МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет вычислительной техники

Кафедра МО и ПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

Онлайн-сервис организации, проведения и обработки результатов психологического тестирования личности в образовательных учреждениях

Студент _____ Бирюков Владислав Евгеньевич
(подпись, дата) (ФИО полностью)

Руководитель _____ Балашова И.Ю.
(подпись, дата) (фамилия, инициалы)

Нормоконтролер _____ Такташкин Д.В
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____№_____)

Заведующий кафедрой _____ Макарычев П.П.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____№____)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2022

Федеральное государственное бюджетное образовательное учреждение высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
Вычислительной техники	МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____П.П. Макарычев
«__»_____ 2022 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Бирюков Владислав Евгеньевич гр. 18ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2.Тема работы Онлайн-сервис организации, проведения и обработки результатов психологического тестирования личности в образовательных учреждениях

Тема утверждена приказом ПГУ № 373/О от "18" апреля 2022 г.

3. Руководитель работы к.т.н., доцент кафедры «МО и ПЭВМ» Балашова И.Ю.

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

1) Назначение: сервис для организации, проведения и обработки результатов психологического тестирования личности в образовательных учреждениях

2) Среда выполнения: операционная система Ubuntu 20.04.

Языковые средства разработки: Java, SQL, UML.

Инструментальные средства: IntelliJI IDEA, PostgreSQL 12, DataGrip, Draw.io, Технологии разработки ООП, RAP-модель, сервисно-ориентированный подход.

3) Основные функции приложения:

а) авторизация;

б) просмотр списка психологических тестов;

в) создание и редактирование психологического теста;

г) администрирование тестируемых пользователей;

д) проведение психологического теста;

е) первичная обработка результатов психологического тестирования;

ж) просмотр результатов психологического теста.

Перечень подлежащих разработке вопросов:

1) *Анализ предметной области и постановка задачи;*
2) *Проектирование;*
3) *Реализация;*
4) *Контроль качества программного обеспечения;*
5) *План разработки бюджета;*
6) *Оформление пояснительной записки.*

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и постановка задачи	10%	13.04.2022 – 27.04.2022	
2	Проектирование	25%	27.04.2022 – 10.05.2022	
3	Реализация	35%	10.05.2022 – 30.06.2022	
4	Контроль качества программного обеспечения	10%	31.05.2022 – 02.06.2022	
5	План разработки бюджета	5%	02.06.2022 – 03.06.2022	
6	Оформление пояснительной записки	15%	03.06.2022 – 17.06.2022	

Дата выдачи задания «22» ____декабря____2021 г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

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

Реферат

Пояснительная записка содержит 79 листов, 33 рисунка, 21 таблица, 13 использованных источников и 2 приложения.

БАЗА ДАННЫХ, ИНФОРМАЦИОННАЯ СИСТЕМА, REST-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ, СУБД, ПРЕДМЕТНАЯ ОБЛАСТЬ, POSTGRESQL.

Объект исследования – сервисы организации, проведения и обработки результатов психологического тестирования личности в школьных учреждениях.

Цель выпускной квалификационной работы – разработать сервис с возможностью проведения, организации и обработки результатов психологического тестирования личности в школьных учреждениях.

Технологии разработки – разработка сервера приложений проводилась на языке программирования Java с использованием фреймворка Spring и ORM Hibernate, СУБД PostgreSQL 12. В качестве среды разработки использовался редактор программного кода IntelliJI IDEA и менеджер баз данных DataGrip.

Среда исполнения – Linux.

Результаты работы – в ходе выполнения выпускной квалификационной работы была создана REST-серверная часть сервиса проведения, организации и обработки результатов психологического тестирования личности для школьных учреждений.

Содержание

Введение 7

1 Анализ сервисов психологического тестирования 9

1.1 Анализ вариантов использования 9

1.2 Сравнительный анализ продуктов-аналогов 12

1.3 Постановка задачи 15

1.4 Анализ требований 16

2 Проектирование сервиса психологического тестирования 19

2.1 Моделирование бизнес-процессов предметной области 19

2.2 Разработка концептуальной модели базы данных 20

2.3 Моделирование базы данных 28

3 Реализация сервиса психологического тестирование 30

3.1 Развертывание и наполнение базы данных тестовыми данными 30

3.1.1 Развертывание базы данных в среде PostgreSQL с помощью средств ORM 30

3.1.2 Наполнение базы данных тестовыми данными с помощью средств ORM 31

3.2 Разработка серверной части приложения 34

3.2.1 Разработка запросов к базе данных и представлений для реализации аналитических бизнес-процессов при помощи ORM 34

3.2.2 Разработка функций для реализации учетных пользовательских бизнес-процессов на стороне REST-сервера при помощи ORM 37

3.3 Тестирование приложения 41

3.3.1 Выбор режима тестирования приложения 41

3.3.2 Тестирование REST-серверного приложения 41

3.4 Анализ качества кода 45

3.4.1 Измерение устойчивости 45

3.4.2 Измерение абстрактности 46

3.4.3 Главная последовательность 46

4 Планирование разработки и оценка бюджета сервиса психологического тестирования 49

Заключение 52

Список использованных источников 53

Приложение А. Глоссарий 55

Приложение Б. Листинг программы 58

Введение

Поддержание психического здоровья личности на разных этапах развития является одним из наиболее значимых направлений психологической практики. Традиционно в контексте психологического сопровождения значительно большее внимание уделяется детям школьного возраста [1].

В зону компетенций школьного психолога включены вопросы психологической диагностики, профилактики, просвещения и коррекции. Особое внимание уделяется диагностическим процедурам.

У школьного психолога в процессе работы возникает потребность самому готовить психологические тесты, для получение необходимой информации о тестируемых, поэтому ему требуется специальный инструмент, который позволит это организовать.

Психологическое тестирование – метод измерения и оценки психологических характеристик человека с помощью специальных техник. Предметом тестирования могут быть любые психологические характеристики человека: психические процессы, состояния, свойства, отношения и т. п. Основой психологического тестирования является психологический тест – стандартизированная система испытаний, позволяющая обнаружить и измерить качественные и количественные индивидуально-психологические различия [2].

Актуальность онлайн-сервиса организации, проведения и обработки результатов психологического тестирования личности в образовательных учреждениях обусловлено тем, что многие бесплатные сервисы-аналоги обладают рядом существенных недостатков, которые не позволяет им в полной мере удовлетворять потребности школьного психолога. К таким недостаткам относятся: отсутствие пользовательской базы, открытость результатов тестирования, закрытый интерфейс взаимодействия, отсутствие базы результатов тестирования

Целью выпускной квалификационной работы является разработка REST-серверного приложения для организации, проведения и обработки результатов психологического тестирования в образовательных учреждениях.

1 Анализ сервисов психологического тестирования

1.1 Анализ вариантов использования

Информационная система предназначена для сбора сведений о психологическом состоянии школьников, родителей и преподавателей посредством психологического тестирования. Также система направлена на упрощение работы психологам. С точки зрения психологов разрабатываемое приложение позволит упростить сбор и систематизацию информации.

Информационная система должна обеспечить пользователя следующими возможностями:

– формирование (создание/редактирование) психологических тестов;

– прохождение психологических тестов;

– получение результатов о прохождении психологических тестов;

– получение первичного анализа результатов прохождения психологических тестов.

На основе сформулированных задач создана диаграмма вариантов использования (рисунок 1).

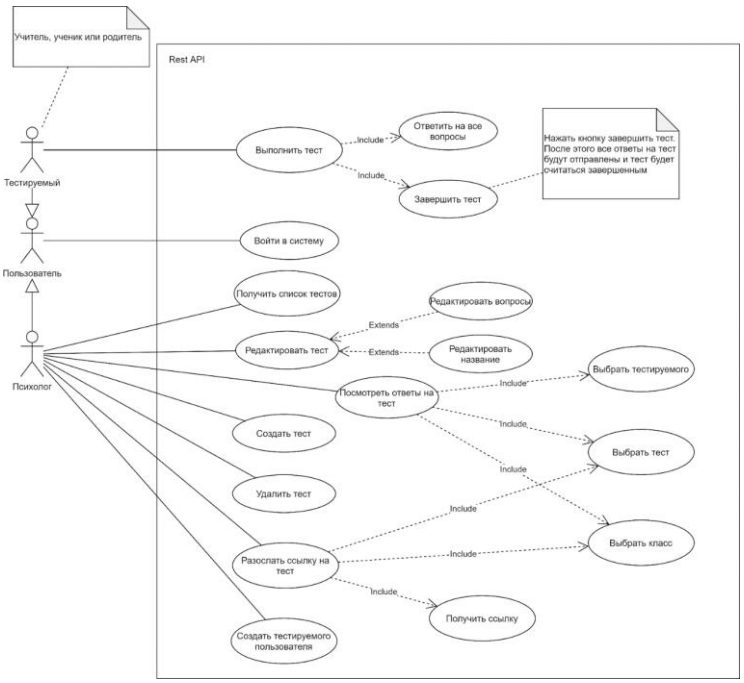


Рисунок 1 – Диаграмма вариантов использования

Составлена спецификация для основных прецедентов из диаграммы вариантов использования.

В таблице 1 приведено описание основного порядка выполнения ролью «Психолог» варианта использования «Создать тест».

Таблица 1 – Спецификация прецедента «Создать тест»

Прецедент: создать тест
ID: 1
Краткое описание: Создание психологического теста
Главные актеры: Психолог
Основной поток: 1. Психолог добавляет вопросы для теста 1.1. Психолог указывает название вопросов 1.2. Психолог указывает возможные ответы для вопросов 2. Психолог посылает запрос системе на сохранение теста 3. Система сохраняет тест в базе данных

В таблице 2 приведено описание основного порядка выполнения ролью «Психолог» варианта использования «Создать тестируемого пользователя».

Таблица 2 – Спецификация прецедента «Создать тестируемого пользователя»

Прецедент: создать тестируемого пользователя
ID: 2
Краткое описание: Психолог создает пользователя, для которого в последующем будут предоставлены тесты
Главные актеры: Психолог

Продолжение таблицы 2

Предусловие: Психолог закреплен за конкретным классом в школе
Основной поток: 1. Психолог вводит информацию о пользователе 1.1. Психолог указывает ФИО 1.2. Психолог указывает дату рождения 1.3. Психолог указывает роль (ученик, преподаватель или родитель) 2. Психолог посылает запрос системе на добавление пользователя Система сохраняет пользователя в базе данных

В таблице 3 приведено описание основного порядка выполнения ролью «Психолог» варианта использования «Разослать ссылку на тест».

Таблица 3 – Спецификация прецедента «Разослать ссылку на тест»

Прецедент: разослать ссылку на тест
ID: 3
Краткое описание: Психолог создает ссылку на тест и отправляет её тестируемым
Главные актеры: Психолог
Предусловие: Психолог закреплен за конкретным классом в школе
Основной поток: 1. Психолог выбирает тест 2. Психолог посылает запрос системе на генерацию ссылки 3. Система генерирует ссылку на тест 4. Психолог получает ссылку на тест Психолог отправляет ссылку тестируемым пользователям (при помощи внешних систем)

В таблице 4 приведено описание основного порядка выполнения ролью «Тестируемый» варианта использования «Выполнить тест».

Таблица 4 – Спецификация прецедента «Выполнить тест»

Прецедент: выполнить тест
ID: 4
Краткое описание: Тестируемый выполняет тест
Главные актеры: Тестируемый
Основной поток: 1. Тестируемый отвечает на вопрос 1.1. Тестируемый выбирает вопрос 1.2. Тестируемый выбирает ответ на вопрос 2. Если тестируемый не ответил на все вопросы, то вернуться к пункту 1. 3. Тестируемый посылает запрос на сохранение результатов теста Система сохраняет результаты выполнения теста в базу данных

В рамках возможности системы не предусмотрен глубокий анализ тестов, только получение первичного анализа.

1.2 Сравнительный анализ продуктов-аналогов

В результате анализа программных продуктов для организации, проведения и обработки результатов психологического тестирования в образовательных учреждениях определены приложения с высоким рейтингом, которые являются приближенными аналогами разрабатываемого онлайн-сервиса.

Для проведения сравнительного анализа, в качестве критериев, выбраны наиболее важные возможности, которые напрямую влияют на функциональность сервиса, а именно:

а) функциональные возможности:

1) администрирование пользовательской базы;
2) администрирование результатов тестирования;
3) группировка результатов тестирования по классам;
4) управления тестами и их свойствами;

б) нефункциональные возможности:

1) закрытость результатов тестирования;
2) открытость программного интерфейса;
3) бесплатность продукта;

в) технические возможности:

1) хранение данных в базе данных (БД);
2) возможность внесения изменений.

Исследовав бизнес-процессы, которые возникают во время использования систем психологического тестирования личности, были составлены коэффициенты значимости критериев, которые представлены в таблице 5.

Таблица 5 – Критерии оценки систем психологического тестирования личности

№	Наименование	Коэффициент значимости
	Функциональные возможности	
1	Администрирование пользовательской базы	1
2	Администрирование результатов тестирования	0,75
3	Группировка результатов тестирования по классам	0,75
4	Управление тестами и их свойствами	0,5
	Нефункциональные возможности	
5	Закрытость результатов тестирования	0,75
6	Открытость программного интерфейса	1
7	Бесплатность продукта	0,75
	Технические возможности	
8	Хранение данных в БД	0,5
9	Возможность внесения изменений	0,75

Для оценки описанных выше критериев используется пятибалльная шкала соответствия, от 0 – «полное несоответствие», до 4 – «полное соответствие» требованиям. В таблице 6 приведены характеристики оценки.

Таблица 6 – Шкала баллов

Балл	Характеристика оценки
0	функция (качество) отсутствует, реализовать ее невозможно
1	функция (качество) отсутствует, но ее возможно реализовать путем доработки системы
2	похожие, но отличные от требуемых, функции или качества системы находятся в стадии разработки
3	реализованы похожие, но несколько отличные от требуемых, функции или качества системы
4	функция или качество безоговорочно реализовано

Были отобраны приближенные аналоги позволяющие проводить общее тестирование:

- psytest.org;
- onlinetestpad.com;
- testometrica.com.

Таблица 7 заполнена на основе оценки описанных выше систем. Расчет аддитивной суммы интегральной оценки для каждого сравниваемого продукта проведен по следующей формуле:

$$O_{\text{пп}} = \sum_{i=1}^{n} (Z_i \cdot K_i),$$

где $O_{\text{пп}}$ – интегральная оценка программного продукта;

n – количество критериев сравнения заявленных продуктов согласно таблице критериев;

Z_i – значение выполнения критерия (4 – выполнен, 0 – не выполнен);

K_i – коэффициент значимости критерия сравнения (от 0 до 1).

Таблица 7 – Критериальная оценка программных продуктов

№	Критерий	Программный продукт №		
		1	2	3
1	Администрирование пользовательской базы	0	1	1
2	Администрирование результатов тестирования	1	2	0
3	Группировка результатов тестирования по классам	0	0	0
4	Управление тестами и их свойствами	1	3	0
5	Закрытость результатов тестирования	0	1	0
6	Открытость программного интерфейса	0	0	0
7	Бесплатность продукта	4	4	4
8	Хранение данных в СУБД	2	3	2
9	Возможность внесения изменений	0	3	0
	Интегральная оценка	5,25	11,5	5

Подводя итог можно сделать вывод, что все рассмотренные сервисы не способны в полной мере выполнить поставленную задачу. Среди всех продуктов-аналогов больше всех подходит портал onlinetestpad.com. Но он не обладает важным требованием закрытости результатов тестирования и открытостью программного интерфейса. Также этим сервисом психологу будет трудозатратно пользоваться, так как база результатов тестирования никак не структурирована и психологу придется вручную каждый раз перебирать все результаты.

1.3 Постановка задачи

Разрабатываемый онлайн-сервис должен предоставлять возможность организации, проведения и обработки результатов психологического тестирования. Для этого необходимо реализовать функции, описанные в анализе вариантов использования (смотри рисунок 1).

В таблице 8 описаны категории пользователей информационной системы.

Таблица 8 – Категория пользователей информационной системы

№	Категория пользователей	Наименование приложения	Фрагмент структуры предметной области	Информационные задачи
1	Администратор	Психолог	Структурная организация тестируемы; набор тестов; результаты тестирования; расписание тестирования	Организация психологических тестов. Сбор и анализ результатов тестирования
2	Пользователь	Ученик	Персональное расписание тестирования; персональные результаты; тест	Прохождение тестирования
3	Пользователь	Преподаватель	Персональное расписание тестирования; персональные результаты; тест	Прохождение тестирования
4	Пользователь	Родитель	Персональное расписание тестирования; персональные результаты; тест	Прохождение тестирования

У категорий пользователей должны различаться права. Тестируемые должны иметь возможность только проходить тесты. Психолог относиться к административной категории, для него доступно: получение, создание, редактирование и удаление тестов; создать тестируемого пользователя.

Для реализациии различных прав у пользователей необходимо предусмотреть систему авторизации.

1.4 Анализ требований

Проанализировав варианты использования онлайн-сервиса организации, проведения и обработки результатов психологического тестирования личности в образовательных учреждения, можно перейти к составлению функциональных и нефункциональных требования и ограничений к программному продукту.

На основе вариантов использования и диаграммы прецедентов сформирована диаграмма деятельности (рисунок 2).

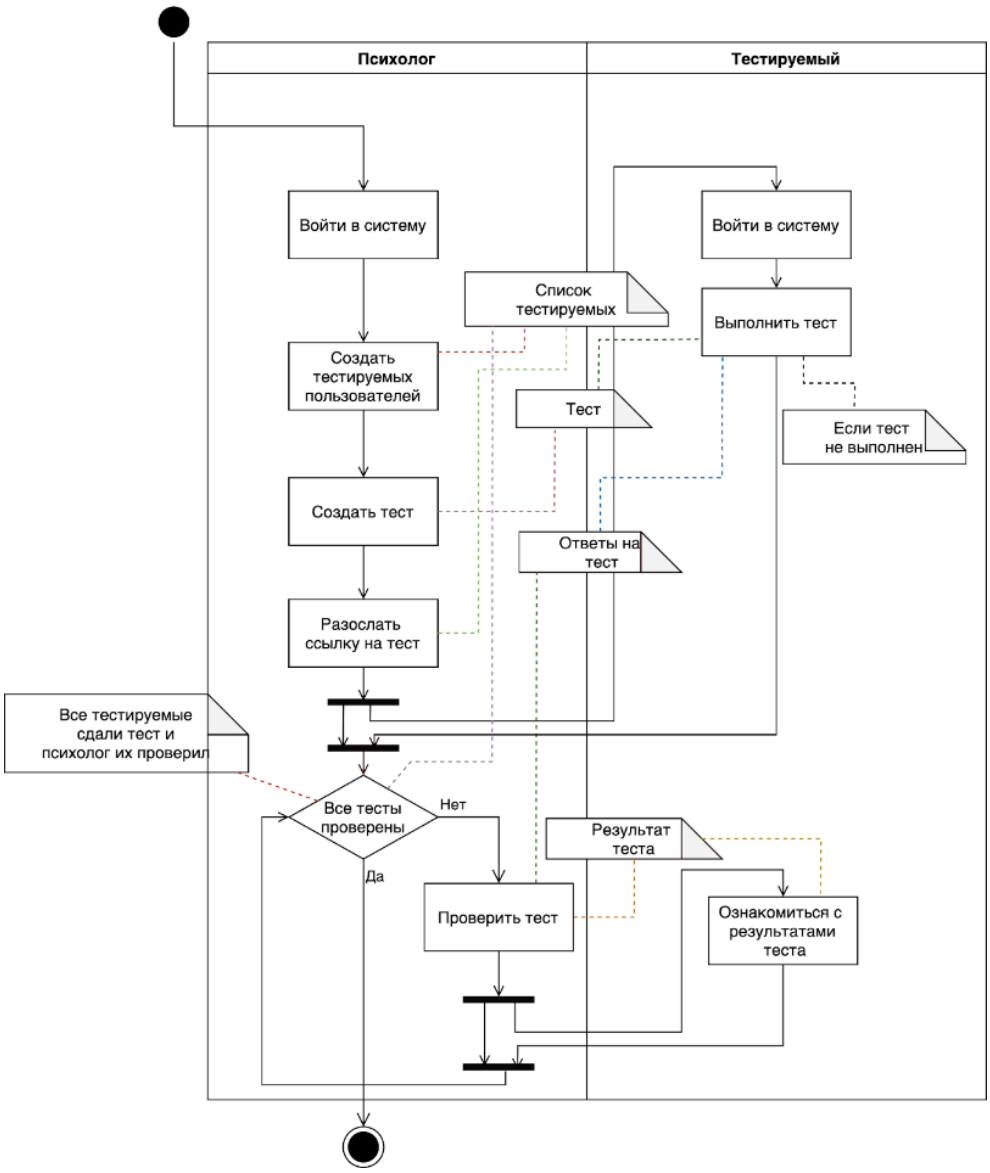


Рисунок 2 – Диаграмма деятельности

На диаграмме деятельности отображена последовательность выполнения основных процессов в системе.

Проанализировав основной цикл взаимодействия с REST-сервером, можно перейти к составлению требований к онлайн сервису организации, проведения и обработки результатов психологического тестирования личности в образовательных учреждениях.

Выделим функциональные требования к программному продукту.

Разрабатываемое REST-серверное приложение обязательно должно обеспечивать пользователей следующими функциями: формирование (создание/редактирование) психологических тестов; прохождение

психологических тестов; получение результатов о прохождении психологических тестов; получение первичного анализа результатов прохождения психологических тестов.

Выделим требования к разделению доступа.

Для любого действия в системе необходима авторизация пользователя.

Если пользователь авторизован под ролью «Психолог», то ему доступна административная часть REST-сервера, которая подразумевает под собой: создание/изменение/удаление психологических тестов и просмотр их результатов.

Если же пользователь авторизован под ролью «Тестируемый», то ему доступно лишь получение информации о тесте (если тест организован для его группы) и возможность выполнить тест.

В иных случаях, клиентское приложение попытается использовать функции, которые не предназначены для конкретной роли, будет выдана ошибка 403 (доступ запрещен).

Если клиентское приложение попробует получить доступ к REST-серверу без авторизации, будет выдана ошибка 401 (отказ в доступе).

Выделим требования к программному обеспечению серверной части.

Для успешной работы REST-сервера требуется: операционная система Windows Server 2003 или Linux; СУБД PostgreSQL версии не ниже 13; среда запуска JRE версии не ниже 8 версии 256 обновления.

Выделим требования к программному обеспечению серверной части.

Для нормального REST-серверного приложения требуются техническое обеспечение со следующими минимальными характеристиками: процессор Intel Core i3-3240; оперативная память 4096 Mb RAM; жесткий диск (HDD) 10Gb свободного места.

2 Проектирование сервиса психологического тестирования

2.1 Моделирование бизнес-процессов предметной области

На основе диаграммы вариантов использования (смотри рисунок 1) были выделены бизнес-процессы для онлайн-сервиса организации, проведения и обработки результатов психологического тестирования личности в образовательных учреждениях (таблица 9).

Таблица 9 – Бизнес-процессы онлайн-сервиса организации, проведения и обработки результатов психологического тестирования личности в образовательных учреждениях

№	Бизнес-процесс	Исполнитель	Входные данные		Выходные данные		Ограничения
			Поставщик	Содержание	Потребитель	Содержание	
1	Получить список тестов	Психолог	Психолог	Сведения о тесте (название, вопросы и возможные варианты ответов)	Психолог	Список тестов	
2	Создать/редактировать тест	Психолог	Психолог	Сведения о тесте (идентификатор теста, учебный класс)	Психолог	Тест	Запрещено менять тесты, которые начали выполнять
3	Сгенерировать ссылку на тест	Психолог	Психолог	Сведения о тестируемом (ФИО, школа, класс, роль)	Тестируемый	Ссылка на тест	Идентификатор теста и учебный класс должны существовать
4	Создать тестируемого пользователя	Психолог	Психолог	Ответы на вопросы, идентификатор тестируемого, идентификатор теста	Психолог	Аккаунт тестируемого пользователя	Запрещено создавать тестируемых с идентичным и наборами данных
5	Выполнить тест	Тестируемый	Тестируемый	Ответы на вопросы, идентификатор тестируемого, идентификатор теста	Психолог	Список ответов тестируемого пользователя	

Продолжение таблицы 9

№	Бизнес-процесс	Исполнитель	Входные данные		Выходные данные		Ограничения
			Поставщик	Содержание	Потребитель	Содержание	
6	Формирование результатов тестируемого	Система	Тестируемый	Ответы на вопросы	Психолог	Шаблонный портрет тестируемого	
7	Получить статистику прохождения теста по классам	Психолог	Психолог	Идентификатор теста, идентификатор школы	Психолог	Процентное соотношение прохождения теста относительно каждого класса	Идентификатор теста и школы должны существовать
8	Получить статистику о длительности выполнения теста по классу	Психолог	Психолог	Идентификатор теста, идентификатор класса	Психолог	Время выполнения теста каждого тестируемого	Идентификатор теста и класса должны существовать

Прежде чем выполнить любой бизнес-процесс в рамках системы, пользователь должен пройти авторизацию. Для авторизации требуется ввести логин и пароль. Логин и пароль выдается психологу от администраторов системы, а тестируемому пользователю выдается от психолога, после создания последнего в системе.

2.2 Разработка концептуальной модели базы данных

Концептуальная модель представляет собой описание основных сущностей (таблиц) и связей между ними без учета принятой модели БД и синтаксиса целевой СУБД. Часто на такой модели отображаются только имена сущностей (таблиц) без указания их атрибутов [3].

Графическое представление концептуальной модели базы данных в виде диаграммы «сущность-связь» для предметной области информационной системы представлено на рисунке 3. Концептуальная модель построена в среде draw.io в нотации IDEF1X.

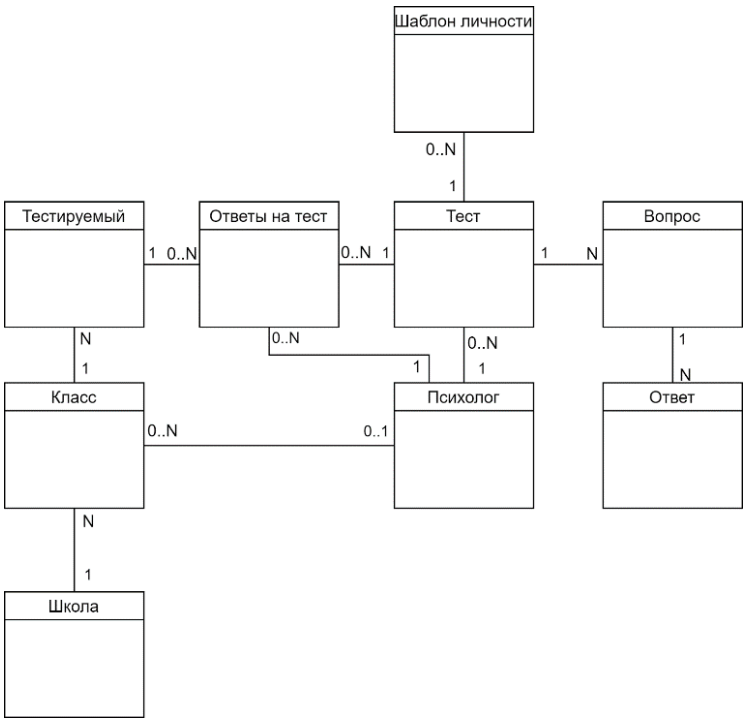


Рисунок 3 – Графическое представление концептуальной модели

В таблице 10 представлены сущности, сформированные на основании входных и выходных данных бизнес-процессов.

Таблица 10 – Описание сущностей

№	Наименование	Описание	Количество экземпляров	Режим обновления
1	Школа	Перечень школ	15	В среднем 2 раза в год
2	Класс	Перечень классов в конкретной школе	25	В среднем 1 раз в год
3	Тестируемый	Перечень тестируемых пользователей (школьников, учителей, родителей)	700	В среднем 1 раз в год
4	Психолог	Перечень психологов, закрепленных за определенным классом	<= количества экземпляров сущности “Классы” конкретной школы	Несколько раз в год
5	Тест	Перечень тестов, созданных психологами (содержат вопросы и ответы на них)	7	Не чаще одного раза в год

Продолжение таблицы 10

№	Наименование	Описание	Количество экземпляров	Режим обновления
6	Вопрос	Перечень вопросов конкретного теста	30	Не чаще одного раза в год
7	Ответ	Перечень ответов, для конкретного вопроса	4	Не чаще одного раза в год
8	Ответы на тест	Перечень ответов, тестируемых на соответствующие тесты	2800	Раз в месяц
9	Шаблон личности	Перечень шаблонов личностей (помогает соотнести личность с конкретным шаблоном)	2	Не чаще одного раза в год

Для сущностей предметной области были сформированы связи (таблица 11).

Таблица 11 – Описание типов связей между сущностями

№	Тип сущности 1	Тип сущности 2	Описание	Мощность	Обязательность
1	Школа	Класс	В школе есть классы/У класса есть школа	1:M	Обязательная
2	Класс	Тестируемый	В классе есть ученики, школьники, родители/У учеников, школьников, родителей есть класс (в котором они учатся/у них учатся дети/они преподают)	1:M	Обязательная
3	Тестируемый	Ответы на тест	Тестируемый отвечает на тест/Тест выполняется тестируемым	1:M	Необязательная (для тестируемого может быть не подготовлено тестов)
4	Психолог	Класс	Класс закреплен за психологом	M:1	Необязательная (за классом может быть не закреплен психолог)
5	Психолог	Ответы на тест	Ответы на тест закрепляются за психологом	M:1	Необязательная (ответов на тест может не быть)
6	Психолог	Тест	Психолог создает тест/Тест создается психологом	1:M	Необязательная (ни один тест может быть не создан)

Продолжение таблицы 11

№	Тип сущности 1	Тип сущности 2	Описание	Мощность	Обязательность
7	Тестируемый	Ответы на тест	Ответы на тест закрепляются за тестируемым	M:1	Необязательная (тестируемый может не пройти ни одного теста)
8	Тест	Ответы на тест	Ответы на тест закрепляются за тестом	M:1	Необязательная (ответов на тест может не быть)
9	Тест	Шаблон личности	Шаблоны личности закрепляются за определенным тестом	M:1	Необязательная (шаблонов личности может не быть для теста)
10	Тест	Вопрос	Тест содержит вопросы/Вопросы находятся в тесте	1:M	Обязательная
11	Вопрос	Ответ	Вопрос содержит ответы/Ответы находятся в вопросе	1:M	Обязательная

Описание атрибутов для сущности «Школа» представлено в таблице 12.

Таблица 12 – Описание атрибутов сущности «Школа»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Идентификатор школы	Уникальный номер школы	Целый	4	0 – 4 294 967 296	Нет	PK	1	Идентификатор школы
Название школы	Название школы	Символьный	128	А-я0-9№.,	Нет		МОУ СОШ №1	Название школы

Описание атрибутов для сущности «Класс» представлено в таблице 13.

Таблица 13 – Описание атрибутов сущности «Класс»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Идентификатор класса	Уникальный номер класса	Целый	4	0 – 4 294 967 296	Нет	PK	1	
Название класса	Название класса	Символьный	8	А-я0-9	Нет		11Б	
Идентификатор школы	Номер, соответствующий номеру школы	Целый	4	0 – 4 294 967 296	Нет	FK	12	

Описание атрибутов для сущности «Тестируемый» представлено в таблице 14.

Таблица 14 – Описание атрибутов сущности «Тестируемый»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Идентификатор тестируемого	Уникальный номер тестируемого	Целый	4	0 – 4 294 967 296	Нет	PK	1	
Имя	Имя тестируемого	Символьный	64	А-я	Нет		Александр	
Фамилия	Фамилия тестируемого	Символьный	64	А-я	Нет		Пушкин	
Отчество	Отчество тестируемого	Символьный	64	А-я	Да			
День рождения	День рождения тестируемого	Дата	8	03.09.2000 – 03.09.2100	Нет		04.05.2022	
Статус	Статус тестируемого	Символьный	64	А-я	Нет		Ученик	Статус
Логин	Логин тестируемого	Символьный	128	A-z.	Нет		anastasia.s	Логин
Пароль	Пароль тестируемого	Символьный	10	A-z0-9	Нет		as1w845sdz	Пароль

Описание атрибутов для сущности «Психолог» представлено в таблице 15.

Таблица 15 – Описание атрибутов сущности «Психолог»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Идентификатор психолога	Уникальный номер психолога	Целый	4	0 – 4 294 967 296	Нет	PK	1	
ФИО	ФИО психолога	Символьный	300	А-я	Нет		Бирюков Владислав Евгеневич	

Продолжение таблицы 15

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Логин	Логин психолога	Символьный	64	A-z0-9_	Нет		qwerty_	
Пароль	Пароль психолога	Символьный	64	A-z0-9	Да		1qwerty1	
Идентификатор класса	Номер, соответствующий номеру класса	Целый	4	0 – 4 294 967 296	Да	FK	1	

Описание атрибутов для сущности «Тест» представлено в таблице 16.

Таблица 16 – Описание атрибутов сущности «Тест»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Идентификатор теста	Уникальный номер теста	Целый	4	0 – 4 294 967 296	Нет	PK	1	
Название	Название теста	Символьный	256	А-я.	Нет		Тест Тьюринга	
Статус теста	Отражает статус теста (запущен тест или нет)	Логический	1	0-1	Нет		1	0 - ложь, 1 - истина

Описание атрибутов для сущности «Вопрос» представлено в таблице 17.

Таблица 17 – Описание атрибутов сущности «Вопрос»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Идентификатор вопроса	Уникальный номер вопроса	Целый	4	0 – 4 294 967 296	Нет	PK	1	
Порядок вопроса	Порядковый номер вопроса в тесте	Целый	2	0 – 65536	Нет		5	

Продолжение таблицы 17

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Текст вопроса	Содержит текст вопроса	Символьный	256	А-я0-9.,-;?!	Нет		Что случится, если рыбы захватят мир?	
Идентификатор теста	Номер, соответствующий номеру теста	Целый	4	0 – 4 294 967 296	Нет	FK	1	

Описание атрибутов для сущности «Ответ» представлено в таблице 18.

Таблица 18 – Описание атрибутов сущности «Ответ»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Идентификатор ответа	Уникальный номер ответа	Целый	4	0 – 4 294 967 296	Нет	PK	1	
Порядок ответа	Порядковый номер ответа в вопросе	Целый	2	0-65536	Нет		5	
Идентификатор вопроса	Номер, соответствующий номеру вопроса	Целый	4	0 – 4 294 967 296	Нет	FK	1	

Описание атрибутов для сущности «Ответ на тест» представлено в таблице 19.

Таблица 19 – Описание атрибутов сущности «Ответ на тест»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Идентификатор ответа на тест	Уникальный номер ответа на тест	Целый	4	0 – 4 294 967 296	Нет	PK	1	
Дата создания	День создания ответа	Дата со временем	8	03.09.2000 15:00:25 – 03.09.2100 19:00:00	Нет		04.05.2019 23:00:00	

Продолжение таблицы 19

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Ответы	Список ответов тестируемого (JSON структура, содержащая информацию о каждом вопросе, когда был произведен ответ, какой ответ и на какой вопрос)	Символьный	20000	Любые символы	Нет			
Идентификатор тестируемого	Номер, соответствующий номеру тестируемого	Целый	4	0 – 4 294 967 296	Нет	FK	1	
Идентификатор теста	Номер, соответствующий номеру теста	Целый	4	0 – 4 294 967 296	Нет	FK	1	
Идентификатор психолога	Номер, соответствующий номеру психолога	Целый	4	0 – 4 294 967 296	Нет	FK	1	

Описание атрибутов для сущности «Шаблон личности» представлено в таблице 20.

Таблица 20 – Описание атрибутов сущности «Шаблон личности»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Идентификатор шаблона личности	Уникальный номер шаблона личности	Целый	4	0 - 4 294 967 296	Нет	PK	1	
Текст шаблона	Текст, содержащий описание личности в соответствии с шаблоном	Символьный	1000	Любые символы	Нет		У личности есть ярко выраженные ораторские способности	

Продолжение таблицы 20

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Ответы	Список ответов (JSON структура, содержащая информацию о некоторых вопросах теста, их идентификаторах и какой ответ должен выбрать тестируемый, чтобы приблизиться к шаблону)	Символьный	20000	Любые символы	Нет			
Идентификатор теста	Номер, соответствующий номеру теста	Целый	4	0 – 4 294 967 296	Нет	FK	1	

Таким образом была разработка концептуальная модель базы данных.

2.3 Моделирование базы данных

Логическая модель – модель базы данных, выраженная в понятиях модели данных [4]. Логическая модель расширяет концептуальную путем определения для сущностей их атрибутов, описаний и ограничений, уточняет состав сущностей и взаимосвязи между ними.

На основании концептуальной модели была сконструирована логическая модель базы данных (рисунок 4).

Физическая модель – логическая модель базы данных, выраженная в терминах языка описания конкретной СУБД [5]. В данной выпускной квалификационной работе – в терминах СУБД PostgreSQL.

На основе концептуальной и логической модели была разработана физическая модель базы данных (рисунок 5).

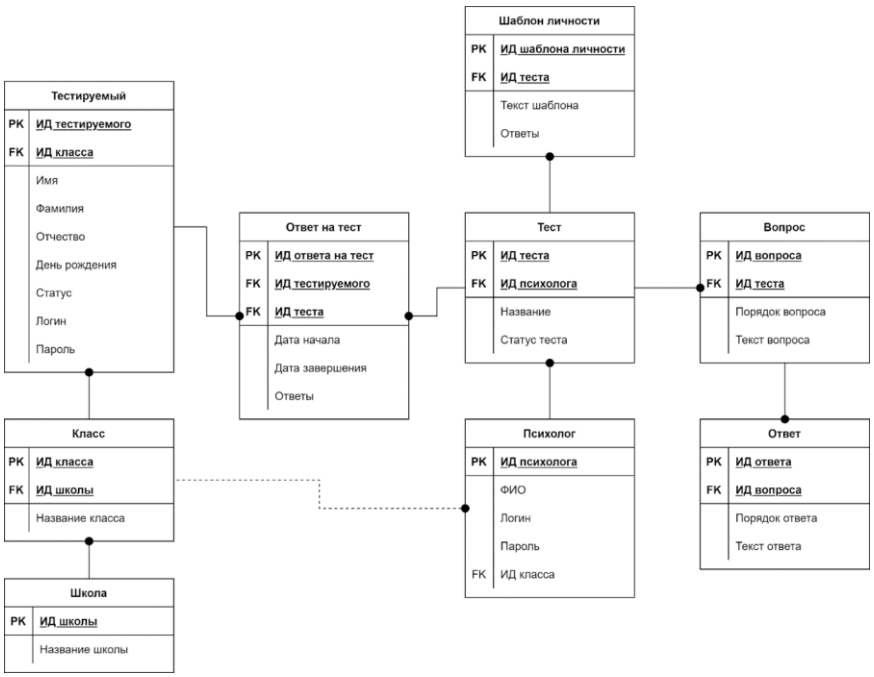


Рисунок 4 – Графическое представление логической модели базы данных

На логической модели базы данных четко видно взаимоотношение сущностей между друг другом.

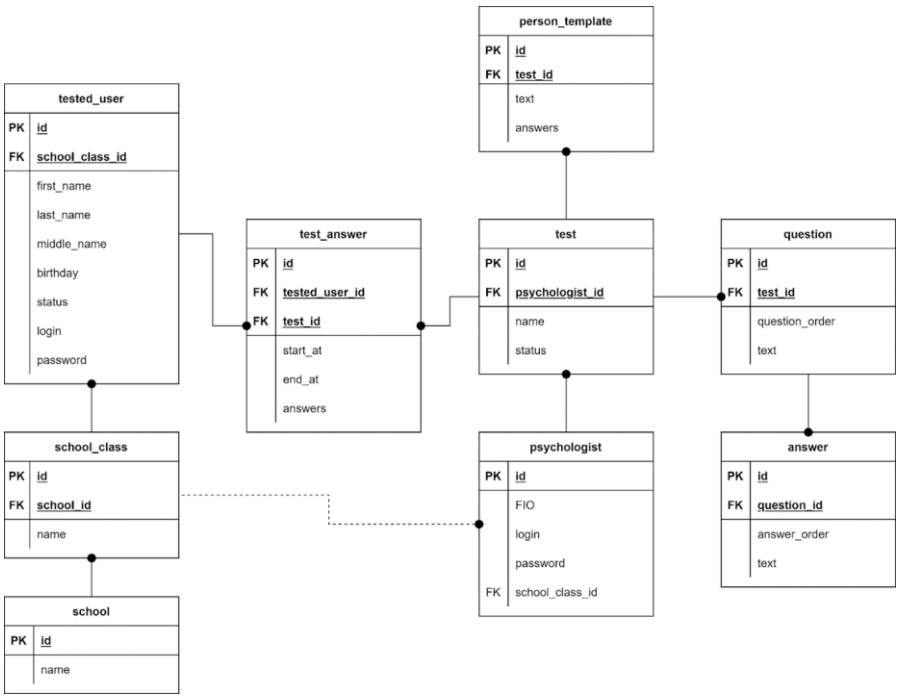


Рисунок 5 – Графическое представление физической модели базы данных

Физическая модель полностью отражает реальную структуру базы данных.

3 Реализация сервиса психологического тестирование

3.1 Развертывание и наполнение базы данных тестовыми данными

3.1.1 Развертывание базы данных в среде PostgreSQL с помощью средств ORM

ORM (англ. Object-Relational Mapping, рус. объектно-реляционное отображение) – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных» [6].

В контексте языка программирования Java за объектно-реляционное отображение отвечает библиотека Hibernate [7], а в частности спецификация JPA (она предоставляет возможность сохранять в удобном виде Java-объекты в базе данных).

Для того, чтобы создать таблицы базы данных при помощи ORM достаточно запустить приложение и указать специальную опцию в конфигурационном файле (рисунок 6).

```
spring:
  jpa:
    show-sql: false
    hibernate:
      ddl-auto: update
```

Рисунок 6 – Фрагмент конфигурационного файла

За автоматическую генерацию (обновление) базы данных отвечает опция «spring.jpa.hibernate.ddl-auto» со значением «update» и ряд классов-сущностей (AnswerEntity.class, QuestionEntity.class и т.д.), на основе которых, производится генерация таблиц, атрибутов и связей. Опция «ddl-auto» предписывает автоматическое создание таблиц базы данных и обновление их в случае, если в объекты сущностей были добавлены новые поля/связи с другими сущностям. Рисунок диаграммы сущность-связь, построенной на основе таблиц БД представлен на рисунке 7.

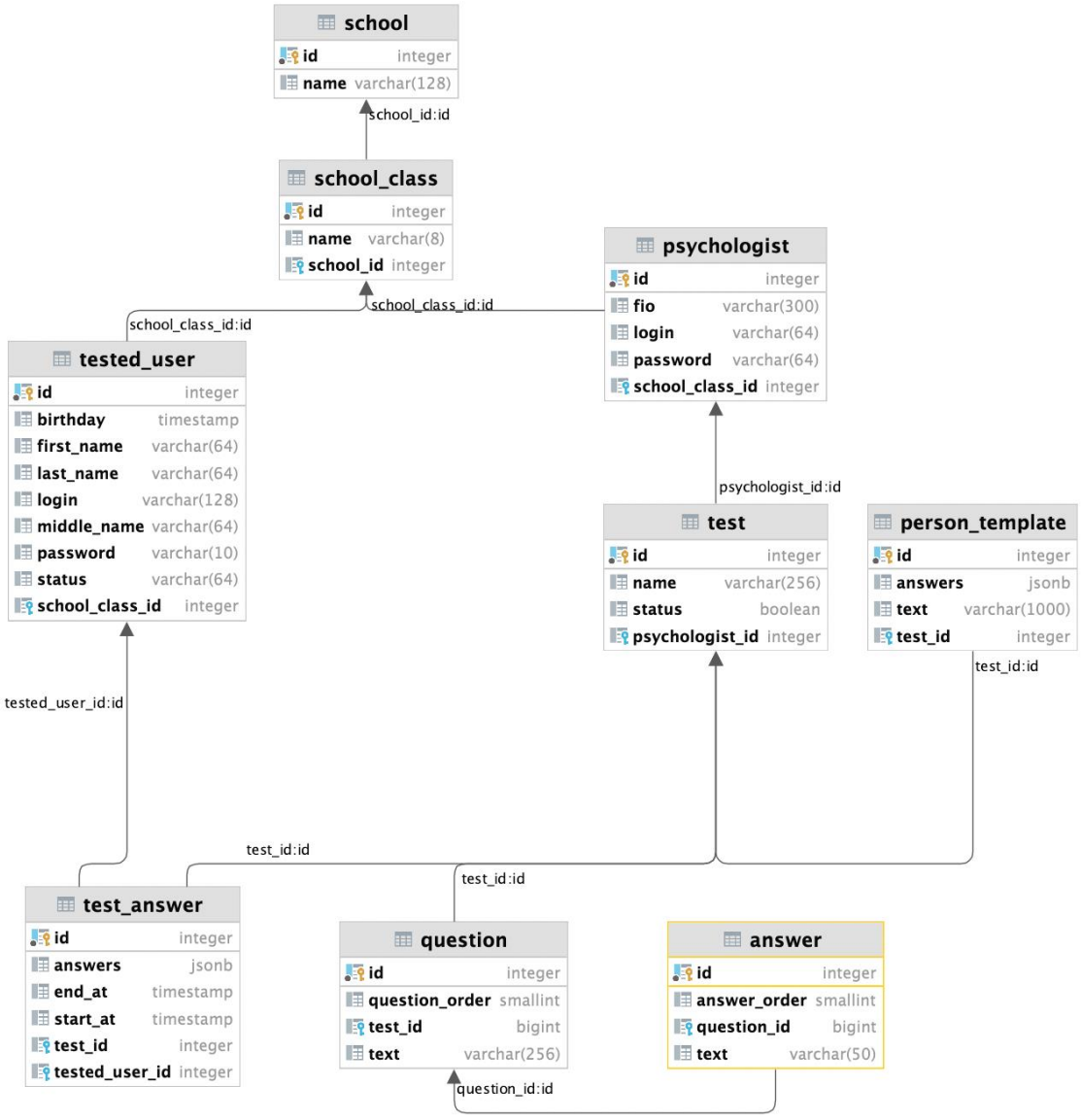


Рисунок 7 – Диаграмма «сущность-связь» итоговой базы данных

Исходный код Java классов, отвечающих за создание базы данных, представлен в приложении Б.

3.1.2 Наполнение базы данных тестовыми данными с помощью средств ORM

За тестовое наполнение данными отвечает класс «TestData». В контексте предметной области этот класс отвечает за создание психолога и школы с классом. Это основные сущности базы данных, поскольку созданием тестируемых пользователей занимается психолог, и за наполнение остальными данными отвечает он и сами тестируемые пользователи.

Отдельно стоит рассмотреть класс «PsychologistTest», он отвечает за наполнение остальными данными для тестирования возможностей психолога в системе.

Содержимое таблицы «tested_user» после наполнения тестовыми данными представлено на рисунке 8.

	id	birthday	first_name	last_name	login	middle_name	password	status	school_class_id
1	206	2021-12-16 14:46:29.914494	Владислав	Бирюков	v.biryukov	Евгеньевич	a6581395a3	STUDENT	166
2	207	2021-12-16 14:46:30.043267	Дибин	Андрей	d.andrey	Юрьевич	5b4379d5bb	STUDENT	166

Рисунок 8 – Содержимое таблицы «tested_user»

Содержимое таблицы «test_answer» после наполнения тестовыми данными представлено на рисунке 9.

	id	answers	end_at	start_at	test_id	tested_user_id
1	223	[{"answer_id": 210, "created_at": 1639655190233, "	2021-12-16 14:46:30.241970	2021-12-16 14:46:30.233379	208	206
2	224	[{"answer_id": 212, "created_at": 1639655190253, "	2021-12-16 14:46:30.258717	2021-12-16 14:46:30.253461	208	207

Рисунок 9 – Содержимое таблицы «test_answer»

Содержимое таблицы «test» после наполнения тестовыми данными представлено на рисунке 10.

	id	name	status	psychologist_id
1	208	Simple test	false	167

Рисунок 10 – Содержимое таблицы «test»

Содержимое таблицы «school_class» после наполнения тестовыми данными представлено на рисунке 11.

	id	name	school_id
1	166	11Б	165

Рисунок 11 – Содержимое таблицы «school_class»

Содержимое таблицы «school» после наполнения тестовыми данными представлено на рисунке 12.

	id	name
1	165	МОУ СОШ №1

Рисунок 12 – Содержимое таблицы «school»

Содержимое таблицы «question» после наполнения тестовыми данными представлено на рисунке 13.

	id	question_order	text	test_id
1	209	1	First question.	208
2	213	2	Second question.	208
3	217	3	Third question.	208

Рисунок 13 – Содержимое таблицы «question»

Содержимое таблицы «psychologist» после наполнения тестовыми данными представлено на рисунке 14.

	id	fio	login	password	school_class_id
1	167	Александров Алексей Александрович	a.alexandrov	a.alexandrov	166

Рисунок 14 – Содержимое таблицы «psychologist»

Содержимое таблицы «person_template» после наполнения тестовыми данными представлено на рисунке 15.

	id	answers	text	test_id
1	221	[{"answer_id": 211, "question_id": 209}, {"answer_id": 216, "question_id": 213	Здоровый ученик	208
2	222	[{"answer_id": 212, "question_id": 209}, {"answer_id": 214, "question_id": 213	Не здоровый ученик	208

Рисунок 15 – Содержимое таблицы «person_template»

Содержимое таблицы «answer» после наполнения тестовыми данными представлено на рисунке 15.

	id	answer_order	text	question_id
1	210	1	First	209
2	211	2	Second	209
3	212	3	Third	209
4	214	1	First	213
5	215	2	Second	213
6	216	3	Third	213
7	218	1	First	217
8	219	2	Second	217
9	220	3	Third	217

Рисунок 16 – Содержимое таблицы «answer»

Листинги классов «TestData» и «PsychologistTes»t так же приведены в приложении Б.

3.2 Разработка серверной части приложения

3.2.1 Разработка запросов к базе данных и представлений для реализации аналитических бизнес-процессов при помощи ORM

При помощи связки Hibernate в составе JPA обычные Java объекты можно представить в виде данных таблиц БД. Достигается это тем, что фреймворк Hibernate отображает представление POJO как сущности базы данных. JPA же является интерфейсом для взаимодействия с этими представлениями. Оба эти инструмента и позволяют создавать и манипулировать данными в БД.

Большим плюсом такого подхода является то, что помимо стандартного SQL, используемого при проектировании запросов, можно использовать JPQL, что позволяет не ограничиваться в выборе СУБД (будь то PostgreSQL или же MySQL), так как этот язык оперирует объектным представлением и все запросы формируются относительно него.

В этом легко убедиться на следующем примере. Есть класс «TestedUserEntity» (рисунок 17) и класс «TestedUserRepository» для работы с таблицей «tested_user» (рисунок 18).

```
@Entity
@Table(name = "tested_user")
public class TestedUserEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    Integer id;

    @Builder.Default
    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true, fetch = FetchType.LAZY)
    @JoinColumn(name = "tested_user_id", referencedColumnName = "id")
    List<TestAnswerEntity> testedUserAnswers = new ArrayList<>();

    @Column(length = 64)
    @NonNull
    String firstName;

    @Column(length = 64)
    @NonNull
    String lastName;

    @Column(length = 64)
    String middleName;
}
```

Рисунок 17 – Класс «TestedUserEntity»

```
public interface TestedUserRepository extends JpaRepository<TestedUserEntity, Integer> {

    // 1) Объявление своего запроса
    @Query("SELECT u FROM TestedUserEntity u WHERE u.id =:userId AND u.schoolClass.id =:schoolClassId")
    Optional<TestedUserEntity> findByIdAndSchoolClassId(Integer userId, Integer schoolClassId);

    // 2) Объявление запроса при помощи языка шаблонов
    Optional<TestedUserEntity> findTopByLoginAndPassword(@NonNull String login, @NonNull String password);
}
```

Рисунок 18 – Класс «TestedUserRepository»

Сущность базы данных описывается декларативно, при помощи аннотаций. Характерной аннотацией, которая объявляет класс сущностью базы данных, является аннотация @Entity. Аннотация @Table позволяет задать иное название таблицы или создать индексы и т.п.

В репозитории для работы с данными видно, что в запросе 1 используется не язык SQL, а именно JPQL, так как JPQL воздействует на объекты, атрибуты и отношения вместо таблиц и столбцов. Таким образом добавляется дополнительный уровень абстракции между программистом и SQL, при этом JPQL также поддерживает многие операторы и функции языка SQL.

Несмотря на то, что этот механизм абстрагирует программиста от реального SQL, это не означает, что сам программист при помощи этих инструментов не может сам построить запрос. JPA позволяет использовать «нативные» SQL запросы в БД (рисунок 19).

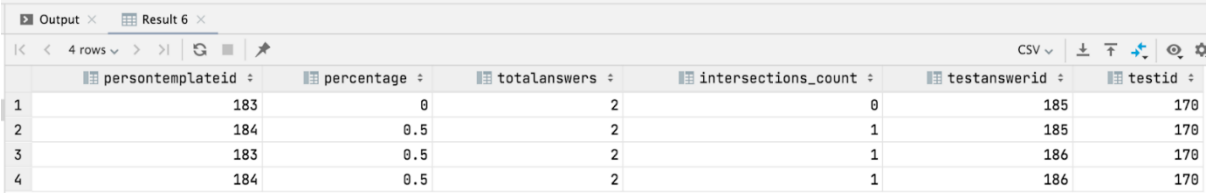
```
public interface TestAnswerRepository extends JpaRepository<TestAnswerEntity, Integer> {

    @Query(value =
            "SELECT pt.id AS personTemplateId, " +
                    "TOTAL_COUNT(pt.id) AS totalAnswers, " +
                    "INTERSECTIONS_COUNT(pt.answers, ta.answers) AS testedUserAnswers, " +
                    "ta.id AS testAnswerId " +
                    "FROM test_answer ta " +
                        "INNER JOIN tested_user tu ON ta.tested_user_id = tu.id " +
                        "INNER JOIN school_class sc ON tu.school_class_id = :schoolClassId " +
                        "INNER JOIN test t ON ta.test_id = :testId " +
                        "INNER JOIN person_template pt ON t.id = pt.test_id " +
                        "GROUP BY ta.id, pt.id",
            nativeQuery = true
    )
    List<IPersonTemplateAnalyze> analyzePersonTemplates(Integer schoolClassId, Integer testId);
}
```

Рисунок 19 – Пример «нативного» запроса JPA

В этом примере по мере завершения запроса формируется психологический портрет тестируемых определенного класса по определенному тесту. Таким образом, при помощи данных механизмов можно реализовать любой, специфический для выбранной СУБД, запрос и ORM не накладывает ограничений на вариативность запроса, она лишь служит вспомогательным инструментом, который берет на себя большой объем шаблонной работы (простые выборки по id, сохранение объекта в БД, удаление по id генерируются автоматически для каждого класса сущности).

Результат выполнения запроса на формирование психологического портрета тестируемых представлен на рисунке 20.


Output Result 6
4 rows CSV

	personTemplateId	percentage	totalAnswers	intersections_count	testAnswerId	testId
1	183	0	2	0	185	170
2	184	0.5	2	1	185	170
3	183	0.5	2	1	186	170
4	184	0.5	2	1	186	170

Рисунок 20 – Результат запроса в базу данных

Часть JSON [8] объекта пользовательского представления приведена в листинге 1.

Листинг 1 – JSON объект, получаемый в ходе запроса результатов теста

```
[
  {
    "test": {
      "id": 208,
      "name": "Simple test"
    },
    "answers": [
      …
    ],
    "personAnalyzes": [
      …
    ],
    "user": {
      "id": 206,
      "firstName": "Владислав",
      "middleName": "Евгеньевич",
      "lastName": "Бирюков"
    }
  },
  {
    "test": {
      "id": 208,
      "name": "Simple test"
```

```
    },
    "answers": [
      …
    ],
    "personAnalyzes": [
      …
    ],
    "user": {
      "id": 207,
      "firstName": "Дибин",
      "middleName": "Юрьевич",
      "lastName": "Андрей"
    }
  }
]
```


В приведенном выше JSON объекте хранится информация о тестируемых пользователях и их шаблонах личности.

3.2.2 Разработка функций для реализации учетных пользовательских бизнес-процессов на стороне REST-сервера при помощи ORM

Для примера рассмотрим процесс создания(модификации) теста. Класс «TestController» отвечает за работу с тестами. Его содержимое представлено на рисунке 21.

```
@RestController
@Transactional
public class TestController {

    // Интерфейс для работы с базой данных
    TestRepository testRepository;

    // Фабричный класс, который отвечает за преобразование
    // данных в пользовтаельский формат (создает DTO)
    TestDtoFactory testDtoFactory;

    public static final String CREATE_OR_UPDATE_TEST = "/api/tests";

    @PostMapping("/api/tests")
    public TestDto createOrUpdateTest(@RequestBody TestDto test) {

        TestEntity testEntity = convertTestToEntity(test);

        testEntity = testRepository.saveAndFlush(testEntity);

        return testDtoFactory.createTestDto(testEntity);
    }

    private TestEntity convertTestToEntity(TestDto dto) {...}

    private QuestionEntity convertQuestionToEntity(QuestionDto dto) {...}

    private AnswerEntity convertAnswerToEntity(AnswerDto dto) {...}
}
```

Рисунок 21 – Класс TestController

Основным методом, на который приходят запросы от пользователей, является метод «createOrUpdateTest». На вход он получает само представление теста. Сам метод содержит в себе работу с сущностями базы данных, о чем говорит вызов интерфейса «testRepository» в контексте сохранения сущности базы данных. Данные операции невозможны без инициализации транзакции. Транзакция инициализируется декларативно, при помощи добавления аннотации @Transactional, стоящая перед классом. На рисунке 22 представлено поведение метода «createOrUpdateTest» с учетом этой аннотации.

Помимо основного метода, в классе еще присутствуют три вспомогательных: «convertTestToEntity» (рисунок 23), «convertQuestionToEntity» и «convertAnswerToEntity».

```
// Перед началом работы метода транзакция открывается
// transaction.begin();
@PostMapping("/api/tests")
public TestDto createOrUpdateTest(@RequestBody TestDto test) {

    TestEntity testEntity = convertTestToEntity(test);

    testEntity = testRepository.saveAndFlush(testEntity);

    return testDtoFactory.createTestDto(testEntity);
}
// transaction.end();
// После завершения работы метода транзакция закрывается
```

Рисунок 22 – Поведение метода «createOrUpdateTest» с учетом аннотации @Transactional

На рисунке 22 приведено описание поведения метода «createOrUpdateTest» в виде комментариев к коду.

```
private TestEntity convertTestToEntity(TestDto dto) {

    Integer testId = dto.getId();

    TestEntity test;
    if (testId == null) {
        test = TestEntity.makeDefault();
    } else {
        test = testRepository
                .findById(testId)
                .orElseThrow(() ->
                        new NotFoundException(String.format("Тест с индентификатором \"%s\" не найден.", testId))
                );
    }

    test.setName(dto.getName());
    test.getQuestions().clear();

    test.getQuestions().addAll(
            dto.getQuestions() List<QuestionDto>
                    .stream() Stream<QuestionDto>
                    .map(this::convertQuestionToEntity) Stream<QuestionEntity>
                    .collect(Collectors.toList()) List<QuestionEntity>
    );

    return test;
}
```

Рисунок 23 – Содержимое метода «convertTestToEntity»

Метод «convertTestToEntity» отвечает за отображение объекта клиентского представления (TestDto) в объект сущность (TestEntity) для взаимодействия с БД (для того же служат и остальные методы). В данном методе решается следующее: создается ли новый тест, или изменяется существующий. Для того, чтобы это было возможно, сначала необходимо получить идентификатор теста, если он пустой – значит пользователь желает создать тест, в ином случае изменить. Если он пытается изменить тест, то в БД ищется запись с соответствующим идентификатором, и если она не была найдена, то пользователю «пробрасывается» человеко-читаемая ошибка, о том, что тест не был найден. То же самое происходит в остальных методах, только отдельно для сущностей QuestionEntity и AnswerEntity. В результате успешного выполнения будет создан/модифицирован тест.

Результат выполнения логики метода «createOrUpdateTest», отображенный в базе данных, представлен на рисунках 24 – 26.

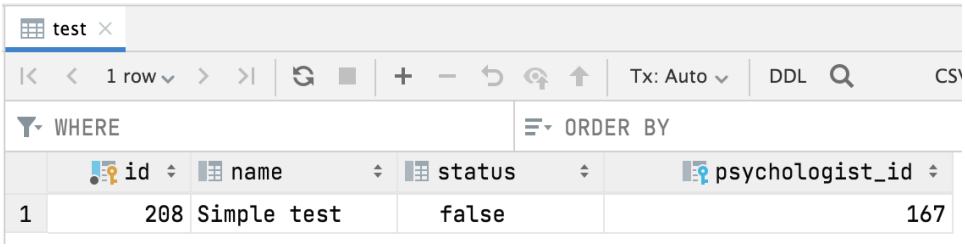


Рисунок 24 – Содержимое таблицы «test»

На рисунке 24 отображено содержание таблицы «test» после выполнения логики метода «createOrUpdateTest».

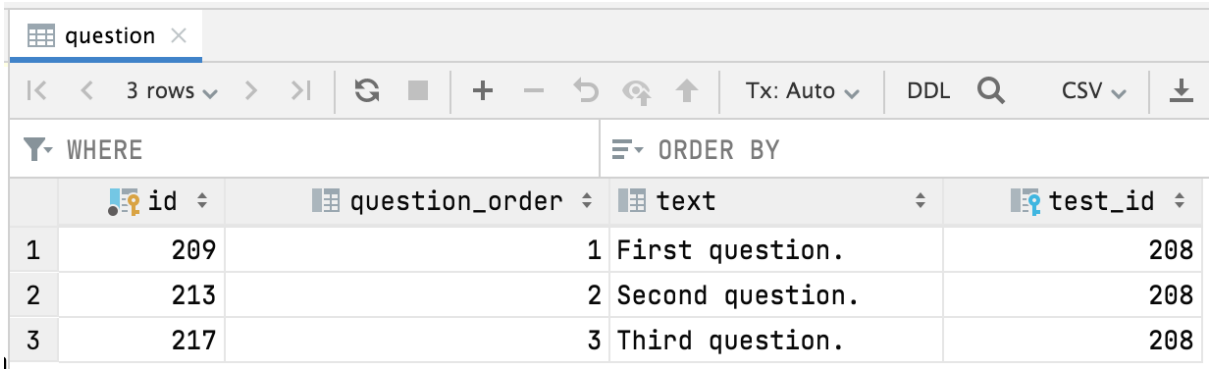


Рисунок 25 – Содержимое таблицы «question»

На рисунке 25 отображено содержание таблицы «question» после выполнения логики метода «createOrUpdateTest».

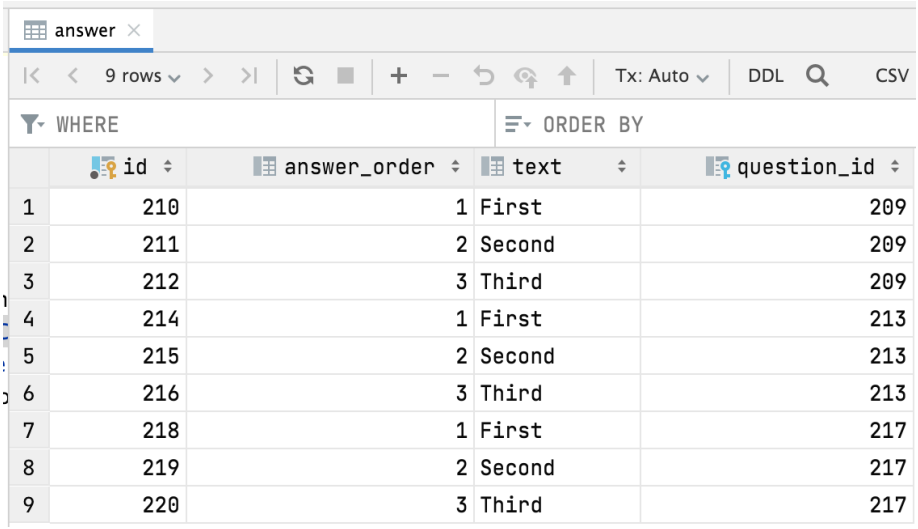


Рисунок 26 – Содержимое таблицы «answer»

На рисунке 26 отображено содержание таблицы «answer» после выполнения логики метода «createOrUpdateTest».

3.3 Тестирование приложения

3.3.1 Выбор режима тестирования приложения

Существует достаточно много видов тестирования программного продукта. Были рассмотрены следующие виды тестирования:

– функциональное тестирование;
– нефункциональное тестирование;
– интеграционное тестирование.

Функциональное тестирование – направлено на проверку того, какие функции приложения реализованы, и что они работают верным образом. Данный метод тестирования является одним из самых применяемых, так как позволяет наиболее полно проверить программу на соответствие требованиям [9].

Нефункциональное тестирование – вид тестирования, направленный на проверку нефункциональных особенностей приложения, таких как удобство использования, совместимость, производительность, безопасность и прочее [10].

Интеграционное тестирование – направлено на проверку взаимодействия между несколькими частями приложения. Этот вид тестирования позволяет проверить корректность взаимодействия частей ПО между собой [11].

Было принято решение осуществить проверку программы с помощью функционального тестирования. Такой выбор обусловлен тем, что к программе были предъявлены в основном функциональные требования, выполнение которых нужно проверить.

3.3.2 Тестирование REST-серверного приложения

Для основных вариантов использования были выполнены функциональные тесты (таблица 21).

Таблица 21 – Функциональные тесты REST-серверного приложения

№	Назначение теста	Описание теста	Результат теста
1	Тест предназначен для проверки варианта использования «Войти в систему» для роли «Психолог»	Входные данные: Логин – a.alexandrov Пароль – a.alexandrov Выходные данные: Токен для аутентификации	Тест пройден успешно. Психологу присвоен токен для аутентификации внутри системы (листинг 2). Самая последняя строчка – токен психолога.
2	Тест предназначен для проверки варианта использования «Получить тесты» для роли «Психолог» без токена аутентификации		Тест пройден успешно. Система сообщила о том, что доступ к ресурсу запрещен и необходимо авторизоваться (листинг 3)
3	Тест предназначен для проверки варианта использования «Получить тесты» для роли «Психолог» с токеном аутентификации	Входные данные: Токен – ba968956-0a9a-45c7-b518-6a036b04578e Выходные данные: Список тестов	Тест пройден успешно (листинг 4).
4	Тест предназначен для проверки варианта использования «Посмотреть ответы на тест» для роли «Психолог»	Входные данные: Токен – ba968956-0a9a-45c7-b518-6a036b04578e Идентификатор теста - 1007	Тест успешно пройден. Психолог получил ответы все ответы на тест и их анализ (листинг 5)
5	Тест предназначен для проверки варианта использования «Удалить тест» для роли «Психолог»	Входные данные: Токен – ba968956-0a9a-45c7-b518-6a036b04578e Идентификатор теста - 1007	Тест успешно пройден (листинг 6). Тест не был удален, так как тестирование запущено.

Ниже приведены листинги выполнения функциональных тестов, представленных в таблице 21.

Листинг 2 – Лог запроса, отображающий успешную авторизацию психолога в системе

```
GET
http://localhost:9125/api/psychologists/authorize?login=a.alexandrov&password
=a.alexandrov

HTTP/1.1 200
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Content-Type: text/plain;charset=ISO-8859-1
Content-Length: 36
Date: Thu, 23 Dec 2021 13:23:04 GMT
Keep-Alive: timeout=60
Connection: keep-alive

ba968956-0a9a-45c7-b518-6a036b04578e # Токен психолога
```

На листинге 2 видно, что психолог смог пройти успешную авторизацию и сервером был возвращен токен для работы с системой.

Листинг 3 – Лог запроса получения списка тестов без токена аутентификации

```
GET http://localhost:9125/api/tests

HTTP/1.1 401
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Content-Type: application/json
Transfer-Encoding: chunked
Date: Thu, 23 Dec 2021 13:20:19 GMT
Keep-Alive: timeout=60
Connection: keep-alive

{
  "error": "Unauthorized",
  "error_description": "Для выполнения этого действия необходимо войти в
систему."
}
```

На листинге 3 видно, что система не пропускает запрос не аутентифицированного пользователя.

Листинг 4 – Лог запроса получения списка тестов с токеном аутентификации

```
GET http://localhost:9125/api/tests

HTTP/1.1 200
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Content-Type: application/json
Transfer-Encoding: chunked
Date: Thu, 23 Dec 2021 13:23:45 GMT
Keep-Alive: timeout=60
Connection: keep-alive

[
  {
    "id": 1007,
    "name": "Simple test",
    "questions": [
      …
    ],
    "personTemplates": [
      …
    ]
  }
]

Response code: 200; Time: 168ms; Content length: 847 bytes
```

На листинге 4 видно успешную работу запроса, и, в качестве результата, сервером был возвращен список тестов

Листинг 5 – Лог запроса на получение ответов тестируемых

```
GET http://localhost:9125/api/psychologists/tests/1007/results

HTTP/1.1 200
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Content-Type: application/json
Transfer-Encoding: chunked
Date: Thu, 23 Dec 2021 13:24:51 GMT
Keep-Alive: timeout=60
Connection: keep-alive

[
  {
    "test": {
      "id": 1007,
      "name": "Simple test"
    },
    "answers": [
      …
    ],
    "personAnalyzes": [
      …
    ],
    "user": {
      "id": 1006,
      "firstName": "Владислав",
      "middleName": "Евгеньевич",
      "lastName": "Бирюков"
    }
  }
]

Response code: 200; Time: 52ms; Content length: 616 bytes
```

На листинге 5 видно успешную работу запроса, и, в качестве результата, сервером был возвращен список ответов тестируемых.

Листинг 6 – Лог запроса на удаление теста

```
DELETE http://localhost:9125/api/tests/1007

HTTP/1.1 400
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Content-Type: application/json
Transfer-Encoding: chunked
Date: Thu, 23 Dec 2021 13:25:50 GMT
Connection: close

{
```

```
  "error": "Bad Request",
  "error_description": "Невозможно удалить тест, сначала необходимо завершить
тестирование."
}
```

На листинге 6 видно неудачное завершение запроса, и, в качестве результата, сервером была возвращена ошибка с детальным описанием.

3.4 Анализ качества кода

3.4.1 Измерение устойчивости

Метрика устойчивости для компоненты отражает количественную меру работы, которую необходимо произвести при изменении других зависимых компонент.

Для вычисления позиционной устойчивости кода используется следующая формула:

$$I = \frac{Ce}{Ca + Ce},$$

где I – мера неустойчивости, изменяется в диапазоне от [0; 1];

Ce – количество классов вне данного компонента, зависящих от классов внутри компонента;

Ca – количество классов внутри данного компонента, зависящих от классов вне компонента.

В созданном приложении можно выделить две логические компоненты:

– компонента, реализующая доменные структуры данных;

– компонента для взаимодействия с клиентскими приложениями.

Для каждой компоненты была просчитана величина I – метрика неустойчивости:

$$I_1 = \frac{0}{0+0} = 0,$$

$$I_2 = \frac{0}{9+0} = 0.$$

Исходя из расчетов, оба компонента имеют максимальную меру устойчивости.

3.4.2 Измерение абстрактности

Значение метрики абстрактности определяется простым отношением количества интерфейсов и абстрактных классов к общему числу классов в компоненте [12].

Для вычисления меры абстрактности используется следующая формула:

$$A = \frac{Na}{Nc},$$

где A – мера абстрактности, изменяется в диапазоне от 0 до 1, где 0 означает полное отсутствие абстрактных классов в компоненте, а 1 означает, что компонент не содержит ничего, кроме абстрактных классов;

Nc – количество классов в компоненте;

Na – количество абстрактных классов в том же компоненте.

Для всех компонентов приложения была вычислена мера абстрактности:

$$A_1 = \frac{2}{4} = 0.5,$$

$$A_2 = \frac{0}{4} = 0.$$

Из расчетов видно, что компонента, которая реализует доменные структуры, наполовину состоит из абстрактных классов. Компонент для взаимодействия с клиентскими приложениями не содержит ни одного абстрактного класса.

3.4.3 Главная последовательность

Вычислив меру устойчивости компоненты и ее абстрактность можно отобразить их на графике. На вертикальной оси необходимо отложить меру абстрактности A, а на горизонтальной оси – меру неустойчивости I.

Компоненты, которые располагаются в верхнем левом углу или в нижнем правым являются максимально качественным. На практике часто случается, так что полученная точка лежит на диагонали между точкой (1,0) и точкой (0,1). Такая диагональ называется главная последовательность (рисунок 27).

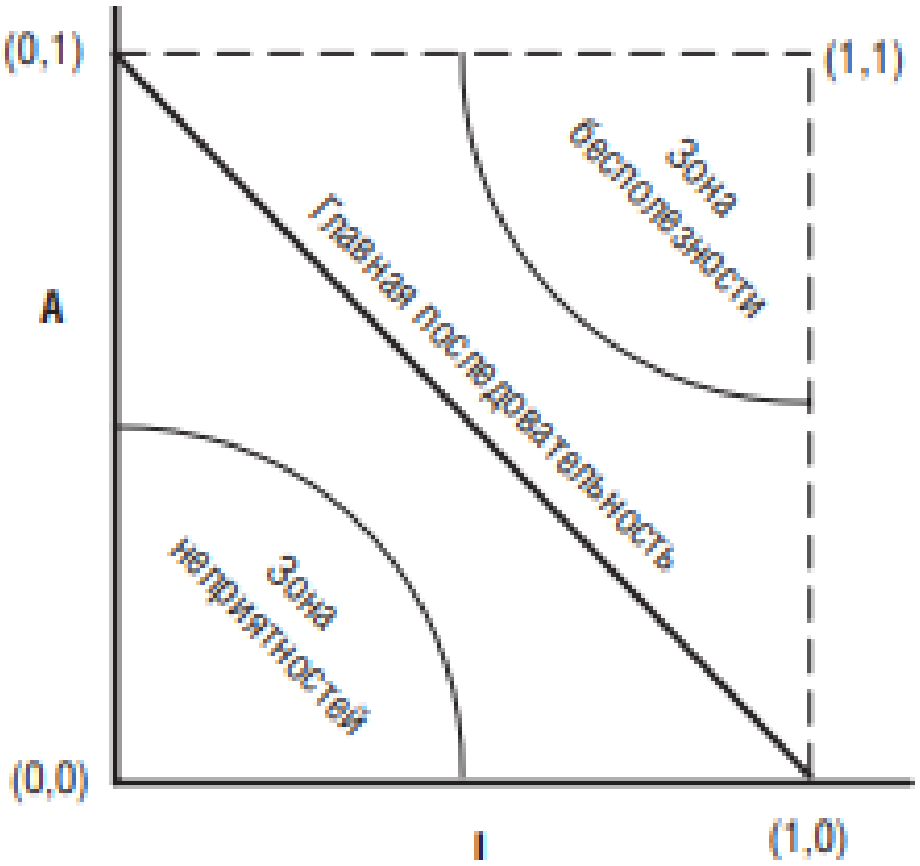


Рисунок 27 – Зоны исключения

Для вычисления количественной оценки используется формула вычисления расстояние до главной последовательности:

$$D = \left|\frac{(A + I - 1)}{\sqrt{2}}\right|,$$

где A – мера абстрактности;

I – мера устойчивости.

Было вычислено расстояние до главной последовательности:

$$D_1 = \left|\frac{0.5 + 0 - 1}{\sqrt{2}}\right| = 0.354,$$

$$D_2 = \left|\frac{0 + 0 - 1}{\sqrt{2}}\right| = 0.707.$$

Для логических компоненты был построен график (рисунок 28).

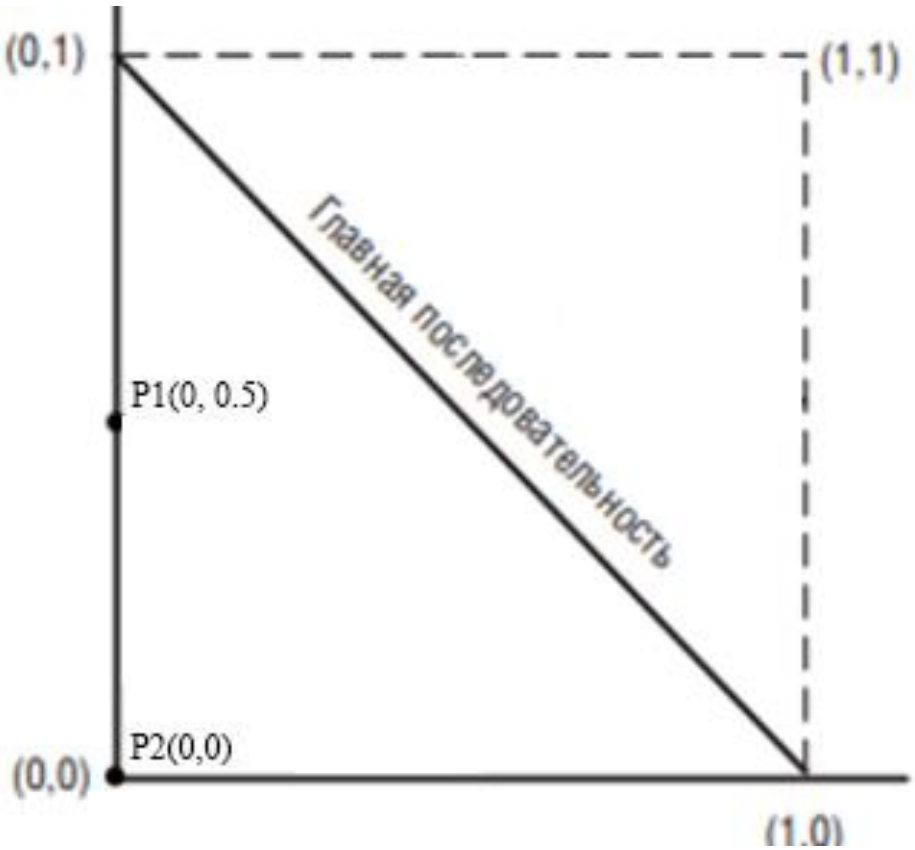


Рисунок 28 – Графическое отображение метрик на графике

Расстояние отношений метрик первой компоненты до главной последовательности достаточно близко к нулю, а значит ее отношение абстрактности к устойчивости является правильным.

Вторая компонента достаточно удалена от главной последовательности (точка P2). Такое расположение компоненты во многом связано с небольшим количеством классов, которые в ней находятся.

4 Планирование разработки и оценка бюджета сервиса психологического тестирования

Планирование разработки и оценка бюджета была выполнена при помощи Microsoft Project. Дата начала проекта – 27 апреля 2022 года. В результате планирования было выделено 14 задач. Задачи проекта, а также их длительность и данные о предшествующих задачах представлены на рисунке 29.

	Название задачи	Длительность	Начало	Окончание	Предшественники	Названия ресурсов
1	Начало проекта	1 день	Ср 13.04.22	Ср 13.04.22		
2	**Анализ требований и постановка задачи**	**7 дней**	**Чт 14.04.22**	**Пт 22.04.22**	**1**	
3	Анализ вариантов использования	1 день	Чт 14.04.22	Чт 14.04.22		Программист 2;Аналитик[50%]
4	Сравнительный анализ продуктов-аналогов	3 дней	Чт 14.04.22	Пн 18.04.22		Аналитик[50%];Программист 1
5	Постановка задачи	2 дней	Вт 19.04.22	Ср 20.04.22	3;4	Аналитик[50%];Программист 2
6	Анализ требований	2 дней	Чт 21.04.22	Пт 22.04.22	5	Аналитик
7	Анализ требований и постановка задачи заверше	0 дней	Пт 22.04.22	Пт 22.04.22	6	
8	**Проектирование**	**15 дней**	**Пн 25.04.22**	**Пт 13.05.22**	**2**	
9	Моделирование бизнес-процессов	7 дней	Пн 25.04.22	Вт 03.05.22		Аналитик[50%];Программист 1[25%];Программист 2[25%]
10	Разработка концептуальной модели базы данных	3 дней	Ср 04.05.22	Пт 06.05.22	9	Аналитик
11	Моделирование базы данных	5 дней	Пн 09.05.22	Пт 13.05.22	10	Программист 1[50%];Программист 2[50%]
12	Проектирование завершено	0 дней	Пт 13.05.22	Пт 13.05.22	11	
13	**Реализация**	**13 дней**	**Пн 16.05.22**	**Ср 01.06.22**	**8**	
14	Развертывание и наполнение базы данных	4 дней	Пн 16.05.22	Чт 19.05.22		Аналитик[25%];Программист 1
15	Разработка серверной части приложения	9 дней	Пт 20.05.22	Ср 01.06.22	14	Программист 1;Программист 2
16	Реализация завершена	0 дней	Ср 01.06.22	Ср 01.06.22	15	
17	**Тестирование**	**6 дней**	**Чт 02.06.22**	**Чт 09.06.22**	**13**	
18	Выбор режима тестирования	2 дней	Чт 02.06.22	Пт 03.06.22		Аналитик
19	Тестирование приложения	4 дней	Пн 06.06.22	Чт 09.06.22	18	Программист 1[50%];Программист 2[50%]
20	Тестирование завершено	0 дней	Чт 09.06.22	Чт 09.06.22	19	
21	**Анализ качества кода**	**3 дней**	**Пт 10.06.22**	**Вт 14.06.22**	**17**	
22	Измерение устойчивости	2 дней	Пт 10.06.22	Пн 13.06.22		Аналитик[50%];Программист 1[50%]
23	Измерение абстрактности	3 дней	Пт 10.06.22	Вт 14.06.22		Программист 2[50%];Аналитик[50%]
24	Анализ качества кода завершен	0 дней	Вт 14.06.22	Вт 14.06.22	23	
25	Составление и оформление документации	3 дней	Ср 15.06.22	Пт 17.06.22	21	Аналитик[50%];Программист 1[25%];Программист 2[25%]
26	Конец проекта	0 дней	Пт 17.06.22	Пт 17.06.22	25	

Рисунок 29 – Список задач проекта

Список ресурсов, используемых при выполнении задач представлен на рисунке 30.

	Название ресурса	Тип	Единицы измерения материалов	Краткое название	Группа	Макс. единиц	Стандартная ставка	Ставка сверхурочных	Затраты на использ.	Начисление	Базовый календарь
1	Программист 1	Трудовой		П		100%	180,00 ₽/ч	360,00 ₽/ч	0,00 ₽	Пропорциональное	Стандартный
2	Программист 2	Трудовой		П		100%	130,00 ₽/ч	260,00 ₽/ч	0,00 ₽	Пропорциональное	Стандартный
3	Аналитик	Трудовой		А		100%	150,00 ₽/ч	300,00 ₽/ч	0,00 ₽	Пропорциональное	Стандартный

Рисунок 30 – Список ресурсов

Диаграмма Ганта [13], которая получена в результате проектирования, представлена на рисунке 31.

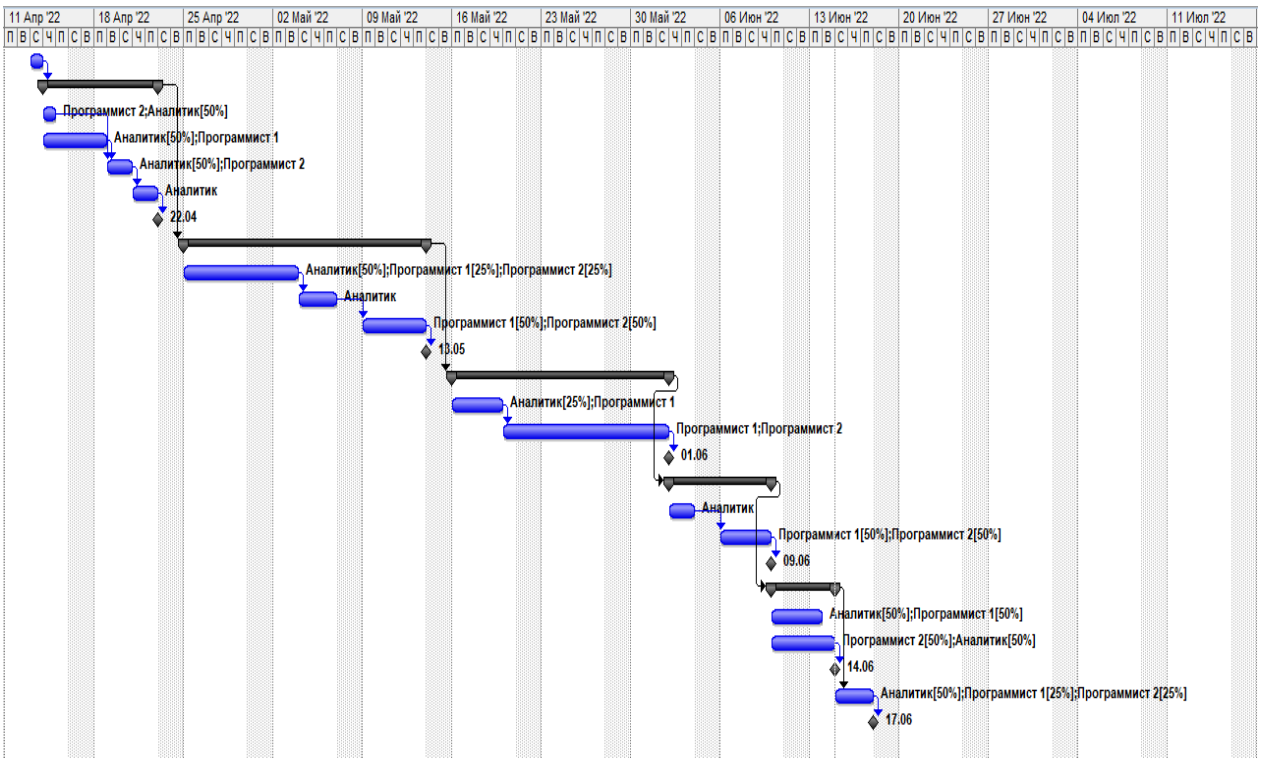


Рисунок 31 – Диаграмма Ганта

На рисунке 32, 33 представлены гистограмма загруженности ресурсов и гистограмма затрат на задачи соответственно.

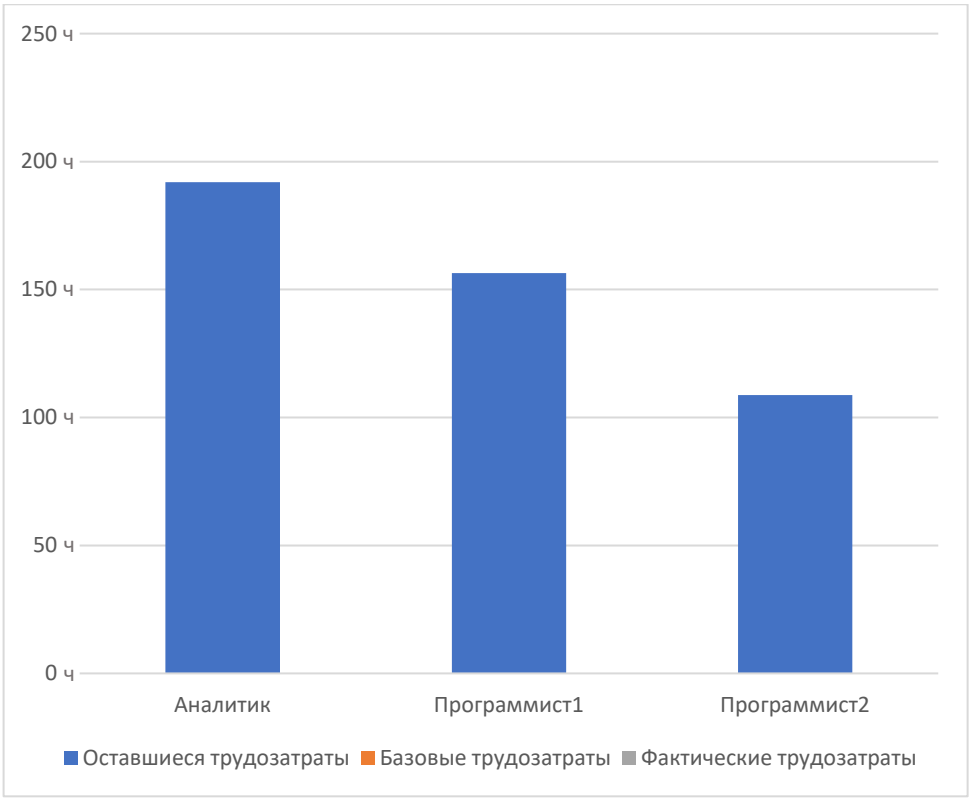


Рисунок 32 – Гистограмма загруженности ресурсов

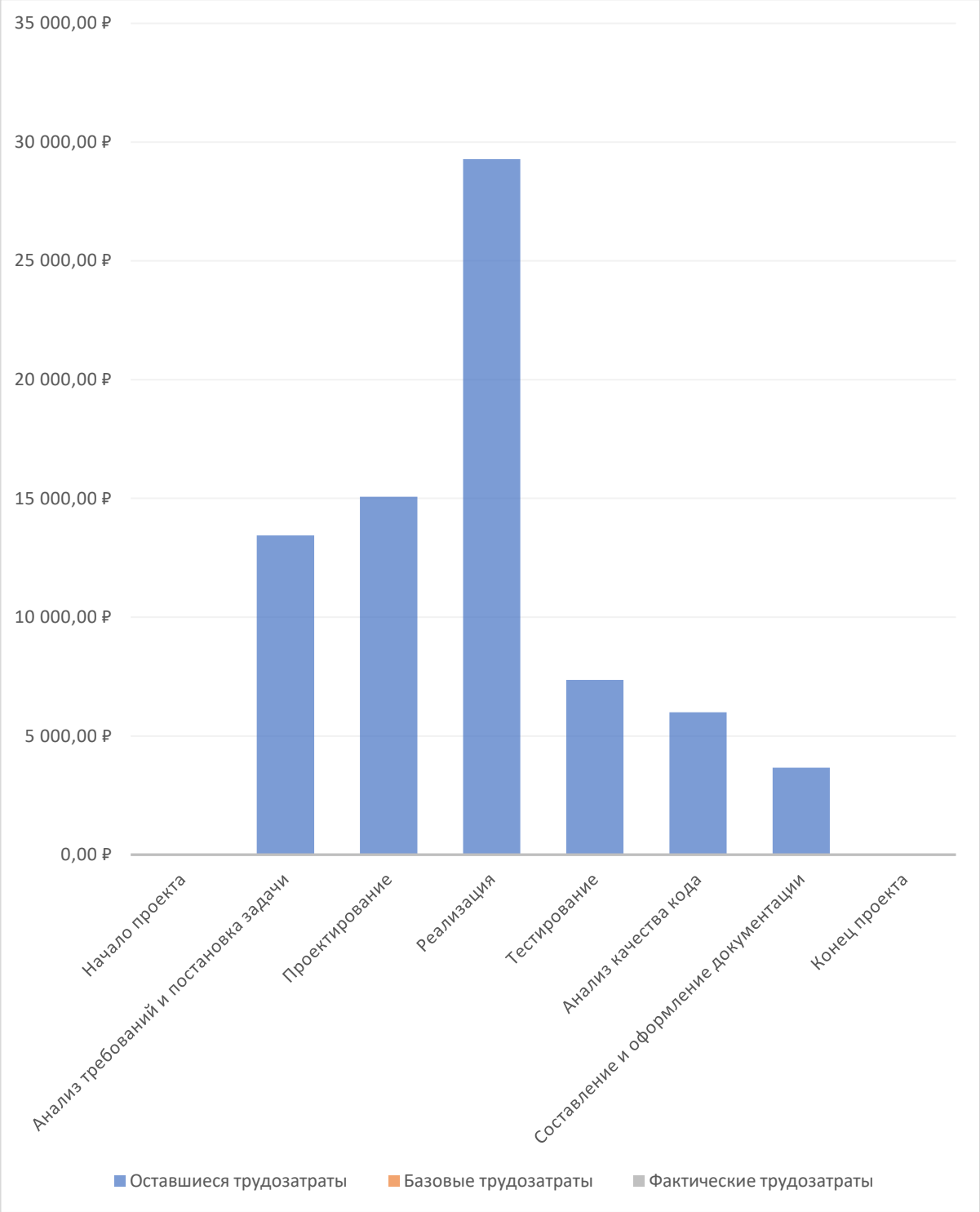


Рисунок 33 – Гистограмма затрат на задачи

В результате оценки бюджета примерная стоимость разработки составила 74812 рублей. Для разработки потребуется 457 человеко-часов. В условиях, указанных при планировании, разработка займет 48 дней.

Заключение

В ходе выполнения выпускной квалификационной работы разработан онлайн-сервис организации, проведения и обработки результатов психологического тестирования личности в образовательных учреждениях.

Проведен анализ предметной области. На основе сравнения имеющихся аналогов и требований задания сформулирована задача на разработку программного средства, а также определены функциональные и нефункциональные требования к будущей системе.

Программные средства спроектированы и реализованы в соответствие с MVC архитектурой. Для работы с базой данных разработан набор классов, работающий по принципу объектно-реляционного отображения.

Разработка программного средства проводилась в среде разработки IntelliJI IDEA на языке программирования Java в операционной системе Ubuntu 20.08. Тестирование программного средства проводилось в той же операционной системе. Все тесты пройдены успешно и в результате тестирования не было выявлено ошибок.

В ходе работы выполнены планирование разработки и оценка бюджета при помощи Microsoft Project. Длительность проекта составила 48 дней, а суммарные затраты равны 74812 рублей.

Таким образом в результате выполнения выпускной квалификационной работы разработан онлайн-сервис организации, проведения и обработки результатов психологического тестирования личности в образовательных учреждениях.

Список использованных источников

1. Печеркина, А.В. Проблемы организации социально-психологического тестирования обучающихся в условиях высшего учебного заведения [Текст] / А.В. Печеркина, С.Ю. Жданова, Л.З. Зарипова // Социальные и гуманитарные науки: теория и практика: сб. науч. тр. / ПГУ. – Пермь, 2020. – С. 395 – 402.

2. Общая характеристика психологического тестирования [Электронный ресурс]. URL: https://bookitut.ru/Eksperimentaljnaya-psikhologiya-konspekt-lekczij-1.32.html (дата обращения: 29.04.2022).

3. Верников Г., Основы методологии IDEF1X [Электронный ресурс]. URL: https://www.cfin.ru/vernikov/idef/idef1x.shtml (дата обращения 29.04.2022).

4. Уровни моделирования ИС [Электронный ресурс]. URL: https://dl.nure.ua/pluginfile.php/550/mod_resource/content/2/content/content4.html (дата обращения 04.05.2022).

5. Физическая модель базы данных [Электронный ресурс]. URL: https://neerc.ifmo.ru/wiki/index.php?title=Физическая_модель_базы_данных (дата обращения 05.05.2022).

6. What is Object/Relation mapping? [Электронный ресурс]. URL: https://hibernate.org/orm/what-is-an-orm/ (дата обращения 06.05.2022).

7. Бауэр К., Java Persistence API и Hibernate / К. Бауэр, Г. Кинг, Г. Грегори – Москва: ДМК Пресс, 2019. – 652с.

8. Что такое JSON [Электронный ресурс]. URL: https://habr.com/ru/post/554274/ (дата обращения 09.05.2022).

9. Функциональное тестирование программного обеспечения [Электронный ресурс]. URL: https://daglab.ru/funkcionalnoe-testirovanie-programmnogo-obespechenija/ (дата обращения 08.05.2022).

10. Нефункциональное тестирование [Электронный ресурс]. URL: https://xbsoftware.ru/testirovanie-po/polnij-tsykl/nefunctional-test/ (дата обращения 08.05.2022).

11. Для чего нужно интеграционное тестирование? [Электронный ресурс]. URL: https://habr.com/ru/post/556002/ (дата обращения 08.05.2022).

12. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения. — Санкт-Петербург.: Питер, 2018. — ISBN 978-5-4461-0772-8.

13. Как диаграммы Ганта упрощают работу с проектами [Электронный ресурс]. URL: https://habr.com/ru/company/hygger/blog/415271/ (дата обращения 1.05.2022).

Приложение А

ГЛОССАРИЙ

(справочное)

СУБД – система управления базами данных.

REST – Representation State Transfer (передача репрезентативного состояния), архитектурный стиль взаимодействия компонентов распределённого приложения в сети.

API – Application Programming Interface (интерфейс программирования приложений), совокупность инструментов и функций в виде интерфейса для создания новых приложений, благодаря которому одна программа будет взаимодействовать с другой.

PostgreSQL – свободная объектно-реляционная система управления базами данных.

Hibernate – библиотека для языка программирования Java, предназначенная для решения задач объектно-реляционного отображения.

PK – Primary Key (первичный ключ), в реляционной модели данных один из потенциальных ключей отношения, выбранный в качестве основного ключа (или ключа по умолчанию).

FK – Foreign Key (внешний ключ), понятие теории реляционных баз данных, относящееся к ограничениям целостности базы данных.

JPA – Java Persistence API, спецификация, предоставляющая возможность сохранять в удобно виде Java-объекты в базе данных.

POJO – Plain Old Java Object (старый добрый Java-объект), простой Java-объект, не унаследованный от какого-то специфического объекта и не реализующий никаких служебных интерфейсов сверх тех, которые нужны для бизнес-модели.

SQL – язык программирования, применяемый для управления базами данных.

JPQL – Java Persistence Query Language, платформенно-независимый объектно-ориентированный язык запросов, являющийся частью спецификации JPA.

JSON – JavaScript Object Notation, текстовый формат, полностью независимый от языка реализации.

IntelliJI IDEA – интегрированная среда разработки программного обеспечения для многих языков программирования, в частности Java.

DataGrip – инструмент от компании JetBrains для управления различными базами данных.

VirtualBox – программный продукт, применяемый для виртуализации операционных систем.

SQL – язык программирования, применяемый для управления базами данных.

UML – Unified Modeling Language (унифицированный язык моделирования).

Приложение Б

ЛИСТИНГ ПРОГРАММЫ

(рекомендуемое)

Листинг Б.1 – Файл Application.class

```
package may.code;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication
                .run(Application.class, args);
    }
}
```

Листинг Б.2 – Файл WebConfig.class

```
package may.code.configs;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import
org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

@Configuration
@EnableWebMvc
public class WebConfig extends WebMvcConfigurerAdapter {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
                .allowedMethods("HEAD", "GET", "PUT", "POST", "DELETE",
"PATCH");
    }
}
```

Листинг Б.3 – Файл AnswerEntity.class

```
package may.code.api.store.entities;

import lombok.*;
import lombok.experimental.FieldDefaults;

import javax.persistence.*;

@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@FieldDefaults(level = AccessLevel.PRIVATE)
@Entity
@Table(name = "answer")
public class AnswerEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    Integer id;

    Short answerOrder;
```

```
    @Column(length = 50)
    String text;

    @ManyToOne
    @JoinColumn(name = "question_id", referencedColumnName = "id")
    QuestionEntity question;

    @Column(name = "question_id", updatable = false, insertable = false)
    Long questionId;

    public static AnswerEntity makeDefault() {
        return builder().build();
    }
}
```

Листинг Б.4 – Файл PersonTemplateEntity.class

```
package may.code.api.store.entities package may.code.api.store.entities;

import com.vladmihalcea.hibernate.type.json.JsonBinaryType;
import lombok.*;
import lombok.experimental.FieldDefaults;
import may.code.api.domains.UserShouldAnswer;
import org.hibernate.annotations.Type;
import org.hibernate.annotations.TypeDef;

import javax.persistence.*;
import java.util.List;

@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@FieldDefaults(level = AccessLevel.PRIVATE)
@Entity
@Table(name = "person_template")
//https://ligerlearn.com/creating-java-jpa-entities-with-a-json-field/
@TypeDef(name = "jsonb", typeClass = JsonBinaryType.class)
public class PersonTemplateEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    Integer id;

    @Column(length = 1000)
    String text;

    @Type(type = "jsonb")
    @Column(columnDefinition = "jsonb", length = 20000)
    @JoinColumn(name = "test_id", referencedColumnName = "id")
    List<UserShouldAnswer> answers;

    @ManyToOne
    @JoinColumn(name = "test_id", referencedColumnName = "id")
    TestEntity test;
}
```

Листинг Б.5 – Файл PsychologistEntity.class

```
package may.code.api.store.entities;
```

```
import lombok.*;
import lombok.experimental.FieldDefaults;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.List;

@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@FieldDefaults(level = AccessLevel.PRIVATE)
@Entity
@Table(name = "psychologist")
public class PsychologistEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    Integer id;

    @Column(length = 300)
    @NonNull
    String fio;

    @Column(length = 64)
    @NonNull
    String login;

    @Column(length = 64)
    @NonNull
    String password;

    @ManyToOne
    @JoinColumn(name = "school_class_id", referencedColumnName = "id")
    SchoolClassEntity schoolClass;

    @Builder.Default
    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true, fetch =
FetchType.LAZY)
    @JoinColumn(name = "psychologist_id", referencedColumnName = "id")
    List<TestEntity> tests = new ArrayList<>();
}
```

Листинг Б.6 – Файл QuestionEntity.class

```
package may.code.api.store.entities;

import lombok.*;
import lombok.experimental.FieldDefaults;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.List;

@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@FieldDefaults(level = AccessLevel.PRIVATE)
@Entity
@Table(name = "question")
```

```
public class QuestionEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    Integer id;

    Short questionOrder;

    @Column(length = 256)
    String text;

    @ManyToOne
    @JoinColumn(name = "test_id", referencedColumnName = "id")
    TestEntity test;

    @Column(name = "test_id", updatable = false, insertable = false)
    Long testId;

    @Builder.Default
    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true, fetch =
FetchType.LAZY)
    @JoinColumn(name = "question_id", referencedColumnName = "id")
    List<AnswerEntity> answers = new ArrayList<>();

    public static QuestionEntity makeDefault() {
        return builder().build();
    }
}
```

Листинг Б.7 – Файл SchoolClassEntity.class

```
package may.code.api.store.entities;

import lombok.*;
import lombok.experimental.FieldDefaults;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.List;

@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@FieldDefaults(level = AccessLevel.PRIVATE)
@Entity
@Table(name = "school_class")
public class SchoolClassEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    Integer id;

    @Column(length = 8)
    @NonNull
    String name;

    @Builder.Default
    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true, fetch =
FetchType.LAZY)
    @JoinColumn(name = "school_class_id", referencedColumnName = "id")
    List<TestedUserEntity> testedUsers = new ArrayList<>();
```

```
    @NonNull
    @ManyToOne
    @JoinColumn(name = "school_id", referencedColumnName = "id")
    SchoolEntity school;

    @Column(name = "school_id", updatable = false, insertable = false)
    Integer schoolId;

    public static SchoolClassEntity makeDefault(String name, SchoolEntity
school) {
        return builder()
                .name(name)
                .school(school)
                .build();
    }
}
```

Листинг Б.8 – Файл SchoolEntity.class

```
package may.code.api.store.entities;

import lombok.*;
import lombok.experimental.FieldDefaults;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.List;

@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@FieldDefaults(level = AccessLevel.PRIVATE)
@Entity
@Table(name = "school")
public class SchoolEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    Integer id;

    @Column(length = 128)
    @NonNull
    String name;

    @Builder.Default
    @OneToMany
    @JoinColumn(name = "school_id", referencedColumnName = "id")
    List<SchoolClassEntity> schoolClasses = new ArrayList<>();

    public static SchoolEntity makeDefault(String schoolName) {

        return builder()
                .name(schoolName)
                .build();
    }
}
```

Листинг Б.9 – Файл TestAnswerEntity.class

```
package may.code.api.store.entities;
```

```
import com.vladmihalcea.hibernate.type.json.JsonBinaryType;
import lombok.*;
import lombok.experimental.FieldDefaults;
import may.code.api.domains.TestedUserAnswer;
import org.hibernate.annotations.Type;
import org.hibernate.annotations.TypeDef;

import javax.persistence.*;
import java.time.Instant;
import java.util.List;

@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@FieldDefaults(level = AccessLevel.PRIVATE)
@Entity
@Table(name = "test_answer")
//https://ligerlearn.com/creating-java-jpa-entities-with-a-json-field/
@TypeDef(name = "jsonb", typeClass = JsonBinaryType.class)
public class TestAnswerEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    Integer id;

    @NonNull
    @ManyToOne
    TestedUserEntity testedUser;

    @NonNull
    @ManyToOne
    @JoinColumn(name = "test_id", referencedColumnName = "id")
    TestEntity test;

    @Type(type = "jsonb")
    @Column(columnDefinition = "jsonb", length = 20000)
    List<TestedUserAnswer> answers;

    Instant startAt;

    @Builder.Default
    Instant endAt = Instant.now();
}
```

Листинг Б.10 – Файл TestedUserEntity.class

```
package may.code.api.store.entities;

import lombok.*;
import lombok.experimental.FieldDefaults;
import may.code.api.domains.TestedUserStatus;

import javax.persistence.*;
import java.time.Instant;
import java.util.ArrayList;
import java.util.List;

@Getter
@Setter
@Builder
```

```
@NoArgsConstructor
@AllArgsConstructor
@FieldDefaults(level = AccessLevel.PRIVATE)
@Entity
@Table(name = "tested_user")
public class TestedUserEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    Integer id;

    @Column(length = 64)
    @NonNull
    String firstName;

    @Column(length = 64)
    @NonNull
    String lastName;

    @Column(length = 64)
    String middleName;

    @NonNull
    Instant birthday;

    @Column(length = 64)
    @NonNull
    @Enumerated(EnumType.STRING)
    TestedUserStatus status;

    @Column(length = 128)
    @NonNull
    String login;

    @Builder.Default
    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true, fetch =
FetchType.LAZY)
    @JoinColumn(name = "tested_user_id", referencedColumnName = "id")
    List<TestAnswerEntity> testedUserAnswers = new ArrayList<>();

    @Column(length = 10)
    @NonNull
    String password;

    @NonNull
    @ManyToOne
    @JoinColumn(name = "school_class_id", referencedColumnName = "id")
    SchoolClassEntity schoolClass;

    public static TestedUserEntity makeDefault(
            String firstName,
            String middleName,
            String lastName,
            String login,
            String password,
            Instant birthday,
            TestedUserStatus status,
            SchoolClassEntity schoolClass) {
        return builder()
                .firstName(firstName)
                .middleName(middleName)
                .lastName(lastName)
                .login(login)
                .password(password)
```

```
                .birthday(birthday)
                .status(status)
                .schoolClass(schoolClass)
                .build();
    }
}
```

Листинг Б.11 – Файл TestEntity.class

```
package may.code.api.store.entities;

import lombok.*;
import lombok.experimental.FieldDefaults;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.List;

@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@FieldDefaults(level = AccessLevel.PRIVATE)
@Entity
@Table(name = "test")
public class TestEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    Integer id;

    @Column(length = 256)
    String name;

    @Builder.Default
    Boolean status = false;

    @ManyToOne
    @JoinColumn(name = "psychologist_id", referencedColumnName = "id")
    PsychologistEntity psychologist;

    @Builder.Default
    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true, fetch =
FetchType.LAZY)
    @JoinColumn(name = "test_id", referencedColumnName = "id")
    List<PersonTemplateEntity> personTemplates = new ArrayList<>();

    @Builder.Default
    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true, fetch =
FetchType.LAZY)
    @JoinColumn(name = "test_id", referencedColumnName = "id")
    List<TestAnswerEntity> testAnswers = new ArrayList<>();

    @Builder.Default
    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true, fetch =
FetchType.LAZY)
    @JoinColumn(name = "test_id", referencedColumnName = "id")
    List<QuestionEntity> questions = new ArrayList<>();

    public static TestEntity makeDefault() {
        return builder().build();
    }
```

```
}
```

Листинг Б.12 – Файл TestAnswerRepository.class

```
package may.code.api.store.repositories;

import may.code.api.domains.IPersonTemplateAnalyze;
import may.code.api.store.entities.TestAnswerEntity;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

import java.util.List;

public interface TestAnswerRepository extends JpaRepository<TestAnswerEntity,
Integer> {

    @Query(value =
            "SELECT pt.id AS personTemplateId, " +
                    "TOTAL_COUNT(pt.id) AS totalAnswers, " +
                    "INTERSECTIONS_COUNT(pt.answers, ta.answers) AS
testedUserAnswers, " +
                    "ta.id AS testAnswerId " +
                    "FROM test_answer ta " +
                        "INNER JOIN tested_user tu ON ta.tested_user_id =
tu.id " +
                        "INNER JOIN school_class sc ON tu.school_class_id
= :schoolClassId " +
                        "INNER JOIN test t ON ta.test_id = :testId " +
                        "INNER JOIN person_template pt ON t.id = pt.test_id "
+
                        "GROUP BY ta.id, pt.id",
            nativeQuery = true
    )
    List<IPersonTemplateAnalyze> analyzePersonTemplates(Integer
schoolClassId, Integer testId);
}
```

Листинг Б.13 – Файл TestService.class

```
package may.code.api.services;

import lombok.AccessLevel;
import lombok.RequiredArgsConstructor;
import lombok.experimental.FieldDefaults;
import may.code.api.domains.IPersonTemplateAnalyze;
import may.code.api.dto.PersonAnalyzeDto;
import may.code.api.exeptions.AccessDeniedException;
import may.code.api.exeptions.BadRequestException;
import may.code.api.exeptions.NotFoundException;
import may.code.api.store.entities.PsychologistEntity;
import may.code.api.store.entities.SchoolClassEntity;
import may.code.api.store.entities.TestEntity;
import may.code.api.store.repositories.TestAnswerRepository;
import may.code.api.store.repositories.TestRepository;
import org.springframework.stereotype.Service;

import java.util.*;

@Service
@RequiredArgsConstructor
@FieldDefaults(level = AccessLevel.PRIVATE, makeFinal = true)
public class TestService {
```

```
    TestRepository testRepository;

    TestAnswerRepository testAnswerRepository;

    public static final String LINK_TEMPLATE = "/test/%s/class/%s";

    public Map<Integer, List<PersonAnalyzeDto>>
getTestAnswerIdToPersonAnalyze(
            SchoolClassEntity schoolClass,
            TestEntity test) {

        List<IPersonTemplateAnalyze> personAnalyzes = testAnswerRepository
                .analyzePersonTemplates(schoolClass.getId(), test.getId());

        Map<Integer, List<PersonAnalyzeDto>> testAnswerIdToPersonAnalyze =
new HashMap<>();

        personAnalyzes
                .forEach(personAnalyze ->
                        testAnswerIdToPersonAnalyze
                                .computeIfAbsent(personAnalyze.getTestAnswerI
d(), key -> new ArrayList<>())
                                .add(PersonAnalyzeDto.builder()
                                        .testedUserAnswers(personAnalyze.getT
estedUserAnswers())
                                        .totalAnswers(personAnalyze.getTotalA
nswers())
                                        .personTemplateId(personAnalyze.getPe
rsonTemplateId())
                                        .build()
                                )
                );
        return testAnswerIdToPersonAnalyze;
    }

    public String generateLinkForTest(PsychologistEntity psychologist,
Integer testId) {

        getTestOrThrowException(psychologist, testId);

        SchoolClassEntity schoolClass = psychologist.getSchoolClass();

        if (Objects.isNull(schoolClass)) {

            throw new BadRequestException(
                    "Вы не можете сгенерировать ссылку, так как не привязаны
ни к одному классу."
            );
        }

        return generateLink(testId, schoolClass.getId());
    }

    public static String generateLink(Integer testId, Integer schoolClassId)
{

        if (Objects.isNull(testId) || testId < 0) {
            throw new IllegalArgumentException("Поле testId не может быть
пустым или меньше 0.");
        }

        if (Objects.isNull(schoolClassId) || schoolClassId < 0) {
            throw new IllegalArgumentException("Поле schoolClassId не может
быть пустым или меньше 0.");
```

```
        }

        return String.format(LINK_TEMPLATE, testId, schoolClassId);
    }

    public TestEntity getTestOrThrowException(PsychologistEntity
psychologist, Integer testId) {

        TestEntity test = testRepository
                .findById(testId)
                .orElseThrow(() ->
                        new NotFoundException(String.format("Тест с
идентификатором \"%s\" не найден.", testId))
                );

        if (!Objects.equals(test.getPsychologist().getId(),
psychologist.getId())) {
            throw new AccessDeniedException("Вы не можете получить доступ к
этому тесту.");
        }

        return test;
    }
}
```

Листинг Б.14 – Файл IPersonTemplateAnalyze.class

```
package may.code.api.domains;

public interface IPersonTemplateAnalyze {
    Integer getPersonTemplateId();
    Integer getTotalAnswers();
    Integer getTestedUserAnswers();
    Integer getTestAnswerId();
}
```

Листинг Б.15 – Файл TestedUserAnswer.class

```
package may.code.api.domains;

import lombok.*;
import lombok.experimental.FieldDefaults;

import java.io.Serializable;

@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@FieldDefaults(level = AccessLevel.PRIVATE)
public class TestedUserAnswer implements Serializable {

    Integer question_id;

    Integer answer_id;

    Long created_at;
}
```

Листинг Б.16 – Файл TestedUserStatus.class

```
package may.code.api.domains;

public enum TestedUserStatus {
    STUDENT,
    PARENT,
    TEACHER
}
```

Листинг Б.17 – Файл UserShouldAnswer.class

```
package may.code.api.domains;

import lombok.*;
import lombok.experimental.FieldDefaults;

import java.io.Serializable;

@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@FieldDefaults(level = AccessLevel.PRIVATE)
public class UserShouldAnswer implements Serializable {

    Integer question_id;

    Integer answer_id;
}
```

Листинг Б.18 – Файл TestController.class

```
package may.code.api.controller;

import lombok.AccessLevel;
import lombok.RequiredArgsConstructor;
import lombok.experimental.FieldDefaults;
import lombok.var;
import may.code.api.domains.TestedUserAnswer;
import may.code.api.domains.UserShouldAnswer;
import may.code.api.dto.*;
import may.code.api.exeptions.BadRequestException;
import may.code.api.exeptions.NotFoundException;
import may.code.api.factory.TestDtoFactory;
import may.code.api.services.ControllerAuthenticationService;
import may.code.api.store.entities.*;
import may.code.api.store.repositories.TestAnswerRepository;
import may.code.api.store.repositories.TestRepository;
import may.code.api.store.repositories.TestedUserRepository;
import org.springframework.web.bind.annotation.*;

import javax.transaction.Transactional;
import java.time.Instant;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Objects;
import java.util.stream.Collectors;

@RequiredArgsConstructor
```

```
@FieldDefaults(makeFinal = true, level = AccessLevel.PRIVATE)
@RestController
@Transactional
public class TestController {

    TestRepository testRepository;

    TestedUserRepository testedUserRepository;

    TestAnswerRepository testAnswerRepository;

    TestDtoFactory testDtoFactory;

    ControllerAuthenticationService authenticationService;

    public static final String FETCH_TESTS = "/api/tests";
    public static final String GET_TEST = "/api/tests/{testId}";
    public static final String CREATE_OR_UPDATE_TEST = "/api/tests";
    public static final String UPDATE_PERSON_TEMPLATES =
"/api/tests/{testId}/person-templates";
    public static final String DELETE_TEST = "/api/tests/{testId}";
    public static final String COMPLETE_TEST = "/api/tested-
users/{testedUserId}/tests/{testId}/compete";

    @GetMapping(FETCH_TESTS)
    public List<TestDto> fetchTests(@RequestParam(defaultValue = "") String
filter,
                                    @RequestHeader(defaultValue = "") String
token) {

        PsychologistEntity psychologist =
authenticationService.authenticate(token);

        boolean isFiltered = !filter.trim().isEmpty();

        List<TestEntity> tests = testRepository.findAllByFilter(isFiltered,
filter, psychologist.getId());

        return testDtoFactory.createTestDtoList(tests);
    }

    @GetMapping(GET_TEST)
    public TestDto getTest(@PathVariable Integer testId) {

        TestEntity test = testRepository
                .findById(testId)
                .orElseThrow(() ->
                        new NotFoundException(String.format("Тест с
идентификатором \"%s\" не найден.", testId))
                );

        return testDtoFactory.createTestDto(test);
    }

    @PostMapping(CREATE_OR_UPDATE_TEST)
    public TestDto createOrUpdateTest(@RequestBody TestDto test,
                                      @RequestHeader(defaultValue = "")
String token) {

        PsychologistEntity psychologist =
authenticationService.authenticate(token);

        TestEntity testEntity = convertTestToEntity(test);
```

```
        testEntity = testRepository.saveAndFlush(testEntity);

        testEntity.setPsychologist(psychologist);

        return testDtoFactory.createTestDto(testEntity);
    }

    @PostMapping(UPDATE_PERSON_TEMPLATES)
    public TestDto updatePersonTemplates(
            @PathVariable Integer testId,
            @RequestBody List<PersonTemplateDto> personTemplates,
            @RequestHeader(defaultValue = "") String token) {

        authenticationService.authenticate(token);

        TestEntity test = testRepository
                .findById(testId)
                .orElseThrow(() ->
                        new NotFoundException(String.format("Тест с
идентификатором \"%s\" не найден.", testId))
                );

        test.getPersonTemplates().clear();
        test.getPersonTemplates().addAll(
               personTemplates
                       .stream()
                       .map(this::convertToPersonTemplateEntity)
                       .collect(Collectors.toList())
        );

        for (PersonTemplateEntity it: test.getPersonTemplates()) {
            it.setTest(test);
        }

        test = testRepository.saveAndFlush(test);

        return testDtoFactory.createTestDto(test);
    }

    @DeleteMapping(DELETE_TEST)
    public AckDto deleteTest(@PathVariable Integer testId,
                             @RequestHeader(defaultValue = "") String token)
{

        PsychologistEntity psychologist =
authenticationService.authenticate(token);

        TestEntity test = psychologist
                .getTests()
                .stream()
                .filter(it -> Objects.equals(it.getId(), testId))
                .findAny()
                .orElseThrow(() ->
                        new NotFoundException(String.format("Тест с \"%s\"
идентификатором не найден.", testId))
                );

        if (test.getStatus()) {
            throw new BadRequestException("Невозможно удалить тест, сначала
необходимо завершить тестирование.");
        }

        test.getQuestions().forEach(it -> it.getAnswers().clear());
        test.getQuestions().clear();
```

```
        test = testRepository.saveAndFlush(test);

        testRepository.delete(test);

        return AckDto.makeDefault(true);
    }

    @PostMapping(COMPLETE_TEST)
    public AckDto completeTest(@PathVariable Integer testedUserId,
                               @PathVariable Integer testId,
                               @RequestParam Instant startAt,
                               @RequestBody List<TestedUserAnswerDto>
testedUserAnswers) {

        TestEntity test = getTestOrThrowException(testId);

        if (testedUserAnswers.size() != test.getQuestions().size()) {
            throw new BadRequestException("Вы ответили не на все вопросы.");
        }

        checkAllAnswers(testedUserAnswers, test);

        TestedUserEntity user = testedUserRepository
                .findById(testedUserId)
                .orElseThrow(() ->
                        new NotFoundException(
                                String.format("Пользователь с идентификатором
\"%s\" не найден.", testedUserId)
                        )
                );

        var answers = testedUserAnswers
                .stream()
                .map(it ->
                        TestedUserAnswer.builder()
                                .answer_id(it.getAnswerId())
                                .question_id(it.getQuestionId())
                                .created_at(it.getCreatedAt().toEpochMilli())
                                .build()
                )
                .collect(Collectors.toList());

        testAnswerRepository.saveAndFlush(
                TestAnswerEntity.builder()
                        .answers(answers)
                        .startAt(startAt)
                        .testedUser(user)
                        .test(test)
                        .build()
        );

        return AckDto.makeDefault(true);
    }

    private void checkAllAnswers(List<TestedUserAnswerDto> testedUserAnswers,
TestEntity test) {

        Map<Integer, Integer> questionIdToAnswerIdMap = new HashMap<>();

        testedUserAnswers
                .forEach(testedUserAnswer ->
```

```
questionIdToAnswerIdMap.put(testedUserAnswer.getQuestionId(),
testedUserAnswer.getAnswerId())
                );

        test
                .getQuestions()
                .forEach(question -> {

                    if
(Objects.isNull(questionIdToAnswerIdMap.get(question.getId()))) {
                        throw new BadRequestException("Вы ответили не на все
вопросы.");
                    }
                });
    }

    private TestEntity getTestOrThrowException(Integer testId) {
        return testRepository
                .findById(testId)
                .orElseThrow(() ->
                        new NotFoundException(String.format("Тест с
идентификатором \"%s\" не найден.", testId))
                );
    }

    private TestEntity convertTestToEntity(TestDto dto) {

        Integer testId = dto.getId();

        TestEntity test;
        if (testId == null) {
            test = TestEntity.makeDefault();
        } else {
            test = testRepository
                    .findById(testId)
                    .orElseThrow(() ->
                            new NotFoundException(String.format("Тест с
индентификатором \"%s\" не найден.", testId))
                    );
        }

        test.setName(dto.getName());
        test.getQuestions().clear();

        test.getQuestions().addAll(
                dto.getQuestions()
                        .stream()
                        .map(this::convertQuestionToEntity)
                        .collect(Collectors.toList())
        );

        return test;
    }

    private PersonTemplateEntity
convertToPersonTemplateEntity(PersonTemplateDto dto) {

        return PersonTemplateEntity.builder()
                .id(dto.getId())
                .text(dto.getText())
                .answers(
                        dto.getUserShouldAnswers()
                                .stream()
```

```
                                .map(it ->
                                        UserShouldAnswer.builder()
                                                .answer_id(it.getAnswerId())
                                                .question_id(it.getQuestionId
())
                                                .build()
                                )
                                .collect(Collectors.toList())
                )
                .build();

    }

    private QuestionEntity convertQuestionToEntity(QuestionDto dto) {

        QuestionEntity question = QuestionEntity.makeDefault();

        question.setId(dto.getId());
        question.setQuestionOrder(dto.getOrder());
        question.setText(dto.getText());
        question.getAnswers().clear();

        question.getAnswers().addAll(
                dto.getAnswers()
                        .stream()
                        .map(this::convertAnswerToEntity)
                        .collect(Collectors.toList())
        );

        return question;
    }

    private AnswerEntity convertAnswerToEntity(AnswerDto dto) {

        AnswerEntity answer = AnswerEntity.makeDefault();

        answer.setId(dto.getId());
        answer.setText(dto.getText());
        answer.setAnswerOrder(dto.getOrder());

        return answer;
    }
}
```

Листинг Б.19 – Файл TestedUserController.class

```
package may.code.api.controller;

import com.ibm.icu.text.Transliterator;
import lombok.AccessLevel;
import lombok.RequiredArgsConstructor;
import lombok.experimental.ExtensionMethod;
import lombok.experimental.FieldDefaults;
import lombok.extern.log4j.Log4j2;
import may.code.api.domains.TestedUserStatus;
import may.code.api.dto.TestedUserDto;
import may.code.api.exeptions.BadRequestException;
import may.code.api.exeptions.NotFoundException;
import may.code.api.factory.TestedUserDtoFactory;
import may.code.api.services.ControllerAuthenticationService;
import may.code.api.store.entities.PsychologistEntity;
import may.code.api.store.entities.SchoolClassEntity;
import may.code.api.store.entities.TestedUserEntity;
```

```
import may.code.api.store.repositories.TestedUserRepository;
import may.code.api.utils.StringChecker;
import org.springframework.web.bind.annotation.*;

import javax.transaction.Transactional;
import java.time.Instant;
import java.util.List;
import java.util.Objects;
import java.util.UUID;

@Log4j2
@RequiredArgsConstructor
@ExtensionMethod(StringChecker.class)
@FieldDefaults(makeFinal = true, level = AccessLevel.PRIVATE)
@RestController
@Transactional
public class TestedUserController {

    TestedUserRepository testedUserRepository;

    TestedUserDtoFactory testedUserDtoFactory;

    ControllerAuthenticationService authenticationService;

    public static final String AUTHORIZE = "/api/tested-users/authorize";
    public static final String GET_SCHOOL_CLASS_TESTED_USERS = "/api/tested-
users";
    public static final String CREATE_USER = "/api/tested-users";

    private static final String CYRILLIC_TO_LATIN = "Russian-Latin/BGN";

    private static final Transliterator toLatinTrans =
Transliterator.getInstance(CYRILLIC_TO_LATIN);

    private static final int PASSWORD_LENGTH = 10;

    @GetMapping(AUTHORIZE)
    public Integer authorize(
            @RequestParam String login,
            @RequestParam String password) {

        return testedUserRepository
                .findTopByLoginAndPassword(login, password)
                .map(TestedUserEntity::getId)
                .orElseThrow(() -> new NotFoundException("Пользователь с
таким логином и паролем не существует."));
    }

    @GetMapping(GET_SCHOOL_CLASS_TESTED_USERS)
    public List<TestedUserDto> fetchUsers(
            @RequestHeader(defaultValue = "") String token) {

        PsychologistEntity psychologist =
authenticationService.authenticate(token);

        SchoolClassEntity schoolClass = psychologist.getSchoolClass();

        if (Objects.isNull(schoolClass)) {
            throw new BadRequestException("Вы не привязаны ни к одному из
классов.");
        }

        List<TestedUserEntity> users = schoolClass.getTestedUsers();
```

```
        return testedUserDtoFactory.createTestedUserDtoList(users);
    }

    @PostMapping(CREATE_USER)
    public TestedUserDto createUser(
            @RequestParam Instant birthday,
            @RequestParam String firstName,
            @RequestParam(defaultValue = "") String middleName,
            @RequestParam String lastName,
            @RequestParam TestedUserStatus testedUserStatus,
            @RequestHeader(defaultValue = "") String token) {

        PsychologistEntity psychologist =
authenticationService.authenticate(token);

        SchoolClassEntity schoolClass = psychologist.getSchoolClass();

        if (Objects.isNull(schoolClass)) {
            throw new BadRequestException(
                    "Вы не можете создать пользователя, " +
                            "так как не привязаны ни к одному из классов"
            );
        }

        firstName = firstName.trim();
        lastName = lastName.trim();

        firstName.checkOnEmpty("firstName");
        lastName.checkOnEmpty("lastName");
        middleName = middleName.trim().isEmpty() ? null : middleName;

        String login = makeLogin(firstName, lastName);
        String password = makePassword();

        TestedUserEntity user = testedUserRepository.saveAndFlush(
                TestedUserEntity.makeDefault(
                        firstName,
                        middleName,
                        lastName,
                        login,
                        password,
                        birthday,
                        testedUserStatus,
                        schoolClass
                )
        );

        return testedUserDtoFactory.createTestedUserDto(user);
    }

    public static String makeLogin(String firstName, String lastName) {

        String firstNameTransliterated =
toLatinTrans.transliterate(firstName.toLowerCase());

        String lastNameTransliterated =
toLatinTrans.transliterate(lastName.toLowerCase());

        return String.format("%s.%s", firstNameTransliterated.charAt(0),
lastNameTransliterated);
    }

    public static String makePassword() {
```

```
        return UUID.randomUUID().toString().replaceAll("-", "").substring(0,
PASSWORD_LENGTH);
    }
}
```

Листинг Б.20 – Файл PsychologistController.class

```
package may.code.api.controller;

import lombok.AccessLevel;
import lombok.RequiredArgsConstructor;
import lombok.experimental.FieldDefaults;
import may.code.api.dto.PersonAnalyzeDto;
import may.code.api.dto.TestAnswerDto;
import may.code.api.exeptions.BadRequestException;
import may.code.api.factory.TestAnswerDtoFactory;
import may.code.api.services.ControllerAuthenticationService;
import may.code.api.services.TestService;
import may.code.api.store.entities.PsychologistEntity;
import may.code.api.store.entities.SchoolClassEntity;
import may.code.api.store.entities.TestEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestHeader;
import org.springframework.web.bind.annotation.RestController;

import javax.transaction.Transactional;
import java.util.List;
import java.util.Map;
import java.util.Objects;

@RequiredArgsConstructor
@FieldDefaults(makeFinal = true, level = AccessLevel.PRIVATE)
@RestController
@Transactional
public class PsychologistController {

    TestService testService;

    TestAnswerDtoFactory testAnswerDtoFactory;

    ControllerAuthenticationService authenticationService;

    public static final String GET_TEST_ANSWERS =
"/api/psychologists/tests/{testId}/results";
    public static final String GENERATE_LINK_FOR_TEST =
"/api/psychologists/tests/{testId}/generate-link";


    @GetMapping(GET_TEST_ANSWERS)
    public List<TestAnswerDto> getTestAnswers(@PathVariable Integer testId,
                                              @RequestHeader(defaultValue =
"") String token) {

        PsychologistEntity psychologist =
authenticationService.authenticate(token);

        SchoolClassEntity schoolClass = psychologist.getSchoolClass();

        if (Objects.isNull(schoolClass)) {
            throw new BadRequestException("Вы не привязаны ни к одному
классу.");
        }
```

```
        TestEntity test = testService.getTestOrThrowException(psychologist,
testId);

        Map<Integer, List<PersonAnalyzeDto>> testAnswerIdToPersonAnalyze =
testService
                .getTestAnswerIdToPersonAnalyze(schoolClass, test);

        return
testAnswerDtoFactory.createTestAnswerDtoList(test.getTestAnswers(),
testAnswerIdToPersonAnalyze);
    }

    @GetMapping(GENERATE_LINK_FOR_TEST)
    public String generateLinkForTest(@PathVariable Integer testId,
@RequestHeader(defaultValue = "") String token) {

        PsychologistEntity psychologist =
authenticationService.authenticate(token);

        return testService.generateLinkForTest(psychologist, testId);
    }
}
```