

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

Направленность Информационные системы и технологии

БАКАЛАВРСКАЯ РАБОТА

на тему

**Информационная система для инвесторов в
виде мобильного приложения на
микросервисах**

Студент _____ Зуйков Евгений Александрович
(подпись, дата) (ФИО полностью)

Руководитель _____ Кольчугина Е.А.
(подпись, дата) (фамилия, инициалы)

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

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

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

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

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

Пенза, 2024

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

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
А.Ю. Козлов
«25» декабря 2023 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

1. Студент Зуйков Евгений Александрович гр. 20ВИ2 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Информационная система для инвесторов в виде мобильного приложения на микросервисах

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

2. Руководитель работы _____

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

Назначение: информационная система для инвесторов в виде мобильного приложения предназначена для предоставления актуальной информации пользователям о биржевых торгах и возможности практического обучения инвестированию

Средства выполнения: Операционная система Windows 10

Средства разработки: IntelliJ IDEA, Android Studio

Язык программирования: Java

Технологии разработки: микросервисная архитектура, MVC, UML

Анализ предметной области и постановка задачи на разработку

Планирование разработки и расчет бюджета

Разработка информационной системы для инвесторов в виде мобильного приложения на микросервисах

Тестирование информационной системы для инвесторов в виде мобильного приложения на микросервисах

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2024 – 13.04.2024	
2	Проектирование программных средств	20%	15.04.2024 – 25.04.2024	
3	Реализация программных средств	30%	26.04.2024 – 12.05.2024	
4	Тестирование программных средств	20%	13.05.2024 – 22.05.2024	
5	Оформление пояснительной записки	20%	23.05.2024 – 03.06.2024	

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 105 листов, 17 таблиц, 33 рисунка, 20 использованных источников и 4 приложения.

ИНВЕСТИРОВАНИЕ, МОБИЛЬНЫЕ ПРИЛОЖЕНИЯ,
МИКРОСЕРВИСЫ, JAVA, SPRING, MONGODB.

Объект исследования: информационная система для инвесторов в виде мобильного приложения на микросервисах.

Цель работы: реализовать мобильное приложение для информационной системы для инвесторов.

Технологии разработки: микросервисная архитектура, MVC, Spring, Java, UML, IntelliJ IDEA, Android Studio.

Задание: спроектировать и реализовать информационную систему для инвесторов в виде мобильного приложения на микросервисах.

Результат работы: спроектирована и разработана информационная система для инвесторов в виде мобильного приложения на микросервисах.

					ПГУ 09.03.02 – 8БР202.09 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	«Информационная система для инвесторов в виде мобильного приложения на микросервисах» Пояснительная записка.	Лит.	Лист	Листов
Разраб.	Зуйков Е. А.						4	105
Провер.	Кольчугина Е. А.					группа 20ВИ2		
Реценз.								
Н. Контр.	Такташкин Д.В							
Утверд.	Козлов А. Ю.							

Содержание

Введение	7
1 Постановка задачи и анализ требований на разработку информационной системы для инвесторов	8
1.1 Постановка задачи на разработку информационной системы для инвесторов	8
1.1.1 Цели разработки и назначение	8
1.1.2 Целевая аудитория информационной системы	9
1.1.3 Формулировка требований на разработку информационной системы	10
1.2 Анализ программного обеспечения аналогичного назначения.....	10
1.3 Формулировка и анализ требований к разрабатываемому программному обеспечению.....	13
1.4 Выбор средств разработки	16
1.5 Планирование процесса разработки программного обеспечения ...	18
2 Проектирование.....	22
2.1 Проектирование структур данных информационной системы.....	22
2.1.1 Проектирование базы данных.....	23
2.1.2 Проектирование серверной части.....	26
2.2 Проектирование структуры программного обеспечения	30
2.2.1 Проектирование структуры серверной части.....	33
2.2.2 Проектирование структуры клиентской части.....	35
2.3 Проектирование интерфейса	38
3 Программная реализация	41
3.1 Реализация базы данных	41
3.2 Реализация серверной части	42
3.3 Реализация клиентской части	47
4 Тестирование информационной системы для инвесторов.....	61

4.1 Обоснование выбора моделей тестирования и метрик программного кода	62
4.1.1 Обоснование выбора моделей тестирования	62
4.1.2 Обоснование выбора метрик программного кода	63
4.2 Тестирование программных средств	64
4.2.1 Тестирование белым ящиком	64
4.2.2 Модульное тестирование.....	68
4.3 Оценка качества программного кода на основе метрик	71
4.3.1 Метрика SLOC	71
4.3.2 Метрика Холстеда.....	73
Заключение	76
Список использованных источников.....	77
Приложение А (рекомендуемое) Исходный код клиентской части	79
Приложение Б (рекомендуемое) Исходный код серверной части.....	89
Приложение В (рекомендуемое) Результаты модульного тестирования	96
Приложение Г (рекомендуемое) Исходный код модульных тестов.....	99

Введение

В наше время все больше людей проявляют интерес к инвестированию, в особенности молодые поколения. Это обусловлено не только доступностью информации в интернете, но и желанием улучшить свое финансовое положение в условиях рыночной неопределенности. Растет число ресурсов, посвященных обучению инвестированию, люди становятся более информированными и финансово грамотными. Доступ к информации о финансовых инструментах и возможность оперативного управления своими вложениями имеют важное значение для инвесторов и трейдеров.

Интерес к инвестированию сегодня также подкрепляется активным развитием технологий, включая использование нейросетей. Нейросети могут быть применены для анализа рыночной динамики, прогнозирования трендов и определения оптимальных стратегий инвестирования.

Мобильные устройства стали неотъемлемой частью повседневной жизни каждого человека. Благодаря удобству, мобильные приложения пользуются высоким спросом на рынке информационных технологий. Приложения для инвестирования являются предпочтительным способом доступа к финансовым рынкам, ведь пользователи имеют возможность управлять своими инвестиционными портфелями непосредственно через свои смартфоны и планшеты в любое удобное время и в любом месте [1].

Информационная система для инвесторов в виде мобильного приложения представляет собой платформу, предназначенную для предоставления клиентам актуальной информации о финансовых инструментах, возможности анализа их динамики, управления инвестиционными портфелями, ведения заметок.

1 Постановка задачи и анализ требований на разработку информационной системы для инвесторов

1.1 Постановка задачи на разработку информационной системы для инвесторов

1.1.1 Цели разработки и назначение

Целью разработки является информационная система для инвесторов. Оно будет представлять собой мобильное приложение на базе микросервисов.

Данная информационная система в виде мобильного приложения должна быть разработана с целью предоставить простой и интуитивно понятный интерфейс для практики в инвестировании как опытным, так и начинающим инвесторам, включая несовершеннолетних пользователей. Приложение должно позволять создавать портфели, анализировать рыночную динамику и принимать инвестиционные решения без финансового риска.

Основная цель разработки заключается в том, чтобы обеспечить безопасное окружение для обучения и практики инвестирования, где пользователи могут приобретать ценный опыт и развивать свои навыки принятия решений. Доступ ко всем функциям приложения будет давать возможность не только изучать теорию инвестирования, но и получать практический опыт, что важно для формирования осознанного и уверенного подхода к реальным инвестиционным ситуациям в будущем.

Этот информационный сервис также должен подходить для опытных инвесторов, предоставлять им инструмент для анализа рынка и принятия решений без необходимости рисковать реальными средствами.

Таким образом, мобильное приложение должно быть удобным и информативным инструментом для изучения и развития навыков инвестирования для широкого круга пользователей, независимо от уровня опыта и возраста.

1.1.2 Целевая аудитория информационной системы

Информационная система для инвесторов в виде мобильного приложения предназначена для широкой аудитории пользователей.

Опытные инвесторы смогут использовать приложение для анализа рынка, отслеживания ценных бумаг, а также для составления инвестиционного портфеля. Данная система сможет предоставлять моментальный доступ к актуальным новостям, аналитике и обновлениям рынка, что поможет пользователям оставаться в курсе событий. Также приложение должно предоставлять продвинутые инструменты анализа данных для помощи в принятии обоснованных инвестиционных решений.

Начинающие инвесторы получают простой и интуитивно понятный интерфейс, что поможет им быстро освоиться. Данное приложение предлагает пользователю создать портфели без каких-либо рисков и вложений, что позволит начинающим инвесторам не только учиться теории инвестирования, но и получать ценный практический опыт. Это поможет им лучше понять механизмы работы рынка и развить свои навыки принятия решений. Таким образом, функция создания портфеля без рисков и вложений не только обеспечит безопасное окружение для начинающих инвесторов, но и поможет им увереннее и осознаннее подходить к реальным инвестиционным решениям в будущем.

Для несовершеннолетних пользователей также будут доступны все функции приложения, поскольку в нем не предусмотрены реальные транзакции. Это означает, что они могут создавать портфели, изучать динамику рынка, анализировать инвестиционные возможности и принимать решения без какого-либо финансового риска. Благодаря доступу ко всем функциям приложения несовершеннолетние пользователи могут получить ценный опыт и понимание механизмов работы рынка, что поможет им развить навыки принятия решений в области инвестирования. Такая возможность дает пользователям уверенность и осознанность в подходе к реальным

инвестиционным ситуациям в будущем, когда они будут готовы совершать реальные сделки на рынке.

Таким образом, была определена целевая аудитория информационной системы. Приложение подходит абсолютно всем категориям пользователей [2].

1.1.3 Формулировка требований на разработку информационной системы

Исходя из целей и назначения информационной системы для инвесторов, были сформулированы следующие требования:

- предоставление пользователю актуальной информации о ценных бумагах и других финансовых инструментах;
- наличие функций для отслеживания и управления инвестиционными портфелями;
- доступ к актуальным новостям и аналитическим материалам о финансовых рынках и компаниях;
- безопасность данных пользователей;
- функциональность заметок и сохранения нужной информации;
- возможность создания множество аккаунтов.

Разрабатываемое программное обеспечение будет реализовано в виде клиент-серверного приложения с микросервисной архитектурой.

В результате получится информационная система для инвесторов в виде мобильного приложения на микросервисах.

1.2 Анализ программного обеспечения аналогичного назначения

В настоящее время существует большой список приложений для инвесторов. Внутренняя часть из них не получили широкого распространения из-за высокой конкуренции. Некоторые имеют довольно сложный интерфейс, доступны только с определенного возраста, требуют финансовых вложений и не ориентированы на пользовательский опыт.

Приложения аналоги приведены ниже.

Тинькофф Инвестиции – это сервис для онлайн-инвестирования, предоставляемый Тинькофф Банком. Он позволяет клиентам инвестировать в акции, облигации, ETF и другие финансовые инструменты через удобный мобильный или веб-интерфейс. Клиенты могут открывать инвестиционные счета, просматривать котировки, проводить анализ рынка и совершать торговые операции на мировых биржах.

Yahoo Finance – это веб-портал, предоставляемый компанией Yahoo, который предлагает широкий спектр финансовой информации и инструментов для инвесторов и трейдеров. Этот сервис предоставляет доступ к различным финансовым данным, включая котировки акций, новости о компаниях, аналитические обзоры, финансовые отчеты, графики цен и многое другое.

Investopedia – это веб-ресурс, предоставляющий обширные сведения о финансах, инвестициях и торговле. Он служит как образовательный ресурс и источник финансовой информации для инвесторов, трейдеров и финансовых профессионалов.

Investing.com – это финансовый веб-ресурс, предоставляющий широкий спектр информации о финансовых рынках, инструментах и инвестиционных возможностях. Сервис предоставляет своим пользователям доступ к актуальным котировкам акций, валют, товаров, индексов и криптовалют, а также к новостям, аналитике, графикам и другим инструментам для анализа и мониторинга рынков.

В целом, представленные программы удовлетворяют потребности инвесторов, имеют все основные функции, такие как: просмотр ценных бумаг в реальном времени, просмотр новостей, анализы рынков, предоставление финансовых отчетов компаний.

Выделим основные критерии оценивания данных веб-ресурсов:

- функциональность портфелей: оценка того, насколько легко и удобно пользователи могут создавать, редактировать и отслеживать свои инвестиционные портфели;

- просмотр информации о ценных бумагах: оценка доступности и полноты информации о ценных бумагах;
- новостной раздел: оценка качества и актуальности новостной ленты;
- функциональность заметок: оценка возможности пользователей создавать заметки, комментарии или отметки касательно определенных ценных бумаг или событий на рынке;
- поддержка нескольких аккаунтов: оценка возможности пользователей управлять несколькими аккаунтами или профилями внутри приложения;
- удобство пользовательского интерфейса: оценка дизайна приложения, его интуитивной навигации и общего удобства использования;
- доступность приложения: возможность использования приложения любому пользователю, независимо от возраста.

В таблице 1 представлено сравнение приложений для инвесторов.

Таблица 1 – Сравнение приложений для инвестирования

Критерии оценки	Программное обеспечение				
	Тинькофф Инвестиции	Yahoo Finance	Investopedia	Investing.com	Разрабатываемое приложение
Функциональность портфелей	средне	сложно	сложно	средне	легко
Просмотр информации о ценных бумагах	легко	сложно	средне	сложно	легко
Новостной раздел	+	+	+	+	+
Функциональность заметок	+	-	-	-	+
Поддержка нескольких аккаунтов	+	-	-	-	+

Продолжение таблицы 1

Критерии оценки	Программное обеспечение				
	Тинькофф Инвестиции	Yahoo Finance	Investopedia	Investing.com	Разрабатываемое приложение
Удобство использования пользовательского интерфейса	средне	средне	сложно	сложно	Легко
Доступность приложения	средняя	полная	полная	полная	полная

Лучшим приложением, с учетом выполняемых функций, является «Тинькофф Инвестиции», но оно также имеет несколько недостатков, а именно: сложность использования функции нескольких аккаунтов, так как для каждого аккаунта, нужен свой номер телефона; недоступность приложения несовершеннолетним; интерфейс, в котором не так просто разобраться; множество лишних функций; открытие всех функций по платной подписке [3].

Представленный обзор позволил выявить оптимальную информационную систему для инвесторов. С учетом недостатков приведенных в таблице программ, можно сказать, что существует необходимость разработки простого в использовании мобильного приложения, предоставляющего пользователю актуальную и полную информацию о ценных бумагах, возможность создания портфелей и управления ими, возможность просмотра последних новостей, возможность создания заметок и получения доступа ко всему функционалу любому пользователю.

1.3 Формулировка и анализ требований к разрабатываемому программному обеспечению

Данная информационная система в виде мобильного приложения разрабатывается с учетом потребностей различных категорий пользователей и должна иметь следующий функционал:

- простой и интуитивно понятный интерфейс: приложение должно обладать пользовательским интерфейсом, который легко понимать и использовать как начинающим, так и опытным инвесторам;
- возможность создания портфелей без рисков и вложений: пользователи должны иметь возможность создавать портфели и управлять ими без финансового риска, что обеспечит безопасную среду для обучения и практики инвестирования;
- отслеживание и анализ доходности портфеля: Пользователи должны иметь возможность отслеживать и анализировать доходность своих инвестиционных портфелей;
- отображение последних новостей о фондовом рынке и компаниях: пользователь должен иметь моментальный доступ к актуальным новостям, что поможет оставаться в курсе событий;
- возможность добавления заметок: пользователь должен иметь возможность добавлять заметки в своих портфелях и отдельных ценных бумагах.

Диаграмма вариантов использования – это одна из разновидностей диаграммы UML, которая используется для описания возможного сценария использования разрабатываемой системы. Она помогает описать, как пользователи будут взаимодействовать с системой и какие функциональные возможности они будут использовать. Основные элементы диаграммы включают варианты использования, актёров, ассоциации.

Диаграмма вариантов использования часто используется на начальных этапах проектирования системы для выявления основных функциональных требований и понимания взаимодействия между компонентами системы и её окружением.

Функциональные возможности информационной системы показаны на диаграмме вариантов использования, изображенной на рисунке 1.

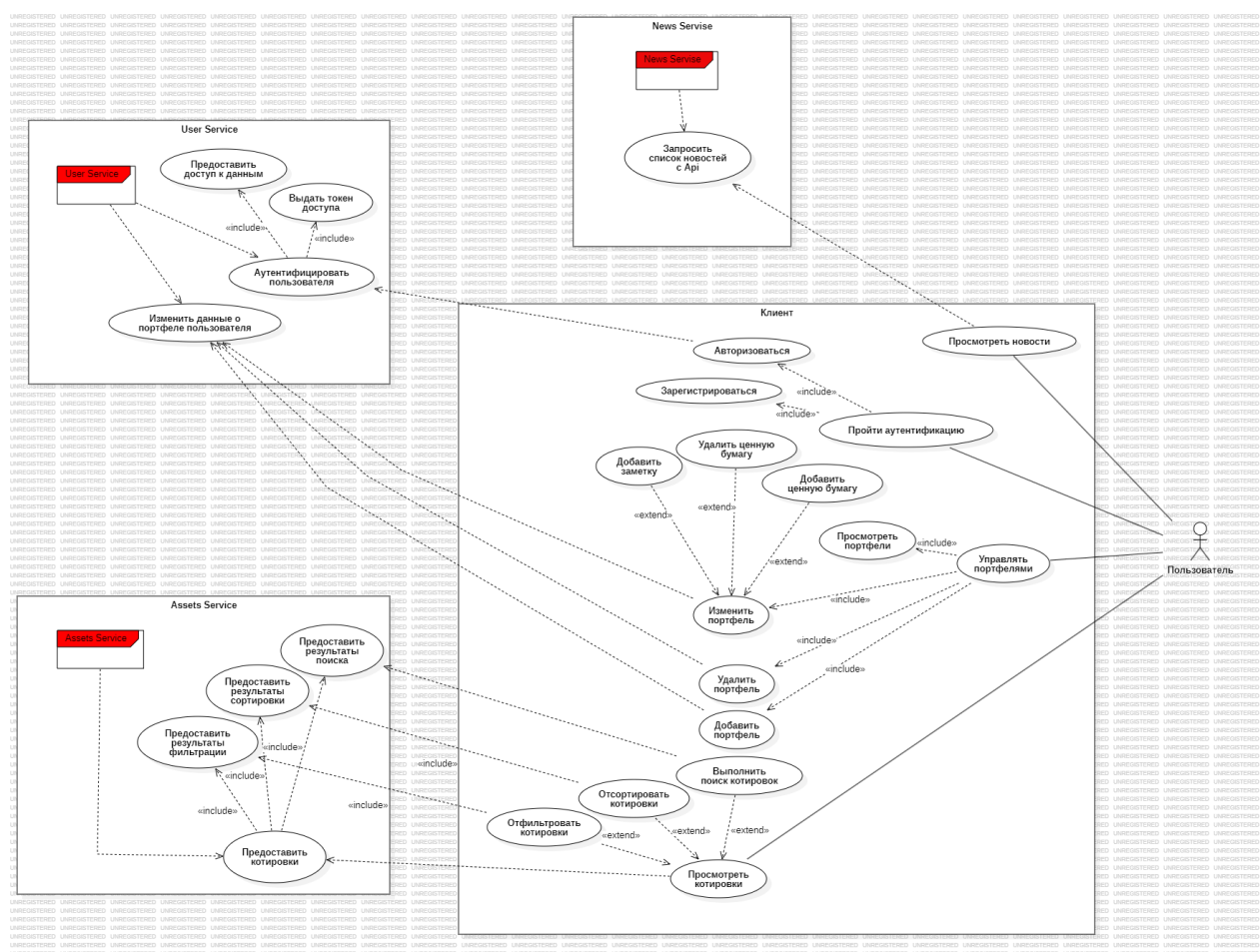


Рисунок 1 – Диаграмма вариантов использования информационной системы

Диаграмма вариантов использования показывает спектр задач, решаемых информационной системой. Пользователь информационной системы имеет следующие варианты использования:

- «Просмотреть новости»;
- «Пройти аутентификацию», содержащий «Авторизоваться», «Зарегистрироваться»;
- «Управлять портфелями», содержащий «Просмотреть портфели», «Удалить портфель», «Добавить портфель», «Изменить портфель», который расширяется «Добавить заметки», «Удалить ценную бумагу», «Добавить ценную бумагу»;
- «Просмотреть котировки», который расширяется «Выполнить поиск котировок», «Отсортировать котировки», «Отфильтровать котировки».

Таким образом, диаграмма вариантов использования показывает прецеденты, происходящие в системе и изменения, которые нужно регистрировать [4].

1.4 Выбор средств разработки

Для быстрого и качественного написания кода, необходимо выбрать наиболее подходящее инструментальное средство разработки программного обеспечения. В качестве таких продуктов были выбраны: IntelliJ IDEA, Android Studio.

IntelliJ IDEA – интегрированная среда разработки программного обеспечения для многих языков программирования, в частности Java, JavaScript, Python, разработанная компанией JetBrains.

IntelliJ IDEA Community Edition – бесплатный вариант для личного и коммерческого использования. Функциональность, по сравнению с версией Ultimate, значительно ограничена: нет встроенного HTTP-клиента, отсутствуют инструменты для работы с базами данных, не поддерживаются совместная работа и удаленный доступ.

IntelliJ IDEA обладает достаточно внушительным спектром функций, которые позволяют повысить разработчикам производительность, а также поддержку таких параметров как:

- интеллектуальное редактирование кода;
- поиск кода;
- проверка кода;
- интеграция с системой управления версиями;
- отладка кода;
- поддержка пользовательской настройки среды разработки;
- экосистема JetBrains;
- предварительные просмотры кода;
- понятный и простой пользовательский интерфейс.

На рисунке 2 показан интерфейс IntelliJ IDEA.

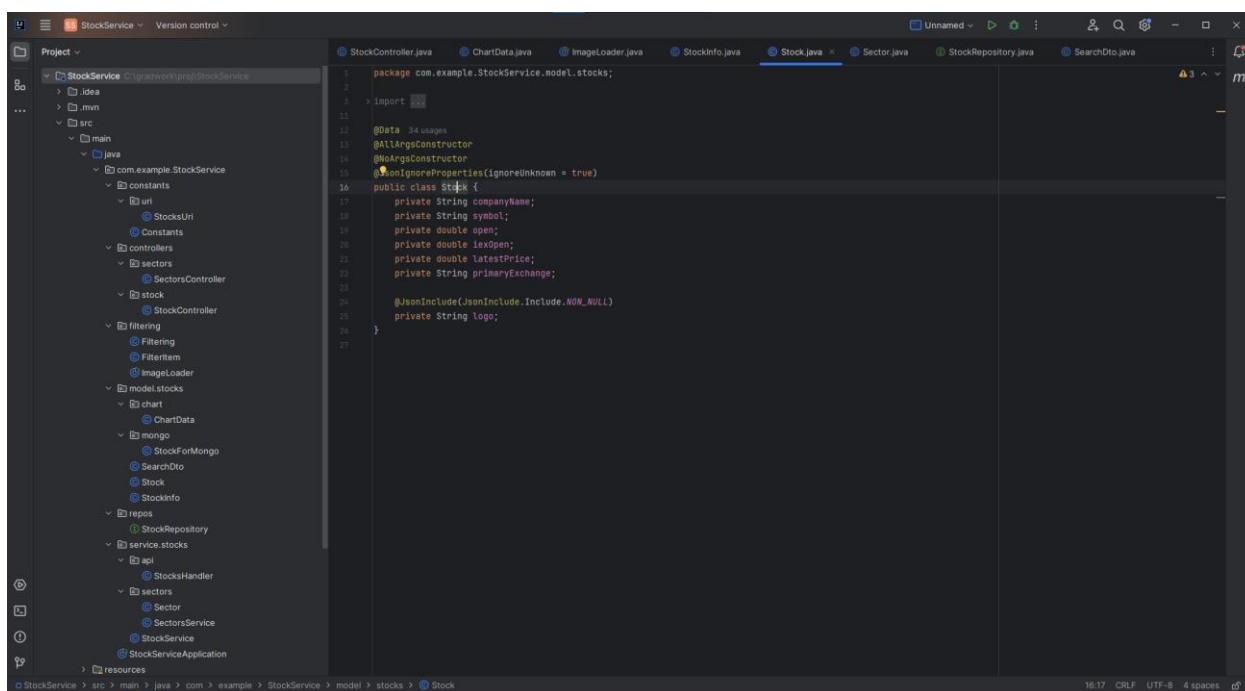


Рисунок 2 – Интерфейс IntelliJ IDEA

IntelliJ IDEA был выбран в качестве средства разработки, так имеет: более удобный интерфейс и внешнего оформления, автозаполнение, позволяющее быстро писать код, возможность простой интеграции с GitHub и управлении версиями кода [5].

Android Studio – это интегрированная среда разработки (integrated development environment) от Google для создания Android-приложений. В ней можно писать код, проектировать графический интерфейс, проводить отладку и сборку приложений для публикации в магазинах.

Android Studio имеет следующие возможности:

- интегрированная среда разработки;
- визуальный редактор макетов;
- эмулятор Android;
- управление проектами;
- анализатор производительности;
- поддержка различных языков программирования;
- интеграция с Google Services;
- предустановленные шаблоны проектов.

На рисунке 3 показан пользовательский интерфейс Android Studio.

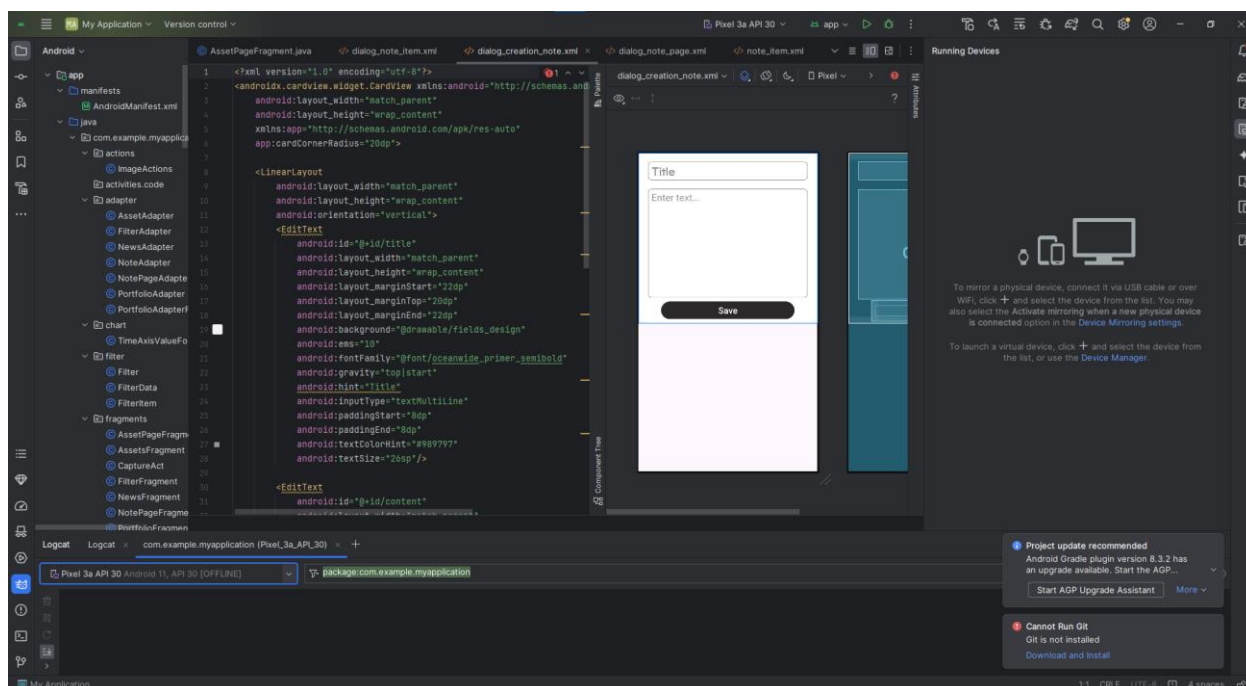


Рисунок 3 – Интерфейс Android Studio

Android Studio был выбран в качестве основного средства разработки андроид приложения, так как предоставляет наиболее полный функционал в плане написания кода и создания макетов приложения и интеграции с Google, а также имеет удобный пользовательский интерфейс [6].

1.5 Планирование процесса разработки программного обеспечения

Одним из важнейших пунктов разработки программного обеспечения является планирование процесса разработки. Особенность процесса планирования в проектной форме заключается в постоянной корректировке плана от начала до стадии завершения.

Для планирования разработки использовалась диаграмма Ганта. Эта диаграмма является одной из самых популярных для построения планирования деятельности, в данном случае для разработки приложения [7].

Для построения диаграммы Ганта был выбран Microsoft Project 2016. Он позволяет отдельно создавать ресурсы и задачи, которые требуются на проект. На рисунке 4 изображены ресурсы проекта.

Название ресурса	Тип	Единицы измерения	Краткое название	Группа	Макс. единиц	Стандартная ставка	Ставка сверхурочн	Затраты на исполыз.	Начисление	Базовый календарь
Аналитик	Трудовой		Ан		100%	100,00 Р/ч	150,00 Р/ч	0,00 Р	Пропорционал	Стандартный
Разработчик ПО	Трудовой		ПО		100%	150,00 Р/ч	210,00 Р/ч	0,00 Р	Пропорционал	Стандартный
Тестировщик	Трудовой		Т		100%	130,00 Р/ч	180,00 Р/ч	0,00 Р	Пропорционал	Стандартный
Android разработчик	Трудовой		Анд		100%	180,00 Р/ч	250,00 Р/ч	0,00 Р	Пропорционал	Стандартный
Интернет	Затраты		И						Пропорционал	

Рисунок 4 – Ресурсы проекта

При планировании проекта необходимо выделить задачи и подзадачи, которые необходимо решить. Нужно задать даты начала и окончания той или иной работы. В Microsoft Project удобно задать дни, а также выделить совокупность задач для одного шага проекта. Также после этого можно задать ресурсы для всего каждого этапа. Общая таблица, которая отображает задачи и их ресурсы, изображена на рисунке 5.

Режим задачи	Название задачи	Длительность	Начало	Окончание	Предшественн	Названия ресурсов
✚	Начало реализации	1 день	Пн 04.03.24	Пн 04.03.24		
✚	▲ Постановка задачи и анализ требований	10 дней	Вт 05.03.24	Пн 18.03.24	1	
✚	Постановка задачи на разработку	2 дней	Вт 05.03.24	Ср 06.03.24	1	Аналитик;Интерне
✚	Анализ программного обеспечен	2 дней	Чт 07.03.24	Пт 08.03.24	3	Аналитик;Интерне
✚	Определение целей разработки	2 дней	Пн 11.03.24	Вт 12.03.24	4	Аналитик;Интерне
✚	Формулировка и анализ требова	2 дней	Ср 13.03.24	Чт 14.03.24	5	Аналитик;Интерне
✚	Выбор средств разработки	2 дней	Пт 15.03.24	Пн 18.03.24	6	Аналитик;Интерне
✚	▲ Проектирование	21 дней	Вт 05.03.24	Вт 02.04.24	1	
✚	Проектирование структур данны	5 дней	Вт 05.03.24	Пн 11.03.24	1	Разработчик ПО
✚	Проектирование структуры ПО	10 дней	Вт 12.03.24	Пн 25.03.24	9	Android разработч
✚	Проектирование интерфейса	6 дней	Вт 26.03.24	Вт 02.04.24	10	Android разработч
✚	▲ Программная реализация	30 дней	Ср 03.04.24	Вт 14.05.24	8	
✚	Реализация базы данных	5 дней	Ср 03.04.24	Вт 09.04.24	8	Разработчик ПО
✚	Реализация серверной части	25 дней	Ср 10.04.24	Вт 14.05.24	13	Разработчик ПО
✚	Реализация клиентской части	25 дней	Ср 10.04.24	Вт 14.05.24	13	Android разработч
✚	▲ Тестирование системы	15 дней	Ср 15.05.24	Вт 04.06.24	12	
✚	Обоснование выбора моделей те	2 дней	Ср 15.05.24	Чт 16.05.24	12	Тестировщик
✚	Тестирование программных сред	10 дней	Пт 17.05.24	Чт 30.05.24	17	Тестировщик
✚	Оценка качества программного к	3 дней	Пт 31.05.24	Вт 04.06.24	18	Тестировщик
✚	▲ Завершение проекта	15 дней	Ср 05.06.24	Вт 25.06.24	16	
✚	Составление документации	10 дней	Ср 05.06.24	Вт 18.06.24	16	Аналитик
✚	Интеграция приложения	5 дней	Ср 19.06.24	Вт 25.06.24	21	Android разработч
✚	Проект завершен	0 дней	Вт 25.06.24	Вт 25.06.24	22	
✚	Конец проекта	0 дней	Ср 26.06.24	Ср 26.06.24	23	

Рисунок 5 – Перечень задач и ресурсов проекта

После этого можно создать графическое представление календарного плана проекта. На рисунке 6 показана диаграмма Ганта.

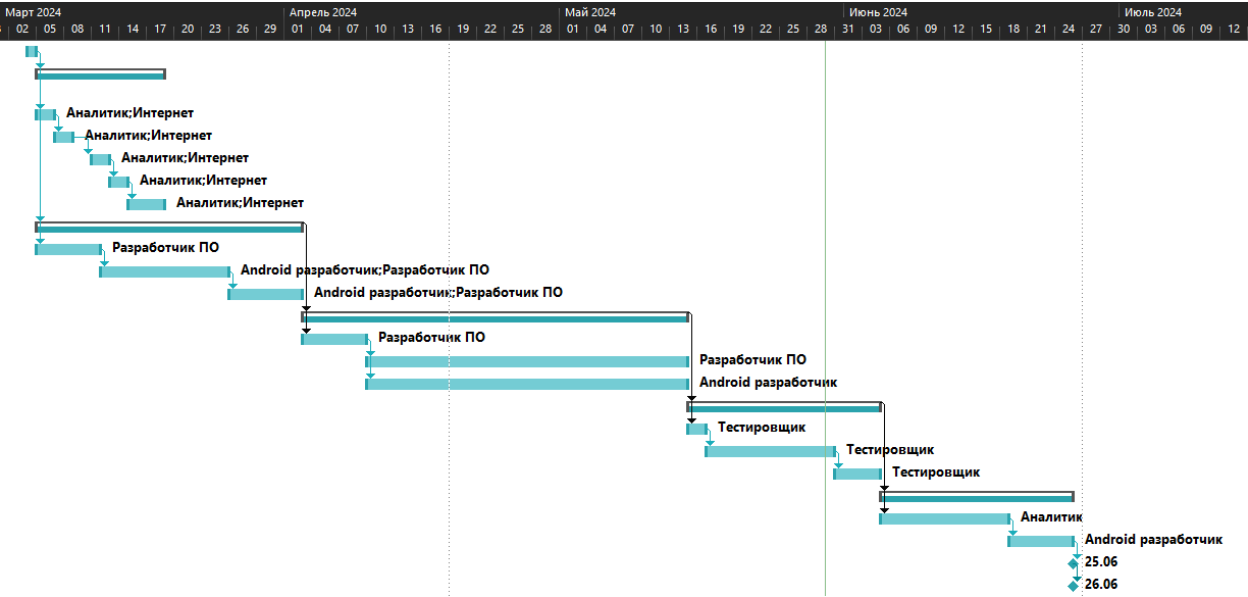


Рисунок 6 – Диаграмма Ганта

Microsoft Project имеет в своем распоряжении набор готовых отчетов и панель мониторинга, контролирующей общее состояние проекта. Также он выполняет проверку и соответствие по графику [8].

На рисунке 7 показан отчет по затратам.

ОБЗОР ЗАТРАТ РЕСУРСОВ

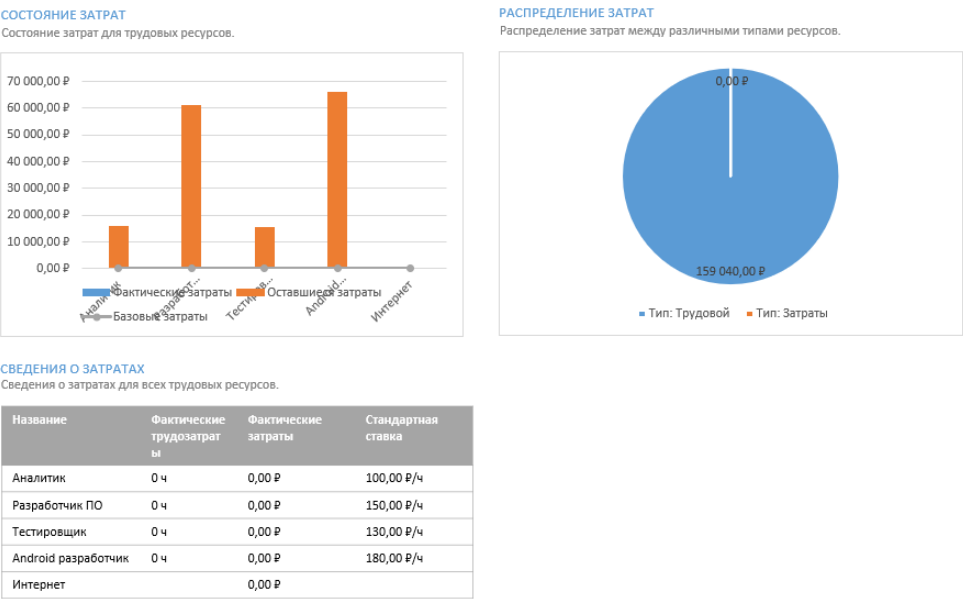


Рисунок 7 – Обзор затрат задачи

По этому отчету видно, что фактические затраты реализации информационной системы для инвесторов в виде мобильного приложения составили 159 040 рублей, длительность проекта, с учетом всех дней (рабочих, праздников и выходных) составила 114 дней.

2 Проектирование информационной системы для инвесторов

2.1 Проектирование структур данных информационной системы

Проектирование эффективных структур данных является критическим аспектом разработки приложений. Выбор подходящих структур данных может существенно повлиять на их производительность и масштабируемость. Эффективные структуры данных позволяют более быстро выполнять операции чтения, записи и поиска данных, уменьшая время ответа приложения и обеспечивая более гладкую пользовательскую интерактивность.

Кроме того, хорошо спроектированные структуры данных обеспечивают оптимальное использование ресурсов, таких как память и процессорное время. Они предоставляют эффективные алгоритмы для выполнения различных операций над данными, что приводит к сокращению накладных расходов и повышению производительности приложения в целом.

При проектировании структур данных информационной системы на базе микросервисов в виде мобильного приложения были выдвинуты следующие требования:

- быстрый доступ к данным: важно, чтобы структуры данных обеспечивали быстрый доступ к нужным данным;
- эффективное использование памяти: структуры данных должны быть спроектированы таким образом, чтобы эффективно использовать выделенную память и минимизировать ее потребление;
- масштабируемость: приложению может потребоваться обрабатывать большие объемы данных и поддерживать высокую производительность даже при увеличении нагрузки, структуры данных должны быть способны масштабироваться и обрабатывать данные эффективно независимо от их объема;
- гибкость и расширяемость: структуры данных должны быть гибкими и легко расширяемыми, чтобы можно было адаптировать их к

изменяющимся требованиям и добавлять новую функциональность без значительных изменений в коде приложения.

2.1.1 Проектирование базы данных

В качестве СУБД для информационной системы была выбрана MongoDB.

MongoDB – документоориентированная система управления базами данных, не требующая описания схемы таблиц. Является одним из классических примеров NoSQL-систем, использует JSON-подобные документы и схему базы данных [9].

Использование данной СУБД дает следующие преимущества при разработке информационной системы в виде Android приложения:

- гибкая модель данных: MongoDB позволяет хранить различные типы данных без жесткой схемы, что особенно полезно в информационных приложениях, где данные могут быть неструктурированными или изменяться со временем;
- быстрый доступ к данным: MongoDB обеспечивает быстрый доступ к данным благодаря своей способности индексировать и оптимизировать запросы, это особенно важно для информационных приложений, где пользователи ожидают мгновенного доступа к актуальным данным;
- масштабируемость: MongoDB легко масштабируется и может работать на нескольких серверах, что позволяет обрабатывать большие объемы данных и поддерживать высокую производительность даже при увеличении нагрузки;
- удобство использования с Java: MongoDB обладает отличной поддержкой для языка программирования Java, она предоставляет официальный драйвер Java, который позволяет легко взаимодействовать с базой данных MongoDB из приложений, написанных на Java.

Модель базы данных представлена в виде ER диаграммы на рисунке 8.

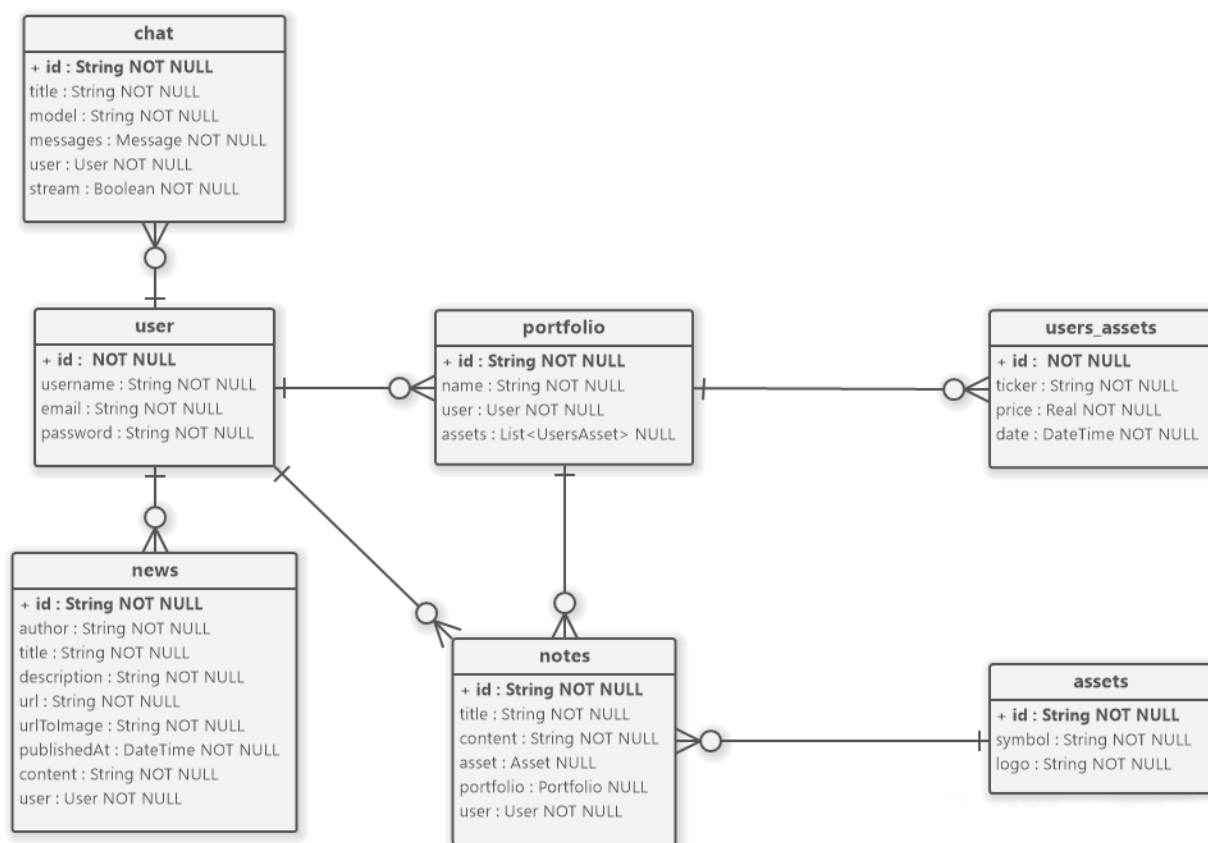


Рисунок 8 – ER диаграмма базы данных

Описание атрибутов объекта «user» представлена в таблице 2.

Таблица 2 – Описание атрибутов объекта «user»

Атрибуты	Нулевое значение	Ключевое поле
id	not null	PK
username	not null	
email	not null	
password	not null	

Описание атрибутов объекта «news» представлена в таблице 3.

Таблица 3 – Описание атрибутов объекта «news»

Атрибуты	Нулевое значение	Ключевое поле
id	not null	PK
author	not null	
title	not null	
description	not null	
url	not null	
urlToImage	not null	
publishedAt	not null	
content	not null	
user	not null	

Описание атрибутов объекта «notes» представлена в таблице 4.

Таблица 4 – Описание атрибутов объекта «notes»

Атрибуты	Нулевое значение	Ключевое поле
id	not null	PK
title	not null	
content	not null	
asset	null	
portfolio	null	
user	not null	

Описание атрибутов объекта «portfolio» представлена в таблице 5.

Таблица 5 – Описание атрибутов объекта «portfolio»

Атрибуты	Нулевое значение	Ключевое поле
id	not null	PK
name	not null	
user	not null	
assets	null	

Описание атрибутов объекта «user_assets» представлена в таблице 6.

Таблица 6 – Описание атрибутов объекта «user_assets»

Атрибуты	Нулевое значение	Ключевое поле
id	not null	PK
ticker	not null	
price	not null	
date	not null	

Описание атрибутов объекта «assets» представлена в таблице 7.

Таблица 7 – Описание атрибутов объекта «assets»

Атрибуты	Нулевое значение	Ключевое поле
id	not null	PK
symbol	not null	
logo	not null	

Описание атрибутов объекта «chat» представлена в таблице 8.

Таблица 8 – Описание атрибутов объекта «chat»

Атрибуты	Нулевое значение	Ключевое поле
id	not null	PK
title	not null	
model	not null	
messages	not null	

Продолжение таблицы 8

Атрибуты	Нулевое значение	Ключевое поле
user	not null	
stream	not null	

Таким образом, была спроектирована структура базы данных для информационной системы для инвесторов в виде мобильного приложения на микросервисах.

2.1.2 Проектирование серверной части

Микросервисы были реализованы на языке Java с использованием фреймворка Spring. Их структуру можно описать с помощью диаграмм классов.

Диаграмма классов микросервиса «UserService» изображена на рисунке 9.

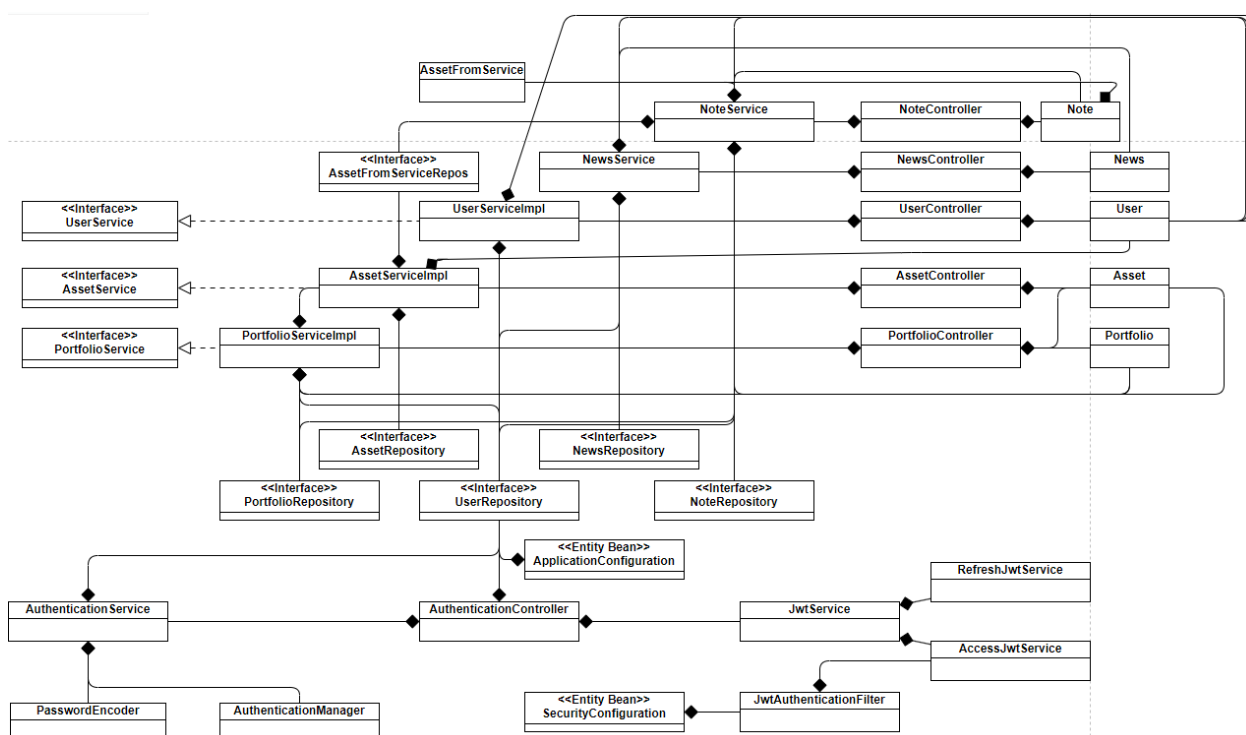


Рисунок 9 – Диаграмма классов микросервиса «UserService»

Микросервис представлен в виде отдельных пакетов, которые создаются библиотекой Spring.

Классы микросервиса «UserService» и их свойства:

- NoteController, NewsController, UserController, AssetController, PortfolioController, AuthenticationController: отвечают за обработку HTTP-запросов, управление потоками данных и взаимодействие с другими компонентами приложения, каждый контроллер содержит энд-поинты для HTTP-запросов и собственный API адрес;
- PortfolioService, AssetService, UserService, NewsService, NoteService, AuthenticationService: выполняют бизнес-логику и управляют операциями, связанными с обработкой данных и выполнением бизнес-задач;
- PortfolioRepository, AssetRepository, UserRepository, NewsRepository, NoteRepository: предоставляют удобный способ взаимодействия с базой данных, абстрагируя сложности работы с ней и предоставляя простой API;
- JwtService, AccessJwtService, RefreshJwtService: отвечают за генерацию и проверку JSON Web Token;
- Note, News, Portfolio, Asset, User: классы, представляющие собой модели данных для хранения информации в БД.

Диаграмма классов микросервиса «NewsService» изображена на рисунке 10.

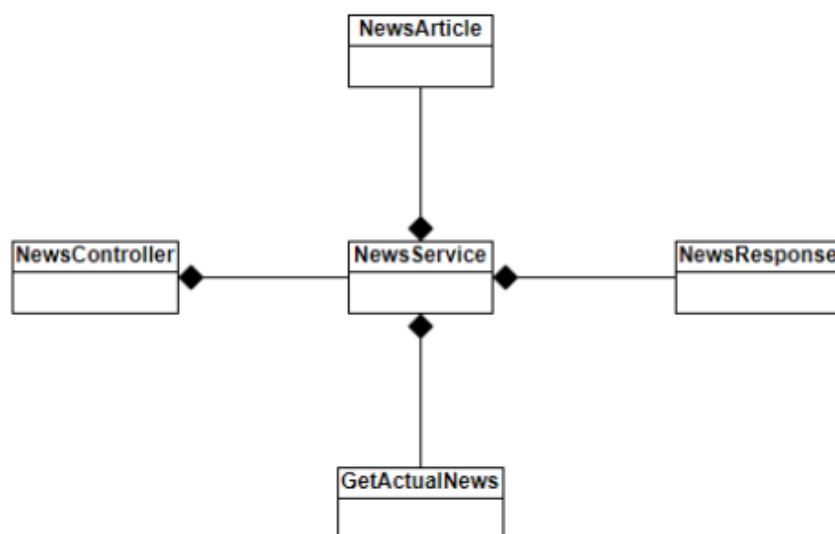


Рисунок 10 – Диаграмма классов микросервиса «NewsService»

Микросервис представлен в виде отдельных пакетов, которые создаются библиотекой Spring.

Классы микросервиса «NewsService» и их свойства:

- NewsController: отвечает за обработку HTTP-запроса, управление потоком данных и взаимодействие с другими компонентами, содержит энд-поинты для HTTP-запросов;
- GetActualNews: класс, получающий все последние новости из новостного API и передающий их в NewsService;
- NewsArticle: класс, содержащий всю информацию о новостях;
- NewsResponse: является моделью данных для отправки HTTP-ответа;
- NewsService: выполняет бизнес-логику и управляет операциями, связанными с обработкой новостей.

Диаграмма классов микросервиса «StockService» изображена на рисунке 11.

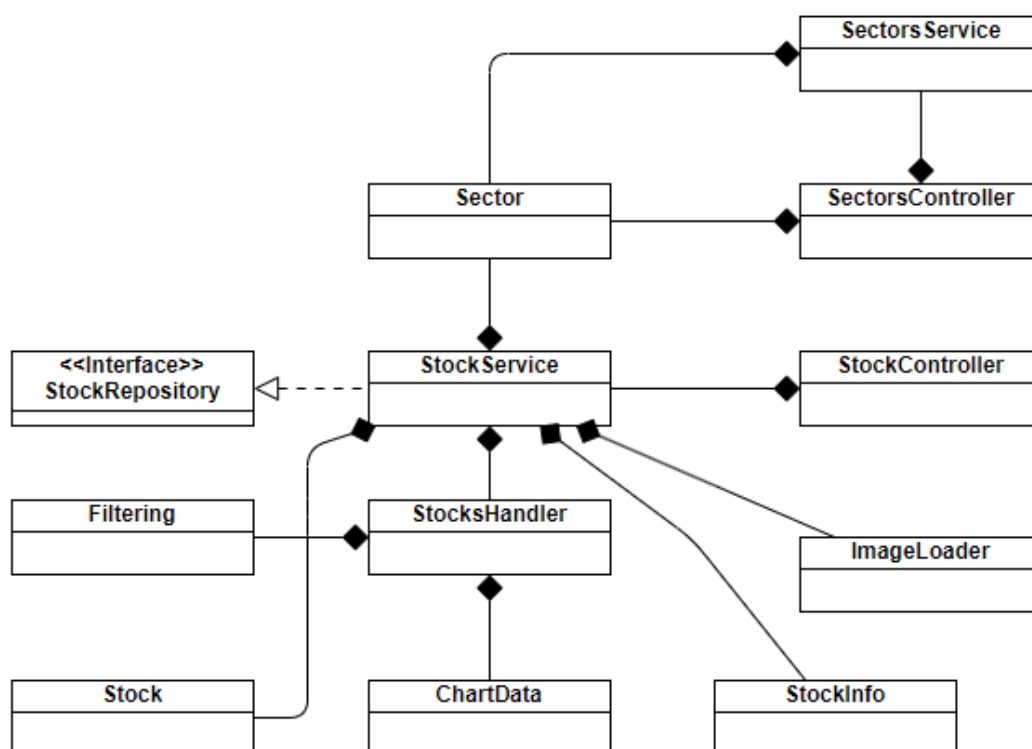


Рисунок 11 – Диаграмма классов микросервиса «StockService»

Микросервис представлен в виде отдельных пакетов, которые создаются библиотекой Spring.

Классы микросервиса «StockService» и их свойства:

- StockController, SectorsController: отвечают за обработку HTTP-запросов, управление потоками данных и взаимодействие с другими компонентами, содержит энд-поинты для HTTP-запросов;
- StockService, SectorsService: выполняют бизнес-логику и управляют операциями, связанными с обработкой данных и выполнением бизнес-задач;
- StocksHandler: класс, из которого отправляются запросы на биржевой API и который возвращает ответы в виде строки JSON;
- Filtering: класс, отвечающий за фильтрацию массивов акций по запросам клиента;
- Stock, ChartData, StockInfo, Sector: классы, представляющие модели данных для HTTP-ответов;
- StockRepository: предоставляет удобный способ взаимодействия с базой данных.

Диаграмма классов микросервиса «ChatService» изображена на рисунке 12.

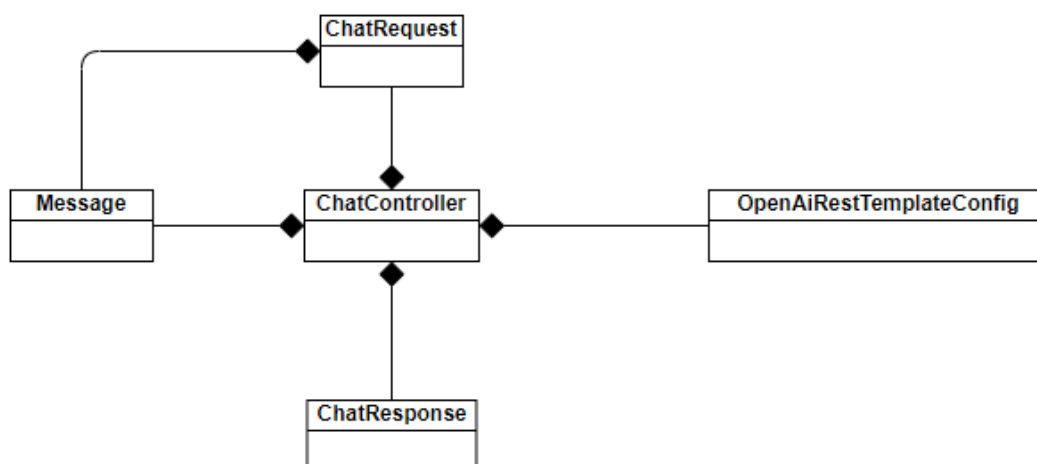


Рисунок 12 – Диаграмма классов микросервиса «ChatService»

Микросервис представлен в виде отдельных пакетов, которые создаются библиотекой Spring.

Классы микросервиса «ChatService» и их свойства:

- ChatController: отвечают за обработку HTTP-запросов, управление потоками данных и взаимодействие с другими компонентами, содержит энд-поинты для HTTP-запросов;
- ChatRequest, ChatResponse: являются моделями данных для отправки и приема данных через HTTP-запросы;
- Message: является моделью данных для сообщений;
- OpenAiRestTemplateConfig: класс конфигурации для отправки запроса на API нейросети.

Таким образом, была спроектирована серверная часть информационной системы для инвесторов на базе микросервисов.

2.2 Проектирование структуры программного обеспечения

При разработке программного обеспечения, производится анализ функциональных возможностей приложения и автоматизируемые им бизнес-процессы с целью выявления наиболее подходящей архитектуры для создаваемой системы или средства.

Микросервисная архитектура приложения – это подход к разработке программного обеспечения, в котором приложение разбивается на небольшие и независимые сервисы. Каждый из них выполняет свою конкретную функцию. В отличие от одной большой монолитной системы, микросервисы взаимодействуют друг с другом через легковесные API, что позволяет вынести следующие преимущества:

- разделение функциональности: приложение разделяется на независимые сервисы, каждый из которых предоставляет определенную функциональность и может быть разработан, развернут и масштабирован независимо от других;

- легковесное взаимодействие: сервисы «общаются» друг с другом посредством API, что позволяет им быть независимыми и заменяемыми;
- отказоустойчивость: если один из микросервисов выходит из строя, остальные сервисы могут продолжать работу;
- быстрая разработка и внедрение изменений: благодаря независимой разработке и развертыванию сервисов, можно быстро внедрять новые функции или исправлять ошибки без необходимости влиять на весь проект;
- масштабируемость: каждый сервис может быть отдельно масштабирован, чтобы удовлетворять изменяющимся потребностям нагрузки, можно масштабировать только необходимые компоненты системы, минимизируя издержки масштабирования.

В качестве архитектуры программного обеспечения для информационной системы для инвесторов была выбрана микросервисная архитектура [10].

Проектирование структуры программного обеспечения является важной частью разработки информационных систем. Оно позволяет определить, как компоненты системы будут взаимодействовать друг с другом и как организованы данные внутри.

Диаграмма развертывания – это особый тип диаграммы компонентов, который отображает физическую конфигурацию программной системы на уровне хостов и сетей. Она позволяет показать, как компоненты распределены по различным узлам и как они взаимодействуют друг с другом через сеть [11].

Диаграмма развертывания информационной системы для инвесторов на базе микросервисов в виде андроид приложения представлена на рисунке 13.

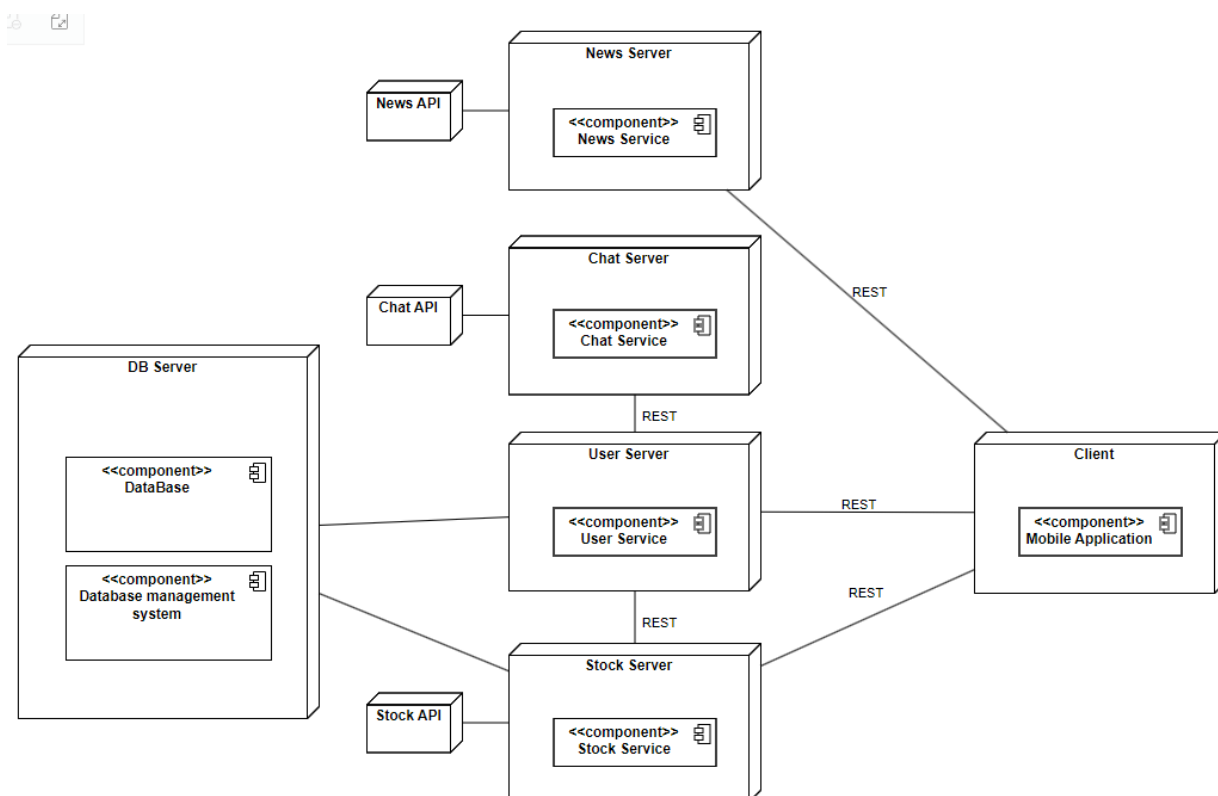


Рисунок 13 – Диаграмма развертывания информационной системы

Клиентская часть запрашивает данные у серверной части посредством REST API. В свою очередь, серверная часть использует сервер базы данных для хранения и обработки информации.

Новостной микросервис предоставляет актуальные биржевые новости, взятые из открытого новостного API в виде массива JSON.

Микросервис по управлению данными пользователя отвечает за аутентификацию пользователя в системе, добавление, хранение и удаление данных пользователя о портфелях, ценных бумагах, заметках и новостях.

Микросервис котировок осуществляет все взаимодействие пользователя с ценными бумагами. Он предоставляет всю актуальную информацию о биржевом рынке, осуществляет выборку данных по запросам пользователя.

Взаимодействие элементов архитектуры для информационной системы для инвесторов можно представить с помощью диаграммы последовательности.

Диаграмма последовательности позволяет описывать последовательность работы в приложении, где определенные действия приводят к какому-то результату [12].

На рисунке 14 представлена диаграмма последовательности взаимодействия компонентов информационной системы.

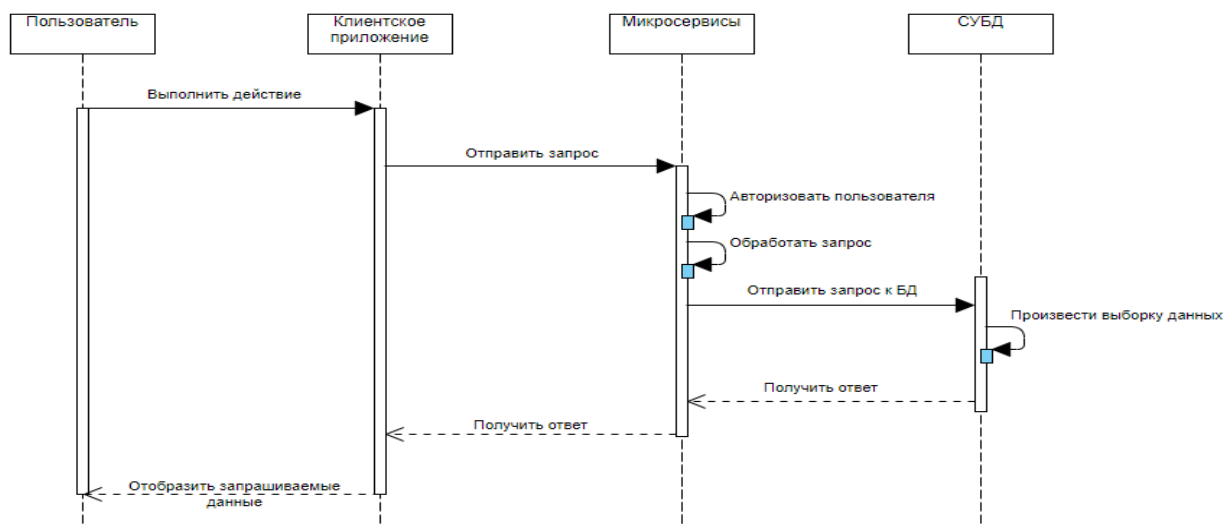


Рисунок 14 – Диаграмма последовательности информационной системы

Таким образом, была разработана архитектура информационной системы на базе микросервисов в виде андроид приложения.

2.2.1 Проектирование структуры серверной части

Диаграмма деятельности представляет собой UML-диаграмму, которая описывает выполняемые действия в системе, специфицирует бизнес-процессы. Она помогает разработчикам понять, как должен работать процесс, находить ошибки или улучшения. Используется для планирования и координации различных этапов разработки системы, обеспечивая понимание последовательности действий и их зависимости [13].

С помощью диаграммы деятельности описана логика работы микросервиса по управлению данными пользователя. Она представлена на рисунке 15.

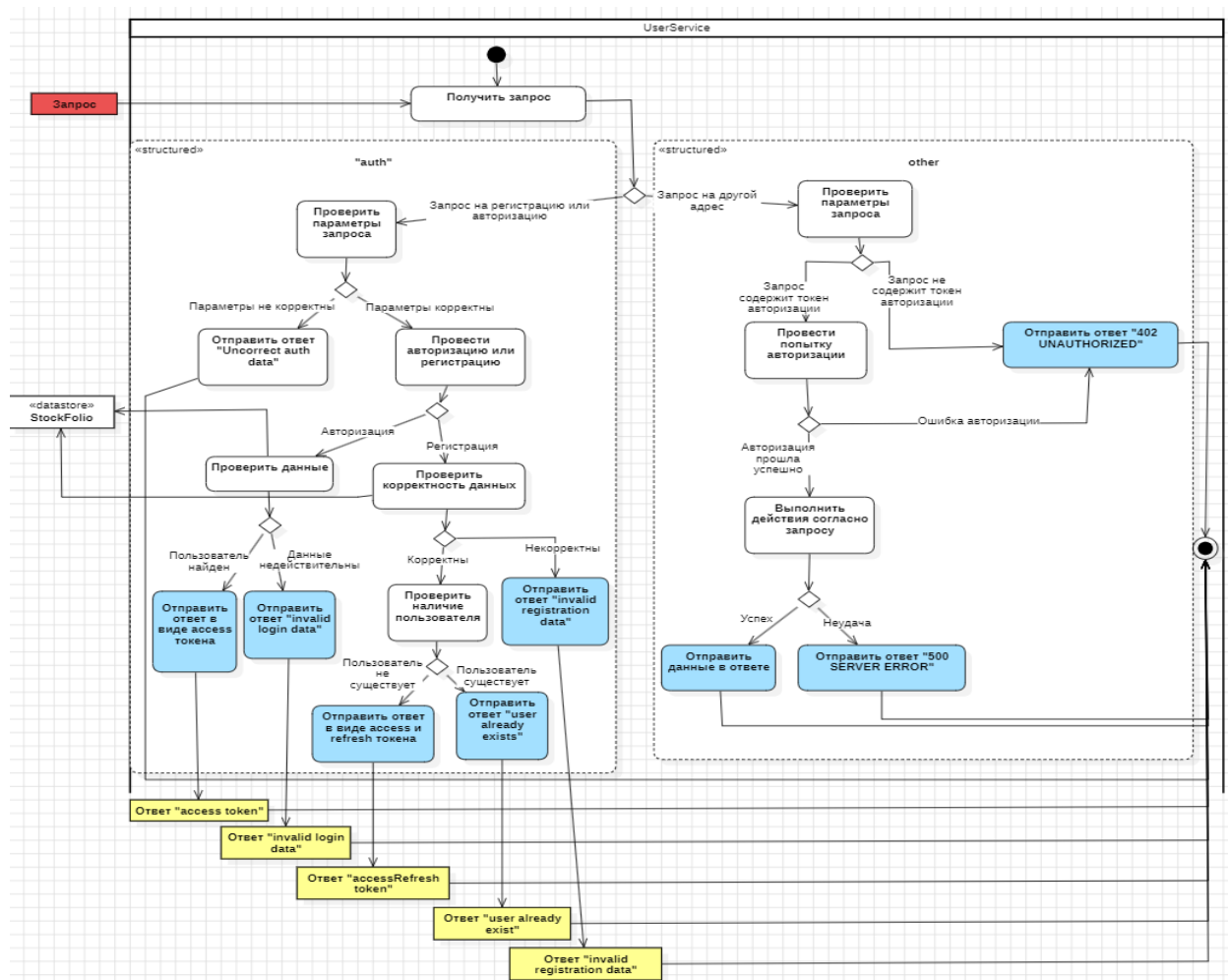


Рисунок 15 – Диаграмма деятельности микросервиса UserService

Пользователь входит в приложение. Открывается страница авторизации, на которой вводятся данные для входа в систему. Если пользователь не имеет аккаунта, он может перейти на страницу регистрации и также заполнить форму. После фиксирования нажатия на кнопку «Войти» или «Зарегистрироваться» отправляется соответствующий запрос на микросервис по управлению данными пользователя. На сервере проводятся всевозможные проверки поступающего запроса и его содержания. В зависимости от запроса, пользователь получает уведомление об ответе сервера.

Подобным образом работают и другие микросервисы. Пользователь выполняет действие, клиент формирует запрос и отправляет его на нужный микросервис. Сервисы обрабатывают запросы и отправляют ответы в зависимости от поступающего запроса. Единственное отличие микросервисов

«Stock Service» и «News Service» от «User Service» – это отсутствие необходимости проведения авторизации пользователя. Это связано с тем, что данные сервисы не содержат никаких данных, связанных с пользователем приложения, а представляют лишь информационное обслуживание приложения.

2.2.2 Проектирование структуры клиентской части

Android Studio является официальной интегрированной средой разработки (IDE) для разработки приложений под платформу Android. Есть несколько причин, почему Android Studio был выбран для разработки информационной системы на базе Android:

- поддержка Google: Android Studio разработана и поддерживается компанией Google, что гарантирует актуальность и соответствие инструментов и функциональности платформе Android;
- интеграция с Android SDK: Android Studio полностью интегрирована с Android Software Development Kit (SDK), что обеспечивает доступ к широкому набору инструментов, библиотек и ресурсов, необходимых для разработки Android приложений;
- улучшенные инструменты разработки: Android Studio предлагает мощные инструменты, такие как расширенный редактор кода, автозавершение, отладчик, интегрированная система сборки Gradle и другие, значительно упрощающие процесс разработки;
- эмуляторы и тестирование: Android Studio предоставляет эмуляторы устройств Android, а также возможность запуска и отладки приложений на физических устройствах, также есть инструменты для тестирования приложений, включая автоматическое тестирование и инструменты для создания юнит-тестов и инструментальных тестов.

Клиентская часть информационной системы для инвесторов на базе микросервисов в виде андроид приложения была реализована с помощью

Android Studio. Диаграмма компонентов описывает ее структуру. Она представлена на рисунке 16.

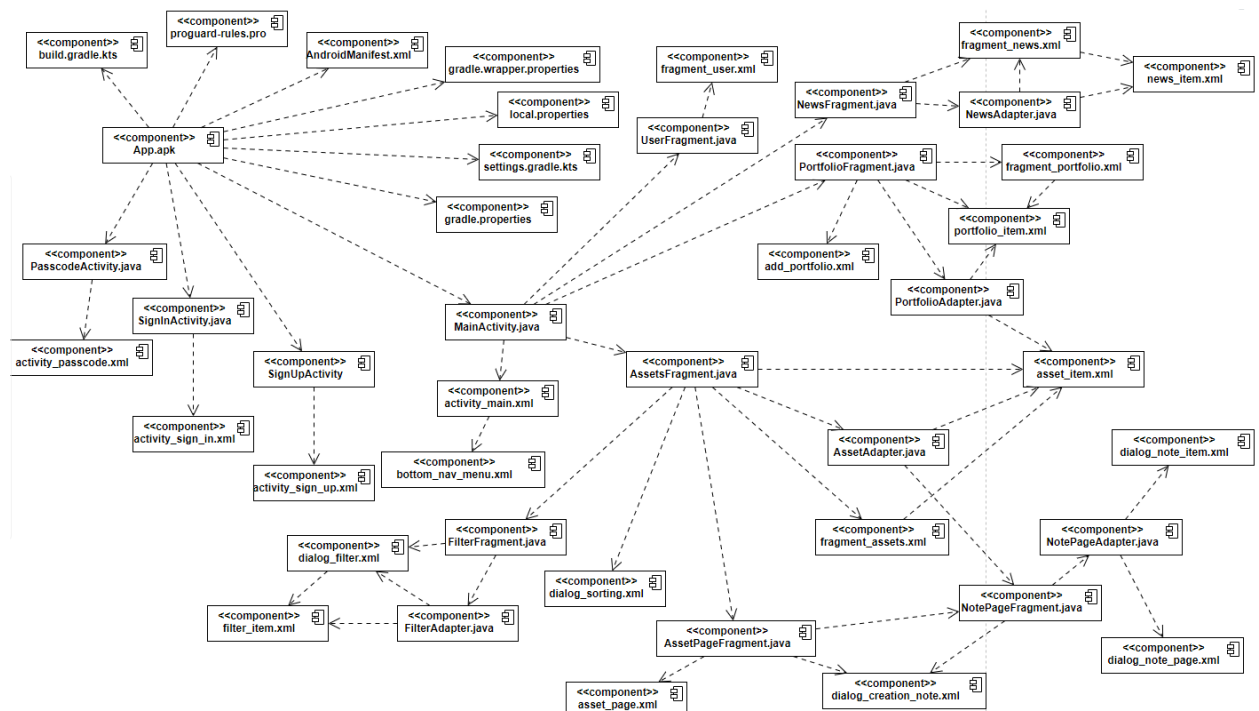


Рисунок 16 – Диаграмма компонентов клиентской части информационной системы

Каждый компонент отвечает за свои функции и отображение данных. Клиентская часть информационной системы содержит компоненты:

- App.apk: это файл, который содержит собранный исполняемый код приложения, он используется для установки и запуска приложения на устройствах Android;
- proguard-rules.pro: этот файл содержит правила ProGuard, которые определяют, какой код должен быть оптимизирован или удален при построении проекта, применение ProGuard может помочь уменьшить размер и улучшить производительность приложения.;
- build.gradle.kts: этот файл является основным скриптом сборки для проекта Android, он содержит информацию о зависимостях, плагинах, настройках сборки, версиях SDK и другие параметры проекта;

- `gradle.properties`: в этом файле задаются пользовательские свойства для системы сборки Gradle, такие как настройки прокси, настройки кэша и другие параметры конфигурации Gradle;
- `settings.gradle.kts`: этот файл используется для настройки структуры проекта Android, в нем указываются модули, которые должны быть включены в сборку приложения.;
- `local.properties`: этот файл содержит путь к SDK Android на компьютере, он автоматически создается Android Studio и используется для указания местоположения SDK;
- `gradle-wrapper.properties`: этот файл содержит настройки для Gradle Wrapper, инструмента, который позволяет использовать конкретную версию Gradle для проекта без необходимости установки Gradle вручную, здесь можно указать версию Gradle, которую следует использовать, и другие параметры конфигурации;
- `AndroidManifest.xml`: этот файл является основным манифестом приложения Android, он содержит информацию о приложении, такую как имя пакета, разрешения, список активностей, службы и другие компоненты приложения;
- `MainActivity.java`, `PasscodeActivity.java`, `SignInActivity.java`, `SignUpActivity.java`: классы, являющиеся основными компонентами приложения, представляют собой экран или интерфейс пользователя, с которым пользователь может взаимодействовать, каждая активность соответствует одному экрану пользовательского интерфейса;
- `AssetPageFragment.java`, `AssetsFragment.java`, `FilterFragment.java`, `NewsFragment.java`, `NotePageFragment.java`, `PortfolioFragment.java`, `UserFragment.java`: представляют собой повторно используемые куски пользовательского интерфейса или поведения, которые могут быть использованы внутри активностей;

– `AssetAdapter.java`, `FilterAdapter.java`, `NewsAdapter.java`, `NoteAdapter.java`, `NotePageAdapter.java`, `PortfolioAdapter.java`: классы, связывающие данные с компонентами пользовательского интерфейса `ListView` или `ExpandableListView`, они предоставляют механизмы для отображения элементов данных в виде списка на экране;

– другие файлы `xml` разметки: используются для определения пользовательского интерфейса и других ресурсов приложения.

Основной точкой входа в приложение Android является `app.apk`, который содержит в себе все компоненты. Главной страницей приложения является компонент `MainActivity.java`, содержащий фрагменты: `PortfolioFragment.java`, `AssetsFragment.java`, `UserFragment.java` и `NewsFragment.java`. Каждый из них содержит адаптеры, другие фрагменты и `xml` файлы с ресурсами формы. При входе в приложение пользователя встречают `SignInActivity.java`, отвечающий за авторизацию, и `SignUpActivity.java`, отвечающий за регистрацию. После аутентификации пользователь должен придумать пароль на странице `PasscodeActivity.java`. Этот пароль будет запрашиваться при каждом открытии приложения.

Таким образом, клиентская часть информационной системы представляет собой совокупность взаимодействующих компонентов, каждый из которых отвечает за свой раздел приложения и содержит свойственную ему информацию.

2.3 Проектирование интерфейса

Проектирование интерфейса является важным шагом в процессе разработки приложения. В данном случае проектирование – это процесс создания макета рабочего интерфейса на основе нарисованных и готовых макетов. Это позволяет изобразить приложение до написания кода [14].

При проектировании интерфейса Android-приложения должны быть учтены следующие требования:

- интуитивность: интерфейс должен быть интуитивно понятным и легко осваиваемым для пользователей разного уровня опыта, элементы управления и действия должны являться понятными и доступными без необходимости чтения инструкций;
- простота: должны быть обеспечены простота и минимализм в дизайне, нельзя допускать излишнего количества информации и элементов на экране, чтобы предотвратить перегруженность и запутанность;
- навигация: должна быть легкая и интуитивная, пользователь должен иметь возможность быстро перемещаться по разделам;
- отзывчивость: приложение должно оперативно реагировать на пользовательский ввод и операции, нужно избегать задержек, чтобы предотвратить ощущение «зависания» приложения;
- цвета и типографика: нужно обеспечить хорошую контрастность текста и фоновых элементов для удобства чтения;
- адаптивность: нужно учесть различные размеры экранов и ориентации устройств, интерфейс должен адаптироваться к разным размерам экранов и обеспечивать хорошую читабельность и удобство использования независимо от устройства.

На рисунке 17 показана диаграмма иерархии окон.

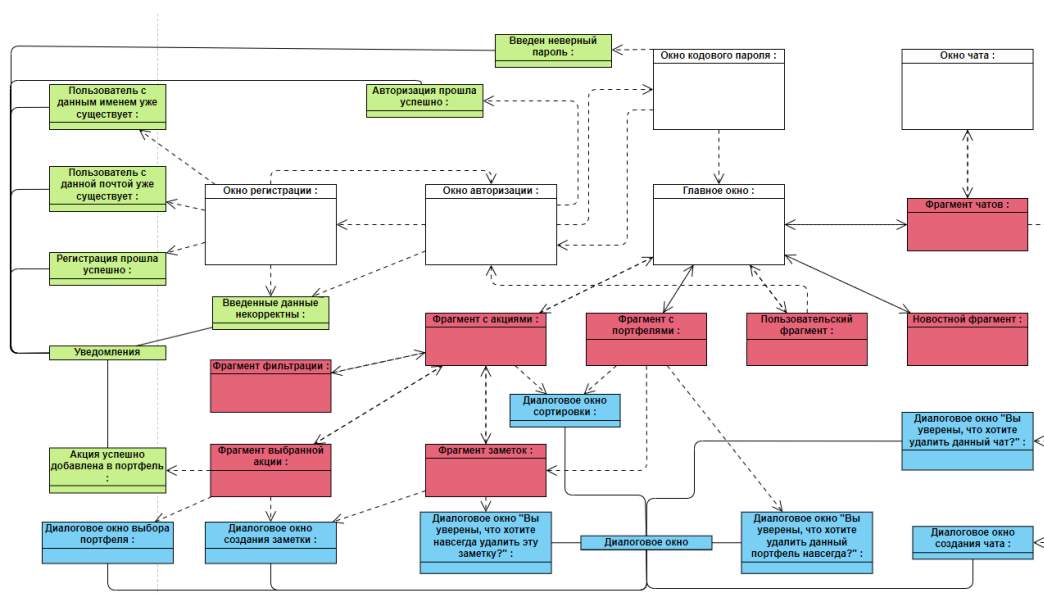


Рисунок 17 – Диаграмма иерархии окон приложения

Зеленым цветом на диаграмме иерархии окон показаны все уведомления, которые есть в приложении, красным фрагменты, синим диалоговые окна, а белым активности.

Основная работа осуществляется в главном окне. Отсюда пользователь может беспрепятственно переходить во фрагменты с помощью нижнего навигационного меню. Все действия сопровождаются уведомлениями. Такие действия, как удаление, не могут проводиться без подтверждения. В окнах авторизации и регистрации пользователь проходит аутентификацию, без которой доступ к главному окну закрыт. После успешной аутентификации пользователь должен придумать код пароль в окне кодового пароля. Это окно будет открываться после успешной авторизации при каждом запуске приложения.

Таким образом, было выполнено проектирование пользовательского интерфейса информационной системы для инвесторов на базе микросервисов в виде андроид приложения. Были выявлены требования к интерфейсу Android приложения и составлена диаграмма иерархии окон, отражающая рабочую область пользователя.

3 Программная реализация

3.1 Реализация базы данных

Для реализации базы данных информационной системы для инвесторов в виде мобильного приложения на микросервисах использовалась документоориентированная СУБД – Mongo DB. Была создана база данных «Stockfolio», содержащая следующие коллекции:

- «portfolios»: коллекция, содержащая информацию о портфелях пользователя;
- «assets»: коллекция, хранящая лого компаний;
- «chats»: коллекция, хранящая пользовательские чаты;
- «news»: коллекция, содержащая сохраненные пользователями новости;
- «notes»: коллекция, содержащая созданные пользователями заметки;
- «users»: коллекция, хранящая информацию о пользователе;
- «users_assets»: коллекция, хранящая добавленные пользователем акции.

Пример коллекции представлен на рисунке 18.

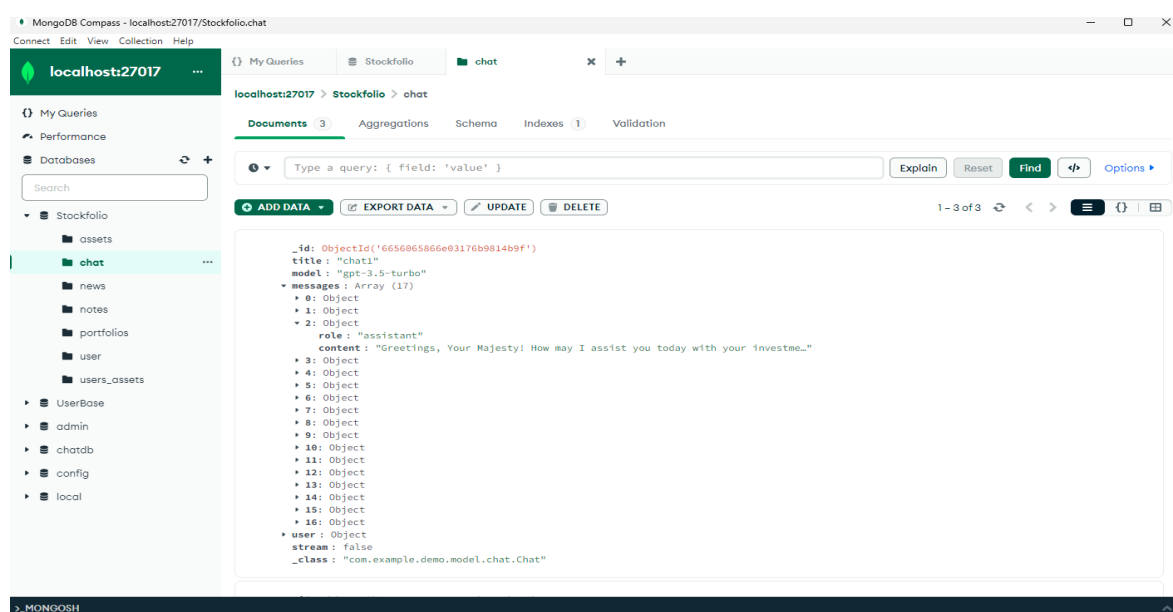


Рисунок 18 – Вид коллекции «chats»

Таким образом, была реализована база данных для хранения информации о пользователе.

3.2 Реализация серверной части

Серверная часть информационной системы реализована на Java с использованием фреймворков Spring Boot и Spring Boot Security. Она состоит из четырех микросервисов, а именно: «User Service», «Stock Service», «Chat Service» и «News Service».

Описание REST API интерфейса микросервиса «User Service» представлено в таблице 9.

Таблица 9 – Описание REST API интерфейса микросервиса «User Service»

Название функции	Строка адреса	Метод REST API	Описание
authenticate	/auth/signin	post	Реализует использование метода авторизации для аутентификации пользователя в системе
register	/auth/signup	post	Реализует использование метода регистрации для добавления пользователя в систему
refreshAccessToken	/auth/refresh	post	Реализует использования метода обновления access токена
validateToken	/auth/validate-token	get	Реализует использования метода проверки токена на действительность
getUsersChats	/chat/all	get	Реализует использование метода получения всех чатов пользователя
createChat	/chat/add	post	Реализует использование метода создания нового пользовательского чата
getAnswer	/chat/reply	post	Реализует использование метода получения ответа от нейросети
addNews	/saved-news/add	post	Реализует использование метода сохранения новости, выбранной пользователем

Продолжение таблицы 9

Название функции	Строка адреса	Метод REST API	Описание
getAllNews	/saved-news/	get	Реализует использование метода получения всех новостей, сохраненных пользователем
deleteNews	/saved-news/delete	post	Реализует использование метода удаления выбранной новости, сохраненной пользователем
checkNews	/saved-news/check	post	Реализует использование метода проверки наличия новости в базе данных
addPortfolio	/portfolios/add	post	Реализует использование метода создания портфеля
deletePortfolio	/portfolios/delete	post	Реализует использование метода удаления портфеля
getPortfolios	/portfolios/	get	Реализует использование метода получения всех портфелей пользователя
addAssetToPortfolio	/portfolios/add-asset	post	Реализует использование метода добавления акции в портфель
addNoteToAsset	/notes/add-to-asset	post	Реализует использование метода добавления заметки об акции
getNoteToAsset	/notes/get-to-asset	post	Реализует использование метода получения заметок об акции
deleteToAsset	/notes/delete-to-asset	post	Реализует использование метода удаления заметок об акции
changeToAsset	/notes/change-to-asset	post	Реализует использование метода изменения заметок об акции
addNoteToPortfolio	/notes/add-to-portfolio	post	Реализует использование метода добавления заметки о портфеле
getNoteToPortfolio	/notes/get-to-portfolio	post	Реализует использование метода получения заметок об портфеле
deleteToPortfolio	/notes/delete-to-portfolio	post	Реализует использование метода удаления заметок об портфеле
changeToPortfolio	/notes/change-to-portfolio	post	Реализует использование метода изменения заметок об портфеле

Все методы являются асинхронными, то есть не требуют обратного ответа от клиентского приложения.

Описание REST API интерфейса микросервиса «News Service» представлено в таблице 10.

Таблица 10 – Описание REST API интерфейса микросервиса «News Service»

Название функции	Строка адреса	Метод REST API	Описание
getNews	/news	post	Реализует использование метода получения актуальных новостей из новостного API

Метод является асинхронными, то есть не требует обратного ответа от клиентского приложения.

Описание REST API интерфейса микросервиса «Stock Service» представлено в таблице 11.

Таблица 11 – Описание REST API интерфейса микросервиса «Stock Service»

Название функции	Строка адреса	Метод REST API	Описание
getSectors	/sectors	get	Реализует использование метода получения списка секторов биржевого рынка
getMostActive	/stocks/most-active	get	Реализует использование метода получения списка акций с меткой «most-active»
getSearchResults	/stocks/search	post	Реализует использование метода получения результатов поиска
getFilteringResults	/stocks/filter	post	Реализует использование метода получения результатов фильтрации
getStockInfo	/stocks/stock-info	post	Реализует использование метода получения информации об акции
getChartData	/stocks/chart	post	Реализует использование метода получения данных для графика
getStockQuote	/stocks/quote	post	Реализует использование метода получения актуальных биржевых данных об акции

Все методы являются асинхронными, то есть не требуют обратного ответа от клиентского приложения.

Описание REST API интерфейса микросервиса «Chat Service» представлено в таблице 12.

Таблица 12 – Описание REST API интерфейса микросервиса «Chat Service»

Название функции	Строка адреса	Метод REST API	Описание
chat	/chat	post	Реализует использование метода получения ответа от нейросети

Метод является асинхронным, то есть не требует обратного ответа от клиентского приложения.

В листинге 1 представлен код REST API интерфейса микросервиса «Stock Service».

Листинг 1 – Код REST API интерфейса микросервиса «Stock Service»

```
@RequestMapping("/stocks")
@RestController
public class StockController {

    @Autowired
    StockService service;

    @GetMapping("/most-active")
    public ArrayList<Stock> getMostActive() {
        return service.getMostActive();
    }

    @PostMapping("/search")
    public ArrayList<Stock> getSearchResults(@RequestBody Map<String,
String> map) {
        return service.getSearchRes(map.get("string"));
    }

    @PostMapping("/filter")
    public ArrayList<Stock> getFilteringResult(@RequestBody
Map<String, List<FilterItem>> map) {
        return service.getFilterRes(map);
    }

    @PostMapping("/stock-info")
    public StockInfo getStockInfo(@RequestBody Map<String, String>
map) {
        return service.getStockInfo(map.get("symbol"));
    }
}
```

```

        @PostMapping("/chart")
        public ArrayList<ChartData> getChartData(@RequestBody Map<String,
String> map) {
            return service.getChartData(map.get("period"),
map.get("symbol"));
        }

        @PostMapping("/quote")
        public ArrayList<Stock> getStockQuote(@RequestBody Map<String,
String> map) {
            return service.getStockQuotes(map.get("symbol"));
        }
    }
}

```

Для того, чтобы передавать данные между клиентом и сервером необходимо формировать JSON строку с определенными полями. Сервер получает от клиента эти данные и десериализует их для передачи в функции. Для этого были реализованы модели данных. Пример такой модели приведен в листинге 2.

Листинг 2 – Модель данных «StockInfo» для передачи

```

@Data
@AllArgsConstructor
@NoArgsConstructor
@JsonIgnoreProperties(ignoreUnknown = true)
public class StockInfo {
    @JsonInclude(JsonInclude.Include.NON_NULL)
    private String companyInfo;
    private String companyName;
    @JsonInclude(JsonInclude.Include.NON_NULL)
    private String logo;

    @JsonInclude(JsonInclude.Include.NON_NULL)
    private double latestPrice;
    @JsonInclude(JsonInclude.Include.NON_NULL)
    private double open;

    @JsonDeserialize(using = StringDeserializer.class)
    private String EBITDA;
    private long marketcap;
    private long sharesOutstanding;

    private double pegRatio;
    private double forwardPERatio;
    private double priceToBook;
    private double priceToSales;

    private double revenuePerShare;
    private double debtToEquity;
    private double profitMargin;

    private double dividendYield;
}

```

```

private String exDividendDate;
private String nextDividendDate;
private double ttmDividendRate;

private double week52high;
private double week52low;
private double beta;
}

```

Полный исходный код серверной части представлен в приложении Б.

Таким образом, была реализована серверная часть информационной системы для инвесторов в виде андроид приложения на микросервисах.

3.3 Реализация клиентской части

Пользовательский интерфейс был разработан в Android Studio. Это интегрированная среда разработки для приложений под операционную систему Android. В качестве цветовой схемы использовалась составная, где основными цветами выступали серый, белый и черный, дополнительными темно-зеленый, красный.

На рисунке 19 показан внешний вид активности «SignInActivity».



Рисунок 19 – Внешний вид окна «SignInActivity»

На данном окне расположены следующие элементы:

- 1 – поля ввода;
- 2 – кнопки;
- 3 – текстовые метки.

На рисунке 20 показан внешний вид активности «SignUpActivity».

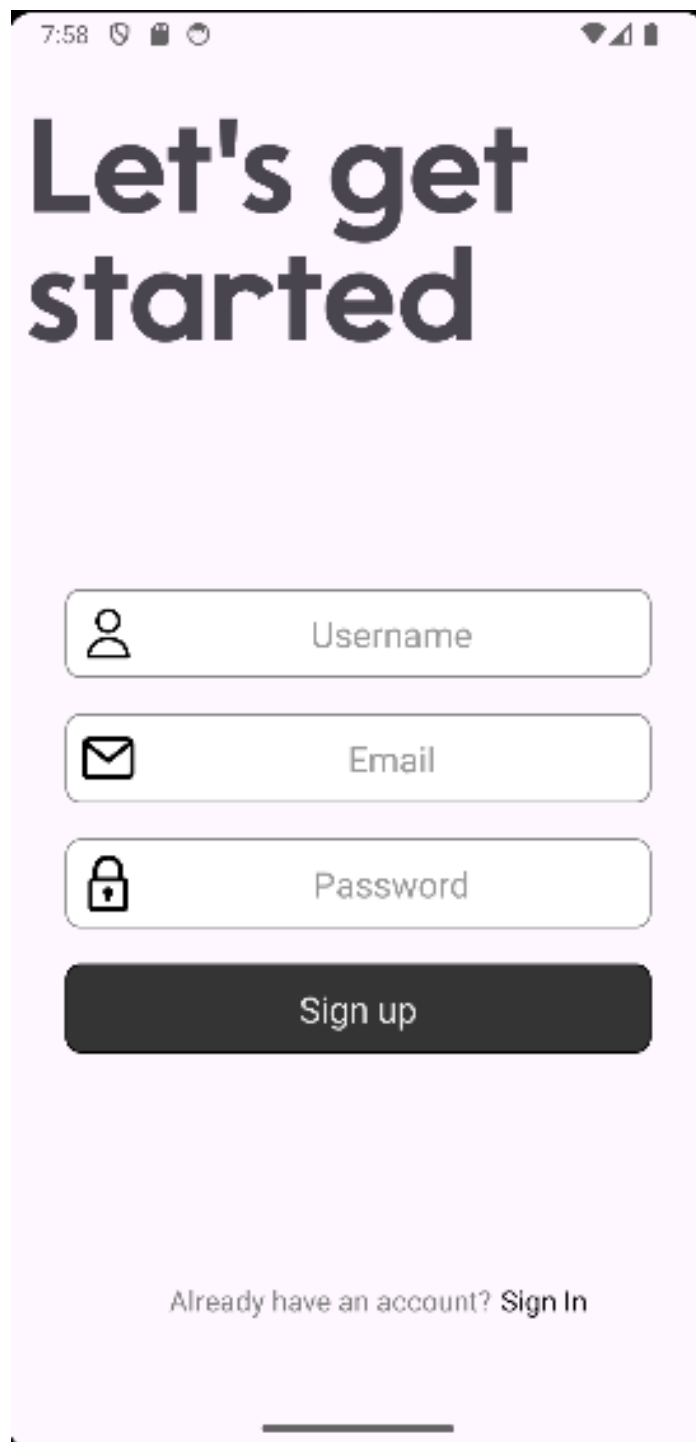


Рисунок 20 – Внешний вид окна «SignUpActivity»

Элементы данного окна аналогичны предыдущему.

На рисунке 21 показан внешний вид активности «PassCodeActivity».

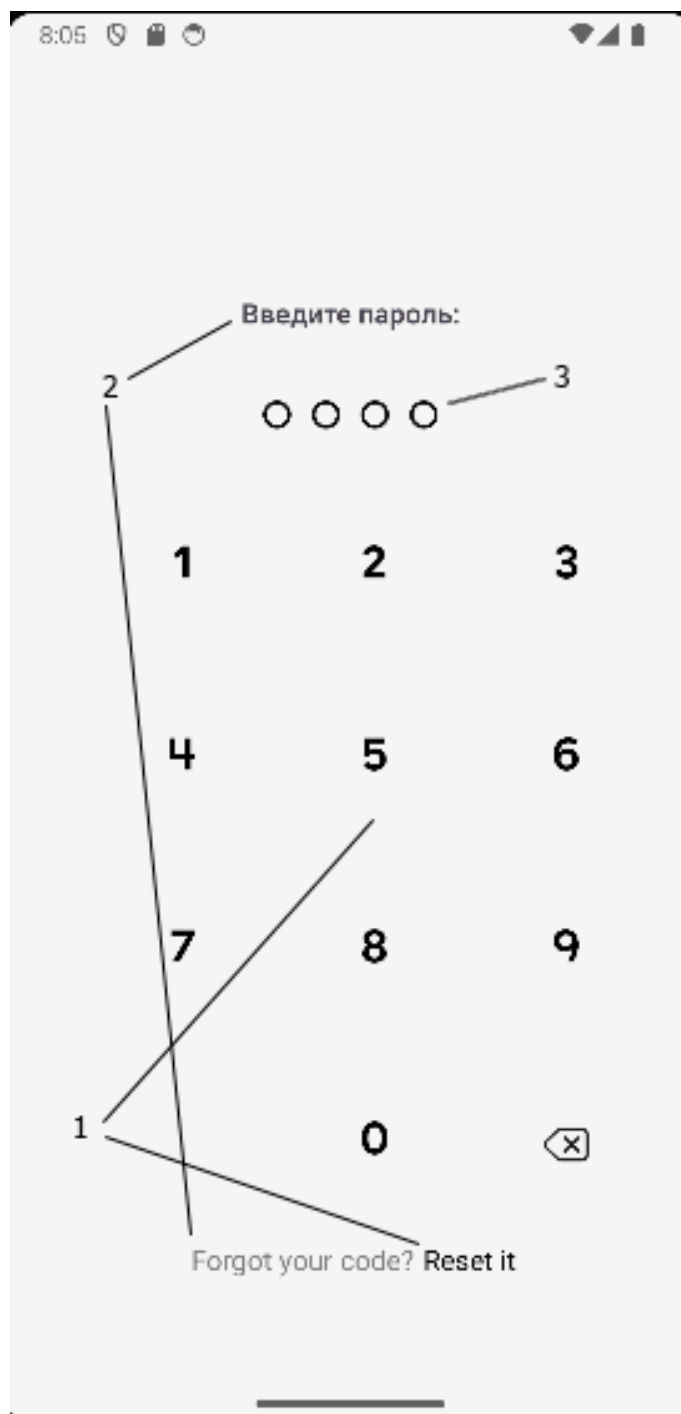


Рисунок 21 – Внешний вид окна «PassCodeActivity»

На данном окне расположены следующие элементы:

- 1 – кнопки;
- 2 – текстовые метки;
- 3 – индикаторы заполнения.

На рисунке 22 показан внешний вид активности «ChatActivity».

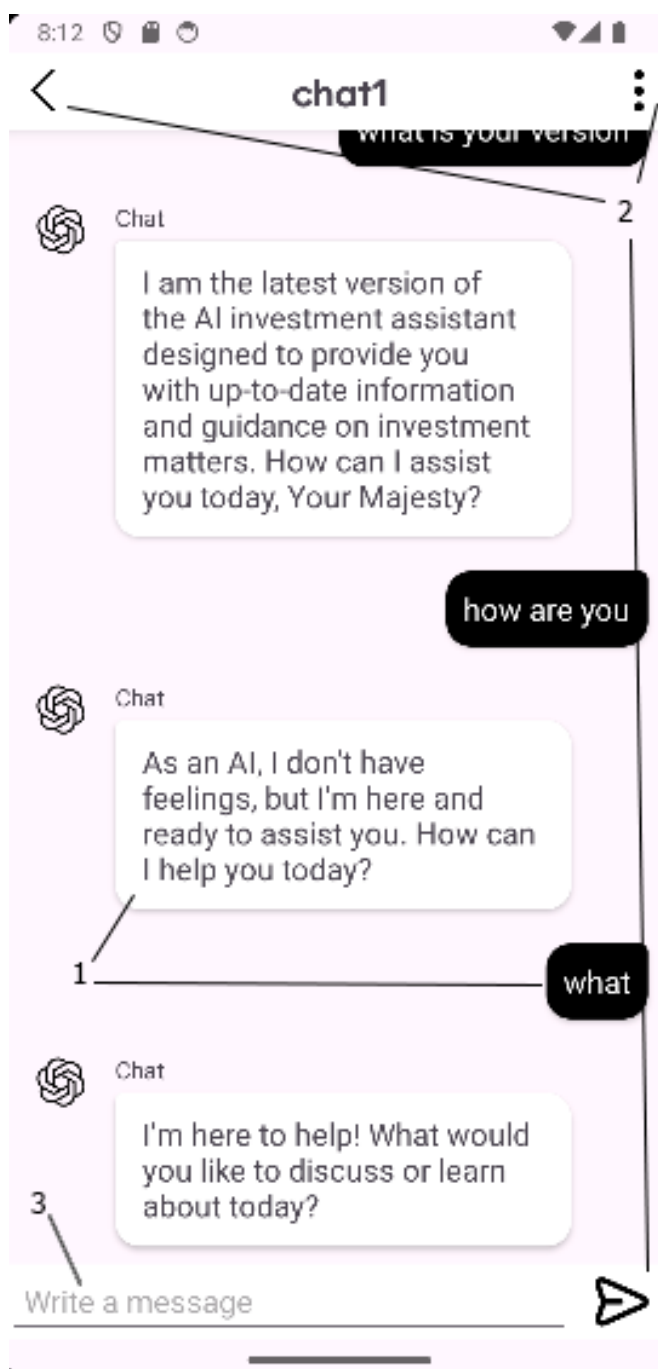


Рисунок 22 – Внешний вид окна «ChatActivity»

На данном окне расположены следующие элементы:

- 1 – макеты сообщений: сообщения пользователя состоят из оболочки и текстовых меток, ответные сообщения состоят из картинки и двух текстовых меток;
- 2 – кнопки;
- 3 – поле текстового ввода.

На рисунке 23 показан внешний вид активности «MainActivity».



Рисунок 23 – Внешний вид окна «MainActivity»

На данном окне расположены следующие элементы:

- 1 – фрагменты: данное оконное пространство предназначено для заменяющих друг друга фрагментов, содержащих информацию, соответствующую области приложения;
- 2 – нижняя навигационная панель, содержащая кнопки в виде иконок для переключения между основными фрагментами.

На рисунке 24 показан фрагмент «AssetsFragment», содержащий списки всех акций.

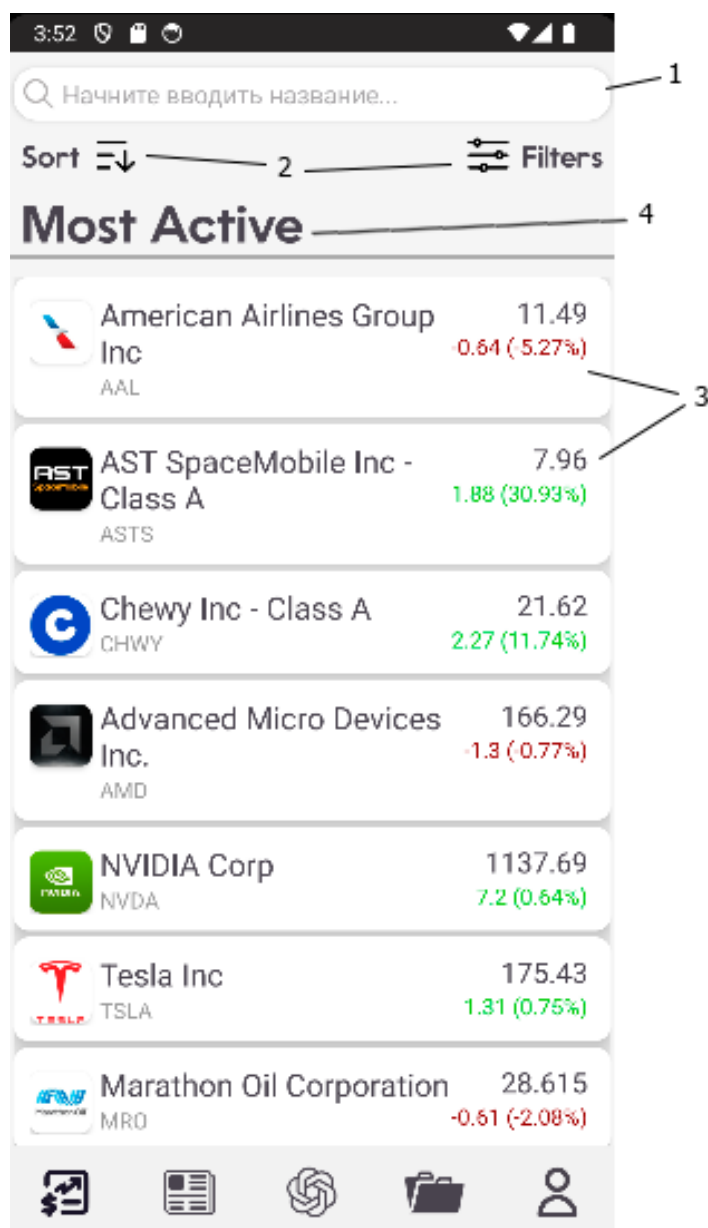


Рисунок 24 – Внешний вид фрагмента «AssetsFragment»

На данном фрагменте расположены следующие элементы:

- 1 – поле текстового ввода;
- 2 – кнопки;
- 3 – макеты акций, содержащие иконки и текстовые метки;
- 4 – текстовые метки.

На рисунке 25 представлен фрагмент «FiltersFragment», содержащий списки фильтров.

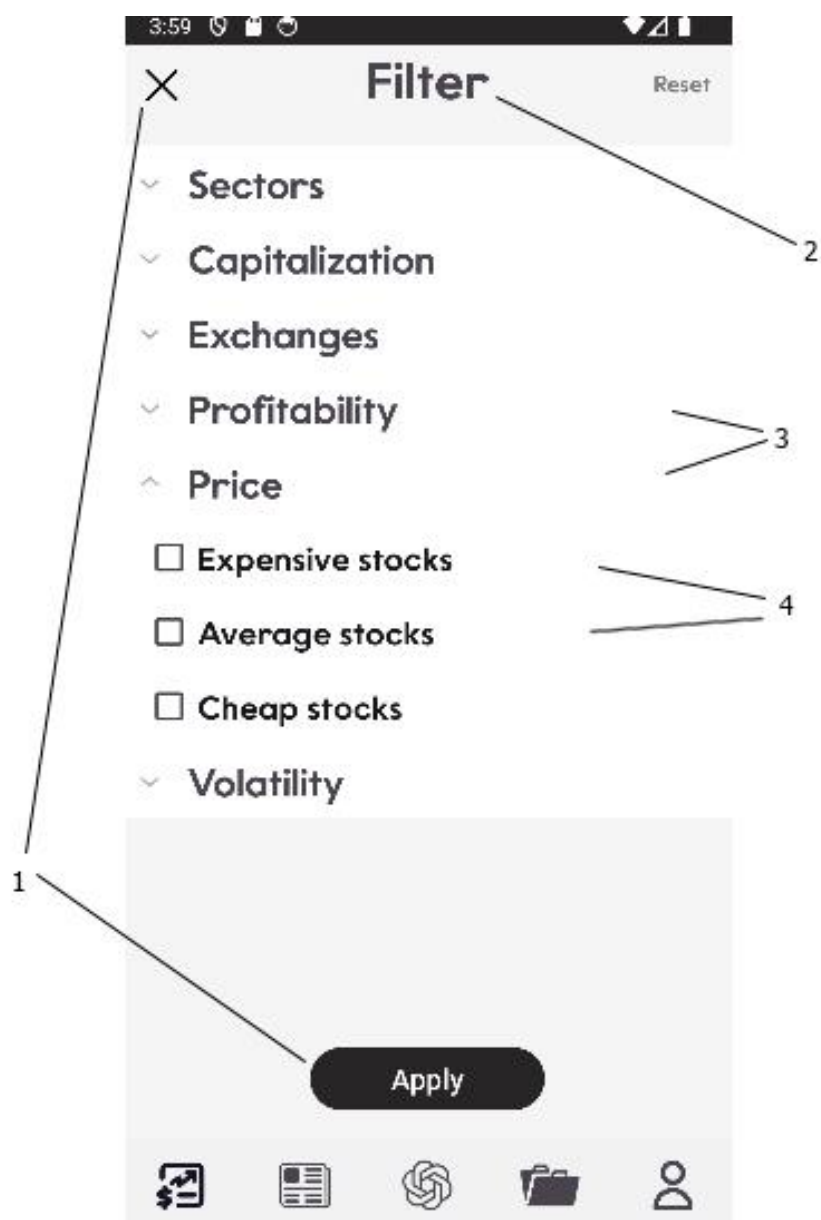


Рисунок 25 – Внешний вид фрагмента «FiltersFragment»

На данном фрагменте расположены следующие элементы:

- 1 – кнопки;
- 2 – текстовые метки;
- 3 – элементы раскрывающегося списка элементов в виде текстовых меток;
- 4 – дочерние элементы раскрывающегося списка элементов в виде чек-боксов.

На рисунке 26 представлен фрагмент «NewsFragment», содержащий новости.



Рисунок 26 – Внешний вид фрагмента «NewsFragment»

На данном фрагменте расположены следующие элементы:

- 1 – текстовые метки;
- 2 – кнопки;
- 3 – макеты новостей, содержащие картинки и текстовые метки.

На рисунке 27 представлен фрагмент «PortfoliosFragment», содержащий все портфели пользователя.

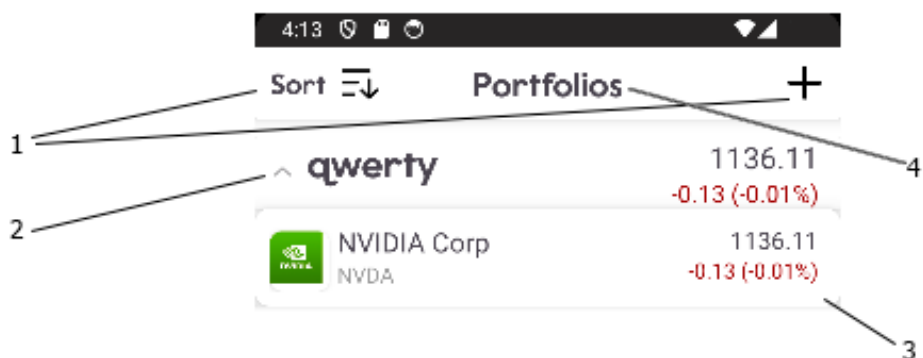


Рисунок 27 – Внешний вид фрагмента «PortfoliosFragment»

На данном фрагменте расположены следующие элементы:

- 1 – кнопки;
- 2 – элемент списка раскрывающихся элементов;
- 3 – дочерний элемент списка раскрывающихся элементов, состоящий из текстовых меток;
- 4 – текстовые метки.

На рисунке 28 показан фрагмент «ChatFragment», содержащий пользовательские чаты.

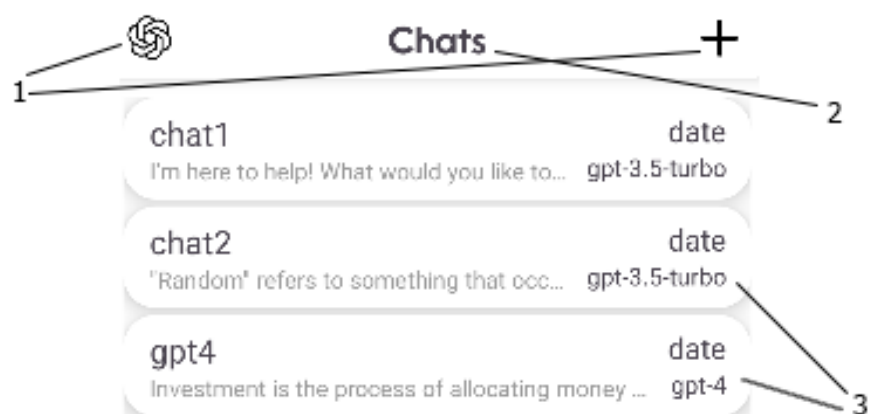


Рисунок 28 – Внешний вид фрагмента «ChatFragment»

На данном фрагменте расположены следующие элементы:

- 1 – кнопки;
- 2 – текстовые метки;
- 3 – элементы списка чатов, содержащие текстовые метки.

На рисунке 29 представлен фрагмент «UserFragment», содержащий информацию о пользователе.

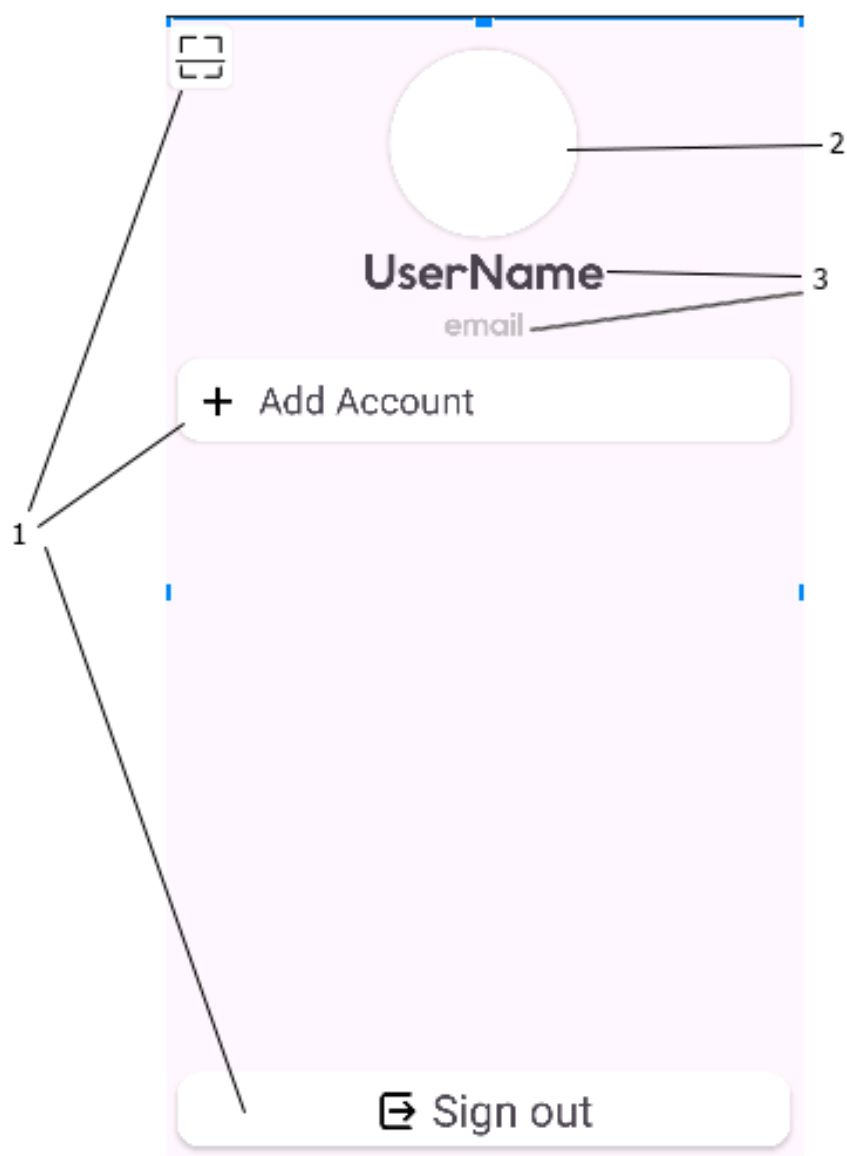


Рисунок 29 – Внешний вид фрагмента «UserFragment»

На данном фрагменте расположены следующие элементы:

- 1 – кнопки;
- 2 – картинка;
- 3 – текстовые метки.

На рисунке 30 представлен фрагмент «AssetPageFragment», содержащий всю информацию об акции.

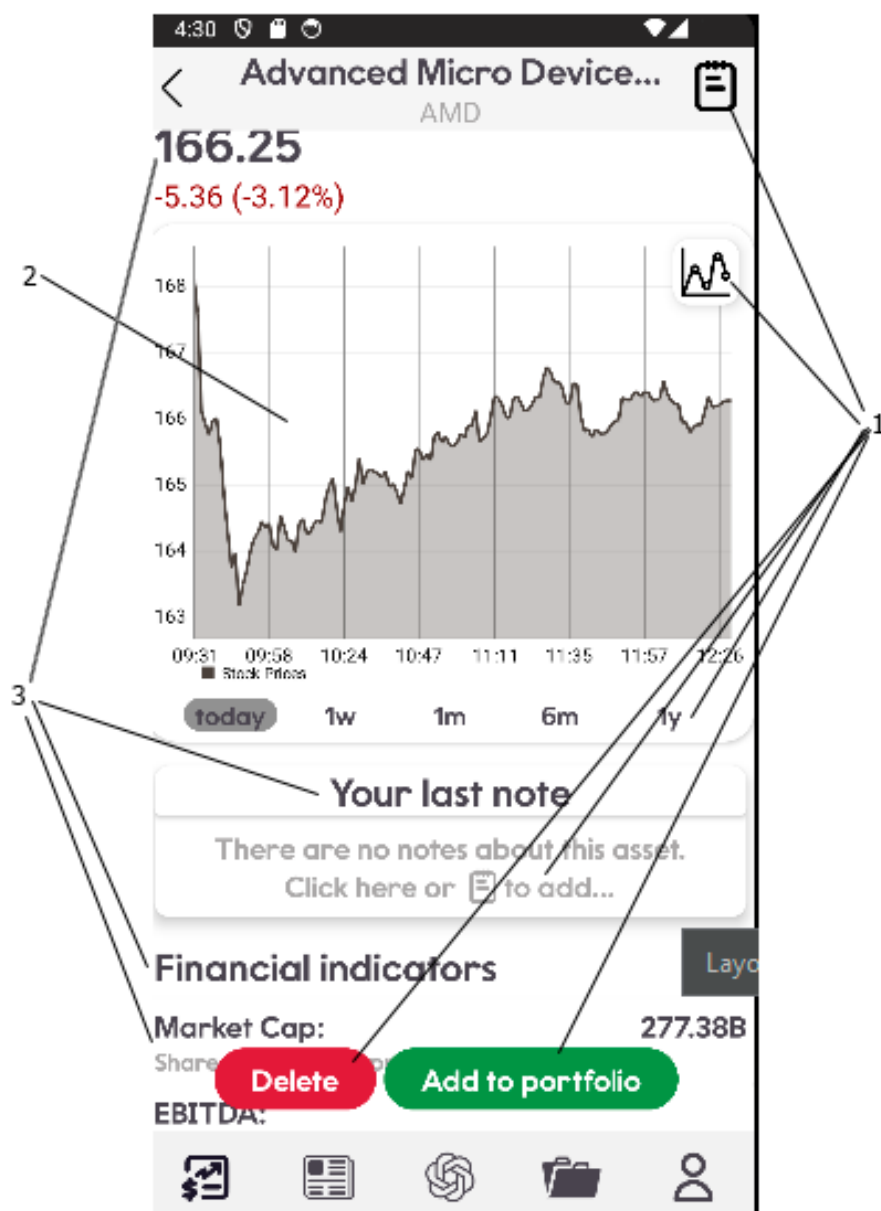


Рисунок 30 – Внешний вид фрагмента «AssetPageFragment»

На данном фрагменте расположены следующие элементы:

- 1 – кнопки;
- 2 – график;
- 3 – текстовые метки.

На рисунке 31 представлен фрагмент «NotePageFragment», содержащий заметки пользователя об определенном объекте.

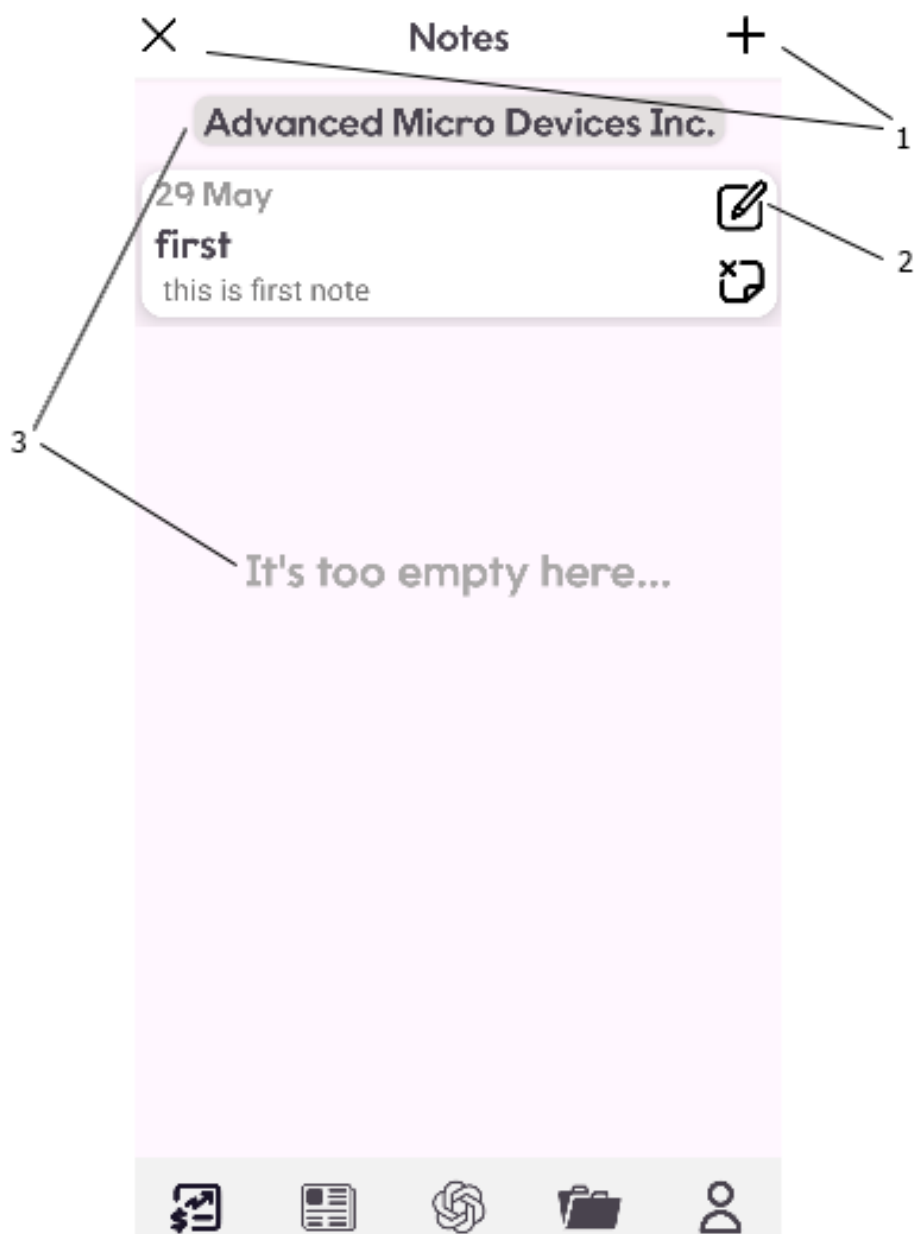


Рисунок 31 – Внешний вид фрагмента «NotePageFragment»

На данном фрагменте расположены следующие элементы:

- 1 – кнопки;
- 2 – элемент списка заметок, содержащий текстовые метки и кнопки удаления и изменения;
- 3 – текстовые метки.

Программное обеспечение клиентской части разрабатывалось на языке программирования Java в Android Studio. Разметка страниц создавалась при помощи xml.

Код главной активности представлен в листинге 3.

Листинг 3 – Код главной страницы «MainActivity»

```
public class MainActivity extends AppCompatActivity {

    ActivityMainBinding binding;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding = ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());
        replaceFragment(new AssetsFragment());

        binding.bottomNavigationView.setOnItemSelectedListener(item ->
        {

            if (item.getItemId() == R.id.assets_part) {
                replaceFragment(new AssetsFragment());
            } else if (item.getItemId() == R.id.news_part) {
                replaceFragment(new NewsFragment());
            } else if (item.getItemId() == R.id.chat_part) {
                replaceFragment(new ChatFragment());
            } else if (item.getItemId() == R.id.portfolios_part) {
                replaceFragment(new PortfolioFragment());
            } else if (item.getItemId() == R.id.user_part) {
                replaceFragment(new UserFragment());
            }

            return true;
        });
    }

    private void replaceFragment(Fragment fragment){

        FragmentManager fragmentManager = getSupportFragmentManager();
        FragmentTransaction fragmentTransaction =
fragmentManager.beginTransaction();

        fragmentTransaction.replace(R.id.frame_layout, fragment);
        fragmentTransaction.commit();
    }
}
```

Полный исходный код клиентской части расположен в приложении А.

Таким образом, была разработана клиентская часть информационной системы в виде мобильного приложения на микросервисах.

4 Тестирование информационной системы для инвесторов

Тестирование приложения – важная часть разработки программного обеспечения. Оно позволяет выявлять ошибки и проверять корректность работы приложения. Существует несколько моделей тестирования:

- функциональное тестирование: проверяет функции приложения на их готовность и корректность работы при различных сценариях использования;
- нефункциональное тестирование: проверяет качественные характеристики компонента или системы, эти характеристики не относятся к конкретной функции или действию пользователя, при этом их можно измерить различными величинами.

Нефункциональное тестирование может включать в себя следующие модели:

- юнит-тестирование;
- тестирование с помощью интеграционных тестов;
- нагрузочное тестирование;
- тестирование пользовательского интерфейса;
- конфигурационное тестирование;
- автоматизированное тестирование.

В качестве моделей тестирования для информационной системы для инвесторов в виде мобильного приложения на микросервисах были выбраны юнит-тестирование и тестирование белого ящика [15].

Метрики кода – количественные показатели, используемые для оценки качества кода, процесса разработки и тестирования. Они нужны для:

- анализа сложности кода, что помогает выявить потенциально проблемные места;
- измерения производительности команды разработчиков и выявления возможных проблем в процессе разработки.

При выборе модели тестирования для информационной системы для инвесторов в виде мобильного приложения на микросервисах также важно учитывать сбор и анализ метрик кода для непрерывного повышения качества приложения и оптимизации процесса разработки [16].

4.1 Обоснование выбора моделей тестирования и метрик программного кода

4.1.1 Обоснование выбора моделей тестирования

Для тестирования информационной системы для инвесторов в виде мобильного приложения на микросервисах были выбраны юнит-тестирование и тестирование белого ящика.

Unit-тестирование (модульное тестирование) – это разновидность тестирования в программной разработке, которая заключается в проверке работоспособности отдельных функциональных модулей, процессов или частей кода приложения.

Unit-тестирование позволяет избежать ошибок или быстро исправить их при обновлении или дополнении программного обеспечения новыми компонентами, не тратя время на проверку программного обеспечения целиком [17].

Преимущества юнит-тестирования:

- быстрая обратная связь: позволяет быстро обнаруживать и исправлять ошибки;
- изоляция ошибок: тестирование отдельных компонентов позволяет точно определить местоположение и причину ошибок;
- способствует раннему обнаружению проблем: помогает предотвратить распространение ошибок на более высокие уровни интеграции.

Юнит тестирование особенно подходит для систем на базе микросервисов, так как:

- позволяет тестировать каждый сервис независимо от других, что соответствует идеологии микросервисной архитектуры;
- обеспечивает стабильность и надежность всей системы, даже при изменениях в одном из микросервисов;
- упрощает поиск и устранение дефектов в отдельных компонентах без влияния на работу других сервисов.

Тестирование белым ящиком (white box testing) – это метод тестирования программного обеспечения, при котором тестирующий имеет доступ к внутренней структуре и коду программы.

Преимущества тестирования белым ящиком:

- более полное покрытие: позволяет тестировать все возможные пути выполнения кода;
- выявление скрытых дефектов: обнаруживает ошибки, связанные с логикой программы или её структурой;
- оптимизация кода: помогает выявить участки кода, требующие оптимизации или улучшения [18].

Таким образом, выбранные модели тестирования обеспечили проверку работы информационной системы для инвесторов в виде мобильного приложения на микросервисах.

4.1.2 Обоснование выбора метрик программного кода

Применение метрик позволяет изучить сложность разрабатываемого проекта, оценить объем работ, стилистику разрабатываемой программы и усилия, потраченные каждым разработчиком для реализации того или иного решения. Существуют количественные метрики, метрики сложности потока программы, метрики сложности потока управления данными.

SLOC – количество строк кода, является самой элементарной метрикой. Данная метрика была изначально разработана для оценки трудозатрат по проекту. Различают физические и логические строки кода. Кроме SLOC к количественным характеристикам относят также: количество пустых строк,

количество комментариев, процент комментариев, среднее число строк для функций, среднее число строк, среднее число строк для модулей.

Метрики Холстеда – группа метрик, основанных на подсчете некоторых единиц в коде программы. При использовании этих метрик частично компенсируются недостатки, связанные с возможностью записи одной и той же функциональности разным количеством строк и операторов [19].

Основными метриками были выбраны метрика SLOC и метрика Холстеда так как они позволяют в целом оценить объем и сложность программного кода.

4.2 Тестирование программных средств

4.2.1 Тестирование белым ящиком

В рамках разработки информационной системы было проведено тестирование белым ящиком алгоритма аутентификации по следующим критериям покрытия кода:

- покрытие операторов;
- покрытие решений;
- покрытие маршрутов.

Блок-схема – это схематичное представление процесса, системы или компьютерного алгоритма.

Блок-схемы часто применяются в разных сферах деятельности, чтобы:

- документировать;
- изучать;
- планировать;
- совершенствовать и объяснять сложные процессы с помощью

простых логических диаграмм.

Блок-схема алгоритма аутентификации представлена на рисунке 32.

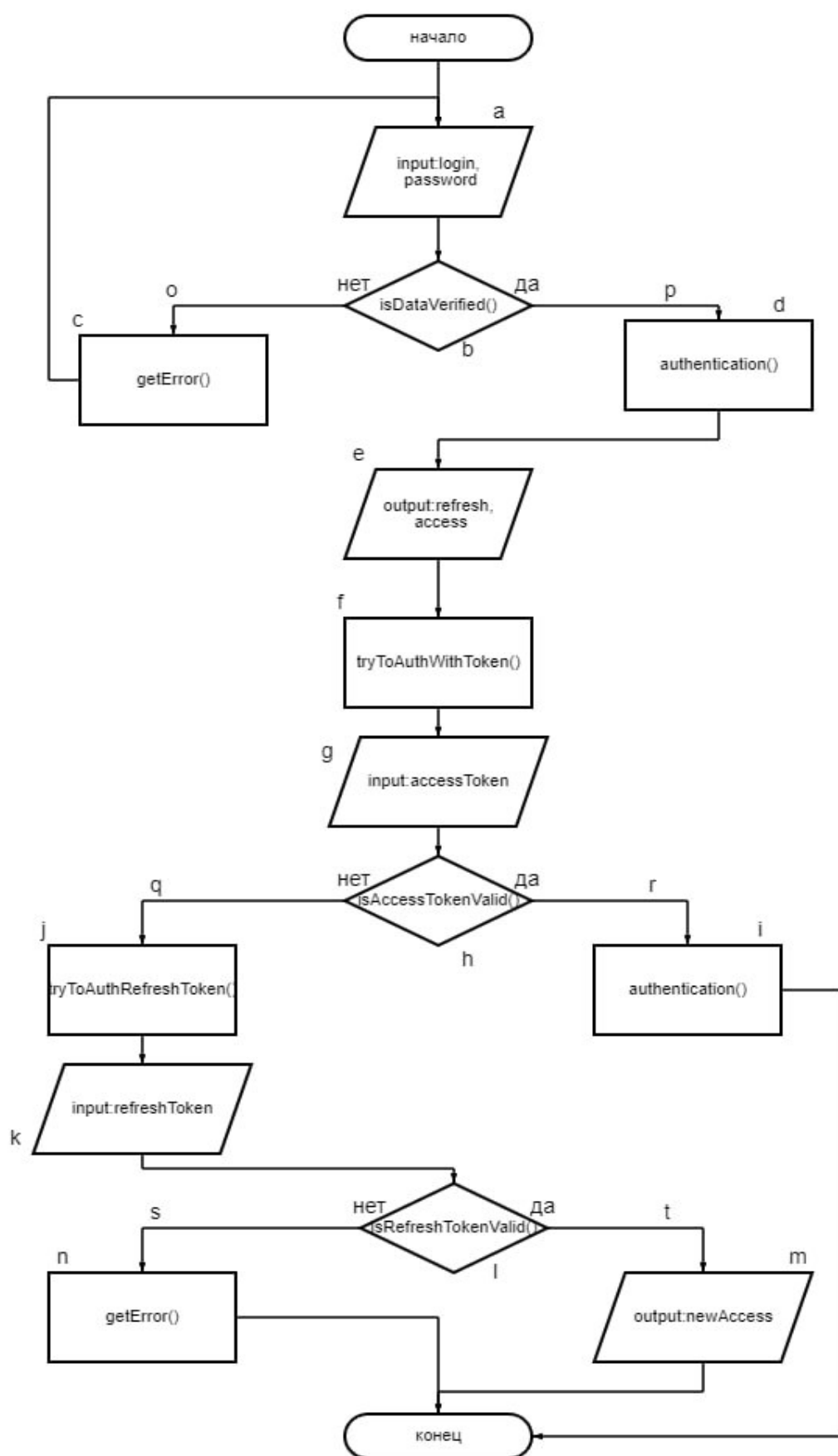


Рисунок 32 – Блок-схема алгоритма аутентификации

Покрывание операторов – один из критериев покрытия кода, который показывает, каждая ли строка исходного кода была выполнена и протестирована. Он подразумевает, что каждый оператор программы должен быть выполнен хотя бы один раз.

Чтобы выполнить каждый оператор не менее одного раза, был составлен единственный тест-кейс со значениями входных данных, указанными в таблице 13.

Таблица 13 – Тест-кейс покрытия операторов

№	Операторы	Входные данные	Выходные данные
1	с, d, f, j, i, n	Корректные login и password	Токен, содержащий refresh и access
		Некорректные login и password	401 UNAUTHORIZED «nvalid user data»
		Корректный access токен	Authenticated user
		Некорректный refresh токен	401 UNAUTHORIZED «invalid refresh token»

Таким образом, тестирование критерия покрытия операторов привело к положительным результатам.

Покрывание решений – один из критериев покрытия кода, который показывает, каждая ли из возможных ветвей из каждой точки решения в исходном коде была выполнена и протестирована.

В ходе тестирования этого критерия необходимо было написать такое число тестов, которого было бы достаточно для того, чтобы получить для каждого условия в программе как истинное, так и ложное значение.

Чтобы выполнить каждую ветвь из каждой точки решения не менее одного раза, было составлено четыре тест-кейса со значениями входных данных, указанных в таблице 14.

Таблица 14 – Тест-кейсы покрытия решений

№	Решения	Входные данные	Выходные данные
1	р(да), г(да)	Корректные login и password Корректный access токен	Authenticated user

Продолжение таблицы 14

№	Решения	Входные данные	Выходные данные
2	p(да), q(нет), t(да)	Корректный login и password Некорректный access токен Корректный refresh токен	newAccess
3	o(нет), p(да), t(да)	Некорректный login и password Корректный login и password Корректный access токен	401 UNAUTHORIZED «invalid user data» Authenticated user
4	p(да), q(нет), s(нет)	Корректный login и password Некорректный access токен Некорректный refresh токен	401 UNAUTHORIZED «invalid refresh token»

Таким образом, тестирование критерия покрытия решений привело к положительным результатам.

Покрывание маршрутов – один из критериев покрытия кода, который показывает, каждая ли из возможных ветвей из каждой точки решения и каждый ли оператор в исходном коде были выполнены и протестированы.

В ходе тестирования этого критерия необходимо было написать такое число тестов, которого было бы достаточно для того, чтобы получить для каждого условия в программе как истинное, так и ложное значение, и каждый оператор был бы выполнен как минимум один раз.

Чтобы выполнить каждую ветвь из каждой точки решения и каждый оператор не менее одного раза, было составлено три тест-кейса со значениями входных данных, указанными в таблице 15.

Таблица 15 – Тест-кейсы покрытия маршрутов

№	Условие	Входные данные	Выходные данные
1	b, p(да), h, r(да)	Корректные login, password Корректный accessToken	Authenticated user «valid access token»
2	b, p(да), h, q(нет), l, t(да)	Корректные login, password Некорректный accessToken Корректный refreshToken	Authenticated user 401 UNAUTHORIZED «invalid access token» newAccess

Продолжение таблицы 15

№	Условие	Входные данные	Выходные данные
3	b, p(да), h, q(нет), l, p(нет)	Корректные login, password Некорректный accessToken Некорректный refreshToken	Authenticated user 401 UNAUTHORIZED «invalid access token» 401 UNAUTHORIZED «invalid refresh token»
4	b, o(нет), b, p(да), h, r(да)	Некорректные login, password Корректные login, password Корректный accessToken	401 UNAUTHORIZED «invalid user data» Authenticated user «valid access token»
5	b, o(нет), b, p(да), h, q(нет), l, t(да)	Некорректные login, password Корректные login, password Некорректный accessToken Корректный refreshToken	401 UNAUTHORIZED «invalid user data» Authenticated user 401 UNAUTHORIZED «invalid access token» newAccess
6	b, o(нет), b, p(да), h, q(нет), l, p(нет)	Некорректные login, password Корректные login, password Некорректный accessToken Некорректный refreshToken	401 UNAUTHORIZED «invalid user data» Authenticated user 401 UNAUTHORIZED «invalid access token» 401 UNAUTHORIZED «invalid refresh token»

Таким образом, тестирование критерия покрытия маршрутов привело к положительным результатам.

4.2.2 Модульное тестирование

Для модульного тестирования были написаны unit-тесты, которые проверяют работу каждого метода класса. При написании тестов использовался фреймворк JUnit, который предоставляет инструменты для создания тестов и проверки результатов.

Модульное тестирование проведено на примере контроллера управления портфелями пользователя. Код unit-тестов для контроллера аутентификации пользователя представлена на листинге 4.

Листинг 4 – Код unit-тестов для контроллера аутентификации пользователя

```
class AuthenticationControllerTest {
```

```

@Mock
private JwtService jwtService;

@Mock
private RefreshJwtService refreshJwtService;

@Mock
private AccessJwtService accessJwtService;

@Mock
private AuthenticationService authenticationService;

@Mock
private UserRepository userRepository;

@InjectMocks
private AuthenticationController authenticationController;

@BeforeEach
void setUp() {
    MockitoAnnotations.openMocks(this); // Initialize mocks

    when(jwtService.getRefresh()).thenReturn(refreshJwtService);
    when(jwtService.getAccess()).thenReturn(accessJwtService);

    authenticationController = new
AuthenticationController(jwtService, authenticationService);
    authenticationController.setUserRepository(userRepository); //
Set the mocked userRepository
}

@Test
void testRegister_WhenUserDoesNotExist_ReturnSuccessMessage() {
    RegisterUserDto registerUserDto = new
RegisterUserDto("username", "password", "email");

    when(userRepository.existsUserByUsername(registerUserDto.getUsername())
    ).thenReturn(false);

    when(userRepository.existsUserByEmail(registerUserDto.getEmail())) .the
nReturn(false);

    when(authenticationService.signup(any(RegisterUserDto.class))).thenRet
urn(new User());

    ResponseEntity<String> responseEntity =
authenticationController.register(registerUserDto);

    assertEquals(HttpStatus.OK, responseEntity.getStatusCode());
    assertEquals("success", responseEntity.getBody());
}

@Test
void testAuthenticate_WhenUserAuthenticated_ReturnJwtToken() {
    LoginUserDto loginUserDto = new LoginUserDto("username",
"password");
    User user = new User();

```

```

when(authenticationService.authenticate(any(LoginUserDto.class))).thenReturn(user);
    when(jwtService.generateToken(user)).thenReturn("mocked-jwt-token");

    ResponseEntity<LoginResponse> responseEntity =
authenticationController.authenticate(loginUserDto);

    assertEquals(HttpStatus.OK, responseEntity.getStatusCode());
    assertEquals("mocked-jwt-token",
responseEntity.getBody().getToken());
}

@Test
void
testRefreshAccessToken_WhenValidRefreshToken_ReturnNewAccessToken() {
    Map<String, String> request = new HashMap<>();
    request.put("refresh_token", "valid-refresh-token");

    User user = new User();

    when(userRepository.findByUsername(any(String.class))).thenReturn(Optional.of(user));
    when(jwtService.getRefresh().isTokenValid("valid-refresh-token")).thenReturn(true);
    when(jwtService.getRefresh().extractUsername("valid-refresh-token")).thenReturn("username");

    when(jwtService.getAccess().generateToken(user)).thenReturn("new-access-token");

    ResponseEntity<?> responseEntity =
authenticationController.refreshAccessToken(request);

    assertEquals(HttpStatus.OK, responseEntity.getStatusCode());
    assertEquals("new-access-token", ((LoginResponse)
responseEntity.getBody()).getToken());
}

@Test
void testValidateToken_WhenTokenIsValid_ReturnOk() {
    String token = "Bearer valid-access-token";
    User user = new User();

    when(userRepository.findByUsername(any(String.class))).thenReturn(Optional.of(user));

    when(jwtService.getAccess().extractUsername(any(String.class))).thenReturn("username");
    when(jwtService.getAccess().isTokenValid("valid-access-token", user)).thenReturn(true);

```

```

        ResponseEntity<?> responseEntity =
authenticationController.validateToken(token);

        assertEquals(HttpStatus.OK, responseEntity.getStatusCode());
        assertEquals("Token is valid", responseEntity.getBody());
    }
}

```

Все ключевые классы – контроллеры и сервисы – прошли модульное тестирование успешно. Результаты тестирования класса контроллера аутентификации пользователя представлены на рисунке 33.

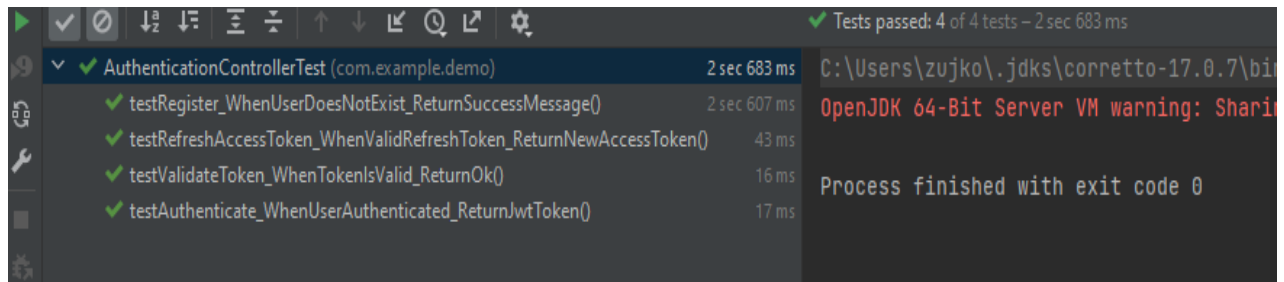


Рисунок 33 – Результат модульного тестирования контроллера аутентификации пользователя

Результаты тестирования остальных модулей представлены в приложении В. Исходный код модульных тестов приведен в приложении Г.

4.3 Оценка качества программного кода на основе метрик

4.3.1 Метрика SLOC

Метрика SLOC представляет собой метрику количества строк кода. Подсчет данной метрики был проведен для модуля, отвечающего за аутентификацию пользователя в системе. Код модуля приведен в листинге 5.

Листинг 5 – Методы register, authenticate, refreshAccessToken, validateToken модуля аутентификации

```

@PostMapping("/signup")
public ResponseEntity<String> register(@RequestBody
RegisterUserDto registerUserDto) {
    if
    (userRepository.existsUserByUsername(registerUserDto.getUsername())) {
        return
        ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Choose different
name");
    }
}

```

```

        if
        (userRepository.existsUserByEmail(registerUserDto.getEmail())) {
            return
            ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Choose different
            email");
        }
        User registeredUser =
        authenticationService.signup(registerUserDto);

        return ResponseEntity.ok("success");
    }

    @PostMapping("/signin")
    public ResponseEntity<LoginResponse> authenticate(@RequestBody
    LoginUserDto loginUserDto) {
        User authenticatedUser =
        authenticationService.authenticate(loginUserDto);

        String jwtToken = jwtService.generateToken(authenticatedUser);

        LoginResponse loginResponse = new LoginResponse();
        loginResponse.setToken(jwtToken);
        loginResponse.setExpiresIn(jwtService.getExpirationTime());

        return ResponseEntity.ok(loginResponse);
    }

    @PostMapping("/refresh")
    public ResponseEntity<?> refreshAccessToken(@RequestBody
    Map<String, String> request, Principal principal) {
        String refreshToken = request.get("refresh_token");
        try {
            Optional<User> user =
            userRepository.findByUsername(jwtService.getRefresh().extractUsername(
            refreshToken));

            if (jwtService.getRefresh().isTokenValid(refreshToken)) {
                String newAccessToken =
                jwtService.getAccess().generateToken(user.get()); // Логика генерации
                нового Access Token
                LoginResponse loginResponse = new LoginResponse();
                loginResponse.setToken(newAccessToken);

                loginResponse.setExpiresIn(jwtService.getAccess().getExpirationTime())
                ;

                return ResponseEntity.ok(loginResponse);
            } else {
                return ResponseEntity.badRequest().body("Invalid
                refresh token");
            }
        } catch (Exception e) {
            return
            ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Invalid refresh
            token");
        }
    }

    @PostMapping("/validate-token")

```

```

    public ResponseEntity<?>
validateToken(@RequestHeader("Authorization") String token) {
    try {
        String tokenValue = token.substring(7); // Получаем чистый
токен
        Optional<User> user =
userRepository.findByUsername(jwtService.getAccess().extractUsername(t
okenValue));

        if (user.isPresent() &&
jwtService.getAccess().isTokenValid(tokenValue, user.get())) {
            return ResponseEntity.ok().body("Token is valid");
        } else {
            return
ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Invalid token or
user not found");
        }
    } catch (NoSuchElementException e) {
        return
ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("User not found");
    } catch (Exception e) {
        return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("An error
occurred while validating token");
    }
}

```

В данной программе имеем:

- 60 физических строк кода;
- 18 логических строк кода;
- 5 строк комментариев.

Таким образом, 18 логических строк кода более полно отражают размер программы, так как исключают пустые строки.

4.3.2 Метрика Холстеда

Для расчёта метрик Холстеда были взяты функции контроллера аутентификации пользователя. Код данного модуля представлен в листинге 2.

Количество используемых операндов и операторов, а также их общее число в программе приведены в таблицах 16 и 17 соответственно.

Таблица 16 – Операторы

Операторы	Число вхождений
=	12
If	4
else	2

Продолжение таблицы 16

Операторы	Число вхождений
return	11
try	2
catch	3
existsUserByUsername()	1
getUsername()	1
existsUserByEmail()	1
getEmail()	1
Status()	6
Body()	8
Signup()	1
Ok()	4
Authenticate()	1
generateToken()	2
setToken()	2
setExpiresIn()	2
getExpirationTime()	2
Get()	3
findByUsername	2
getRefresh()	2
extractUsername()	2
isTokenValid()	2
getAccess()	4
badRequest()	1
Substring()	1
isPresent()	1
n1 = 28	N1 = 84

Таблица 17 – Операнды

Операнды	Число вхождений
registerUserDto	4
BAD_REQUEST	2
«Choose different name»	1
«Choose different email»	1
«success»	1
registeredUser	1
loginUserDto	2
authenticatedUser	1
loginResponse	8
jwtToken	2
jwtService	11
Request	2
«refresh_token»	1
User	8
newAccessToken	2
«Invalid refresh token»	1
«Invalid refresh token»	1
UNAUTHORIZED	3
Token	2
tokenValue	3
«Token is valid»	1

Продолжение таблицы 16

Операторы	Число вхождений
«Invalid token or user not found»	1
«User not found»	1
INTERNAL_SERVER_ERROR	1
«An error occurred while validating token»	1
e	4
userRepository	5
authenticationService	5
n2 = 30	N2 = 76

Длина программы рассчитывается по формуле:

$$N = N1 + N2 = 84 + 76 = 160.$$

Словарь программы рассчитывается по формуле:

$$n = n1 + n2 = 28 + 30 = 58.$$

Объем программы рассчитывается по формуле:

$$H_V = N * \log_2(n) = 160 * 5,857 = 937,17.$$

Сложность программы рассчитывается по формуле:

$$HD = (n1 / 2) * (N2 / n2) = (28 / 2) * (76 / 30) = 35,42.$$

Полученные показатели объема и сложности программы удовлетворяют требованиям по метрикам программного кода [20].

Заключение

В ходе выполнения преддипломной практики был выполнен анализ предметной области и постановка задачи на разработку информационной системы для инвесторов в виде мобильного приложения на микросервисах, а также проектирование этой системы.

В анализе предметной области и постановке задачи выполнено:

- определение целей разработки и назначения программного обеспечения;
- определение целевой аудитории информационной системы;
- сравнительный анализ приложений аналогов;
- постановка задачи на разработку;
- планирование разработки программного обеспечения.

В проектировании программных средств было выполнено:

- моделирование функциональных возможностей приложения;
- разработка архитектуры программного обеспечения;
- прототипирование пользовательского интерфейса;
- обоснование выбора средств разработки.

Таким образом было выполнен анализ и проектирование для разработки информационной системы для инвесторов в виде мобильного приложения на микросервисах.

Список использованных источников

1. Роль мобильных приложений в жизни современного человека. [Электронный ресурс] Ссылка: <https://cyberleninka.ru/article/n/rol-mobilnyh-prilozheniy-v-zhizni-sovremennogo-cheloveka> (дата обращения 13.03.2024).
2. Как найти целевую аудиторию для Вашего мобильного приложения или продукта [Электронный ресурс] Ссылка: <https://allrival.com/post/articles/kak-nayti-celevuyu-auditoriyu-dlya-vashego-mobilnogo-prilozheniya-ili-produkta> (дата обращения 14.05.2024)
3. Мобильные приложения для инвестиций. [Электронный ресурс] Ссылка: <https://quasa.io/ru/media/mobilnye-prilozheniya-dlya-investiciy> (дата обращения 13.03.2024).
4. Use Case Diagrams. [Электронный ресурс] Ссылка: <https://www.geeksforgeeks.org/use-case-diagram/> (дата обращения 13.03.2024).
5. IntelliJ IDEA – а ведущая IDE для разработки на Java и Kotlin. [Электронный ресурс] Ссылка: <https://www.jetbrains.com/ru-ru/idea/> (дата обращения 14.03.2024).
6. Get starter with Android Studio [Электронный ресурс] Ссылка: <https://developer.android.com/studio/intro> (дата обращения 15.03.2024).
7. Диаграмма Ганта: как и где построить, шаблоны и примеры. [Электронный ресурс] Ссылка: <https://getcompass.ru/blog/posts/diagramma-ganta> (дата обращения 20.03.2024).
8. ПО для управления проектами | Microsoft Project [Электронный ресурс] Ссылка: <https://www.microsoft.com/ru-ru/microsoft-365/project/project-management-software> (дата обращения 16.03.2024).
9. MongoDB: The Developer Data Platform | MongoDB [Электронный ресурс] Ссылка: <https://www.mongodb.com/> (дата обращения 17.03.2024).
10. Микросервисная архитектура [Электронный ресурс] Ссылка: https://ru.wikipedia.org/wiki/Микросервисная_архитектура (дата обращения 27.03.2024).

11. Учебное пособие «Диаграмма развертывания». [Электронный ресурс] Ссылка: <https://creately.com/blog/ru/uncategorized-ru/учебное-пособие-по-диаграмме-развёрт/> (дата обращения 7.04.2024)
12. Диаграмма последовательности. [Электронный ресурс] Ссылка: https://ru.wikipedia.org/wiki/Диаграмма_последовательности (дата обращения 7.04.2024)
13. Activity Diagram Tutorial [Электронный ресурс] Ссылка: <https://online.visual-paradigm.com/diagrams/tutorials/activity-diagram-tutorial/> (дата обращения 19.03.2024)
14. Проектирование графического интерфейса пользователя. [Электронный ресурс] Ссылка: <https://habr.com/ru/articles/208966/> (дата обращения 14.05.2024)
15. Что такое тестирование и как его проводить [Электронный ресурс] Ссылка: <https://sky.pro/media/chto-takoe-testirovanie-i-kak-ego-provodit/> (дата обращения 14.05.2024)
16. Программный код и его метрики [Электронный ресурс] Ссылка: <https://habr.com/ru/companies/intel/articles/106082/> (дата обращения 14.05.2024)
17. Java Unit Testing: методики, понятия, практика [Электронный ресурс] Ссылка: <https://javarush.com/groups/posts/2500-vse-o-unit-testing-metodiki-ponjatija-praktika> (дата обращения 14.05.2024)
18. Тестирование белого ящика [Электронный ресурс] Ссылка: https://ru.wikipedia.org/wiki/Тестирование_белого_ящика (дата обращения 14.05.2024)
19. Программный код и его метрики [Электронный ресурс] Ссылка: <https://habr.com/ru/companies/intel/articles/106082/> (дата обращения 14.05.2024)
20. Программные показатели Холстеда [Электронный ресурс] Ссылка: <https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/> (дата обращения 14.05.2024)

Приложение А
(рекомендуемое)
Исходный код клиентской части

Листинг А.1 – Код класса «AssetAdapter»

```

public class AssetAdapter extends ArrayAdapter<Asset> {
    public AssetAdapter(@NonNull Context context, ArrayList<Asset>
assets) {
        super(context, R.layout.asset_item, assets);
    }

    @NonNull
    @Override
    public View getView(int position, @Nullable View view, @NonNull
ViewGroup parent) {
        Asset asset = getItem(position);

        if (view == null){
            view =
LayoutInflater.from(getContext()).inflate(R.layout.asset_item,
parent, false);
        }

        CardView asset_item = view.findViewById(R.id.asset_item);
        ImageView logo = view.findViewById(R.id.company_logo);
        TextView title = view.findViewById(R.id.asset_title);
        TextView ticker = view.findViewById(R.id.asset_ticker);
        TextView price = view.findViewById(R.id.asset_price);
        TextView diff = view.findViewById(R.id.asset_diff);

        //      image.setImageDrawable(news.getImage());
        Picasso.get().load(asset.getLogo()).into(logo);
        title.setText(asset.getCompanyName());
        ticker.setText(asset.getSymbol());
        price.setText(String.valueOf(asset.getLatestPrice()));
        double scale = Math.pow(10, 2);
        double difference = Math.ceil((asset.getLatestPrice() -
asset.getOpen()) * scale) / scale;
        double percents = Math.ceil(((asset.getLatestPrice() -
asset.getOpen()) / asset.getOpen() * 100) * scale) / scale;

        diff.setText(difference + " (" + percents + "%)");
        if (asset.getOpen() < (asset.getLatestPrice())) {

diff.setTextColor(view.getResources().getColor(R.color.green));
        } else {

diff.setTextColor(view.getResources().getColor(R.color.red));
        }

        asset_item.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                openAssetPage(asset.getSymbol());
            }
        });

        return view;
    }
}

```

```

        private void openAssetPage(String symbol) {
            FragmentManager fragmentManager =
                ((FragmentActivity)getContext()).getSupportFragmentManager();

            // Создаем экземпляр FilterFragment
            AssetPageFragment newFragment = new AssetPageFragment(symbol);
            // Начинаем транзакцию фрагмента
            FragmentTransaction transaction =
                fragmentManager.beginTransaction();

            // Заменяем текущий фрагмент на FilterFragment, если есть
            // какой-то текущий фрагмент,
            // или добавляем его, если текущий фрагмент отсутствует
            transaction.replace(R.id.frame_layout, newFragment);

            // Добавляем транзакцию в стек возврата, чтобы пользователь
            // мог вернуться к предыдущему состоянию фрагмента
            transaction.addToBackStack(null);

            // Применяем транзакцию
            transaction.commit();
        }
    }
}

```

Листинг A.2 – Код класса «Filter»

```

public class Filter {
    private String name;
    private List<FilterItem> subFilters;

    public Filter(String name, List<FilterItem> subFilters) {
        this.name = name;
        this.subFilters = subFilters;
    }

    public String getName() {
        return name;
    }

    public List<FilterItem> getSubFilters() {
        return subFilters;
    }
}

```

Листинг A.3 – Код класса «PortfolioFragment»

```

public class PortfolioFragment extends Fragment {

    ArrayList<Portfolio> portfolios;
    List<Asset> assets;
    PortfolioAdapter adapter;
    String portfolioName;
    Retrofit retrofit;
    ApiService apiService;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
        container,

```

```

        Bundle savedInstanceState) {
            View view = inflater.inflate(R.layout.fragment_portfolio,
container, false);
            ExpandableListView expandableListView =
view.findViewById(R.id.portfolio_list);
            initialize();
            initializeRetrofit();

            SharedPreferences preferences =
getActivity().getSharedPreferences("my_prefs", Context.MODE_PRIVATE);

            updatePortfolios(preferences);

            expandableListView.setAdapter(adapter);
            expandableListView.setGroupExpandListener(new
ExpandableListView.OnGroupExpandListener() {
                int lastExpandedPosition = -1;
                @Override
                public void onGroupExpand(int groupPosition) {
                    if (lastExpandedPosition != -1 && groupPosition !=
lastExpandedPosition){
expandableListView.collapseGroup(lastExpandedPosition);
                    }
                    lastExpandedPosition = groupPosition;
                }
            });

            ImageView add_portfolio = view.findViewById(R.id.button);
            add_portfolio.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    AlertDialog.Builder builder = new
AlertDialog.Builder(getActivity());
                    View dialogView =
getLayoutInflater().inflate(R.layout.add_portfolio, null);
                    builder.setView(dialogView);

                    final EditText editTextPortfolioName =
dialogView.findViewById(R.id.editText_portfolio_name);

                    builder.setPositiveButton("Добавить", new
DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int
which) {
                            // Получить название портфеля
                            portfolioName =
editTextPortfolioName.getText().toString();

                            Call<String> call =
apiService.addPortfolio("Bearer " +
                                preferences.getString("access", ""),
                                Map.of("name", portfolioName));
                            call.enqueue(new Callback<String>() {
                                @Override
                                public void onResponse(Call<String> call,
Response<String> response) {

```

```

        if (response.isSuccessful()) {
            updatePortfolios(preferences);
            Toast.makeText(getContext(),
"Success", Toast.LENGTH_SHORT).show();
        } else {

            Toast.makeText(getContext(),
"Failure", Toast.LENGTH_SHORT).show();
        }
    }

    @Override
    public void onFailure(Call<String> call,
Throwable t) {
        Log.e("Network Error", "Code: " +
call.isCanceled() + ", Message: " + t.getMessage());
    }
});

    // Добавить портфель в список
    // Обновить ExpandableListView
}
});

    builder.setNegativeButton("Отмена", null);

    AlertDialog dialog = builder.create();
    dialog.show();
}
});
// Inflate the layout for this fragment
return view;
}

private void initialize() {
    portfolios = new ArrayList<>();
    assets = new ArrayList<>();
    adapter = new PortfolioAdapter(getActivity(),
getActivity().getSupportFragmentManager(), portfolios);
}

private void initializeRetrofit() {
    Gson gson = new GsonBuilder()
        .setLenient()
        .create();

    retrofit = new Retrofit.Builder()
        .baseUrl("http://10.0.2.2:8081")
        .addConverterFactory(ScalarsConverterFactory.create())

.addConverterFactory(GsonConverterFactory.create(gson))
        .build();
    apiService = retrofit.create(ApiService.class);
}

private void updatePortfolios(SharedPreferences preferences) {
    Log.e("TOKENS", "access: " + preferences.getString("access",
"") + " refresh: " + preferences.getString("refresh", ""));
}

```

```

        Call<List<Portfolio>> call = apiService.getPortfolios("Bearer
" + preferences.getString("access", ""));
        call.enqueue(new Callback<List<Portfolio>>() {
            @Override
            public void onResponse(Call<List<Portfolio>> call,
Response<List<Portfolio>> response) {
                if (response.isSuccessful()) {
                    portfolios.clear();
                    portfolios.addAll(response.body());
                    Log.e("EEEEEEEE", portfolios.toString());
                    adapter.notifyDataSetChanged();
                } else {
                    try {
                        Log.e("ERRORRRRRR",
response.errorBody().string());
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            }

            @Override
            public void onFailure(Call<List<Portfolio>> call,
Throwable t) {
                Toast.makeText(getContext(), "Network error",
Toast.LENGTH_SHORT).show();
            }
        });
    }
}

```

Листинг А.4 – Код класса «TokenRefreshService»

```

public class TokenRefreshService {

    private ApiService apiService;
    private Context context;
    private ProgressDialog progressDialog;

    public TokenRefreshService(Context context) {
        this.context = context;

        // Инициализация Retrofit и ApiService
        Gson gson = new GsonBuilder()
            .setLenient()
            .create();

        Retrofit retrofit = new Retrofit.Builder()
            .baseUrl("http://10.0.2.2:8081")

            .addConverterFactory(GsonConverterFactory.create(gson))
            .build();

        apiService = retrofit.create(ApiService.class);
    }
}

```

```

        public void refreshToken(String refreshToken, final
TokenRefreshCallback callback) {
            progressDialog = new ProgressDialog(context);
            progressDialog.setMessage("Refreshing token...");
            progressDialog.setCancelable(false);
            progressDialog.show();

            Map<String, String> map = new HashMap<>();
            map.put("refresh_token", refreshToken);
            Call<AuthResponse> call = apiService.refreshToken(map);
            call.enqueue(new Callback<AuthResponse>() {
                @Override
                public void onResponse(Call<AuthResponse> call,
Response<AuthResponse> response) {

                    if (response.isSuccessful()) {
                        AuthResponse authResponse = response.body();

                        // Обработка успешного обновления токена

callback.onTokenRefreshed(authResponse.getToken());
                    } else {
                        // Обработка ошибки обновления токена
                        callback.onTokenRefreshFailed();
                    }
                    progressDialog.dismiss(); // Скрыть ProgressDialog
после выполнения запроса
                }

                @Override
                public void onFailure(Call<AuthResponse> call, Throwable
t) {

                    // Обработка ошибок сети при попытке обновления токена
                    callback.onTokenRefreshFailed();
                    progressDialog.dismiss(); // Скрыть ProgressDialog в
случае ошибки
                }
            });
        }

        public interface TokenRefreshCallback {
            void onTokenRefreshed(String newAccessToken);

            void onTokenRefreshFailed();
        }
    }
}

```

Листинг А.5 – Код класса «Asset»

```

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Asset {
    private String companyName;
    private String symbol;
    private double open;
    private double latestPrice;
}

```

```

    private String primaryExchange;
    private String logo;
}

```

Листинг А.6 – Код интерфейса «ApiService»

```

public interface ApiService {
    @POST("/auth/signup")
    Call<String> registerUser(@Body User user);

    @POST("/auth/signin")
    Call<AuthResponse> authUser(@Body User user);

    @GET("/secured/user")
    Call<String> getUserDetails(@Header("Authorization") String
authToken);

    @POST("/auth/refresh")
    Call<AuthResponse> refreshToken(@Body Map<String, String> token);

    @POST("/auth/validate-token")
    Call<String> checkToken(@Header("Authorization") String token);

    @GET("/portfolios/")
    Call<List<Portfolio>> getPortfolios(@Header("Authorization")
String token);

    @POST("/portfolios/add")
    Call<String> addPortfolio(@Header("Authorization") String token,
@Body Map<String, String> name);

    @POST("/portfolios/delete")
    Call<String> deletePortfolio(@Header("Authorization") String
token, @Body Portfolio portfolio);

    @POST("/portfolios/add-asset")
    Call<String> addAssetToPortfolio(@Header("Authorization") String
token, @Body Map<String, String> map);

    @GET("/stocks/most-active")
    Call<List<Asset>> getMostActiveAssets();

    @POST("/stocks/search")
    Call<List<Asset>> searchAssets(@Body Map<String, String> string);

    @GET("/sectors")
    Call<List<FilterItem>> getSectors();

    @POST("/stocks/filter")
    Call<List<Asset>> filtering(@Body Map<String, List<FilterItem>>
map);

    @POST("/stocks/stock-info")
    Call<AssetInfo> getStockInfo(@Body Map<String, String> map);

    @POST("/stocks/chart")
    Call<List<ChartData>> getChartData(@Body Map<String, String> map);
}

```

```

    @POST("/stocks/quote")
    Call<Asset> getQuote(@Body Map<String, String> map);

    @POST("/notes/add-to-asset")
    Call<String> addNoteToAsset(@Header("Authorization") String token,
    @Body Map<String, String> map);

    @POST("/notes/get-to-asset")
    Call<List<Note>> getNoteToAsset(@Header("Authorization") String
    token, @Body Map<String, String> map);

    @POST("/notes/add-to-portfolio")
    Call<String> addNoteToPortfolio(@Header("Authorization") String
    token, @Body Map<String, String> map);

    @POST("/notes/get-to-portfolio")
    Call<List<Note>> getNoteToPortfolio(@Header("Authorization")
    String token, @Body Map<String, String> map);

    @POST("/notes/delete-to-asset")
    Call<String> deleteToAsset(@Header("Authorization") String token,
    @Body Note note);

    @POST("/notes/delete-to-portfolio")
    Call<String> deleteToPortfolio(@Header("Authorization") String
    token, @Body Note note);

    @POST("/notes/change-to-asset")
    Call<String> changeToAsset(@Header("Authorization") String token,
    @Body ChangingNote note);

    @POST("/notes/change-to-portfolio")
    Call<String> changeToPortfolio(@Header("Authorization") String
    token, @Body ChangingNote note);

    @POST("/news")
    Call<List<News>> getNews(@Body Map<String, String> map);

    @POST("/saved-news/add")
    Call<String> saveNews(@Header("Authorization") String token, @Body
    News news);

    @GET("/saved-news/")
    Call<List<News>> getSavedNews(@Header("Authorization") String
    token);

    @POST("/saved-news/delete")
    Call<String> deleteNews(@Header("Authorization") String token,
    @Body News news);

    @POST("/saved-news/check")
    Call<String> checkNews(@Header("Authorization") String token,
    @Body Map<String, String> news);
}

```

Листинг А.7 – Код класса «MainActivity»

```

public class MainActivity extends AppCompatActivity {

    ActivityMainBinding binding;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding = ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());
        replaceFragment(new AssetsFragment());

        binding.bottomNavigationView.setOnItemSelectedListener(item ->
{
            if (item.getItemId() == R.id.assets_part) {
                replaceFragment(new AssetsFragment());
            } else if (item.getItemId() == R.id.news_part) {
                replaceFragment(new NewsFragment());
            } else if (item.getItemId() == R.id.portfolios_part) {
                replaceFragment(new PortfolioFragment());
            } else if (item.getItemId() == R.id.user_part) {
                replaceFragment(new UserFragment());
            }

            return true;
        });
    }

    private void replaceFragment(Fragment fragment){

        FragmentManager fragmentManager = getSupportFragmentManager();
        FragmentTransaction fragmentTransaction =
fragmentManager.beginTransaction();

        fragmentTransaction.replace(R.id.frame_layout, fragment);
        fragmentTransaction.commit();
    }
}

```

Приложение Б
(рекомендуемое)
Исходный код серверной части

Листинг Б.1 – Код класса «SecurityConfiguration»

```

@Configuration
@EnableWebSecurity
public class SecurityConfiguration {
    private final AuthenticationProvider authenticationProvider;
    private final JwtAuthenticationFilter jwtAuthenticationFilter;

    public SecurityConfiguration(
        JwtAuthenticationFilter jwtAuthenticationFilter,
        AuthenticationProvider authenticationProvider
    ) {
        this.authenticationProvider = authenticationProvider;
        this.jwtAuthenticationFilter = jwtAuthenticationFilter;
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http)
    throws Exception {
        http
            .csrf(AbstractHttpConfigurer::disable)
            .cors(httpSecurityCorsConfigurer ->
                httpSecurityCorsConfigurer.configurationSource(request ->
                    new
                    CorsConfiguration().applyPermitDefaultValues()))
            // .exceptionHandling(exceptions -> exceptions
            // .authenticationEntryPoint(new
            HttpStatusEntryPoint(HttpStatus.UNAUTHORIZED)))
            .sessionManagement(session -> session
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authorizeHttpRequests(authorize -> authorize
                .requestMatchers("/auth/**").permitAll()
                .requestMatchers("/users/**").fullyAuthenticated()
                .requestMatchers("/portfolios/**").authenticated()
                .anyRequest().authenticated()
            )
            .addFilterBefore(jwtAuthenticationFilter,
                UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }
}

```

Листинг Б.2 – Код класса «AuthenticationController»

```

@RequestMapping("/auth")
@RestController
public class AuthenticationController {
    private final JwtService jwtService;
    @Autowired
    @Setter
    private UserRepository userRepository;
}

```

```

        private final AuthenticationService authenticationService;

        public AuthenticationController(JwtService jwtService,
        AuthenticationService authenticationService) {
            this.jwtService = jwtService;
            this.authenticationService = authenticationService;
        }

        @PostMapping("/signup")
        public ResponseEntity<String> register(@RequestBody
        RegisterUserDto registerUserDto) {
            if
            (userRepository.existsUserByUsername(registerUserDto.getUsername())) {
                return
                ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Choose different
                name");
            }
            if
            (userRepository.existsUserByEmail(registerUserDto.getEmail())) {
                return
                ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Choose different
                email");
            }
            User registeredUser =
            authenticationService.signup(registerUserDto);

            return ResponseEntity.ok("success");
        }

        @PostMapping("/signin")
        public ResponseEntity<LoginResponse> authenticate(@RequestBody
        LoginUserDto loginUserDto) {
            User authenticatedUser =
            authenticationService.authenticate(loginUserDto);

            String jwtToken = jwtService.generateToken(authenticatedUser);

            LoginResponse loginResponse = new LoginResponse();
            loginResponse.setToken(jwtToken);
            loginResponse.setExpiresIn(jwtService.getExpirationTime());

            return ResponseEntity.ok(loginResponse);
        }

        @PostMapping("/refresh")
        public ResponseEntity<?> refreshAccessToken(@RequestBody
        Map<String, String> request) {
            String refreshToken = request.get("refresh_token");
            try {
                Optional<User> user =
                userRepository.findByUsername(jwtService.getRefresh().extractUsername(
                refreshToken));

                if (jwtService.getRefresh().isTokenValid(refreshToken)) {
                    String newAccessToken =
                    jwtService.getAccess().generateToken(user.get()); // Логика генерации
                    нового Access Token
                }
            }
        }
    }

```

```

        LoginResponse loginResponse = new LoginResponse();
        loginResponse.setToken(newAccessToken);

loginResponse.setExpiresIn(jwtService.getAccess().getExpirationTime())
;
        return ResponseEntity.ok(loginResponse);
    } else {
        return ResponseEntity.badRequest().body("Invalid
refresh token");
    }
    } catch (Exception e) {
        return
ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Invalid refresh
token");
    }
}

@PostMapping("/validate-token")
public ResponseEntity<?>
validateToken(@RequestHeader("Authorization") String token) {
    try {
        String tokenValue = token.substring(7); // Получаем чистый
токен

        Optional<User> user =
userRepository.findByUsername(jwtService.getAccess().extractUsername(t
okenValue));

        if (user.isPresent() &&
jwtService.getAccess().isTokenValid(tokenValue, user.get())) {
            return ResponseEntity.ok().body("Token is valid");
        } else {
            return
ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Invalid token or
user not found");
        }
    } catch (NoSuchElementException e) {
        return
ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("User not found");
    } catch (Exception e) {
        return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("An error
occurred while validating token");
    }
}
}
}

```

Листинг Б.3 – Код класса « LoginUserDto»

```

@Data
@AllArgsConstructor
@NoArgsConstructor
public class LoginUserDto {
    private String password;
    private String username;
}

```

Листинг Б.4 – Код класса «JwtService»

```

@Service
public class JwtService {
    @Value("${authentication.app.secret}")
    private String secretKey;
    @Value("${authentication.app.lifetime}")
    private long jwtExpiration;
    @Getter
    private RefreshJwtService refresh;
    @Getter
    private AccessJwtService access;

    public JwtService(RefreshJwtService refreshJwtService,
AccessJwtService accessJwtService) {
        refresh = refreshJwtService;
        access = accessJwtService;
    }

    public String generateToken(UserDetails userDetails) {
        return generateToken(Map.of("refresh",
refresh.generateToken(userDetails),
"access", access.generateToken(userDetails)));
    }

    public String generateToken(Map<String, Object> extraClaims) {
        return buildToken(extraClaims, jwtExpiration);
    }

    public long getExpirationTime() {
        return jwtExpiration;
    }

    private String buildToken(
        Map<String, Object> extraClaims,
        long expiration
    ) {
        return Jwts
            .builder()
            .setClaims(extraClaims)
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() +
expiration))
            .signWith(getSignInKey(), SignatureAlgorithm.HS256)
            .compact();
    }

    private Key getSignInKey() {
        byte[] keyBytes = Decoders.BASE64.decode(secretKey);
        return Keys.hmacShaKeyFor(keyBytes);
    }
}

```

Листинг Б.5 – Код класса «News»

```

@Data
@Document("news")
@NoArgsConstructor
@AllArgsConstructor
public class News {

```

```

@Id
private String id;

@Field("author")
private String author;
@Field("title")
private String title;
@Field("description")
private String description;
@Field("url")
private String url;
@Field("urlToImage")
private String urlToImage;
@Field("publishedAt")
private String publishedAt;
@Field("content")
private String content;

@JsonIgnore
@Field("user")
User user;
}

```

Листинг Б.6 – Код класса «NewsRepository»

```

@Repository
public interface NewsRepository extends MongoRepository<News, String>
{
    List<News> findAllByUser(User user);
    boolean existsNewsByUserAndTitleAndPublishedAt(User user, String
title, String date);
    void deleteByUserAndTitleAndPublishedAt(User user, String title,
String date);
}

```

Листинг Б.7 – Код класса «NewsService»

```

@Service
public class NewsService {
    @Autowired
    NewsRepository repository;
    @Autowired
    UserRepository userRepository;

    public String saveNews(News news, Principal principal) {
        Optional<User> user =
userRepository.findByUsername(principal.getName());
        news.setUser(user.get());
        if
(repository.existsNewsByUserAndTitleAndPublishedAt(user.get(),
news.getTitle(), news.getPublishedAt())) {
            return "already added";
        } else {
            repository.save(news);
            return "success";
        }
    }
}

```

```

        public List<News> getAllNews(Principal principal) {
            Optional<User> user =
userRepository.findByUsername(principal.getName());
            List<News> list = new ArrayList<>();
            list.addAll(repository.findAllByUser(user.get()));
            return list;
        }

        public void deleteNews(Principal principal, News news) {
            Optional<User> user =
userRepository.findByUsername(principal.getName());
            repository.deleteByUserAndTitleAndPublishedAt(user.get(),
news.getTitle(), news.getPublishedAt());
        }

        public String checkNews(Principal principal, String title, String
date) {
            Optional<User> user =
userRepository.findByUsername(principal.getName());
            if
(repository.existsNewsByUserAndTitleAndPublishedAt(user.get(), title,
date)) {
                return "exists";
            } else {
                return "not";
            }
        }
    }
}

```

Приложение В
(рекомендуемое)
Результаты модульного тестирования

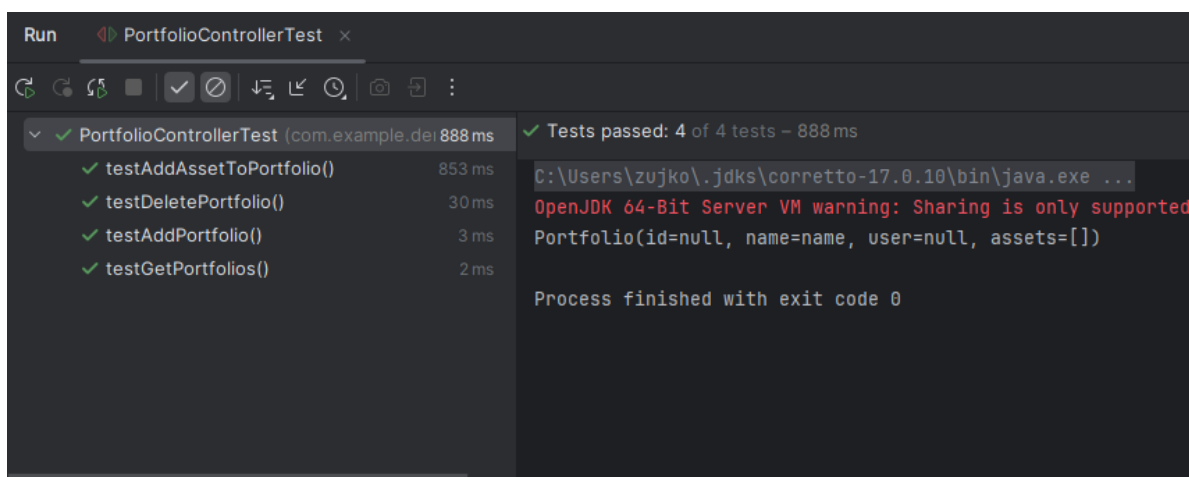


Рисунок В.1 – Результат модульного тестирования контроллера управления портфелями пользователя

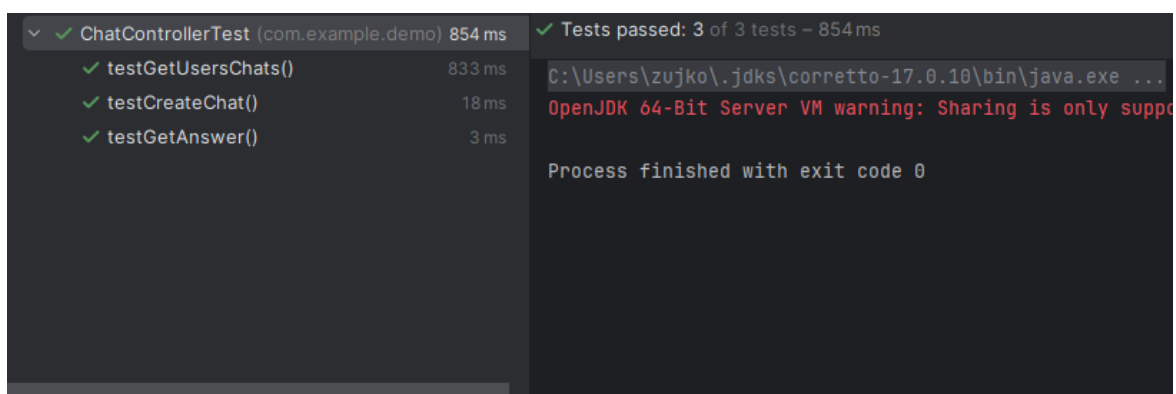


Рисунок В.2 – Результат модульного тестирования контроллера управления чатами пользователя

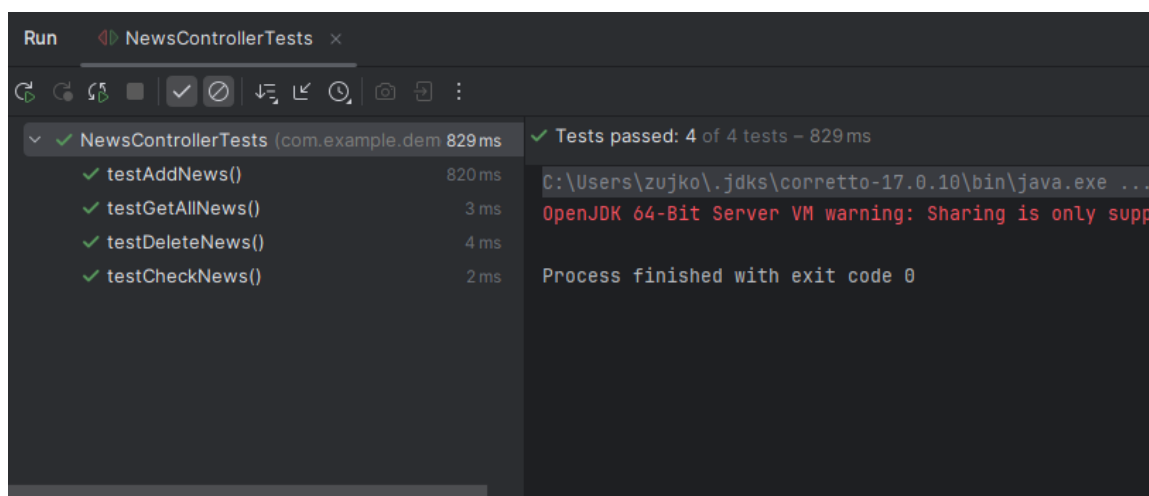


Рисунок В.3 – Результат модульного тестирования контроллера управления новостями пользователя

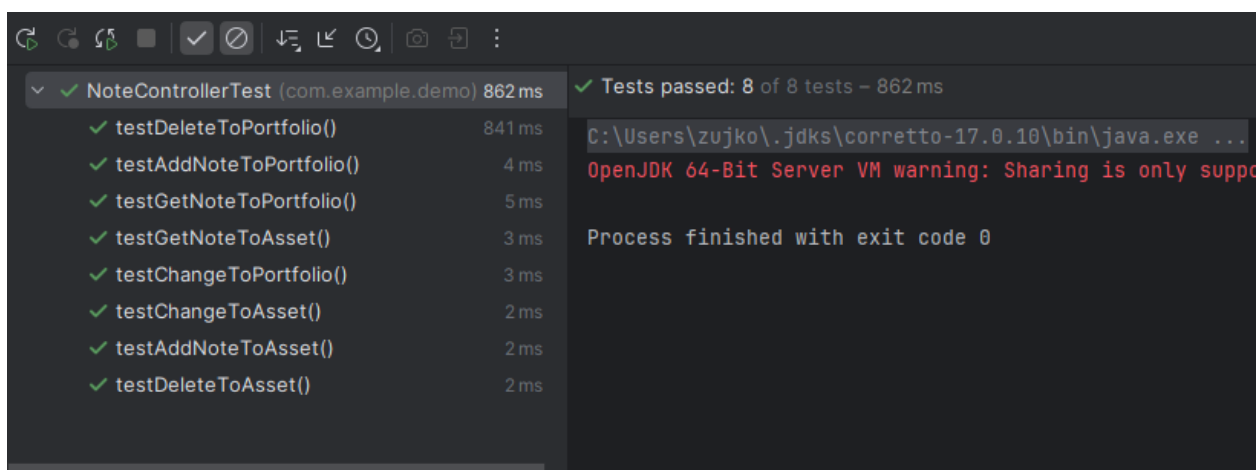


Рисунок В.4 – Результат модульного тестирования контроллера управления заметками пользователя

Приложение Г
(рекомендуемое)
Исходный код модульных тестов

Листинг Г.1 – Код модульных тестов для контроллера управления портфелями пользователя

```

@ExtendWith(MockitoExtension.class)
class PortfolioControllerTest {

    @Mock
    PortfolioService portfolioService;

    @InjectMocks
    PortfolioController portfolioController;

    @Test
    void testAddPortfolio() {
        String request = new StringRequest("Portfolio Name",
        "");
        Principal principal = mock(Principal.class);
        when(portfolioService.createPortfolio(anyString(),
        any(Principal.class))).thenReturn("Portfolio Created");

        ResponseEntity<String> response =
        portfolioController.addPortfolio(request, principal);

        assertEquals("Portfolio Created", response.getBody());
        verify(portfolioService, times(1)).createPortfolio("Portfolio
        Name", principal);
    }

    @Test
    void testDeletePortfolio() {
        Portfolio portfolio = new Portfolio("name", new
        ArrayList<>());
        Principal principal = mock(Principal.class);
        when(portfolioService.deletePortfolio(any(Portfolio.class),
        any(Principal.class))).thenReturn("Portfolio Deleted");

        ResponseEntity<String> response =
        portfolioController.deletePortfolio(portfolio, principal);

        assertEquals("Portfolio Deleted", response.getBody());
        verify(portfolioService, times(1)).deletePortfolio(portfolio,
        principal);
    }

    @Test
    void testGetPortfolios() {
        Principal principal = mock(Principal.class);
        List<PortfoliosResponse> portfolios = Arrays.asList(new
        PortfoliosResponse(), new PortfoliosResponse());

        when(portfolioService.getAllPortfolios(principal)).thenReturn(portfoli
        os);

        ResponseEntity<List<PortfoliosResponse>> response =
        portfolioController.getPortfolios(principal);
    }
}

```

```

        assertEquals(portfolios, response.getBody());
        verify(portfolioService,
times(1)).getAllPortfolios(principal);
    }

    @Test
    void testAddAssetToPortfolio() {
        Principal principal = mock(Principal.class);
        Map<String, String> map = Map.of("name", "Asset Name",
"symbol", "Asset Symbol", "price", "100.0");

        doNothing().when(portfolioService).addAssetToPortfolio(principal,
"Asset Name", "Asset Symbol", 100.0);

        String result = portfolioController.addAssetToPortfolio(map,
principal);

        assertEquals("success", result);
        verify(portfolioService,
times(1)).addAssetToPortfolio(principal, "Asset Name", "Asset Symbol",
100.0);
    }
}

```

Листинг Г.2 – Код модульных тестов для контроллера управления чатами

ПОЛЬЗОВАТЕЛЯ

```

@ExtendWith(MockitoExtension.class)
class ChatControllerTests {

    @Mock
    ChatService chatService;

    @InjectMocks
    ChatController chatController;

    @Test
    void testGetUsersChats() {
        Principal principal = mock(Principal.class);
        List<Chat> chats = Arrays.asList(new Chat(), new Chat());
        when(chatService.getAllByUser(principal)).thenReturn(chats);

        List<Chat> result = chatController.getUsersChats(principal);

        assertEquals(chats, result);
        verify(chatService, times(1)).getAllByUser(principal);
    }

    @Test
    void testCreateChat() {
        Principal principal = mock(Principal.class);
        Map<String, String> map = Map.of("title", "Chat Title",
"model", "Chat Model");
        when(chatService.createChat("Chat Title", "Chat Model",
principal, false)).thenReturn(new Chat());
    }
}

```

```

        ResponseEntity<Chat> response = chatController.createChat(map,
principal);

        assertEquals(new Chat(), response.getBody());
        verify(chatService, times(1)).createChat("Chat Title", "Chat
Model", principal, false);
    }

    @Test
    void testGetAnswer() {
        Principal principal = mock(Principal.class);
        Chat chat = new Chat();
        when(chatService.getAnswer(chat, principal)).thenReturn(new
Messages());

        Messages result = chatController.getAnswer(chat, principal);

        assertEquals(new Messages(), result);
        verify(chatService, times(1)).getAnswer(chat, principal);
    }
}

```

Листинг Г.3 – Код модульных тестов для контроллера управления новостями ПОЛЬЗОВАТЕЛЯ

```

@ExtendWith(MockitoExtension.class)
class NewsControllerTests {

    @Mock
    NewsService newsService;

    @InjectMocks
    NewsController newsController;

    @Test
    void testAddNews() {
        Principal principal = mock(Principal.class);
        News news = new News();
        when(newsService.saveNews(news, principal)).thenReturn("News
Added");

        String result = newsController.addNews(principal, news);

        assertEquals("News Added", result);
        verify(newsService, times(1)).saveNews(news, principal);
    }

    @Test
    void testGetAllNews() {
        Principal principal = mock(Principal.class);
        List<News> newsList = Arrays.asList(new News(), new News());
        when(newsService.getAllNews(principal)).thenReturn(newsList);

        List<News> result = newsController.getAllNews(principal);

        assertEquals(newsList, result);
        verify(newsService, times(1)).getAllNews(principal);
    }
}

```

```

    }

    @Test
    void testDeleteNews() {
        Principal principal = mock(Principal.class);
        News news = new News();
        // Assuming deleteNews returns void, use doNothing() for void
methods    doNothing().when(newsService).deleteNews(principal, news);

        String result = newsController.deleteNews(principal, news);

        assertEquals("success", result); // Assuming you return
"success" directly
        verify(newsService, times(1)).deleteNews(principal, news);
    }

    @Test
    void testCheckNews() {
        Principal principal = mock(Principal.class);
        Map<String, String> map = Map.of("title", "News Title",
"date", "2024-06-06");
        when(newsService.checkNews(principal, "News Title", "2024-06-
06")).thenReturn("Checked");

        String result = newsController.checkNews(principal, map);

        assertEquals("Checked", result);
        verify(newsService, times(1)).checkNews(principal, "News
Title", "2024-06-06");
    }
}

```

Листинг Г.4 – Код модульных тестов для контроллера управления заметками

ПОЛЬЗОВАТЕЛЯ

```

@ExtendWith(MockitoExtension.class)
class NoteControllerTests {

    @Mock
    NoteService noteService;

    @InjectMocks
    NoteController noteController;

    @Test
    void testAddNoteToAsset() {
        Principal principal = mock(Principal.class);
        Map<String, String> map = Map.of("title", "Note Title",
"content", "Note Content", "symbol", "Asset Symbol");
        doNothing().when(noteService).addNoteToAsset(principal, "Note
Title", "Note Content", "Asset Symbol");

        String result = noteController.addNoteToAsset(principal, map);

        assertEquals("success", result);
    }
}

```

```

        verify(noteService, times(1)).addNoteToAsset(principal, "Note
Title", "Note Content", "Asset Symbol");
    }

    @Test
    void testGetNoteToAsset() {
        Principal principal = mock(Principal.class);
        Map<String, String> map = Map.of("symbol", "Asset Symbol");
        List<NoteResponse> notes = Arrays.asList(new NoteResponse(),
new NoteResponse());
        when(noteService.getNotesToAsset(principal, "Asset
Symbol")).thenReturn(notes);

        List<NoteResponse> result =
noteController.getNoteToAsset(principal, map);

        assertEquals(notes, result);
        verify(noteService, times(1)).getNotesToAsset(principal,
"Asset Symbol");
    }

    @Test
    void testAddNoteToPortfolio() {
        Principal principal = mock(Principal.class);
        Map<String, String> map = Map.of("title", "Note Title",
"content", "Note Content", "name", "Portfolio Name");
        doNothing().when(noteService).addNoteToPortfolio(principal,
"Note Title", "Note Content", "Portfolio Name");

        String result = noteController.addNoteToPortfolio(principal,
map);

        assertEquals("success", result);
        verify(noteService, times(1)).addNoteToPortfolio(principal,
"Note Title", "Note Content", "Portfolio Name");
    }

    @Test
    void testGetNoteToPortfolio() {
        Principal principal = mock(Principal.class);
        Map<String, String> map = Map.of("name", "Portfolio Name");
        List<NoteResponse> notes = Arrays.asList(new NoteResponse(),
new NoteResponse());
        when(noteService.getNotesToPortfolio(principal, "Portfolio
Name")).thenReturn(notes);

        List<NoteResponse> result =
noteController.getNoteToPortfolio(principal, map);

        assertEquals(notes, result);
        verify(noteService, times(1)).getNotesToPortfolio(principal,
"Portfolio Name");
    }

    @Test
    void testDeleteToAsset() {
        Principal principal = mock(Principal.class);
        NoteResponse response = new NoteResponse();

```

```

        doNothing().when(noteService).deleteNoteToAsset(principal,
response);

        String result = noteController.deleteToAsset(principal,
response);

        assertEquals("success", result);
        verify(noteService, times(1)).deleteNoteToAsset(principal,
response);
    }

    @Test
    void testChangeToAsset() {
        Principal principal = mock(Principal.class);
        ChangingNoteResponse noteResponse = new
ChangingNoteResponse();
        doNothing().when(noteService).changeNoteToAsset(principal,
noteResponse);

        String result = noteController.changeToAsset(principal,
noteResponse);

        assertEquals("success", result);
        verify(noteService, times(1)).changeNoteToAsset(principal,
noteResponse);
    }

    @Test
    void testDeleteToPortfolio() {
        Principal principal = mock(Principal.class);
        NoteResponse response = new NoteResponse();
        doNothing().when(noteService).deleteNoteToPortfolio(principal,
response);

        String result = noteController.deleteToPortfolio(principal,
response);

        assertEquals("success", result);
        verify(noteService, times(1)).deleteNoteToPortfolio(principal,
response);
    }

    @Test
    void testChangeToPortfolio() {
        Principal principal = mock(Principal.class);
        ChangingNoteResponse noteResponse = new
ChangingNoteResponse();
        doNothing().when(noteService).changeNoteToPortfolio(principal,
noteResponse);

        String result = noteController.changeToPortfolio(principal,
noteResponse);

        assertEquals("success", result);
        verify(noteService, times(1)).changeNoteToPortfolio(principal,
noteResponse);
    }
}

```