

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

Кафедра

вычислительной техники

«Вычислительная техника»

Специальность: 09.05.01 «Применение и эксплуатация
автоматизированных систем специального назначения»
Специализация: «Эксплуатация вычислительных машин, комплексов,
систем и сетей специального назначения»

ДИПЛОМНЫЙ ПРОЕКТ

на тему

«Автоматизированная система психодиагностики
для задач профотбора»

Студент

Кокорин Кокорин Артем Александрович
(подпись, дата) 01.06.2026 (ФИО полностью)

Руководитель

Гурин Е.И. Гурин Е.И.
(подпись, дата) (фамилия, инициалы)

Консультанты:

ПО (наименование раздела) (подпись, дата) (фамилия, инициалы)

Нормоконтролёр

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

Рецензент

Мат. ЧИТ (должность, место работы) Севастьянов А.В.
(подпись, дата) (фамилия, инициалы)

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

Заведующий кафедрой

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

Работа защищена с отметкой отлично (протокол заседания ГЭК от

10 июня 2026 № 2)

Секретарь ГЭК

Гурин Е.И. Гурин

Пенза 2026

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

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

«Утверждаю»

Заведующий кафедрой

М.А. Митрохин

« 11 » 02 20 26 г.

ЗАДАНИЕ НА ДИПЛОМНЫЙ ПРОЕКТ

1. Студент Кокорин Артем Александрович гр. 21ВОЭ1 факультета Вычислительной техники

2. Тема работы «Автоматизированная система психодиагностики для задач профотбора»

Тема утверждена приказом ПГУ № 122/о от "11.02.2026" 2026 г.

3. Руководитель работы д.т.н., профессор кафедры ВТ Гурин Е.И.

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

Назначением разработки является создание веб-приложения для автоматизации психологического тестирования, обработки его результатов и формирования рекомендаций по профессиональной направленности. Система должна обеспечивать централизованное хранение данных о пользователях, методиках, попытках прохождения тестов, каталоге профессий и рекомендациях, а также предоставлять интерфейс для двух категорий пользователей: обычного пользователя и администратора. Исходными данными служат существующие аналоги систем профориентации, требования к разграничению прав доступа, стандартизированные психодиагностические методики и эталонные профили требуемых черт для каталога профессий.

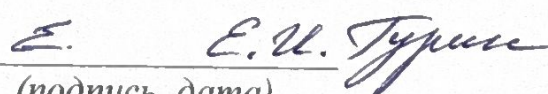
5. Перечень подлежащих разработке вопросов

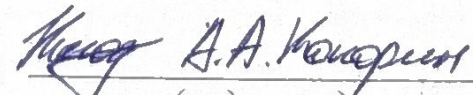
В ходе выполнения дипломного проекта необходимо провести анализ предметной области и существующих аналогов, обосновать актуальность разработки и сформулировать техническое задание; выполнить выбор технологического стека и системы управления базой данных; спроектировать архитектуру приложения, базу данных и пользовательский интерфейс с применением UML-диаграмм. Требуется разработать серверную часть на базе FastAPI с REST API, аутентификацией на основе JWT-токенов, модулем расчёта рекомендации по методу косинусной близости, а также клиентскую часть на Next.js и React с компонентной архитектурой, адаптивным интерфейсом и поддержкой двух ролей. По итогам прохождения всех методик должна формироваться рекомендация с возможностью скачивания итогового отчёта в формате PDF. Кроме того, необходимо обеспечить безопасность хранения персональных данных и провести тестирование системы. В составе пояснительной записки должны быть представлены разделы по безопасности жизнедеятельности и экономическому обоснованию проекта.

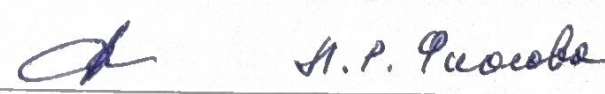
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области и анализ существующих аналогов	40	11.02.2026-16.02.2026	
2.	Анализ требований и постановка задачи	20	17.02.2026-20.02.2026	
3.	Выбор технологий проектирования и разработки (Python, FastAPI, SQLAlchemy, TypeScript, Next.js, React, MySQL)	30	21.02.2026-25.02.2026	
4.	Проектирование архитектуры системы, базы данных, модуля расчёта рекомендации и пользовательского интерфейса	60	26.02.2026-06.03.2026	
5.	Разработка серверной части (FastAPI, REST API, JWT)	80	07.03.2026-27.03.2026	
6.	Разработка клиентской части (Next.js, React, взаимодействие с API)	80	01.05.2026-11.05.2026	
7.	Интеграция компонентов, обеспечение безопасности персональных данных, тестирование	40	12.05.2026-17.05.2026	
8.	Оформление пояснительной записки (включая БЖД и экономику)	20	18.05.2026-21.05.2026	

Дата выдачи задания " 11 " 02 20 26 г.

Руководитель работы 
(подпись, дата)

Задание к исполнению принял студент 
(подпись, дата)

Работу к защите допустить
Декан факультета 
09.06.2026 (подпись, дата)

Реферат

Пояснительная записка содержит 200 листов, 44 рисунка, 16 таблиц, 26 источников и 3 приложения.

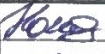
АВТОМАТИЗИРОВАННАЯ СИСТЕМА, ПСИХОДИАГНОСТИКА, ПРОФЕССИОНАЛЬНЫЙ ОТБОР, ТЕСТИРОВАНИЕ, РЕКОМЕНДАТЕЛЬНАЯ СИСТЕМА, БАЗА ДАННЫХ, ВЕБ-ПРИЛОЖЕНИЕ, КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА, REST API, КОСИНУСНАЯ БЛИЗОСТЬ, PYTHON, FASTAPI, SQLALCHEMY, NEXT.JS, REACT, TYPESCRIPT, MYSQL, JWT

Цель работы – разработка автоматизированной системы психодиагностики для задач профессионального отбора в виде веб-приложения, позволяющего проводить психологическое тестирование по нескольким стандартизированным методикам, автоматически обрабатывать результаты с контролем достоверности ответов и формировать персонализированные рекомендации по подходящим профессиям без участия специалиста-психолога.

Для достижения поставленной цели было разработано веб-приложение, отвечающее следующим задачам: анализ предметной области и существующих аналогов; обоснование выбора методик, технологий и СУБД; проектирование архитектуры, базы данных, модуля расчёта и интерфейса; реализация серверной и клиентской частей с тестированием, расчётом рекомендации и формированием отчёта; обеспечение безопасности персональных данных.

В результате разработана автоматизированная система психодиагностики, объединяющая пять методик единый профиль пользователя по девяти шкалам, контроль достоверности ответов, расчёт рейтинга профессий по косинусной близости, каталог из 37 профессий, личный кабинет, выгрузку отчёта в формате PDF.

ПГУ 1.090501.07.001 ПЗ

Изм	Лист	№ докум.	Подпись	Дата			
Разраб.		Кокорин А.А.		21.06.20	Автоматизированная система психодиагностики для задач профотбора Пояснительная записка	п	Лист
Пров.		Гурин Е.И.		21.06.20			Листов
							2
							200
Н. Контр.		Бычков А.С.		21.06.20		ФВТ, каф. ВТ гр. 21ВОЭ1	
Утв.		Митрохин М.А.		21.06.20			

Содержание

Введение.....	6
1 Автоматизированная система психодиагностики для задач профотбора.....	10
1.1 Анализ предметной области.....	10
1.1.1 Актуальность разработки.....	10
1.1.2 Анализ существующих аналогов.....	13
1.1.3 Обоснование актуальности разработки программы.....	20
1.1.4 Постановка задачи.....	23
1.2 Выбор технологий и средств разработки приложения.....	30
1.2.1 Выбор языка программирования для серверной части.....	30
1.2.2 Выбор языка программирования для клиентской части.....	35
1.2.3 Выбор СУБД.....	39
1.2.4 Выбор фреймворков и библиотек.....	41
1.2.5 Выбор среды разработки.....	44
1.3 Концепция системы и содержательная основа автоматизации.....	47
1.3.1 Концепция системы и состав психодиагностических методик.....	47
1.3.2 Структура шкал оценивания и обоснование ключевых показателей.....	50
1.3.3 Каталог профессий и эталонные профили требуемых черт.....	52
1.4 Проектирование информационной системы.....	55
1.4.1 Архитектура приложения.....	55
1.4.2 Проектирование базы данных.....	64
1.4.3 Проектирование модуля расчёта рекомендации.....	73
1.4.4 Проектирование интерфейса.....	77
1.5 Разработка автоматизированной системы.....	80
1.5.1 Реализация базы данных.....	80
1.5.2 Реализация серверной части.....	86
1.5.3 Реализация клиентской части.....	93
1.5.4 Реализация модуля расчёта и формирования отчёта.....	101
1.5.5 Обеспечение безопасности персональных данных.....	106

1.6 Тестирование и руководство пользователя.....	109
1.6.1 Описание интерфейса системы.....	109
1.6.2 Развёртывание и запуск системы.....	111
1.6.3 Руководство пользователя.....	116
1.6.4 Руководство администратора.....	125
1.6.5 Тестирование системы.....	130
2 Охрана труда.....	133
3 Экономика.....	136
Заключение.....	139
Список использованных источников.....	142
Приложение А – UML-диаграммы.....	145
Приложение Б – Листинг программы.....	155
Приложение В – Презентация.....	195

Перечень сокращений и обозначений

API (Application Programming Interface) – программный интерфейс для взаимодействия между клиентской и серверной частями системы

CSS (Cascading Style Sheets) – язык описания внешнего вида веб-страниц

FastAPI – веб-фреймворк на языке Python для разработки серверной части

HTML (HyperText Markup Language) – язык разметки веб-страниц

HTTP (HyperText Transfer Protocol) – протокол передачи данных между клиентом и сервером

HTTPS – защищённая версия протокола HTTP с использованием шифрования

JSON (JavaScript Object Notation) – текстовый формат обмена данными между клиентом и сервером

JWT (JSON Web Token) – стандарт защищённой передачи данных в виде подписанного токена, используется для аутентификации пользователей

MySQL – система управления реляционными базами данных

Next.js – фреймворк на основе библиотеки React для клиентской части

ORM (Object-Relational Mapping) – объектно-реляционное отображение

React – библиотека JavaScript для создания пользовательских интерфейсов

REST (Representational State Transfer) – архитектурный стиль построения веб

SHA (Secure Hash Algorithm) – семейство криптографических хеш-функций

SQL (Structured Query Language) – структурированный язык запросов к реляционным базам данных

UI (User Interface) – пользовательский интерфейс

UML (Unified Modeling Language) – унифицированный язык моделирования

URL (Uniform Resource Locator) – универсальный адрес ресурса в сети Интернет

UX (User Experience) – пользовательский опыт, удобство взаимодействия с системой

Введение

В современном обществе выбор профессии остаётся одним из наиболее значимых и сложных жизненных решений. Уже в школьные годы человек сталкивается с вопросами «Кто я?», «Кем быть?», «Что мне действительно интересно?». Статистика показывает, что многие молодые люди в России допускают серьёзные ошибки при выборе специальности: по различным оценкам, до 60 % выпускников вузов в итоге работают не по полученной профессии. Основные причины – ориентация преимущественно на престиж, мнение родителей или друзей, недостаточное знание рынка труда, игнорирование собственных интересов и психологических особенностей, а также слабая системная профориентационная поддержка. Каждому свойственно ошибаться, падать и вставать, но вопрос заключается не в том, стоит ли расстраиваться из-за неверного выбора, а в том, как сделать этот выбор максимально осознанным и соответствующим внутренним качествам человека.

Традиционно помощь в профессиональном самоопределении оказывают квалифицированные специалисты – психологи и профконсультанты. Они проводят тестирование, беседуют с человеком, анализируют его мотивацию, склонности и личностные качества. Однако такой подход имеет существенные ограничения: высокая субъективность интерпретации, зависимость от компетенций и загруженности специалиста, значительные временные и финансовые затраты, а также риск неискренних ответов тестируемых, стремящихся подобрать результаты под желаемую вакансию или ожидания общества. Чтобы облегчить задачу как для обычного человека, так и для специалистов, возникает потребность в создании автоматизированных систем, которые возьмут на себя часть этой работы. [1]

В последние годы активно развивается государственная политика в сфере профориентации.

С 2023 года внедряется Единая модель профессиональной ориентации обучающихся, обновляются методические рекомендации Минпросвещения, усиливается связь профориентации с прогнозами рынка труда и кадровой потребностью регионов. Это подчёркивает актуальность создания современных цифровых инструментов, способных дополнить или частично заменить ручной труд экспертов.

На рынке существуют различные программные продукты, решающие задачи тестирования и оценки персонала. Однако существующие системы часто имеют общие недостатки: ограничиваются простым подсчётом баллов по одному тесту, требуют обязательного участия психолога для глубокой интерпретации, либо распространяются на коммерческой основе и недоступны массовому пользователю. Кроме того, большинство из них опирается на одну методику, что снижает достоверность общей оценки: данные одного теста легко исказить, тогда как согласованные результаты нескольких независимых методик дают значительно более устойчивую картину.

Автоматизированная система представляет собой комплекс технических и программных средств, предназначенный для выполнения задач без прямого участия человека или с минимальным его вмешательством. Применительно к задаче профотбора это означает построение веб-приложения, которое самостоятельно проводит сразу несколько проверенных психодиагностических методик, формирует интегральный профиль пользователя по единому набору шкал, сравнивает его с заранее подготовленной базой профессий и выдаёт итоговую рекомендацию вместе с текстовым пояснением. При этом разные методики имеют разные веса в итоговом расчёте, а встроенный контроль достоверности отслеживает признаки искажения ответов и при необходимости предлагает пройти тестирование заново. Поскольку психологические особенности и профессиональные интересы человека могут меняться с течением жизни – в разное время мы раскрываем разные возможности, – возможность повторного прохождения тестирования и получения актуальных рекомендаций делает такой инструмент особенно ценным на протяжении всей карьеры. [2]

Таким образом, возникает объективная потребность в разработке автоматизированной системы психодиагностики, которая минимизирует влияние человеческого фактора, повышает достоверность оценки за счёт сочетания нескольких методик, делает процесс доступным для широкой аудитории и способствует более осознанному профессиональному самоопределению. Именно поэтому выбор темы «Автоматизированная система психодиагностики для задач профотбора» в качестве выпускной квалификационной работы является актуальным.

Разработка веб–приложения, объединяющего пять психодиагностических методик, автоматический расчёт интегрального профиля, контроль достоверности ответов и детерминированное сопоставление с базой профессий, позволяет получить готовый программный продукт, способный повысить объективность и доступность профориентационной диагностики для широкого круга пользователей.

Целью выпускной квалификационной работы является разработка автоматизированной системы психодиагностики для задач профотбора в виде веб–приложения, реализующего полный цикл от регистрации пользователя до формирования итогового отчёта с рекомендацией.

Для достижения поставленной цели необходимо решить следующие задачи:

- провести исследование предметной области и анализ существующих программных решений в сфере автоматизированной психодиагностики и профотбора;
- выбрать и обосновать состав психодиагностических методик, шкалы оценивания, технологии разработки и систему управления базой данных;
- спроектировать архитектуру веб–приложения, структуру базы данных и пользовательский интерфейс;
- разработать алгоритм детерминированного расчёта интегрального профиля пользователя на основе ответов по нескольким методикам и его сравнения с эталонными профилями профессий;

- реализовать модуль психологического тестирования с автоматическим подсчётом результатов по стандартизированным ключам и контролем достоверности ответов;
- реализовать клиентскую и серверную части веб–приложения, включая личный кабинет пользователя, каталог профессий, формирование текстового отчёта и административную панель со статистикой;
- обеспечить безопасность хранения персональных данных пользователей;
- провести тестирование разработанной системы и оценить её работоспособность.

Объектом исследования выступает процесс автоматизированной психодиагностики применительно к задачам профессионального отбора и профориентации.

Предметом исследования являются методы и программные средства построения многомодульной автоматизированной системы психодиагностики на базе веб–приложения.

Практическая значимость работы заключается в создании доступного цифрового инструмента, позволяющего пользователям самостоятельно и объективно оценивать свои профессиональные склонности, получать персонализированные рекомендации по выбору карьерного пути и развитию необходимых качеств. Разработанная система поможет минимизировать риски ошибочного выбора профессии, повысить осознанность в профессиональном самоопределении, а также окажет поддержку работодателям и психологам в задачах первичного профотбора за счёт снижения субъективности и временных затрат. Применение единого набора шкал и нескольких независимых методик повышает достоверность итоговой рекомендации по сравнению с системами, опирающимися на одну методику.

1 Автоматизированная система психодиагностики для задач профотбора

1.1 Анализ предметной области

1.1.1 Актуальность разработки

Проблема профессионального самоопределения и правильного выбора карьеры существует на протяжении всей истории человечества, поскольку от адекватной оценки собственных способностей, интересов и мотивации напрямую зависит удовлетворённость жизнью, продуктивность труда и вклад в общество. В современном мире, когда рынок труда быстро меняется под влиянием технологий и экономических сдвигов, эта проблема приобретает особую остроту.

Проблема соответствия человека избираемой профессии имеет глубокие исторические корни. Ещё в доиндустриальную эпоху существовали механизмы передачи профессиональных навыков, от ремесленных цехов до семейных династий, где вопрос пригодности решался через длительное наблюдение и ученичество. С развитием психологической науки в XX веке сформировался научный подход к оценке профессиональной пригодности, основанный на стандартизированных методиках. К настоящему времени создано множество валидных инструментов: личностные опросники Big Five, методики изучения профессиональных интересов Климов, Холланд, мотивационные шкалы и тесты ценностных ориентаций Рокича, а также методики оценки способностей по предметным областям. Однако наличие богатого арсенала средств диагностики не решает главной задачи – помощи человеку в осознанном выборе, поскольку эти средства существуют разрозненно и требуют квалифицированной интеграции. [3]

Современная ситуация на рынке труда характеризуется парадоксом: при формальном росте доступности профориентационных услуг качество профессионального самоопределения остаётся низким.

					ПГУ 1.090501.07.001 ПЗ	Лист
						10
Изм.	Лист	№ докум.	Подпись	Дата		

Исследования показывают, что значительная часть выпускников учебных заведений испытывает трудности с трудоустройством по специальности. По данным социологических опросов, около половины работающих россиян заняты в сферах, не связанных с полученным образованием. Особенно остро эта проблема стоит среди молодёжи: многие выпускники признаются, что при выборе вуза ориентировались на внешние факторы – советы родителей, мнение друзей, представления о престижности, – а не на собственные склонности и способности. Это приводит к разочарованию в профессии, снижению продуктивности и, в конечном счёте, к необходимости переобучения и смены карьеры.

Анализ существующей практики профориентационной работы выявляет ряд системных недостатков. Во-первых, наблюдается фрагментарность диагностических подходов: различные методики применяются изолированно, их результаты не интегрируются в целостную картину. Во-вторых, интерпретация полученных данных требует участия квалифицированного специалиста–психолога, что делает процедуру дорогостоящей и труднодоступной для широких слоёв населения. В-третьих, даже при обращении к специалисту сохраняется элемент субъективности, обусловленный различиями в профессиональной подготовке и личном опыте консультантов. В-четвёртых, прохождение каждой методики у отдельного специалиста занимает значительное время, что плохо согласуется с потребностью школьников и студентов в массовой первичной диагностике в сжатые сроки.

Особую проблему представляет достоверность получаемых результатов. Практика показывает, что при прохождении тестирования в условиях реального отбора (при приёме на работу, поступлении в учебное заведение) значительная часть респондентов стремится представить себя в более выгодном свете, сознательно или подсознательно искажая ответы. Стандартные методики предусматривают шкалы достоверности и контрольные вопросы, позволяющие выявлять такие искажения, однако их корректная интерпретация в ручном режиме также требует участия специалиста и не всегда применяется на практике. [4]

Автоматизированная проверка согласованности ответов в рамках одной методики (например, расхождения между объективной оценкой и самооценкой по одним и тем же показателям) или между разными методиками способна выявлять признаки недобросовестных ответов без участия эксперта.

Существующие программные продукты, позиционируемые как автоматизированные системы профотбора, лишь частично решают обозначенные проблемы. Большинство из них реализуют функцию подсчёта баллов по одной методике, но не предлагают глубокой аналитики и персонализированных рекомендаций. Те же системы, которые претендуют на комплексный подход, либо распространяются на коммерческой основе и недоступны массовому пользователю, либо требуют дополнительного сопровождения со стороны психологов. Таким образом, сохраняется потребность в создании доступного инструмента, который объединил бы несколько проверенных диагностических методик в едином сценарии, автоматизировал их интерпретацию и обеспечил пользователя понятными, персонализированными рекомендациями.

Развитие современных веб-технологий создаёт техническую базу для решения этой задачи. Реализация системы в виде веб-приложения снимает необходимость установки специального программного обеспечения, обеспечивает доступ с любого устройства с браузером и позволяет проходить тестирование в удобное время. Объединение нескольких независимых психодиагностических методик в одном программном продукте, приведение их результатов к общему набору шкал и применение детерминированного алгоритма сравнения с заранее подготовленной базой эталонных профилей профессий обеспечивают объективность интерпретации без обращения к специалисту. При этом разные методики получают разные веса в итоговом расчёте, что позволяет учитывать их различную диагностическую ценность для задачи профотбора. Автоматизация открывает возможность многократного использования системы без дополнительных затрат, что особенно важно в условиях массовой профориентационной работы со школьниками и студентами.

Решение, разработанное в рамках выпускной квалификационной работы, призвано исключить вышеперечисленные проблемы. Автоматизированная система психодиагностики реализована в виде веб–приложения с бесплатной регистрацией, объединяющего пять психодиагностических методик, единый личный кабинет с отслеживанием прогресса, автоматический контроль достоверности ответов, детерминированное сопоставление профиля пользователя с базой профессий и формирование скачиваемого отчёта с пояснением. Это обеспечивает доступность инструмента для всех категорий пользователей – от школьников, стоящих перед первым выбором, до взрослых людей, планирующих смену профессиональной траектории.

1.1.2 Анализ существующих аналогов

Анализ существующих аналогов является неотъемлемым этапом разработки любого программного продукта, позволяющим определить текущее состояние рынка, выявить сильные и слабые стороны конкурирующих решений, а также сформулировать требования к создаваемой системе. Для автоматизированной системы психодиагностики в сфере профотбора особенно важно изучить не только функциональные возможности, но и ценовую политику, целевую аудиторию, а также методологические подходы, используемые в аналогичных проектах. Это даёт возможность не повторять уже существующие недостатки и сосредоточиться на создании действительно востребованного и универсального инструмента.

Среди аналогов систем автоматизации профотбора существуют различные онлайн–платформы и сервисы, предлагающие тестирование и рекомендации по профессиям. Ниже приведён краткий анализ ключевых решений с указанием их основных возможностей и недостатков.

Profilum.ru предоставляет: комплексную систему профориентации для школ и регионов (диагностика интересов и способностей, каталог 500+ профессий, уроки по приоритетным отраслям, индивидуальные рекомендации по образовательным траекториям, прогнозы на основе данных рынка труда).

Недостатки: ориентирована преимущественно на школьников и региональные внедрения (не универсальна для взрослых), требует организационной поддержки и внедрения в школах и региональных органах власти, недоступна для самостоятельного использования индивидуальным пользователем. Сайт отображен на рисунке 1.

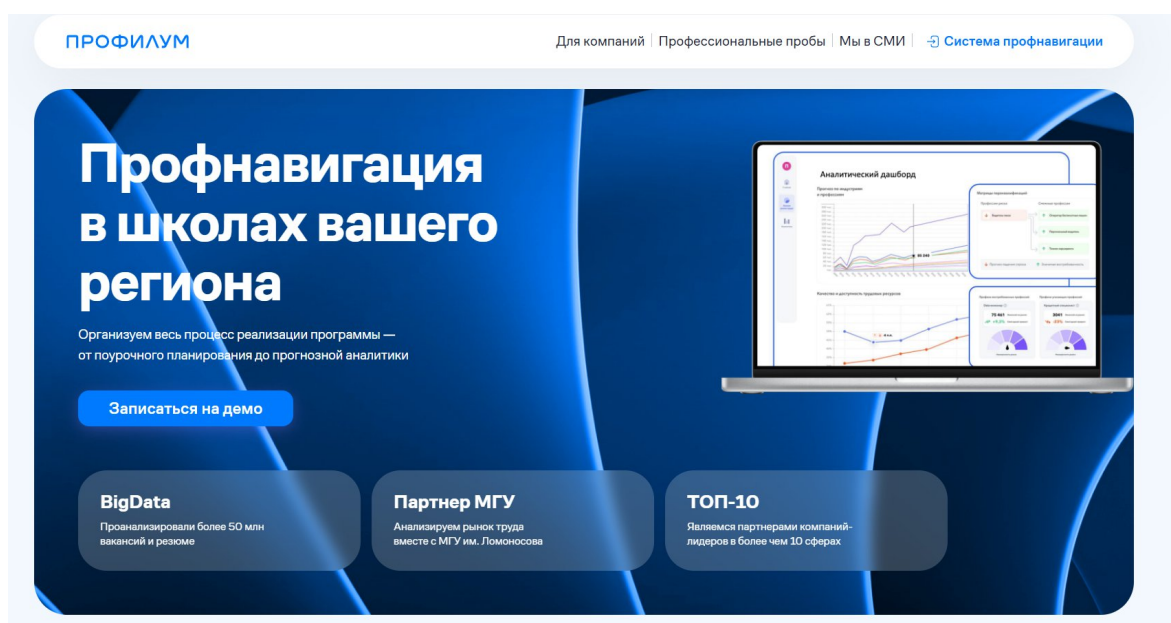


Рисунок 1 – Profilum.ru

Proforientator.ru предоставляет: бесплатные онлайн-тесты на личность, интересы, способности и роли в профессии (например, «Личность и профессии», «Профперспектива», «Гуманитарий или технарь»), автоматические рекомендации подходящих профессий и направлений образования.

Недостатки: тесты простые и самостоятельные, без встроенных шкал достоверности (легко поддаются искажению под желаемый результат), рекомендации остаются базовыми без комплексного интегрального профиля; результаты разных тестов не объединяются в единый профиль и не учитывают динамику изменений со временем. Сайт отображен на рисунке 2.

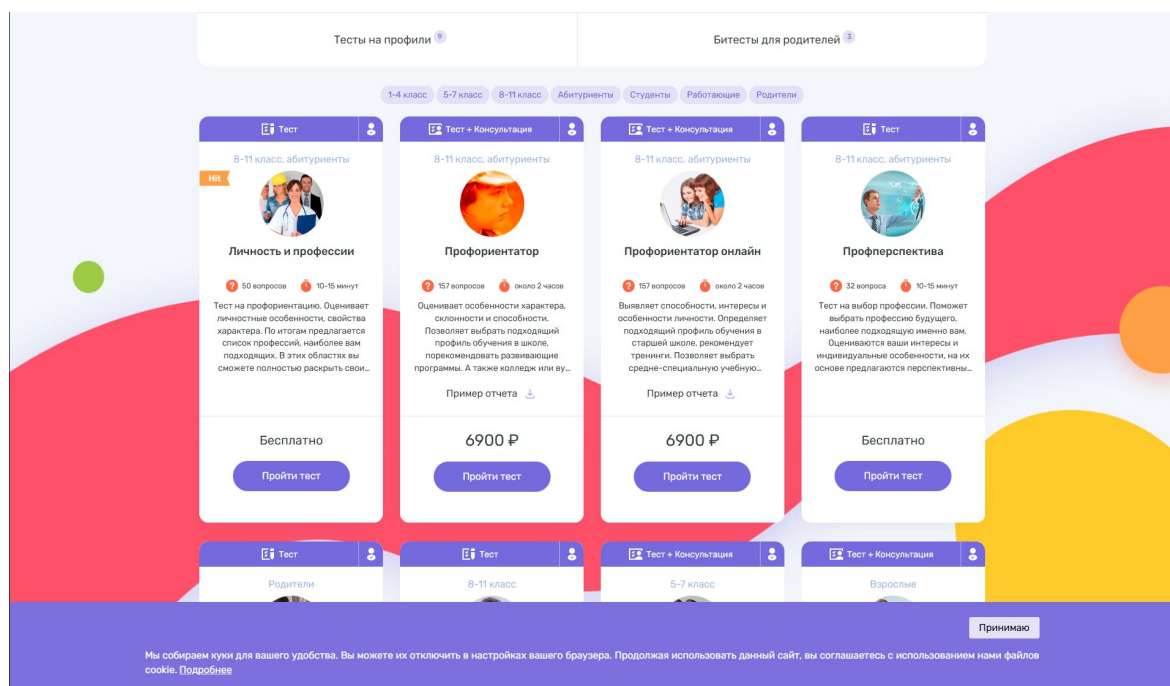


Рисунок 2 – Proforientator.ru

Skillbox Proftest предоставляет: бесплатный быстрый тест (8 минут) на личностные качества, сильные стороны, интересы и ожидания от карьеры, выдаёт 4 рекомендованные профессии в 9 сферах (с акцентом на IT, дизайн, маркетинг, креатив), плюс консультация специалиста и бот для уточнений.

Недостатки: поверхностный и скоростной формат, сильный уклон в сторону профессий, связанных с собственными курсами платформы (что вносит коммерческую предвзятость), нет механизмов выявления неискренности, анализ не глубокий и не учитывает многомерный психологический профиль. Сайт отображен на рисунке 3.

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.07.001 ПЗ

Лист

15

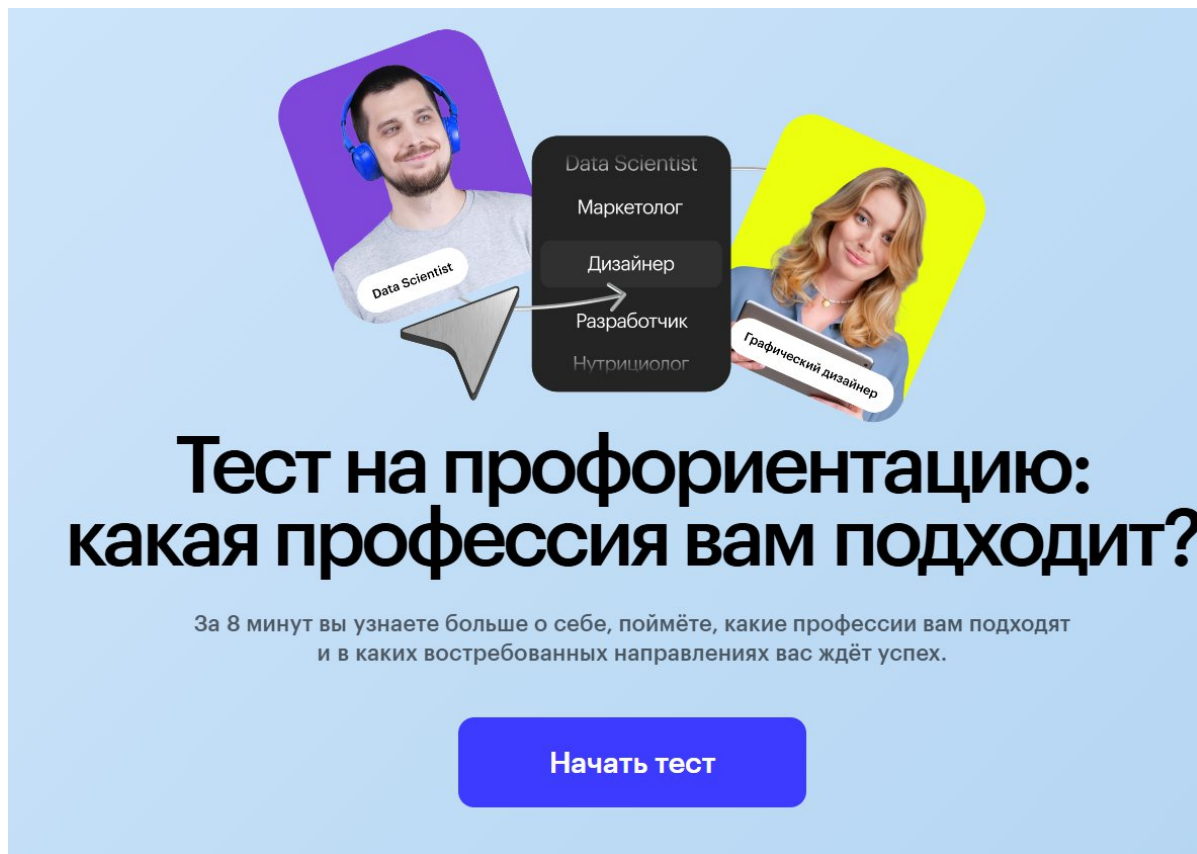


Рисунок 3 – Skillbox ProfTest

Нейропрофориентация.рф предоставляет: платформу на основе технологии Persona Talent (визуальный нейропрофайлинг за 2 минуты), анализ когнитивных талантов, мотивации и поведения, персонализированные рекомендации по 35+ профессиям с научным обоснованием (заявленная точность 87,2 %), корректировка искажений ответов.

Недостатки: распространяется на платной основе (стоимость зависит от партнёров, для индивидуальных пользователей в большинстве случаев платная), фокус преимущественно на школьниках и студентах, требует специального доступа через партнёров, что ограничивает массовую доступность. Сайт отображен на рисунке 4.

					ПГУ 1.090501.07.001 ПЗ	Лист
						16
Изм.	Лист	№ докум.	Подпись	Дата		

Neuro

AI

/ технология /

Откройте новую эру
Искусственного
Интеллекта

«НейроПрофориентация» — это не просто сервис, это комплексное решение, которое помогает учащимся и их родителям принимать взвешенные решения относительно будущей карьеры.

Эта инновационная платформа разработана с применением новейших технологий искусственного интеллекта и нейроскрининга Persona Talent, что позволяет осуществлять глубокий анализ когнитивных талантов, навыков и поведения человека.

➤ Узнать про возможности для Вас

Рисунок 4 – Нейропрофориентация.рф

Proektoria.online предоставляет: платформу для школьников и молодёжи с тестами, видео о профессиях, интерактивными материалами и рекомендациями по выбору карьеры (государственно–поддерживаемый проект).

К числу существенных ограничений рассматриваемой платформы следует отнести, во-первых, её преимущественную направленность на подростковую возрастную группу и решение задач в рамках школьной профориентации. Во-вторых, формируемые системой рекомендации носят в значительной степени обобщённый и образовательно-ознакомительный характер, что выражается в отсутствии механизмов углублённого анализа профилей пользователей, а также процедур проверки достоверности предоставляемых ответов. В-третьих, отмеченные особенности делают платформу непригодной для применения взрослыми пользователями, уже обладающими сформированным практическим опытом трудовой деятельности. Сайт отображен на рисунке 5.

					ПГУ 1.090501.07.001 ПЗ	Лист
						17
Изм.	Лист	№ докум.	Подпись	Дата		

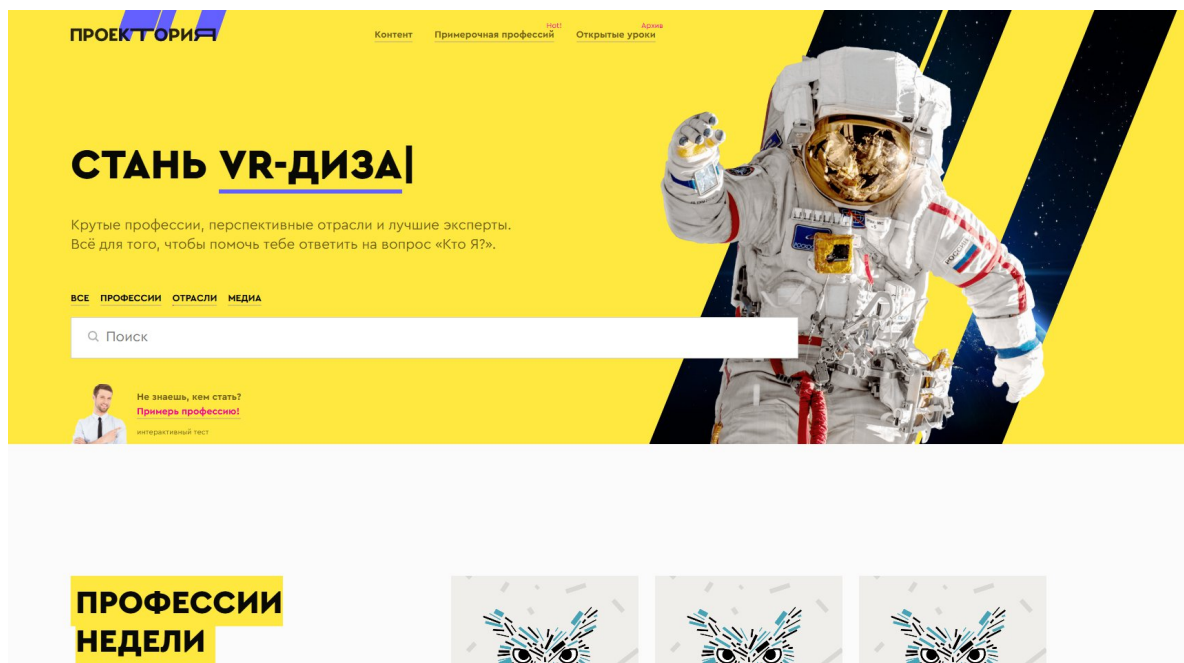


Рисунок 5 - Proektoria.online

Postupi.online (тест «Выбор профессии») предоставляет: сверхкороткий бесплатный диагностический инструмент, прохождение которого занимает не более одной минуты и включает всего два вопроса, касающихся предпочтений по объекту деятельности и виду деятельности. В основе теста лежит методика Е. А. Резапкина. По итогам прохождения система формирует перечень подходящих профессий, а также выдаёт рекомендации по программам обучения, соответствующим указанным профилям.

Недостатки: крайне упрощённый формат процедуры, ограничивающийся всего двумя вопросами, что обуславливает поверхностный и обобщённый характер выдаваемых рекомендаций. В тесте полностью отсутствуют механизмы проверки искренности ответов пользователя, а также элементы углублённого психологического анализа индивидуальных особенностей. Вследствие указанных ограничений результат теста является легко предсказуемым, и пользователь, при желании, может подобрать ответы, направленные на получение той или иной желаемой профессии, что снижает диагностическую ценность инструмента. Сайт отображен на рисунке 6.

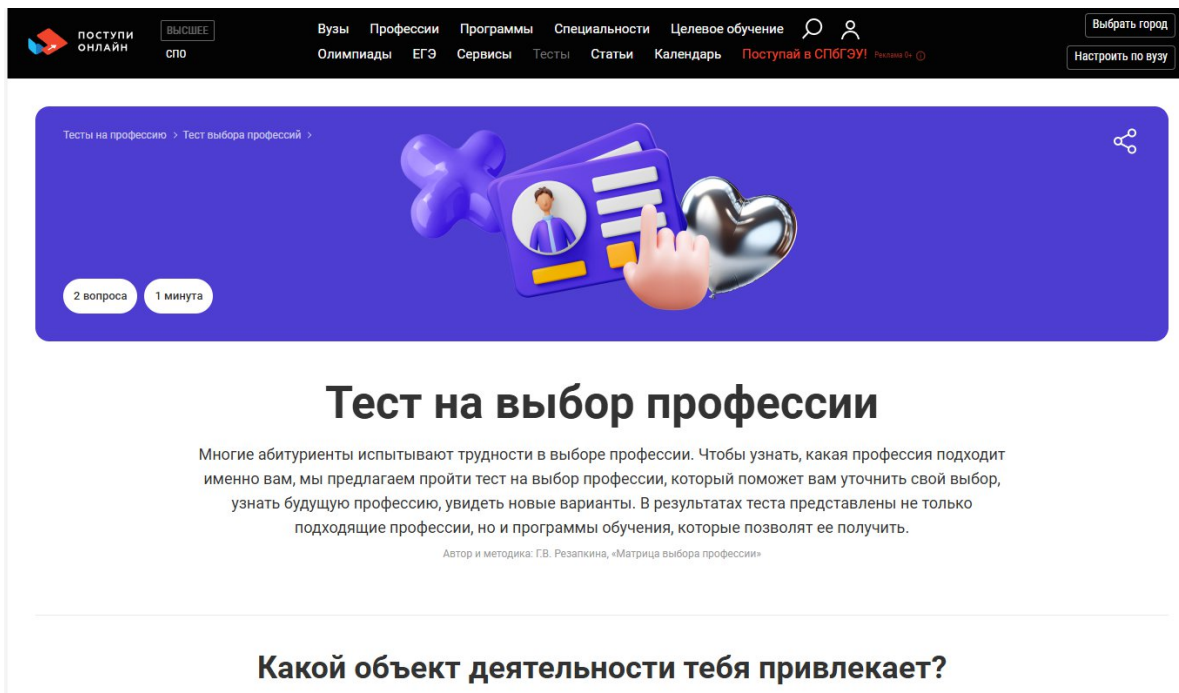


Рисунок 6 – Postupi.online

Trudvsem.ru (раздел профориентации) предоставляет: государственную платформу с бесплатными тестами на интересы, ценностные ориентации и индивидуальные особенности, автоматические рекомендации по направлениям и связь с вакансиями и рынком труда.

Недостатки: применяемые в рамках данного подхода тестовые методики характеризуются базовым уровнем сложности, а их обработка осуществляется по упрощённым алгоритмам, не предполагающим многофакторного анализа. Рекомендации, формируемые по итогам тестирования, в большинстве случаев ограничиваются указанием на обобщённые группы профессий, без выделения конкретных специальностей или узких направлений. При этом пользователю не предоставляется ни глубокого персонализированного профиля, ни развёрнутых советов по развитию профессионально значимых качеств, необходимых для успешной реализации в выбранной сфере. Вследствие указанных ограничений получаемые выводы обладают низкой прогностической ценностью и не могут служить надёжной основой для принятия карьерных решений. Сайт отображен на рисунке 7.



Рисунок 7 – Trudvsem.ru (раздел профориентации)

Проведённый анализ выявляет явные недостатки, характерные для ряда специализированных интернет-ресурсов в области профориентационного тестирования.

1.1.3 Обоснование актуальности разработки программы

Разработанная в данной работе автоматизированная система психодиагностики обладает рядом преимуществ по сравнению с существующими аналогами. Главным из них является полностью бесплатный доступ для всех категорий пользователей – от школьников и студентов до взрослых людей, планирующих смену карьеры. В отличие от большинства конкурентов, предлагающих либо платные расширенные отчёты, либо требующих оплаты за консультацию психолога, либо распространяемых через коммерческих партнёров и образовательные организации, разрабатываемая система не взимает платы за регистрацию, прохождение тестов, формирование рекомендации и скачивание итогового отчёта.

Система последовательно закрывает каждую из проблем, выявленных при анализе существующих решений.

Первая проблема заключается в фрагментарности диагностических подходов. Большинство бесплатных сервисов реализует только одну методику, что не позволяет составить целостную картину личности и интересов пользователя.

В разрабатываемой системе объединены пять психодиагностических методик: стартовый предметный тест «Школьные ориентиры» (18 школьных предметов с тремя шкалами), дифференциально–диагностический опросник Е. А. Климова (ДДО), методика профессиональных интересов Дж. Холланда (RIASEC), пятифакторный личностный опросник Big Five и тест ценностных ориентаций М. Рокича. Результаты всех методик сводятся к единому набору шкал черт личности, что позволяет использовать данные разных тестов согласованно, а не изолированно.

Вторая проблема связана с отсутствием контроля достоверности ответов. В большинстве бесплатных аналогов пользователь может подобрать ответы под желаемый результат без каких–либо последствий. В разрабатываемой системе реализован встроенный модуль контроля достоверности: при прохождении стартового теста отслеживается среднее расхождение между объективной оценкой по предмету и самооценкой способностей пользователя, а также общий уровень мотивации. При превышении пороговых значений система формирует предупреждение и предлагает пройти тестирование повторно. Это снижает риск получения недостоверного результата без обращения к специалисту–психологу.

Третья проблема состоит в поверхностности анализа и выдаче «сырых» баллов. Многие сервисы ограничиваются выдачей суммарных баллов по тесту без интерпретации. В разрабатываемой системе на основе ответов пользователя автоматически формируется интегральный профиль личности по единому набору шкал, который затем сравнивается с заранее подготовленными эталонными профилями профессий по методу косинусной близости. Каждой методике при агрегировании присвоен собственный весовой коэффициент, отражающий её диагностическую ценность для задачи профотбора. В итоге пользователь получает не «сырые» баллы, а упорядоченный по убыванию релевантности рейтинг подходящих профессий с указанием ключевых совпадений по чертам личности и автоматически сформированным текстовым пояснением.

Четвёртая проблема выражается в зависимости от человеческого фактора при интерпретации. В традиционной профконсультации результат тестирования зависит от компетенций, опыта и текущей загруженности специалиста. В разрабатываемой системе вся интерпретация выполняется детерминированным алгоритмом: при одинаковых ответах результат всегда будет одинаковым, без субъективного разброса.

При этом сохраняется возможность участия специалиста – в системе предусмотрена административная панель со статистикой прохождения тестов и возможностью выгрузки отчёта по любому пользователю, что делает её удобной не только для самостоятельного использования, но и в работе профконсультантов и кадровых служб.

Пятая проблема – узкая целевая аудитория аналогов. Государственные и образовательные платформы ориентированы преимущественно на школьников и студентов и не учитывают потребности взрослых пользователей. Разрабатываемая система рассчитана на широкую аудиторию – от школьников старших классов до взрослых пользователей, рассматривающих смену сферы деятельности. Каталог профессий содержит как массовые, так и узкоспециализированные направления; описание каждой профессии включает уровень популярности на российском рынке труда, преимущества, недостатки, направления подготовки и типичные затруднения, что одинаково полезно для всех категорий пользователей.

Шестая проблема – барьеры доступа и угрозы персональным данным. Часть аналогов требует длительной регистрации с подтверждением учётной записи через образовательную организацию или партнёра. В разрабатываемой системе предусмотрена простая бесплатная регистрация по электронной почте, доступ через любой современный веб-браузер без установки дополнительного программного обеспечения, а также возможность повторного прохождения тестирования в любой момент – это важно, поскольку психологические особенности и профессиональные интересы человека меняются с течением жизни.

Хранение персональных данных организовано с учётом современных требований: пароли хранятся в виде криптографических хешей, для авторизации используются токены с ограниченным сроком действия, в административной панели применяется маскирование адресов электронной почты, а удаление учётной записи пользователя реализовано без потери целостности обезличенной статистики.

Таким образом, разрабатываемая система заполняет существующую нишу доступного, объективного и комплексного инструмента автоматизированной психодиагностики для задач профотбора, сочетающего одновременно бесплатность, объединение нескольких проверенных методик, единый интегральный профиль, автоматический контроль достоверности и универсальность по целевой аудитории – то, что в существующих аналогах присутствует только по отдельности, но не в одном программном продукте.

1.1.4 Постановка задачи

Требуется разработать автоматизированную систему психодиагностики для задач профотбора, представляющую собой веб–приложение, которое будет соответствовать следующим функциональным требованиям.

Модуль регистрации и управления пользователями:

- регистрация новых пользователей с указанием минимально необходимых данных (адрес электронной почты, никнейм, пароль, пол, возраст);
- авторизация зарегистрированных пользователей в системе по электронной почте и паролю;
- выпуск и проверка токенов сессии с ограниченным сроком действия;
- выход из системы;
- хранение паролей в защищённом виде (хеширование);
- возможность удаления собственной учётной записи пользователем без потери целостности обезличенной статистики.

Модуль психологического тестирования:

- отображение каталога доступных психодиагностических методик с кратким описанием и индикатором статуса прохождения каждой методики для текущего пользователя («не начата», «в процессе», «пройдена»);
- последовательное предъявление вопросов выбранной методики с фиксацией ответов пользователя;
- автоматический подсчёт результатов тестирования по стандартизированным ключам для каждой методики;
- реализация встроенного контроля достоверности ответов, выявляющего значительное расхождение между объективной оценкой и самооценкой пользователя, а также аномально низкий уровень мотивации;
- информирование пользователя о возможной недостоверности результата и предложение пройти тестирование повторно;
- сохранение истории всех попыток прохождения тестов;
- возможность повторного прохождения любой методики для отслеживания динамики изменений во времени.

База профессий:

- ведение структурированной базы профессий, актуальных для российского рынка труда;
- для каждой профессии хранение описания, перечня преимуществ и недостатков, рекомендуемых направлений подготовки, типичных затруднений и оценки уровня популярности на рынке труда;
- для каждой профессии хранение эталонного набора требуемых черт личности с весовыми коэффициентами, используемого при расчёте рекомендации;
- предоставление пользователю алфавитного каталога профессий с поиском по названию и возможностью просмотра подробной карточки выбранной профессии.

Модуль расчёта рекомендации:

- формирование интегрального профиля пользователя по единому набору шкал черт личности на основе совокупности результатов всех пройденных методик;
- учёт весовых коэффициентов каждой методики при агрегировании итогового профиля;
- сопоставление полученного профиля с эталонными профилями профессий из базы по методу косинусной близости;
- формирование персонализированного рейтинга, включающего список наиболее подходящих профессий с указанием степени соответствия и ключевых совпадений по чертам;
- автоматическое формирование текстового пояснения к рейтингу.

Личный кабинет пользователя:

- отображение анкетных данных пользователя и общего прогресса по методикам (количество пройденных тестов от общего числа);
- просмотр истории прохождений с детализацией результатов по каждой методике;
- просмотр итоговой рекомендации с рейтингом подходящих профессий и пояснением;
- сохранение и скачивание итогового отчёта в формате PDF для личного использования или предоставления работодателю.

Административный модуль:

- разделение пользователей по ролям (обычный пользователь и администратор);
- просмотр списка зарегистрированных пользователей с маскированием персональных данных (адресов электронной почты);

					ПГУ 1.090501.07.001 ПЗ	Лист
						25
Изм.	Лист	№ докум.	Подпись	Дата		

- отображение обезличенной статистики использования системы (количество регистраций, количество начатых и завершённых тестов, средний процент завершённости);
- просмотр каталога системных методик;
- добавление, изменение и удаление позиций каталога профессий;
- выгрузка итогового отчёта по любому пользователю в формате PDF для работы профконсультанта.

Разрабатываемая автоматизированная система должна соответствовать следующим техническим требованиям:

- тип приложения: клиент–серверное веб–приложение с адаптивным интерфейсом, доступное через современные веб–браузеры на различных устройствах (персональные компьютеры, планшеты, смартфоны).
- клиентская часть: TypeScript, Next.js (React);
- серверная часть: Python, FastAPI;
- база данных: MySQL.
- архитектура взаимодействия: обмен данными между клиентом и сервером по протоколу HTTP в формате JSON через REST API; аутентификация на основе токенов сессии с ограниченным сроком действия.
- удобный и интуитивно понятный интерфейс, обеспечивающий быстрое ориентирование пользователя; расположение элементов должно соответствовать логике процесса тестирования; поддержка масштабирования интерфейса под разные разрешения экранов (адаптивная вёрстка).
- производительность: время отклика системы при выполнении основных операций (авторизация, загрузка тестов, расчёт рекомендации, формирование отчёта) не должно превышать 10 секунд.
- система должна функционировать круглосуточно, обеспечивая доступ пользователей из любой точки при наличии подключения к интернету.
- безопасность:

- хранение паролей в виде криптографических хешей;
- передача токена сессии в каждом запросе с проверкой прав доступа;
- ролевое разграничение доступа к административным функциям;
- маскирование персональных данных в административной панели;
- обязательное информирование пользователя о том, что результаты тестирования и сформированные рекомендации носят исключительно справочно–информационный характер, не являются окончательным диагнозом или гарантией профессиональной пригодности, и в сложных жизненных или карьерных ситуациях рекомендуется обращаться к квалифицированному специалисту–психологу;
- соответствие требованиям Федерального закона № 152–ФЗ «О персональных данных» (обеспечение конфиденциальности и защиты персональных данных пользователей).

Требования к клиентскому оборудованию:

- операционная система: любая с поддержкой современного веб–браузера (Windows, macOS, Linux, Android, iOS);
- браузер: актуальные версии Google Chrome, Mozilla Firefox, Safari, Microsoft Edge, Opera, Yandex Browser;
- наличие подключения к сети Интернет со скоростью не менее 1 Мбит/с.

Требования к серверному оборудованию (минимальные):

- процессор: 2 ядра, 2 ГГц;
- оперативная память: не менее 4 ГБ;
- дисковое пространство: не менее 20 ГБ (под систему и базу данных);
- подключение к сети Интернет.

Разрабатываемая система должна быть готова к опытной эксплуатации, пройти тестирование на соответствие функциональным требованиям и обеспечить решение задач, сформулированных в цели работы.

Диаграмма вариантов использования: Пользователь

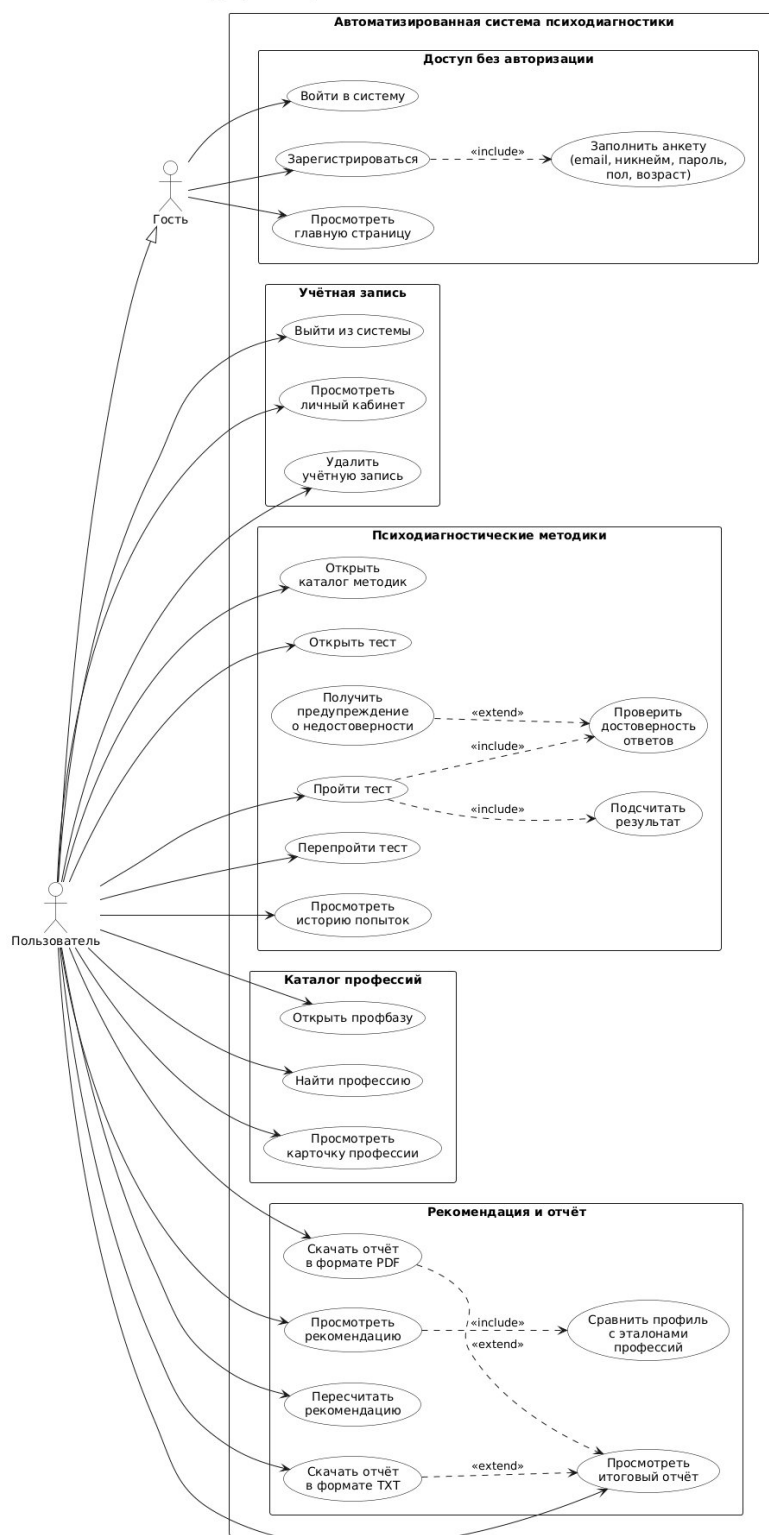


Рисунок 8 –Диаграмма вариантов использования пользователя

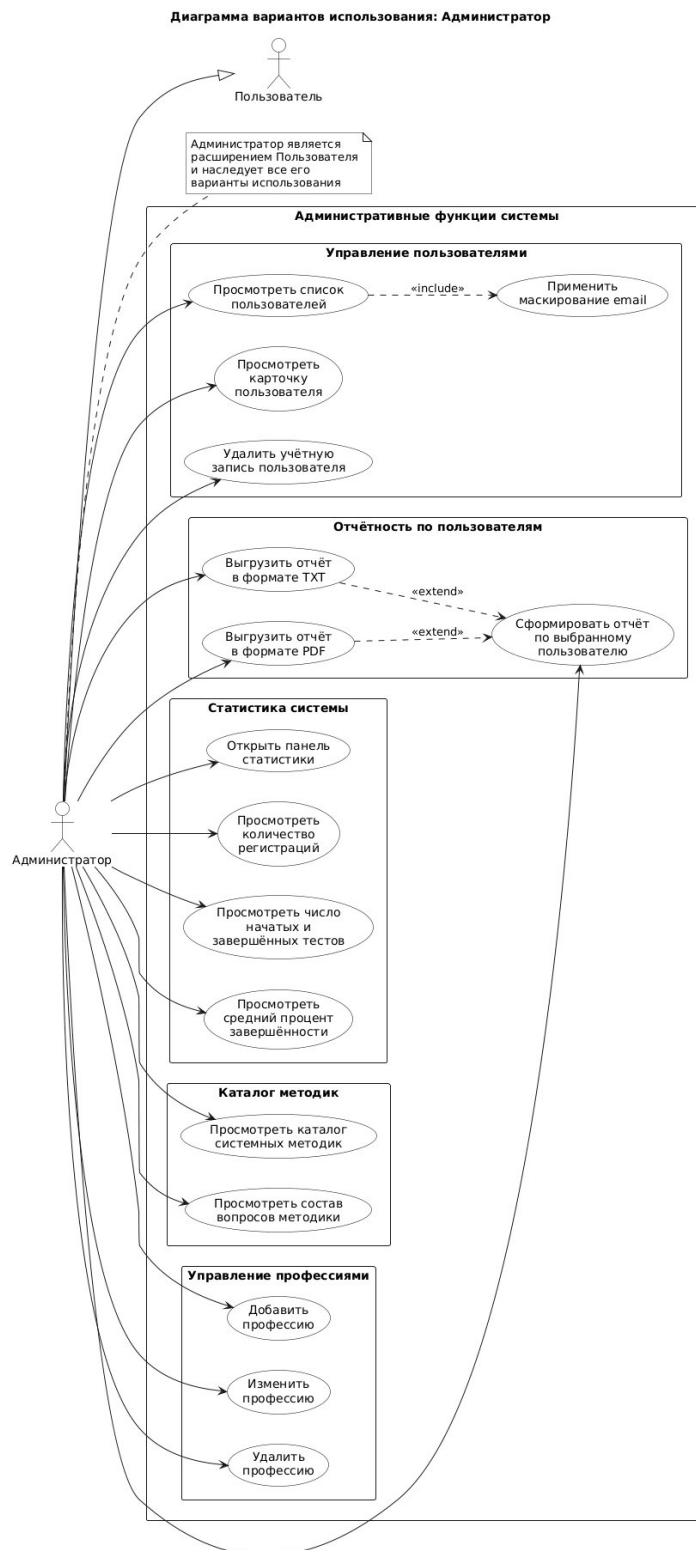


Рисунок 9 –Диаграмма вариантов использования администратора

1.2 Выбор технологий и средств разработки приложения

1.2.1 Выбор языка программирования для серверной части

Разработка автоматизированной системы психодиагностики для профотбора требует тщательного подбора инструментария, обеспечивающего не только реализацию заявленного функционала, но и высокую производительность, безопасность, удобство сопровождения и возможность дальнейшего расширения. В данном разделе обосновывается выбор языка программирования для серверной части, клиентской части, системы управления базами данных, фреймворков, среды разработки и способов взаимодействия между компонентами системы.

В результате анализа выбраны следующие технологии и средства разработки:

- язык программирования для серверной части – Python 3.11;
- язык программирования для клиентской части – TypeScript 5.7 в сочетании с библиотекой React 19 (компонентный интерфейс) и фреймворком Next.js 15 (серверный рендеринг, маршрутизация, оптимизация сборки);
- система управления базами данных (СУБД) – MySQL 8.x, выбранная за надёжность, поддержку транзакций ACID, нативный тип JSON, совместимость с библиотекой SQLAlchemy через драйвер PyMySQL, бесплатное распространение и повсеместную поддержку российских хостинг-провайдеров;
- фреймворки и библиотеки серверной части:
 - FastAPI 0.115 – асинхронный веб-фреймворк;
 - a) SQLAlchemy 2.0 – объектно-реляционный отображатель;
 - b) Alembic 1.14 – система версионирования и миграций схемы бд;
 - c) Pydantic 2.10 – библиотека для валидации JSON-структур;
 - d) PyJWT 2.10 – реализация JSON Web Token для аутентификации;
 - e) passlib + bcrypt – хеширование паролей пользователей;
 - f) PyMySQL 1.1 – драйвер подключения к MySQL;
- фреймворки и библиотеки клиентской части:
 - a) Next.js 15 и React 19 – основа клиентского приложения;

- b) TypeScript – статическая типизация исходного кода;
 - c) CSS(styles/globals.css) – оформление интерфейса
 - d) Vitest и React Testing Library –тестирование компонентов;
- среда разработки:
 - a) Visual Studio Code – основная среда написания кода;
 - b) MySQL Workbench 8.0 – проектирование и администрирование бд;
 - c) Git – система контроля версий;
 - d) PowerShell – оболочка командной строки;
- способы взаимодействия клиента и сервера – обмен данными по протоколу HTTP в формате JSON через REST API; аутентификация на основе JWT-токенов, передаваемых в заголовке Authorization каждого запроса.

Серверная часть автоматизированной системы отвечает за обработку запросов пользователей, выполнение бизнес-логики, взаимодействие с базой данных, реализацию алгоритма расчёта интегрального профиля и формирование персонализированных рекомендаций. От правильного выбора языка программирования зависит скорость разработки, производительность системы, её надёжность и возможность интеграции с необходимыми библиотеками.

При выборе языка для серверной части учитывались следующие критерии.

Производительность и масштабируемость – способность обрабатывать множество одновременных запросов, особенно в перспективе роста числа пользователей.

Зрелость веб-фреймворков и наличие готовых решений для REST API – поддержка построения HTTP-эндпоинтов, автоматической валидации входных данных, генерации документации API.

Качество средств работы с реляционной базой данных – наличие развитого ORM с поддержкой MySQL, системы миграций, удобной модели сессий.

Удобство реализации вычислительной логики – расчёт векторов черт, нормализация, агрегирование результатов нескольких методик, вычисление косинусной близости должны выражаться компактным и читаемым кодом.

Простота разработки и поддержки – возможность быстрого прототипирования, понятный синтаксис, обширное сообщество и документация.

Совместимость с выбранной СУБД и клиентской частью – лёгкость интеграции с MySQL и с REST–клиентом на Next.js.

Кроссплатформенность – возможность развёртывания на различных операционных системах (Linux, Windows, macOS).

Безопасность – наличие проверенных библиотек для хеширования паролей, работы с JWT–токенами и защиты от типовых уязвимостей.

На современном рынке языков программирования для серверной разработки наиболее популярны: Java, C#, Node.js (JavaScript), PHP, Go и Python. Рассмотрим их соответствие указанным критериям.

Java – мощный, строго типизированный язык с высокой производительностью и широким набором фреймворков (Spring, Spring Boot). Он отлично подходит для крупных корпоративных систем, однако характеризуется высокой сложностью разработки, необходимостью написания значительного объёма «инфраструктурного» кода (конфигурационных классов, аннотаций, описаний бинов) и менее гибкими возможностями для быстрого прототипирования. Для дипломного проекта со сжатыми сроками разработки этот фактор существенно замедляет работу.

C# – язык от Microsoft, тесно интегрированный с платформой .NET. Он обеспечивает высокую производительность, удобную среду разработки Visual Studio и хорошие возможности для создания веб–приложений (ASP.NET Core). Однако исторически он сильнее ориентирован на Windows–среду (хотя .NET Core кроссплатформенен), а его экосистема в части бесплатного веб–хостинга в России менее распространена, чем экосистема Python.

Node.js (JavaScript) – позволяет писать серверную часть на JavaScript, что обеспечивает единообразие языка с клиентской частью. Node.js асинхронен и хорошо подходит для приложений с большим количеством операций ввода–вывода.

Однако написание серверной логики без статической типизации усложняет сопровождение, а реализация вычислений (нормализация векторов, агрегация по нескольким методикам) на JavaScript выглядит менее компактно, чем на Python. Кроме того, использование одного и того же языка на сервере и клиенте в данном проекте не является обязательным, поскольку клиентская часть уже использует TypeScript, и преимущества «единого языка» нивелируются.

PHP – широко используется для веб–разработки, имеет множество фреймворков (Laravel, Symfony). Однако PHP исторически создавался для генерации HTML–страниц и менее удобен для построения REST API и реализации сложной бизнес–логики, требующей работы с математическими операциями над массивами. Современные тенденции показывают снижение популярности PHP для новых проектов в области аналитики и интегральной обработки данных.

Go – язык от Google, отличающийся высокой производительностью, простотой и отличной поддержкой конкурентности. Он хорошо подходит для создания высоконагруженных веб–сервисов. Однако его экосистема в части ORM (наиболее популярный GORM) уступает зрелостью SQLAlchemy для Python; кроме того, обязательная статическая типизация и явное управление ошибками увеличивают объём кода в задачах, где основной фокус – на бизнес–логике, а не на низкоуровневой оптимизации.

Python – интерпретируемый язык высокого уровня, занимающий одно из ведущих мест в современной веб–разработке. Его основные преимущества применительно к данному проекту это зрелая экосистема веб–фреймворков: FastAPI, Django, Flask. FastAPI, выбранный в проекте, обеспечивает автоматическую генерацию документации OpenAPI/Swagger, встроенную валидацию входных данных через Pydantic и асинхронную обработку запросов. Один из лучших ORM в индустрии – SQLAlchemy: полностью поддерживает MySQL через драйвер PyMySQL, имеет зрелую систему миграций Alembic, удобную модель сессий и широко применяется в коммерческих проектах.

Это критично для системы, в основе которой лежат сложные связанные сущности (пользователи, методики, попытки, профили, профессии). Лаконичный синтаксис для математических операций: расчёт вектора черт пользователя, нормализация, агрегирование результатов пяти методик с весовыми коэффициентами и вычисление косинусной близости выражаются на Python в виде компактных и читаемых функций, не требующих подключения тяжёлых внешних библиотек. Быстрая разработка: динамическая типизация (с возможностью использовать аннотации типов и Pydantic для контроля), огромное количество готовых модулей и краткий синтаксис позволяют создавать рабочие прототипы в кратчайшие сроки, что важно в условиях дипломного проектирования. Зрелые библиотеки информационной безопасности: passlib и bcrypt для хеширования паролей, PyJWT для работы с токенами сессий – все они проверены в коммерческих проектах. Кроссплатформенность и простота развёртывания: Python работает на всех популярных операционных системах, поддерживается практически всеми хостинг-провайдерами и легко контейнеризируется. Большое сообщество и обширная документация – при возникновении вопросов всегда можно найти готовые решения и примеры на русском и английском языках.

Недостатки Python – более низкая скорость выполнения по сравнению с компилируемыми языками (Java, Go) – в данном проекте не являются критичными, так как основная нагрузка связана с обработкой HTTP-запросов и работой с базой данных, а не с интенсивными вычислениями. К тому же асинхронные возможности FastAPI позволяют эффективно обрабатывать множество одновременных подключений.

Таким образом, учитывая необходимость зрелой работы с реляционной базой данных, скорость разработки, удобство выражения вычислительной логики и широкую распространённость языка, Python является наиболее обоснованным выбором для серверной части разрабатываемой автоматизированной системы. В качестве конкретного фреймворка использован FastAPI. [5]

1.2.2 Выбор языка программирования для клиентской части

Клиентская часть автоматизированной системы отвечает за пользовательский интерфейс, взаимодействие с пользователем и отправку запросов к серверной части. От правильного выбора технологий зависят удобство использования, скорость работы приложения, адаптивность под различные устройства и простота дальнейшей поддержки.

При выборе языка и фреймворка для клиентской части учитывались следующие критерии:

- производительность и отзывчивость интерфейса – приложение должно быстро реагировать на действия пользователя, плавно отображать страницы прохождения тестов и результаты.
- адаптивность – система должна корректно отображаться на компьютерах, планшетах и смартфонах.
- удобство разработки и поддержки – модульность, повторное использование компонентов, наличие готовых библиотек.
- современный стек и экосистема – популярность технологии, наличие сообщества, документации и инструментов разработки.
- совместимость с серверной частью – возможность удобного обмена данными в формате JSON через REST API.
- снижение количества ошибок при работе с данными – поскольку клиент обменивается со сервером сложными структурами (профили пользователя, рейтинги профессий, результаты тестов), желательно иметь средства предотвращения ошибок типов на этапе написания кода.

На сегодняшний день для разработки клиентских веб–приложений наиболее популярны три основных подхода.

Чистый JavaScript с HTML и CSS – не требует изучения дополнительных фреймворков, но приводит к сложности поддержки больших проектов, отсутствию структурированности и низкой скорости разработки. Многократное обращение к DOM–узлам и ручное обновление интерфейса при изменении данных делает код многословным и подверженным ошибкам.

jQuery – библиотека, упрощающая манипуляции с DOM и обработку событий, но не решающая проблему организации кода в сложных приложениях. На текущий момент относится к устаревающим подходам и не используется в новых проектах.

Современные фреймворки и библиотеки – Angular, Vue.js, React – предлагают компонентный подход, реактивное обновление интерфейса и широкие возможности. Рассмотрим их подробнее.

Angular – полноценный фреймворк от Google, использующий TypeScript «из коробки». Он предоставляет всё необходимое для создания крупных корпоративных приложений, но имеет крутой порог вхождения, многословный синтаксис, обязательную привязку к ряду собственных понятий (модули, сервисы, инжекторы) и избыточен для проекта среднего размера, каким является разрабатываемая система.

Vue.js – прогрессивный фреймворк, сочетающий простоту и гибкость. Он легче Angular и хорошо подходит для небольших и средних проектов. Однако его экосистема несколько меньше, чем у React, а распространённость на российском рынке труда ниже, что важно с точки зрения дальнейшей поддержки и развития проекта другими разработчиками.

React – библиотека для создания пользовательских интерфейсов от Meta. Она предлагает компонентный подход, виртуальный DOM для высокой производительности и крупнейшую среди современных решений экосистему. React проще в изучении, чем Angular, позволяет создавать как простые, так и сложные интерфейсы и является наиболее популярным выбором для современных веб-приложений. [6]

Для разрабатываемой системы выбран React, так как он обеспечивает:

- компонентную архитектуру, что упрощает разработку и поддержку кода (каждая часть интерфейса оформляется как отдельный переиспользуемый компонент);

- высокую производительность за счёт виртуального DOM и точечного обновления изменённых элементов;
- богатую экосистему готовых компонентов и библиотек;
- возможность реализации адаптивной вёрстки средствами CSS–модулей без подключения тяжёлых стилевых фреймворков.

Выбор языка: JavaScript или TypeScript

После выбора библиотеки React встаёт вопрос о языке исходного кода. React можно использовать как с JavaScript, так и с TypeScript – расширением JavaScript, добавляющим статическую типизацию. В разрабатываемой системе использован TypeScript по следующим причинам:

- раннее обнаружение ошибок – типы проверяются на этапе компиляции, а не во время работы пользователя. Для приложения, обменивающегося со сервером сложными структурами данных (профиль пользователя, рейтинг профессий, результаты тестов), это снижает количество ошибок, проявляющихся уже в браузере.
- подсказки в редакторе – Visual Studio Code на основе типов предлагает автодополнение полей и сигнатур функций, ускоряя разработку.
- документирование контрактов – описание типов ответов сервера в одном месте служит для разработчика своеобразной документацией API и упрощает работу с новыми эндпоинтами.
- полная совместимость с JavaScript – TypeScript не требует переписывания кода: любой JavaScript–код является корректным TypeScript–кодом, а проект может постепенно вводить типизацию. [7]

Выбор фреймворка поверх React: Next.js

Для полноценного веб–приложения поверх React удобно использовать фреймворк, который добавляет серверный рендеринг, маршрутизацию и оптимизацию сборки. Таким фреймворком является Next.js.

Next.js – это популярный React–фреймворк, который предоставляет:

- серверный рендеринг (SSR) и статическую генерацию (SSG) – улучшает индексацию поисковыми системами и сокращает время первого отображения, что важно для главной страницы с описанием системы;
- файловую маршрутизацию – создание страниц через структуру каталога `app/` упрощает навигацию и избавляет от необходимости описывать маршруты в отдельных конфигурационных файлах;
- автоматическую оптимизацию изображений и скриптов – повышает производительность загрузки;
- поддержку API–маршрутов – возможность создавать простые серверные функции прямо в Next.js (в проекте эта возможность не используется, поскольку вся серверная логика вынесена в отдельное приложение на Python);
- удобную разработку с горячей перезагрузкой – изменения в коде немедленно отражаются в браузере без перезапуска сервера разработки. [8]

Стилизация интерфейса

В качестве средства оформления интерфейса использован собственный набор CSS–модулей и единый файл глобальных стилей (`styles/globals.css`) без подключения внешних стилевых фреймворков (Tailwind, Bootstrap, Material UI). Такой подход выбран по двум причинам: полный контроль над оформлением без избыточных зависимостей, а также соответствие требованиям к адаптивной вёрстке, которое легко достигается стандартными CSS–инструментами (медиа–запросы, флексбокс–разметка, относительные единицы измерения). [9]

Тестирование клиентской части

Для проверки корректности работы компонентов используется библиотека Vitest в сочетании с React Testing Library – это современный стандарт автоматического тестирования React–приложений, обеспечивающий быстрый запуск тестов и проверку поведения компонентов с точки зрения пользователя.

Таким образом, использование TypeScript + React + Next.js позволяет создать современное, быстрое, адаптивное и устойчивое к ошибкам клиентское приложение, удобное как для разработчиков, так и для конечных пользователей.

1.2.3 Выбор СУБД

Подсистема хранения данных является основой автоматизированной системы психодиагностики. В ней хранятся учётные записи пользователей, каталоги методик и профессий, результаты прохождения тестов, рассчитанные профили пользователей, рейтинги рекомендаций и сформированные отчёты. От выбора системы управления базой данных (СУБД) зависят надёжность хранения, скорость работы приложения, возможность одновременного доступа большого числа пользователей и удобство сопровождения системы.

При выборе СУБД учитывались следующие критерии:

- поддержка реляционной модели и ссылочной целостности – данные системы описываются жёстко связанными сущностями (пользователь – попытка – методика – вопрос – ответ – профессия – рекомендация), что естественным образом ложится на реляционную модель;
- поддержка транзакций ACID – сохранение результатов теста и формирование рекомендации представляют собой логически неделимые операции, при сбое которых промежуточные данные не должны попадать в выборку;
- поддержка стандартного языка запросов SQL – для построения аналитических выборок в административной панели (количество регистраций, число попыток, средняя завершённость) необходим декларативный язык с агрегирующими функциями и соединениями;
- совместимость с используемым ORM (SQLAlchemy) – наличие официально поддерживаемого драйвера для Python;
- поддержка типа JSON – в моделях профессий используются неструктурированные поля (преимущества, недостатки, направления подготовки, типичные затруднения), которые удобно хранить как массивы JSON;
- многопользовательский режим работы – система рассчитана на одновременное прохождение тестов несколькими пользователями;
- бесплатность и открытость – учебный проект с возможностью свободного распространения исключает использование коммерческих СУБД;

- распространённость на хостинг–провайдерах РФ – упрощает дальнейшее развёртывание системы в эксплуатации;
- наличие зрелых инструментов администрирования – графический клиент с возможностью построения ER–диаграмм, выполнения SQL–запросов, резервного копирования и редактирования данных.

В качестве системы управления базой данных выбрана MySQL 8.x – одна из самых популярных открытых реляционных СУБД, широко используемая в веб–приложениях и корпоративных системах.

MySQL полностью соответствует всем перечисленным критериям. Используемый по умолчанию движок хранения InnoDB обеспечивает поддержку транзакций ACID, внешних ключей и построчных блокировок, что позволяет нескольким пользователям одновременно проходить тестирование без блокировки базы целиком. Нативный тип JSON даёт возможность хранить неструктурированные данные профессий в одном столбце без создания дополнительных таблиц. Драйвер PyMySQL официально поддерживается библиотекой SQLAlchemy 2.0 и используется в проекте через строку подключения вида `mysql+pymysql://`. Распространение по открытой лицензии GPL не накладывает финансовых ограничений на разработку и распространение учебного проекта. MySQL поддерживается всеми основными операционными системами (Windows, Linux, macOS) и включается в линейки услуг большинства хостинг–провайдеров Российской Федерации, что упрощает дальнейшее развёртывание системы. [10]

Отдельным аргументом в пользу выбора MySQL является развитая экосистема визуальных инструментов администрирования. Помимо командной строки (`mysql`), доступны графические клиенты, упрощающие повседневную работу с базой: phpMyAdmin (веб–интерфейс, часто включаемый в комплект хостинга), HeidiSQL, DBeaver, а также официальный клиент компании Oracle – MySQL Workbench.

MySQL Workbench выбран в качестве основного средства администрирования базы данных разрабатываемой системы. Этот клиент предоставляет всё необходимое для проектирования, разработки и сопровождения базы:

- просмотр структуры таблиц, столбцов, ключей и индексов в наглядном виде;
- визуальный редактор для добавления, изменения и удаления записей;
- редактор SQL-запросов с подсветкой синтаксиса и автодополнением;
- автоматическое построение ER-диаграмм по существующей базе (функция Reverse Engineer), что особенно полезно для оформления раздела «Проектирование базы данных» в дипломной работе;
- экспорт схемы и данных в виде SQL-скрипта (Data Export);
- резервное копирование и восстановление базы.

Таким образом, выбор MySQL в сочетании с MySQL Workbench обеспечивает надёжное хранение данных, простое сопровождение системы и удобную работу с базой как на этапе разработки, так и при последующей эксплуатации.

1.2.4 Выбор фреймворков и библиотек

Необходимо определить набор фреймворков и библиотек, которые обеспечат эффективную реализацию функциональных требований системы.

Выбор обусловлен необходимостью ускорить разработку, повысить безопасность хранения данных, обеспечить удобство сопровождения и согласованную интеграцию между серверной и клиентской частями.

Фреймворки и библиотеки предназначены для оптимизации процесса разработки программного обеспечения за счёт повторного использования готовых, отлаженных компонентов и типовых архитектурных решений. Их применение позволяет сократить временные и трудовые затраты на написание кода с нуля, минимизировать вероятность возникновения ошибок, а также обеспечить стандартизацию и поддерживаемость проектов.

Основным фреймворком для создания веб–приложения выбран FastAPI. Его преимущества:

- высокая производительность – FastAPI входит в число самых быстрых Python–фреймворков благодаря асинхронной архитектуре (asyncio) и использованию Starlette в качестве низкоуровневой основы;
- автоматическая документация – генерация интерактивной документации OpenAPI (Swagger UI) и ReDoc, что упрощает тестирование эндпоинтов и согласованную работу с клиентской частью;
- валидация данных – встроенная поддержка библиотеки Pydantic для проверки типов и структуры входящих и исходящих JSON–структур, что критически важно для корректной обработки результатов тестирования и форм регистрации;
- простота и интуитивность – лаконичный синтаксис, основанный на аннотациях типов Python, и качественная документация позволяют быстро создавать эндпоинты и бизнес–логику. [11]

Для работы с базой данных используется SQLAlchemy 2.0 – одна из наиболее зрелых ORM (Object–Relational Mapping) в индустрии. SQLAlchemy позволяет описывать таблицы базы данных в виде классов Python, автоматически генерировать SQL–запросы, прозрачно управлять сессиями и транзакциями, а также легко переключаться между разными СУБД (в нашем случае MySQL через драйвер PyMySQL) без изменения бизнес–логики. Все 12 таблиц разрабатываемой системы описаны как классы SQLAlchemy в каталоге backend/app/models. [12]

Для версионирования структуры базы данных применяется Alembic – система миграций, совместимая с SQLAlchemy. Каждое изменение схемы (добавление таблицы, столбца, индекса) оформляется как отдельный сценарий миграции, что обеспечивает воспроизводимое развёртывание системы на любой среде и возможность отката при необходимости.

Для обеспечения безопасности используются:

- passlib в связке с bcrypt – для хеширования паролей пользователей перед сохранением в базу данных; [13]
- PyJWT – для выпуска и проверки JSON Web Token, на которых построена система аутентификации; [14]
- email-validator – для валидации адресов электронной почты при регистрации.

Для драйверного уровня работы с базой данных используется PyMySQL – официально поддерживаемый библиотекой SQLAlchemy драйвер подключения к MySQL.

Для автоматического тестирования серверной части применяется pytest – фактический стандарт в Python-разработке, в сочетании с библиотекой httpx, позволяющей программно отправлять запросы к FastAPI-приложению в рамках тестов. [15]

Для логирования и обработки ошибок используется стандартная библиотека logging в сочетании с механизмом middleware FastAPI, что обеспечивает централизованный сбор журналов работы серверной части.

Клиентская часть строится на фреймворке Next.js 15, который поверх библиотеки React 19 добавляет:

- маршрутизацию на основе файловой системы – каждая страница приложения соответствует отдельной папке в каталоге app/, что упрощает навигацию и избавляет от необходимости описывать маршруты в отдельных конфигурационных файлах;
- серверный рендеринг (SSR) и статическую генерацию – улучшают время загрузки и индексацию поисковыми системами, особенно для главной страницы с описанием системы;
- оптимизацию изображений и скриптов – повышает производительность загрузки;
- горячую перезагрузку – изменения в коде немедленно отражаются в браузере без перезапуска сервера разработки.

Языком разработки клиентской части является TypeScript 5.7 – статически типизированное расширение JavaScript, обеспечивающее раннее обнаружение ошибок и удобную работу с типами данных, передаваемыми между клиентом и сервером.

Для оформления интерфейса применяются CSS–модули и единый файл глобальных стилей (styles/globals.css) без подключения внешних стилевых фреймворков. Такой подход обеспечивает полный контроль над визуальным оформлением, отсутствие избыточных зависимостей и реализацию адаптивной вёрстки средствами стандартного CSS (медиа–запросы, флексбокс–разметка, относительные единицы измерения).

1.2.5 Выбор среды разработки

Среда разработки (IDE) играет ключевую роль в процессе создания программного продукта, поскольку от её функциональности, удобства и производительности зависит скорость написания кода, его отладки и тестирования. При выборе инструментов для реализации автоматизированной системы психодиагностики учитывались требования к поддержке используемых технологий (Python, TypeScript, Next.js, MySQL), возможность интеграции с системами контроля версий, наличие средств автодополнения, рефакторинга и отладки, а также личный опыт разработчика.

В качестве основной интегрированной среды разработки выбрана Visual Studio Code (VS Code) – бесплатный кроссплатформенный редактор кода от компании Microsoft. VS Code зарекомендовал себя как один из наиболее популярных инструментов среди веб–разработчиков благодаря следующим характеристикам:

– лёгкость и быстродействие – редактор не требователен к ресурсам, быстро запускается и работает даже на компьютерах со средними характеристиками;

– расширяемость – обширный каталог расширений позволяет адаптировать среду под конкретные задачи. Для данного проекта установлены расширения для поддержки Python (Python, Pylance), TypeScript и React, а также для работы с Git;

– встроенный терминал – позволяет выполнять команды операционной системы, запускать серверы, управлять зависимостями через пакетные менеджеры (pip, npm) без переключения между окнами;

– интеграция с Git – встроенные средства контроля версий упрощают фиксацию изменений, работу с ветками и удалёнными репозиториями;

– отладка – графический отладчик для Python и TypeScript, позволяющий устанавливать точки останова, просматривать значения переменных и стек вызовов;

– поддержка удалённой разработки – возможность подключаться к удалённым серверам или контейнерам, что полезно при последующем развёртывании системы.

Для проектирования и администрирования базы данных MySQL используется MySQL Workbench 8.0. Данный инструмент предоставляет:

– визуальное проектирование баз данных – создание ER–диаграмм, наглядное представление структуры таблиц и связей между ними, что облегчает моделирование данных и документирование;

– построение ER–диаграммы по существующей базе (Reverse Engineer) автоматическое формирование схемы из уже созданных таблиц, что использовано при подготовке иллюстраций для подраздела о проектировании базы данных;

– выполнение SQL–запросов – встроенный редактор с подсветкой синтаксиса, автодополнением и возможностью сохранять часто используемые скрипты;

– администрирование сервера – управление пользователями, настройка параметров, мониторинг подключений и производительности;

- прямой доступ к данным – просмотр и редактирование записей в таблицах, что удобно при отладке и наполнении базы тестовыми данными;
- экспорт схемы и данных в виде SQL-скрипта (Data Export) для приложения к работе.

В процессе разработки также активно используются встроенные средства браузера, в первую очередь Chrome DevTools, для отладки клиентской части – анализ сетевых запросов, проверка адаптивности и профилирование производительности. Для тестирования API применяется встроенная документация Swagger UI, автоматически генерируемая FastAPI по адресу /docs, а также утилита Postman для ручной проверки отдельных эндпоинтов.

Для системы контроля версий используется Git – фактический стандарт в современной разработке программного обеспечения. Все изменения исходного кода фиксируются в виде последовательных коммитов, что обеспечивает прозрачную историю разработки и возможность возврата к любой предыдущей версии проекта.

Для работы с терминалом и запуска команд проекта (установка зависимостей, запуск серверной и клиентской частей, выполнение тестов, миграции базы данных) используется PowerShell – встроенная оболочка командной строки операционной системы Windows.

Таким образом, в качестве основных инструментов разработки выбраны Visual Studio Code – мощная и гибкая среда для написания кода, MySQL Workbench – удобное средство для проектирования и администрирования базы данных, Chrome DevTools и Swagger UI – для отладки клиентской и серверной частей соответственно, Git – для контроля версий, PowerShell – для работы с командной строкой. Данный набор полностью обеспечивает реализацию всех этапов создания автоматизированной системы психодиагностики: от проектирования до тестирования и развёртывания.

1.3 Концепция системы и содержательная основа автоматизации

1.3.1 Концепция системы и состав психодиагностических методик

Идея разрабатываемой системы строится на простом принципе: чем больше независимых психодиагностических методик пройдёт пользователь, тем устойчивее будет полученная картина его профессиональных склонностей. Одиночный тест легко исказить, подгоняя ответы под желаемый результат, тогда как согласованные данные нескольких разных методик дают значительно более надёжный итог. На этом принципе и построена концепция автоматизации: один пользователь последовательно проходит пять проверенных тестов, система самостоятельно сводит их результаты в единый профиль и сопоставляет его с заранее подготовленной базой профессий.

Для пользователя процесс выглядит как «один сценарий – много данных»: достаточно зарегистрироваться, по очереди пройти пять тестов в личном кабинете и получить итоговую рекомендацию. Всё остальное – нормализацию, агрегирование, контроль достоверности, сравнение с эталонными профилями и формирование текстового пояснения – система выполняет автоматически, без участия специалиста–психолога.

В состав системы включены пять методик, отобранных по принципу взаимодополнения: каждая измеряет свой срез личности и интересов, и вместе они покрывают и интересы, и темперамент, и ценностную сферу.

Тест 1 – «Школьные ориентиры» (стартовая методика, авторская):

Стартовый тест на основе 18 школьных предметов, по каждому из которых пользователь указывает три значения: текущую оценку, самооценку способностей и уровень мотивации. Эта методика выполняет двойную функцию: формирует первичный набросок интересов на знакомом школьном материале и одновременно служит источником данных для контроля достоверности ответов. Школьный материал выбран сознательно – он понятен большинству пользователей и не требует психологической подготовки.

Тест 2 – ДДО Е. А. Климова:

Классический дифференциально–диагностический опросник, относящий пользователя к одному из пяти типов профессий («человек – человек», «человек – техника», «человек – знаковая система», «человек – художественный образ», «человек – природа»). Реализован в виде 20 пар утверждений, из которых пользователь выбирает наиболее близкое. Методика стабильно работает в задачах профориентации, лежит в основе многих государственных и образовательных систем и хорошо узнаваема. [16]

Тест 3 – Holland RIASEC:

Тест профессиональных интересов Дж. Холланда, оценивающий пользователя по шести типам: реалистический (R), исследовательский (I), артистический (A), социальный (S), предпринимательский (E), конвенциональный (C). Реализован как 30 утверждений с пятибалльной шкалой согласия. В отличие от ДДО, который опирается на бинарные предпочтения, RIASEC даёт количественную оценку каждого из шести типов и позволяет учитывать смешанные профили. [17]

Тест 4 – Big Five (Пятифакторная модель личности):

Современный стандарт оценки личностных черт, измеряющий пять факторов: экстраверсию (E), доброжелательность (A), добросовестность (C), эмоциональную устойчивость (N), открытость опыту (O). Реализован как 25 утверждений с пятибалльной шкалой согласия. Big Five добавляет в общую картину то, чего нет у методик чисто профессиональной направленности: оценку темперамента и поведения в коллективе. [18]

Тест 5 – Ценностные ориентации М. Рокича:

Методика ранжирования 36 ценностей (терминальных – «цели жизни» – и инструментальных – «способы их достижения»). Пользователь упорядочивает ценности по приоритетности от 1 до 18 в каждой группе. Эта методика дополняет картину мотивационным контекстом: две одинаковые по способностям личности могут стремиться к разным сферам в зависимости от того, что они считают для себя по–настоящему важным. [19]

Каждой методике в системе присвоен весовой коэффициент, отражающий её диагностическую значимость для задачи профотбора. Коэффициенты подобраны так, чтобы основной вес ложился на наиболее устоявшиеся в отечественной и зарубежной практике профессионально–ориентированные методики – ДДО Климова и RIASEC Холланда, – а вспомогательные методики (стартовый тест, Big Five, Рокич) формировали уточняющий контекст. Итоговый расчет представлен в таблице 1.

Таблица 1 – Веса психодиагностических методик в итоговом расчёте

Методика	Тип данных	Весовой коэффициент
Тест 1. Школьные ориентиры	Три шкалы (оценка, самооценка, мотивация) по 18 предметам	0,15
Тест 2. ДДО Климова	Бинарный выбор из пары утверждений (20 пар)	0,25
Тест 3. Holland RIASEC	Согласие по шкале 0–4 (30 утверждений)	0,25
Тест 4. Big Five	Согласие по шкале 1–5 (25 утверждений)	0,20
Тест 5. Ценностные ориентации Рокича	Ранжирование 36 ценностей по приоритетности	0,15
Итого	Итоговое значение	1,00

В итоге система получает пять независимых «взглядов» на одного и того же пользователя и сводит их в один согласованный профиль. Именно сочетание нескольких методик и их автоматическая агрегация по фиксированным весам позволяют говорить о системе как об автоматизированной: данные пяти проверенных тестов уже сами по себе дают достаточно надёжный материал для рекомендации, без необходимости очной интерпретации психологом.

1.3.2 Структура шкал оценивания и обоснование ключевых показателей

Чтобы результаты пяти разных методик можно было сравнить и агрегировать, в системе введён единый набор шкал черт личности, общий для всех тестов и для каталога профессий. Каждая методика на выходе формирует свой вектор по этим шкалам, после чего векторы можно складывать с весами и сравнивать с эталонными профилями профессий.

Всего в системе используется девять универсальных шкал, охватывающих основные срезы профессиональной направленности, они указаны в таблице 2.

Таблица 2 – Универсальные шкалы черт личности

№	Шкала	Что отражает
1	Люди	Склонность к работе с людьми, общению, обучению
2	Техника	Интерес к технике, машинам, механизмам, задачам
3	Знаки	Работа с символами, текстами, кодом, формулами
4	Творчество	Художественное самовыражение, создание новых образов
5	Природа	Интерес к живым организмам, экологии
6	Аналитика	Склонность к исследованию, анализу данных
7	Практика	Предпочтение физической, осязаемой деятельности
8	Структура	Потребность в порядке, правилах, чётких процедурах и документации
9	Управление	Склонность к организации, координации, принятию решений и руководству

Этот набор намеренно сделан широким и универсальным: шкалы 1–5 близки к классическим типам Климова и Холланда (человек, техника, знаки, художественный образ, природа), а шкалы 6–9 описывают рабочий стиль и темперамент, что важно для разделения близких по интересам профессий (например, отделить аналитика от практика в технической сфере).

Хотя итоговый профиль приводится к единому виду, внутри каждой методики используются разные шкалы оценивания, исторически закреплённые в первоисточниках. Это позволяет сохранить психометрическую корректность каждой методики. Шкалы оценивания представлены в таблице 3.

Таблица 3 – Шкалы оценивания внутри методик

Методика	Формат ответа	Преобразование в универсальный вектор
Школьные ориентиры	Три числа по каждому предмету (оценка, самооценка, мотивация).	Мотивация нормируется и взвешивается коэффициентами «предмет – шкала».
ДДО Климова	Бинарный выбор «А» или «В» из пары утверждений.	Подсчёт частот по 5 типам Климова, далее пересчёт в 9 универсальных шкал.
Holland RIASEC	Шкала Лайкерта 0–4 («0 – совсем не про меня» ... «4 – полностью про меня»).	Усреднение по 6 типам RIASEC, далее пересчёт в 9 универсальных шкал.
Big Five	Шкала Лайкерта 1–5 («полностью не согласен» ... «полностью согласен»).	Усреднение по 5 факторам, далее пересчёт в 9 универсальных шкал.
Ценностные ориентации Рокича	Ранжирование от 1 (самое важное) до 18 (наименее важное).	Пересчёт ранга в приоритет

После того как каждая методика дала свой вектор по 9 шкалам, все векторы агрегируются с весами из и итоговый вектор пользователя нормируется в диапазоне от 0 до 1 (значение 1 соответствует наиболее ярко выраженной шкале у данного пользователя).

Особо стоит остановиться на трёх шкалах стартового теста «Школьные ориентиры», поскольку именно в них заложена одна из ключевых идей всей системы. По каждому из 18 школьных предметов пользователь указывает три разные величины:

- оценка – текущая школьная оценка по предмету (объективный внешний показатель);
- самооценка – собственная оценка способностей по этому предмету (внутренний субъективный показатель);
- мотивация – насколько пользователю интересно и хочется заниматься этим предметом.

При итоговом расчёте профиля в качестве основного сигнала используется именно мотивация, а не оценка. Это сознательное решение, продиктованное самой задачей профотбора: высокая оценка по предмету говорит лишь о том, что человек справляется с программой, но не о том, что эта деятельность подходит ему как будущая профессия. Множество людей хорошо учились по математике в школе, но не стали ни математиками, ни программистами – потому что предмет им давался, но не нравился. Мотивация же напрямую отражает интерес, а интерес главный долгосрочный двигатель профессионального развития.

Шкалы «оценка» и «самооценка» не отбрасываются – они используются для контроля достоверности ответов. Если у пользователя по большинству предметов наблюдается значительное расхождение между объективной оценкой и самооценкой (например, тройка по предмету при максимальной самооценке 10), или если средняя мотивация по всем предметам аномально низкая, система формирует предупреждение о возможной недостоверности результата и предлагает пройти тестирование повторно.

Это та самая встроенная проверка, отсутствие которой является одним из главных недостатков большинства бесплатных аналогов.

Таким образом, единый набор из девяти шкал на выходе и продуманные шкалы оценивания внутри каждой методики обеспечивают одновременно две задачи: сравнимость результатов разных тестов (агрегация в один вектор) и сохранение исходной диагностической ценности каждой методики (использование родного для неё формата ответа).

1.3.3 Каталог профессий и эталонные профили требуемых черт

Чтобы результат тестирования можно было преобразовать в конкретную рекомендацию, в системе сформирован каталог профессий, где для каждой позиции определён эталонный профиль требуемых черт, – то есть формализованный набор психологических и профессионально значимых характеристик, который служит образцом для сравнения с индивидуальными показателями пользователя.

Каталог содержит 37 базовых профессий, охватывающих основные сферы российского рынка труда: социальную, такие как психолог, учитель, врач, социальный работник, управленческую, такие как менеджер, юрист, HR, техническую и инженерную, такие как программист, инженер, системный аналитик, творческую, такие как дизайнер, художник, актёр, музыкант, естественнонаучную, такие как биолог, эколог, ветеринар, финансово-экономическую, такие как экономист, бухгалтер, финансист, производственную и силовую, такие как строитель, электрик, военнослужащий, спасатель, сервисную, такие как повар, водитель, логист.

Каждая профессия в системе описывается одновременно двумя способами – содержательно (для пользователя) и количественно (для алгоритма расчёта).

Содержательная часть (отображается пользователю в профбазе и в карточке профессии):

- название и краткое описание;
- список преимуществ работы по этой профессии;
- список недостатков и типичных трудностей;
- возможные направления обучения и подготовки;
- типичные затруднения, с которыми сталкиваются специалисты;
- оценка популярности профессии на российском рынке труда (высокая, средняя, низкая).

Количественная часть (используется для расчёта рекомендации):

- эталонный профиль требуемых черт – набор значений от 0 до 1 по каждой из 9 универсальных шкал, описывающий, насколько ярко должна быть выражена та или иная черта у успешного представителя этой профессии.

Именно количественная часть и обеспечивает работу алгоритма подбора: профиль пользователя сравнивается с эталонными профилями всех профессий в каталоге, и список выстраивается по убыванию степени совпадения.

Чем ближе значение к 1, тем сильнее требуется выраженность данной черты для успешной работы по профессии; значение 0 означает, что черта для этой профессии не существенна.

В качестве эталонной базы выступила выборка из 37 профессий, для каждой из которых в Excel–таблице задан профиль требуемых профессионально важных качеств (9 шкал). Фрагмент указанной таблицы приведён на рисунке 10.

Эталонные профили требуемых черт (фрагмент каталога)										
	A	B	C	D	E	F	G	H	I	J
1	Профессия	Люди	Техника	Знаки	Творчество	Природа	Аналитика	Практика	Структура	Управление
2	Психолог	1,0	0,0	0,6	0,4	0,0	0,7	0,1	0,5	0,2
3	Программист	0,2	0,8	1,0	0,3	0,0	1,0	0,1	0,8	0,2
4	Менеджер	0,9	0,3	0,6	0,4	0,0	0,7	0,3	0,8	1,0
5	Биолог	0,2	0,2	0,6	0,3	1,0	0,8	0,4	0,6	0,1
6	Дизайнер	0,5	0,3	0,4	1,0	0,0	0,5	0,2	0,4	0,2
7	Электрик	0,2	1,0	0,4	0,1	0,0	0,7	0,9	0,8	0,2

Рисунок 10 – Эталонные профили требуемых черт (фрагмент каталога)

Видно, как профили значительно различаются между собой и при этом остаются сопоставимыми за счёт единого набора шкал. У психолога максимум по шкале «Люди» при практически нулевой «Технике»; у программиста, наоборот, максимум по «Знакам» и «Аналитике» при минимальной выраженности «Людей»; менеджер сочетает высокие значения «Людей» и «Управления»; биолог – единственный в этой выборке с высоким значением шкалы «Природа»; дизайнер выделяется максимальной «Творчеством»; электрик – высокими «Техникой» и «Практикой».

После того как пользователь прошёл методики и система рассчитала его собственный вектор по тем же 9 шкалам, алгоритм поочередно сравнивает этот вектор с эталонными профилями всех 37 профессий из каталога. Для сравнения используется косинусная близость векторов – мера, показывающая, насколько похожи два набора значений по своему общему «направлению» вне зависимости от их абсолютной величины. Чем ближе значение косинусной близости к 1, тем выше степень совпадения профиля пользователя с эталонным профилем профессии. Полученные значения сортируются по убыванию, и пользователь получает упорядоченный рейтинг подходящих профессий.

1.4 Проектирование информационной системы

1.4.1 Архитектура приложения

В разработке программного обеспечения применяются различные архитектурные шаблоны: MVC, MVVM, микросервисная архитектура, многоуровневая (слоистая) архитектура и другие. В разрабатываемой системе использована трёхзвенная многоуровневая архитектура в сочетании с компонентным подходом на клиенте – наиболее естественное решение для современных одностраничных веб–приложений на стеке Next.js + REST API + СУБД. [20]

Для полного описания архитектуры применён язык UML (Unified Modeling Language), позволяющий представить систему с разных точек зрения. По принятой в UML классификации диаграммы делятся на две большие группы: статические (описывают структуру системы – какие компоненты в неё входят и как они связаны) и поведенческие (описывают функционирование системы – как компоненты взаимодействуют во времени). Ниже последовательно приводятся обе группы диаграмм с краткими пояснениями. Детальная реализация каждого элемента раскрывается в последующих подразделах.

Диаграмма классов показывает состав классов серверной части системы и связи между ними. Каждый класс соответствует одной сущности предметной области и одновременно одной таблице базы данных, так как в проекте использован объектно–реляционный отображатель SQLAlchemy.

Данный инструмент моделирования является полезным и информативным, поскольку он позволяет разработчикам, аналитикам и техническим специалистам в наглядной форме воспринять организацию серверной логики без погружения в исходный код. Диаграмма классов представлена на рисунке 11.

Диаграмма классов серверной части

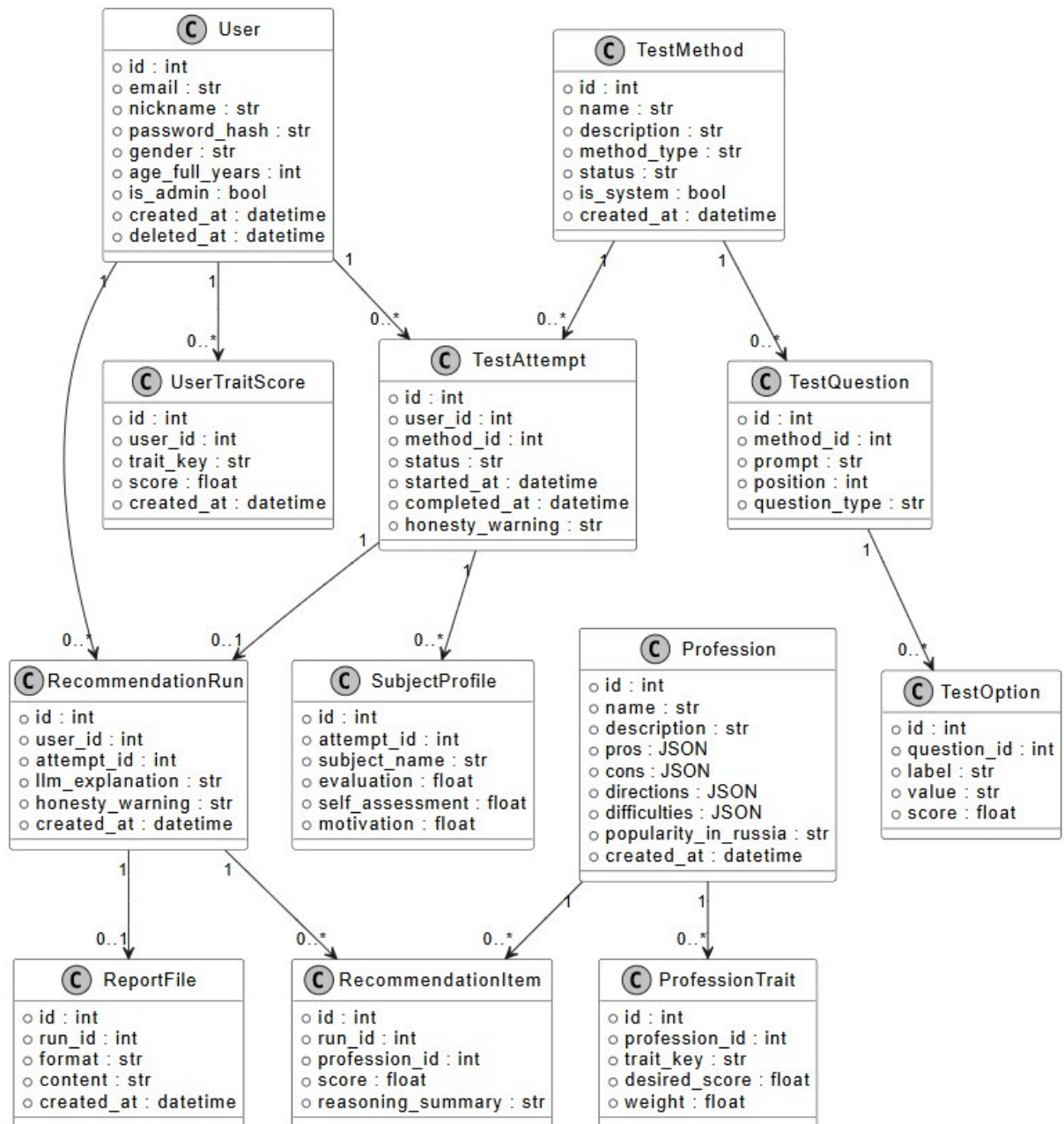


Рисунок 11 – Диаграмма классов серверной части

Из диаграммы видно, что центральным классом является User, от которого отходят связи к попыткам прохождения тестов, баллам по чертам и расчётам рекомендации. Каталоги методик и профессий образуют отдельные ветви, объединяющиеся через сущность RecommendationItem.

ER–диаграмма базы данных описывает те же сущности с точки зрения реляционной модели – в виде таблиц с первичными и внешними ключами. Диаграмма строится автоматически средствами MySQL Workbench (функция Reverse Engineer) по уже существующей базе.

ER–диаграмма базы данных представлена на рисунке 12 и 13.

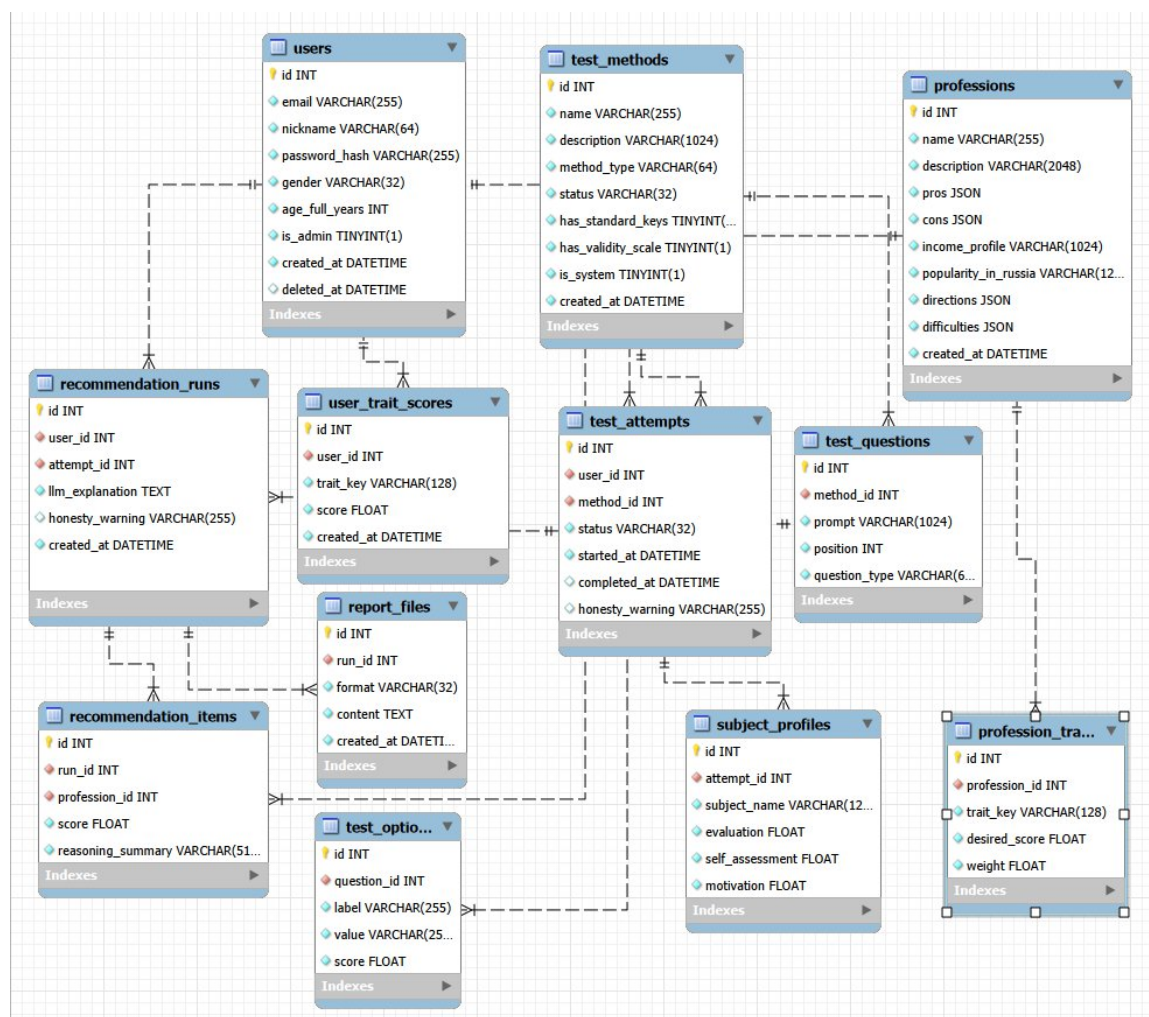


Рисунок 12 – ER–диаграмма базы данных (MySQL Workbench)

ER-диаграмма базы данных psychodiagnostics

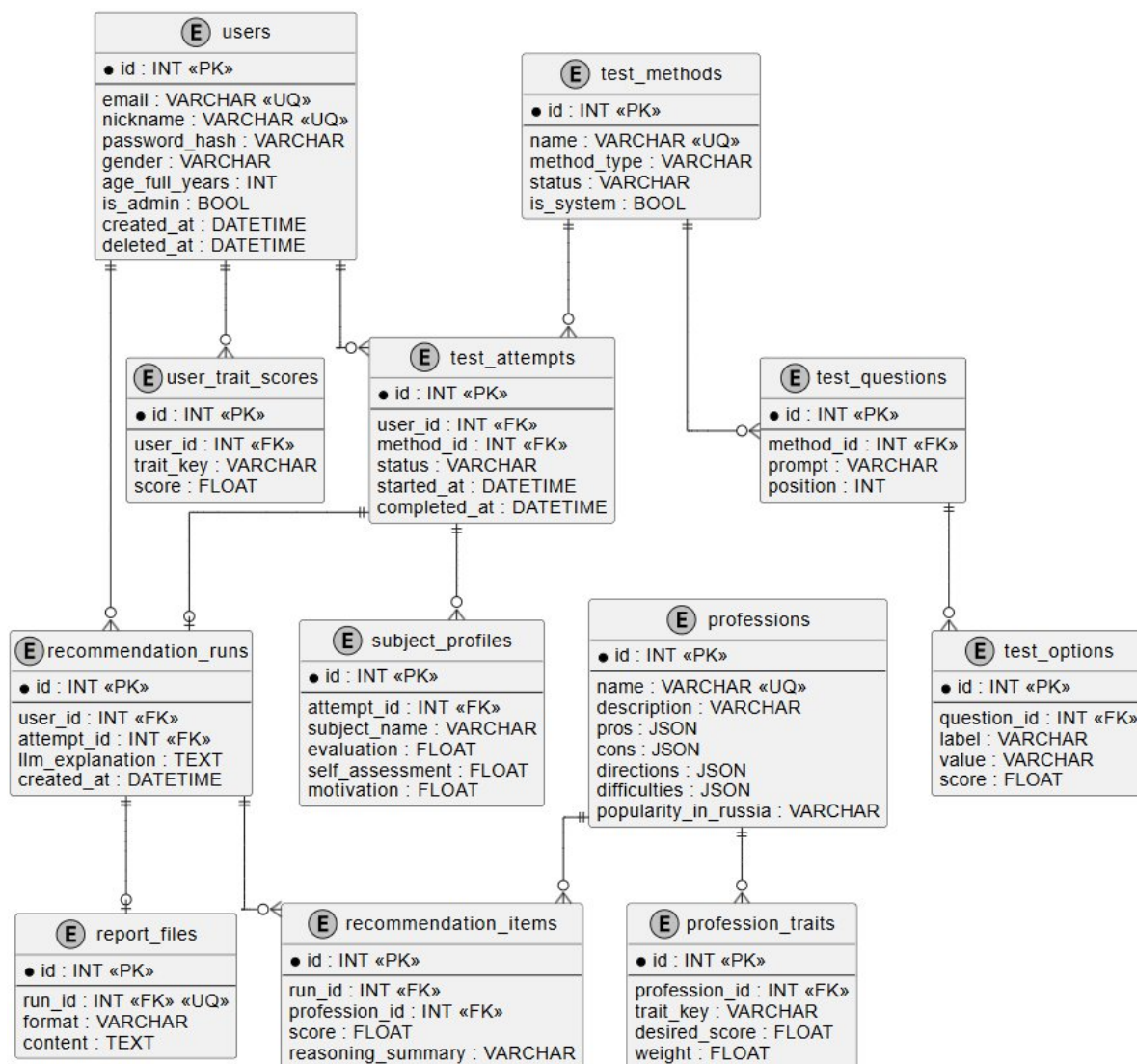


Рисунок 13 – ER–диаграмма базы данных (PlantUML)

База данных содержит 12 таблиц, объединённых в четыре логических блока: учётные данные пользователей, методики и попытки прохождения, профессии и их требования, рекомендации и отчёты.

Компонентная диаграмма отражает разделение системы на три уровня (представление, бизнес–логика, данные) и внутренние модули каждого уровня. Представлена на рисунке 14.

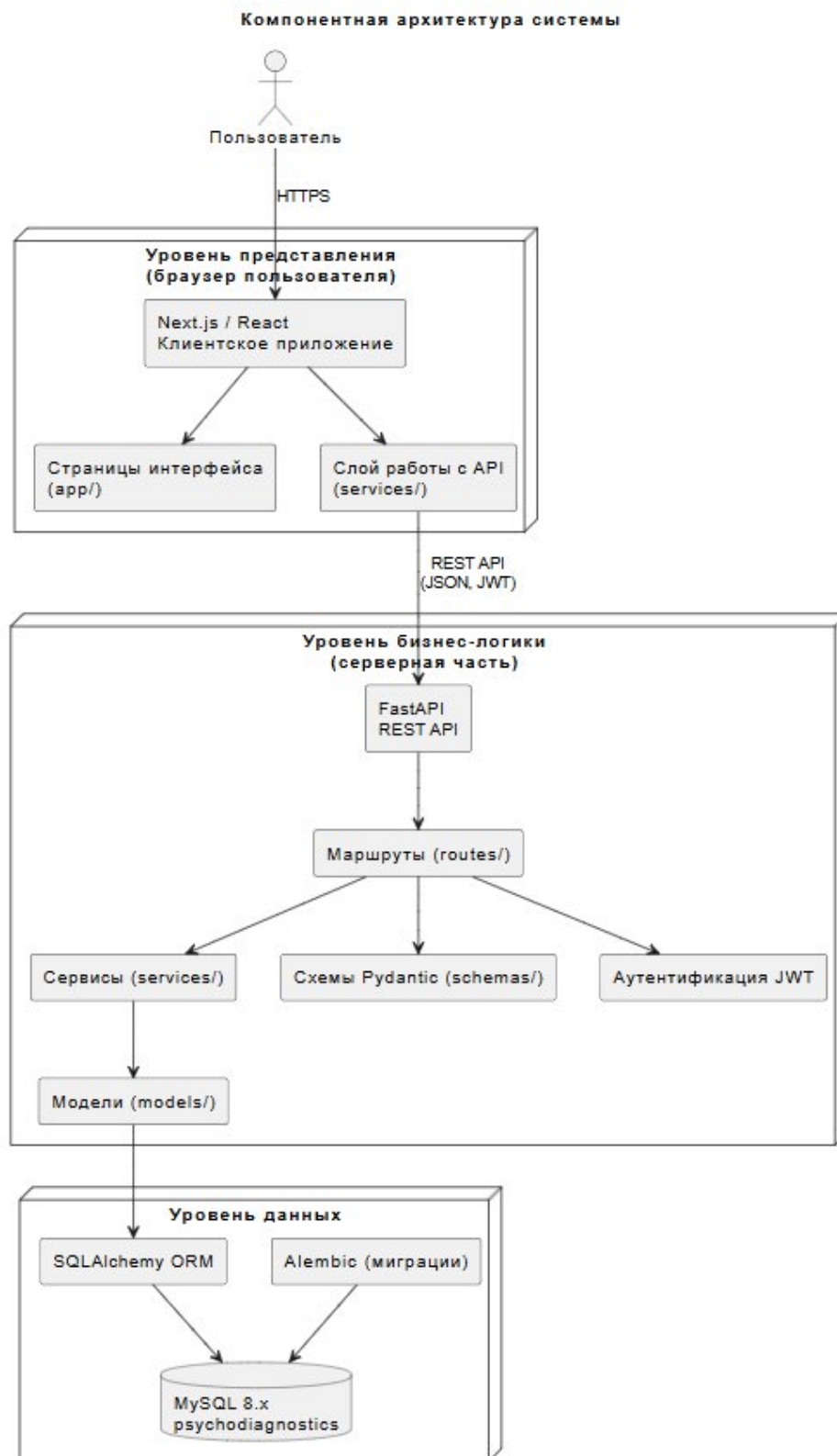


Рисунок 14 – Компонентная архитектура системы

Диаграмма показывает, что уровни взаимодействуют через чётко определённые интерфейсы (REST API между клиентом и сервером, ORM между сервером и базой), что позволяет заменять реализацию любого уровня независимо от остальных.

Диаграмма развёртывания описывает физическое размещение компонентов системы на серверах в режиме рабочей эксплуатации. Представлена на рисунке 15.

Диаграмма развёртывания системы

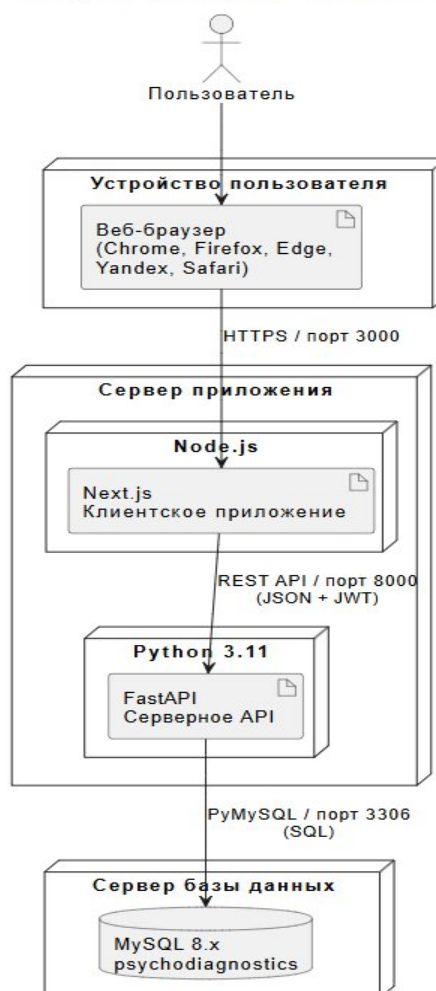


Рисунок 15 – Диаграмма развертывания системы

Из диаграммы видно, что пользователь обращается через веб–браузер к фронтенд–серверу Next.js, который через REST API взаимодействует с API–сервером FastAPI, а тот, в свою очередь, обращается к серверу СУБД MySQL. Каждый узел может быть размещён на отдельной физической машине.

Диаграмма вариантов использования для пользователя описывает все сценарии работы зарегистрированного пользователя и гостя – от регистрации до скачивания отчёта. На диаграмме отражены 21 вариант использования, сгруппированные в пять смысловых пакетов: доступ без авторизации, учётная запись, психодиагностические методики, каталог профессий, рекомендация и отчёт.

Диаграмма вариантов использования для администратора описывает дополнительные административные функции, доступные пользователю с признаком администратора. Администратор наследует все возможности обычного пользователя и получает дополнительно 15 административных вариантов использования: управление пользователями, отчётность по другим пользователям, просмотр статистики и редактирование каталога профессий.

Диаграмма последовательности показывает порядок обмена сообщениями между компонентами системы на примере ключевого сценария «пользователь отправляет ответы теста и получает рекомендацию».

Данный инструмент моделирования является полезным и информативным, поскольку он позволяет разработчикам, аналитикам и техническим специалистам в наглядной форме воспринять хронологию взаимодействия между отдельными модулями, сервисами и объектами системы без необходимости погружения в исходный код.

Диаграмма последовательности представлена на рисунке 16.

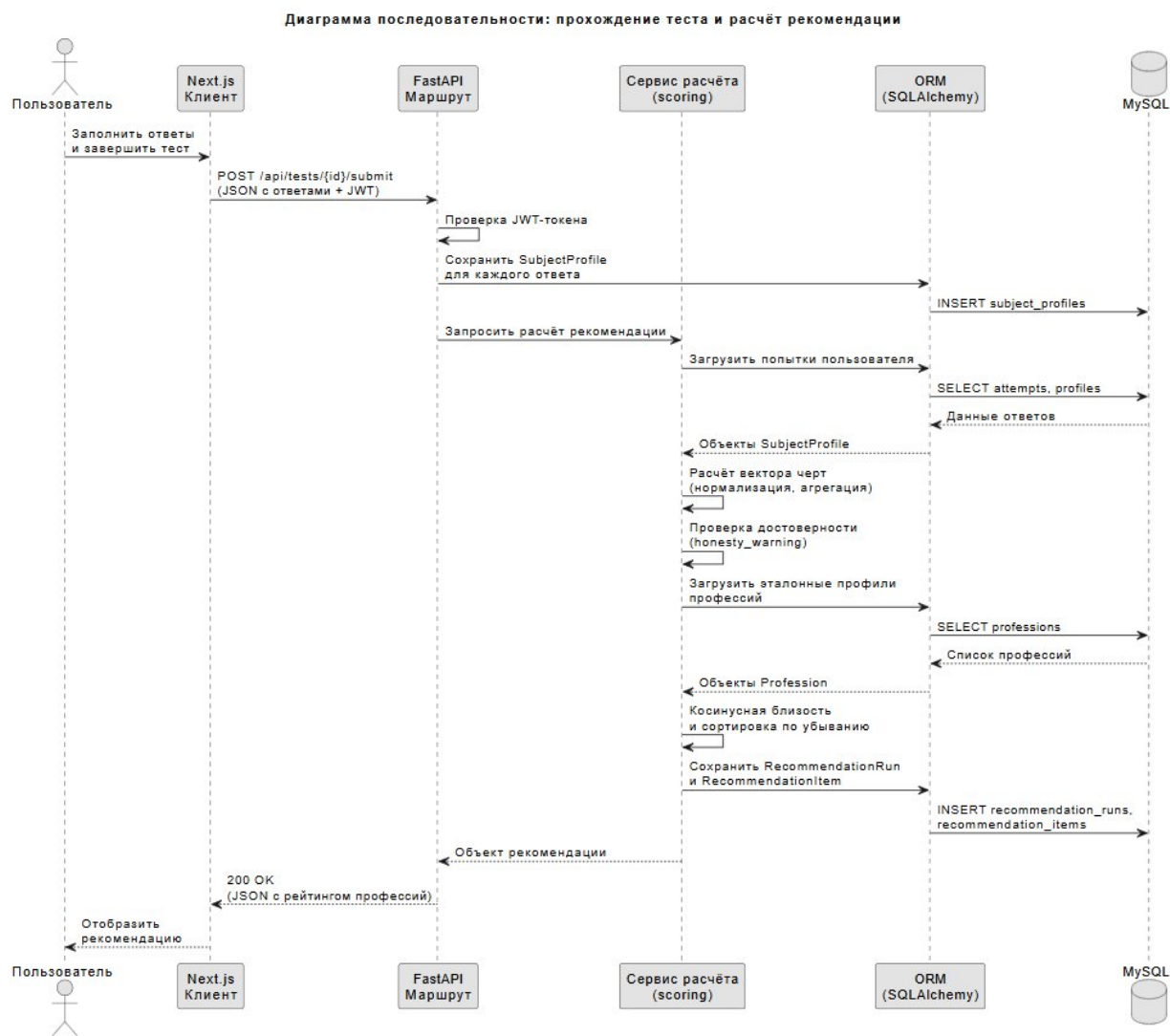


Рисунок 16 – Диаграмма последовательности для сценария расчёта рекомендации

Диаграмма показывает, что один запрос пользователя инициирует цепочку из нескольких вызовов на сервере (сохранение ответов, расчёт вектора черт, сравнение с эталонами, формирование пояснения), которые выполняются в рамках единой транзакции и возвращают пользователю готовый результат.

Диаграмма деятельности описывает полный пользовательский сценарий работы с системой – от регистрации до получения отчёта. Представлена на рисунке 17.

Диаграмма деятельности: пользовательский сценарий

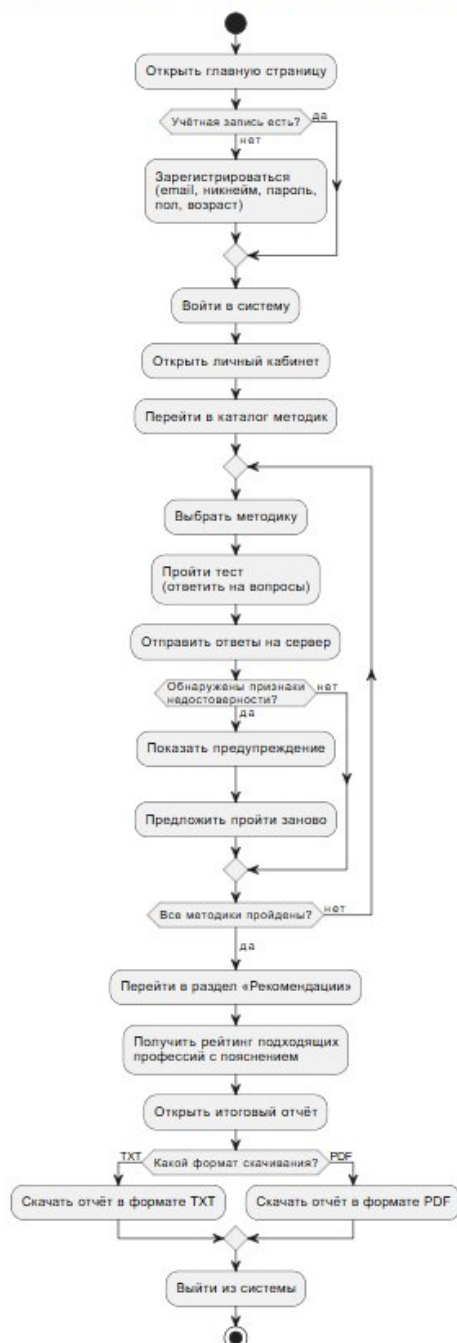


Рисунок 17 – Диаграмма деятельности пользователя системы

Диаграмма наглядно представляет порядок действий пользователя, точки ветвления (проверка достоверности ответов, возможность повторного прохождения теста) и итоговые состояния процесса.

Приведённый набор из нескольких UML–диаграмм даёт целостное представление об архитектуре системы как со стороны структуры (классы, сущности, компоненты, физическое размещение), так и со стороны поведения (варианты использования, последовательность взаимодействия, поток работы пользователя). Подробная реализация каждого элемента раскрывается в последующих подразделах.

1.4.2 Проектирование базы данных

Для реализации реляционной базы данных в качестве инструмента проектирования был выбран MySQL Workbench. Данное средство является промышленным стандартом, обеспечивает надёжное хранение данных и удобный визуальный интерфейс для работы с большими объёмами информации, а также позволяет выполнять прямое моделирование структур, построение ER–диаграмм и генерацию SQL–кода.

На первом этапе было создано новое подключение к серверу баз данных. В окне настройки указаны параметры подключения: имя соединения, метод подключения (Standard TCP/IP), адрес сервера (127.0.0.1), порт (3306), имя пользователя (root) и пароль, сохранённый в защищённом хранилище. Представлено на рисунке 18.

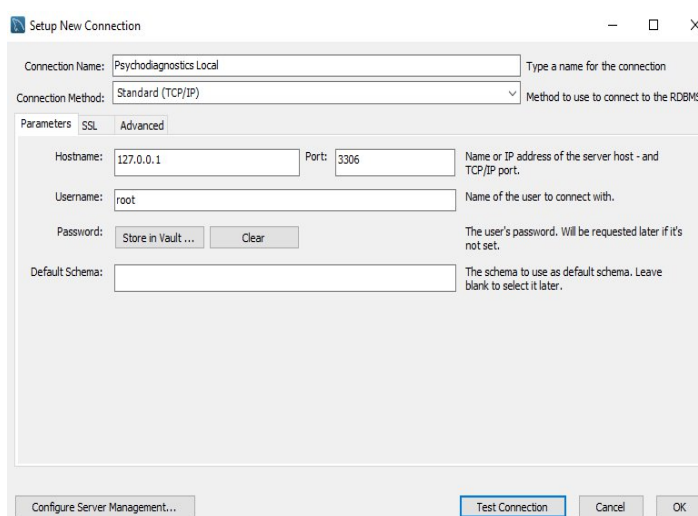


Рисунок 18 – Создание подключения к серверу MySQL в MySQL Workbench

После заполнения параметров было выполнено тестирование соединения с помощью кнопки Test Connection. Успешный результат подтвердил, что сервер MySQL запущен и доступен по указанному адресу с указанными учётными данными. Представлено на рисунке 19.

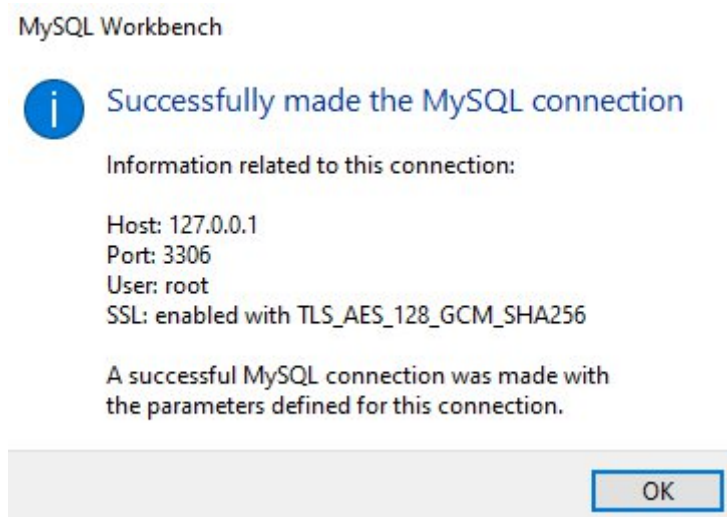


Рисунок 19 – Подтверждение успешного подключения к серверу MySQL

После сохранения подключения была создана целевая схема (база данных) с именем psychodiagnostics. В дальнейшем в этой базе размещались все таблицы системы, связи между ними и данные. Представлено на рисунке 20.

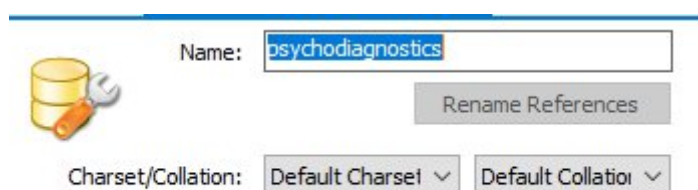


Рисунок 20 – Создание схемы psychodiagnostics в MySQL Workbench

Для хранения информации разработанной системы создано 12 таблиц, объединённых в четыре логических блока: учётные данные пользователей, методики и попытки прохождения, профессии и их требования, рекомендации и отчёты. Перечень таблиц:

- users – содержит данные учётных записей зарегистрированных пользователей. Данные представлены в таблице 4 ;
- user_trait_scores – содержит агрегированные баллы пользователя по чертам личности. Данные представлены в таблице 5 ;
- test_methods – содержит каталог психодиагностических методик системы. Данные представлены в таблице 6 ;
- test_questions – содержит вопросы методик. Данные представлены в таблице 7 ;
- test_options – содержит варианты ответов на вопросы. Данные представлены в таблице 8 ;
- test_attempts – содержит попытки прохождения методик пользователями. Данные представлены в таблице 9 ;
- subject_profiles – содержит результаты ответов пользователя по предметам в рамках попытки. Данные представлены в таблице 10 ;
- professions – содержит каталог профессий с описаниями. Данные представлены в таблице 11 ;
- profession_traits – содержит эталонные значения требуемых черт для каждой профессии. Данные представлены в таблице 12 ;
- recommendation_runs – содержит факты расчёта рекомендации для пользователя. Данные представлены в таблице 13 ;
- recommendation_items – содержит позиции рейтинга подходящих профессий в рамках одной рекомендации. Данные представлены в таблице 14 ;
- report_files – содержит содержимое сформированных итоговых отчётов. Данные представлены в таблице 15 ;

Таблица 4 – Атрибуты таблицы users

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INT	Идентификатор пользователя	PK	1
email	VARCHAR(255)	Адрес электронной почты	UQ	user@mail.ru
nickname	VARCHAR(64)	Никнейм пользователя	UQ	ivan_petrov
password_hash	VARCHAR(255)	Хеш пароля по алгоритму bcrypt		2b\$12 KIXh...
gender	VARCHAR(32)	Пол пользователя		мужской
age_full_years	INT	Возраст в полных годах		22
is_admin	BOOLEAN	Признак администратора		false
created_at	DATETIME	Дата и время регистрации		46169.4375
deleted_at	DATETIME	Дата и время удаления учётной записи		null

Таблица 5 – Атрибуты таблицы user_trait_scores

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INT	Идентификатор записи балла	PK	1
user_id	INT	Идентификатор пользователя	FK	1
trait_key	VARCHAR(128)	Название черты (универсальной шкалы)		Аналитика
score	FLOAT	Балл пользователя по черте (от 0 до 1)		0.85
created_at	DATETIME	Дата и время расчёта		2026-05-27 11:00

Таблица 6 – Атрибуты таблицы test_methods

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INT	Идентификатор методики	PK	2
name	VARCHAR(255)	Название методики	UQ	Тест 2 – ДДО Климова
description	VARCHAR(1024)	Краткое описание методики		Определяет предпочтительный тип...
method_type	VARCHAR(64)	Технический тип методики		klimov_ddo
status	VARCHAR(32)	Статус методики		NOT_STARTED
has_standard_keys	BOOLEAN	Признак стандартизированных ключей		true
has_validity_scale	BOOLEAN	Признак наличия шкалы достоверности		true
is_system	BOOLEAN	Признак системной (нередактируемой) методики		true
created_at	DATETIME	Дата и время создания записи		2026-05-27 09:00

Таблица 7 – Атрибуты таблицы test_questions

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INT	Идентификатор вопроса	PK	1
method_id	INT	Идентификатор методики	FK	2
prompt	VARCHAR(1024)	Текст вопроса		Что вам ближе: А) Общаться с людьми / В) Работать с техникой
position	INT	Порядковый номер вопроса в методике		1
question_type	VARCHAR(64)	Технический тип вопроса		klimov_choice

Таблица 8 – Атрибуты таблицы test_options

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INT	Идентификатор варианта ответа	PK	1
question_id	INT	Идентификатор вопроса	FK	1
label	VARCHAR(255)	Текст варианта, отображаемый пользователю		А) Общаться с людьми
value	VARCHAR(255)	Машинное значение варианта		А
score	FLOAT	Балл, присваиваемый за выбор варианта		1.0
Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INT	Идентификатор варианта ответа	PK	1
question_id	INT	Идентификатор вопроса	FK	1

Таблица 9 – Атрибуты таблицы test_attempts

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INT	Идентификатор попытки	PK	1
user_id	INT	Идентификатор пользователя	FK	1
method_id	INT	Идентификатор методики	FK	2
status	VARCHAR(32)	Статус попытки		COMPLETED
started_at	DATETIME	Дата и время начала попытки		2026-05-27 10:35
completed_at	DATETIME	Дата и время завершения попытки		2026-05-27 10:48
honesty_warning	VARCHAR(255)	Текст предупреждения о возможной недостоверности		null

Таблица 10 – Атрибуты таблицы subject_profiles

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INT	Идентификатор записи результата	PK	1
attempt_id	INT	Идентификатор попытки	FK	1
subject_name	VARCHAR(128)	Название предмета или показателя		Математика
evaluation	FLOAT	Объективная оценка (или числовое значение ответа)		5.0
self_assessment	FLOAT	Самооценка пользователя		8.0
motivation	FLOAT	Уровень мотивации		9.0

Таблица 11 – Атрибуты таблицы professions

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INT	Идентификатор профессии	PK	1
name	VARCHAR (255)	Название профессии	UQ	Психолог
description	VARCHAR (2048)	Описание профессии		Помогает людям...
pros	JSON	Список преимуществ работы по профессии		["востребованность", "социальная значимость"]
cons	JSON	Список недостатков и трудностей		["эмоциональная нагрузка"]
income_profile	VARCHAR (1024)	Описание уровня дохода		60 000 – 120 000 Р
popularity_in_russia	VARCHAR (128)	Оценка популярности на российском рынке		средняя
directions	JSON	Возможные направления подготовки		["клиническая психология", "педагогика"]
difficulties	JSON	Типичные затруднения		["длительное обучение"]
created_at	DATETIME	Дата добавления в каталог		2026-05-27 09:00

Таблица 12 – Атрибуты таблицы profession_traits

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INT	Идентификатор записи требования	PK	1
profession_id	INT	Идентификатор профессии	FK	1
trait_key	VARCHAR(128)	Название черты (универсальной шкалы)		Люди
desired_score	FLOAT	Эталонное значение черты (от 0 до 1)		1.0
weight	FLOAT	Весовой коэффициент черты в итоговом расчёте		1.0

Таблица 13 – Атрибуты таблицы recommendation_runs

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INT	Идентификатор расчёта рекомендации	PK	1
user_id	INT	Идентификатор пользователя	FK	1
attempt_id	INT	Идентификатор попытки, послужившей основой	FK	1
llm_explanation	TEXT	Сформированное текстовое пояснение к рекомендации		Ваш профиль наиболее близок к профессии...
honesty_warning	VARCHAR(255)	Текст предупреждения о возможной недостоверности		null
created_at	DATETIME	Дата и время расчёта		2026-05-27 11:05

Таблица 14 – Атрибуты таблицы recommendation_items

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INT	Идентификатор позиции рейтинга	PK	1
run_id	INT	Идентификатор расчёта рекомендации	FK	1
profession_id	INT	Идентификатор профессии	FK	11
score	FLOAT	Степень соответствия профилю пользователя (процент)		92.45
reasoning_summary	VARCHAR(512)	Краткое пояснение причин включения		Ключевые совпадения: Знаки, Аналитика, Структура

Таблица 15 – Атрибуты таблицы report_files

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INT	Идентификатор файла отчёта	PK	1
run_id	INT	Идентификатор расчёта рекомендации	FK, UQ	1
format	VARCHAR(32)	Формат отчёта		pdf
content	TEXT	Текстовое содержимое отчёта		Отчёт по результатам тестирования...
created_at	DATETIME	Дата и время формирования отчёта		2026-05-27 11:05

Эти таблицы обеспечивают полное отображение всех данных, используемых в системе.

1.4.3 Проектирование модуля расчёта рекомендации

Модуль расчёта рекомендации – это набор формул, который превращает ответы пользователя на пять психодиагностических методик в рейтинг подходящих профессий из каталога. В каталоге системы заранее подготовлено 37 профессий, каждая из которых описана эталонным набором значений по 9 универсальным шкалам черт личности (Люди, Техника, Знаки, Творчество, Природа, Аналитика, Практика, Структура, Управление). Задача модуля – построить такой же 9–значный вектор для пользователя по его ответам и сравнить его с каждой профессией из каталога.

Алгоритм работы модуля состоит из шести последовательных этапов.

Этап 1. Сбор ответов пользователя. Модуль загружает из базы данных последние завершённые попытки прохождения по каждой методике. Если методика не пройдена, она просто не учитывается; если не пройдено ни одной – расчёт не запускается.

Этап 2. Построение вектора по каждой методике. Для каждой пройденной методики строится свой вектор из 9 значений – по тому же набору универсальных шкал. У каждой методики свой способ преобразования ответов в вектор: у школьного теста используется мотивация пользователя по предметам, у ДДО Климова – частоты выборов типов профессий, у RIASEC и Big Five – средние баллы по группам утверждений, у Рокича – ранги ценностей.

Этап 3. Нормализация и агрегация. Каждый вектор сначала нормируется – все значения делятся на максимум, чтобы наиболее выраженная шкала получила значение 1. Затем все векторы складываются с заранее заданными весами методик в один итоговый вектор пользователя. Веса методик приведены в таблице 1.3 (ДДО Климова – 0,25; Holland RIASEC – 0,25; Big Five – 0,20; стартовый тест – 0,15; Рокич – 0,15).

Этап 4. Контроль достоверности. Параллельно с расчётом анализируется согласованность ответов в стартовом тесте: среднее расхождение между объективной оценкой и самооценкой пользователя по предметам, а также средний уровень мотивации. Если расхождение слишком большое или мотивация слишком низкая, система формирует предупреждение и предлагает пройти тест повторно.

Этап 5. Сравнение с эталонами профессий. Итоговый вектор пользователя сравнивается с эталонными векторами всех 37 профессий из каталога по методу косинусной близости. Косинусная близость показывает, насколько похожи два набора значений по своему общему «направлению», и принимает значения от 0 (полная непохожесть) до 1 (полное совпадение). Полученное значение умножается на 100 и округляется – это и есть процент соответствия, который видит пользователь.

Этап 6. Сортировка и сохранение результата. Профессии упорядочиваются по убыванию степени соответствия, для итоговой рекомендации отбираются пять верхних, и для каждой формируется краткое пояснение «Ключевые совпадения» – это три шкалы, по которым профиль пользователя ближе всего к эталонному профилю профессии. Все результаты сохраняются в базу данных: запись расчёта рекомендации, пять позиций рейтинга и итоговый текстовый отчёт для последующего скачивания.

В качестве меры сравнения векторов выбрана именно косинусная близость, а не, например, евклидово расстояние. Это сделано потому, что для задачи профотбора важно сходство пропорции черт между собой, а не их абсолютные значения. Два пользователя с одинаковыми соотношениями шкал, но разной общей выраженностью ответов должны получать одни и те же профессиональные рекомендации – именно это свойство и обеспечивает косинусная близость.

Спроектированный модуль реализует детерминированный алгоритм: при одинаковых ответах пользователь всегда получает одну и ту же рекомендацию.

Это отличает разрабатываемую систему от систем на основе машинного обучения, где результат может зависеть от обучающей выборки и быть непредсказуемым. Использование заранее заданных весов методик и эталонных профилей профессий обеспечивает прозрачность и объективность работы алгоритма.

Нормализация вектора пользователя по отдельной методике. Каждое значение шкалы делится на максимальное значение в наборе, в результате наиболее выраженная шкала получает значение 1, а остальные пропорционально уменьшаются в диапазон от 0 до 1. Представлено на формуле 1.

$$\hat{v}_i = \frac{v_i}{\max(v_1, v_2, \dots, v_9)} \quad (1)$$

$\hat{v}_i$ – значение i-й шкалы в исходном векторе пользователя по отдельной методике;

v_i – нормализованное значение i-й шкалы;

$\max(v_1, v_2, \dots, v_9)$ – максимальное значение среди всех девяти универсальных шкал вектора.

Агрегация векторов пяти методик в один итоговый вектор пользователя. Каждый нормализованный вектор по методике умножается на свой весовой коэффициент, все произведения суммируются, и сумма делится на сумму использованных весов, что является классическим взвешенным средним. Представлено на формуле 2.

$$v_{user} = \frac{\sum_{m=1}^M \omega_m \cdot \hat{v}_m}{\sum_{m=1}^M \omega_m} \quad (2)$$

v_{user} – итоговый агрегированный вектор пользователя;

M – количество пройденных пользователем методик;

ω_m – весовой коэффициент m-й методики;

$\hat{v}_m$ – нормализованный вектор пользователя по m-й методике.

Расчёт показателей достоверности ответов пользователя по стартовому тесту. Вычисляются два усреднённых показателя: среднее расхождение между объективной оценкой и самооценкой по предметам и средний уровень мотивации по всем предметам. Представлено на формуле 3.

$$\Delta avg = \frac{1}{N} \sum_{i=1}^N |o_i - s_i|, \quad Mavg = \frac{1}{N} \sum_{i=1}^N m_i \quad (3)$$

Δavg – среднее расхождение между объективной оценкой и самооценкой по предметам;

$Mavg$ – средний уровень мотивации пользователя по всем предметам;

N – количество предметов стартового теста;

o_i – объективная оценка по i-му предмету;

s_i – самооценка пользователя по i-му предмету;

m_i – уровень мотивации пользователя по i-му предмету.

Косинусная близость между итоговым вектором пользователя и эталонным вектором профессии. Метрика оценивает сходство пропорций двух векторов, независимо от их абсолютной величины. Значение функции лежит в диапазоне от 0 (полная непохожесть профилей) до 1 (полное совпадение направлений векторов). Представлено на формуле 4.

$$\cos(V_{user}, V_{prof}) = \frac{\sum_{i=1}^9 u_i p_i}{\sqrt{\sum_{i=1}^9 u_i^2} \sqrt{\sum_{i=1}^9 p_i^2}} \quad (4)$$

V_{user} – итоговый вектор пользователя;

V_{prof} – эталонный вектор требуемых черт профессии;

u_i – значение i-й шкалы в векторе пользователя;

p_i – значение i -й шкалы в эталонном векторе профессии.

Перевод значения косинусной близости в проценты. Полученное значение представляет собой степень соответствия профиля пользователя профессии в процентах с округлением до двух знаков после запятой, отображаемую пользователю в итоговом рейтинге. Представлено на формуле 5.

$$score = \left(\cos(V_{user}, V_{prof}) \right) \cdot 100 \quad (5)$$

$score$ – степень соответствия профиля пользователя эталонному профилю профессии, выраженная в процентах;

$\left(\cos(V_{user}, V_{prof}) \right)$ – значение косинусной близости, вычисляемое по формуле.

Расчёт близости по отдельной шкале. Используется при формировании краткого пояснения «Ключевые совпадения». По каждой из девяти универсальных шкал вычисляется значение близости, и три шкалы с наибольшим значением включаются в пояснение к рекомендации. Представлено на формуле 6.

$$closeness = 1 - \frac{|u_i - p_i|}{\dots} \quad (6)$$

$closeness$ – близость по i -й шкале (чем больше значение, тем сильнее совпадение по данной шкале);

u_i – значение i -й шкалы в векторе пользователя;

p_i – значение i -й шкалы в эталонном векторе профессии.

1.4.4 Проектирование интерфейса

Пользовательский интерфейс системы спроектирован с ориентацией на массового пользователя – школьников, студентов и взрослых, не имеющих

специальной подготовки в области информационных технологий. При проектировании были положены в основу следующие базовые принципы.

Единый визуальный стиль. Все страницы выполнены в одной цветовой палитре (мягкие бирюзовые и серо-голубые оттенки), используют единый набор шрифтов и одинаковые элементы оформления (карточки, кнопки, поля ввода). Это снижает когнитивную нагрузку и даёт пользователю ощущение целостности продукта.

Сквозная боковая навигация. В левой части экрана на всех страницах после авторизации присутствует сайдбар с пунктами «Пользователь», «Тестирование», «Профбаза», «Пользователи» (для администратора), «Администрирование» (для администратора). Активный раздел подсвечен, что позволяет пользователю всегда понимать, где он находится.

Логическая группировка функций. Страницы сгруппированы по смысловым блокам: вход и регистрация, личный кабинет, прохождение методик, каталог профессий, рекомендации и отчёт, административная панель. Каждый блок отвечает за свой сценарий и не пересекается с другими.

Ролевая видимость элементов. Обычный пользователь не видит административных разделов; администратор получает дополнительные пункты сайдбара. Это исключает путаницу и снижает риск случайных действий.

Адаптивная вёрстка. Интерфейс корректно отображается на устройствах с различным разрешением экрана – от персональных компьютеров до планшетов и смартфонов. Использование стандартных средств CSS (медиа-запросы, флексбокс-разметка, относительные единицы измерения) обеспечивает автоматическое масштабирование без необходимости поддержки отдельной мобильной версии.

Прозрачная индикация состояний. Для каждой методики в каталоге отображается её статус – «не начата», «в процессе», «пройдена». В личном кабинете виден общий прогресс в виде доли пройденных методик. Все длительные операции сопровождаются индикатором загрузки, ошибки отображаются понятными русскоязычными сообщениями.

Подсказки и валидация форм. Поля ввода сопровождаются пояснениями (например, требование к минимальной длине пароля), а проверка корректности заполнения выполняется как на стороне клиента (мгновенный отклик), так и на стороне сервера (защита от обхода).

Один сценарий – одна кнопка. На каждой странице чётко выделена основная целевая кнопка (войти, зарегистрироваться, открыть тест, скачать отчёт), оформленная контрастным цветом. Вторичные действия (отмена, выход) визуально менее заметны, что направляет внимание пользователя на основной сценарий.

Полная русификация. Все элементы интерфейса, тексты вопросов методик, описания профессий и сообщения системы выполнены на русском языке, что соответствует целевой аудитории – пользователям из Российской Федерации.

Выбор темы фона приведен ниже на рисунке 21.

					ПГУ 1.090501.07.001 ПЗ	Лист
						79
Изм.	Лист	№ докум.	Подпись	Дата		



Рисунок 21 – Основной фон

1.5 Разработка приложения

1.5.1 Реализация БД

Перед началом реализации базы данных её необходимо создать с помощью SQL-запроса:

```
CREATE DATABASE IF NOT EXISTS psychodiagnostics
DEFAULT CHARACTER SET utf8mb4
DEFAULT COLLATE utf8mb4_0900_ai_ci;
```

Кодировка utf8mb4 выбрана для корректного хранения русскоязычного текста и расширенных символов. Затем внутри базы создаются 12 таблиц. Ниже приведены SQL-запросы создания каждой таблицы и краткое описание её назначения.

Листинг 1 – Пользователи (users)

```
CREATE TABLE users (
    id INT PRIMARY KEY AUTO_INCREMENT,
    email VARCHAR(255) NOT NULL UNIQUE,
    nickname VARCHAR(64) NOT NULL UNIQUE,
    password_hash VARCHAR(255) NOT NULL,
    gender VARCHAR(32) NOT NULL,
    age_full_years INT NOT NULL,
    is_admin BOOLEAN DEFAULT FALSE,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    deleted_at DATETIME NULL,
    INDEX idx_users_email (email),
    INDEX idx_users_nickname (nickname)
);
```

Назначение: учётные записи пользователей системы. Поддерживает мягкое удаление через поле deleted_at, ролевое разграничение через is_admin. Пароли хранятся в виде bcrypt-хешей. Индексы ускоряют поиск по email и никнейму при авторизации.

Листинг 2 – Методики (test_methods)

```
CREATE TABLE test_methods (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL UNIQUE,
    description VARCHAR(1024) NOT NULL,
    method_type VARCHAR(64) DEFAULT 'subject_starter',
    status VARCHAR(32) DEFAULT 'NOT_STARTED',
    has_standard_keys BOOLEAN DEFAULT TRUE,
    has_validity_scale BOOLEAN DEFAULT TRUE,
    is_system BOOLEAN DEFAULT FALSE,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

Назначение: каталог психодиагностических методик системы (5 методик: стартовый предметный тест, ДДО Климова, RIASEC, Big Five, Рокич). Поле method_type хранит технический идентификатор методики, используемый алгоритмом расчёта; is_system отмечает методики, недоступные для редактирования через административную панель.

Листинг 3 – Вопросы (test_questions)

```
CREATE TABLE test_questions (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    method_id INT NOT NULL,  
    prompt VARCHAR(1024) NOT NULL,  
    position INT NOT NULL,  
    question_type VARCHAR(64) DEFAULT 'subject_profile',  
    FOREIGN KEY (method_id) REFERENCES test_methods(id) ON  
DELETE CASCADE,  
    INDEX idx_questions_method (method_id)  
);
```

Назначение: библиотека вопросов методик. Поле position задаёт порядок вопросов в рамках одной методики, что позволяет менять последовательность без переделки данных. Связь с test_methods каскадная – при удалении методики удаляются и все её вопросы.

Листинг 4 – Варианты ответов (test_options)

```
CREATE TABLE test_options (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    question_id INT NOT NULL,  
    label VARCHAR(255) NOT NULL,  
    value VARCHAR(255) NOT NULL,  
    score FLOAT DEFAULT 0.0,  
    FOREIGN KEY (question_id) REFERENCES test_questions(id) ON  
DELETE CASCADE,  
    INDEX idx_options_question (question_id)  
);
```

Назначение: варианты ответов для каждого вопроса. Поле label хранит текст, отображаемый пользователю; value – машинное значение, которое сохраняется при выборе; score – балл, присваиваемый за выбор варианта (используется алгоритмом расчёта).

Листинг 5 – Попытки прохождения (test_attempts)

```
CREATE TABLE test_attempts (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    user_id INT NOT NULL,  
    method_id INT NOT NULL,  
    status VARCHAR(32) DEFAULT 'IN_PROGRESS',  
    started_at DATETIME DEFAULT CURRENT_TIMESTAMP,  
    completed_at DATETIME NULL,  
    honesty_warning VARCHAR(255) NULL,
```

```

        FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE
CASCADE,
        FOREIGN KEY (method_id) REFERENCES test_methods(id) ON
DELETE CASCADE,
        INDEX idx_attempts_user (user_id),
        INDEX idx_attempts_method (method_id)
);

```

Назначение: факт прохождения методики пользователем. Поле status принимает значения IN_PROGRESS, COMPLETED, RETAKE_REQUIRED. Пользователь может пройти одну методику несколько раз – алгоритм расчёта использует последнюю завершённую попытку. Поле honesty_warning заполняется при обнаружении признаков недостоверных ответов.

Листинг 6 – Результаты по предметам (subject_profiles)

```

CREATE TABLE subject_profiles (
    id INT PRIMARY KEY AUTO_INCREMENT,
    attempt_id INT NOT NULL,
    subject_name VARCHAR(128) NOT NULL,
    evaluation FLOAT NOT NULL,
    self_assessment FLOAT NOT NULL,
    motivation FLOAT NOT NULL,
    FOREIGN KEY (attempt_id) REFERENCES test_attempts(id) ON
DELETE CASCADE,
    UNIQUE KEY uk_attempt_subject (attempt_id, subject_name),
    INDEX idx_profiles_subject (subject_name)
);

```

Назначение: результаты ответов пользователя по предметам (для стартового теста) или числовые значения ответов (для остальных методик). Хранит три шкалы – evaluation, self_assessment, motivation. Ограничение уникальности на пару полей (attempt_id, subject_name) исключает дублирование одного предмета в рамках одной попытки.

Листинг 7 – Баллы пользователя по чертам (user_trait_scores)

```

CREATE TABLE user_trait_scores (
    id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT NOT NULL,
    trait_key VARCHAR(128) NOT NULL,
    score FLOAT NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE
CASCADE,
    INDEX idx_trait_user (user_id),

```

```

        INDEX idx_trait_key (trait_key)
    );

```

Назначение: хранит итоговые баллы пользователя по каждой из 9 универсальных шкал (Люди, Техника, Знаки и так далее). Заполняется автоматически при каждом расчёте рекомендации. Используется для отображения профиля пользователя в личном кабинете и при пересчёте рекомендации.

Листинг 8 – Профессии (professions)

```

CREATE TABLE professions (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL UNIQUE,
    description VARCHAR(2048) NOT NULL,
    pros JSON,
    cons JSON,
    income_profile VARCHAR(1024) DEFAULT '',
    popularity_in_russia VARCHAR(128) DEFAULT '',
    directions JSON,
    difficulties JSON,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

```

Назначение: каталог из 37 профессий, актуальных для российского рынка труда. Поля типа JSON хранят списки строк (преимущества, недостатки, направления подготовки, типичные затруднения), что избавляет от необходимости создавать отдельные вспомогательные таблицы. Поле popularity_in_russia принимает значения «низкая», «средняя», «высокая».

Листинг 9 – Эталонные черты профессий (profession_traits)

```

CREATE TABLE profession_traits (
    id INT PRIMARY KEY AUTO_INCREMENT,
    profession_id INT NOT NULL,
    trait_key VARCHAR(128) NOT NULL,
    desired_score FLOAT NOT NULL,
    weight FLOAT NOT NULL,
    FOREIGN KEY (profession_id) REFERENCES professions(id) ON
DELETE CASCADE,
    INDEX idx_traits_profession (profession_id),
    INDEX idx_traits_key (trait_key)
);

```

Назначение: задаёт эталонный профиль профессии – желаемые значения по каждой из 9 универсальных шкал. Поле `desired_score` лежит в диапазоне от 0 до 1, `weight` – весовой коэффициент важности данной черты для профессии. Эти данные использует алгоритм при расчёте косинусной близости.

Листинг 10 – Расчёты рекомендаций (`recommendation_runs`)

```
CREATE TABLE recommendation_runs (
  id INT PRIMARY KEY AUTO_INCREMENT,
  user_id INT NOT NULL,
  attempt_id INT NOT NULL,
  llm_explanation TEXT NOT NULL,
  honesty_warning VARCHAR(255) NULL,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE
CASCADE,
  FOREIGN KEY (attempt_id) REFERENCES test_attempts(id) ON
DELETE CASCADE,
  INDEX idx_runs_user (user_id),
  INDEX idx_runs_attempt (attempt_id)
);
```

Назначение: факт расчёта рекомендации для пользователя. Хранит сформированное текстовое пояснение и предупреждение о возможной недостоверности ответов (если оно было сформировано). Один пользователь может иметь несколько записей расчёта – при пересчёте рекомендации создаётся новая запись, а не перезаписывается старая.

Листинг 11 – Позиции рейтинга (`recommendation_items`)

```
CREATE TABLE recommendation_items (
  id INT PRIMARY KEY AUTO_INCREMENT,
  run_id INT NOT NULL,
  profession_id INT NOT NULL,
  score FLOAT NOT NULL,
  reasoning_summary VARCHAR(512) NOT NULL,
  FOREIGN KEY (run_id) REFERENCES recommendation_runs(id) ON
DELETE CASCADE,
  FOREIGN KEY (profession_id) REFERENCES professions(id),
  INDEX idx_items_run (run_id),
```

```

        INDEX idx_items_profession (profession_id)
    );

```

Назначение: позиции итогового рейтинга профессий в рамках одного расчёта рекомендации. Поле score хранит степень соответствия в процентах (от 0 до 100), reasoning_summary – краткое пояснение «Ключевые совпадения: ...». Для одного расчёта обычно сохраняется пять позиций.

Листинг 12 – Файлы отчётов (report_files)

```

CREATE TABLE report_files (
    id INT PRIMARY KEY AUTO_INCREMENT,
    run_id INT NOT NULL UNIQUE,
    format VARCHAR(32) DEFAULT 'txt',
    content TEXT NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (run_id) REFERENCES recommendation_runs(id) ON
DELETE CASCADE,
    INDEX idx_reports_run (run_id)
);

```

Назначение: содержимое сформированного итогового отчёта пользователя. Связь «один-к-одному» с recommendation_runs обеспечивается ограничением UNIQUE на поле run_id. Поле format хранит формат отчёта (pdf), content – само текстовое содержимое.

После выполнения всех приведённых запросов база данных полностью готова к работе. Структура из 12 нормализованных таблиц с явной ссылочной целостностью, индексами на внешние ключи и ограничениями уникальности обеспечивает надёжное хранение данных и быстрое выполнение типовых запросов системы (загрузка ответов пользователя, расчёт рекомендации, формирование статистики для администратора).

В разрабатываемом проекте создание таблиц выполняется автоматически при первом запуске серверной части благодаря использованию ORM SQLAlchemy: при старте приложения система проверяет наличие таблиц и создаёт недостающие. Альтернативно структура базы может быть применена вручную через выполнение приведённых SQL-запросов в MySQL Workbench, что особенно удобно при развёртывании системы на удалённом сервере.

1.5.2 Реализация серверной части

Серверная часть автоматизированной системы – это веб-приложение на языке Python, которое принимает запросы от браузера пользователя, выполняет всю бизнес-логику (проверку пароля, расчёт результатов теста, формирование рекомендации) и работает с базой данных. Серверная часть не имеет собственного интерфейса: она отдаёт данные в формате JSON, а отображением занимается клиентская часть.

Весь серверный код размещён в каталоге backend/ и организован следующим образом:

- main.py – точка входа в приложение
- config – настройки и переменные окружения
- db – подключение к базе данных
- models – классы-таблицы (SQLAlchemy)
- schemas – структуры для проверки запросов и ответов (Pydantic)
- routes – HTTP-эндпоинты (адреса вида /api/...)
- services – бизнес-логика
- tests – автоматические тесты на pytest

Серверная часть автоматизированной системы приведена на рисунке 22.

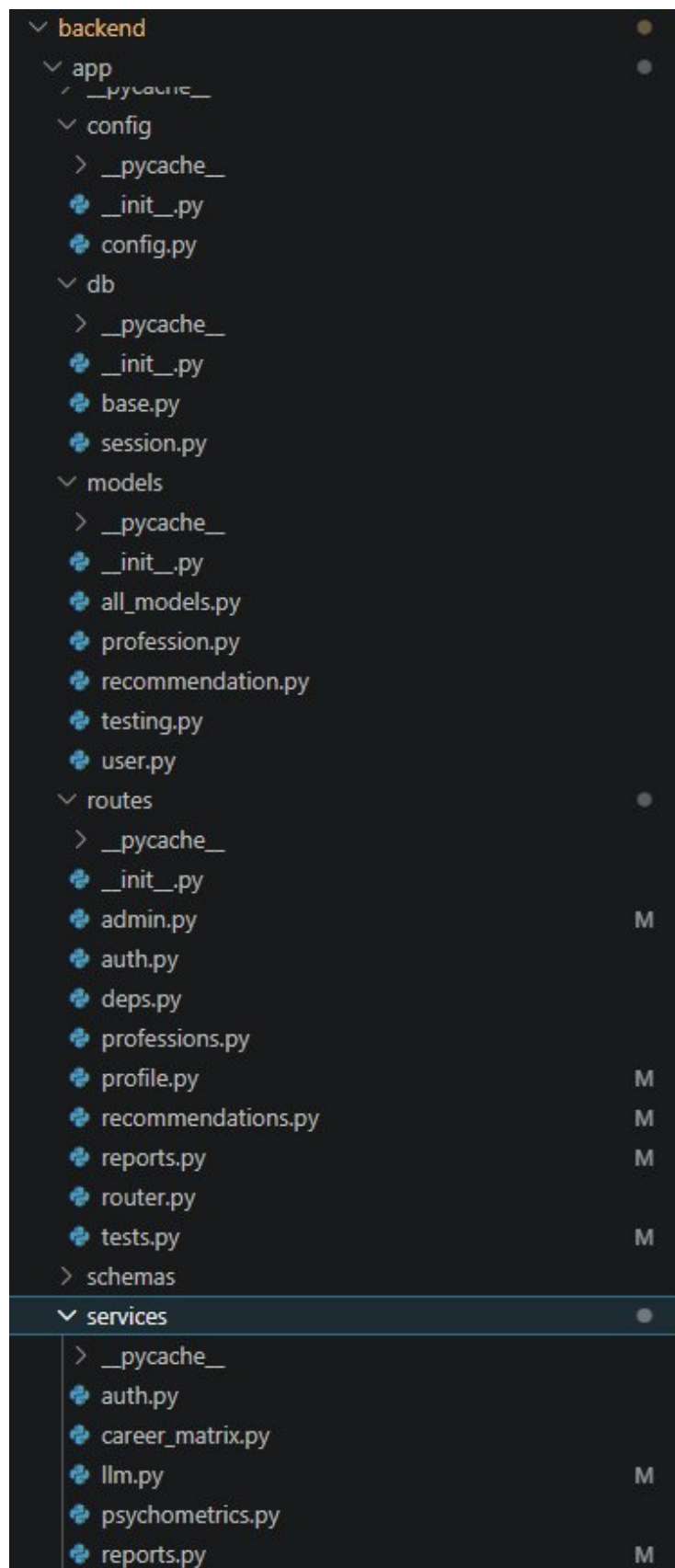


Рисунок 22 – Серверная часть автоматизированной системы

Такое разделение реализует принцип слоистой архитектуры: каждая папка отвечает за свою задачу и не пересекается с другими. Если нужно изменить структуру таблицы – правится файл в `models/`; если изменить логику расчёта – `services/scoring.py`; если изменить адрес или формат HTTP-запроса – соответствующий файл в `routes/`. Это упрощает поиск нужного места при доработке и снижает риск что-то сломать в другом месте.

Ниже в листинге 13 подробнее рассмотрены ключевые файлы серверной части.

Листинг 13 – Точка входа: `backend/app/main.py`

```
from contextlib import asynccontextmanager
from fastapi import FastAPI
from app.config.config import get_settings
from app.db.base import Base
from app.db.session import SessionLocal, engine
from app.models import all_models
from app.routes.router import api_router
from app.services.seed import seed_database
@asynccontextmanager
async def lifespan(_ : FastAPI):
    Base.metadata.create_all(bind=engine)
    with SessionLocal() as session:
        seed_database(session)
    yield
def create_app() -> FastAPI:
    settings = get_settings()
    app = FastAPI(title="Psychodiagnostics API",
version="0.1.0", lifespan=lifespan)
    app.include_router(api_router, prefix=settings.api_prefix)
    @app.get("/health")
    def healthcheck() -> dict[str, str]:
        return "status": "ok"
```

Назначение: главный файл приложения. При первом запуске автоматически создаёт все 12 таблиц в базе данных и наполняет их начальными данными (методики, профессии, учётная запись администратора). Подключает все HTTP-маршруты с префиксом `/api`. Добавляет служебный эндпоинт `/health` для проверки работоспособности сервера.

Листинг 14 – Настройки: backend/app/config/config.py

```
from functools import lru_cache
from pydantic_settings import BaseSettings, SettingsConfigDict
class Settings(BaseSettings):
    app_env: str = "development"
    api_prefix: str = "/api"
    database_url: str =
"sqlite:///./database/runtime/psychodiagnostics.db"
    jwt_secret: str = "change-me"
    jwt_expire_minutes: int = 720
    admin_email: str = "admin@example.com"
    admin_password: str = "admin12345"
    model_config = SettingsConfigDict(env_file=".env",
env_file_encoding="utf-8")
    def get_settings() -> Settings:
        return Settings()
```

Назначение: хранит все настройки приложения в одном месте. Реальные значения подгружаются из файла .env в корне проекта (адрес базы данных, секрет JWT, данные администратора). Если файла нет – используются значения по умолчанию. Это позволяет менять настройки без правки кода.

Листинг 15 – Подключение к базе данных: backend/app/db/session.py

```
from sqlalchemy import create_engine
from sqlalchemy.orm import Session, sessionmaker
from app.config.config import get_settings
settings = get_settings()
engine = create_engine(settings.database_url, future=True)
SessionLocal = sessionmaker(bind=engine, autoflush=False,
autocommit=False, future=True)
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

Назначение: создаёт подключение к базе данных по адресу из настроек и предоставляет функцию get_db(), которая выдаёт новую сессию работы с базой на каждый HTTP-запрос. После завершения запроса сессия автоматически закрывается, а при ошибке выполняется откат транзакции – это исключает попадание частично записанных данных в базу.

Листинг 16 – Описание таблицы: backend/app/models/user.py

```
from sqlalchemy import Boolean, DateTime, Integer, String
from sqlalchemy.orm import Mapped, mapped_column, relationship
from app.db.base import Base
class User(Base):
    __tablename__ = "users"
    id: Mapped[int] = mapped_column(primary_key=True)
    email: Mapped[str] = mapped_column(String(255),
unique=True, index=True)
    nickname: Mapped[str] = mapped_column(String(64),
unique=True, index=True)
    password_hash: Mapped[str] = mapped_column(String(255))
    gender: Mapped[str] = mapped_column(String(32))
    age_full_years: Mapped[int] = mapped_column(Integer)
    is_admin: Mapped[bool] = mapped_column(Boolean,
default=False)
    attempts = relationship("TestAttempt",
back_populates="user")
```

Назначение: Python-класс User одновременно описывает таблицу users в базе данных. Каждое поле класса – это столбец таблицы со своим типом данных. Атрибут relationship задаёт связь с другой таблицей (test_attempts) – это и есть тот самый внешний ключ, но описанный в виде объекта. По такому же образцу устроены и остальные 11 моделей в каталоге models/.

Листинг 17 – Проверка входных данных: backend/app/schemas/auth.py (фрагмент)

```
from pydantic import BaseModel, EmailStr, Field
class RegisterRequest(BaseModel):
    email: EmailStr
    nickname: str = Field(min_length=2, max_length=64)
    password: str = Field(min_length=8)
    gender: str
    age_full_years: int = Field(ge=14, le=99)
```

Назначение: описание того, как должен выглядеть JSON-запрос на регистрацию. Pydantic автоматически проверяет, что email имеет правильный формат, никнейм – от 2 до 64 символов, пароль – не короче 8 символов, возраст – от 14 до 99 лет. Если данные не проходят проверку, пользователь получает структурированное сообщение об ошибке ещё до того, как запрос попадёт в бизнес-логику.

(фрагмент)

```

from fastapi import APIRouter, Depends, HTTPException, status
from sqlalchemy.orm import Session
from app.db.session import get_db
from app.models.user import User
from app.schemas.auth import RegisterRequest, AuthResponse
from app.services.auth import hash_password,
create_access_token
router = APIRouter(prefix="/auth", tags=["auth"])

    @router.post("/register", response_model=AuthResponse,
status_code=status.HTTP_201_CREATED)
    def register(payload: RegisterRequest, db: Session =
Depends(get_db)):
        if db.query(User).filter(User.email ==
payload.email).one_or_none():
            raise HTTPException(status_code=409, detail="Email
already registered")
        user = User(
            email=payload.email,
            nickname=payload.nickname,
            password_hash=hash_password(payload.password),
            gender=payload.gender,
            age_full_years=payload.age_full_years,
        )
        db.add(user)
        db.commit()
        return AuthResponse(access_token=create_access_token(user),
user=user)

```

Назначение: обработчик HTTP–запроса POST /api/auth/register. Принимает уже проверенные данные регистрации, проверяет, что email ещё не занят, создаёт пользователя в базе с хешированным паролем и возвращает JWT–токен для последующей авторизации. Заметно, что сам обработчик короткий и не содержит сложной логики – он только координирует работу: «проверь – создай – верни». Вся работа (хеширование, выпуск токена) вынесена в сервисный слой.

Листинг 19 – Бизнес–логика: backend/app/services/scoring.py (фрагмент)

```
def generate_recommendation_for_user(db: Session, user: User)
-> RecommendationRun:
    attempts_by_type = latest_completed_attempts(db, user.id)
    if not attempts_by_type:
        raise ValueError("No completed attempts")
    test_vectors, honesty_warning =
compute_test_vectors(attempts_by_type)
    user_vector = aggregate_user_vector(test_vectors)
    ranked = score_professions(db, user_vector)
    llm_explanation = build_explanation(ranked[:5],
honesty_warning)
    run = RecommendationRun(
        user_id=user.id,
        attempt_id=anchor_attempt.id,
        llm_explanation=llm_explanation,
        honesty_warning=honesty_warning,
    )
    db.add(run)
    db.commit()
    return run
```

Назначение: ключевая функция системы – расчёт рекомендации для пользователя. Шаг за шагом выполняет всё, что описано в подразделе 1.4.3: загружает попытки прохождения, строит векторы по методикам, агрегирует их в итоговый профиль пользователя, сравнивает с эталонными профилями профессий, формирует пояснение и сохраняет результат в базу. По такому же принципу устроены и остальные сервисы – каждая функция делает одну понятную работу.

Листинг 20 – Сборка всех маршрутов: backend/app/routes/router.py

```
from fastapi import APIRouter
from app.routes.auth import router as auth_router
from app.routes.profile import router as profile_router
from app.routes.tests import router as tests_router
from app.routes.professions import router as professions_router
from app.routes.recommendations import router as
recommendations_router
from app.routes.reports import router as reports_router
from app.routes.admin import router as admin_router
api_router = APIRouter()
api_router.include_router(auth_router)
api_router.include_router(profile_router)
api_router.include_router(tests_router)
api_router.include_router(professions_router)
api_router.include_router(recommendations_router)
api_router.include_router(reports_router)
api_router.include_router(admin_router)
```

Назначение: собирает все группы маршрутов в один общий роутер, который подключается в main.py. В итоге получаются эндпоинты вида /api/auth/login, /api/tests/catalog, /api/recommendations/latest, /api/admin/stats и так далее. Полный список из 22 эндпоинтов доступен по адресу <http://127.0.0.1:8000/docs> (автоматически генерируемая документация Swagger UI).

Серверная часть организована по принципу слоистой архитектуры: точка входа, настройки, подключение к базе, описание таблиц, проверка входных данных, обработчики HTTP-запросов и бизнес-логика разнесены по отдельным каталогам. Каждый слой имеет чёткую зону ответственности, что упрощает разработку, поиск ошибок и сопровождение системы. Использование фреймворка FastAPI и ORM SQLAlchemy позволило сосредоточиться на бизнес-логике.

1.5.3 Реализация клиентской части

Клиентская часть автоматизированной системы – это веб-приложение на платформе Next.js, которое работает в браузере пользователя. Она отвечает за отображение интерфейса (страницы, формы, кнопки, таблицы), обработку действий пользователя и обмен данными с серверной частью по REST API. Вся работа (проверка пароля, расчёт результатов, формирование рекомендации) выполняется на сервере, а клиентская часть только показывает результаты и принимает ввод.

- app/ – страницы приложения (по одной папке на страницу)
- layout.tsx – корневой шаблон (общая обёртка всех страниц)
- page.tsx – главная страница (лендинг + форма входа)
- login/ – страница входа
- register/ – страница регистрации
- dashboard/ – личный кабинет
- tests/ – каталог методик и страница прохождения теста
- professions/ – профбаза
- recommendations/ – итоговая рекомендация
- report/ – отчёт и скачивание

- admin/ – административная панель
- components/ – переиспользуемые блоки интерфейса
- auth/ – формы входа и регистрации
- dashboard/ – карточки личного кабинета
- layout/ – сайдбар и общий каркас страницы
- testing/ – компоненты прохождения тестов
- professions/ – карточка и список профессий
- recommendations/ – панель рекомендации
- services/ – слой работы с серверным API
- api-client.ts – клиент для запросов к серверу
- api-types.ts – TypeScript-типы для API
- styles/ – стили интерфейса
- globals.css – глобальные стили
- tests/ – автоматические тесты компонентов (Vitest)
- package.json – зависимости и команды
- tsconfig.json – настройки TypeScript
- next.config.ts – настройки Next.js

Такое разделение реализует принцип «папка под страницу – папка под компоненты». Если нужно изменить адрес или содержимое страницы – правится файл в app/; если изменить внешний вид общего блока (например, сайдбара или карточки профессии) – соответствующий компонент в components/. Если изменить адрес или способ обращения к API – единый файл services/api-client.ts. Это упрощает поиск нужного места при доработке и исключает дублирование одинакового кода на разных страницах.

На рисунке 23 подробнее рассмотрены ключевые файлы клиентской части.

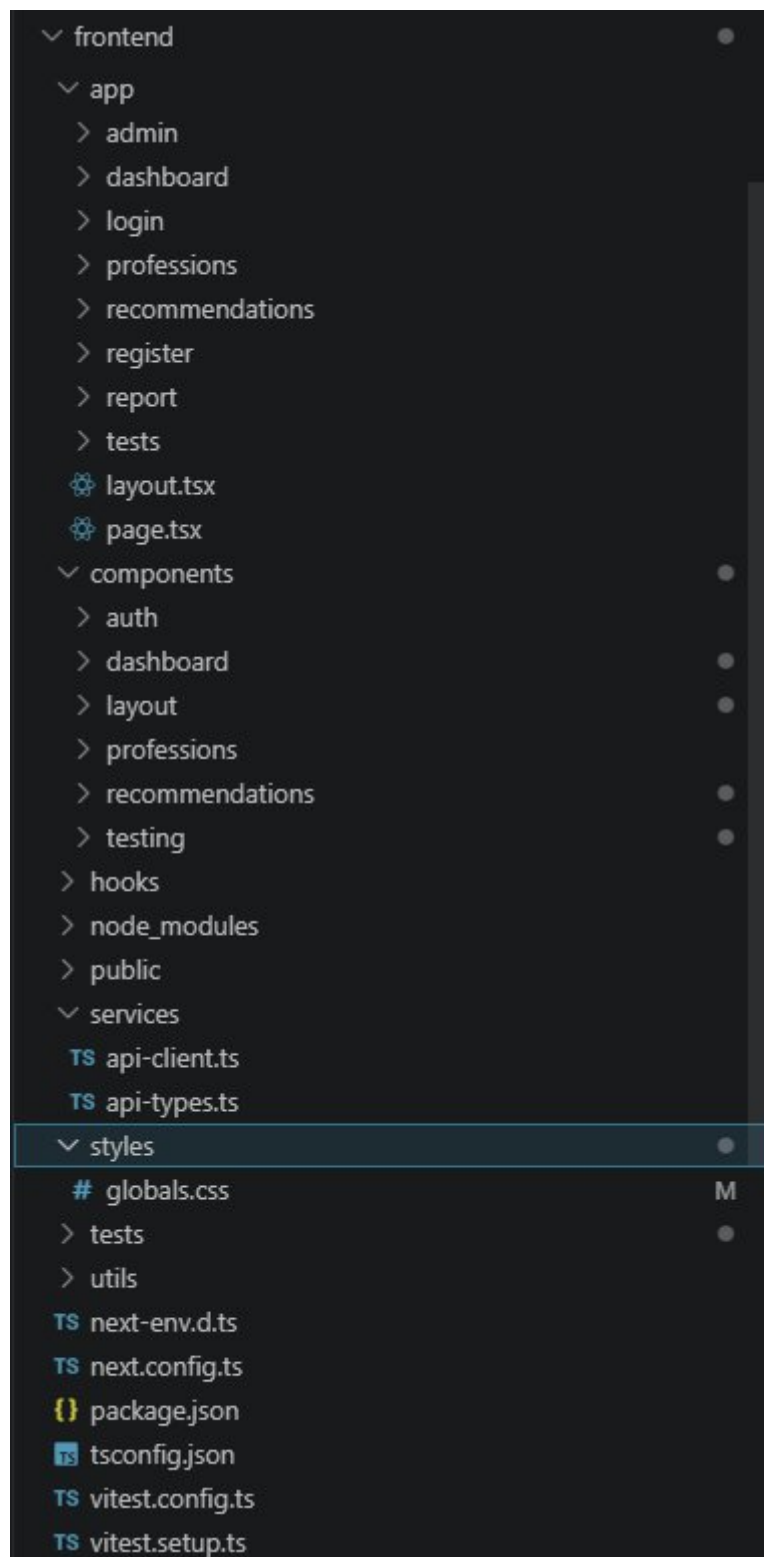


Рисунок 23 – Клиентская часть автоматизированной системы

Листинг 21 – Корневой шаблон: frontend/app/layout.tsx

```
import "../styles/globals.css";
import type Metadata from "next";
export const metadata: Metadata =
  title: "Psychodiagnostics Workspace",
  description: "Автоматизированная система психодиагностики для
задач профотбора",
;
export default function RootLayout(
  children,
: Readonly< children: React.ReactNode >)
return (
  <html lang="ru">
    <body>children</body>
  </html>
);
```

Назначение: корневой шаблон всех страниц приложения. Подключает единый файл стилей, задаёт заголовок вкладки браузера, описание для поисковых систем и язык содержимого (ru). Всё, что находится в этом файле, автоматически применяется ко всем страницам – без необходимости повторять одно и то же в каждой.

Листинг 22 – Главная страница: frontend/app/page.tsx

```
"use client";
import useEffect, useState from "react";
import AuthShell from "@components/auth/auth-shell";
import apiRequest, clearToken, loadToken from
"@services/api-client";
export default function HomePage()
const [isReady, setIsReady] = useState(false);
useEffect(() =>
  const token = loadToken();
  if (!token)
    setIsReady(true);
  return;

  apiRequest("/api/me", , token)
    .then(() => window.location.replace("/dashboard"))
    .catch(() =>
      clearToken();
      setIsReady(true);
    );
, []);
if (!isReady) return null;
return <AuthShell onAuthenticated={() =>
window.location.replace("/dashboard")} />;
```

Назначение: главная страница сайта по адресу /. При открытии страница проверяет, есть ли у пользователя сохранённый токен авторизации в браузере. Если токен есть и он действителен – пользователь автоматически перенаправляется в личный кабинет. Если токена нет – отображается форма входа и регистрации. Это типичный сценарий для веб-приложений: «не заставлять пользователя входить заново, если он уже входил».

Листинг 23 – Слой работы с API: frontend/services/api-client.ts (фрагмент)

```
const API_BASE_URL = process.env.NEXT_PUBLIC_API_BASE_URL ??
"";

export async function apiRequest<T>(
  path: string,
  options: RequestInit = ,
  token?: string,
): Promise<T>
  const headers = new Headers(options.headers ?? );
  if (!headers.has("Content-Type") && options.body)
    headers.set("Content-Type", "application/json");

  if (token)
    headers.set("Authorization", `Bearer $token`);

  const response = await
fetch(`${API_BASE_URL}${path}`, ...options, headers );
  return parseResponse<T>(response);

export function saveToken(token: string)
  window.localStorage.setItem("psychodiagnostics-token",
token);

export function loadToken()
  return
window.localStorage.getItem("psychodiagnostics-token") ?? "";
```

Назначение: единственная точка обращения клиентской части к серверу. Все компоненты вызывают эту функцию, передавая адрес эндпоинта (например, /api/auth/login) и при необходимости – токен авторизации. Функция автоматически добавляет нужные заголовки, отправляет запрос, обрабатывает ответ и возвращает результат в виде типизированного объекта. Кроме того, в этом же файле находятся функции сохранения, загрузки и удаления токена в локальном хранилище браузера.

(фрагмент)

```

"use client";
import { useState } from "react";
import { apiRequest, saveToken } from "@services/api-client";
export function LoginForm( onSuccess : onSuccess: () => void )
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState<string | null>(null);
  async function handleSubmit(event: React.FormEvent)
    event.preventDefault();
    setError(null);
    try
      const response = await apiRequest< access_token:
string >(
      "/api/auth/login",
      method: "POST", body: JSON.stringify( email,
password ) ,
    );
    saveToken(response.access_token);
    onSuccess();
    catch (err)
      setError((err as Error).message);

  return (
    <form onSubmit=handleSubmit>
      <input type="email" value=email onChange=(e) =>
setEmail(e.target.value) />
      <input type="password" value=password onChange=(e) =>
setPassword(e.target.value) />
      error && <p className="error">error</p>
      <button type="submit">Войти</button>
    </form>
  );

```

Назначение: компонент формы входа. Хранит в собственном состоянии введённые email и пароль, при отправке формы вызывает функцию обращения к серверу, сохраняет полученный токен и уведомляет родительский компонент об успешном входе. Если сервер возвращает ошибку (например, неверный пароль), она отображается под формой. По такому же образцу устроены и другие формы – регистрации, прохождения теста, поиска по профбазе.

frontend/components/layout/app–sidebar.tsx (фрагмент)

```
"use client";
const NAV_ITEMS = [
  label: "Пользователь", href: "/dashboard" ,
  label: "Тестирование", href: "/tests" ,
  label: "Профбаза", href: "/professions" ,
];

const ADMIN_ITEMS = [
  label: "Пользователи", href: "/dashboard/users" ,
  label: "Администрирование", href: "/admin" ,
];

export function AppSidebar( isAdmin : isAdmin: boolean )
  return (
    <nav>
      <ul>
        NAV_ITEMS.map((item) => (
          <li key=item.href>
            <a href=item.href>item.label</a>
          </li>
        ))

        isAdmin && ADMIN_ITEMS.map((item) => (
          <li key=item.href>
            <a href=item.href>item.label</a>
          </li>

        ))

      </ul>

    </nav>

  );
```

Назначение: боковая навигация, отображаемая на всех страницах после авторизации. Содержит общие пункты для всех пользователей и дополнительные для администраторов. Признак `isAdmin` передаётся из родительского компонента, который получает его из ответа сервера на запрос `/api/me`. Благодаря этому администратор автоматически видит дополнительные разделы, а обычный пользователь – нет.

Листинг 26 – Прохождение теста: frontend/components/testing/test-runner.tsx

```
"use client";
export function TestRunner( method, onSubmit )
  const [answers, setAnswers] = useState();

  function handleAnswerChange(questionId: number, value:
string)
    setAnswers((prev) => ( ...prev, [questionId]: value ));

  async function handleFinish()
    await apiRequest(`/api/tests/$method.id/submit`,
      method: "POST",
      body: JSON.stringify( answers ),
      , loadToken());
    onSubmit();

  return (
    <div>
      method.questions.map((q) => (
        <div key=q.id>
          <p>q.prompt</p>
          q.options.map((o) => (
            <label key=o.id>
              <input
                type="radio"
                name=`q-$q.id`
                value=o.value
                onChange=(e) => handleAnswerChange(q.id,
e.target.value)
              />
              o.label
            </label>
          ))
        </div>
      ))
      <button onClick=handleFinish>Завершить тест</button>
    </div>
  );
```

Назначение: универсальный компонент прохождения методики. Получает на вход данные методики (вопросы и варианты ответов), хранит в своём состоянии ответы пользователя по мере заполнения и отправляет их на сервер по нажатию кнопки «Завершить тест». Один и тот же компонент используется для разных методик – отличаются только данные. Для специфических методик (Рокич с ранжированием, стартовый тест с тремя шкалами) есть отдельные обёртки в той же папке testing/.

В проекте используются обычные CSS–стили без подключения внешних стилевых фреймворков (Tailwind, Bootstrap не используются). Все глобальные правила – цветовая палитра, шрифты, размеры отступов, оформление кнопок и форм – собраны в одном файле `globals.css`, который подключается в корневом шаблоне `layout.tsx`. Этот подход обеспечивает полный контроль над визуальным оформлением и отсутствие лишних зависимостей.

Клиентская часть организована по принципу разделения «страница – компонент – слой работы с API». Каждая страница в каталоге `app/` отвечает за свой адрес и сценарий, переиспользуемые блоки вынесены в `components/`, а единственная точка обращения к серверу собрана в `services/api-client.ts`. Использование платформы Next.js и библиотеки React позволило сосредоточиться на содержательной части – оформлении интерфейса и работе с данными, – а рутинные задачи (маршрутизация, сборка, серверный рендеринг) обеспечиваются стандартными средствами выбранного стека. Применение TypeScript обеспечивает раннее обнаружение ошибок типов в данных, передаваемых между клиентом и сервером, что повышает надёжность работы приложения.

1.5.4 Реализация модуля расчёта и формирования отчёта

Модуль расчёта и формирования отчёта реализует ключевой сценарий системы: преобразование ответов пользователя на пять психодиагностических методик в итоговый рейтинг подходящих профессий и формирование текстового отчёта, доступного для скачивания. Программно модуль состоит из трёх файлов, размещённых в каталоге `backend/app/services/`:

- `scoring.py` – расчёт интегрального профиля и рейтинга профессий
- `llm.py` – формирование текстового пояснения к рекомендации
- `reports.py` – сборка итогового отчёта в формате PDF

Ниже приведено описание ключевых функций каждого файла.

Расчёт степени соответствия профессий (`scoring.py`)

В основе расчёта рейтинга лежит функция вычисления косинусной близости двух векторов:

Листинг 27 – Вычисления косинусной близости двух векторов

```
def cosine_similarity(left: dict[str, float], right: dict[str, float]) -> float:
    numerator = sum(left[s] * right[s] for s in SCALES)
    left_norm = sqrt(sum(left[s] ** 2 for s in SCALES))
    right_norm = sqrt(sum(right[s] ** 2 for s in SCALES))
    if left_norm == 0 or right_norm == 0:
        return 0.0
    return numerator / (left_norm * right_norm)
```

Функция принимает два вектора по девяти универсальным шкалам (вектор пользователя и эталонный вектор профессии) и возвращает значение от 0 до 1, отражающее степень их совпадения. Используется на этапе сравнения с каталогом профессий.

Листинг 28 – Расчёт рейтинга:

```
def score_professions(db: Session, user_vector: dict[str, float]):
    professions =
db.query(Profession).order_by(Profession.name.asc()).all()
    ranked = []
    for profession in professions:
        profession_vector = profile_for_profession(profession)
        if profession_vector is None:
            continue
        similarity = cosine_similarity(user_vector,
profession_vector)
        score = round(similarity * 100, 2)
        ranked.append((profession, score,
summarize_match(user_vector, profession_vector)))
        ranked.sort(key=lambda item: (-item[1], item[0].name))

    return ranked
```

Функция последовательно сравнивает вектор пользователя с эталонными векторами всех профессий каталога, для каждой пары вычисляет процент соответствия (округлённое значение косинусной близости, умноженное на 100) и формирует краткое пояснение «Ключевые совпадения: ...». Полученный список сортируется по убыванию степени соответствия.

Листинг 29 – generate_recommendation_for_user:

```
def generate_recommendation_for_user(db: Session, user: User)
-> RecommendationRun:
    attempts_by_type = latest_completed_attempts(db, user.id)
    if not attempts_by_type:
        raise ValueError("No completed attempts")

    test_vectors, honesty_warning =
compute_test_vectors(attempts_by_type)
    user_vector = aggregate_user_vector(test_vectors)
    ranked = score_professions(db, user_vector)
    llm_explanation = build_explanation(ranked[:5],
honesty_warning)

    run = RecommendationRun(
        user_id=user.id,
        attempt_id=anchor_attempt.id,
        llm_explanation=llm_explanation,
        honesty_warning=honesty_warning,
    )
    db.add(run)
    db.flush()

    for profession, score, reasoning in ranked[:5]:
        db.add(RecommendationItem(
            run_id=run.id,
            profession_id=profession.id,
            score=score,
            reasoning_summary=reasoning,
        ))

    report_text = render_report_text(
        user=user, run=run, items=ranked[:5],
        attempts_by_type=attempts_by_type,
test_vectors=test_vectors,
    )
    db.add(ReportFile(run_id=run.id, content=report_text,
format="txt"))
    db.commit()
    return run
```

Функция последовательно выполняет все шесть этапов алгоритма: загрузка пройденных попыток, построение векторов по методикам, агрегирование в итоговый профиль пользователя, проверка достоверности, расчёт рейтинга профессий, формирование пояснения. Все результаты сохраняются в трёх связанных таблицах базы данных: recommendation_runs (запись расчёта), recommendation_items (пять верхних позиций рейтинга) и report_files (текст отчёта для последующей выгрузки).

Листинг 30 – Формирование текстового пояснения (llm.py)

```
def build_explanation(ranked_items, honesty_warning):
    top_names = ", ".join(item[0].name for item in
ranked_items[:3])
    base = (
        "Рекомендация сформирована на основе результатов пяти
методик и "
        "сопоставления с профилями профессий. "
        f"Наиболее близкими направлениями для текущего профиля
являются: top_names."
    )
    if honesty_warning:
        return f"base Дополнительно обратите внимание:
honesty_warning"
    return base
```

Функция формирует пояснительный текст к рекомендации, перечисляя три наиболее подходящие профессии. При наличии предупреждения о недостоверности ответов оно добавляется отдельной фразой. Текст сохраняется в поле llm_explanation записи расчёта рекомендации и отображается пользователю в личном кабинете.

Листинг 31 – Формирование текстового отчёта (reports.py)

```
def render_report_text(*, user, run, items, attempts_by_type,
test_vectors) -> str:
    klimov_type =
_dominant_klimov_type(attempts_by_type.get("klimov_ddo"))
    motivation_scale =
_leading_scale_name(test_vectors.get("subject_starter", ))
    personality_summary = _describe_personality(test_vectors)

    lines = [
        "Отчет по профессиональному профилю",
        "",
        f"Имя пользователя: user.nickname",
        "Краткий итог: На основе пройденного тестирования
сформирован профиль.",
        "",
        "Итоговый профиль",
        f"Тип по Климову: klimov_type.",
        f"Ведущая мотивационная шкала: motivation_scale.",
        personality_summary,
        "",
        "Рекомендации по профессиям",
    ]
    for profession, _score, _reasoning in items:
        lines.extend([profession.name, profession.description,
        ""])

    if run.honesty_warning:
        lines.extend(["", f"Примечание: run.honesty_warning"])
    return "\n".join(lines)
```

Функция собирает структурированный текстовый отчёт, включающий имя пользователя, итоговый профиль (доминирующий тип по Климову, ведущая мотивационная шкала, краткое описание личностных особенностей по результатам RIASEC, Big Five и Рокича) и список рекомендованных профессий с их описаниями. При наличии предупреждения о возможной недостоверности ответов оно выводится в конце отчёта.

Формирование отчёта в формате PDF

Для пользователей, которым необходим документ в формате PDF (например, для предоставления работодателю или сохранения в личном архиве), реализована функция конвертации текстового отчёта в PDF с использованием библиотеки reportlab:

```
def render_report_pdf(content: str, target: BytesIO) -> None:
    from reportlab.lib.pagesizes import A4
    from reportlab.pdfbase import pdfmetrics
    from reportlab.pdfbase.ttfonts import TTFont
    from reportlab.pdfgen import canvas

    font_path = _resolve_pdf_font_path()
    pdfmetrics.registerFont(TTFont("PsychodiagnosticsFont",
font_path))

    pdf = canvas.Canvas(target, pagesize=A4)
    text = pdf.beginText(48, A4[1] - 56)
    text.setFont("PsychodiagnosticsFont", 11)
    text.setLeading(16)

    for line in content.splitlines():
        text.textLine(line)
    pdf.drawText(text)
    pdf.save()
```

Функция создаёт PDF–документ формата A4, регистрирует системный шрифт с поддержкой кириллицы (Arial, Tahoma или Calibri в зависимости от их наличия в операционной системе) и построчно выводит содержимое текстового отчёта. При выходе за нижнюю границу страницы автоматически создаётся новая. Результат записывается в переданный поток BytesIO, что позволяет отправить файл пользователю напрямую через HTTP–ответ без сохранения на диск.

Модуль расчёта и формирования отчёта реализован в виде трёх взаимосвязанных файлов: `scoring.py` обеспечивает математическую основу алгоритма (расчёт векторов, агрегирование, косинусная близость), `llm.py` – формирование пояснительного текста, `reports.py` – сборку отчёта в формате PDF. Все функции имеют чёткую зону ответственности и последовательно вызываются из единой управляющей функции `generate_recommendation_for_user`, что обеспечивает прозрачность работы алгоритма и упрощает его сопровождение. Использование стандартных библиотек Python (`math`, `reportlab`) и отказ от подключения внешних сервисов гарантируют автономность модуля и независимость работы системы от внешних факторов.

1.5.5 Обеспечение безопасности персональных данных

Безопасность данных – это комплекс мер, направленных на защиту информации от несанкционированного доступа, изменения, уничтожения или раскрытия третьим лицам. Применительно к веб–приложениям безопасность подразумевает защиту трёх категорий данных: учётных данных пользователей (логин, пароль), персональных данных (имя, возраст, контакты) и пользовательской информации (результаты тестов, рекомендации). [21]

В разрабатываемой системе пользователи указывают свой адрес электронной почты, возраст, пол и проходят психодиагностические методики, результаты которых отражают их личностные особенности. Утечка такой информации может привести к нарушению неприкосновенности частной жизни, дискриминации при найме на работу или другим негативным последствиям. Кроме того, использование одного и того же пароля на нескольких сайтах является распространённой практикой, поэтому утечка паролей из одной системы создаёт риск компрометации учётных записей пользователей и на других ресурсах. Согласно Федеральному закону № 152–ФЗ «О персональных данных», оператор персональных данных обязан принимать организационные и технические меры для их защиты. [22]

В разрабатываемой системе реализованы следующие меры безопасности:

- хеширование паролей – пароли пользователей не хранятся в открытом виде, а преобразуются в криптографический хеш по алгоритму pbkdf2-sha256;
- аутентификация на основе JWT-токенов – каждый запрос пользователя сопровождается подписанным токеном с ограниченным сроком действия;
- ролевое разграничение доступа – административные функции доступны только пользователям с признаком is_admin;
- маскирование персональных данных в административной панели – адреса электронной почты пользователей отображаются администратору в сокращённом виде;
- мягкое удаление учётных записей – при удалении пользователя его данные не стираются физически, а помечаются полем deleted_at, что сохраняет историчность обезличенной статистики и позволяет восстановить запись при ошибочном удалении;
- валидация всех входных данных на стороне сервера средствами Pydantic, что исключает обработку некорректных или вредоносных запросов.

Ключевой механизм защиты – хеширование пароля при регистрации и его проверка при входе:

```
from passlib.context import CryptContext
pwd_context = CryptContext(schemes=["pbkdf2_sha256"],
deprecatеd="auto")
def hash_password(password: str) -> str:
    return pwd_context.hash(password)
def verify_password(password: str, password_hash: str) -> bool:
    return pwd_context.verify(password, password_hash)
```

При регистрации введённый пользователем пароль преобразуется в хеш функцией hash_password и сохраняется в базу данных. При следующем входе введённый пароль не сравнивается напрямую с сохранённым – вместо этого функция verify_password вычисляет хеш от введённого пароля и сравнивает его с сохранённым. Сам исходный пароль в системе нигде не хранится, поэтому даже в случае получения злоумышленником доступа к базе данных восстановить пароли пользователей невозможно.

SHA (Secure Hash Algorithm) – семейство криптографических хеш–функций, разработанных Агентством национальной безопасности США и являющихся международным стандартом обработки данных. Хеш–функция принимает на вход произвольный текст и возвращает строку фиксированной длины – так называемый «хеш» или «отпечаток».

Криптографические хеш–функции обладают тремя важными свойствами:

- односторонность – по хешу невозможно восстановить исходный текст;
- детерминированность – для одного и того же входа всегда получается один и тот же хеш;
- чувствительность к изменениям – даже минимальное изменение входного текста (например, замена одной буквы) приводит к полностью другому хешу.

Наиболее широко распространёнными версиями являются SHA–1 (устарела), SHA–256 и SHA–512 (актуальные). В разрабатываемой системе используется алгоритм PBKDF2–SHA256 – это специальный режим работы SHA–256, разработанный именно для безопасного хранения паролей. Особенность PBKDF2 заключается в том, что он применяет хеш–функцию многократно (в реализации passlib – десятки тысяч раз), что значительно усложняет подбор пароля методом перебора. Кроме того, PBKDF2 автоматически добавляет к паролю случайную «соль» – уникальную последовательность символов, исключающую возможность использования заранее подготовленных таблиц для расшифровки.

Применённый в разрабатываемой системе набор мер обеспечивает соответствие основным требованиям к защите персональных данных пользователей. Хеширование паролей по алгоритму PBKDF2–SHA256, аутентификация на основе JWT–токенов, и мягкое удаление учётных записей в совокупности существенно снижают риски, связанные с возможной утечкой информации, и соответствуют требованиям Федерального закона № 152–ФЗ «О персональных данных».

1.6 Разработка приложения

1.6.1 Описание интерфейса системы

Пользовательский интерфейс автоматизированной системы психодиагностики спроектирован с ориентацией на массового пользователя – школьников, студентов, взрослых, а также специалистов профконсультантов и кадровых служб. Основная идея, заложенная при разработке интерфейса, – простота и понятность: пользователь должен без специальной подготовки разобраться в работе системы и за минимальное число действий пройти полный цикл от регистрации до получения итогового отчёта.

Все страницы приложения выполнены в едином визуальном стиле с мягкой цветовой палитрой (преимущественно бирюзовые, серо–голубые и белые тона) и единым набором шрифтов. Такой подход снижает когнитивную нагрузку, делает интерфейс целостным и не отвлекает пользователя от содержательной работы – прохождения методик и анализа результатов.

Структура интерфейса построена по принципу «сайдбар + основная область». В левой части экрана на всех страницах после авторизации присутствует постоянная боковая навигация с пятью основными пунктами (Пользователь, Тестирование, Профбаза, Пользователи, Администрирование), причём последние два видны только администраторам. Активный раздел всегда подсвечен, что позволяет пользователю в любой момент понимать, в какой части системы он находится. В правой части отображается содержимое выбранного раздела, разбитое на смысловые карточки.

Информация на страницах расположена по принципу постепенного раскрытия: на первом уровне отображается обобщённая сводка (например, прогресс прохождения или краткие сведения профессии), а детали (полные результаты, полное описание профессии, эталонный профиль) доступны по дополнительному действию. Это позволяет одновременно сохранить простоту восприятия для начинающего пользователя и обеспечить доступ к подробной информации для тех, кому она требуется.

					ПГУ 1.090501.07.001 ПЗ	Лист
						110
Изм.	Лист	№ докум.	Подпись	Дата		

Все действия в системе сопровождаются прозрачной индикацией состояний: для каждой методики отображается статус её прохождения («не начата», «в процессе», «пройдена»), длительные операции сопровождаются индикатором загрузки, ошибки выводятся понятными русскоязычными сообщениями непосредственно у того поля, где они возникли. Формы ввода снабжены подсказками (требования к минимальной длине пароля, формат электронной почты, допустимый диапазон возраста), а проверка корректности выполняется одновременно на стороне клиента (мгновенный отклик) и на стороне сервера (защита от обхода).

На каждой странице выделена одна основная целевая кнопка, оформленная контрастным цветом (Войти, Зарегистрироваться, Открыть, Завершить тест, Скачать отчёт). Вторичные действия (Отмена, Выйти, переключение между вкладками) визуально менее заметны, что направляет внимание пользователя на основной сценарий и снижает риск ошибочных действий.

Интерфейс полностью русифицирован: все элементы управления, тексты вопросов методик, описания профессий и системные сообщения выполнены на русском языке, что соответствует целевой аудитории – пользователям из Российской Федерации.

Реализована адаптивная вёрстка: интерфейс корректно отображается на устройствах с различным разрешением экрана – от персональных компьютеров с большими мониторами до планшетов и смартфонов. Использование стандартных средств CSS (медиа-запросы, флексбокс-разметка, относительные единицы измерения) обеспечивает автоматическое масштабирование без необходимости поддержки отдельной мобильной версии приложения.

Доступ к разным разделам системы организован по принципу ролевой видимости: обычный пользователь не видит административных разделов в сайдбаре, что исключает путаницу и снижает риск случайных действий. Администратор после входа автоматически получает дополнительные пункты навигации и доступ к расширенному набору функций – статистике системы, списку пользователей, выгрузке отчётов и редактированию каталога профессий.

1.6.2 Развёртывание и запуск системы

Для развёртывания и запуска автоматизированной системы психодиагностики необходимо подготовить программное окружение, установить зависимости, настроить подключение к базе данных и поочерёдно запустить серверную и клиентскую части. Все команды выполняются из корневой папки проекта.

Требования к программному окружению

На машине, на которой выполняется развёртывание системы, должны быть установлены следующие средства:

- Python 3.11 или новее – среда исполнения серверной части;
- Node.js 20 LTS или новее – среда исполнения клиентской части и менеджер пакетов npm;
- MySQL Server 8.x – система управления базой данных;
- MySQL Workbench 8.0 – графический клиент для администрирования базы;
- Visual Studio Code – основная среда разработки и работы с проектом;
- Современный веб-браузер (Google Chrome, Mozilla Firefox, Microsoft Edge, Yandex Browser) – для работы с пользовательским интерфейсом.

После установки и запуска MySQL Server необходимо убедиться, что соответствующая служба Windows (MySQL80) запущена и установлена на автоматический запуск.

Исходный код проекта открывается в Visual Studio Code с указанием корневой папки проекта. После открытия в левой панели отображается дерево каталогов проекта: backend/, frontend/, database/, scripts/, docs/, а также конфигурационные файлы package.json, .env и другие.

На рисунке 24 представлена директория рабочего проекта.

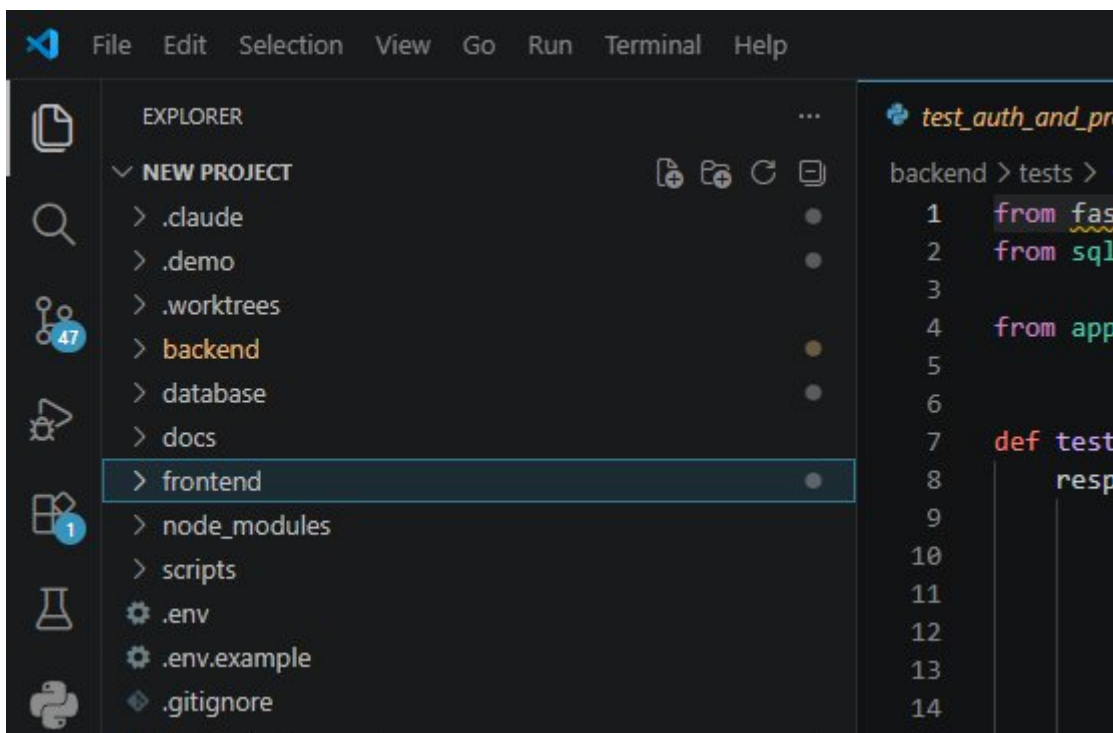


Рисунок 24 – Открытый проект в среде разработки Visual Studio Code

Установка библиотек выполняется однократно при первом развёртывании системы. Команды выполняются во встроенном терминале Visual Studio Code (открывается сочетанием клавиш Ctrl + `).

Для клиентской части:

```
npm install
```

Для серверной части:

```
py -m pip install --upgrade pip setuptools wheel
py -m pip install fastapi==0.115.6 uvicorn==0.34.0
sqlalchemy==2.0.36 ^
alembic==1.14.0 pydantic==2.10.4
pydantic-settings==2.7.0 ^
email-validator==2.2.0 passlib==1.7.4
PyJWT==2.10.1 ^
pymysql==1.1.1 pytest pytest-cov

httpx==0.28.1 reportlab
```

В корневой папке проекта создаётся файл .env со строкой подключения к серверу MySQL:

```
DATABASE_URL=mysql+pymysql://root:Пароль@127.0.0.1:3306/psychodiagnostics?charset=utf8mb4
JWT_SECRET=diploma-secret-change-in-production
ADMIN_EMAIL=admin@example.com
ADMIN_PASSWORD=admin12345
NEXT_PUBLIC_API_BASE_URL=http://127.0.0.1:8000
```

Перед первым запуском системы в MySQL Workbench создаётся пустая схема (база данных) с именем psychodiagnostics и кодировкой utf8mb4. Создание таблиц выполняется автоматически серверной частью при первом запуске благодаря использованию ORM SQLAlchemy.

В первом окне терминала выполняется команда:

```
npm run backend:dev
```

Команда запускает веб-сервер Uvicorn с автоматической перезагрузкой кода при изменениях. После успешного запуска в окне терминала отображаются служебные сообщения: создание таблиц базы данных при первом запуске, загрузка демо-данных, готовность сервера принимать запросы по адресу <http://127.0.0.1:8000>.

После запуска серверной части в MySQL Workbench при обновлении схемы отображаются все 12 автоматически созданных таблиц.

Для проверки работоспособности достаточно открыть в браузере адрес <http://127.0.0.1:8000/health> – должен вернуться ответ "status":"ok", результат представлен на рисунке 25. Дополнительно по адресу <http://127.0.0.1:8000/docs> доступна автоматически сгенерированная документация REST API в формате Swagger UI представлен на рисунке 26.

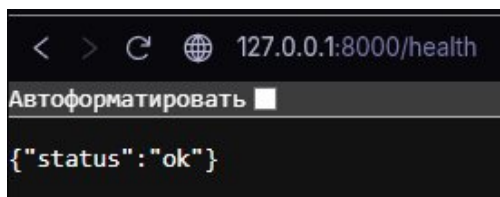


Рисунок 25 – Проверка работоспособности

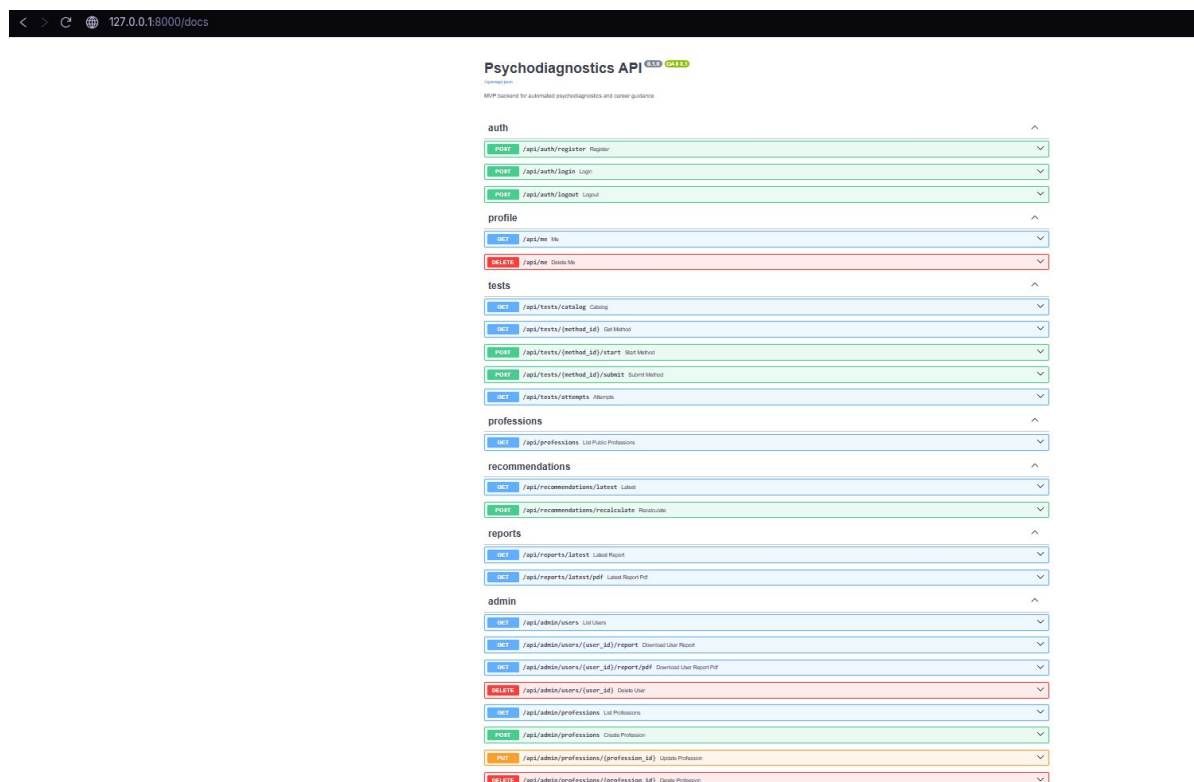


Рисунок 26 – Автоматически сгенерированная документация Swagger UI

Во втором окне терминала выполняется команда:

```
npm run frontend:dev
```

Команда запускает сервер разработки Next.js. После успешного запуска в окне терминала отображается готовность приложения к работе по адресу <http://127.0.0.1:3000>.

После запуска клиентской части можно открывать в браузере главную страницу системы.

Рабочий вход в систему тестирования и аналитической интерпретации представлен на рисунке 27.

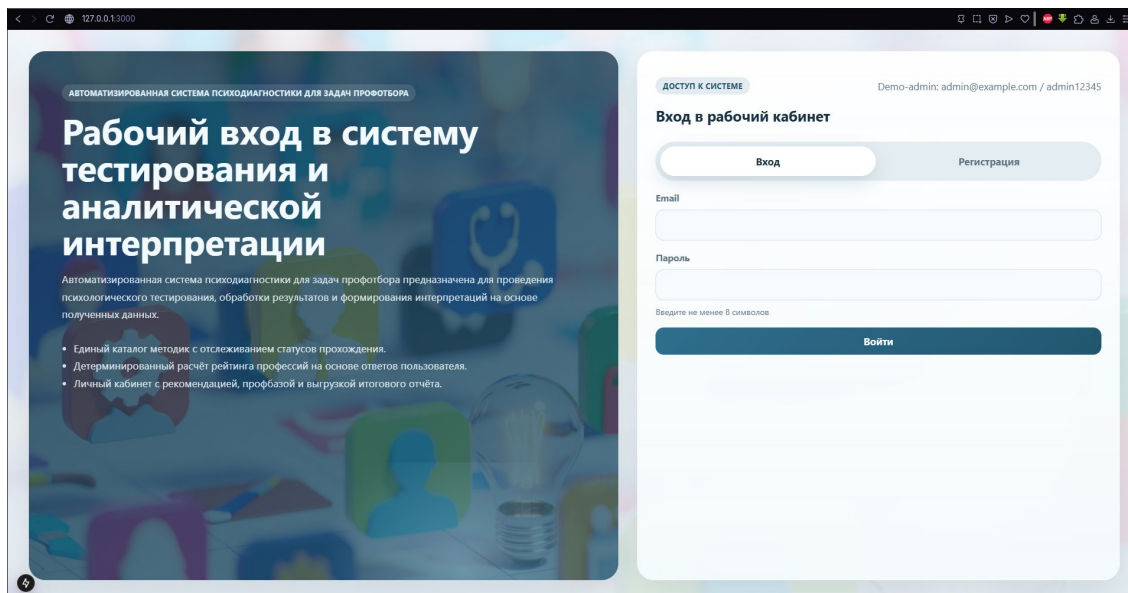


Рисунок 27 – Главная страница системы, открытая в браузере

Проверка работоспособности

После выполнения всех шагов развёртывания система готова к использованию. Основные адреса для проверки:

Главная страница системы – <http://127.0.0.1:3000>

Проверка работоспособности серверной части – <http://127.0.0.1:8000/health>

Документация REST API (Swagger UI) – <http://127.0.0.1:8000/docs>

Для проверки административных функций при первом запуске системы автоматически создаётся учётная запись администратора со следующими параметрами:

Адрес электронной почты – admin@example.com

Пароль – admin12345

В рабочей эксплуатации пароль администратора должен быть изменён через настройки в файле .env перед первым запуском системы.

Для проверки корректности работы программного кода предусмотрены автоматические тесты обеих частей системы. Запуск тестов выполняется командами:

```
npm run backend:test
npm test
```

Первая команда запускает тесты серверной части (фреймворк pytest), вторая – тесты клиентской части (фреймворк Vitest и библиотека React Testing Library).

Для перевода клиентской части в режим рабочей эксплуатации (с оптимизированной сборкой) выполняется команда:

```
npm run build
```

После успешной сборки оптимизированная версия запускается командой

```
npm run start
```

И работает в режиме рабочего веб-сервера.

1.6.3 Руководство пользователя

Настоящее руководство описывает полный сценарий работы пользователя с автоматизированной системой психодиагностики – от регистрации новой учётной записи до получения итогового отчёта с рекомендацией.

Для начала работы с системой пользователь открывает главную страницу и переключается на вкладку «Регистрация» в правой части экрана.

В появившейся форме необходимо заполнить пять полей: адрес электронной почты, никнейм, пароль длиной не менее 8 символов, пол и возраст в полных годах.

После заполнения всех полей следует нажать кнопку «Зарегистрироваться». При успешной регистрации пользователь автоматически перенаправляется в личный кабинет.

При попытке регистрации с уже занятым адресом электронной почты или никнеймом, а также при некорректном заполнении полей под формой отображается соответствующее сообщение об ошибке.

На рисунке 28 представлена одна из заполненных форм регистрации для дальнейшего тестирования программы и откладки, такие как Email, Имя, Пароль, Пол, Полных лет.

АВТОМАТИЗИРОВАННАЯ СИСТЕМА ПСИХОДИАГНОСТИКИ ДЛЯ ЗАДАЧ ПРОФОТБОРА

Рабочий вход в систему тестирования и аналитической интерпретации

Автоматизированная система психодиагностики для задач профотбора предназначена для проведения психологического тестирования, обработки результатов и формирования интерпретаций на основе полученных данных.

- Единый каталог методик с отслеживанием статусов прохождения.
- Детерминированный расчёт рейтинга профессий на основе ответов пользователя.
- Личный кабинет с рекомендацией, профбазой и выгрузкой итогового отчёта.

ДОСТУП К СИСТЕМЕ Demo-admin: admin@example.com / admin12345

Регистрация пользователя

Вход Регистрация

Email
diplom@example.com

Ник
diplom0000

Пароль

Введите не менее 8 символов

Пол
Мужской

Полных лет
23

Создать аккаунт

Рисунок 28 – Форма регистрации нового пользователя

Если у пользователя уже есть учётная запись, на главной странице на вкладке «Вход» необходимо ввести адрес электронной почты и пароль, после чего нажать кнопку «Войти».

После успешной авторизации система автоматически открывает личный кабинет.

При неверно введённых учётных данных под формой отображается соответствующее сообщение, и пользователь может повторить попытку.

Пробный вход в рабочий кабинет под созданным

Email: diplom@example.com представлена на рисунке 29.

ДОСТУП К СИСТЕМЕ

Demo-admin: admin@example.com / admin12345

Вход в рабочий кабинет

Вход

Регистрация

Email

diplom@example.com

Пароль

diplom@example.com

.....

Управление паролями

Введите не менее 8 символов

Войти

Рисунок 29 – Форма входа в систему

После входа открывается личный кабинет пользователя. В его центральной части отображаются анкетные данные (никнейм, адрес электронной почты, пол, возраст), карточка общего прогресса прохождения методик в виде дроби «N/5» и процентного индикатора, а также кнопка удаления учётной записи. В левой части экрана расположен постоянный сайдбар с пунктами навигации «Пользователь», «Тестирование» и «Профбаза». В правом верхнем углу присутствует кнопка выхода из системы.

Личный кабинет пользователя представлен на рисунке 30.

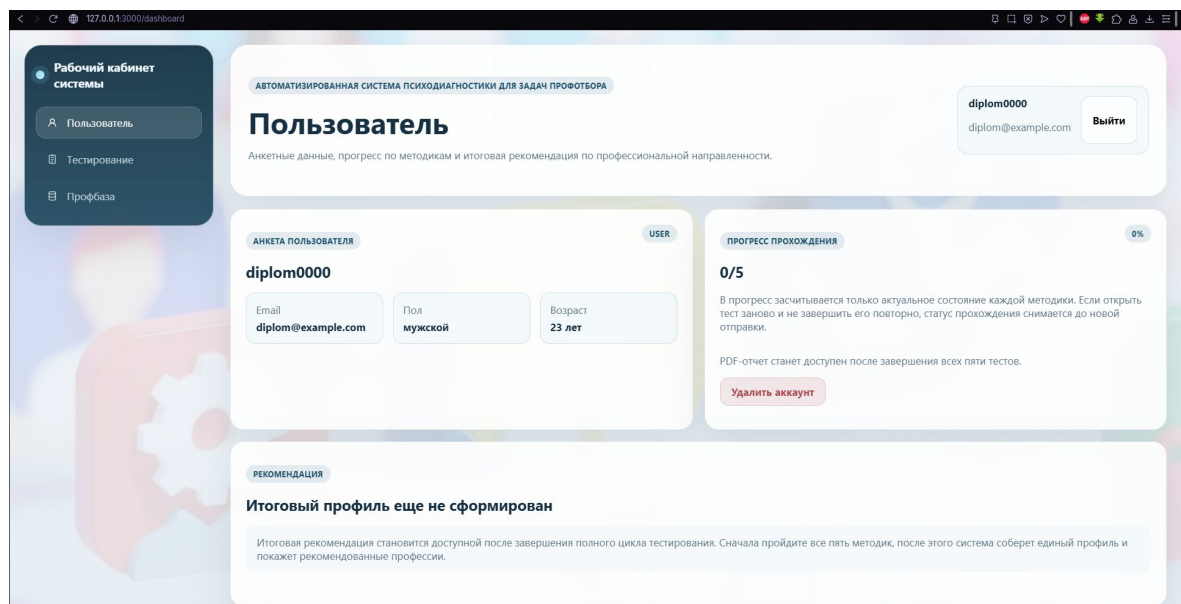


Рисунок 30 – Личный кабинет пользователя

Для прохождения методик пользователь нажимает в сайдбаре пункт «Тестирование» и открывает каталог из пяти доступных психодиагностических методик: «Школьные ориентиры», «ДДО Климова», «Holland RIASEC», «Big Five» и «Ценностные ориентации Рокича». У каждой методики отображается краткое описание и индикатор статуса прохождения – «не начата», «в процессе» или «пройдена». Представлено на рисунках 31 и 32.

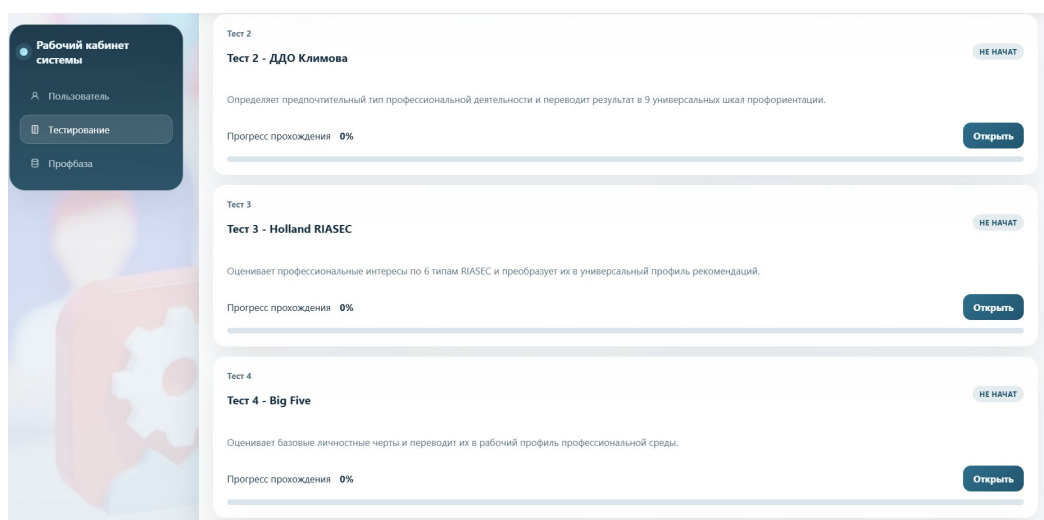


Рисунок 31 – Каталог психодиагностических методик

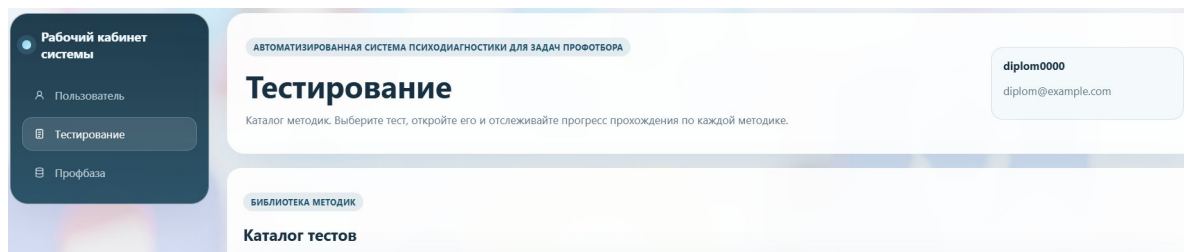


Рисунок 32 – Каталог психодиагностических методик

Для начала прохождения выбранной методики пользователь нажимает кнопку «Открыть» в её карточке. На странице методики последовательно отображаются вопросы с вариантами ответов. Формат ответа зависит от типа методики: бинарный выбор А или В для ДДО Климова, пятибалльная шкала согласия для Big Five, четырёхбалльная шкала для RIASEC, ранжирование от 1 до 18 для Рокича, тройка оценок (текущая оценка, самооценка и мотивация по каждому предмету) для стартового теста «Школьные ориентиры». По завершении ответов пользователь нажимает кнопку «Завершить тест», после чего ответы отправляются на сервер для обработки, а статус методики в каталоге меняется на «пройдена». Представлено на рисунках 33 и 34.

Рисунок 33 – Страница прохождения методики

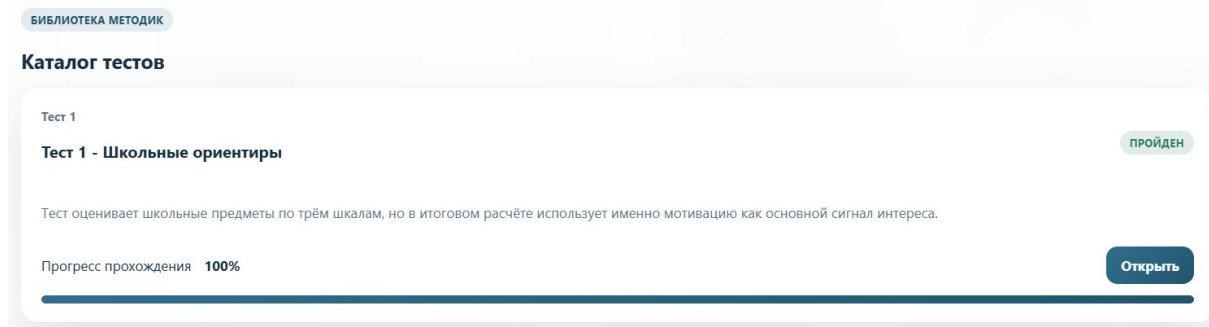


Рисунок 34 – Страница прохождения методики

Для просмотра каталога профессий пользователь нажимает в сайдбаре пункт «Профбаза», представлено на рисунке 35. В левой части экрана отображается алфавитный фильтр и поле поиска по названию, в правой – подробная карточка выбранной профессии с описанием, преимуществами, недостатками, направлениями подготовки, типичными затруднениями и оценкой популярности на российском рынке труда.

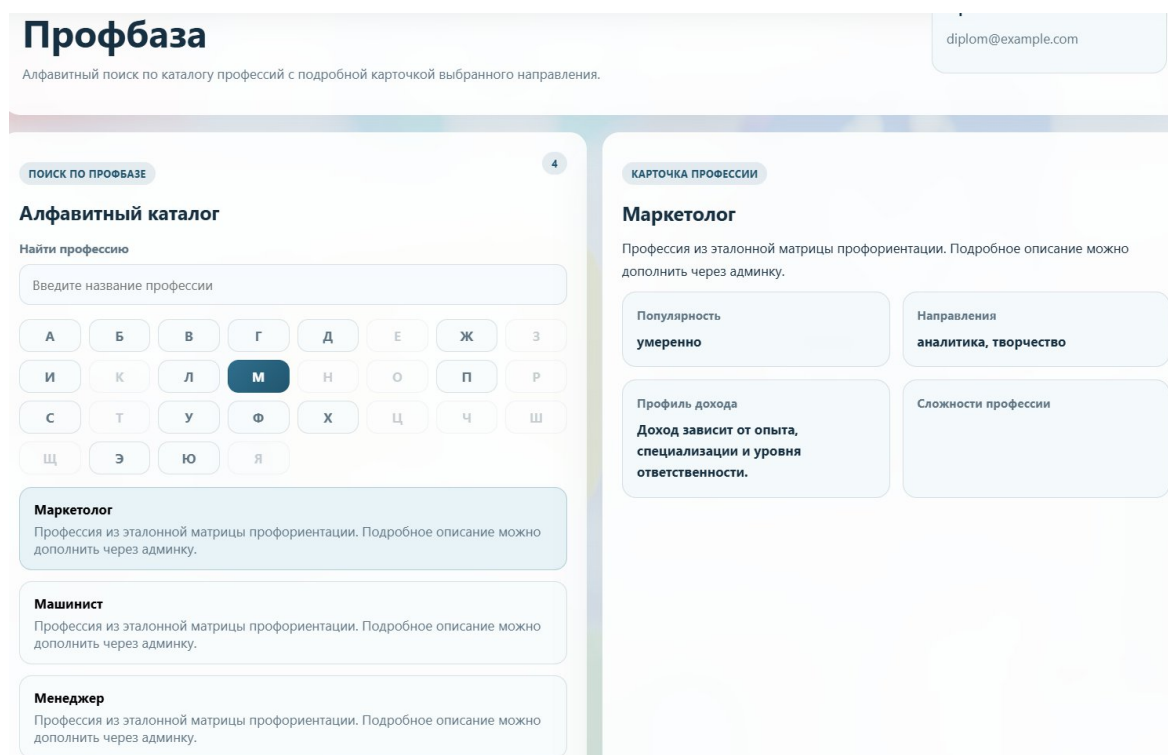


Рисунок 35 – Каталог профессий с подробной карточкой

После того как пользователь прошёл хотя бы одну методику, на странице рекомендации отображается рейтинг из пяти наиболее подходящих профессий, упорядоченных по убыванию степени соответствия профилю пользователя. По каждой профессии указывается процент совпадения и краткое пояснение с указанием трёх ключевых черт, по которым наблюдается наибольшее совпадение. Под рейтингом размещён обобщающий текстовый комментарий, формируемый автоматически по результатам всех пройденных методик. Представлено на рисунках 36 и 37.

Тест 1

Тест 1 - Школьные ориентиры пройден

Тест оценивает школьные предметы по трём шкалам, но в итоговом расчёте использует именно мотивацию как основной сигнал интереса.

Прогресс прохождения **100%** Открыть

Тест 2

Тест 2 - ДДО Климова пройден

Определяет предпочтительный тип профессиональной деятельности и переводит результат в 9 универсальных шкал профориентации.

Прогресс прохождения **100%** Открыть

Тест 3

Тест 3 - Holland RIASEC пройден

Оценивает профессиональные интересы по 6 типам RIASEC и преобразует их в универсальный профиль рекомендаций.

Прогресс прохождения **100%** Открыть

Тест 4

Тест 4 - Big Five пройден

Оценивает базовые личностные черты и переводит их в рабочий профиль профессиональной среды.

Прогресс прохождения **100%** Открыть

Рисунок 36 – Пройденные методики

ИТОГОВАЯ РЕКОМЕНДАЦИЯ

27.05.2026

Профессиональный профиль

На основе завершеного тестирования сформирован общий профессиональный профиль пользователя. Ниже приведены направления деятельности, которые сейчас наиболее близки по интересам и личностным особенностям.

Рекомендация сформирована на основе результатов пяти методик и сопоставления с профилями профессий. Наиболее близкими направлениями для текущего профиля являются: Менеджер, Маркетолог, Логист.

РЕКОМЕНДАЦИЯ 1

Менеджер

Профессия из эталонной матрицы профориентации. Подробное описание можно дополнить через админку.

Популярность
умеренно

Направления
управление, люди

Профиль дохода
Доход зависит от опыта, специализации и уровня ответственности.

Сложности профессии

РЕКОМЕНДАЦИЯ 2

Маркетолог

Профессия из эталонной матрицы профориентации. Подробное описание можно дополнить через админку.

Популярность
умеренно

Направления
аналитика, творчество

Профиль дохода
Доход зависит от опыта, специализации и уровня ответственности.

Сложности профессии

Рисунок 37 – Итоговая рекомендация с рейтингом подходящих профессий

Для получения итогового отчёта пользователь открывает страницу «Отчёт», на которой отображается полный текст с результатами тестирования и рекомендациями. На странице доступна кнопка загрузки документа: «Скачать отчет PDF». При нажатии на любую из них браузер инициирует загрузку файла с отчётом, который пользователь может сохранить, распечатать или передать работодателю. Представлено на рисунках 38 и 39.

АВТОМАТИЗИРОВАННАЯ СИСТЕМА ПСИХОДИАГНОСТИКИ ДЛЯ ЗАДАЧ ПРОФОБОРА

diplom0000

diplom@example.com

Выйти

Пользователь

Анкетные данные, прогресс по методикам и итоговая рекомендация по профессиональной направленности.

АНКЕТА ПОЛЬЗОВАТЕЛЯ

USER

ПРОГРЕСС ПРОХОЖДЕНИЯ

100%

diplom0000

Email
diplom@example.com

Пол
мужской

Возраст
23 лет

5/5

В прогресс засчитывается только актуальное состояние каждой методики. Если открыть тест заново и не завершить его повторно, статус прохождения снимается до новой отправки.

Скачать отчет PDF

Удалить аккаунт

Рисунок 38 – Страница итогового отчёта с кнопками скачивания

Отчет по профессиональному профилю

Имя пользователя: diplom0000

Краткий итог: На основе пройденного тестирования сформирован следующий профессиональный профил

Итоговый профиль

Тип по Климову: человек-человек.

Ведущая мотивационная шкала: Природа.

В профессиональных интересах заметна ориентация на направления: аналитика, структура. В работе наиболее комфортна среда, где ценятся люди, структура. В выборе деятельности важны структура, аналитика.

Рекомендации по профессиям

Менеджер

Рисунок 39 – Итоговый отчёт

При необходимости пользователь может пройти любую методику повторно – для этого в каталоге методик нужно снова нажать «Открыть» у выбранной методики. После повторного прохождения статус методики обновляется, а итоговая рекомендация автоматически пересчитывается с учётом последних ответов. Это позволяет отслеживать динамику изменений профессионального профиля во времени, поскольку психологические особенности и интересы человека могут меняться.

В случае подозрения системы на недостоверность ответов (значительное расхождение между объективной оценкой и самооценкой в стартовом тесте либо аномально низкий уровень мотивации) в личном кабинете и на странице рекомендации отображается соответствующее предупреждение с предложением пройти тестирование повторно.

Для завершения работы с системой пользователь нажимает кнопку «Выйти» в правом верхнем углу личного кабинета. Сессия завершается, токен авторизации удаляется из локального хранилища браузера, и пользователь возвращается на главную страницу. При необходимости в систему можно войти повторно в любой момент.

					ПГУ 1.090501.07.001 ПЗ	Лист
						125
Изм.	Лист	№ докум.	Подпись	Дата		

1.6.4 Руководство администратора

Настоящее руководство описывает работу пользователя с правами администратора. Администратор обладает всеми возможностями обычного пользователя (регистрация, прохождение методик, получение рекомендаций, скачивание отчётов), а также получает доступ к дополнительным разделам системы – просмотру статистики, управлению пользователями, выгрузке отчётов других пользователей и редактированию каталога профессий.

Для входа под учётной записью администратора на главной странице системы используется демонстрационная учётная запись с адресом электронной почты `admin@example.com` и паролем `admin12345`. В рабочей эксплуатации параметры администраторской учётной записи задаются в файле `.env` перед первым запуском системы. После успешной авторизации пользователь автоматически перенаправляется в личный кабинет, при этом в сайдбаре отображаются дополнительные пункты «Пользователи» и «Администрирование», недоступные обычным пользователям. Представлено на рисунке 40.

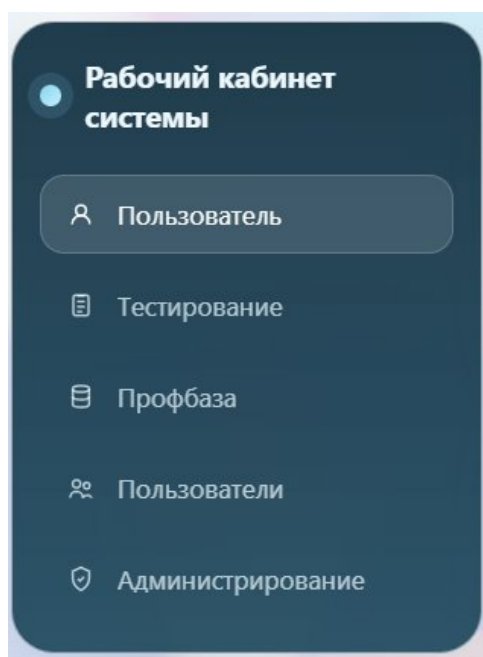


Рисунок 40 – Сайдбар с дополнительными пунктами навигации для администратора

Для просмотра обезличенной статистики использования системы администратор нажимает в сайдбаре пункт «Администрирование». В верхней части экрана отображаются ключевые показатели: общее количество зарегистрированных пользователей, число начатых попыток прохождения методик, число завершённых попыток и средний процент завершённости. Эти данные не содержат персональной информации о пользователях и предназначены для оценки активности использования системы. Представлено на рисунке 41.

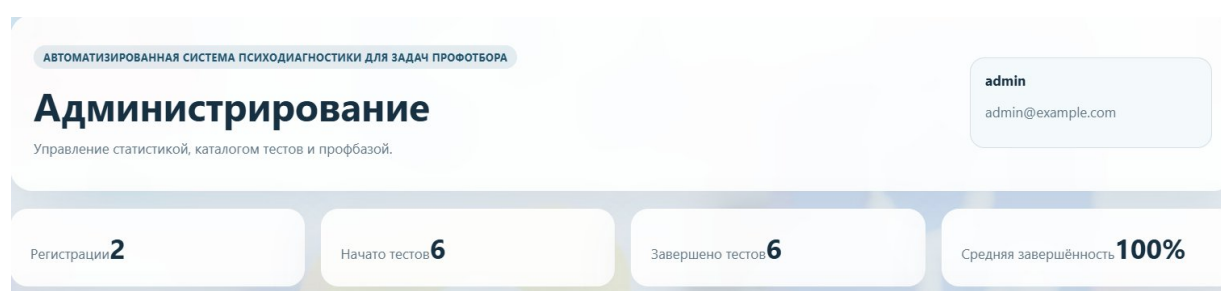


Рисунок 41 – Панель статистики системы

Для работы со списком зарегистрированных пользователей администратор переходит в раздел «Пользователи».

В нём отображается список учётных записей с указанием никнейма, пола, возраста и маскированного адреса электронной почты (например, adm***@e*****.com) – маскирование выполняется автоматически на стороне сервера в целях защиты персональных данных.

Учётные записи администраторов помечены соответствующим значком и надписью.

Данный раздел представлен на рисунке 42.

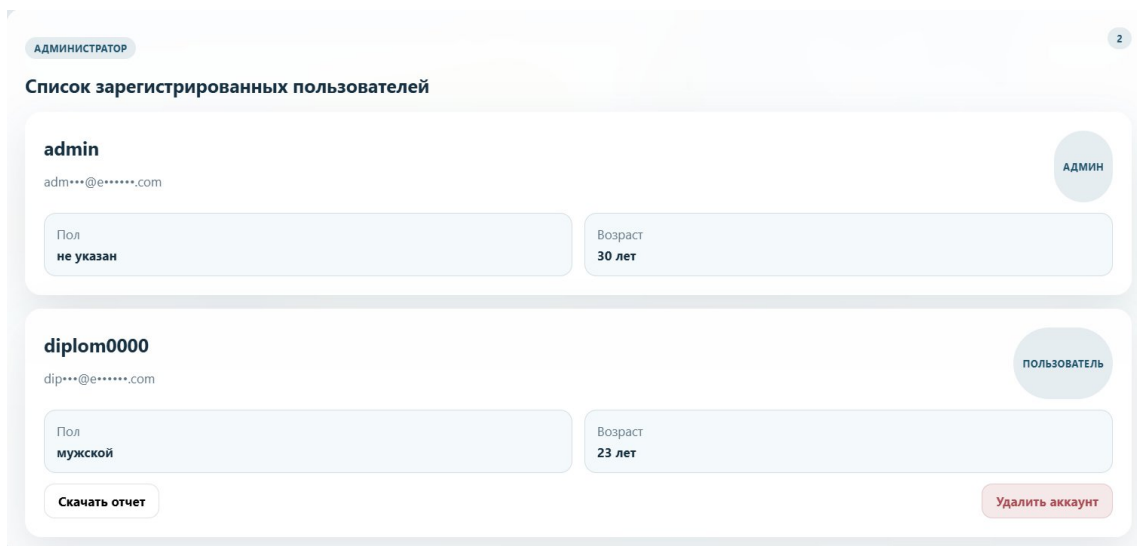


Рисунок 42 – Список зарегистрированных пользователей с маскированием адресов электронной почты

Для просмотра подробной информации о конкретном пользователе администратор раскрывает его карточку в списке. В карточке отображаются анкетные данные, дата регистрации, список пройденных методик и кнопки загрузки итогового отчёта в формате PDF. Это позволяет администратору сформировать документ по результатам тестирования любого пользователя – например, для последующего анализа профконсультантом или для предоставления отчётности.

При необходимости удалить учётную запись пользователя администратор нажимает кнопку «Удалить аккаунт» в его карточке. Удаление выполняется по принципу мягкого удаления: запись пользователя помечается полем deleted, после чего она исключается из всех пользовательских сценариев, однако сохраняется в обезличенной форме для целостности статистики системы.

В разделе «Администрирование» под панелью статистики расположен блок «Системные методики», в котором отображается перечень всех пяти методик, доступных пользователям. В демонстрационной версии возможность редактирования методик отключена для сохранения целостности данных, представлено на рисунке 43.

Системные методики

В текущей демо-версии каталог тестов зафиксирован. Просмотр состава методик доступен, но изменение, добавление и удаление через админку отключены, чтобы не нарушить рабочую структуру базы данных.

Тест 1 - Школьные ориентиры

Тест оценивает школьные предметы по трём шкалам, но в итоговом расчёте использует именно мотивацию как основной сигнал интереса.

Тип: subject_starter

Вопросов: 18

Тест 2 - ДДО Климова

Определяет предпочтительный тип профессиональной деятельности и переводит результат в 9 универсальных шкал профориентации.

Тип: klimov_ddo

Вопросов: 20

Тест 3 - Holland RIASEC

Оценивает профессиональные интересы по 6 типам RIASEC и преобразует их в универсальный профиль рекомендаций.

Тип: holland_riasec

Рисунок 43 – Блок системных методик в панели администрирования

Для управления каталогом профессий в разделе «Администрирование» предусмотрен блок «Редактирование профессий». В нём отображается полный список из 37 профессий, доступных пользователям, и форма редактирования выбранной профессии. Администратор может изменить название профессии, описание, перечень преимуществ и недостатков, направления подготовки, типичные затруднения и оценку популярности на российском рынке труда.

Для сохранения внесённых изменений необходимо нажать кнопку «Сохранить». Кроме того, доступны операции добавления новой профессии в каталог и удаления устаревшей. Представлено на рисунке 44.

ПРОФБАЗА

Редактирование профессий

HR-менеджер
 Профессия из эталонной матрицы профориентации. Подробное описание можно дополнить через админку.

Актер
 Профессия из эталонной матрицы профориентации. Подробное описание можно дополнить через админку.

Архитектор
 Профессия из эталонной матрицы профориентации. Подробное описание можно дополнить через админку.

Биолог
 Профессия из эталонной матрицы профориентации. Подробное описание можно дополнить через админку.

Бухгалтер
 Профессия из эталонной матрицы профориентации. Подробное описание можно дополнить через админку.

Ветеринар
 Профессия из эталонной матрицы профориентации. Подробное описание можно дополнить через админку.

Водитель
 Профессия из эталонной матрицы

Название профессии
 HR-менеджер

Описание
 Профессия из эталонной матрицы профориентации. Подробное описание можно дополнить через админку.

Профиль дохода
 Доход зависит от опыта, специализации и уровня ответственности.

Популярность в России
 умеренно

Направления
 люди
 структура

Сложности

Сохранить изменения

Новая карточка

Удалить

Рисунок 44 – Форма редактирования профессии в каталоге

Для завершения работы с системой администратор нажимает кнопку «Выйти» в правом верхнем углу личного кабинета. Сессия завершается, токен авторизации удаляется из локального хранилища браузера, и пользователь возвращается на главную страницу. При повторном входе под учётной записью обычного пользователя административные разделы скрываются – это обеспечивается за счёт проверки признака администратора в каждом запросе к серверу.

1.6.5 Тестирование системы

Для проверки корректности работы разработанной автоматизированной системы психодиагностики применены два взаимодополняющих подхода: автоматическое тестирование программного кода и ручное тестирование пользовательских сценариев с проверкой обработки ошибок.

Для проверки серверной части системы реализованы автоматические тесты на фреймворке `pytest`, размещённые в каталоге `backend/tests/`. Семь файлов тестов покрывают ключевые сценарии: авторизацию и регистрацию пользователей, работу с личным кабинетом, прохождение методик и расчёт рекомендаций, отображение прогресса по методикам, каталог профессий, административные функции и сбор статистики, корректность ранжирования по методике Рокича. Запуск тестов выполняется командой `npm run backend:test`.

Для проверки клиентской части используется фреймворк `Vitest` в сочетании с библиотекой `React Testing Library`. Семь файлов тестов, размещённых в каталоге `frontend/tests/`, проверяют работу основных компонентов интерфейса: боковой навигации, форм входа и регистрации, каталога методик, компонента прохождения теста, компонента ранжирования Рокича, отображения рекомендаций и каталога профессий. Запуск тестов выполняется командой `npm test`.

В ходе разработки все автоматические тесты регулярно прогонялись после внесения изменений в код. Результаты ручного тестирования по указанным сценариям представлены в таблице 16.

					ПГУ 1.090501.07.001 ПЗ	Лист
						131
Изм.	Лист	№ докум.	Подпись	Дата		

Таблица 16– Тест–кейсы ручного тестирования наиболее распространённых пользовательских ошибок

№	Наименование	Описание	Шаги для воспроизведения	Ожидаемый результат
1	Регистрация с пустыми обязательными полями	При регистрации в системе пользователь должен заполнить все поля формы. По статистике, около трети всех ошибок при работе с веб–формами связано именно с пропуском обязательных полей	1. Открыть главную страницу. 2. Перейти на вкладку «Регистрация». 3. Оставить любое из обязательных полей пустым. 4. Нажать кнопку «Зарегистрироваться»	Под незаполненным полем отображается сообщение об обязательности его заполнения; форма не отправляется на сервер, учётная запись не создаётся
2	Вход с неверным паролем	Для входа в систему требуются корректные учётные данные. Ошибка ввода пароля является самой распространённой проблемой при авторизации – пользователи забывают пароль, используют пароль от другого сервиса или совершают опечатки при наборе	1. Открыть главную страницу. 2. Перейти на вкладку «Вход». 3. Ввести существующий адрес электронной почты. 4. Ввести неверный пароль. 5. Нажать кнопку «Войти»	На экране отображается сообщение «Invalid credentials» («Неверные учётные данные»); пользователь остаётся на главной странице и может повторить ввод
3	Регистрация с уже занятым адресом электронной почты	Адрес электронной почты должен быть уникальным в системе. Часто пользователи забывают, что уже регистрировались ранее, и пытаются создать новую учётную запись с тем же адресом	1. Открыть форму регистрации. 2. Ввести адрес электронной почты уже существующего пользователя. 3. Корректно заполнить остальные поля. 4. Нажать кнопку «Зарегистрироваться»	На экране отображается сообщение «Email already registered» («Адрес электронной почты уже зарегистрирован»); пользователю предлагается перейти к форме входа

В дополнение к автоматическим тестам проведено ручное тестирование пользовательских сценариев с акцентом на проверку обработки исключительных ситуаций. Анализ работы систем подобного класса показывает, что наиболее частыми причинами обращений пользователей в техническую поддержку являются три типовые ситуации: попытка отправки формы с незаполненными обязательными полями, ошибка при вводе пароля во время авторизации и попытка повторной регистрации с уже использованным адресом электронной почты. Именно эти три сценария являются наиболее распространёнными среди всех пользовательских ошибок, поэтому корректность их обработки была проверена в первую очередь.

В ходе автоматического и ручного тестирования все запланированные сценарии успешно отработаны: положительные сценарии (корректный ввод, корректная авторизация, корректное прохождение методик) выполняются в полном объёме, а наиболее распространённые исключительные ситуации корректно обрабатываются системой с выдачей пользователю понятных русскоязычных сообщений. Это подтверждает работоспособность разработанной автоматизированной системы психодиагностики и её готовность к опытной эксплуатации.

2 Охрана труда

Современное законодательство в области охраны труда при работе с электронно–вычислительными машинами (ЭВМ) выдвигает следующие требования:

Помещение, в котором осуществляется работа с ПК, должно соответствовать следующим требованиям: [23]

- освещение должно соответствовать стандартам, а в случае отсутствия естественного освещения должны быть предоставлены расчеты, доказывающие безопасность работы без него;
- оконные проемы должны иметь регулируемые устройства, такие как жалюзи или занавеси;
- компьютеры не должны размещаться в местах, где есть риск попадания влаги.
- рабочие места с компьютерами должны быть оборудованы защитным заземлением;
- в помещениях с компьютерами следует проводить ежедневную влажную уборку и проветривание каждый час;
- рекомендуемая температура воздуха в помещении с компьютерами составляет $+19^{\circ}\text{C} - +21^{\circ}\text{C}$;
- рабочий стол должен быть спроектирован таким образом, чтобы обеспечить оптимальное расположение используемого оборудования в соответствии с выполняемой работой, разрешается использование различных типов рабочих столов, соответствующих современным стандартам эргономики;
- в помещениях с компьютерами следует проводить ежедневную влажную уборку и проветривание каждый час. [24]

Рабочие места с персональными компьютерами должны быть организованы следующим образом:

– при организации рабочих мест с компьютерами необходимо учитывать, что расстояние между рабочими столами, на которых установлены видеомониторы, должно быть не менее 2,0 метров в направлении между ними и не менее 1,2 метра между боковыми поверхностями видеомониторов;

– размещение компьютеров должно предусматривать расстояние не менее 1 метра от источников тепла. Следует избегать прямого попадания солнечного света на них;

– запрещается размещать рабочие места с компьютерами рядом с силовыми кабелями, высоковольтными трансформаторами и другим оборудованием, способным создавать помехи в работе компьютеров;

– при установке компьютеров следует избегать прохождения интерфейсных и электропитания кабелей по полу. В случае необходимости, их следует закрыть защитными коробками;

– экран видеомонитора должен находиться от глаз пользователя на расстоянии 600–700 мм, но не ближе 500 мм с учетом размеров алфавитно–цифровых знаков и символов;

– системный блок и монитор следует разместить так, чтобы расстояние от вентиляционных отверстий этих устройств до стены (или стенки мебели) составляло не менее 10 сантиметров;

– искусственное освещение в помещениях для эксплуатации компьютеров должно осуществляться системой общего равномерного освещения. Освещенность на поверхности рабочего стола в зоне размещения ПК должна быть 300–500 люкс. Освещение не должно создавать бликов на поверхности экрана. Освещенность поверхности экрана не должна быть более 300 люкс.

В случае работы на компьютере в течение 8–часовой смены следует предусматривать перерывы в соответствии с категорией работ:

					ПГУ 1.090501.07.001 ПЗ	Лист
						135
Изм.	Лист	№ докум.	Подпись	Дата		

– для работников I категории перерывы устанавливаются через 2 часа после начала смены и через 2 часа после обеденного перерыва продолжительностью 15 минут каждый;

– работники II категории должны иметь перерыв через 2 часа после начала смены и через 1,5 – 2,0 часа после обеденного перерыва продолжительностью 15 минут каждый или продолжительностью 10 минут через каждый час работы;

– для работников III категории рекомендуется устанавливать перерыв через 1,5 – 2,0 часа после начала смены и также через 1,5 – 2,0 часа после обеденного перерыва продолжительностью 20 минут каждый или продолжительностью 15 минут через каждый час работы.

При 12-часовой смене регламентированные перерывы должны быть предусмотрены в первые 8 часов работы так же, как и при 8-часовой смене. В течение последних 4 часов работы, независимо от категории и вида работ, необходимо предусматривать перерыв каждый час продолжительностью 15 минут. [25]

3 Экономика

Разработка автоматизированной системы психодиагностики, выполненная в рамках выпускной квалификационной работы, представляет собой полноценный программный продукт уровня коммерческого проекта. В настоящем разделе приводится оценка стоимости такой разработки в случае её реализации силами профессиональной команды разработчиков на российском рынке труда. Подобный расчёт позволяет соотнести объём фактически проделанной работы с её рыночной стоимостью.

Общая длительность разработки составила пять календарных месяцев работы по графику пять рабочих дней в неделю при восьмичасовом рабочем дне, что соответствует приблизительно 105 рабочим дням. Для коммерческой реализации проекта аналогичного масштаба и сложности потребовалась бы команда из нескольких специалистов разных профилей. Минимальный состав команды включает разработчика серверной части (Python, FastAPI, SQLAlchemy, проектирование базы данных), разработчика клиентской части (TypeScript, React, Next.js, вёрстка интерфейса), а также частичное привлечение специалиста по пользовательскому интерфейсу, психолога–консультанта для согласования состава и весов методик и менеджера проекта для координации работ.

Согласно открытым данным сервисов hh.ru и «Хабр Карьера» по состоянию на 2026 год, средняя ежемесячная заработная плата специалистов уровня middle на российском рынке труда составляет: для разработчика серверной части на языке Python – около 220.000 рублей, для разработчика клиентской части на React и Next.js – около 190.000 рублей, для специалиста по пользовательскому интерфейсу – около 160.000 рублей, для психолога–консультанта – около 100.000 рублей, для менеджера проекта – около 180.000 рублей. С учётом полной занятости разработчиков серверной и клиентской частей на протяжении всех пяти месяцев и частичной занятости остальных специалистов фонд оплаты труда команды составил бы порядка 2.750.000 рублей. Дополнительно работодатель

					ПГУ 1.090501.07.001 ПЗ	Лист
						137
Изм.	Лист	№ докум.	Подпись	Дата		

уплачивает страховые взносы в размере 30,2 процента от фонда оплаты труда, что добавляет к расчётной стоимости ещё около 830.000 рублей. [26]

Помимо затрат на оплату труда, при реализации проекта необходимо учесть инфраструктурные расходы: лицензии на специализированное программное обеспечение и среды разработки (около 50.000 рублей), аренду виртуального сервера для размещения системы и регистрацию доменного имени в зоне .ru на год эксплуатации (около 20.000 рублей), а также прочие сопутствующие расходы на тестирование, документирование и подготовку к опытной эксплуатации (порядка 50.000 рублей). Итоговая совокупная стоимость разработки и первого года эксплуатации системы в коммерческой среде ориентировочно составляет 3.700.000 рублей.

Несмотря на то, что разработанная система предоставляется пользователям бесплатно, в долгосрочной перспективе проект способен достичь экономической окупаемости за счёт размещения тематической рекламы и партнёрских интеграций. Целевая аудитория системы – школьники, студенты и взрослые, выбирающие профессиональный путь – представляет интерес для рекламодателей сразу из нескольких сегментов: образовательных платформ и онлайн–курсов, маркетплейсов одежды и аксессуаров для учащихся (Wildberries, Ozon, Lamoda), издательств учебной литературы, а также сервисов трудоустройства. Размещение баннерной и контекстной рекламы на страницах личного кабинета, каталога профессий и итоговой рекомендации, по средним рыночным ставкам российских рекламных сетей с показателем CPM от 50 до 150 рублей за тысячу показов, при условии достижения посещаемости порядка 10.000 уникальных пользователей в месяц позволяет получать ежемесячный доход от 15.000 до 50.000 рублей. Дополнительный доход может быть получен от партнёрских программ маркетплейсов – за каждый переход пользователя по партнёрской ссылке и совершённую покупку владелец сайта получает комиссию в размере от 3 до 8 процентов.

					ПГУ 1.090501.07.001 ПЗ	Лист
						138
Изм.	Лист	№ докум.	Подпись	Дата		

При устойчивом росте посещаемости и расширении состава рекламных партнёров проект способен выйти на самоокупаемость в течение 18–24 месяцев эксплуатации, после чего обеспечивает стабильный пассивный доход при сохранении бесплатного доступа для конечного пользователя.

Таким образом, с экономической точки зрения проект, разработанный в рамках выпускной квалификационной работы, с учётом заработной платы сотрудников, страховых отчислений и инфраструктурных расходов, оценивается примерно в 3,5–4,0 миллиона рублей. Полученная оценка показывает, что выполненная работа по объёму и сложности соответствует полноценному коммерческому проекту. При этом наличие потенциала монетизации через размещение тематической рекламы и партнёрские интеграции с маркетплейсами обеспечивает возможность выхода проекта на самоокупаемость в среднесрочной перспективе при сохранении бесплатного доступа для конечного пользователя, что особенно важно для системы социально значимого назначения, ориентированной на широкую аудиторию школьников, студентов и взрослых, выбирающих профессиональный путь.

					ПГУ 1.090501.07.001 ПЗ	Лист
						139
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

В ходе выполнения данной выпускной квалификационной работы была спроектирована и разработана автоматизированная система психодиагностики для задач профессионального отбора, представляющая собой полноценное веб–приложение, доступное пользователю через современный веб–браузер с любого устройства.

В результате был проведён анализ предметной области профессионального самоопределения и профессионального отбора, рассмотрено текущее состояние государственной политики Российской Федерации в сфере профориентации, изучены основные психодиагностические методики, применяемые в данной области. Был выполнен подробный анализ существующих программных аналогов – рассмотрены семь наиболее распространённых на российском рынке систем (Profilum.ru, Proforientator.ru, Skillbox Profest, Нейропрофориентация.рф, Proektoria.online, Postupi.online, Trudvsem.ru), выявлены их сильные и слабые стороны, на основании чего сформулированы требования к собственной разработке, призванной закрыть выявленную нишу доступного, объективного и комплексного инструмента автоматизированной психодиагностики.

Перед началом разработки продукта были выбраны программные средства и инструменты реализации. В качестве языка программирования серверной части выбран Python 3.11 с фреймворком FastAPI, в качестве объектно–реляционного отображателя – SQLAlchemy 2.0, для версионирования схемы базы – Alembic, для аутентификации – PyJWT, для хеширования паролей – passlib с алгоритмом PBKDF2–SHA256. Клиентская часть реализована на языке TypeScript с использованием библиотеки React и фреймворка Next.js. В качестве системы управления базой данных выбрана MySQL 8.x в связке с графическим клиентом MySQL Workbench, упрощающим процесс администрирования и проектирования. Основной средой разработки выступила Visual Studio Code.

На основе приведённых технологий удалось спроектировать, разработать и протестировать веб–приложение со следующим функционалом:

- регистрация и авторизация пользователей с защищённым хранением паролей в виде криптографических хешей и аутентификацией на основе JWT–токенов;
- личный кабинет пользователя с отображением анкетных данных и прогресса прохождения психодиагностических методик;
- проведение пяти психодиагностических методик: стартового предметного теста «Школьные ориентиры», дифференциально–диагностического опросника Е. А. Климова, методики профессиональных интересов Дж. Холланда (RIASEC), пятифакторной личностной модели Big Five и теста ценностных ориентаций М. Рокича;
- автоматический контроль достоверности ответов пользователя на основе анализа согласованности оценок и самооценок, а также общего уровня мотивации;
- детерминированный расчёт интегрального профиля пользователя по единому набору из девяти универсальных шкал черт личности и сопоставление его с эталонными профилями 37 профессий каталога по методу косинусной близости;
- формирование рейтинга наиболее подходящих профессий с указанием степени соответствия и автоматически генерируемого текстового пояснения;
- каталог профессий с алфавитным поиском и подробной карточкой по каждой позиции;
- формирование и скачивание итогового отчёта в формате PDF;
- административная панель с обезличенной статистикой использования системы, управлением учётными записями пользователей, маскированием персональных данных, выгрузкой отчётов по конкретным пользователям и редактированием каталога профессий.

Корректность работы реализованных функций подтверждена двумя взаимодополняющими подходами: автоматическими тестами на фреймворках pytest и Vitest, а также ручным тестированием наиболее распространённых пользовательских сценариев и обработки исключительных ситуаций.

Разработанная в рамках данной выпускной квалификационной работы система на текущий момент содержит весь необходимый базовый функционал для начала комфортной работы. Однако, как и большинство других программных продуктов, она допускает дальнейшее развитие в плане реализации новых функций и улучшения работы действующих. В качестве перспективных направлений доработки могут рассматриваться: расширение каталога профессий до нескольких сотен позиций, добавление новых психодиагностических методик в систему, реализация модуля партнёрских интеграций с маркетплейсами и образовательными платформами для обеспечения экономической самокупаемости проекта, а также подключение модуля сравнения результатов пользователя во времени для отслеживания динамики профессионального самоопределения.

Созданная автоматизированная система психодиагностики, призванная упростить и сделать доступным для широкой аудитории процесс профессионального самоопределения, полностью готова к опытной эксплуатации. Её бесплатное распространение, объединение пяти проверенных психодиагностических методик в едином сценарии, наличие автоматического контроля достоверности ответов, объективный детерминированный алгоритм формирования рекомендации и универсальность по целевой аудитории закрывают ту нишу, в которой существующие аналоги представлены лишь частично, что делает разработанный программный продукт востребованным как для самостоятельного использования широким кругом пользователей, так и в составе инструментария профконсультантов и кадровых служб, занимающихся задачами первичного профессионального отбора.

Список использованных источников

1. Бодров, В.А. Психология профессиональной пригодности: учебное пособие для вузов / В.А. Бодров. – 2-е изд. – Москва: ПЕР СЭ, 2006. – 511 с.
2. Носс, И.Н. Профессиональная психодиагностика: психологический отбор персонала / И.Н. Носс. – Москва: Психотерапия, 2009. – 464 с.
3. Кулагин, Б.В. Основы профессиональной психодиагностики / Б.В. Кулагин. – Ленинград: Медицина, 1984. – 216 с.
4. Бурлачук, Л.Ф. Психодиагностика: учебник для вузов / Л.Ф. Бурлачук. – Санкт-Петербург: Питер, 2006. – 351 с.
5. Доусон, М. Програмируем на Python / М. Доусон. – Санкт-Петербург: Питер, 2014. – 416 с.
6. React Documentation. Официальный сайт библиотеки. [Электронный ресурс]. – Режим доступа: <https://react.dev/> (дата обращения 24.04.2026).
7. TypeScript Handbook. Официальная документация языка программирования. [Электронный ресурс]. – Режим доступа: <https://www.typescriptlang.org/docs/> (дата обращения 14.05.2026).
8. Next.js Documentation. Официальный сайт фреймворка. [Электронный ресурс]. – Режим доступа: <https://nextjs.org/docs> (дата обращения 16.04.2026).
9. MDN Web Docs. Документация по веб-технологиям (HTML, CSS, JavaScript). [Электронный ресурс]. – Режим доступа: <https://developer.mozilla.org/ru/> (дата обращения 10.04.2026).
10. MySQL 8.0 Reference Manual. Официальная документация системы управления базами данных. [Электронный ресурс]. – Режим доступа: <https://dev.mysql.com/doc/refman/8.0/en/> (дата обращения 28.03.2026).
11. FastAPI Documentation. Официальная документация фреймворка. [Электронный ресурс]. – Режим доступа: <https://fastapi.tiangolo.com/ru/> (дата обращения 03.03.2026).

12. SQLAlchemy 2.0 Documentation. Официальный сайт ORM-библиотеки. [Электронный ресурс]. – Режим доступа: <https://docs.sqlalchemy.org/en/20/> (дата обращения 20.03.2026).

13. passlib Documentation. Документация библиотеки хеширования паролей. [Электронный ресурс]. – Режим доступа: <https://passlib.readthedocs.io/en/stable/> (дата обращения 30.04.2026).

14. PyJWT Documentation. Документация библиотеки работы с JSON Web Token. [Электронный ресурс]. – Режим доступа: <https://pyjwt.readthedocs.io/en/stable/> (дата обращения 07.05.2026).

15. pytest Documentation. Документация фреймворка автоматического тестирования. [Электронный ресурс]. – Режим доступа: <https://docs.pytest.org/en/stable/> (дата обращения 11.04.2026).

16. Климов, Е.А. Психология профессионального самоопределения: учебное пособие для студентов высших учебных заведений / Е.А. Климов. – 4-е изд., стер. – Москва: Академия, 2010. – 304 с.

17. Профессиональная ориентация личности с позиции типологии Дж. Холланда. Научная электронная библиотека «КиберЛенинка». [Электронный ресурс]. – Режим доступа: <https://cyberleninka.ru/article/n/professionalnaya-orientatsiya-lichnosti-s-pozitsii-tipologii-dzh-hollanda> (дата обращения 28.02.2026).

18. Применение пятифакторной модели личности Big Five в задачах профессионального отбора. Научная электронная библиотека «КиберЛенинка». [Электронный ресурс]. – Режим доступа: <https://cyberleninka.ru/article/n/pyatifaktornaya-model-lichnosti-big-five> (дата обращения 09.04.2026).

19. Методика ценностных ориентаций М. Рокича: теоретическое обоснование и практика применения. Научная электронная библиотека «КиберЛенинка». [Электронный ресурс]. – Режим доступа: <https://cyberleninka.ru/article/n/tsennostnye-orientatsii-m-rokicha> (дата обращения 17.03.2026).

20. Современные подходы к построению архитектуры веб-приложений. Хабр. [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/articles/498620/> (дата обращения 25.03.2026).

21. Реализация JSON Web Token в задачах аутентификации веб-приложений. Хабр. [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/articles/340146/> (дата обращения 02.04.2026).

22. Федеральный закон от 27 июля 2006 года № 152-ФЗ «О персональных данных» (с изменениями и дополнениями). [Электронный ресурс]. – Режим доступа: http://www.consultant.ru/document/cons_doc_LAW_61801/ (дата обращения 12.05.2026).

23. Инструкция по охране труда при работе с ЭВМ. [Электронный ресурс]. – Режим доступа: https://kpfu.ru/portal/docs/F_1086113514/1815.pdf (дата обращения 12.05.2026).

24. ГОСТ 12.2.032–78. Система стандартов безопасности труда. Рабочее место при выполнении работ сидя. Общие эргономические требования. – Москва: Стандартиформ, 2008. – 9 с.

25. СанПиН 1.2.3685–21. Гигиенические нормативы и требования к обеспечению безопасности и (или) безвредности для человека факторов среды обитания. – Москва: Главный государственный санитарный врач Российской Федерации, 2021. – 469 с.

26. Статистика рынка труда ИТ-специалистов. Хабр Карьера. [Электронный ресурс]. – Режим доступа: <https://career.habr.com/salaries> (дата обращения 19.05.2026).

Приложение А

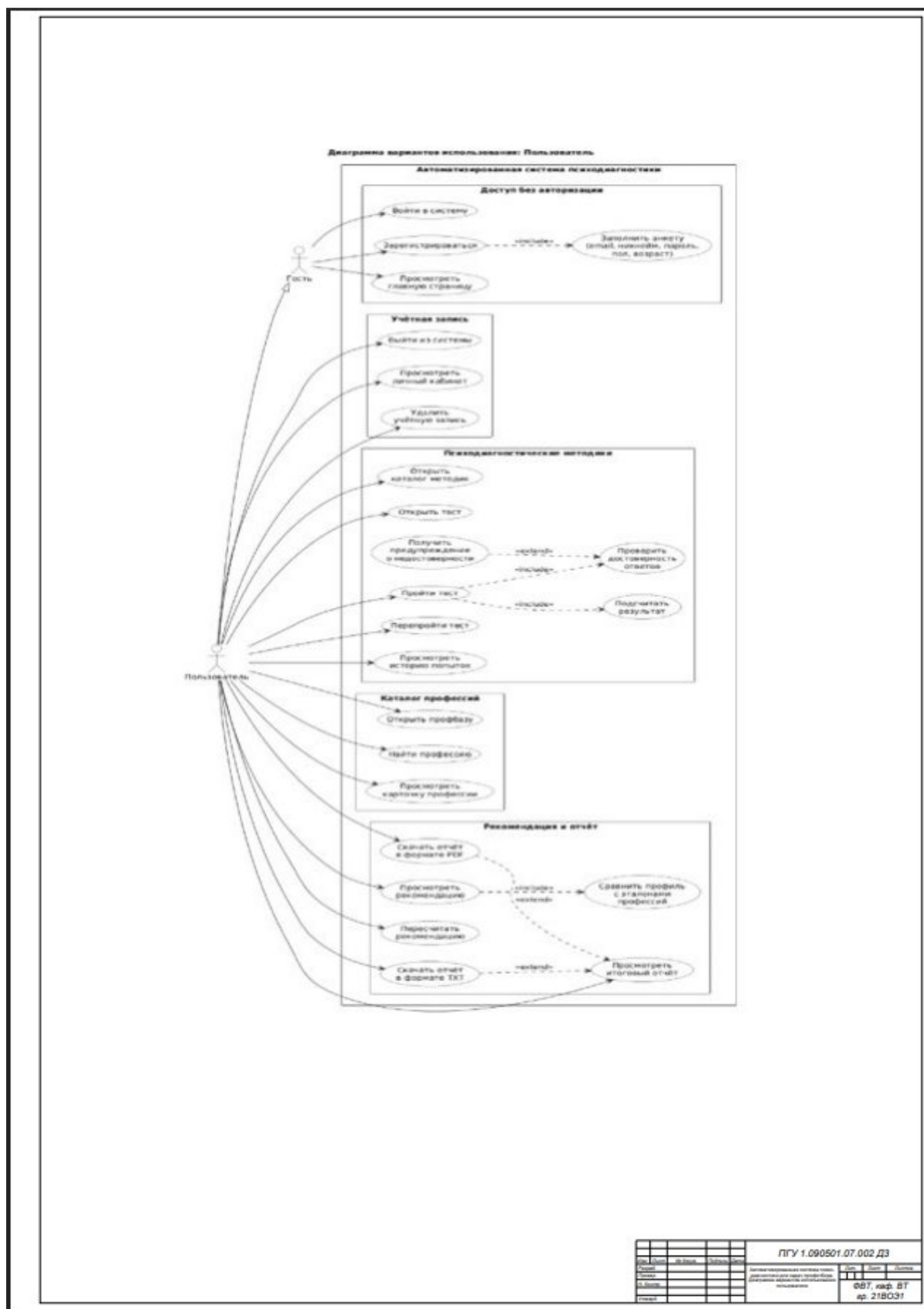
(обязательное)

Автоматизированная система психодиагностики для задач профотбора

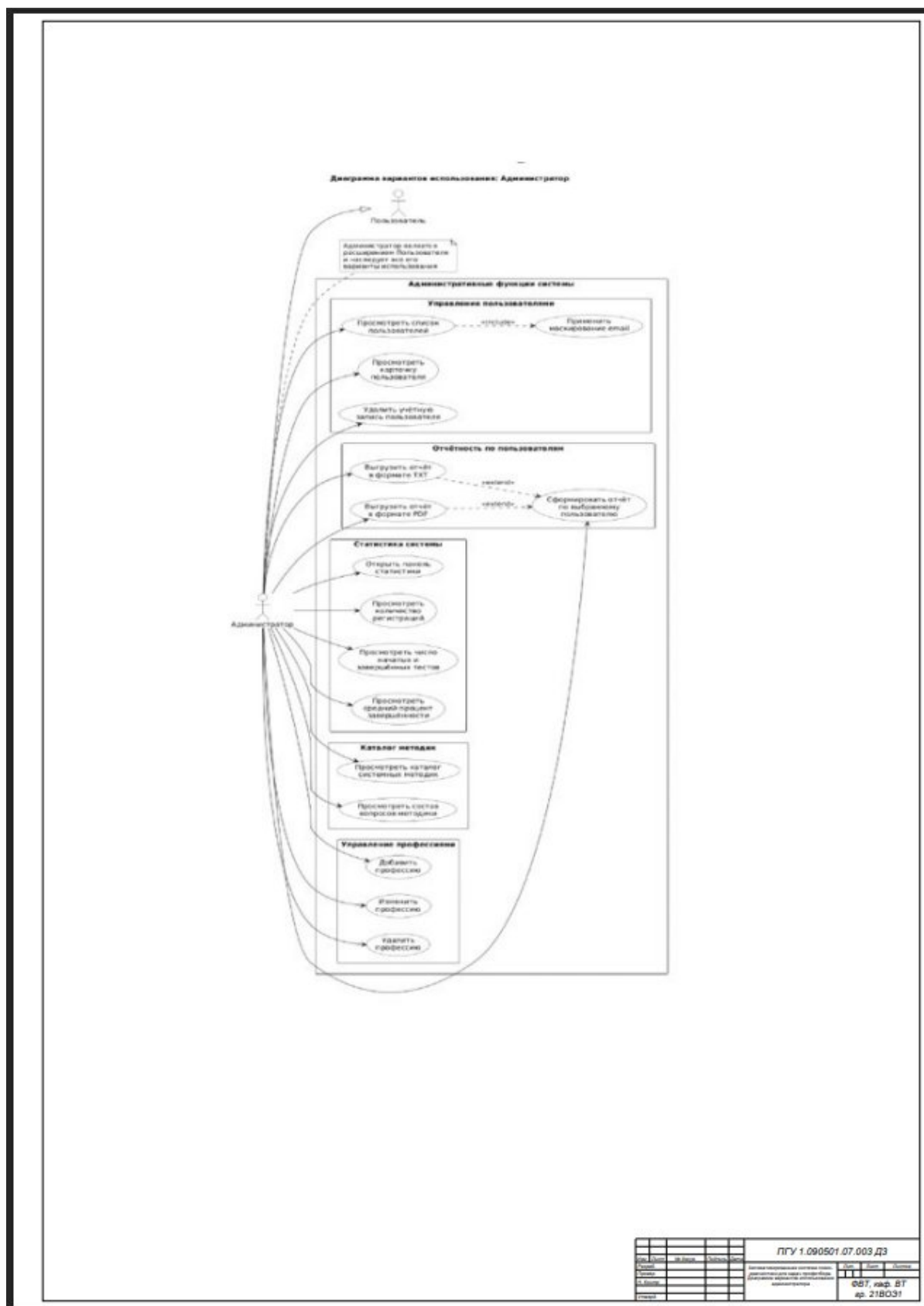
UML–диаграммы

					ПГУ 1.090501.07.001 ПЗ	Лист
						146
Изм.	Лист	№ докум.	Подпись	Дата		

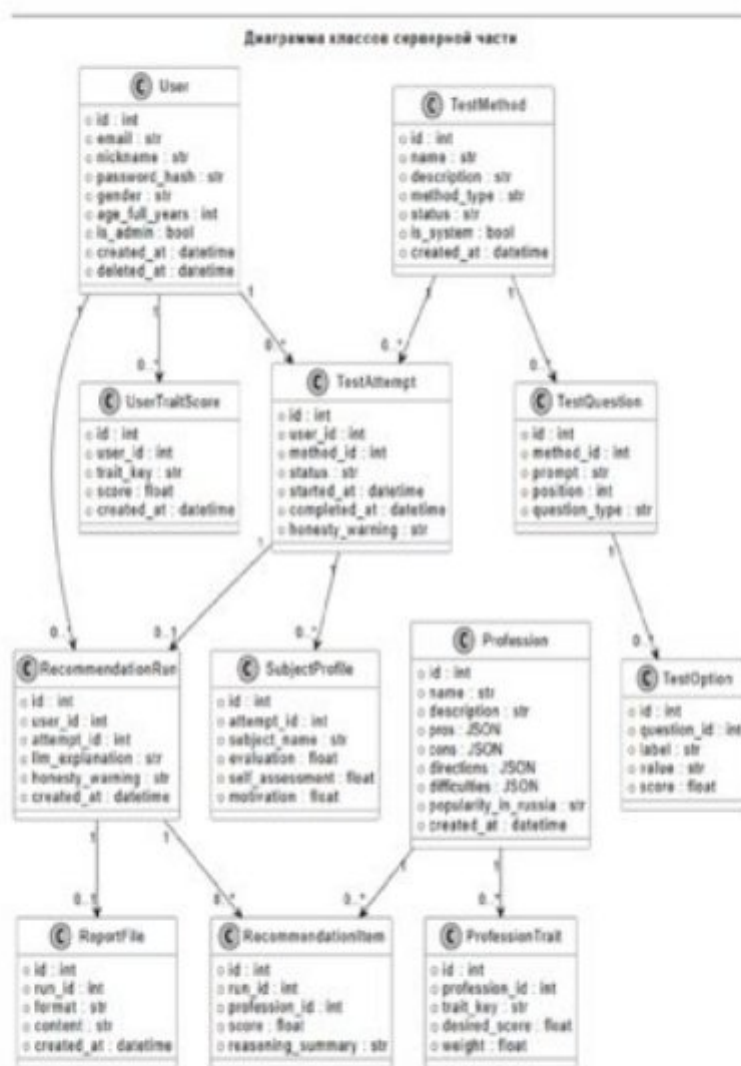
Приложение А.1 – Диаграмма вариантов использования пользователя



Приложение А.2 – Диаграмма вариантов использования администратора



Приложение А.3 – Диаграмма классов серверной части



ПГУ 1.090501.07.004 Д11			
Имя	Фамилия	Имя	Фамилия
Подпись	Подпись	Подпись	Подпись
Дата	Дата	Дата	Дата

ФБТ, вып. БТ
ар. 218031

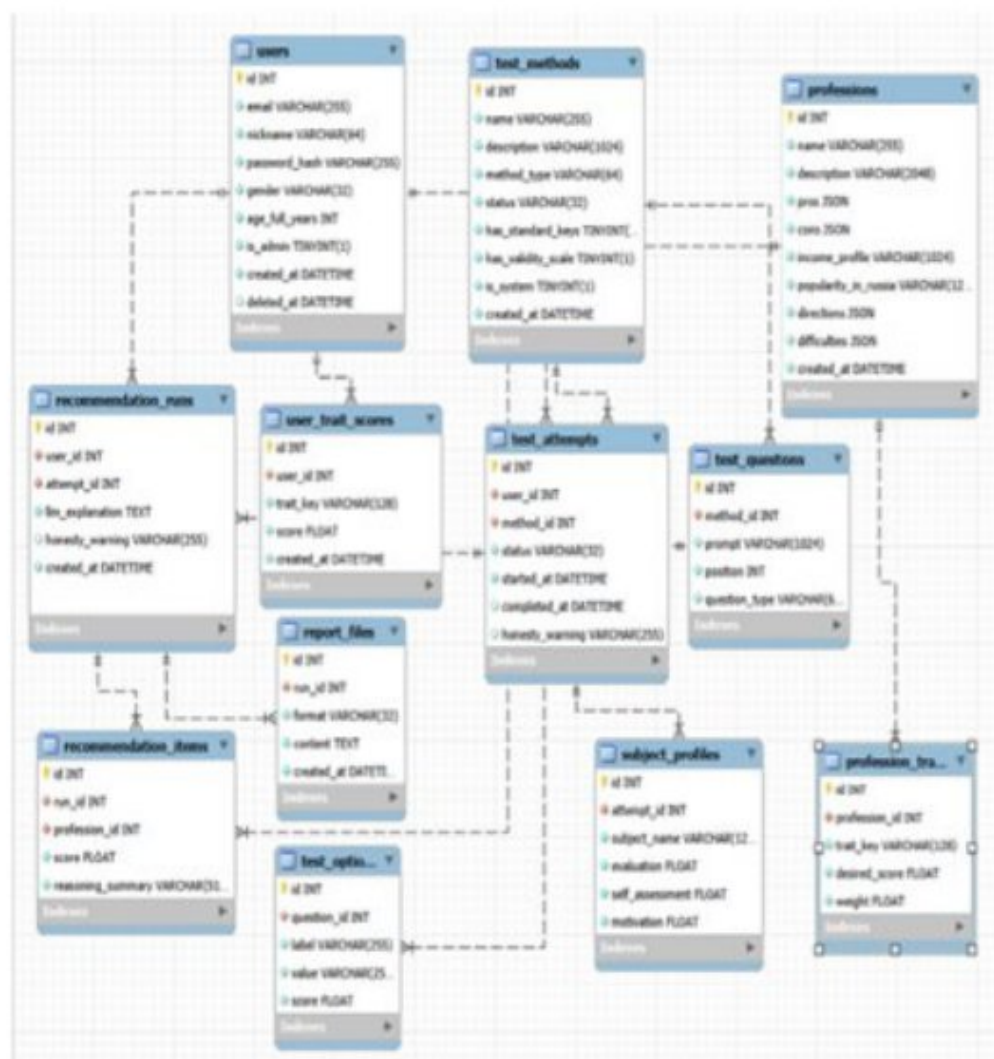
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.07.001 ПЗ

Лист

149

Приложение А.4 – ER-диаграмма базы данных (MySQL Workbench)



ПГУ 1.090501.07.005 Д1			
Исполн.	Провер.	Дата	Статус
Форм. 1.090501.07.005 Д1			
ФБТ, шаг БТ			
ар. 218031			

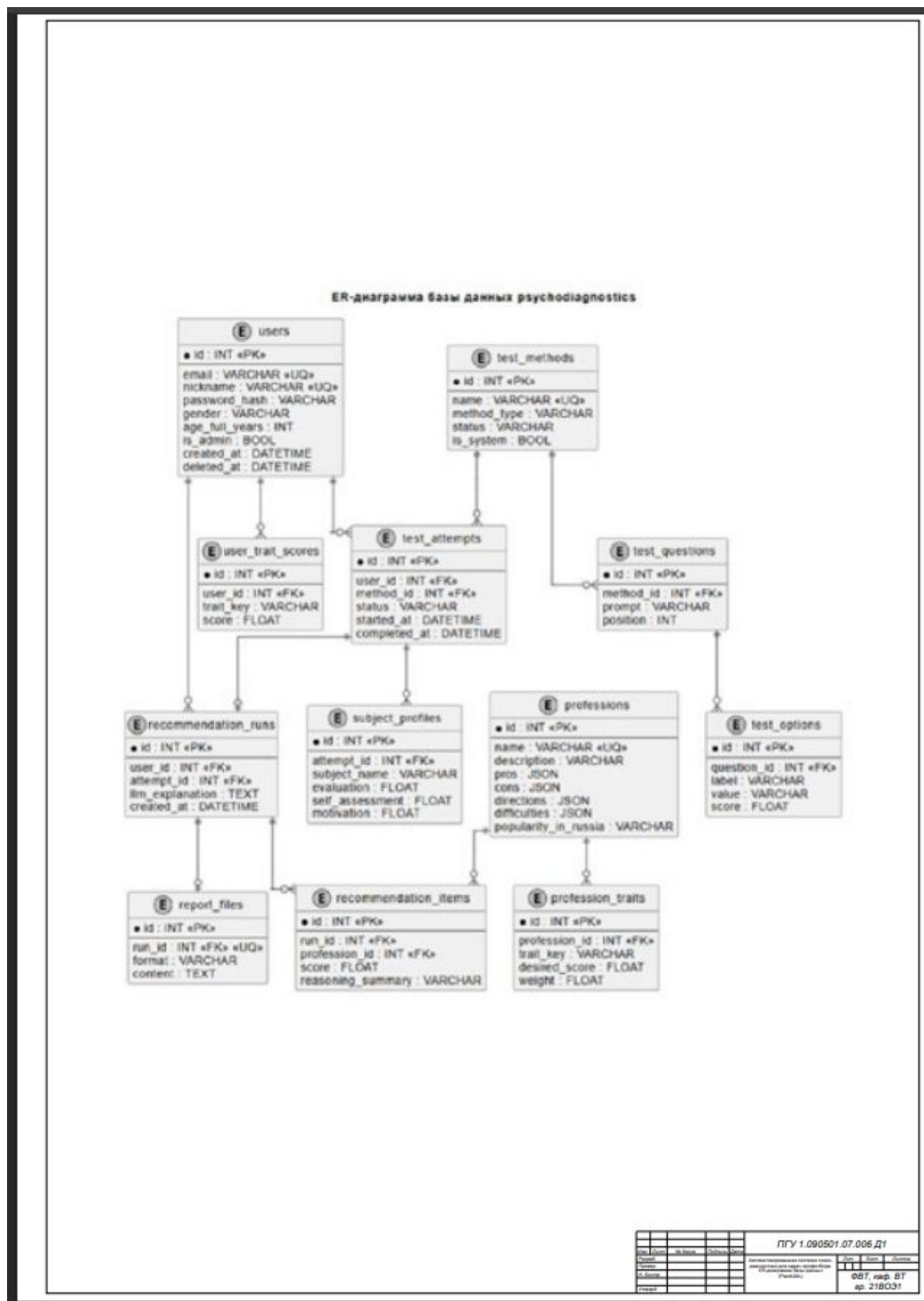
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.07.001 ПЗ

Лист

150

Приложение А.5 – ER-диаграмма базы данных (PlantUML)



ПГУ 1.090501.07.006 Д1			
Имя	Фамилия	Отчество	Подпись
Дата	Время	Место	Подпись
С. 1	С. 2	С. 3	С. 4
С. 5	С. 6	С. 7	С. 8
С. 9	С. 10	С. 11	С. 12

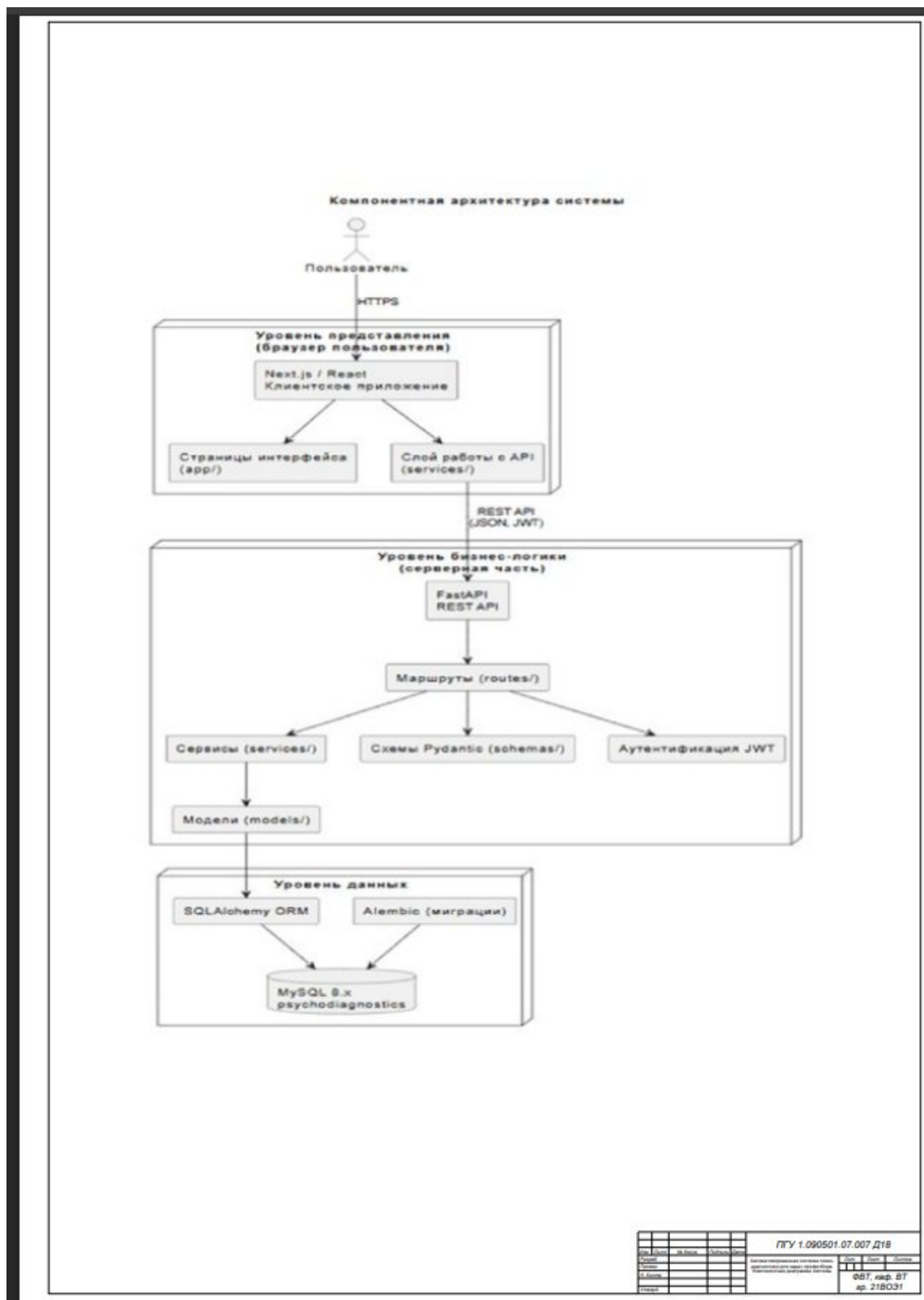
Изм.	Лист	№ докум.	Подпись	Дата

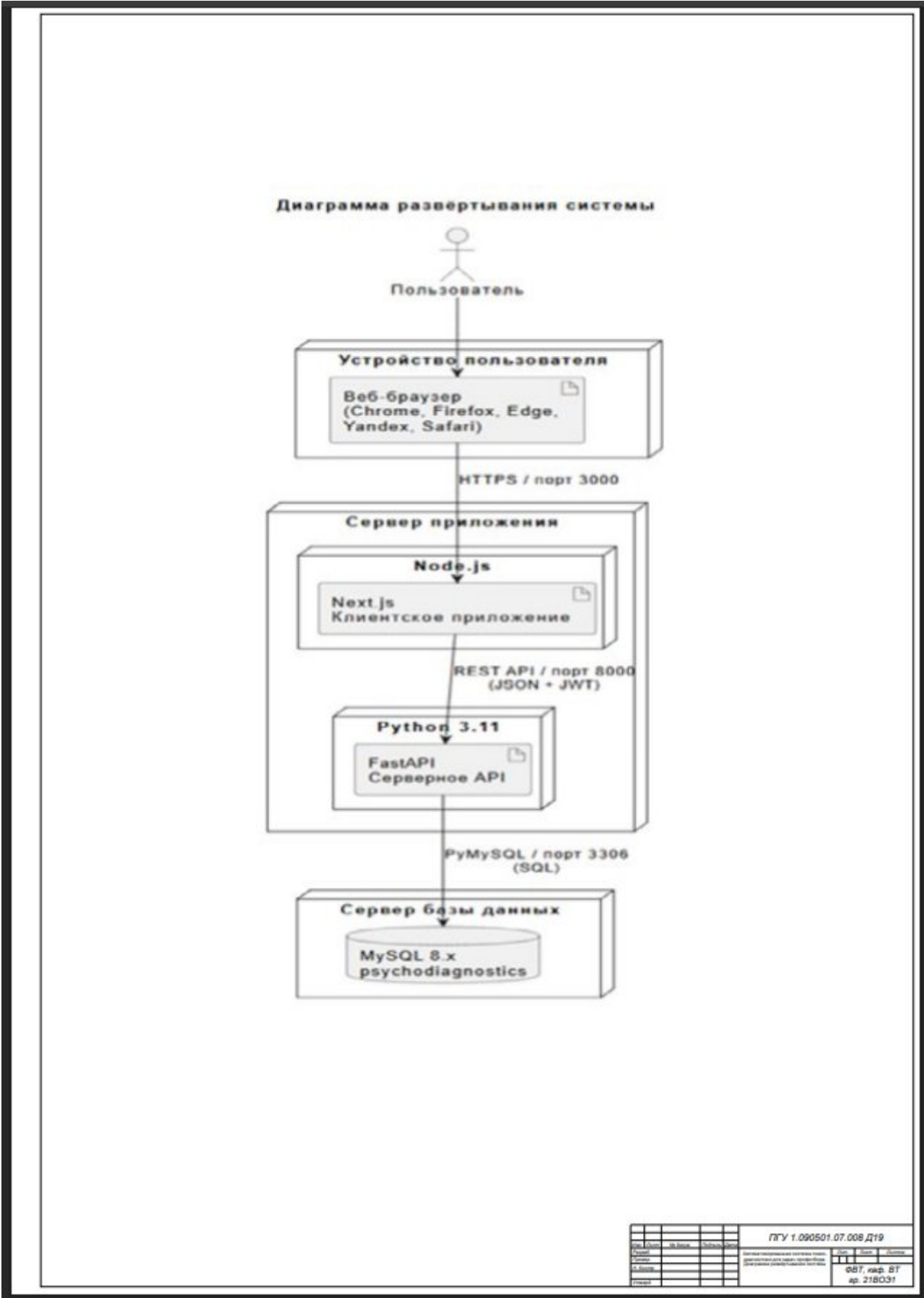
ПГУ 1.090501.07.001 ПЗ

Лист

151

Приложение А.6 – Компонентная диаграмма системы





Приложение А.8 – Диаграмма последовательности для сценария расчёта рекомендации

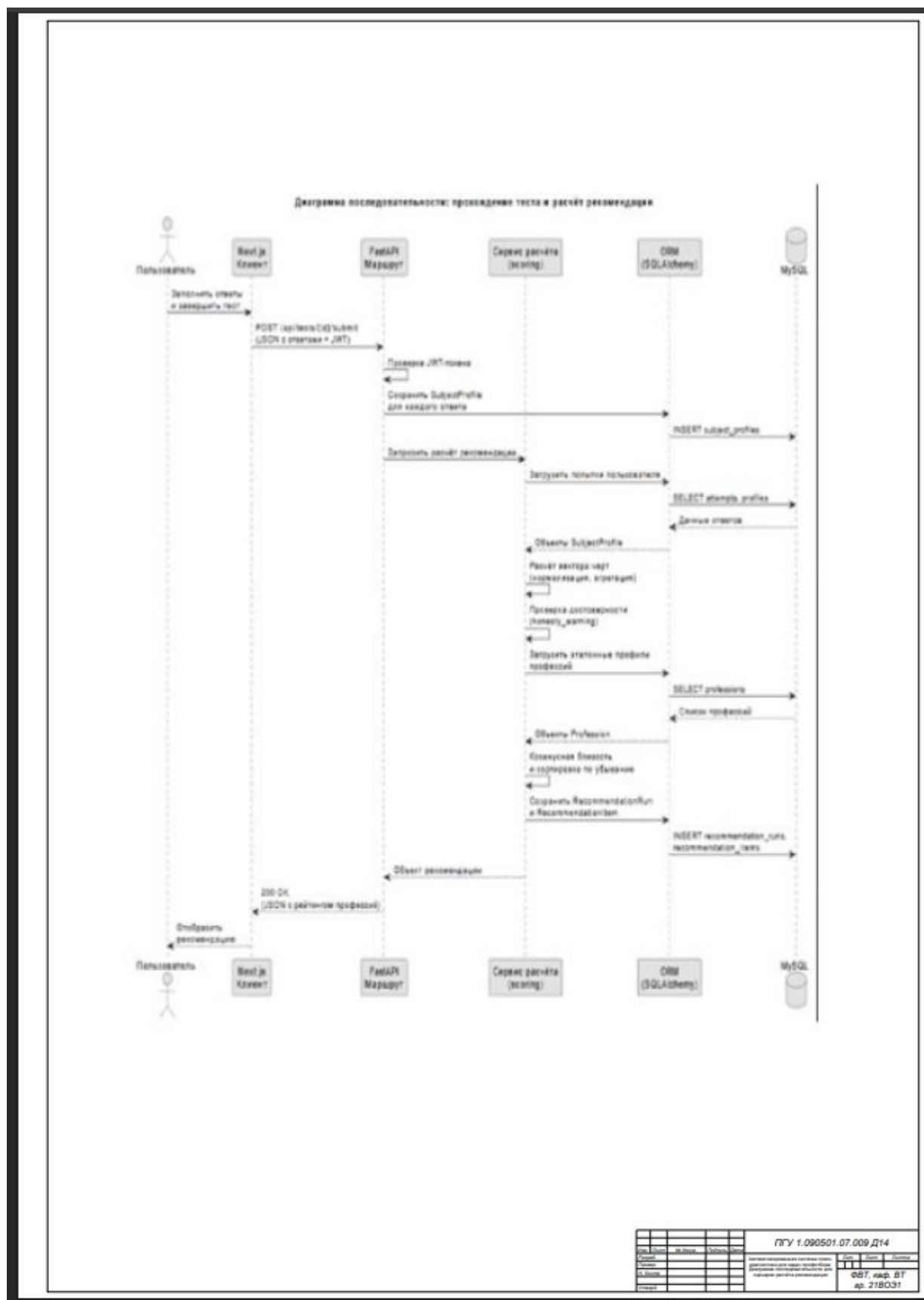
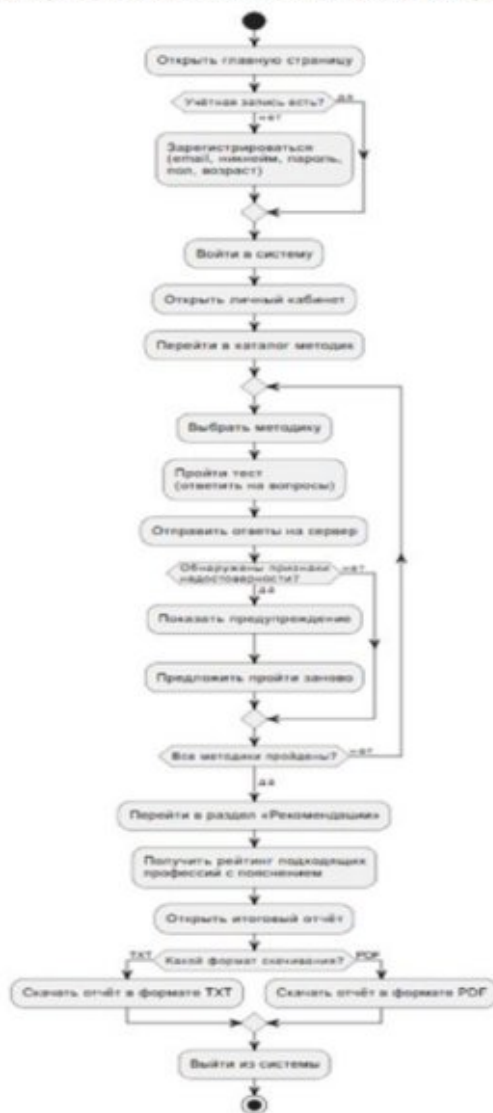


Диаграмма деятельности: пользовательский сценарий



ПГУ 1.090501.07.010 Д17			
Исполн.	Провер.	Утверд.	Дата
Созд.	Провер.	Утверд.	Дата
Исполн.	Провер.	Утверд.	Дата
Созд.	Провер.	Утверд.	Дата

ФБГ, илф. БГ
пр. 218031

Приложение Б

(обязательное)

Автоматизированная система психодиагностики для задач профотбора

Листинги программы

					ПГУ 1.090501.07.001 ПЗ	Лист
						156
Изм.	Лист	№ докум.	Подпись	Дата		

Листинг А.1 – Файл «psychodiagnostics.sql – создание базы данных и всех таблиц»

```

CREATE DATABASE IF NOT EXISTS psychodiagnostics
  DEFAULT CHARACTER SET utf8mb4
  DEFAULT COLLATE utf8mb4_0900_ai_ci;

USE psychodiagnostics;

CREATE TABLE users (
  id INT PRIMARY KEY AUTO_INCREMENT,
  email VARCHAR(255) NOT NULL UNIQUE,
  nickname VARCHAR(64) NOT NULL UNIQUE,
  password_hash VARCHAR(255) NOT NULL,
  gender VARCHAR(32) NOT NULL,
  age_full_years INT NOT NULL,
  is_admin BOOLEAN DEFAULT FALSE,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  deleted_at DATETIME NULL,
  INDEX idx_users_email (email),
  INDEX idx_users_nickname (nickname)
);

CREATE TABLE test_methods (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL UNIQUE,
  description VARCHAR(1024) NOT NULL,
  method_type VARCHAR(64) DEFAULT 'subject_starter',
  status VARCHAR(32) DEFAULT 'NOT_STARTED',
  has_standard_keys BOOLEAN DEFAULT TRUE,
  has_validity_scale BOOLEAN DEFAULT TRUE,
  is_system BOOLEAN DEFAULT FALSE,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE test_questions (
  id INT PRIMARY KEY AUTO_INCREMENT,
  method_id INT NOT NULL,
  prompt VARCHAR(1024) NOT NULL,
  position INT NOT NULL,
  question_type VARCHAR(64) DEFAULT 'subject_profile',
  FOREIGN KEY (method_id) REFERENCES test_methods(id) ON DELETE
  CASCADE,
  INDEX idx_questions_method (method_id)
);

CREATE TABLE test_options (
  id INT PRIMARY KEY AUTO_INCREMENT,
  question_id INT NOT NULL,
  label VARCHAR(255) NOT NULL,
  value VARCHAR(255) NOT NULL,
  score FLOAT DEFAULT 0.0,
  FOREIGN KEY (question_id) REFERENCES test_questions(id) ON DELETE
  CASCADE,

```

```

        INDEX idx_options_question (question_id)
    );

CREATE TABLE test_attempts (
    id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT NOT NULL,
    method_id INT NOT NULL,
    status VARCHAR(32) DEFAULT 'IN_PROGRESS',
    started_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    completed_at DATETIME NULL,
    honesty_warning VARCHAR(255) NULL,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (method_id) REFERENCES test_methods(id) ON DELETE
CASCADE,
    INDEX idx_attempts_user (user_id),
    INDEX idx_attempts_method (method_id)
);

CREATE TABLE subject_profiles (
    id INT PRIMARY KEY AUTO_INCREMENT,
    attempt_id INT NOT NULL,
    subject_name VARCHAR(128) NOT NULL,
    evaluation FLOAT NOT NULL,
    self_assessment FLOAT NOT NULL,
    motivation FLOAT NOT NULL,
    FOREIGN KEY (attempt_id) REFERENCES test_attempts(id) ON DELETE
CASCADE,
    UNIQUE KEY uk_attempt_subject (attempt_id, subject_name),
    INDEX idx_profiles_subject (subject_name)
);

CREATE TABLE user_trait_scores (
    id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT NOT NULL,
    trait_key VARCHAR(128) NOT NULL,
    score FLOAT NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    INDEX idx_trait_user (user_id),
    INDEX idx_trait_key (trait_key)
);

CREATE TABLE professions (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL UNIQUE,
    description VARCHAR(2048) NOT NULL,
    pros JSON,
    cons JSON,
    income_profile VARCHAR(1024) DEFAULT '',
    popularity_in_russia VARCHAR(128) DEFAULT '',
    directions JSON,
    difficulties JSON,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP

```

);

```
CREATE TABLE profession_traits (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    profession_id INT NOT NULL,  
    trait_key VARCHAR(128) NOT NULL,  
    desired_score FLOAT NOT NULL,  
    weight FLOAT NOT NULL,  
    FOREIGN KEY (profession_id) REFERENCES professions(id) ON DELETE  
CASCADE,  
    INDEX idx_traits_profession (profession_id),  
    INDEX idx_traits_key (trait_key)  
);
```

```
CREATE TABLE recommendation_runs (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    user_id INT NOT NULL,  
    attempt_id INT NOT NULL,  
    llm_explanation TEXT NOT NULL,  
    honesty_warning VARCHAR(255) NULL,  
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,  
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,  
    FOREIGN KEY (attempt_id) REFERENCES test_attempts(id) ON DELETE  
CASCADE,  
    INDEX idx_runs_user (user_id),  
    INDEX idx_runs_attempt (attempt_id)  
);
```

```
CREATE TABLE recommendation_items (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    run_id INT NOT NULL,  
    profession_id INT NOT NULL,  
    score FLOAT NOT NULL,  
    reasoning_summary VARCHAR(512) NOT NULL,  
    FOREIGN KEY (run_id) REFERENCES recommendation_runs(id) ON DELETE  
CASCADE,  
    FOREIGN KEY (profession_id) REFERENCES professions(id),  
    INDEX idx_items_run (run_id),  
    INDEX idx_items_profession (profession_id)  
);
```

```
CREATE TABLE report_files (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    run_id INT NOT NULL UNIQUE,  
    format VARCHAR(32) DEFAULT 'txt',  
    content TEXT NOT NULL,  
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,  
    FOREIGN KEY (run_id) REFERENCES recommendation_runs(id) ON DELETE  
CASCADE,  
    INDEX idx_reports_run (run_id)  
);
```

Листинг А.2 – Файл «backend/app/main.py – точка входа серверной части»

```
from contextlib import asynccontextmanager

from fastapi import FastAPI

from app.config.config import get_settings
from app.db.base import Base
from app.db.session import SessionLocal, engine
from app.models import all_models
from app.routes.router import api_router
from app.services.seed import seed_database

@asynccontextmanager
async def lifespan(_: FastAPI):
    Base.metadata.create_all(bind=engine)
    with SessionLocal() as session:
        seed_database(session)
    yield

def create_app() -> FastAPI:
    settings = get_settings()
    app = FastAPI(
        title="Psychodiagnostics API",
        version="0.1.0",
        description="MVP backend for automated psychodiagnostics and
career guidance",
        lifespan=lifespan,
    )
    app.include_router(api_router, prefix=settings.api_prefix)

    @app.get("/health", tags=["health"])
    def healthcheck() -> dict[str, str]:
        return {"status": "ok"}

    return app

app = create_app()
```

Листинг А.3 – Файл «backend/app/config/config.py – настройки приложения»

```
from functools import lru_cache

from pydantic_settings import BaseSettings, SettingsConfigDict

class Settings(BaseSettings):
    app_env: str = "development"
    api_prefix: str = "/api"
    database_url: str =
"sqlite:///./database/runtime/psychodiagnostics.db"
    jwt_secret: str = "change-me"
    jwt_expire_minutes: int = 720
```

					ПГУ 1.090501.07.001 ПЗ	Лист
						160
Изм.	Лист	№ докум.	Подпись	Дата		

```

llm_api_key: str = ""
admin_email: str = "admin@example.com"
admin_password: str = "admin12345"

model_config = SettingsConfigDict(
    env_file=".env",
    env_file_encoding="utf-8",
    extra="ignore",
)

@lru_cache
def get_settings() -> Settings:
    return Settings()

```

Листинг А.4 – Файл «backend/app/db/session.py – подключение к базе данных»

```

from collections.abc import Generator

from sqlalchemy import create_engine
from sqlalchemy.orm import Session, sessionmaker

from app.config.config import get_settings

settings = get_settings()
connect_args = {"check_same_thread": False} if
settings.database_url.startswith("sqlite") else {}
engine = create_engine(settings.database_url, future=True,
connect_args=connect_args)
SessionLocal = sessionmaker(bind=engine, autoflush=False,
autocommit=False, future=True)

def get_db() -> Generator[Session, None, None]:
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

```

Листинг А.5 – Файл «backend/app/models/user.py – модель пользователя»

```

from __future__ import annotations

from datetime import datetime
from typing import Optional

from sqlalchemy import Boolean, DateTime, ForeignKey, Integer, String
from sqlalchemy.orm import Mapped, mapped_column, relationship

from app.db.base import Base

class User(Base):

```

```

__tablename__ = "users"

id: Mapped[int] = mapped_column(primary_key=True)
email: Mapped[str] = mapped_column(String(255), unique=True,
index=True)
nickname: Mapped[str] = mapped_column(String(64), unique=True,
index=True)
password_hash: Mapped[str] = mapped_column(String(255))
gender: Mapped[str] = mapped_column(String(32))
age_full_years: Mapped[int] = mapped_column(Integer)
is_admin: Mapped[bool] = mapped_column(Boolean, default=False)
created_at: Mapped[datetime] = mapped_column(DateTime,
default=datetime.utcnow)
deleted_at: Mapped[Optional[datetime]] = mapped_column(DateTime,
nullable=True)

attempts = relationship("TestAttempt", back_populates="user",
cascade="all, delete-orphan")
trait_scores = relationship("UserTraitScore",
back_populates="user", cascade="all, delete-orphan")
recommendation_runs = relationship(
    "RecommendationRun",
    back_populates="user",
    cascade="all, delete-orphan",
)

class UserTraitScore(Base):
    __tablename__ = "user_trait_scores"

    id: Mapped[int] = mapped_column(primary_key=True)
    user_id: Mapped[int] = mapped_column(ForeignKey("users.id"),
index=True)
    trait_key: Mapped[str] = mapped_column(String(128), index=True)
    score: Mapped[float]
    created_at: Mapped[datetime] = mapped_column(DateTime,
default=datetime.utcnow)

    user = relationship("User", back_populates="trait_scores")

```

Листинг А.6 – Файл «backend/app/models/testing.py – модели методик и попыток»

```

from __future__ import annotations

from datetime import datetime
from enum import Enum
from typing import Optional

from sqlalchemy import Boolean, DateTime, Float, ForeignKey, Integer,
String, UniqueConstraint
from sqlalchemy.orm import Mapped, mapped_column, relationship

from app.db.base import Base

```

					ПГУ 1.090501.07.001 ПЗ	Лист
						162
Изм.	Лист	№ докум.	Подпись	Дата		

```

class TestStatus(str, Enum):
    NOT_STARTED = "NOT_STARTED"
    COMPLETED = "COMPLETED"
    RETAKE_REQUIRED = "RETAKE_REQUIRED"
    IN_PROGRESS = "IN_PROGRESS"

class TestMethod(Base):
    __tablename__ = "test_methods"

    id: Mapped[int] = mapped_column(primary_key=True)
    name: Mapped[str] = mapped_column(String(255), unique=True)
    description: Mapped[str] = mapped_column(String(1024))
    method_type: Mapped[str] = mapped_column(String(64),
default="subject_starter")
    status: Mapped[str] = mapped_column(String(32),
default=TestStatus.NOT_STARTED.value)
    has_standard_keys: Mapped[bool] = mapped_column(Boolean,
default=True)
    has_validity_scale: Mapped[bool] = mapped_column(Boolean,
default=True)
    is_system: Mapped[bool] = mapped_column(Boolean, default=False)
    created_at: Mapped[datetime] = mapped_column(DateTime,
default=datetime.utcnow())

    questions = relationship("TestQuestion", back_populates="method",
cascade="all, delete-orphan")
    attempts = relationship("TestAttempt", back_populates="method",
cascade="all, delete-orphan")

class TestQuestion(Base):
    __tablename__ = "test_questions"

    id: Mapped[int] = mapped_column(primary_key=True)
    method_id: Mapped[int] =
mapped_column(ForeignKey("test_methods.id"), index=True)
    prompt: Mapped[str] = mapped_column(String(1024))
    position: Mapped[int] = mapped_column(Integer)
    question_type: Mapped[str] = mapped_column(String(64),
default="subject_profile")

    method = relationship("TestMethod", back_populates="questions")
    options = relationship("TestOption", back_populates="question",
cascade="all, delete-orphan")

class TestOption(Base):
    __tablename__ = "test_options"

    id: Mapped[int] = mapped_column(primary_key=True)
    question_id: Mapped[int] =
mapped_column(ForeignKey("test_questions.id"), index=True)
    label: Mapped[str] = mapped_column(String(255))
    value: Mapped[str] = mapped_column(String(255))

```

```

        score: Mapped[float] = mapped_column(Float, default=0.0)

        question = relationship("TestQuestion", back_populates="options")

class TestAttempt(Base):
    __tablename__ = "test_attempts"

    id: Mapped[int] = mapped_column(primary_key=True)
    user_id: Mapped[int] = mapped_column(ForeignKey("users.id"),
index=True)
    method_id: Mapped[int] =
mapped_column(ForeignKey("test_methods.id"), index=True)
    status: Mapped[str] = mapped_column(String(32),
default=TestStatus.IN_PROGRESS.value)
    started_at: Mapped[datetime] = mapped_column(DateTime,
default=datetime.utcnow)
    completed_at: Mapped[Optional[datetime]] = mapped_column(DateTime,
nullable=True)
    honesty_warning: Mapped[Optional[str]] =
mapped_column(String(255), nullable=True)

    user = relationship("User", back_populates="attempts")
    method = relationship("TestMethod", back_populates="attempts")
    subject_profiles = relationship(
        "SubjectProfile",
        back_populates="attempt",
        cascade="all, delete-orphan",
    )

class SubjectProfile(Base):
    __tablename__ = "subject_profiles"
    __table_args__ = (UniqueConstraint("attempt_id", "subject_name",
name="uq_attempt_subject"),)

    id: Mapped[int] = mapped_column(primary_key=True)
    attempt_id: Mapped[int] =
mapped_column(ForeignKey("test_attempts.id"), index=True)
    subject_name: Mapped[str] = mapped_column(String(128), index=True)
    evaluation: Mapped[float]
    self_assessment: Mapped[float]
    motivation: Mapped[float]

    attempt = relationship("TestAttempt",
back_populates="subject_profiles")

```

Листинг А.7 – Файл «backend/app/models/profession.py – модели профессий»

```

from __future__ import annotations

from datetime import datetime

from sqlalchemy import DateTime, Float, ForeignKey, JSON, String

```

```

from sqlalchemy.orm import Mapped, mapped_column, relationship

from app.db.base import Base

class Profession(Base):
    __tablename__ = "professions"

    id: Mapped[int] = mapped_column(primary_key=True)
    name: Mapped[str] = mapped_column(String(255), unique=True)
    description: Mapped[str] = mapped_column(String(2048))
    pros: Mapped[list[str]] = mapped_column(JSON, default=list)
    cons: Mapped[list[str]] = mapped_column(JSON, default=list)
    income_profile: Mapped[str] = mapped_column(String(1024),
default="")
    popularity_in_russia: Mapped[str] = mapped_column(String(128),
default="")
    directions: Mapped[list[str]] = mapped_column(JSON, default=list)
    difficulties: Mapped[list[str]] = mapped_column(JSON,
default=list)
    created_at: Mapped[datetime] = mapped_column(DateTime,
default=datetime.utcnow)

    traits = relationship("ProfessionTrait",
back_populates="profession", cascade="all, delete-orphan")
    recommendation_items = relationship("RecommendationItem",
back_populates="profession")

class ProfessionTrait(Base):
    __tablename__ = "profession_traits"

    id: Mapped[int] = mapped_column(primary_key=True)
    profession_id: Mapped[int] =
mapped_column(ForeignKey("professions.id"), index=True)
    trait_key: Mapped[str] = mapped_column(String(128), index=True)
    desired_score: Mapped[float] = mapped_column(Float)
    weight: Mapped[float] = mapped_column(Float)

    profession = relationship("Profession", back_populates="traits")

```

Листинг А.8 – Файл «backend/app/models/recommendation.py – модели рекомендаций»

```

from __future__ import annotations

from datetime import datetime
from typing import Optional

from sqlalchemy import DateTime, Float, ForeignKey, String, Text
from sqlalchemy.orm import Mapped, mapped_column, relationship

from app.db.base import Base

```

					ПГУ 1.090501.07.001 ПЗ	Лист
						165
Изм.	Лист	№ докум.	Подпись	Дата		

```

class RecommendationRun(Base):
    __tablename__ = "recommendation_runs"

    id: Mapped[int] = mapped_column(primary_key=True)
    user_id: Mapped[int] = mapped_column(ForeignKey("users.id"),
index=True)
    attempt_id: Mapped[int] =
mapped_column(ForeignKey("test_attempts.id"), index=True)
    llm_explanation: Mapped[str] = mapped_column(Text)
    honesty_warning: Mapped[Optional[str]] =
mapped_column(String(255), nullable=True)
    created_at: Mapped[datetime] = mapped_column(DateTime,
default=datetime.utcnow)

    user = relationship("User", back_populates="recommendation_runs")
    items = relationship("RecommendationItem", back_populates="run",
cascade="all, delete-orphan")
    report = relationship("ReportFile", back_populates="run",
uselist=False, cascade="all, delete-orphan")

class RecommendationItem(Base):
    __tablename__ = "recommendation_items"

    id: Mapped[int] = mapped_column(primary_key=True)
    run_id: Mapped[int] =
mapped_column(ForeignKey("recommendation_runs.id"), index=True)
    profession_id: Mapped[int] =
mapped_column(ForeignKey("professions.id"), index=True)
    score: Mapped[float] = mapped_column(Float)
    reasoning_summary: Mapped[str] = mapped_column(String(512))

    run = relationship("RecommendationRun", back_populates="items")
    profession = relationship("Profession",
back_populates="recommendation_items")

class ReportFile(Base):
    __tablename__ = "report_files"

    id: Mapped[int] = mapped_column(primary_key=True)
    run_id: Mapped[int] =
mapped_column(ForeignKey("recommendation_runs.id"), unique=True,
index=True)
    format: Mapped[str] = mapped_column(String(32), default="txt")
    content: Mapped[str] = mapped_column(Text)
    created_at: Mapped[datetime] = mapped_column(DateTime,
default=datetime.utcnow)

    run = relationship("RecommendationRun", back_populates="report")

```

Листинг А.9 – Файл «backend/app/services/auth.py – хеширование паролей и JWT»

					ПГУ 1.090501.07.001 ПЗ	Лист
						166
Изм.	Лист	№ докум.	Подпись	Дата		

```

from __future__ import annotations

from datetime import datetime, timedelta

import jwt
from fastapi import HTTPException, status
from passlib.context import CryptContext
from sqlalchemy.orm import Session

from app.config.config import get_settings
from app.models.user import User

pwd_context = CryptContext(schemes=["pbkdf2_sha256"],
deprecatеd="auto")

def hash_password(password: str) -> str:
    return pwd_context.hash(password)

def verify_password(password: str, password_hash: str) -> bool:
    return pwd_context.verify(password, password_hash)

def create_access_token(user: User) -> str:
    settings = get_settings()
    payload = {
        "sub": str(user.id),
        "email": user.email,
        "is_admin": user.is_admin,
        "exp": datetime.utcnow() +
timedelta(minutes=settings.jwt_expire_minutes),
    }
    return jwt.encode(payload, settings.jwt_secret, algorithm="HS256")

def decode_access_token(token: str) -> dict[str, object]:
    settings = get_settings()
    try:
        return jwt.decode(token, settings.jwt_secret,
algorithms=["HS256"])
    except jwt.PyJWTError as exc:
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED,
detail="Invalid token") from exc

def authenticate_user(db: Session, email: str, password: str) -> User:
    user = db.query(User).filter(User.email == email).one_or_none()
    if user is None or user.deleted_at is not None or not
verify_password(password, user.password_hash):
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED,
detail="Invalid credentials")
    return user

```

Листинг А.10 – Файл «backend/app/routes/deps.py – проверка авторизации»

					ПГУ 1.090501.07.001 ПЗ	Лист
						167
Изм.	Лист	№ докум.	Подпись	Дата		

```

from typing import Annotated, Optional

from fastapi import Depends, Header, HTTPException, status
from sqlalchemy.orm import Session

from app.db.session import get_db
from app.models.user import User
from app.services.auth import decode_access_token

def get_current_user(
    authorization: Annotated[Optional[str], Header()] = None,
    db: Session = Depends(get_db),
) -> User:
    if authorization is None or not authorization.startswith("Bearer
"):
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED,
detail="Missing token")
        token = authorization.split(" ", maxsplit=1)[1]
        payload = decode_access_token(token)
        user = db.query(User).filter(User.id ==
int(payload["sub"])).one_or_none()
        if user is None or user.deleted_at is not None:
            raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED,
detail="User not available")
        return user

def get_admin_user(current_user: User = Depends(get_current_user)) ->
User:
    if not current_user.is_admin:
        raise HTTPException(status_code=status.HTTP_403_FORBIDDEN,
detail="Admin access required")
    return current_user

```

Листинг А.11 – Файл «backend/app/routes/auth.py – маршруты регистрации и входа»

```

from fastapi import APIRouter, Depends, HTTPException, Response,
status
from sqlalchemy.orm import Session

from app.db.session import get_db
from app.models.user import User
from app.schemas.auth import AuthResponse, LoginRequest,
RegisterRequest, UserResponse
from app.services.auth import authenticate_user, create_access_token,
hash_password

router = APIRouter(prefix="/auth", tags=["auth"])

@router.post("/register", response_model=AuthResponse,
status_code=status.HTTP_201_CREATED)
def register(payload: RegisterRequest, db: Session = Depends(get_db))
-> AuthResponse:

```

```

        if db.query(User).filter(User.email ==
payload.email).one_or_none():
            raise HTTPException(status_code=status.HTTP_409_CONFLICT,
detail="Email already registered")
        if db.query(User).filter(User.nickname ==
payload.nickname).one_or_none():
            raise HTTPException(status_code=status.HTTP_409_CONFLICT,
detail="Nickname already registered")
        user = User(
            email=payload.email,
            nickname=payload.nickname,
            password_hash=hash_password(payload.password),
            gender=payload.gender,
            age_full_years=payload.age_full_years,
            is_admin=False,
        )
        db.add(user)
        db.commit()
        db.refresh(user)
        token = create_access_token(user)
        return AuthResponse(access_token=token,
user=_serialize_user(user))

@router.post("/login", response_model=AuthResponse)
def login(payload: LoginRequest, db: Session = Depends(get_db)) ->
AuthResponse:
    user = authenticate_user(db, payload.email, payload.password)
    return AuthResponse(access_token=create_access_token(user),
user=_serialize_user(user))

@router.post("/logout", status_code=status.HTTP_204_NO_CONTENT)
def logout() -> Response:
    return Response(status_code=status.HTTP_204_NO_CONTENT)

def _serialize_user(user: User) -> UserResponse:
    return UserResponse(
        id=user.id,
        email=user.email,
        nickname=user.nickname,
        gender=user.gender,
        age_full_years=user.age_full_years,
        is_admin=user.is_admin,
        password_hash=None,
    )

```

Листинг А.12 – Файл «backend/app/routes/tests.py – маршруты прохождения методик»

```

from datetime import datetime

from fastapi import APIRouter, Depends, HTTPException, status
from sqlalchemy.orm import Session

```

```

from app.routes.deps import get_current_user
from app.routes.profile import to_recommendation_response
from app.db.session import get_db
from app.models.testing import SubjectProfile, TestAttempt,
TestMethod, TestStatus
from app.models.user import User
from app.schemas.testing import (
    StartAttemptResponse,
    TestAttemptResponse,
    TestCatalogItem,
    TestMethodResponse,
    TestSubmitRequest,
    TestSubmitResponse,
)
from app.services.scoring import completion_progress,
generate_recommendation_for_user

router = APIRouter(prefix="/tests", tags=["tests"])

def validate_responses(method_type: str, payload: TestSubmitRequest)
-> None:
    if method_type == "rokeach_values":
        for response_item in payload.responses:
            values = [response_item.evaluation,
response_item.self_assessment, response_item.motivation]
            if any(value < 1 or value > 18 for value in values):
                raise HTTPException(
                    status_code=status.HTTP_422_UNPROCESSABLE_ENTITY,
                    detail="Для теста Рокича допустимы только ранги от
1 до 18.",
                )
            return

        for response_item in payload.responses:
            values = [response_item.evaluation,
response_item.self_assessment, response_item.motivation]
            if any(value < 0 or value > 10 for value in values):
                raise HTTPException(
                    status_code=status.HTTP_422_UNPROCESSABLE_ENTITY,
                    detail="Для этого теста допустимы значения только от 0
до 10.",
                )

@router.get("/catalog", response_model=list[TestCatalogItem])
def catalog(db: Session = Depends(get_db)) -> list[TestCatalogItem]:
    methods = db.query(TestMethod).order_by(TestMethod.id.asc()).all()
    return [TestCatalogItem.model_validate(method) for method in
methods]

@router.get("/{method_id}", response_model=TestMethodResponse)
def get_method(method_id: int, db: Session = Depends(get_db)) ->
TestMethodResponse:

```

```

        method = db.query(TestMethod).filter(TestMethod.id ==
method_id).one_or_none()
        if method is None:
            raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Method not found")
        return TestMethodResponse.model_validate(method)

@router.post("/{method_id}/start",
response_model=StartAttemptResponse)
def start_method(
    method_id: int,
    current_user: User = Depends(get_current_user),
    db: Session = Depends(get_db),
) -> StartAttemptResponse:
    method = db.query(TestMethod).filter(TestMethod.id ==
method_id).one_or_none()
    if method is None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Method not found")
    attempt = TestAttempt(user_id=current_user.id,
method_id=method_id, status=TestStatus.IN_PROGRESS.value)
    db.add(attempt)
    db.commit()
    db.refresh(attempt)
    return StartAttemptResponse(attempt_id=attempt.id,
status=attempt.status)

@router.post("/{method_id}/submit", response_model=TestSubmitResponse)
def submit_method(
    method_id: int,
    payload: TestSubmitRequest,
    current_user: User = Depends(get_current_user),
    db: Session = Depends(get_db),
) -> TestSubmitResponse:
    attempt = (
        db.query(TestAttempt)
        .filter(
            TestAttempt.id == payload.attempt_id,
            TestAttempt.user_id == current_user.id,
            TestAttempt.method_id == method_id,
        )
        .one_or_none()
    )
    if attempt is None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Attempt not found")

    method = db.query(TestMethod).filter(TestMethod.id ==
method_id).one()
    validate_responses(method.method_type, payload)
    db.query(SubjectProfile).filter(SubjectProfile.attempt_id ==
attempt.id).delete()
    for response_item in payload.responses:

```

```

        db.add(
            SubjectProfile(
                attempt_id=attempt.id,
                subject_name=response_item.subject_name,
                evaluation=response_item.evaluation,
                self_assessment=response_item.self_assessment,
                motivation=response_item.motivation,
            )
        )

    attempt.status = TestStatus.COMPLETED.value
    attempt.completed_at = datetime.utcnow()
    db.flush()
    db.refresh(attempt)

    completed, total = completion_progress(db, current_user.id)
    if total > 0 and completed == total:
        run = generate_recommendation_for_user(db, current_user)
        return TestSubmitResponse(status=attempt.status,
            recommendation=to_recommendation_response(run))
    db.commit()
    return TestSubmitResponse(status=attempt.status,
        recommendation=None)

@router.get("/attempts", response_model=list[TestAttemptResponse])
def attempts(
    current_user: User = Depends(get_current_user),
    db: Session = Depends(get_db),
) -> list[TestAttemptResponse]:
    attempts_list = (
        db.query(TestAttempt)
        .filter(TestAttempt.user_id == current_user.id)
        .order_by(TestAttempt.started_at.desc(),
            TestAttempt.id.desc())
        .all()
    )
    return [TestAttemptResponse.model_validate(attempt) for attempt in
        attempts_list]

```

Листинг А.13 – Файл «backend/app/routes/admin.py – административные маршруты»

```

from io import BytesIO

from fastapi import APIRouter, Depends, HTTPException, Response,
status
from sqlalchemy.orm import Session, joinedload

from app.routes.deps import get_admin_user
from app.db.session import get_db
from app.models.profession import Profession, ProfessionTrait
from app.models.recommendation import RecommendationItem
from app.models.testing import TestMethod, TestOption, TestQuestion

```

```

from app.models.user import User
from app.schemas.admin import AdminUserResponse,
ProfessionAdminResponse, ProfessionCreateRequest,
ProfessionTraitInput, StatsResponse, TestMethodUpsertRequest
from app.schemas.testing import TestMethodResponse,
TestOptionResponse, TestQuestionResponse
from app.services.reports import render_report_pdf
from app.services.scoring import latest_recommendation
from app.services.stats import build_stats_response

router = APIRouter(prefix="/admin", tags=["admin"])

@router.get("/users", response_model=list[AdminUserResponse])
def list_users(_: User = Depends(get_admin_user), db: Session =
Depends(get_db)) -> list[AdminUserResponse]:
    users =
db.query(User).filter(User.deleted_at.is_(None)).order_by(User.is_admin.desc(), User.nickname.asc(), User.email.asc()).all()
    return [AdminUserResponse(id=user.id, email=user.email,
nickname=user.nickname, gender=user.gender,
age_full_years=user.age_full_years, is_admin=user.is_admin) for user
in users]

@router.get("/users/{user_id}/report")
def download_user_report(user_id: int, _: User =
Depends(get_admin_user), db: Session = Depends(get_db)) -> Response:
    user = db.query(User).filter(User.id == user_id,
User.deleted_at.is_(None)).one_or_none()
    if user is None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="User not found")
    run = latest_recommendation(db, user_id)
    if run is None or run.report is None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Report not found")
    return Response(
        content=run.report.content,
        media_type="text/plain; charset=utf-8",
        headers={"Content-Disposition": f'attachment; filename="user-
{user_id}-report.txt"'},
    )

@router.get("/users/{user_id}/report/pdf")
def download_user_report_pdf(user_id: int, _: User =
Depends(get_admin_user), db: Session = Depends(get_db)) -> Response:
    user = db.query(User).filter(User.id == user_id,
User.deleted_at.is_(None)).one_or_none()
    if user is None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="User not found")
    run = latest_recommendation(db, user_id)
    if run is None or run.report is None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,

```

```

detail="Report not found")
pdf_buffer = BytesIO()
render_report_pdf(run.report.content, pdf_buffer)
return Response(
    content=pdf_buffer.getvalue(),
    media_type="application/pdf",
    headers={"Content-Disposition": f'attachment; filename="user-
{user_id}-report.pdf"'},
)

@router.delete("/users/{user_id}",
status_code=status.HTTP_204_NO_CONTENT)
def delete_user(user_id: int, current_admin: User =
Depends(get_admin_user), db: Session = Depends(get_db)) -> Response:
    if current_admin.id == user_id:
        raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST,
detail="Admin cannot delete own account")
    user = db.query(User).filter(User.id == user_id).one_or_none()
    if user is None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="User not found")
    db.delete(user)
    db.commit()
    return Response(status_code=status.HTTP_204_NO_CONTENT)

@router.get("/professions",
response_model=list[ProfessionAdminResponse])
def list_professions(_: User = Depends(get_admin_user), db: Session =
Depends(get_db)) -> list[ProfessionAdminResponse]:
    professions =
db.query(Profession).options(joinedload(Profession.traits)).order_by(P
rofession.name.asc()).all()
    return [_serialize_profession(profession) for profession in
professions]

@router.post("/professions", response_model=ProfessionAdminResponse,
status_code=status.HTTP_201_CREATED)
def create_profession(payload: ProfessionCreateRequest, _: User =
Depends(get_admin_user), db: Session = Depends(get_db)) ->
ProfessionAdminResponse:
    if db.query(Profession).filter(Profession.name ==
payload.name).one_or_none():
        raise HTTPException(status_code=status.HTTP_409_CONFLICT,
detail="Profession already exists")
    profession = Profession(
        name=payload.name,
        description=payload.description,
        pros=payload.pros,
        cons=payload.cons,
        income_profile=payload.income_profile,
        popularity_in_russia=payload.popularity_in_russia,
        directions=payload.directions,
        difficulties=payload.difficulties,

```

```

    )
    db.add(profession)
    db.flush()
    for trait in payload.traits:
        db.add(ProfessionTrait(profession_id=profession.id,
            trait_key=trait.trait_key, desired_score=trait.desired_score,
            weight=trait.weight))
        db.commit()
    profession =
db.query(Profession).options(joinedload(Profession.traits)).filter(Pro
fession.id == profession.id).one()
    return _serialize_profession(profession)

@router.put("/professions/{profession_id}",
response_model=ProfessionAdminResponse)
def update_profession(profession_id: int, payload:
ProfessionCreateRequest, _: User = Depends(get_admin_user), db:
Session = Depends(get_db)) -> ProfessionAdminResponse:
    profession = db.query(Profession).filter(Profession.id ==
profession_id).one_or_none()
    if profession is None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Profession not found")
    profession.name = payload.name
    profession.description = payload.description
    profession.pros = payload.pros
    profession.cons = payload.cons
    profession.income_profile = payload.income_profile
    profession.popularity_in_russia = payload.popularity_in_russia
    profession.directions = payload.directions
    profession.difficulties = payload.difficulties
    db.query(ProfessionTrait).filter(ProfessionTrait.profession_id ==
profession_id).delete()
    for trait in payload.traits:
        db.add(ProfessionTrait(profession_id=profession_id,
            trait_key=trait.trait_key, desired_score=trait.desired_score,
            weight=trait.weight))
        db.commit()
    profession =
db.query(Profession).options(joinedload(Profession.traits)).filter(Pro
fession.id == profession_id).one()
    return _serialize_profession(profession)

@router.delete("/professions/{profession_id}",
status_code=status.HTTP_204_NO_CONTENT)
def delete_profession(profession_id: int, _: User =
Depends(get_admin_user), db: Session = Depends(get_db)) -> Response:
    profession = db.query(Profession).filter(Profession.id ==
profession_id).one_or_none()
    if profession is None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Profession not found")

```

```

db.query(RecommendationItem).filter(RecommendationItem.profession_id
== profession_id).delete()
    db.delete(profession)
    db.commit()
    return Response(status_code=status.HTTP_204_NO_CONTENT)

@router.get("/methods", response_model=list[TestMethodResponse])
def list_methods(_: User = Depends(get_admin_user), db: Session =
Depends(get_db)) -> list[TestMethodResponse]:
    methods =
db.query(TestMethod).options(joinedload(TestMethod.questions).joinedlo
ad(TestQuestion.options)).order_by(TestMethod.id.asc()).all()
    return [_serialize_method(method) for method in methods]

@router.post("/methods", response_model=TestMethodResponse,
status_code=status.HTTP_201_CREATED)
def create_method(payload: TestMethodUpsertRequest, _: User =
Depends(get_admin_user), db: Session = Depends(get_db)) ->
TestMethodResponse:
    raise HTTPException(
        status_code=status.HTTP_403_FORBIDDEN,
        detail="В демо-версии редактирование каталога тестов
отключено. Системные методики доступны только для просмотра.",
    )

@router.put("/methods/{method_id}", response_model=TestMethodResponse)
def update_method(method_id: int, payload: TestMethodUpsertRequest, _:
User = Depends(get_admin_user), db: Session = Depends(get_db)) ->
TestMethodResponse:
    raise HTTPException(
        status_code=status.HTTP_403_FORBIDDEN,
        detail="В демо-версии редактирование каталога тестов
отключено. Системные методики доступны только для просмотра.",
    )

@router.delete("/methods/{method_id}",
status_code=status.HTTP_204_NO_CONTENT)
def delete_method(method_id: int, _: User = Depends(get_admin_user),
db: Session = Depends(get_db)) -> Response:
    raise HTTPException(
        status_code=status.HTTP_403_FORBIDDEN,
        detail="В демо-версии редактирование каталога тестов
отключено. Системные методики доступны только для просмотра.",
    )

@router.get("/stats", response_model=StatsResponse)
def stats(_: User = Depends(get_admin_user), db: Session =
Depends(get_db)) -> StatsResponse:
    return build_stats_response(db)

def _serialize_profession(profession: Profession) ->
ProfessionAdminResponse:
    return ProfessionAdminResponse(

```

```

        id=profession.id,
        name=profession.name,
        description=profession.description,
        pros=profession.pros,
        cons=profession.cons,
        income_profile=profession.income_profile,
        popularity_in_russia=profession.popularity_in_russia,
        directions=profession.directions,
        difficulties=profession.difficulties,
        traits=[ProfessionTraitInput(trait_key=trait.trait_key,
desired_score=trait.desired_score, weight=trait.weight) for trait in
sorted(profession.traits, key=lambda current: current.trait_key)],
    )

def _replace_method_questions(method: TestMethod, payload:
TestMethodUpsertRequest, db: Session) -> None:
    existing_questions =
db.query(TestQuestion).filter(TestQuestion.method_id ==
method.id).all()
    for question in existing_questions:
        db.delete(question)
    db.flush()

    for question_input in sorted(payload.questions, key=lambda
current: current.position):
        question = TestQuestion(
            method_id=method.id,
            prompt=question_input.prompt,
            position=question_input.position,
            question_type=question_input.question_type,
        )
        db.add(question)
        db.flush()

        for option_input in question_input.options:
            db.add(
                TestOption(
                    question_id=question.id,
                    label=option_input.label,
                    value=option_input.value,
                    score=option_input.score,
                )
            )

def _serialize_method(method: TestMethod) -> TestMethodResponse:
    return TestMethodResponse(
        id=method.id,
        name=method.name,
        description=method.description,
        method_type=method.method_type,
        status=method.status,
        has_standard_keys=method.has_standard_keys,
        has_validity_scale=method.has_validity_scale,

```

```

questions=[
    TestQuestionResponse(
        id=question.id,
        prompt=question.prompt,
        position=question.position,
        question_type=question.question_type,
        options=[
            TestOptionResponse(
                id=option.id,
                label=option.label,
                value=option.value,
                score=option.score,
            )
            for option in question.options
        ],
    )
    for question in sorted(method.questions, key=lambda
current: current.position)
],
)

```

Листинг А.14 – Файл «backend/app/services/scoring.py – модуль расчёта рекомендации»

```

from __future__ import annotations

from collections import defaultdict
from datetime import datetime
from typing import Optional

from sqlalchemy.orm import Session, joinedload

from app.models.profession import Profession
from app.models.recommendation import RecommendationItem,
RecommendationRun, ReportFile
from app.models.testing import SubjectProfile, TestAttempt,
TestMethod, TestStatus
from app.models.user import User, UserTraitScore
from app.services.llm import build_explanation
from app.services.psychometrics import (
    BIG_FIVE_TO_SCALE,
    KLIMOV_KEY,
    KLIMOV_TYPE_TO_SCALE,
    PROFESSION_PROFILES,
    PROFESSION_PROFILE_ALIASES,
    RIASEC_TYPE_TO_SCALE,
    ROKEACH_GROUP_TO_SCALE,
    ROKEACH_INSTRUMENTAL_GROUPS,
    ROKEACH_TERMINAL_GROUPS,
    SCALES,
    SCHOOL_SUBJECT_SCALE_WEIGHTS,
    TEST_WEIGHTS,
)

```

```

blend_vectors,
cosine_similarity,
empty_vector,
normalize_vector,
top_scales,
)
from app.services.reports import render_report_text

def build_honesty_warning(subjects: list[SubjectProfile]) ->
Optional[str]:
    if not subjects:
        return None
    discrepancy = sum(abs(item.evaluation - item.self_assessment) for
item in subjects) / len(subjects)
    motivation = sum(item.motivation for item in subjects) /
len(subjects)
    if discrepancy >= 4 or motivation <= 2:
        return "Обнаружены признаки искажения ответов. Рекомендуется
перепройти стартовый тест."
    return None

def subject_vector(subjects: list[SubjectProfile]) -> dict[str,
float]:
    accumulator = empty_vector()
    weights = defaultdict(float)
    for subject in subjects:
        subject_weight_map =
SCHOOL_SUBJECT_SCALE_WEIGHTS.get(subject.subject_name)
        if not subject_weight_map:
            continue
        motivation = max(0.0, min(subject.motivation / 10.0, 1.0))
        for scale, weight in subject_weight_map.items():
            accumulator[scale] += motivation * weight
            weights[scale] += weight

    normalized = {
        scale: (accumulator[scale] / weights[scale]) if
weights.get(scale) else 0.0
        for scale in SCALES
    }
    return normalize_vector(normalized)

def parse_klimov_answer(score: float) -> str:
    return "A" if int(round(score)) <= 1 else "B"

def klimov_vector(subjects: list[SubjectProfile]) -> dict[str, float]:
    counts = defaultdict(float)
    ordered_subjects = sorted(subjects, key=lambda current:
current.id)
    for index, subject in enumerate(ordered_subjects, start=1):
        if index not in KLIMOV_KEY:
            continue
        answer = parse_klimov_answer(subject.evaluation)

```

```

        counts[KLIMOV_KEY[index][answer]] += 1.0

total = sum(counts.values())
if total <= 0:
    return empty_vector()

weighted_vectors = [
    (count / total, KLIMOV_TYPE_TO_SCALE[klimov_type])
    for klimov_type, count in counts.items()
]
return normalize_vector(blend_vectors(weighted_vectors))

def riasec_vector(subjects: list[SubjectProfile]) -> dict[str, float]:
    ordered_subjects = sorted(subjects, key=lambda current:
current.id)
    blocks = {
        "R": ordered_subjects[0:5],
        "I": ordered_subjects[5:10],
        "A": ordered_subjects[10:15],
        "S": ordered_subjects[15:20],
        "E": ordered_subjects[20:25],
        "C": ordered_subjects[25:30],
    }
    weighted_vectors = []
    for riasec_type, answers in blocks.items():
        if not answers:
            continue
        block_score = sum(answer.evaluation for answer in answers) /
20.0
        weighted_vectors.append((block_score,
RIASEC_TYPE_TO_SCALE[riasec_type]))
    return normalize_vector(blend_vectors(weighted_vectors))

def big_five_vector(subjects: list[SubjectProfile]) -> dict[str,
float]:
    ordered_subjects = sorted(subjects, key=lambda current:
current.id)
    blocks = {
        "E": ordered_subjects[0:5],
        "A": ordered_subjects[5:10],
        "C": ordered_subjects[10:15],
        "N": ordered_subjects[15:20],
        "O": ordered_subjects[20:25],
    }
    weighted_vectors = []
    for factor, answers in blocks.items():
        if not answers:
            continue
        factor_score = sum(answer.evaluation for answer in answers) /
25.0
        weighted_vectors.append((factor_score,
BIG_FIVE_TO_SCALE[factor]))
    return normalize_vector(blend_vectors(weighted_vectors))

```

```

def rokeach_value_priority(rank: float) -> float:
    bounded = max(1.0, min(rank, 18.0))
    return (19.0 - bounded) / 18.0

def canonical_rokeach_name(name: str) -> str:
    return name.split("(", 1)[0].strip()

def rokeach_vector(subjects: list[SubjectProfile]) -> dict[str, float]:
    priorities = {canonical_rokeach_name(subject.subject_name):
rokeach_value_priority(subject.evaluation) for subject in subjects}
    weighted_vectors = []

    for group_name, group_values in (**ROKEACH_TERMINAL_GROUPS,
**ROKEACH_INSTRUMENTAL_GROUPS).items():
        group_scores = [priorities[value] for value in group_values if
value in priorities]
        if not group_scores:
            continue
        weighted_vectors.append((sum(group_scores) /
len(group_scores), ROKEACH_GROUP_TO_SCALE[group_name]))

    return normalize_vector(blend_vectors(weighted_vectors))

def latest_completed_attempts(db: Session, user_id: int) -> dict[str,
TestAttempt]:
    attempts = (
        db.query(TestAttempt)
        .join(TestMethod, TestMethod.id == TestAttempt.method_id)
        .options(joinedload(TestAttempt.subject_profiles),
joinedload(TestAttempt.method))
        .filter(TestAttempt.user_id == user_id, TestAttempt.status ==
TestStatus.COMPLETED.value)
        .order_by(TestAttempt.completed_at.desc(),
TestAttempt.id.desc())
        .all()
    )
    latest: dict[str, TestAttempt] = {}
    for attempt in attempts:
        if attempt.method.method_type not in latest:
            latest[attempt.method.method_type] = attempt
    return latest

def completion_progress(db: Session, user_id: int) -> tuple[int, int]:
    methods = db.query(TestMethod).order_by(TestMethod.id.asc()).all()
    attempts = (
        db.query(TestAttempt)
        .filter(TestAttempt.user_id == user_id)
        .order_by(TestAttempt.started_at.desc(),
TestAttempt.id.desc())
        .all()
    )

```

```

latest_attempt_by_method: dict[int, TestAttempt] = {}
for attempt in attempts:
    if attempt.method_id not in latest_attempt_by_method:
        latest_attempt_by_method[attempt.method_id] = attempt
    completed = sum(1 for attempt in latest_attempt_by_method.values()
if attempt.status == TestStatus.COMPLETED.value)
    return completed, len(methods)

def compute_test_vectors(attempts_by_type: dict[str, TestAttempt]) ->
tuple[dict[str, dict[str, float]], Optional[str]]:
    vectors: dict[str, dict[str, float]] = {}
    honesty_warning: Optional[str] = None

    subject_attempt = attempts_by_type.get("subject_starter")
    if subject_attempt:
        vectors["subject_starter"] =
subject_vector(list(subject_attempt.subject_profiles))
        honesty_warning =
build_honesty_warning(list(subject_attempt.subject_profiles))

    klimov_attempt = attempts_by_type.get("klimov_ddo")
    if klimov_attempt:
        vectors["klimov_ddo"] =
klimov_vector(list(klimov_attempt.subject_profiles))

    holland_attempt = attempts_by_type.get("holland_riasec")
    if holland_attempt:
        vectors["holland_riasec"] =
riasec_vector(list(holland_attempt.subject_profiles))

    big_five_attempt = attempts_by_type.get("big_five")
    if big_five_attempt:
        vectors["big_five"] =
big_five_vector(list(big_five_attempt.subject_profiles))

    rokeach_attempt = attempts_by_type.get("rokeach_values")
    if rokeach_attempt:
        vectors["rokeach_values"] =
rokeach_vector(list(rokeach_attempt.subject_profiles))

    return vectors, honesty_warning

def aggregate_user_vector(test_vectors: dict[str, dict[str, float]])
-> dict[str, float]:
    weighted_vectors = [
        (TEST_WEIGHTS[method_type], vector)
        for method_type, vector in test_vectors.items()
        if method_type in TEST_WEIGHTS
    ]
    return normalize_vector(blend_vectors(weighted_vectors))

def profile_for_profession(profession: Profession) ->
Optional[dict[str, float]]:

```

```

        source_name = PROFESSION_PROFILE_ALIASES.get(profession.name,
profession.name)
        profile = PROFESSION_PROFILES.get(source_name)
        if profile is None:
            return None
        return {scale: profile.get(scale, 0.0) for scale in SCALES}

def summarize_match(user_vector: dict[str, float], profession_vector:
dict[str, float]) -> str:
    matches = []
    for scale in SCALES:
        closeness = 1 - abs(user_vector.get(scale, 0.0) -
profession_vector.get(scale, 0.0))
        matches.append((closeness, scale))
    strongest = [scale for _score, scale in sorted(matches,
reverse=True)[:3]]
    return "Ключевые совпадения: " + ", ".join(strongest)

def score_professions(db: Session, user_vector: dict[str, float]) ->
list[tuple[Profession, float, str]]:
    professions =
db.query(Profession).order_by(Profession.name.asc()).all()
    ranked: list[tuple[Profession, float, str]] = []
    for profession in professions:
        profession_vector = profile_for_profession(profession)
        if profession_vector is None:
            continue
        similarity = cosine_similarity(user_vector, profession_vector)
        score = round(similarity * 100, 2)
        ranked.append((profession, score, summarize_match(user_vector,
profession_vector)))
    ranked.sort(key=lambda item: (-item[1], item[0].name))
    return ranked

def generate_recommendation_for_user(db: Session, user: User) ->
RecommendationRun:
    attempts_by_type = latest_completed_attempts(db, user.id)
    if not attempts_by_type:
        raise ValueError("No completed attempts")

    test_vectors, honesty_warning =
compute_test_vectors(attempts_by_type)
    user_vector = aggregate_user_vector(test_vectors)
    anchor_attempt = sorted(
        attempts_by_type.values(),
        key=lambda attempt: (attempt.completed_at or datetime.min,
attempt.id),
        reverse=True,
    )[0]

    db.query(UserTraitScore).filter(UserTraitScore.user_id ==
user.id).delete()
    for scale, score in user_vector.items():

```

```

        db.add(UserTraitScore(user_id=user.id, trait_key=scale,
score=score))

    ranked = score_professions(db, user_vector)
    llm_explanation = build_explanation(ranked[:5], honesty_warning)
    run = RecommendationRun(
        user_id=user.id,
        attempt_id=anchor_attempt.id,
        llm_explanation=llm_explanation,
        honesty_warning=honesty_warning,
    )
    db.add(run)
    db.flush()

    for profession, score, reasoning in ranked[:5]:
        db.add(RecommendationItem(run_id=run.id,
profession_id=profession.id, score=score,
reasoning_summary=reasoning))

    db.flush()
    report_text = render_report_text(
        user=user,
        run=run,
        items=ranked[:5],
        attempts_by_type=attempts_by_type,
        test_vectors=test_vectors,
    )
    db.add(ReportFile(run_id=run.id, content=report_text,
format="txt"))
    db.commit()
    db.refresh(run)
    return run

def latest_recommendation(db: Session, user_id: int) ->
Optional[RecommendationRun]:
    return (
        db.query(RecommendationRun)
        .options(joinedload(RecommendationRun.items).joinedload(Recomm
endationItem.profession), joinedload(RecommendationRun.report))
        .filter(RecommendationRun.user_id == user_id)
        .order_by(RecommendationRun.created_at.desc(),
RecommendationRun.id.desc())
        .first()
    )

def completion_stats(db: Session) -> dict[str, float]:
    attempts = db.query(TestAttempt).all()
    if not attempts:
        return {"started": 0, "completed": 0, "ratio": 0.0}
    started = len(attempts)
    completed = sum(1 for attempt in attempts if attempt.status ==
TestStatus.COMPLETED.value)
    return {"started": started, "completed": completed, "ratio":

```

```
round(completed / started, 2)}
```

Листинг А.15 – Файл «backend/app/services/reports.py – формирование отчётов»

```
from __future__ import annotations

from collections import defaultdict
from collections.abc import Sequence
from io import BytesIO
from pathlib import Path

from app.models.profession import Profession
from app.models.recommendation import RecommendationRun
from app.models.testing import TestAttempt
from app.models.user import User
from app.services.psychometrics import KLIMOV_KEY, top_scales

KLIMOV_LABELS = {
    "ч1": "человек-человек",
    "ч2": "человек-техника",
    "ч3": "человек-природа",
    "ч4": "человек-знаковая система",
    "ч5": "человек-художественный образ",
}

def _klimov_answer(score: float) -> str:
    return "А" if int(round(score)) <= 1 else "Б"

def _dominant_klimov_type(klimov_attempt: TestAttempt | None) -> str:
    if klimov_attempt is None:
        return "не определен"

    counts = defaultdict(int)
    ordered_answers = sorted(klimov_attempt.subject_profiles,
key=lambda current: current.id)
    for index, answer in enumerate(ordered_answers, start=1):
        if index not in KLIMOV_KEY:
            continue
        counts[KLIMOV_KEY[index][_klimov_answer(answer.evaluation)]]
+= 1

    if not counts:
        return "не определен"

    dominant = sorted(counts.items(), key=lambda item: (-item[1],
item[0]))[0][0]
    return KLIMOV_LABELS.get(dominant, "не определен")

def _leading_scale_name(vector: dict[str, float]) -> str:
    top = top_scales(vector, limit=1)
    return top[0][0] if top else "не определена"
```

					ПГУ 1.090501.07.001 ПЗ	Лист
						185
Изм.	Лист	№ докум.	Подпись	Дата		

```

def _describe_personality(test_vectors: dict[str, dict[str, float]])
-> str:
    parts: list[str] = []

    riasec_vector = test_vectors.get("holland_riasec")
    if riasec_vector:
        directions = ", ".join(scale.lower() for scale, _ in
top_scales(riasec_vector, limit=2))
        parts.append(f"В профессиональных интересах заметна ориентация
на направления: {directions}.")

    big_five_vector = test_vectors.get("big_five")
    if big_five_vector:
        traits = ", ".join(scale.lower() for scale, _ in
top_scales(big_five_vector, limit=2))
        parts.append(f"В работе наиболее комфортна среда, где ценятся
{traits}.")

    rokeach_vector = test_vectors.get("rokeach_values")
    if rokeach_vector:
        values = ", ".join(scale.lower() for scale, _ in
top_scales(rokeach_vector, limit=2))
        parts.append(f"В выборе деятельности важны {values}.")

    return " ".join(parts) if parts else "Профиль формируется на
основе сочетания интересов, личностных особенностей и жизненных
приоритетов."

def render_report_text(
    *,
    user: User,
    run: RecommendationRun,
    items: Sequence[tuple[Profession, float, str]],
    attempts_by_type: dict[str, TestAttempt],
    test_vectors: dict[str, dict[str, float]],
) -> str:
    klimov_type =
_dominant_klimov_type(attempts_by_type.get("klimov_ddo"))
    motivation_scale =
_leading_scale_name(test_vectors.get("subject_starter", {}))
    personality_summary = _describe_personality(test_vectors)

    lines = [
        "Отчет по профессиональному профилю",
        "",
        f"Имя пользователя: {user.nickname}",
        "Краткий итог: На основе пройденного тестирования сформирован
следующий профессиональный профиль.",
        "",
        "Итоговый профиль",
        f"Тип по Климову: {klimov_type}.",
        f"Ведущая мотивационная шкала: {motivation_scale}.",
        personality_summary,
    ]

```

```

        "",
        "Рекомендации по профессиям",
    ]

    for profession, _score, _reasoning in items:
        lines.extend([profession.name, profession.description, ""])

    if run.honesty_warning:
        lines.extend(["", f"Примечание: {run.honesty_warning}"])
    return "\n".join(lines)

def _resolve_pdf_font_path() -> str:
    candidates = [
        Path("C:/Windows/Fonts/arial.ttf"),
        Path("C:/Windows/Fonts/arialuni.ttf"),
        Path("C:/Windows/Fonts/tahoma.ttf"),
        Path("C:/Windows/Fonts/calibri.ttf"),
    ]
    for candidate in candidates:
        if candidate.exists():
            return str(candidate)
    raise FileNotFoundError("No suitable TrueType font found for PDF
export")

def render_report_pdf(content: str, target: BytesIO) -> None:
    from reportlab.lib.pagesizes import A4
    from reportlab.pdfbase import pdfmetrics
    from reportlab.pdfbase.ttfonts import TTFont
    from reportlab.pdfgen import canvas

    font_name = "PsychodiagnosticsFont"
    font_path = _resolve_pdf_font_path()
    if font_name not in pdfmetrics.getRegisteredFontNames():
        pdfmetrics.registerFont(TTFont(font_name, font_path))

    pdf = canvas.Canvas(target, pagesize=A4)
    width, height = A4
    pdf.setTitle("Psychodiagnostics Report")

    text = pdf.beginText(48, height - 56)
    text.setFont(font_name, 11)
    text.setLeading(16)

    for raw_line in content.splitlines():
        line = raw_line.strip()
        if not line:
            text.textLine("")
            continue

        current = line
        while current:
            chunk = current[:95]
            current = current[95:]

```

```

        text.textLine(chunk)
        if text.getY() < 56:
            pdf.drawText(text)
            pdf.showPage()
            text = pdf.beginText(48, height - 56)
            text.setFont(font_name, 11)
            text.setLeading(16)

pdf.drawText(text)
pdf.save()

```

Листинг А.16 – Файл «frontend/app/layout.tsx – корневой шаблон клиентской части»

```

import "../styles/globals.css";
import type { Metadata } from "next";

export const metadata: Metadata = {
  title: "Psychodiagnostics Workspace",
  description: "Автоматизированная система психодиагностики для задач профотбора",
};

export default function RootLayout({
  children,
}: Readonly<{ children: React.ReactNode }>) {
  return (
    <html lang="ru">
      <body>{children}</body>
    </html>
  );
}

```

Листинг А.17 – Файл «frontend/app/page.tsx – главная страница»

```

"use client";

import { useEffect, useState } from "react";

import { AuthShell } from "@components/auth/auth-shell";
import { apiRequest, clearToken, loadToken } from "@services/api-client";
import { DashboardPayload } from "@services/api-types";

export default function HomePage() {
  const [isReady, setIsReady] = useState(false);

  useEffect(() => {
    const token = loadToken();
    if (!token) {
      setIsReady(true);
      return;
    }
  });
}

```

```

    }

    apiRequest<DashboardPayload>("/api/me", {}, token)
      .then(() => {
        window.location.replace("/dashboard");
      })
      .catch(() => {
        clearToken();
        setIsReady(true);
      });
  }, []);

  if (!isReady) {
    return null;
  }

  return <AuthShell onAuthenticated={() =>
    window.location.replace("/dashboard")} />;
}

```

Листинг А.18 – Файл «frontend/services/api-client.ts – слой работы с API»

```

const API_BASE_URL = process.env.NEXT_PUBLIC_API_BASE_URL ?? "";

function buildUrl(path: string) {
  if (path.startsWith("http://") || path.startsWith("https://")) {
    return path;
  }
  return `${API_BASE_URL}${path}`;
}

function extractErrorMessage(payload: unknown): string {
  if (typeof payload === "string") {
    return payload;
  }
  if (payload && typeof payload === "object" && "detail" in payload) {
    const detail = (payload as { detail: unknown }).detail;
    if (typeof detail === "string") {
      return detail;
    }
    if (Array.isArray(detail) && detail.length > 0) {
      const first = detail[0] as { msg?: string } | undefined;
      if (first?.msg) {
        return first.msg;
      }
    }
  }
  return "Request failed";
}

async function parseResponse<T>(response: Response): Promise<T> {
  if (!response.ok) {

```

```

        const contentType = response.headers.get("content-type") ?? "";
        if (contentType.includes("application/json")) {
            const payload = await response.json();
            throw new Error(extractErrorMessage(payload));
        }
        const errorText = await response.text();
        throw new Error(errorText || "Request failed");
    }
    if (response.status === 204 || typeof response.json !== "function")
    {
        return undefined as T;
    }
    return response.json() as Promise<T>;
}

export async function apiRequest<T>(
    path: string,
    options: RequestInit = {},
    token?: string,
): Promise<T> {
    const headers = new Headers(options.headers ?? {});
    if (!headers.has("Content-Type") && options.body) {
        headers.set("Content-Type", "application/json");
    }
    if (token) {
        headers.set("Authorization", `Bearer ${token}`);
    }
    const response = await fetch(buildUrl(path), {
        ...options,
        headers,
    });
    return parseResponse<T>(response);
}

export async function apiTextRequest(path: string, options:
RequestInit = {}, token?: string): Promise<string> {
    const headers = new Headers(options.headers ?? {});
    if (token) {
        headers.set("Authorization", `Bearer ${token}`);
    }
    const response = await fetch(buildUrl(path), {
        ...options,
        headers,
    });
    if (!response.ok) {
        const contentType = response.headers.get("content-type") ?? "";
        if (contentType.includes("application/json")) {
            const payload = await response.json();
            throw new Error(extractErrorMessage(payload));
        }
        throw new Error((await response.text()) || "Request failed");
    }
    return response.text();
}

```

```

}

export function saveToken(token: string) {
  if (typeof window !== "undefined") {
    window.localStorage.setItem("psychodiagnostics-token", token);
  }
}

export function loadToken() {
  if (typeof window === "undefined") {
    return "";
  }
  return window.localStorage.getItem("psychodiagnostics-token") ?? "";
}

export function clearToken() {
  if (typeof window !== "undefined") {
    window.localStorage.removeItem("psychodiagnostics-token");
  }
}

```

Листинг А.19 – Файл «frontend/components/auth/login-form.tsx – форма входа»

```

"use client";

import React, { FormEvent, useState } from "react";

import { apiRequest } from "@services/api-client";

type LoginFormProps = {
  onSuccess?: (token: string) => void;
};

const EMAIL_MESSAGE = "Введите корректный email";
const PASSWORD_MESSAGE = "Введите не менее 8 символов";

function validateLogin(email: string, password: string) {
  if (!email.includes("@")) {
    return EMAIL_MESSAGE;
  }
  if (password.trim().length < 8) {
    return PASSWORD_MESSAGE;
  }
  return "";
}

export function LoginForm({ onSuccess }: LoginFormProps) {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState("");
  const [isSubmitting, setIsSubmitting] = useState(false);

```

```

async function handleSubmit(event: FormEvent<HTMLFormElement>) {
  event.preventDefault();
  const validationError = validateLogin(email, password);
  setError(validationError);
  if (validationError) {
    return;
  }

  setIsSubmitting(true);
  try {
    const result = await apiRequest<{ access_token:
string }>("/api/auth/login", {
      method: "POST",
      body: JSON.stringify({ email, password }),
    });
    onSuccess?.(result.access_token);
  } catch (requestError) {
    setError(requestError instanceof Error ? requestError.message :
"Не удалось выполнить вход.");
  } finally {
    setIsSubmitting(false);
  }
}

return (
  <form className="auth-form" onSubmit={handleSubmit}>
    <label className="field">
      <span>Email</span>
      <input aria-label="Email" type="email" value={email}
onChange={ (event) => setEmail(event.target.value)} />
    </label>
    <label className="field">
      <span>Пароль</span>
      <input
        aria-label="Пароль"
        type="password"
        value={password}
        onChange={ (event) => setPassword(event.target.value)}
      />
      <small className="field-hint">Введите не менее 8
символов</small>
    </label>
    {error ? <p className="warning-text">{error}</p> : null}
    <button type="submit" className="primary-button"
disabled={isSubmitting}>
      {isSubmitting ? "Выполняется вход..." : "Войти"}
    </button>
  </form>
);
}

```

Листинг А.20 – Файл «frontend/components/layout/app-sidebar.tsx – боковая навигация»

```
import React from "react";
import Link from "next/link";

type SidebarItem = "profile" | "testing" | "professions" | "users" |
"admin";

type AppSidebarProps = {
  activeItem: SidebarItem;
  isAdmin?: boolean;
};

const baseItems = [
  { id: "profile" as const, label: "Пользователь", href: "/dashboard",
  icon: UserIcon },
  { id: "testing" as const, label: "Тестирование", href: "/tests",
  icon: ClipboardIcon },
  { id: "professions" as const, label: "Профбаза", href:
"/professions", icon: DatabaseIcon },
];

export function AppSidebar({ activeItem, isAdmin = false }:
AppSidebarProps) {
  return (
    <aside className="app-sidebar">
      <div className="sidebar-brand">
        <span className="brand-dot" />
        <div>
          <strong>Рабочий кабинет системы</strong>
        </div>
      </div>
      <nav className="sidebar-nav" aria-label="Основная навигация">
        {baseItems.map((item) => (
          <Link
            key={item.id}
            href={item.href}
            className={activeItem === item.id ? "sidebar-link is-
active" : "sidebar-link"}
          >
            <item.icon />
            <span>{item.label}</span>
          </Link>
        ))}
        {isAdmin ? (
          <>
            <Link href="/admin/users" className={activeItem ===
"users" ? "sidebar-link is-active" : "sidebar-link"}>
              <UsersIcon />
              <span>Пользователи</span>
            </Link>
          </>
        ) : null}
      </nav>
    </aside>
  );
}
```

```

        <Link href="/admin" className={activeItem === "admin" ?
"sidebar-link is-active" : "sidebar-link"}>
            <ShieldIcon />
            <span>Администрирование</span>
        </Link>
    </>
    ) : null}
</nav>
</aside>
);
}

function iconPath(children: React.ReactNode) {
    return (
        <svg viewBox="0 0 24 24" fill="none" stroke="currentColor"
strokeWidth="1.8" aria-hidden="true">
            {children}
        </svg>
    );
}

function UserIcon() {
    return iconPath(
        <>
            <path d="M12 13a4 4 0 1 0 0-8 4 4 0 0 0 0 8Z" />
            <path d="M5 20a7 7 0 0 1 14 0" />
        </>,
    );
}

function UsersIcon() {
    return iconPath(
        <>
            <path d="M8.5 12a3 3 0 1 0 0-6 3 3 0 0 0 0 6Z" />
            <path d="M16.5 13a2.5 2.5 0 1 0 0-5 2.5 2.5 0 0 0 0 5Z" />
            <path d="M3.5 20a5 5 0 0 1 10 0" />
            <path d="M13.5 20a4 4 0 0 1 7 0" />
        </>,
    );
}

function ClipboardIcon() {
    return iconPath(
        <>
            <rect x="6" y="4" width="12" height="16" rx="2" />
            <path d="M9 4.5h6M9 9h6M9 13h6M9 17h4" />
        </>,
    );
}

function DatabaseIcon() {
    return iconPath(
        <>

```

```

        <ellipse cx="12" cy="6" rx="6.5" ry="2.5" />
        <path d="M5.5 6v6c0 1.4 2.9 2.5 6.5 2.5s6.5-1.1 6.5-2.5V6" />
        <path d="M5.5 12v6c0 1.4 2.9 2.5 6.5 2.5s6.5-1.1 6.5-2.5v-6" />
    </>,
);
}

function ShieldIcon() {
    return iconPath(
        <>
        <path d="M12 3 5.5 6v5.5c0 4.1 2.8 7.8 6.5 9.5 3.7-1.7 6.5-5.4
6.5-9.5V6L12 3Z" />
        <path d="m9.5 12 1.7 1.8 3.3-3.6" />
        </>,
    );
}

```

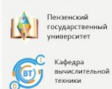
Приложение В

(обязательное)

Автоматизированная система психодиагностики для задач профотбора

Презентация

					ПГУ 1.090501.07.001 ПЗ	Лист
						196
Изм.	Лист	№ докум.	Подпись	Дата		



ДИПЛОМНЫЙ ПРОЕКТ

на тему

«Автоматизированная система психодиагностики для задач профотбора»

Руководитель работы
д.т.н., профессор
кафедры ВТ Гурин Е.И.

Выполнил: ст. гр. 21В031
Кокорин Артем
Александрович

Пенза 2026

Актуальность



«до 60 %» — выпускников работают не по специальности.

Субъективность
интерпретации
психолога

Разрозненность
диагностических
методик

Недостоверность
ответов в тестах



**Нужен бесплатный, объективный,
комплексный инструмент**



Цель и задачи



Разработка автоматизированной системы психодиагностики в виде удобного клиент-серверного веб приложения для задач профотбора



Анализ предметной области и аналогов



Выбор технологического стека



Проектирование архитектуры и базы данных



Реализация серверной и клиентской частей

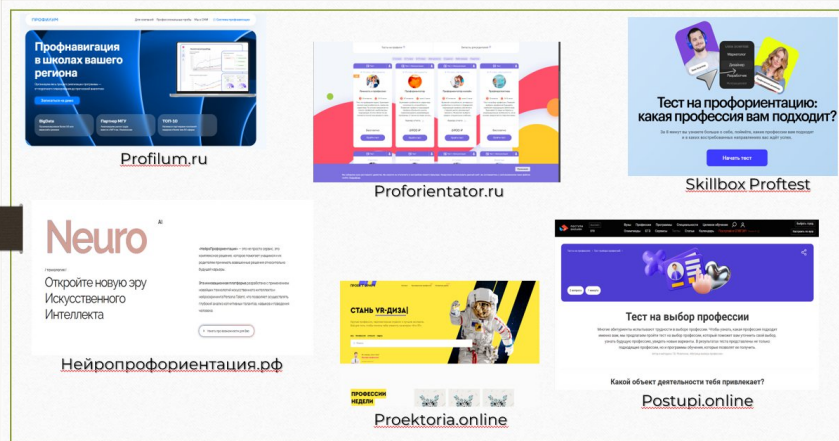


Реализация модуля расчета рекомендации с контролем достоверности



Обеспечение безопасности персональных данных

Анализ аналогов



Технологический стек

СЕРВЕРНАЯ

Серверная часть:

Python 3.11 ·
FastAPI ·
SQLAlchemy ·
MySQL-драйвер
PyMySQL ·
PyJWT ·
passlib (PBKDF2-
SHA256) ·



Клиентская часть:

TypeScript 5.7 ·
Next.js 15 ·
React 19 ·
CSS ·



База данных:

MySQL 8.x ·
MySQL
Workbench ·

Архитектура системы



Диаграмма развертывания системы

Компонентная архитектура системы

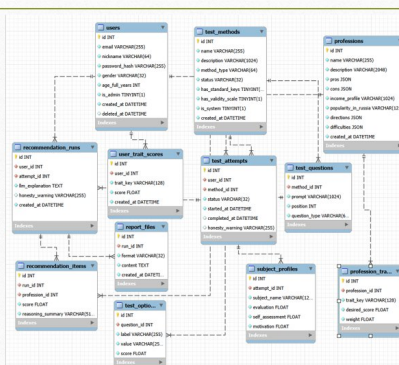
Методики

Методика	Тип данных	Весовой коэффициент
Тест 1. Школьные ориентиры	Три шкалы (оценка, самооценка, мотивация) по 18 предметам	0,15
Тест 2. ДДО Климova	Бинарный выбор из пары утверждений (20 пар)	0,25
Тест 3. Holland RIASEC	Согласие по шкале 0-4 (30 утверждений)	0,25
Тест 4. Big Five	Согласие по шкале 1-5 (25 утверждений)	0,20
Тест 5. Ценностные ориентации Рокича	Ранжирование 36 ценностей по приоритетности	0,15
Итого	Итоговое значение	1,00

Шкалы

№	Шкала	Что отражает
1	Люди	Склонность к работе с людьми, общению, обучению
2	Техника	Интерес к технике, машинам, механизмам, задачам
3	Знаки	Работа с символами, текстами, кодом, формулами
4	Творчество	Художественное самовыражение, создание новых образов
5	Природа	Интерес к живым организмам, экологии
6	Аналитика	Склонность к исследованию, анализу данных
7	Практика	Предпочтение физической, осязаемой деятельности
8	Структура	Потребность в порядке, правилах, чётких процедурах и документации
9	Управление	Склонность к организации, координации, принятию решений и руководству

База данных



ER-диаграмма базы данных (MySQL Workbench)

Охрана труда (БЖД)

ОХРАНА ТРУДА ПРИ РАБОТЕ ЗА КОМПЬЮТЕРОМ

ПРАВИЛЬНОЕ ПОЛОЖЕНИЕ ЗА СТОЛОМ

Взгляд на уровне глаз. Держите голову поднятой, даже если приходится смотреть вниз на экран.

Расстояние от глаза до монитора. Расстояние от глаза до монитора должно составлять не менее 50-70 см.

Спина. Сядьте прямо, лопатки опираются на спинку стула. Предплечья должны быть параллельны полу.

ОБЯЗАТЕЛЬНО СДЕЛАЙ ЗАРЯДКУ

1. Наклон головы по 3 подхода (3 сек.)
2. Наклон шеи по 3 подхода (10 сек.)
3. Поворот головы по 3 подхода (10 сек.)
4. Наклоны в стороны и вперед

ПОРА СДЕЛАТЬ ПЕРЕРЫВ

Не игнорируя время на перерыв. Делай его через каждые полчаса по 10-15 минут.

НЕ ЗАБЫВАЙ ПРО ГЛАЗА

Экономика

Блок 1 — Состав команды

Роль	Сумма
Backend-разработчик (Python, FastAPI)	220 000 руб/мес
Frontend-разработчик (TypeScript, React)	190 000 руб/мес
UX-специалист (частично)	160 000 руб/мес
Психолог-консультант (частично)	100 000 руб/мес
Менеджер проекта (частично)	180 000 руб/мес

Блок 3 — Окупаемость

Доход от рекламы и партнёров:

15 000 – 50 000
Р/мес

Доступ для пользователей остаётся бесплатным

Выход на окупаемость:

18 – 24 месяца

Блок 2 — Расчёт стоимости

Составляющая	Сумма
Фонд оплаты труда (ФОТ)	2 780 000 Р
Страховые взносы (30,2 %)	830 000 Р
Инфраструктура и лицензии	120 000 Р
Итого	~3,7 млн Р

Заключение

✓ Разработана

Автоматизированная система психодиагностики для задач профотбора

Перспективы

- Расширение каталога профессий
- Добавление новых методик
- Мобильное приложение
- Партнёрские интеграции для самоокупаемости

psychodiagnostics

Psychodiagnostics qr

Psychodiagnostics-сайт

Спасибо вам!