



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий
Кафедра инструментального и прикладного программного обеспечения

РАБОТА ДОПУЩЕНА К ЗАЩИТЕ

Заведующий
кафедрой _____ Р. Г. Болбаков

« ____ » _____ 2022 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
по направлению подготовки бакалавров 09.03.04 Программная инженерия

На тему: Мобильное приложение на основе Чистой архитектуры для контроля
ИТ-проектов

Обучающийся


подпись

Лаухина Александра Сергеевна
Фамилия, имя, отчество

шифр 18И0467
группа ИКБО-01-18

Руководитель работы


подпись

к.т.н., доцент

Алпатов А.Н.

Консультант


подпись

к.ю.н., доцент,
доцент

Филаткина А.П.

Москва 2022 г.



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий

Кафедра инструментального и прикладного программного обеспечения

СОГЛАСОВАНО

Заведующий
кафедрой

подпись

Болбаков Роман Геннадьевич

«20» апреля 2022 г.

УТВЕРЖДАЮ

Директор
института

подпись

Зуев Андрей Сергеевич

«20» апреля 2022 г.

ЗАДАНИЕ

на выполнение выпускной квалификационной работы бакалавра

Обучающийся

Лаухина Александра Сергеевна

Фамилия Имя Отчество

Шифр

18И0467

Направление
подготовки

09.03.04

индекс направления

Программная инженерия

наименование направления

Группа

ИКБО-01-18

1. Тема выпускной квалификационной работы: Мобильное приложение на основе Чистой архитектуры для контроля ИТ-проектов




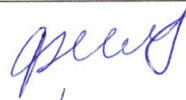
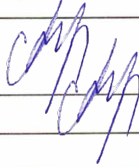
2. Цель и задачи выпускной квалификационной работы

Цель работы: спроектировать, положив в основу Чистую архитектуру, и разработать приложение для контроля ИТ-проектов.

Задачи работы: провести анализ предметной области, в том числе аналогов разрабатываемого программного приложения и архитектур, используемых для мобильной разработки; определить информационные процессы предметной области и формализовать их; формализовать задачи на проектирование и разработку приложения; спроектировать приложение для контроля ИТ-проектов (архитектуру, интерфейс и базу данных); определить и обосновать информационные, технические, программные средства для разработки приложения; разработать приложение для контроля ИТ-проектов; произвести расчет вычислительной и ёмкостной сложности приложения в наихудшем случае; произвести тестирование приложения; рассчитать экономическую эффективность и


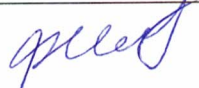
стоимость проведения работ; оформить пояснительную записку согласно ГОСТ 7.32-2017 (Отчет о научно-исследовательской работе).


3. Этапы выпускной квалификационной работы

№ этапа	Содержание этапа выпускной квалификационной работы	Результат выполнения этапа ВКР	Срок выполнения
1	Исследовательский раздел		27.04.2022
1.1	Обобщённая характеристика предметной области		
1.2	Поиск и анализ аналогов в среде приложений навигации и приложений дополненной реальности		
1.3	Поиск и анализ архитектур, применяемых в мобильной разработке		
1.4	Определение информационных процессов предметной области		
1.5	Постановка задач к проектированию и разработке приложения		
2	Проектный раздел		04.05.2022
2.1	Проектирование архитектуры приложения		
2.2	Проектирование мобильного клиентского приложения для контроля ИТ-проектов		
2.3	Проектирование схемы базы данных		
2.4	Выбор средств разработки и видов обеспечения		
3	Технологический раздел		11.05.2022
3.1	Разработка интерфейса мобильного приложения		
3.2	Разработка клиентского мобильного приложения для контроля ИТ-проектов		
3.3	Разработка серверного приложения для контроля ИТ-проектов		
3.4	Расчёт вычислительной и ёмкостной сложности		
3.5	Тестирование приложения		
4	Экономический раздел		17.05.2022
4.1	Организация и планирование работ по теме		
4.2	Расчет стоимости проведения работ по теме		25.05.2022
5	Введение, заключение, список источников, приложения		26.05.2022
6	Презентация		27.05.2022
7	Нормоконтроль		


4. Перечень разрабатываемых документов и графических материалов: печатная и электронная версии выпускной квалификационной работы бакалавра, презентационный материал с основными результатами выпускной квалификационной работы бакалавра.

5. Руководитель и консультант выпускной квалификационной работы

Функциональные обязанности	Должность в Университете	Фамилия, имя, отчество	Подпись
Руководитель ВКР	к.т.н., доцент	Алпатов Алексей Николаевич	
Консультант по экономическому разделу	к.ю.н., доцент	Филаткина Анна Павловна	

Задание выдал
Руководитель ВКР:  подпись

«20» апреля 2022 г.

Задание принял к исполнению
Обучающийся:  подпись

«20» апреля 2022 г.

Руководитель ВКР: к.т.н., доцент А.Н. Алпатов

Консультант по экономическому разделу: к.ю.н., доцент А.П. Филаткина

Лаухина А.С. Выпускная квалификационная работа направления профессиональной подготовки бакалавриата 09.03.04 «Программная инженерия» по профилю «Разработка программных продуктов и проектирование информационных систем» на тему «Мобильное приложение на основе Чистой архитектуры для контроля ИТ-проектов» / Руководитель: Алпатов А.Н., к.т.н., доцент / : М. 2022 г., МИРЭА – Российский технологический университет (РТУ МИРЭА), Институт информационных технологий (ИИТ), кафедра инструментального и прикладного программного обеспечения (ИиППО) – 50 стр., 19 илл., 7 табл., 9 формул, 33 ист. лит (в т.ч. 9 на английском яз.), 3 прил.

Ключевые слова: *архитектура программного обеспечения, Чистая архитектура, REST API, трехуровневая архитектура, Spring.*

Целью работы является разработка мобильного приложения для контроля ИТ-проектов. Были проведены исследования рынка систем управления проектами и архитектур, применяемых при мобильной разработке. Также была спроектирована инновационная архитектура, основой которой послужила Чистая архитектура. Разработаны база данных, REST API сервера, мобильное клиентское приложение. Описание полученной работы было опубликовано в статье «Проектирование архитектуры мобильного приложения для управления проектами» в научном журнале «Инновации. Наука. Образование», индексируемом в РИНЦ.

Laukhina A.S. Final qualifying work of the bachelor's degree training direction 09.03.04 "Software engineering" of the profile "Software product development and information systems design" on the topic "Mobile application based on the Clean architecture for IT project control" / Supervisor: Alpatov A.N., Candidate of Technical Sciences, docent /: M. 2022, MIREA - Russian Technological University (RTU MIREA), Institute of Information Technologies (IIT), Department of the Tool and Applied Software (Department of TAS) - 50 pages, 19 illustrations, 7 tables, 9 formulas, 33 ist. Litas (incl. foreign 9), 3 adj

Keywords: *software architecture, the Clean architecture, REST API, three-level architecture, Spring.*

The purpose of the work is to develop a mobile application for monitoring IT projects. Market research of project management systems and research of architectures used in mobile development were conducted. A new architecture based on the Clean Architecture was also designed. A database, a server REST API, and a mobile client application have been also developed. The description of the architecture was published in a scientific journal indexed in the RSCI.

РТУ МИРЭА: 119454, Москва, пр-т Вернадского, д. 78

кафедра инструментального и прикладного программного обеспечения (ИиППО)

Тираж: 1 экз. (на правах рукописи)

Файл: 090304_18И0467_Лаухина.docx, исполнитель Лаухина А.С.

© А.С. Лаухина

СОДЕРЖАНИЕ

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ.....	7
ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ	8
ВВЕДЕНИЕ	9
1 Анализ предметной области	10
1.1 Анализ существующих программных решений	10
1.1.1 Битрикс24.....	11
1.1.2 ПланФикс	11
1.1.3 YouGile	12
1.1.4 Обобщение результатов анализа рынка.....	12
1.2 Анализ архитектур, применяемых в мобильных приложениях	13
1.2.1 Семейство архитектур MV-X.....	13
1.3 Выводы	17
1.3.1 Чистая архитектура (Clean Architecture).....	17
2 Проектирование.....	20
2.1 Проектирование архитектуры системы в целом.....	20
2.2 Проектирование структуры базы данных.....	21
2.3 Проектирование архитектуры клиентского приложения	22
2.4 Выбор средств разработки	24
3 Реализация приложения и его тестирование.....	26
3.1 Реализация мобильного приложения	26
3.1.1 Разработка пользовательского интерфейса	26
3.1.2 Структура проекта.....	27
3.1.3 Entities (Сущности)	27

3.1.4 UseCase (Сценарии или Варианты использования).....	28
3.1.5 Presentation (Представление).....	29
3.1.6 Repositories (Репозитории)	31
3.2 Реализация серверной части	31
3.2.1 Пакет Entity	32
3.2.2 Пакет Repository	32
3.2.3 Пакет Service.....	33
3.2.4 Пакет Config.....	34
3.2.5 Пакет Controller	34
3.2.6 Пакеты Response и Request.....	35
3.2.7 Развертывание	36
3.3 Тестирование	36
3.3.1 Тестирование серверного приложения	36
3.3.2 Теоретические вычислительная и ёмкостная сложности	38
4 Экономическая часть	40
4.1 Организация и планирование работ	40
4.2 Расчёт стоимости проведения работ	42
4.2.1 Статья 1 «Материалы, покупные изделия и полуфабрикаты»	42
4.2.2 Статья 2 «Специальное оборудование»	42
4.2.3 Статья 3 «Основная заработная плата»	43
4.2.4 Статья 4 «Дополнительная заработная плата».....	43
4.2.5 Статья 5 «Страховые отчисления»	44
4.2.6 Статья 6 «Командировочные расходы»	44
4.2.7 Статья 7 «Контрагентские услуги»	44
4.2.8 Статья 8 «Накладные расходы».....	44

4.2.9 Статья 9 «Прочие расходы»	44
4.2.10 Расчет полной себестоимости проекта	44
4.3 Расчет договорной цены	45
ЗАКЛЮЧЕНИЕ	46
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	47
ПРИЛОЖЕНИЕ А	51
ПРИЛОЖЕНИЕ Б	55
ПРИЛОЖЕНИЕ В	57

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

В настоящей выпускной квалификационной работе были использованы следующие термины с соответствующими определениями:

1) Data Transfer Object – один из шаблонов проектирования, используется для передачи данных между подсистемами приложения, не содержит никакого поведения [1]

2) HyperText Transfer Protocol – протокол прикладного уровня передачи данных, изначально — в виде гипертекстовых документов в формате HTML, в настоящее время используется для передачи произвольных данных [2]

3) JavaScript Object Notation – текстовый формат обмена структурированными данными, основанный на парах ключ-значение и упорядоченных списках [3]

4) Интегрированная среда разработки – программный комплекс, предназначенный для продуктивной разработки прикладных систем и состоящий: из редакторов исходных текстов и ресурсов; из компилятора; из отладчика и пр. [4]

5) Система контроля версий - система , которая ведет учет модификаций файлов проекта. Такая система позволяет в случае необходимости вернуться к более ранним версиям проекта [5]

6) Система управления базами данных – совокупность программных, языковых и прочих средств, которые предназначены для создания, управления, контролирования, администрирования и совместного использования БД разными пользователями [6]

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей выпускной квалификационной работе были применяются следующие сокращения:

DTO – Data Transfer Object

HTTP – HyperText Transfer Protocol

IDE – Integrated Development Environment (интегрированная среда разработки)

JSON – JavaScript Object Notation

UI – User Interface (пользовательский интерфейс)

БД – База данных

ГИП – Графический интерфейс пользователя

ОС – Операционная система

ПО – Программное обеспечение

СУБД – Система управления базами данных

ВВЕДЕНИЕ

Проекты – неотъемлемая часть любой современной организации, осуществляющей деятельность в сфере информационных технологий [7].

Актуальность: Российский рынок систем управления проектами заполнен преимущественно иностранным ПО, а изучение отечественных решений показало, что они недостаточно качественны и удобны для пользователей. Таким образом разработка приложения для контроля ИТ-проектов – актуальное направление.

Цель работы – внедрение Чистой архитектуры в мобильную разработку на примере приложения для контроля ИТ-проектов, что позволит исправить недостатки существующих на российском рынке ИТ-решений, а также повысить гибкость и тестируемость ПО.

Для достижения цели необходимо выполнить следующие **задачи**: изучить предметную область, изучить основные практики разработки мобильных приложений, спроектировать архитектуру приложения, разработать и протестировать приложение, рассчитать стоимость разработки.

Новизна работы заключается в имплементации Чистой архитектуры для мобильной разработки и применении к разработке полученной архитектуры.

Объект исследования: Архитектуры и паттерны разработки мобильного программного обеспечения.

Предмет исследования: Имплементация Чистой архитектуры для мобильно разработки.

Гипотеза: внедрение Чистой архитектуры в промышленную разработку мобильных систем позволит упростить процесс разработки, повысить читаемость кода проектов, тестируемость, гибкость систем и п.р.

Стандарты, используемые в работе: ФГОС ВО 09.03.04, СМКО МИРЭА 7.5.1/03.П.67-19, ГОСТ 7.32-2017.

На защиту выносятся: инновационная архитектура для мобильной разработки, а также метод ее внедрения в проект, разработанное мобильное клиент-серверное приложение для контроля ИТ-проектов.

1 Анализ предметной области

1.1 Анализ существующих программных решений

Рассмотрим существующие на рынке системы управления проектами. В ходе исследования были изучены 13 систем контроля проектов. Сводная информация о рассмотренных системах представлена в таблице 1.1.

Таблица 1.1 – Сводная таблица рассмотренных систем

№	Название	Страна	Поддержка орг. структуры	Поддержка русского языка	UX	UI	Моб. прил.
1	YouGile	Россия	+	+	-	-	+
2	Asana	США	-	-	+	+	+
3	Jira	Австралия	-	-	-	-	+
4	Trello	Австралия	-	+	+	-	+
5	Битрикс24	Россия	+	+	-	-	-
6	Wrike	США	-	+	-	+	-
7	GanttPro	Беларусь	+	+	-	-	-
8	Basecamp	США	-	-	+	-	+
9	MS To-Do	США	-	+	+	-	+
10	Hygger	США	-	-	+	+	+
11	RedMine	-	-	+	-	-	-
12	ПланФикс	Россия	+	+	-	+	+
13	TogglPlan	Эстония	+	-	+	+	+

Из приведенной выше таблицы видно, что рынок заполнен преимущественно зарубежными программными продуктами. Это говорит о ярко выраженном дефиците отечественных ИТ-решений на российском рынке, что ведет к зависимости российских компаний от иностранного ПО.

Также отмечается частое отсутствие поддержки организационной структуры организаций. Несоблюдение иерархических связей может привести к нарушениям процессов управления проектами, появлению информационных потоков, которых быть не должно, несоблюдению порядка распределения трудовой нагрузки по уровням управления и конкретным сотрудникам. Более чем половина рассмотренных систем обладают непривлекательным или сложным интерфейсом.

Далее рассмотрим подробнее системы управления проектами российского производства.

1.1.1 Битрикс24

Битрикс24 – российский сервис для управления бизнесом, включающий в себя в том числе и управление проектами.

Данная система доступна для использования на следующих платформах: Windows, MacOS, iOS и Android. Также доступ к системе можно получить с помощью web-приложение через любой браузер. Многообразие платформ усложняет и повышает стоимость выпуска обновлений, поддержки и разработки в целом.

Битрикс24 относится, скорее, к классу CRM-систем, т.к. ее функциональность очень широка и может подойти не только для управления проектами, но и для использования в отделах продаж и маркетинга. В свою очередь, в контексте управления проектами, такой набор функций избыточен и только усложняет работу с системой.

Внедрение Битрикс24 потребует времени и обучения персонала: интерфейсы перегружены, слишком много функций, с каждой из которых нужно разбираться.

Кроме того, пользователи крайне негативно отзываются о работе системы: большое количество ошибок, низкий уровень технической поддержки, долгие загрузки, медленная работа. Также пользователями отмечается, что с каждым годом число ошибок в системе только растет [8].

1.1.2 ПланФикс

ПланФикс позиционируется как «конструктор», с помощью которого пользователь может «построить» свою систему управления проектами и даже компанией.

Итак, к системе можно получить доступ через web-приложение или с помощью мобильных приложений для Android или iOS.

Создатели системы убеждают, что настроить систему можно самостоятельно, без привлечения технической поддержки или интеграторов. Однако, если изучить отзывы пользователей [9], можно обнаружить, что пользователи часто остаются недовольны сложностью настройки,

«разорванностью» интерфейса (т.е. функции, принадлежащие одной группе, могут быть разнесены по всему интерфейсу). Более того, отмечается, что настройка системы партнерами-интеграторами – очень дорогостоящая услуга, а настроить систему самостоятельно вовсе невозможно.

1.1.3 YouGile

YouGile представляет собой систему управления проектами, в которой каждая задача – это чат. Рассматриваемая система обладает очень широкой функциональностью, что делает ее громоздкой и сложной для первоначальной настройки, а также требует довольно долгого периода обучения работе с ней.

Система YouGile может быть поставлена пользователям в следующих формах: веб-приложение, мобильные приложения для Android и iOS, приложения для ОС Windows, Linux и MacOS. Разнообразие поддерживаемых платформ делает выпуск обновлений дорогостоящим и трудным процессом, что, естественно, сказывается и на стоимости программного продукта.

Решение представить каждую задачу в виде отдельного чата также выглядит сомнительным, поскольку чаты повышают объем информационного шума: вся необходимая для работы информация может затеряться в многочисленных сообщениях сотрудников, принимающих участие в решении задачи.

Кроме того, пользователи отмечают, что мобильные приложения данной системы выглядят «сырыми», а создаваемые в системе задачи могут не сохраниться или пропадать после сохранения [10]. Такие ошибки в работе говорят о неверно спроектированной архитектуре, что, безусловно, затрудняет разработчикам системы поиск ошибок, поддержку и дальнейшее развитие.

1.1.4 Обобщение результатов анализа рынка

Итак, на российском рынке имеется нехватка качественных, удобных программных продуктов с мобильными приложениями. Этот факт делает разработку в данном направлении актуальной. Чтобы начать разработку мобильного приложения необходимо определиться с архитектурой, что подробно рассмотрено в следующих разделах.

1.2 Анализ архитектур, применяемых в мобильных приложениях

Архитектурные ошибки – источник затруднений при разработке, тестировании, эксплуатации и поддержке приложения. Поэтому выбор паттерна и детальное проектирование архитектуры составляет основу всей разработки и требует всестороннего, тщательного обдумывания.

Итак, в общем случае, все архитектуры мобильных приложений можно свести к схеме, изображенной на рисунке 1.1.

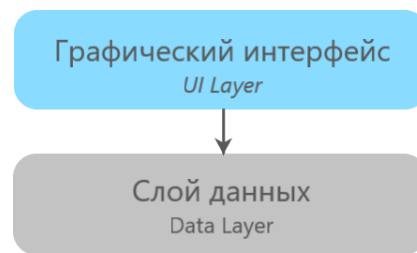


Рисунок 1.1 – Обобщенная схема архитектуры для Android-приложения (авторский рисунок)

Общая черта архитектур приложений – поддержка принципа разделения ответственностей, т.е. обработка событий интерфейса и основная логика приложения разделены на отдельные слои, но каждая из архитектур предлагает свой способ разделения.

В данном разделе будут рассмотрены наиболее часто используемые архитектуры Android-приложений. Для каждой архитектуры будут выделены достоинства и недостатки, а также характерные особенности.

1.2.1 Семейство архитектур MV-X

Для решения в целом повторяющихся для ПО проблем могут быть применены архитектуры, которые используют общие подходы.

Наиболее известной группой таких архитектур является семейство MV-X паттернов. В семейство входят такие архитектуры, как MVC (Model – View – Controller, MVP (Model – View – Presenter), MVVM (Model – View – ViewModel) и пр. Все архитектуры, входящие в рассматриваемую группу, представляют собой производные от MVC. Основные части архитектур этого семейства – так или иначе, это те же Модели, Представления и Контроллеры.

Рассмотрим далее подробнее архитектуры MVC, MVP и MVVM.

1.2.1.1 Model – View – Controller (MVC)

MVC – фундамент MV-X семейства. Идея данного архитектурного решения заключается в отделении ГИП от бизнес-логики приложения, а бизнес-логики от самих данных. Таким образом, приложение может быть разделено на три независимых друг от друга компонента: *Модель (Model)*, *Представление (View)* и *Контроллер (Controller)* [11]. Далее, на рисунке 1.2, приведена схема MVC.

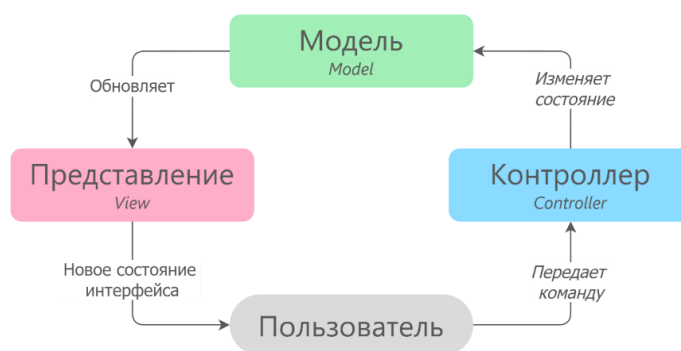


Рисунок 1.2 – Схема MVC (авторский рисунок)

Под *Моделью* подразумевается часть кода, которая содержит функциональную бизнес-логику приложения, т.е. фактически *Моделью* может быть некий набор классов, который служит источником данных для приложения. Слой *Представлений* реализует отображение данных, полученных из Модели, в интерфейсе. *Контроллер* – это компонент, который должен обрабатывать действия пользователя, например, нажатие на кнопку, и оповещать *Модель* о необходимости ее изменений.

В контексте разработки приложений для Android в качестве слоя *Представлений* могут выступать XML-файлы разметки и собственные View-классы, а в качестве Контроллеров – классы Fragment и Activity.

Но в разделении обязанностей между *Контроллером* и *Представлением* кроется проблема: отдельного слоя *Представлений* как такового нет, т.к. как правило, весь код работы и с *Представлениями*, и с *Моделями* помещается в код активности или фрагмента (т.е. в *Контроллер*), что может привести к разрастанию кода до огромного количества строк и смешению зон ответственности слоев.

1.2.1.2 Model – View – Presenter (MVP)

MVP – производный от MVC шаблон проектирования, содержащий три слоя: *Модель (Model)*, *Представление (View)* и *Представитель (Presenter)*.

Ниже, на рисунке 1.3, представлена схема рассматриваемой архитектуры.

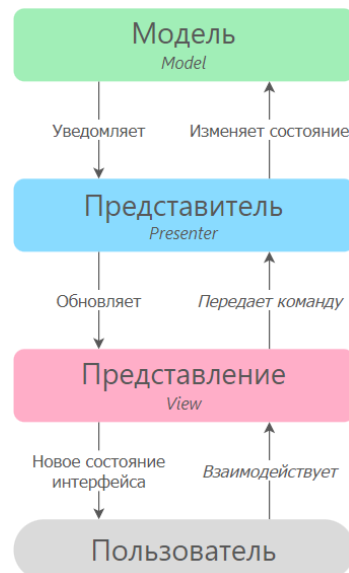


Рисунок 1.3 – Схема MVP (авторский рисунок)

Как и в MVC, *Модель* и *Представление* отвечают за бизнес-логику приложения и отображение данных в интерфейсе, соответственно [12].

Отличие данной архитектуры от MVC заключается в компоненте *Представитель*. Фактически, этот компонент занял место *Контроллера* и точно так же отвечает за передачу данных из *Представления* в *Модель*. Однако, *Представитель* отвечает еще и за обновление *Представления* при изменении состояния *Модели*.

Представитель обменивается данными с *Представлением* через интерфейс, который позволяет увеличить тестируемость, так как *Модель* может быть заменена на специальный макет для модульных тестов.

Таким образом, *Модель* четко отделена от *Представления*, и все их взаимодействие осуществляется через *Представитель*, что делает MVP более гибким и тестируемым вариантом архитектуры по сравнению с MVC.

1.2.1.3 Model – View – ViewModel (MVVM)

MVVM – еще одна производная от MVC архитектура, созданная для возможности обхода ограничений MVC/MVP. Она тоже состоит из трех компонентов: Модель (Model), Представление (View) и Модель представления (ViewModel) (рисунок 1.4).

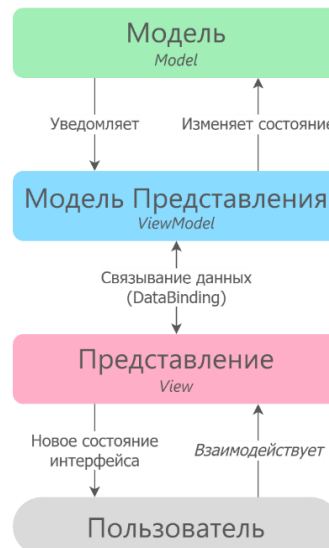


Рисунок 1.4 – Схема MVVM (авторский рисунок)

Аналогично двум ранее рассмотренным архитектурам, Модель и Представление определяют бизнес-логику приложения и пользовательский интерфейс соответственно. Третий же компонент, *Модель представления*, использует технологию связывания данных.

С одной стороны, *Модель представления* – это абстракция представления, а с другой – обертка для данных, которые принадлежать связыванию [13]. Свойства *Представления* совпадают со свойствами *Модели представления*. При этом *ViewModel* не имеет ссылки на интерфейс *Представления*. Изменение состояния *Модели представления* автоматически изменяет *Представление*, и наоборот. То есть *ViewModel* действует как связующее звено между *Моделью* и *Представлением*.

Если рассматривать архитектуру MVVM в контексте разработки для ОС Android, то для ее реализации можно использовать Android Data Binding. IDE Android Studio также предлагает готовое решение – Фрагмент с Моделью представления (Fragment with ViewModel).

1.3 Выводы

Итак, как уже было отмечено в п 1.1, разработка мобильного приложения для управления ИТ-проектами – актуальное направление в текущих условиях.

Также были изучены архитектуры семейства MV-X. Их общее происхождение обеспечивает наличие, в целом, общих недостатков: выделение в проекте слишком большого количества сущностей, что затрудняет разработку, сложность реализации в «чистом виде» (на практике зоны ответственности компонентов часто пересекаются), отсутствие общепризнанного мнения, где именно должна располагаться бизнес-логика приложения (в Модели или в Контроллере), также не установлено, какой из компонентов должен отвечать за валидацию вводимых пользователем данных.

В дополнение, для простоты расширения функциональности следует четко разделять бизнес-логику, модели данных и представления, а ни одна архитектура семейства MV-X не может этого обеспечить.

Проблемы паттернов MV-X решает Чистая архитектура.

1.3.1 Чистая архитектура (Clean Architecture)

Чистая архитектура – современный подход к проектированию программного обеспечения, предложенный Робертом Мартином (рисунок 1.5).

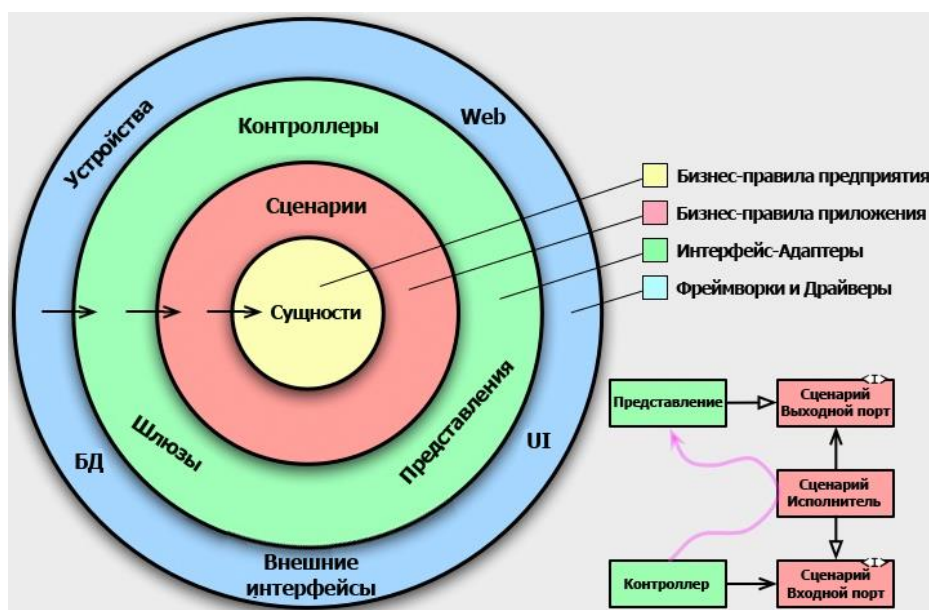


Рисунок 1.5 – Схема Чистой архитектуры Роберта Мартина [14]

Чистая архитектура, подобно паттернам MV-X, подразумевает разделение ПО на компоненты (слои) с разными областями ответственности.

Слой *Сущностей* инкапсулирует бизнес-правила или бизнес-объекты приложения. Они с наименьшей вероятностью изменятся, если изменится что-то внешнее. Например, едва ли возможно, что на эти объекты повлияют изменения в навигации по приложению или изменения в безопасности. Никакие операционные изменения в каком-либо конкретном приложении не должны влиять на уровень *Сущностей*. *Сущности* могут быть объектами с методами или наборами структур данных и функций.

Сценарии инкапсулируют и реализуют все варианты использования системы (бизнес-правила, характерные для конкретного приложения). Эти варианты использования организуют поток данных к *Сущностям* и от них, а также направляют эти *Сущности* на использование своих общеорганизационных бизнес-правил для достижения целей *Сценария*. Изменения в этом слое не должны влиять на слой *Сущностей*. Кроме того, на *Сценарии* не должны оказывать воздействие изменения внешних факторов, таких как база данных, пользовательский интерфейс или какие-либо общие фреймворки.

Третий слой состоит из контроллеров, шлюзов и представлений. Фактически, этот слой представляет собой набор адаптеров, которые преобразуют данные из формата, удобного для *Сценариев* и *Сущностей*, в формат, подходящий для какой-либо внешней составляющей (БД или Интернет).

Последний слой как правило состоит из фреймворков и внешних элементов, например, БД, веб-фреймворки и пр. Обычно в этом слое пишут код «склеивания», который связывается со следующим внутренним кругом.

Компоненты Чистой архитектуры расположены в виде слоев, что обеспечивает соблюдение правила зависимостей.

Правило зависимостей – главное правило для рассматриваемой архитектуры, гласящее, что никакие сущности внутреннего круга не должны

ничего знать о сущностях внешнего [14]. За счет этого достигается независимость от библиотек, баз данных и прочих внешних сервисов, т.е. при необходимости их можно заменить на другие, не меняя логику внутренних слоев. Обеспечивается независимость от UI, что дает возможность изменять интерфейс, не затрагивая остальные компоненты. В свою очередь, независимость слоев повышает тестируемость и добавляет возможность повторного использования некоторых компонентов в других проектах.

Чистая архитектура, в силу своей относительной новизны, еще не имеет общепризнанных реализаций и стандартов, а это оставляет широкий простор для творчества системных архитекторов. Также, она представляет собой очень обобщенную схему, которую следует адаптировать под конкретную платформу и конкретные задач.

2 Проектирование

2.1 Проектирование архитектуры системы в целом

Для хранения и обработки данных о проектах и пользователях необходимо обеспечить наличие в системе сервера. Следовательно, система должна иметь клиент серверную архитектуру. Рассмотрим способы ее организации.

Существуют две основных разновидности клиент-серверной архитектуры: двухуровневая и трехуровневая [15].

Двухуровневая клиент-серверная архитектура характеризуется тем, что в ней используются толстый клиент. Толстый клиент осуществляет и отображение информации, и обработку всех данных. В то время как сервер используется лишь для хранения и предоставления данных клиенту.

Так, например, в контексте разработки приложения для управления ИТ-проектами, на сервере можно было бы разместить базу данных, а всю бизнес-логику перенести на клиентское приложение.

Однако, такой подход является весьма сомнительным, поскольку несет в себе следующие риски:

- Большое количество одновременно установленных соединений могут вызвать значительное замедление обработки запросов.

- Использование толстых клиентов – крайне небезопасная практика, т.к. запросы клиента содержат информацию о структуре базы данных. Это открывает широкое поле деятельности для потенциальных злоумышленников.

- Прямое обращение клиента к базе данных вызывает трудности при обновлении логики клиентского приложения, т.к. требуется обновить приложение абсолютно на всех клиентских устройствах. В противном случае, клиент со старой версией приложения может нарушить консистентность данных.

Таким образом, использование трехуровневой архитектуры не требует дополнительных аргументов. Рассмотрим организацию данной архитектуры подробнее, схема представлена на рисунке 2.1.



Рисунок 2.1 – Схема трехуровневой архитектуры [15]

Отличительной особенностью трехуровневой архитектуры является наличие сервера приложений, на который вынесена большая часть бизнес-логики. Соответственно, клиентское приложение осуществляет только отображение данных, что делает его так называемым «тонким» клиентом.

Итак, трехуровневая архитектура лишена недостатков двухуровневой, поэтому выбор, очевидно, был сделан в ее пользу. Таким образом, в контексте работы, система имеет структуру, представленную на рисунке 2.2.

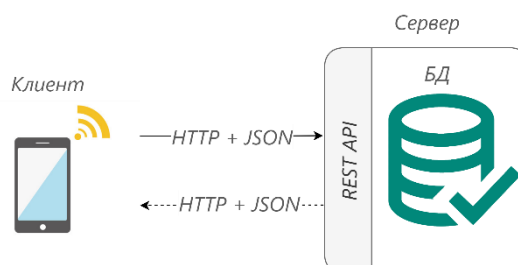


Рисунок 2.2 – Архитектурная схема системы (авторский рисунок)

2.2 Проектирование структуры базы данных

Хорошо спроектированная структура базы данных позволяет сохранять данные удобным образом, а также эффективно производить поиск по ним. Для создания базы данных предложено использовать реляционную модель данных, которая предполагает выделение отдельной таблицы для каждой сущности предметной области [16].

Управлением проектами называется деятельность по решению задач для достижения поставленных целей проекта. В процессах управления принимают

участие владелец проекта (менеджер проекта) и команда разработки [17]. Таким образом, управление ИТ-проектом сводится к эффективному распределению рабочей нагрузки (задач) по сотрудникам и осуществлению контроля за выполнением задач руководителем. Следовательно, можно выделить следующие сущности: Проект, Задача, Участник проекта.

Проект обладает следующими атрибутами: наименование, описание основной идеи, дата начала, планируемая дата завершения. Задача имеет те же атрибуты, что и проект, но кроме них указываются еще создатель задачи и ответственный за нее. В контексте приложения, все участники проекта – пользователи. К атрибутам этой сущности относятся фамилия, имя, отчество, наименование должности, телефон, непосредственный руководитель, адрес электронной почты и пароль.

Итак, получим модель базы данных, приведенную на рисунке 2.3.

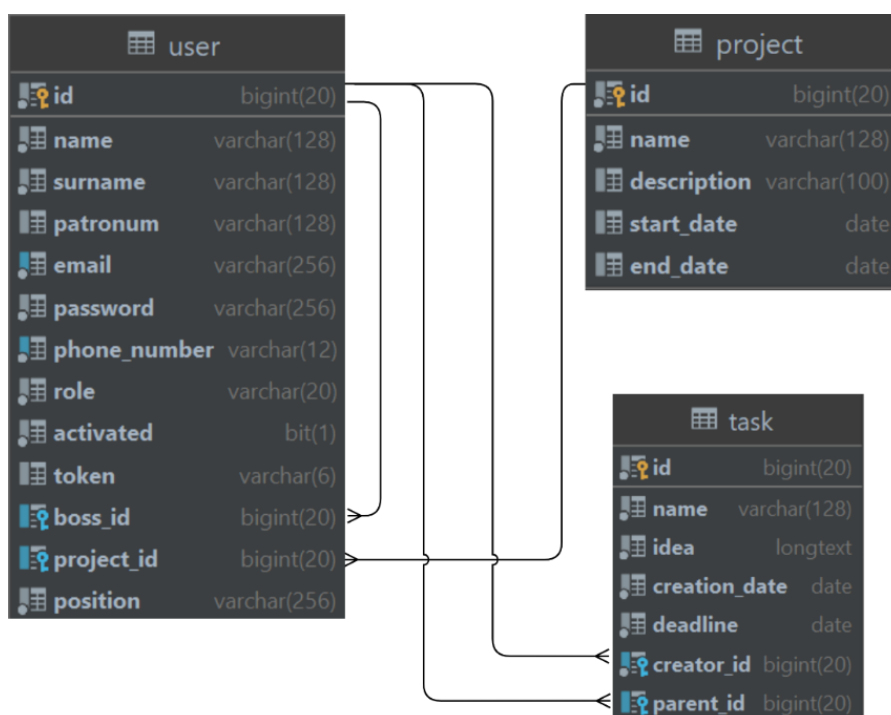


Рисунок 2.3 – Структура базы данных (авторский рисунок)

2.3 Проектирование архитектуры клиентского приложения

Итак, если рассматривать проектирование архитектуры для мобильного приложения для управления проектами, то приложению потребуется

обращаться к серверу и к файлам разметки. Получается, что внешний слой архитектуры содержит следующие компоненты: Web и XML-файлы разметки.

Контроллеры и Представления оригинальной схемы (см п.1.3.1) в данном случае представлены классами активностей (Activities) и фрагментов (Fragments), которые обрабатывают события и подготавливают данные к выводу на экран.

Данные с сервера поступают в приложение через классы-Репозитории (Repository). Затем эти данные передаются в следующий слой – Use Cases. Перед передачей данные «оборачиваются» в удобный для обработки вид – Data Transfer Objects. Слой Use Cases содержит бизнес-правила для данного приложения, а Entities – классы, представляющие бизнес-объекты приложения.

Изобразив все вышеописанное, получим следующую схему (рисунок 2.4).

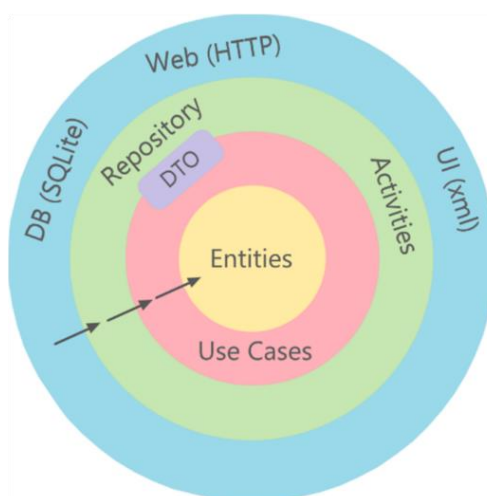


Рисунок 2.4 – Видоизмененная Clean Architecture (авторский рисунок)

Важно отметить, что правило зависимостей также должно быть соблюдено, как и в оригинальном варианте архитектуры. Однако, не всегда бывает просто соблюсти этот принцип. Например, когда пользователь нажимает кнопку регистрации, Активность должна обработать это событие, вызвав Use Case для отправки запроса к серверу с помощью Репозитория. Но

правило зависимостей запрещает внутренним слоям обращаться к внешним, т.е. из слоя Use Case нельзя напрямую обращаться к слою Repository.

Эту проблему можно решить, используя внедрение зависимостей (Dependency injection, DI). Идея этого метода заключается в том, чтобы скрыть конкретную реализацию класса верхнего слоя от классов нижнего. Для этого необходимо создать интерфейс, имплементирующий класс верхнего слоя, а класс нижнего слоя сможет использовать объекты, реализующие этот интерфейс. В контексте рассматриваемого примера это значит, что для Репозитория необходимо создать интерфейс. Через этот интерфейс и будет происходить взаимодействие Use Case и Repository (рисунок 2.5). Таким образом, несмотря на обращение Use Case к Repository, Use Case все равно не зависит от конкретной реализации вышележащего слоя.

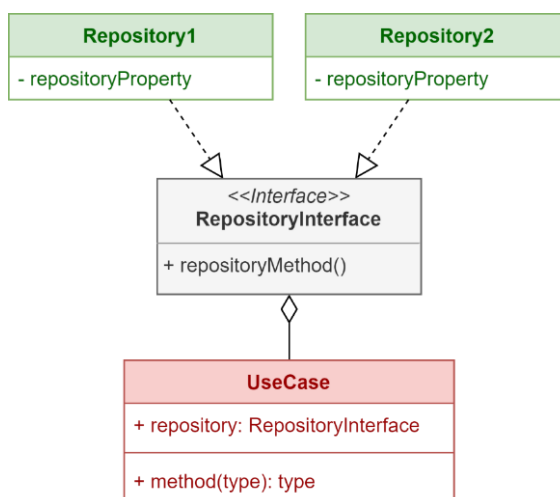


Рисунок 2.5 – UML-диаграмма классов для внедрения зависимостей (авторский рисунок)

2.4 Выбор средств разработки

Для разработки серверного и клиентского мобильного приложения был выбран язык Java. Java – один из самых популярных языков программирования для backend. Он отличается универсальностью, т.к. он работает на виртуальной машине JVM (Java Virtual Machine) [18]. Также Java8 позволяет использовать такие инструменты как стримы, лямбда-выражения, новую библиотеку java.time [19]. За счет своей популярности, Java имеет большое сообщество разработчиков, благодаря чему существует широкий спектр

статей и обучающих материалов, а также библиотек и фреймворков, что делает разработку на этом языке простой и понятной.

Для разработки серверной части был выбран Spring Boot. Spring делает программирование на Java более быстрым, простым и безопасным [20]. Гибкий и всеобъемлющий набор расширений и сторонних библиотек Spring позволяет разработчикам создавать практически любые приложения. По своей сути функции Inversion of Control (IoC) и Dependency Injection (DI) обеспечивают основу для широкого набора функциональных возможностей. Spring Boot – это надстройка над классическим Spring, которая позволяет упростить процессы конфигурации и развертывания приложения.

Также для уменьшения количества написанного кода и повышения читаемости программы было решено использовать библиотеку Lombok. Она позволяет заменить объемный код конструкторов, геттеров, сеттеров и пр. на аннотации, которые занимают значительно меньше места.

Сама разработка ведется в IDE IntelliJ Idea Ultimate и Android Studio. Эти среды имеют массу преимуществ: интеграция с Git, наличие инструментов для работы с базами данных, глубокое понимание кода, удобный рефакторинг и т.д. Все эти инструменты повышают скорость разработки и качество кода.

Удобным инструментом для работы с базами данных является библиотека LiquiBase. Она позволяет версионировать структуру БД, а также автоматически применять изменения при обновлении приложения. Изменения в базу вносятся посредством написания скриптов. Они могут быть представлены как в виде SQL-запросов, так и в виде независимых от реализации стандарта SQL форматах: YAML, JSON, XML.

Для упрощения развертывания приложения и СУБД используется средство контейнеризации Docker. С помощью специальных скриптов можно автоматизировать процесс установки, а также настроить взаимосвязь между различными компонентами системы, например приложением и СУБД. Кроме того, контейнеризация позволяет ограничить доступ программных компонентов к ресурсам системы.

3 Реализация приложения и его тестирование

3.1 Реализация мобильного приложения

3.1.1 Разработка пользовательского интерфейса

Разработка пользовательского интерфейса велась посредством написания xml-разметок активностей, фрагментов и прочих элементов. Такой метод создания ГИП позволяет быстро располагать элементы на экране как с помощью кода, так и в встроенном редакторе, что очень удобно.

В интерфейсе были использованы такие элементы как кнопки, выпадающие списки, карточки, поля ввода и т.п.

Файлы разметок хранятся в директории /res/layout. Скриншоты графического интерфейса приведены на рисунке 3.1 и в приложении А (рисунки А.1-А.9).

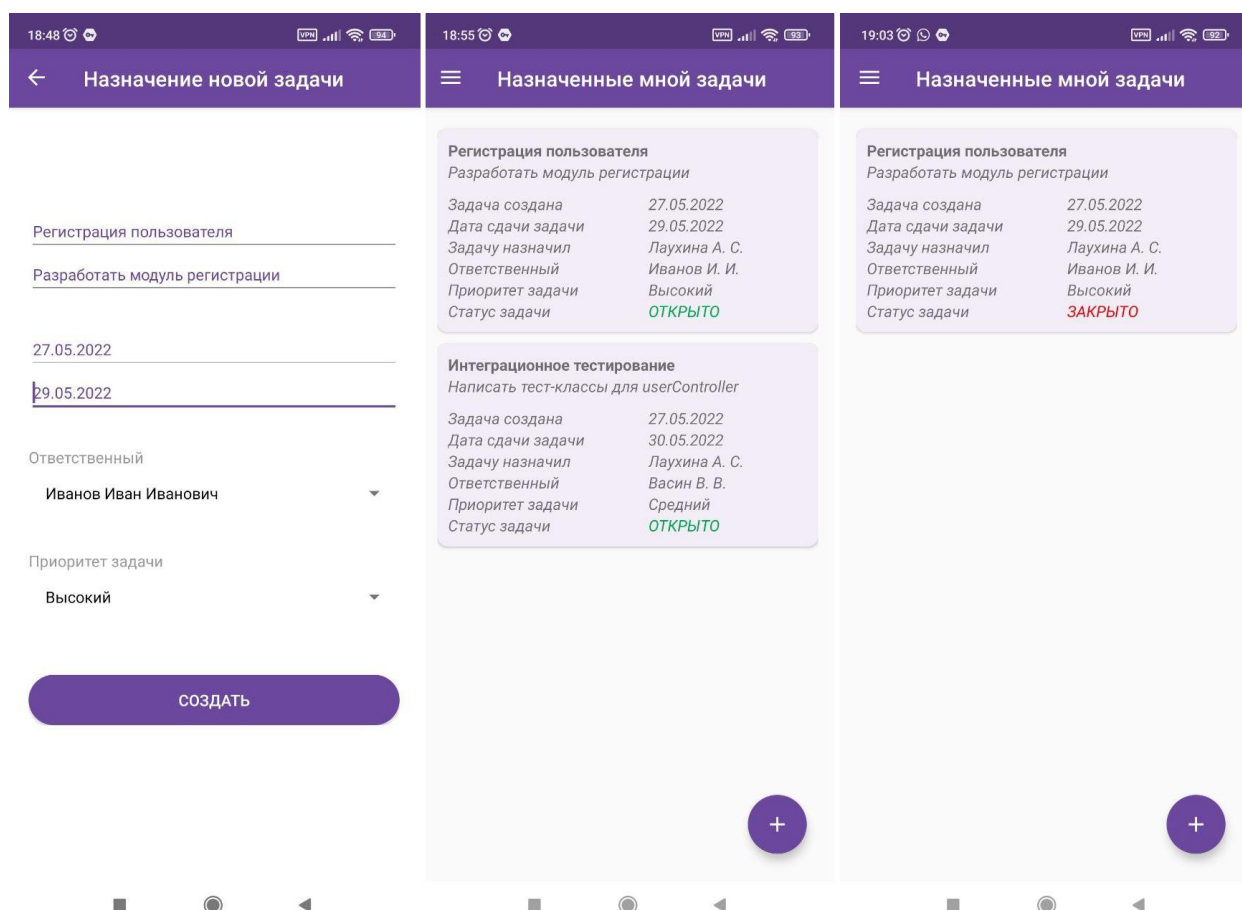


Рисунок 3.1 – Функции назначения задач (авторский рисунок)

3.1.2 Структура проекта

Разделим проект на несколько пакетов, соответствующих слоям архитектуры (рисунок 3.2).

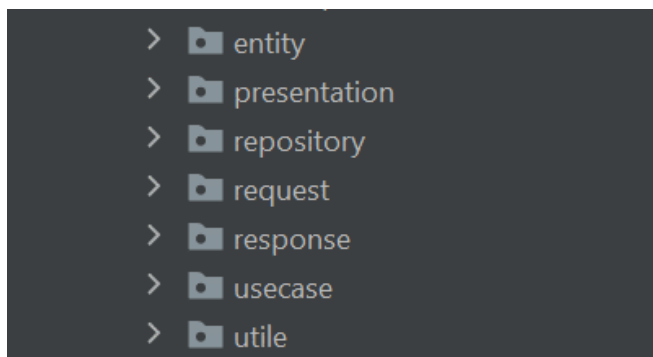


Рисунок 3.2 – Структура проекта (авторский рисунок)

Из рисунка также видно, что кроме пакетов-слоев есть также дополнительные пакеты: request и utile.

Пакет request содержит DTO-классы, которые представляют из себя обертку для объектов, пересылаемых между активностями и репозиториями (на схеме 2.2 изображен в виде фиолетового прямоугольника). Таким образом, достигается независимость активностей от изменений в репозиториях.

В пакете utile располагаются вспомогательные классы, которые нельзя отнести к конкретному слою, например, класс констант.

Далее рассмотрим каждый из основных пакетов подробнее.

3.1.3 Entities (Сущности)

Пакет Entities соответствует одноименному слою архитектуры, спроектированной в п. 2.3, поэтому в нем содержатся классы, соответствующие сущностям.

Логично выделить следующие сущности: «Пользователь», «Сотрудник», «Проект» и «Задача» (рисунок 3.3).

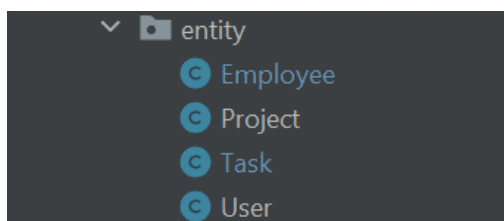


Рисунок 3.3 – Содержимое пакета Entities (авторский рисунок)

Итак, класс Project соответствует сущности «Проект» и обладает следующими атрибутами (полями): название, описание, дата начала и планируемая дата завершения.

Класс User (сущность «Пользователь») содержит следующие поля: имя, фамилия, отчество, должность, телефон, адрес электронной почты и пароль аккаунта.

Класс «Employee» почти полностью дублирует класс User и представляет собой сущность сотрудника. Но в отличие от «Пользователя», «Сотрудник» не содержит пароля аккаунта и используется для представления данных коллег пользователя в его аккаунте.

Класс Task представляет собой сущность «Задача» и имеет такие атрибуты: название, краткое описание (идея), дата создания, дедлайн, ответственный сотрудник (Employee), создатель задачи (Employee), приоритет задачи, статус и идентификатор.

Чтобы защитить атрибуты от непредвиденных изменений, все поля в классах определены как приватные. Использование аннотаций библиотеки @Data, @AllArgsConstructor, @NoArgsConstructor библиотеки Lombok позволяют автоматически генерировать геттеры, сеттеры, конструкторы по умолчанию и конструкторы с полным набором параметров, не засоряя при этом код [21].

Важно отметить, что данные классы не содержат обращений к классам внешних слоев архитектуры, что обеспечивает соблюдение правила зависимостей – основного правила Чистой архитектуры.

3.1.4 UseCase (Сценарии или Варианты использования)

Данный пакет соответствует слою UseCase. Здесь расположены классы, которые содержат в себе специфичную для реализуемого приложения бизнес-логику. Например, к таким классам относятся классы-валидаторы, которые реализуют правила проверки паролей и прочих данных, которые вводятся пользователями.

Аналогично классам-сущностям, классы слоя UseCase не имеют обращений к классам вышележащих слоев. Все зависимости направлены от внешних слоев к внутренним, в данном случае от UseCase к Entities.

3.1.5 Presentation (Представление)

Классы этого пакета относятся к слою – Activities/Repositories. Здесь располагаются классы активностей и фрагментов, которые условно можно разделить еще на две группы по принадлежности к определенной части приложения: start и account (рисунок 3.4).

Подраздел start содержит стартовую активность и фрагменты, которые и отвечают за обеспечение функций регистрации и авторизации.

Подраздел account содержит главную активность аккаунта пользователя и все фрагменты, относящиеся к аккаунту (профиль, раздел сотрудников, раздел созданных задач, раздел полученных задач и пр.)

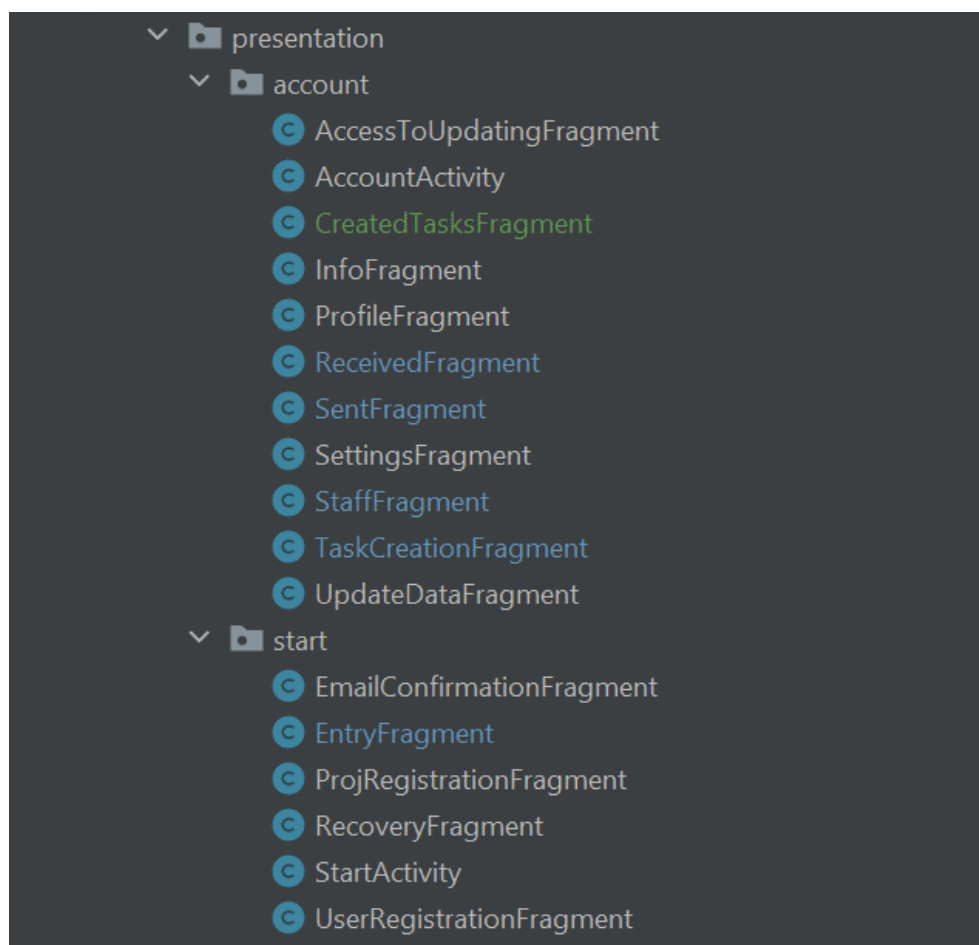


Рисунок 3.4 – Структура пакета Presentation (авторский рисунок)

Итак, в таблице 3.1 приведены наименования активностей, фрагментов, которые к ним относятся, и соответствующие им зоны ответственности.

Таблица 3.1 – Активности, их фрагменты и функции

Класс	Функция
StartActivity	Контейнер для фрагментов
EntryFragment	Авторизация
ProjRegistrationFragment	Форма регистрации организации
UserRegistrationFragment	Форма регистрации пользователя
EmailConfirmationFragment	Подтверждение адреса электронной почты
AccountFragment	Контейнер для фрагментов личного кабинета
ProfileFragment	Профиль пользователя
AccessToUpdateFragment	Получение доступа к редактированию данных
UpdateDataFargment	Редактирование данных
ReceivedFragment	Просмотр списка полученных задач
SentFragment	Просмотр отправленных задач (закрытых и открытых)
StaffFragment	Просмотр списка подчиненных
TaskCreationFragment	Форма создания новой задачи

Классы активностей не содержат никакой логики и отвечают лишь за навигацию по фрагментам, которая осуществляется с помощью компонента NavController и навигационного графа.

Фрагменты ответственны за обновление UI, перехват действий пользователя и взаимодействие с репозиториями.

Во фрагментах, откуда должно осуществляться взаимодействие с сервером определены специальные вложенные приватные классы. Каждый такой приватный класс наследуется от AsyncTask и осуществляет создание DTO-объектов и обращение к репозиториям. Непосредственное взаимодействие с сервером делегируется на репозитории, более подробное описание которых представлено в разделе 3.1.5. Внутренние классы фрагментов также выполняют обновление интерфейса после того, как репозиторий закончил взаимодействие с сервером.

Таким образом, сами активности не обращаются напрямую к серверу, а делают это через классы репозитория в фоновом потоке, чтобы не загружать основной поток программы.

3.1.6 Repositories (Репозитории)

Классы-репозитории – классы, отвечающие за формирование и отправку HTTP-запросов серверу, а также за обработку его ответа, содержащего HTTP-код ответа и JSON-файл.

Методы классов-репозиториях обрабатывают ответ сервера и представляют его в виде, удобном для работы фрагмента (т.е. они возвращают DTO).

Обращение к репозиториям осуществляется только внутри класса, унаследованного от AsyncTask. Таким образом, приложение осуществляет все сетевое взаимодействие в фоновом потоке, т.к. для основного потока эти задачи слишком трудоемки и могут вызвать «зависание» приложения.

Для формирования HTTP-запросов использовался фреймворк OkHTTP.

Также среди классов репозиториях есть класс Preferences, отвечающий за взаимодействие с файлом настроек приложения. В этом файле сохраняются логин и пароль для авторизации. Данный файл хранится в системной папке приложения на устройстве и к нему нельзя получить доступ через проводник. Вносить, изменять и удалять данные имеет право только само приложение.

Содержимое пакета repository приведено на рисунке 3.5.

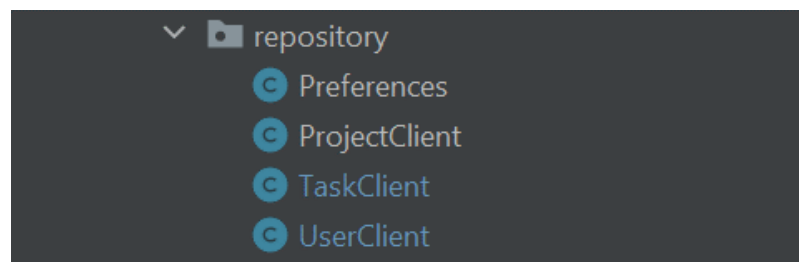


Рисунок 3.5 – Содержимое пакета repository (авторский рисунок)

Пример исходного кода класса-репозитория приведен в листинге Б.1 приложения Б.

3.2 Реализация серверной части

Для удобства написания кода и повышения простоты расширения и читаемости кода классы распределены по пакетам (рисунок 3.6). Каждый пакет содержит классы, реализующие слои Spring-приложения.

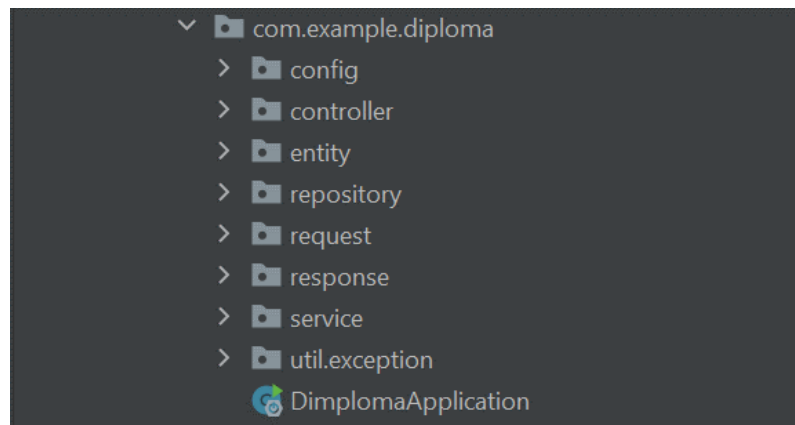


Рисунок 3.6 – Структура проекта серверного приложения (авторский рисунок)

Итак, рассмотрим подробнее каждый пакет приложения далее.

3.2.1 Пакет **Entity**

Данный пакет содержит классы, представляющие собой сущности базы данных. К классам сущностей относятся **User** (пользователь), **Project** (проект), **Task** (задача).

Каждой сущности в соответствие ставится таблица базы с помощью специальной аннотации `@Table`. Каждое поле класса соответствует атрибуту сущности базы, т.е. колонке таблицы. Колонка указывается с помощью аннотации `@Column` [22]. Также необходимо пометить аннотацией `@Id` поле, содержащее первичный ключ. Если значение поля генерируется базой (например, уникальный идентификатор), необходимо использовать аннотацию `@GeneratedValue`.

3.2.2 Пакет **Repository**

Доступ к БД осуществляется посредством встроенного в фреймворк Spring механизма репозитория. Они представляют собой набор интерфейсов, наследующих интерфейс `JpaRepository`.

Фреймворк неявно создает объекты, имплементирующие интерфейсы репозитория. Эти объекты обращаются к базе данных, используя спецификацию `Java Persistence API`. Для каждой из сущностей определен свой репозиторий.

Также, возможно расширить набор определенных фреймворком стандартных методов, добавив свои собственные. Spring создаст реализацию метода по его названию, записанного определенным образом.

Имена производных методов состоят из двух основных частей, разделенных первым ключевым словом `By`. Первая часть, такая как `find`, является вводным, а остальные, такие как `ByName`, являются критериями.

Spring Data JPA поддерживает `find`, `read`, `query`, `count` и `get`, также для ограничения результата или исключения дубликатов можно использовать `Distinct`, `First` или `Top`. `Or` и `And` позволяют объединять выражения.

Так, например, для репозитория `UserRepository` были добавлены методы поиска пользователя по адресу электронной почты, по адресу или телефону, а также метод для поиска всех подчиненных сотрудника по его идентификатору.

Исходный код интерфейса представлен в приложении Б в листинге Б.2

3.2.3 Пакет Service

Пакет `Service` содержит классы, реализующие бизнес-логику приложения. В целом, работа сервисов заключается в обращении к репозиториям для получения данных и преобразовании этих данных для передачи в контроллеры (эндпоинты).

Аналогично репозиториям, каждому сервису соответствует одна сущность. То есть сервисы охватывают варианты использования одного вида сущностей, а каждый метод представляет собой одно действие, например, регистрация нового пользователя, или поиск пользователя. Таким образом, в проекте определено 4 сервиса: `ProjectService`, `UserService`, `MailService` и `TaskService`.

Для того, чтобы обозначить класс как сервис используется Spring-аннотация `@Service` [22].

Пример кода класса-сервиса приведен в листинге Б.3 приложения Б.

3.2.4 Пакет Config

Данный пакет содержит набор классов, используемых для реализации механизма авторизации: `CustomUserDetails`, `CustomUserDetailsService` и `SecurityConfiguration`.

`CustomUserDetails` представляет собой данные пользователя, которые необходимы для авторизации, т.е. это обертка для сущности пользователя, которая не содержит никаких дополнительных данных, кроме логина, пароля, идентификатора и отметки об активации аккаунта.

Класс `CustomUserDetailsService` отвечает за поиск пользователя с заданным логином в базе. Поиск осуществляется через основную службу для сущности Пользователя – `UserService`.

Два вышеописанных класса используются для осуществления аутентификации в классе `SecurityConfiguration`. Этот класс также настраивает права доступа для эндпоинтов: адреса `/public/**` доступны без авторизации, а все остальные – только после ввода логина и пароля. Для обеспечения доступа пользователям к ресурсам сервера используется встроенный в Spring механизм – базовая HTTP аутентификация.

3.2.5 Пакет Controller

При создании серверного приложения использовался архитектурный стиль REST, который подразумевает использование протокола HTTP и предоставление списка URL-адресов, с помощью которых сервер может принимать запросы на получение, сохранение, обновление и удаление данных [23]. За каждым URL-адресом закреплен эндпоинт (или точка входа). Компоненты, которые обрабатывают запросы, приходящие на конкретные эндпоинты, называются контроллерами и расположены в пакете `Controller`.

Каждый класс-контроллер отмечен аннотацией `@RestController`. Контроллер может содержать несколько URL-адресов, каждый из которых привязан к определенному методу контроллера с помощью следующих аннотаций: `@GetMapping` (для GET-запросов), `@PostMapping` (для POST-запросов), `@PatchMapping` (для PATCH-запросов) и др.

Получив запрос на определенный адрес, контроллер вызывает метод, ассоциированный с этим адресом. Каждый из методов контроллера обращается к соответствующему сервису, а тот, в свою очередь, обращается к репозиторию для взаимодействия с базой данных.

Также как репозитории и сервисы, контроллеры ассоциированы с сущностями. Некоторые контроллеры, относящиеся к одной и той же сущности, были разделены на публичные и закрытые. Публичные контроллеры определяют работу эндпоинтов, которые доступны неавторизованным пользователям. К таким входным точкам относятся, например, эндпоинт для регистрации проекта. Закрытые эндпоинты доступны только авторизованным пользователям (например, эндпоинт для изменения персональных данных пользователя). В ответ на запрос контроллеры могут генерировать специальные объекты - DTO, которые подробно рассмотрены в следующем разделе.

Пример кода класса-контролера представлен в листинге Б.4 приложения Б.

3.2.6 Пакеты Response и Request

Оба этих пакета содержат классы DTO.

DTO – специальные объекты, используемые для передачи данных сущностей между клиентом и сервером [24].

Все DTO делятся на те, что приходят от клиента, и те, что отправляются сервером обратно клиенту. Соответственно, в состав пакета Request входят классы представляющие данные, которые присылает клиент, а в состав пакета Response – те, в которые сервер оборачивает данные для ответа клиенту.

Классы DTO отмечены аннотацией Lombok `@Jacksonized`. Эта аннотация позволяет использовать механизм автоматической JSON - сериализации и десериализации объектов.

Таким образом, при поступлении запроса контроллер автоматически преобразует JSON в теле запроса в Request-объект и передает его нужному сервису для дальнейшей обработки. И наоборот, после всех необходимых

манипуляций с данными контроллер подготавливает Response-объект, который затем автоматически сериализуется в JSON и помещается в тело ответа клиенту.

Использование DTO способствует уменьшению взаимной зависимости между слоем представления (клиент) и предметным слоем (сущности) [24], позволяя приложению быть более гибким и уменьшая сложность его дальнейшей разработки.

3.2.7 Развертывание

Для развертывания приложения на сервере необходимо создать 2 файла: `.dockerfile` и `docker-compose.yml`.

`.dockerfile` отвечает за генерацию образа контейнера с собранной программой. Предварительно нужно собрать проект. После сборки образа контейнера он загружается на сервер, на котором проводится развертывание.

Файл `docker-compose.yml` содержит в себе информацию, необходимую для запуска контейнеров: версию СУБД и параметры подключения к ней, порядок запуска СУБД и приложения, порты для подключений.

Контейнер с приложением будет запущен только после запуска контейнера с СУБД и создания пустой базы данных, чтобы избежать преждевременного обращения к базе.

Далее, когда приложение будет запущено, оно создаст все нужные таблицы в базе данных с помощью `.yaml` скриптов, находящихся в директории `\resources\db.changelog`.

3.3 Тестирование

3.3.1 Тестирование серверного приложения

Для тестирования приложения был написан набор тестов. В каждом из тестов проверяется работоспособность каждой конечной точки входа (эндпоинт) приложения в нескольких сценариях с помощью имитационных объектов (`mock`, заглушка) (рисунок 3.7). Использование таких объектов позволяет провести тестирование отдельных модулей без случайного перехода в интеграционное тестирование [25].

Test Results	5 sec 524 ms
Тест класса PublicProjectController	346 ms
Успешная регистрация компании и пользователя	346 ms
Тест класса PublicUserController	21 ms
Успешная активация пользователя	21 ms
Тест класса TaskController	2 sec 662 ms
Успешное закрытие задачи	666 ms
Успешный поиск созданных задач (открытых)	579 ms
Успешный поиск назначенных задач	542 ms
Успешный поиск созданных задач (закрытых)	445 ms
Успешное создание задачи	430 ms
Тест класса UserController	2 sec 495 ms
Ошибка 401 при неправильном пароле	426 ms
Ошибка 409, если пользователь уже есть	395 ms
Успешное получение данных пользователя	401 ms
Успешная регистрация пользователя с проставлением руководителя	459 ms
Успешное обновление данных пользователя	407 ms
Успешное получение списка подчиненных	407 ms

Рисунок 3.7 – Результат выполнения тестов (авторский рисунок)

В ходе выполнения работы было написано несколько тестовых классов, симулирующих обращения к одному из эндпоинтов и сравнивающих ожидаемый результат с фактическим. Тестовые классы были написаны с использованием библиотеки Junit.

При выполнении тестов поднимается in-memory база данных H2, которая хранит все данные в оперативной памяти и используется тестами вместо основной. По завершении тестов in-memory база данных автоматически удаляется. При выполнении тестов выполняются скрипты LiquiBase для инициализации тестовой базы данных, что упрощает написание тестов, т.к. структура тестовой БД идентична структуре основной БД.

Также с помощью библиотеки Jasoco [26] была оценена такая метрика тестирования как покрытие тестами (test coverage) (рисунок 3.8).

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Cxty	Missed Lines	Lines	Missed Methods	Methods	Missed Classes	Classes
com.example.diploma.service	<div></div>	86 %	<div></div>	50 %	11	39	5	170	7	35	0	5
com.example.diploma.controller	<div></div>	100 %		n/a	0	17	0	25	0	17	0	5
Total	99 of 859	88 %	4 of 8	50 %	11	56	5	195	7	52	0	10

Рисунок 3.8 – Оценка покрытия тестами (авторский рисунок)

Как видно из рисунка, покрытие тестами составило 88%.

3.3.2 Теоретические вычислительная и емкостная сложности

Анализ сложности алгоритма позволяет оценить объем ресурсов, необходимых для его выполнения. К таким ресурсам относят время выполнения и объем выделяемой в процессе выполнения оперативной памяти.

Итак, временная сложность – это характеристика алгоритма, определяющая временные затраты на его исполнение, емкостная сложность – характеристика, показывающая, сколько памяти потребуется выделить [27].

Чтобы определить теоретическую сложность алгоритма можно воспользоваться методом подсчета числа критических операций, т.е. операций, которые выполняются наиболее часто или составляют большую часть времени выполнения алгоритма [28]. Прочие операции не вносят значительных временных и емкостных затрат по сравнению с критическими, поэтому они могут игнорироваться при расчетах.

Рассмотрим метод получения списка подчиненных сотрудника серверного приложения. Данный метод преобразует список объектов класса User в список объектов класса EmployeeResponse с помощью цикла, что делает его самым ресурсозатратным.

Тело цикла выполняется n раз. В теле цикла выполняется метод toEmployeeResponse, содержащий 14 операций. После цикла выполняется еще 3 операции (таблица 3.2).

Таблица 3.2 – Подсчет критических операций

Операция	Количество выполнений
toEmployeeResponse	$14n$
UserEmployeesResponse.builder()	1
.employees(employees)	1
.build();	1

Рассчитаем наихудший случай, когда все n элементов списка используются в вычислениях (3.1)

$$T(n) = 14n + 1 + 1 + 1 = 14n + 3 \quad (3.1)$$

Определим порядок роста $T(n)$: числа 14 и 3 - константы, рост определяется значением n . Очевидна линейная зависимость количества

операций от количества элементов n , рост времени пропорционален объему входных данных n . Таким образом, временная сложность данного алгоритма – $O(n)$.

Для работы алгоритма необходимо зарезервировать память для n новых объектов. Следовательно, объем выделяемой памяти пропорционален количеству элементов списка, а значит емкостная сложность – $O(n)$.

Аналогичный алгоритм конвертации массива объектов класса `TaskResponse` в массив с объектами класса `Task` есть и в клиентском приложении. Логично, что временная и емкостная сложности этого алгоритма так же прямо пропорциональны количеству элементов массива. Следовательно, теоретически временная и емкостная сложности алгоритма составляют $O(n)$.

4 Экономическая часть

4.1 Организация и планирование работ

В выполнении данной работы принимали участие 3 человека:

— Руководитель (Алпатов А.Н., к.т.н., доцент, кафедра ИиППО) – отвечает за корректную постановку задачи, контролирует отдельные этапы работы, вносит необходимые поправки и оценивает выполненную работу в целом;

— Консультант (Филаткина А.П., к.ю.н., доцент, кафедра экономики) – консультирует по вопросам написания экономической части, оценивает качество выполненной экономической части;

— Разработчик (Лаухина А.С., группа ИКБО-01-18) – отвечает за анализ предметной области, проектирование и непосредственную разработку приложения для контроля ИТ-проектов, а также за проведение тестирования готового продукта и подготовку проектной документации.

Составим перечень работ, выполняемых в рамках разработки приложения для контроля ИТ-проектов [29].

Выполнение работ производится в период с 20 апреля 2022г. по 27 мая 2022г, что составляет 28 дней.

В таблице 4.1 приведен перечень работ, их продолжительность, а также трудоемкость для каждого из участников. Трудоемкость указывается в количестве дней, в течение которых то или иное лицо принимало участие в работах.

Также для наглядного представления графика работ использована диаграмма Ганта (ленточная диаграмма).

Данный вид диаграммы представляет собой набор полос, которые ориентированы вдоль временной оси. Каждая полоса иллюстрирует отдельную задачу проекта. Вертикальная ось диаграммы – перечень задач.

Таблица 4.1 – Перечень задач проекта

№	Название работ	Трудоемкость, чел/дни	Исполнитель	Длительность, чел/дни
1	Исследовательский этап			
1.1	Обобщённая характеристика предметной области	1	Разработчик	7
1.2	Поиск и анализ аналогов в среде приложений навигации и приложений дополненной реальности	2	Разработчик	
		1	Руководитель	
1.3	Поиск и анализ архитектур, применяемых в мобильной разработке	2	Разработчик	
		1	Руководитель	
1.4	Определение информационных процессов предметной области	1	Разработчик	
1.5	Постановка задач к проектированию и разработке приложения	1	Разработчик	
		1	Руководитель	
2	Проектирование			
2.1	Проектирование архитектуры приложения	1	Разработчик	7
2.2	Проектирование мобильного клиентского приложения для контроля ИТ-проектов	3	Разработчик	
		2	Руководитель	
2.3	Проектирование схемы базы данных	2	Разработчик	
2.4	Выбор средств разработки и видов обеспечения	1	Разработчик	
		1	Руководитель	
3	Разработка			
3.1	Разработка интерфейса мобильного приложения	1	Разработчик	14
3.2	Разработка клиентского мобильного приложения для контроля ИТ-проектов	5	Разработчик	
3.3	Разработка серверного приложения для контроля ИТ-проектов	6	Разработчик	
3.4	Расчёт вычислительной и ёмкостной сложности	1	Разработчик	
		1	Руководитель	
3.5	Тестирование готового продукта	1	Разработчик	
4	Подготовка документации	10	Разработчик	10
		4	Руководитель	
		3	Консультант	
Итого				38

Ниже, на рисунке 4.1 приведена диаграмма Ганта. Из рисунка видно, что общая длительной работ составляет 38 дней. Наглядность диаграммы позволяет также удобно и быстро оценивать длительность выполнения задач.

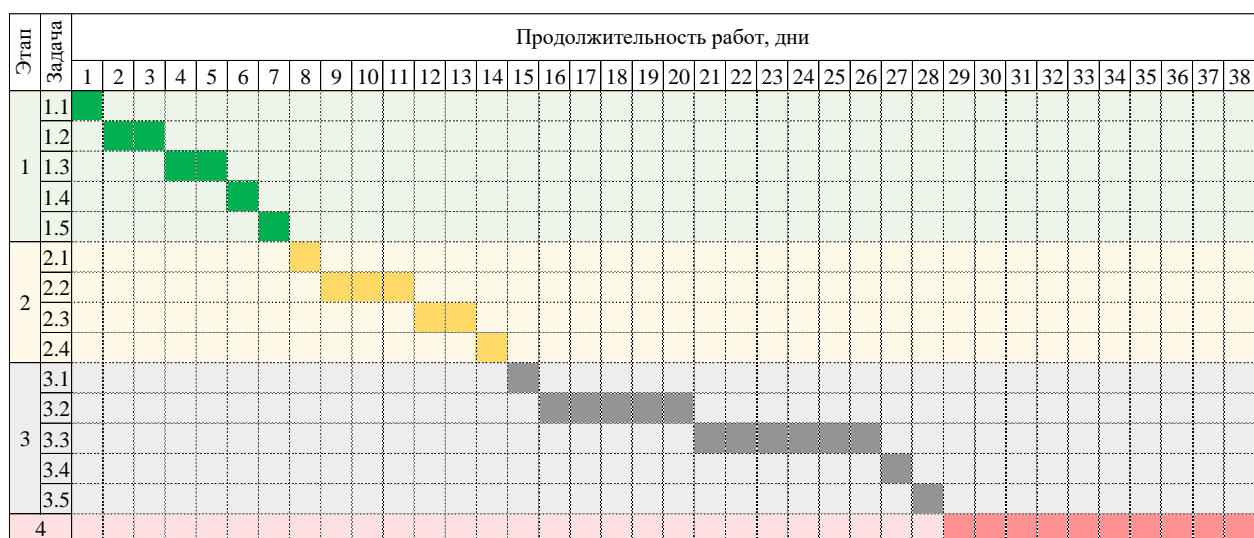


Рисунок 4.1 – Диаграмма Ганта (авторский рисунок)

4.2 Расчёт стоимости проведения работ

Объем затрат, необходимых для проектирования и реализации приложения для контроля ИТ-проектов был оценен методом калькулирования. Итак, затраты включают в себя 9 статей [30], который далее будут рассмотрены подробно.

4.2.1 Статья 1 «Материалы, покупные изделия и полуфабрикаты»

К данной статье относятся стоимость материалов, покупных изделий, полуфабрикатов, комплектующих изделий и других материальных ценностей, расходуемых непосредственно в процессе выполнения работ. Перечень затрат по данной статье приведен в таблице 4.2.

Таблица 4.2 – Перечень затрат по Статье 1

№	Наименование материалов	Единицы измерения	Количество	Цена, руб	Стоимость, руб
1	Кабель USB – USB Type C	шт	1	550	550
2	Белая офисная бумага А4	пачка	1	530	530
3	Картридж для принтера, черный	шт	1	2 530	2 530
4	Ручка шариковая синяя, Pilot	шт	2	86	172
Итого материалов				3 782	
Транспортно-заготовительные расходы				715	
Итого				4 497	

4.2.2 Статья 2 «Специальное оборудование»

Для выполнения запланированных работы специальное оборудование не приобреталось. Таким образом, затраты по данной статье отсутствуют.

4.2.3 Статья 3 «Основная заработная плата»

Основная заработная плата (ОЗП) участников формируется на основе трудоемкости работ каждого из исполнителей и заработной платы персонала, приходящейся на один человеко-день. Данная величина определяется путем деления месячного должностного оклада на 22 дня. Расчеты основной заработной платы сотрудников представлены в таблице 4.3.

Таблица 4.3 – Расчет ОЗП

№	Наименование этапа	Исполнитель (должность)	Мес. оклад (руб)	Трудоемкость (чел/дни)	Оплата за день (руб)	Оплата за этап (руб)
1	Исследовательский этап	Руководитель	80 000	3	3 636	10 908
		Разработчик	30 000	7	1 364	9 548
2	Проектирование	Руководитель	80 000	3	3 636	10 908
		Разработчик	30 000	7	1 364	9 548
3	Разработка	Руководитель	80 000	1	3 636	3 636
		Разработчик	30 000	14	1 364	19 096
4	Подготовка документации	Руководитель	80 000	4	3 636	14 544
		Разработчик	30 000	10	1 364	13 640
		Консультант	45 000	3	2 045	6 136
Итого			97 965			

4.2.4 Статья 4 «Дополнительная заработная плата»

Данная статья включает в себя выплаты, предусмотренные трудовым законодательством за неотработанное по уважительным причинам время, оплату очередных и дополнительных отпусков, выплаты вознаграждения за выслугу лет и т.п. В рамках данной работы эти выплаты составляют 20% от суммы ОЗП) (4.1).

$$\text{ДЗП} = \text{ОЗП} \times 0,2 = 97\,965 \times 0,2 = 19\,593 \quad (4.1)$$

Рассчитаем теперь фонд оплаты труда (далее ФОТ), представляющий собой сумму основной и дополнительной заработных плат (4.2).

$$\text{ФОТ} = \text{ОЗП} + \text{ДЗП} = 97\,965 + 19\,593 = 117\,558 \quad (4.2)$$

Фонд оплаты труда используется при расчете взносов в социальные фонды, что рассмотрено в следующем разделе.

4.2.5 Статья 5 «Страховые отчисления»

Определим величину страховых взносов. Отчисления на социальные нужды составляют 30% от ФОТ (4.3), вычисленного в предыдущем разделе.

$$СВ = \text{ФОТ} \times 0,3 = 117\,558 \times 0,3 = 35\,267 \quad (4.3)$$

4.2.6 Статья 6 «Командировочные расходы»

Командировочные расходы для данного проекта отсутствуют.

4.2.7 Статья 7 «Контрагентские услуги»

К этой статье относится стоимость контрагентских работ, осуществляемых сторонними организациями, например, стоимость изготовления и испытания макетов и опытных образцов и т.п.

В процессе разработки данного проекта услуги сторонних организаций не использовались.

4.2.8 Статья 8 «Накладные расходы»

К накладным расходам относятся расходы на содержание и ремонт зданий, сооружений, оборудования, инвентаря. Это затраты, сопутствующие основному производству, но не связанные с ним напрямую, не входящие в стоимость труда и материалов. Расходы по данной статье отсутствуют.

4.2.9 Статья 9 «Прочие расходы»

Для развертывания серверного приложения был арендован сервер со статическим глобальным адресом. Стоимость аренды за месяц составляет 300 рублей. Сервер был арендован на время проведения разработки, таким образом размер прочих расходов составил 300 рублей.

4.2.10 Расчет полной себестоимости проекта

Рассчитаем полную себестоимость реализации проекта. Стоимость работ составляет сумму затрат по всем статьям. Расчет приведен в таблице 4.4.

Таблица 4.4 – Расчет полной себестоимости

№	Номенклатура статей расходов	Затраты (руб.)
1	Материалы, покупные изделия и полуфабрикаты (за вычетом отходов)	4 497
2	Специальное оборудование для научных (экспериментальных) работ	0
3	Основная заработная плата научного и производственного персонала	97 965
4	Дополнительная заработная плата научного и производственного персонала	19 593
5	Страховые взносы в социальные фонды	35 267
6	Расходы на научные и производственные командировки	0
7	Оплата работ, выполненных сторонними организациями и предприятиями	0
8	Накладные расходы	0
9	Прочие прямые расходы	300
Итого		157 622

4.3 Расчет договорной цены

Для последующей реализации проекта требуется рассчитать договорную цену (ДЦ) (4.4).

$$\text{ДЦ} = \text{себестоимость} + \text{прибыль} + \text{НДС} \quad (4.4)$$

Планируемая прибыль составляет 30% от полной себестоимости (4.5).

$$\text{Прибыль} = \text{себестоимость} \times 0,3 = 157622 \times 0,3 = 47287 \quad (4.5)$$

Оптовая цена предприятия рассчитывается из себестоимости и прибыли (4.6).

$$\begin{aligned} \text{Оптовая цена} &= \text{себестоимость} + \text{прибыль} = \\ &= 157622 + 47287 = 204909 \end{aligned} \quad (4.6)$$

НДС определяется по ставке в 20% от оптовой цены предприятия (4.7).

$$\begin{aligned} \text{НДС} &= \text{Оптовая цена предприятия} \times 0,2 = \\ &= 204909 \times 0,2 = 40982 \end{aligned} \quad (4.7)$$

Таким образом, договорная цена реализуемого продукта, рассчитываемая по формуле (4.4) составляет 245891 рублей (4.8).

$$\text{ДЦ} = 157622 + 47287 + 40982 = 245891 \quad (4.8)$$

ЗАКЛЮЧЕНИЕ

Итак, в процессе выполнения данной выпускной квалификационной работы было разработано мобильное клиент-серверное приложение для контроля ИТ-проектов с поддержкой вертикальной организационной структуры команд. Полученное приложение позволяет назначать задачи подчиненным, а также контролировать сроки их выполнения.

Приложение было также протестировано. Ошибок выявлено не было. Рассчитанное покрытие тестами составило 88%.

Для реализации клиентского приложения была разработана инновационная архитектура для мобильных Android-приложений на основе Чистой архитектуры, описание которой было опубликовано в научном журнале «Наука. Инновации. Образование», индексируемом в РИНЦ. Сертификат о публикации представлен в приложении В.

Из профессионального стандарта 06.001 «Программист» уровня 4 были приобретены необходимые навыки «Разработка тестовых наборов данных», «Проверка работоспособности программного обеспечения», «Рефакторинг и оптимизация программного кода», «Исправление дефектов, зафиксированных в базе данных дефектов».

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Создание объектов передачи данных (DTO) : [Электронный ресурс] - Microsoft Docs, 2022. – URL: <https://docs.microsoft.com/ru-ru/aspnet/web-api/overview/data/using-web-api-with-entity-framework/part-5> (Дата обращения: 27.04.2022).
2. Богданова, В. С. Описание протокола передачи данных HTTP / В. С. Богданова // Наука и производство Урала. – 2019. – № 15. – С. 72-75.
3. Коптева, А. В. Анализ проблемы преобразования данных формата JSON в строго типизированных языках программирования на примере Golang / А. В. Коптева, И. В. Князев // Проблемы науки. – 2021. – № 7(66). – С. 5-10.
4. Артемов, А. А. Анализ современных сред разработки мобильных приложений / А. А. Артемов, Р. Р. Акжигитов, Э. Н. Гаджимурадов // Инновации. Наука. Образование. – 2021. – № 30. – С. 857-868.
5. Акинина, А. А. Рассмотрение принципов работы системы контроля версий на примере GIT / А. А. Акинина, А. С. Маликова // Синергия Наук. – 2019. – № 34. – С. 372-392.
6. Тортика А. С. Обзор и сравнительный анализ современных систем управления базами данных // Вестник Саратовского государственного технического университета. – 2020. – № 4(87). – С. 79-82.
7. SWOT анализ стандарта СТ РК ISO 21500-2014 «Руководство по управлению проектами» // Инновационные подходы в современной науке: материалы Международной (заочной) научно-практической конференции, Прага, Чехия, 25 декабря 2019 года. – Прага, Чехия: Научно-издательский центр "Мир науки" (ИП Вострецов Александр Ильич), 2019. – Р. 130-133.
8. Отзывы о сервисе Битрикс24 : [Электронный ресурс]. – URL: https://startpack.ru/application/1c-bitrix24/reviews?sort_by=grade&sort_desc=asc (Дата обращения: 29.04.2022).
9. Отзывы о сервисе ПланФикс : [Электронный ресурс]. – URL: <https://startpack.ru/application/planfix-project-management/reviews> (Дата обращения: 04.05.2022).

10. Отзывы о сервисе YouGile : [Электронный ресурс]. – URL: https://startpack.ru/application/yougile/reviews?sort_by=grade&sort_desc=asc (Дата обращения: 06.05.2022).

11. Калюжный, Е. Р. Архитектура и технологии, используемые при разработке современных мобильных приложений / Е. Р. Калюжный // Электронные средства и системы управления. Материалы докладов Международной научно-практической конференции. – 2018. – С. 224-226.

12. Реализация MVP архитектуры в мобильном приложении / Н. Н. Куликова, О. А. Маширов, А. Д. Соломыков, М. Ю. Волошко // Фундаментальные и прикладные научные исследования: актуальные вопросы, достижения и инновации : сборник статей XXXVI Международной научно-практической конференции : в 2 ч., Пенза, 27 июля 2020 года. – Пенза: "Наука и Просвещение" (ИП Гуляев Г.Ю.), 2020. – С. 155-157.

13. Грузин, Н. А. Сравнение шаблонов проектирования архитектуры приложения: MVC, MVP и MVVM / Н. А. Грузин // Modern Science. – 2021. – № 1-1. – С. 434-440.

14. The Clean Code Blog by Robert C. Martin [Электронный ресурс] // 2012 г. – URL: <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html> (Дата обращения: 12.05.2022).

15. Архитектура информационных систем : учеб. пособие для СПО / М. В. Рыбальченко. — М. : Издательство Юрайт, 20167 — 91 с. — Серия : Профессиональное образование.

16. Тимофеева, Н. Е. Сравнительный анализ реляционной и не реляционной модели хранения служебной информации централизованной распределенной базы данных / Н. Е. Тимофеева, К. А. Дмитриева // Вестник Российского нового университета. Серия: Сложные системы: модели, анализ и управление. – 2019. – № 1. – С. 66-74.

17. Баширова, М. М. Технология управления проектами и проектными командами на основе методологии гибкого управления проектами / М. М. Баширова // Наука: общество, экономика, право. – 2020. – № 2. – С. 178-183.

18. Эккель Б. Философия Java. 4-е полное изд. — СПб.: Питер, 2018 — 1168 с.: ил. — (Серия «Классика computer science»).
19. Хорстманн, Кей С. Java SE 8. Базовый курс. : Пер. с англ. — М. : ООО "И .Д. Вильямс"— 464 с. : ил. — Парал. тит. англ.
20. Why Spring? [Электронный ресурс] – URL: <https://spring.io/why-spring> (Дата обращения: 13.05.2022).
21. Официальная документация Lombok. [Электронный ресурс] – URL: <https://projectlombok.org/api/> (Дата обращения: 13.05.2022).
22. Уоллс К. Spring в действии. – М.: ДМК Пресс, 2013 – 752 с.: ил. ISBN 978-5-94074-568-6.
23. Алпатов, А. Н. Архитектура многослойного клиент-серверного приложения с разделением логики представления и бизнес-логики / А. Н. Алпатов, Е. В. Попова // Инновации. Наука. Образование. – 2021. – № 48. – С. 1210-1215.
24. The DTO Pattern (Data Transfer Object) : [Электронный ресурс]. – URL: <https://www.baeldung.com/java-dto-pattern> (Дата обращения: 16.05.2022).
25. Алпатов, А. Н. Имитационный сервер rest API на основе спецификации для тестирования веб-приложений / А. Н. Алпатов // Современные наукоемкие технологии. – 2020. – № 11-2. – С. 254-260.
26. Официальная документация Jасосо : [Электронный ресурс] – URL: <https://www.jacoco.org/jacoco/>(Дата обращения: 17.05.2022).
27. Воронухин, М. Е. Анализ вычислительной сложности алгоритмов оптимизации / М. Е. Воронухин // Наука и образование транспорту. – 2018. – № 2. – С. 19-23.
28. Лафоре Р. Структуры данных и алгоритмы в Java. Классика Computers Science. 2-е изд. — СПб.: Питер. — 704 с.: илл.
29. Методические рекомендации по выполнению организационно-экономической части выпускных квалификационных работ / метод. указания / Т. Ю. Гавриленко, О. В. Григоренко, Е. К. Ткаченко. — М.: РТУ МИРЭА, 2019.

30. Экономика предприятия. Учебно-методическое пособие / И.А. Назарова, А.С. Вихрова. – М.: РТУ МИРЭА, 2021. – 71 с.

31. Порядок проведения государственной итоговой аттестации по образовательным программам высшего образования – программам бакалавриата, программам специалитета и программам магистратуры СМКО МИРЭА 7.5.1/03.П.30-19.

32. Положение о выпускной квалификационной работе студентов, обучающихся по образовательным программам подготовки бакалавров СМКО МИРЭА 7.5.1/03.П.67-19.

33. Федеральный государственный образовательный стандарт высшего образования - бакалавриат по направлению подготовки 09.03.04 Программная инженерия (ФГОС ВО 3++).

ПРИЛОЖЕНИЕ А

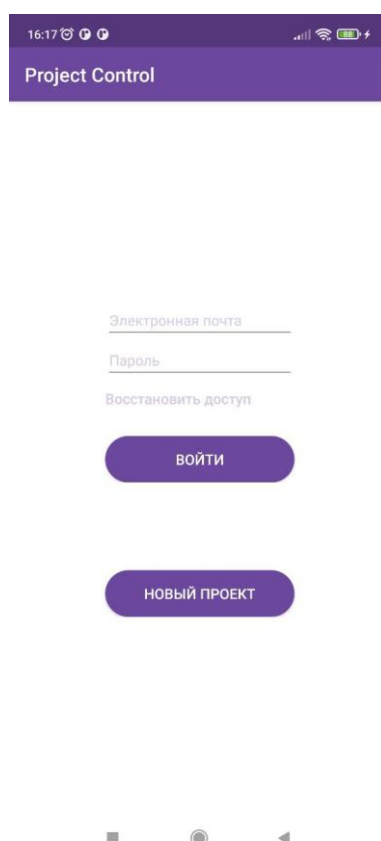


Рисунок А.1 – Экран авторизации

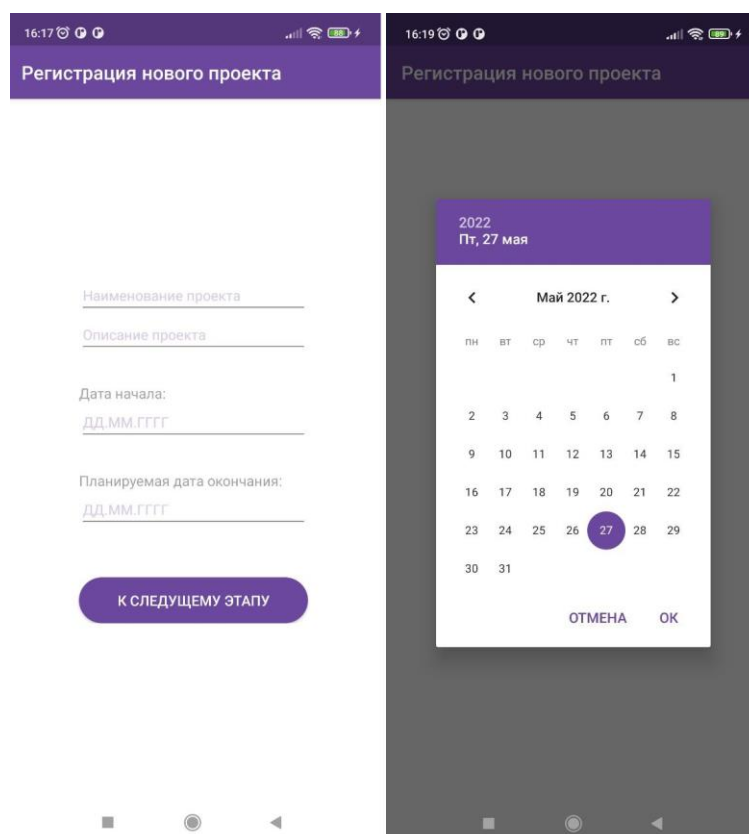


Рисунок А.2 – Экраны регистрации нового проекта

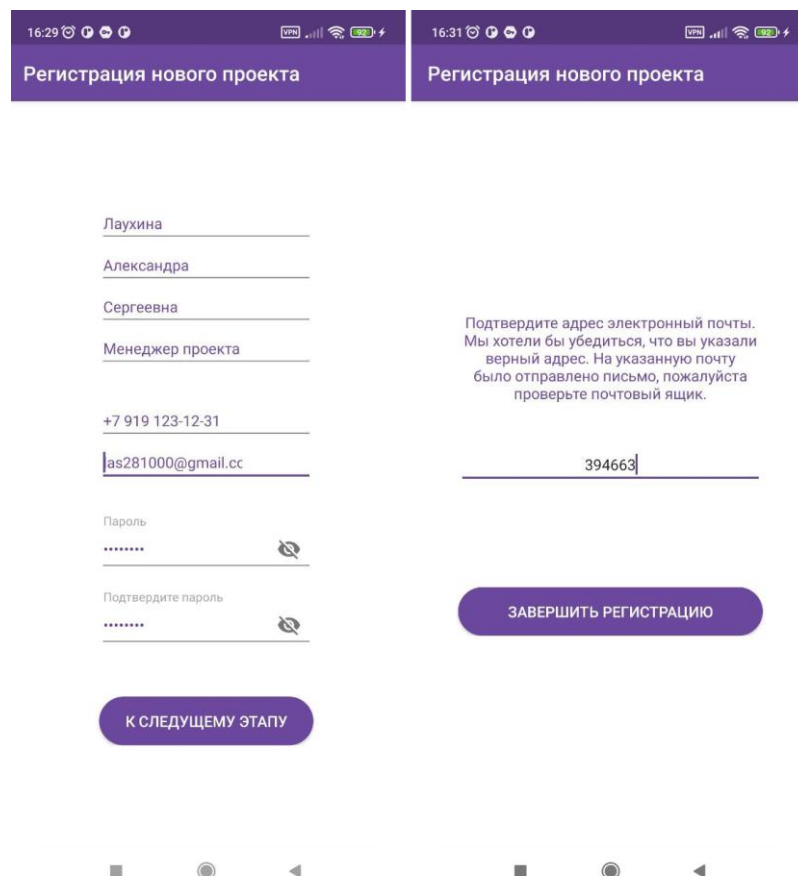


Рисунок А.3 – Экраны регистрации владельца проекта

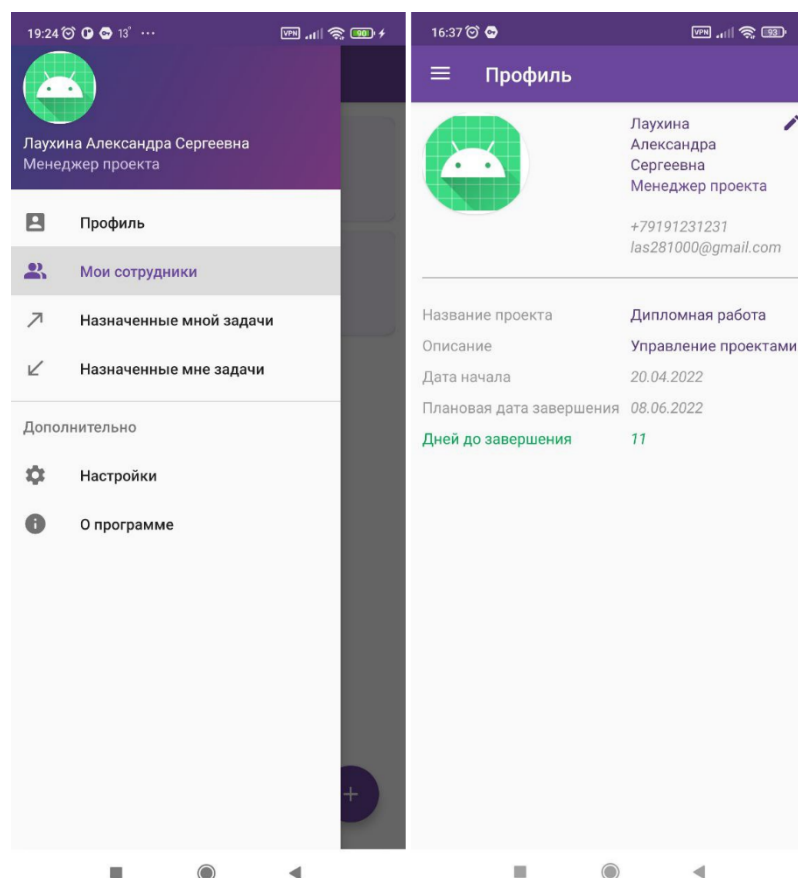


Рисунок А.5 – Профиль сотрудника

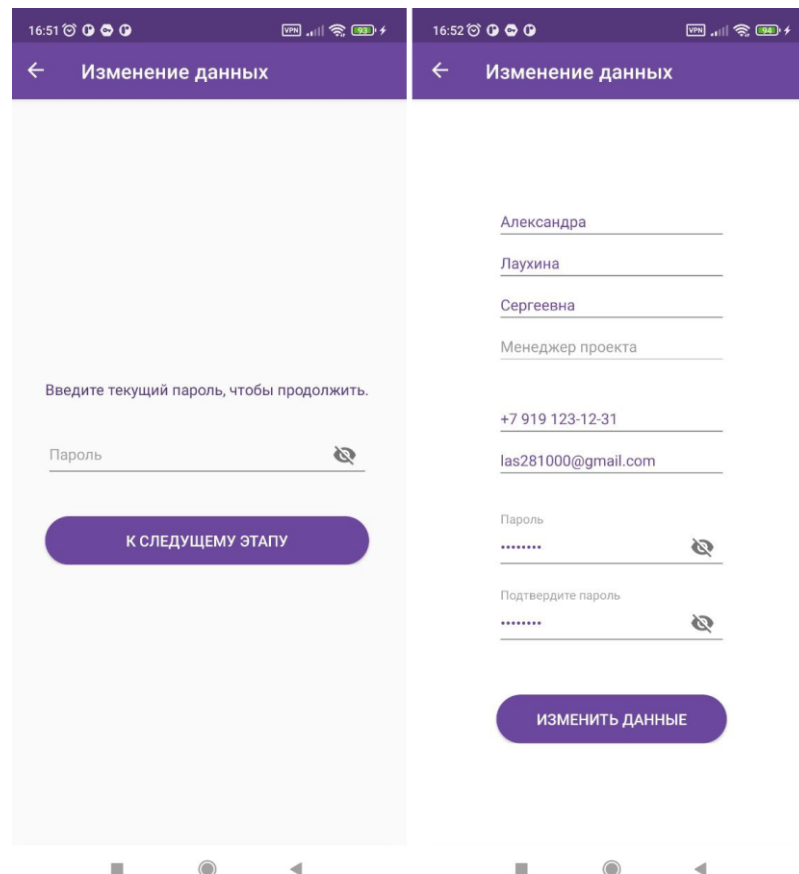


Рисунок А.6 – Экраны редактирования персональных данных

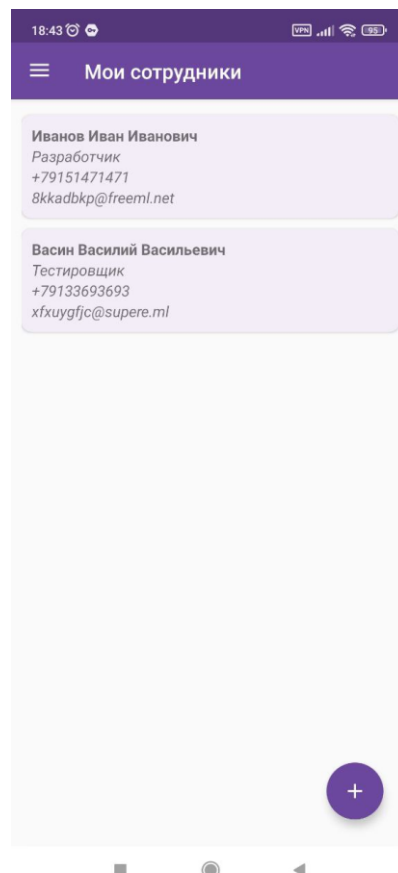


Рисунок А.7 – Экран со списком сотрудников пользователя

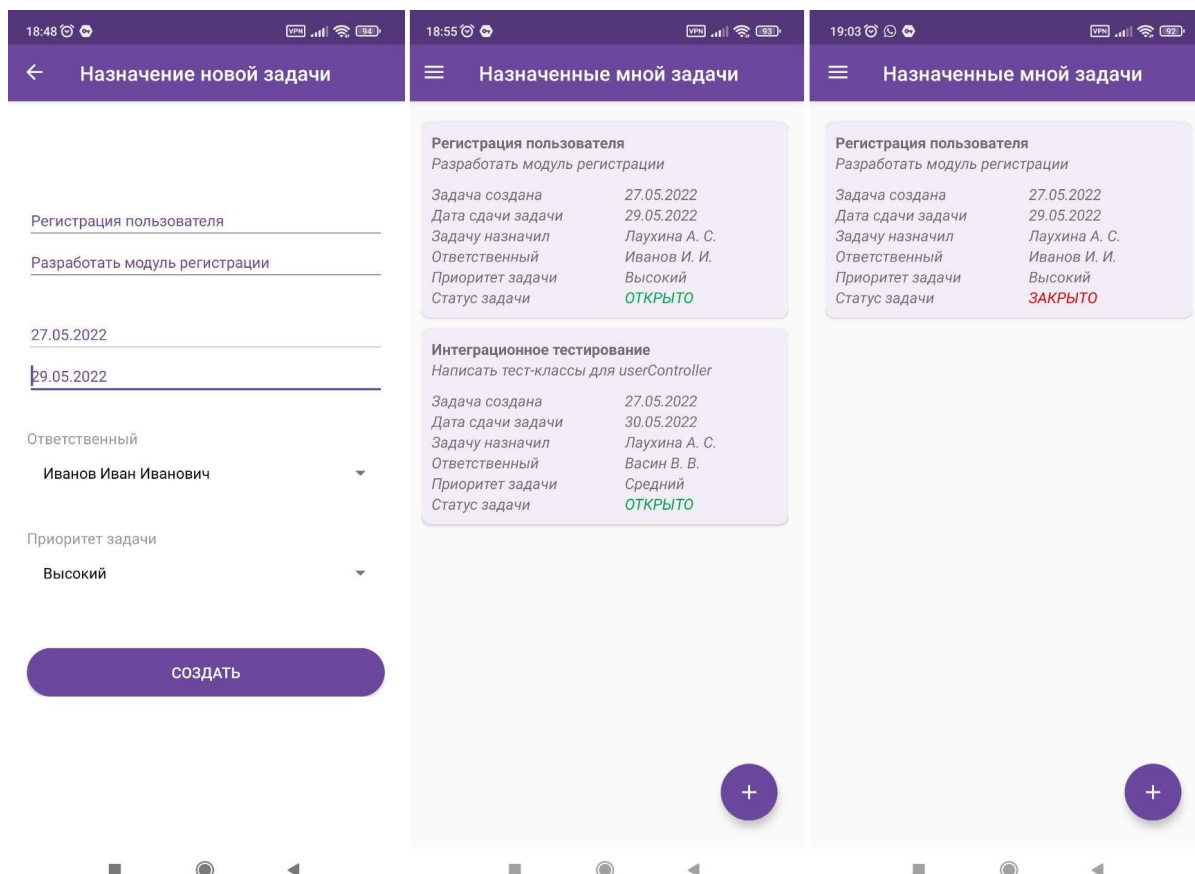


Рисунок А8 – Функции назначения задач

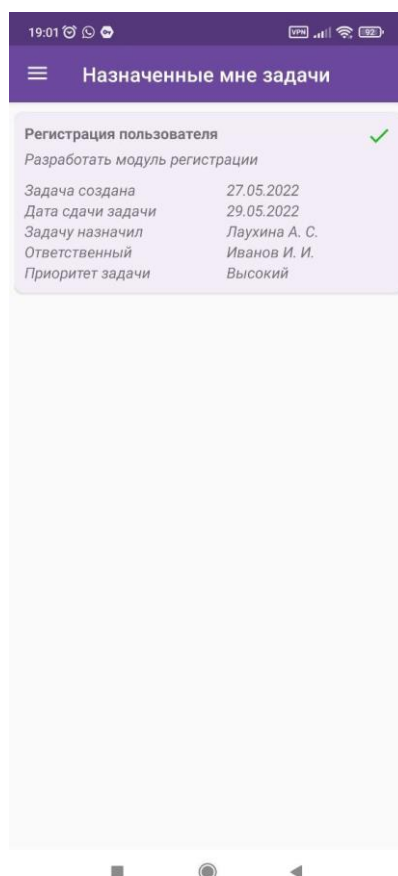


Рисунок А.9 – Страницы со списком назначенных пользователю задач

ПРИЛОЖЕНИЕ Б

Листинг Б.1 – Класс-репозиторий ProjectRepository

```
public class ProjectClient {
    private static final String TAG = "PROJECT_CLIENT"; //Для ошибок
    private static final String URL = "http://checka.ru:3333";
    private static final String regURL = "/public/project";

    public static boolean register(Project project, User user) {
        UserProjectRequest userProjectRequest = new UserProjectRequest(
            project.getName(),
            project.getDescription(),
            project.getStartDate().toString(),
            project.getEndDate().toString(),
            user.getName(),
            user.getSurname(),
            user.getPatronym(),
            user.getPosition(),
            user.getPhoneNumber(),
            user.getEmail(),
            user.getPassword()
        );

        String jsonObject = new Gson().toJson(userProjectRequest);

        OkHttpClient client = new OkHttpClient.Builder()
            .retryOnConnectionFailure(false)
            .build();

        Request request = new Request.Builder()
            .post(RequestBody.create(JSON, jsonObject))
            .url(URL + regURL).build();

        try {
            Response response = client.newCall(request).execute();
            Log.e(TAG, response.toString());
            return response.code() == 200;
        } catch (IOException | JsonSyntaxException e) {
            Log.e(TAG, e.getMessage());
            return false;
        }
    }
}
```

Листинг Б.2 – Интерфейс-репозиторий UserRepository

```
@Repository
public interface UserRepository extends JpaRepository<User, Long> {

    Optional<User> findFirstByEmail(String email);
    Optional<User> findFirstByEmailOrPhoneNumber(String username, String
phoneNumber);
    List<User> findAllByBossId(Long id);
}
```


Листинг Б.3 – Класс-сервис ProjectService

```
package com.example.diploma.service;
@Slf4j
@Service
@Transactional
@RequiredArgsConstructor
public class ProjectService {

    private final ProjectRepository repository;
    private final MailService mailService;
    private final UserService userService;

    public void register(ProjectRegistrationRequest request) {
        log.info("Registering new company: {}", request.getProjectName());
        final var user = userService.createUser(request);
        final var company = Project.builder()
            .name(request.getProjectName())
            .description(request.getDescription())
            .startDate(request.getStartDate())
            .endDate(request.getEndDate())
            .build();
        company.addWorker(user);
        repository.save(company);
        mailService.sendAuthToken(user.getEmail(), user.getToken());
    }
}
```

Листинг Б.4 – Класс-контроллер PublicProjectController

```
@RestController
@RequiredArgsConstructor
@RequestMapping(value = "/public/project", produces =
APPLICATION_JSON_VALUE, consumes = APPLICATION_JSON_VALUE)
public class PublicProjectController {

    private final ProjectService projectService;

    @PostMapping
    public void register(@RequestBody ProjectRegistrationRequest request)
    {
        projectService.register(request);
    }
}
```

ПРИЛОЖЕНИЕ В

ИННОВАЦИИ. НАУКА. ОБРАЗОВАНИЕ ЭЛЕКТРОННОЕ ПЕРИОДИЧЕСКОЕ ИЗДАНИЕ	
СЕРТИФИКАТ	
Лаухина Александра Сергеевна опубликовал(а) статью в научном журнале «Инновации. Наука. Образование» (РИНЦ)	
ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ УПРАВЛЕНИЯ ПРОЕКТАМИ Научный руководитель: Алпатов Алексей Николаевич	
Глав. ред. журнала Сафронов А.И Номер договора РИНЦ №.185-03/215 от 26.03.15	Номер 52, февраль 2022 года Дата публикации: 18.02.2022 www.innovjourn.ru