

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 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Бизнес-проект «Мобильное приложение,
предназначенное для повышения осознанности
потребления кофе»**

Студент

(подпись, дата) Левин Андрей Алексеевич
(ФИО полностью)

Руководитель

(подпись, дата) Козлов А.Ю.
(фамилия, инициалы)

Нормоконтролёр

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

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

Заведующий кафедрой

(подпись) Козлов А.Ю.
(фамилия, инициалы)

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

Секретарь ГЭК

(подпись) Попова Н.А.
(фамилия, инициалы)

Пенза, 2025

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
«20» декабря 2024 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ В ФОРМЕ
СТАРТАП-ПРОЕКТА**

1. Студент Левин Андрей Алексеевич гр. 21ВП2 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Бизнес-проект «Мобильное приложение, предназначенное для повышения осознанности потребления кофе»

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

2. Руководитель работы д.т.н., проф., зав. кафедрой «МОиПЭВМ» Козлов А.Ю.

3. Общая часть ВКР:

Подготовка компьютерной презентации бизнес-проекта (стартапа).

4. Индивидуальная часть ВКР:

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

Назначение: программное средство предназначено для учета и анализа потребления кофе, отслеживания финансовых расходов, мониторинга влияния кофе на эмоциональное состояние, а также визуализации собранных данных

Основные функции:

1) Управление справочником видов кофе;

2) Ведение записей о потреблении кофе;

3) Ведение записей эмоционального состояния;

4) Просмотр статистики и аналитики;

5) Настройка персональных целей.

Технологии разработки: ООП

Язык программирования: Kotlin

Инструментальные средства разработки: Android Studio

Среда исполнения: Android 9+

4.2. Перечень подлежащих разработке вопросов

1) Анализ предметной области и требований к системе

2) Проектирование программных средств

3) Реализация программных средств

4) Тестирование программных средств

5) Оформление пояснительной записки

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2025 – 14.04.2025	
2	Проектирование программных средств	20%	15.04.2025 – 25.04.2025	
3	Реализация программных средств	30%	26.04.2025 – 12.05.2025	
4	Тестирование программных средств	20%	13.05.2025 – 22.05.2025	
5	Оформление пояснительной записки	20%	23.05.2025 – 03.06.2025	

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы

(подпись, дата)

Задание к исполнению принял студент

(подпись, дата)

Работу к защите допустить

Декан ФВТ

(подпись, дата)

Л.Р. Фионова

Реферат

Пояснительная записка содержит 170 листов, 31 рисунок, 14 таблиц, 3 листинга, 22 использованных источника, 5 приложений.

БИЗНЕС-ПРОЕКТ, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, АНАЛИТИКА ПОТРЕБЛЕНИЯ, КОНТРОЛЬ НАСТРОЕНИЯ, УЧЕТ РАСХОДОВ, MVVM-АРХИТЕКТУРА, ROOM

Объект исследования: мобильные приложения для мониторинга потребления напитков и анализа привычек.

Целью выпускной квалификационной работы как стартапа является разработка бизнес-плана проекта «Мобильное приложение, предназначенное для повышения осознанности потребления кофе».

Целью индивидуальной части выпускной квалификационной работы как стартапа является разработка мобильного приложения для осознанного потребления кофе с возможностью ведения записей, анализа статистики и формирования персональных рекомендаций.

Технология разработки: ООП.

Среда исполнения: Android.

Область применения: забота о здоровье и личная эффективность.

Разработка мобильного приложения осуществлялась на языке программирования Kotlin в среде разработки Android Studio. Разработка велась в операционной системе Windows 10.

					ПГУ 09.03.04– 08BP212.14 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Бизнес-проект «Мобильное приложение, предназначенное для повышения осознанности потребления кофе» Пояснительная записка	Лит.	Лист	Листов
Разраб.		Левин А.А.						
Пров.		Козлов А.Ю.					4	170
						Группа 21ВП2		
Н. Контр.		Такташкин Д.В						
Утв.		Козлов А.Ю.						

Содержание

Введение.....	7
1 Бизнес-проект мобильного приложения «Coffies»	8
1.1 Анализ рынка потребления кофе.....	8
1.2 Описание бизнес-идеи мобильного приложения.....	9
1.3 Постановка цели и задач бизнес-проекта	9
1.4 Описание задумки бизнес-проекта.....	10
1.5 Маркетинговый план бизнес-проекта	11
1.6 Финансовый план бизнес-проекта.....	12
2 Мобильное приложение, предназначенное для повышения осознанности потребления кофе	13
2.1 Анализ требований и постановка задач на разработку мобильного приложения осознанного потребления кофе	13
2.1.1 Анализ предметной области	13
2.1.2 Сравнительный анализ программных продуктов-аналогов	14
2.1.2.1 Анализ приложения CoffeeCup	14
2.1.2.2 Анализ приложения CaffèInMe	15
2.1.2.3 Анализ приложения Caffeine Clock	16
2.1.2.4 Результаты проведенного анализа приложений.....	17
2.1.3 Анализ требований мобильного приложения	18
2.1.3.1 Анализ функциональных требований	18
2.1.3.2 Анализ нефункциональных требований	22
2.1.4 Постановка цели и задачи на разработку мобильного приложения	23
2.1.5 Анализ разработки и оценка бюджета мобильного приложения	24
2.1.5.1 Роли и задачи участников проекта	24
2.1.5.2 Оценка бюджета на разработку мобильного приложения	25
2.2 Проектирование программных средств для разработки мобильного приложения для осознанного потребления кофе.....	28
2.2.1 Проектирование архитектуры мобильного приложения	28
2.2.2 Проектирование базы данных мобильного приложения	31
2.2.2.1 Проектирование концептуальной модели базы данных.....	32
2.2.2.2 Проектирование логической модели базы данных	33
2.2.2.3 Проектирование физической модели базы данных	33

2.2.3	Проектирование пользовательского интерфейса мобильного приложения.....	34
2.3	Реализация мобильного приложения осознанного потребления кофе.....	36
2.3.1	Реализация архитектуры мобильного приложения.....	36
2.3.2	Реализация базы данных мобильного приложения.....	40
2.3.3	Реализация пользовательского интерфейса мобильного приложения.....	42
2.4	Тестирование мобильного приложения для осознанного потребления кофе.....	44
2.4.1	Обоснование выбора режима тестирования и метрик кода	44
2.4.2	Функциональное тестирование мобильного приложения.....	46
2.4.3	Нефункциональное тестирование мобильного приложения.....	49
2.4.4	Модульное тестирование мобильного приложения.....	52
2.4.5	Метрики кода мобильного приложения	54
	Заключение	56
	Список использованных источников	58
	Приложение А (обязательное) Код мобильного приложения.....	61
	Приложение Б (рекомендуемое) Рекомендательное письмо.....	134
	Приложение В (рекомендуемое) Схемы взаимодействия компонентов.....	136
	Приложение Г (рекомендуемое) Реализация пользовательского интерфейса .	141
	Приложение Д (рекомендуемое) Код модульных тестов	161

Введение

Кофе прочно вошёл в повседневную жизнь россиян. Сегодня это не просто утренний ритуал, а целый рынок, который стремительно расширяется и меняет привычки потребителей. По данным отраслевых аналитиков, с 2020 по 2023 год общий объём потребления кофе в России увеличился примерно на 15% и превысил 210 тысяч тонн. Более 70% взрослого населения страны ежедневно пьют кофе, что относит этот напиток к числу массовых и формирует стабильный спрос на новые сервисы, связанные с кофейной индустрией.

Одновременно усиливается тенденция осознанного потребления: всё больше людей задумываются о влиянии кофеина на здоровье, следят за количеством выпитого кофе и контролируют связанные с этим расходы. В результате возникает спрос на инструменты, позволяющие не только лучше понимать собственные привычки, но и управлять ими.

Тем не менее современные цифровые решения пока не обеспечивают в полной мере возможности для осознанного управления потреблением кофе. Большинство существующих приложений акцентируют внимание на программах лояльности отдельных сетей либо ограничиваются простым подсчётом чашек, без учёта медицинских рекомендаций или финансовых аспектов. С учётом широкой распространённости смартфонов такой дефицит сервисов выглядит особенно странным: пользователи привыкли отслеживать шаги и калории, но практически не имеют инструментов для контроля одного из самых популярных напитков.

Бизнес-проект «Мобильное приложение для осознанного потребления кофе» нацелен на решение этой проблемы путем создания мобильного приложения, объединяющего трекинг потребления кофеина, анализ эмоционального состояния и учёт финансовых расходов. Такой подход будет способствовать не только повышению личного благополучия пользователей, но и формированию более зрелой, ответственной и устойчивой кофейной культуры в целом.

1 Бизнес-проект мобильного приложения «Coffies»

1.1 Анализ рынка потребления кофе

Согласно данным аналитического агентства «Газета.ру», российский рынок кофе демонстрирует стабильный рост: потребление кофе увеличилось с 186 тысяч тонн за 2020 год до 214 тысяч тонн в 2023 году. Более 70% взрослого населения страны употребляют кофе ежедневно. Особенно активно растет рынок кофе навынос, который только за последний год вырос на 15%. Динамика развития рынка кофе навынос (его объем и стоимость за последние 5 лет) представлена на рисунке 1 [1 – 3].

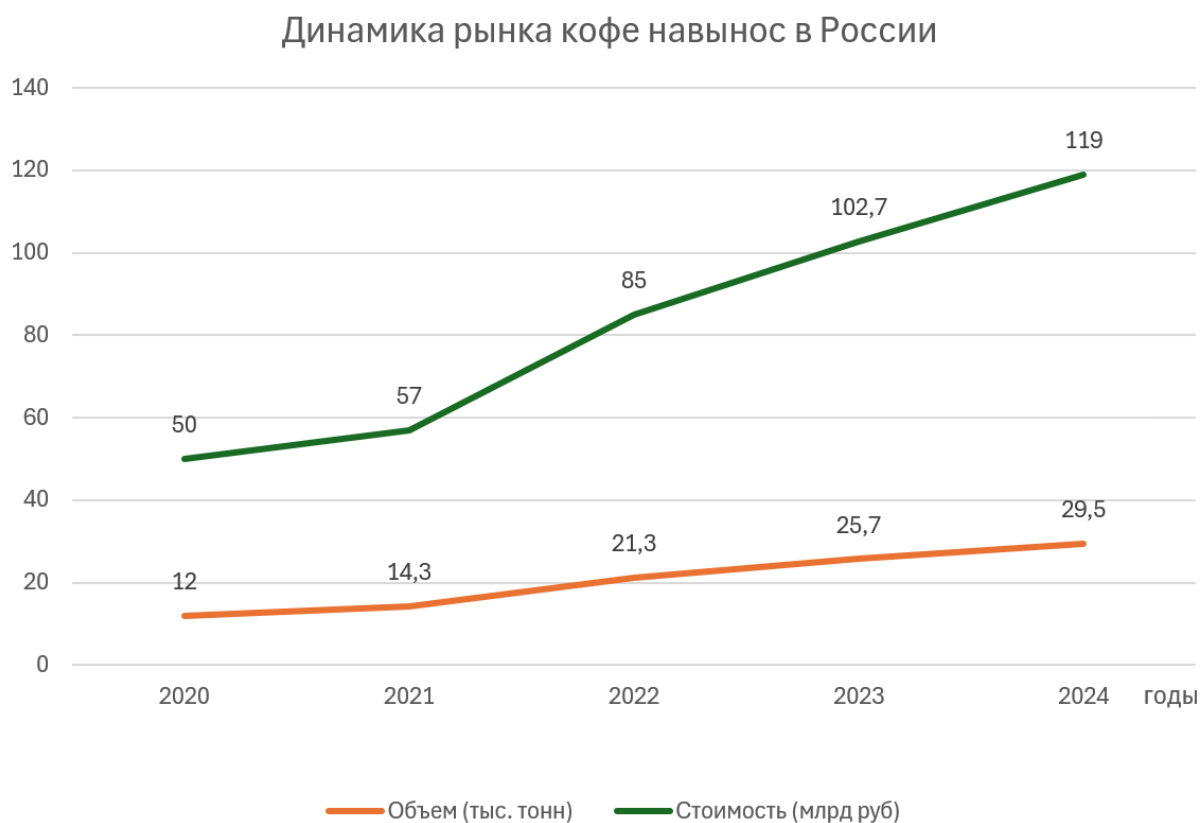


Рисунок 1 – Динамика рынка кофе навынос

Однако несмотря на такой активный рост рынка кофе и массовое проникновение смартфонов, остается недостаточно цифровых решений для какого-либо контроля и осознания потребления. Необходим удобный инструмент для понимания своих привычек, контроля расходов и сохранения

здоровья, что в свою очередь поможет сформировать культуру осознанного потребления кофе.

1.2 Описание бизнес-идеи мобильного приложения

Отдельные потребители кофе нуждаются в инструменте, который объединит учет потребленного кофе и финансовых расходов, свяжет их с самочувствием пользователя и предоставит персональные рекомендации. Однако существующие решения не способны полноценно удовлетворить эти потребности. В то же время небольшие и средние кофейни нуждаются в инструменте, который позволит привлечь и удержать клиентов.

Для решения проблем как пользователей, так и бизнеса предлагается:

- разработать собственное мобильное приложения для формирования осознанного подхода к потреблению кофе;
- распространять приложение среди целевой аудитории отдельных потребителей кофе по модели подписки, а среди бизнеса – по договорам сотрудничества.

Реализация указанных задач позволит удовлетворить потребности как индивидуальных потребителей, так и представителей малого бизнеса. В результате на рынке появится инновационный продукт, способствующий развитию культуры осознанного потребления кофе и обеспечивающий новые возможности для роста кофейных заведений.

1.3 Постановка цели и задач бизнес-проекта

Целью данного бизнес-проекта выступает создание мобильного приложения, ориентированного на формирование культуры осознанного потребления кофе среди пользователей, а также развитие партнерской сети с кофейнями.

Реализация целей данного бизнес-проекта предполагает решение ряда следующих задач:

- проведение анализа рынка и потенциальной экономической ниши;

- описание бизнес-идеи и разрабатываемого продукта;
- разработка мобильного приложения с необходимым функционалом;
- разработка маркетингового плана, описывающего способ и стоимость продажи продукта;
- поиск партнеров проекта.

Реализация перечисленных задач позволит создать конкурентоспособное мобильное приложение, вывести его на рынок, что в дальнейшем станет основой для масштабирования проекта и привлечения инвестиций для дальнейшего развития.

1.4 Описание задумки бизнес-проекта

Мобильное приложение будет предназначено для платформы android и будет выпускаться в двух версиях: базовой (бесплатной) и расширенной (доступной по подписке).

Базовая версия приложения будет сфокусирована на ключевых функциях, необходимых для формирования привычки осознанного потребления кофе и будет включать в себя следующие возможности:

- учет выпитых чашек кофе и полученного кофеина с возможностью аналитики потребления;
- фиксация потраченных средств с возможностью анализа расходов;
- отслеживание эмоционального состояния пользователя до и после приема кофе для выявления индивидуальных зависимостей;
- предоставление базовых персональных рекомендации на основе введенных данных.

Расширенная версия приложения откроет дополнительные функции для более требовательных пользователей и будет включать:

- персональные рекомендации на базе искусственного интеллекта;
- продвинутый финансовый контроль с возможностью постановки индивидуальных целей;

- интеграцию интерактивной карты кофеен с персональными предложениями.

Стоит отметить, что в рамках выполнения данной квалификационной работы в форме стартапа (ВКРС), предлагается разработать только базовую версию продукта для стартап-проекта «Coffies», мобильного приложения для осознанного подхода к кофе.

1.5 Маркетинговый план бизнес-проекта

Целевой аудиторией разрабатываемого проекта будут являться:

- люди (14 – 45 лет): те, кто стремится к осознанному потреблению, хочет контролировать свои кофейные привычки и расходы, а также интересуется улучшением эмоционального состояния и его связью с потреблением кофе;
- бизнес: кофейни и производители, которые хотят продвигать устойчивое потребление и привлекать осознанных потребителей.

Коммерциализацию предлагаемого стартап-проекта предлагается производить по двум схемам взаимодействия:

- «бизнес для потребителя» (b2c) – подписка для конечных пользователей с расширенным функционалом;
- «бизнес для бизнеса» (b2b) – интеграция кофеен для прямого взаимодействия с клиентами [4].

В качестве каналов продвижения продукта будут использоваться:

- социальные сети и профильные сообщества;
- прямые продажи и партнерства с кофейнями;
- контент-маркетинг;
- реферальные и бонусные системы.

Пензенский государственный университет выразил заинтересованность в реализации и продвижении проекта, соответствующее письмо поддержки приведено в приложении Б.

В результате внедрения выбранных моделей монетизации и маркетинговых каналов планируется обеспечить стабильный рост пользовательской базы, повысить узнаваемость бренда и создать дополнительные источники дохода как для стартапа, так и для партнерских кофеен.

1.6 Финансовый план бизнес-проекта

Для конечных пользователей планируется стоимость подписки в размере 100 рублей в месяц, что составляет меньше стоимости 1 чашки кофе и должно являться комфортной ценой. В то время как для бизнеса планируется бесплатное участие в запуске и тестировании продукта, с дальнейшим переходом на гибкие условия.

На ближайшие 3 года была разработана финансовая модель, она представлена в таблице 1.

Таблица 1 – Финансовая модель

Показатель	Год 1	Год 2	Год 3
Пользователи (всего)	5000	20000	50000
b2c-подписчики	500	2000	10000
b2b-кофейни	10	40	200
Выручка b2c (млн руб.)	0,6	2,4	12
Выручка b2b (млн руб.)	0	0,5	2 – 3
Выручка всего (млн руб.)	0,6	2,9	14 – 15
Расходы (млн руб.)	0,79	1,35	2,72
Операционная прибыль (млн руб.)	-0,19	1,55	11,3

Согласно финансовой модели, предлагаемый к разработке проект выйдет на точку безубыточности в конце второго года, что подтверждает экономическую целесообразность разработки проекта.

2 Мобильное приложение, предназначенное для повышения осознанности потребления кофе

2.1 Анализ требований и постановка задач на разработку мобильного приложения осознанного потребления кофе

2.1.1 Анализ предметной области

Кофе остается одним из самых популярных и востребованных напитков во всем мире, и Россия не является исключением. В последние годы отечественный рынок кофе демонстрирует стабильный рост, в особенности формата «кофе с собой». Этому способствуют широкое распространение вендинговых автоматов и кофе-пойнтов, установленных в местах массового скопления людей. Несмотря на текущий прирост продаж ежегодно, аналитические данные свидетельствуют о потенциале дальнейшего увеличения объемов потребления кофе. По данным на 2022 год, среднедушевое потребление кофе в России составляет лишь 2,7 кг в год, что заметно ниже аналогичных показателей в ряде соседних государств. Например, в Казахстане этот показатель достигает 4 кг в год, в Беларуси – 8,5 кг в год, а в таких традиционно кофейных странах, как Финляндия и Эстония, 12 и 17 кг в год соответственно [5, 6].

Параллельно с ростом спроса на кофе, набирает силу тренд его осознанного потребления. Все больше людей задумываются о влиянии кофеина на их здоровье и самочувствие и стремятся к получению пользы от кофе (прилив бодрости и повышение эффективности) и минимизации возможных побочных эффектов (частые головные боли или тревожность). Доступность информации и популярность трекинговых мобильных приложений также стимулируют интерес к заботе о собственном здоровье и формируют привычку отслеживать свое самочувствие и управлять повседневной активностью. Одновременно становится всё более заметной тенденция к альтернативным вариантам напитка: помимо привычного кофе на коровьем молоке с сахаром доступны к покупке: кофе без сахара, кофе на растительном молоке, кофе с пониженным содержанием

кофеина и другие вариации. Также часть сетей поощряют экологичность, предлагая скидки за использование собственных кружек [7].

Учитывая масштабы потребления и описанные тенденции, разработка приложения для повышения осознанности потребления кофе крайне актуальна. Она не только вписывается в тренды заботы о здоровье и разумном потреблении, но и имеет широкую потенциальную аудиторию. Пользователи получают инструмент, позволяющий им балансировать удовольствием от кофе с личным благополучием и рациональностью.

2.1.2 Сравнительный анализ программных продуктов-аналогов

На рынке уже существуют приложения, позволяющие отслеживать потребление кофе или связанные с ним аспекты. Рассмотрим функционал некоторых из них, чтобы оценить текущий уровень конкуренции, выявить сильные и слабые стороны существующих продуктов, найти отсутствующие функции и сформировать ценностное предложение для целевой аудитории, опираясь на незакрытые потребности рынка. Такое исследование поможет более чётко определить позиционирование будущего продукта и сделать его максимально полезным и востребованным.

2.1.2.1 Анализ приложения CoffeeCup

Мобильное приложение CoffeeCup ориентировано прежде всего на количественный учёт потребления кофе. Пользователь может фиксировать каждую выпитую чашку, а также указывать её стоимость. После внесения новой записи приложение автоматически подсчитывает общее количество выпитого кофе и сумму расходов за выбранный период. А встроенная система наглядной статистики позволяет пользователю анализировать свои привычки и отслеживать предпочтения по разным видам напитков. Интерфейс CoffeeCup показан на рисунке 2.

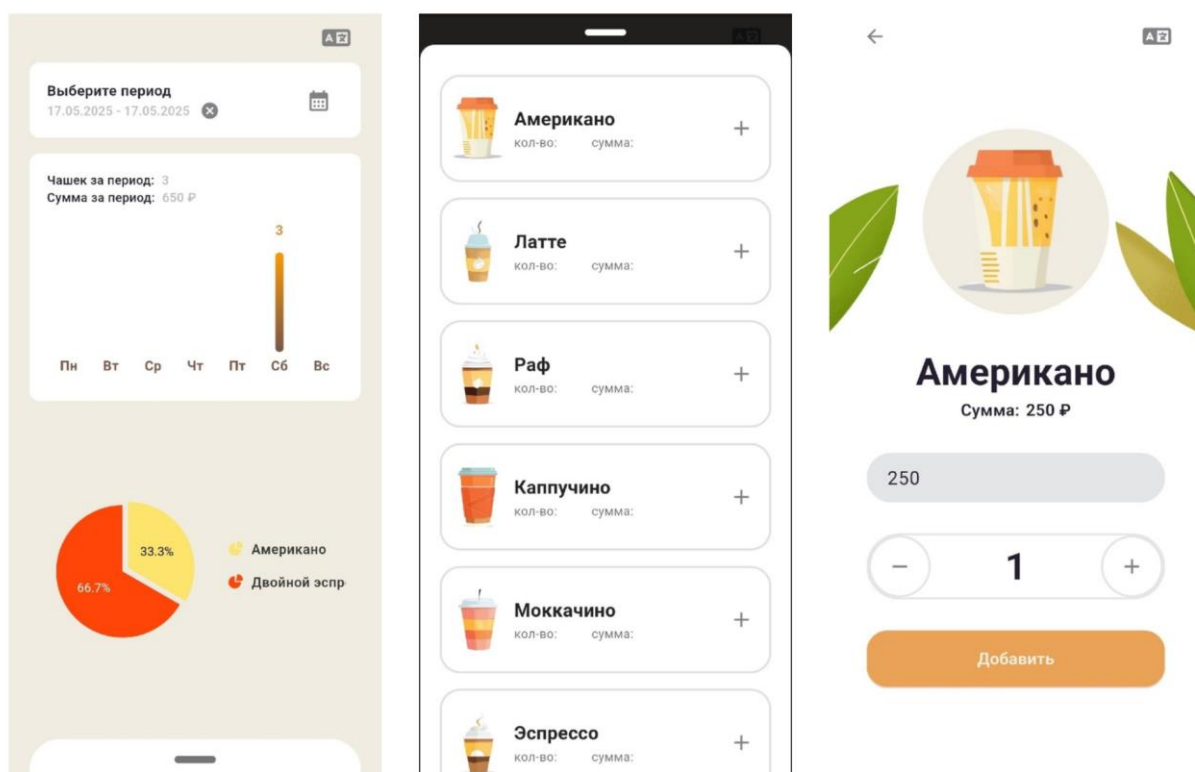


Рисунок 2 – Приложение CoffeeCup

Приложение распространяется бесплатно, доступен русский и английский языки, имеет простой и лаконичный дизайн. Оно работает офлайн и не требует подключения к аккаунту.

2.1.2.2 Анализ приложения CaffeInMe

Мобильное приложение CaffeInMe сфокусировано на детальном учете кофеина из различных источников. Оно позволяет учитывать потребление не только кофе, но и энергетиков, чая и других напитков, содержащих кофеин. Это дает возможность пользователю следить за текущим уровнем кофеина в организме и прогнозировать его значения через несколько часов. Помимо этого, приложение предоставляет статистику потребления за различные периоды и позволяет гибко персонализировать настройки: задавать дневную норму потребления, указывать желаемое время отхода ко сну и контролировать допустимый уровень кофеина вечером. Интерфейс CaffeInMe показан на рисунке 3.

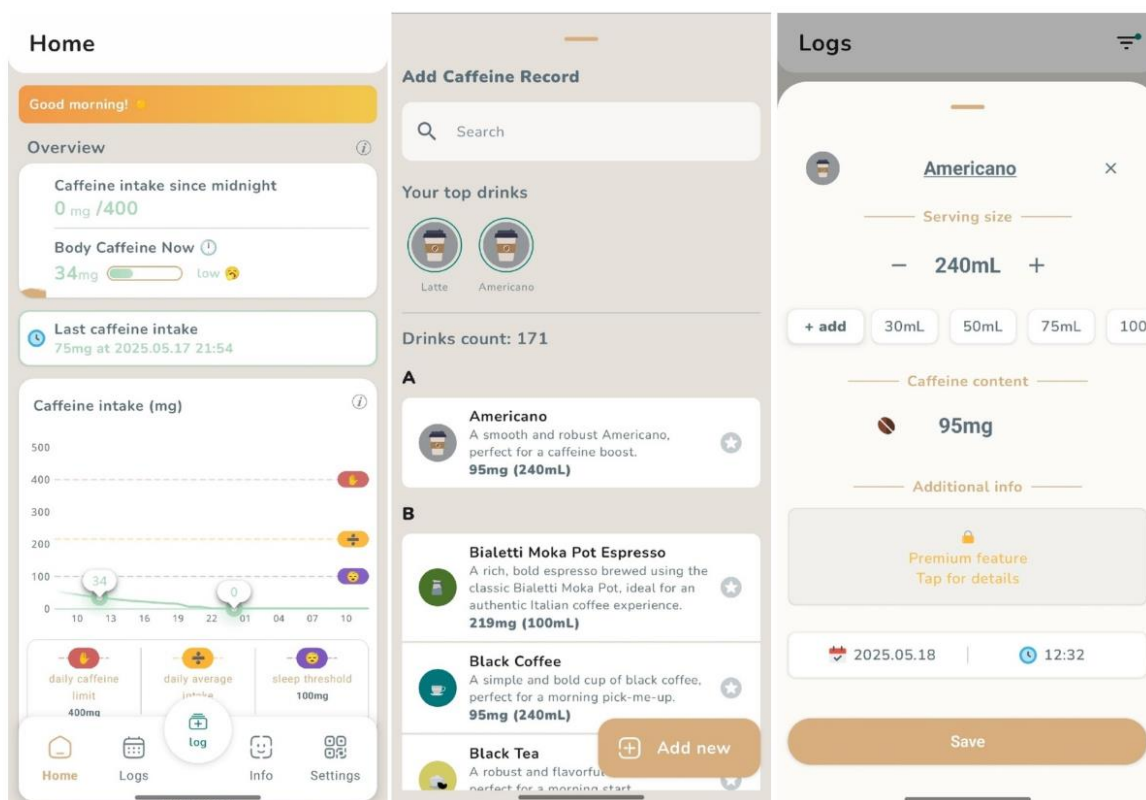


Рисунок 3 – Приложение CaffeInMe

Базовая версия приложения доступна бесплатно, однако обладает ограниченным функционалом и не поддерживает русский язык. В базовой версии доступны только обобщенные напитки, для которых значения кофеина могут значительно отличаться от реальных. Для добавления напитков конкретных производителей требуется приобретение подписки.

Платная версия приложения предоставляет ряд дополнительных преимуществ: возможность оставлять комментарии к каждой записи потребления кофеина, хранить резервные копии данных в облачном хранилище (при этом работа офлайн поддерживается) и добавлять виджеты на главный экран устройства.

2.1.2.3 Анализ приложения Caffeine Clock

Мобильное приложение Caffeine Clock предназначено для отслеживания текущего уровня кофеина в организме и контроля состояния бодрости.

Пользователь может учитывать различные источники кофеина, задавать целевой уровень его содержания и получать прогноз времени, необходимого для снижения текущего уровня до заданного. Приложение также позволяет просматривать историю потребления кофеина за текущий день и неделю. Интерфейс Caffeine Clock показан на рисунке 4.

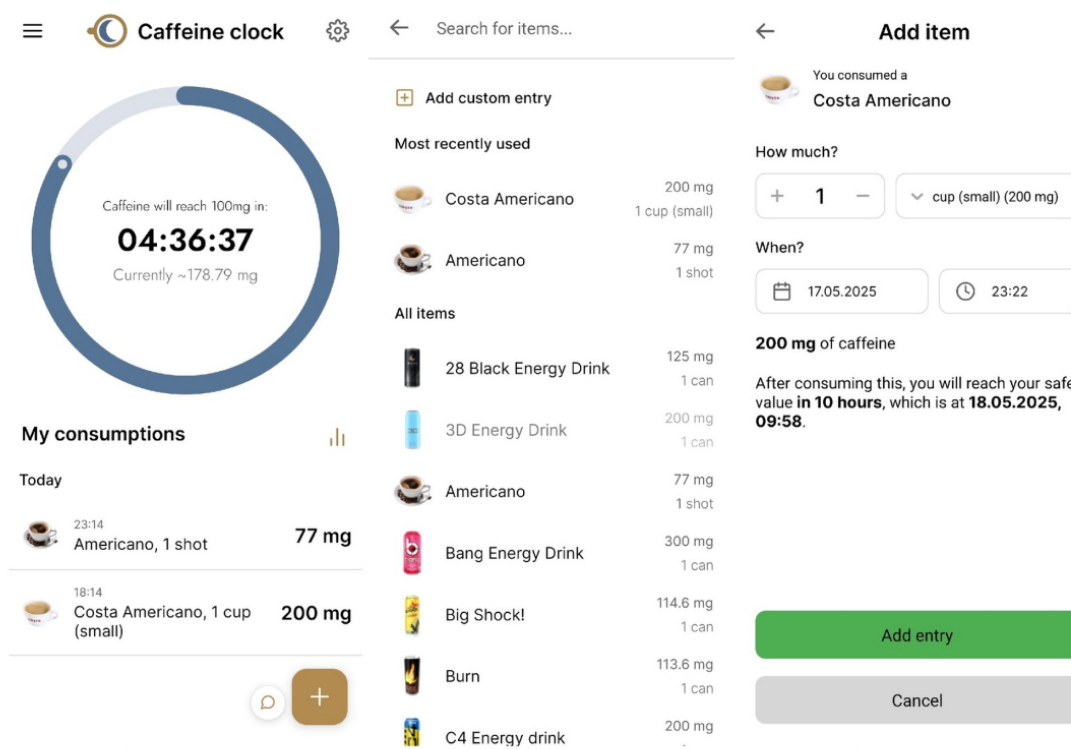


Рисунок 4 – Приложение Caffeine Clock

Приложение доступно бесплатно, отличается понятным интерфейсом, может работать офлайн и не требует регистрации аккаунта. Однако на данный момент доступен только английский язык.

2.1.2.4 Результаты проведенного анализа приложений

По результатам анализа аналогов была сформулирована сравнительная таблица ключевых характеристик приложений для отслеживания кофе и кофеина (таблица 2). Это позволило выявить сильные и слабые стороны существующих решений и определить потенциальные направления для развития нового продукта.

Таблица 2 – Ключевые характеристики аналогичных приложений

Характеристика	CoffeeCup	CaffeInMe	Caffeine Clock
Контроль кофеина	Нет	Есть	Есть
Контроль финансов	Есть	Нет	Нет
Поддерживаемые напитки	Только кофе	Любые источники кофеина	Любые источники кофеина
Возможность добавления своих напитков	Нет	Только в платной версии	Есть
Период просмотра статистики	Произвольный период	Фиксированные периоды (неделя, месяц, 3 месяца, год, все записи)	1 неделя

Таким образом, ни одно из рассмотренных приложений не объединяет все необходимые функции для комплексного и осознанного подхода к учету потребления кофе и кофеина, а также контроля финансовых и персональных аспектов. Каждый из аналогов реализует только часть из возможных сценариев использования, что оставляет возможность реализации конкурентоспособного приложения.

2.1.3 Анализ требований мобильного приложения

2.1.3.1 Анализ функциональных требований

Анализ функциональных требований позволяет определить, что именно должно выполнять программное обеспечение с точки зрения пользователя, что обеспечивает соответствие продукта ожиданиям целевой аудитории [8]. Так,

разрабатываемое приложение для осознанного потребления кофе должно предоставлять следующий функционал:

- добавление персонализированных кофейных напитков;
- учет количества и видов употребленного кофе;
- учет финансовых затрат на покупку напитка;
- учет настроения как до, так и после употребления напитка;
- анализ потребления за день (количество выпитых чашек, полученного кофеина, потраченных денег и преобладающего эмоционального состояния);
- просмотр статистики за выбранный промежуток времени;
- настройку персонализации.

Для наглядного описания поведения разрабатываемого мобильного приложения была составлена диаграмма вариантов использования (рисунок 5), на которой изображен актер и выполняемые им прецеденты [9].

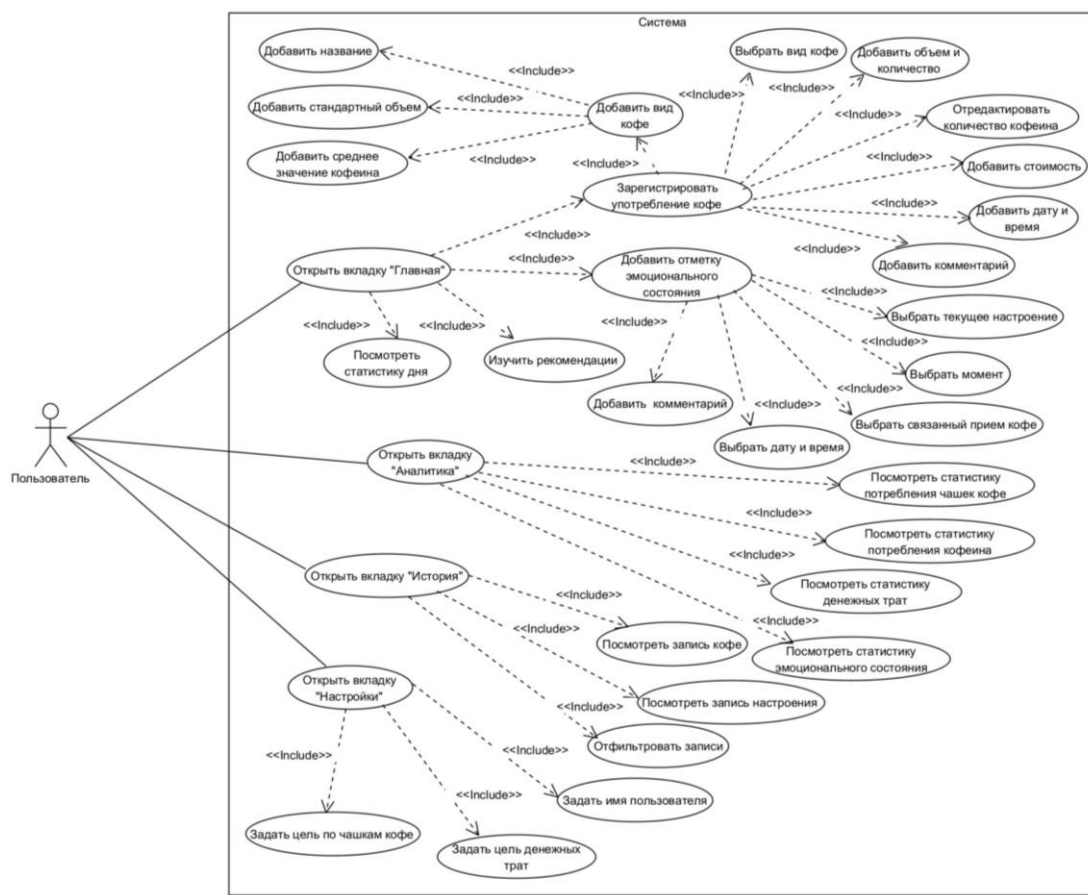


Рисунок 5 – Диаграмма вариантов использования

Для пяти вариантов использования были описаны их спецификации. Спецификация прецедента «Изучить рекомендации» представлена на рисунке 6.

Прецедент: Изучить рекомендации
ID: 1
Краткое описание: Пользователь изучает персональные рекомендации
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Пользователь открыл вкладку «Главная»
Основной поток: <ol style="list-style-type: none"> 1) Система подготавливает персональные рекомендации; 2) Система отображает список рекомендаций; 3) Пользователь изучает рекомендации системы.
Постусловия: Пользователь ознакомлен с актуальными рекомендациями
Альтернативные потоки: Нет.

Рисунок 6 – Спецификация прецедента «Изучить рекомендации»

Спецификация прецедента «Выбрать вид кофе» представлена на рисунке 7.

Прецедент: Выбрать вид кофе
ID: 2
Краткое описание: Пользователь выбирает вид выпитого кофе
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: <ol style="list-style-type: none"> 1. Существует как минимум 1 вид кофе 2. Пользователь начал добавлять употребление кофе
Основной поток: <ol style="list-style-type: none"> 1) Система запрашивает все имеющиеся виды кофе из базы данных; 2) Система отображает записи; 3) Пользователь выбирает нужную запись; 4) Система сохраняет выбранный вид кофе.
Постусловия: Пользователь выбрал вид выпитого кофе
Альтернативные потоки: Нет.

Рисунок 7 – Спецификация прецедента «Выбрать вид кофе»

Спецификация прецедента «Просмотреть статистику потребления чашек кофе» представлена на рисунке 8.

Прецендент: Просмотреть статистику потребления чашек кофе
ID: 3
Краткое описание: Пользователь просматривает диаграмму количества выпитых чашек кофе за последние 30 дней
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Пользователь открыл вкладку «Аналитика»
Основной поток: <ol style="list-style-type: none"> 1) Система из базы данных запрашивает записи потребления кофе за последние 30 дней; 2) Система подготавливает данные; 3) Система отображает диаграмму потребления и текстовое сообщение о среднем потреблении за последние 30 дней; 4) Пользователь изучает информацию.
Постусловия: Пользователь изучил статистику потребления чашек кофе
Альтернативные потоки: Нет.

Рисунок 8 – Спецификация прецендента «Просмотреть статистику потребления чашек кофе»

Спецификация прецендента «Посмотреть запись кофе» представлена на рисунке 9.

Прецендент: Посмотреть запись кофе
ID: 4
Краткое описание: Пользователь просматривает подробную информацию о конкретной записи кофе
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Пользователь открыл вкладку «История» и видит список записей кофе
Основной поток: <ol style="list-style-type: none"> 1) Пользователь нажимает на интересующую его запись кофе в списке истории; 2) Система делает запрос в базу данных для получения всей записи потребления; 3) Система отображает подробную информацию о выбранной записи; 4) Пользователь изучает информацию.
Постусловия: Пользователь ознакомился с подробностями выбранной записи
Альтернативные потоки: Нет.

Рисунок 9 – Спецификация прецендента «Посмотреть запись кофе»

Спецификация прецедента «Задать цель денежных трат» представлена на рисунке 10.

Прецедент: Задать цель денежных трат
ID: 5
Краткое описание: Пользователь задает целевую сумму трат
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Пользователь начал редактировать настройки
Основной поток: 1) Пользователь вводит новую цель по тратам; 2) Система проверяет значение; 3) Система возвращает результат обновления.
Постусловия: Цель изменена или обработана ошибка
Альтернативные потоки: Нет.

Рисунок 10 – Спецификация прецедента «Задать цель денежных трат»

В ходе изучения функциональных требований были выделены прецеденты работы системы и создана диаграмма вариантов использования.

2.1.3.2 Анализ нефункциональных требований

Нефункциональные требования в свою очередь отражают качественные характеристики будущего ПО. Они более абстрактны и, как правило, с точки зрения пользователя их невозможно, либо очень сложно проверить [8]. Так, к разрабатываемому мобильному приложению для осознанного потребления кофе были выдвинуты следующие нефункциональные требования:

- корректная работа в режиме офлайн;
- хранение данных непосредственно на устройстве;
- интуитивно понятный и удобный пользовательский интерфейс;
- высокая степень надежности работы;
- совместимость с различными устройствами;
- адаптируемость пользовательского интерфейса;
- сопровождение переходов между экранами плавной анимацией.

Также к нефункциональным требованиям можно отнести требования к интерфейсу, которые необходимы для обеспечения удобства и понятности взаимодействия пользователя с ПО. К разрабатываемому мобильному приложению были выдвинуты следующие требования к интерфейсу:

- простота навигации;
- интуитивный ввод данных;
- адаптивность дизайна;
- единообразие элементов интерфейса;
- доступность информации.

Устройства, на которых планируется запуск должны иметь следующие характеристики:

- операционная система Android 9 (API 28) и старше;
- минимум 4-ядерный процессор с частотой 1,4 ГГц и выше;
- минимум 2 Гб оперативной памяти (рекомендуется 4 Гб);
- наличие как минимум 100 Мб свободного места в хранилище;
- дисплей как минимум с разрешением HD (рекомендуется Full HD и выше) и соотношением «16:9» (рекомендуется «20:9»).

На данном этапе был проведен анализ предметной области, что позволило разработать функциональные и нефункциональные требования к приложению.

2.1.4 Постановка цели и задачи на разработку мобильного приложения

Целью данной выпускной квалификационной работы (ВКР) является разработка мобильного приложения «Coffies», которое позволит пользователям осознанно подойти к вопросу потребления кофе и его влиянию на их состояние.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- проанализировать предметную область, разработать требования и выявить аналогии;

- произвести оценку бюджета на разработку мобильного приложения;
- спроектировать базу данных и пользовательский интерфейс;
- разработать android-приложение;
- произвести тестирование и оценить качество кода с использованием различных метрик.

При решении поставленных задач необходимо учитывать простоту взаимодействия пользователя с приложением, что позволит ему быстро и удобно отслеживать потребление кофе, расход денежных средств и своего эмоционального состояния.

2.1.5 Анализ разработки и оценка бюджета мобильного приложения

2.1.5.1 Роли и задачи участников проекта

Для успешной реализации мобильного приложения для осознанного потребления кофе требуется участие следующих специалистов:

- менеджер проекта, который будет отвечать за составление графика и бюджета, подготовку документации, контроля качества и координацию команды;
- аналитик, который будет изучать предметную область, формулировать требования к продукту и вести техническую документацию;
- проектировщик, который будет разрабатывать общую структуру программного обеспечения (ПО), определять используемые технологии, а также оптимизировать используемые алгоритмы;
- android-разработчик, который будет реализовывать на практике архитектурные решения, бизнес логику и пользовательский интерфейс;
- тестировщик, который будет проверять корректность работы ПО, готовить отчеты и рекомендации для исправления ошибок [10].

Привлечение указанных специалистов позволит организовать эффективную работу над проектом, обеспечить его соответствие техническим требованиям и ожиданиям пользователей.

2.1.5.2 Оценка бюджета на разработку мобильного приложения

При планировании сроков разработки и оценки расходов на проект использовался программный продукт Microsoft Project. На первоначальном этапе был разработан детализированный список задач (рисунок 11), на основании которого была составлена диаграмма Ганта (рисунок 12).

Название задачи	Длительность	Начало	Окончание	Предш	Названия ресурсов
1 - Начало реализации проекта	123 дней	Пн 02.12.24	Ср 21.05.25		
2 Закупка оборудования и ПО	2 дней	Пн 02.12.24	Вт 03.12.24		Принтер[1 Устройство];Компьютер[1 Устройство];MS Windows 11[1];MS Office Professional[1];Android Studio[1];Менеджер проекта
3 - Разработка бизнес-плана	14 дней	Ср 04.12.24	Пн 23.12.24		
4 Описание резюме проекта	2 дней	Ср 04.12.24	Чт 05.12.24	2	Менеджер проекта;Электроэнергия[14 КВт*ч]
5 Описание инновационной идеи	2 дней	Пт 06.12.24	Пн 09.12.24	4	Менеджер проекта;Электроэнергия[14 КВт*ч]
6 Описание продукции	2 дней	Вт 10.12.24	Ср 11.12.24	5	Менеджер проекта;Электроэнергия[14 КВт*ч]
7 Анализ рынка	3 дней	Чт 12.12.24	Пн 16.12.24	6	Менеджер проекта;Электроэнергия[21 КВт*ч]
8 Составление маркетингового плана	3 дней	Вт 17.12.24	Чт 19.12.24	7	Менеджер проекта;Электроэнергия[21 КВт*ч]
9 Составление финансового плана	2 дней	Пт 20.12.24	Пн 23.12.24	8	Менеджер проекта;Электроэнергия[14 КВт*ч]
10 Завершение разработки бизнес-плана	0 дней	Пн 23.12.24	Пн 23.12.24	9	
11 - Разработка мобильного приложения для с	107 дней	Вт 24.12.24	Ср 21.05.25		
12 - Анализ разработки программных сре	13 дней	Вт 24.12.24	Чт 09.01.25		
13 Анализ предметной области	3 дней	Вт 24.12.24	Чт 26.12.24	12	Аналитик[50%];Менеджер проекта[50%];Электроэнергия[21 КВт*ч]
14 Технический анализ конкурентов	2 дней	Пт 27.12.24	Пн 30.12.24	13	Аналитик[50%];Менеджер проекта[50%];Электроэнергия[14 КВт*ч]
15 Постановка задачи	1 день	Вт 31.12.24	Вт 31.12.24	14	Аналитик[50%];Менеджер проекта[25%];Проектировщик[25%];Электроэнергия[7 КВт*ч]
16 Анализ требований к программным	3 дней	Ср 01.01.25	Пт 03.01.25	15	Аналитик[50%];Менеджер проекта[25%];Проектировщик[25%];Электроэнергия[21 КВт*ч];Бумага[1 Пачка]
17 Выбор технологий и средств разра	4 дней	Пн 06.01.25	Чт 09.01.25	16	Аналитик[25%];Менеджер проекта[25%];Проектировщик[50%];Электроэнергия[14 КВт*ч]
18 Завершение анализа разработки	0 дней	Чт 09.01.25	Чт 09.01.25	17	
19 - Проектирование программных средс	8 дней	Пт 10.01.25	Вт 21.01.25		
20 Проектирование архитектуры моби	3 дней	Пт 10.01.25	Вт 14.01.25	19	Проектировщик;Электроэнергия[21 КВт*ч]
21 Проектирование структуры базы да	2 дней	Ср 15.01.25	Чт 16.01.25	20	Проектировщик[75%];Электроэнергия[14 КВт*ч];Аналитик[25%]
22 Проектирование интерфейса мобил	3 дней	Пт 17.01.25	Вт 21.01.25	21	Проектировщик;Электроэнергия[21 КВт*ч]
23 Завершение проектирования	0 дней	Вт 21.01.25	Вт 21.01.25	22	
24 - Реализация программных средств	70 дней	Ср 22.01.25	Вт 29.04.25		
25 Реализация модели данных	10 дней	Ср 22.01.25	Вт 04.02.25	24	Android-разработчик;Электроэнергия[70 КВт*ч]
26 Реализация мобильного приложения	60 дней	Ср 05.02.25	Вт 29.04.25	25	Android-разработчик;Электроэнергия[420 КВт*ч]
27 Завершение реализации	0 дней	Вт 29.04.25	Вт 29.04.25	26	
28 - Тестирование программных средств	9 дней	Ср 30.04.25	Пн 12.05.25		
29 Функциональное тестирование	3 дней	Ср 30.04.25	Пт 02.05.25	28	Тестировщик;Электроэнергия[21 КВт*ч]
30 Оценка метрик покрытия операторс	2 дней	Пн 05.05.25	Вт 06.05.25	29	Тестировщик;Электроэнергия[14 КВт*ч]
31 Оценка метрик качества кода	2 дней	Ср 07.05.25	Чт 08.05.25	30	Тестировщик;Электроэнергия[14 КВт*ч]
32 Модульное тестирование	2 дней	Пт 09.05.25	Пн 12.05.25	31	Тестировщик;Электроэнергия[14 КВт*ч]
33 Завершение тестирования	0 дней	Пн 12.05.25	Пн 12.05.25	32	
34 Составление и оформление документа	7 дней	Вт 13.05.25	Ср 21.05.25	33	Менеджер проекта;Бумага[2 Пачка];Электроэнергия[49 КВт*ч]
35 Завершение разработки мобильного приложени	0 дней	Ср 21.05.25	Ср 21.05.25	34	
36 Завершение реализации проекта	0 дней	Ср 21.05.25	Ср 21.05.25	35	

Рисунок 11 – Список задач проекта

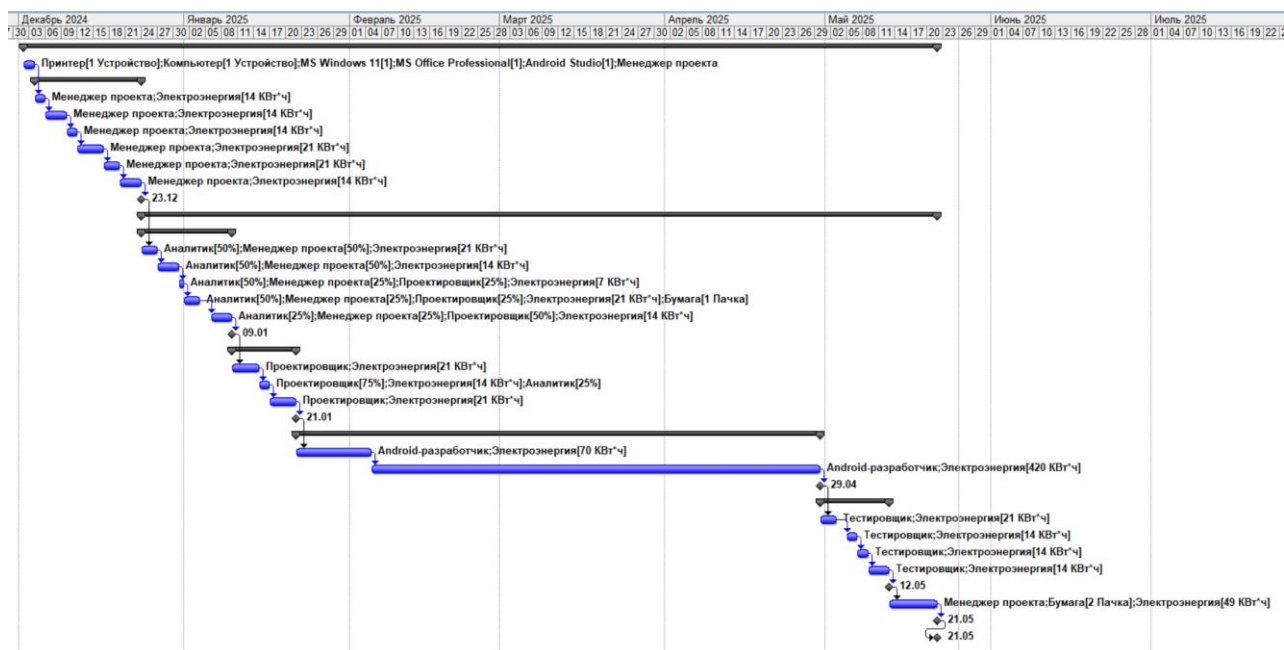


Рисунок 12 – Диаграмма Ганта

Из общего отчета проекта (рисунок 13), видно, что реализация проекта займет 123 рабочих дня (с 2 декабря 2024 года по 21 мая 2025 года).

Даты			
Начало:	Пн 02.12.24	Окончание:	Ср 21.05.25
Баз. начало:	НД	Баз. окончание:	НД
Факт. начало:	НД	Факт. окончание:	НД
Отклон. начала:	0 дней	Отклон. окончания:	0 дней
Длительность			
Текущий план:	123 дней	Оставшаяся:	123 дней
Базовый план:	0 дней?	Фактическая:	0 дней
Отклонение:	123 дней	Процент завершения:	0%
Трудозатраты			
Текущий план:	965,12 ч	Оставшиеся:	965,12 ч
Базовый план:	0 ч	Фактические:	0 ч
Отклонение:	965,12 ч	Процент завершения:	0%
Затраты			
Текущий план:	347 684,13 Р	Оставшиеся:	347 684,13 Р
Базовый план:	0,00 Р	Фактические:	0,00 Р
Отклонение:	347 684,13 Р		
Состояние задачи		Состояние ресурса	
Еще не начатые задачи	36	Трудовые ресурсы:	5
Выполняющиеся задачи	0	Трудовые ресурсы с прев. дос:	0
Завершенные задачи:	0	Материальные ресурсы:	7
Итого задач:	36	Итого ресурсов:	12

Рисунок 13 – Общий отчет проекта

Стоимость разработки проекта включает в себя затраты как на заработную плату исполнителей, так и на материальные ресурсы. Как видно из рисунка 14, в котором представлены ставки заработной платы, а также из распределения задач по исполнителям (рисунок 15), наиболее загружен Android-разработчик (560 часов). Это, в свою очередь, делает реализацию мобильного приложения наиболее затратной задачей.

Название ресурса	Тип	Стандартная ставка	Ставка сверхурочных	Начисление
Менеджер проекта	Трудовой	250,00 Р/ч	400,00 Р/ч	Пропорциональное
Аналитик	Трудовой	200,00 Р/ч	350,00 Р/ч	Пропорциональное
Проектировщик	Трудовой	300,00 Р/ч	450,00 Р/ч	Пропорциональное
Android-разработчик	Трудовой	275,00 Р/ч	450,00 Р/ч	Пропорциональное
Тестировщик	Трудовой	200,00 Р/ч	350,00 Р/ч	Пропорциональное

Рисунок 14 – Ставка заработной платы

Менеджер проекта		216 ч				
Ид.	Название задачи	Единицы	Трудозатраты	Задержка	Начало	Окончание
5	Описание инновационной идеи	100%	16ч	0 дней	Пт 06.12.24	Пн 09.12.24
6	Описание продукции	100%	16ч	0 дней	Вт 10.12.24	Ср 11.12.24
7	Анализ рынка	100%	24ч	0 дней	Чт 12.12.24	Пн 16.12.24
8	Составление маркетингового плана	100%	24ч	0 дней	Вт 17.12.24	Чт 19.12.24
9	Составление финансового плана	100%	16ч	0 дней	Пт 20.12.24	Пн 23.12.24
2	Закупка оборудования и ПО	100%	16ч	0 дней	Пн 02.12.24	Вт 03.12.24
4	Описание резюме проекта	100%	16ч	0 дней	Ср 04.12.24	Чт 05.12.24
13	Анализ предметной области	50%	12ч	0 дней	Вт 24.12.24	Чт 26.12.24
14	Технический анализ конкурентов	50%	8ч	0 дней	Пт 27.12.24	Пн 30.12.24
15	Постановка задачи	25%	2ч	0 дней	Вт 31.12.24	Вт 31.12.24
16	Анализ требований к программным	25%	6ч	0 дней	Ср 01.01.25	Пт 03.01.25
17	Выбор технологии и средств разраб	25%	4ч	0 дней	Пн 06.01.25	Вт 07.01.25
34	Составление и оформление докумен	100%	56ч	0 дней	Вт 13.05.25	Ср 21.05.25
Аналитик		45,92 ч				
Ид.	Название задачи	Единицы	Трудозатраты	Задержка	Начало	Окончание
14	Технический анализ конкурентов	50%	8ч	0 дней	Пт 27.12.24	Пн 30.12.24
16	Анализ требований к программным	50%	12ч	0 дней	Ср 01.01.25	Пт 03.01.25
17	Выбор технологии и средств разраб	25%	8ч	0 дней	Пн 06.01.25	Чт 09.01.25
13	Анализ предметной области	50%	12ч	0 дней	Вт 24.12.24	Чт 26.12.24
15	Постановка задачи	50%	4ч	0 дней	Вт 31.12.24	Вт 31.12.24
21	Проектирование структуры базы да	30%	1,92ч	0 дней	Ср 15.01.25	Ср 15.01.25
Проектировщик		71,2 ч				
Ид.	Название задачи	Единицы	Трудозатраты	Задержка	Начало	Окончание
21	Проектирование структуры базы данных	70%	11,2ч	0 дней	Ср 15.01.25	Чт 16.01.25
22	Проектирование интерфейса мобильного	100%	24ч	0 дней	Пт 17.01.25	Вт 21.01.25
15	Постановка задачи	25%	2ч	0 дней	Вт 31.12.24	Вт 31.12.24
16	Анализ требований к программным средст	25%	6ч	0 дней	Ср 01.01.25	Пт 03.01.25
17	Выбор технологии и средств разработки	50%	4ч	0 дней	Пн 06.01.25	Пн 06.01.25
20	Проектирование архитектуры мобильного	100%	24ч	0 дней	Пт 10.01.25	Вт 14.01.25
Android-разработчик		560 ч				
Ид.	Название задачи	Единицы	Трудозатраты	Задержка	Начало	Окончание
25	Реализация модели данных	100%	80ч	0 дней	Ср 22.01.25	Вт 04.02.25
26	Реализация мобильного прил	100%	480ч	0 дней	Ср 05.02.25	Вт 29.04.25
Тестировщик		72 ч				
Ид.	Название задачи	Единицы	Трудозатраты	Задержка	Начало	Окончание
30	Оценка метрик покрытия оператором	100%	16ч	0 дней	Пн 05.05.25	Вт 06.05.25
31	Оценка метрик качества кода	100%	16ч	0 дней	Ср 07.05.25	Чт 08.05.25
32	Модульное тестирование	100%	16ч	0 дней	Пт 09.05.25	Пн 12.05.25
29	Функциональное тестирование	100%	24ч	0 дней	Ср 30.04.25	Пт 02.05.25

Рисунок 15 – Распределение задач по исполнителям

Также происходила закупка оборудования и ПО, список стоимостей использованных материальных ресурсов представлен на рисунке 16.

Название ресурса	Тип	Стандартная ставка	Единицы измерения материалов
Компьютер	Материальный	50 000,00 Р	Устройство
Принтер	Материальный	10 000,00 Р	Устройство
MS Windows 11	Материальный	8 000,00 Р	
Android Studio	Материальный	0,00 Р	
MS Office Professional	Материальный	22 000,00 Р	
Электроснабжение	Материальный	4,61 Р	КВт*ч
Бумага	Материальный	300,00 Р	Пачка

Рисунок 16 – Материальные ресурсы

Общая стоимость разработки проекта составила 347684,13 рублей, денежные траты для каждой задачи представлены на рисунке 17.

Ид.	Название задачи	Фиксированные затраты	Начисление фикс. затрат	Общие затраты
26	Реализация мобильного приложения	0,00 Р	Пропорциональное	133 936,20 Р
2	Закупка оборудования и ПО	0,00 Р	Пропорциональное	94 000,00 Р
25	Реализация модели данных	0,00 Р	Пропорциональное	22 322,70 Р
34	Составление и оформление докумен	0,00 Р	Пропорциональное	14 825,89 Р
20	Проектирование архитектуры мобил	0,00 Р	Пропорциональное	7 296,81 Р
22	Проектирование интерфейса мобил	0,00 Р	Пропорциональное	7 296,81 Р
7	Анализ рынка	0,00 Р	Пропорциональное	6 096,81 Р
8	Составление маркетингового плана	0,00 Р	Пропорциональное	6 096,81 Р
16	Анализ требований к программным	0,00 Р	Пропорциональное	6 096,81 Р
13	Анализ предметной области	0,00 Р	Пропорциональное	5 496,81 Р
29	Функциональное тестирование	0,00 Р	Пропорциональное	4 896,81 Р
4	Описание резюме проекта	0,00 Р	Пропорциональное	4 064,54 Р
5	Описание инновационной идеи	0,00 Р	Пропорциональное	4 064,54 Р
6	Описание продукции	0,00 Р	Пропорциональное	4 064,54 Р
9	Составление финансового плана	0,00 Р	Пропорциональное	4 064,54 Р
17	Выбор технологии и средств разраб	0,00 Р	Пропорциональное	3 864,54 Р
21	Проектирование структуры базы да	0,00 Р	Пропорциональное	3 808,54 Р
14	Технический анализ конкурентов	0,00 Р	Пропорциональное	3 664,54 Р
30	Оценка метрик покрытия операторо	0,00 Р	Пропорциональное	3 264,54 Р
31	Оценка метрик качества кода	0,00 Р	Пропорциональное	3 264,54 Р
32	Модульное тестирование	0,00 Р	Пропорциональное	3 264,54 Р
15	Постановка задачи	0,00 Р	Пропорциональное	1 932,27 Р
10	Завершение разработки бизнес-пла	0,00 Р	Пропорциональное	0,00 Р
18	Завершение анализа разработки	0,00 Р	Пропорциональное	0,00 Р
23	Завершение проектирования	0,00 Р	Пропорциональное	0,00 Р
27	Завершение реализации	0,00 Р	Пропорциональное	0,00 Р
33	Завершение тестирования	0,00 Р	Пропорциональное	0,00 Р
35	Завершение разработки мобильного	0,00 Р	Пропорциональное	0,00 Р
36	Завершение реализации проекта	0,00 Р	Пропорциональное	0,00 Р
		0,00 Р		347 684,13 Р

Рисунок 17 – Общие затраты на проект

Длительность проекта соответствует проекту по разработке мобильного приложения средней сложности. Наибольшую нагрузку и основные расходы несет Android-разработчик, что подчеркивает важность проектирования на ранних этапах.

2.2 Проектирование программных средств для разработки мобильного приложения для осознанного потребления кофе

2.2.1 Проектирование архитектуры мобильного приложения

На этапе проектирования архитектуры мобильного приложения предстоит выбрать наиболее подходящие свойства архитектуры, соответствующие особенностям проанализированной предметной области: разработка будет содержать большое количество графических элементов и использоваться для создания стартап-проекта, что может привести к активному добавлению нового функционала и изменению уже существующего. В связи с этим, к требуемым свойствам относятся:

- расширяемость;

- модульность;
- производительность;
- простота разработки;
- надежность;
- тестируемость;
- удобство и простота использования [11].

В таблице 3 представлено проявление каждого из свойств в наиболее популярных архитектурах мобильных приложений: Model-View-Controller (MVC), Model-View-Presenter (MVP), Model-View-ViewModel (MVVM), Model-View-Intent (MVI) [12 – 15].

Таблица 3 – Сравнение архитектур мобильных приложений

№	MVC	MVP	MVVM	MVI
Расширяемость	При увеличении кодовой базы сложность поддержки возрастает экспоненциально	За счет разделения логики, легче добавлять новые функции	За счет разделения логики, легче добавлять новые функции	Единый поток данных облегчает рост кода по мере усложнения проекта
Модульность	UI и бизнес-логика тесно связаны между собой	UI и бизнес-логика слабо связаны между собой	UI отделен от бизнес-логики	UI лишь отражает состояние, бизнес-логика обособлена
Производительность	Тесное взаимодействие UI и логики замедляет отображение	UI отделен от фоновых процессов	UI отделен от фоновых процессов	Возможны задержки вследствие перегрузки потоков действий и переходов
Простота разработки	Легко понять и использовать для разработки	Требуется знание шаблонной структуры кода	Требуется понимание работы архитектурных компонентов	Требуется знания реактивного программирования
Надежность	Смешение логики приводит к множеству ошибок и трудной отладке	Разделение логики делает приложение более предсказуемым	Учитывается также жизненный цикл приложения, что позволяет получить большую стабильность	За счет единого неизменяемого состояния и одностороннего потока данных поведение легко прогнозировать

Продолжение таблицы 3

№	MVC	MVP	MVVM	MVI
Тестируемость	Тесно переплетенный код трудно тестировать	Бизнес-логику можно тестировать без использования UI	Для тестирования достаточно выдачу данных при взаимодействии	Четкий поток позволяет изолированно тестировать слои
Удобство и простота использования	Удобно и быстро в начале, однако в долгосрочной перспективе становится очень сложно поддерживать	Приходится писать много шаблонного кода, однако это позволяет поддерживать структуру для долгосрочной поддержки. Хорошо задокументирован	Для старта требуется понимание основ. Имеет множество учебных материалов и готовых решений.	Требуется глубокое понимание реактивных библиотек. Требуется самостоятельная реализация потока данных. Меньше поддерживается сообществом

В таблице 4 представлено распределение оценок свойств по шкале, от 1 до 4, где 1 – минимум, а 4 – максимум.

Таблица 4 – Оценки сравнения архитектур

№	MVC	MVP	MVVM	MVI
Расширяемость	1	3	3	4
Модульность	1	3	3	4
Производительность	2	4	4	3
Простота разработки	4	2	2	1
Надежность	1	2	3	4
Тестируемость	1	3	4	4
Удобство и простота использования	3	3	3	1
Всего	13	20	22	21

В результате суммирования баллов за каждый критерий, архитектуры MVP, MVVM и MVI показали похожие результаты. Однако соблюдение баланса

между простотой и мощностью, а также официальная поддержка со стороны компании Google стали ключевыми факторами для выбора: разработка мобильного приложения будет идти по архитектуре Model-View-ViewModel (MVVM).

2.2.2 Проектирование базы данных мобильного приложения

База данных является ключевым компонентом приложения, предназначенного для хранения, структурирования и обеспечения эффективного доступа к данным. Формирование логичной и структурированной системы хранения данных происходит в ходе поэтапного проектирования базы данных.

Всего выделяют четыре этапа:

- анализ требований;
- концептуальное проектирование;
- логическое проектирование;
- физическое проектирование.

Поскольку анализ требований уже был проведен, рассмотрим только этапы с проектированием.

В ходе концептуального проектирования создается черновая модель базы данных, в которой отражаются основные сущности предметной области и связи между ними. На этапе логического проектирования полученная ранее концептуальная модель преобразуется в логическую модель, совместимую с системой управления базой данных (СУБД). Также на этом этапе производится нормализация данных, с целью устранения избыточности и повышения эффективности хранения. На финальном этапе выбираются конкретные технологии и механизмы хранения данных, задаются параметры таблиц и индексов и учитываются требования к безопасности. Дополнительно разрабатываются схемы резервного копирования, процедуры миграции данных и механизмы контроля целостности, чтобы обеспечить устойчивость и надежность работы приложения в различных сценариях эксплуатации [16].

2.2.2.1 Проектирование концептуальной модели базы данных

В ходе анализа предметной области мобильного приложения для осознанного потребления кофе были выявлены следующие основные сущности:

- виды кофе, для описания видов кофе с указанием объема и количества кофеина;
- записи кофе, для сохранения пользователем приема кофе;
- записи настроения, для отслеживания пользователя своего настроения;
- настройки, для сохранения настроек и целей пользователя.

Концептуальная модель базы данных, с представленными выше сущностями, представлена на рисунке 18. Такая структура обеспечивает полноту учета данных и гибкость расширения функциональности в будущем.

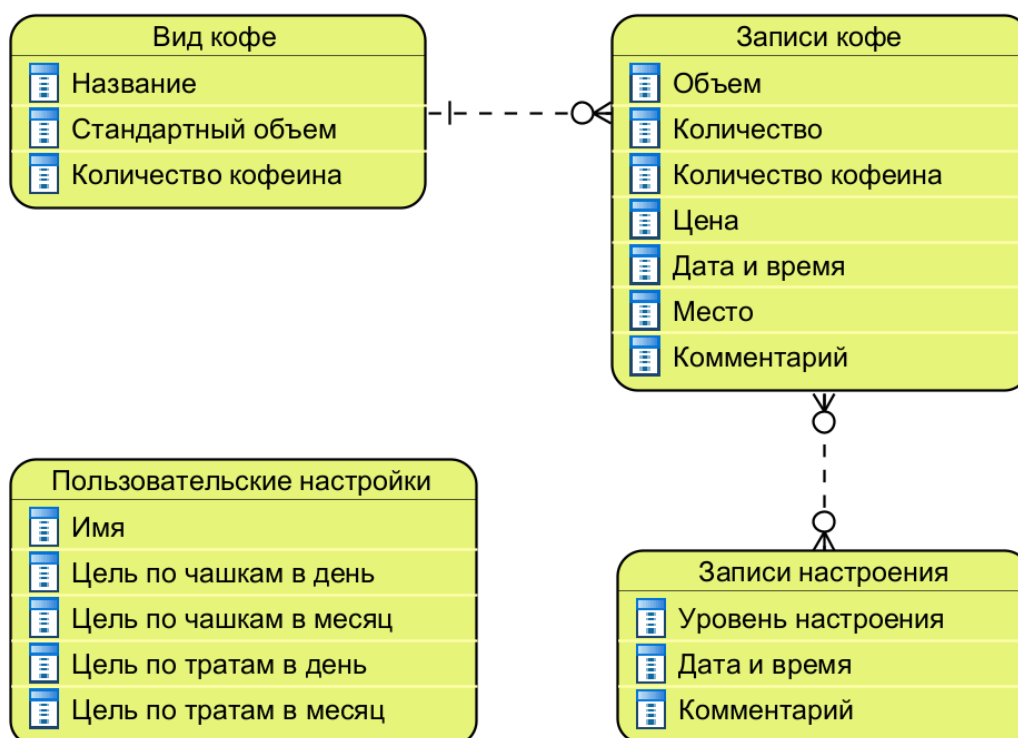


Рисунок 18 – Концептуальная модель базы данных

В процессе дальнейшего проектирования в эту модель могут быть внесены изменения для полного соответствия предметной области.

2.2.2.2 Проектирование логической модели базы данных

На основе концептуальной модели базы данных была создана логическая модель базы данных, представленная на рисунке 19.

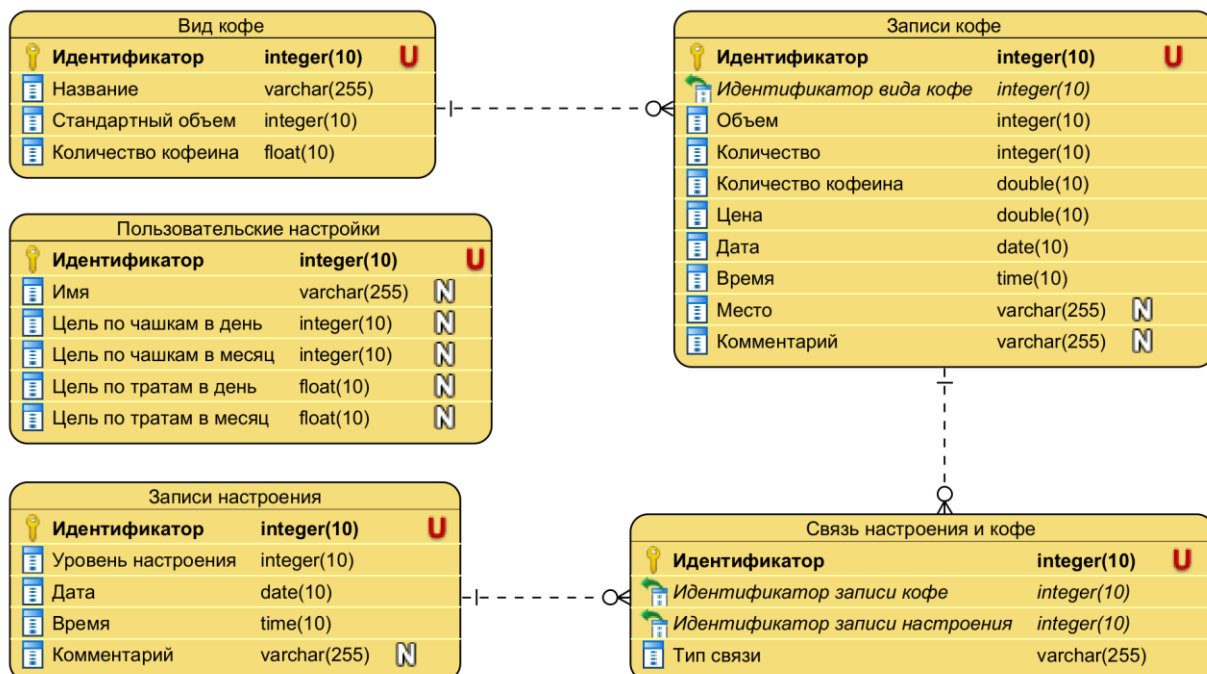


Рисунок 19 – Логическая модель базы данных

Связь «многие-ко-многим» между сущностями «Записи кофе» и «Записи настройки» была реализована с использованием дополнительной таблицы «Связь настройки и кофе». Для каждой из таблиц также были добавлены идентификаторы, которые будут являться первичными ключами, и проанализированы ограничения для каждого из атрибутов.

Данная логическая модель находится в третьей нормальной форме, поскольку между не ключевыми атрибутами нет транзитивных зависимостей, а все не ключевые атрибуты зависят только от первичного ключа, который является атомарным.

2.2.2.3 Проектирование физической модели базы данных

В заключение, была спроектирована физическая модель базы данных мобильного приложения для осознанного потребления кофе (рисунок 20).

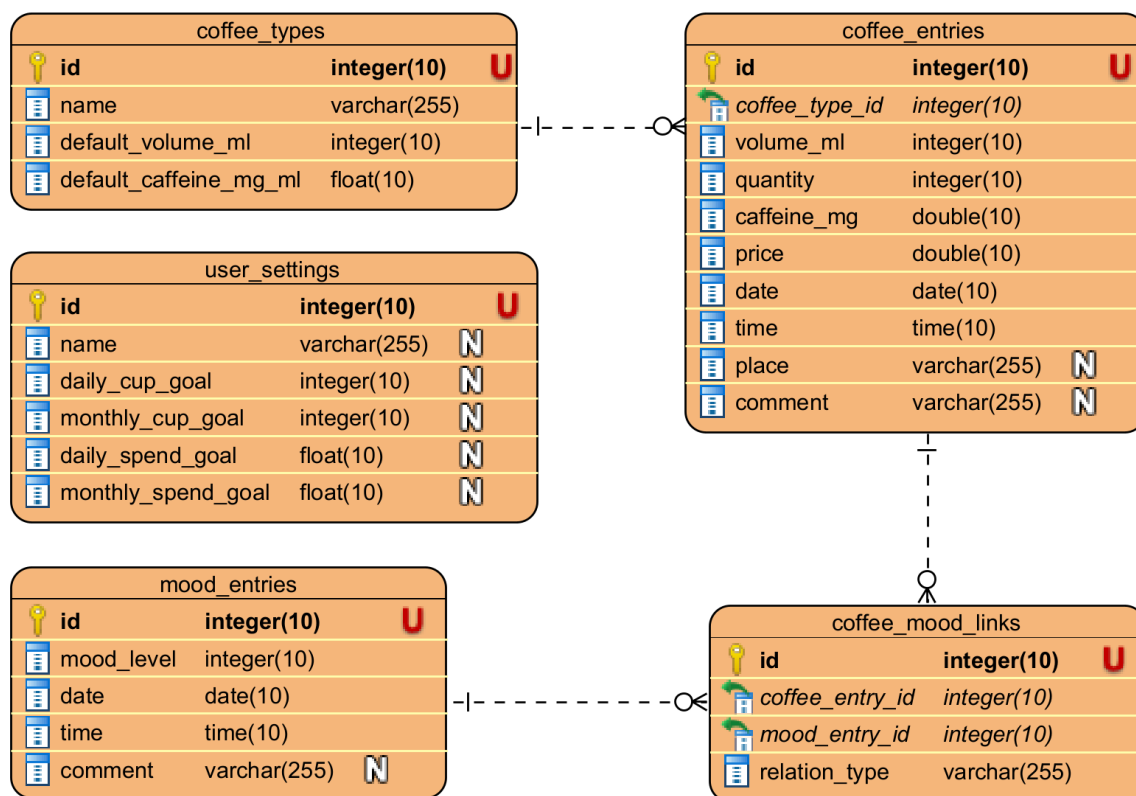


Рисунок 20 – Физическая модель базы данных

Таким образом, были спроектированы концептуальная, логическая и физическая модели базы данных всех сущностей, необходимых для удобного и эффективного хранения данных в мобильном приложении осознанного потребления кофе.

2.2.3 Проектирование пользовательского интерфейса мобильного приложения

Пользовательский интерфейс мобильного приложения для осознанного потребления кофе разрабатывался с применением Figma – современного инструмента для проектирования интерфейсов и быстрой сборки интерактивных прототипов.

Цветовая палитра построена на основе кофейных тонов: основным выбран светло-бежевый цвет, который гармонирует с темно-бежевым и белым цветом. Дополнительно используются нейтральные цвета: коричневый, черный и серый (рисунок 21).

Цвета приложения

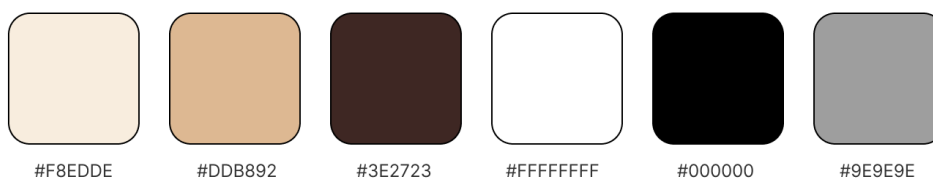


Рисунок 21 – Цветовая схема мобильного приложения

В ходе проектирования были детально проработаны макеты и интерактивные прототипы всех ключевых экранов:

- главной страницы;
- страницы добавления приема кофе;
- страницы фиксации настроения;
- страницы истории;
- страницы информации о приеме кофе;
- страницы информации об отмеченном настроении.

Макеты страниц представлены на рисунках 22 – 23.

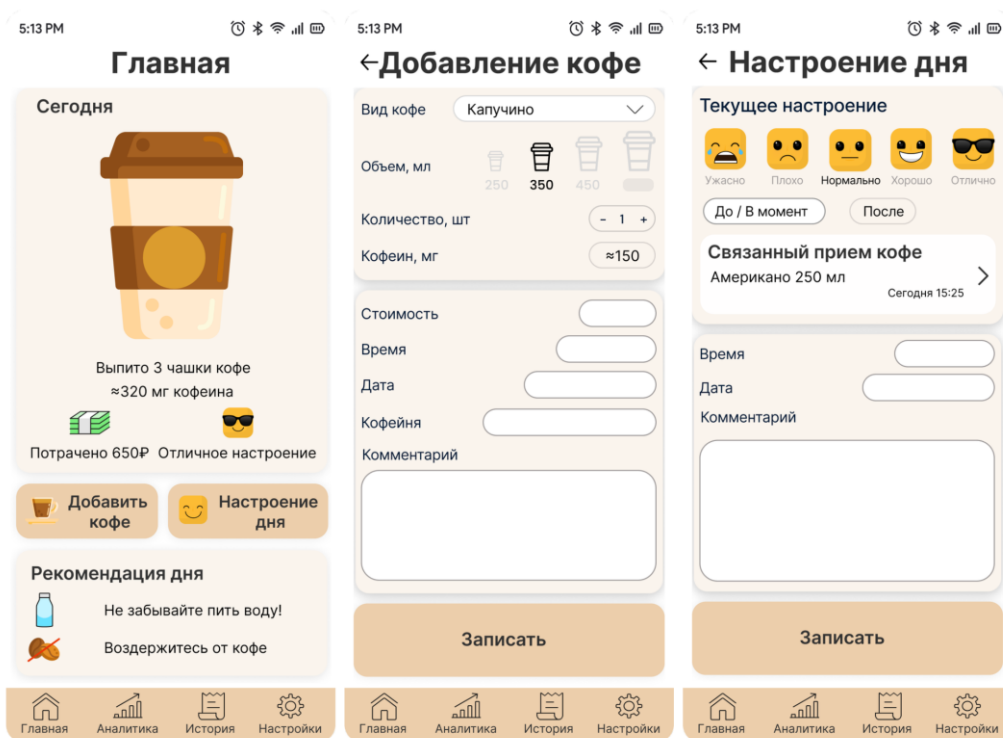


Рисунок 22 – Макеты страниц «Главная», «Добавление кофе» и «Настроение дня»

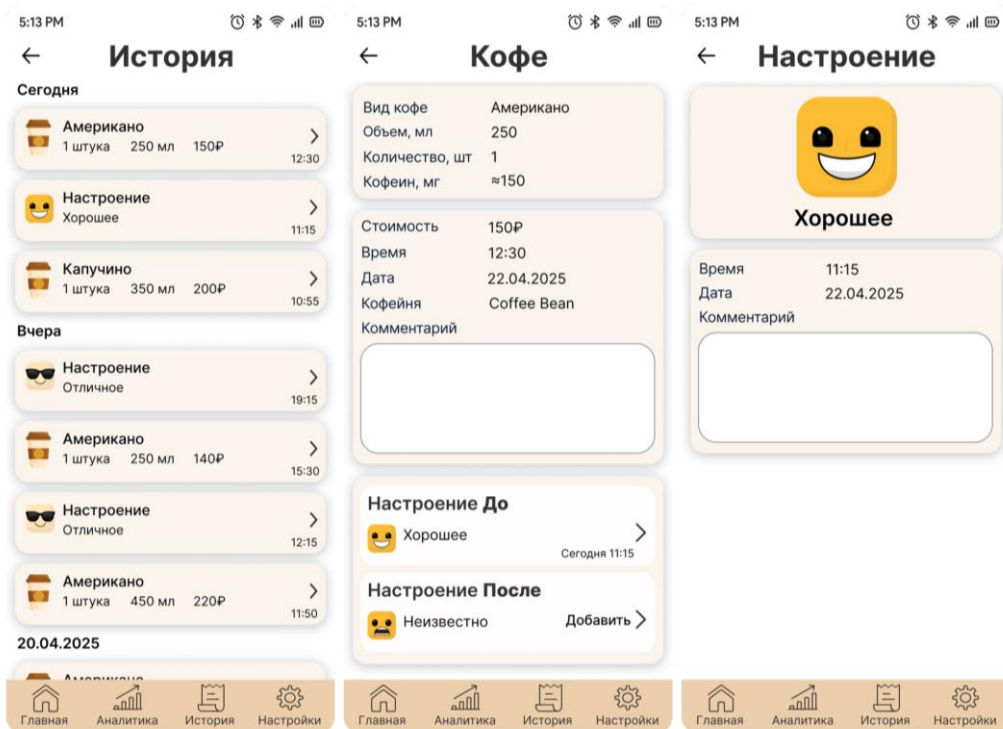


Рисунок 23 – Макеты страниц «История», «Кофе» и «Настроение»

По итогам проектирования были реализованы рабочие прототипы наиболее важных страниц приложения: просмотр статистики дня, добавление новых записей и просмотр истории записей. Это позволило не только оценить удобство навигации между экранами, но и повысить общий комфорт взаимодействия с приложением.

2.3 Реализация мобильного приложения осознанного потребления кофе

2.3.1 Реализация архитектуры мобильного приложения

Реализация мобильного приложения происходила по архитектуре Model-View-ViewModel (MVVM) на языке Kotlin. Основными элементами архитектуры являются:

- «Model» – здесь происходит обработка данных, выполняются операции и взаимодействие с источниками данных;

- «View» – пользовательский интерфейс приложения, с которым взаимодействует конечный потребитель;
- «ViewModel» – посредник между «Model» и «View», который содержит и логику работы с интерфейсом и предоставляет необходимые данные [17].

Разработанные компоненты были сгруппированы в пакеты:

- «ui» для компонентов «View» и «ViewModel»;
- «database» для компонентов «Model»;
- «layout» для разметки каждого из экранов.

В свою очередь компоненты пакета «ui» были сгруппированы в другие пакеты в зависимости от их функционального назначения:

- «addcoffee» для компонентов, отвечающих за добавление нового приема кофе;
- «addmood» для компонентов, отвечающих за добавление новой записи настроения;
- «analytics» для компонентов, отвечающих за просмотр вкладки «Аналитика»;
- «coffeedetails» для компонентов, отвечающих за просмотр деталей выпитого кофе;
- «coffeetype» для компонентов, отвечающих за добавление нового вида кофе;
- «history» для компонентов, отвечающих за просмотр вкладки «История»;
- «main» для компонентов, отвечающих за просмотр вкладки «Главная»;
- «mooddetails» для компонентов, отвечающих за просмотр деталей отмеченного настроения;
- «settings» для компонентов, отвечающих за просмотр вкладки «Настройки».

Рассмотрим архитектуру MVVM на примере функционала добавления записи кофе. Соответствующая диаграмма компонентов с точки зрения архитектуры представлена на рисунке 24.

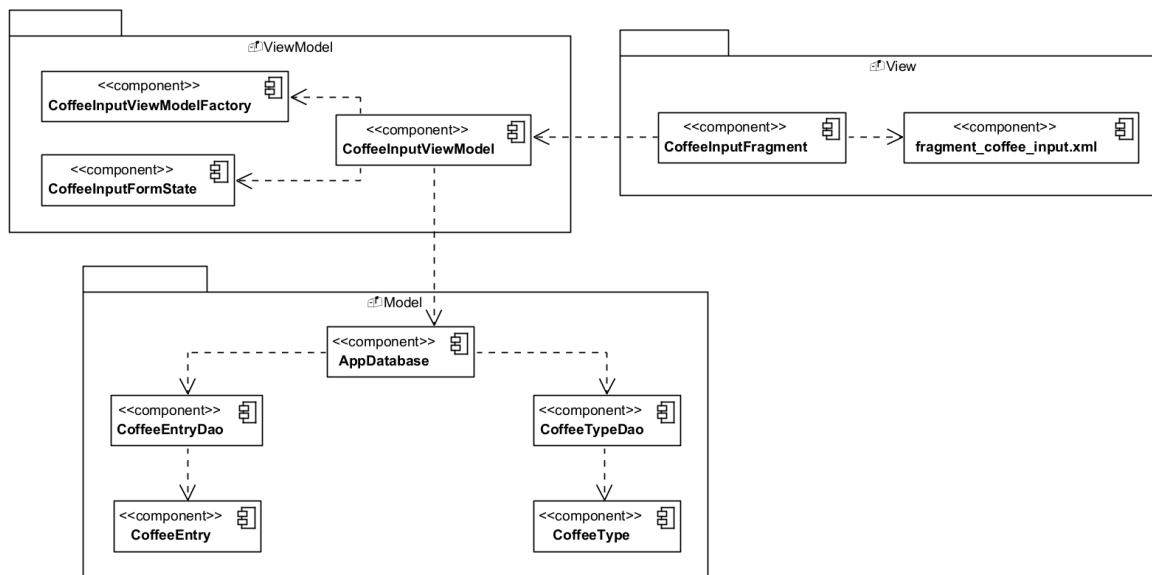


Рисунок 24 – Диаграмма компонентов функционала добавления кофе

Рассмотрим назначение каждого из компонентов:

- «fragment_coffee_input.xml» определяет структуру пользовательского интерфейса для добавления записи о приеме кофе;
- «CoffeeInputFragment» реализует логику UI (описанную через xml разметку), подписывается на изменение состояния и отправляет события в «CoffeeInputViewModel»;
- «CoffeeInputViewModel» хранит и обновляет состояние формы (CoffeeInputFormState) и взаимодействует с данными;
- «CoffeeInputFormState» хранит текущее состояние формы и ее возможные ошибки;
- «CoffeeInputViewModelFactory» отвечает за передачу зависимостей при создании «CoffeeInputViewModel»;
- «AppDatabase» обеспечивает доступ к таблицам базы данных через DAO («CoffeeEntryDao» и «CoffeeTypeDao»);

– «CoffeeEntryDao» и «CoffeeTypeDao» отвечают за взаимодействие с данными хранящихся в сущностях «CoffeeEntry» и «CoffeeType» соответственно;

– «CoffeeEntry» и «CoffeeType» отвечают за структуру данных записи приема кофе и видов кофе соответственно.

Схемы взаимодействия остальных функциональных компонентов приведены в приложении В.

Рассмотрим диаграмму классов на примере компонента для добавления нового вида кофе (рисунок 25).

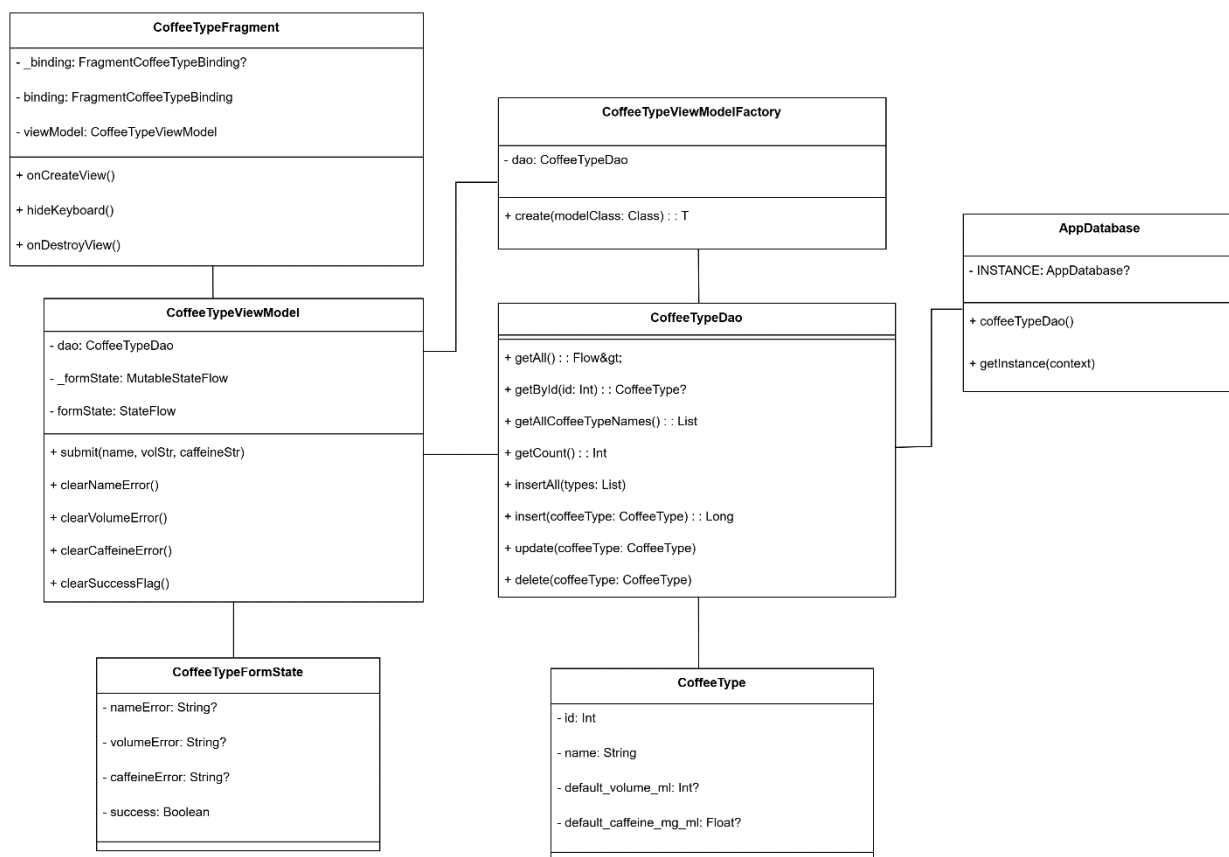


Рисунок 25 – Диаграмма классов для добавления вида кофе

На диаграмме представлены следующие классы:

- «CoffeeTypeFragment», который отвечает за пользовательский интерфейс и взаимодействие с пользователем на экране добавления вида кофе;
- «CoffeeTypeViewModel», который содержит логику обработки данных формы и взаимодействия с базой данных;

- «CoffeeTypeFormState», который хранит состояние формы, включая ошибки валидации и статус успешного сохранения;
- «CoffeeTypeViewModelFactory», который создает экземпляры ViewModel с необходимыми зависимостями;
- «CoffeeTypeDao», который является интерфейсом для доступа к данным вида кофе в базе данных;
- «CoffeeType», который является моделью данных, представляющей вид кофе с его свойствами;
- «AppDatabase», который является классом базы данных, предоставляющий доступ к DAO и управлению данными.

Разработка мобильного приложения происходила в среде разработки Android Studio, в которой происходила сборка «.apk» файла, для его дальнейшей установки на конечные устройства.

Исходный код реализации основного функционала представлен в приложении А. Также была подана заявка на получение свидетельства о государственной регистрации программы для ЭВМ «Мобильное приложение для осознанного потребления кофе».

2.3.2 Реализация базы данных мобильного приложения

Для работы с базой данных в приложении использовалась библиотека Room. К преимуществам ее применения можно отнести:

- проверка SQL-запросов в момент компиляции;
- интеграция с компонентами android;
- автоматическое управление миграциями [18].

На примере таблицы записей настроения («mood_entries») рассмотрим особенности описания сущностей в библиотеке Room (листинг 1).

Листинг 1 – Объявление сущности «MoodEntry»

```
@Entity(tableName = "mood_entries")
data class MoodEntry(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
```

```

        val mood_level: Int,
        val date: String,
        val time: String,
        val comment: String?
    )

```

Каждая сущность представляет собой класс, в котором каждое свойство соответствует столбцу таблицы. В аннотации класса «`@Entity(tableName = «mood_entries»)`» указывается будущее название таблицы. Также отмечено аннотацией («`@PrimaryKey(autoGenerate = true)`») и поле `id`, что позволяет автоматически генерировать уникальные идентификаторы записей.

Рассмотрим также способ взаимодействия с данными на примере нескольких операций, реализованных в интерфейсе доступа к данным «`MoodEntryDao`» (листинг 2).

Листинг 2 – Интерфейс «`MoodEntryDao`»

```

@Dao
interface MoodEntryDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insert(entry: MoodEntry): Long

    @Update
    suspend fun update(entry: MoodEntry)

    @Query("SELECT * FROM mood_entries ORDER BY date DESC, time DESC")
    fun getAll(): Flow<List<MoodEntry>>

    @Delete
    suspend fun delete(entry: MoodEntry)
}

```

Интерфейсы DAO (Data Access Object) в Room описывают способы взаимодействия с базой данных и аннотируются с помощью «`@Dao`». Методы внутри DAO соответствуют типовым операциям: вставке, обновлению, удалению и выполнению SQL-запроса; аннотации «`@Insert`», «`@Update`», «`@Delete`» и «`@Query`» соответственно. Данные можно получать в виде потока (Flow), что позволяет отслеживать изменения данных в реальном времени.

Доступ к папке с базой данных на устройстве ограничен приложением и пользователем с root-правами, что обеспечивает безопасность данных.

Структурная схема разработанной базы данных, реализованной с использованием библиотеки Room, представлена на рисунке 26.

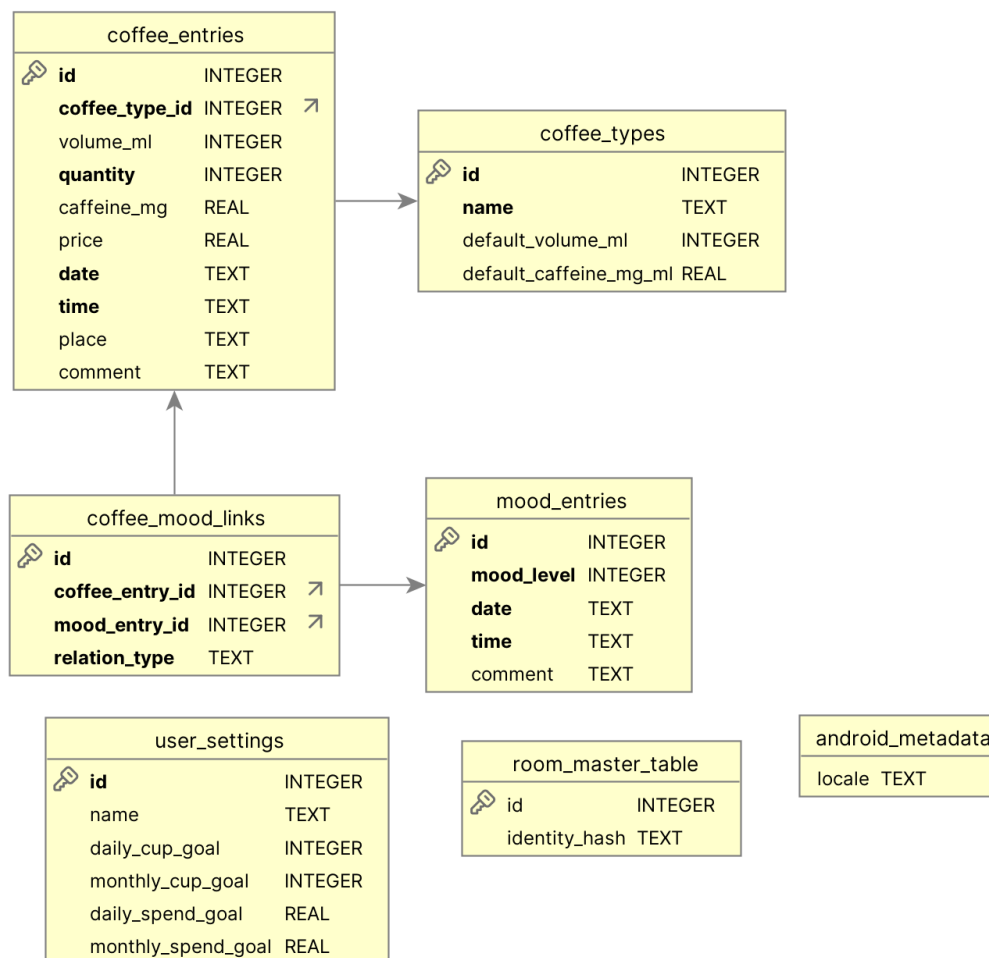


Рисунок 26 – Схема базы данных мобильного приложения

Таким образом, при помощи библиотеки Room была реализована база данных, которая не только описывает требуемую структуру таблиц, но и служит связующим звеном между объектной моделью приложения и реляционной базой данных.

2.3.3 Реализация пользовательского интерфейса мобильного приложения

В ходе разработки был реализован пользовательский интерфейс мобильного приложения для осознанного потребления кофе.

На рисунке 27 приведен скриншот главного экрана приложения (вкладка «Главная»).



Рисунок 27 – Главный экран приложения

С главной страницы возможно открыть как страницы добавления записей кофе и настроения, так и другие вкладки приложения, которые также как и главная вкладка после перехода становятся начальной точкой для открытия новых страниц. Граф навигации (карта экранов приложения) между экранами представлен на рисунке 28.

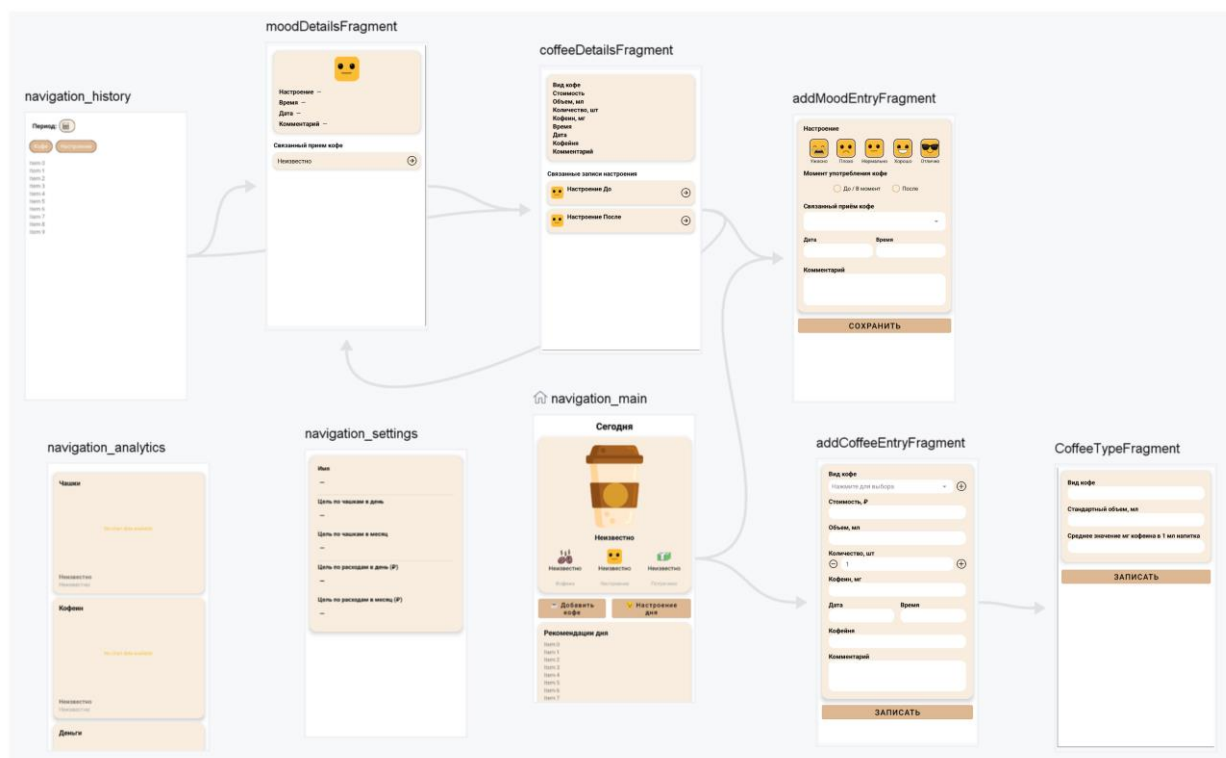


Рисунок 28 – Граф навигации

Стоит отметить, что на графе навигации страницы представлены в виде xml разметки, итоговый вид экранов пользовательского интерфейса приведен в приложении Г.

2.4 Тестирование мобильного приложения для осознанного потребления кофе

2.4.1 Обоснование выбора режима тестирования и метрик кода

Для устранения возможных ошибок в работе программного обеспечения проводится комплексное тестирование, которое направлено на повышение надежности приложения и улучшение пользовательского опыта взаимодействия с ним [19].

Наиболее популярны следующие виды тестирования:

- функциональное тестирование;
- нефункциональное тестирование;
- модульное тестирование;

- тестирование производительности;
- интеграционное тестирование.

В качестве метода ручного тестирования мобильного приложения было выбрано функциональное тестирование. Данный подход позволяет понять, насколько реализованные функции соответствуют заявленным требованиям, моделируя типичные сценарии взаимодействия пользователя с системой. Это обеспечивает целостную оценку корректности поведения приложения в различных сценариях.

В свою очередь, в рамках автоматического тестирования применялся метод модульного тестирования. Он позволяет изолированно проверять компоненты или функции приложения, обеспечивая быструю и точную диагностику ошибок на ранних этапах разработки. Как правило, модульные тесты автоматизируются и выполняются часто, что способствует поддержанию стабильности кода при его написании [20].

Качество программного обеспечения также можно оценить при помощи метрики программного кода. Они позволяют подробно анализировать технические характеристики продукта, прогнозировать трудозатраты на его разработку, количественно характеризовать качество принятых инженерных решений, а также определять сложность и надежность разрабатываемых систем. Все это способствует повышению общей стабильности и производительности итогового программного продукта [21].

Для оценки качества разработанного мобильного приложения была выбрана одна из наиболее распространённых программных метрик – метрика Холстеда. Она основывается на анализе количества уникальных операторов и операндов, а также общего числа их вхождений в программном коде. В рамках этой метрики вычисляются такие показатели, как длина и словарь программы, объём, сложность, а также предполагаемое время и усилия, необходимые для её понимания и сопровождения.

Метрика Холстеда позволяет объективно оценить сложность реализованных алгоритмов, сравнить различные фрагменты кода между собой, а также выявить участки, которые могут требовать оптимизации или более тщательной проверки [22].

2.4.2 Функциональное тестирование мобильного приложения

После завершения разработки мобильного приложения было проведено функциональное тестирование персональных рекомендаций с использованием подхода «белый ящик». В процессе тестирования были проверены все возможные сценарии работы функции «generateRecommendations», включая различные комбинации входных данных. Такой подход позволил оценить корректность работы внутренней логики, покрыть все ветвления кода и убедиться в отсутствии скрытых ошибок [20]. Код функции представлен в листинге 3.

Листинг 3 – Функция определения персональных рекомендаций

```
fun generateRecommendations(
    coffeeIntakes: List<CoffeeIntake>,
    spendGoal: Float
): List<String> = buildList {

    val totalCaffeine = coffeeIntakes.sumOf { it.volumeMl *
it.caffeineMgPerMl }
    val totalSpent = coffeeIntakes.sumOf { it.costRub.toDouble()
}.toFloat()

    when {
        totalCaffeine > 400 -> add("Превышен дневной лимит
кофеина. Рекомендуется снизить потребление.")
    }

    if (spendGoal > 0f) {
        when {
            totalSpent > spendGoal -> add("Расходы на кофе
превысили цель. Попробуйте проанализировать траты.")
        }
    }
}
```

Схема функции «generateRecommendations» представлена на рисунке 28.



Рисунок 29 – Схема функции «generateRecommendations»

С целью проверки, что каждый отдельный оператор в коде был выполнен хотя бы один раз, было проведено тестирование покрытием операторов (таблица 5).

Таблица 5 – Покрытие операторов

№	Операторы	Входные данные	Выходные данные
1	a, b, c, d, f	b=норма, c=норма	Без дополнительных сообщений
2	a, b, c, d, e, f	b=много, c=норма	Предупреждение о превышении потребления кофеина
3	a, b, c, d, f, g	b=норма, c=много	Предупреждение о превышении цели расхода средств
4	a, b, c, d, e, f, g	b=много, c=много	Предупреждение о превышении потребления кофеина и цели расхода средств

Также были проверены логические решения на все возможные исходы (таблица 6).

Таблица 6 – Покрытие решений

№	Решения	Входные данные	Выходные данные
1	d-да, f-да	b=норма, c=норма	Без дополнительных сообщений
2	d-нет, f-да	b=много, c=норма	Предупреждение о превышении потребления кофеина
3	d-да, f-нет	b=норма, c=много	Предупреждение о превышении цели расхода средств
4	d-нет, f-нет	b=много, c=много	Предупреждение о превышении потребления кофеина и цели расхода средств

Покрытие условий подразумевает составление такого набора тестов, при котором все возможные варианты исполнения каждого условия в решении оказываются протестированы хотя бы один раз. Однако данный подход не обеспечивает гарантированного прохождения всех операторов кода, поэтому

дополнительно требуется обеспечить выполнение каждого оператора минимум один раз.

В случае подбора рекомендаций данное требование выполняется, поскольку каждый оператор действительно выполняется хотя бы однократно. Из чего можно сделать вывод, что в построении таблиц тестов нет необходимости. Покрытие решений и условий представлено в таблице 7.

Таблица 7 – Покрытие решений и условий

№	Условие	Входные данные	Выходные данные
1	a, b, c, d (да), f(да)	b=норма, c=норма	Без дополнительных сообщений
2	a, b, c, d (нет), f(да)	b=много, c=норма	Предупреждение о превышении потребления кофеина
3	a, b, c, d (да), f(нет)	b=норма, c=много	Предупреждение о превышении цели расхода средств
4	a, b, c, d (нет), f(нет)	b=много, c=много	Предупреждение о превышении потребления кофеина и цели расхода средств

Функционал создания персональных рекомендаций успешно прошел функциональное тестирование на примере тестирования методом «белого ящика», показав правильность работы программных средств и соответствие функциональным требованиям.

2.4.3 Нефункциональное тестирование мобильного приложения

После проведения функционального тестирования, для оценки качества работы мобильного приложения в реальных условиях, было проведено нефункциональное тестирование. Оно проводится с целью выявить узкие места в производительности, проблемы с адаптацией под разные устройства и неудобства в пользовательском интерфейсе.

С целью проверить скорость работы приложения и его отклик под нагрузкой было проведено тестирование производительностью. План тестирования представлен в таблице 8. Тестирование включало в себя запуск

основных пользовательских сценариев при различных уровнях нагрузки для выявления возможных узких мест и оценки масштабируемости системы.

Таблица 8 – Тестирование производительности

Тест	Условие	Ожидаемый результат	Результат
Время отклика при добавлении кофе	Открыть экран добавления, ввести данные, сохранить	Отклик не более 1 секунды	Тест пройден
Открытие главного экрана с большими данными	Загрузка списка с большим количеством записей	Отображение не более 2 секунд	Тест пройден
Пролистывание списка кофе	Быстрое пролистывание списка из 500 записей	Плавная прокрутка без подтормаживаний	Тест пройден

С целью проверки стабильности работы приложения при длительном использовании было проведено тестирование надежности. План тестирования представлен в таблице 9. В ходе тестирования оценивалась устойчивость приложения к сбоям, отсутствие утечек памяти и корректная работа при многократном выполнении однотипных операций.

Таблица 9 – Тестирование надежности

Тест	Условие	Ожидаемый результат	Результат
Непрерывная работа 12 часов	Приложение активно без перезапуска	Отсутствие утечек памяти, сбоев, зависаний	Тест пройден
Повторное добавление и удаление записей	Многократное добавление и удаление записей	Без сбоев и потери данных	Тест пройден
Выход из приложения и повторный запуск	Закрытие и запуск приложения 10 раз подряд	Корректное восстановление состояния	Тест пройден
Ожидание на экране без действий	Приложение открыто, пользователь не взаимодействует в течение часа	Стабильность, отсутствие утечек памяти	Тест пройден

С целью проверки мобильного приложения на разных устройствах платформы Android, было проведено тестирование совместимости. План тестирования представлен в таблице 10. В ходе тестирования учитывались различные версии операционной системы и типы устройств, чтобы гарантировать корректную работу приложения в широком диапазоне пользовательских условий.

Таблица 10 – Тестирование совместимости

Тест	Условие	Ожидаемый результат	Результат
Запуск на Android 9, 10, 11, 12+	Установка и запуск на различных версиях ОС	Без ошибок и сбоев	Тест пройден
Отображение на экранах 16х9, 18х9 и 20х9	Запуск на устройствах с разными соотношениями экрана	Корректный адаптивный интерфейс	Тест пройден
Проверка работы в режиме энергосбережения	Включение энергосбережения на устройстве	Стабильная работа без сбоев	Тест пройден

С целью проверки удобства интерфейса и взаимодействия было проведено тестирование удобства использования. План тестирования представлен в таблице 11. В рамках тестирования учитывались сценарии работы новых пользователей, понятность сообщений об ошибках и простота навигации по основным разделам приложения.

Таблица 11 – Тестирование удобства использования

Тест	Условие	Ожидаемый результат	Результат
Обработка ошибок ввода	Ввод некорректных данных	Появление понятного сообщения об ошибке	Тест пройден
Навигация между экранами	Переходы между разделами приложения	Быстрый и простой доступ ко всем функциям	Тест пройден
Использование приложением с одной рукой	Проверка удобства на больших экранах	Все элементы доступны и читаемы	Тест пройден

С целью проверки сохранения данных и восстановления после аварийных ситуаций было проведено тестирование восстанавливаемости. План тестирования представлен в таблице 12. В процессе тестирования моделировались сценарии неожиданного завершения работы приложения и перезагрузки устройства, чтобы убедиться в сохранности данных и корректном восстановлении работы приложения.

Таблица 12 – Тестирование восстанавливаемости

Тест	Условие	Ожидаемый результат	Результат
Восстановление после закрытия	Закреть и заново открыть приложение	Все данные сохраняются	Тест пройден
Восстановление после перезагрузки устройства	Перезагрузить телефон во время работы приложения	Данные не потеряны	Тест пройден
Прерывание процесса добавления записи	Прервать добавление записи (закрывать экран)	Нет частично сохранённых или повреждённых данных	Тест пройден
Восстановление после сбоя приложения	Принудительное закрытие приложения во время работы	При следующем запуске приложение корректно работает	Тест пройден

Результаты нефункционального тестирования показывают, что мобильное приложение для осознанного потребления кофе стабильно работает под нагрузкой и в течение длительного времени работы, успешно восстанавливает работу после сбоев. А интерфейс приложения удобен и интуитивно понятен.

2.4.4 Модульное тестирование мобильного приложения

Для проверки работоспособности бизнес-логики мобильного приложения для осознанного потребления кофе было проведено модульное тестирование, которое позволило в автоматическом режиме проверить работоспособность функций приложения.

Реализация тестирования происходила при помощи библиотек junit (для запуска и организации тестовых методов) и Mock (для создания и настройки DAO-заглушек).

В качестве примера в листинге 1 приложения Д приведен исходный код модульных тестов для бизнес-логики страницы аналитики, результаты прохождения тестов приведены на рисунке 29.

✓ AnalyticsViewModelTest	785 ms
✓ defaults for all fields when no data	702 ms
✓ no money goal means no excess	6 ms
✓ mood points filled for days with mood	4 ms
✓ empty db gives default state	5 ms
✓ cups sum includes all entries	13 ms
✓ no coffee gives dash in cupsSum	5 ms
✓ one coffee entry, caffeine and cups shown	4 ms
✓ money sum does not exceed goal	4 ms
✓ money sum exceeds goal	4 ms
✓ many cups in one day summed correctly	4 ms
✓ empty days filled with zeros	4 ms
✓ mood average calculated correctly	5 ms
✓ caffeine average over 400 is exceeded	5 ms
✓ money sum formatted with ruble sign	4 ms
✓ no cup goal means no excess	4 ms
✓ coffee for several days, caffeine average cor	4 ms
✓ no moods gives moodAvg zero	4 ms
✓ cups goal exceeded is true	4 ms

Рисунок 30 – Результаты модульного тестирования бизнес-логики страницы аналитики

Бизнес-логика других страниц также была протестирована (рисунок 30).

✓ Test Results	2 sec 291 ms	✓ Tests passed: 82 of 82 tests – 2 sec 291 ms
> ✓ AnalyticsViewModelTest	785 ms	> Task :app:createDebugCompatibleScreen
> ✓ CoffeeDetailsViewModelTest	1 sec 199 ms	> Task :app:extractDeepLinksDebug UP-7
> ✓ CoffeeInputViewModelTest	41 ms	> Task :app:processDebugMainManifest
> ✓ CoffeeTypeViewModelTest	22 ms	> Task :app:processDebugManifest
> ✓ HistoryViewModelTest	74 ms	> Task :app:javaPreCompileDebug UP-T0-
> ✓ MoodDetailsViewModelTest	32 ms	> Task :app:preDebugUnitTestBuild UP-7
> ✓ MoodInputViewModelTest	93 ms	> Task :app:javaPreCompileDebugUnitTes
> ✓ RecommendationAdapterTest	4 ms	> Task :app:processDebugManifestForPar
> ✓ SettingsViewModelTest	41 ms	> Task :app:processDebugResources
		> Task :app:kspDebugKotlin UP-T0-DATE

Рисунок 31 – Результаты модульного тестирования бизнес-логики всех страниц

Мобильное приложение успешно прошло все разработанные модульные тесты, что подтверждает корректность его работы в различных ситуациях.

2.4.5 Метрики кода мобильного приложения

Для оценки метрик кода мобильного приложения была применена метрика Холстеда на примере функции «loadGroupedHistoryItems», которая реализует функционал подготовки элементов истории по дням. Код этой функции приведен в листинге 2 приложения Д.

Для вычисления данной метрики потребуется подсчитать число вхождения уникальных операторов (N1) и операндов (N2), а также их общее количество (n1 и n2 соответственно). Операторы и операнды функции «loadGroupedHistoryItems» и число их вхождений приведены в таблице 13.

Таблица 13 – Списки операторов и операндов функции «loadGroupedHistoryItems»

№	Операторы		Операнды	
	Оператор	Количество	Операнд	Количество
1	=	7	start	2
2	.	41	end	2
3	()	19	dateRange	1
4	{}	14	startStr	3
5	,	7	endStr	3
6	if	2	DateTimeFormatter	2
7	else	2	ISO_LOCAL_DATE	2
8	runCatching	3	coffeeTypeMap	2
9	getOrDefault	3	db	3
10	associate	1	coffeeTypeDao	1
11	to	1	coffeeEntryDao	1
12	map	2	moodEntryDao	1
13	getEntriesForPeriod	2	emptyMap	1
14	toHistoryItem	2	emptyList	4
15	+	2	filters	3
16	sortedByDescending	1	showCoffee	1
17	when	2	showMood	1
18	is	6	coffee	2
19	->	7	mood	2
20	groupBy	1	allItems	2
21	toSortedMap	1	HistoryItem	8
22	compareByDescending	1	date	3

Продолжение таблицы 13

№	Операторы		Операнды	
	Оператор	Количество	Операнд	Количество
23	flatMap	1	it	12
24	listOf	1		
25	return	1		
	Всего	130	Всего	62

Данные, полученные в ходе обработки программного кода функции «loadGroupedHistoryItems» были использованы для расчета характеристик программы по формулам из метрики Холстеда. Результаты вычислений представлены в таблице 14.

Таблица 14 – Результаты вычислений по метрике Холстеда

Значение	Результат
Словарь программы	48
Длина программы	192
Объем программы	1072
Сложность программы	33

Таким образом, все полученные значения находятся в рамках рекомендованных диапазонов для функций средней сложности. Это свидетельствует о том, что реализация функции является оптимальной: код читаемый, не избыточный и не слишком сложный для поддержки и модификации.

Заключение

В рамках выполнения выпускной квалификационной работы в формате стартапа был разработан бизнес-план мобильного приложения, предназначенного для осознанного потребления кофе. В процессе работы выполнены следующие задачи:

- проведен анализ рынка потребления кофе, определены целевые группы пользователей и выявлены тенденции развития отрасли;
- описана концепция мобильного приложения, его ключевая идея, структура и основные технико-экономические характеристики;
- составлен маркетинговый план с описанием стратегий продвижения, подходов к ценообразованию и выбором наиболее подходящего способа продажи продукта;
- разработан финансовый план на ближайшие 3 следующих года, в котором рассчитаны прогнозные показатели, необходимые для достижения целей проекта.

В процессе выполнения выпускной квалификационной работы было реализовано мобильное приложение, предназначенное для осознанного потребления кофе. Для этого были выполнены следующие задачи:

- проведен анализ предметной области и выявлены ключевые особенности рынка потребления кофе;
- выполнен сравнительный анализ существующих мобильных приложений-аналогов и определены их преимущества и недостатки;
- сформулированы функциональные и нефункциональные требования к разрабатываемому мобильному приложению;
- поставлены цели и задачи разработки приложения, а также проведена оценка бюджета и распределение ролей в проекте;
- спроектирована архитектура мобильного приложения, структура базы данных и пользовательский интерфейс;

- реализованы архитектурные решения, база данных и пользовательский интерфейс;
- проведено функциональное, нефункциональное и модульное тестирование приложения и оценка качества кода с помощью метрик.

В ходе выполнения выпускной квалификационной работы было создано мобильное приложение для осознанного потребления кофе, соответствующее всем требованиям технического задания. Также была подготовлена и подана заявка на регистрацию программы для ЭВМ, подтверждающая оригинальность и правовую защищённость разработанного продукта.

Список использованных источников

1. Одинцов Е. Россияне стали чаще пить кофе // АО «Газета.Ру». [Электронный ресурс]. URL: <https://www.gazeta.ru/social/news/2025/01/26/24922910.shtml> (дата обращения 10.02.2025).
2. Морозова А. Рынок кофе с собой в России вырос за год более чем на 20% // Forbes. [Электронный ресурс]. URL: <https://www.forbes.ru/biznes/504480-rynok-kofe-s-soboj-v-rossii-vyros-za-god-bolee-chem-na-20> (дата обращения 10.02.2025).
3. Ионова Л. Как в России развивается рынок «кофе на вынос» // Российская газета. [Электронный ресурс]. URL: <https://rg.ru/2024/04/02/reg-ufo/kak-v-rossii-razvivaetsia-rynok-kofe-na-vynos.html> (дата обращения 10.02.2025).
4. Старшова Е. B2B и B2C: в чем разница? Как их объединяют в eCommerce? // SimTech Development. [Электронный ресурс]. URL: <https://simtechdev.ru/blog/b2b-vs-b2c/> (дата обращения 15.02.2025).
5. Бодрящий тренд: как рост цен на кофе повлиял на его продажи в России // РБК Отрасли. [Электронный ресурс]. URL: <https://www.rbc.ru/industries/news/67e13b5d9a7947e45d1fb71e> (дата обращения 20.02.2025).
6. Coffee Consumption by Country 2025 // World Population Review. [Электронный ресурс]. URL: <https://worldpopulationreview.com/country-rankings/coffee-consumption-by-country> (дата обращения 21.02.2025).
7. Как изменились предпочтения потребителей кофе в последние годы // РБК Компании. [Электронный ресурс]. URL: <https://companies.rbc.ru/news/PlbT9kJT5T/kak-izmenilis-predpochteniya-potrebiteldej-kofe-v-poslednie-godyi> (дата обращения 22.02.2025).
8. Функциональные и нефункциональные требования: различия и особенности // Блог Яндекс Практикума. [Электронный ресурс]. URL: <https://practicum.yandex.ru/blog/funkcionalnye-i-nefunkcionalnye-trebovaniya/> (дата обращения 01.03.2025).

9. Джим Арлоу. UML2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование / Джим Арлоу, Айла Нейштадт. – Санкт-Петербург, Издательство Символ-Плюс, 2007. – 624с.

10. Роли в разработке ИТ-продукта // Тематический блог «Хабр». [Электронный ресурс]. URL: <https://habr.com/ru/articles/695944/> (дата обращения 10.03.2025).

11. Ричардс, М. Фундаментальный подход к программной архитектуре: паттерны, свойства, проверенные методы / М. Ричардс, Н. Форд – СПб. : Питер, 2023. – 448 с.

12. Komilov Y. Android Architecture Patterns: Comparing MVC, MVP, MVVM, and MVI // Medium. [Электронный ресурс]. URL: <https://medium.com/@YodgorbekKomilo/android-architecture-patterns-comparing-mvc-mvp-mvvm-and-mvi-b282712cfb46> (дата обращения 15.03.2025).

13. Android Architecture Patterns – MVC, MVP, MVVM, MVI, Clean Architecture // Medium. [Электронный ресурс]. URL: <https://medium.com/droidblogs/android-architecture-patterns-mvc-mvp-mvvm-mvi-clean-architecture-cde8029b8f37> (дата обращения 15.03.2025).

14. Android Architecture Patterns: MVC vs MVVM vs MVP // daily.dev. [Электронный ресурс]. URL: <https://daily.dev/blog/android-architecture-patterns-mvc-vs-mvvm-vs-mvp> (дата обращения 16.03.2025).

15. Model-View-Intent (MVI) Pattern in Reactive Programming: A Comprehensive Overview // GeeksforGeeks. [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/model-view-intent-mvi-pattern-in-reactive-programming-a-comprehensive-overview/> (дата обращения 16.03.2025).

16. Проектирование баз данных // DecoSystems. [Электронный ресурс]. URL: <https://www.decosystems.ru/proektirovanie-baz-dannykh/> (дата обращения 20.03.2025).

17. Простая архитектура с использованием MVVM и делегатов в Android. Оптимальное решение для малых проектов // Тематический блог

«Хабр». [Электронный ресурс]. URL: <https://habr.com/ru/articles/776344/> (дата обращения 25.03.2025).

18. Виноградов Д. Урок 5. Room. Основы // StartAndroid. [Электронный ресурс]. URL: <https://startandroid.ru/ru/courses/architecture-components/27-course/architecture-components/529-urok-5-room-osnovy.html> (дата обращения 01.04.2025).

19. Крауч, Р. Тестирование ПО. Основы, методы и термины / Р. Крауч, В.Д. Влиет. – СПб.: Питер, 2003. – 256 с

20. Куликов, С.С. Тестирование программного обеспечения. Базовый курс / С.С. Куликов – Минск: ЕРАМ Systems, 2021. – 300 с.

21. Программный код и его метрики // Тематический блог «Хабр». [Электронный ресурс]. URL: <https://habr.com/ru/companies/intel/articles/106082/> (дата обращения 01.05.2025).

22. Метрики как средства управления качеством // OSP. [Электронный ресурс]. URL: <https://www.osp.ru/os/2009/08/10748698> (дата обращения 03.05.2025).

Приложение А
(обязательное)

Код мобильного приложения

Листинг А.1 – Файл MainActivity.kt

```
package com.example.coffies

class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        val db = AppDatabase.getInstance(this)
        CoroutineScope(Dispatchers.IO).launch {
            insertDefaultCoffeeTypesIfNeeded(db)
        }

        val navView: BottomNavigationView = binding.navView
        val navController =
            findNavController(R.id.nav_host_fragment_activity_main)

        val appBarConfiguration = AppBarConfiguration(
            setOf(
                R.id.navigation_main,
                R.id.navigation_analytics,
                R.id.navigation_history,
                R.id.navigation_settings
            )
        )
        setupActionBarWithNavController(navController,
            appBarConfiguration)

        navController.addOnDestinationChangedListener { _,
            destination, _ ->
                when (destination.id) {
                    R.id.navigation_main ->
                        navView.menu.findItem(R.id.navigation_main).isChecked = true
                    R.id.navigation_analytics ->
                        navView.menu.findItem(R.id.navigation_analytics).isChecked = true
                    R.id.navigation_history ->
                        navView.menu.findItem(R.id.navigation_history).isChecked = true
                    R.id.navigation_settings ->
                        navView.menu.findItem(R.id.navigation_settings).isChecked = true
                }
            }
        navView.setOnItemSelectedListener { item ->
            val destinationId = when (item.itemId) {
                R.id.navigation_main -> R.id.navigation_main
                R.id.navigation_analytics ->
                    R.id.navigation_analytics
            }
        }
    }
}
```

```

        R.id.navigation_history -> R.id.navigation_history
        R.id.navigation_settings ->
R.id.navigation_settings
        else -> null
    }
    destinationId?.let {
        if (navController.currentDestination?.id != it) {
            navController.popBackStack(it, false)

            val navOptions = NavOptions.Builder()
                .setLaunchSingleTop(true)
                .setEnterAnim(R.anim.fade_in)
                .setExitAnim(R.anim.fade_out)
                .setPopEnterAnim(R.anim.fade_in)
                .setPopExitAnim(R.anim.fade_out)
                .setRestoreState(true)
                .build()

            navController.navigate(it, null, navOptions)
        }
        true
    } ?: false
}
}
override fun onSupportNavigateUp(): Boolean {
    val navController =
findNavController(R.id.nav_host_fragment_activity_main)
    return navController.navigateUp() ||
super.onSupportNavigateUp()
}
}

```

Листинг A.2 – Файл AppDatabase.kt

```

package com.example.coffies.database

@Database(
    entities = [
        CoffeeType::class,
        CoffeeEntry::class,
        MoodEntry::class,
        CoffeeMoodLink::class,
        UserSettings::class
    ],
    version = 1,
    exportSchema = false
)
abstract class AppDatabase : RoomDatabase() {
    abstract fun coffeeTypeDao(): CoffeeTypeDao
    abstract fun coffeeEntryDao(): CoffeeEntryDao
    abstract fun moodEntryDao(): MoodEntryDao
    abstract fun coffeeMoodLinkDao(): CoffeeMoodLinkDao
    abstract fun userSettingsDao(): UserSettingsDao
}

```

```

companion object {
    @Volatile
    private var INSTANCE: AppDatabase? = null
    fun getInstance(context: Context): AppDatabase {
        return INSTANCE ?: synchronized(this) {
            val instance = Room.databaseBuilder(
                context.applicationContext,
                AppDatabase::class.java,
                "coffee_tracker_db"
            ).build()
            INSTANCE = instance
            instance
        }
    }
}
}

```

Листинг А.3 – Файл CoffeeEntry.kt

```

package com.example.coffies.database.coffeentry

@Entity(
    tableName = "coffee_entries",
    foreignKeys = [
        ForeignKey(
            entity = CoffeeType::class,
            parentColumns = ["id"],
            childColumns = ["coffee_type_id"],
            onDelete = ForeignKey.CASCADE
        )
    ],
    indices = [Index("coffee_type_id")]
)
data class CoffeeEntry(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val coffee_type_id: Int,
    val volume_ml: Int?,
    val quantity: Int = 1,
    val caffeine_mg: Double?,
    val price: Double?,
    val date: String,
    val time: String,
    val place: String?,
    val comment: String?
)

```

Листинг А.4 – Файл CoffeeEntryDao.kt

```

package com.example.coffies.database.coffeentry

@Dao
interface CoffeeEntryDao {

```

```

@Insert(onConflict = OnConflictStrategy.REPLACE)
suspend fun insert(entry: CoffeeEntry): Long

@Update
suspend fun update(entry: CoffeeEntry)

@Query("SELECT * FROM coffee_entries ORDER BY date DESC, time DESC")
fun getAll(): Flow<List<CoffeeEntry>>

@Query("SELECT * FROM coffee_entries WHERE id = :id")
suspend fun getById(id: Int): CoffeeEntry?

@Query("SELECT * FROM coffee_entries ORDER BY date DESC, time DESC LIMIT :limit")
fun getAllDescLimited(limit: Int): Flow<List<CoffeeEntry>>

@Query("SELECT * FROM coffee_entries WHERE date >= :from AND date <= :to ORDER BY date DESC, time DESC")
suspend fun getEntriesForPeriod(
    from: String,
    to: String
): List<CoffeeEntry>

@Query("SELECT coalesce(SUM(quantity), 0) FROM coffee_entries WHERE date = :today")
suspend fun getCupsCountForDate(today: String): Int
@Query("SELECT coalesce(SUM(coalesce(caffeine_mg,0) * quantity), 0) FROM coffee_entries WHERE date = :today")
suspend fun getTotalCaffeineForDate(today: String): Double
@Query("SELECT coalesce(SUM(coalesce(price,0)), 0) FROM coffee_entries WHERE date = :today")
suspend fun getTotalSpentForDate(today: String): Double
@Query("SELECT * FROM coffee_entries WHERE date BETWEEN :from AND :to")
suspend fun getAllBetweenDates(from: String, to: String): List<CoffeeEntry>
@Delete
suspend fun delete(entry: CoffeeEntry)
@Query("DELETE FROM coffee_entries WHERE id = :id")
suspend fun deleteById(id: Int)

}

```

Листинг А.5 – Файл CoffeeType.kt

```
package com.example.coffies.database.coffeetype
```

```
@Entity(tableName = "coffee_types")
data class CoffeeType(
```

```
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
```

```

        val name: String,
        val default_volume_ml: Int?,
        val default_caffeine_mg_ml: Float?
    )

```

```

data class CoffeeTypeIdName(
    val id: Int,
    val name: String
)

```

Листинг A.6 – Файл CoffeeTypeDao.kt

```

package com.example.coffies.database.coffeetype

@Dao
interface CoffeeTypeDao {

    @Query("SELECT * FROM coffee_types")
    fun getAll(): Flow<List<CoffeeType>>
    @Query("SELECT * FROM coffee_types WHERE id = :id")
    suspend fun getById(id: Int): CoffeeType?
    @Query("SELECT id, name FROM coffee_types")
    suspend fun getAllCoffeeTypeNames(): List<CoffeeTypeIdName>
    @Query("SELECT COUNT(*) FROM coffee_types")
    suspend fun getCount(): Int
    @Insert
    suspend fun insertAll(types: List<CoffeeType>)
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insert(coffeeType: CoffeeType): Long
    @Update
    suspend fun update(coffeeType: CoffeeType)
    @Delete
    suspend fun delete(coffeeType: CoffeeType)
}

```

Листинг A.7 – Файл CoffeeMoodLink.kt

```

package com.example.coffies.database.coffemoodlink

@Entity(
    tableName = "coffee_mood_links",
    foreignKeys = [
        ForeignKey(
            entity = CoffeeEntry::class,
            parentColumns = ["id"],
            childColumns = ["coffee_entry_id"],
            onDelete = ForeignKey.CASCADE
        ),
        ForeignKey(
            entity = MoodEntry::class,
            parentColumns = ["id"],
            childColumns = ["mood_entry_id"],

```

```

        onDelete = ForeignKey.CASCADE
    )
],
indices = [
    Index("coffee_entry_id"),
    Index("mood_entry_id")
]
)
data class CoffeeMoodLink(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val coffee_entry_id: Int,
    val mood_entry_id: Int,
    val relation_type: String // 'before and during', 'after'
)

```

Листинг А.8 – Файл CoffeeMoodLinkDao.kt

```

package com.example.coffies.database.coffemoodlink

@Dao
interface CoffeeMoodLinkDao {

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insert(link: CoffeeMoodLink): Long

    @Query("SELECT * FROM coffee_mood_links WHERE coffee_entry_id = :coffeeEntryId")
    suspend fun getLinksForCoffeeEntry(coffeeEntryId: Int):
    List<CoffeeMoodLink>

    @Query("SELECT * FROM coffee_mood_links WHERE mood_entry_id = :moodEntryId")
    suspend fun getLinksForMoodEntry(moodEntryId: Int):
    List<CoffeeMoodLink>

    @Query("SELECT coffee_entry_id FROM coffee_mood_links WHERE relation_type = :type")
    suspend fun getCoffeeEntryIdsByType(type: String): List<Int>

    @Delete
    suspend fun delete(link: CoffeeMoodLink)
}

```

Листинг А.9 – Файл MoodEntry.kt

```

package com.example.coffies.database.moodentry

@Entity(tableName = "mood_entries")
data class MoodEntry(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val mood_level: Int,
    val date: String,
    val time: String,

```

```
        val comment: String?
    )
}
```

Листинг A.10 – Файл MoodEntryDao.kt

```
package com.example.coffies.database.moodentry

@Dao
interface MoodEntryDao {

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insert(entry: MoodEntry): Long

    @Update
    suspend fun update(entry: MoodEntry)

    @Query("SELECT * FROM mood_entries ORDER BY date DESC, time DESC")
    fun getAll(): Flow<List<MoodEntry>>

    @Query("SELECT * FROM mood_entries WHERE id = :id")
    suspend fun getById(id: Int): MoodEntry?

    @Query("""
        SELECT * FROM mood_entries
        WHERE date >= :from AND date <= :to
        ORDER BY date DESC, time DESC
    """)
    suspend fun getEntriesForPeriod(
        from: String,
        to: String
    ): List<MoodEntry>

    @Query("SELECT * FROM mood_entries WHERE date BETWEEN :from AND :to")
    suspend fun getAllBetweenDates(from: String, to: String): List<MoodEntry>

    @Query("SELECT AVG(mood_level) FROM mood_entries WHERE date = :today")
    suspend fun getAverageMoodLevelForDate(today: String): Double?

    @Delete
    suspend fun delete(entry: MoodEntry)
}
```

Листинг A.11 – Файл UserSettings.kt

```
package com.example.coffies.database.usersettings

@Entity(tableName = "user_settings")
data class UserSettings(
    @PrimaryKey val id: Int = 1,

```

```

    val name: String? = null,
    val daily_cup_goal: Int? = null,
    val monthly_cup_goal: Int? = null,
    val daily_spend_goal: Float? = null,
    val monthly_spend_goal: Float? = null
)

```

Листинг A.12 – Файл UserSettingsDao.kt

```

package com.example.coffies.database.usersettings

@Dao
interface UserSettingsDao {

    @Query("SELECT * FROM user_settings WHERE id = 1 LIMIT 1")
    fun getSettings(): Flow<UserSettings?>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertOrUpdate(settings: UserSettings)

    @Query("DELETE FROM user_settings WHERE id = 1")
    suspend fun clearSettings()

    @Query("SELECT * FROM user_settings WHERE id = 1 LIMIT 1")
    suspend fun getSettingsOnce(): UserSettings?
}

```

Листинг A.13 – Файл CoffeeInputFormState.kt

```

package com.example.coffies.ui.addcoffee

import com.example.coffies.database.coffeetype.CoffeeType

data class CoffeeInputFormState(
    val coffeeType: CoffeeType? = null,
    val volume: String = "",
    val quantity: String = "1",
    val caffeine: String = "",
    val date: String = "",
    val dateIso: String = "",
    val time: String = "",
    val cost: String = "",
    val place: String = "",
    val comment: String = "",
    val coffeeTypeError: String? = null,
    val volumeError: String? = null,
    val quantityError: String? = null,
    val caffeineError: String? = null,
    val costError: String? = null,
    val dateError: String? = null,
    val timeError: String? = null,
    val success: Boolean = false,

```

```

        val errorMsg: String? = null
    )

```

Листинг А.14 – Файл CoffeeInputFragment.kt

```

package com.example.coffies.ui.addcoffee

class CoffeeInputFragment : Fragment() {

    private var _binding: FragmentCoffeeInputBinding? = null
    private val binding get() = _binding!!

    private val viewModel: CoffeeInputViewModel by viewModels {
        CoffeeInputViewModelFactory(AppDatabase.getInstance(requireContext())
        )
    }

    private var coffeeTypes: List<CoffeeType> = emptyList()
    private var spinnerAdapter: ArrayAdapter<String>? = null

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentCoffeeInputBinding.inflate(inflater,
        container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState:
    Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.addCoffeeType.setOnClickListener {

            findNavController().navigate(R.id.action_main_to_addTypeCoffee)
        }

        lifecycleScope.launch {
            coffeeTypes = viewModel.getCoffeeTypesOnce()
            val names = coffeeTypes.map { it.name }
            spinnerAdapter = ArrayAdapter(
                requireContext(),
                android.R.layout.simple_dropdown_item_1line,
                names
            )
            binding.coffeeTypeSpinner.setAdapter(spinnerAdapter)
        }

        fun handleCoffeeTypeClick() {
            if (coffeeTypes.isEmpty()) {
                Toast.makeText(

```

```

        requireContext(),
        "Добавьте хотя бы один вид кофе для
продолжения.",
        Toast.LENGTH_SHORT
    ).show()
    } else {
        binding.coffeeTypeSpinner.showDropDown()
    }
}

binding.coffeeTypeInputLayout.setOnClickListener {
handleCoffeeTypeClick() }
binding.coffeeTypeInputLayout.setEndIconOnClickListener {
handleCoffeeTypeClick() }
binding.coffeeTypeSpinner.setOnClickListener {
handleCoffeeTypeClick() }

binding.coffeeTypeSpinner.setOnItemClickListener { _, _,
position, _ ->
    val type = coffeeTypes.getOrNull(position)
    if (type != null) {
        viewModel.onCoffeeTypeSelected(type)
        binding.coffeeTypeInputLayout.isHintEnabled =
false
    }
}

binding.coffeeTypeSpinner.addTextChangedListener {
    if (it.isNullOrEmpty()) {
        binding.coffeeTypeInputLayout.isHintEnabled = true
    }
}

binding.volumeInput.addTextChangedListener {
    viewModel.onFieldChanged("volume", it?.toString() ?:
"" )
}
binding.quantityValue.addTextChangedListener {
    viewModel.onFieldChanged("quantity", it?.toString() ?:
"" )
}
binding.caffeineInput.addTextChangedListener {
    viewModel.onFieldChanged("caffeine", it?.toString() ?:
"" )
}
binding.costInput.addTextChangedListener {
    viewModel.onFieldChanged("cost", it?.toString() ?: "" )
}
binding.dateInput.setOnClickListener {
    val cal = Calendar.getInstance()
    val dialog = DatePickerDialog(
        requireContext(), R.style.CoffiesDialogTheme,

```

```

        { _, y, mon, day ->
            val display = String.format("%02d.%02d.%04d",
day, mon + 1, y)
            val iso = String.format("%04d-%02d-%02d", y,
mon + 1, day)
            viewModel.onDatePicked(display, iso)
        },
        cal.get(Calendar.YEAR),
        cal.get(Calendar.MONTH),
        cal.get(Calendar.DAY_OF_MONTH)
    )
    dialog.datePicker.maxDate = System.currentTimeMillis()
    dialog.show()
}
binding.timeInput.setOnClickListener {
    val cal = Calendar.getInstance()
    TimePickerDialog(
        requireContext(), R.style.CoffiesDialogTheme,
        { _, h, m ->
            val formatted = String.format("%02d:%02d", h,
m)
            viewModel.onTimePicked(formatted)
        },
        cal.get(Calendar.HOUR_OF_DAY),
        cal.get(Calendar.MINUTE),
        true
    ).show()
}

binding.coffeeShopInput.addTextChangedListener {
    viewModel.onFieldChanged("place", it?.toString() ?:
"" )
}
binding.commentInput.addTextChangedListener {
    viewModel.onFieldChanged("comment", it?.toString() ?:
"" )
}

binding.quantityPlus.setOnClickListener {
    val current =
binding.quantityValue.text.toString().toIntOrNull() ?: 1
    binding.quantityValue.setText((current +
1).toString())
}
binding.quantityMinus.setOnClickListener {
    val current =
binding.quantityValue.text.toString().toIntOrNull() ?: 1
    if (current > 1) {
        binding.quantityValue.setText((current -
1).toString())
    }
}
}

```

```

        binding.saveButton.setOnClickListener {
            viewModel.submit()
        }

        lifecycleScope.launch {
            viewModel.formState.collect { state ->
                if (binding.coffeeTypeSpinner.text.toString() !=
                    state.coffeeType?.name.orEmpty()) {

binding.coffeeTypeSpinner.setText(state.coffeeType?.name.orEmpty()
, false)

                    binding.coffeeTypeInputLayout.isHintEnabled =
state.coffeeType?.name.isNullOrEmpty()
                }
                if (binding.volumeInput.text.toString() !=
state.volume) {
                    binding.volumeInput.setText(state.volume)
                }
                if (binding.quantityValue.text.toString() !=
state.quantity) {
                    binding.quantityValue.setText(state.quantity)
                }
                if (binding.caffeineInput.text.toString() !=
state.caffeine) {
                    binding.caffeineInput.setText(state.caffeine)
                }
                if (binding.dateInput.text.toString() !=
state.date) {
                    binding.dateInput.setText(state.date)
                }
                if (binding.timeInput.text.toString() !=
state.time) {
                    binding.timeInput.setText(state.time)
                }
                if (binding.costInput.text.toString() !=
state.cost) {
                    binding.costInput.setText(state.cost)
                }
                if (binding.coffeeShopInput.text.toString() !=
state.place) {
                    binding.coffeeShopInput.setText(state.place)
                }
                if (binding.commentInput.text.toString() !=
state.comment) {
                    binding.commentInput.setText(state.comment)
                }

                    binding.coffeeTypeInputLayout.error =
state.coffeeTypeError
                    binding.volumeInput.error = state.volumeError
                    binding.quantityValue.error = state.quantityError

```

```

        binding.caffeineInput.error = state.caffeineError
        binding.costInput.error = state.costError
        binding.dateInput.error = state.dateError
        binding.timeInput.error = state.timeError

        state.errorMsg?.let {
            Toast.makeText(requireContext(), "Ошибка при
сохранении: $it", Toast.LENGTH_LONG).show()
        }
        if (state.success) {
            clearFields()
            Toast.makeText(requireContext(),
R.string.saved, Toast.LENGTH_LONG).show()
            viewModel.clearSuccessFlag()
        }
    }
}

private fun clearFields() {
    binding.coffeeTypeSpinner.setText("")
    binding.volumeInput.text?.clear()
    binding.caffeineInput.text?.clear()
    binding.costInput.text?.clear()
    binding.timeInput.text?.clear()
    binding.dateInput.text?.clear()
    binding.dateInput.tag = null
    binding.coffeeShopInput.text?.clear()
    binding.commentInput.text?.clear()
    binding.quantityValue.setText("1")
}

override fun onDestroyView() {
    super.onDestroyView()
    _binding = null
}
}

```

Листинг A.15 – Файл CoffeeInputViewModel.kt

```

package com.example.coffies.ui.addcoffee

class CoffeeInputViewModel(
    private val db: AppDatabase
) : ViewModel() {

    val _formState = MutableStateFlow(CoffeeInputFormState())
    val formState: StateFlow<CoffeeInputFormState> = _formState

    suspend fun getCoffeeTypesOnce(): List<CoffeeType> =
        withContext(Dispatchers.IO) {
            db.coffeeTypeDao().getAll().first()
        }
}

```

```

fun onCoffeeTypeSelected(type: CoffeeType?) {
    if (type == null) {
        _formState.value = CoffeeInputFormState()
        return
    }
    val now = Calendar.getInstance().time
    val dispDate = SimpleDateFormat("dd.MM.yyyy",
Locale.getDefault()).format(now)
    val isoDate = SimpleDateFormat("yyyy-MM-dd",
Locale.getDefault()).format(now)
    val time = SimpleDateFormat("HH:mm",
Locale.getDefault()).format(now)
    _formState.value = _formState.value.copy(
        coffeeType = type,
        volume = type.default_volume_ml?.toString() ?: "",
        caffeine = type.default_caffeine_mg_ml?.let { mg ->
type.default_volume_ml?.let { v -> "%.1f".format(mg * v) } } ?:
"",
        quantity = "1",
        date = dispDate,
        dateIso = isoDate,
        time = time,
        coffeeTypeError = null
    )
}

fun onFieldChanged(field: String, value: String) {
    _formState.value = when (field) {
        "volume" -> _formState.value.copy(volume = value,
volumeError = null)
        "quantity" -> _formState.value.copy(quantity = value,
quantityError = null)
        "caffeine" -> _formState.value.copy(caffeine = value,
caffeineError = null)
        "cost" -> _formState.value.copy(cost = value,
costError = null)
        "date" -> _formState.value.copy(date = value,
dateError = null)
        "time" -> _formState.value.copy(time = value,
timeError = null)
        "place" -> _formState.value.copy(place = value)
        "comment" -> _formState.value.copy(comment = value)
        else -> _formState.value
    }
}

fun clearSuccessFlag() {
    _formState.value = _formState.value.copy(success = false,
errorMsg = null)
}

fun submit() {

```

```

val state = _formState.value
var ok = true

var coffeeTypeError: String? = null
var volumeError: String? = null
var quantityError: String? = null
var caffeineError: String? = null
var costError: String? = null
var dateError: String? = null
var timeError: String? = null

val volume = state.volume.toIntOrNull()
val quantity = state.quantity.toIntOrNull()
val caffeine = state.caffeine.replace(',', ' ',
'.').toDoubleOrNull()
val cost = state.cost.replace(',', ' ').toDoubleOrNull()

if (state.coffeeType == null) {
    coffeeTypeError = "Выберите тип кофе"
    ok = false
}
if (volume == null || volume <= 0) {
    volumeError = "Введите объем"
    ok = false
}
if (quantity == null || quantity <= 0) {
    quantityError = "Введите количество"
    ok = false
}
if (caffeine == null || caffeine <= 0.0) {
    caffeineError = "Введите кофеин"
    ok = false
}
if (state.cost.isBlank()) {
    costError = "Введите стоимость"
    ok = false
} else if (cost == null || cost < 0.0) {
    costError = "Некорректная цена"
    ok = false
}

if (state.dateIso.isBlank()) {
    dateError = "Введите дату"
    ok = false
}
if (state.time.isBlank()) {
    timeError = "Введите время"
    ok = false
}

if (!ok) {
    _formState.value = state.copy(

```

```

        coffeeTypeError = coffeeTypeError,
        volumeError = volumeError,
        quantityError = quantityError,
        caffeineError = caffeineError,
        costError = costError,
        dateError = dateError,
        timeError = timeError
    )
    return
}

viewModelScope.launch {
    try {
        val entry = CoffeeEntry(
            coffee_type_id = state.coffeeType!!.id,
            volume_ml = volume!!,
            quantity = quantity!!,
            caffeine_mg = caffeine!!,
            price = cost,
            date = state.dateIso,
            time = state.time,
            place = state.place.ifBlank { null },
            comment = state.comment.ifBlank { null }
        )
        withContext(Dispatchers.IO) {
            db.coffeeEntryDao().insert(entry)
        }
        _formState.value = CoffeeInputFormState(success =
true)
    } catch (e: Exception) {
        _formState.value = _formState.value.copy(errorMsg
= e.message)
    }
}

fun onDatePicked(displayDate: String, isoDate: String) {
    _formState.value = _formState.value.copy(date =
displayDate, dateIso = isoDate, dateError = null)
}

fun onTimePicked(time: String) {
    _formState.value = _formState.value.copy(time = time,
timeError = null)
}
}

```

Листинг A.16 – Файл MoodInputFormState.kt

```

package com.example.coffies.ui.addmood

import com.example.coffies.database.coffeentry.CoffeeEntry

```

```
data class MoodInputFormState(
    val moodLevel: Int = 0,
    val momentType: MoodMomentType? = MoodMomentType.BEFORE,
    val relatedCoffeeId: Int? = null,
    val relatedCoffeeTime: String? = null,
    val relatedCoffeeDate: String? = null,
    val coffeeList: List<CoffeeEntry> = emptyList(),
    val coffeeTypeNameMap: Map<Int, String> = emptyMap(),
    val date: String = "",
    val displayDate: String = "",
    val time: String = "",
    val comment: String = "",
    val moodError: String? = null,
    val momentError: String? = null,
    val relatedCoffeeError: String? = null,
    val dateError: String? = null,
    val timeError: String? = null,
    val success: Boolean = false
)
```

Листинг А.17 – Файл MoodInputFragment.kt

```
package com.example.coffies.ui.addmood

class MoodInputFragment : Fragment() {

    private var _binding: FragmentMoodInputBinding? = null
    private val binding get() = _binding!!
    private val viewModel: MoodInputViewModel by viewModels {

        MoodInputViewModelFactory(AppDatabase.getInstance(requireContext())
        )
    }

    override fun onCreateView(inflater: LayoutInflater, container:
    ViewGroup?, savedInstanceState: Bundle?): View {
        _binding = FragmentMoodInputBinding.inflate(inflater,
        container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState:
    Bundle?) {
        prefillFromArguments()
        observeState()

        listOf(binding.mood1, binding.mood2, binding.mood3,
        binding.mood4, binding.mood5).forEachIndexed { idx, iv ->
            iv.setOnClickListener { viewModel.onMoodSelected(idx +
            1) }
        }

        binding.momentRadioGroup.check(R.id.moment_before)
```

```

        binding.momentRadioGroup.setOnCheckedChangeListener { _,
checkedId ->
            val moment = when (checkedId) {
                R.id.moment_before -> MoodMomentType.BEFORE
                R.id.moment_after -> MoodMomentType.AFTER
                else -> null
            }
            viewModel.onMomentSelected(moment)
        }

        binding.relatedEntryInputLayout.setEndIconOnClickListener
{ showCoffeeDialog() }
        binding.relatedEntrySpinner.setOnClickListener {
showCoffeeDialog() }

        binding.relatedEntrySpinner.setOnFocusChangeListener { _,
hasFocus ->
            if (hasFocus &&
binding.relatedEntrySpinner.text.isEmpty()) {
                binding.relatedEntrySpinner.setText("")
            }
        }

        binding.dateInput.setOnClickListener { showDatePicker() }
        binding.timeInput.setOnClickListener { showTimePicker() }

        binding.commentInput.addTextChangedListener {
viewModel.onCommentChanged(it?.toString() ?: "") }

        binding.saveButton.setOnClickListener { viewModel.submit()
    }

    private fun prefillFromArguments() {
        val args = arguments ?: return

        val relatedCoffeeId = args.getInt("relatedCoffeeId", -
1).takeIf { it > 0 }
        val relatedCoffeeTime =
args.getString("relatedCoffeeTime")
        val relatedCoffeeDate =
args.getString("relatedCoffeeDate")
        val momentTypeStr = args.getString("momentType")
        val momentType = when (momentTypeStr) {
            "before and during", "before", "BEFORE" ->
MoodMomentType.BEFORE
            "after", "AFTER" -> MoodMomentType.AFTER
            else -> null
        }

        viewModel.prefill(relatedCoffeeId, relatedCoffeeTime,

```

```

relatedCoffeeDate, momentType)
    }

    private fun observeState() {
        lifecycleScope.launch {
            viewModel.formState.collect { st ->
                listOf(binding.mood1, binding.mood2,
binding.mood3, binding.mood4, binding.mood5).forEachIndexed { idx,
iv ->
                    iv.isSelected = idx + 1 == st.moodLevel
                    iv.background = if (iv.isSelected)
ContextCompat.getDrawable(requireContext(),
R.drawable.mood_selector_background) else null
                }
                binding.momentRadioGroup.check(
                    when (st.momentType) {
                        MoodMomentType.BEFORE ->
R.id.moment_before
                        MoodMomentType.AFTER -> R.id.moment_after
                        else -> -1
                    }
                )
                val relatedText = st.relatedCoffeeId?.let { id ->
                    st.coffeeList.firstOrNull { it.id == id }?.let
{ entry ->
                        val typeName =
st.coffeeTypeNameMap[entry.coffee_type_id] ?: "Тип
${entry.coffee_type_id}"
                        val displayDate = entry.date.split("-"
).let {
                            if (it.size == 3)
"${it[2]}.${it[1]}.${it[0]}" else entry.date
                        }
                        "$typeName • ${entry.volume_ml} мл •
${entry.time} $displayDate"
                    }
                } ?: ""
                if (relatedText.isEmpty() &&
binding.relatedEntrySpinner.text.isNotEmpty()) {
                    binding.relatedEntrySpinner.setText("")
                }
                if (relatedText.isNotEmpty() &&
binding.relatedEntrySpinner.text.toString() != relatedText) {
binding.relatedEntrySpinner.setText(relatedText, false)
                }
                if (relatedText.isEmpty()) {
                    binding.relatedEntrySpinner.hint = "Выберите
приём кофе"
                } else {
                    binding.relatedEntrySpinner.hint = ""
                }
            }
        }
    }

```

```

        }
        binding.dateInput.setText(st.displayDate)
        binding.timeInput.setText(st.time)
        if (binding.commentInput.text.toString() !=
st.comment) {
            binding.commentInput.setText(st.comment)

binding.commentInput.setSelection(st.comment.length)
        }

        binding.moodError.text = st.moodError
        binding.moodError.visibility = if (st.moodError !=
null) View.VISIBLE else View.GONE
        binding.momentError.text = st.momentError
        binding.momentError.visibility = if
(st.momentError != null) View.VISIBLE else View.GONE
        binding.relatedCoffeeError.text =
st.relatedCoffeeError
        binding.relatedCoffeeError.visibility = if
(st.relatedCoffeeError != null) View.VISIBLE else View.GONE

        val datetimeError = st.dateError ?: st.timeError
        binding.datetimeError.text = datetimeError
        binding.datetimeError.visibility = if
(datetimeError != null) View.VISIBLE else View.GONE

        if (st.success) {
            Toast.makeText(requireContext(), "Настройка
успешно сохранено", Toast.LENGTH_SHORT).show()
            binding.relatedEntrySpinner.setText("", false)
            binding.relatedEntrySpinner.hint = "Выберите
приём кофе"

            viewModel.resetForm()
        }
    }
}

private fun showCoffeeDialog() {
    val st = viewModel.formState.value
    val context = requireContext()
    val list = st.coffeeList
    if (list.isEmpty()) {
        Toast.makeText(context, "Нет подходящих приемов кофе",
Toast.LENGTH_SHORT).show()
        return
    }
    val items = list.map { entry ->
        val typeName =
st.coffeeTypeNameMap[entry.coffee_type_id] ?: "Тип
${entry.coffee_type_id}"
        val displayDate = entry.date.split("-").let {

```

```

        if (it.size == 3) "${it[2]}.${it[1]}.${it[0]}"
else entry.date
    }
    "$typeName • ${entry.volume_ml} мл • ${entry.time}
$displayDate"
    }.toTypedArray()

    AlertDialog.Builder(context)
        .setTitle("Выберите прием кофе")
        .setItems(items) { _, which ->
            viewModel.onRelatedCoffeeSelected(list[which])
        }
        .setNegativeButton("Отмена", null)
        .show()
    }

private fun showDatePicker() {
    val now = Calendar.getInstance()
    val dialog = DatePickerDialog(
        requireContext(),
        R.style.CoffiesDialogTheme,
        { _, y, m, d ->
            val display = String.format("%02d.%02d.%04d", d, m
+ 1, y)
            val iso = String.format("%04d-%02d-%02d", y, m +
1, d)
            viewModel.onDateChanged(display, iso)
        },
        now.get(Calendar.YEAR),
        now.get(Calendar.MONTH),
        now.get(Calendar.DAY_OF_MONTH)
    )
    dialog.datePicker.maxDate = System.currentTimeMillis()
    dialog.show()
}

private fun showTimePicker() {
    val now = Calendar.getInstance()
    val dialog = TimePickerDialog(
        requireContext(),
        R.style.CoffiesDialogTheme,
        { _, h, min ->
            val formatted = String.format("%02d:%02d", h, min)
            viewModel.onTimeChanged(formatted)
        },
        now.get(Calendar.HOUR_OF_DAY),
        now.get(Calendar.MINUTE),
        true
    )
    dialog.show()
}

```

```

        override fun onDestroyView() {
            super.onDestroyView()
            _binding = null
        }
    }
}

```

Листинг А.18 – Файл MoodInputViewModel.kt

```

package com.example.coffies.ui.addmood

class MoodInputViewModel(private val db: AppDatabase) :
    ViewModel() {

    private val _formState =
        MutableStateFlow(MoodInputFormState(momentType =
            MoodMomentType.BEFORE))
    val formState: StateFlow<MoodInputFormState> =
        _formState.asStateFlow()

    init {
        loadCoffeeTypeNames()
        refreshCoffeeList()
    }

    fun prefill(
        relatedCoffeeId: Int?,
        relatedCoffeeTime: String?,
        relatedCoffeeDate: String?,
        momentType: MoodMomentType?
    ) {
        _formState.update { st ->
            st.copy(
                relatedCoffeeId = relatedCoffeeId,
                relatedCoffeeTime = relatedCoffeeTime,
                relatedCoffeeDate = relatedCoffeeDate,
                momentType = momentType,
                date = relatedCoffeeDate ?: "",
                displayDate = relatedCoffeeDate?.split("-")?.reversed()?.joinToString(".") ?: "",
                time = relatedCoffeeTime ?: ""
            )
        }
        refreshCoffeeList()
    }

    private fun loadCoffeeTypeNames() {
        viewModelScope.launch {
            val map =
                db.coffeeTypeDao().getAllCoffeeTypeNames().associate { it.id to
                    it.name }
            _formState.update { it.copy(coffeeTypeNameMap = map) }
        }
    }
}

```

```

    }

    fun onMoodSelected(level: Int) {
        _formState.update { it.copy(moodLevel = level, moodError =
null) }
    }

    fun onMomentSelected(moment: MoodMomentType?) {
        _formState.update { st ->
            val relatedTime = st.relatedCoffeeTime
            val relatedDate = st.relatedCoffeeDate
            var newTime = st.time
            var newDate = st.date

            if (moment != null && relatedTime != null &&
relatedDate != null) {
                val cmp =
"$${st.date}T${st.time}".compareTo("$relatedDate" + "T" +
relatedTime)
                if (moment == MoodMomentType.BEFORE && cmp > 0) {
                    newTime = relatedTime
                    newDate = relatedDate
                }
                if (moment == MoodMomentType.AFTER && cmp < 0) {
                    val nowTime =
java.time.LocalDateTime.now().toString().substring(0,5)
                    newTime = nowTime
                    newDate = relatedDate
                }
            }
            st.copy(
                momentType = moment,
                momentError = null,
                time = newTime,
                date = newDate,
                displayDate = if (newDate.isNotEmpty())
newDate.split("-").reversed().joinToString(".") else ""
            )
        }
        refreshCoffeeList()
    }

    fun onRelatedCoffeeSelected(entry: CoffeeEntry?) {
        _formState.update { st ->
            if (entry == null) return@update st.copy(
                relatedCoffeeId = null,
                relatedCoffeeError = null,
                relatedCoffeeTime = null,
                relatedCoffeeDate = null,
                date = "",
                displayDate = "",
                time = ""
            )
        }
    }

```

```

    )
    val newDate = entry.date
    val newTime = entry.time
    st.copy(
        relatedCoffeeId = entry.id,
        relatedCoffeeError = null,
        relatedCoffeeTime = entry.time,
        relatedCoffeeDate = entry.date,
        date = newDate,
        displayDate = if (newDate.isNotEmpty())
newDate.split("-").reversed().joinToString(".") else "",
        time = newTime
    )
    }
}

fun onDateChanged(display: String, iso: String) {
    _formState.update { it.copy(date = iso, displayDate =
display, dateError = null, timeError = null) }
}

fun onTimeChanged(value: String) {
    _formState.update { it.copy(time = value, dateError =
null, timeError = null) }
}

fun onCommentChanged(text: String) {
    _formState.update { it.copy(comment = text) }
}

private fun refreshCoffeeList() {
    viewModelScope.launch {
        val usedBefore =
db.coffeeMoodLinkDao().getCoffeeEntryIdsByType(MoodMomentType.BEFO
RE.dbValue)
        val usedAfter =
db.coffeeMoodLinkDao().getCoffeeEntryIdsByType(MoodMomentType.AFTE
R.dbValue)
        val coffeeList =
db.coffeeEntryDao().getAllDescLimited(20).first()
        val filter = _formState.value.momentType
        val filtered = when (filter) {
            MoodMomentType.BEFORE -> coffeeList.filter { it.id
!in usedBefore }
            MoodMomentType.AFTER -> coffeeList.filter { it.id
!in usedAfter }
            else -> coffeeList
        }

        val currentRelatedId =
_formState.value.relatedCoffeeId
        val isRelatedValid = currentRelatedId != null &&

```

```

filtered.any { it.id == currentRelatedId }
    _formState.update {
        it.copy(
            coffeeList = filtered,
            relatedCoffeeId = if (isRelatedValid)
currentRelatedId else null,
            relatedCoffeeTime = if (isRelatedValid)
it.relatedCoffeeTime else null,
            relatedCoffeeDate = if (isRelatedValid)
it.relatedCoffeeDate else null
        )
    }
}

fun submit() {
    viewModelScope.launch {
        val st = _formState.value
        var valid = true
        var moodError: String? = null
        var momentError: String? = null
        var relatedCoffeeError: String? = null
        var dateError: String? = null
        var timeError: String? = null

        if (st.moodLevel !in 1..5) {
            moodError = "Выберите настроение"
            valid = false
        }
        if (st.momentType == null) {
            momentError = "Выберите момент"
            valid = false
        }
        if (st.relatedCoffeeId == null) {
            relatedCoffeeError = "Свяжите с приемом кофе"
            valid = false
        }
        if (st.date.isBlank()) {
            dateError = "Выберите дату"
            valid = false
        }
        if (st.time.isBlank()) {
            timeError = "Выберите время"
            valid = false
        }
        if (st.momentType != null && st.relatedCoffeeTime !=
null && st.relatedCoffeeDate != null) {
            val cmp =
"$${st.date}T${st.time}".compareTo("$${st.relatedCoffeeDate}T${st.re
latedCoffeeTime}")
            if (st.momentType == MoodMomentType.BEFORE && cmp
> 0) {

```

```

        timeError = "Дата/время должны быть не позже
приёма кофе"
        valid = false
    }
    if (st.momentType == MoodMomentType.AFTER && cmp <
0) {
        timeError = "Дата/время должны быть не раньше
приёма кофе"
        valid = false
    }
}

val now = java.time.LocalDateTime.now()
val inputDT = try {
java.time.LocalDateTime.parse("${st.date}T${st.time}") } catch (_:
Exception) { null }
    if (inputDT != null && inputDT > now) {
        timeError = "Дата/время не могут быть из будущего"
        valid = false
    }

    _formState.update {
        it.copy(
            moodError = moodError,
            momentError = momentError,
            relatedCoffeeError = relatedCoffeeError,
            dateError = dateError,
            timeError = timeError
        )
    }

    if (!valid) return@launch

    try {
        val moodId = db.moodEntryDao().insert(
            MoodEntry(
                mood_level = st.moodLevel,
                date = st.date,
                time = st.time,
                comment = st.comment.ifBlank { null }
            )
        ).toInt()
        db.coffeeMoodLinkDao().insert(
            CoffeeMoodLink(
                coffee_entry_id = st.relatedCoffeeId!!,
                mood_entry_id = moodId,
                relation_type = st.momentType!!.dbValue
            )
        )
        _formState.update { it.copy(success = true) }
        resetForm()
    } catch (e: Exception) {

```

```

        // handle error
    }
}

fun resetForm() {
    _formState.value = MoodInputFormState(momentType =
MoodMomentType.BEFORE)
    loadCoffeeTypeNames()
    refreshCoffeeList()
}
}

```

Листинг A.19 – Файл MoodMomentType.kt

```

package com.example.coffies.ui.addmood

enum class MoodMomentType(val dbValue: String) {
    BEFORE("before and during"),
    AFTER("after")
}

```

Листинг A.20 – Файл AnalyticsFragment.kt

```

package com.example.coffies.ui.analitics

class AnalyticsFragment : Fragment() {
    private var _binding: FragmentAnalyticsBinding? = null
    private val binding get() = _binding!!

    private val viewModel: AnalyticsViewModel by viewModels {
        AnalyticsViewModelFactory(requireContext())
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentAnalyticsBinding.inflate(inflater,
container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        binding.caffeineCard.setOnClickListener { }
        binding.cupsCard.setOnClickListener { }
        binding.moneyCard.setOnClickListener { }
        binding.moodCard.setOnClickListener { }

        viewLifecycleOwner.lifecycleScope.launch {
            viewModel.state.collect { state ->
                binding.caffeineTitle.text =

```

```

getString(R.string.caffeine_legend)
        binding.cupsTitle.text =
getString(R.string.cups_legend)
        binding.moneyTitle.text =
getString(R.string.money_legend)
        binding.moodTitle.text =
getString(R.string.mood_legend)

        setupLineChart(
            chart = binding.caffeineChart,
            points = state.caffeinePoints,
            lineColor =
ContextCompat.getColor(requireContext(), R.color.brown),
            fillColor =
ContextCompat.getColor(requireContext(), R.color.dark_beige),
            yValueFormatter = null
        )
        binding.caffeineSummary.text =
getString(R.string.caffeine_summary, state.caffeineAvg)
        if (state.caffeineGoal != null) {
            binding.caffeineGoal.visibility = View.VISIBLE
            binding.caffeineGoal.text =
getString(R.string.caffeine_goal_template, state.caffeineGoal)
        } else {
            binding.caffeineGoal.visibility = View.GONE
        }
        binding.caffeineSummary.setTextColor(
            if (state.isCaffeineExceeded)
                ContextCompat.getColor(requireContext(),
R.color.red)
            else ContextCompat.getColor(requireContext(),
R.color.black)
        )

        setupLineChart(
            chart = binding.cupsChart,
            points = state.cupsPoints,
            lineColor =
ContextCompat.getColor(requireContext(), R.color.black),
            fillColor =
ContextCompat.getColor(requireContext(), R.color.dark_beige),
            yValueFormatter = null
        )
        binding.cupsSummary.text =
getString(R.string.cups_summary, state.cupsSum)
        if (state.cupsGoal != null) {
            binding.cupsGoal.visibility = View.VISIBLE
            binding.cupsGoal.text =
getString(R.string.cups_goal_template, state.cupsGoal)
        } else {
            binding.cupsGoal.visibility = View.GONE
        }
    }

```

```

        binding.cupsSummary.setTextColor(
            if (state.isCupsExceeded)
                ContextCompat.getColor(requireContext(),
R.color.red)
            else ContextCompat.getColor(requireContext(),
R.color.black)
        )

        setupLineChart(
            chart = binding.moneyChart,
            points = state.moneyPoints,
            lineColor =
ContextCompat.getColor(requireContext(), R.color.black),
            fillColor =
ContextCompat.getColor(requireContext(), R.color.dark_beige),
            yValueFormatter = MoneyYFormatter()
        )
        binding.moneySummary.text =
getString(R.string.money_summary, state.moneySum)
        if (state.moneyGoal != null) {
            binding.moneyGoal.visibility = View.VISIBLE
            binding.moneyGoal.text =
getString(R.string.money_goal_template, state.moneyGoal)
        } else {
            binding.moneyGoal.visibility = View.GONE
        }
        binding.moneySummary.setTextColor(
            if (state.isMoneyExceeded)
                ContextCompat.getColor(requireContext(),
R.color.red)
            else ContextCompat.getColor(requireContext(),
R.color.black)
        )

        setupLineChart(
            chart = binding.moodChart,
            points = state.moodPoints,
            lineColor =
ContextCompat.getColor(requireContext(), R.color.black),
            fillColor =
ContextCompat.getColor(requireContext(), R.color.dark_beige),
            yValueFormatter = MoodAxisFormatter()
        )
        binding.moodSummary.text =
getString(R.string.mood_summary, moodLevelToText(state.moodAvg))

        binding.moodSummary.setTextColor(ContextCompat.getColor(requireCon
text(), R.color.black))
    }
}
}

```

```

override fun onDestroyView() {
    super.onDestroyView()
    _binding = null
}

private fun setupLineChart(
    chart: LineChart,
    points: List<Pair<String, Double>>,
    lineColor: Int,
    fillColor: Int,
    yValueFormatter: ValueFormatter? = null
) {
    val entries = points.mapIndexed { index, pair ->
        Entry(index.toFloat(), pair.second.toFloat())
    }
    val dataSet = LineDataSet(entries, "").apply {
        mode = LineDataSet.Mode.CUBIC_BEZIER
        setDrawValues(false)
        setDrawCircles(true)
        lineWidth = 2.2f
        circleRadius = 3.5f
        color = lineColor
        setCircleColor(lineColor)
        fillAlpha = 60
        setDrawFilled(true)
        setFillColor(fillColor)
    }
    chart.data = LineData(dataSet)
    chart.axisRight.isEnabled = false
    chart.description.isEnabled = false
    chart.legend.isEnabled = false
    chart.setExtraOffsets(0f, 0f, 0f, 20f)
    chart.axisLeft.setDrawGridLines(true)

    chart.setScaleEnabled(false)
    chart.isDoubleTapToZoomEnabled = false
    chart.isDragEnabled = false

    chart.setTouchEnabled(true)
    chart.isHighlightPerTapEnabled = true
    chart.isHighlightPerDragEnabled = true

    chart.xAxis.position = XAxis.XAxisPosition.BOTTOM
    chart.xAxis.setDrawGridLines(false)
    chart.xAxis.labelRotationAngle = -45f
    chart.xAxis.granularity = 1f
    chart.xAxis.valueFormatter = DayAxisFormatter(points.map {
it.first })
    chart.xAxis.axisMinimum = 0f
    chart.xAxis.axisMaximum = (entries.lastIndex).toFloat()

    if (yValueFormatter != null) {

```

```

        chart.axisLeft.valueFormatter = yValueFormatter
    } else {
        chart.axisLeft.valueFormatter = DefaultYFormatter()
    }

    val markerView = ChartMarkerView(
        requireContext(),
        points.map { it.first },
        { y -> yValueFormatter?.getFormattedValue(y) ?:
String.format("%.2f", y) }
    )
    markerView.chartView = chart
    chart.marker = markerView

    chart.invalidate()
}

class DayAxisFormatter(private val dates: List<String>) :
ValueFormatter() {
    override fun getFormattedValue(value: Float): String {
        val idx = value.toInt()
        return if (idx in dates.indices && dates[idx].length
== 10)
"$${dates[idx].substring(8,10)}.$${dates[idx].substring(5,7)}"
        else ""
    }
}

class DefaultYFormatter : ValueFormatter() {
    override fun getFormattedValue(value: Float): String {
        return value.toInt().toString()
    }
}

class MoneyYFormatter : ValueFormatter() {
    override fun getFormattedValue(value: Float): String {
        return if (value > 0) "%d".format(value.toInt()) else
"0"
    }
}

class MoodAxisFormatter : ValueFormatter() {
    private val moods = arrayOf("?", "😞", "😓", "😐", "😊",
"😄")
    override fun getFormattedValue(value: Float): String {
        val idx = value.toInt().coerceIn(0, 5)
        return moods[idx]
    }
}

```

```

private fun moodLevelToText(level: Int): String {
    return when (level) {
        1 -> getString(R.string.mood_1_short)
        2 -> getString(R.string.mood_2_short)
        3 -> getString(R.string.mood_3_short)
        4 -> getString(R.string.mood_4_short)
        5 -> getString(R.string.mood_5_short)
        else -> getString(R.string.no_data)
    }
}

```

Листинг A.21 – Файл AnalyticsState.kt

```

package com.example.coffies.ui.analitics

data class AnalyticsState(
    val caffeinePoints: List<Pair<String, Double>> = emptyList(),
    val caffeineAvg: String = "-",
    val caffeineGoal: String? = null,
    val isCaffeineExceeded: Boolean = false,
    val cupsPoints: List<Pair<String, Double>> = emptyList(),
    val cupsSum: String = "-",
    val cupsGoal: String? = null,
    val isCupsExceeded: Boolean = false,
    val moneyPoints: List<Pair<String, Double>> = emptyList(),
    val moneySum: String = "-",
    val moneyGoal: String? = null,
    val isMoneyExceeded: Boolean = false,
    val moodPoints: List<Pair<String, Double>> = emptyList(),
    val moodAvg: Int = 0
)

```

Листинг A.22 – Файл AnalyticsViewModel.kt

```

package com.example.coffies.ui.analitics

class AnalyticsViewModel(
    private val db: AppDatabase
) : ViewModel() {

    private val _state = MutableStateFlow<AnalyticsState>()
    val state: StateFlow<AnalyticsState> = _state

    init {
        loadAnalytics()
    }

    private fun loadAnalytics() {
        viewModelScope.launch {
            val formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd")

            val today = LocalDate.now()

```

```

        val days = (0..29).map { today.minusDays((29 -
it).toLong()) }
        val dayStrings = days.map { it.format(formatter) }

        val entries =
db.coffeeEntryDao().getAllBetweenDates(dayStrings.first(),
dayStrings.last())
        val moods =
db.moodEntryDao().getAllBetweenDates(dayStrings.first(),
dayStrings.last())
        val userSettings =
db.userSettingsDao().getSettingsOnce()

        val caffeineByDay = dayStrings.map { date ->
            val sumCaffeine = entries.filter { it.date == date
}.sumOf { it.caffeine_mg ?: 0.0 }
            date to sumCaffeine
        }
        val cupsByDay = dayStrings.map { date ->
            val sumCups = entries.filter { it.date == date
}.sumOf { it.quantity }.toDouble()
            date to sumCups
        }
        val moneyByDay = dayStrings.map { date ->
            val sumMoney = entries.filter { it.date == date
}.sumOf { it.price ?: 0.0 }
            date to sumMoney
        }
        val moodByDay = dayStrings.map { date ->
            val moodsForDay = moods.filter { it.date == date }
            val avgMood = if (moodsForDay.isNotEmpty())
moodsForDay.map { it.mood_level }.average() else 0.0
            date to avgMood
        }

        val caffeineAvgs = caffeineByDay.map { it.second
}.filter { it > 0 }
        val caffeineAvgNum = if (caffeineAvgs.isNotEmpty())
caffeineAvgs.average() else 0.0
        val caffeineAvg = if (caffeineAvgNum > 0) "%.0f
mΓ".format(caffeineAvgNum) else "-"
        val isCaffeineExceeded = caffeineAvgNum > 400
        val caffeineGoal = "400 mΓ"

        val cupsSumNum = cupsByDay.map { it.second }.filter {
it > 0 }.sum()
        val cupsSum = if (cupsSumNum > 0)
 "%.0f".format(cupsSumNum) else "-"
        val cupsGoal = userSettings?.monthly_cup_goal?.takeIf
{ it > 0 }?.toString()
        val isCupsExceeded =
userSettings?.monthly_cup_goal?.let { it > 0 && cupsSumNum > it }

```

```

?: false

        val moneySumNum = moneyByDay.map { it.second }.filter
{ it > 0 }.sum()
        val moneySum = if (moneySumNum > 0) "%.2f
₽".format(moneySumNum) else "-"
        val moneyGoal =
userSettings?.monthly_spend_goal?.takeIf { it > 0f }?.let { "%.2f
₽".format(it) }
        val isMoneyExceeded =
userSettings?.monthly_spend_goal?.let { it > 0f && moneySumNum >
it } ?: false

        val moodAvgs = moodByDay.map { it.second }.filter { it
> 0 }
        val moodAvg = if (moodAvgs.isNotEmpty())
ceil(moodAvgs.average()).toInt() else 0

        _state.value = AnalyticsState(
            caffeinePoints = caffeineByDay,
            caffeineAvg = caffeineAvg,
            caffeineGoal = caffeineGoal,
            isCaffeineExceeded = isCaffeineExceeded,
            cupsPoints = cupsByDay,
            cupsSum = cupsSum,
            cupsGoal = cupsGoal,
            isCupsExceeded = isCupsExceeded,
            moneyPoints = moneyByDay,
            moneySum = moneySum,
            moneyGoal = moneyGoal,
            isMoneyExceeded = isMoneyExceeded,
            moodPoints = moodByDay,
            moodAvg = moodAvg
        )
    }
}
}

```

Листинг A.23 – Файл CoffeeDetailsFragment.kt

```

package com.example.coffies.ui.coffeedetails

class CoffeeDetailsFragment : Fragment() {

    private var _binding: FragmentCoffeeDetailsBinding? = null
    private val binding get() = _binding!!

    private val args: CoffeeDetailsFragmentArgs by navArgs()
    private val viewModel: CoffeeDetailsViewModel by viewModels {
        CoffeeDetailsViewModelFactory(

com.example.coffies.database.AppDatabase.getInstance(requireContex
t()),

```

```

        args.coffeeEntryId
    )
}

override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View {
    _binding = FragmentCoffeeDetailsBinding.inflate(inflater,
container, false)
    return binding.root
}

override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
    viewModel.state
        .flowWithLifecycle(viewLifecycleOwner.lifecycle,
Lifecycle.State.STARTED)
        .onEach { state ->
            if (state.isLoading) return@onEach
            state.coffee?.let { bindCoffee(it, state) }
            bindMoodBlocks(state)
            state.error?.let {
                Toast.makeText(requireContext(), it,
Toast.LENGTH_SHORT).show()
            }
        }
        .launchIn(viewLifecycleOwner.lifecycleScope)

    requireActivity().addMenuProvider(object : MenuProvider {
        override fun onCreateMenu(menu: Menu, menuInflater:
MenuInflater) {
            menuInflater.inflate(R.menu.coffee_details_menu,
menu)
        }

        override fun onMenuItemSelected(menuItem: MenuItem):
Boolean {
            return when (menuItem.itemId) {
                R.id.action_delete -> {
                    showDeleteConfirmationDialog()
                    true
                }
                else -> false
            }
        }
    }, viewLifecycleOwner, Lifecycle.State.RESUMED)
}

private fun bindCoffee(entry:
com.example.coffies.database.coffeentry.CoffeeEntry, state:
CoffeeDetailsState) {

```

```

        binding.coffeeTypeValue.text =
state.coffeeTypeNames[entry.coffee_type_id] ?: "Тип
${entry.coffee_type_id}"
        binding.volumeValue.text = entry.volume_ml.toString()
        binding.quantityValue.text = entry.quantity.toString()
        binding.caffeineValue.text = String.format("%.0f",
entry.caffeine_mg)
        binding.priceValue.text = entry.price?.let { "${it}₽" } ?:
""
        binding.dateValue.text = formatDate(entry.date)
        binding.timeValue.text = entry.time
        binding.placeValue.text = entry.place ?:
getString(R.string.no_data)
        binding.commentValue.text = entry.comment ?:
getString(R.string.no_data)
    }

private fun bindMoodBlocks(state: CoffeeDetailsState) {
    bindMoodBlock(
        container = binding.moodBeforeContainer,
        label = binding.moodBeforeLabel,
        value = binding.moodBeforeValue,
        icon = binding.moodBeforeIcon,
        mood = state.moodMap["before and during"],
        momentType = "before and during"
    )
    bindMoodBlock(
        container = binding.moodAfterContainer,
        label = binding.moodAfterLabel,
        value = binding.moodAfterValue,
        icon = binding.moodAfterIcon,
        mood = state.moodMap["after"],
        momentType = "after"
    )
}

private fun bindMoodBlock(
    container: View,
    label: View,
    value: android.widget.TextView,
    icon: android.widget.ImageView,
    mood: com.example.coffies.database.moodentry.MoodEntry?,
    momentType: String
) {
    container.setOnClickListener {
        val entry = viewModel.state.value.coffee ?:
return@setOnClickListener
        if (mood != null) {
            findNavController().navigate(
                R.id.action_coffeeDetails_to_moodDetails,
                bundleOf("moodEntryId" to mood.id)
            )
        }
    }
}

```

```

        } else {
            findNavController().navigate(
                R.id.action_coffeeDetails_to_addMood,
                bundleOf(
                    "relatedCoffeeId" to entry.id,
                    "relatedCoffeeTime" to entry.time,
                    "relatedCoffeeDate" to entry.date,
                    "momentType" to momentType
                )
            )
        }
    }
    if (mood != null) {
        value.text = getString(
            R.string.mood_brief_template,
            moodLevelToText(mood.mood_level),
            mood.time,
            formatDate(mood.date)
        )
        icon.setImageResource(moodLevelToDrawable(mood.mood_level))
    } else {
        value.text = getString(R.string.no_data)
        icon.setImageResource(R.drawable.mood_3)
    }
}

private fun formatDate(iso: String): String {
    return try {
        val parsed = SimpleDateFormat("yyyy-MM-dd",
            Locale.getDefault()).parse(iso)
        SimpleDateFormat("dd.MM.yyyy",
            Locale.getDefault()).format(parsed!!)
    } catch (e: Exception) {
        iso
    }
}

private fun moodLevelToText(level: Int): String {
    return when (level) {
        1 -> getString(R.string.mood_1_short)
        2 -> getString(R.string.mood_2_short)
        3 -> getString(R.string.mood_3_short)
        4 -> getString(R.string.mood_4_short)
        5 -> getString(R.string.mood_5_short)
        else -> getString(R.string.no_data)
    }
}

private fun moodLevelToDrawable(level: Int): Int {
    return when (level) {
        1 -> R.drawable.mood_1
        2 -> R.drawable.mood_2
    }
}

```

```

        3 -> R.drawable.mood_3
        4 -> R.drawable.mood_4
        5 -> R.drawable.mood_5
        else -> R.drawable.mood_3
    }
}

private fun showDeleteConfirmationDialog() {
    androidx.appcompat.app.AlertDialog.Builder(
        requireContext(),
        R.style.CoffiesDeleteDialogTheme
    )
        .setTitle("Удалить запись?")
        .setMessage("Вы действительно хотите удалить эту запись кофе и все связанные с ней настройки? Это действие нельзя отменить.")
        .setPositiveButton("Удалить") { dialog, _ ->
            viewModel.deleteWithMoods {
                parentFragmentManager.setFragmentResult(
                    "COFFEE_DELETED",
                    bundleOf("deleted_id" to
args.coffeeEntryId)
                )
                activity?.runOnUiThread {
findNavController().popBackStack() }
            }
            dialog.dismiss()
        }
        .setNegativeButton("Отмена") { dialog, _ ->
dialog.dismiss() }
        .show()
    }

    override fun onResume() {
        super.onResume()
        viewModel.loadAll()
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}

```

Листинг А.24 – Файл CoffeeDetailsState.kt

```

package com.example.coffies.ui.coffeedetails

import com.example.coffies.database.coffeentry.CoffeeEntry
import com.example.coffies.database.moodentry.MoodEntry

data class CoffeeDetailsState(
    val coffee: CoffeeEntry? = null,

```

```

        val moodMap: Map<String, MoodEntry?> = emptyMap(),
        val coffeeTypeNames: Map<Int, String> = emptyMap(),
        val isLoading: Boolean = false,
        val error: String? = null
    )

```

Листинг А.25 – Файл CoffeeDetailsViewModel.kt

```

package com.example.coffies.ui.coffeedetails

class CoffeeDetailsViewModel(
    private val db: AppDatabase,
    private val coffeeEntryId: Int,
) : ViewModel() {

    private val _state = MutableStateFlow(CoffeeDetailsState())
    val state: StateFlow<CoffeeDetailsState> = _state

    fun loadAll() {
        _state.value = _state.value.copy(isLoading = true, error = null)

        viewModelScope.launch {
            val newState = kotlin.runCatching {
                val entry =
db.coffeeEntryDao().getById(coffeeEntryId)

                val coffeeTypeNames = db.coffeeTypeDao()
                    .getAllCoffeeTypeNames()
                    .associate { it.id to it.name }

                val moodLinks =
db.coffeeMoodLinkDao().getLinksForCoffeeEntry(coffeeEntryId)

                val moodMap = buildMap<String, MoodEntry?> {
                    for (link in moodLinks) {
                        val mood =
db.moodEntryDao().getById(link.mood_entry_id)
                        put(link.relation_type, mood)
                    }
                }

                CoffeeDetailsState(
                    isLoading = false,
                    coffee = entry,
                    moodMap = moodMap,
                    coffeeTypeNames = coffeeTypeNames,
                    error = null,
                )
            }.getOrElse { throwable ->
                _state.value.copy(isLoading = false, error =
throwable.message)
            }
        }
    }
}

```

```

        _state.value = newState
    }
}

fun deleteWithMoods(onDeleted: () -> Unit) {
    viewModelScope.launch {
        kotlin.runCatching {
            val links =
                db.coffeeMoodLinkDao().getLinksForCoffeeEntry(coffeeEntryId)
            val moodIds = links.map { it.mood_entry_id }

            links.forEach { db.coffeeMoodLinkDao().delete(it) }

            moodIds.forEach { id ->
                db.moodEntryDao().getById(id)?.let {
                    db.moodEntryDao().delete(it) }
            }
            db.coffeeEntryDao().deleteById(coffeeEntryId)
        }
        onDeleted()
    }
}
}
}

```

Листинг A.26 – Файл CoffeeTypeFormState.kt

```

package com.example.coffies.ui.coffeetype

data class CoffeeTypeFormState(
    val nameError: String? = null,
    val volumeError: String? = null,
    val caffeineError: String? = null,
    val success: Boolean = false
)

```

Листинг A.27 – Файл CoffeeTypeFragment.kt

```

package com.example.coffies.ui.coffeetype

class CoffeeTypeFragment : Fragment() {

    private var _binding: FragmentCoffeeTypeBinding? = null
    private val binding get() = _binding!!

    private val viewModel: CoffeeTypeViewModel by viewModels {
        val dao =
            AppDatabase.getInstance(requireContext()).coffeeTypeDao()
            CoffeeTypeViewModelFactory(dao)
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,

```

```

        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentCoffeeTypeBinding.inflate(inflater,
container, false)

        binding.saveTypeButton.setOnClickListener {
            viewModel.submit(
                binding.nameInput.text.toString().trim(),
                binding.defaultVolumeInput.text.toString().trim(),
binding.defaultCaffeineInput.text.toString().trim()
            )
        }

        binding.nameInput.addTextChangedListener {
viewModel.clearNameError() }
        binding.defaultVolumeInput.addTextChangedListener {
viewModel.clearVolumeError() }
        binding.defaultCaffeineInput.addTextChangedListener {
viewModel.clearCaffeineError() }

        lifecycleScope.launch {
            viewModel.formState.collectLatest { state ->
                binding.nameInput.error = state.nameError
                binding.defaultVolumeInput.error =
state.volumeError
                binding.defaultCaffeineInput.error =
state.caffeineError

                if (state.success) {
                    binding.nameInput.text?.clear()
                    binding.defaultVolumeInput.text?.clear()
                    binding.defaultCaffeineInput.text?.clear()
                    hideKeyboard()
                    Toast.makeText(requireContext(),
com.example.coffies.R.string.saved, Toast.LENGTH_LONG).show()
                    viewModel.clearSuccessFlag()
                }
            }
        }

        return binding.root
    }

    private fun hideKeyboard() {
        val imm =
requireContext().getSystemService(Context.INPUT_METHOD_SERVICE) as
InputMethodManager
        val view = requireActivity().currentFocus ?: binding.root
        imm.hideSoftInputFromWindow(view.windowToken, 0)
    }

```

```

        override fun onDestroyView() {
            super.onDestroyView()
            _binding = null
        }
    }
}

```

Листинг А.28 – Файл CoffeeTypeViewModel.kt

```

package com.example.coffies.ui.coffeetype

class CoffeeTypeViewModel(private val dao: CoffeeTypeDao) :
    ViewModel() {

    private val _formState =
        MutableStateFlow(CoffeeTypeFormState())
    val formState: StateFlow<CoffeeTypeFormState> = _formState

    fun submit(name: String, volStr: String, caffeineStr: String)
    {
        var valid = true
        var nameError: String? = null
        var volumeError: String? = null
        var caffeineError: String? = null

        val vol = volStr.toIntOrNull()
        val caffeine = caffeineStr.toFloatOrNull()

        if (name.isBlank()) {
            nameError = "Введите название"
            valid = false
        }
        if (volStr.isBlank()) {
            volumeError = "Введите объем"
            valid = false
        } else if (vol == null || vol <= 0) {
            volumeError = "Некорректное число"
            valid = false
        }
        if (caffeineStr.isBlank()) {
            caffeineError = "Введите кофеин (мг/мл)"
            valid = false
        } else if (caffeine == null || caffeine < 0f) {
            caffeineError = "Некорректное число"
            valid = false
        }

        if (!valid) {
            _formState.value = CoffeeTypeFormState(
                nameError = nameError,
                volumeError = volumeError,
                caffeineError = caffeineError,
                success = false
            )
        }
    }
}

```

```

        return
    }

    viewModelScope.launch {
        val coffeeType = CoffeeType(
            name = name,
            default_volume_ml = vol,
            default_caffeine_mg_ml = caffeine
        )
        dao.insert(coffeeType)
        _formState.value = CoffeeTypeFormState(success = true)
    }
}

fun clearNameError() {
    _formState.value = _formState.value.copy(nameError = null)
}

fun clearVolumeError() {
    _formState.value = _formState.value.copy(volumeError =
null)
}

fun clearCaffeineError() {
    _formState.value = _formState.value.copy(caffeineError =
null)
}

fun clearSuccessFlag() {
    _formState.value = _formState.value.copy(success = false)
}
}

```

Листинг А.29 – Файл HistoryAdapter.kt

```

package com.example.coffies.ui.history

class HistoryAdapter : ListAdapter<HistoryItem,
RecyclerView.ViewHolder>(DiffCallback()) {

    var onItemClick: ((HistoryItem) -> Unit)? = null

    override fun getItemViewType(position: Int): Int = when
(getItem(position)) {
        is HistoryItem.DateHeader -> VIEW_TYPE_DATE_HEADER
        is HistoryItem.CoffeeItem -> VIEW_TYPE_COFFEE
        is HistoryItem.MoodItem -> VIEW_TYPE_MOOD
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType:
Int): RecyclerView.ViewHolder {
        val inflater = LayoutInflater.from(parent.context)
        return when (viewType) {
            VIEW_TYPE_DATE_HEADER -> DateHeaderViewHolder(
                ItemHistoryDateHeaderBinding.inflate(inflater,
parent, false)

```

```

        )
        VIEW_TYPE_COFFEE -> CoffeeViewHolder(
            ItemHistoryEntryBinding.inflate(inflater, parent,
false)
        )
        VIEW_TYPE_MOOD -> MoodViewHolder(
            ItemHistoryEntryBinding.inflate(inflater, parent,
false)
        )
        else -> throw IllegalArgumentException("Unknown
viewType $viewType")
    }
}

    override fun onBindViewHolder(holder: RecyclerView.ViewHolder,
position: Int) {
        when (val item = getItem(position)) {
            is HistoryItem.DateHeader -> (holder as
DateHeaderViewHolder).bind(item)
            is HistoryItem.CoffeeItem -> (holder as
CoffeeViewHolder).bind(item)
            is HistoryItem.MoodItem -> (holder as
MoodViewHolder).bind(item)
        }
    }

    inner class DateHeaderViewHolder(private val binding:
ItemHistoryDateHeaderBinding) :
RecyclerView.ViewHolder(binding.root) {
        fun bind(item: HistoryItem.DateHeader) {
            val today = LocalDate.now()
            val yesterday = today.minusDays(1)
            binding.dateHeaderText.text = when (item.date) {
                today -> "Сегодня"
                yesterday -> "Вчера"
                else ->
item.date.format(DateTimeFormatter.ofPattern("dd.MM.yyyy"))
            }
        }
    }

    inner class CoffeeViewHolder(private val binding:
ItemHistoryEntryBinding) : RecyclerView.ViewHolder(binding.root) {
        init {
            binding.root.setOnClickListener {
                val position = adapterPosition
                if (position != RecyclerView.NO_POSITION) {
                    onItemClick?.invoke(getItem(position))
                }
            }
        }
    }
}

```

```

        fun bind(item: HistoryItem.CoffeeItem) {
            with(binding) {

entryIcon.setImageResource(R.drawable.coffee_cup_large)
                entryTitle.text = "Кофе: ${item.type}"
                val details = buildString {
                    append("${item.quantity} шт")
                    item.volume?.let { append(" • $it мл") }
                    item.price?.let { append(" • ${it}₽") }
                }
                entryDetails.text = details
                entryTime.text =
item.dateTime.format(DateTimeFormatter.ofPattern("HH:mm"))
            }
        }
    }

    inner class MoodViewHolder(private val binding:
ItemHistoryEntryBinding) : RecyclerView.ViewHolder(binding.root) {
        init {
            binding.root.setOnClickListener {
                val position = adapterPosition
                if (position != RecyclerView.NO_POSITION) {
                    onItemClick?.invoke(getItem(position))
                }
            }
        }
    }

    fun bind(item: HistoryItem.MoodItem) {
        with(binding) {

entryIcon.setImageResource(getMoodIcon(item.level))
                entryTitle.text = "Настроение"
                entryDetails.text = getMoodText(item.level)
                entryTime.text =
item.dateTime.format(DateTimeFormatter.ofPattern("HH:mm"))
            }
        }

        private fun getMoodIcon(level: Int): Int {
            return when (level) {
                1 -> R.drawable.mood_1
                2 -> R.drawable.mood_2
                3 -> R.drawable.mood_3
                4 -> R.drawable.mood_4
                5 -> R.drawable.mood_5
                else -> R.drawable.mood_3
            }
        }

        private fun getMoodText(level: Int): String {
            return when (level) {

```

```

        1 -> "Очень плохое"
        2 -> "Плохое"
        3 -> "Нормальное"
        4 -> "Хорошее"
        5 -> "Отличное"
        else -> "Неизвестно"
    }
}

class DiffCallback : DiffUtil.ItemCallback<HistoryItem>() {
    override fun areItemsTheSame(oldItem: HistoryItem,
newItem: HistoryItem): Boolean {
        return when {
            oldItem is HistoryItem.DateHeader && newItem is
HistoryItem.DateHeader -> oldItem.date == newItem.date
            oldItem is HistoryItem.CoffeeItem && newItem is
HistoryItem.CoffeeItem -> oldItem.id == newItem.id
            oldItem is HistoryItem.MoodItem && newItem is
HistoryItem.MoodItem -> oldItem.id == newItem.id
            else -> false
        }
    }

    override fun areContentsTheSame(oldItem: HistoryItem,
newItem: HistoryItem): Boolean {
        return oldItem == newItem
    }
}

companion object {
    private const val VIEW_TYPE_DATE_HEADER = 0
    private const val VIEW_TYPE_COFFEE = 1
    private const val VIEW_TYPE_MOOD = 2
}
}

```

Листинг А.30 – Файл HistoryFragment.kt

```

package com.example.coffies.ui.history

class HistoryFragment : Fragment() {

    private var _binding: FragmentHistoryBinding? = null
    private val binding get() = _binding!!

    private val viewModel: HistoryViewModel by viewModels {
        HistoryViewModelFactory(AppDatabase.getInstance(requireContext()))
    }

    private val historyAdapter = HistoryAdapter()
}

```

```

        override fun onCreateView(
            inflater: LayoutInflater,
            container: ViewGroup?,
            savedInstanceState: Bundle?
        ): View {
            _binding = FragmentHistoryBinding.inflate(inflater,
container, false)
            return binding.root
        }

        override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
            setupRecyclerView()
            setupDateRangePicker()
            setupSwipeRefresh()
            setupChipFilters()
            observeViewModel()
            observeDeleteResult()
        }

        private fun setupRecyclerView() {
            binding.historyRecyclerView.apply {
                layoutManager = LinearLayoutManager(requireContext())
                adapter = historyAdapter
            }

            historyAdapter.onItemClick = { item ->
                when (item) {
                    is HistoryItem.CoffeeItem -> {
                        val action =
HistoryFragmentDirections.actionHistoryToCoffeeDetails(item.id)
                        findNavController().navigate(action)
                    }
                    is HistoryItem.MoodItem -> {
                        val action =
HistoryFragmentDirections.actionHistoryToMoodDetails(item.id)
                        findNavController().navigate(action)
                    }
                    else -> Unit
                }
            }
        }

        private fun setupSwipeRefresh() {
            binding.swipeRefresh.setOnRefreshListener {
                viewModel.forceReload()
            }
        }

        private fun setupDateRangePicker() {
            binding.dateRangeLayout.setOnClickListener {
                val currentStart = viewModel.dateRange.value.first

```

```

        val currentEnd = viewModel.dateRange.value.second

        val todayMillis =
MaterialDatePicker.todayInUtcMilliseconds()
        val validator = MaxDateValidator(todayMillis)
        val constraints = CalendarConstraints.Builder()
            .setValidator(validator)
            .build()

        val picker =
MaterialDatePicker.Builder.dateRangePicker()
            .setTheme(R.style.CoffiesCalendarPickerTheme)
            .setTitleText("Выберите период")
            .setCalendarConstraints(constraints)
            .setSelection(
                androidx.core.util.Pair(

currentStart.atStartOfDay(ZoneId.of("UTC")).toInstant().toEpochMilli(),

currentEnd.atStartOfDay(ZoneId.of("UTC")).toInstant().toEpochMilli()
                )
            )
            .build()

        picker.addOnPositiveButtonClickListener { selection ->
            val start = Instant.ofEpochMilli(selection.first)
                .atZone(ZoneId.systemDefault()).toLocalDate()
            val end = Instant.ofEpochMilli(selection.second)
                .atZone(ZoneId.systemDefault()).toLocalDate()
            viewModel.updateDateRange(start, end)
        }

        picker.show(parentFragmentManager,
"date_range_picker")
    }

    private fun setupChipFilters() {
        binding.chipCoffee.setOnCheckedChangeListener { _,
isChecked ->
            viewModel.toggleCoffeeFilter(isChecked)
        }
        binding.chipMood.setOnCheckedChangeListener { _, isChecked
->
            viewModel.toggleMoodFilter(isChecked)
        }
    }

    private fun observeViewModel() {
        viewModel.historyItems

```

```

        .flowWithLifecycle(viewLifecycleOwner.lifecycle,
Lifecycle.State.STARTED)
        .onEach { items ->
            historyAdapter.submitList(items)
            binding.emptyView.visibility = if
(items.isEmpty()) View.VISIBLE else View.GONE
        }
        .launchIn(viewLifecycleOwner.lifecycleScope)

viewModel.dateRange
    .flowWithLifecycle(viewLifecycleOwner.lifecycle,
Lifecycle.State.STARTED)
    .onEach { (start, end) ->
        binding.dateRangeText.text =
formatDateRange(start, end)
    }
    .launchIn(viewLifecycleOwner.lifecycleScope)

viewModel.filters
    .flowWithLifecycle(viewLifecycleOwner.lifecycle,
Lifecycle.State.STARTED)
    .onEach { filters ->
        if (binding.chipCoffee.isChecked !=
filters.showCoffee)
            binding.chipCoffee.isChecked =
filters.showCoffee
        if (binding.chipMood.isChecked !=
filters.showMood)
            binding.chipMood.isChecked = filters.showMood
    }
    .launchIn(viewLifecycleOwner.lifecycleScope)

viewModel.isLoading
    .flowWithLifecycle(viewLifecycleOwner.lifecycle,
Lifecycle.State.STARTED)
    .onEach { isLoading ->
        binding.swipeRefresh.isRefreshing = isLoading
    }
    .launchIn(viewLifecycleOwner.lifecycleScope)
}

private fun formatDateRange(start: LocalDate, end: LocalDate):
String {
    val formatter = DateTimeFormatter.ofPattern("dd.MM.yyyy")
    return when {
        start == end && start == LocalDate.now() -> "Сегодня"
        start == end && start == LocalDate.now().minusDays(1)
-> "Вчера"
        start == end -> start.format(formatter)
        else -> "${start.format(formatter)} -
${end.format(formatter)}"
    }
}

```

```

    }

    private fun observeDeleteResult() {
        parentFragmentManager.setFragmentResultListener(
            "COFFEE_DELETED",
            viewLifecycleOwner
        ) { _, bundle ->
            val deletedId = bundle.getInt("deleted_id")
            viewModel.forceReload()
            Toast.makeText(requireContext(), "Запись удалена",
                Toast.LENGTH_SHORT).show()
        }
        parentFragmentManager.setFragmentResultListener(
            "MOOD_DELETED",
            viewLifecycleOwner
        ) { _, bundle ->
            val deletedId = bundle.getInt("deleted_id")
            viewModel.forceReload()
            Toast.makeText(requireContext(), "Настройка удалено",
                Toast.LENGTH_SHORT).show()
        }
    }

    override fun onResume() {
        super.onResume()
        viewModel.forceReload()
    }
    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}

```

Листинг А.31 – Файл HistoryItem.kt

```

package com.example.coffies.ui.history

import java.time.LocalDate
import java.time.LocalDateTime

sealed class HistoryItem {
    data class DateHeader(val date: LocalDate) : HistoryItem()

    data class CoffeeItem(
        val id: Int,
        val type: String,
        val volume: Int?,
        val quantity: Int,
        val price: Double?,
        val dateTime: LocalDateTime,
        val place: String?,
        val comment: String?
    ) : HistoryItem()
}

```

```

    data class MoodItem(
        val id: Int,
        val level: Int,
        val dateTime: LocalDateTime,
        val comment: String?
    ) : HistoryItem()
}

data class HistoryFilters(
    val showCoffee: Boolean = true,
    val showMood: Boolean = true
)

```

Листинг А.32 – Файл HistoryViewModel.kt

```

package com.example.coffies.ui.history

class HistoryViewModel(
    private val db: AppDatabase,
) : ViewModel() {

    private val _dateRange =
MutableStateFlow(LocalDate.now().minusDays(7) to LocalDate.now())
    val dateRange: StateFlow<Pair<LocalDate, LocalDate>> =
    _dateRange.asStateFlow()

    private val _filters = MutableStateFlow(HistoryFilters())
    val filters: StateFlow<HistoryFilters> =
    _filters.asStateFlow()

    private val reloadTrigger =
MutableSharedFlow<Unit>(extraBufferCapacity = 1)

    private val _isLoading = MutableStateFlow(false)
    val isLoading: StateFlow<Boolean> = _isLoading.asStateFlow()

    private val _historyItems =
MutableStateFlow<List<HistoryItem>>(emptyList())
    val historyItems: StateFlow<List<HistoryItem>> =
    _historyItems.asStateFlow()

    init {
        viewModelScope.launch {
            combine(_dateRange, _filters) { range, filters ->
range to filters }
                .flatMapLatest { (range, filters) ->
                    reloadTrigger
                        .onStart { emit(Unit) }
                        .map {
                            _isLoading.value = true
                            val items =
loadGroupedHistoryItems(range, filters)

```

```

        _isLoading.value = false
        items
    }
}
}
.catch { throwable ->
    _isLoading.value = false
    _historyItems.value = emptyList()
}
.collect { items ->
    _historyItems.value = items
}
}
}

fun updateDateRange(from: LocalDate, to: LocalDate) {
    _dateRange.value = from to to
    forceReload()
}

fun toggleCoffeeFilter(show: Boolean) {
    _filters.value = _filters.value.copy(showCoffee = show)
    forceReload()
}

fun toggleMoodFilter(show: Boolean) {
    _filters.value = _filters.value.copy(showMood = show)
    forceReload()
}

fun forceReload() {
    reloadTrigger.tryEmit(Unit)
}

private suspend fun loadGroupedHistoryItems(
    dateRange: Pair<LocalDate, LocalDate>,
    filters: HistoryFilters,
): List<HistoryItem> {
    val (start, end) = dateRange
    val startStr =
start.format(DateTimeFormatter.ISO_LOCAL_DATE)
    val endStr = end.format(DateTimeFormatter.ISO_LOCAL_DATE)

    val coffeeTypeMap = runCatching {
        db.coffeeTypeDao().getAllCoffeeTypeNames().associate {
it.id to it.name }
    }.getOrElse(emptyMap())

    val coffee = if (filters.showCoffee) {
        runCatching {
            db.coffeeEntryDao().getEntriesForPeriod(startStr,
endStr)
                .map { it.toHistoryItem(coffeeTypeMap) }

```

```

        }.getOrElse(emptyList())
    } else emptyList()

    val mood = if (filters.showMood) {
        runCatching {
            db.moodEntryDao().getEntriesForPeriod(startStr,
endStr)
                .map { it.toHistoryItem() }
        }.getOrElse(emptyList())
    } else emptyList()

    val allItems = (coffee + mood).sortedByDescending {
        when (it) {
            is HistoryItem.CoffeeItem -> it.dateTime
            is HistoryItem.MoodItem -> it.dateTime
            is HistoryItem.DateHeader -> LocalDateTime.MIN
        }
    }
    return allItems
        .groupBy {
            when (it) {
                is HistoryItem.CoffeeItem ->
it.dateTime.toLocalDate()
                is HistoryItem.MoodItem ->
it.dateTime.toLocalDate()
                is HistoryItem.DateHeader -> it.date
            }
        }
        .toSortedMap(compareByDescending { it })
        .flatMap { (date, items) ->
listOf(HistoryItem.DateHeader(date)) + items }
    }

    private fun CoffeeEntry.toHistoryItem(coffeeTypeMap: Map<Int,
String>): HistoryItem.CoffeeItem =
        HistoryItem.CoffeeItem(
            id = id,
            type = coffeeTypeMap[coffee_type_id] ?: "Неизвестно",
            volume = volume_ml,
            quantity = quantity,
            price = price,
            dateTime = LocalDateTime.parse("${date}T${time}"),
            place = place,
            comment = comment,
        )

    private fun MoodEntry.toHistoryItem(): HistoryItem.MoodItem =
        HistoryItem.MoodItem(
            id = id,
            level = mood_level,
            dateTime = LocalDateTime.parse("${date}T${time}"),
            comment = comment,

```

```

        )
    }
}

```

Листинг A.33 – Файл MaxDateValidator.kt

```

package com.example.coffies.ui.history

class MaxDateValidator(private val maxDate: Long) : DateValidator
{
    override fun isValid(date: Long): Boolean = date <= maxDate
    override fun writeToParcel(dest: Parcel, flags: Int) {}
    override fun describeContents(): Int = 0

    companion object CREATOR :
        android.os.Parcelable.Creator<MaxDateValidator> {
            override fun createFromParcel(parcel: android.os.Parcel):
                MaxDateValidator {
                val maxDate = parcel.readLong()
                return MaxDateValidator(maxDate)
            }
            override fun newArray(size: Int): Array<MaxDateValidator?>
                = arrayOfNulls(size)
        }
}

```

Листинг A.34 – Файл MainFragment.kt

```

package com.example.coffies.ui.main

class MainFragment : Fragment() {

    private var _binding: FragmentMainBinding? = null
    private val binding get() = _binding!!

    private val viewModel: MainViewModel by viewModels {
        MainViewModelFactory(AppDatabase.getInstance(requireContext()))
    }
    private val recAdapter = RecommendationAdapter()

    private val colorRed by lazy {
        ContextCompat.getColor(requireContext(), R.color.red) }
    private val colorBlack by lazy {
        ContextCompat.getColor(requireContext(), R.color.black) }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentMainBinding.inflate(inflater,
            container, false)
        return binding.root
    }
}

```

```

        override fun onCreateView(view: View, savedInstanceState:
Bundle?) {
            binding.recommendationsRecycler.layoutManager =
LinearLayoutManager(requireContext())
            binding.recommendationsRecycler.adapter = recAdapter

            binding.addCoffeeButton.setOnClickListener {

findNavController().navigate(R.id.action_main_to_addCoffee)
            }
            binding.addMoodButton.setOnClickListener {

findNavController().navigate(R.id.action_main_to_addMood)
            }

            viewLifecycleOwner.lifecycleScope.launch {

viewLifecycleOwner.repeatOnLifecycle(androidx.lifecycle.Lifecycle.
State.STARTED) {
                viewModel.stats.collect { stats ->
                    binding.cupsText.text = when {
                        stats?.cupsCount != null && stats.cupGoal
!= null ->

getString(R.string.cups_count_with_goal, stats.cupsCount,
stats.cupGoal)

                        stats?.cupsCount != null ->
                            getString(R.string.cups_count_simple,
stats.cupsCount)

                        else -> getString(R.string.no_data)
                    }
                    val isCupExceeded = stats?.cupGoal != null &&
stats.cupsCount != null && stats.cupGoal > 0 && stats.cupsCount >
stats.cupGoal
                    binding.cupsText.setTextColor(if
(isCupExceeded) colorRed else colorBlack)

                    binding.caffeineText.text =
stats?.caffeine?.let { "$it mГ" } ?: getString(R.string.no_data)
                    val isCaffeineExceeded = stats?.caffeine !=
null && stats.caffeine > 400
                    binding.caffeineText.setTextColor(if
(isCaffeineExceeded) colorRed else colorBlack)
                    binding.caffeineLimitText.text = "/ 400 mГ"
                    binding.caffeineLimitText.setTextColor(if
(isCaffeineExceeded) colorRed else colorBlack)

                    if (stats?.avgMood != null && stats.avgMood >
0) {

binding.currentMoodIcon.setImageResource(moodLevelToDrawable(stats

```

```

.avgMood))
                binding.moodText.text =
moodLevelToText(stats.avgMood)
                binding.moodText.setTextColor(colorBlack)
            } else {

binding.currentMoodIcon.setImageResource(R.drawable.mood_3)
                binding.moodText.text =
getString(R.string.no_data)
                binding.moodText.setTextColor(colorBlack)
            }

                binding.moodLimitText.text = "/" + "Отличное"
binding.moodLimitText.setTextColor(colorBlack)

                binding.spentText.text =
stats?.totalSpent?.let { "%.2f ₸".format(it) } ?:
getString(R.string.no_data)
                val isSpendExceeded = stats?.spendGoal != null
&& stats.totalSpent != null && stats.totalSpent > stats.spendGoal
                binding.spentText.setTextColor(if
(isSpendExceeded) colorRed else colorBlack)
                binding.spentLimitText.text =
                    if (stats?.spendGoal != null) "/" + "%.2f
₸".format(stats.spendGoal) else "/" + " +
getString(R.string.infinity)
                binding.spentLimitText.setTextColor(if
(isSpendExceeded) colorRed else colorBlack)

                binding.recommendationTitle.text = if
(!stats?.userName.isNullOrEmpty()) {
                    "${stats!!.userName}, вот ваши
рекомендации дня"
                } else {
                    getString(R.string.recomendation)
                }
                recAdapter.updateFromStats(stats)
            }
        }

        viewModel.loadStatsForToday()
    }

    override fun onResume() {
        super.onResume()
        viewModel.loadStatsForToday()
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }

```

```

    }

    private fun moodLevelToDrawable(level: Int): Int {
        return when (level) {
            1 -> R.drawable.mood_1
            2 -> R.drawable.mood_2
            3 -> R.drawable.mood_3
            4 -> R.drawable.mood_4
            5 -> R.drawable.mood_5
            else -> R.drawable.mood_3
        }
    }

    private fun moodLevelToText(level: Int): String {
        return when (level) {
            1 -> getString(R.string.mood_1_short)
            2 -> getString(R.string.mood_2_short)
            3 -> getString(R.string.mood_3_short)
            4 -> getString(R.string.mood_4_short)
            5 -> getString(R.string.mood_5_short)
            else -> getString(R.string.no_data)
        }
    }
}

```

Листинг A.35 – Файл MainScreenStats.kt

```

package com.example.coffies.ui.main

data class MainScreenStats(
    val cupsCount: Int? = null,
    val caffeine: Int? = null,
    val totalSpent: Double? = null,
    val avgMood: Int? = null,
    val recommendations: List<String> = emptyList(),
    val cupGoal: Int? = null,
    val spendGoal: Float? = null,
    val userName: String? = null
)

```

Листинг A.36 – Файл MainViewModel.kt

```

package com.example.coffies.ui.main

class MainViewModel(
    private val db: AppDatabase,
    private val ioDispatcher: CoroutineDispatcher = Dispatchers.IO
) : ViewModel() {

    private val _stats =
MutableStateFlow<MainScreenStats?>(MainScreenStats())
    val stats: StateFlow<MainScreenStats?> = _stats

```

```

        fun loadStatsForToday() {
            viewModelScope.launch(ioDispatcher) {
                val today =
                    LocalDate.now().format(DateTimeFormatter.ISO_LOCAL_DATE)

                val cups =
                    db.coffeeEntryDao().getCupsCountForDate(today)

                val caffeine =
                    db.coffeeEntryDao().getTotalCaffeineForDate(today)

                val spent =
                    db.coffeeEntryDao().getTotalSpentForDate(today)

                val moodRaw =
                    db.moodEntryDao().getAverageMoodLevelForDate(today) ?: 0.0

                val avgMood = ceil(moodRaw).toInt().takeIf { it > 0 }

            }

            val settings =
                db.userSettingsDao().getSettingsOnce()

            _stats.emit(
                MainScreenStats(
                    cupsCount = cups.takeIf { it > 0 },
                    caffeine = caffeine.takeIf { it > 0.0 }
                ).toInt(),
                totalSpent = spent.takeIf { it > 0.0 },
                avgMood = avgMood,
                cupGoal = settings?.daily_cup_goal,
                spendGoal = settings?.daily_spend_goal,
                userName = settings?.name
            )
        }
    }
}

```

Листинг A.37 – Файл MainViewModelFactory.kt

```

package com.example.coffies.ui.main

class MainViewModelFactory(private val db: AppDatabase) :
    ViewModelProvider.Factory {
    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        if
            (modelClass.isAssignableFrom(MainViewModel::class.java)) {
            @Suppress("UNCHECKED_CAST")
            return MainViewModel(db) as T
        }

        throw IllegalArgumentException("Unknown ViewModel class")
    }
}

```

```
    }
}
```

Листинг A.38 – Файл RecommendationAdapter.kt

```
package com.example.coffies.ui.main

class RecommendationAdapter :
    RecyclerView.Adapter<RecommendationAdapter.RecViewHolder>() {
    private val data = mutableListOf<String>()

    fun updateFromStats(stats: MainScreenStats?) {
        data.clear()
        stats?.let { data += generateRecommendations(it) }
        try {
            notifyDataSetChanged()
        } catch (_: NullPointerException) { }
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType:
    Int): RecViewHolder {
        val v = LayoutInflater.from(parent.context)
            .inflate(R.layout.item_recommendation, parent, false)
        return RecViewHolder(v)
    }
    override fun onBindViewHolder(holder: RecViewHolder, position:
    Int) =
        holder.bind(data[position])
    override fun getItemCount() = data.size

    class RecViewHolder(v: View) : RecyclerView.ViewHolder(v) {
        private val tv: TextView =
        v.findViewById(R.id.recommendation_text)
        fun bind(text: String) { tv.text = text }
    }

    companion object {
        fun generateRecommendations(s: MainScreenStats):
        List<String> = buildList {
            if (s.caffeine != null) when {
                s.caffeine > 400 -> add("☕ Превышен дневной
                лимит кофеина. Рекомендуется снизить потребление.")
                s.caffeine >= 300 -> add("⚠ Вы близки к
                лимиту кофеина. Будьте внимательны!")
                s.caffeine < 100 -> add("🌱 Сегодня мало
                кофеина — вы молодец, организм скажет спасибо!")
            }
            if (s.spendGoal != null && s.totalSpent != null) when
            {
                s.totalSpent > s.spendGoal -> add("💰
                Расходы на кофе превысили цель. Попробуйте проанализировать
                траты.")
            }
        }
    }
}
```

```

        s.totalSpent < s.spendGoal * 0.5f    -> add("👍 Вы
хорошо контролируете траты на кофе!")
        (s.spendGoal - s.totalSpent) < 100    -> add("$
До финансового лимита осталось меньше 100 ₽.")
    }
    if (s.avgMood != null) when (s.avgMood) {
        in 1..2 -> add("😊 Постарайтесь больше отдыхать.
Хорошее настроение – залог продуктивности!")
        3        -> add("😊 Ваше настроение сегодня
стабильное.")
        4        -> add("😄 Ваше настроение сегодня
хорошее.")
        5        -> add("🎉 Отличное настроение сегодня!
Так держать!")
    }
    if (isEmpty()) {
        add("💡 Не забывайте пить воду и делать небольшие
перерывы!")
        add("❤️ Всё в норме! Продолжайте в том же духе.")
    }
    }.distinct().take(3)
}
}

```

Листинг A.39 – Файл MoodDetailsFragment.kt

```

package com.example.coffies.ui.mooddetails

class MoodDetailsFragment : Fragment() {

    private var _binding: FragmentMoodDetailsBinding? = null
    private val binding get() = _binding!!
    private val args: MoodDetailsFragmentArgs by navArgs()

    private val viewModel: MoodDetailsViewModel by viewModels {
        MoodDetailsViewModelFactory(AppDatabase.getInstance(requireContext
()), args.moodEntryId)
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentMoodDetailsBinding.inflate(inflater,
container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {

```

```

        requireActivity().addMenuProvider(object : MenuProvider {
            override fun onCreateMenu(menu: Menu, menuInflater:
MenuInflater) {
                menuInflater.inflate(R.menu.mood_details_menu,
menu)
            }
            override fun onMenuItemSelected(item: MenuItem):
Boolean {
                return when (item.itemId) {
                    R.id.action_delete -> {
                        showDeleteConfirmationDialog()
                        true
                    }
                    else -> false
                }
            }
        }, viewLifecycleOwner, Lifecycle.State.RESUMED)

        viewModel.state
            .flowWithLifecycle(viewLifecycleOwner.lifecycle,
Lifecycle.State.STARTED)
            .onEach { state ->
                if (state.loading) return@onEach
                if (state.error != null) {
                    Toast.makeText(requireContext(), state.error,
Toast.LENGTH_LONG).show()
                    findNavController().popBackStack()
                    return@onEach
                }
                val mood = state.mood
                val coffee = state.coffee
                if (mood == null || coffee == null) {
                    Toast.makeText(requireContext(), "Запись или
связанный кофе не найдены", Toast.LENGTH_SHORT).show()
                    findNavController().popBackStack()
                    return@onEach
                }
            }

        binding.moodIcon.setImageResource(viewModel.moodLevelToDrawable(mo
od.mood_level))
        binding.moodLevelValue.text =
viewModel.moodLevelToText(mood.mood_level)
        binding.dateValue.text =
viewModel.formatDate(mood.date)
        binding.timeValue.text = mood.time
        binding.commentValue.text = mood.comment ?:
getString(R.string.no_data)
        binding.linkedCoffeeInfo.text =
viewModel.getLinkedCoffeeInfo()
        binding.linkedCoffeeContainer.setOnClickListener {
            val action =
MoodDetailsFragmentDirections.actionMoodDetailsToCoffeeDetails(cof

```

```

fee.id)
        findNavController().navigate(action)
    }
}
    .launchIn(viewLifecycleOwner.lifecycleScope)
}
private fun showDeleteConfirmationDialog() {
    AlertDialog.Builder(requireContext(),
R.style.CoffiesDialogTheme)
        .setTitle("Удалить запись?")
        .setMessage("Вы действительно хотите удалить это
настроение? Это действие нельзя отменить.")
        .setPositiveButton("Удалить") { dialog, _ ->
            viewModel.deleteMood {
                parentFragmentManager.setFragmentResult(
                    "MOOD_DELETED",
                    bundleOf("deleted_id" to args.moodEntryId)
                )
                findNavController().popBackStack()
            }
            dialog.dismiss()
        }
        .setNegativeButton("Отмена", null)
        .show()
}
override fun onDestroyView() {
    super.onDestroyView()
    _binding = null
}
}

```

Листинг А.40 – Файл MoodDetailsState.kt

```

package com.example.coffies.ui.mooddetails

import com.example.coffies.database.coffeentry.CoffeeEntry
import com.example.coffies.database.moodentry.MoodEntry

data class MoodDetailsState(
    val mood: MoodEntry? = null,
    val coffee: CoffeeEntry? = null,
    val coffeeTypeName: String = "",
    val relationType: String = "",
    val loading: Boolean = true,
    val error: String? = null
)

```

Листинг А.41 – Файл MoodDetailsViewModel.kt

```

package com.example.coffies.ui.mooddetails

class MoodDetailsViewModel(
    private val db: AppDatabase,

```

```

        private val moodEntryId: Int
    ) : ViewModel() {

        private val _state = MutableStateFlow(MoodDetailsState())
        val state: StateFlow<MoodDetailsState> = _state.asStateFlow()

        init {
            loadMoodDetails()
        }

        private fun loadMoodDetails() {
            viewModelScope.launch {
                try {
                    val mood = db.moodEntryDao().getById(moodEntryId)
                    if (mood == null) {
                        _state.value = MoodDetailsState(error =
"Запись не найдена", loading = false)
                        return@launch
                    }
                    val link =
db.coffeeMoodLinkDao().getLinksForMoodEntry(mood.id).firstOrNull()
                    val coffee = link?.let {
db.coffeeEntryDao().getById(it.coffee_entry_id) }
                    val typeName = coffee?.let {

db.coffeeTypeDao().getAllCoffeeTypeNames().find { t -> t.id ==
coffee.coffee_type_id }?.name ?: ""
                    } ?: ""
                    _state.value = MoodDetailsState(
                        mood = mood,
                        coffee = coffee,
                        coffeeTypeName = typeName,
                        relationType = link?.relation_type ?: "",
                        loading = false
                    )
                } catch (e: Exception) {
                    _state.value = MoodDetailsState(error = e.message
?: "Ошибка", loading = false)
                }
            }
        }

        fun deleteMood(onDeleted: () -> Unit) {
            viewModelScope.launch {
                val st = _state.value
                val mood = st.mood ?: return@launch
                db.moodEntryDao().delete(mood)
                onDeleted()
            }
        }
    }

```

```

fun getLinkedCoffeeInfo(): String {
    val st = _state.value
    val coffee = st.coffee ?: return "-"
    val typeName = st.coffeeTypeName.ifEmpty { "-" }
    val volume = coffee.volume_ml
    val time = coffee.time
    val date = try {
        val parsed = SimpleDateFormat("yyyy-MM-dd",
Locale.getDefault()).parse(coffee.date)
        SimpleDateFormat("dd.MM.yyyy",
Locale.getDefault()).format(parsed!!)
    } catch (e: Exception) {
        coffee.date
    }

    val moment = when (st.relationType) {
        "before and during" -> "до/во время"
        "after" -> "после"
        else -> ""
    }
    return "$typeName • $volume мл • $time $date • $moment"
}

fun moodLevelToText(level: Int): String = when (level) {
    1 -> "Очень плохо"
    2 -> "Плохо"
    3 -> "Нормально"
    4 -> "Хорошо"
    5 -> "Отлично"
    else -> "-"
}

fun moodLevelToDrawable(level: Int): Int = when (level) {
    1 -> com.example.coffies.R.drawable.mood_1
    2 -> com.example.coffies.R.drawable.mood_2
    3 -> com.example.coffies.R.drawable.mood_3
    4 -> com.example.coffies.R.drawable.mood_4
    5 -> com.example.coffies.R.drawable.mood_5
    else -> com.example.coffies.R.drawable.mood_3
}

fun formatDate(iso: String): String {
    return try {
        val parsed = SimpleDateFormat("yyyy-MM-dd",
Locale.getDefault()).parse(iso)
        SimpleDateFormat("dd.MM.yyyy",
Locale.getDefault()).format(parsed!!)
    } catch (e: Exception) {
        iso
    }
}
}

```

Листинг A.42 – Файл SettingsFragment.kt

```

package com.example.coffies.ui.settings

class SettingsFragment : Fragment() {

    private var _binding: FragmentSettingsBinding? = null
    private val binding get() = _binding!!
    private val viewModel: SettingsViewModel by viewModels {
        SettingsViewModelFactory(AppDatabase.getInstance(requireContext())
        )
    }

    private var isDialogShown = false
    private var isResetDialogShown = false
    private var isUpdatingEditText = false

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentSettingsBinding.inflate(inflater,
        container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState:
    Bundle?) {
        requireActivity().addMenuProvider(object : MenuProvider {
            override fun onCreateMenu(menu: Menu, menuInflater:
            MenuInflater) {
                menuInflater.inflate(R.menu.menu_settings_actions,
                menu)
            }
            override fun onMenuItemSelected(menuItem: MenuItem):
            Boolean {
                return when (menuItem.itemId) {
                    R.id.action_edit -> {
                        if (viewModel.state.value.isEditing) {
                            if (viewModel.hasChanges()) {
                                if (!isDialogShown) {
                                    isDialogShown = true
                                    showUnsavedChangesDialog()
                                }
                            }
                        } else {
                            viewModel.exitEditMode()
                            hideKeyboard()
                        }
                    } else {
                        viewModel.enterEditMode()
                    }
                }
            }
        }, true
    }

```

```

        }
        R.id.action_reset -> {
            if (!isResetDialogShown) showResetDialog()
            true
        }
        else -> false
    }
}
}, viewLifecycleOwner, Lifecycle.State.RESUMED)

binding.nameEdit.addTextChangedListener {
    if (viewModel.state.value.isEditing &&
!isUpdatingEditText) {
        viewModel.onNameChanged(it?.toString().orEmpty())
    }
}
binding.cupGoalEdit.addTextChangedListener {
    if (viewModel.state.value.isEditing &&
!isUpdatingEditText) {
viewModel.onCupGoalChanged(it?.toString().orEmpty())
    }
}
binding.monthlyCupGoalEdit.addTextChangedListener {
    if (viewModel.state.value.isEditing &&
!isUpdatingEditText) {
viewModel.onMonthlyCupGoalChanged(it?.toString().orEmpty())
    }
}
binding.spendGoalEdit.addTextChangedListener {
    if (viewModel.state.value.isEditing &&
!isUpdatingEditText) {
viewModel.onSpendGoalChanged(it?.toString().orEmpty())
    }
}
binding.monthlySpendGoalEdit.addTextChangedListener {
    if (viewModel.state.value.isEditing &&
!isUpdatingEditText) {
viewModel.onMonthlySpendGoalChanged(it?.toString().orEmpty())
    }
}

binding.saveButton.setOnClickListener {
    viewModel.save()
    hideKeyboard()
    Toast.makeText(requireContext(), "Настройки
сохранены", Toast.LENGTH_SHORT).show()
}

```

```

        viewModel.state
            .flowWithLifecycle(viewLifecycleOwner.lifecycle,
Lifecycle.State.STARTED)
            .onEach { state -> render(state) }
            .launchIn(viewLifecycleOwner.lifecycleScope)

requireActivity().onBackPressedDispatcher.addCallback(
    viewLifecycleOwner,
    object : OnBackPressedCallback(true) {
        override fun handleOnBackPressed() {
            if (viewModel.state.value.isEditing) {
                if (viewModel.hasChanges()) {
                    if (!isDialogShown) {
                        isDialogShown = true
                        showUnsavedChangesDialog()
                    }
                } else {
                    viewModel.exitEditMode()
                    hideKeyboard()
                }
            } else {
                isEnabled = false
            }
        }
    })

requireActivity().onBackPressedDispatcher.onBackPressed()
    }
}

private fun showUnsavedChangesDialog() {
    android.app.AlertDialog.Builder(requireContext(),
R.style.CoffiesDialogTheme)
        .setTitle("Сохранить изменения?")
        .setMessage("У вас есть несохранённые изменения.
Сохранить перед выходом?")
        .setPositiveButton("Сохранить") { dialog, _ ->
            viewModel.save()
            dialog.dismiss()
            Toast.makeText(requireContext(), "Настройки
сохранены", Toast.LENGTH_SHORT).show()
        }
        .setNegativeButton("Выйти без сохранения") { dialog, _
->
            viewModel.discardChangesAndExitEditMode()
            hideKeyboard()
            dialog.dismiss()
        }
        .setOnDismissListener { isDialogShown = false }
        .show()
}

```

```

private fun showResetDialog() {
    isResetDialogShown = true
    android.app.AlertDialog.Builder(requireContext(),
R.style.CoffiesDeleteDialogTheme)
        .setTitle("Сбросить настройки?")
        .setMessage("Вы уверены, что хотите сбросить настройки
пользователя? Это действие удалит все ваши настройки и не может
быть отменено.")
        .setPositiveButton("Сбросить") { dialog, _ ->
            viewModel.resetSettings()
            dialog.dismiss()
            Toast.makeText(requireContext(), "Настройки
сброшены", Toast.LENGTH_SHORT).show()
        }
        .setNegativeButton("Отмена") { dialog, _ ->
            dialog.dismiss()
        }
        .setOnDismissListener { isResetDialogShown = false }
        .show()
}

private fun render(state: SettingsViewState) {
    if (state.isEditing) {
        isUpdatingEditText = true
        if (binding.nameEdit.text.toString() != state.name)
            binding.nameEdit.setText(state.name)
        if (binding.cupGoalEdit.text.toString() !=
(state.cupGoal?.toString() ?: ""))
binding.cupGoalEdit.setText(state.cupGoal?.toString() ?: "")
        if (binding.monthlyCupGoalEdit.text.toString() !=
(state.monthlyCupGoal?.toString() ?: ""))
binding.monthlyCupGoalEdit.setText(state.monthlyCupGoal?.toString(
) ?: "")
        if (binding.spendGoalEdit.text.toString() !=
(state.spendGoal?.let { formatMoney(it) } ?: ""))
            binding.spendGoalEdit.setText(state.spendGoal?.let
{ formatMoney(it) } ?: "")
        if (binding.monthlySpendGoalEdit.text.toString() !=
(state.monthlySpendGoal?.let { formatMoney(it) } ?: ""))
binding.monthlySpendGoalEdit.setText(state.monthlySpendGoal?.let {
formatMoney(it) } ?: "")
        isUpdatingEditText = false
    }
    binding.nameText.text = if (state.name.isNotBlank())
state.name else "Не задано"
    binding.cupGoalText.text = state.cupGoal?.toString() ?:
"Не задано"
    binding.monthlyCupGoalText.text =
state.monthlyCupGoal?.toString() ?: "Не задано"

```

```

        binding.spendGoalText.text = state.spendGoal?.let {
            formatMoney(it) + " ₺" } ?: "Не задано"
        binding.monthlySpendGoalText.text =
            state.monthlySpendGoal?.let { formatMoney(it) + " ₺" } ?: "Не
            задано"

        setEditMode(state.isEditing)
    }

    private fun setEditMode(isEditing: Boolean) {
        binding.nameText.visibility = if (isEditing) View.GONE
        else View.VISIBLE
        binding.nameEdit.visibility = if (isEditing) View.VISIBLE
        else View.GONE

        binding.cupGoalText.visibility = if (isEditing) View.GONE
        else View.VISIBLE
        binding.cupGoalEdit.visibility = if (isEditing)
        View.VISIBLE else View.GONE

        binding.monthlyCupGoalText.visibility = if (isEditing)
        View.GONE else View.VISIBLE
        binding.monthlyCupGoalEdit.visibility = if (isEditing)
        View.VISIBLE else View.GONE

        binding.spendGoalText.visibility = if (isEditing)
        View.GONE else View.VISIBLE
        binding.spendGoalEdit.visibility = if (isEditing)
        View.VISIBLE else View.GONE

        binding.monthlySpendGoalText.visibility = if (isEditing)
        View.GONE else View.VISIBLE
        binding.monthlySpendGoalEdit.visibility = if (isEditing)
        View.VISIBLE else View.GONE

        binding.saveButton.visibility = if (isEditing)
        View.VISIBLE else View.GONE
    }

    private fun hideKeyboard() {
        val view = activity?.currentFocus ?: view
        view?.let {
            val imm =
                requireContext().getSystemService(android.content.Context.INPUT_ME
                THOD_SERVICE) as android.view.inputmethod.InputMethodManager
            imm.hideSoftInputFromWindow(it.windowToken, 0)
        }
    }

    private fun formatMoney(value: Float): String {
        return if (value == value.toInt().toFloat())
            value.toInt().toString() else value.toString()
    }

```

```

    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}

```

Листинг А.43 – Файл SettingsViewModel.kt

```

package com.example.coffies.ui.settings

class SettingsViewModel(
    private val db: AppDatabase,
    private val io: CoroutineDispatcher = Dispatchers.IO
) : ViewModel() {

    private val _state = MutableStateFlow(SettingsViewState())
    val state: StateFlow<SettingsViewState> = _state.asStateFlow()

    private var lastDbSettings: UserSettings? = null

    init {
        viewModelScope.launch(io) {
            db.userSettingsDao().getSettings().collect { settings
->
                if (!_state.value.isEditing) {
                    lastDbSettings = settings
                    _state.value = settings.toViewState()
                }
            }
        }
    }

    fun enterEditMode() = update { copy(isEditing = true) }
    fun exitEditMode() = update { copy(isEditing = false) }

    fun discardChangesAndExitEditMode() {
        _state.value = lastDbSettings.toViewState()
    }

    fun onNameChanged(value: String) = update {
copy(name = value) }
    fun onCupGoalChanged(value: String) = update {
copy(cupGoal = value.toIntOrNull()) }
    fun onMonthlyCupGoalChanged(value: String) = update {
copy(monthlyCupGoal = value.toIntOrNull()) }
    fun onSpendGoalChanged(value: String) = update {
copy(spendGoal = value.toFloatOrNull()) }
    fun onMonthlySpendGoalChanged(value: String) = update {
copy(monthlySpendGoal = value.toFloatOrNull()) }

    fun hasChanges(): Boolean {

```

```

        val s    = lastDbSettings
        val cur = _state.value
        if (s == null && cur == SettingsViewState()) return false
        return cur.name                != (s?.name.orEmpty())

    ||
        cur.cupGoal                    != s?.daily_cup_goal
    ||
        cur.monthlyCupGoal             != s?.monthly_cup_goal
    ||
        cur.spendGoal                  != s?.daily_spend_goal
    ||
        cur.monthlySpendGoal           != s?.monthly_spend_goal
    }

    fun save() {
        val st = _state.value
        viewModelScope.launch(io) {
            db.userSettingsDao().insertOrUpdate(
                UserSettings(
                    name                = st.name,
                    daily_cup_goal      = st.cupGoal,
                    monthly_cup_goal    = st.monthlyCupGoal,
                    daily_spend_goal    = st.spendGoal,
                    monthly_spend_goal  = st.monthlySpendGoal
                )
            )
            withContext(Dispatchers.Main) { exitEditMode() }
        }
    }

    fun resetSettings() {
        viewModelScope.launch(io) {
            db.userSettingsDao().clearSettings()
            withContext(Dispatchers.Main) { _state.value =
SettingsViewState() }
        }
    }

    private inline fun update(block: SettingsViewState.() ->
SettingsViewState) {
        _state.value = _state.value.block()
    }

    private fun UserSettings?.toViewState() = SettingsViewState(
        isEditing        = false,
        name              = this?.name.orEmpty(),
        cupGoal           = this?.daily_cup_goal,
        monthlyCupGoal    = this?.monthly_cup_goal,
        spendGoal         = this?.daily_spend_goal,
        monthlySpendGoal  = this?.monthly_spend_goal
    )
}

```

Листинг А.44 – Файл SettingsViewState.kt

```
package com.example.coffies.ui.settings

data class SettingsViewState(
    val isEditing: Boolean = false,
    val name: String = "",
    val cupGoal: Int? = null,
    val monthlyCupGoal: Int? = null,
    val spendGoal: Float? = null,
    val monthlySpendGoal: Float? = null
)
```

Приложение Б
(рекомендуемое)
Рекомендательное письмо



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Пензенский государственный университет»
(ФГБОУ ВО «ПГУ»)

ул. Красная, д. 40, г. Пенза, Россия, 440026
 Тел: (841-2) 66-62-85, факс: (841-2) 66-63-32, e-mail: cnit@pnzgu.ru, <http://www.pnzgu.ru>
 ОКПО 02069042, ОГРН 1025801440620, ИНН/КПП 5837003736/583701001

14.05.2025 № 29-319

Генеральному директору
 Фонда содействия инновациям
 С.Г. Полякову

Уважаемый Сергей Геннадьевич!

ФГБОУ ВО «Пензенский государственный университет» настоящим письмом подтверждает свою заинтересованность в сотрудничестве с Левиным Андреем Алексеевичем с целью реализации проекта «Мобильное приложения для осознанного потребления кофе».

Данный проект важен для ФГБОУ ВО «ПГУ», поскольку он соответствует приоритетным направлениям государственной стратегии цифровой трансформации и укрепления здоровья населения: развитию цифровых платформ в социальной сфере и здравоохранении, а также популяризации здорового образа жизни.

Проект нацелен на создание удобного мобильного помощника, позволяющего пользователям фиксировать выпитые чашки кофе и потребленный кофеин, отслеживать собственное настроение в течение дня и контролировать расходы. Визуализация данных в виде наглядных графиков поможет понять, как привычки в потреблении кофе влияют на самочувствие и личный бюджет.

ФГБОУ ВО «ПГУ» готов оказать консультативную и информационную поддержку проекту, а также экспертную поддержку в лице своих сотрудников.

Проректор по научной работе
 и инновационной деятельности

С.М. Васин

Участник конкурса
 «Студенческий стартап»

А.А. Левин

Приложение В
(рекомендуемое)
Схемы взаимодействия компонентов

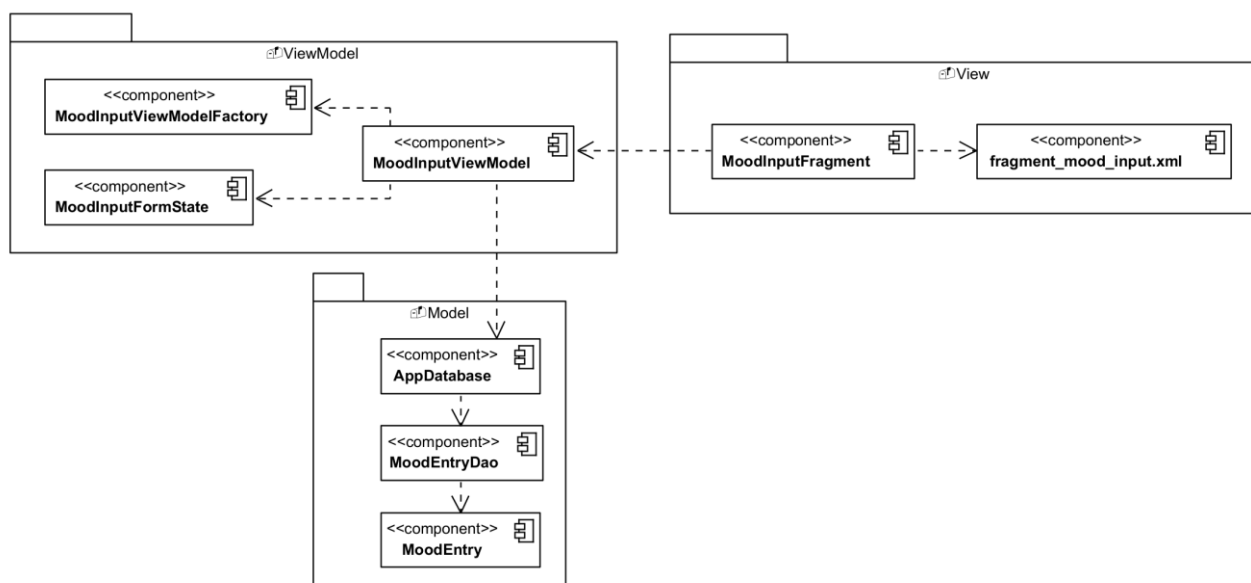


Рисунок В.1 – Диаграмма компонентов функционала добавления настроения

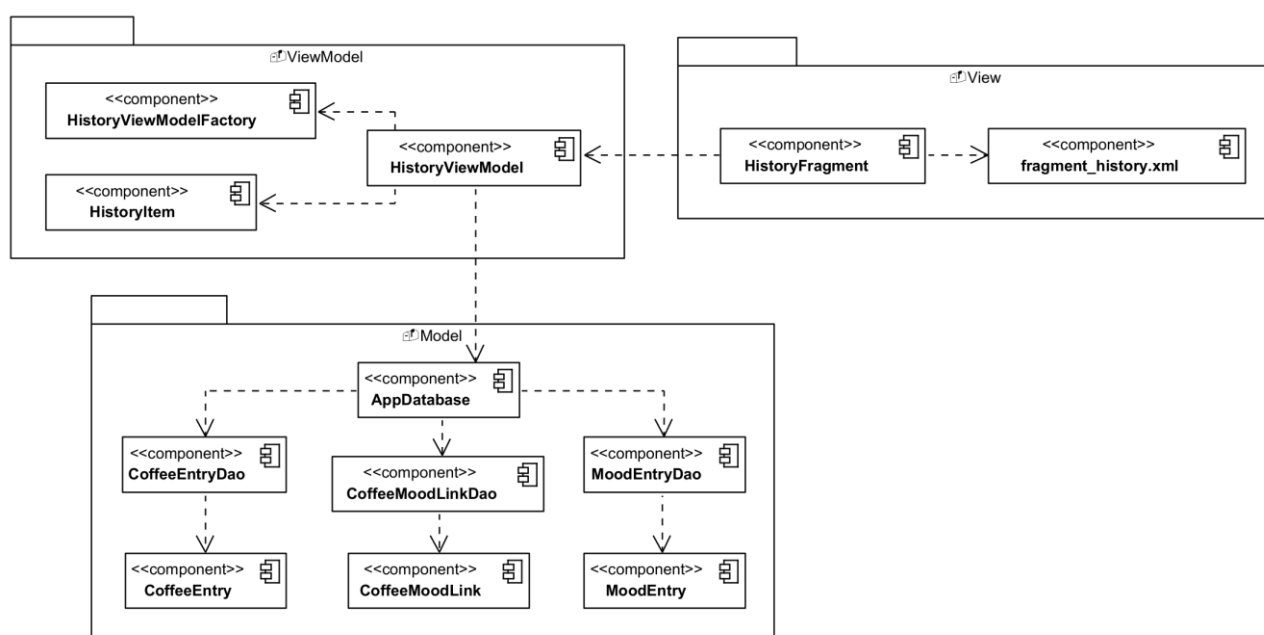


Рисунок В.2 – Диаграмма компонентов функционала просмотра истории

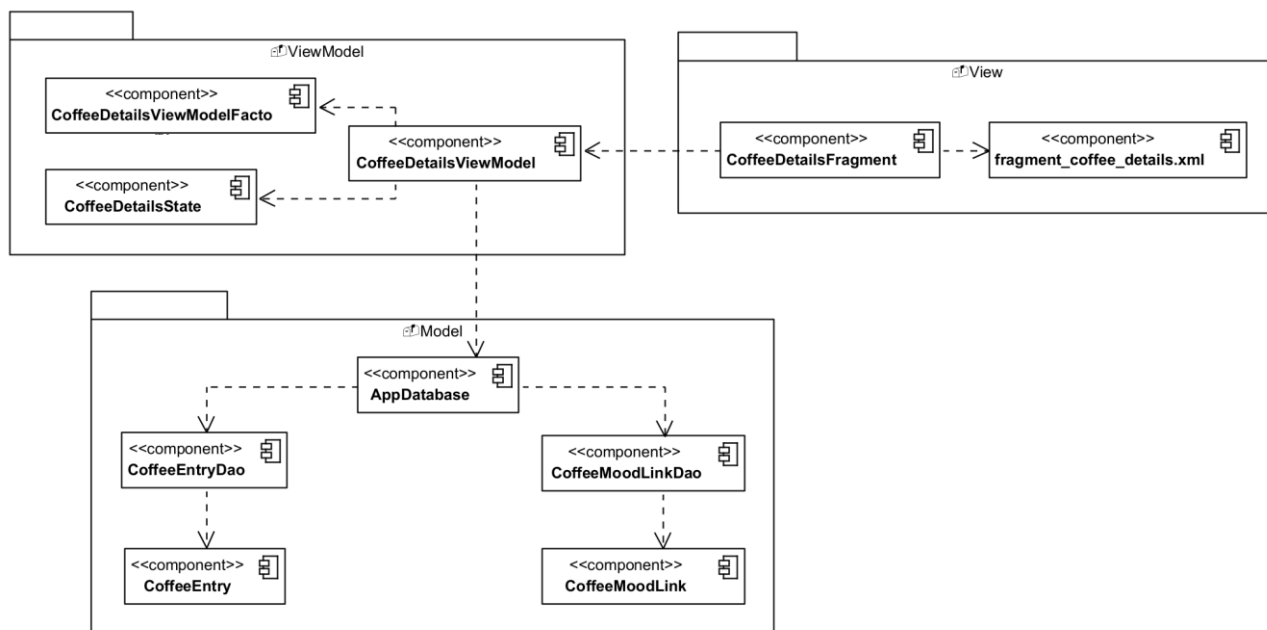


Рисунок В.3 – Диаграмма компонентов функционала просмотра деталей потребления кофе

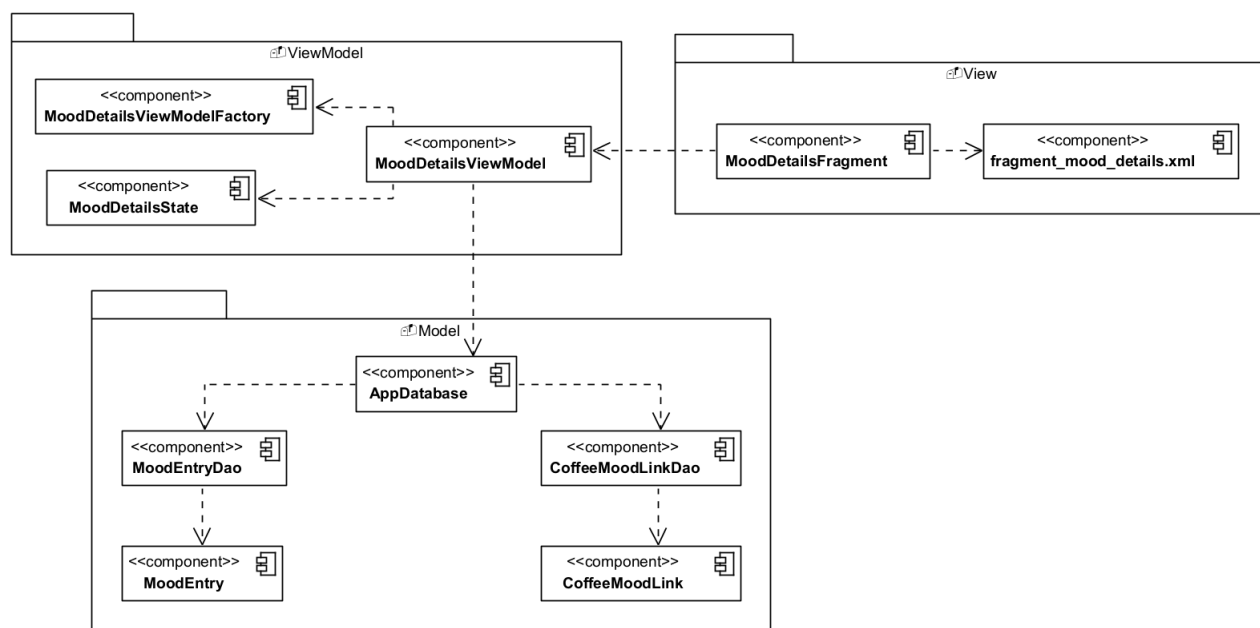


Рисунок В.4 – Диаграмма компонентов функционала просмотра деталей записи настроения

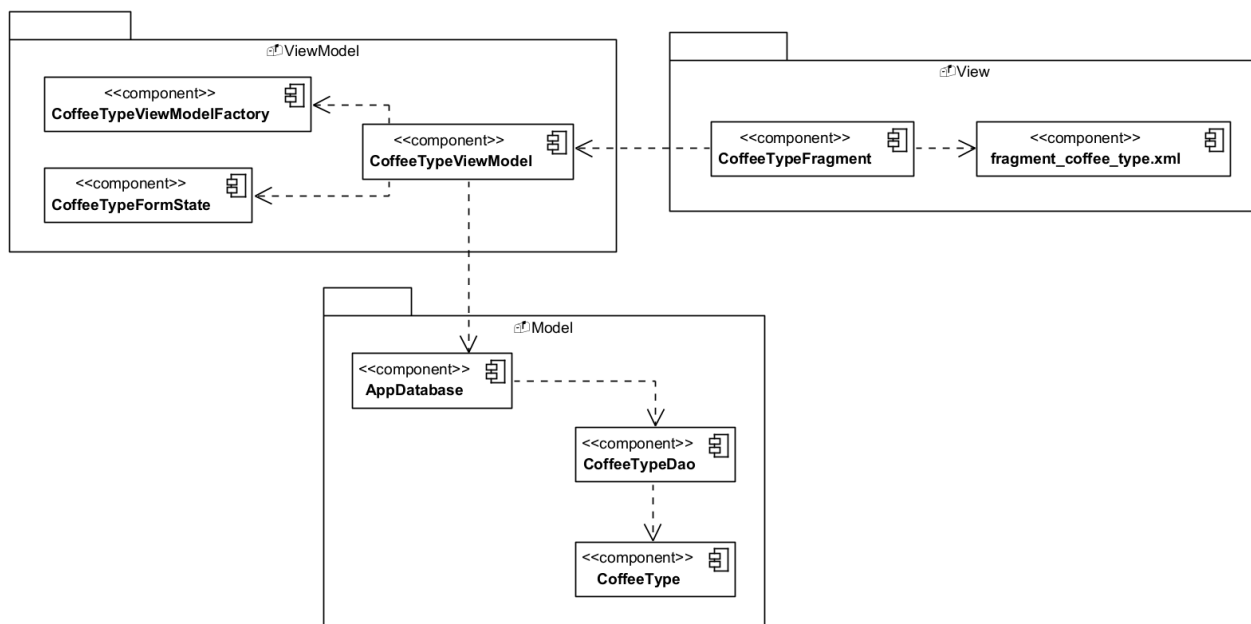


Рисунок В.5 – Диаграмма компонентов функционала добавления нового вида кофе

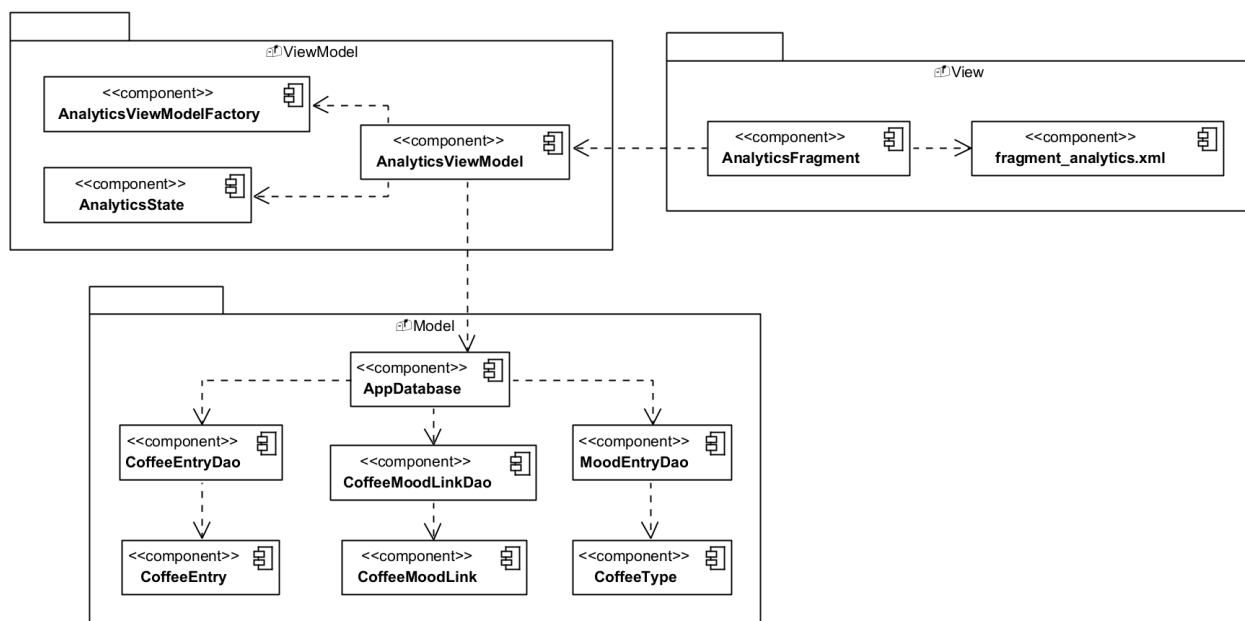


Рисунок В.6 – Диаграмма компонентов функционала просмотра экрана «Аналитика»

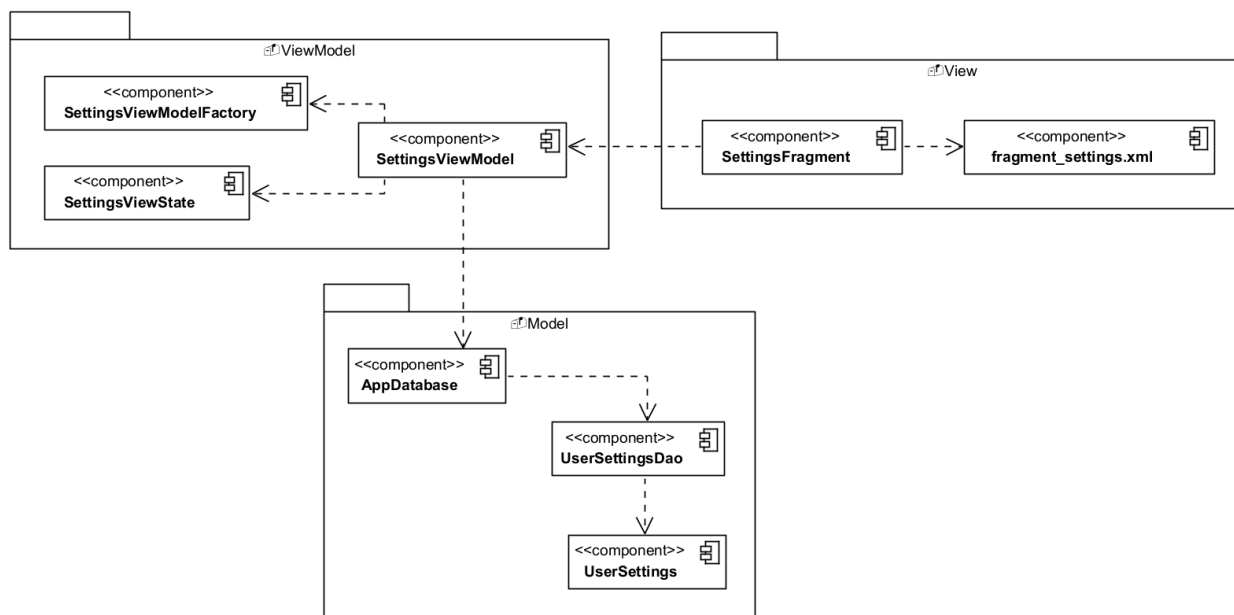


Рисунок В.7 – Диаграмма компонентов функционала просмотра экрана
«Настройки»

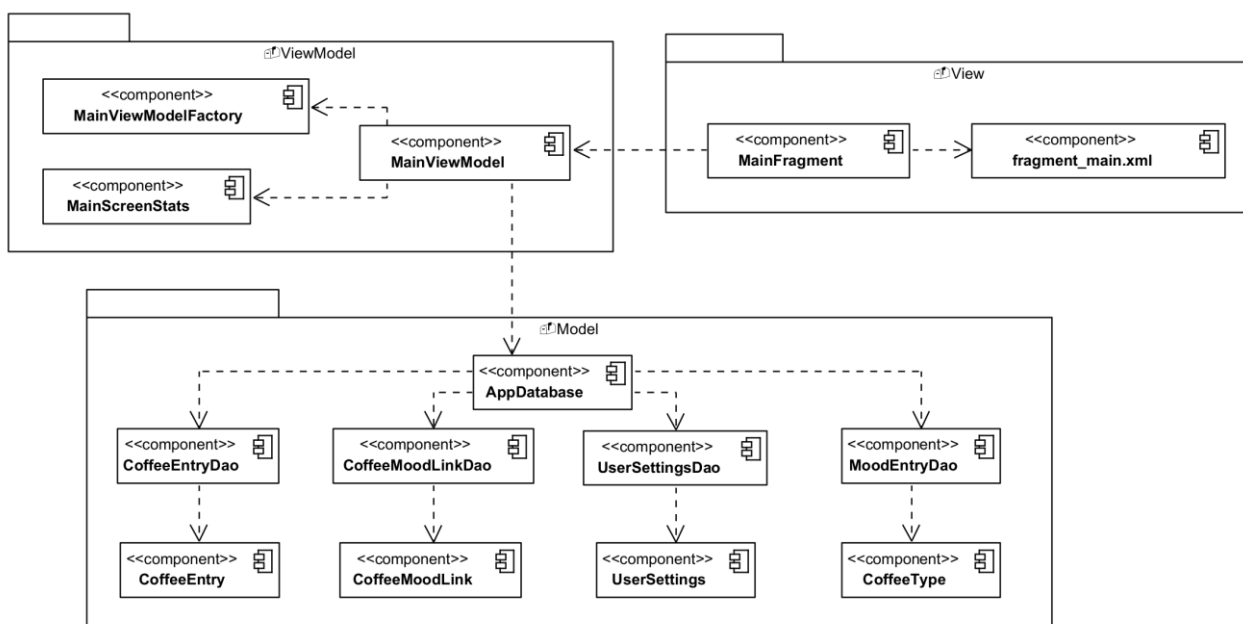


Рисунок В.8 – Диаграмма компонентов функционала просмотра экрана
«Главная»

Приложение Г
(рекомендуемое)

Реализация пользовательского интерфейса

Главная

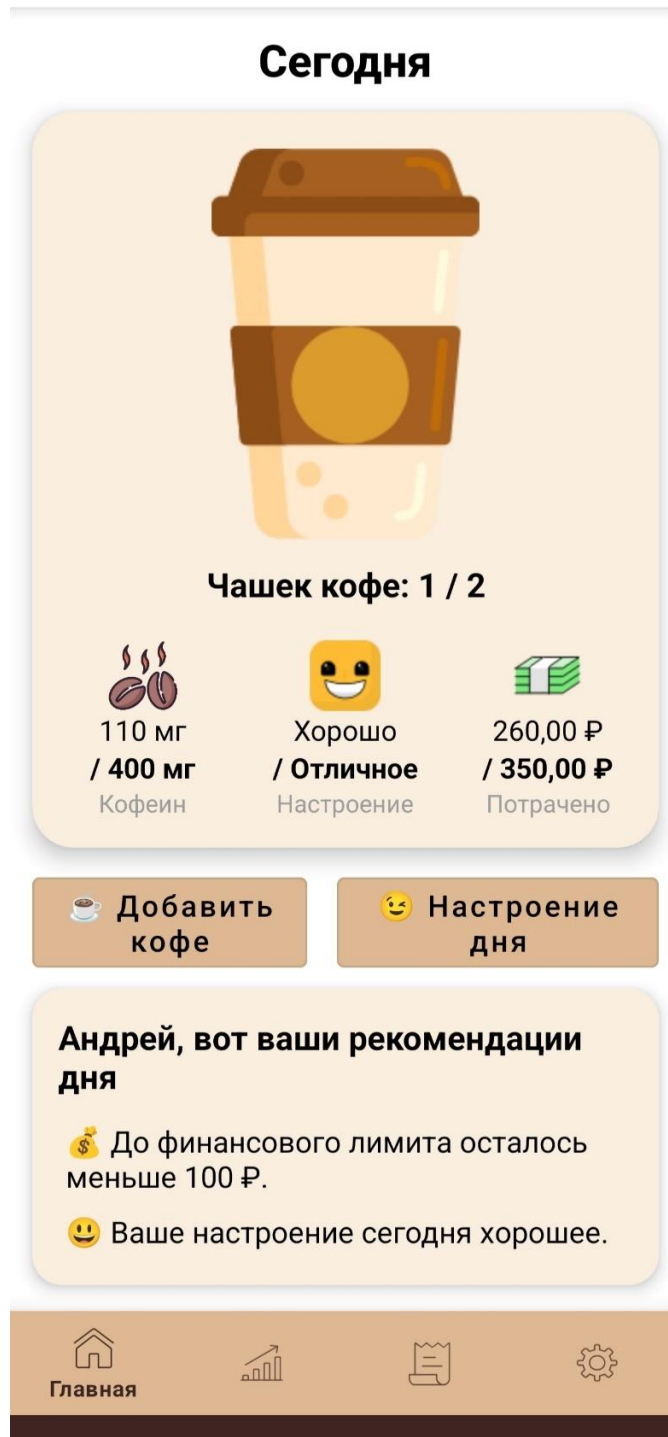


Рисунок Г.1 – Экран вкладки «Главная»

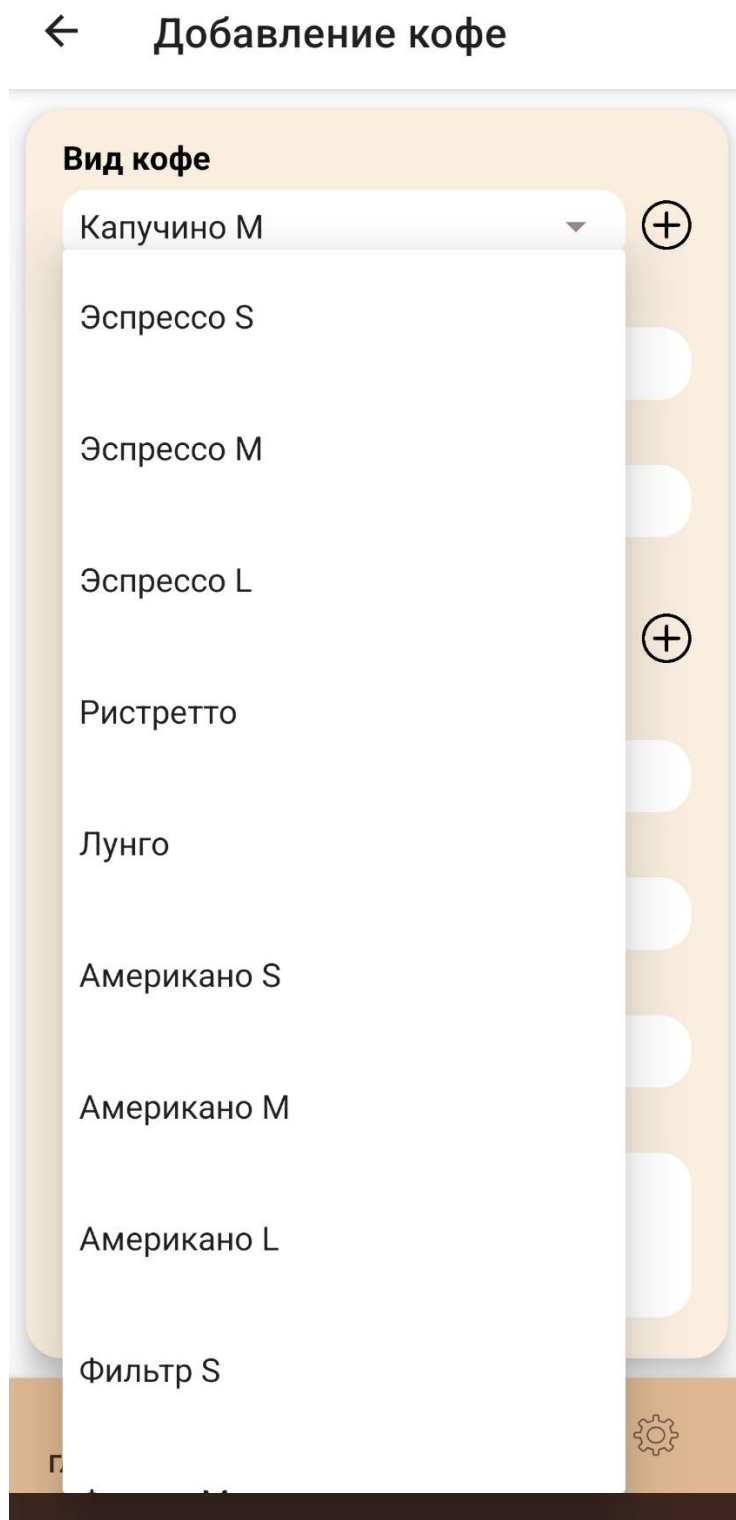


Рисунок Г.2 – Список видов кофе на экране «Добавление кофе»

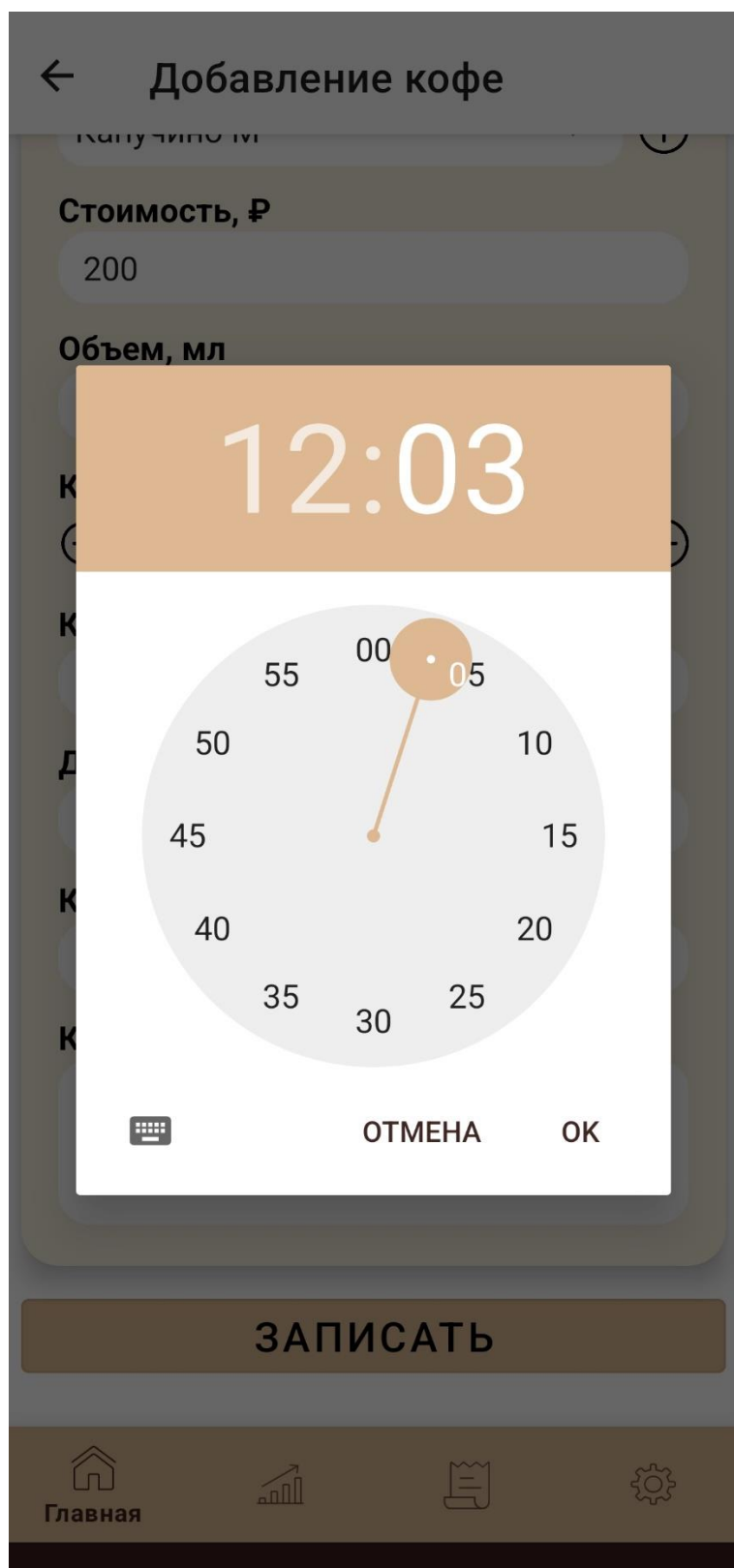


Рисунок Г.3 – Окно выбора времени на экране «Добавление кофе»

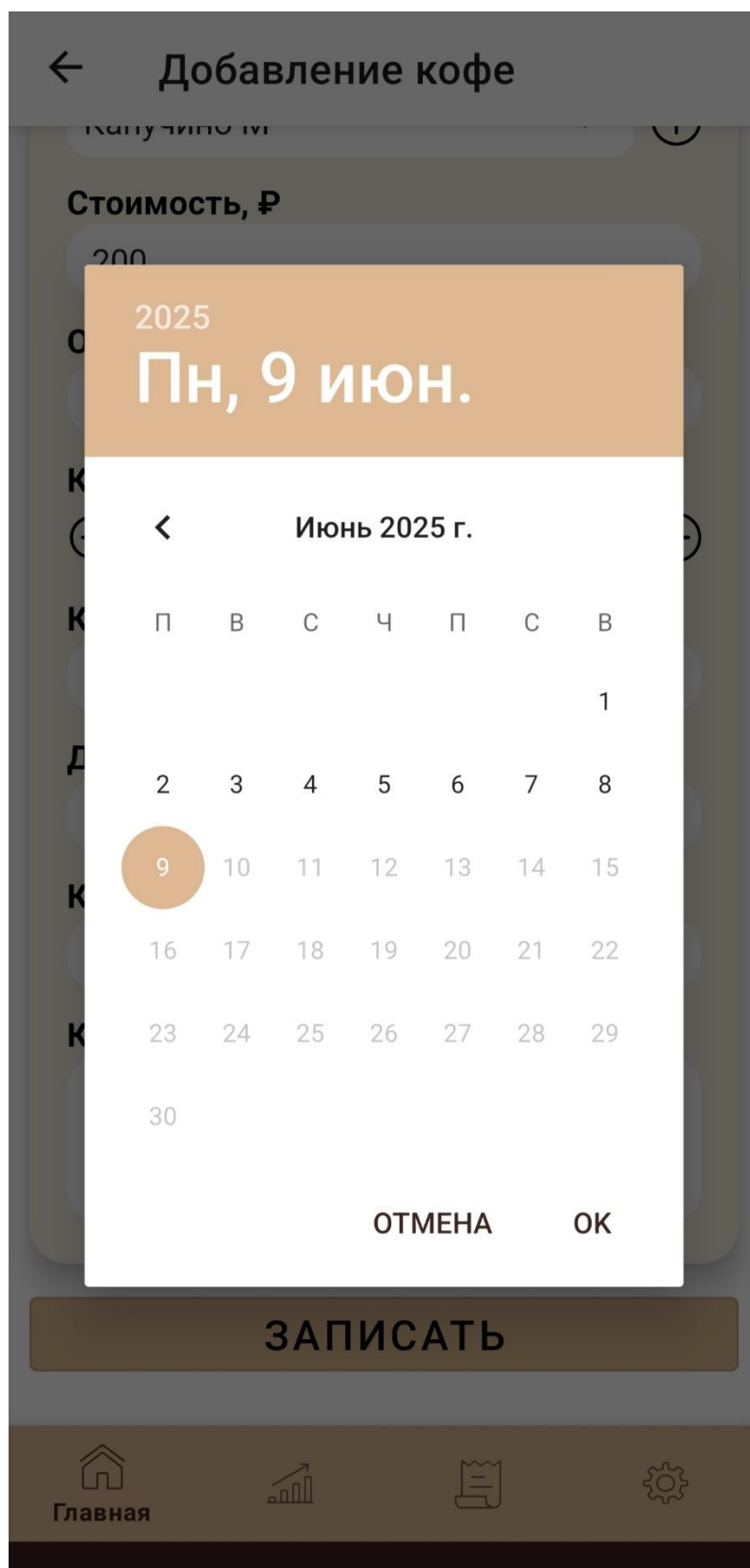


Рисунок Г.4 – Окно выбора даты на экране «Добавление кофе»

 **Добавление кофе**

Вид кофе
Капучино М  

Стоимость, ₽
200

Объем, мл
200

Количество, шт
 1 

Кофеин, мг
64,0

Дата
09.06.2025

Время
07:02

Кофейня
Шишки орешки

Комментарий
Вкусный!

ЗАПИСАТЬ

 Главная   

Рисунок Г.5 – Экран «Добавление кофе»

←

Добавление вида кофе

Вид кофе

Американо

Стандартный объем, мл

250

Среднее значение мг кофеина в 1 мл напитка

0.5

ЗАПИСАТЬ


Главная







Рисунок Г.6 – Экран «Добавление вида кофе»

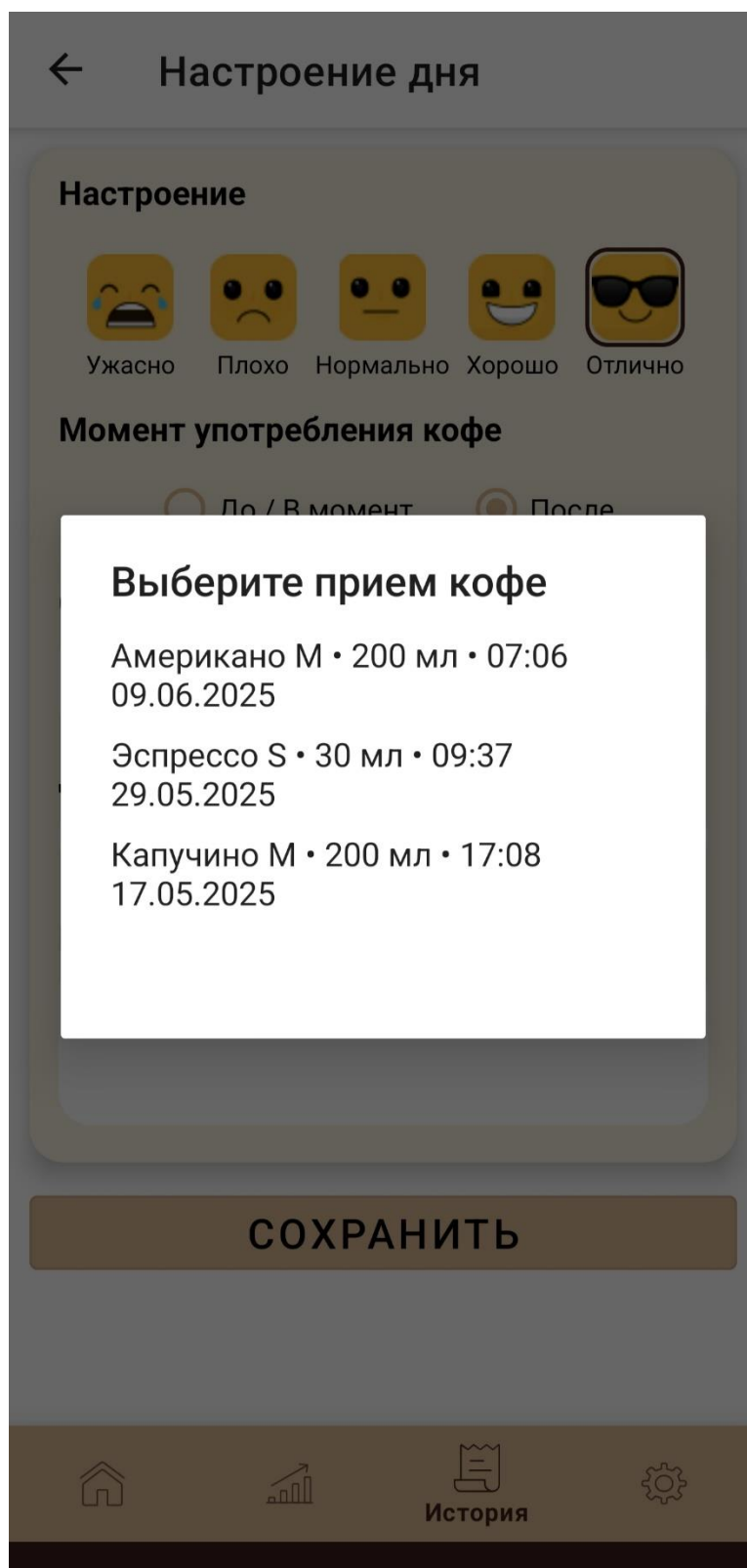


Рисунок Г.7 – Окно выбора приема кофе на экране «Настроение дня»

 **Настроение дня**

Настроение



Ужасно Плохо Нормально Хорошо Отлично

Момент употребления кофе

☐ До / В момент ☒ После

Связанный приём кофе

Американо М • 200 мл • 07:06
09.06.2025

Дата **Время**

09.06.2025 07:07

Комментарий

Обожаю кофе

СОХРАНИТЬ

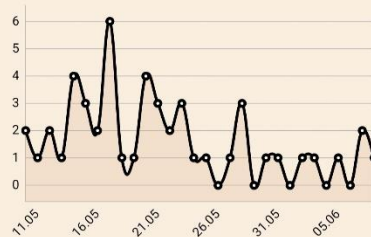
   

История

Рисунок Г.8 – Экран «Настроение дня»

Аналитика

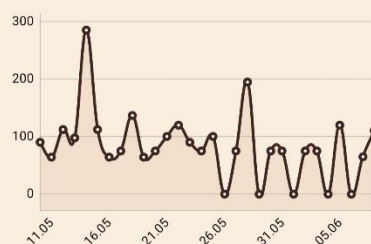
Чашки (за 30 дней)



Количество за месяц: 49

Цель за месяц не более 50

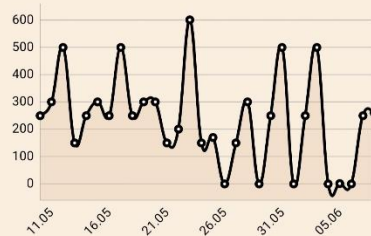
Кофеин, мг (за 30 дней)



Среднее за месяц: 101 мг

Цель среднего значения не более 400 мг

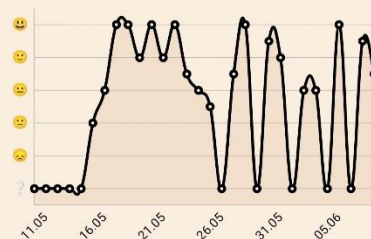
Деньги, ₽ (за 30 дней)



Сумма за месяц: 7081,00 ₽

Цель за месяц не более 10000,00 ₽

Настроение (за 30 дней)



Среднее за месяц: Хорошо



Рисунок Г.9 – Экран «Аналитика»

История

Период:  02.06.2025 - 09.06.2025

Кофе

Настроение

Сегодня



Настроение

Отличное

07:07



Кофе: Американо М

1 шт • 200 мл • 260.0₽

07:06



Настроение

Плохое

07:06

Вчера



Настроение

Отличное

17:25



Кофе: Макиато

2 шт • 50 мл • 250.0₽

17:02



Настроение

Хорошее

17:02

06.06.2025

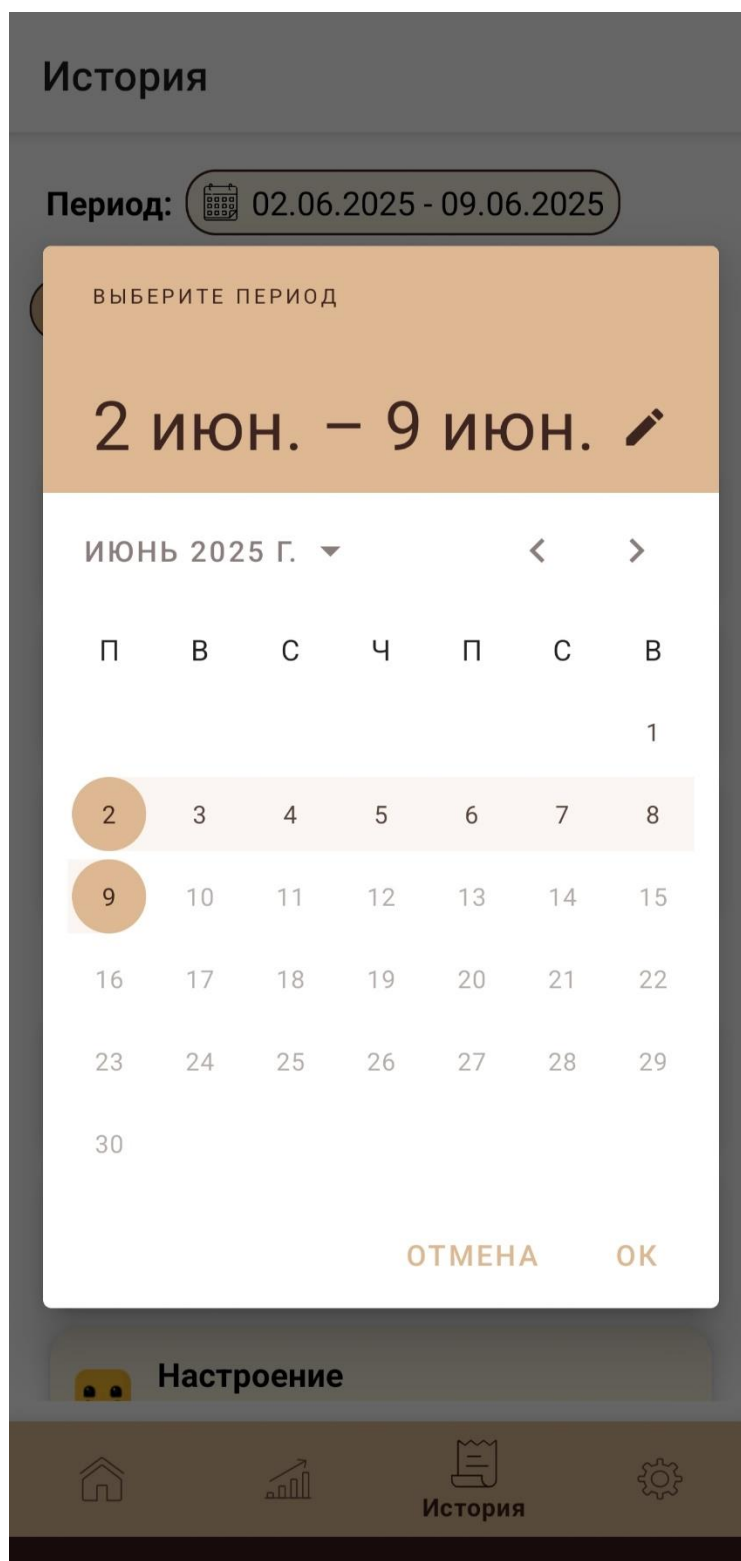


Рисунок Г.11 – Окно выбора периода отображения на экране «История»

История

Период:  02.06.2025 - 09.06.2025

Кофе

Настроение

Сегодня



Кофе: Американо М

1 шт • 200 мл • 260.0₽

07:06

Вчера



Кофе: Макиато

2 шт • 50 мл • 250.0₽

17:02

06.06.2025



Кофе: Лунго

1 шт • 80 мл • 1.0₽

22:55

04.06.2025



Кофе: Эспрессо S

1 шт • 30 мл • 500.0₽

08:27

03.06.2025



История



Рисунок Г.12 – Отображение только записей кофе на экране «История»

История

Период:  02.06.2025 - 09.06.2025

Кофе

Настроение

Сегодня



Настроение

Отличное

07:07



Настроение

Плохое

07:06

Вчера



Настроение

Отличное

17:25



Настроение

Хорошее

17:02

06.06.2025



Настроение

Отличное

23:35



История



Рисунок Г.13 – Отображение только записей настроения на экране «История»

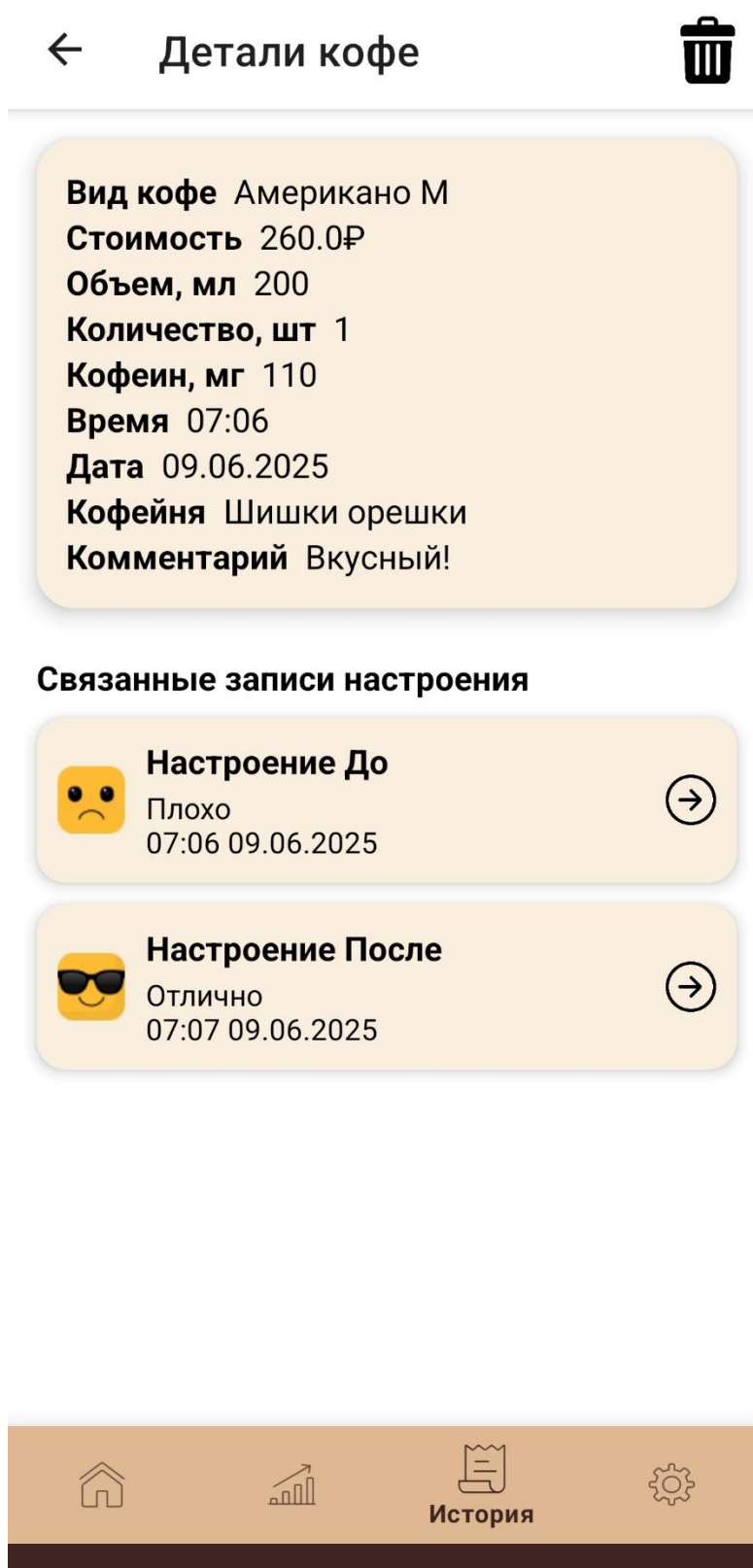


Рисунок Г.14 – Экран «Детали кофе»

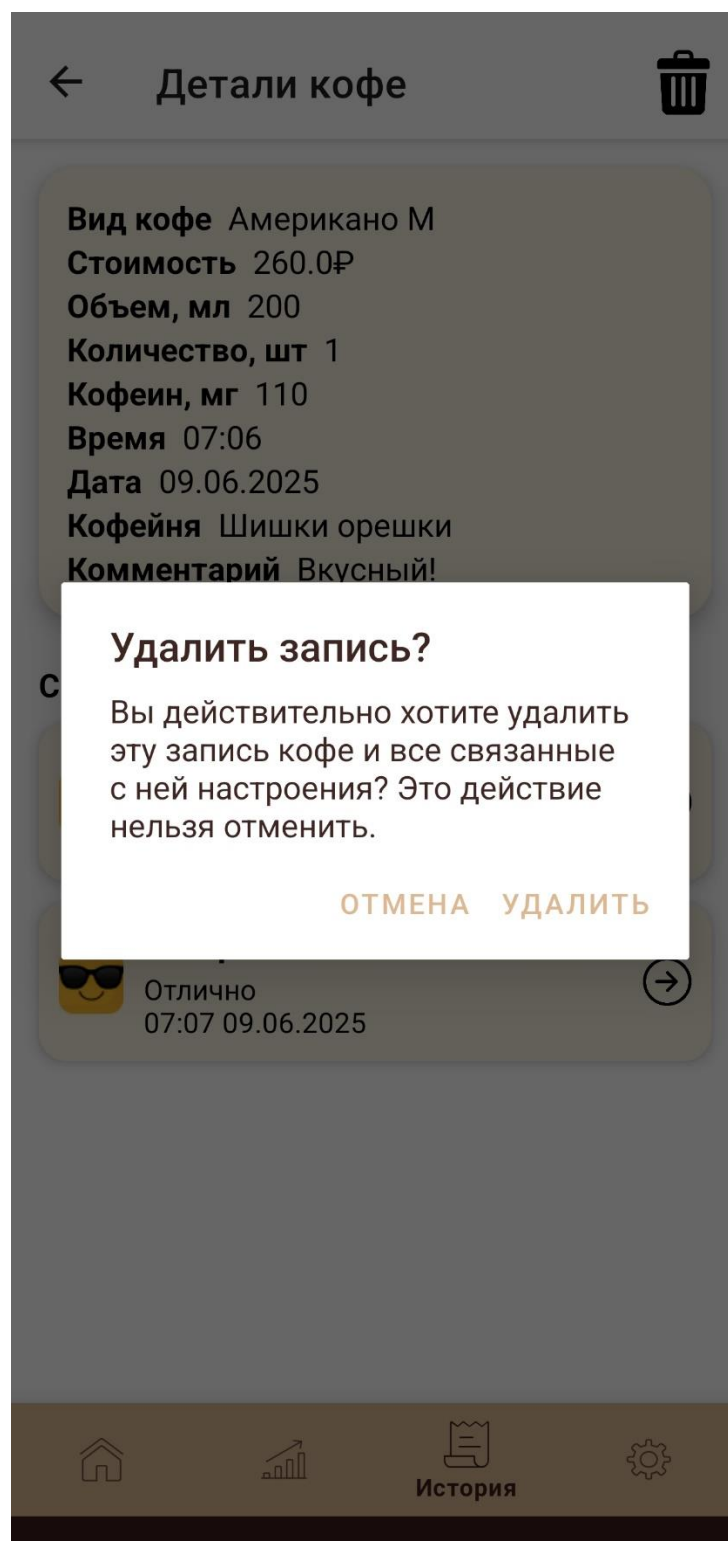


Рисунок Г.15 – Окно подтверждения удаления на экране «Детали кофе»

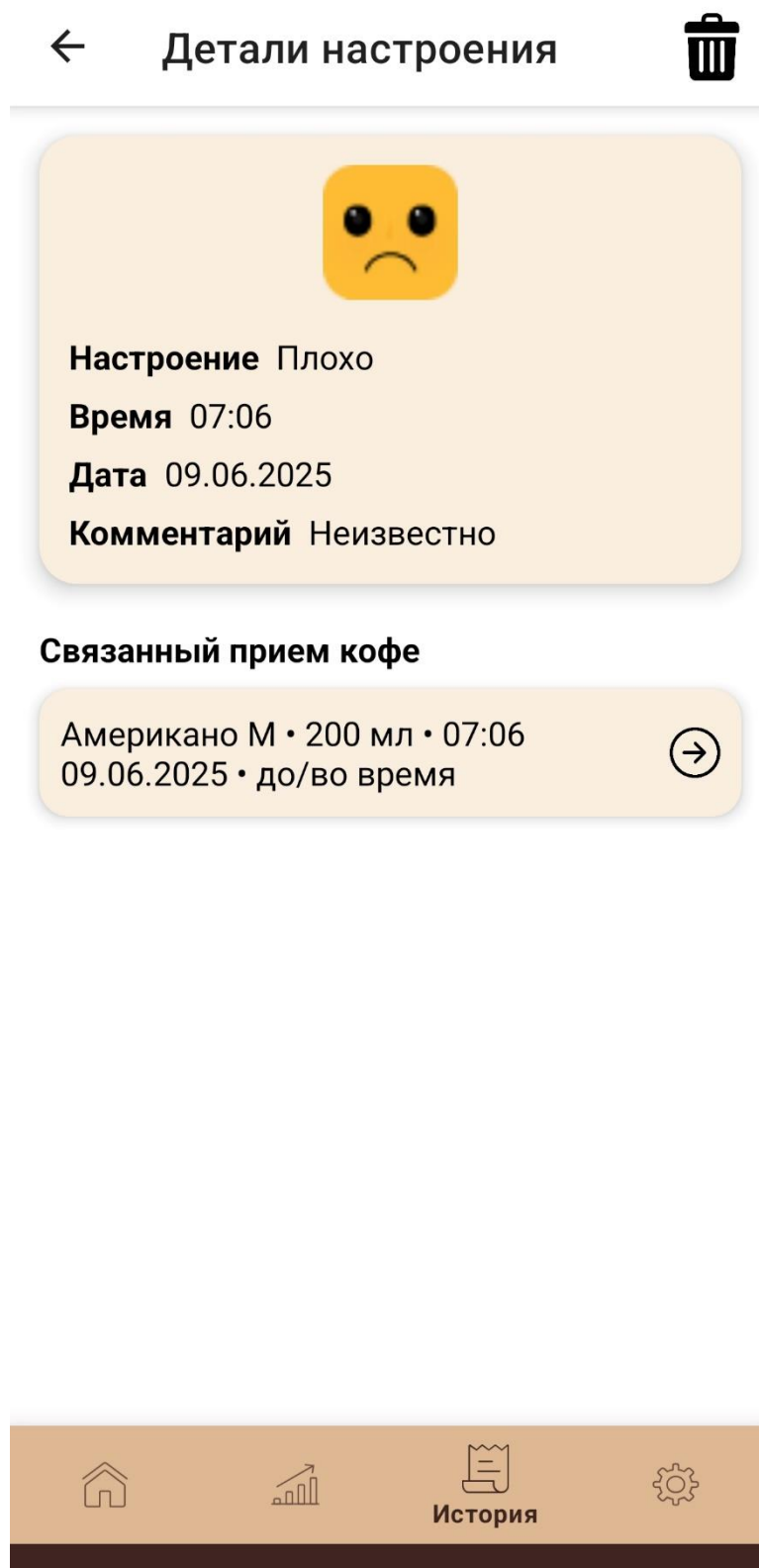


Рисунок Г.16 – Экран «Детали настройки»

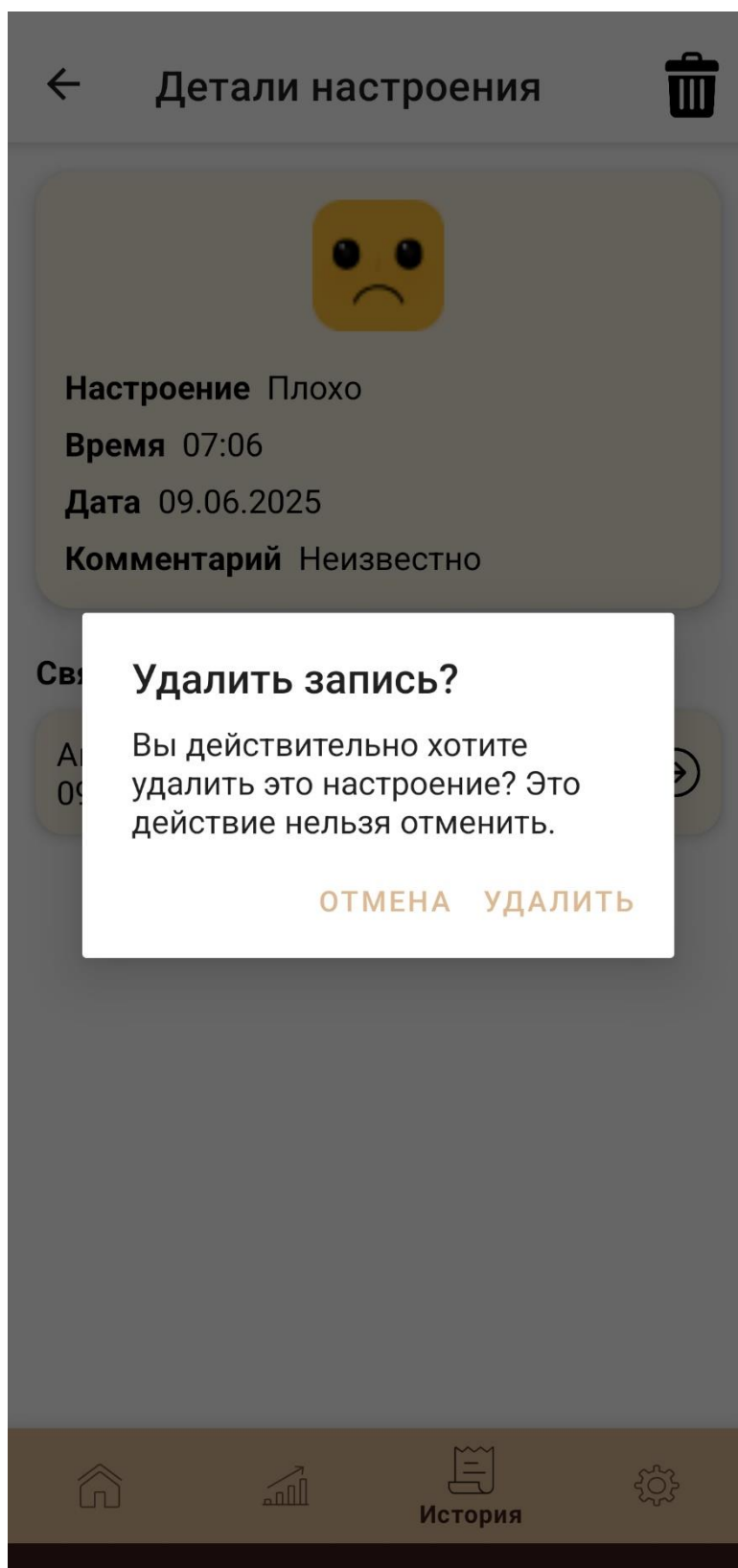


Рисунок Г.17 – Окно подтверждения удаления на экране «Детали кофе»

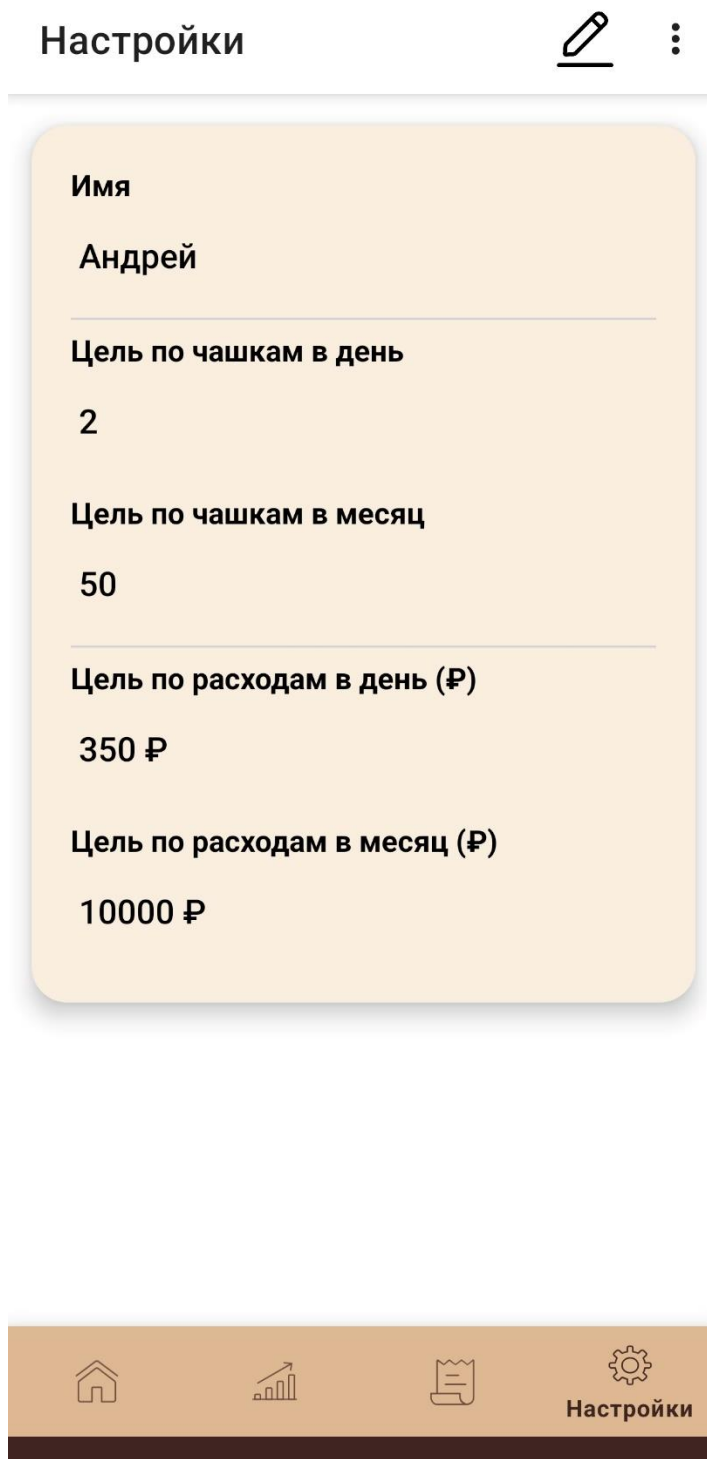


Рисунок Г.18 – Экран «Настройки» в режиме просмотра

Настройки



Имя

Цель по чашкам в день

Цель по чашкам в месяц

Цель по расходам в день (₽)

Цель по расходам в месяц (₽)

Сохранить



Настройки

Рисунок Г.19 – Экран «Настройки» в режиме редактирования

Приложение Д
(рекомендуемое)
Код модульных тестов

Листинг Д.1 – Код модульных тестов для бизнес-логики страницы аналитики

```

package com.example.coffies

@ExperimentalCoroutinesApi
class AnalyticsViewModelTest {

    private val dispatcher = StandardTestDispatcher()

    @Mock
    lateinit var db: AppDatabase
    @Mock
    lateinit var coffeeEntryDao:
com.example.coffies.database.coffeentry.CoffeeEntryDao
    @Mock
    lateinit var moodEntryDao:
com.example.coffies.database.moodentry.MoodEntryDao
    @Mock
    lateinit var userSettingsDao:
com.example.coffies.database.usersettings.UserSettingsDao

    private lateinit var viewModel: AnalyticsViewModel

    @Before
    fun setUp() {
        MockitoAnnotations.openMocks(this)
        Dispatchers.setMain(dispatcher)
        whenever(db.coffeeEntryDao()).thenReturn(coffeeEntryDao)
        whenever(db.moodEntryDao()).thenReturn(moodEntryDao)
        whenever(db.userSettingsDao()).thenReturn(userSettingsDao)
    }

    @After
    fun tearDown() {
        Dispatchers.resetMain()
    }

    private fun coffee(
        id: Int = 1, date: String, caffeine: Double, quantity: Int
        = 1, price: Double = 100.0
    ) = CoffeeEntry(id, 1, 100, quantity, caffeine, price, date,
        "09:00", null, null)

    private fun mood(id: Int, level: Int, date: String) =
MoodEntry(id, level, date, "10:00", null)

    @Test
    fun `empty db gives default state`() = runTest {
        whenever(coffeeEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(emptyList())
        whenever(moodEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(emptyList())
    }
}

```

```

whenever(userSettingsDao.getSettingsOnce()).thenReturn(null)
    viewModel = AnalyticsViewModel(db)
    advanceUntilIdle()
    val state = viewModel.state.value
    Assert.assertEquals("-", state.caffeineAvg)
    Assert.assertEquals("-", state.cupsSum)
    Assert.assertEquals("-", state.moneySum)
    Assert.assertEquals(0, state.moodAvg)
}

@Test
fun `one coffee entry, caffeine and cups shown`() = runTest {
    val now = java.time.LocalDate.now().toString()
    whenever(
        coffeeEntryDao.getAllBetweenDates(
            any(),
            any()
        )
    ).thenReturn(listOf(coffee(date = now, caffeine = 150.0)))
    whenever(moodEntryDao.getAllBetweenDates(any(),
any()))).thenReturn(emptyList())

whenever(userSettingsDao.getSettingsOnce()).thenReturn(null)
    viewModel = AnalyticsViewModel(db)
    advanceUntilIdle()
    val state = viewModel.state.value
    Assert.assertTrue(state.caffeineAvg.contains("mg"))
    Assert.assertEquals("1", state.cupsSum)
}

@Test
fun `coffee for several days, caffeine average correct`() =
runTest {
    val now = java.time.LocalDate.now()
    val coffeeList = listOf(
        coffee(date = now.minusDays(2).toString(), caffeine =
200.0),
        coffee(date = now.minusDays(1).toString(), caffeine =
400.0)
    )
    whenever(coffeeEntryDao.getAllBetweenDates(any(),
any()))).thenReturn(coffeeList)
    whenever(moodEntryDao.getAllBetweenDates(any(),
any()))).thenReturn(emptyList())

whenever(userSettingsDao.getSettingsOnce()).thenReturn(null)
    viewModel = AnalyticsViewModel(db)
    advanceUntilIdle()
    val state = viewModel.state.value
    Assert.assertTrue(state.caffeineAvg.contains("mg"))
}

```

```

@Test
fun `mood average calculated correctly`() = runTest {
    val now = java.time.LocalDate.now().toString()
    whenever(coffeeEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(emptyList())
    whenever(moodEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(
        listOf(
            mood(1, 3, now), mood(2, 5, now), mood(3, 1, now)
        )
    )

    whenever(userSettingsDao.getSettingsOnce()).thenReturn(null)
    viewModel = AnalyticsViewModel(db)
    advanceUntilIdle()
    val state = viewModel.state.value
    Assert.assertEquals(3, state.moodAvg)
}

@Test
fun `cups goal exceeded is true`() = runTest {
    val now = java.time.LocalDate.now().toString()
    whenever(coffeeEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(
        listOf(
            coffee(date = now, caffeine = 50.0, quantity = 2),
            coffee(date = now, caffeine = 50.0, quantity = 3)
        )
    )
    whenever(moodEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(emptyList())
    whenever(userSettingsDao.getSettingsOnce()).thenReturn(
        UserSettings(
            monthly_cup_goal = 4,
            monthly_spend_goal = 0f
        )
    )
    viewModel = AnalyticsViewModel(db)
    advanceUntilIdle()
    Assert.assertTrue(viewModel.state.value.isCupsExceeded)
}

@Test
fun `money sum does not exceed goal`() = runTest {
    val now = java.time.LocalDate.now().toString()
    whenever(coffeeEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(
        listOf(
            coffee(date = now, caffeine = 20.0, price = 120.0)
        )
    )
}

```

```

        whenever(moodEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(emptyList())
        whenever(userSettingsDao.getSettingsOnce()).thenReturn(
            UserSettings(
                monthly_cup_goal = 0,
                monthly_spend_goal = 300f
            )
        )
        viewModel = AnalyticsViewModel(db)
        advanceUntilIdle()
        Assert.assertFalse(viewModel.state.value.isMoneyExceeded)
    }

    @Test
    fun `money sum exceeds goal`() = runTest {
        val now = java.time.LocalDate.now().toString()
        whenever(coffeeEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(
            listOf(
                coffee(date = now, caffeine = 20.0, price = 400.0)
            )
        )
        whenever(moodEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(emptyList())
        whenever(userSettingsDao.getSettingsOnce()).thenReturn(
            UserSettings(
                monthly_cup_goal = 0,
                monthly_spend_goal = 300f
            )
        )
        viewModel = AnalyticsViewModel(db)
        advanceUntilIdle()
        Assert.assertTrue(viewModel.state.value.isMoneyExceeded)
    }

    @Test
    fun `caffeine average over 400 is exceeded`() = runTest {
        val now = java.time.LocalDate.now().toString()
        whenever(coffeeEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(
            listOf(
                coffee(date = now, caffeine = 500.0)
            )
        )
        whenever(moodEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(emptyList())

        whenever(userSettingsDao.getSettingsOnce()).thenReturn(null)
        viewModel = AnalyticsViewModel(db)
        advanceUntilIdle()

        Assert.assertTrue(viewModel.state.value.isCaffeineExceeded)
    }

```

```

    }

    @Test
    fun `money sum formatted with ruble sign`() = runTest {
        val now = java.time.LocalDate.now().toString()
        whenever(coffeeEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(
            listOf(
                coffee(date = now, caffeine = 20.0, price =
123.45)
            )
        )
        whenever(moodEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(emptyList())

        whenever(userSettingsDao.getSettingsOnce()) .thenReturn(null)
        viewModel = AnalyticsViewModel(db)
        advanceUntilIdle()

        Assert.assertTrue(viewModel.state.value.moneySum.contains("₽"))
    }

    @Test
    fun `mood points filled for days with mood`() = runTest {
        val now = java.time.LocalDate.now()
        val moods = (0..5).map { mood(it, it % 5 + 1,
now.minusDays(it.toLong()).toString()) }
        whenever(coffeeEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(emptyList())
        whenever(moodEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(moods)

        whenever(userSettingsDao.getSettingsOnce()) .thenReturn(null)
        viewModel = AnalyticsViewModel(db)
        advanceUntilIdle()
        Assert.assertTrue(viewModel.state.value.moodPoints.any {
it.second > 0 })
    }

    @Test
    fun `cups sum includes all entries`() = runTest {
        val now = java.time.LocalDate.now().toString()
        whenever(coffeeEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(
            listOf(
                coffee(date = now, caffeine = 10.0, quantity = 2),
                coffee(date = now, caffeine = 5.0, quantity = 3)
            )
        )
        whenever(moodEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(emptyList())
    }

```

```

whenever(userSettingsDao.getSettingsOnce()).thenReturn(null)
    viewModel = AnalyticsViewModel(db)
    advanceUntilIdle()
    Assert.assertEquals("5", viewModel.state.value.cupsSum)
}

@Test
fun `no coffee gives dash in cupsSum`() = runTest {
    whenever(coffeeEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(emptyList())
    whenever(moodEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(emptyList())

whenever(userSettingsDao.getSettingsOnce()).thenReturn(null)
    viewModel = AnalyticsViewModel(db)
    advanceUntilIdle()
    Assert.assertEquals("-", viewModel.state.value.cupsSum)
}

@Test
fun `no moods gives moodAvg zero`() = runTest {
    whenever(coffeeEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(emptyList())
    whenever(moodEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(emptyList())

whenever(userSettingsDao.getSettingsOnce()).thenReturn(null)
    viewModel = AnalyticsViewModel(db)
    advanceUntilIdle()
    Assert.assertEquals(0, viewModel.state.value.moodAvg)
}

@Test
fun `many cups in one day summed correctly`() = runTest {
    val now = java.time.LocalDate.now().toString()
    whenever(
        coffeeEntryDao.getAllBetweenDates(
            any(),
            any()
        )
    ).thenReturn(List(10) { coffee(date = now, caffeine =
10.0, quantity = 1) })
    whenever(moodEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(emptyList())

whenever(userSettingsDao.getSettingsOnce()).thenReturn(null)
    viewModel = AnalyticsViewModel(db)
    advanceUntilIdle()
    Assert.assertEquals("10", viewModel.state.value.cupsSum)
}

@Test

```

```

        fun `defaults for all fields when no data`() = runTest {
            whenever(coffeeEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(emptyList())
            whenever(moodEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(emptyList())

            whenever(userSettingsDao.getSettingsOnce()) .thenReturn(null)
            viewModel = AnalyticsViewModel(db)
            advanceUntilIdle()
            val state = viewModel.state.value
            Assert.assertEquals("-", state.caffeineAvg)
            Assert.assertEquals("-", state.moneySum)
            Assert.assertEquals("-", state.cupsSum)
            Assert.assertEquals(0, state.moodAvg)
        }

        @Test
        fun `no cup goal means no excess`() = runTest {
            val now = java.time.LocalDate.now().toString()
            whenever(coffeeEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(
                listOf(
                    coffee(date = now, caffeine = 20.0, quantity = 10)
                )
            )
            whenever(moodEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(emptyList())
            whenever(userSettingsDao.getSettingsOnce()) .thenReturn(
                UserSettings(
                    monthly_cup_goal = 0,
                    monthly_spend_goal = 0f
                )
            )
            viewModel = AnalyticsViewModel(db)
            advanceUntilIdle()
            Assert.assertFalse(viewModel.state.value.isCupsExceeded)
        }

        @Test
        fun `no money goal means no excess`() = runTest {
            val now = java.time.LocalDate.now().toString()
            whenever(coffeeEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(
                listOf(
                    coffee(date = now, caffeine = 20.0, price =
1000.0)
                )
            )
            whenever(moodEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(emptyList())
            whenever(userSettingsDao.getSettingsOnce()) .thenReturn(
                UserSettings(

```

```

        monthly_cup_goal = 0,
        monthly_spend_goal = 0f
    )
)
viewModel = AnalyticsViewModel(db)
advanceUntilIdle()
Assert.assertFalse(viewModel.state.value.isMoneyExceeded)
}

@Test
fun `empty days filled with zeros`() = runTest {
    whenever(coffeeEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(emptyList())
    whenever(moodEntryDao.getAllBetweenDates(any(),
any())) .thenReturn(emptyList())

    whenever(userSettingsDao.getSettingsOnce()) .thenReturn(null)
    viewModel = AnalyticsViewModel(db)
    advanceUntilIdle()
    val points = viewModel.state.value.caffeinePoints
    Assert.assertTrue(points.all { it.second == 0.0 })
}

}

```

Листинг Д.2 – Код функции «loadGroupedHistoryItems»

```

private suspend fun loadGroupedHistoryItems(
    dateRange: Pair<LocalDate, LocalDate>,
    filters: HistoryFilters,
): List<HistoryItem> {
    val (start, end) = dateRange
    val startStr = start.format(DateTimeFormatter.ISO_LOCAL_DATE)
    val endStr = end.format(DateTimeFormatter.ISO_LOCAL_DATE)

    val coffeeTypeMap = runCatching {
        db.coffeeTypeDao().getAllCoffeeTypeNames().associate {
it.id to it.name }
    }.getOrElse(emptyMap())

    val coffee = if (filters.showCoffee) {
        runCatching {
            db.coffeeEntryDao().getEntriesForPeriod(startStr,
endStr)
                .map { it.toHistoryItem(coffeeTypeMap) }
            }.getOrElse(emptyList())
        } else emptyList()

    val mood = if (filters.showMood) {
        runCatching {
            db.moodEntryDao().getEntriesForPeriod(startStr,
endStr)
                .map { it.toHistoryItem() }

```

```

        }.getOrElse(emptyList())
    } else emptyList()

    val allItems = (coffee + mood).sortedByDescending {
        when (it) {
            is HistoryItem.CoffeeItem -> it.dateTime
            is HistoryItem.MoodItem -> it.dateTime
            is HistoryItem.DateHeader -> LocalDateTime.MIN
        }
    }

    return allItems
        .groupBy {
            when (it) {
                is HistoryItem.CoffeeItem ->
it.dateTime.toLocalDate()
                is HistoryItem.MoodItem ->
it.dateTime.toLocalDate()
                is HistoryItem.DateHeader -> it.date
            }
        }
        .toSortedMap(compareByDescending { it })
        .flatMap { (date, items) ->
listOf(HistoryItem.DateHeader(date)) + items }
    }

```