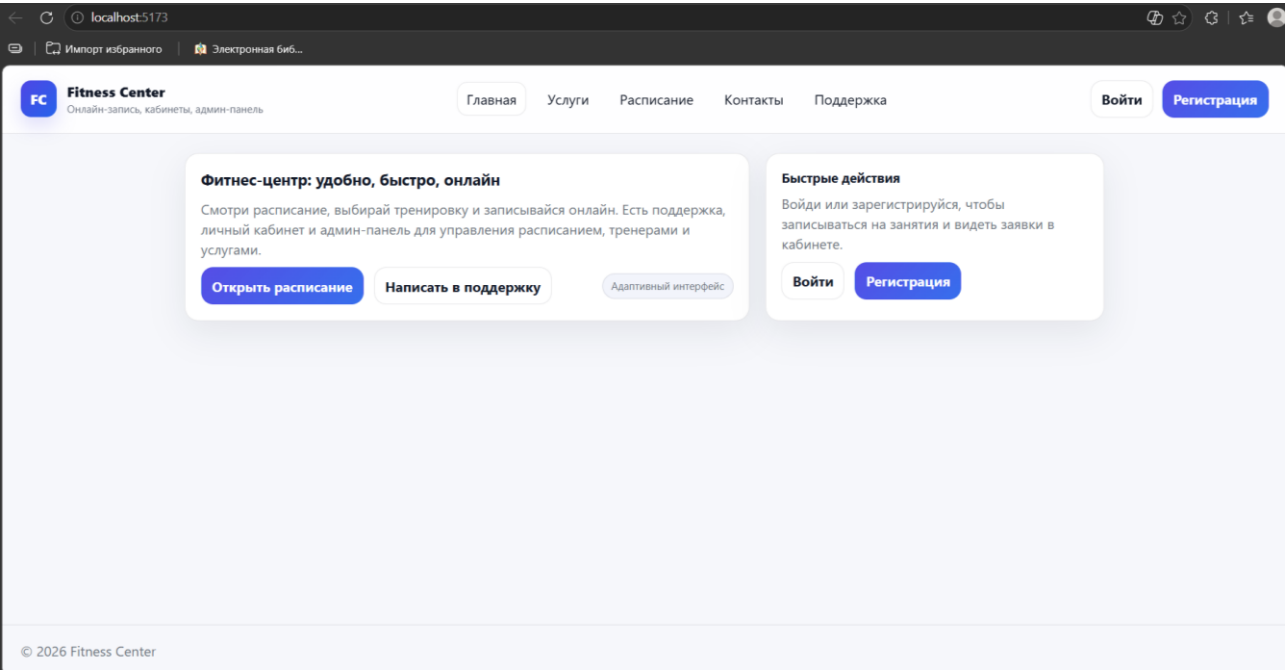


Рисунок 19 — Главный экран программы

Сразу после запуска пользователю доступны все публичные разделы приложения. Для удобства работы экран разделен на несколько функциональных областей: верхняя панель навигации с логотипом, навигационным меню и блоком авторизации, центральная область контента для отображения страниц, нижний колонтитул с информацией о годе создания и копирайтом. На главной странице также присутствуют карточки с быстрыми действиями и приветственная информация.

1.9.1 Сценарий «Регистрация нового пользователя»

Для начала работы с приложением новому пользователю необходимо создать учетную запись. Для этого необходимо нажать кнопку «Регистрация»

в правом верхнем углу экрана, после чего откроется страница регистрации, представленная на рисунке 20.

FC Fitness Center
Онлайн-запись, кабинеты, админ-панель

Главная Услуги Расписание Контакты Поддержка

Войти Регистрация

Регистрация

Email: you@example.com

Телефон (опционально): +7...

Имя: Имя

Пароль: Минимум 8 символов

Создать аккаунт

© 2026 Fitness Center

Рисунок 20 — Страница регистрации

Пользователь вводит свои данные: адрес электронной почты (обязательно), имя (обязательно), номер телефона (опционально) и пароль, требующий минимум 8 символов). При вводе пароля поле скрывает символы для обеспечения безопасности. После заполнения всех полей пользователь нажимает кнопку «Создать аккаунт». При успешной регистрации пользователь автоматически входит в систему и перенаправляется в личный кабинет. В случае, если указанный email уже зарегистрирован, отображается сообщение об ошибке «Этот email уже занят». При несоответствии пароля требованиям длины выводится соответствующее предупреждение. Результат успешной регистрации представлен на рисунке 21.

FC Fitness Center
Онлайн-запись, кабинеты, админ-панель

Главная Услуги Расписание Контакты Поддержка Кабинет

Виктор Арестов Выйти

Личный кабинет

Здесь отображаются твои записи на занятия.
Записей пока нет. Перейди в расписание и запишись.

webofscience@mahle.ru

Рисунок 21 — Личный кабинет после регистрации

1.9.2 Сценарий «Авторизация в системе»

Для входа в систему зарегистрированному пользователю необходимо нажать кнопку «Войти» в правом верхнем углу экрана, после чего откроется страница входа, представленная на рисунке 22.

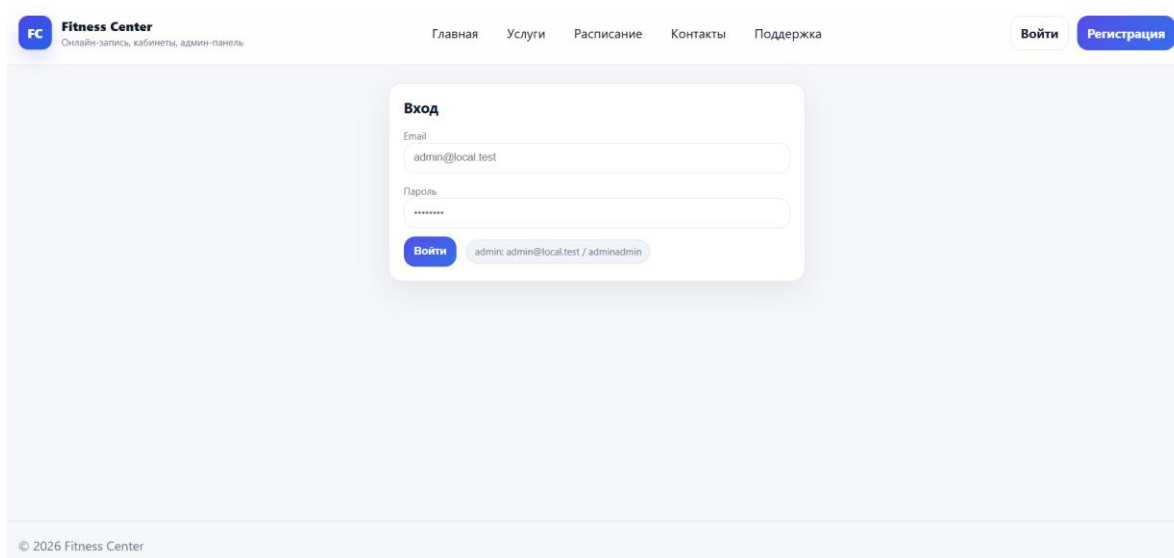


Рисунок 22 — Страница входа

Пользователь вводит адрес электронной почты и пароль в соответствующие поля, после чего нажимает кнопку «Войти». При успешной авторизации пользователь перенаправляется в личный кабинет, а в правом верхнем углу отображается имя пользователя и кнопка «Выйти». В случае ввода неверных учетных данных отображается сообщение «Неверные данные». Результат успешной авторизации представлен на рисунке 23.

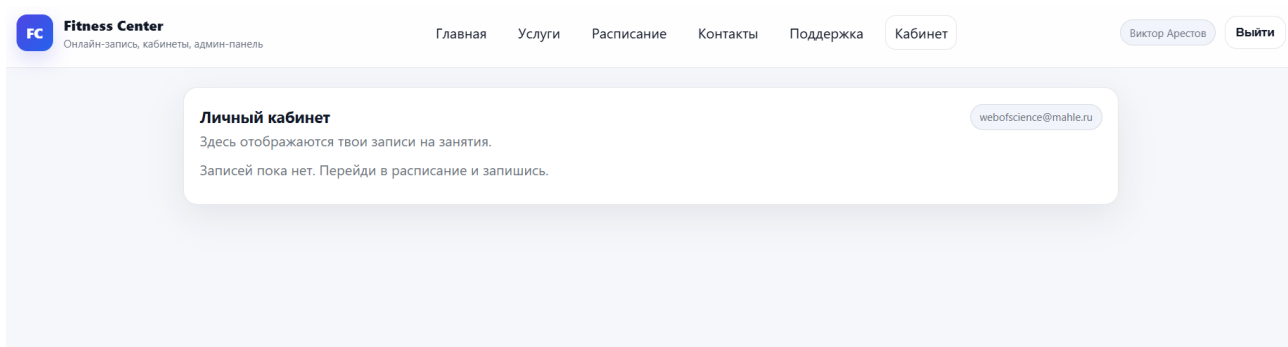


Рисунок 23 — Интерфейс после успешного входа

1.9.3 Сценарий «Просмотр расписания и запись на занятие»

После успешной авторизации пользователь может просмотреть актуальное расписание занятий, перейдя по ссылке «Расписание» в навигационном меню. Страница расписания представлена на рисунке 24.

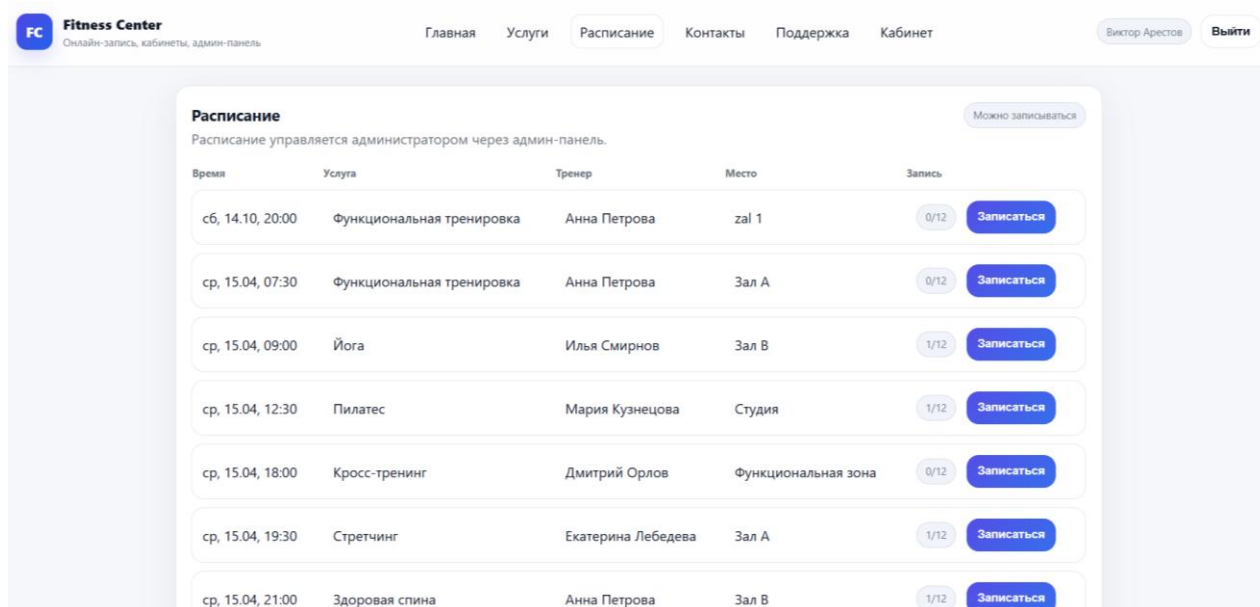


Рисунок 24 — Страница расписания

На странице отображается таблица со списком всех активных занятий на ближайшие дни. Для каждого занятия указаны: дата и время начала, название услуги, имя тренера, место проведения (локация), а также количество занятых мест и общая вместимость в виде «X/Y», где X — количество одобренных записей, Y — максимальная вместимость зала. Для записи на занятие пользователь нажимает кнопку «Записаться» в соответствующей строке таблицы, как показано на рисунке 25.

ср, 15.04, 12:30	Пилатес	Мария Кузнецова	Студия	1/12	Записаться
ср, 15.04, 18:00	Кросс-тренинг	Дмитрий Орлов	Функциональная зона	0/12	Записаться
ср, 15.04, 19:30	Стретчинг	Екатерина Лебедева	Зал А	1/12	Записаться
ср, 15.04, 21:00	Здоровая спина	Анна Петрова	Зал В	1/12	Записаться
чт, 16.04, 07:30	Кросс-тренинг	Илья Смирнов	Зал В	0/12	Записаться
чт, 16.04, 09:00	Стретчинг	Мария Кузнецова	Студия	1/12	Записаться

Заявка на запись отправлена

Рисунок 25 — Запись на занятие

При нажатии кнопки программа отправляет запрос на сервер и при успешном создании заявки отображает всплывающее уведомление «Заявка на запись отправлена». В случае, если места на занятие закончились, отображается сообщение «Мест нет». Если пользователь уже записывался на это занятие ранее, выводится уведомление «Ты уже отправлял заявку на это занятие». Результат создания заявки в личном кабинете представлен на рисунке 26.

Личный кабинет					webofscience@mahle.ru
Здесь отображаются твои записи на занятия.					
Создано	Время занятия	Услуга	Тренер	Статус	
28.05, 01:40	15.04, 18:00	Кросс-тренинг	Дмитрий Орлов	Ожидает	
28.05, 01:38	14.10, 20:00	Функциональная тренировка	Анна Петрова	Ожидает	

Рисунок 26 — Личный кабинет после записи

1.9.4 Сценарий «Просмотр заявок в личном кабинете»

Для получения подробной информации о своих заявках пользователь переходит в личный кабинет по ссылке «Кабинет» в навигационном меню. Страница личного кабинета представлена на рисунке 27.

Личный кабинет

Здесь отображаются твои записи на занятия.

webofscience@mahle.ru

Создано	Время занятия	Услуга	Тренер	Статус
28.05, 01:40	15.04, 18:00	Кросс-тренинг	Дмитрий Орлов	Одобрено
28.05, 01:38	14.10, 20:00	Функциональная тренировка	Анна Петрова	Отменено

Рисунок 27 — Личный кабинет

В личном кабинете отображается таблица всех заявок текущего пользователя. Для каждой заявки указаны: дата и время создания записи, дата и время выбранного занятия, название услуги, имя тренера и статус заявки. Статусы отображаются в виде цветных пилюль: «Ожидает» (желтый/оранжевый фон), «Одобрено» (зеленый фон), «Отклонено» (красный фон), «Отменено» (серый фон). При отсутствии заявок отображается сообщение «Записей пока нет. Перейди в расписание и запишись».

1.9.5 Сценарий «Отправка обращения в поддержку»

Для связи с администрацией фитнес-центра пользователь может отправить обращение через страницу поддержки, перейдя по ссылке «Поддержка» в навигационном меню. Страница поддержки представлена на рисунке 28.

Поддержка

Можно писать как гостем, так и из кабинета (тогда сообщение привяжется к пользователю).

Тема

Например: Вопрос по абонементу

Сообщение

Опиши проблему или вопрос

Отправить

Отправка от аккаунта

Рисунок 28 — Страница поддержки для авторизованного пользователя

Для авторизованных пользователей форма содержит только поля «Тема» и «Сообщение», так как данные пользователя автоматически привязываются к обращению. Пользователь вводит тему и текст сообщения, после чего нажимает кнопку «Отправить». При успешной отправке отображается уведомление «Сообщение отправлено», а поля формы очищаются. Для неавторизованных пользователей форма также содержит поля для ввода email, имени и телефона, как показано на рисунке 29.

Рисунок 29 — Страница поддержки для гостя

1.9.6 Сценарий «Управление тренерами»

Для администратора фитнес-центра доступна административная панель, открываемая по ссылке «Админ» в навигационном меню. Административная панель содержит несколько вкладок, включая управление тренерами, услугами, расписанием, заявками и поддержкой. Страница управления тренерами представлена на рисунке 30.

Админ-панель

Обновить

Расписание

Тренеры

Услуги

Заявки

Поддержка

Добавить тренера

Имя

Віо

Добавить

Список

Екатерина Лебедева активен

Удалить

Дмитрий Орлов активен

Удалить

Мария Кузнецова активен

Удалить

Илья Смирнов активен

Удалить

Анна Петрова активен

Удалить

Рисунок 30 — Управление тренерами в административной панели

Для добавления нового тренера администратор заполняет поля «Имя» и опциональное поле «Віо» в левой части вкладки, после чего нажимает кнопку «Добавить». Новый тренер появляется в списке справа. Для удаления тренера администратор нажимает кнопку «Удалить» рядом с соответствующим тренером в списке. Перед удалением программа отображает диалоговое окно с подтверждением «Удалить тренера?». Результат добавления тренера представлен на рисунке 31.

Админ-панель

Обновить

Расписание

Тренеры

Услуги

Заявки

Поддержка

Добавить тренера

Имя

Віо

Добавить

Список

Штормголец Александр активен

Удалить

Екатерина Лебедева активен

Удалить

Дмитрий Орлов активен

Удалить

Мария Кузнецова активен

Удалить

Илья Смирнов активен

Удалить

Анна Петрова активен

Удалить

Рисунок 31 — Список тренеров после добавления

1.9.7 Сценарий «Управление расписанием»

Для управления расписанием администратор переходит на вкладку «Расписание» в административной панели. Страница управления расписанием представлена на рисунке 32.

Админ-панель Обновить

Расписание | Тренеры | Услуги | Заявки | Поддержка

Добавить занятие

Дата/время:

Длительность (мин): Мест:

Локация:

Тренер: Услуга:

Добавить

Расписание

Время	Услуга	Тренер	Записано
14.10.2000, 20:00:00	Функциональная тренировка	Анна Петрова	0/12 Удалить
15.04.2026, 07:30:00	Функциональная тренировка	Анна Петрова	0/12 Удалить
15.04.2026, 09:00:00	Йога	Илья Смирнов	1/12 Удалить
15.04.2026, 12:30:00	Пилатес	Мария Кузнецова	1/12 Удалить
15.04.2026, 18:00:00	Кросс-тренинг	Дмитрий Орлов	1/12 Удалить

Рисунок 32 — Управление расписанием в административной панели

Для добавления нового занятия администратор заполняет форму: выбирает дату и время, указывает длительность в минутах, количество мест, локацию, выбирает тренера и услугу из выпадающих списков, после чего нажимает кнопку «Добавить». Новое занятие появляется в таблице расписания справа. В таблице отображаются дата и время занятия, название услуги, имя тренера, количество одобренных заявок и вместимость. Для удаления занятия администратор нажимает кнопку «Удалить» рядом с соответствующим занятием. Результат добавления занятия представлен на рисунке 33.

06.06.2026,
06:06:00

Функциональная
тренировка

Штормголец
Александр

0/6

Удалить

Рисунок 33 — Расписание после добавления занятия

1.9.8 Сценарий «Обработка заявок клиентов»

Для обработки заявок клиентов администратор переходит на вкладку «Заявки» в административной панели. Страница управления заявками представлена на рисунке 34.

Админ-панель

Обновить

Расписание

Тренеры

Услуги

Заявки

Поддержка

Заявки на запись

Пользователь	Время занятия	Услуга	Тренер	Статус	Действия
webofscience@mahle.ru	15.04.2026, 18:00:00	Кросс-тренинг	Дмитрий Орлов	Одобрено	Отменить
webofscience@mahle.ru	14.10.2000, 20:00:00	Функциональная тренировка	Анна Петрова	Отменено	
admin@local.test	15.04.2026, 21:00:00	Здоровая спина	Анна Петрова	Ожидает	Одобрить Отклонить
admin@local.test	15.04.2026, 18:00:00	Кросс-тренинг	Дмитрий Орлов	Ожидает	Одобрить Отклонить

Рисунок 34 — Управление заявками в административной панели

В таблице отображаются все заявки от всех клиентов. Для каждой заявки указаны: email пользователя, дата и время занятия, название услуги, имя тренера, текущий статус (ожидает, одобрена, отклонена, отменена). Для заявок со статусом «Ожидает» доступны кнопки «Одобрить» и «Отклонить». Для заявок со статусом «Одобрено» доступна кнопка «Отменить». При нажатии на кнопку «Одобрить» статус заявки изменяется на «Одобрено», при этом система проверяет, не превышен ли лимит мест на занятии. Если лимит достигнут, отображается ошибка «Мест нет». При успешном одобрении

отображается уведомление «Заявка одобрена», и таблица обновляется. Результат одобрения заявки представлен на рисунке 35.

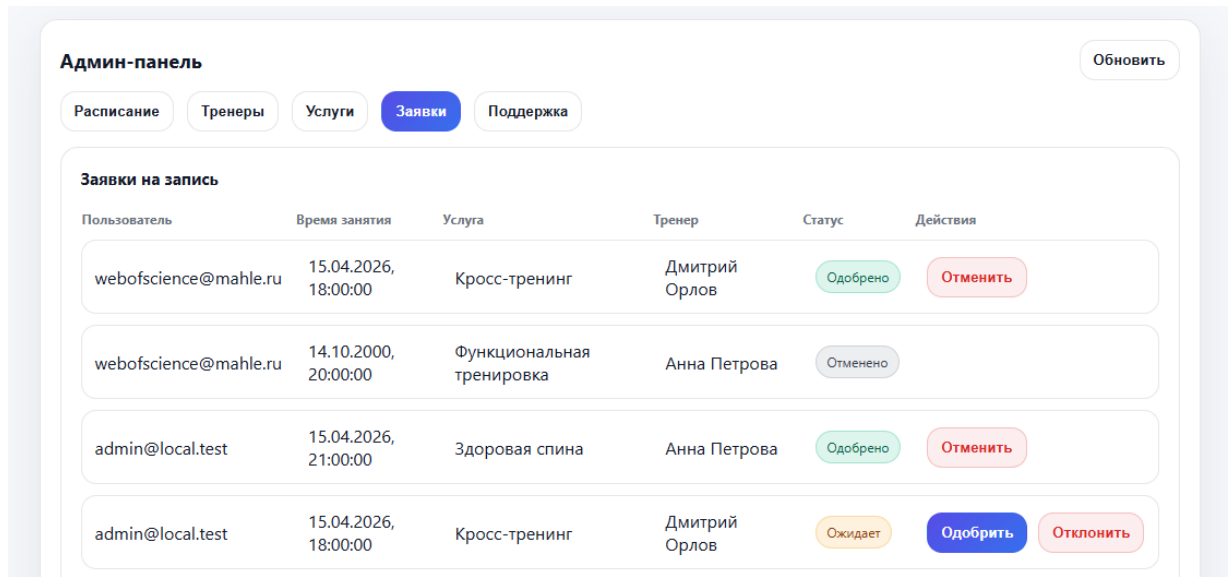


Рисунок 35 — Заявки после одобрения

1.9.9 Сценарий «Просмотр услуг»

Для ознакомления с перечнем тренировок, предлагаемых фитнес-центром, пользователь может перейти по ссылке «Услуги» в навигационном меню. Страница услуг представлена на рисунке 36. Данная страница является публичной и доступна как авторизованным, так и неавторизованным пользователям, что позволяет потенциальным клиентам ознакомиться с ассортиментом тренировок до регистрации в системе. Интерфейс страницы выполнен в виде сетки карточек, что обеспечивает удобное визуальное восприятие информации и позволяет легко просматривать доступные направления..

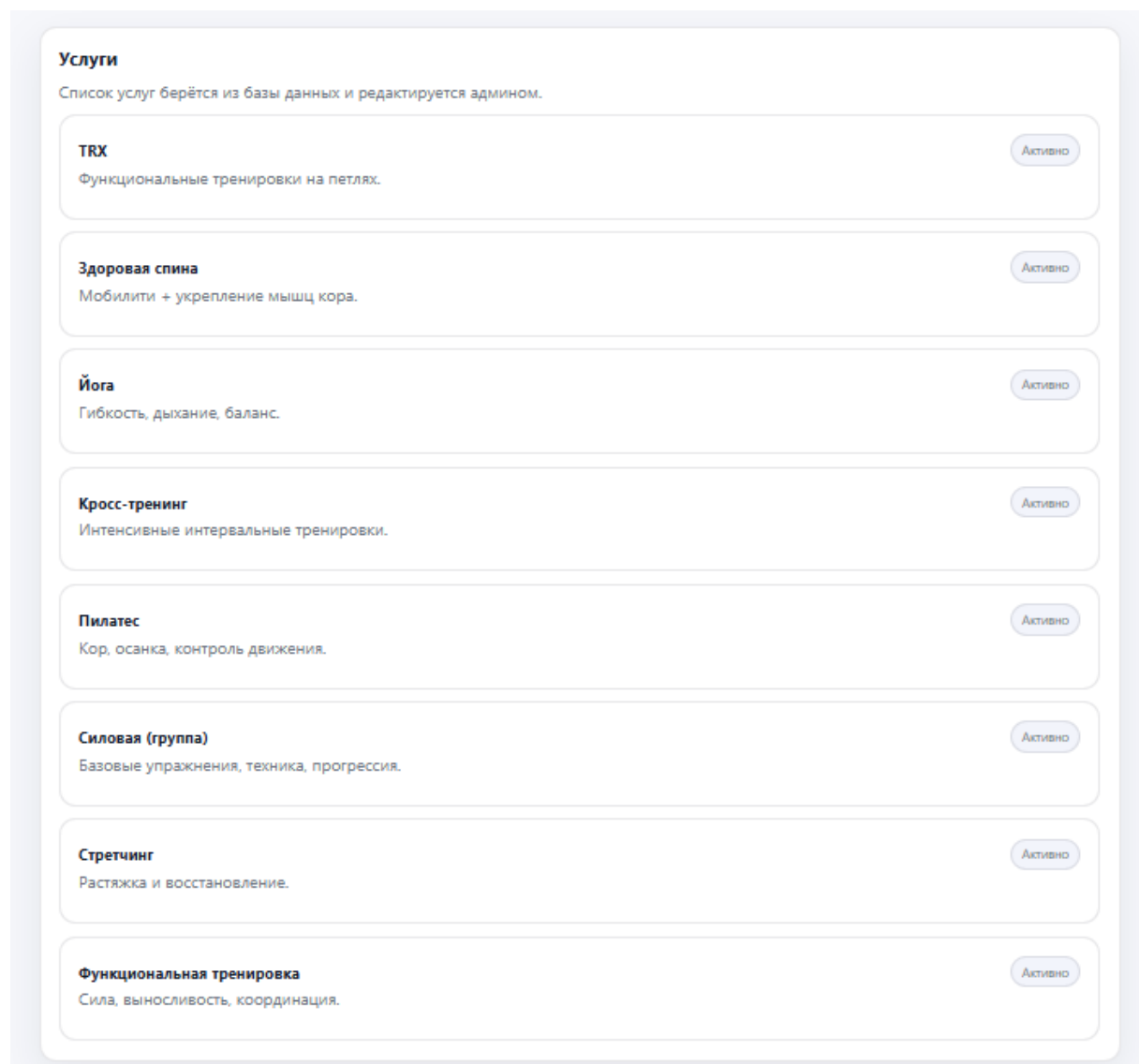


Рисунок 36 — Страница услуг

На странице отображается список всех активных услуг фитнес-центра. Каждая услуга представлена в виде отдельной карточки, содержащей название услуги и краткое описание. Если для услуги не указано описание, отображается текст «Описание не указано». В правом верхнем углу каждой карточки расположена пиллюля «Активно», указывающая на то, что услуга доступна для записи. Услуги загружаются из базы данных и могут изменяться администратором через административную панель. При отсутствии активных услуг отображается сообщение «Пока нет активных

услуг». Страница услуг доступна как авторизованным, так и неавторизованным пользователям.

1.9.10 Сценарий «Просмотр контактов»

Для получения контактной информации фитнес-центра пользователь переходит по ссылке «Контакты» в навигационном меню. Страница контактов представлена на рисунке 37.

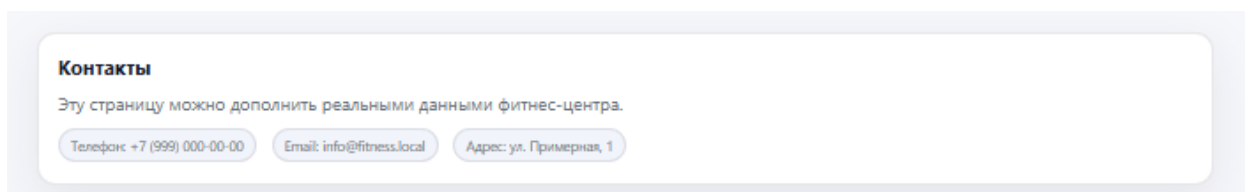


Рисунок 37 — Страница контактов

На странице отображается карточка с контактной информацией, содержащая следующие данные: номер телефона фитнес-центра, адрес электронной почты и физический адрес. Контактная информация представлена в виде цветных пилюль для удобства визуального восприятия. В реальном разворачивании приложения данная информация может быть заменена на актуальные данные конкретного фитнес-центра. Страница контактов доступна всем пользователям без авторизации.

1.9.11 Сценарий «Управление услугами»

Для управления услугами администратор переходит на вкладку «Услуги» в административной панели. Страница управления услугами представлена на рисунке 38.

Админ-панель

Обновить

Расписание

Тренеры

Услуги

Заявки

Поддержка

Добавить услугу

Название

Описание

Добавить

Список

Силовая (группа) активно

Удалить

TRX активно

Удалить

Здоровая спина активно

Удалить

Стретчинг активно

Удалить

Кросс-тренинг активно

Удалить

Пилатес активно

Удалить

Функциональная тренировка активно

Удалить

Йога активно

Удалить

Рисунок 38 — Управление услугами в административной панели

В левой части вкладки расположена форма для добавления новой услуги, содержащая поле «Название» (обязательное) и поле «Описание» (опциональное). Для добавления услуги администратор заполняет поля и нажимает кнопку «Добавить». Новая услуга появляется в списке справа. В правой части вкладки отображается таблица существующих услуг, где для каждой услуги указаны название, статус активности («активно» или «отключено») и кнопка «Удалить». Для удаления услуги администратор нажимает кнопку «Удалить» рядом с соответствующей услугой. Перед удалением программа отображает диалоговое окно с подтверждением «Удалить услугу?». Результат добавления услуги представлен на рисунке 39.

Админ-панель

Обновить

Расписание

Тренеры

Услуги

Заявки

Поддержка

Добавить услугу

Название

Подтягивания

Описание

Добавить

Список

Подтягивания

активно

Удалить

Силовая (группа)

активно

Удалить

TRX

активно

Удалить

Здоровая спина

активно

Удалить

Стретчинг

активно

Удалить

Кросс-тренинг

активно

Удалить

Пилатес

активно

Удалить

Функциональная тренировка

активно

Удалить

Йога

активно

Удалить

Рисунок 39 — Список услуг после добавления

1.9.12 Сценарий «Управление обращениями в поддержку»

Для просмотра и управления обращениями клиентов администратор переходит на вкладку «Поддержка» в административной панели. Страница управления обращениями представлена на рисунке 40.

Админ-панель

Обновить

Расписание

Тренеры

Услуги

Заявки

Поддержка

Сообщения поддержки

Когда	Тема	От	Статус	Текст		
15.04.2026, 15:39:41	Гость: Вопрос по расписанию	guest1@example.com	Открыто	Когда будут утренние занятия по йоге?	Показать	Удалить
15.04.2026, 15:39:41	Гость: Цена	гость	Открыто	Сколько стоит разовое посещение трена...	Показать	Удалить
15.04.2026, 15:39:41	Вопрос #1: Абонемент	user1@local.test	Открыто	Подскажите, можно ли заморозить абон...	Показать	Удалить
15.04.2026, 15:39:41	Вопрос #2: Тренировка	user1@local.test	Открыто	Есть ли ограничения по уровню подгото...	Показать	Удалить
15.04.2026, 15:39:41	Вопрос #1: Абонемент	mail@mail.ru	Открыто	Подскажите, можно ли заморозить абон...	Показать	Удалить
15.04.2026, 15:39:41	Вопрос #2: Тренировка	mail@mail.ru	Открыто	Есть ли ограничения по уровню подгото...	Показать	Удалить

Рисунок 40 — Управление обращениями в административной панели

В таблице отображаются все обращения, поступившие от пользователей и гостей. Для каждого обращения указаны: дата и время создания, тема обращения, отправитель (email авторизованного пользователя или email, указанный гостем), статус обращения («Открыто» или «Закрыто»), текст сообщения и кнопка «Удалить». Текст сообщения отображается с обрезкой до 90 символов для экономии места, при этом полный текст доступен после нажатия кнопки «Показать». При нажатии на кнопку «Показать» под строкой с обращением разворачивается карточка с полным текстом сообщения, а кнопка меняет название на «Скрыть». Для удаления обращения администратор нажимает кнопку «Удалить» рядом с соответствующим обращением. Перед удалением программа отображает диалоговое окно с подтверждением «Удалить обращение?». При отсутствии обращений отображается сообщение «Сообщений пока нет». Результат удаления обращения представлен на рисунке 41.

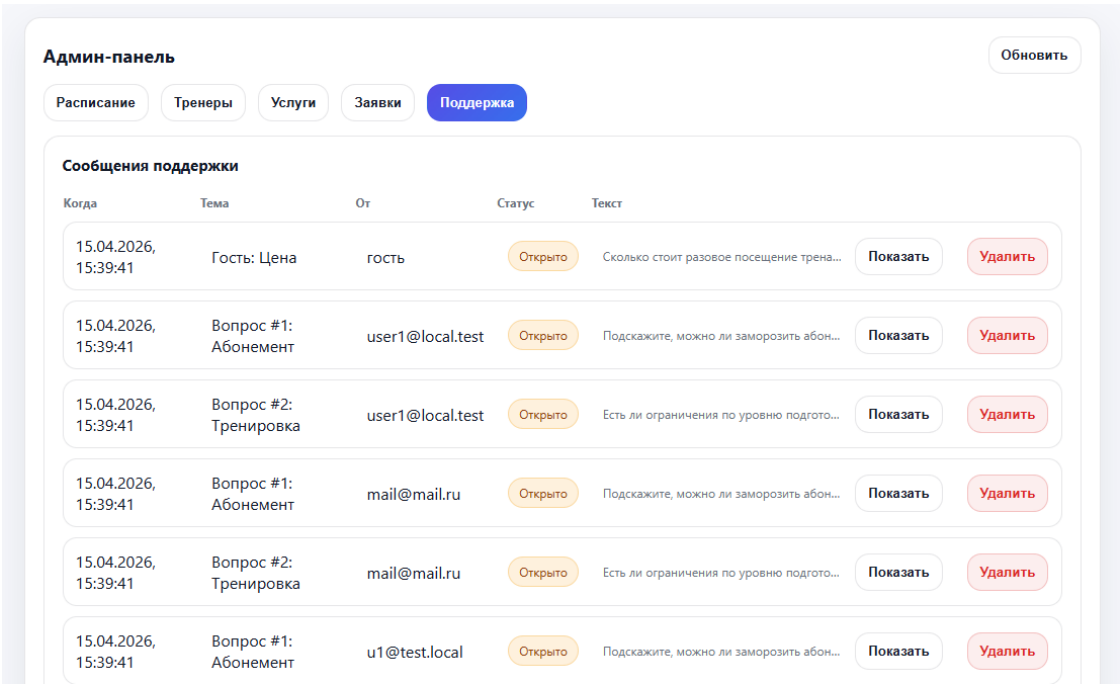


Рисунок 41 — Список обращений после удаления

Программа работает корректно и без технических сбоев.

2 Требования безопасности по охране труда при работе с ПК

Разработка программного обеспечения связана с длительной статической нагрузкой на мышцы, высокой зрительной нагрузкой и работой с электрооборудованием. Для сохранения здоровья разработчика и предотвращения профессиональных заболеваний необходимо соблюдать комплекс организационных, санитарно-гигиенических и технических мер.

2.1 Организация рабочего места и профилактика мышечного переутомления

Неправильная поза при длительной работе за компьютером приводит к развитию остеохондроза, синдрома запястного канала и болям в спине. Для профилактики костно-мышечного утомления рабочее место программиста организуется особым образом.

Используется компьютерный стул с регулируемой высотой сиденья, подлокотниками и поясничным подпором (валиком). Высота стула должна быть такой, чтобы стопы полностью стояли на полу, а угол в коленях составлял 90°. Стол должен иметь достаточную глубину для размещения монитора на расстоянии вытянутой руки [6].

Положение монитора и клавиатуры должно быть правильным. Верхний край экрана должен находиться на уровне глаз или чуть ниже, опущен на 15–20°. Клавиатура располагается на расстоянии 10–15 см от края стола, чтобы обеспечить опору для предплечий. Локти при работе с клавиатурой должны быть согнуты под углом 90°, запястья — прямые [6].

Обязательно делать динамические паузы. Каждые 45–60 минут работы необходимо вставать, выполнять наклоны, вращения головой и плечами. Рекомендуется менять позу и режим работы стоя, при наличии регулируемого стола.

					ПГУ1.090301.47.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		70

2.2 Профилактика зрительного утомления и требования к освещению

Глаза программиста испытывают нагрузку из-за мерцания экрана, мелкого шрифта и фокусировки на близком расстоянии. Основные меры защиты зрения:

- освещение: рабочее место должно быть расположено боком к окну. Искусственное освещение должно быть равномерным, комбинированным. Нормируемая освещенность на поверхности стола в зоне размещения документов — 300–500 лк. Категорически запрещается работать в темноте только при свете монитора;

- параметры экрана: яркость монитора должна соответствовать яркости окружающего освещения (не слишком тускло и не ослепляюще). Контрастность — максимально комфортная для чтения. Рекомендуется включить режим «тёплый свет» (синий свет уменьшает утомляемость, но ухудшает цветопередачу, поэтому настраивается индивидуально) [6];

- правило 20-20-20: каждые 20 минут отводить взгляд от монитора на 20 секунд, фокусируясь на предмете, удалённом на 6 метров (20 футов). Это снижает спазм аккомодации [6];

- упражнения для глаз: круговые движения, частое моргание, перевод взгляда с ближней точки на дальнюю.

2.3 Пожарная безопасность

Помещения, где ведётся разработка ПО, относятся к категории пожароопасности В пожароопасные по НПБ 105-03, так как содержат электрооборудование, горючие материалы. Основные требования:

- причины пожара: обычно это короткое замыкание, перегрузка сети, при включении многих приборов в один удлинитель, перегрев блока питания, неисправность розеток;

– меры предотвращения: запрещается использовать самодельные и повреждённые удлинители, оставлять включённое оборудование без присмотра (кроме серверов), закрывать вентиляционные отверстия системного блока;

– средства тушения: в помещении должен быть углекислотный или порошковый огнетушитель (марки ОУ или ОП). Вода и пенные огнетушители для тушения электроустановок под напряжением запрещены;

– план действий: при запахе гари обесточить оборудование, вызвать пожарную охрану, приступить к тушению небольших очагов огнетушителем. Эвакуация через основные и запасные выходы.

2.4 Санитарно-гигиенические требования и организационные мероприятия

Данный раздел регламентирует микроклимат, режим труда и электробезопасность:

– микроклимат: по СанПиН 1.2.3685-21 для помещений с ПК температура в тёплый период — 23–25 °С, в холодный — 22–24 °С. Относительная влажность воздуха — 40–60% [7];

– электробезопасность: все розетки должны иметь заземляющий контакт. питание компьютера осуществляется через сетевые фильтры (источники бесперебойного питания защищают не только от скачков, но и от потери данных). Запрещается работать с открытыми крышками системного блока [7];

– режим труда и отдыха: установлен регламентированный перерыв 10–15 минут через каждый час работы. Общая продолжительность работы за компьютером за смену не должна превышать 6 часов (при творческой деятельности — до 8 часов с перерывами) [7];

– гигиена: проводится влажная уборка помещения не реже 1 раза в день, экран монитора протирается специальными салфетками без спирта для удаления пыли и статического заряда.

					ПГУ1.090301.47.001 ПЗ	Лист
						73
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

Разработанное в рамках выпускной квалификационной работы веб-приложение для фитнес-центра представляет собой современное и востребованное решение в области автоматизации спортивного бизнеса. В условиях современного общества, где время клиента становится главной ценностью, а конкуренция между фитнес-клубами постоянно растёт, подобное приложение может стать надёжным инструментом для повышения лояльности посетителей и оптимизации работы административного персонала.

Ожидается, что внедрение разработанной системы не только упростит работу с большим потоком клиентов и снизит нагрузку на ресепшн, но и позволит руководству фитнес-центра получать наглядную аналитику о посещаемости, популярности различных групп занятий и загрузке тренеров. Система способна показать те нюансы в организации работы, над которыми стоит поработать для увеличения прибыли и удержания клиентов.

В дальнейшем проект может быть расширен путём добавления дополнительных функций, например, интеграции с системами оплаты и абонеменами, push-уведомлений об отмене или переносе тренировки, личного кабинета с историей посещений и прогрессом в тренировках и усложнения логики их взаимодействия.

					ПГУ1.090301.47.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		74

Список использованных источников

1. FitBase [Электронный ресурс] URL: <https://fitbase.io/> (дата обращения 15.04.2026)
2. Фитнес клуб XFIT . [Электронный ресурс] Режим доступа: <https://www.xfit.ru/> свободный. (дата обращения: 15.04.2026).
3. Фитнес-клубы World Class [Электронныйресурс] / Режим доступа: <https://www.worldclass.ru/> свободный (дата обращения: 15.04.2026).
4. Goldberg, Josh. Learning TypeScript. " O'Reilly Media, Inc.", 2022. [Электронный ресурс]. URL: <https://books.google.com/books?hl=vi&lr=&id=YD5zEAAAQBAJ&oi=fnd&pg=PP1&dq=TypeScript+&ots=FG46DpkGo8&sig=uiNd76XisRU-q7jNqscb7KWDGLU> (дата обращения 02.05.2026)
5. Visual Studio Code [Электронный ресурс]. URL: <https://code.visualstudio.com/> (дата обращения 05.05.2026)
6. Яковлева, Е. В., and В. Е. Семенов. "Здоровьесберегающие технологии офисных работников." Вестник техносферной безопасности и сельского развития 2 (41) (2025): 17-20. [Электронный ресурс] Режим доступа: <https://cyberleninka.ru/article/n/zdoroviesberegayuschie-tehnologii-ofisnyh-rabotnikov> свободный. (дата обращения: 15.05.2026).
7. Федорович, Г. В. "СанПиН 1.2. 3685-21-стартовая позиция нормотворческой работы." Безопасность и охрана труда 1 (2021): 31-38. [Электронный ресурс] / Режим доступа: <https://elibrary.ru/item.asp?id=45630856> свободный. (дата обращения: 18.05.2026)

Приложение А

(обязательное)

Веб-приложение фитнес-центра

UML диаграммы

					ПГУ1.090301.47.001 ПЗ	Лист
						76
Изм.	Лист	№ докум.	Подпись	Дата		



UML – Диаграмма прецедентов авторизованного пользователя



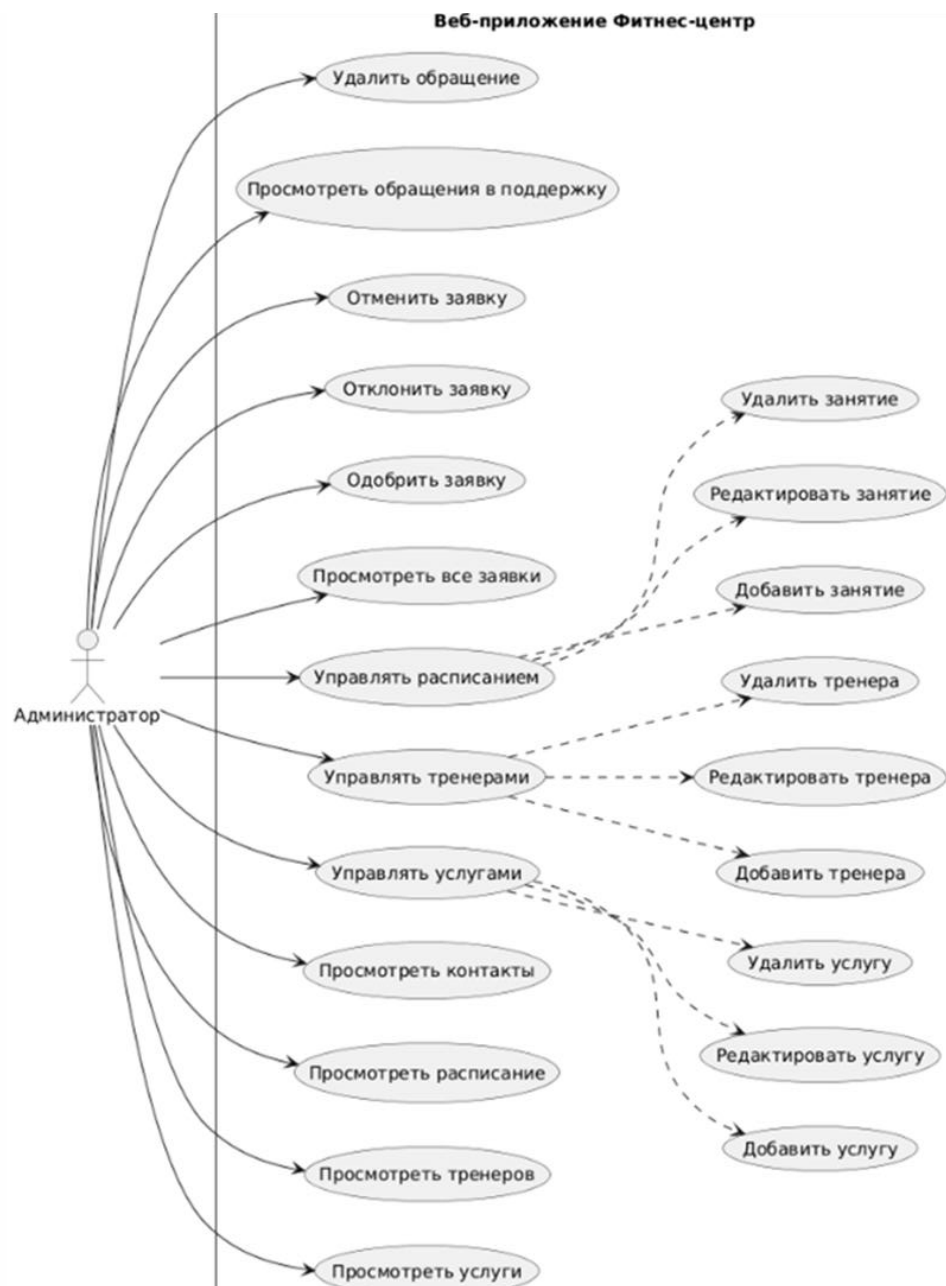
UML - Диаграмма прецедентов администратора

Изм.	Лист	№ докум.	Подпись	Дата

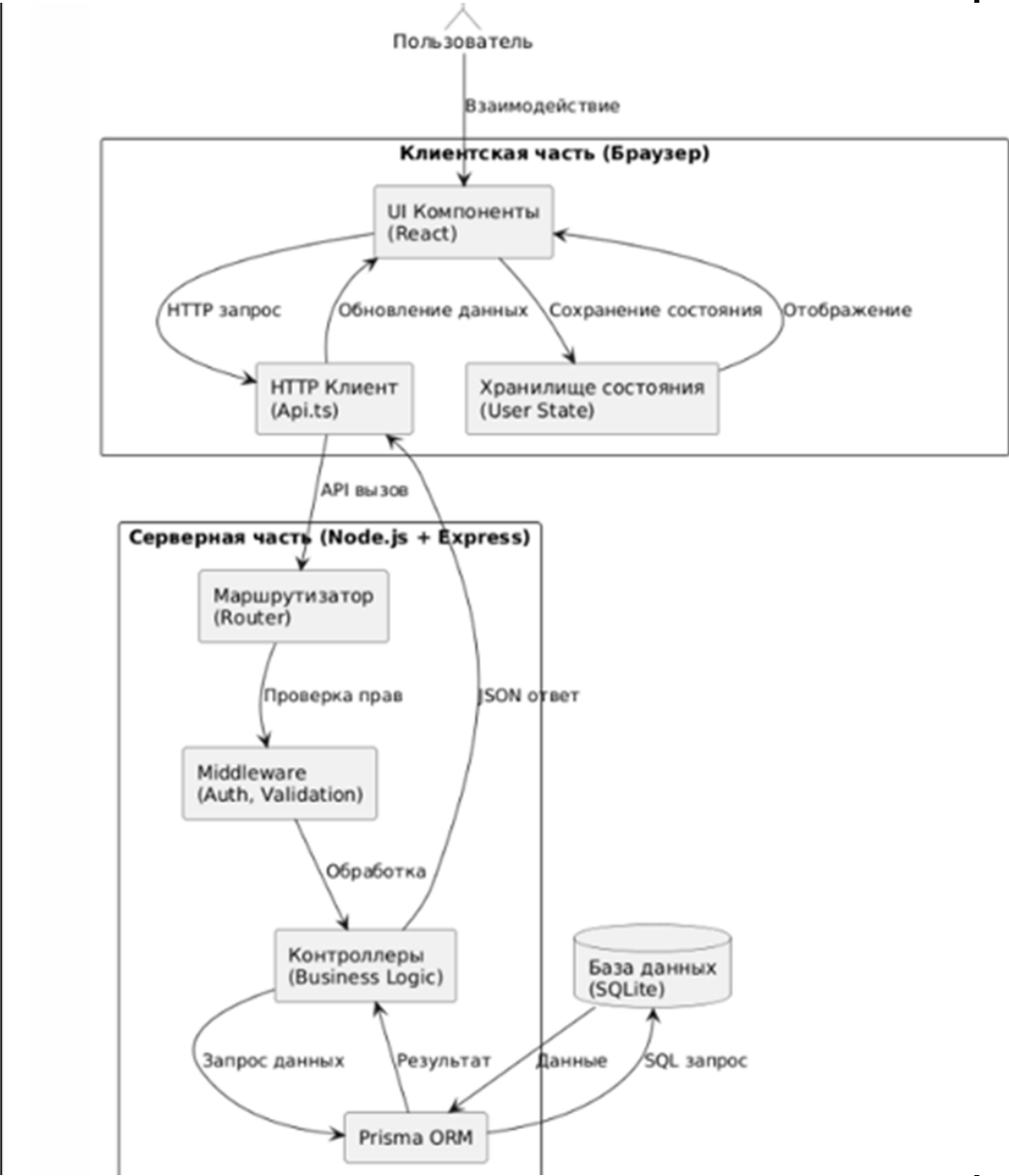
ПГУ1.090301.47.001 ПЗ

Лист

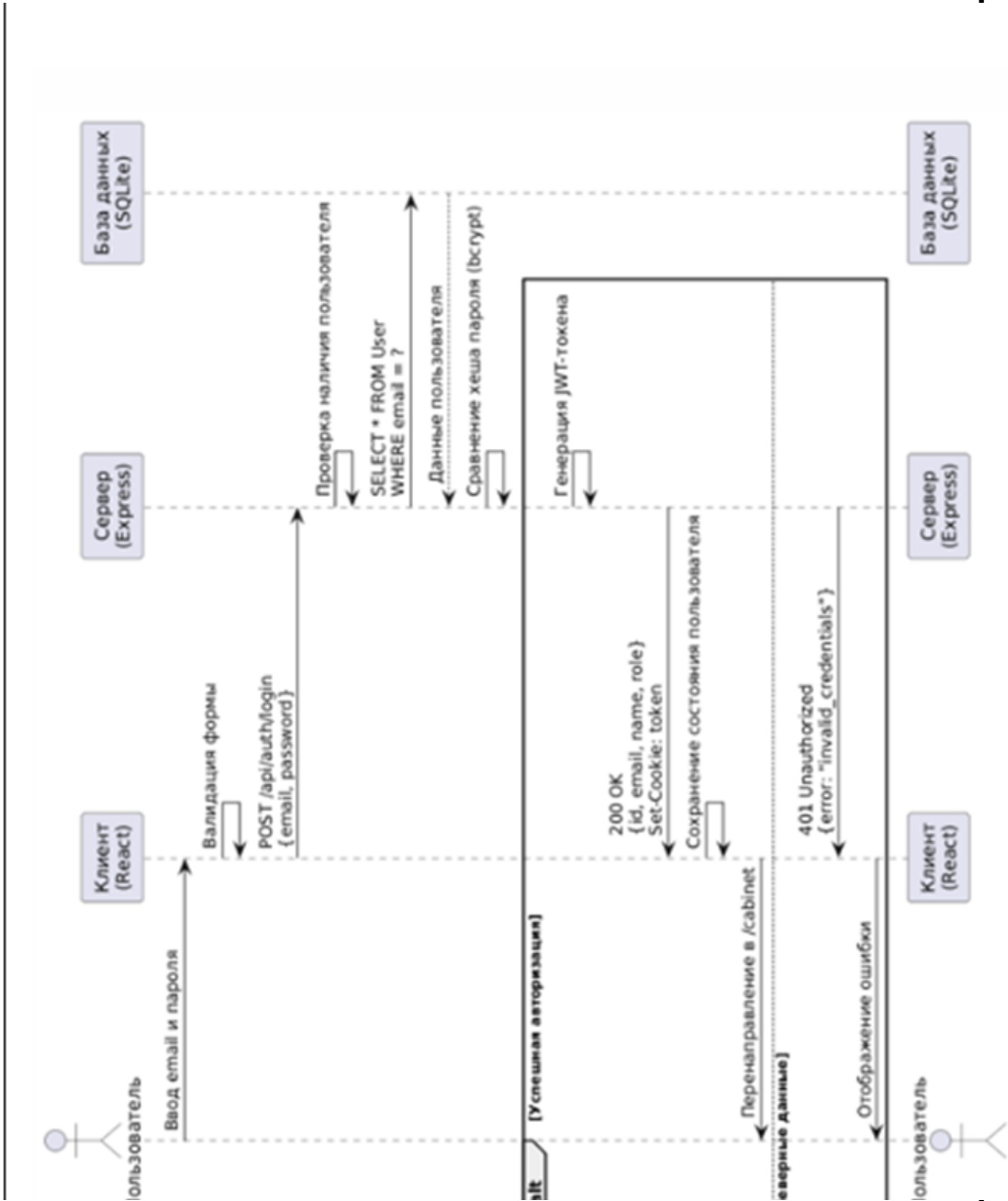
78



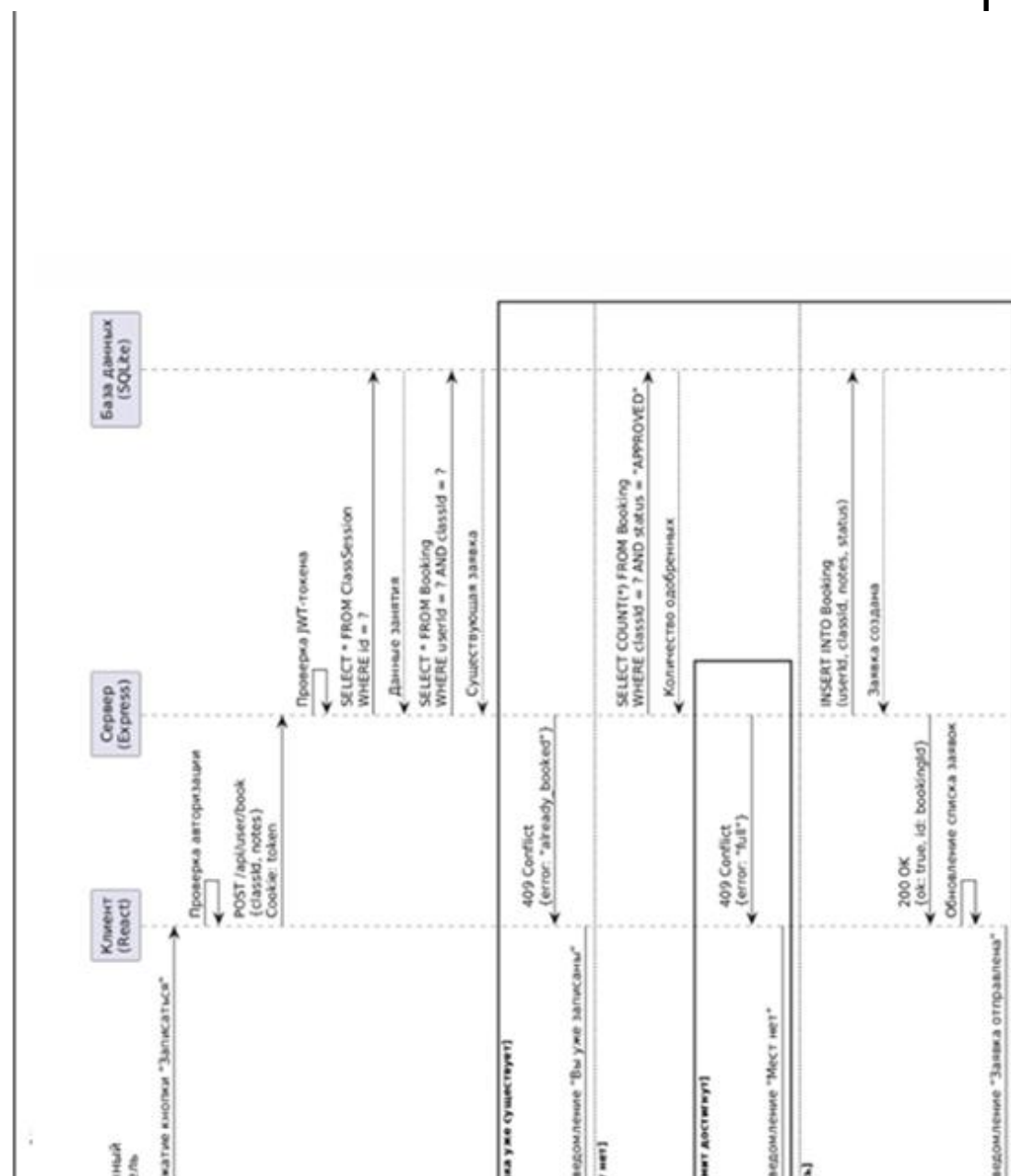
UML – Диаграмма функционирования работы преокта



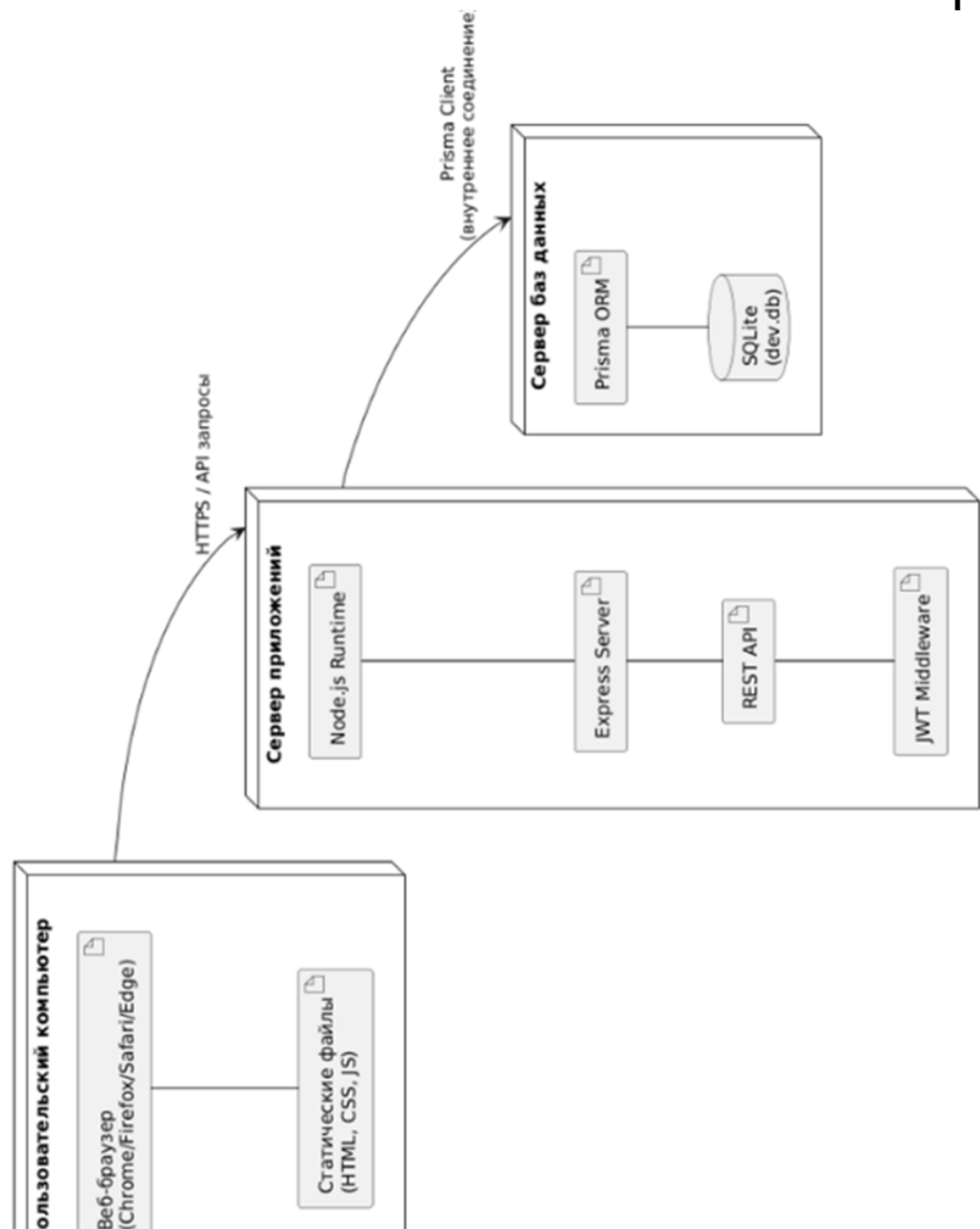
UML – Диаграмма последовательностей «Авторизация»



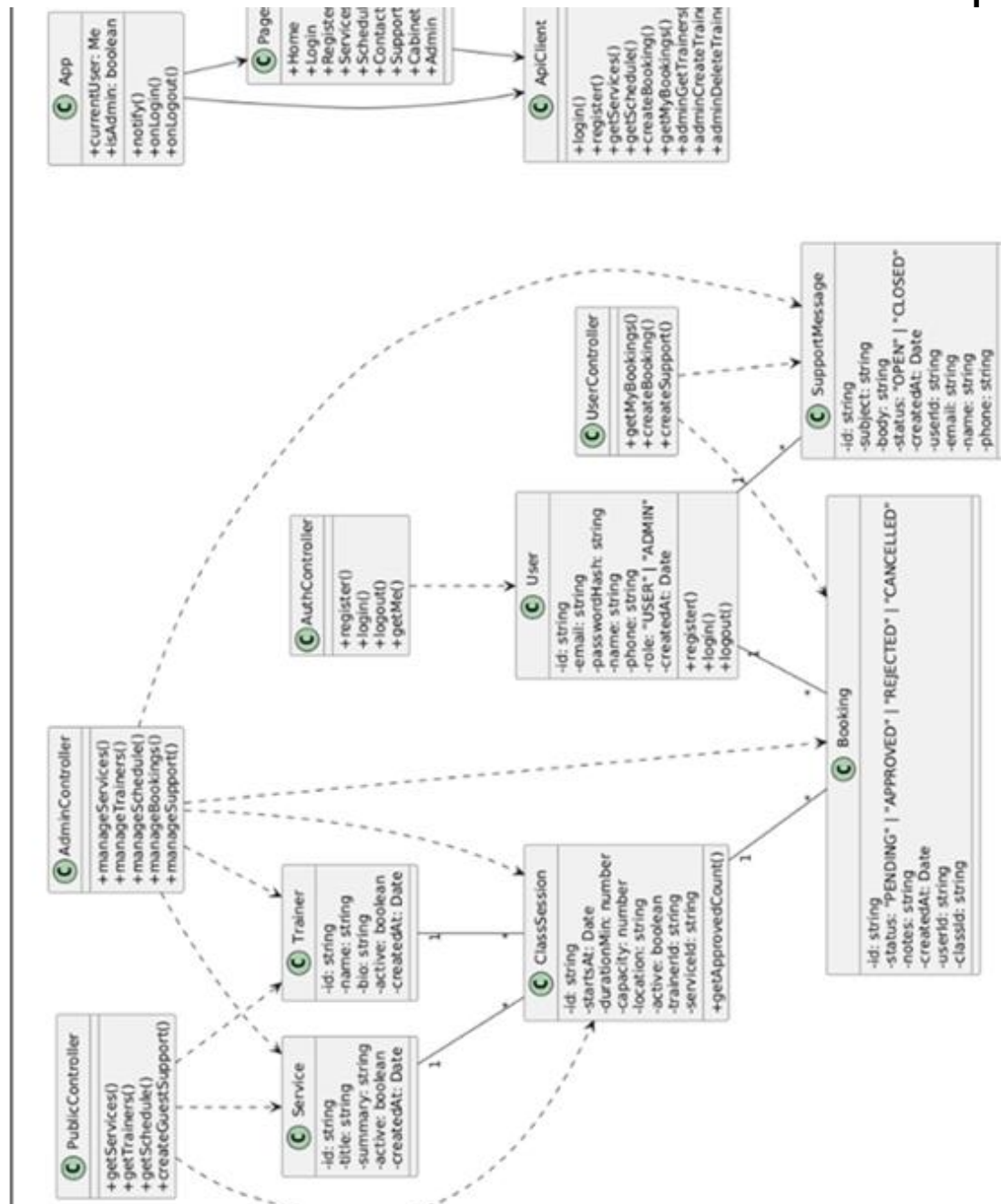
UML – Диаграмма последовательностей «Запись на занятие»



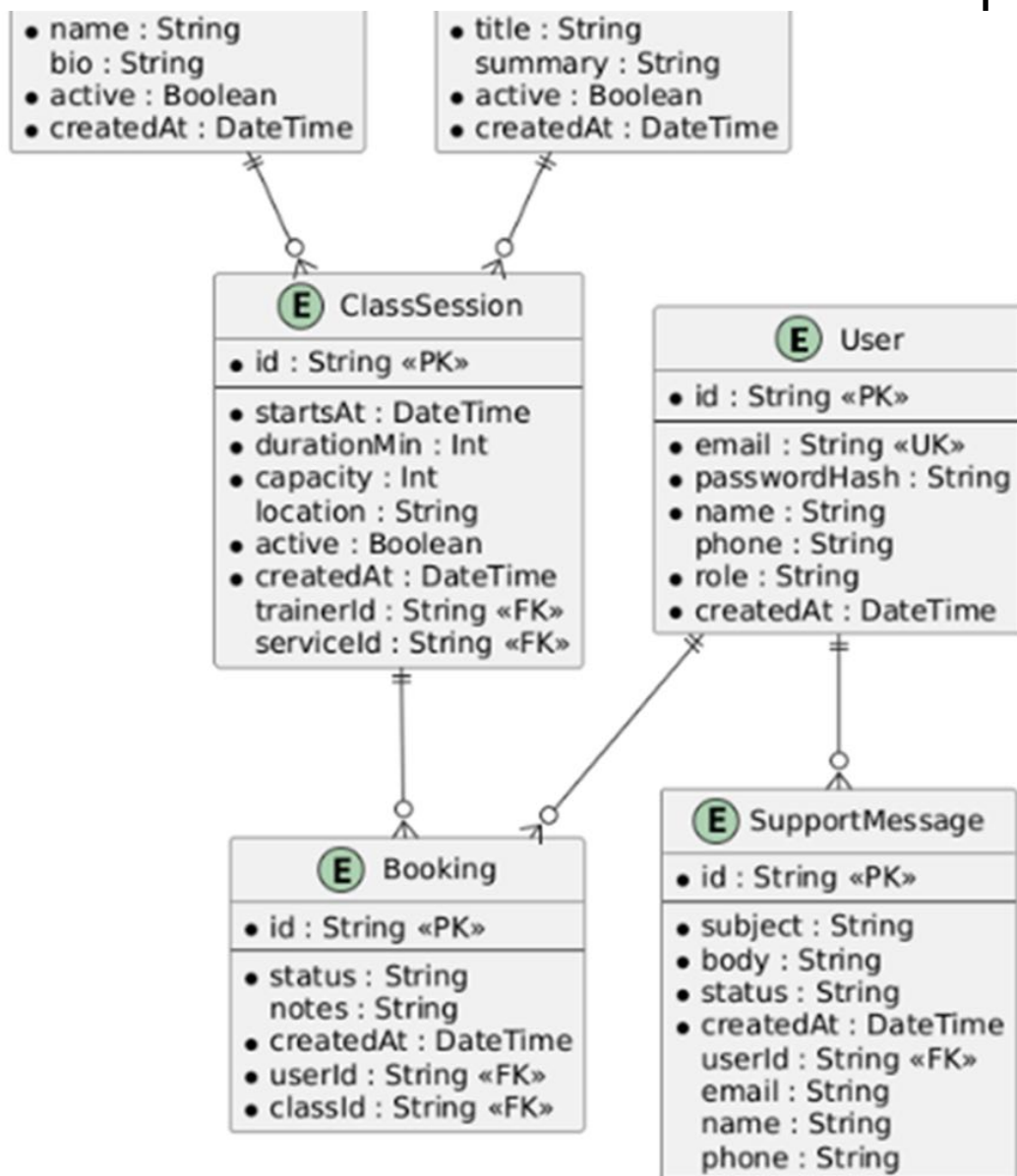
UML – Диаграмма развертывания приложения



UML — Диаграмма развертывания приложения



UML — Диаграмма развертывания приложения



Приложение Б

(обязательное)

«Веб-приложение фитнес-центра»

Листинг программы

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.47.001 ПЗ

Лист

85

Файл «schema.prisma»

```

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "sqlite"
  url      = env("DATABASE_URL")
}

enum UserRole {
  USER
  ADMIN
}

model User {
  id          String    @id @default(cuid())
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt

  role        UserRole  @default(USER)
  email        String    @unique
  passwordHash String
  name         String
  phone        String?

  bookings    Booking[]
  messages    SupportMessage[]
}

model Trainer {
  id          String    @id @default(cuid())
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt

  name        String
  bio         String?
  active       Boolean   @default(true)

  classes     ClassSession[]
}

model Service {
  id          String    @id @default(cuid())
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt

  title       String
  summary     String?
  active       Boolean   @default(true)

```

```

    classes    ClassSession[]
}

model ClassSession {
    id          String    @id @default(cuid())
    createdAt   DateTime  @default(now())
    updatedAt   DateTime  @updatedAt

    startsAt    DateTime
    durationMin Int
    capacity     Int      @default(12)
    location     String?
    active       Boolean   @default(true)

    trainerId   String?
    trainer      Trainer? @relation(fields: [trainerId],
references: [id], onDelete: SetNull)

    serviceId   String?
    service      Service? @relation(fields: [serviceId],
references: [id], onDelete: SetNull)

    bookings    Booking[]

    @@index([startsAt])
}

model Booking {
    id          String    @id @default(cuid())
    createdAt   DateTime  @default(now())
    updatedAt   DateTime  @updatedAt

    status      String    @default("PENDING")
    notes       String?

    userId      String
    user         User      @relation(fields: [userId], references:
[id], onDelete: Cascade)

    classId     String
    class        ClassSession @relation(fields: [classId],
references: [id], onDelete: Cascade)

    @@unique([userId, classId])
}

model SupportMessage {
    id          String    @id @default(cuid())
    createdAt   DateTime  @default(now())

    subject     String

```

```

body      String
status    String    @default("OPEN")

userId     String?
user       User?     @relation(fields: [userId], references:
[id], onDelete: SetNull)

email      String?
name       String?
phone      String?
}

```

Файл «seed.ts»

```

import bcrypt from "bcryptjs";
import { prisma } from "../lib/prisma";

function addDays(d: Date, days: number) {
  const x = new Date(d);
  x.setDate(x.getDate() + days);
  return x;
}

function atTime(base: Date, hh: number, mm: number) {
  const x = new Date(base);
  x.setHours(hh, mm, 0, 0);
  return x;
}

async function main() {
  const adminEmail = "admin@local.test";
  const adminPass = "adminadmin";

  const existing = await prisma.user.findUnique({ where: {
email: adminEmail } });
  if (!existing) {
    const passwordHash = await bcrypt.hash(adminPass, 12);
    await prisma.user.create({
      data: {
        email: adminEmail,
        passwordHash,
        name: "Admin",
        role: "ADMIN"
      }
    });
  }

  // Services
  const servicesToEnsure = [
    { title: "Функциональная тренировка", summary: "Сила,
выносливость, координация." },

```

```

    { title: "Йога", summary: "Гибкость, дыхание, баланс." },
    { title: "Пилатес", summary: "Кор, осанка, контроль
движения." },
    { title: "Кросс-тренинг", summary: "Интенсивные интервальные
тренировки." },
    { title: "Стретчинг", summary: "Растяжка и восстановление."
},
    { title: "Здоровая спина", summary: "Мобилити + укрепление
мышц кора." },
    { title: "TRX", summary: "Функциональные тренировки на
петлях." },
    { title: "Силовая (группа)", summary: "Базовые упражнения,
техника, прогрессия." }
  ] as const;

  for (const s of servicesToEnsure) {
    const exists = await prisma.service.findFirst({ where: {
title: s.title } });
    if (!exists) await prisma.service.create({ data: { ...s,
active: true } });
  }

  // Trainers
  const trainersToEnsure = [
    { name: "Анна Петрова", bio: "Йога, пилатес. 6 лет опыта."
},
    { name: "Илья Смирнов", bio: "Силовые и функциональные. 8
лет опыта." },
    { name: "Мария Кузнецова", bio: "Стретчинг, восстановление,
mobility." },
    { name: "Дмитрий Орлов", bio: "TRX, кросс-тренинг,
интервалы." },
    { name: "Екатерина Лебедева", bio: "Здоровая спина, пилатес,
осанка." }
  ] as const;

  for (const t of trainersToEnsure) {
    const exists = await prisma.trainer.findFirst({ where: {
name: t.name } });
    if (!exists) await prisma.trainer.create({ data: { ...t,
active: true } });
  }

  // Demo users (besides admin)
  const demoUsers = [
    { email: "user1@local.test", name: "Иван", phone:
"+79990000001" },
    { email: "user2@local.test", name: "Ольга", phone:
"+79990000002" },
    { email: "user3@local.test", name: "Никита", phone:
"+79990000003" }
  ]

```

```

    ] as const;
    const demoPass = "password1";
    for (const u of demoUsers) {
        const exists = await prisma.user.findUnique({ where: {
            email: u.email } });
        if (!exists) {
            const passwordHash = await bcrypt.hash(demoPass, 12);
            await prisma.user.create({ data: { ...u, passwordHash,
            role: "USER" } });
        }
    }

    // Schedule generation (next 21 days)
    const services = await prisma.service.findMany({ where: {
        active: true } });
    const trainers = await prisma.trainer.findMany({ where: {
        active: true } });
    if (services.length > 0) {
        const now = new Date();
        const base = new Date(now.getFullYear(), now.getMonth(),
        now.getDate(), 0, 0, 0);

        const timeSlots = [
            { hh: 7, mm: 30 },
            { hh: 9, mm: 0 },
            { hh: 12, mm: 30 },
            { hh: 18, mm: 0 },
            { hh: 19, mm: 30 },
            { hh: 21, mm: 0 }
        ];

        const locations = ["Зал А", "Зал В", "Студия",
        "Функциональная зона"];

        let created = 0;
        for (let day = 0; day < 21; day++) {
            const d = addDays(base, day);
            const weekday = d.getDay(); // 0 Sun .. 6 Sat
            // slightly fewer on weekends
            const slotsToday = weekday === 0 ? timeSlots.slice(2, 5) :
            weekday === 6 ? timeSlots.slice(1, 5) : timeSlots;

            for (let i = 0; i < slotsToday.length; i++) {
                // deterministic-ish pick
                const svc = services[(day * 3 + i) % services.length];
                const tr = trainers.length ? trainers[(day + i) %
                trainers.length] : null;
                const startsAt = atTime(d, slotsToday[i].hh,
                slotsToday[i].mm);
                const durationMin = svc.title.includes("Йога") ||
                svc.title.includes("Пилатес") ? 75 : 60;
                const capacity = svc.title.includes("TRX") ? 10 :
                svc.title.includes("Силовая") ? 14 : 12;
            }
        }
    }

```

```

    const location = locations[(day + i) %
locations.length];

    const exists = await prisma.classSession.findFirst({
      where: {
        startsAt,
        serviceId: svc.id
      }
    });
    if (exists) continue;

    await prisma.classSession.create({
      data: {
        startsAt,
        durationMin,
        capacity,
        location,
        active: true,
        serviceId: svc.id,
        trainerId: tr?.id ?? null
      }
    });
    created++;
  }
}

// Add some bookings + support messages to make it look real
const users = await prisma.user.findMany({ where: { role:
"USER" } });
const classes = await prisma.classSession.findMany({
  where: { startsAt: { gte: base } },
  orderBy: { startsAt: "asc" }
});

// A few support messages (idempotent via subject+createdAt
isn't reliable -> check by subject prefix + user)
for (const u of users.slice(0, 3)) {
  const already = await prisma.supportMessage.findFirst({
    where: { userId: u.id, subject: { startsWith: "Вопрос #"
  } }
});
  if (!already) {
    await prisma.supportMessage.createMany({
      data: [
        {
          userId: u.id, subject: "Вопрос #1: Абонемент",
body: "Подскажите, можно ли заморозить абонемент на 2 недели?"
        },
        {
          userId: u.id, subject: "Вопрос #2: Тренировка",
body: "Есть ли ограничения по уровню подготовки для TRX?"
        }
      ]
    });
  }
}

```

```

    }

    // Create some bookings if not present
    // Make some APPROVED so that occupancy in schedule looks
    alive.
    for (let idx = 0; idx < Math.min(classes.length, 18); idx++)
    {
        const cls = classes[idx];
        const u = users[idx % users.length];
        const exists = await prisma.booking.findFirst({ where: {
        userId: u.id, classId: cls.id } });
        if (exists) continue;
        await prisma.booking.create({
            data: {
                userId: u.id,
                classId: cls.id,
                status: idx % 3 === 0 ? "PENDING" : "APPROVED",
                notes: idx % 4 === 0 ? "Хочу место ближе к зеркалу" :
null
            }
        });
    }

    // Add some guest support messages too
    const guestExists = await prisma.supportMessage.findFirst({
    where: { userId: null, subject: { startsWith: "Гость:" } } });
    if (!guestExists) {
        await prisma.supportMessage.createMany({
            data: [
                { subject: "Гость: Вопрос по расписанию", body: "Когда
будут утренние занятия по йоге?", email: "guest1@example.com",
name: "Алексей" },
                { subject: "Гость: Цена", body: "Сколько стоит разовое
посещение тренажерного зала?", phone: "+79990000099", name:
"Светлана" }
            ]
        });
    }

    // eslint-disable-next-line no-console
    console.log(`Seeded: created classes=${created} (if new)`);
}

main()
    .then(async () => {
        await prisma.$disconnect();
    })
    .catch(async (e) => {
        // eslint-disable-next-line no-console
        console.error(e);
        await prisma.$disconnect();
    });

```

```

    process.exit(1);
  });

```

Файл «server.ts»

```

import express from "express";
import cors from "cors";
import cookieParser from "cookie-parser";
import { env } from "../lib/env";
import { authRouter } from "../routes/auth";
import { publicRouter } from "../routes/public";
import { userRouter } from "../routes/user";
import { adminRouter } from "../routes/admin";

const app = express();

app.use(
  cors({
    origin: env.APP_ORIGIN,
    credentials: true
  })
);

app.use(express.json({ limit: "1mb" }));
app.use(cookieParser());

app.get("/health", (_req, res) => res.json({ ok: true }));

app.use("/api/auth", authRouter);
app.use("/api/public", publicRouter);
app.use("/api/user", userRouter);

```

```

app.use("/api/admin", adminRouter);

app.listen(env.PORT, () => {
  // eslint-disable-next-line no-console
  console.log(`API listening on http://localhost:${env.PORT}`);
});

```

Файл «admin.ts»

```

import { Router } from "express";
import { z } from "zod";
import { prisma } from "../lib/prisma";
import { requireAdmin } from "../middleware/requireAuth";

export const adminRouter = Router();
adminRouter.use(requireAdmin);

adminRouter.get("/support", async (_req, res) => {
  const msgs = await prisma.supportMessage.findMany({
    orderBy: { createdAt: "desc" },
    include: { user: { select: { id: true, email: true, name: true } } }
  });
  return res.json(msgs);
});

adminRouter.delete("/support/:id", async (req, res) => {
  await prisma.supportMessage.delete({ where: { id: req.params.id } });
  return res.json({ ok: true });
});

```

```

});

adminRouter.get("/bookings", async (_req, res) => {
  const bookings = await prisma.booking.findMany({
    orderBy: { createdAt: "desc" },
    include: {
      user: { select: { id: true, email: true, name: true } },
      class: { include: { trainer: true, service: true } }
    }
  });
  return res.json(bookings);
});

adminRouter.patch("/bookings/:id", async (req, res) => {
  const schema = z.object({
    status: z.enum(["PENDING", "APPROVED", "REJECTED",
"CANCELLED"])
  });

  const parsed = schema.safeParse(req.body);

  if (!parsed.success) return res.status(400).json({ error:
"invalid_input" });

  const booking = await prisma.booking.findUnique({
    where: { id: req.params.id },
    include: { class: true }
  });

  if (!booking) return res.status(404).json({ error: "not_found"
});

```

```

    if (parsed.data.status === "APPROVED") {

        const approved = await prisma.booking.count({

            where: { classId: booking.classId, status: "APPROVED",
NOT: { id: booking.id } }

        });

        if (approved >= booking.class.capacity) return
res.status(409).json({ error: "full" });

    }

    const updated = await prisma.booking.update({

        where: { id: booking.id },

        data: { status: parsed.data.status }

    });

    return res.json(updated);

});

adminRouter.get("/trainers", async (_req, res) => {

    const trainers = await prisma.trainer.findMany({ orderBy: {
createdAt: "desc" } });

    return res.json(trainers);

});

adminRouter.post("/trainers", async (req, res) => {

    const schema = z.object({ name: z.string().min(1), bio:
z.string().optional(), active: z.boolean().optional() });

    const parsed = schema.safeParse(req.body);

    if (!parsed.success) return res.status(400).json({ error:
"invalid_input" });

    const trainer = await prisma.trainer.create({ data: {
...parsed.data, active: parsed.data.active ?? true } });

```

```

    return res.json(trainer);
});

adminRouter.patch("/trainers/:id", async (req, res) => {

    const schema = z.object({ name: z.string().min(1).optional(),
    bio: z.string().optional(), active: z.boolean().optional() });

    const parsed = schema.safeParse(req.body);

    if (!parsed.success) return res.status(400).json({ error:
    "invalid_input" });

    const trainer = await prisma.trainer.update({ where: { id:
    req.params.id }, data: parsed.data });

    return res.json(trainer);
});

adminRouter.delete("/trainers/:id", async (req, res) => {

    await prisma.trainer.delete({ where: { id: req.params.id } });

    return res.json({ ok: true });
});

adminRouter.get("/services", async (_req, res) => {

    const services = await prisma.service.findMany({ orderBy: {
    createdAt: "desc" } });

    return res.json(services);
});

adminRouter.post("/services", async (req, res) => {

    const schema = z.object({ title: z.string().min(1), summary:
    z.string().optional(), active: z.boolean().optional() });

    const parsed = schema.safeParse(req.body);

```

					ПГУ1.090301.47.001 ПЗ	Лист
						97
Изм.	Лист	№ докум.	Подпись	Дата		

```

    if (!parsed.success) return res.status(400).json({ error:
"invalid_input" });

    const service = await prisma.service.create({ data: {
...parsed.data, active: parsed.data.active ?? true } });

    return res.json(service);

});

adminRouter.patch("/services/:id", async (req, res) => {

    const schema = z.object({ title: z.string().min(1).optional(),
summary: z.string().optional(), active: z.boolean().optional()
});

    const parsed = schema.safeParse(req.body);

    if (!parsed.success) return res.status(400).json({ error:
"invalid_input" });

    const service = await prisma.service.update({ where: { id:
req.params.id }, data: parsed.data });

    return res.json(service);

});

adminRouter.delete("/services/:id", async (req, res) => {

    await prisma.service.delete({ where: { id: req.params.id } });

    return res.json({ ok: true });

});

adminRouter.get("/classes", async (_req, res) => {

    const classes = await prisma.classSession.findMany({

        orderBy: { startsAt: "asc" },

        include: { trainer: true, service: true }

    });

```

```

const ids = classes.map((c) => c.id);

const approvedCounts = await prisma.booking.groupBy({
  by: ["classId"],
  where: { classId: { in: ids }, status: "APPROVED" },
  _count: { _all: true }
});

const m = new Map(approvedCounts.map((r) => [r.classId,
r._count._all]));

return res.json(
  classes.map((c) => ({
    ...c,
    approvedCount: m.get(c.id) ?? 0
  }))
);
});

adminRouter.post("/classes", async (req, res) => {
  const schema = z.object({
    startsAt: z.string().datetime(),
    durationMin: z.number().int().positive(),
    capacity: z.number().int().positive().optional(),
    location: z.string().optional(),
    active: z.boolean().optional(),
    trainerId: z.string().optional().nullable(),
    serviceId: z.string().optional().nullable()
  });

  const parsed = schema.safeParse(req.body);

  if (!parsed.success) return res.status(400).json({ error:
  "invalid_input" });

```

```

const cls = await prisma.classSession.create({
  data: {
    startsAt: new Date(parsed.data.startsAt),
    durationMin: parsed.data.durationMin,
    capacity: parsed.data.capacity ?? 12,
    location: parsed.data.location,
    active: parsed.data.active ?? true,
    trainerId: parsed.data.trainerId ?? null,
    serviceId: parsed.data.serviceId ?? null
  }
});

return res.json(cls);
});

adminRouter.patch("/classes/:id", async (req, res) => {
  const schema = z.object({
    startsAt: z.string().datetime().optional(),
    durationMin: z.number().int().positive().optional(),
    capacity: z.number().int().positive().optional(),
    location: z.string().optional(),
    active: z.boolean().optional(),
    trainerId: z.string().optional().nullable(),
    serviceId: z.string().optional().nullable()
  });

  const parsed = schema.safeParse(req.body);

  if (!parsed.success) return res.status(400).json({ error:
"invalid_input" });

```

					ПГУ1.090301.47.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		100

```

const cls = await prisma.classSession.update({
  where: { id: req.params.id },
  data: {
    ...parsed.data,
    ...(parsed.data.startsAt ? { startsAt: new
Date(parsed.data.startsAt) } : {})
  }
});

return res.json(cls);
});

adminRouter.delete("/classes/:id", async (req, res) => {
  await prisma.classSession.delete({ where: { id: req.params.id
} });
  return res.json({ ok: true });
});

```

Файл «auth.ts»

```

import { Router } from "express";
import bcrypt from "bcryptjs";
import { z } from "zod";
import { prisma } from "../lib/prisma";
import { env } from "../lib/env";
import { signSessionToken } from "../lib/auth";
import { requireAuth, type AuthedRequest } from
"../middleware/requireAuth";

export const authRouter = Router();

```

```

const registerSchema = z.object({
  email: z.string().email(),
  password: z.string().min(8),
  name: z.string().min(1),
  phone: z.string().min(5).optional()
});

authRouter.post("/register", async (req, res) => {
  const parsed = registerSchema.safeParse(req.body);

  if (!parsed.success) return res.status(400).json({ error:
"invalid_input" });

  const { email, password, name, phone } = parsed.data;

  const exists = await prisma.user.findUnique({ where: { email }
});

  if (exists) return res.status(409).json({ error: "email_taken"
});

  const passwordHash = await bcrypt.hash(password, 12);

  const user = await prisma.user.create({
    data: { email, passwordHash, name, phone }
  });

  const token = signSessionToken({ sub: user.id, role: user.role
});

  res.cookie(env.COOKIE_NAME, token, {
    httpOnly: true,
    sameSite: "lax",
    secure: env.COOKIE_SECURE

```

```

    });

    return res.json({ id: user.id, role: user.role, email:
user.email, name: user.name });

});

const loginSchema = z.object({
  email: z.string().email(),
  password: z.string().min(1)
});

authRouter.post("/login", async (req, res) => {
  const parsed = loginSchema.safeParse(req.body);

  if (!parsed.success) return res.status(400).json({ error:
"invalid_input" });

  const { email, password } = parsed.data;

  const user = await prisma.user.findUnique({ where: { email }
});

  if (!user) return res.status(401).json({ error:
"invalid_credentials" });

  const ok = await bcrypt.compare(password, user.passwordHash);

  if (!ok) return res.status(401).json({ error:
"invalid_credentials" });

  const token = signSessionToken({ sub: user.id, role: user.role
});

  res.cookie(env.COOKIE_NAME, token, {
    httpOnly: true,

```

```

        sameSite: "lax",

        secure: env.COOKIE_SECURE

    });

    return res.json({ id: user.id, role: user.role, email:
user.email, name: user.name });

});

authRouter.post("/logout", async (_req, res) => {

    res.clearCookie(env.COOKIE_NAME);

    return res.json({ ok: true });

});

authRouter.get("/me", requireAuth, async (req: AuthedRequest,
res) => {

    const user = await prisma.user.findUnique({

        where: { id: req.user!.id },

        select: { id: true, role: true, email: true, name: true,
phone: true, createdAt: true }

    });

    return res.json(user);

});

```

Файл «public.ts»

```

import { Router } from "express";

import { z } from "zod";

import { prisma } from "../lib/prisma";

export const publicRouter = Router();

```

```

publicRouter.get("/services", async (_req, res) => {
    const services = await prisma.service.findMany({
        where: { active: true },
        orderBy: { title: "asc" }
    });
    return res.json(services);
});

publicRouter.get("/trainers", async (_req, res) => {
    const trainers = await prisma.trainer.findMany({
        where: { active: true },
        orderBy: { name: "asc" }
    });
    return res.json(trainers);
});

publicRouter.get("/schedule", async (req, res) => {
    const from = typeof req.query.from === "string" ? new
Date(req.query.from) : undefined;

    const to = typeof req.query.to === "string" ? new
Date(req.query.to) : undefined;

    const where: Record<string, unknown> = { active: true };
    if (from || to) {
        where.startsAt = {
            ...(from ? { gte: from } : {}),
            ...(to ? { lte: to } : {})
        };
    }
}

```

```

const classes = await prisma.classSession.findMany({
  where,
  orderBy: { startsAt: "asc" },
  include: { trainer: true, service: true }
});

const ids = classes.map((c) => c.id);

const approvedCounts = await prisma.booking.groupBy({
  by: ["classId"],
  where: { classId: { in: ids }, status: "APPROVED" },
  _count: { _all: true }
});

const m = new Map(approvedCounts.map((r) => [r.classId,
r._count._all]));

return res.json(
  classes.map((c) => ({
    ...c,
    approvedCount: m.get(c.id) ?? 0
  }))
);
});

const supportSchema = z.object({
  subject: z.string().min(1),
  body: z.string().min(1),
  email: z.string().email().optional(),

```

```

    name: z.string().min(1).optional(),
    phone: z.string().min(5).optional()
  });

publicRouter.post("/support", async (req, res) => {
  const parsed = supportSchema.safeParse(req.body);

  if (!parsed.success) return res.status(400).json({ error:
    "invalid_input" });

  const msg = await prisma.supportMessage.create({ data:
    parsed.data });

  return res.json({ ok: true, id: msg.id });
});

```

Файл «user.ts»

```

import { Router } from "express";

import { z } from "zod";

import { prisma } from "../lib/prisma";

import { requireAuth, type AuthedRequest } from
  "../middleware/requireAuth";

export const userRouter = Router();

userRouter.use(requireAuth);

userRouter.get("/bookings", async (req: AuthedRequest, res) => {
  const bookings = await prisma.booking.findMany({
    where: { userId: req.user!.id },
    orderBy: { createdAt: "desc" },
  });
});

```

```

        include: { class: { include: { trainer: true, service: true
    } } }

    });

    return res.json(bookings);

});

const bookSchema = z.object({

    classId: z.string().min(1),

    notes: z.string().max(500).optional()

});

userRouter.post("/book", async (req: AuthedRequest, res) => {

    const parsed = bookSchema.safeParse(req.body);

    if (!parsed.success) return res.status(400).json({ error:
"invalid_input" });

    const cls = await prisma.classSession.findUnique({

        where: { id: parsed.data.classId }

    });

    if (!cls || !cls.active) return res.status(404).json({ error:
"not_found" });

    const approved = await prisma.booking.count({ where: {
classId: cls.id, status: "APPROVED" } });

    if (approved >= cls.capacity) return res.status(409).json({
error: "full" });

    try {

        const booking = await prisma.booking.create({

            data: {

```

```

        classId: cls.id,
        userId: req.user!.id,
        notes: parsed.data.notes,
        status: "PENDING"
    }
});

return res.json({ ok: true, id: booking.id });
} catch {
    return res.status(409).json({ error: "already_booked" });
}
});

userRouter.post("/support", async (req: AuthedRequest, res) => {
    const schema = z.object({
        subject: z.string().min(1),
        body: z.string().min(1)
    });

    const parsed = schema.safeParse(req.body);

    if (!parsed.success) return res.status(400).json({ error:
"invalid_input" });

    const msg = await prisma.supportMessage.create({
        data: { ...parsed.data, userId: req.user!.id }
    });

    return res.json({ ok: true, id: msg.id });
});

```

Файл «requireAuth.ts»

```
import type { Request, Response, NextFunction } from "express";
```

					ПГУ1.090301.47.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		109

```

import { verifySessionToken } from "../lib/auth";

import { env } from "../lib/env";

export type AuthedRequest = Request & {
  user?: { id: string; role: "USER" | "ADMIN" };
};

export function requireAuth(req: AuthedRequest, res: Response,
next: NextFunction) {
  const token = req.cookies?.[env.COOKIE_NAME];

  if (!token) return res.status(401).json({ error:
"unauthorized" });

  try {
    const payload = verifySessionToken(token);

    req.user = { id: payload.sub, role: payload.role };

    next();
  } catch {
    return res.status(401).json({ error: "unauthorized" });
  }
}

export function requireAdmin(req: AuthedRequest, res: Response,
next: NextFunction) {
  requireAuth(req, res, () => {
    if (req.user?.role !== "ADMIN") return
res.status(403).json({ error: "forbidden" });

    next();
  });
}

```

Файл «auth.ts»

```
import jwt from "jsonwebtoken";

import { env } from "./env";

import type { UserRole } from "@prisma/client";

export type SessionTokenPayload = {
  sub: string;
  role: UserRole;
};

export function signSessionToken(payload: SessionTokenPayload) {
  return jwt.sign(payload, env.JWT_SECRET, { expiresIn: "7d" });
}

export function verifySessionToken(token: string) {
  return jwt.verify(token, env.JWT_SECRET) as
  SessionTokenPayload;
}
```

Файл «prisma.ts»

```
import { PrismaClient } from "@prisma/client";

export const prisma = new PrismaClient();
```

Файл «env.ts»

```
import { z } from "zod";

import dotenv from "dotenv";

dotenv.config();

const schema = z.object({
```

					ПГУ1.090301.47.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		111

```

    DATABASE_URL: z.string().min(1),

    PORT: z.coerce.number().int().positive().default(3001),

    APP_ORIGIN:
z.string().min(1).default("http://localhost:5173"),

    JWT_SECRET: z.string().min(16),

    COOKIE_NAME: z.string().min(1).default("fc_session"),

    COOKIE_SECURE: z

        .enum(["true", "false"])

        .default("false")

        .transform((v) => v === "true")

});

export const env = schema.parse(process.env);

```

Файл «style.css»

```

:root{

    --bg:#f6f7fb;

    --panel:#ffffff;

    --panel2:#f2f4fb;

    --text:#111827;

    --muted:#6b7280;

    --border:rgba(17,24,39,.10);

    --accent:#4f46e5;

    --accent2:#2563eb;

    --danger:#dc2626;

    --ok:#059669;

    --shadow:0 14px 40px rgba(17,24,39,.08);

    --r:16px;

}

*{box-sizing:border-box}

```

					ПГУ1.090301.47.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		112

```

html,body{height:100%}

/* Ensure full-height layout for footer pinning */

#root{min-height:100%}

body{

    margin:0;

    font-family: ui-sans-serif, system-ui, -apple-system, Segoe
    UI, Roboto, "Helvetica Neue", Arial, "Noto Sans", "Liberation
    Sans", sans-serif;

    background: var(--bg);

    color:var(--text);

}

a{color:inherit;text-decoration:none}

.muted{color:var(--muted)}

.container{width:min(1100px, 100%); margin:0 auto; padding: 22px
18px 44px}

.app{min-height:100vh; display:flex; flex-direction:column}

.container{flex:1}

.topbar{

    display:flex; gap:18px; align-items:center; justify-
    content:space-between;

    padding:18px 22px;

    border-bottom:1px solid var(--border);

    background: rgba(255,255,255,.92);

    backdrop-filter: blur(8px);

    position:sticky; top:0; z-index:5;

}

.brand{display:flex; gap:12px; align-items:center}

```

```

.logo{
  width:42px;height:42px;border-radius:12px;

  background: linear-gradient(135deg, var(--accent), var(--
accent2));

  box-shadow:0 10px 22px rgba(79,70,229,.18);

  display:grid;place-items:center;font-weight:800;

  color:#fff;
}

.brand-title{font-weight:800;letter-spacing:.2px}

.brand-sub{font-size:12px;color:var(--muted);margin-top:2px}

.nav{display:flex; gap:14px; flex-wrap:wrap; align-items:center}

.nav a{padding:8px 10px;border-radius:12px;border:1px solid
transparent}

.nav a.active{border-color:var(--border);background:var(--
panel)}

.authbox{display:flex; gap:10px; align-items:center}

.grid{display:grid; grid-template-columns: 1.25fr .75fr;
gap:18px}

@media (max-width: 960px){ .grid{grid-template-columns:1fr}
.nav{display:none} }

.card{

  border:1px solid var(--border);

  background: var(--panel);

  border-radius: var(--r);

  box-shadow: var(--shadow);

  padding: 18px;

}

.card h2{margin:0 0 10px; font-size:18px}

```

```

.card h3{margin:0 0 8px; font-size:15px}

.card p{margin:0 0 10px; color:var(--muted); line-height:1.45}


.row{display:flex; gap:12px; flex-wrap:wrap; align-items:center}

.actions{display:flex; gap:10px; flex-wrap:nowrap; align-items:center}

.actions > *{white-space:nowrap}

.spacer{flex:1}

.pill{display:inline-flex; align-items:center; border:1px solid var(--border); background: var(--panel2); padding:6px 10px; border-radius:999px; font-size:12px; color:var(--muted)}

.pill.status-pending{
    background: rgba(245, 158, 11, .14);
    border-color: rgba(245, 158, 11, .28);
    color: #92400e;
}

.pill.status-approved{
    background: rgba(16, 185, 129, .14);
    border-color: rgba(16, 185, 129, .28);
    color: #065f46;
}

.pill.status-rejected{
    background: rgba(220, 38, 38, .10);
    border-color: rgba(220, 38, 38, .24);
    color: #991b1b;
}

.pill.status-cancelled{
    background: rgba(107, 114, 128, .12);

```

```

border-color: rgba(107, 114, 128, .22);
color: #374151;
}

.btn{
border:1px solid var(--border);
background: var(--panel);
color:var(--text);
padding:10px 12px;border-radius:14px;
cursor:pointer;
display:inline-flex; align-items:center; gap:8px;
font-weight:700;
}

.btn:hover{background:var(--panel2)}

.btn.primary{border-color: transparent; background: linear-
gradient(135deg, rgba(79,70,229,.96), rgba(37,99,235,.92));
color:#fff}

.btn.primary:hover{filter: brightness(1.02)}

.btn.danger{border-color: rgba(220,38,38,.22); background:
rgba(220,38,38,.08); color: var(--danger)}

form{display:grid; gap:10px}

label{font-size:12px;color:var(--muted)}

input, textarea, select{
width:100%;
padding:10px 12px;
border-radius:14px;
border:1px solid var(--border);
background: #ffffff;

```

```

    color:var(--text);

    outline:none;

}

textarea{min-height:120px; resize:vertical}

input:focus, textarea:focus, select:focus{border-color:
rgba(79,70,229,.55)}


.table{width:100%; border-collapse:separate; border-spacing:0
10px}

.table th{font-size:12px; text-align:left; color:var(--muted);
font-weight:700}

.table td{padding:12px 12px; background: #ffffff; border-top:1px
solid var(--border); border-bottom:1px solid var(--border)}

.table td:first-child{border-left:1px solid var(--border);
border-radius:14px 0 0 14px}

.table td:last-child{border-right:1px solid var(--border);
border-radius:0 14px 14px 0}


.footer{

    padding: 18px 22px;

    border-top:1px solid var(--border);

    display:flex; justify-content:space-between; gap:12px; flex-
wrap:wrap;

    color:var(--muted);

    margin-top:auto;

}


.toast{

    position: fixed;

    bottom: 16px;

    left: 50%;

```

```

transform: translateX(-50%);
background: rgba(255,255,255,.96);
border:1px solid var(--border);
backdrop-filter: blur(10px);
padding:10px 12px;
border-radius: 14px;
box-shadow: var(--shadow);
color: var(--text);
min-width: min(560px, calc(100vw - 28px));
}

```

Файл «App.tsx»

```

import { NavLink, Route, Routes, useNavigate } from "react-router-dom";

import { useEffect, useMemo, useState } from "react";

import { api, type Me } from "../api";

import { Home } from "../pages/Home";

import { Services } from "../pages/Services";

import { Schedule } from "../pages/Schedule";

import { Contacts } from "../pages/Contacts";

import { Support } from "../pages/Support";

import { Login } from "../pages/Login";

import { Register } from "../pages/Register";

import { Cabinet } from "../pages/Cabinet";

import { Admin } from "../pages/Admin";

function Toast({ message }: { message: string }) {

  if (!message) return null;

  return <div className="toast">{message}</div>;
}

```

```

}

export function App() {
  const [me, setMe] = useState<Me | null>(null);
  const [toast, setToast] = useState("");
  const nav = useNavigate();

  const isAdmin = useMemo(() => me?.role === "ADMIN", [me]);

  useEffect(() => {
    api.me().then(setMe).catch(() => setMe(null));
  }, []);

  function notify(msg: string) {
    setToast(msg);
    window.setTimeout(() => setToast(""), 2200);
  }

  async function onLogout() {
    await api.logout();
    setMe(null);
    notify("Вы вышли из аккаунта");
    nav("/");
  }

  return (
    <div className="app">
      <header className="topbar">

```

```

<div className="brand">

  <div className="logo">FC</div>

  <div>

    <div className="brand-title">Fitness Center</div>

    <div className="brand-sub">Онлайн-запись, кабинеты,
админ-панель</div>

  </div>

</div>

<nav className="nav">

  <NavLink to="/" end>

    Главная

  </NavLink>

  <NavLink to="/services">Услуги</NavLink>

  <NavLink to="/schedule">Расписание</NavLink>

  <NavLink to="/contacts">Контакты</NavLink>

  <NavLink to="/support">Поддержка</NavLink>

  {me && <NavLink to="/cabinet">Кабинет</NavLink>}

  {isAdmin && <NavLink to="/admin">Админ</NavLink>}

</nav>

<div className="authbox">

  {!me ? (

    <>

      <NavLink className="btn" to="/login">

        Войти

      </NavLink>

      <NavLink className="btn primary" to="/register">

```

```

        Регистрация

    </NavLink>

</>

) : (

<>

    <span className="pill">{me.name}</span>

    <button className="btn" onClick={onLogout}>

        Выйти

    </button>

</>

)}

</div>

</header>

<div className="container">

    <Routes>

        <Route path="/" element={<Home me={me} />} />

        <Route path="/services" element={<Services />} />

        <Route path="/schedule" element={<Schedule me={me}
notify={notify} />} />

        <Route path="/contacts" element={<Contacts />} />

        <Route path="/support" element={<Support me={me}
notify={notify} />} />

        <Route path="/login" element={<Login onLogin={ (u) =>
(setMe(u), notify("Добро пожаловать!"), nav("/cabinet"))} />} />

        <Route

            path="/register"

            element={<Register onRegister={ (u) => (setMe(u),
notify("Аккаунт создан"), nav("/cabinet"))} />}

    </Routes>

```

```

        />

        <Route path="/cabinet" element={<Cabinet me={me} />}
/>

        <Route path="/admin" element={<Admin me={me}
notify={notify} />} />

    </Routes>

</div>

<footer className="footer">

    <div>© {new Date().getFullYear()} Fitness Center</div>

    <div className="muted"></div>

</footer>

    <Toast message={toast} />

</div>

);
}

```

Файл «Api.ts»

```

export type Me = {
    id: string;
    role: "USER" | "ADMIN";
    email: string;
    name: string;
    phone?: string | null;
};

const API_BASE = "";

```

					ПГУ1.090301.47.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		122

```

export class ApiError extends Error {
    status: number;
    data: any;
    constructor(status: number, message: string, data: any) {
        super(message);
        this.name = "ApiError";
        this.status = status;
        this.data = data;
    }
}

```

```

async function req<T>(path: string, init?: RequestInit):
Promise<T> {
    const res = await fetch(`${API_BASE}${path}`, {
        credentials: "include",
        headers: { "content-type": "application/json",
        ...(init?.headers || {}) },
        ...init
    });
    if (!res.ok) {
        const ct = res.headers.get("content-type") || "";
        if (ct.includes("application/json")) {
            const data = await res.json().catch(() => null);
            const msg = (data && (data.error || data.message)) ?
String(data.error || data.message) : `HTTP ${res.status}`;
            throw new ApiError(res.status, msg, data);
        }
        const text = await res.text().catch(() => "");
    }
}

```

```

        throw new ApiError(res.status, text || `HTTP ${res.status}`,
null);

    }

    return (await res.json()) as T;
}

export const api = {

    me: () => req<Me | null>("/api/auth/me"),

    login: (email: string, password: string) =>

        req<Me>("/api/auth/login", { method: "POST", body:
JSON.stringify({ email, password }) }),

    register: (payload: { email: string; password: string; name:
string; phone?: string }) =>

        req<Me>("/api/auth/register", { method: "POST", body:
JSON.stringify(payload) }),

    logout: () => req<{ ok: boolean }>("/api/auth/logout", {
method: "POST", body: "{}" }),


    services: () => req<any[]>("/api/public/services"),

    trainers: () => req<any[]>("/api/public/trainers"),

    schedule: () => req<any[]>("/api/public/schedule"),

    supportPublic: (payload: any) =>
req<any>("/api/public/support", { method: "POST", body:
JSON.stringify(payload) }),

    myBookings: () => req<any[]>("/api/user/bookings"),

    book: (classId: string, notes?: string) =>

        req<any>("/api/user/book", { method: "POST", body:
JSON.stringify({ classId, notes }) }),

    supportUser: (payload: any) => req<any>("/api/user/support", {
method: "POST", body: JSON.stringify(payload) }),

```

```

adminSupport: () => req<any[]>("/api/admin/support"),

adminSupportDelete: (id: string) =>
req<any>(`/api/admin/support/${id}`, { method: "DELETE" }),

adminBookings: () => req<any[]>("/api/admin/bookings"),

adminBookingUpdate: (id: string, payload: any) =>

    req<any>(`/api/admin/bookings/${id}`, { method: "PATCH",
body: JSON.stringify(payload) }),

adminTrainers: () => req<any[]>("/api/admin/trainers"),

adminTrainerCreate: (payload: any) =>
req<any>("/api/admin/trainers", { method: "POST", body:
JSON.stringify(payload) }),

adminTrainerUpdate: (id: string, payload: any) =>

    req<any>(`/api/admin/trainers/${id}`, { method: "PATCH",
body: JSON.stringify(payload) }),

adminTrainerDelete: (id: string) =>
req<any>(`/api/admin/trainers/${id}`, { method: "DELETE" }),

adminServices: () => req<any[]>("/api/admin/services"),

adminServiceCreate: (payload: any) =>
req<any>("/api/admin/services", { method: "POST", body:
JSON.stringify(payload) }),

adminServiceUpdate: (id: string, payload: any) =>

    req<any>(`/api/admin/services/${id}`, { method: "PATCH",
body: JSON.stringify(payload) }),

adminServiceDelete: (id: string) =>
req<any>(`/api/admin/services/${id}`, { method: "DELETE" }),

adminClasses: () => req<any[]>("/api/admin/classes"),

adminClassCreate: (payload: any) =>
req<any>("/api/admin/classes", { method: "POST", body:
JSON.stringify(payload) }),

adminClassUpdate: (id: string, payload: any) =>

    req<any>(`/api/admin/classes/${id}`, { method: "PATCH",
body: JSON.stringify(payload) }),

adminClassDelete: (id: string) =>
req<any>(`/api/admin/classes/${id}`, { method: "DELETE" })

```

```
};
```

Файл «main.tsx»

```
import React from "react";

import ReactDOM from "react-dom/client";

import { BrowserRouter } from "react-router-dom";

import { App } from "../ui/App";

import "../ui/styles.css";

ReactDOM.createRoot(document.getElementById("root")!).render(

  <React.StrictMode>

    <BrowserRouter>

      <App />

    </BrowserRouter>

  </React.StrictMode>

);
```

Файл «Admin.tsx»

```
import { useEffect, useState } from "react";

import { api, type Me } from "../api";

import { Link } from "react-router-dom";

function statusView(status: string): { label: string; cls: string } {

  switch (status) {

    case "PENDING":

      return { label: "Ожидает", cls: "status-pending" };

    case "APPROVED":

      return { label: "Одобрено", cls: "status-approved" };

    case "REJECTED":
```

					ПГУ1.090301.47.001 ПЗ	Лист
						126
Изм.	Лист	№ докум.	Подпись	Дата		

```

        return { label: "Отклонено", cls: "status-rejected" };
    case "CANCELLED":
        return { label: "Отменено", cls: "status-cancelled" };
    default:
        return { label: status, cls: "" };
    }
}

function supportStatusView(status: string): { label: string;
cls: string } {
    switch (status) {
        case "OPEN":
            return { label: "Открыто", cls: "status-pending" };
        case "CLOSED":
            return { label: "Закрыто", cls: "status-approved" };
        default:
            return { label: status, cls: "" };
    }
}

export function Admin({ me, notify }: { me: Me | null; notify:
(s: string) => void }) {

    const [tab, setTab] = useState<"classes" | "trainers" |
"services" | "bookings" | "support">("classes");

    const [classes, setClasses] = useState<any[] | null>(null);
    const [trainers, setTrainers] = useState<any[] | null>(null);
    const [services, setServices] = useState<any[] | null>(null);
    const [bookings, setBookings] = useState<any[] | null>(null);
    const [support, setSupport] = useState<any[] | null>(null);

```

```

    const [openSupportId, setOpenSupportId] = useState<string |
    null>(null);

    async function refresh() {

        try {

            const [c, t, s] = await Promise.all([api.adminClasses(),
            api.adminTrainers(), api.adminServices()]);

            setClasses(c);

            setTrainers(t);

            setServices(s);

        } catch {

            // ignore

        }

    }

    useEffect(() => {

        if (me?.role !== "ADMIN") return;

        refresh();

    }, [me]);

    if (!me) {

        return (

            <div className="card">

                <h2>Админ-панель</h2>

                <p>

                    Нужно <Link to="/login">войти</Link>.

                </p>

            </div>

        );

    }

```

```

    }

    if (me.role !== "ADMIN") {

        return (

            <div className="card">

                <h2>Админ-панель</h2>

                <p className="muted">Доступ запрещён.</p>

            </div>

        );

    }

    async function openTab(next: typeof tab) {

        setTab(next);

        try {

            if (next === "bookings") setBookings(await
api.adminBookings());

            if (next === "support") setSupport(await
api.adminSupport());

            if (next === "classes") setClasses(await
api.adminClasses());

            if (next === "trainers") setTrainers(await
api.adminTrainers());

            if (next === "services") setServices(await
api.adminServices());

        } catch {

            notify("Ошибка загрузки");

        }

    }

    async function addTrainer(e: React.FormEvent<HTMLFormElement>)
{

```

```

    e.preventDefault();

    const fd = new FormData(e.currentTarget);

    const name = String(fd.get("name") || "").trim();

    const bio = String(fd.get("bio") || "").trim();

    if (!name) return notify("Имя тренера обязательно");

    await api.adminTrainerCreate({ name, bio: bio || undefined,
active: true });

    notify("Тренер добавлен");

    e.currentTarget.reset();

    setTrainers(await api.adminTrainers());
}

async function delTrainer(id: string) {
    if (!confirm("Удалить тренера?")) return;

    await api.adminTrainerDelete(id);

    notify("Тренер удалён");

    setTrainers(await api.adminTrainers());
}

async function delService(id: string) {
    if (!confirm("Удалить услугу?")) return;

    await api.adminServiceDelete(id);

    notify("Услуга удалена");

    setServices(await api.adminServices());
}

async function delClass(id: string) {
    if (!confirm("Удалить занятие?")) return;

```

```

    await api.adminClassDelete(id);

    notify("Занятие удалено");

    setClasses(await api.adminClasses());
}

async function deleteSupport(id: string) {
    if (!confirm("Удалить обращение?")) return;

    await api.adminSupportDelete(id);

    notify("Обращение удалено");

    setSupport(await api.adminSupport());
}

async function approveBooking(id: string) {
    await api.adminBookingUpdate(id, { status: "APPROVED" });

    notify("Заявка одобрена");

    setBookings(await api.adminBookings());

    setClasses(await api.adminClasses());
}

async function rejectBooking(id: string) {
    await api.adminBookingUpdate(id, { status: "REJECTED" });

    notify("Заявка отклонена");

    setBookings(await api.adminBookings());
}

async function cancelBooking(id: string) {
    await api.adminBookingUpdate(id, { status: "CANCELLED" });

    notify("Заявка отменена");
}

```

```

    setBookings(await api.adminBookings());

    setClasses(await api.adminClasses());
  }

  async function addService(e: React.FormEvent<HTMLFormElement>)
  {
    e.preventDefault();

    const fd = new FormData(e.currentTarget);

    const title = String(fd.get("title") || "").trim();

    const summary = String(fd.get("summary") || "").trim();

    if (!title) return notify("Название услуги обязательно");

    await api.adminServiceCreate({ title, summary: summary ||
undefined, active: true });

    notify("Услуга добавлена");

    e.currentTarget.reset();

    setServices(await api.adminServices());
  }

  async function addClass(e: React.FormEvent<HTMLFormElement>) {
    e.preventDefault();

    const fd = new FormData(e.currentTarget);

    const startsAt = String(fd.get("startsAt") || "");

    const durationMin = Number(fd.get("durationMin") || 60);

    const capacity = Number(fd.get("capacity") || 12);

    const location = String(fd.get("location") || "").trim();

    const trainerId = String(fd.get("trainerId") || "");

    const serviceId = String(fd.get("serviceId") || "");

    if (!startsAt) return notify("Дата/время обязательно");

    await api.adminClassCreate({

```

```

        startsAt: new Date(startsAt).toISOString(),
        durationMin,
        capacity,
        location: location || undefined,
        trainerId: trainerId || null,
        serviceId: serviceId || null,
        active: true
    });

    notify("Занятие добавлено");
    e.currentTarget.reset();
    setClasses(await api.adminClasses());
}

return (
    <div className="card">
        <div className="row" style={{ marginBottom: 10 }}>
            <h2 style={{ margin: 0 }}>Админ-панель</h2>
            <div className="spacer" />
            <button className="btn" onClick={refresh}>
                ОБНОВИТЬ
            </button>
        </div>

        <div className="row" style={{ marginBottom: 14 }}>
            <button className={`btn ${tab === "classes" ? "primary"
: ""}`} onClick={() => openTab("classes")}>
                Расписание
            </button>
        </div>
    </div>
)

```

```

        <button className={`btn ${tab === "trainers" ? "primary"
: ""}`} onClick={() => openTab("trainers")}>

            Тренеры

        </button>

        <button className={`btn ${tab === "services" ? "primary"
: ""}`} onClick={() => openTab("services")}>

            Услуги

        </button>

        <button className={`btn ${tab === "bookings" ? "primary"
: ""}`} onClick={() => openTab("bookings")}>

            Заявки

        </button>

        <button className={`btn ${tab === "support" ? "primary"
: ""}`} onClick={() => openTab("support")}>

            Поддержка

        </button>

    </div>

    {tab === "trainers" && (

        <div className="grid" style={{ gridTemplateColumns: "1fr
1fr" }}>

            <section className="card" style={{ boxShadow: "none"
}}>

                <h3>Добавить тренера</h3>

                <form onSubmit={addTrainer}>

                    <div>

                        <label>Имя</label>

                        <input name="name" />

                    </div>

                    <div>

```

```

        <label>Bio</label>

        <textarea name="bio" />

    </div>

    <button className="btn primary" type="submit">

        Добавить

    </button>

</form>

</section>

<section className="card" style={{ boxShadow: "none"
}}>

    <h3>Список</h3>

    {!trainers ? (

        <p className="muted">Загрузка...</p>

    ) : (

        <div style={{ display: "grid", gap: 10 }}>

            {trainers.map((t) => (

                <div key={t.id} className="row">

                    <span className="pill">{t.name}</span>

                    <span className="muted" style={{ fontSize:
12 }}>

                        {t.active ? "активен" : "отключён"}

                    </span>

                    <div className="spacer" />

                    <button className="btn danger" onClick={()
=> delTrainer(t.id)}>

                        Удалить

                    </button>

                </div>

            ) ) }

        </div>

    ) ) }

```

```

        </div>

    })

</section>

</div>

})

{tab === "services" && (

    <div className="grid" style={{ gridTemplateColumns: "1fr
1fr" }}>

        <section className="card" style={{ boxShadow: "none"
}}>

            <h3>Добавить услугу</h3>

            <form onSubmit={addService}>

                <div>

                    <label>Название</label>

                    <input name="title" />

                </div>

                <div>

                    <label>Описание</label>

                    <textarea name="summary" />

                </div>

                <button className="btn primary" type="submit">

                    Добавить

                </button>

            </form>

        </section>

        <section className="card" style={{ boxShadow: "none"
}}>

            <h3>Список</h3>

```

```

        {!services ? (
            <p className="muted">Загрузка...</p>
        ) : (
            <div style={{ display: "grid", gap: 10 }}>
                {services.map((s) => (
                    <div key={s.id} className="row">
                        <span className="pill">{s.title}</span>
                        <span className="muted" style={{ fontSize:
12 }}>
                            {s.active ? "активно" : "отключено"}
                        </span>
                        <div className="spacer" />
                        <button className="btn danger" onClick={()
=> delService(s.id)}>
                            Удалить
                        </button>
                    </div>
                )))}
            </div>
        )}
    </section>
</div>
)}

{tab === "classes" && (
    <div className="grid" style={{ gridTemplateColumns: "1fr
1.2fr" }}>
        <section className="card" style={{ boxShadow: "none"
}}>

```

```

<h3>Добавить занятие</h3>

<form onSubmit={addClass}>

  <div>

    <label>Дата/время</label>

    <input name="startsAt" type="datetime-local" />

  </div>

  <div className="row">

    <div style={{ flex: 1 }}>

      <label>Длительность (мин)</label>

      <input name="durationMin" type="number"
defaultValue={60} min={15} />

    </div>

    <div style={{ flex: 1 }}>

      <label>Мест</label>

      <input name="capacity" type="number"
defaultValue={12} min={1} />

    </div>

  </div>

  <div>

    <label>Локация</label>

    <input name="location" placeholder="Зал А" />

  </div>

  <div className="row">

    <div style={{ flex: 1 }}>

      <label>Тренер</label>

      <select name="trainerId" defaultValue="">

        <option value="">—</option>

        {(trainers || []).map((t) => (

          <option key={t.id} value={t.id}>

```

```

        {t.name}
      </option>
    )})
  </select>
</div>

<div style={{ flex: 1 }}>
  <label>Услуга</label>
  <select name="serviceId" defaultValue="">
    <option value="">-</option>
    {(services || []).map((s) => (
      <option key={s.id} value={s.id}>
        {s.title}
      </option>
    ))}
  </select>
</div>
</div>

<button className="btn primary" type="submit">
  Добавить
</button>
</form>
</section>

<section className="card" style={{ boxShadow: "none"
}}>

  <h3>Расписание</h3>

  {!classes ? (
    <p className="muted">Загрузка...</p>
  ) : classes.length === 0 ? (

```

```

    <p className="muted">Пока пусто.</p>
  ) : (
    <table className="table">
      <thead>
        <tr>
          <th>Время</th>
          <th>Услуга</th>
          <th>Тренер</th>
          <th>Записано</th>
        </tr>
      </thead>
      <tbody>
        {classes.map((c) => (
          <tr key={c.id}>
            <td>{new
Date(c.startsAt).toLocaleString("ru-RU")}</td>
            <td>{c.service?.title ?? "-"}</td>
            <td>{c.trainer?.name ?? "-"}</td>
            <td>
              <div className="row">
                <span className="pill">
                  {(c.approvedCount ??
c._count?.bookings ?? 0)}/{c.capacity}
                </span>
                <button className="btn danger"
onClick={() => delClass(c.id)}>
                  Удалить
                </button>
              </div>
            </td>
          </tr>
        ))}
      </tbody>
    </table>
  )
}

```

```

        </td>

    </tr>

    )})

</tbody>

</table>

    })

</section>

</div>

    })

{tab === "bookings" && (

    <section className="card" style={{ boxShadow: "none" }}>

        <h3>Заявки на запись</h3>

        {!bookings ? (

            <p className="muted">Загрузка...</p>

        ) : bookings.length === 0 ? (

            <p className="muted">Заявок пока нет.</p>

        ) : (

            <table className="table">

                <thead>

                    <tr>

                        <th>Пользователь</th>

                        <th>Время занятия</th>

                        <th>Услуга</th>

                        <th>Тренер</th>

                        <th>Статус</th>

                        <th>Действия</th>

                    </tr>

```

```

</thead>

<tbody>

    {bookings.map((b) => (

        <tr key={b.id}>

            <td>{b.user?.email ?? "-"}</td>

            <td>{new
Date(b.class.startsAt).toLocaleString("ru-RU")}</td>

            <td>{b.class.service?.title ?? "-"}</td>

            <td>{b.class.trainer?.name ?? "-"}</td>

            <td>

                <span className={`pill
${statusView(b.status).cls}`}>{statusView(b.status).label}</span>
>

            </td>

            <td>

                <div className="actions">

                    {b.status === "PENDING" && (

                        <>

                            <button className="btn primary"
onClick={() => approveBooking(b.id)}>

                                Одобрить

                            </button>

                            <button className="btn danger"
onClick={() => rejectBooking(b.id)}>

                                Отклонить

                            </button>

                        </>

                    )}

                    {b.status === "APPROVED" && (

```

```

                                <button className="btn danger"
onClick={ () => cancelBooking(b.id) }>

                                Отменить

                                </button>

                                )}

                                </div>

                                </td>

                                </tr>

                                </tbody>

                                </table>

                                )}

                                </section>

                                )}

```

```

{tab === "support" && (
    <section className="card" style={{ boxShadow: "none" }}>
        <h3>Сообщения поддержки</h3>
        {!support ? (
            <p className="muted">Загрузка...</p>
        ) : support.length === 0 ? (
            <p className="muted">Сообщений пока нет.</p>
        ) : (
            <table className="table">
                <thead>
                    <tr>
                        <th>Когда</th>
                        <th>Тема</th>

```

```

        <th>От</th>

        <th>Статус</th>

        <th>Текст</th>

        <th></th>

    </tr>

</thead>

<tbody>

    {support.map((m) => {

        const opened = openSupportId === m.id;

        const sv = supportStatusView(m.status);

        return (

            <tr key={m.id}>

                <td>{new
Date(m.createdAt).toLocaleString("ru-RU")}</td>

                <td>{m.subject}</td>

                <td>{m.user?.email ?? m.email ??
"Гость">}</td>

                <td>

                    <span className={`pill
${sv.cls}`}>{sv.label}</span>

                </td>

                <td style={{ maxWidth: 360 }}>

                    <div className="actions" style={{
justifyContent: "space-between", width: "100%" }}>

                        <span

                            className="muted"

                            style={{ fontSize: 12, overflow:
"hidden", textOverflow: "ellipsis", whiteSpace: "nowrap" }}

                            title={m.body}

                        >

```

```

|| "-"}

                                {String(m.body || "").slice(0, 90)

                                </span>

                                <button className="btn" onClick={() =>
setOpenSupportId((prev) => (prev === m.id ? null : m.id))}>

                                {opened ? "Скрыть" : "Показать"}

                                </button>

                                </div>

                                {opened && (

                                <div className="card" style={{
boxShadow: "none", marginTop: 10, background: "var(--panel2)"
}}>

                                <p style={{ margin: 0, whiteSpace:
"pre-wrap" }}>{m.body}</p>

                                </div>

                                )}

                                </td>

                                <td>

                                <button className="btn danger"
onClick={() => deleteSupport(m.id)}>

                                Удалить

                                </button>

                                </td>

                                </tr>

                                );

                                }}}

                                </tbody>

                                </table>

                                )}

                                </section>

```

```

    })

</div>

);

}

Файл «Cabinet.tsx»

import { useEffect, useState } from "react";
import { api, type Me } from "../api";
import { Link } from "react-router-dom";

function fmt(iso: string) {

    return new Date(iso).toLocaleString("ru-RU", { day: "2-digit",
month: "2-digit", hour: "2-digit", minute: "2-digit" });

}

function statusView(status: string): { label: string; cls:
string } {

    switch (status) {

        case "PENDING":

            return { label: "Ожидает", cls: "status-pending" };

        case "APPROVED":

            return { label: "Одобрено", cls: "status-approved" };

        case "REJECTED":

            return { label: "Отклонено", cls: "status-rejected" };

        case "CANCELLED":

            return { label: "Отменено", cls: "status-cancelled" };

        default:

            return { label: status, cls: "" };

    }

}

```

```

}

export function Cabinet({ me }: { me: Me | null }) {
  const [items, setItems] = useState<any[] | null>(null);

  useEffect(() => {
    if (!me) return;

    api.myBookings().then(setItems).catch(() => setItems([]));
  }, [me]);

  if (!me) {
    return (
      <div className="card">
        <h2>Личный кабинет</h2>
        <p>
          Нужно <Link to="/login">войти</Link>.
        </p>
      </div>
    );
  }

  return (
    <div className="card">
      <div className="row">
        <h2 style={{ margin: 0 }}>Личный кабинет</h2>
        <div className="spacer" />
        <span className="pill">{me.email}</span>
      </div>
    </div>
  );
}

```

```
<p className="muted">Здесь отображаются твои записи на занятия.</p>
```

```
{!items ? (
```

```
<p className="muted">Загрузка...</p>
```

```
) : items.length === 0 ? (
```

```
<p className="muted">Записей пока нет. Перейди в расписание и запишись.</p>
```

```
) : (
```

```
<table className="table">
```

```
<thead>
```

```
<tr>
```

```
<th>Создано</th>
```

```
<th>Время занятия</th>
```

```
<th>Услуга</th>
```

```
<th>Тренер</th>
```

```
<th>Статус</th>
```

```
</tr>
```

```
</thead>
```

```
<tbody>
```

```
{items.map((b) => (
```

```
<tr key={b.id}>
```

```
<td>{fmt (b.createdAt) }</td>
```

```
<td>{fmt (b.class.startsAt) }</td>
```

```
<td>{b.class.service?.title ?? "-"}</td>
```

```
<td>{b.class.trainer?.name ?? "-"}</td>
```

```
<td>
```

```

        <span className={`pill
${statusView(b.status).cls}`}>{statusView(b.status).label}</span
>

        </td>

    </tr>

    )})

</tbody>

</table>

    )}

</div>

    );

}

```

Файл «Contacts.tsx»

```

export function Contacts() {

    return (

        <div className="card">

            <h2>Контакты</h2>

            <p>Эту страницу можно дополнить реальными данными
фитнес-центра.</p>

            <div className="row" style={{ gap: 14 }}>

                <span className="pill">Телефон: +7 (999)
000-00-00</span>

                <span className="pill">Email: info@fitness.local</span>

                <span className="pill">Адрес: ул. Примерная, 1</span>

            </div>

        </div>

    );

}

```

Файл «Home.tsx»

```

import type { Me } from "../api";

import { Link } from "react-router-dom";

export function Home({ me }: { me: Me | null }) {
  return (
    <div className="grid">
      <section className="card">
        <h2>Фитнес-центр: удобно, быстро, онлайн</h2>
        <p>
          Смотри расписание, выбирай тренировку и записывайся
          онлайн. Есть поддержка, личный кабинет и админ-панель для
          управления расписанием, тренерами и услугами.
        </p>
        <div className="row">
          <Link className="btn primary" to="/schedule">
            Открыть расписание
          </Link>
          <Link className="btn" to="/support">
            Написать в поддержку
          </Link>
          <div className="spacer" />
          <span className="pill">Адаптивный интерфейс</span>
        </div>
      </section>

      <aside className="card">
        <h3>Быстрые действия</h3>
        {!me ? (

```

<>

<p>Войди или зарегистрируйся, чтобы записываться на занятия и видеть заявки в кабинете.</p>

<div className="row">

<Link className="btn" to="/login">

Войти

</Link>

<Link className="btn primary" to="/register">

Регистрация

</Link>

</div>

</>

) : (

<>

<p>Ты вошёл как {me.name}. Перейди в кабинет, чтобы посмотреть свои записи.</p>

<div className="row">

<Link className="btn primary" to="/cabinet">

Кабинет

</Link>

{me.role === "ADMIN" && (

<Link className="btn" to="/admin">

Админ-панель

</Link>

)}

</div>

</>

)}

</aside>

					ПГУ1.090301.47.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		151

```

        </div>

    );
}

Файл «Login.tsx»

import { useState } from "react";
import { api, type Me } from "../api";

export function Login({ onLogin }: { onLogin: (me: Me) => void }) {

    const [email, setEmail] = useState("");
    const [password, setPassword] = useState("");
    const [err, setErr] = useState("");

    async function submit(e: React.FormEvent) {

        e.preventDefault();

        setErr("");

        try {

            const me = await api.login(email, password);

            onLogin(me);

        } catch {

            setErr("Неверные данные");

        }

    }

    return (

        <div className="card" style={{ maxWidth: 520, margin: "0 auto" }}>

            <h2>Вход</h2>

```

```

    <form onSubmit={submit}>

      <div>

        <label>Email</label>

        <input value={email} onChange={ (e) =>
setEmail(e.target.value)} placeholder="admin@local.test" />

      </div>

      <div>

        <label>Пароль</label>

        <input type="password" value={password} onChange={ (e)
=> setPassword(e.target.value)} placeholder="....." />

      </div>

      {err && <p style={{ color: "var(--danger)", margin: 0
}}>{err}</p>}

      <div className="row">

        <button className="btn primary" type="submit">

          Войти

        </button>

        <span className="pill">admin: admin@local.test /
adminadmin</span>

      </div>

    </form>

  </div>

);
}

```

Файл «Register.tsx»

```

import { useState } from "react";

import { ApiError, api, type Me } from "../api";

export function Register({ onRegister }: { onRegister: (me: Me)
=> void }) {

```

```

const [email, setEmail] = useState("");
const [password, setPassword] = useState("");
const [name, setName] = useState("");
const [phone, setPhone] = useState("");
const [err, setErr] = useState("");

async function submit(e: React.FormEvent) {
  e.preventDefault();

  setErr("");

  if (!email.trim() || !email.includes("@")) {
    setErr("Укажи корректный email");
    return;
  }

  if (!name.trim()) {
    setErr("Имя обязательно");
    return;
  }

  if (password.length < 8) {
    setErr("Пароль минимум 8 символов");
    return;
  }

  try {
    const me = await api.register({ email, password, name,
phone: phone || undefined });

    onRegister(me);
  } catch (e: any) {

    if (e instanceof ApiError) {

      if (e.data?.error === "email_taken") return setErr("Этот
email уже занят");
    }
  }
}

```

					ПГУ1.090301.47.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		154

```

        if (e.data?.error === "invalid_input") return
        setErr("Проверь поля формы (email/имя/пароль/телефон)");

    }

    setErr("Не удалось зарегистрироваться (ошибка сервера)");

}

}

return (

    <div className="card" style={{ maxWidth: 560, margin: "0
auto" }}>

        <h2>Регистрация</h2>

        <form onSubmit={submit}>

            <div className="grid" style={{ gridTemplateColumns: "1fr
1fr", gap: 12 }}>

                <div>

                    <label>Email</label>

                    <input value={email} onChange={(e) =>
setEmail(e.target.value)} placeholder="you@example.com" />

                </div>

                <div>

                    <label>Телефон (опционально)</label>

                    <input value={phone} onChange={(e) =>
setPhone(e.target.value)} placeholder="+7..." />

                </div>

                <div style={{ gridColumn: "1 / -1" }}>

                    <label>Имя</label>

                    <input value={name} onChange={(e) =>
setName(e.target.value)} placeholder="Имя" />

                </div>

                <div style={{ gridColumn: "1 / -1" }}>

```

```

        <label>Пароль</label>

        <input type="password" value={password}
onChange={(e) => setPassword(e.target.value)}
placeholder="Минимум 8 символов" />

    </div>

</div>

    {err && <p style={{ color: "var(--danger)", margin: 0
}}>{err}</p>}

    <div className="row">

        <button className="btn primary" type="submit">

            Создать аккаунт

        </button>

    </div>

</form>

</div>

);
}

```

Файл «Shedule.tsx»

```

import { useEffect, useMemo, useState } from "react";
import { ApiError, api, type Me } from "../api";
import { Link } from "react-router-dom";

function formatDT(iso: string) {

    const d = new Date(iso);

    return d.toLocaleString("ru-RU", { weekday: "short", day: "2-
digit", month: "2-digit", hour: "2-digit", minute: "2-digit" });

}

export function Schedule({ me, notify }: { me: Me | null;
notify: (s: string) => void }) {

```

```

const [items, setItems] = useState<any[] | null>(null);

const canBook = !!me;

useEffect(() => {

    api.schedule().then(setItems).catch(() => setItems([]));

}, []);

const rows = useMemo(() => items ?? [], [items]);

async function book(classId: string) {

    if (!me) return;

    try {

        await api.book(classId);

        notify("Заявка на запись отправлена");

    } catch (e: any) {

        if (e instanceof ApiError) {

            if (e.status === 401) return notify("Нужно войти в
аккаунт");

            const code = e.data?.error;

            if (code === "full") return notify("Мест нет (все места
заняты одобренными заявками)");

            if (code === "already_booked") return notify("Ты уже
отправлял заявку на это занятие");

            if (code === "not_found") return notify("Занятие не
найденно (возможно удалено админом)");

        }

        notify("Не удалось записаться (ошибка сервера)");

    }

}

```

```

return (

  <div className="card">

    <div className="row">

      <h2 style={{ margin: 0 }}>Расписание</h2>

      <div className="spacer" />

      {!me ? (

        <span className="pill">

          Чтобы записаться: <Link to="/login">войти</Link>

        </span>

      ) : (

        <span className="pill">Можно записываться</span>

      )}

    </div>

    <p className="muted">Расписание управляется администратором через админ-панель.</p>

    {!items ? (

      <p className="muted">Загрузка...</p>

    ) : rows.length === 0 ? (

      <p className="muted">Занятий пока нет.</p>

    ) : (

      <table className="table">

        <thead>

          <tr>

            <th>Время</th>

            <th>Услуга</th>

            <th>Тренер</th>


```

```

        <th>Место</th>

        <th>Запись</th>

    </tr>
</thead>
<tbody>
    {rows.map((c) => (
        <tr key={c.id}>
            <td>{formatDT(c.startsAt)}</td>
            <td>{c.service?.title ?? c.service_title ?? "-"}</td>
            <td>{c.trainer?.name ?? c.trainer_name ?? "-"}</td>
            <td>{c.location ?? "-"}</td>
            <td>
                <div className="row">
                    <span className="pill">
                        {c.approvedCount ?? c._count?.bookings ??
c.booked ?? 0}/{c.capacity}
                    </span>
                    <button className="btn primary"
disabled={!canBook} onClick={() => book(c.id)}>
                        Записаться
                    </button>
                </div>
            </td>
        </tr>
    )})}
</tbody>
</table>

```

```

    })
  </div>

  );
}

Файл «Services.tsx»

import { useEffect, useState } from "react";
import { api } from "../api";

export function Services() {
  const [items, setItems] = useState<any[] | null>(null);

  useEffect(() => {
    api.services().then(setItems).catch(() => setItems([]));
  }, []);

  return (
    <div className="card">
      <h2>Услуги</h2>

      <p>Список услуг берётся из базы данных и редактируется админом.</p>

      {!items ? (
        <p className="muted">Загрузка...</p>
      ) : items.length === 0 ? (
        <p className="muted">Пока нет активных услуг.</p>
      ) : (
        <div style={{ display: "grid", gap: 12 }}>
          {items.map((s) => (

```

```

        <div key={s.id} className="card" style={{ boxShadow:
"none" }}>

            <div className="row">

                <h3 style={{ margin: 0 }}>{s.title}</h3>

                <div className="spacer" />

                <span className="pill">Активно</span>

            </div>

            {s.summary ? <p>{s.summary}</p> : <p
className="muted">Описание не указано.</p>}

        </div>

    )))

</div>

)}

</div>

);

}

```

Файл «Supports.tsx»

```

import { useState } from "react";

import { api, type Me } from "../api";

export function Support({ me, notify }: { me: Me | null; notify:
(s: string) => void }) {

    const [subject, setSubject] = useState("");

    const [body, setBody] = useState("");

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

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

    const [phone, setPhone] = useState("");

    async function submit(e: React.FormEvent) {

```

```

e.preventDefault();

if (!subject.trim() || !body.trim()) {

    notify("Заполни тему и сообщение");

    return;

}

try {

    if (me) {

        await api.supportUser({ subject, body });

    } else {

        await api.supportPublic({ subject, body, email: email ||
undefined, name: name || undefined, phone: phone || undefined
});

    }

    notify("Сообщение отправлено");

    setSubject("");

    setBody("");

    setEmail("");

    setName("");

    setPhone("");

} catch {

    notify("Не удалось отправить");

}

}

return (

    <div style={{ maxWidth: 760, margin: "0 auto" }}>

        <section className="card">

            <h2>Поддержка</h2>

```

```

    <p className="muted">Можно писать как гостем, так и из
кабинета (тогда сообщение привяжется к пользователю).</p>

    <form onSubmit={submit}>

        <div>

            <label>Тема</label>

            <input value={subject} onChange={(e) =>
setSubject(e.target.value)} placeholder="Например: Вопрос по
абонементу" />

        </div>

        <div>

            <label>Сообщение</label>

            <textarea value={body} onChange={(e) =>
setBody(e.target.value)} placeholder="Опиши проблему или вопрос"
/>

        </div>

        {!me && (

            <div className="grid" style={{ gridTemplateColumns:
"1fr 1fr", gap: 12 }}>

                <div>

                    <label>Email (опционально)</label>

                    <input value={email} onChange={(e) =>
setEmail(e.target.value)} placeholder="you@example.com" />

                </div>

                <div>

                    <label>Имя (опционально)</label>

                    <input value={name} onChange={(e) =>
setName(e.target.value)} placeholder="Имя" />

                </div>

                <div style={{ gridColumn: "1 / -1" }}>

                    <label>Телефон (опционально)</label>

                    <input value={phone} onChange={(e) =>
setPhone(e.target.value)} placeholder="+7..." />

                </div>

            </div>

        )}

    </form>

```

```

        </div>

    </div>

    )}

    <div className="row">

        <button className="btn primary" type="submit">

            Отправить

        </button>

        <span className="pill">{me ? "Отправка от аккаунта"
: "Отправка как гость"}</span>

    </div>

</form>

</section>

</div>

);

}

```

Файл «index.html»

```

<!doctype html>

<html lang="ru">

    <head>

        <meta charset="UTF-8" />

        <meta name="viewport" content="width=device-width, initial-
scale=1.0" />

        <title>Fitness Center</title>

    </head>

    <body>

        <div id="root"></div>

        <script type="module" src="/src/main.tsx"></script>

    </body>

```

Изм.	Лист	№ докум.	Подпись	Дата

</html>

Файл «vite.config.ts»

```
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";

export default defineConfig({
  plugins: [react()],
  server: {
    port: 5173,
    proxy: {
      "/api": {
        target: "http://localhost:3001",
        changeOrigin: true
      }
    }
  }
});
```

					ПГУ1.090301.47.001 ПЗ	Лист
						165
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение В
«Веб-приложение фитнес-центра»
Презентация

					ПГУ1.090301.47.001 ПЗ	<i>Лист</i>
						166
<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>		

Веб-приложение фитнес-центра.

АВТОР РАБОТЫ: СТУДЕНТ ГРУППЫ 22ВВП2 ТАРАСОВ Н.А.

НАУЧНЫЙ РУКОВОДИТЕЛЬ: ДОЦЕНТ КАФЕДРЫ ВТ СОРОКИН А.А.

ПЕНЗА, 2026.

Цели работы:

Целью работы являлась разработка системы для автоматизации основных процессов фитнес-центра, таких как запись клиентов на занятия, управление расписанием и обработка заявок.

Для достижения поставленной цели были решены следующие задачи:

- Анализ предметной области
- Проектирование архитектуры
- Разработка клиентской и серверной части
- Создание БД
- Тестирование системы

Результатом работы стало функционирующее веб-приложение фитнес-центра, обеспечивающее регистрацию пользователей, просмотр расписания, запись на занятия, работу личного кабинета клиента, административной панели и системы поддержки пользователей.

2

					ПГУ1.090301.47.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		167

Область применения:

Разработанное веб-приложение предназначено для автоматизации работы фитнес-центров, спортивных клубов и студий групповых тренировок.

Система позволяет:

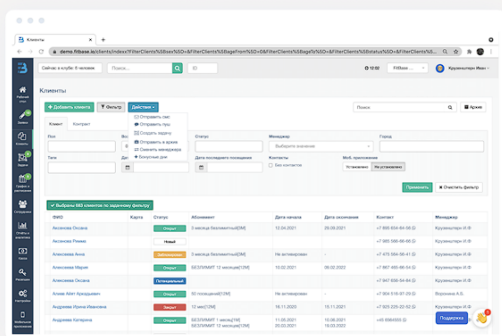
- организовать онлайн-запись клиентов на занятия;
- автоматизировать управление расписанием тренировок;
- вести учет заявок клиентов;
- управлять информацией о тренерах и услугах;
- обрабатывать обращения пользователей через систему поддержки.

Использование приложения позволяет **сократить** нагрузку на административный персонал, **повысить** качество обслуживания клиентов и **уменьшить** количество ошибок при обработке записей.

3

Анализ аналогов.

Fitbase



Преимущества:

CRM-система для фитнес-клубов;
онлайн-запись клиентов;
учет посещений и абонементов;
инструменты аналитики и отчетности.

Недостатки:

высокая стоимость внедрения и обслуживания;
ежемесячная абонентская плата;
избыточный функционал для небольших фитнес-центров.

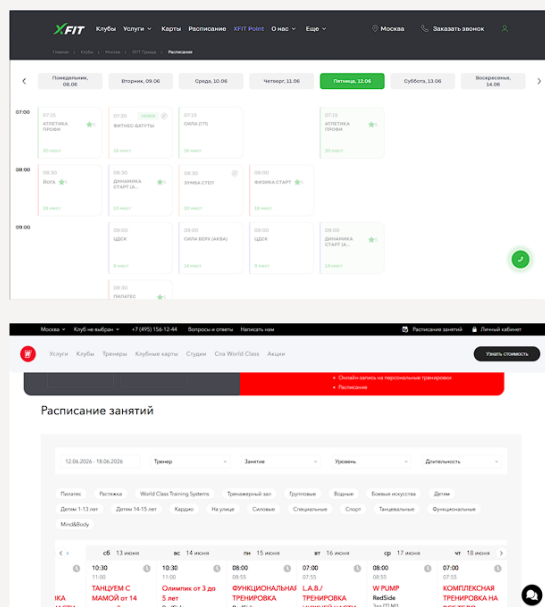
4

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.47.001 ПЗ

Лист

168



XFIT

Преимущества:

развитая цифровая инфраструктура;
удобная запись на тренировки;
высокий уровень автоматизации.

Недостатки:

предназначена только для сети XFIT;
не является самостоятельным программным продуктом;
отсутствует возможность использования сторонними организациями.

World Class

Преимущества:

современный пользовательский интерфейс;
удобное расписание занятий;
возможность записи на тренировки;
интеграция с сервисами фитнес-клуба.

Недостатки:

ориентировано исключительно на клиентов сети World Class;
отсутствует возможность развертывания в других фитнес-центрах;
закрытая экосистема.

5

Стек технологий:

1.Frontend-React



2.Backend-Node.js+Express



3.База данных-SQLite



4.ORM-Prisma



5.Авторизация-JWT



6

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.47.001 ПЗ

Лист

169

UML-диаграмма прецедентов.

Диаграмма прецедентов неавторизованного пользователя.

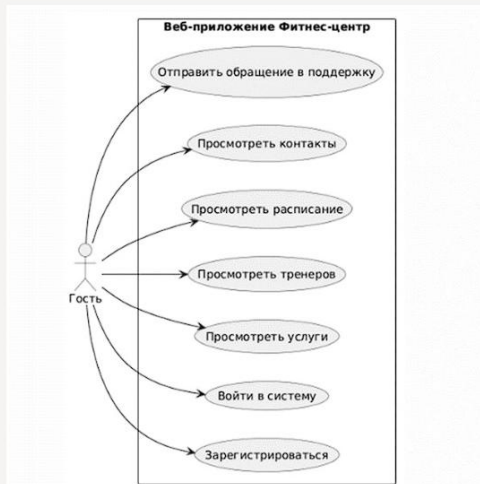
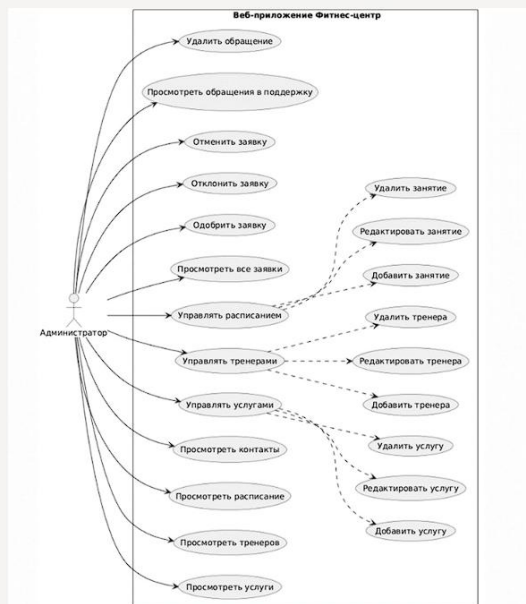


Диаграмма прецедентов авторизованного пользователя.



7

Диаграмма прецедентов администратора.



8

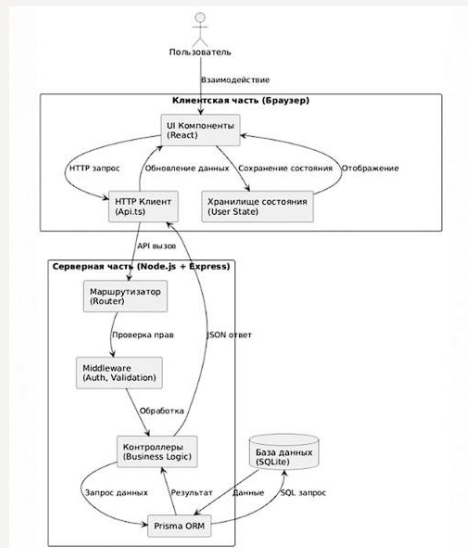
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.47.001 ПЗ

Лист

170

Архитектура системы.



Функциональная схема проекта.

9

Структура базы данных .

Основные сущности системы:

Пользователи

Тренеры

Услуги

Занятия

Заявки на запись

Обращения в поддержку

Связи между сущностями:

Пользователь может создавать заявки на занятия.

Пользователь может отправлять обращения в службу поддержки.

Каждая заявка связана с конкретным занятием.

Каждое занятие проводится определённым тренером.

Каждое занятие относится к выбранной услуге.

10

Изм.	Лист	№ докум.	Подпись	Дата

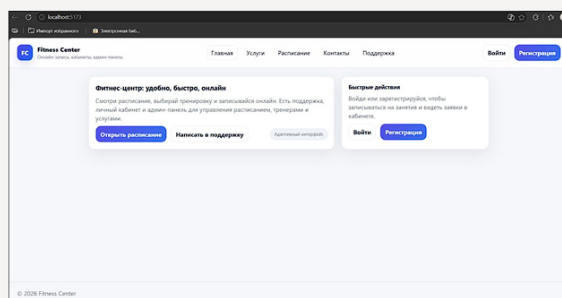
ПГУ1.090301.47.001 ПЗ

Лист

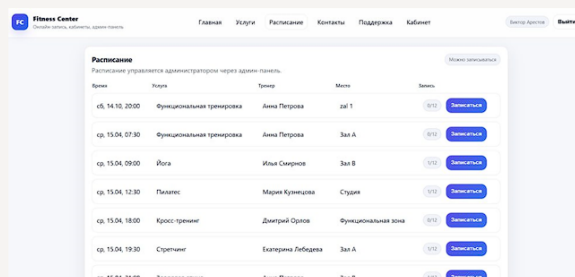
171

Интерфейс приложения.

Главная страница.

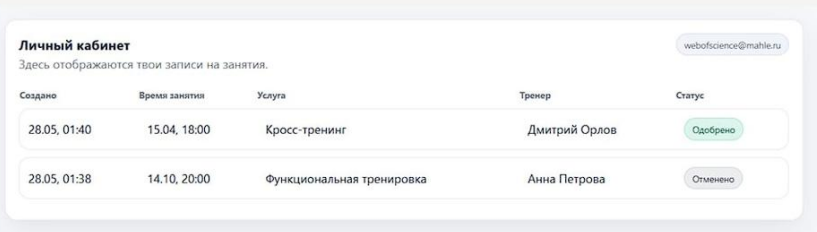


Страница расписания.

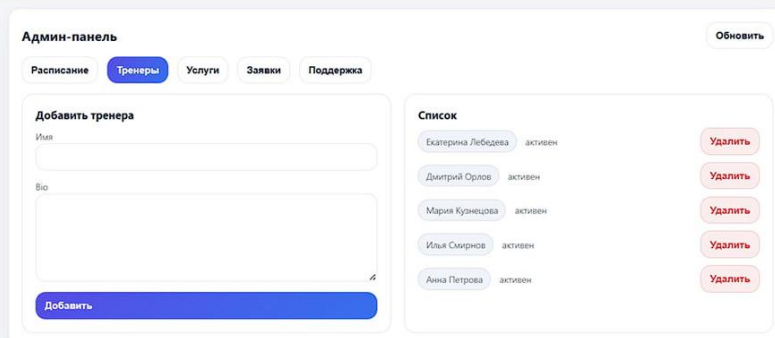


11

Личный кабинет



Админ-панель



12

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.47.001 ПЗ

Лист

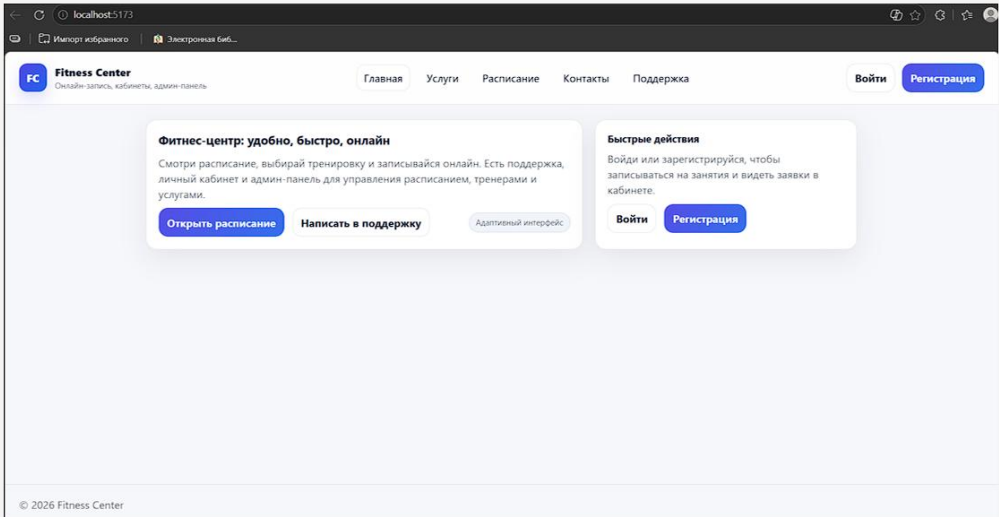
172

Результат разработки.

- ✓ Реализована система регистрации и авторизации пользователей
- ✓ Разработан модуль онлайн-записи на занятия
- ✓ Создан личный кабинет клиента для отслеживания заявок
- ✓ Реализована административная панель управления системой
- ✓ Добавлена система обработки обращений пользователей
- ✓ Обеспечена защита от двойной записи и разграничение прав доступа

13

Демонстрация работы веб-приложения фитнес-центра



14

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.47.001 ПЗ

Лист
173

Спасибо за внимание!

					ПГУ1.090301.47.001 ПЗ	Лист
						174
Изм.	Лист	№ докум.	Подпись	Дата		