

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

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

Факультет

Вычислительной техники

Кафедра

МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

Направленность Информационные системы и технологии

БАКАЛАВРСКАЯ РАБОТА

на тему

Электронный журнал питания с подбором блюд

Студент Курносов Михаил Алексеевич
(подпись, дата) (ФИО полностью)

Руководитель Генералова А.А.
(подпись, дата) (фамилия, инициалы)

Нормоконтролер Такташкин Д.В.
(подпись, дата) (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 _____ 202____)

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

Пенза, 2026

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

Факультет
Вычислительной техники

Кафедра
МОиПЭВМ

«Утверждаю»
Заведующий кафедрой
А.Ю. Козлов
«26» декабря 2025 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Курносков Михаил Алексеевич гр. 22ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Электронный журнал питания с подбором блюд

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

2. Руководитель работы к.т.н., доцент кафедры «МОиПЭВМ» Генералова А.А.

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

1) Разработать мобильное приложение для платформы Android, предназначенное для ведения электронного журнала питания, подсчета калорий и подбора блюд с использованием интеллектуального помощника;

2) Среда функционирования: мобильное устройство или эмулятор Android под управлением операционной системы Android;

3) Среда разработки: ОС Windows 10
Языковые средства разработки: Kotlin, SQL, UML;
Инструментальные средства: Android Studio, Gradle, Jetpack Compose, Room, Retrofit, OkHttp, Gson, GigaChat API, Draw.io, Visual Paradigm;
Средства тестирования: Android Emulator, JUnit 4;

4) Основные функции приложения:
а) регистрация и авторизация пользователей;
б) заполнение анкеты пользователя с указанием возраста, пола, роста, веса, уровня физической активности, цели и аллергий;
в) расчет индивидуальной дневной нормы калорий и отображение данных профиля; пользователя;
г) ведение дневника питания с возможностью добавления и удаления блюд;

д) отображение суммарной калорийности и количества белков, жиров и углеводов за день;			
е) генерация индивидуального рациона питания с использованием ИИ-диетолога;			
ж) корректировка составленного рациона по запросу пользователя на естественном языке;			
з) сохранение составленного ИИ-рациона в дневник питания;			
и) хранение данных пользователей, профилей, блюд, записей дневника и истории взаимодействия с ИИ в локальной базе данных.			
5) Особенности: приложение использует локальную базу данных Root для хранения пользовательских данных и дневника питания. Подбор рациона выполняется с учетом индивидуальных параметров пользователя, целевой калорийности, цели питания и указанных аллергий. Взаимодействие с ИИ-диетологом осуществляется через API GigaChat, что позволяет пользователю изменять состав рациона в диалоговом режиме, например добавлять, удалять или заменять продукты.			
4.	Перечень	подлежащих	разработке
вопросов			
1) Анализ предметной области и постановка задачи;			
2) Проектирование;			
3) Реализация;			
4) Отладка и тестирование;			
5) Контроль качества приложения;			
6) 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%	08.04.2026 – 14.04.2026	
2	Проектирование программных средств	20%	15.04.2026 – 25.04.2026	
3	Реализация программных средств	30%	26.04.2026 – 12.05.2026	
4	Тестирование программных средств	20%	13.05.2026 – 22.05.2026	
5	Оформление пояснительной записки	20%	23.05.2026 – 03.06.2026	

Дата выдачи задания «26» декабря 2025г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

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

Реферат

Пояснительная записка содержит 145 листов, 33 рисунка, 20 таблиц, 20 использованных источников и 6 приложений.

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, ANDROID, KOTLIN, JETPACK COMPOSE, ROOM, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ, GIGASNAT API, ДНЕВНИК ПИТАНИЯ, MVVM, ТЕСТИРОВАНИЕ.

Объект исследования – приложение контроля питания и составления индивидуального рациона.

Цель выпускной квалификационной работы – спроектировать и разработать мобильное приложение для платформы Android, позволяющее пользователям вести учет питания и получать персонализированные рекомендации по составлению рациона с помощью ИИ-диетолога.

Результаты работы – в ходе выполнения выпускной квалификационной работы было разработано полнофункциональное мобильное приложение. Реализованы модули многопользовательской аутентификации, анкетирования пользователя, ведения дневника питания и чата с ИИ-диетологом. Разработан алгоритм пропорционального распределения весов продуктов для точного попадания в целевую калорийность. Все поставленные задачи выполнены.

					ПГУ 09.03.02– 8BP211.09 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Курносов М.А.			Электронный журнал питания с подбором блюд. Пояснительная записка	Лит.	Лист	Листов
Провер.		Генералова А.А.					4	145
Реценз.						Группа 22ВИ1		
Н. Контр.		Такташкин Д.В.						
Утверд.		Козлов А.Ю.						

Содержание

Введение	7
1 Анализ предметной области и постановка задачи	9
1.1 Постановка задачи	10
1.2 Анализ предметной области	12
1.3 Анализ существующих аналогов	14
1.4 Анализ требований к разрабатываемому приложению	20
1.5 Выбор методов и средств разработки	23
2 Проектирование приложения	27
2.1 Архитектура приложения	27
2.2 Проектирование структуры данных	30
2.3 Проектирование пользовательского интерфейса	34
2.4 Проектирование алгоритмов приложения	40
2.5 Проектирование классов приложения	46
3 Реализация мобильного приложения	50
3.1 Общая организация проекта	50
3.2 Реализация локальной базы данных	54
3.3 Реализация регистрации и авторизации пользователя	56
3.4 Реализация анкеты пользователя и расчета дневной нормы калорий	58
3.5 Реализация дневника питания	61
3.6 Реализация ИИ-диетолога	64
4 Отладка и тестирование приложения	69
4.1 Обоснование выбора режима тестирования	69
4.2 Функциональное ручное тестирование пользовательского ввода	71
4.3 Модульное тестирование	75
5 Контроль качества приложения	78
5.1 Обоснование выбора метрик кода	79
5.2 Оценка качества программного кода	80
5.2.1 Метрика Холстеда	80
5.2.2 Метрика Чепина	81

5.3 Вывод по результатам контроля качества.....	6 83
6 Расчет бюджета разработки приложения.....	84
6.1 Планирование разработки и определение трудоемкости	84
6.2 Расчет заработной платы разработчика.....	85
6.3 Расчет страховых взносов, накладных расходов и итогового бюджета разработки.....	87
Заключение.....	89
Список использованных источников.....	91
Приложение А (справочное) Глоссарий	93
Приложение Б (рекомендуемое) Атрибуты сущностей базы данных.....	96
Приложение В (обязательное) UML-диаграммы приложения NutritionDiary	100
Приложение Г (обязательное) Ключевые фрагменты программной логики, локальной базы данных и сетевого взаимодействия	111
Приложение Д (рекомендуемое) Результаты функционального тестирования.....	132
Приложение Е (рекомендуемое) Листинг модульных тестов приложения...	143

Введение

В современном мире все больше людей уделяют внимание контролю рациона, калорийности питания и балансу макроэлементов: белков, жиров и углеводов. Это связано как с распространением заболеваний, вызванных несбалансированным питанием, так и с ростом интереса к здоровому образу жизни. Одним из наиболее доступных инструментов контроля питания являются мобильные приложения, позволяющие фиксировать приемы пищи, рассчитывать энергетическую ценность продуктов и отслеживать выполнение дневной нормы калорий.

Однако большинство существующих решений имеют ряд ограничений. Пользователю часто приходится вручную искать продукты, указывать массу порции и самостоятельно подбирать блюда под суточную норму калорий. Кроме того, многие приложения предлагают универсальные рекомендации и не всегда учитывают индивидуальные параметры человека: возраст, пол, рост, вес, уровень физической активности, наличие аллергий, пищевые предпочтения и цель питания. В результате процесс ведения дневника питания может становиться трудоемким, а рекомендации – недостаточно персонализированными.

Развитие технологий искусственного интеллекта позволяет расширить возможности классических дневников питания. Интеграция большой языковой модели в мобильное приложение дает возможность не только вести учет съеденных продуктов, но и получать индивидуальный рацион, сформированный с учетом параметров пользователя. Дополнительно пользователь может взаимодействовать с интеллектуальным помощником в диалоговом режиме: запрашивать замену продуктов, изменение калорийности или корректировку отдельных приемов пищи.

В рамках данной выпускной квалификационной работы разрабатывается мобильное Android-приложение «Электронный журнал питания с подбором блюд». Приложение предназначено для ведения дневника питания, расчета индивидуальной дневной нормы калорий, учета

белков, жиров и углеводов, а также формирования персонализированного рациона с использованием ИИ-диетолога на основе GigaChat API.

Целью выпускной квалификационной работы является разработка мобильного приложения для платформы Android, предназначенного для ведения электронного журнала питания и подбора блюд с использованием интеллектуального помощника.

Для достижения поставленной цели необходимо решить следующие задачи:

- провести анализ предметной области и существующих программных аналогов;
- сформулировать функциональные и нефункциональные требования к разрабатываемому приложению;
- выбрать методы и средства разработки мобильного приложения;
- спроектировать архитектуру приложения, структуру локальной базы данных и основные пользовательские сценарии;
- реализовать модули регистрации и авторизации, анкетирования пользователя, расчета дневной нормы калорий, ведения дневника питания и взаимодействия с ИИ-диетологом;
- реализовать алгоритм формирования и корректировки рациона с учетом целевой калорийности и пользовательских ограничений;
- провести отладку, функциональное и модульное тестирование приложения;
- выполнить оценку качества программного кода.

Выполнение поставленных задач позволит разработать мобильное приложение, объединяющее функции классического дневника питания и интеллектуального помощника для персонализированного подбора блюд.

1 Анализ предметной области и постановка задачи

Перед началом разработки программного продукта необходимо выполнить анализ предметной области, определить назначение системы, круг пользователей, основные функции приложения и ограничения, которые необходимо учитывать при проектировании. Для мобильного приложения, предназначенного для ведения электронного журнала питания, особое значение имеют корректный расчет пищевой ценности продуктов, хранение пользовательских данных, персонализация рекомендаций и удобство ежедневного использования.

Разрабатываемое приложение должно объединять функции классического дневника питания и интеллектуального помощника. Пользователь должен иметь возможность вручную добавлять блюда в дневник, отслеживать суммарную калорийность и количество белков, жиров и углеводов за день, а также получать автоматически сформированный рацион с учетом индивидуальных параметров. К таким параметрам относятся возраст, пол, рост, вес, уровень физической активности, цель питания и наличие аллергий.

Особенностью разрабатываемой системы является использование ИИ-диетолога, взаимодействие с которым осуществляется через GigaChat API. Интеллектуальный помощник используется для формирования рациона и его последующей корректировки по запросу пользователя на естественном языке. При этом приложение не должно полностью полагаться на ответ внешнего сервиса: продукты должны проверяться по локальной базе, а блюда с аллергенами должны исключаться из итогового рациона. Такой подход позволяет рассматривать ИИ-диетолога как рекомендательный модуль, а не как единственный источник данных, что повышает контролируемость и надежность работы приложения.

Таким образом, в рамках первой главы необходимо определить постановку задачи, рассмотреть предметную область контроля питания,

провести анализ существующих аналогов, сформулировать требования к приложению и обосновать выбор методов и средств разработки.

1.1 Постановка задачи

Основной задачей выпускной квалификационной работы является разработка мобильного приложения для платформы Android, предназначенного для ведения электронного журнала питания и подбора блюд с использованием интеллектуального помощника.

Приложение должно предоставлять пользователю возможность регистрироваться и входить в систему. После первичной авторизации пользователь должен заполнить анкету, содержащую персональные данные, необходимые для расчета индивидуальной дневной нормы калорий. К таким данным относятся имя, возраст, пол, рост, вес, уровень физической активности, цель питания, пищевые предпочтения и аллергии. На основании введенных данных приложение должно рассчитать рекомендуемую дневную калорийность.

Основной пользовательский сценарий приложения связан с ведением дневника питания. Пользователь должен иметь возможность выбирать блюда из встроенной базы, указывать массу порции и тип приема пищи: завтрак, обед, ужин или перекус. После добавления блюда приложение должно автоматически рассчитывать калорийность, количество белков, жиров и углеводов с учетом указанной массы порции. В дневнике должны отображаться записи за выбранный день, а также суммарные показатели питания за сутки.

Дополнительно приложение должно предоставлять функцию генерации индивидуального рациона с использованием ИИ-диетолога. Рацион должен формироваться с учетом рассчитанной дневной нормы калорий, цели питания и указанных пользователем ограничений. При формировании рациона необходимо исключать продукты, содержащие аллергены пользователя. Сформированный рацион должен быть распределен по приемам пищи и

содержать информацию о массе продуктов, калорийности и количестве белков, жиров и углеводов.

Пользователь должен иметь возможность взаимодействовать с ИИ-диетологом в диалоговом режиме. В рамках такого взаимодействия пользователь может задавать вопросы о питании, просить заменить отдельный продукт, убрать нежелательное блюдо, добавить новый продукт или изменить калорийность рациона. Если запрос пользователя связан с изменением текущего рациона, приложение должно выполнить корректировку и отобразить обновленный вариант.

Сформированный ИИ-рацион должен иметь возможность сохранения в дневник питания. При сохранении продукты из рациона добавляются в соответствующие приемы пищи текущего дня. Для предотвращения повторного добавления приложение должно учитывать, был ли уже сохранен рацион за выбранную дату.

В результате разработки должно быть создано мобильное приложение, обеспечивающее выполнение следующих основных функций:

- регистрация и авторизация пользователей;
- заполнение и редактирование анкеты пользователя;
- расчет индивидуальной дневной нормы калорий;
- хранение данных пользователей, профилей, блюд, записей дневника, сообщений чата и сохраненных рационов в локальной базе данных;
- ведение дневника питания с возможностью добавления и удаления блюд;
- отображение суммарной калорийности и количества белков, жиров и углеводов за день;
- генерация индивидуального рациона с использованием ИИ-диетолога;
- фильтрация продуктов с учетом аллергий пользователя;
- корректировка рациона по запросу пользователя на естественном языке;

– сохранение сформированного рациона в дневник питания.

Разрабатываемое приложение должно функционировать на мобильных устройствах под управлением операционной системы Android. Для хранения данных должна использоваться локальная база данных, что позволит сохранять пользовательскую информацию на устройстве и обеспечивать быстрый доступ к дневнику питания. Для взаимодействия с ИИ-диетологом необходимо реализовать сетевое взаимодействие с внешним API.

1.2 Анализ предметной области

Предметной областью данной выпускной квалификационной работы является контроль питания и составление индивидуального рациона с использованием мобильного приложения и технологий искусственного интеллекта.

Контроль питания представляет собой процесс наблюдения за составом и энергетической ценностью рациона человека. Основными показателями, которые обычно учитываются при ведении дневника питания, являются калорийность, количество белков, жиров и углеводов. Калорийность характеризует энергетическую ценность пищи, а белки, жиры и углеводы отражают состав рациона и позволяют оценить его сбалансированность.

На практике контроль питания может выполняться различными способами. Наиболее простой вариант заключается в ручной записи съеденных продуктов и самостоятельном подсчете их энергетической ценности. Такой подход требует от пользователя постоянного обращения к таблицам калорийности, справочникам или онлайн-калькуляторам. Кроме того, ручной подсчет часто приводит к ошибкам, так как пользователь может неточно указать массу продукта, неправильно рассчитать пищевую ценность порции или не учесть отдельные ингредиенты блюда.

С развитием мобильных технологий широкое распространение получили приложения для ведения дневника питания. Они позволяют фиксировать приемы пищи, выбирать продукты из базы, указывать массу

порции и автоматически рассчитывать калорийность и количество макроэлементов. Такие приложения существенно упрощают ежедневный учет питания, однако во многих случаях пользователь все равно самостоятельно подбирает продукты и контролирует соответствие рациона своей цели.

Важной задачей в области контроля питания является персонализация. Суточная потребность в калориях зависит от пола, возраста, роста, массы тела и уровня физической активности человека. Кроме того, рацион должен учитывать цель пользователя: снижение массы тела, поддержание текущего веса или набор мышечной массы. Например, при снижении веса необходимо создать умеренный дефицит калорий, а при наборе мышечной массы – обеспечить повышенную энергетическую ценность рациона.

Отдельное значение имеют индивидуальные ограничения пользователя. К ним относятся пищевые аллергии, непереносимость отдельных продуктов и личные предпочтения. Наличие аллергии требует исключения из рациона продуктов, содержащих соответствующие аллергены. Поэтому приложение, предназначенное для подбора блюд, должно не только рассчитывать калорийность, но и проверять состав предлагаемых продуктов с учетом ограничений пользователя.

Использование искусственного интеллекта позволяет расширить возможности классического дневника питания. Интеллектуальный помощник может сформировать рацион на основе пользовательских данных, предложить распределение продуктов по приемам пищи, объяснить рекомендации и изменить меню по запросу пользователя. В отличие от шаблонных планов питания, диалоговое взаимодействие позволяет учитывать уточняющие пожелания: заменить продукт, убрать нежелательное блюдо, изменить калорийность или скорректировать отдельный прием пищи.

При этом использование внешнего ИИ – сервиса требует дополнительной программной обработки результатов. Ответ модели может содержать продукты, отсутствующие в локальной базе приложения, либо

блюда, которые не подходят пользователю из-за аллергий. Поэтому в разрабатываемом приложении ИИ-диетолог должен использоваться не как единственный источник данных, а как рекомендательный компонент. Итоговый рацион должен проверяться средствами приложения: продукты должны сопоставляться с локальной базой, аллергены должны исключаться, а масса порций и показатели КБЖУ должны рассчитываться программно.

Таким образом, предметная область разрабатываемого приложения включает несколько взаимосвязанных процессов: хранение пользовательских данных, расчет индивидуальной нормы калорий, ведение дневника питания, учет КБЖУ, фильтрацию продуктов по ограничениям и интеллектуальный подбор рациона. Совмещение этих процессов в одном мобильном приложении позволяет повысить удобство контроля питания и сделать рекомендации более персонализированными.

1.3 Анализ существующих аналогов

Для определения функциональных требований к разрабатываемому приложению необходимо рассмотреть существующие программные решения аналогичного назначения. В качестве аналогов были выбраны популярные приложения и сервисы, связанные с контролем питания, подсчетом калорий и получением рекомендаций: MyFitnessPal, YAZIO, FatSecret, а также сервис GigaChat как пример интеллектуального помощника, который может использоваться для консультаций по питанию.

MyFitnessPal является одним из наиболее известных приложений для ведения дневника питания и подсчета калорий. Приложение предоставляет пользователю возможность фиксировать приемы пищи, выбирать продукты из базы, указывать массу порции и отслеживать выполнение дневной нормы калорий. Также в приложении предусмотрен учет белков, жиров и углеводов [1].

К достоинствам MyFitnessPal можно отнести большую базу продуктов, наличие инструментов для отслеживания прогресса и удобный механизм

добавления записей в дневник. Приложение подходит пользователям, которые уже имеют представление о принципах подсчета калорий и готовы самостоятельно планировать рацион.

Недостатком MyFitnessPal является то, что основной сценарий работы все равно связан с ручным вводом данных. Пользователь самостоятельно выбирает продукты, рассчитывает порции и формирует рацион. Персональные рекомендации и дополнительные функции часто ограничены платной версией. Кроме того, приложение не ориентировано на полноценное диалоговое взаимодействие с интеллектуальным помощником, который мог бы корректировать рацион по естественному запросу пользователя.

Интерфейс приложения MyFitnessPal представлен на рисунке 1.

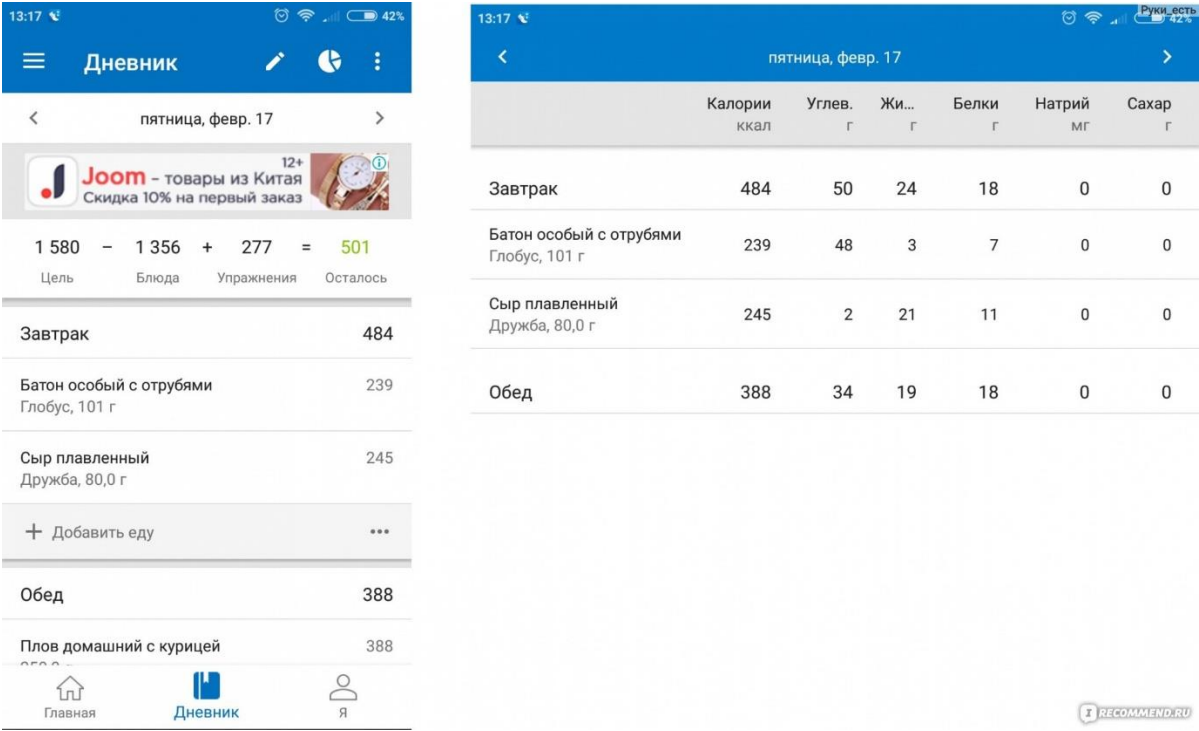


Рисунок 1 – Интерфейс дневника питания в MyFitnessPal

YAZIO представляет собой мобильное приложение для контроля питания, подсчета калорий и формирования планов питания. Приложение имеет современный интерфейс, позволяет задавать цель, отслеживать рацион и использовать готовые рецепты. Пользователь может выбирать предпочтения питания и получать рекомендации по приемам пищи [2].

Преимуществами YAZIO являются удобный интерфейс, наличие готовых планов питания и рецептов, а также возможность отслеживания основных показателей рациона. Приложение хорошо подходит для пользователей, которым необходим не только дневник питания, но и визуально понятный инструмент планирования.

К недостаткам YAZIO можно отнести ограниченность персонализации в бесплатной версии и отсутствие свободного диалогового режима с ИИ-диетологом. Пользователь получает рекомендации в рамках заранее подготовленных сценариев, но не может гибко изменять рацион с помощью естественного языка, например попросить заменить конкретный продукт или перестроить меню под новую калорийность.

Интерфейс приложения YAZIO представлен на рисунке 2.

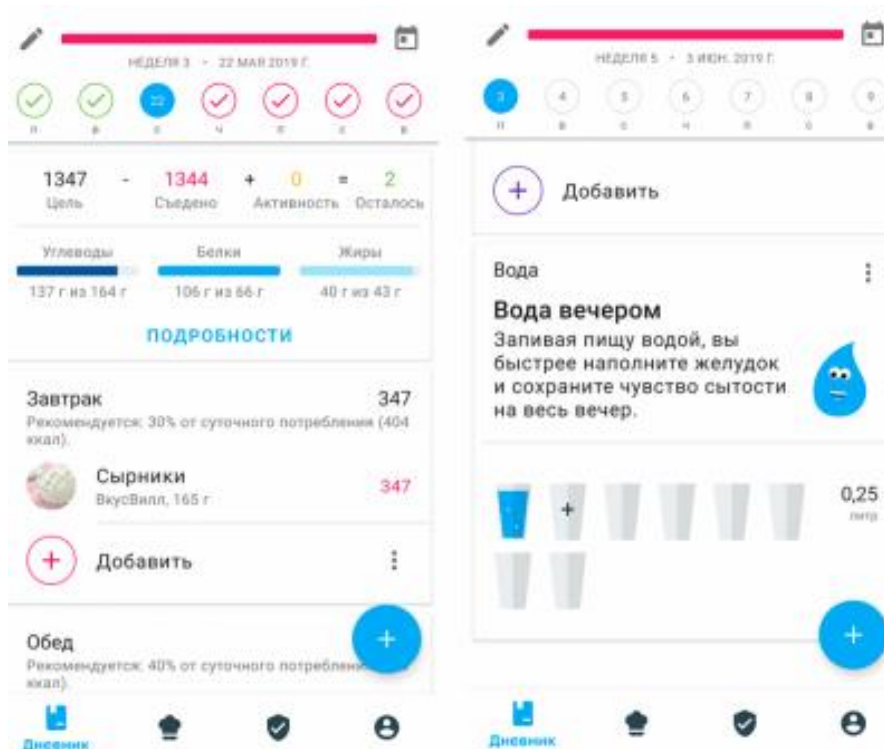


Рисунок 2 – Интерфейс YAZIO

FatSecret является сервисом для подсчета калорий и ведения дневника питания. Приложение содержит базу продуктов, поддерживает добавление приемов пищи и позволяет отслеживать калорийность рациона. Одним из

преимуществ сервиса является наличие русскоязычной базы продуктов, что делает его удобным для пользователей из России [3].

К достоинствам FatSecret можно отнести простоту использования, доступность основных функций и наличие базы продуктов на русском языке. Приложение позволяет быстро фиксировать съеденные продукты и контролировать показатели питания за день.

Недостатком FatSecret является отсутствие интеллектуального подбора рациона. Пользователь самостоятельно выбирает продукты и формирует меню. Приложение не предоставляет полноценного ИИ-помощника, способного составить рацион с учетом индивидуальных параметров, аллергий и цели питания, а затем изменить его по запросу пользователя.

Интерфейс приложения FatSecret представлен на рисунке 3.

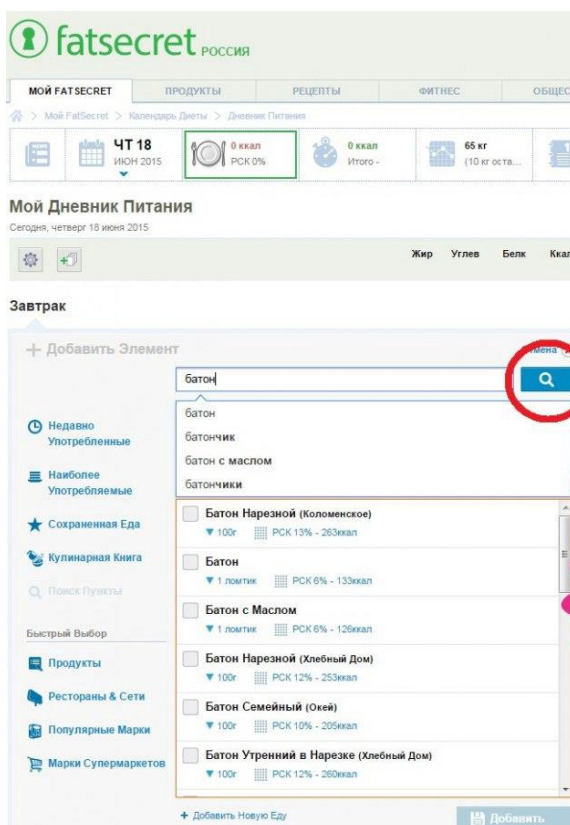


Рисунок 3 – Интерфейс поиска продуктов в FatSecret

GigaChat является сервисом на основе большой языковой модели, позволяющим пользователю вести диалог на естественном языке и получать ответы по различным темам, включая питание. Сервис может объяснять

общие принципы здорового питания, предлагать идеи блюд и отвечать на уточняющие вопросы [4].

Преимуществом GigaChat является возможность свободного диалогового взаимодействия. Пользователь может формулировать запросы в естественной форме, например попросить предложить рацион, заменить продукт или объяснить рекомендации. Такой формат общения удобен и понятен для пользователя.

Однако как самостоятельный сервис GigaChat не является специализированным дневником питания. Он не хранит пользовательский профиль в структуре мобильного приложения, не ведет дневник приемов пищи, не имеет встроенной локальной базы продуктов и не выполняет автоматический расчет КБЖУ по сохраненным записям. Кроме того, ответы модели требуют проверки и программной обработки, так как сгенерированный рацион может содержать продукты, отсутствующие в базе приложения, либо не учитывать все ограничения пользователя.

Пример диалога с GigaChat на тему питания представлен на рисунке 4.

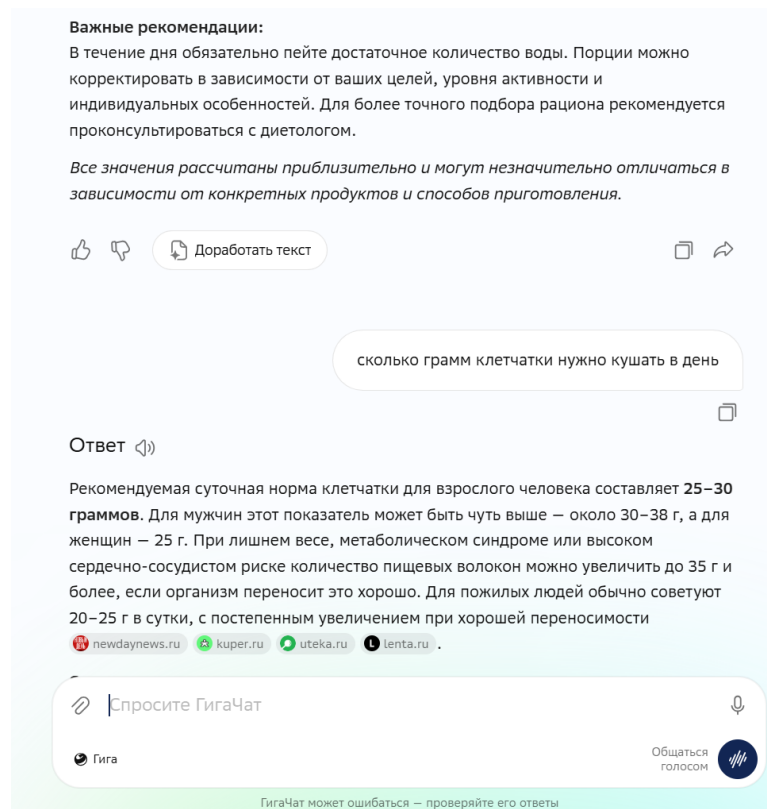


Рисунок 4 – Диалог с GigaChat на тему питания

Для сравнения рассмотренных решений была составлена таблица 1.

Таблица 1 – Сравнение приложений аналогичного назначения

Критерий	MyFitnessPal	Yazio	FatSecret	GigaChat	NutritionDiaryApp
Ручной ввод продуктов	+	+	+	–	+
База продуктов на русском	+	+	+	–	+ (встроенная)
Генерация ИИ-рациона	± (платно)	± (шаблоны)	–	+ (без привязки к КБЖУ)	+
Диалог с диетологом (чат)	–	–	–	+	+
Учет аллергий	+	+	+	–	+
Корректировка рациона по запросу	–	–	–	+ (вручную)	+
Хранение данных локально	+	+	+	–	+

Обозначения в таблице: «+» – функция реализована, «–» – функция отсутствует, «±» – функция реализована частично или имеет ограничения.

По результатам анализа можно сделать вывод, что существующие приложения хорошо решают задачу ручного ведения дневника питания и подсчета калорий, однако не обеспечивают полноценного сочетания локального дневника, персонализированного подбора рациона и диалогового взаимодействия с ИИ-диетологом. Отдельный ИИ-сервис, наоборот, предоставляет удобный диалоговый режим, но не является специализированным приложением для учета питания.

Разрабатываемое приложение должно объединить достоинства рассмотренных решений: ведение дневника питания, расчет калорий и БЖУ, хранение базы продуктов, персонализацию по данным пользователя и возможность формирования рациона с помощью ИИ. Дополнительным преимуществом является возможность сохранения сформированного ИИ-рациона в дневник и его корректировка по запросу пользователя на естественном языке.

1.4 Анализ требований к разрабатываемому приложению

На основе анализа предметной области, постановки задачи и рассмотрения существующих аналогов были сформулированы требования к разрабатываемому мобильному приложению. Требования можно разделить на функциональные, нефункциональные, требования к данным и требования к взаимодействию с внешним API.

Функциональные требования определяют основные действия, которые пользователь должен иметь возможность выполнять в приложении. Разрабатываемое приложение должно обеспечивать регистрацию и авторизацию пользователей. При регистрации пользователь указывает логин и пароль, после чего данные сохраняются в локальной базе данных. При последующем входе приложение должно проверять корректность введенных данных и предоставлять доступ только зарегистрированному пользователю.

После входа в приложение пользователь должен иметь возможность заполнить анкету. В анкете указываются имя, возраст, пол, рост, вес, уровень физической активности, цель питания, аллергии и пищевые предпочтения. На основе этих данных приложение должно рассчитывать индивидуальную дневную норму калорий. Расчет должен учитывать базовый обмен веществ, коэффициент физической активности и выбранную цель пользователя.

Приложение должно предоставлять возможность ведения дневника питания. Пользователь должен иметь возможность выбирать блюдо из локальной базы, указывать массу порции и тип приема пищи. После добавления записи приложение должно рассчитывать калорийность, количество белков, жиров и углеводов с учетом указанной массы. В дневнике должны отображаться все записи за выбранный день, а также суммарные показатели питания за сутки.

Для удобства пользователя приложение должно отображать прогресс выполнения дневной нормы калорий. Это позволит пользователю понимать, сколько калорий уже получено за день и насколько текущий рацион

соответствует рассчитанной норме. Также должны отображаться суммарные значения белков, жиров и углеводов.

Отдельным функциональным требованием является генерация индивидуального рациона с использованием ИИ-диетолога. Пользователь должен иметь возможность запросить рацион, сформированный с учетом его профиля, дневной нормы калорий, цели питания и аллергий. Полученный рацион должен быть распределен по приемам пищи и содержать сведения о продуктах, их массе и пищевой ценности.

Приложение должно поддерживать корректировку сформированного рациона в диалоговом режиме. Пользователь может ввести запрос на естественном языке, например попросить заменить продукт, убрать блюдо, добавить другой продукт или изменить калорийность рациона. Если запрос относится к изменению рациона, приложение должно обработать его и отобразить обновленный вариант. Если запрос является общим вопросом о питании, приложение должно предоставить ответ в формате консультации без изменения текущего рациона.

Сформированный ИИ-рацион должен иметь возможность сохранения в дневник питания. При сохранении продукты из рациона добавляются в соответствующие приемы пищи текущего дня. Для исключения дублирования приложение должно учитывать факт сохранения рациона за выбранную дату.

Для наглядного представления функциональных возможностей разрабатываемого приложения была построена диаграмма вариантов использования, представленная на рисунке В.1 приложения В.

К требованиям к данным относится хранение информации о пользователях, профилях, блюдах, записях дневника, сообщениях чата и сформированных рационах. Данные должны быть связаны с конкретным пользователем, чтобы обеспечить раздельное хранение информации при использовании приложения несколькими пользователями на одном устройстве. База продуктов должна содержать сведения о названии блюда,

калорийности, количестве белков, жиров и углеводов, категории, типе продукта и возможных аллергенах.

Нефункциональные требования определяют свойства приложения, не связанные напрямую с отдельными функциями. Пользовательский интерфейс должен быть понятным и удобным для ежедневного использования. Навигация между основными разделами приложения должна быть простой и очевидной. Основные действия пользователя, такие как добавление блюда, просмотр дневника, генерация рациона и переход к профилю, должны выполняться без лишних промежуточных шагов.

Приложение должно обеспечивать корректную обработку пользовательского ввода. Поля анкеты должны проходить валидацию: имя не должно быть пустым, возраст, рост и вес должны находиться в допустимых диапазонах, а обязательные параметры, такие как пол, активность и цель питания, должны быть выбраны. При некорректном вводе приложение должно отображать понятное сообщение об ошибке.

Для хранения данных должна использоваться локальная база данных. Это позволит сохранять информацию пользователя на устройстве и обеспечивать быстрый доступ к дневнику питания. При этом взаимодействие с ИИ-диетологом требует подключения к сети, так как генерация и корректировка рациона выполняются через внешний API.

Требования к взаимодействию с внешним API включают авторизацию, отправку запроса и обработку ответа. Приложение должно получать токен доступа, формировать запрос к GigaChat API, передавать в него сведения о пользователе и доступных продуктах, а затем обрабатывать полученный ответ. Ответ ИИ должен проходить дополнительную проверку: продукты должны сопоставляться с локальной базой, блюда с аллергенами должны исключаться, а итоговые значения калорийности и БЖУ должны рассчитываться средствами приложения.

Таким образом, разрабатываемое приложение должно обеспечивать полный цикл работы с дневником питания: от заполнения профиля и расчета

нормы калорий до ручного учета блюд, генерации индивидуального рациона, его корректировки и сохранения в дневник.

1.5 Выбор методов и средств разработки

Для реализации мобильного приложения необходимо определить методы разработки и выбрать средства, обеспечивающие создание Android-приложения, локальное хранение данных, построение пользовательского интерфейса, взаимодействие с внешним API и проверку корректности работы программных компонентов.

При выполнении работы использовался объектно-ориентированный подход. Функциональность приложения была разделена на отдельные компоненты, отвечающие за регистрацию и авторизацию пользователя, заполнение анкеты, расчет дневной нормы калорий, ведение дневника питания, работу с ИИ-диетологом, хранение данных и сетевое взаимодействие. Такой подход позволяет представить приложение как совокупность связанных программных объектов и упростить дальнейшее сопровождение проекта.

Для описания структуры и поведения приложения применялось UML-моделирование. Диаграмма вариантов использования использовалась для отображения основных пользовательских сценариев, а диаграмма классов – для представления ключевых программных компонентов и связей между ними. Это позволило заранее определить состав основных экранов, моделей данных, компонентов базы данных и сетевого слоя.

При проектировании локального хранилища использовался реляционный подход к организации данных. В структуре базы данных были выделены сущности пользователя, профиля пользователя, блюда, записи дневника питания, сообщения чата и сохраненного ИИ-рациона. Для каждой сущности были определены основные атрибуты и связи с другими сущностями.

Разработка приложения выполнялась модульно. Отдельные части приложения реализовывались как самостоятельные функциональные блоки: модуль авторизации, модуль анкеты пользователя, модуль дневника питания, модуль ИИ-диетолога, модуль локального хранения данных и модуль сетевого взаимодействия. Это позволило разделить ответственность между компонентами и упростить проверку отдельных частей приложения.

Для проверки корректности работы приложения использовались функциональное и модульное тестирование. Функциональное тестирование применялось для проверки пользовательского ввода и основных сценариев работы с анкетой профиля. Модульное тестирование использовалось для проверки отдельных программных компонентов, не зависящих от пользовательского интерфейса.

В качестве языка программирования был выбран Kotlin. Kotlin является современным языком разработки для платформы Android и поддерживается средой Android Studio. Он обладает лаконичным синтаксисом, средствами null-безопасности, поддержкой объектно-ориентированного и функционального программирования. Использование Kotlin позволяет сократить объем шаблонного кода и повысить читаемость программной реализации [5].

Средой разработки выбрана Android Studio. Данная среда является основным инструментом разработки Android-приложений, предоставляет средства создания пользовательского интерфейса, отладки, запуска приложения на эмуляторе и физических устройствах, а также интеграцию с системой сборки Gradle. Android Studio позволяет удобно управлять зависимостями проекта и выполнять модульное тестирование [6].

Для построения пользовательского интерфейса используется Jetpack Compose. Это декларативный фреймворк для разработки интерфейсов Android-приложений [7]. В отличие от классического подхода с XML-разметкой, Jetpack Compose позволяет описывать интерфейс непосредственно на языке Kotlin. Такой подход упрощает разработку экранов, ускоряет

внесение изменений и хорошо сочетается с архитектурой приложения, основанной на состоянии.

Для локального хранения данных выбрана библиотека Room, являющаяся надстройкой над SQLite [8][9]. Room предоставляет удобный способ описания сущностей базы данных, объектов доступа к данным и запросов. Использование Room позволяет хранить данные пользователей, профили, блюда, записи дневника питания, историю сообщений и сформированные рационы. Кроме того, Room обеспечивает проверку SQL-запросов на этапе компиляции, что повышает надежность работы с данными.

Для организации сетевого взаимодействия используется библиотека Retrofit [10]. Она позволяет описывать HTTP-запросы в виде интерфейсов Kotlin и обрабатывать ответы внешнего API. В разрабатываемом приложении Retrofit применяется для взаимодействия с GigaChat API. Для настройки сетевого клиента и обработки HTTP-запросов используется OkHttp. Данная библиотека позволяет добавлять заголовки авторизации, настраивать параметры запросов и выполнять отладку сетевого взаимодействия [11].

Для преобразования данных между объектами Kotlin и форматом JSON используется Gson. Эта библиотека необходима для формирования тела запросов к API и обработки ответов, полученных от сервиса искусственного интеллекта [12]. Использование Gson упрощает извлечение сгенерированного текста и дальнейшую обработку результата.

В качестве внешнего интеллектуального сервиса используется GigaChat API. Данный сервис применяется для генерации индивидуального рациона и ответов на вопросы пользователя, связанные с питанием. Взаимодействие с GigaChat API осуществляется через HTTP-запросы с использованием авторизации. При этом итоговая обработка рациона выполняется на стороне приложения: продукты сопоставляются с локальной базой, исключаются аллергены, рассчитываются граммовки и показатели КБЖУ.

В качестве архитектурного подхода используется MVVM. Данный подход предполагает разделение приложения на модель данных, слой

представления и ViewModel. Модель отвечает за данные и бизнес-логику, представление отображает интерфейс пользователя, а ViewModel связывает данные с интерфейсом и управляет состоянием экранов. Использование MVVM позволяет разделить логику приложения от пользовательского интерфейса и упростить сопровождение проекта.

Для асинхронного выполнения операций используются корутины Kotlin. Они позволяют выполнять обращение к базе данных и внешнему API без блокировки основного потока приложения.

Зависимости проекта подключаются и настраиваются с использованием Gradle [13]. Система сборки Gradle применяется для подключения библиотек, настройки параметров Android-проекта, сборки приложения и запуска тестов.

Для тестирования отдельных компонентов приложения используется JUnit 4 [14]. С его помощью проверяется корректность работы функций валидации пользовательских данных и отдельных алгоритмов приложения. Модульное тестирование позволяет убедиться в корректности работы логики приложения независимо от пользовательского интерфейса.

Совместное использование выбранных технологий обеспечивает единый цикл разработки: от проектирования интерфейса и хранения данных до сетевого обмена, проверки логики и сборки готового Android-приложения.

Таким образом, выбранные методы и средства разработки позволяют реализовать мобильное приложение с современным пользовательским интерфейсом, локальным хранением данных, сетевым взаимодействием с внешним API, разделением ответственности между компонентами и возможностью модульного тестирования.

2 Проектирование приложения

После анализа предметной области, постановки задачи и выбора средств разработки необходимо выполнить проектирование приложения. На данном этапе определяется общая архитектура программного продукта, структура локальной базы данных, основные компоненты приложения, пользовательские сценарии и принципы взаимодействия между слоями системы.

Проектирование позволяет заранее определить состав модулей приложения, распределить ответственность между ними и снизить сложность дальнейшей реализации. Для мобильного приложения дневника питания особенно важно обеспечить разделение пользовательского интерфейса, бизнес-логики, работы с локальной базой данных и сетевого взаимодействия с внешним API.

В рамках данной главы рассматриваются архитектура приложения, проектирование структуры данных, пользовательского интерфейса и основных классов, обеспечивающих работу электронного журнала питания и ИИ-диетолога.

2.1 Архитектура приложения

Разрабатываемое приложение представляет собой мобильное Android-приложение, работающее на устройстве пользователя и использующее локальную базу данных для хранения основной информации. Внешний сервис GigaChat API используется только для функций, связанных с генерацией и корректировкой рациона, а также для ответов ИИ-диетолога в диалоговом режиме.

Для приложения выбран архитектурный подход MVVM. Данный подход позволяет разделить программный продукт на несколько логических частей: пользовательский интерфейс, слой представления, слой бизнес-логики и слой данных. Такое разделение повышает сопровождаемость

проекта и упрощает внесение изменений, так как изменение одного слоя не требует полной переработки остальных частей приложения.

Общая архитектура приложения включает следующие основные слои:

- слой пользовательского интерфейса;
- слой представления;
- слой предметной логики;
- слой данных;
- слой сетевого взаимодействия.

Общая схема архитектуры разрабатываемого приложения представлена на рисунке В.2 приложения В.

Слой пользовательского интерфейса реализован с использованием Jetpack Compose. Он содержит экраны регистрации, авторизации, анкеты пользователя, профиля, дневника питания и ИИ-диетолога. Задача данного слоя заключается в отображении состояния приложения и обработке действий пользователя: нажатий кнопок, ввода текста, выбора блюд, отправки сообщений и переходов между экранами.

Слой представления представлен ViewModel-классами. ViewModel отвечает за хранение состояния экранов, обработку пользовательских действий и передачу данных между интерфейсом и другими слоями приложения. Например, ViewModel дневника питания получает записи за выбранный день, рассчитывает суммарную калорийность и передает данные на экран. ViewModel ИИ-диетолога отвечает за отправку сообщений, получение ответа от внешнего API, обработку текущего рациона и сохранение результата в дневник.

Слой предметной логики содержит алгоритмы, не зависящие напрямую от пользовательского интерфейса. К нему относятся расчет базового обмена веществ, определение дневной нормы калорий, валидация данных анкеты, проверка ограничений пользователя, расчет пищевой ценности порции и обработка сформированного рациона. Выделение данной логики в отдельные классы упрощает тестирование и повторное использование кода.

Слой данных отвечает за локальное хранение информации. Для этого используется база данных Room. В базе данных хранятся пользователи, профили, блюда, записи дневника питания, сообщения чата и сохраненные ИИ-рационы. Доступ к данным осуществляется через DAO-интерфейсы, которые содержат методы для добавления, изменения, удаления и получения записей.

Для объединения источников данных используется репозиторий. Репозиторий скрывает детали работы с базой данных и внешним API от остальных частей приложения. Благодаря этому ViewModel не обращается напрямую к DAO или сетевому клиенту, а использует методы репозитория. Такой подход упрощает архитектуру и делает код более гибким.

Слой сетевого взаимодействия используется для обращения к GigaChat API. Он включает Retrofit-интерфейс для описания HTTP-запросов, OkHttp-клиент для настройки сетевого взаимодействия и классы авторизации для получения токена доступа. Ответы внешнего API обрабатываются приложением: из них извлекается текст, после чего выполняется проверка продуктов, фильтрация аллергенов и расчет итоговых показателей рациона.

Физически приложение может быть представлено как набор взаимосвязанных компонентов. Пользователь взаимодействует с экранами Jetpack Compose. Экраны обращаются к ViewModel, ViewModel вызывает методы репозитория, репозиторий получает данные из локальной базы Room или обращается к GigaChat API. После получения результата данные возвращаются обратно во ViewModel и отображаются в пользовательском интерфейсе.

Такой архитектурный подход позволяет обеспечить разделение ответственности между компонентами приложения. Пользовательский интерфейс отвечает только за отображение данных, ViewModel управляет состоянием экранов, слой данных обеспечивает хранение информации, а сетевой слой отвечает за взаимодействие с внешним сервисом. Это делает

приложение более структурированным, удобным для сопровождения и расширения.

2.2 Проектирование структуры данных

Для хранения данных разрабатываемого приложения используется локальная реляционная база данных Room, основанная на SQLite [15]. Использование локальной базы данных позволяет сохранять информацию непосредственно на устройстве пользователя и обеспечивать быстрый доступ к данным дневника питания, профиля и истории взаимодействия с ИИ-диетологом.

При проектировании структуры данных необходимо определить основные сущности предметной области, их атрибуты и связи между ними. Поскольку на данном этапе выполняется концептуальное проектирование базы данных, модель должна отражать не физическую структуру таблиц, а основные информационные объекты приложения и их взаимосвязи. Поэтому в концептуальной модели используются предметные названия сущностей, а технические идентификаторы и внешние ключи не отображаются.

В разрабатываемом приложении выделяются следующие основные сущности:

- пользователь;
- профиль пользователя;
- блюдо;
- запись дневника питания;
- сообщение чата;
- ИИ-рацион.

Сущность «Пользователь» предназначена для описания учетной записи пользователя приложения. Она содержит логин и пароль, необходимые для регистрации и последующей авторизации. Логин пользователя является уникальным, так как по нему приложение определяет конкретную учетную запись и отделяет данные одного пользователя от данных другого.

Сущность «Профиль пользователя» содержит персональные данные, необходимые для расчета индивидуальной дневной нормы калорий и формирования рекомендаций по питанию. К таким данным относятся имя, возраст, пол, рост, вес, уровень физической активности, цель питания, аллергии, пищевые предпочтения и рассчитанная дневная норма калорий. Профиль связан с пользователем связью один к одному, так как для каждой учетной записи должен храниться один набор персональных параметров.

Сущность «Блюдо» описывает продукт или готовое блюдо, доступное в локальной базе приложения. Для блюда указываются название, калорийность на 100 граммов, количество белков, жиров и углеводов на 100 граммов, категория, тип приема пищи, сведения об аллергенах и признак возможности использования при формировании ИИ-рациона. Данная сущность используется как при ручном добавлении записей в дневник, так и при проверке и обработке рациона, полученного от ИИ-диетолога.

Сущность «Запись дневника питания» предназначена для описания факта добавления блюда в дневник питания пользователя. Она содержит дату, тип приема пищи и массу порции. Связь с пользователем показывает, кому принадлежит запись дневника, а связь с блюдом определяет, какой продукт или блюдо было добавлено. На основе данных о блюде и массе порции приложение рассчитывает калорийность, количество белков, жиров и углеводов для конкретной записи.

Сущность «Сообщение чата» используется для хранения истории взаимодействия пользователя с ИИ-диетологом. В ней фиксируются текст сообщения, отправитель и время сообщения. Наличие данной сущности позволяет отображать историю диалога и отделять сообщения пользователя от ответов интеллектуального помощника.

Сущность «ИИ-рацион» предназначена для хранения рациона питания, сформированного интеллектуальным помощником. Она содержит дату формирования, содержимое рациона, признак сохранения в дневник и время создания. Данная сущность необходима для отображения ранее

сформированного рациона и предотвращения повторного добавления одного и того же ИИ-рациона в дневник питания.

Описание основных сущностей концептуальной модели базы данных представлено в таблице 2.

Таблица 2 – Описание сущностей концептуальной модели базы данных

Название сущности	Основные атрибуты	Описание сущности
Пользователь	Логин; пароль	Содержит данные учетной записи пользователя, необходимые для регистрации и авторизации
Профиль пользователя	Имя; возраст; пол; рост; вес; уровень активности; цель питания; аллергии; предпочтения; дневная норма калорий	Содержит персональные данные пользователя для расчета нормы калорий и формирования рекомендаций
Блюдо	Название блюда; калорийность на 100 г; белки на 100 г; жиры на 100 г; углеводы на 100 г; категория; тип приема пищи; аллергены; используется ИИ	Содержит сведения о продуктах и блюдах, доступных в приложении
Запись дневника питания	Дата; тип приема пищи; масса порции	Содержит информацию о блюде, добавленном пользователем в дневник питания за определенную дату
Сообщение чата	Текст сообщения; отправитель; время сообщения	Содержит историю сообщений пользователя и ИИ-диетолога
ИИ-рацион	Дата формирования; содержимое рациона; признак сохранения в дневник; время создания	Содержит сформированный ИИ-рацион и сведения о его сохранении в дневник питания

Между сущностями концептуальной модели существуют следующие связи. Один пользователь имеет один профиль пользователя. Один пользователь может иметь множество записей дневника питания, множество сообщений чата и множество сформированных ИИ-рационов. Каждая запись дневника питания относится к одному пользователю и связана с одним блюдом. Одно блюдо может использоваться во множестве записей дневника питания.

Концептуальная модель базы данных разрабатываемого приложения представлена на рисунке 5.

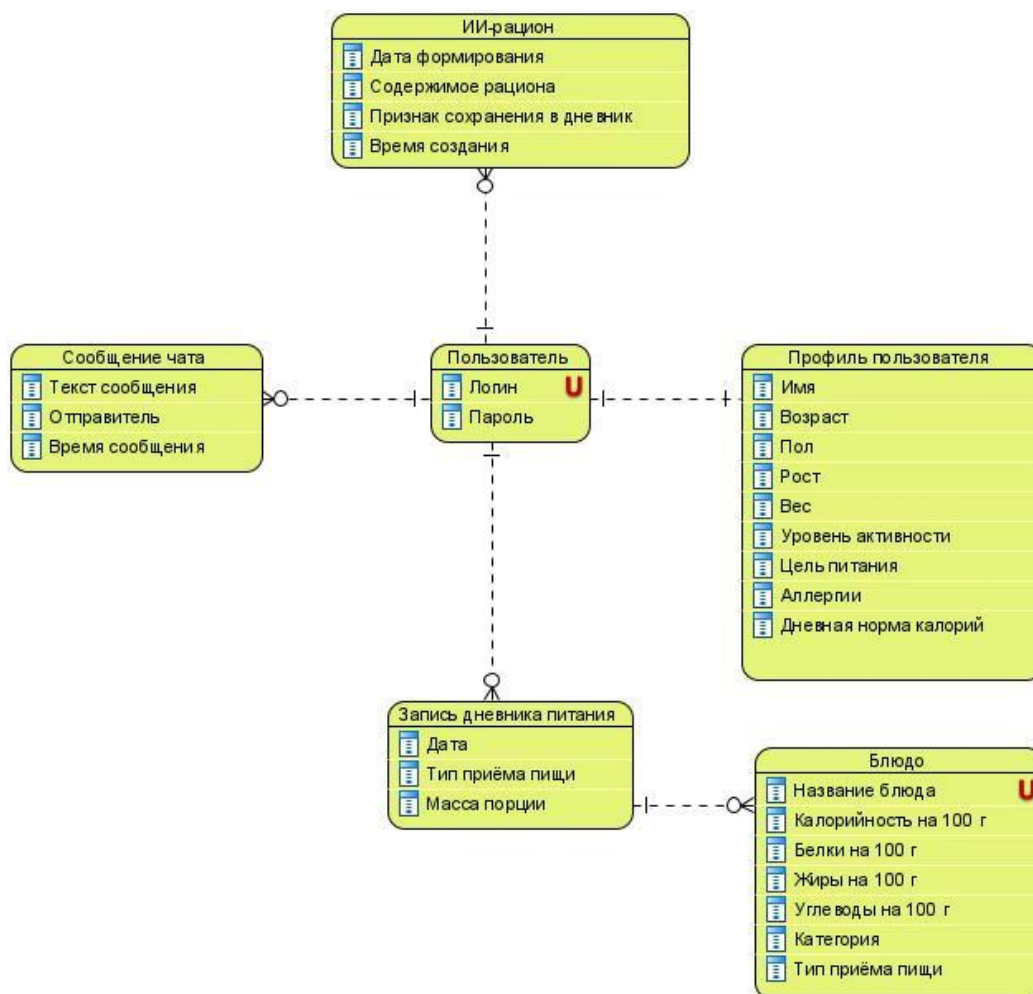


Рисунок 5 – Концептуальная модель базы данных

Диаграмма построена в нотации «сущность-связь» и отражает основные информационные объекты приложения без привязки к физической реализации базы данных. Центральной сущностью является пользователь, с которым связаны профиль, записи дневника питания, сообщения чата и сформированные ИИ-рационы. Запись дневника питания дополнительно связана с сущностью «Блюдо», что позволяет определить, какой продукт был добавлен пользователем, и рассчитать показатели КБЖУ для указанной массы порции. Атрибуты сущностей локальной базы данных приведены в приложении Б.

Таким образом, концептуальная модель базы данных отражает структуру предметной области приложения и показывает, какие данные

необходимы для работы электронного журнала питания, профиля пользователя, дневника, истории общения с ИИ-диетологом и сформированных рационов.

2.3 Проектирование пользовательского интерфейса

Пользовательский интерфейс разрабатываемого мобильного приложения должен обеспечивать быстрый доступ к основным функциям электронного журнала питания: заполнению профиля, просмотру дневника питания, добавлению блюд, контролю калорийности и взаимодействию с ИИ-диетологом. Так как приложение предназначено для ежедневного использования, при проектировании интерфейса необходимо обеспечить простую навигацию, минимальное количество действий для выполнения основных операций и понятное отображение результатов расчета калорий, белков, жиров и углеводов.

Интерфейс приложения проектируется с учетом особенностей мобильных устройств под управлением Android. Основное взаимодействие пользователя с приложением выполняется через вертикально ориентированные экраны. Для экранов, содержащих большое количество данных, предусматривается вертикальная прокрутка. Такой подход применяется на экране анкеты пользователя, в профиле, дневнике питания и на экране ИИ-диетолога.

Для построения пользовательского интерфейса используется декларативный подход Jetpack Compose. Каждый экран приложения представлен отдельным composable-компонентом, отвечающим за отображение определенного пользовательского сценария. Основными экранами приложения являются экран входа, экран регистрации, экран анкеты, экран профиля, экран дневника питания, экран добавления блюда и экран ИИ-диетолога.

Навигация в приложении строится поэтапно. При первом запуске пользователь попадает на экран входа. Если учетная запись отсутствует, он

может перейти на экран регистрации. После успешной регистрации или авторизации приложение проверяет наличие сохраненного профиля пользователя. Если профиль отсутствует, открывается экран анкеты, где пользователь указывает персональные параметры, необходимые для расчета дневной нормы калорий. После заполнения анкеты открывается основной интерфейс приложения.

Основной интерфейс приложения содержит нижнюю навигационную панель с тремя разделами: «Профиль», «Дневник» и «ИИ-диетолог». Такой вариант навигации выбран потому, что эти разделы являются основными и должны быть доступны пользователю без перехода через дополнительные меню. Раздел «Профиль» используется для просмотра персональных данных и дневной нормы калорий, раздел «Дневник» – для ручного учета питания, а раздел «ИИ-диетолог» – для генерации и корректировки индивидуального рациона.

Схема переходов между основными экранами приложения приведена на рисунке В.3 приложения В.

Экран входа предназначен для авторизации ранее зарегистрированного пользователя. На экране располагаются поля ввода логина и пароля, кнопка входа, ссылка для перехода к регистрации и область отображения сообщения об ошибке. Поле пароля предусматривает возможность скрывать и отображать введенные символы. Кнопка входа становится доступной только после заполнения обязательных полей. При вводе неверных данных приложение должно вывести пользователю понятное сообщение об ошибке.

Экран регистрации предназначен для создания новой локальной учетной записи. На нем размещаются поля для ввода логина, пароля и подтверждения пароля. Для повышения удобства пользователя на экране предусмотрена проверка совпадения пароля и его подтверждения. Если пароли не совпадают, приложение выводит сообщение об ошибке. Также предусматривается обработка ситуации, когда пользователь пытается зарегистрировать логин, уже существующий в локальной базе данных. После

успешной регистрации пользователь автоматически переходит к следующему этапу работы с приложением.

Экран анкеты пользователя является одним из ключевых экранов приложения, так как от введенных на нем данных зависит расчет индивидуальной дневной нормы калорий и дальнейшая персонализация рекомендаций. На экране анкеты пользователь вводит имя, возраст, рост и вес, выбирает пол, уровень физической активности, цель питания и отмечает имеющиеся аллергии. Для выбора пола, активности и цели используются переключатели, а для выбора аллергий – флажки. Такой способ ввода снижает вероятность ошибок и делает форму более понятной.

При проектировании анкеты необходимо учитывать, что часть данных вводится вручную, поэтому требуется проверка корректности пользовательского ввода. Имя не должно быть пустым, возраст, рост и вес должны находиться в допустимых диапазонах, а пол, уровень активности и цель питания должны быть выбраны обязательно. При обнаружении ошибки приложение должно отобразить сообщение рядом с формой, не завершая сохранение профиля. После корректного заполнения анкеты выполняется расчет дневной нормы калорий, а профиль пользователя сохраняется в локальную базу данных.

Экран профиля предназначен для отображения персональных данных пользователя и рассчитанной дневной нормы калорий. Информация на экране группируется в отдельные карточки: основные сведения о пользователе, параметры активности и цели, дневная норма калорий и список аллергий. Такое деление делает экран более читаемым и позволяет пользователю быстро найти нужную информацию. В верхней части экрана предусматривается кнопка редактирования профиля, при нажатии на которую пользователь возвращается к экрану анкеты с уже заполненными данными. Также на экране профиля предусматривается кнопка выхода из учетной записи.

Экран дневника питания предназначен для ежедневного контроля рациона. В верхней части экрана отображается текущая дата и карточка прогресса по калориям. В данной карточке показываются текущая сумма потребленных калорий, целевая дневная норма, оставшееся количество калорий или превышение нормы, а также суммарные значения белков, жиров и углеводов. Для наглядного отображения прогресса используется горизонтальный индикатор. Если пользователь превысил дневную норму, интерфейс должен визуально выделять это состояние как предупреждение.

Ниже карточки прогресса размещается список добавленных за день блюд. Каждая запись дневника должна содержать название блюда, тип приема пищи, массу порции и рассчитанную калорийность. Для каждой записи предусматривается кнопка удаления. При удалении записи приложение пересчитывает суммарные показатели за день. Если за текущий день нет добавленных блюд, приложение должно вывести сообщение с подсказкой, что пользователь может добавить блюдо вручную или сформировать рацион с помощью ИИ-диетолога.

Добавление блюда в дневник выполняется через отдельный экран или модальное представление, вызываемое с экрана дневника питания. Для быстрого доступа к данной функции в разделе дневника используется плавающая кнопка добавления. На экране добавления блюда пользователь сначала выбирает продукт или блюдо из локальной базы. Для удобства предусматривается строка поиска по названию. После выбора блюда пользователь указывает массу порции в граммах и выбирает тип приема пищи: завтрак, обед, ужин или перекус. Кнопка добавления становится доступной только при корректном значении массы порции.

Экран ИИ-диетолога предназначен для генерации персонального рациона и диалогового взаимодействия пользователя с интеллектуальным помощником. Интерфейс данного экрана проектируется в формате чата, так как пользователь может задавать вопросы и отправлять команды на естественном языке. В верхней части экрана располагается заголовок «ИИ-

диетолог» и действие для очистки истории сообщений. Основная область экрана содержит список сообщений пользователя и ответов ИИ. В нижней части располагается поле ввода текста и кнопка отправки сообщения.

Если рацион еще не сформирован, на экране ИИ-диетолога отображается кнопка составления рациона на рассчитанную дневную норму калорий. После нажатия на эту кнопку приложение формирует запрос к интеллектуальному помощнику и отображает результат в виде карточки рациона. Карточка рациона должна содержать список блюд, распределенных по приемам пищи: завтрак, обед, ужин и перекусы. Для каждого элемента рациона отображается название продукта, количество граммов и калорийность. Также в карточке выводятся итоговые значения калорийности, белков, жиров и углеводов.

Для сформированного ИИ-рациона предусматриваются две основные кнопки: «В дневник» и «Другой». Кнопка «В дневник» сохраняет предложенный рацион в дневник питания за текущий день. Кнопка «Другой» позволяет сформировать новый вариант рациона. Дополнительно пользователь может вводить текстовые команды, например попросить заменить продукт, убрать блюдо, добавить другой продукт или изменить калорийность. Такой способ взаимодействия делает интерфейс более гибким и позволяет корректировать рацион без ручного пересоставления всего меню.

Состав экранов пользовательского интерфейса представлен в таблице 3.

Таблица 3 – Состав экранов пользовательского интерфейса

Экран	Назначение	Основные элементы	Основные действия пользователя
Экран входа	Авторизация пользователя	Поле логина, поле пароля, кнопка входа, ссылка на регистрацию, сообщение об ошибке	Ввести логин и пароль, выполнить вход, перейти к регистрации
Экран регистрации	Создание новой учетной записи	Поле логина, поле пароля, подтверждение пароля, кнопка регистрации, ссылка на вход	Создать учетную запись, вернуться на экран входа
Экран анкеты	Ввод персональных параметров пользователя	Поля имени, возраста, роста, веса, выбор пола, активности, цели и аллергий	Заполнить анкету, рассчитать дневную норму калорий, сохранить профиль

Продолжение таблицы 3

Экран	Назначение	Основные элементы	Основные действия пользователя
Экран профиля	Просмотр данных пользователя	Карточки персональных данных, дневная норма калорий, аллергии, кнопки редактирования и выхода	Просмотреть профиль, перейти к редактированию, выйти из учетной записи
Экран дневника питания	Контроль питания за день	Карточка калорий, показатели БЖУ, список записей, кнопка добавления блюда	Просмотреть дневной рацион, удалить запись, перейти к добавлению блюда
Экран добавления блюда	Добавление продукта в дневник	Поиск блюда, список продуктов, поле массы порции, выбор приема пищи, кнопка добавления	Найти блюдо, указать граммы, выбрать прием пищи, добавить запись
Экран ИИ-диетолога	Генерация и корректировка рациона	Список сообщений, поле ввода, кнопка отправки, кнопка генерации рациона, карточка рациона	Сформировать рацион, задать вопрос, изменить рацион, сохранить рацион в дневник

При проектировании интерфейса также необходимо учитывать обработку пустых состояний и ошибок. Если дневник за текущий день пуст, пользователь должен видеть не пустой экран, а подсказку о возможных действиях. Если данные введены некорректно, сообщение об ошибке должно быть сформулировано понятным языком. Если выполняется длительная операция, например загрузка профиля или обращение к ИИ-диетологу, интерфейс должен отображать состояние ожидания и не создавать впечатление зависания приложения.

Важным требованием является сохранение логической связи между разделами приложения. Данные, введенные на экране анкеты, должны отображаться на экране профиля и использоваться на экране ИИ-диетолога. Блюда, добавленные вручную или сохраненные из ИИ-рациона, должны отображаться в дневнике питания. При этом пользователь не должен повторно вводить одни и те же данные: профиль сохраняется в локальной базе данных и используется при следующих запусках приложения.

Таким образом, спроектированный пользовательский интерфейс обеспечивает выполнение всех основных сценариев работы с приложением: регистрацию и вход пользователя, заполнение анкеты, просмотр профиля, ведение дневника питания, добавление блюд, генерацию персонального

рациона и сохранение ИИ-рациона в дневник. Использование нижней навигации, карточек, форм с проверкой ввода и диалогового интерфейса ИИ-диетолога делает приложение удобным для повседневного использования и соответствует назначению электронного журнала питания.

2.4 Проектирование алгоритмов приложения

Разрабатываемое мобильное приложение выполняет не только хранение пользовательских данных, но и ряд вычислительных и логических операций, связанных с персонализацией рациона. К таким операциям относятся расчет индивидуальной дневной нормы калорий, вычисление пищевой ценности порции, формирование рациона с использованием ИИ-диетолога, корректировка рациона по текстовому запросу пользователя и сохранение сформированного рациона в дневник питания. Для обеспечения корректной работы приложения данные алгоритмы должны быть спроектированы до этапа реализации.

Основными входными данными для алгоритмов приложения являются параметры пользователя, введенные при заполнении анкеты: пол, возраст, рост, вес, уровень физической активности, цель питания и список аллергий. Эти данные используются при расчете дневной нормы калорий и при формировании персонального рациона. Дополнительно приложение использует локальную базу блюд, в которой для каждого продукта или блюда указаны калорийность, белки, жиры и углеводы на 100 граммов.

Первым важным алгоритмом является алгоритм расчета индивидуальной дневной нормы калорий. Он выполняется после заполнения пользователем анкеты. Расчет базируется на формуле Миффлина – Сан Жеора, позволяющей определить базовый обмен веществ с учетом пола, возраста, роста и веса пользователя.

Для мужчин базовый обмен веществ рассчитывается по формуле (1), для женщин используется формула (2):

$$BMR_M = 10 \times W + 6,25 \times H - 5 \times A + 5, \quad (1)$$

где BMR_m – базовый обмен веществ для мужчин, ккал;

W – масса пользователя, кг;

H – рост пользователя, см;

A – возраст пользователя, лет;

$$BMR_{\text{ж}} = 10 \times W + 6,25 \times H - 5 \times A - 161, \quad (2)$$

где $BMR_{\text{ж}}$ – базовый обмен веществ для женщин, ккал;

После вычисления базового обмена веществ результат умножается на коэффициент физической активности. Чем выше уровень активности пользователя, тем больше итоговая дневная норма калорий. Затем результат корректируется в зависимости от выбранной цели питания. При цели снижения веса из рассчитанного значения вычитается 500 ккал, при поддержании веса значение остается без изменений, а при наборе мышечной массы к значению добавляется 300 ккал. Для предотвращения некорректных результатов итоговая калорийность ограничивается допустимым диапазоном.

Алгоритм расчета дневной нормы калорий можно представить следующей последовательностью действий. Сначала приложение получает данные профиля пользователя. Затем проверяется пол пользователя и выбирается соответствующая формула расчета BMR. После этого базовый обмен веществ умножается на коэффициент активности. Далее приложение проверяет выбранную цель питания и изменяет результат в соответствии с этой целью. Полученное значение округляется и сохраняется в профиле пользователя как целевая дневная калорийность. Схемы алгоритмов разработаны с учетом правил выполнения схем программ и данных [16].

Схема алгоритма расчета индивидуальной дневной нормы калорий приведена на рисунке В.4 приложения В.

Следующим важным алгоритмом является алгоритм добавления блюда в дневник питания. Пользователь выбирает блюдо из локальной базы данных, указывает массу порции в граммах и выбирает тип приема пищи. После этого приложение рассчитывает фактическую пищевую ценность указанной порции. Так как значения калорийности, белков, жиров и углеводов хранятся

в базе данных на 100 граммов продукта, пересчет выполняется по формуле (3):

$$\text{значение порции} = \text{значение на 100 г} \times \text{масса порции} / 100, \quad (3)$$

Данная формула применяется отдельно для калорий, белков, жиров и углеводов. После расчета запись сохраняется в дневник питания с указанием пользователя, выбранного блюда, массы порции, типа приема пищи и даты. Затем приложение обновляет список записей за текущий день и пересчитывает суммарные показатели КБЖУ.

Алгоритм добавления блюда в дневник начинается с открытия экрана добавления блюда. Пользователь выбирает продукт из списка или находит его с помощью поиска. Далее вводится масса порции и выбирается тип приема пищи. Если масса указана некорректно, например равна нулю или не является числом, приложение не выполняет сохранение и выводит сообщение об ошибке. Если данные корректны, выполняется пересчет пищевой ценности порции, после чего запись добавляется в локальную базу данных.

Схема алгоритма добавления блюда в дневник питания приведена на рисунке В.5 приложения В.

Наиболее сложным алгоритмом приложения является алгоритм формирования рациона с использованием ИИ-диетолога. Его назначение состоит в том, чтобы на основе профиля пользователя, целевой калорийности и локальной базы продуктов сформировать примерный рацион на день. При этом приложение не передает внешней модели произвольную задачу без ограничений. Перед обращением к ИИ выполняется подготовка контролируемого набора продуктов, которые могут быть использованы при генерации.

Сначала приложение получает профиль текущего пользователя и проверяет наличие рассчитанной дневной нормы калорий. Затем из локальной базы данных выбираются продукты и блюда, разрешенные для использования в ИИ-генерации. После этого выполняется фильтрация продуктов по аллергиям пользователя. Например, если пользователь указал

аллергию на молочные продукты, из списка исключаются блюда, содержащие молоко или связанные с ним ингредиенты. Такая фильтрация выполняется до формирования запроса к ИИ, чтобы снизить вероятность появления неподходящих продуктов в рационе.

После подготовки списка продуктов приложение формирует текстовый запрос к GigaChat API. В запрос включаются параметры пользователя, его цель питания, дневная норма калорий, список ограничений и перечень доступных продуктов. ИИ должен вернуть рацион, распределенный по основным приемам пищи: завтрак, обед, ужин и перекусы. Полученный ответ не используется приложением напрямую без проверки. После получения ответа приложение анализирует результат, сопоставляет названия продуктов с локальной базой данных и исключает продукты, которых нет в базе или которые противоречат ограничениям пользователя.

После проверки состава рациона приложение выполняет расчет граммовок и итоговых показателей КБЖУ. Калорийность распределяется между приемами пищи неравномерно: завтрак, обед, ужин и перекусы получают разные доли от общей дневной нормы. Такой подход позволяет сформировать более реалистичный рацион, в котором основная часть калорий приходится на основные приемы пищи. Если итоговая калорийность сильно отличается от целевой, приложение корректирует порции продуктов, уменьшая или увеличивая их в допустимых пределах.

Алгоритм формирования ИИ-рациона включает следующие этапы: получение профиля пользователя, загрузку доступных продуктов, фильтрацию аллергенов, формирование запроса к ИИ, получение ответа, проверку продуктов по локальной базе данных, повторную фильтрацию ограничений, расчет граммовок, пересчет КБЖУ и отображение результата пользователю. Такой подход позволяет использовать ИИ как рекомендательный модуль, но сохранять контроль над итоговым результатом на стороне приложения.

Схема алгоритма формирования рациона с использованием ИИ-диетолога приведена на рисунке В.6 приложения В.

Отдельного проектирования требует алгоритм сохранения сформированного ИИ-рациона в дневник питания. После того как пользователь получил подходящий рацион, он может выбрать его на экране ИИ-диетолога и нажать кнопку сохранения в дневник. Приложение должно проверить, не был ли данный рацион уже добавлен ранее. Такая проверка необходима для предотвращения повторного сохранения одних и тех же блюд и искажения дневной статистики.

Если рацион уже был сохранен, приложение выводит пользователю сообщение о повторном сохранении и не выполняет добавление записей в дневник. Если рацион еще не сохранялся, приложение получает список блюд из ИИ-рациона и проверяет наличие соответствующих блюд в локальной базе данных. При отсутствии отдельных блюд пользователю выводится сообщение о невозможности сохранения некоторых элементов рациона, после чего неподходящие блюда исключаются из списка.

После проверки приложение создает записи дневника питания. Для каждой записи указывается дата, тип приема пищи, выбранное блюдо и масса порции. Далее приложение рассчитывает показатели КБЖУ для каждой записи, сохраняет записи в локальной базе данных и устанавливает признак сохранения ИИ-рациона. После этого обновляется дневник питания, пересчитываются суммарные показатели КБЖУ за день и пользователю отображается обновленный дневник. Такой механизм гарантирует надежность учета и исключает риск дублирования записей при случайном повторном нажатии кнопки сохранения

Схема алгоритма сохранения ИИ-рациона в дневник питания приведена на рисунке В.7 приложения В.

Основные алгоритмы приложения и их назначение представлены в таблице 4.

Таблица 4 – Основные алгоритмы приложения

Алгоритм	Назначение	Входные данные	Результат выполнения
Расчет дневной нормы калорий	Определение индивидуальной целевой калорийности пользователя	Пол, возраст, рост, вес, уровень активности, цель питания	Рассчитанная дневная норма калорий
Добавление блюда в дневник	Сохранение выбранного блюда в дневник питания	Блюдо, масса порции, тип приема пищи, дата	Запись дневника с рассчитанными КБЖУ
Формирование ИИ-рациона	Получение персонального рациона на день	Профиль пользователя, дневная норма калорий, аллергии, локальная база блюд	Список блюд по приемам пищи с рассчитанными граммовками и КБЖУ
Сохранение ИИ-рациона в дневник	Перенос сформированного рациона в дневник питания	Сформированный рацион, дата, идентификатор пользователя	Записи дневника питания за текущий день

Таким образом, проектирование алгоритмов приложения позволяет определить порядок обработки пользовательских данных, расчета калорийности, взаимодействия с ИИ-диетологом и сохранения результатов в локальную базу данных. При этом итоговый результат контролируется приложением: данные проверяются, продукты сопоставляются с локальной

базой, аллергены исключаются, а показатели КБЖУ рассчитываются программно.

2.5 Проектирование классов приложения

Данный этап позволяет определить основные программные компоненты, их назначение и связи между ними до непосредственной реализации приложения.

Разрабатываемое приложение имеет слоистую структуру, поэтому классы можно разделить на несколько групп: классы пользовательского интерфейса, классы предметной области, классы локального хранения данных и классы сетевого взаимодействия. Такое разделение соответствует выбранной архитектуре приложения и позволяет изолировать пользовательский интерфейс от логики хранения данных и взаимодействия с внешним API [17].

Классы пользовательского интерфейса отвечают за отображение экранов приложения и обработку действий пользователя. К ним относятся экраны авторизации, регистрации, анкеты, профиля, дневника питания, добавления блюда и ИИ-диетолога. В проекте интерфейс реализуется с использованием Jetpack Compose, поэтому экраны представлены composable-компонентами. Основная навигация между экранами и инициализация компонентов приложения выполняются в классе MainActivity.

Классы предметной области описывают основные сущности и операции, не зависящие напрямую от способа хранения данных или особенностей пользовательского интерфейса. К ним относятся модели профиля пользователя, рациона, отдельного элемента рациона, а также перечисления пола, уровня активности и цели питания. Отдельно выделяется класс BmrCalculator, выполняющий расчет индивидуальной дневной нормы калорий, и ProfileValidationService, отвечающий за проверку корректности данных анкеты пользователя.

Слой локального хранения данных реализован с использованием библиотеки Room. Центральным классом данного слоя является `AppDatabase`, который объединяет сущности базы данных и предоставляет доступ к DAO-интерфейсам. Сущности UserEntity, UserProfileEntity, DishEntity, DiaryEntryEntity, ChatMessageEntity и AiMealPlanEntity соответствуют таблицам локальной базы данных. Для работы с ними используются DAO-интерфейсы UserDao, UserProfileDao, DishDao, DiaryDao, ChatDao и AiMealPlanDao.

Для отделения предметной модели профиля от структуры хранения используется класс ProfileRepository. Он выполняет преобразование данных между объектом профиля пользователя и сущностью базы данных.

Сетевой слой приложения предназначен для взаимодействия с внешним сервисом GigaChat API. Он включает интерфейс GigaChatApi, класс GigaChatAuth, объект RetrofitClient и классы передачи данных, используемые при формировании запросов и обработке ответов. GigaChatAuth отвечает за получение токена доступа, RetrofitClient создает настроенный экземпляр Retrofit, а GigaChatApi описывает методы обращения к внешнему API.

Особое место в проекте занимает класс DietologistViewModel, который связывает пользовательский интерфейс ИИ-диетолога, локальную базу данных и сетевой слой. Он получает профиль пользователя, список блюд, историю сообщений и сохраненные рационы, формирует запросы к GigaChat API, проверяет полученный рацион, выполняет фильтрацию аллергенов, рассчитывает граммовки и сохраняет результат в дневник питания.

Для наглядного представления основных классов приложения была разработана UML-диаграмма классов [18]. Она отражает основные программные компоненты приложения, их принадлежность к слоям и связи между ними. UML-диаграмма классов приложения приведена на рисунке В.8 приложения В.

Основные классы приложения и их назначение представлены в таблице 5.

Таблица 5 – Основные классы приложения NutritionDiary

Класс или интерфейс	Слой приложения	Назначение
MainActivity	Пользовательский интерфейс	Инициализация приложения, создание базы данных, управление авторизацией и основной навигацией
LoginScreen	Пользовательский интерфейс	Отображение экрана входа и обработка авторизации пользователя
RegisterScreen	Пользовательский интерфейс	Отображение экрана регистрации и создание новой учетной записи
SurveyScreen	Пользовательский интерфейс	Отображение анкеты пользователя и передача данных профиля для сохранения
ProfileScreen	Пользовательский интерфейс	Отображение персональных данных пользователя и рассчитанной нормы калорий
DiaryScreen	Пользовательский интерфейс	Отображение дневника питания, списка записей и суммарных показателей КБЖУ
AddDishScreen	Пользовательский интерфейс	Выбор блюда, ввод массы порции и добавление записи в дневник питания
DietologistScreen	Пользовательский интерфейс	Отображение чата с ИИ-диетологом и карточки сформированного рациона
DietologistViewModel	Слой управления состоянием	Генерация, корректировка и сохранение ИИ-рациона, работа с чатом и локальными данными
BmrCalculator	Предметная логика	Расчет индивидуальной дневной нормы калорий пользователя
ProfileValidationService	Предметная логика	Проверка корректности данных анкеты пользователя

Продолжение таблицы 5

Класс или интерфейс	Слой приложения	Назначение
ProfileRepository	Слой данных	Преобразование и сохранение профиля пользователя в локальной базе данных
AppDatabase	Слой данных	Описание Room-базы данных и предоставление доступа к DAO-интерфейсам
UserDao	Слой данных	Работа с данными пользователей
UserProfileDao	Слой данных	Работа с профилями пользователей
DishDao	Слой данных	Работа со справочником блюд и продуктов
DiaryDao	Слой данных	Работа с записями дневника питания
ChatDao	Слой данных	Работа с историей сообщений ИИ-диетолога
AiMealPlanDao	Слой данных	Работа с сохраненными ИИ-рационами
GigaChatApi	Сетевой слой	Описание HTTP-запросов к GigaChat API
GigaChatAuth	Сетевой слой	Получение токена доступа для обращения к GigaChat API
RetrofitClient	Сетевой слой	Создание и настройка Retrofit-клиента для сетевого взаимодействия

Проектирование классов позволяет показать внутреннюю организацию приложения и распределение ответственности между его компонентами. Пользовательский интерфейс отвечает за взаимодействие с пользователем, предметная логика выполняет расчеты и проверки, слой данных обеспечивает локальное хранение информации, а сетевой слой реализует обращение к внешнему сервису ИИ-диетолога.

3 Реализация мобильного приложения

3.1 Общая организация проекта

Реализация мобильного приложения NutritionDiary выполнена в среде Android Studio на языке Kotlin. Проект представляет собой Android-приложение с использованием Gradle Kotlin DSL, Jetpack Compose для построения пользовательского интерфейса, Room для локального хранения данных и Retrofit/OkHttp для сетевого взаимодействия с GigaChat API.

Исходный код приложения расположен в модуле app и разделен на пакеты в соответствии с выполняемыми функциями. Такая организация позволяет отделить экраны пользовательского интерфейса от логики обработки данных, локальной базы данных и сетевого взаимодействия. Основная часть кода размещена в пакетах data, domain и ui.

Пакет ui содержит компоненты пользовательского интерфейса. В нем расположены экраны входа, регистрации, анкеты пользователя, профиля, дневника питания, добавления блюда и ИИ-диетолога. Пользовательский интерфейс реализован с помощью Jetpack Compose, поэтому каждый экран представлен отдельной composable-функцией или набором composable-компонентов.

Пакет domain содержит классы, связанные с предметной логикой приложения. В нем размещены доменные модели пользователя и рациона, перечисления пола, активности и цели питания, класс расчета дневной нормы калорий, а также сервис проверки корректности данных анкеты. Эти классы используются разными экранами приложения и не зависят напрямую от способа отображения интерфейса.

Пакет data содержит реализацию работы с данными. В него входят компоненты локальной базы данных Room, DAO-интерфейсы, сущности таблиц, репозиторий профиля пользователя, начальные данные справочника блюд и классы сетевого взаимодействия с GigaChat API. В данном пакете сосредоточена работа с локальным хранилищем и внешним ИИ-сервисом.

Файл `MainActivity.kt` является точкой входа в приложение. В нем выполняется инициализация локальной базы данных, получение DAO-интерфейсов, настройка начального состояния приложения и выбор экрана, который должен быть показан пользователю. После запуска приложение отображает экран входа или регистрации, а после авторизации проверяет наличие заполненного профиля пользователя.

Зависимости проекта подключаются через Gradle. В приложении используются библиотеки Jetpack Compose, Room, Retrofit, OkHttp, Gson, Kotlin Coroutines и JUnit. Jetpack Compose применяется для построения интерфейса, Room – для хранения пользователей, профилей, блюд, дневника и истории взаимодействия с ИИ, Retrofit и OkHttp – для обращения к GigaChat API, а JUnit – для модульного тестирования логики валидации.

Выбранный набор библиотек определяет не только функциональные возможности приложения, но и структуру его исходного кода. Компоненты, связанные с пользовательским интерфейсом, вынесены в отдельный пакет, что позволяет изменять внешний вид экранов без переработки логики хранения и обработки данных. Классы, отвечающие за расчет дневной нормы калорий, проверку анкеты и формирование моделей рациона, размещены отдельно от экранов приложения, так как они относятся к предметной логике. Слой данных также выделен в самостоятельную часть проекта: в нем расположены сущности Room, DAO-интерфейсы, репозиторий профиля, начальные данные справочника блюд и классы сетевого взаимодействия с GigaChat API.

Кроме того, такое расположение файлов удобно при демонстрации проекта: по структуре каталогов можно быстро определить, где находятся модули авторизации, анкеты пользователя, дневника питания, ИИ-диетолога, локальной базы данных и сетевого API. Подобная модульная организация также существенно упрощает процесс отладки и написания изолированных модульных тестов для компонентов предметной логики.

Структура исходного кода проекта представлена на рисунке 6.

Структура исходного кода проекта NutritionDiary

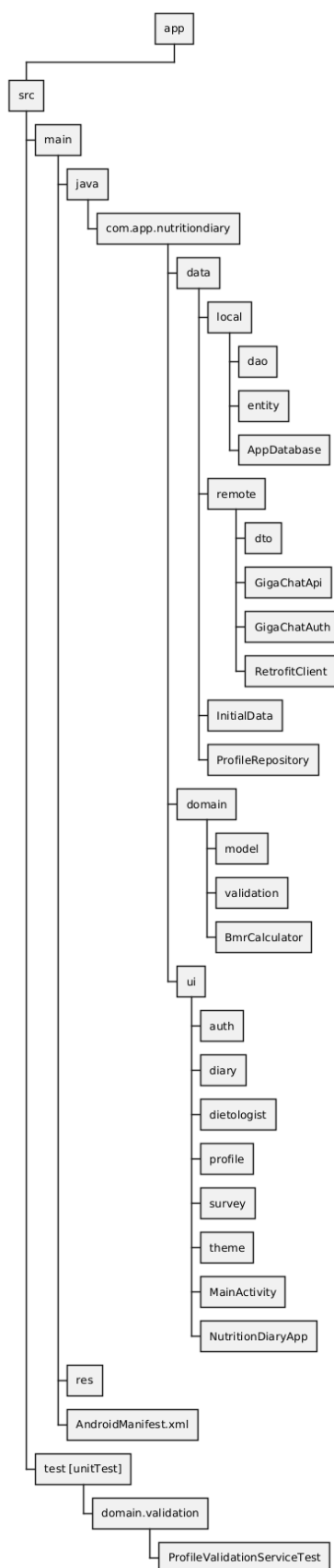


Рисунок 6 – Структура исходного кода проекта NutritionDiary

Основные пакеты проекта представлены в таблице 6.

Таблица 6 – Основные пакеты проекта NutritionDiary

Пакет	Содержимое	Назначение
data	Room-база данных, сущности, DAO, репозиторий, начальные данные, сетевой API	Работа с локальными и удаленными данными приложения
data.local	Класс базы данных Room, DAO-интерфейсы и сущности	Организация локального хранения данных
data.remote	Retrofit-клиент, интерфейс GigaChat API, авторизация в сервисе	Взаимодействие с внешним ИИ-сервисом
domain	Расчет калорий, модели, валидация анкеты	Реализация предметной логики приложения
domain.model	Модели профиля, рациона, элементов рациона и перечисления	Описание объектов предметной области
domain.validation	Сервис проверки данных анкеты и результат валидации	Проверка корректности пользовательского ввода
ui	Экраны приложения, основная активность и тема оформления	Реализация пользовательского интерфейса
ui.auth	Экраны входа и регистрации	Авторизация и создание локальной учетной записи
ui.survey	Экран анкеты пользователя	Ввод персональных параметров пользователя
ui.profile	Экран профиля	Отображение данных профиля и дневной нормы калорий
ui.diary	Экран дневника и экран добавления блюда	Ведение дневника питания
ui.dietologist	Экран ИИ-диетолога и ViewModel	Генерация, корректировка и сохранение ИИ-рациона
ui.theme	Цвета, типографика и тема приложения	Оформление интерфейса приложения

Такая организация проекта упрощает дальнейшее сопровождение приложения. При необходимости можно изменять отдельные части программы без полной переработки всей структуры: например, расширить локальную базу блюд, изменить экран дневника, доработать алгоритм расчета калорий или заменить внешний ИИ-сервис.

Ключевые фрагменты программной логики, локальной базы данных и сетевого взаимодействия приведены в листингах Г.1 - Г.5 приложения Г.

3.2 Реализация локальной базы данных

Для хранения данных приложения используется локальная база данных Room, основанная на SQLite. Данный подход позволяет сохранять информацию непосредственно на устройстве пользователя и обеспечивать работу основных функций приложения без необходимости обращаться к удаленному серверу для получения данных профиля, дневника питания и истории взаимодействия с ИИ-диетологом.

Локальная база данных реализована в классе AppDatabase. Данный класс является центральным компонентом слоя хранения данных и содержит описание используемых сущностей, версию базы данных и методы получения DAO-интерфейсов. В приложении база данных включает сущности пользователей, профилей пользователей, блюд, записей дневника питания, сообщений чата и сохраненных ИИ-рационов.

Сущность UserEntity используется для хранения данных учетной записи пользователя. Она содержит логин и пароль, необходимые для регистрации и авторизации. Для исключения дублирования учетных записей логин пользователя должен быть уникальным.

Сущность UserProfileEntity предназначена для хранения персональных данных пользователя. В нее входят имя, возраст, пол, рост, вес, уровень физической активности, цель питания, сведения об аллергиях, пищевые предпочтения и рассчитанная дневная норма калорий. Профиль связан с конкретным пользователем, что позволяет хранить разные данные для разных учетных записей на одном устройстве.

Сущность DishEntity описывает блюдо или продукт, доступный в локальной базе приложения. Для каждого блюда хранятся название, калорийность на 100 граммов, количество белков, жиров и углеводов на 100 граммов, категория, тип приема пищи, сведения об аллергенах и признак возможности использования при формировании ИИ-рациона. Эти данные применяются как при ручном добавлении блюда в дневник, так и при обработке рациона, полученного от ИИ-диетолога.

Сущность `DiaryEntryEntity` используется для хранения записей дневника питания. Каждая запись содержит сведения о пользователе, выбранном блюде, дате, типе приема пищи и массе порции. На основе массы порции и пищевой ценности блюда приложение рассчитывает фактическую калорийность и показатели БЖУ для конкретной записи.

Сущность `ChatMessageEntity` предназначена для хранения истории сообщений на экране ИИ-диетолога. В ней сохраняются текст сообщения, признак отправителя и время создания сообщения. Благодаря этому пользователь может видеть историю общения с интеллектуальным помощником.

Сущность `AiMealPlanEntity` используется для хранения сформированных ИИ-рационов. Она содержит дату, текстовое или структурированное представление рациона, признак сохранения в дневник и время создания. Наличие признака сохранения позволяет предотвратить повторное добавление одного и того же рациона в дневник питания.

Для работы с таблицами базы данных используются DAO-интерфейсы. `UserDao` отвечает за операции с учетными записями пользователей, `UserProfileDao` – за работу с профилями, `DishDao` – за получение блюд из справочника, `DiaryDao` – за добавление, удаление и получение записей дневника питания, `ChatDao` – за сохранение и получение сообщений чата, `AiMealPlanDao` – за работу со сформированными ИИ-рационами.

При запуске приложения выполняется инициализация локальной базы данных. Для справочника блюд предусмотрено начальное заполнение данными, что позволяет пользователю сразу выбирать продукты при ведении дневника питания и использовать их при формировании ИИ-рациона. Наличие локального справочника также необходимо для проверки результатов, полученных от внешнего ИИ-сервиса.

Использование `Room` упрощает работу с базой данных, так как позволяет описывать таблицы в виде Kotlin-классов, а запросы – в виде методов DAO-интерфейсов. Это снижает количество ручного SQL-кода и

делает структуру слоя данных более понятной. Кроме того, Room выполняет проверку запросов на этапе компиляции, что повышает надежность приложения.

Таким образом, реализованная локальная база данных обеспечивает хранение всех основных данных приложения: учетных записей, профилей пользователей, справочника блюд, записей дневника питания, истории сообщений и сформированных ИИ-рационов. Это позволяет приложению сохранять пользовательскую информацию между запусками и обеспечивать корректную работу дневника питания и ИИ-диетолога.

3.3 Реализация регистрации и авторизации пользователя

Одним из основных модулей приложения является модуль регистрации и авторизации пользователя. Он необходим для разделения данных между разными пользователями, работающими с приложением на одном устройстве. После входа в систему приложение получает идентификатор текущего пользователя и использует его при загрузке профиля, записей дневника питания, сообщений чата и сформированных ИИ-рационов.

Реализация регистрации и авторизации выполняется средствами локальной базы данных Room. Для хранения учетных записей используется сущность `UserEntity`, содержащая логин и пароль пользователя. Операции получения, добавления и проверки пользователей выполняются через интерфейс `UserDao`. Такой подход позволяет хранить данные пользователей локально и не использовать удаленный сервер авторизации.

При запуске приложения пользователь попадает на экран входа. Если у него уже есть учетная запись, он вводит логин и пароль. После нажатия кнопки входа приложение проверяет наличие пользователя с указанными данными в локальной базе данных. Если пользователь найден, его идентификатор сохраняется как идентификатор текущего пользователя, после чего приложение проверяет наличие заполненного профиля. Если профиль

отсутствует, открывается экран анкеты. Если профиль уже был создан ранее, пользователь переходит к основному интерфейсу приложения.

Если учетная запись отсутствует, пользователь может перейти на экран регистрации. На данном экране он вводит логин, пароль и подтверждение пароля. Перед сохранением выполняется проверка корректности введенных данных: поля не должны быть пустыми, пароль и подтверждение пароля должны совпадать, а логин не должен повторяться в базе данных. Если проверка не пройдена, приложение выводит сообщение об ошибке и не создает новую запись.

При успешной регистрации создается новый объект `UserEntity`, который сохраняется в таблицу пользователей. После этого приложение получает созданного пользователя из базы данных и использует его как текущего. Далее пользователь переходит к заполнению анкеты, так как для новой учетной записи профиль еще не создан.

Для реализации переходов между экранами используется состояние приложения, определяющее текущий этап работы пользователя. На начальном этапе отображается экран входа или регистрации. После успешной авторизации выполняется переход к проверке профиля. В зависимости от результата проверки открывается экран анкеты или основной интерфейс приложения. Такой подход позволяет отделить логику выбора экрана от визуального представления и упростить дальнейшее сопровождение кода.

Схема алгоритма регистрации и авторизации пользователя приведена на рисунке В.9 приложения В.

Таким образом, реализованный модуль регистрации и авторизации обеспечивает создание локальных учетных записей, проверку данных при входе и переход пользователя к дальнейшей работе с приложением. Использование локальной базы данных позволяет хранить несколько учетных записей на одном устройстве и связывать данные профиля, дневника питания и ИИ-диетолога с конкретным пользователем.

3.4 Реализация анкеты пользователя и расчета дневной нормы калорий

После регистрации или первого входа в приложение пользователь должен заполнить анкету. Данный этап необходим для получения персональных параметров, на основе которых приложение рассчитывает индивидуальную дневную норму калорий и в дальнейшем формирует рекомендации по питанию.

Экран анкеты реализован в виде composable-компонента SurveyScreen. На экране размещены поля ввода имени, возраста, роста и веса пользователя, а также элементы выбора пола, уровня физической активности, цели питания и аллергий. Для параметров, имеющих ограниченный набор значений, используются переключатели и варианты выбора. Такой подход уменьшает вероятность ошибок при заполнении анкеты и делает интерфейс более удобным для пользователя.

Перед сохранением профиля выполняется проверка корректности введенных данных. Для этого используется класс ProfileValidationService. Он проверяет, что обязательные поля заполнены, а числовые значения находятся в допустимых диапазонах. Например, возраст, рост и вес пользователя не должны быть нулевыми или отрицательными. Если введенные данные некорректны, приложение отображает сообщение об ошибке и не выполняет сохранение профиля.

После успешной проверки данных выполняется расчет индивидуальной дневной нормы калорий. Для этого используется класс BmrCalculator. Расчет выполняется на основе пола, возраста, роста, веса, уровня физической активности и цели питания пользователя. Сначала определяется базовый обмен веществ, после чего полученное значение умножается на коэффициент физической активности. Затем результат корректируется в зависимости от цели пользователя: при снижении веса создается дефицит калорий, при поддержании веса значение остается без изменений, а при наборе массы добавляется дополнительная калорийность.

Результатом работы алгоритма является рассчитанная дневная норма калорий. Она сохраняется вместе с остальными данными профиля и отображается пользователю на экране профиля. Благодаря этому пользователь может видеть свою целевую калорийность и ориентироваться на нее при ведении дневника питания.

Экран профиля пользователя предназначен для отображения сохраненных персональных данных, выбранной цели питания, уровня физической активности, списка аллергий и рассчитанной дневной нормы калорий. Также на экране предусмотрены кнопка редактирования профиля и кнопка выхода из учетной записи. Пример отображения профиля пользователя представлен на рисунке 7.



Рисунок 7 – Экран профиля пользователя

Для хранения профиля используется сущность `UserProfileEntity`. Она содержит персональные параметры пользователя, сведения об аллергиях, пищевых предпочтениях и рассчитанную дневную норму калорий. Сохранение и получение профиля выполняется через `ProfileRepository`, который преобразует данные между доменной моделью `UserProfile` и сущностью базы данных. Такой подход позволяет отделить представление профиля в интерфейсе от структуры хранения данных в Room.

После сохранения анкеты приложение связывает профиль с текущим пользователем. Это необходимо для того, чтобы при последующих запусках приложения автоматически загружать персональные данные конкретной учетной записи. Если пользователь уже заполнял анкету ранее, приложение может открыть основной интерфейс без повторного ввода данных.

Экран анкеты пользователя представлен на рисунке 8.

5:15

Давайте настроим ваш профиль

Ваше имя

Пол

☐ Мужской ☐ Женский

Возраст

Рост (см)

Вес (кг)

Уровень активности

☐ Сидячий образ жизни

☐ Лёгкая активность (1-3 раза в неделю)

☐ Умеренная активность (3-5 раза в неделю)

☐ Высокая активность (6-7 раза в неделю)

☐ Экстремальная активность

Цель

☐ Похудение

☐ Поддержание веса

☐ Набор мышечной массы

Аллергии (выберите при наличии)

☐ Молоко

☐ Орехи

☐ Глютен

☐ Яйца

☐ Рыба

☐ Соевые продукты

Рассчитать норму калорий

Рисунок 8 – Экран анкеты пользователя

Таким образом, реализация анкеты пользователя обеспечивает сбор персональных данных, проверку корректности ввода, расчет индивидуальной дневной нормы калорий и сохранение профиля в локальной базе данных. Эти данные используются в дальнейшем при отображении профиля, ведении дневника питания и формировании ИИ-рациона.

3.5 Реализация дневника питания

Дневник питания является одним из основных функциональных модулей приложения. Он предназначен для ежедневного учета съеденных блюд, расчета калорийности и отображения суммарных показателей белков, жиров и углеводов за выбранный день.

Экран дневника питания реализован в виде composable-компонента DiaryScreen. На данном экране отображается информация о текущей дате, рассчитанной дневной норме калорий, суммарном количестве потребленных калорий и показателях БЖУ. Также на экране выводится список блюд, добавленных пользователем за выбранный день.

Для хранения записей дневника используется сущность DiaryEntryEntity. Каждая запись содержит идентификатор пользователя, идентификатор выбранного блюда, дату, тип приема пищи и массу порции. Такая структура позволяет отделять записи разных пользователей друг от друга и получать дневник питания только для текущей учетной записи.

Данные о блюдах хранятся в сущности DishEntity. Для каждого блюда указаны название, калорийность на 100 граммов, количество белков, жиров и углеводов на 100 граммов, категория, тип приема пищи и сведения об аллергенах. При добавлении блюда в дневник пользователь выбирает продукт из локального справочника и указывает массу порции в граммах.

Добавление блюда выполняется через экран AddDishScreen. На нем пользователь выбирает дату, тип приема пищи, блюдо из списка и массу порции. Перед сохранением записи приложение проверяет корректность введенных данных. Масса порции должна быть положительным числом, а

блюдо и тип приема пищи должны быть выбраны. Если данные некорректны, пользователю выводится сообщение об ошибке.

После успешной проверки данных приложение создает новую запись дневника питания и сохраняет ее в локальной базе данных через DiaryDao. После сохранения список записей дневника обновляется, а суммарные показатели за выбранный день пересчитываются.

Расчет пищевой ценности порции выполняется на основе данных блюда на 100 граммов. Для этого значение калорийности, белков, жиров и углеводов умножается на массу порции и делится на 100. Такой расчет применяется отдельно для каждого показателя. Благодаря этому пользователь может указывать любую массу блюда, а приложение автоматически рассчитывает фактическую пищевую ценность порции.

На экране дневника питания отображается не только список записей, но и итоговая статистика за день. Приложение суммирует калорийность, белки, жиры и углеводы по всем записям выбранной даты. Полученные значения используются для отображения прогресса выполнения дневной нормы калорий и позволяют пользователю оценить текущее состояние рациона.

Для каждой записи дневника предусмотрена возможность удаления. При удалении запись удаляется из локальной базы данных через DiaryDao, после чего список блюд и суммарные показатели за день обновляются. Это позволяет пользователю исправлять ошибочно добавленные продукты и поддерживать актуальность дневника питания. Все операции изменения дневника выполняются асинхронно с использованием корутин Kotlin, что исключает блокировку пользовательского интерфейса при работе с базой данных.

Если за выбранный день отсутствуют записи, приложение отображает пустое состояние с подсказкой о необходимости добавить блюдо вручную или сформировать рацион с помощью ИИ-диетолога. Такой подход делает интерфейс более понятным и помогает пользователю быстрее перейти к основным действиям.

Экран дневника питания представлен на рисунке 9.

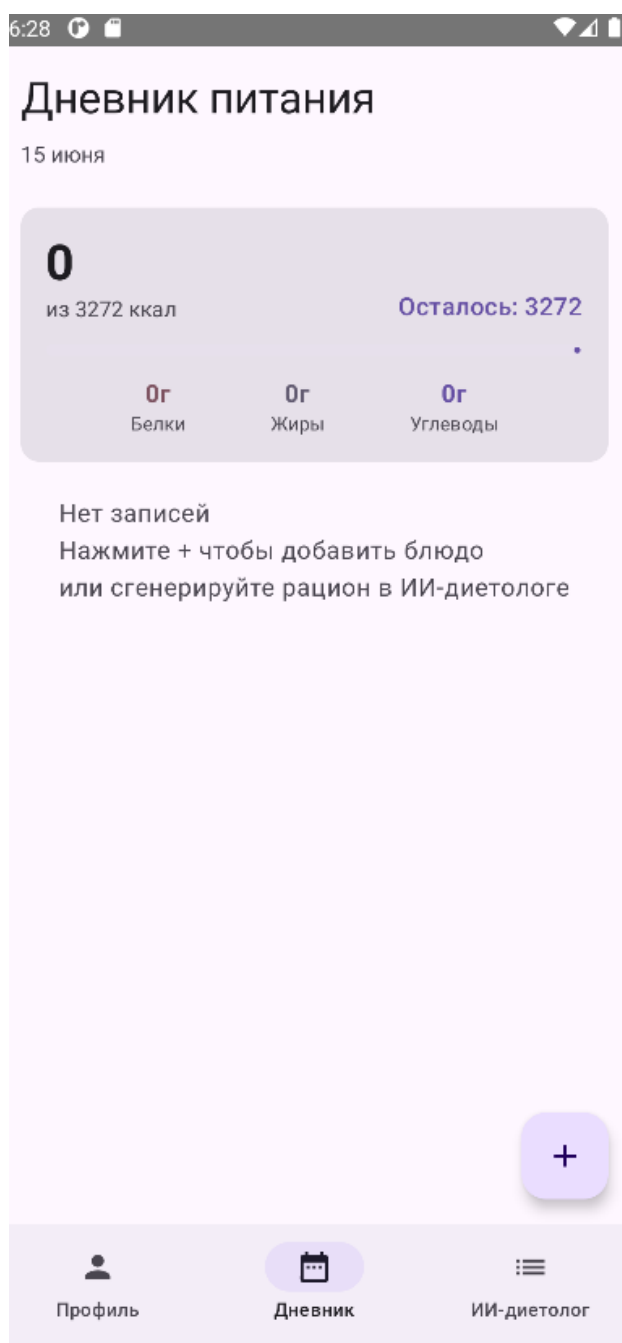


Рисунок 9 – Экран дневника питания

Таким образом, реализация дневника питания обеспечивает добавление, хранение, отображение и удаление записей о приемах пищи. Автоматический расчет калорийности и показателей БЖУ позволяет пользователю контролировать рацион за день и сравнивать фактическое потребление с рассчитанной дневной нормой калорий.

3.6 Реализация ИИ-диетолога

Одной из ключевых особенностей разрабатываемого приложения является наличие ИИ-диетолога, который позволяет пользователю получать персонализированный рацион и задавать вопросы о питании в диалоговом режиме. Данный модуль объединяет пользовательский интерфейс чата, работу с локальной базой данных и сетевое взаимодействие с внешним сервисом GigaChat API.

Экран ИИ-диетолога реализован в виде composable-компонента DietologistScreen. Основная область экрана представляет собой список сообщений пользователя и ответов интеллектуального помощника. В нижней части экрана расположено поле ввода текста и кнопка отправки сообщения. Такой интерфейс выбран потому, что диалоговая форма взаимодействия является привычной для пользователя и позволяет формулировать запросы на естественном языке.

Для управления состоянием экрана используется класс DietologistViewModel. Он отвечает за загрузку истории сообщений, сохранение новых сообщений, отправку запросов к GigaChat API, обработку ответов, генерацию рациона, корректировку рациона и сохранение результата в локальную базу данных. Выделение данной логики во ViewModel позволяет отделить пользовательский интерфейс от обработки данных и сетевых запросов.

Перед формированием рациона приложение получает профиль текущего пользователя, рассчитанную дневную норму калорий, цель питания и список аллергий. Эти данные используются при составлении запроса к ИИ-диетологу. Дополнительно из локальной базы данных загружается список блюд, разрешенных для использования при генерации рациона. Продукты, содержащие аллергены пользователя, исключаются из списка доступных вариантов.

Для обращения к GigaChat API используется сетевой слой приложения. Интерфейс GigaChatApi описывает HTTP-запросы к внешнему сервису, класс

GigaChatAuth отвечает за получение токена доступа, а объект RetrofitClient создает настроенный экземпляр Retrofit. Запрос к сервису формируется в формате JSON и содержит системное сообщение, параметры пользователя, список доступных блюд и текст пользовательского запроса.

После получения ответа от GigaChat API приложение не использует его без дополнительной проверки. Сначала из ответа извлекается текст сформированного рациона или консультации. Если ответ содержит рацион, приложение проверяет его на наличие продуктов, противоречащих ограничениям пользователя. Также выполняется сопоставление предложенных блюд с локальной базой данных. Это необходимо для того, чтобы итоговый рацион мог быть корректно сохранен в дневник питания и чтобы значения КБЖУ рассчитывались по данным приложения.

История общения с ИИ-диетологом сохраняется в локальной базе данных с использованием сущности ChatMessageEntity и интерфейса ChatDao. Для каждого сообщения сохраняются текст, признак отправителя и время создания. Благодаря этому после повторного открытия экрана пользователь может продолжить работу с сохраненной историей диалога.

Сформированный рацион сохраняется отдельно с использованием сущности AiMealPlanEntity и интерфейса AiMealPlanDao. В записи хранится дата формирования, содержимое рациона, время создания и признак сохранения в дневник питания. Это позволяет отображать ранее сформированный рацион и предотвращать его повторное добавление в дневник.

Карточка рациона отображает распределение блюд по приемам пищи, а также итоговую калорийность, что позволяет сразу оценить соответствие предложенного меню рассчитанной дневной норме. Наличие поля ввода делает экран не только средством просмотра результата, но и инструментом интерактивной корректировки: пользователь может попросить заменить продукт, изменить состав приема пищи или сформировать другой вариант

рациона. После сохранения выбранного варианта данные передаются в дневник питания и используются в общей статистике за день.

Экран ИИ-диетолога представлен на рисунке 10.

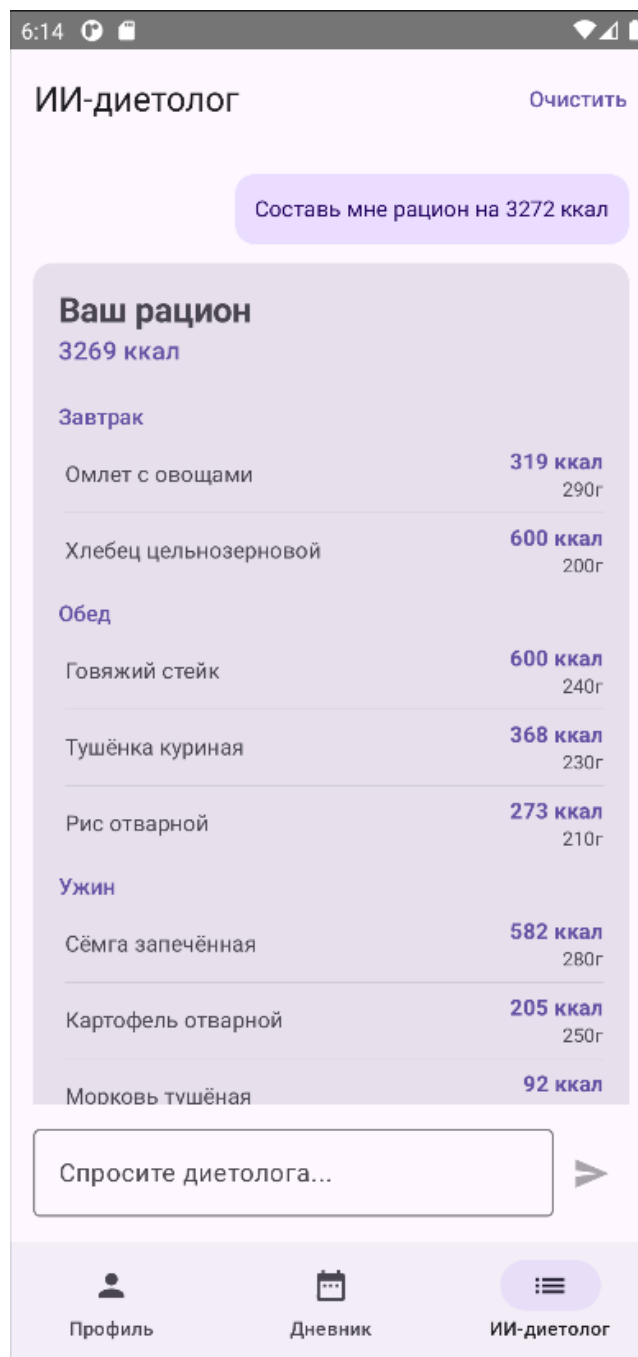


Рисунок 10 – Экран ИИ-диетолога

Таким образом, реализация ИИ-диетолога обеспечивает генерацию персонализированного рациона, диалоговое взаимодействие пользователя с интеллектуальным помощником, сохранение истории сообщений и проверку полученных результатов перед использованием в дневнике питания.

Ключевые фрагменты реализации ИИ-диетолога приведены в листингах Г.6-Г.8 приложения Г.

3.7 Реализация сохранения ИИ-рациона в дневник питания

Функция сохранения ИИ-рациона в дневник питания реализована в модуле ИИ-диетолога и связывает результат генерации рациона с основным дневником приложения. После формирования рациона пользователь может нажать кнопку сохранения, после чего приложение переносит элементы рациона в записи дневника питания за текущую дату.

Основная логика данной функции размещена в классе `DietologistViewModel`. Он получает текущий сформированный рацион, проверяет признак его сохранения, сопоставляет блюда из рациона с локальным справочником `DishEntity` и формирует набор записей `DiaryEntryEntity`. Для сохранения данных используются DAO-интерфейсы `AiMealPlanDao`, `DishDao` и `DiaryDao`.

Перед добавлением записей приложение проверяет значение признака сохранения в объекте `AiMealPlanEntity`. Если рацион уже был сохранен ранее, повторное добавление не выполняется, а пользователь получает соответствующее сообщение. Такая проверка необходима для защиты дневника от дублирования записей и повторного увеличения суммарной калорийности за день.

Если рацион еще не был сохранен, приложение создает записи дневника питания для каждого блюда, которое удалось сопоставить с локальной базой. В каждую запись заносится идентификатор пользователя, идентификатор блюда, дата, тип приема пищи и масса порции. После сохранения записей через `DiaryDao` приложение обновляет признак сохранения ИИ-рациона через `AiMealPlanDao`. После сопоставления блюд приложение создает записи дневника питания. Для каждого элемента рациона формируется объект `DiaryEntryEntity`, в котором указываются идентификатор пользователя, идентификатор блюда, дата, тип приема пищи и масса порции.

Созданные записи сохраняются в локальной базе данных через интерфейс DiaryDao.

При добавлении записей в дневник приложение рассчитывает пищевую ценность каждой порции. Значения калорийности, белков, жиров и углеводов берутся из сущности DishEntity на 100 граммов продукта и пересчитываются пропорционально указанной массе порции. После сохранения всех записей дневник питания обновляется, а суммарные показатели за день пересчитываются.

После успешного сохранения приложение изменяет признак сохранения ИИ-рациона в AiMealPlanEntity. Это позволяет при последующем открытии экрана определить, что данный рацион уже был перенесен в дневник, и не допустить повторного сохранения. Пользователь получает уведомление об успешном добавлении рациона. Результат сохранения ИИ-рациона отображается на экране дневника питания.

Пример сохраненного ИИ-рациона в дневнике питания представлен на рисунке 11.

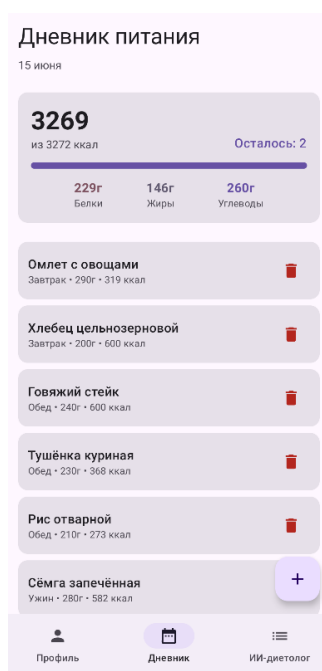


Рисунок 11 – Сохраненный ИИ-рацион в дневнике питания

Фрагмент реализации сохранения ИИ-рациона в дневник питания приведен в листинге Г.9 приложения Г.

4 Отладка и тестирование приложения

После завершения реализации основных модулей приложения необходимо выполнить отладку и тестирование. Данный этап позволяет проверить корректность работы программного продукта, выявить ошибки пользовательского ввода, убедиться в правильности расчетов и подтвердить соответствие реализованных функций сформулированным требованиям.

В рамках выпускной квалификационной работы тестирование проводилось для основных пользовательских сценариев приложения: регистрации и авторизации пользователя, заполнения анкеты, расчета дневной нормы калорий, ведения дневника питания, добавления блюд, взаимодействия с ИИ-диетологом и сохранения сформированного ИИ-рациона в дневник питания.

Для проверки приложения использовались функциональное ручное тестирование и модульное тестирование. Функциональное тестирование применялось для проверки работы приложения с точки зрения пользователя. Модульное тестирование использовалось для проверки отдельных компонентов предметной логики, не зависящих от пользовательского интерфейса.

4.1 Обоснование выбора режима тестирования

Тестирование программного обеспечения представляет собой процесс проверки программы с целью обнаружения ошибок и подтверждения корректности ее работы. Для мобильного приложения дневника питания особенно важно проверить обработку пользовательского ввода, корректность расчета дневной нормы калорий, сохранение данных в локальной базе и правильность переходов между экранами.

В разрабатываемом приложении часть функций связана с взаимодействием пользователя с интерфейсом. К таким функциям относятся регистрация, вход, заполнение анкеты, добавление блюд в дневник и работа с ИИ-диетологом. Для проверки этих сценариев целесообразно использовать

функциональное тестирование методом «черного ящика». При таком подходе проверяется соответствие входных данных и результата без анализа внутренней реализации программного кода.

Функциональное тестирование позволяет убедиться, что приложение корректно реагирует на действия пользователя: принимает допустимые данные, отклоняет некорректные значения, отображает сообщения об ошибках и выполняет переходы между экранами. Данный вид тестирования особенно важен для формы анкеты пользователя, так как от корректности введенных данных зависит расчет индивидуальной дневной нормы калорий и дальнейшая персонализация рациона.

Для проверки отдельных компонентов предметной логики используется модульное тестирование. Оно позволяет проверить работу функций и классов независимо от пользовательского интерфейса. В рамках данной работы модульное тестирование применяется для класса `ProfileValidationService`, отвечающего за проверку данных анкеты пользователя. Такой подход позволяет убедиться в корректной обработке пустых значений, недопустимых символов, неправильных диапазонов возраста, роста и веса, а также отсутствия обязательного выбора пола, активности и цели питания.

Также в процессе разработки выполнялась отладка приложения в среде Android Studio. Приложение запускалось на Android Emulator, после чего проверялась работа экранов, переходов, сохранения данных в базе Room и обработки ответов от GigaChat API. При возникновении ошибок анализировались сообщения Logcat, состояние переменных и результаты выполнения запросов к локальной базе данных.

Таким образом, выбранные режимы тестирования позволяют проверить приложение как со стороны пользователя, так и на уровне отдельных компонентов логики. Функциональное тестирование подтверждает соответствие реализованных сценариев требованиям, а модульное тестирование повышает надежность отдельных программных компонентов.

Пользователь получает возможность не только сформировать персональный рацион, но и сразу использовать его для ежедневного учета питания.

4.2 Функциональное ручное тестирование пользовательского ввода

Одним из наиболее важных сценариев приложения является заполнение анкеты пользователя. На основе данных анкеты рассчитывается индивидуальная дневная норма калорий, поэтому приложение должно корректно проверять введенные значения до сохранения профиля.

Для формы анкеты пользователя были определены следующие требования к валидации:

- имя является обязательным полем, должно иметь длину от 2 до 50 символов и содержать только буквы и пробелы;
- возраст должен быть целым числом от 1 до 120;
- рост должен быть целым числом от 50 до 250 см;
- вес должен быть числом от 10,0 до 300,0 кг;
- пол, уровень активности и цель питания должны быть выбраны обязательно;
- аллергии являются необязательными параметрами и выбираются пользователем при наличии.

Для проверки формы анкеты пользователя были выделены причины и следствия событий. Причины описывают условия, которым должны соответствовать входные данные, а следствия определяют результат, возникающий при нарушении этих условий. Такой подход позволяет представить проверку пользовательского ввода в виде функциональной модели и на ее основе сформировать набор тест-кейсов. Подобная формализация требований позволяет систематизировать процесс проверки и гарантировать, что ни одно граничное значение или логическое условие не останется без внимания.

Причины и следствия для проверки анкеты профиля представлены в таблице 7.

Таблица 7 – Причины и следствия

Обозначение	Описание
П1	Имя имеет корректную длину от 2 до 50 символов
П2	Имя содержит только буквы и пробелы
П3	Возраст находится в диапазоне от 1 до 120
П4	Рост находится в диапазоне от 50 до 250 см
П5	Вес находится в диапазоне от 10,0 до 300,0 кг
П6	Пол выбран
П7	Уровень активности выбран
П8	Цель питания выбрана
С1	Ошибка «Некорректная длина имени»
С2	Ошибка «Недопустимые символы в имени»
С3	Ошибка «Возраст вне диапазона»
С4	Ошибка «Рост вне диапазона»
С5	Ошибка «Вес вне диапазона»
С6	Ошибка «Пол не выбран»
С7	Ошибка «Активность не выбрана»
С8	Ошибка «Цель не выбрана»
С9	Профиль успешно сохранен

Используя таблицу причин и следствий, была построена функциональная диаграмма проверки анкеты пользователя. На диаграмме показано соответствие между условиями пользовательского ввода и результатами проверки. Если причина не выполняется, приложение должно вывести соответствующее сообщение об ошибке. Если все причины удовлетворены, профиль пользователя сохраняется.

Функциональная диаграмма проверки анкеты пользователя представлена на рисунке 12.

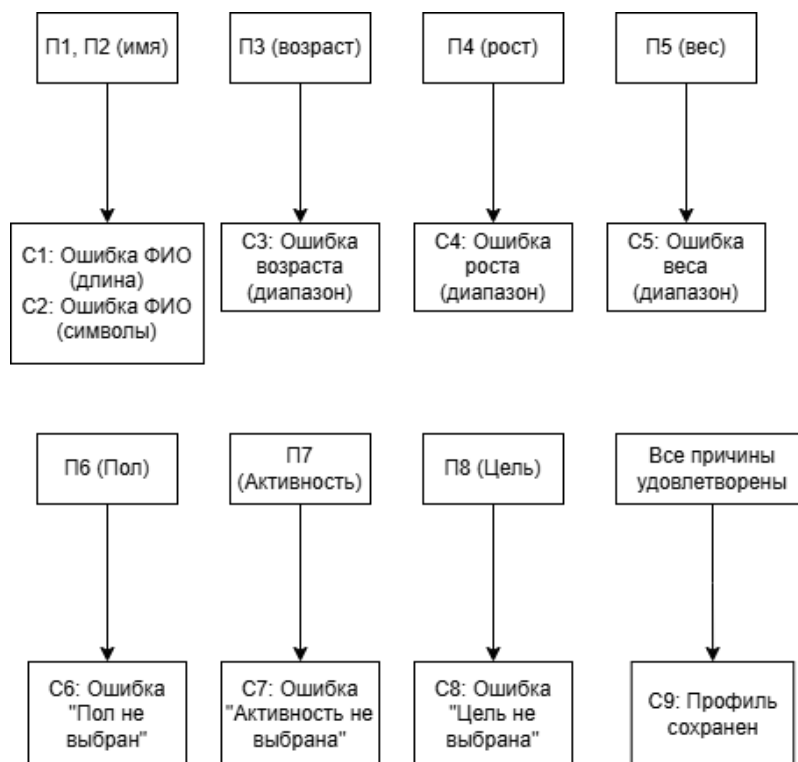


Рисунок 12 – Функциональная диаграмма проверки анкеты пользователя

Так как полная проверка всех возможных сочетаний входных данных приводит к большому количеству вариантов, для формирования набора тест-кейсов был использован подход попарного тестирования. Он позволяет сократить количество проверок, сохранив покрытие основных сочетаний значений входных параметров.

На основе причин и следствий были сформированы тест-кейсы для анкеты профиля, представленные в таблице 8.

Таблица 8 – Тест-кейсы для анкеты профиля

№	Имя	Возраст	Рост	Вес	Пол	Активность	Цель	Ожидаемый результат
1	А	25	170	70	М	LIGHT	MAINTAIN	Ошибка «Имя должно содержать хотя 2 символа»

Продолжение таблицы 8

№	Имя	Возраст	Рост	Вес	Пол	Активность	Цель	Ожидаемый результат
2	Антон#	25	170	70	М	LIGHT	MAINTAIN	Ошибка «Имя содержит недопустимые символы»
3	Антон	0	170	70	F	LIGHT	MAINTAIN	Ошибка «Возраст должен быть от 1 до 120 лет»
4	Антон	25	40	70	М	LIGHT	MAINTAIN	Ошибка «Рост должен быть от 50 до 250 см»
5	Антон	25	300	70	М	LIGHT	MAINTAIN	Ошибка «Рост должен быть от 50 до 250 см»
6	Антон	25	170	5	М	LIGHT	MAINTAIN	Ошибка «Вес должен быть от 10 до 300 кг»
7	Антон	25	170	350	М	LIGHT	MAINTAIN	Ошибка «Вес должен быть от 10 до 300 кг»
8	Антон	25	170	70	–	LIGHT	MAINTAIN	Ошибка «Выберите пол»
9	Антон	25	170	70	М	–	MAINTAIN	Ошибка «Выберите уровень активности»
10	Антон	25	170	70	М	LIGHT	–	Ошибка «Цель не выбрана»

Все тест-кейсы были выполнены вручную на эмуляторе Android. Фактические результаты совпали с ожидаемыми: приложение корректно отображало сообщения об ошибках при некорректном вводе и сохраняло профиль при корректном заполнении всех обязательных полей. Результаты функционального тестирования приведены на рисунках Д.1–Д.10 приложения Д.

4.3 Модульное тестирование

Для проверки логики валидации данных анкеты был создан класс `ProfileValidationService`. Он содержит методы, отвечающие за проверку отдельных полей формы профиля пользователя: имени, возраста, роста, веса, а также выбора пола, уровня активности и цели питания.

Класс `ProfileValidationService` содержит следующие методы:

- `validateName(name: String): ValidationResult;`
- `validateAge(age: String): ValidationResult;`
- `validateHeight(height: String): ValidationResult;`
- `validateWeight(weight: String): ValidationResult;`
- `checkSelections(gender: Gender?, activity: ActivityLevel?, goal: Goal?): ValidationResult.`

Тесты были написаны с использованием JUnit 4 и Kotlin. Для каждого метода были подготовлены входные данные, позволяющие проверить как корректные, так и ошибочные варианты пользовательского ввода. Это позволило убедиться, что класс `ProfileValidationService` возвращает успешный результат при корректных данных и сообщение об ошибке при нарушении требований к полям анкеты.

Фрагмент тестового класса `ProfileValidationServiceTest` приведен в листинге Е.1 приложения Е.

Функциональная спецификация unit-тестов представлена в таблице 9.

Таблица 9 – Функциональная спецификация unit-тестов

№	Название теста	Входные данные	Ожидаемый результат
1	validateName empty	«»	Ошибка «Введите имя»
2	validateName too short	«А»	Ошибка «Некорректная длина»
3	validateName with digits	«Иван1»	Ошибка «Недопустимые символы»
4	validateAge out of range	«0»	Ошибка «Возраст должен быть от 1 до 120»
5	validateAge valid	«25»	ОК
6	validateHeight out of range	«300»	Ошибка «Рост вне диапазона»
7	validateWeight out of range	«5.0»	Ошибка «Вес вне диапазона»
8	checkSelections missing gender	gender = null	Ошибка «Выберите пол»
9	checkSelections all valid	все параметры заданы	ОК

Все 9 unit-тестов были выполнены. Все они завершились успешно. Результаты выполнения unit-тестов представлены на рисунке 13.

```

✓ 9 tests passed 9 tests total, 24 ms

Executing tasks: [:app:testDebugUnitTest, --tests, com.app.nutritiondiary.domain.validation.ProfileValidationServiceTest] in project D:\A

> Task :app:checkKotlinGradlePluginConfigurationErrors SKIPPED
> Task :app:preBuild UP-TO-DATE
> Task :app:preDebugBuild UP-TO-DATE
> Task :app:generateDebugResValues
> Task :app:mapDebugSourceSetPaths
> Task :app:generateDebugResources
> Task :app:checkDebugAarMetadata
> Task :app:packageDebugResources
> Task :app:mergeDebugResources
> Task :app:createDebugCompatibleScreenManifests
> Task :app:extractDeepLinksDebug
> Task :app:parseDebugLocalResources
> Task :app:processDebugMainManifest
> Task :app:processDebugManifest
> Task :app:preDebugUnitTestBuild UP-TO-DATE
> Task :app:processDebugManifestForPackage
> Task :app:javaPreCompileDebugUnitTest
> Task :app:javaPreCompileDebug
> Task :app:processDebugResources
> Task :app:kspDebugKotlin
> Task :app:compileDebugKotlin
> Task :app:compileDebugJavaWithJavac NO-SOURCE
> Task :app:bundleDebugClassesToCompileJar
> Task :app:bundleDebugClassesToRuntimeJar
> Task :app:processDebugJavaRes
> Task :app:kspDebugUnitTestKotlin
> Task :app:compileDebugUnitTestKotlin
> Task :app:compileDebugUnitTestJavaWithJavac NO-SOURCE

```

Рисунок 13 – Результат unit-тестов

Полученный результат подтверждает, что реализованные модульные тесты успешно проверяют как корректные, так и ошибочные варианты заполнения анкеты пользователя. Это позволяет считать проверку логики валидации достаточной для подтверждения работоспособности выбранного программного компонента.

5 Контроль качества приложения

5.1 Обоснование выбора метрик кода

Контроль качества программного кода необходим для оценки сложности, читаемости и сопровождаемости разработанного приложения [19]. Даже при корректной работе программы важно оценить, насколько удобно ее сопровождать, изменять и расширять в дальнейшем.

В рамках данной выпускной квалификационной работы для оценки качества программного кода используются метрики Холстеда и Чепина. Данные метрики позволяют количественно оценить сложность отдельного программного модуля и проанализировать характер использования переменных внутри метода.

Метрика Холстеда основывается на подсчете операторов и операндов в программном коде. С ее помощью можно определить словарь программы, длину программы, объем кода, вычислительную сложность и усилия, необходимые для понимания или реализации фрагмента программы.

Метрика Чепина применяется для оценки сложности программы с точки зрения использования переменных. Она разделяет переменные на группы в зависимости от их назначения: входные данные, модифицируемые переменные, управляющие переменные и служебные переменные. Такой анализ позволяет определить, насколько метод зависит от входных данных, изменяет ли состояние программы и содержит ли лишние переменные.

Для оценки качества был выбран метод `calculateBmr` класса `BmrCalculator`. Данный метод отвечает за вычисление базового обмена веществ пользователя на основе массы, роста, возраста и пола. Он относится к предметной логике приложения и используется при расчете индивидуальной дневной нормы калорий. Поэтому корректность и простота данного метода имеют важное значение для работы приложения.

Фрагмент метода `calculateBmr` приведен в листинге Г.1 приложения Г.

5.2 Оценка качества программного кода

5.2.1 Метрика Холстеда

Метрика Холстеда была рассчитана для метода `calculateBmr` класса `BmrCalculator`. Данный метод выполняет вычисление базового обмена веществ пользователя по формуле Миффлина – Сан Жеора. В методе сначала рассчитывается базовое значение, зависящее от массы, роста и возраста пользователя, после чего результат корректируется в зависимости от пола [20].

Для расчета метрик Холстеда необходимо определить четыре исходных значения:

- $n1$ – количество уникальных операторов;
- $n2$ – количество уникальных операндов;
- $N1$ – общее количество операторов;
- $N2$ – общее количество операндов.

Словарь программы определяется по формуле (4):

$$n = n1 + n2, \quad (4)$$

где n – словарь программы;

$n1$ – количество уникальных операторов;

$n2$ – количество уникальных операндов.

Длина программы определяется по формуле (5):

$$N = N1 + N2 \quad (5)$$

где N – длина программы;

$N1$ – общее количество операторов;

$N2$ – общее количество операндов.

Объем программы вычисляется по формуле (6):

$$V = N \times \log_2(n), \quad (6)$$

где V – объем программы.

Вычислительная сложность определяется по формуле (7):

$$D = (n1 \div 2) \times (N2 \div n2), \quad (7)$$

где D – вычислительная сложность.

Усилия вычисляются по формуле (8):

$$E = D \times V, \quad (8)$$

где E – усилия, необходимые для понимания или реализации фрагмента программы.

Для метода calculateBmr были получены следующие значения:

- $n1 = 13$;
- $n2 = 6$;
- $N1 = 21$;
- $N2 = 8$.

Результаты расчета метрик Холстеда представлены в таблице 10.

Таблица 10 – Метрики Холстеда для метода calculateBmr

Характеристика	Значение
Словарь программы n	19
Длина программы N	29
Объем программы V	123,25
Вычислительная сложность D	8,67
Усилия E	1067

Полученные значения показывают, что метод calculateBmr имеет небольшой объем и невысокую вычислительную сложность. Объем программы $V = 123,25$ является небольшим, что говорит о простоте анализируемого фрагмента кода. Значение усилий $E = 1067$ также является невысоким, следовательно, метод легко понять, проверить и сопровождать.

5.2.2 Метрика Чепина

Метрика Чепина позволяет оценить сложность программного модуля на основе анализа переменных, используемых в методе. Все переменные делятся на четыре группы:

- Р – вводимые переменные, которые используются как исходные данные и не изменяются внутри метода;
- М – модифицируемые переменные, значения которых изменяются в процессе выполнения метода;
- С – управляющие переменные, влияющие на выбор ветви алгоритма;
- Т – служебные переменные, которые используются временно и не имеют существенного значения для логики метода.

Для расчета метрики используется формула (9):

$$Q = a1 \times P + a2 \times M + a3 \times C + a4 \times T, \quad (9)$$

где Q – значение метрики Чепина;

a1, a2, a3, a4 – весовые коэффициенты групп переменных.

Для расчета используются следующие коэффициенты:

a1 = 1; a2 = 2; a3 = 3; a4 = 0,5.

В методе calculateBmr входными данными являются объект profile и его поля: weightKg, heightCm, age и gender. Они используются для вычисления результата и не изменяются внутри метода. Модифицируемые переменные отсутствуют, так как метод не изменяет состояние объекта. Управляющей переменной является gender, поскольку от значения пола пользователя зависит выбор формулы. Служебной переменной является base, в которой хранится промежуточное значение базового обмена веществ.

Распределение переменных по группам представлено в таблице 11.

Таблица 11 – Метрика Чепина для метода calculateBmr

Группа переменных	Переменные	Количество
Р	profile; profile.weightKg; profile.heightCm; profile.age; profile.gender	5
М	–	0

Продолжение таблицы 11

Группа переменных	Переменные	Количество
C	profile.gender	1
T	base	1
Значение метрики	$Q = 1 \times 5 + 2 \times 0 + 3 \times 1 + 0,5 \times 1 = 8,5$	

Полученное значение $Q = 8,5$ говорит о низкой сложности метода. Метод использует небольшое количество переменных, не изменяет состояние объекта и выполняет одну конкретную задачу – расчет базового обмена веществ. Наличие управляющей переменной связано только с выбором формулы в зависимости от пола пользователя, что является необходимой частью алгоритма.

5.3 Вывод по результатам контроля качества

В результате анализа качества программного кода были рассчитаны метрики Холстеда и Чепина для метода calculateBmr класса BmrCalculator. Данный метод был выбран для анализа, так как он относится к предметной логике приложения и используется при расчете индивидуальной дневной нормы калорий пользователя.

Метрика Холстеда показала небольшой объем и невысокую вычислительную сложность метода. Это означает, что код является компактным и понятным для сопровождения. Метрика Чепина показала, что метод использует в основном входные данные и одну управляющую переменную, не содержит модифицируемых переменных и не изменяет состояние приложения.

6 Расчет бюджета разработки приложения

6.1 Планирование разработки и определение трудоемкости

Расчет бюджета разработки необходим для оценки затрат, связанных с созданием мобильного приложения. В рамках данной выпускной квалификационной работы разрабатывается Android-приложение, включающее локальную базу данных, пользовательский интерфейс, модуль авторизации, дневник питания, расчет дневной нормы калорий и взаимодействие с внешним сервисом GigaChat API.

Для планирования сроков выполнения разработки была построена диаграмма Ганта. Данная диаграмма позволяет наглядно представить последовательность выполнения этапов работы, сроки начала и окончания задач, а также общую продолжительность разработки приложения. Диаграмма Ганта разработки приложения NutritionDiary представлена на рисунке В.10 приложения В.

Для оценки трудоемкости разработки были выделены основные этапы выполнения работы. К ним относятся анализ предметной области, проектирование приложения, реализация программных модулей, тестирование, отладка и оформление пояснительной записки. Каждый этап требует определенных трудозатрат, выраженных в человеко-часах.

Оценка трудоемкости разработки представлена в таблице 12.

Таблица 12 – Оценка трудоемкости разработки приложения

Этап работы	Трудоемкость, чел.-ч
Анализ предметной области и требований	32
Проектирование архитектуры и базы данных	40
Проектирование пользовательского интерфейса и алгоритмов	36

Продолжение таблицы 12

Реализация регистрации, авторизации и анкеты пользователя	48
Реализация локальной базы данных и дневника питания	56
Реализация ИИ-диетолога и взаимодействия с GigaChat API	64
Реализация сохранения ИИ-рациона в дневник питания	24
Отладка и тестирование приложения	40
Оформление пояснительной записки	60
Итого	400

Суммарная трудоемкость разработки составила 400 человеко-часов. Данное значение учитывает как непосредственную разработку программного кода, так и проектирование, тестирование, отладку и оформление материалов выпускной квалификационной работы.

6.2 Расчет заработной платы разработчика

Для расчета затрат на оплату труда необходимо определить часовую ставку разработчика. В расчете принимается условная среднемесячная заработная плата разработчика мобильных приложений, равная 70000 руб. Количество рабочих часов в месяце принимается равным 168.

Часовая ставка рассчитывается по формуле (10):

$$Сч = Зм / Тм, \quad (10)$$

где Сч – часовая ставка разработчика, руб./ч;

Зм – месячная заработная плата, руб.;

Тм – количество рабочих часов в месяце, ч.

$$Сч = 70000 / 168 = 416,67 \text{ руб./ч.},$$

Основная заработная плата рассчитывается по формуле (11):

$$Зосн = Сч \cdot Т, \quad (11)$$

где $Z_{осн}$ – основная заработная плата, руб.;

$Сч$ – часовая ставка разработчика, руб./ч;

T – трудоемкость разработки, чел.-ч.

$$Z_{осн} = 416,67 \cdot 400 = 166668 \text{ руб.},$$

Дополнительная заработная плата принимается в размере 10 % от основной заработной платы (12):

$$Z_{доп} = Z_{осн} \cdot 0,1, \quad (12)$$

где $Z_{доп}$ – дополнительная заработная плата, руб.;

$Z_{осн}$ – основная заработная плата, руб.

$$Z_{доп} = 166668 \cdot 0,1 = 16667 \text{ руб.},$$

Фонд оплаты труда определяется как сумма основной и дополнительной заработной платы (13):

$$\Phi OT = Z_{осн} + Z_{доп}, \quad (13)$$

где ΦOT – фонд оплаты труда, руб.;

$Z_{осн}$ – основная заработная плата, руб.;

$Z_{доп}$ – дополнительная заработная плата, руб.

$$\Phi OT = 166668 + 16667 = 183335 \text{ руб.},$$

Расчет фонда оплаты труда представлен в таблице 13.

Таблица 13 – Расчет фонда оплаты труда

Показатель	Значение
Месячная заработная плата, руб.	70000
Количество рабочих часов в месяце, ч	168
Часовая ставка, руб./ч	416,67
Трудоемкость разработки, чел.-ч	400
Основная заработная плата, руб.	166668
Дополнительная заработная плата, руб.	16667
Фонд оплаты труда, руб.	183335

Таким образом, фонд оплаты труда разработчика мобильного приложения составляет 183335 рублей.

6.3 Расчет страховых взносов, накладных расходов и итогового бюджета разработки

После определения фонда оплаты труда необходимо рассчитать страховые взносы и накладные расходы, связанные с разработкой мобильного приложения.

Страховые взносы принимаются в размере 30 % от фонда оплаты труда и рассчитываются по формуле (14):

$$Cв = ФОТ \cdot 0,3, \quad (14)$$

где $Cв$ – страховые взносы, руб.;

$ФОТ$ – фонд оплаты труда, руб.

$$Cв = 183335 \cdot 0,3 = 55001 \text{ руб.}$$

Накладные расходы включают затраты, связанные с использованием рабочего места, электроэнергии, программного обеспечения, сети Интернет и других ресурсов, необходимых для выполнения разработки. В рамках расчета накладные расходы принимаются в размере 20 % от фонда оплаты труда.

Накладные расходы рассчитываются по формуле (15):

$$Нр = ФОТ \cdot 0,2, \quad (15)$$

где $Нр$ – накладные расходы, руб.;

$ФОТ$ – фонд оплаты труда, руб.

$$Нр = 183335 \cdot 0,2 = 36667 \text{ руб.,}$$

Итоговый бюджет разработки определяется как сумма фонда оплаты труда, страховых взносов и накладных расходов (16):

$$Б = ФОТ + Cв + Нр, \quad (16)$$

где $Б$ – итоговый бюджет разработки, руб.;

$ФОТ$ – фонд оплаты труда, руб.;

$Cв$ – страховые взносы, руб.;

$Нр$ – накладные расходы, руб.

$$Б = 183335 + 55001 + 36667 = 275003 \text{ руб.,}$$

Итоговый расчет бюджета разработки мобильного приложения представлен в таблице 14.

Таблица 14 – Итоговый бюджет разработки приложения

Статья затрат	Сумма, руб.
Фонд оплаты труда	183335
Страховые взносы	55001
Накладные расходы	36667
Итого	275003

Таким образом, общий бюджет разработки мобильного приложения «Электронный журнал питания с подбором блюд» составляет 275003 руб.

Заключение

В ходе выполнения выпускной квалификационной работы было разработано мобильное приложение «Электронный журнал питания с подбором блюд», предназначенное для ведения дневника питания, расчета индивидуальной дневной нормы калорий и формирования персонализированного рациона с использованием ИИ-диетолога.

На первом этапе был выполнен анализ предметной области. Были рассмотрены особенности контроля питания, ведения дневника калорий, учета белков, жиров и углеводов, а также необходимость персонализации рекомендаций с учетом возраста, пола, роста, веса, уровня физической активности, цели питания и аллергий пользователя. Также был проведен анализ существующих программных аналогов, таких как MyFitnessPal, YAZIO, FatSecret и GigaChat. По результатам анализа были выявлены их достоинства и недостатки, а также обоснована необходимость разработки приложения, объединяющего функции классического дневника питания и интеллектуального помощника.

На этапе проектирования были определены функциональные и нефункциональные требования к приложению, разработана архитектура программного продукта, спроектирована структура локальной базы данных, пользовательский интерфейс и основные алгоритмы работы системы. Особое внимание было уделено алгоритмам расчета дневной нормы калорий, добавления блюд в дневник питания, формирования рациона с использованием ИИ-диетолога и сохранения сгенерированного рациона в дневник питания.

На этапе реализации было создано Android-приложение с использованием языка программирования Kotlin и фреймворка Jetpack Compose. Для локального хранения данных была применена библиотека Room, обеспечивающая работу с базой данных SQLite. Для сетевого взаимодействия с ИИ-диетологом использовались Retrofit, OkHttp, Gson и GigaChat API. В приложении были реализованы регистрация и авторизация

пользователей, заполнение анкеты профиля, расчет индивидуальной дневной нормы калорий, ведение дневника питания, добавление и удаление блюд, отображение суммарной калорийности и количества белков, жиров и углеводов за день, а также генерация, корректировка и сохранение ИИ-рациона.

Для проверки корректности работы приложения были проведены функциональное и модульное тестирование. В рамках функционального тестирования была проверена анкета пользователя, включая валидацию имени, возраста, роста, веса, выбора пола, уровня активности и цели питания. Для модульного тестирования был использован JUnit 4, с помощью которого проверялась логика работы класса `ProfileValidationService`.

Также был выполнен контроль качества программного кода. Для анализа был выбран метод `calculateBmr` класса `BmrCalculator`, отвечающий за расчет базового обмена веществ. Оценка была проведена с использованием метрик Холстеда и Чепина. Полученные значения показали, что рассматриваемый фрагмент кода соответствует требованиям к сопровождаемости программного обеспечения.

В экономической части был выполнен расчет бюджета разработки приложения. Была определена трудоемкость разработки, рассчитан фонд оплаты труда, страховые взносы и накладные расходы. Итоговый бюджет разработки мобильного приложения составил 275003 руб.

Таким образом, цель выпускной квалификационной работы была достигнута. В результате работы было спроектировано и разработано мобильное Android-приложение, позволяющее пользователю вести учет питания, рассчитывать дневную норму калорий и получать персонализированный рацион с использованием ИИ-диетолога. Разработанное приложение объединяет функции дневника питания, локального хранения данных и интеллектуального подбора блюд, что делает его удобным инструментом для контроля питания и формирования индивидуального рациона.

Список использованных источников

1. MyFitnessPal. Официальный сайт приложения для контроля питания [Электронный ресурс]. URL: <https://www.myfitnesspal.com/ru> (дата обращения: 15.05.2026).
2. YAZIO. Официальный сайт приложения для подсчета калорий и ведения дневника питания [Электронный ресурс]. URL: <https://www.yazio.com/ru> (дата обращения: 15.05.2026).
3. FatSecret. Официальный сайт приложения для подсчета калорий [Электронный ресурс]. URL: <https://www.fatsecret.ru> (дата обращения: 15.05.2026).
4. GigaChat. Документация для разработчиков [Электронный ресурс]. URL: <https://developers.sber.ru/docs/ru/gigachat/overview> (дата обращения: 15.05.2026).
5. Kotlin. Официальная документация языка программирования Kotlin [Электронный ресурс]. URL: <https://kotlinlang.org/docs/home.html> (дата обращения: 16.05.2026).
6. Android Developers. Официальная документация по разработке Android-приложений [Электронный ресурс]. URL: <https://developer.android.com/docs> (дата обращения: 16.05.2026).
7. Android Developers. Документация по Jetpack Compose [Электронный ресурс]. URL: <https://developer.android.com/compose> (дата обращения: 16.05.2026).
8. Android Developers. Документация по библиотеке Room [Электронный ресурс]. URL: <https://developer.android.com/training/data-storage/room> (дата обращения: 16.05.2026).
9. SQLite. Официальная документация системы управления базами данных SQLite [Электронный ресурс]. URL: <https://www.sqlite.org/docs.html> (дата обращения: 16.05.2026).

10. Retrofit. Официальная документация библиотеки Retrofit [Электронный ресурс]. URL: <https://square.github.io/retrofit/> (дата обращения: 16.05.2026).
11. OkHttp. Официальная документация библиотеки OkHttp [Электронный ресурс]. URL: <https://square.github.io/okhttp/> (дата обращения: 16.05.2026).
12. Gson. Официальный репозиторий библиотеки Gson [Электронный ресурс]. URL: <https://github.com/google/gson> (дата обращения: 16.05.2026).
13. Gradle. Официальная документация системы сборки Gradle [Электронный ресурс]. URL: <https://docs.gradle.org/current/userguide/userguide.html> (дата обращения: 17.05.2026).
14. JUnit 4. Официальная документация фреймворка модульного тестирования [Электронный ресурс]. URL: <https://junit.org/junit4/> (дата обращения: 17.05.2026).
15. Коннолли, Т. Базы данных. Проектирование, реализация и сопровождение. Теория и практика / Т. Коннолли, К. Бегг. – 3-е изд. – М. : Вильямс, 2003. – 1440 с.
16. ГОСТ 19.701–90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. – М. : Стандартиформ, 2010
17. Гамма, Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. – СПб. : Питер, 2020. – 368 с.
18. Fowler, M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. – 3rd ed. – Boston : Addison-Wesley, 2004. – 208 p.
19. Мартин, Р. Чистый код. Создание, анализ и рефакторинг / Р. Мартин. – СПб. : Питер, 2021. – 464 с.
20. Halstead, M. H. Elements of Software Science. – New York : Elsevier, 1977. – 127 p.

Приложение А
(справочное)
Глоссарий

Перечень сокращений и обозначений

В данной работе применяются следующие сокращения и обозначения:

Android – Операционная система для мобильных устройств, на которой функционирует разработанное приложение.

Kotlin – Язык программирования, использованный для реализации мобильного приложения.

Jetpack Compose – Декларативный фреймворк для разработки пользовательского интерфейса Android-приложений.

Room – Библиотека Android Jetpack, предназначенная для работы с локальной базой данных SQLite.

SQLite – Встраиваемая реляционная система управления базами данных, используемая для локального хранения данных приложения.

Retrofit – Библиотека для организации сетевого взаимодействия с REST API.

OkHttp – HTTP-клиент, используемый для выполнения сетевых запросов.

Gson – Библиотека для сериализации и десериализации данных в формате JSON.

GigaChat API – Программный интерфейс сервиса GigaChat, используемый для взаимодействия с ИИ-диетологом.

API – Программный интерфейс приложения, обеспечивающий взаимодействие между программными компонентами.

ИИ – Искусственный интеллект.

БД – База данных.

КБЖУ – Калорийность, белки, жиры и углеводы.

BMR – Базовый обмен веществ, то есть количество энергии, необходимое организму в состоянии покоя.

MVVM – Архитектурный паттерн, разделяющий приложение на модель, представление и модель представления.

DAO – Объект доступа к данным, используемый для выполнения операций с базой данных.

Entity – Класс-сущность, описывающий структуру таблицы базы данных.

UI – Пользовательский интерфейс приложения.

Unit-тест – Модульный тест, проверяющий работу отдельной функции или класса программы.

JUnit – Фреймворк для модульного тестирования программного кода.

Приложение Б
(рекомендуемое)
Атрибуты сущностей базы данных

Таблица Б.1 – Атрибуты сущности UserEntity

Атрибут	Тип данных	Описание
Id	Int	Уникальный идентификатор пользователя
username	String	Имя пользователя для входа в приложение
password	String	Пароль пользователя
createdAt	Long	Дата и время создания учетной записи

Таблица Б.2 – Атрибуты сущности UserProfileEntity

Атрибут	Тип данных	Описание
Id	Int	Уникальный идентификатор профиля
userId	Int	Идентификатор пользователя
Name	String	Имя пользователя
Age	Int	Возраст пользователя
Gender	String	Пол пользователя
heightCm	Int	Рост пользователя в сантиметрах
weightKg	Double	Вес пользователя в килограммах
activityLevel	String	Уровень физической активности
Goal	String	Цель питания
Allergies	String	Список аллергий пользователя
dailyCalories	Int	Рассчитанная дневная норма калорий

Таблица Б.3 – Атрибуты сущности DishEntity

Атрибут	Тип данных	Описание
Id	Int	Уникальный идентификатор блюда
Name	String	Наименование блюда или продукта
Calories	Double	Калорийность на 100 г продукта
Proteins	Double	Количество белков на 100 г продукта
Fats	Double	Количество жиров на 100 г продукта
Carbs	Double	Количество углеводов на 100 г продукта
Allergens	String	Список аллергенов, связанных с блюдом

Таблица Б.4 – Атрибуты сущности DiaryEntryEntity

Атрибут	Тип данных	Описание
Id	Int	Уникальный идентификатор записи дневника
userId	Int	Идентификатор пользователя
dishId	Int	Идентификатор блюда
date	String	Дата записи в дневнике питания
mealType	String	Тип приема пищи
weightGrams	Double	Масса порции в граммах
calories	Double	Калорийность порции
proteins	Double	Количество белков в порции
fats	Double	Количество жиров в порции
carbs	Double	Количество углеводов в порции

Таблица Б.5 – Атрибуты сущности ChatMessageEntity

Атрибут	Тип данных	Описание
id	Int	Уникальный идентификатор сообщения
userId	Int	Идентификатор пользователя
role	String	Роль отправителя сообщения
content	String	Текст сообщения
timestamp	Long	Дата и время отправки сообщения

Таблица Б.6 – Атрибуты сущности AiMealPlanEntity

Атрибут	Тип данных	Описание
id	Int	Уникальный идентификатор ИИ-рациона
userId	Int	Идентификатор пользователя
date	String	Дата формирования или сохранения рациона
mealType	String	Тип приема пищи
dishName	String	Наименование блюда
weightGrams	Double	Масса продукта или блюда в граммах
calories	Double	Калорийность
proteins	Double	Количество белков
fats	Double	Количество жиров
carbs	Double	Количество углеводов
isSavedToDiary	Boolean	Признак сохранения рациона в дневник питания

Приложение В

(обязательное)

UML-диаграммы приложения NutritionDiary

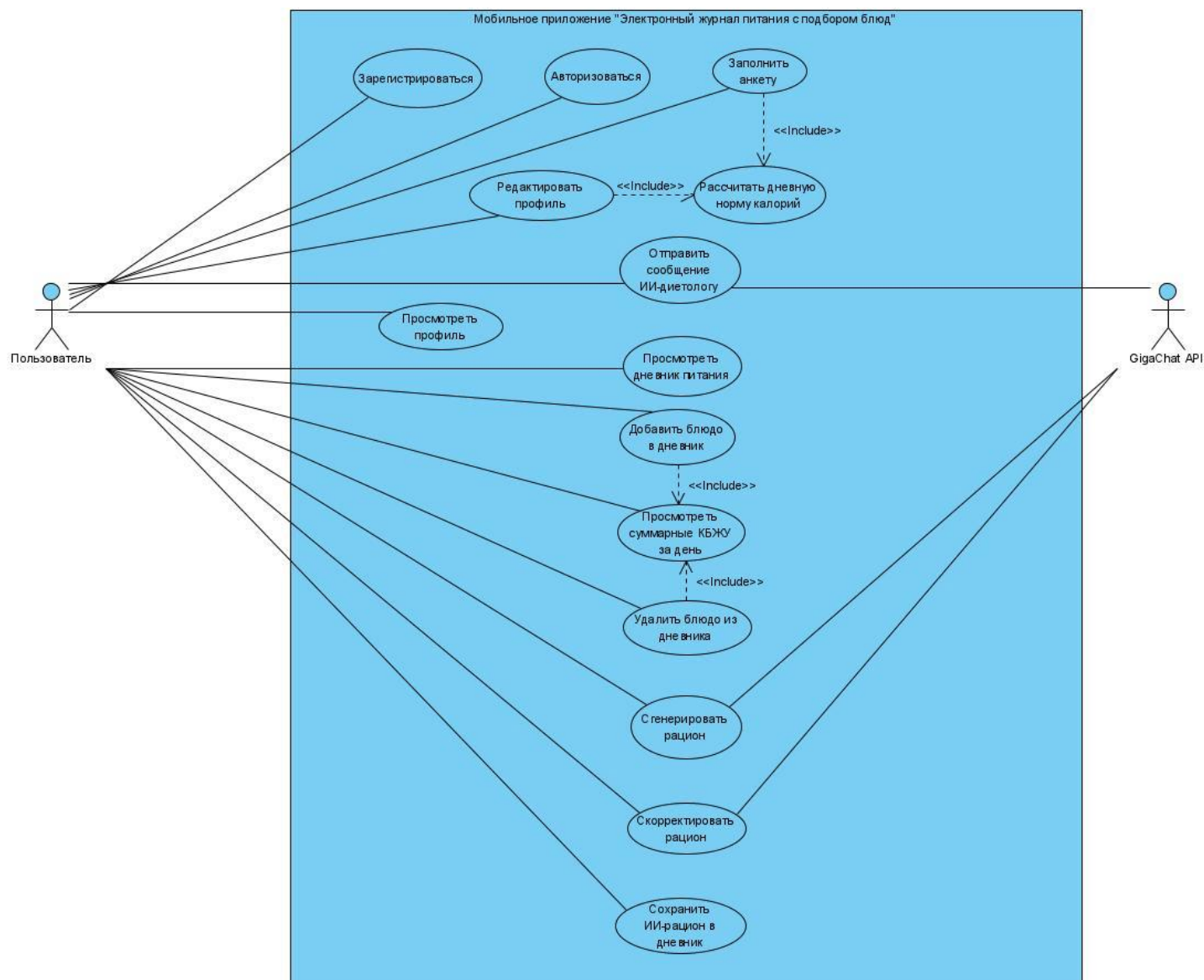


Рисунок В.1 – Диаграмма вариантов использования приложения

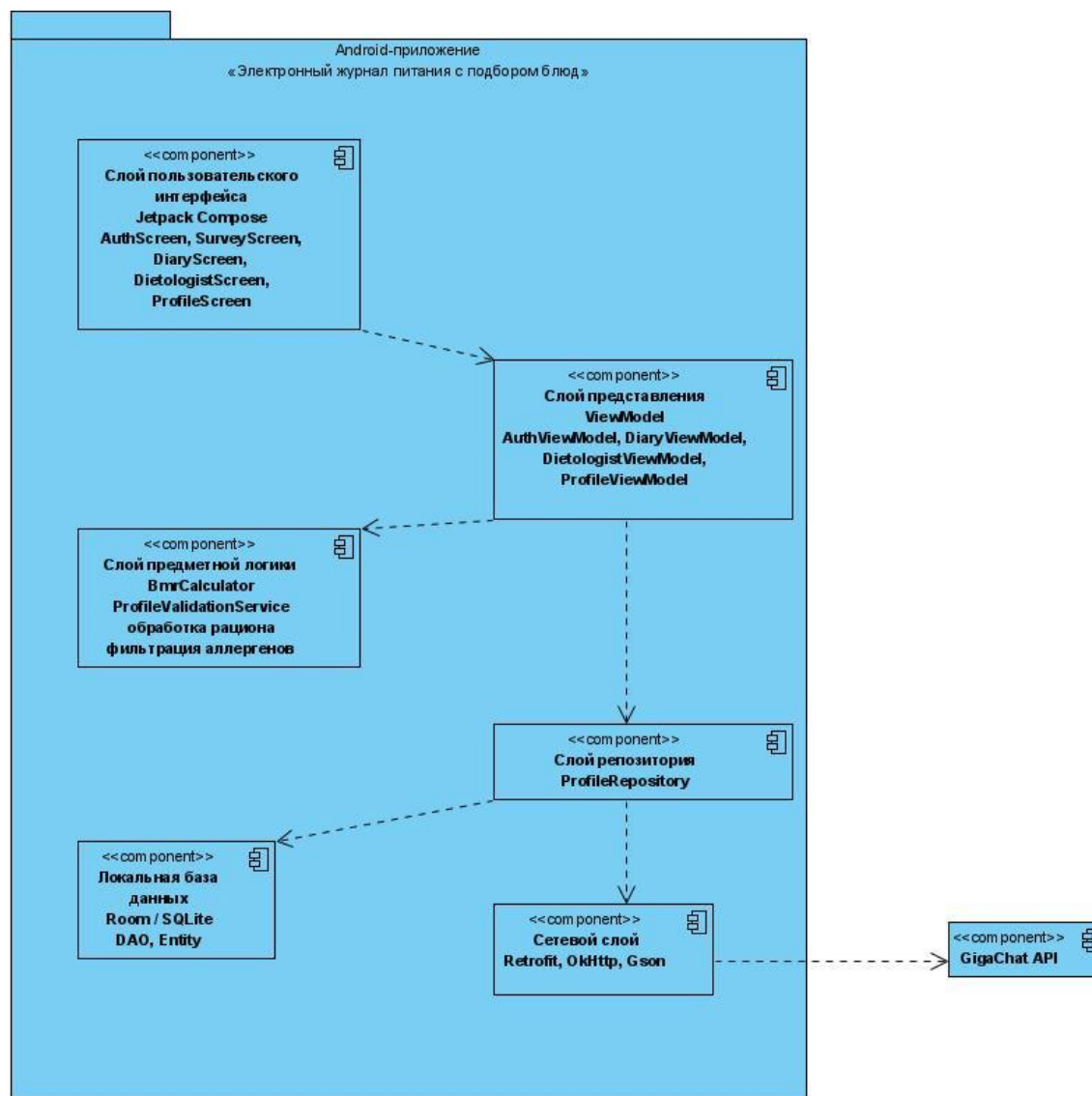


Рисунок В.2 – Архитектура мобильного приложения

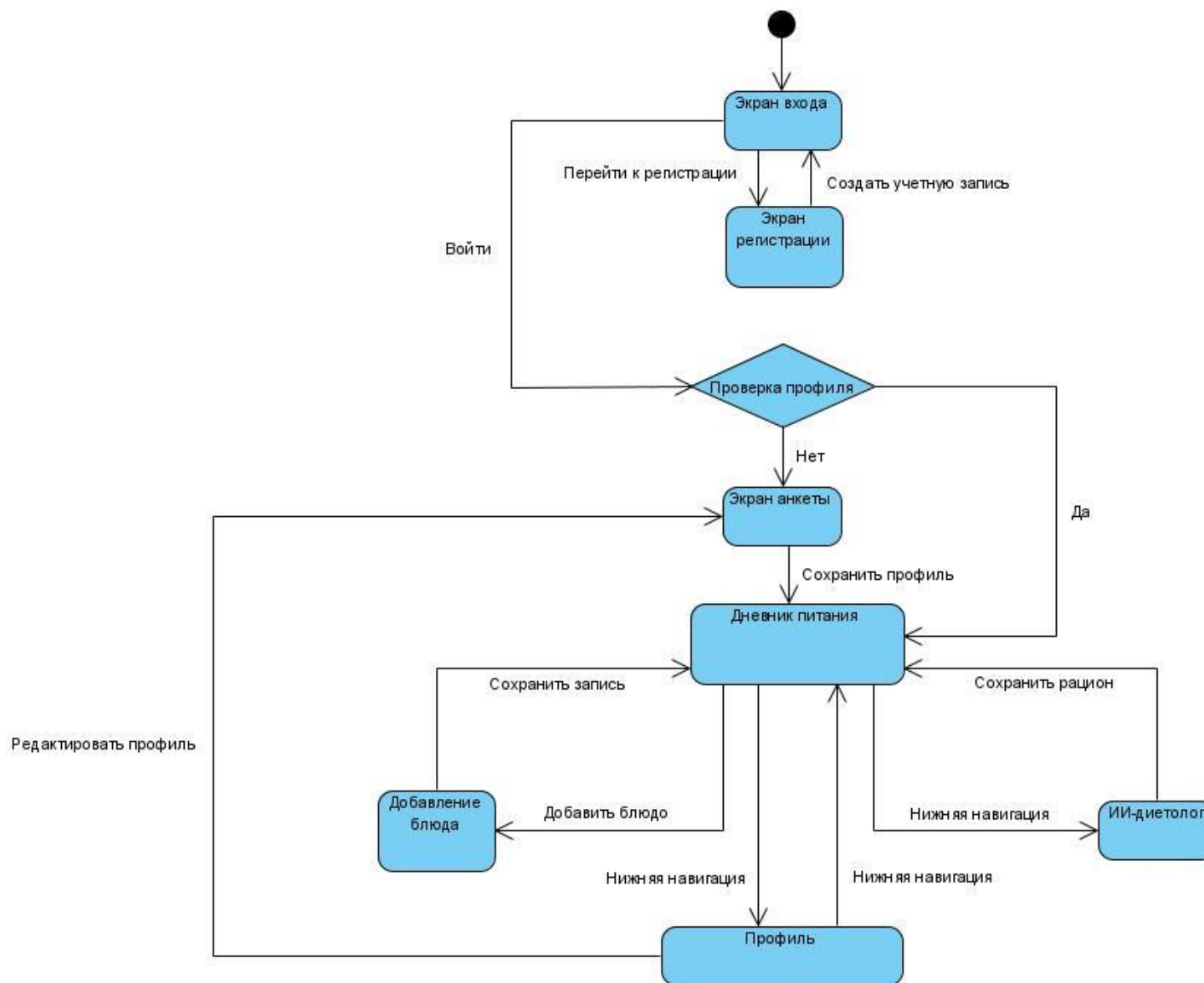


Рисунок В.3 – Структура пользовательского интерфейса приложения NutritionDiary

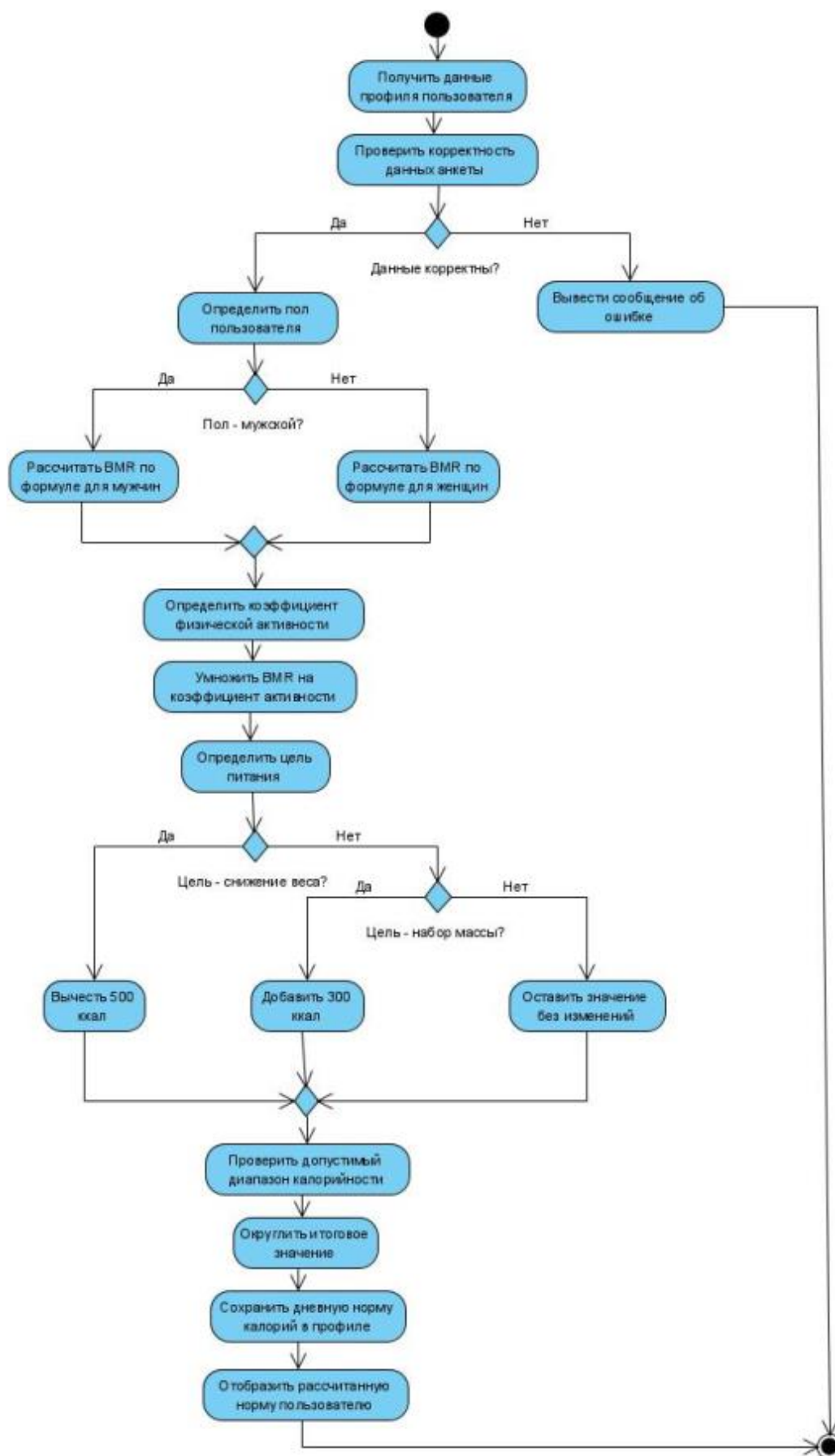


Рисунок В.4 – Алгоритм расчета индивидуальной дневной нормы калорий

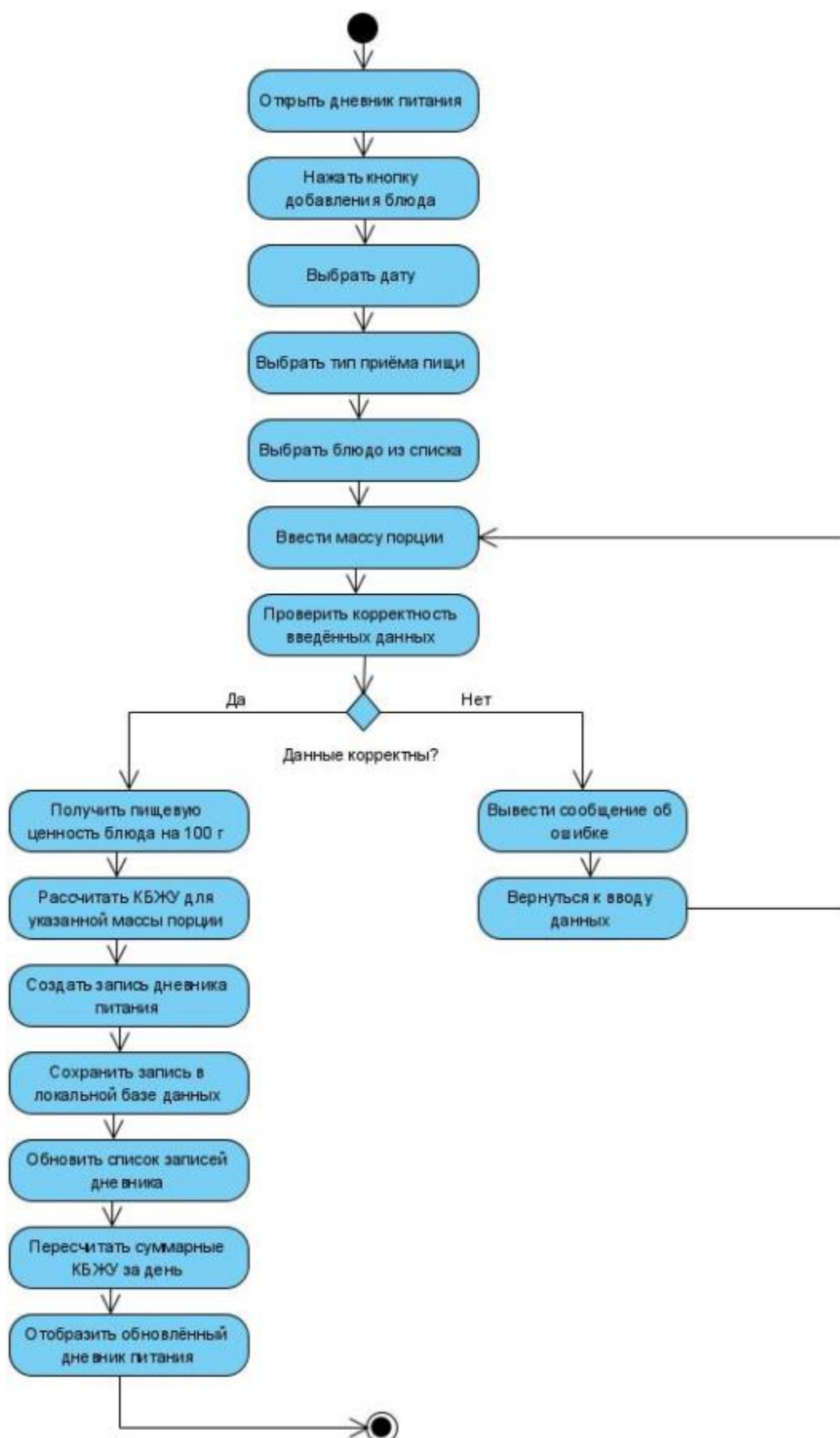


Рисунок В.5 – Алгоритм добавления блюда в дневник питания

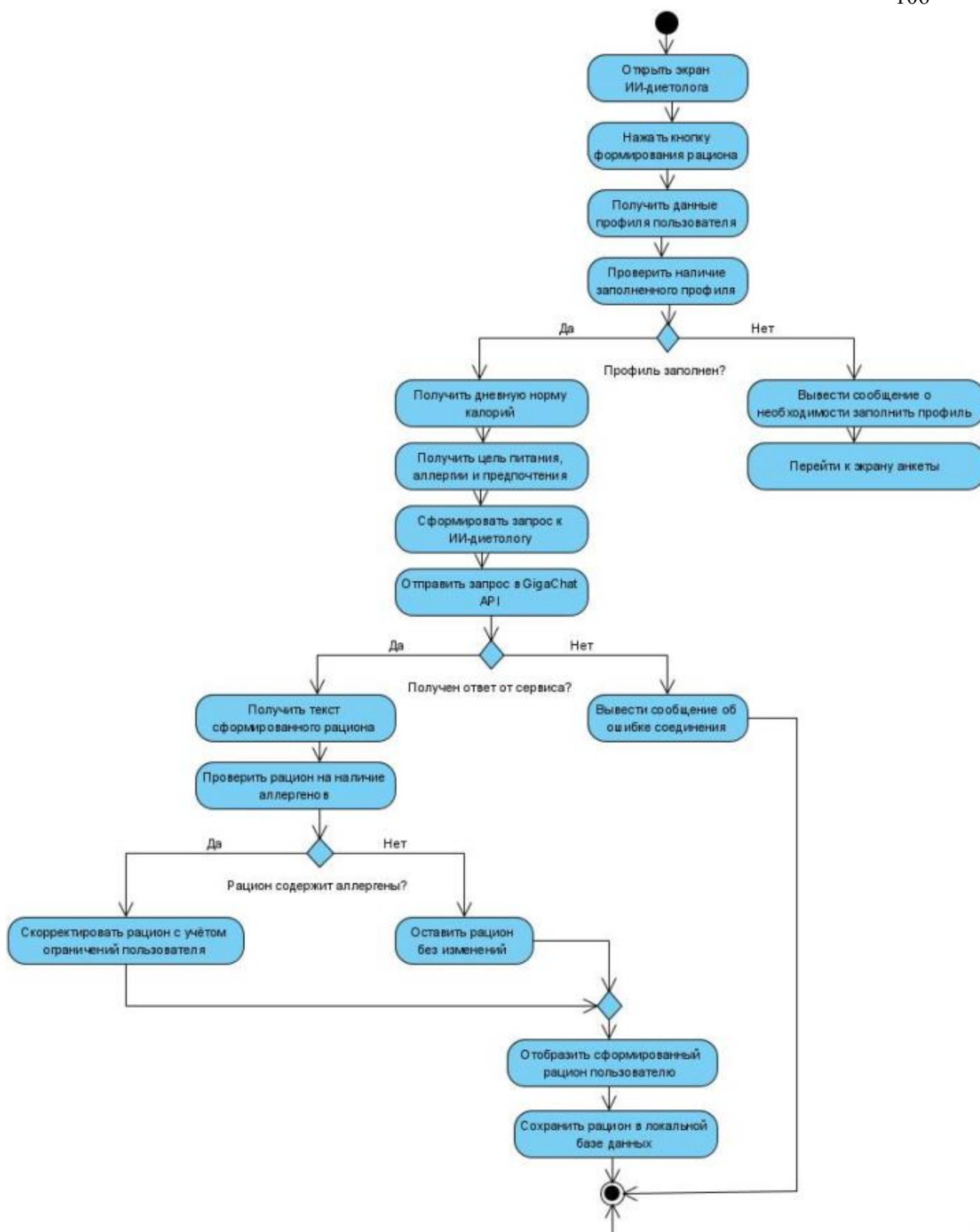


Рисунок В.6 – Алгоритм формирования рациона с использованием ИИ-диетолога

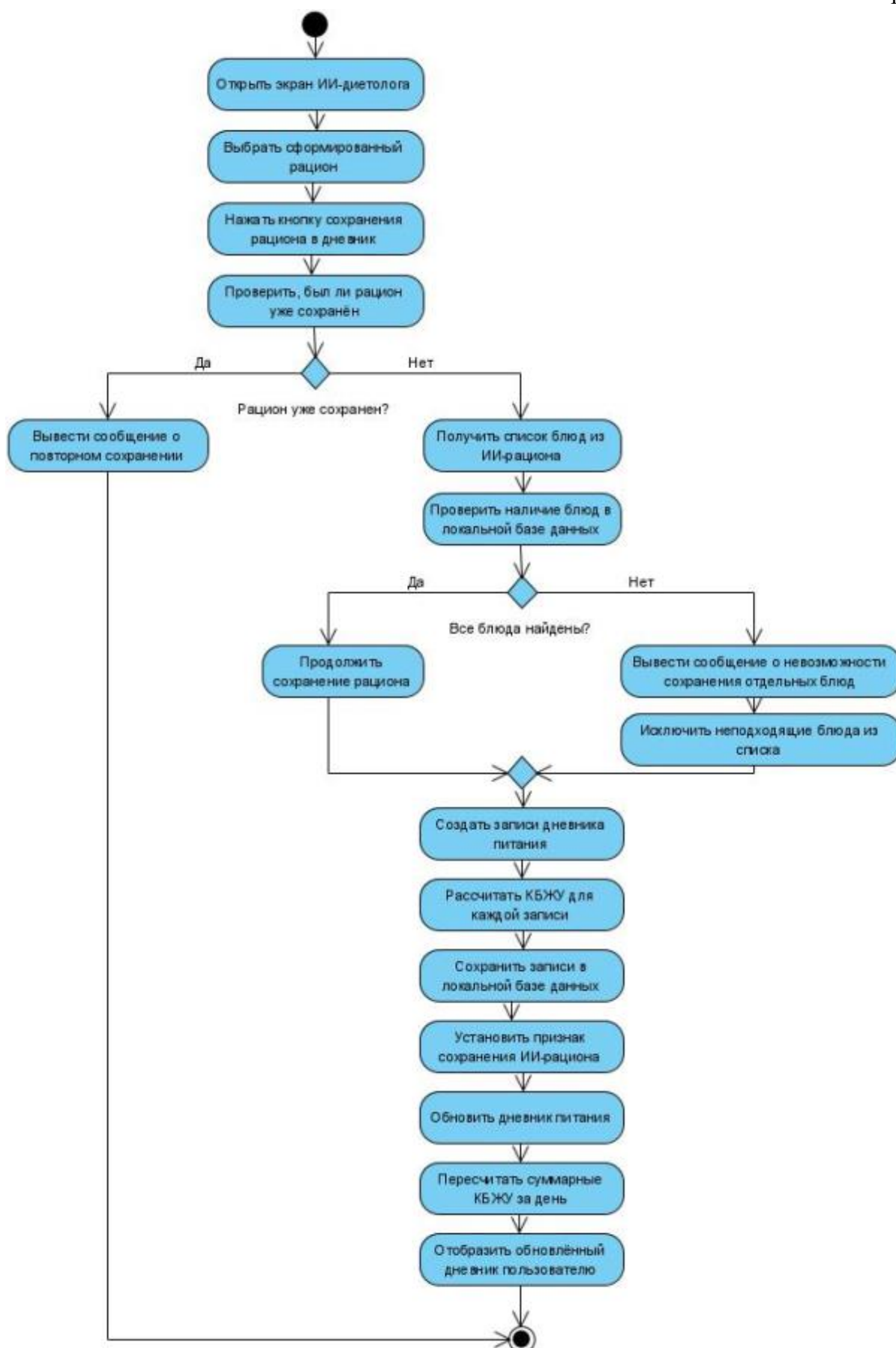


Рисунок В.7 – Алгоритм сохранения ИИ-рациона в дневник питания

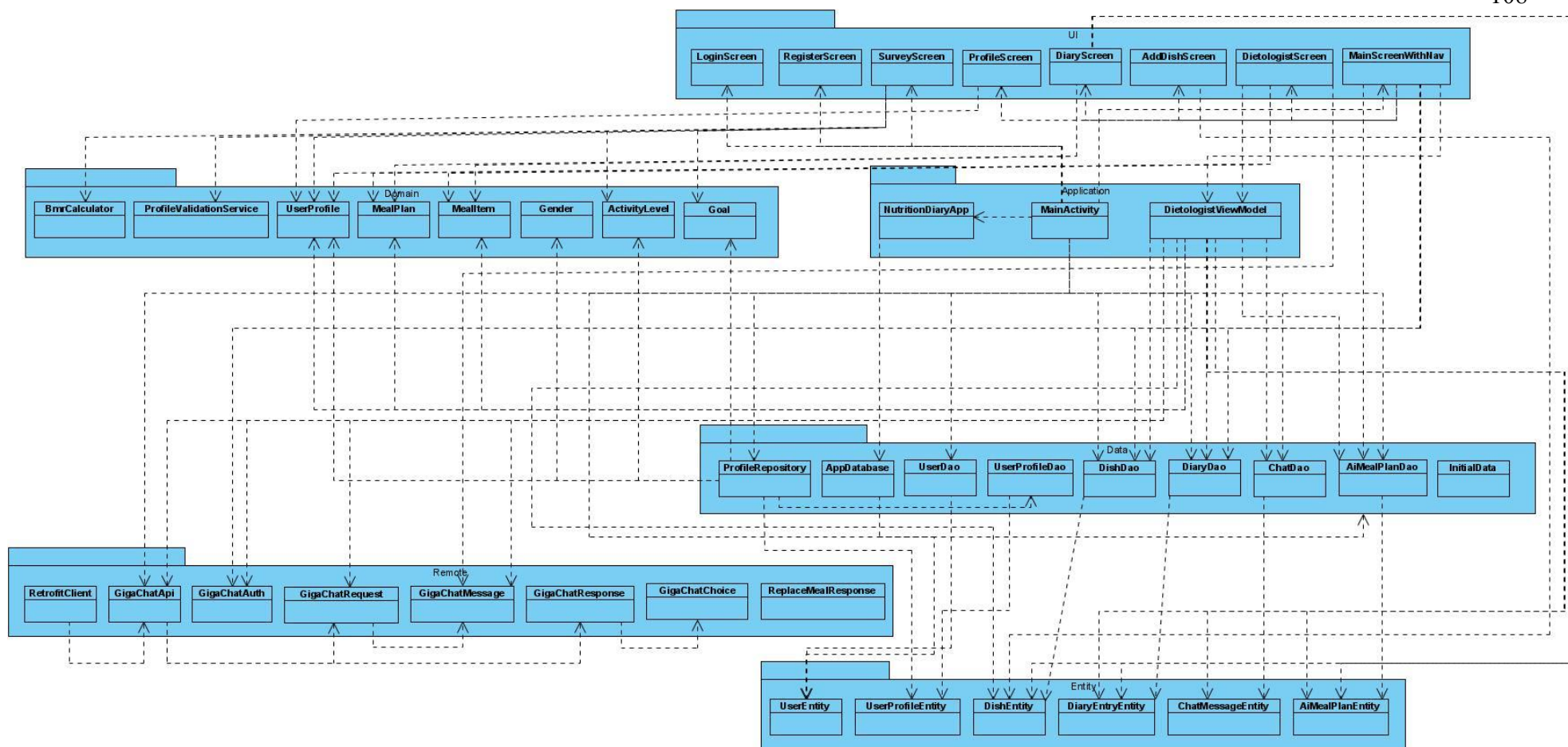


Рисунок В.8 – UML-диаграмма классов приложения NutritionDiary

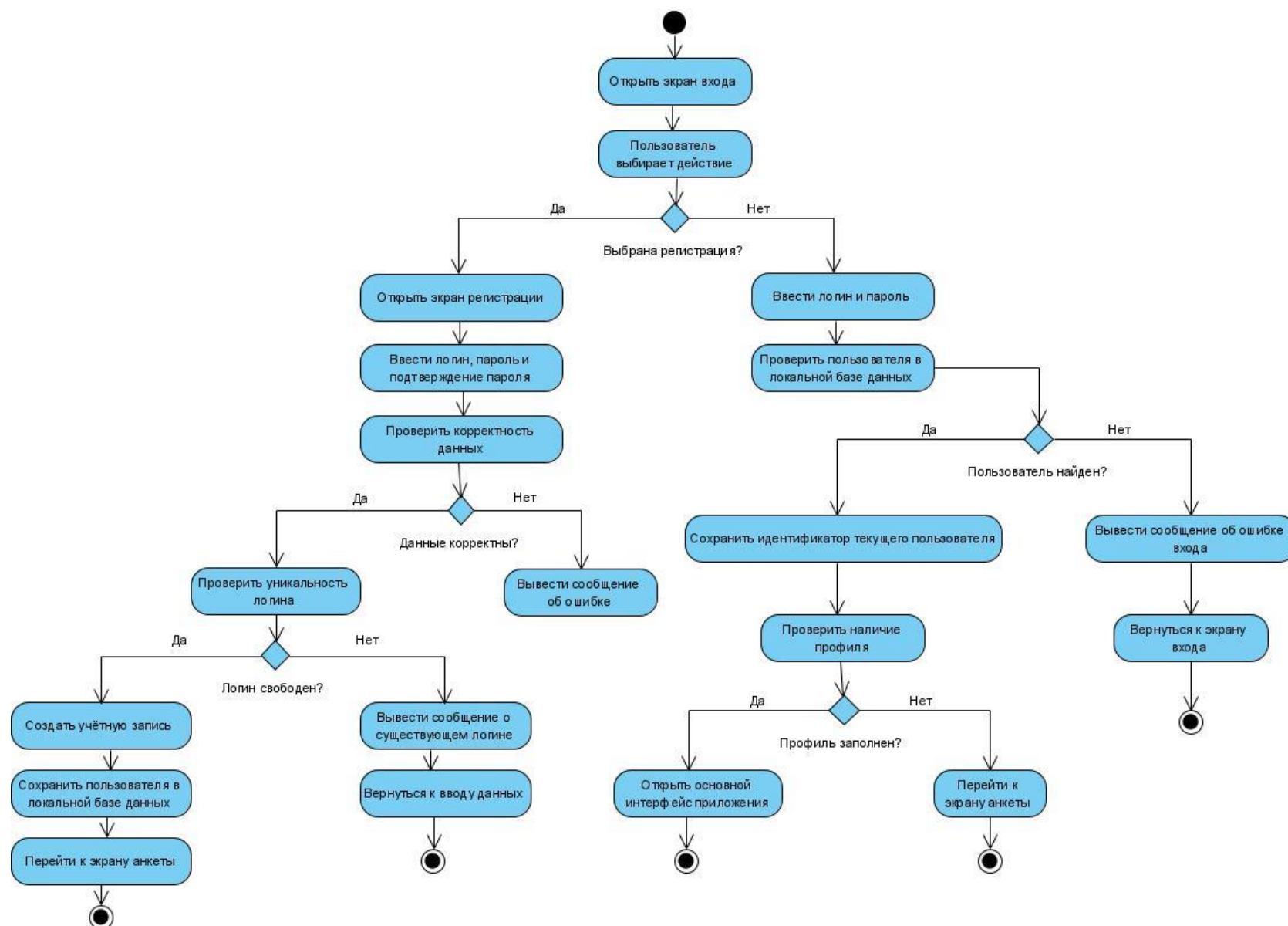


Рисунок В.9 – Алгоритм регистрации и авторизации пользователя

Диаграмма Ганта

Premium Asset

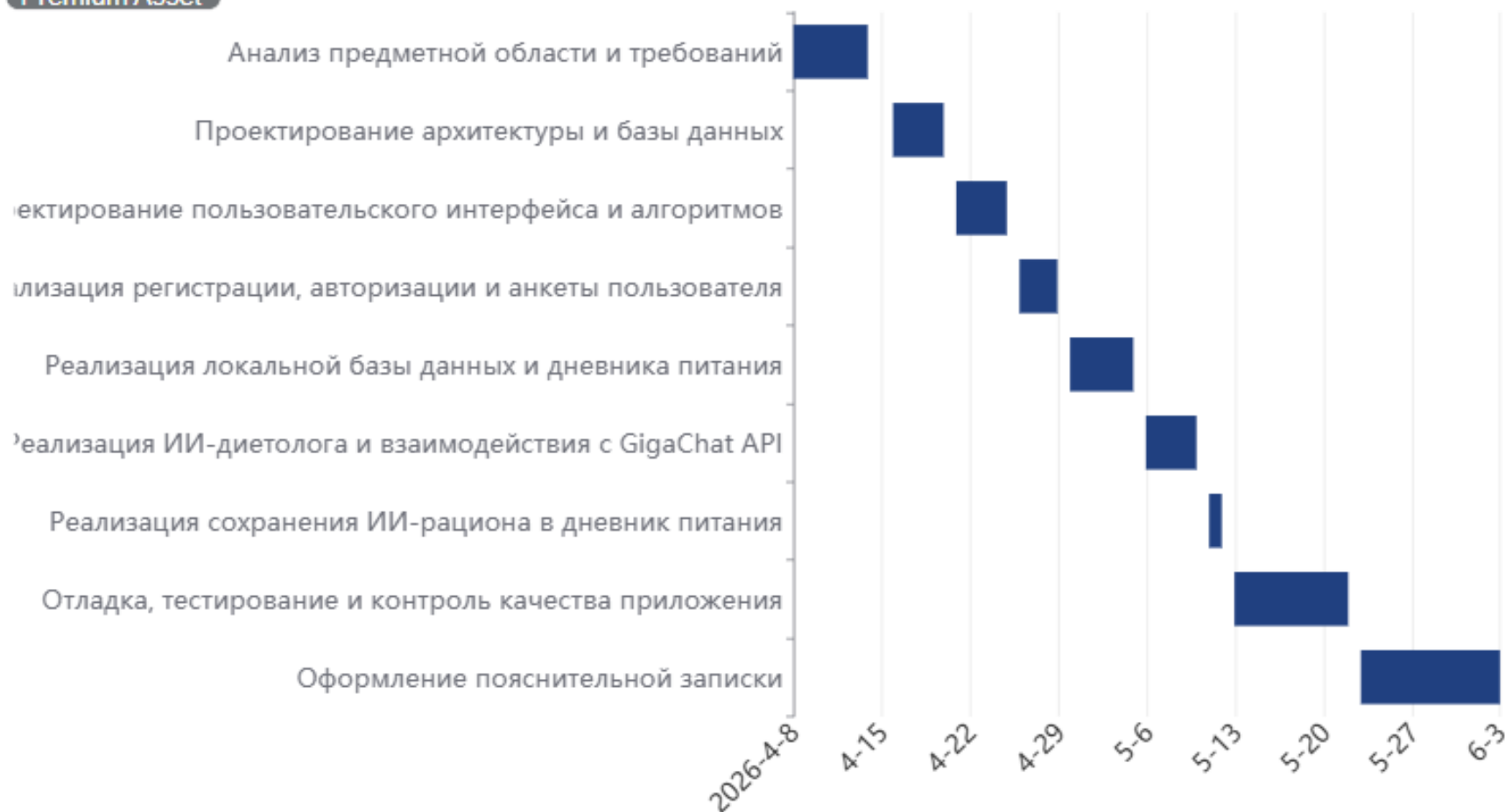


Рисунок В.10 – Диаграмма Ганта приложения NutritionDiary

Приложение Г

(обязательное)

**Ключевые фрагменты программной логики, локальной базы данных и
сетевого взаимодействия**

Полный листинг программы размещен в публичном репозитории GitHub по адресу: URL: <https://github.com/mishajordan/NutritionDiary> (дата обращения: 05.06.2026). Ниже приведены фрагменты наиболее значимых классов.

Листинг Г.1 – Файл BmrCalculator.kt

```
package com.app.nutritiondiary.domain

import com.app.nutritiondiary.domain.model.Gender
import com.app.nutritiondiary.domain.model.UserProfile

object BmrCalculator {
    fun calculateBmr(profile: UserProfile): Double {
        val base = 10 * profile.weightKg + 6.25 *
profile.heightCm - 5 * profile.age
        return when (profile.gender) {
            Gender.MALE -> base + 5
            Gender.FEMALE -> base - 161
        }
    }

    fun calculateTdee(profile: UserProfile): Double {
        return calculateBmr(profile) *
profile.activityLevel.multiplier
    }

    fun calculateTargetCalories(profile: UserProfile): Double {
        val tdee = calculateTdee(profile)
        val rawTarget = tdee + profile.goal.calorieDelta
        // Ограничиваем разумным диапазоном (800-5000 ккал)
        return rawTarget.coerceIn(800.0, 5000.0)
    }
}
```

Листинг Г.2 – Файл ProfileValidationService.kt

```
sealed class ValidationResult {
    object Ok : ValidationResult()
    data class Error(val message: String) : ValidationResult()
}

class ProfileValidationService {

    fun validateName(name: String): ValidationResult {
        if (name.isBlank()) return
ValidationResult.Error("Введите имя")
        if (name.length < 2) return ValidationResult.Error("Имя
должно содержать хотя бы 2 символа")
        if (name.length > 50) return ValidationResult.Error("Имя
не должно превышать 50 символов")
    }
}
```

```

        if (!name.matches(Regex("^[a-zA-Za-яА-Я\\s]+$")) return
ValidationResult.Error("Имя содержит недопустимые символы")
        return ValidationResult.Ok
    }

    fun validateAge(age: String): ValidationResult {
        val ageVal = age.toIntOrNull() ?: return
ValidationResult.Error("Введите корректный возраст")
        if (ageVal !in 1..120) return
ValidationResult.Error("Возраст должен быть от 1 до 120 лет")
        return ValidationResult.Ok
    }

    fun validateHeight(height: String): ValidationResult {
        val heightVal = height.toIntOrNull() ?: return
ValidationResult.Error("Введите корректный рост")
        if (heightVal !in 50..250) return
ValidationResult.Error("Рост должен быть от 50 до 250 см")
        return ValidationResult.Ok
    }

    fun validateWeight(weight: String): ValidationResult {
        val weightVal = weight.toDoubleOrNull() ?: return
ValidationResult.Error("Введите корректный вес")
        if (weightVal !in 10.0..300.0) return
ValidationResult.Error("Вес должен быть от 10 до 300 кг")
        return ValidationResult.Ok
    }

    fun checkSelections(gender: Gender?, activity:
ActivityLevel?, goal: Goal?): ValidationResult {
        if (gender == null) return
ValidationResult.Error("Выберите пол")
        if (activity == null) return
ValidationResult.Error("Выберите уровень активности")
        if (goal == null) return
ValidationResult.Error("Выберите цель")
        return ValidationResult.Ok
    }
}

```

Листинг Г.3 – Файл AppDataBase.kt

```

@Database(
    entities = [
        UserEntity::class,
        UserProfileEntity::class,
        DiaryEntryEntity::class,
        DishEntity::class,
        ChatMessageEntity::class,
        AiMealPlanEntity::class
    ],

```

```

        version = 6,
        exportSchema = false
    )
    abstract class AppDatabase : RoomDatabase() {
        abstract fun userDao(): UserDao
        abstract fun userProfileDao(): UserProfileDao
        abstract fun diaryDao(): DiaryDao
        abstract fun dishDao(): DishDao
        abstract fun chatDao(): ChatDao
        abstract fun aiMealPlanDao(): AiMealPlanDao

        companion object {
            @Volatile
            private var INSTANCE: AppDatabase? = null

            fun getDatabase(context: Context): AppDatabase {
                return INSTANCE ?: synchronized(this) {
                    val instance = Room.databaseBuilder(
                        context.applicationContext,
                        AppDatabase::class.java,
                        "nutrition_diary_db"
                    )
                        .addCallback(object : Callback() {
                            override fun onCreate(db:
SupportSQLiteDatabase) {
                                super.onCreate(db)
                                // Заполняем блюда сразу при
создании базы

CoroutineScope(Dispatchers.IO).launch {
                                INSTANCE?.dishDao()?.let { dao -
>
                                    InitialData.dishes.forEach {
dao.insertDish(it) }
                                }
                            }
                        })
                        .fallbackToDestructiveMigration(true)
                        .build()
                    INSTANCE = instance
                    instance
                }
            }
        }
    }
}

```

Листинг Г.4 – DAO-интерфейсы для дневника, блюд и ИИ-рациона

```

//DiaryDao.kt
@Insert
suspend fun addEntry(entry: DiaryEntryEntity)

@Query("""

```

```

SELECT * FROM diary_entries
WHERE date = :dayStart AND userId = :userId
ORDER BY
    CASE mealType
        WHEN 'BREAKFAST' THEN 1
        WHEN 'LUNCH' THEN 2
        WHEN 'DINNER' THEN 3
        WHEN 'SNACK' THEN 4
        ELSE 5
    END,
    id ASC
""")
suspend fun getEntriesForDay(dayStart: Long, userId: Long):
List<DiaryEntryEntity>

    @Query("DELETE FROM diary_entries WHERE id = :entryId AND
userId = :userId")
suspend fun deleteEntry(entryId: Int, userId: Long)

    @Query("""
        SELECT SUM(d.caloriesPer100g * e.grams / 100.0) as
totalCalories,
                SUM(d.proteinPer100g * e.grams / 100.0) as
totalProtein,
                SUM(d.fatPer100g * e.grams / 100.0) as totalFat,
                SUM(d.carbsPer100g * e.grams / 100.0) as
totalCarbs
        FROM diary_entries e
        JOIN dishes d ON e.dishId = d.id
        WHERE e.date = :dayStart AND e.userId = :userId
    """)
suspend fun getDayStats(dayStart: Long, userId: Long):
DayStats?

    @Query("DELETE FROM diary_entries WHERE userId = :userId")
suspend fun deleteAllForUser(userId: Long)

    @Query("""
        SELECT COUNT(*)
        FROM diary_entries
        WHERE date = :dayStart AND userId = :userId
    """)
suspend fun getEntriesCountForDay(dayStart: Long, userId:
Long): Int

    data class DayStats(
        val totalCalories: Double?,
        val totalProtein: Double?,
        val totalFat: Double?,
        val totalCarbs: Double?
    )

```

//DishDao.kt

```

@Query("SELECT * FROM dishes WHERE isForAiGeneration = 1")
suspend fun getAiGenerationDishes(): List<DishEntity>

@Query("SELECT * FROM dishes WHERE name = :name LIMIT 1")
suspend fun getDishByName(name: String): DishEntity?

//AiMealPlanDao.kt
@Insert
suspend fun insert(plan: AiMealPlanEntity): Long

@Query("SELECT * FROM ai_meal_plans WHERE date = :date AND
userId = :userId LIMIT 1")
suspend fun getByDate(date: String, userId: Long):
AiMealPlanEntity?

@Query("DELETE FROM ai_meal_plans WHERE date = :date AND userId
= :userId")
suspend fun deleteByDate(date: String, userId: Long)

```

Листинг Г.5 – Взаимодействие с GigaChat API

```

//GigaChatApi.kt
interface GigaChatApi {

    @POST("api/v1/chat/completions")
    suspend fun generateMealPlan(
        @Header("Authorization") token: String,
        @Body request: GigaChatRequest
    ): GigaChatResponse
}

//GigaChatAuth.kt
suspend fun getAccessToken(): String =
withContext(Dispatchers.IO) {
    if (clientId.isBlank() || clientSecret.isBlank()) {
        throw IllegalStateException(
            "Не заданы GIGACHAT_CLIENT_ID и
GIGACHAT_CLIENT_SECRET в local.properties"
        )
    }

    val credentials = "$clientId:$clientSecret"
    val base64Credentials =
android.util.Base64.encodeToString(
        credentials.toByteArray(Charsets.UTF_8),
        android.util.Base64.NO_WRAP
    )

    val request = Request.Builder()
        .url(AUTH_URL)
        .post(
            FormBody.Builder()
                .add("scope", "GIGACHAT_API_PERS")
                .build()
        )

```

```

        )
        .header("Authorization", "Basic $base64Credentials")
        .header("RqUID", UUID.randomUUID().toString())
        .header("Content-Type", "application/x-www-form-
urlencoded")
        .build()

        val response = client.newCall(request).execute()
        val responseBody = response.body?.string()
        ?: throw Exception("Пустой ответ от сервера
авторизации")

        if (!response.isSuccessful) {
            throw Exception("Ошибка авторизации:
${response.code} - $responseBody")
        }

        val json = JSONObject(responseBody)
        json.getString("access_token")
    }

//RetrofitClient.kt
object RetrofitClient {

    private const val BASE_URL =
"https://gigachat.devices.sberbank.ru/"

    private val logging = HttpLoggingInterceptor().apply {
        redactHeader("Authorization")
        level = if (BuildConfig.DEBUG) {
            HttpLoggingInterceptor.Level.BASIC
        } else {
            HttpLoggingInterceptor.Level.NONE
        }
    }

    private val client = OkHttpClient.Builder()
        .addInterceptor(logging)
        .connectTimeout(60, TimeUnit.SECONDS)
        .readTimeout(60, TimeUnit.SECONDS)
        .writeTimeout(60, TimeUnit.SECONDS)
        .build()

    val retrofit: Retrofit = Retrofit.Builder()
        .baseUrl(BASE_URL)
        .client(client)
        .addConverterFactory(GsonConverterFactory.create())
        .build()

    val gigaChatApi: GigaChatApi =
retrofit.create(GigaChatApi::class.java)
}

```

```
//GigaChatRequest.kt
data class GigaChatRequest(
    val model: String = "GigaChat",
    val messages: List<GigaChatMessage>,
    val temperature: Double = 0.1,
    val max_tokens: Int = 1500
)

data class GigaChatMessage(
    val role: String,
    val content: String
)
```

Листинг Г.6 – Генерация ИИ-рациона (файл DietologistViewModel.kt)

```
data class AiProductSelection(
    @SerializedName("breakfast") val breakfast:
List<String>,
    @SerializedName("lunch") val lunch: List<String>,
    @SerializedName("dinner") val dinner: List<String>,
    @SerializedName("snacks") val snacks: List<String>
)

fun onInputChange(text: String) { inputText = text }

fun sendMessage() {
    val text = inputText.trim()
    if (text.isEmpty()) return
    inputText = ""

    viewModelScope.launch {
        chatDao.insert(
            ChatMessageEntity(
                userId = userId,
                role = "user",
                content = text
            )
        )

        if (currentMealPlan != null &&
isMealPlanEditRequest(text)) {
            refineMealPlan(text)
        } else {
            getAiResponse(text)
        }
    }
}

fun generateMealPlan(profile: UserProfile, targetCalories:
Double) {
    viewModelScope.launch {
        isLoading = true
        lastUserAllergens = profile.allergies
        val target = targetCalories.toInt()
    }
}
```

```

        lastGeneratedTargetCalories = target

        chatDao.insert(ChatMessageEntity(userId = userId,
role = "user",
        content = "Составь мне рацион на $target ккал"))

        try {
            val token = "Bearer
${GigaChatAuth.getAccessToken()}"
            if (target !in 500..6000) {
                chatDao.insert(ChatMessageEntity(userId =
userId, role = "assistant",
                    content = "⚠️ Некорректная дневная норма
калорий ($target ккал).")
                    isLoading = false
                    return@launch
            }

            val portionLimit = when (profile.goal) {
                Goal.GAIN_MUSCLE -> 600
                Goal.LOSE_WEIGHT -> 400
                Goal.MAINTAIN -> 500
            }
            lastPortionLimit = portionLimit

            val dishes = getDishes()
                .filter { !hasForbiddenAllergen(it,
profile.allergies) }
            val prompt = buildSelectionPrompt(profile,
target, dishes)
            val request = GigaChatRequest(
                model = "GigaChat",
                messages = listOf(
                    GigaChatMessage(role = "system",
                        content = "Ты — диетолог. Выбирай
продукты только из предоставленного списка. " +
                        "Не придумывай новых
названий. Отвечай строго JSON."),
                    GigaChatMessage(role = "user", content =
prompt)
                ),
                temperature = 0.3,
                max_tokens = 1000
            )
            val response = api.generateMealPlan(token,
request)
            val json =
response.choices.first().message.content
                .replace("`json", "").replace("`",
"").trim()

            Log.d("Dietologist", "ИИ выбрал: $json")

```

```

        var selection = try {
            Gson().fromJson(json,
AiProductSelection::class.java)
        } catch (e: Exception) {
            Log.e("Dietologist", "Ошибка парсинга:
${e.message}")

            val bf = dishes.filter { it.category ==
"breakfast" }.shuffled().take(2).map { it.name }
            val lu = dishes.filter { it.category ==
"lunch" }.shuffled().take(3).map { it.name }
            val dn = dishes.filter { it.category ==
"dinner" }.shuffled().take(2).map { it.name }
            val sn = dishes.filter { it.category ==
"snack" }.shuffled().take(2).map { it.name }
            AiProductSelection(bf, lu, dn, sn)
        }

        selection = filterExistingProducts(selection,
dishes)
        selection = removeAllergenProducts(selection,
dishes, profile.allergies)
        selection =
enforceMinimumRealProducts(selection, dishes, profile.allergies)

        val bfCal = (target * 0.28).toInt()
        val luCal = (target * 0.38).toInt()
        val dnCal = (target * 0.27).toInt()
        val snCal = target - bfCal - luCal - dnCal

        val breakfastItems =
calculateWeights(selection.breakfast, bfCal, portionLimit)
        val lunchItems =
calculateWeights(selection.lunch, luCal, portionLimit)
        val dinnerItems =
calculateWeights(selection.dinner, dnCal, portionLimit)
        val snackItems =
calculateWeights(selection.snacks, snCal, portionLimit)

        val namedPlan = MealPlan(breakfastItems,
lunchItems, dinnerItems, snackItems)

        currentMealPlan = namedPlan

        val totalCal = namedPlan.totalCalories()
        val prot = namedPlan.totalProtein().toInt()
        val fat = namedPlan.totalFat().toInt()
        val carb = namedPlan.totalCarbs().toInt()

        chatDao.insert(ChatMessageEntity(
            userId = userId, role = "assistant",
            content = "Рацион готов! Всего $totalCal
ккал. Б: ${prot}г, Ж: ${fat}г, У: ${carb}г"
        ))

```

```

    } catch (e: Exception) {
        Log.e("Dietologist", "ОШИБКА ГЕНЕРАЦИИ", e)

        chatDao.insert(
            ChatMessageEntity(
                userId = userId,
                role = "assistant",
                content = "Ошибка генерации: ${e.message}
?: e::class.java.simpleName}"
            )
        )
    } finally {
        isLoading = false
    }
}

private fun buildSelectionPrompt(profile: UserProfile, target:
Int, dishes: List<DishEntity>): String {
    val allergies = profile.allergies.joinToString().ifEmpty
{ "нет" }
    val breakfastCal = (target * 0.28).toInt()
    val lunchCal = (target * 0.38).toInt()
    val dinnerCal = (target * 0.27).toInt()
    val snacksCal = target - breakfastCal - lunchCal -
dinnerCal
    val list = dishes.joinToString("\n") { "- ${it.name}" }

    val goalInst = when (profile.goal) {
        Goal.LOSE_WEIGHT -> "Похудение. Выбирай
низкокалорийные продукты, больше овощей."
        Goal.GAIN_MUSCLE -> "Набор массы. Выбирай калорийные
продукты с высоким белком."
        else -> "Поддержание. Сбалансированное питание."
    }

    val sampleBreakfast = listOf("Овсянка на воде", "Творог
5%").joinToString("\", \"")
    val sampleLunch = listOf("Куриная грудка", "Гречка
отварная", "Салат овощной").joinToString("\", \"")
    val sampleDinner = listOf("Запеченная рыба", "Брокколи
на пару").joinToString("\", \"")
    val sampleSnacks = listOf("Яблоко", "Грецкий
орех").joinToString("\", \"")

    return ""
    $goalInst
    Калорий: $target ккал.
    Аллергии: $allergies. Исключай продукты с этими
аллергенами!
    Завтрак $breakfastCal ккал (2 продукта), обед $lunchCal
ккал (3 продукта), ужин $dinnerCal ккал (2 продукта), перекусы
$snacksCal ккал (2 продукта).

```

Используй ТОЛЬКО продукты из списка ниже (точные названия). Не придумывай своих!

Список продуктов:

\$list

Пример ответа (обязательно используй реальные названия из списка):

```
{"breakfast":["$sampleBreakfast"],"lunch":["$sampleLunch"],"dinner":["$sampleDinner"],"snacks":["$sampleSnacks"]}
```

Твой ответ должен быть только таким JSON.

```
"".trimIndent()
```

```
}
```

```
private fun normalizeAllergen(value: String): String {
    val v = value.trim().lowercase()
```

```
    return when {
        v.contains("молок") || v.contains("молоч") ->
```

```
        "молоко"
```

```
        v.contains("орех") -> "орехи"
```

```
        v.contains("глют") -> "глютен"
```

```
        v.contains("яйц") -> "яйца"
```

```
        v.contains("рыб") -> "рыба"
```

```
        v.contains("соев") || v.contains("соя") -> "соевые"
```

```
        else -> v
```

```
    }
```

```
}
```

```
private fun hasForbiddenAllergen(dish: DishEntity, allergens:
List<String>): Boolean {
```

```
    val forbidden = allergens
```

```
        .map(::normalizeAllergen)
```

```
        .filter { it.isNotBlank() }
```

```
        .toSet()
```

```
    if (forbidden.isEmpty()) return false
```

```
    return dish.allergens
```

```
        .split(",")
```

```
        .map(::normalizeAllergen)
```

```
        .filter { it.isNotBlank() }
```

```
        .any { it in forbidden }
```

```
}
```

```
private fun filterExistingProducts(sel: AiProductSelection,
dishes: List<DishEntity>): AiProductSelection {
```

```
    fun List<String>.keepExisting() = filter { name ->
```

```
        dishes.any { it.name.equals(name, ignoreCase = true)
```

```
}
```

```
}
```

```
    return sel.copy(
```

```
        breakfast = sel.breakfast.keepExisting(),
```

```

        lunch = sel.lunch.keepExisting(),
        dinner = sel.dinner.keepExisting(),
        snacks = sel.snacks.keepExisting()
    )
}

private fun removeAllergenProducts(
    sel: AiProductSelection,
    dishes: List<DishEntity>,
    allergens: List<String>
): AiProductSelection {
    if (allergens.isEmpty()) return sel

    fun List<String>.safe() = filter { name ->
        val dish = dishes.find { it.name.equals(name,
            ignoreCase = true) }
        ?: return@filter false

        !hasForbiddenAllergen(dish, allergens)
    }

    return sel.copy(
        breakfast = sel.breakfast.safe(),
        lunch = sel.lunch.safe(),
        dinner = sel.dinner.safe(),
        snacks = sel.snacks.safe()
    )
}

private fun enforceMinimumRealProducts(
    sel: AiProductSelection,
    dishes: List<DishEntity>,
    allergens: List<String>
): AiProductSelection

```

Листинг Г.7 – Расчет граммовок продуктов (файл DietologistViewModel.kt)

```

private suspend fun calculateWeights(
    names: List<String>,
    targetCal: Int,
    maxGram: Int
): List<MealItem> {
    val dishes = getDishes()

    val products = names.mapNotNull { name ->
        dishes.find { it.name.equals(name, ignoreCase =
true) }
    }

    if (products.isEmpty()) {
        return listOf(
            MealItem(
                name = "Неизвестный продукт",
                calories = targetCal.coerceAtLeast(1),

```

```

        protein = 0.0,
        fat = 0.0,
        carbs = 0.0,
        grams = 100
    )
)
}

val priority = listOf(
    "meat",
    "fish",
    "garnish",
    "vegetable",
    "dairy",
    "fruit",
    "nuts",
    "bread",
    "snack",
    "drink",
    "oil",
    "breakfast"
)

val sorted = products.sortedBy { product ->
    priority.indexOf(product.type).let { index ->
        if (index >= 0) index else 100
    }
}

val minMap = mapOf(
    "meat" to 120,
    "fish" to 120,
    "garnish" to 100,
    "vegetable" to 80,
    "dairy" to 100,
    "fruit" to 100,
    "nuts" to 15,
    "oil" to 5,
    "bread" to 30,
    "snack" to 30,
    "drink" to 150,
    "breakfast" to 100
)

fun createItem(product: DishEntity, grams: Int):
MealItem {
    val safeGrams = grams.coerceIn(1, maxGram)

    return MealItem(
        name = product.name,
        calories = (product.caloriesPer100g * safeGrams
/ 100.0).roundToInt().coerceAtLeast(1),

```

```

        protein = round1(product.proteinPer100g *
safeGrams / 100.0),
        fat = round1(product.fatPer100g * safeGrams /
100.0),
        carbs = round1(product.carbsPer100g * safeGrams
/ 100.0),
        grams = safeGrams
    )
}

val items = sorted.map { product ->
    val startGrams = minMap[product.type] ?: 80
    createItem(product, startGrams)
}.toMutableList()

val startCalories = items.sumOf { it.calories }

if (startCalories <= 0) {
    return items
}

// Если минимальные порции уже больше цели,
// уменьшаем ВСЕ продукты пропорционально, а не один
продукт до 1 г.
if (startCalories > targetCal) {
    val factor = targetCal.toDouble() /
startCalories.toDouble()

    return sorted.mapIndexed { index, product ->
        val oldItem = items[index]
        val newGrams = (oldItem.grams * factor)
            .roundToInt()
            .coerceIn(10, maxGram)

        createItem(product, newGrams)
    }
}

// Если калорий не хватает, равномерно подбираем до цели.
var remaining = targetCal - startCalories

while (remaining > 10) {
    var addedSomething = false

    for (i in items.indices) {
        if (remaining <= 10) break

        val product = sorted[i]
        val currentItem = items[i]

        if (currentItem.grams >= maxGram) continue
    }
}

```

```

        val addGrams = 10.coerceAtMost(maxGram -
currentItem.grams)
        if (addGrams <= 0) continue

        val newGrams = currentItem.grams + addGrams
        val newItem = createItem(product, newGrams)

        val addedCalories = newItem.calories -
currentItem.calories
        if (addedCalories <= 0) continue

        items[i] = newItem
        remaining -= addedCalories
        addedSomething = true
    }

    if (!addedSomething) break
}

// Финальная мягкая корректировка, чтобы попасть ближе к
цели.
val currentTotal = items.sumOf { it.calories }
val diff = targetCal - currentTotal

if (kotlin.math.abs(diff) > 15) {
    val indexToAdjust = sorted.indexOfFirst {
        it.type in listOf("meat", "fish", "garnish",
"breakfast", "dairy")
    }.takeIf { it >= 0 } ?: 0

    val product = sorted[indexToAdjust]
    val currentItem = items[indexToAdjust]

    val caloriesPerGram = product.caloriesPer100g /
100.0

    if (caloriesPerGram > 0) {
        val gramsDiff = (diff / caloriesPerGram)
            .roundToInt()

        val newGrams = (currentItem.grams + gramsDiff)
            .coerceIn(10, maxGram)

        items[indexToAdjust] = createItem(product,
newGrams)
    }
}

return items
}

```

Листинг Г.8 – Корректировка ИИ-рациона по запросу пользователя (файл DietologistViewModel.kt)

```
private enum class MealSlot {
    BREAKFAST,
    LUNCH,
    DINNER,
    SNACKS
}

private enum class MealEditAction {
    REMOVE,
    ADD,
    REPLACE
}

private data class ParsedMealEdit(
    val action: MealEditAction,
    val fromDish: DishEntity? = null,
    val toDish: DishEntity? = null,
    val slot: MealSlot? = null
)

private fun isMealPlanEditRequest(text: String): Boolean {
    val t = text
        .lowercase()
        .replace('ё', 'е')
        .replace(Regex("\\s+"), " ")
        .trim()

    val questionStarters = listOf(
        "сколько",
        "что",
        "почему",
        "зачем",
        "как",
        "какой",
        "какая",
        "какие",
        "можно ли",
        "нужно ли",
        "стоит ли",
        "расскажи",
        "объясни",
        "подскажи"
    )

    val startsAsQuestion = questionStarters.any { starter ->
        t == starter || t.startsWith("$starter ")
    }

    val editCommands = listOf(
        "измени",

```

```

        "изменить",
        "поменяй",
        "поменять",
        "замени",
        "заменить",
        "убери",
        "убрать",
        "исключи",
        "исключить",
        "удали",
        "удалить",
        "добавь",
        "добавить",
        "включи",
        "включить",
        "уменьши",
        "уменьшить",
        "увеличь",
        "увеличить",
        "снизь",
        "снизить",
        "повысь",
        "повысить",
        "убавь",
        "убавить",
        "сделай меньше",
        "сделай больше",
        "сделай рацион",
        "пересчитай",
        "пересчитать",
        "скорректируй",
        "скорректировать",
        "отредактируй",
        "отредактировать",
        "переделай",
        "переделать",
        "перегенерируй",
        "перегенерировать",
        "составь заново",
        "другой рацион",
        "другой вариант",
        "новый рацион",
        "новый вариант",
        "хочу без",
        "сделай без",
        "давай без"
    )

    val hasEditCommand = editCommands.any { command ->
        t.contains(command)
    }

    if (!hasEditCommand) return false

```

```

        if (startsWithQuestion) return false

        return true
    }
    private fun normalizeUserText(text: String): String {
        return text
            .lowercase()
            .replace('ё', 'е')
            .replace(Regex("[,.;!()?\\[\\]\\"]"), " ")
            .replace(Regex("\\s+"), " ")
            .trim()
    }
    private fun detectMealSlot(text: String): MealSlot? {
        val normalized = normalizeUserText(text)

        return when {
            listOf("завтрак", "завтрака", "на завтрак", "в
завтрак").any { normalized.contains(it) } -> {
                MealSlot.BREAKFAST
            }

            listOf("обед", "обеда", "на обед", "в обед").any {
normalized.contains(it) } -> {
                MealSlot.LUNCH
            }

            listOf("ужин", "ужина", "на ужин", "в ужин").any {
normalized.contains(it) } -> {
                MealSlot.DINNER
            }

            listOf("перекус", "перекусы", "перекусов", "в
перекус", "на перекус").any { normalized.contains(it) } -> {
                MealSlot.SNACKS
            }

            else -> null
        }
    }

    private fun findDishMention(
        text: String,
        dishes: List<DishEntity>,
        excludeDish: DishEntity? = null
    ): DishEntity?
    private fun parseMealEditInstruction(
        userMessage: String,
        dishes: List<DishEntity>
    ): ParsedMealEdit?
    private suspend fun tryApplyControlledMealEdit(
        oldPlan: MealPlan,
        userMessage: String,

```

```

        dishes: List<DishEntity>,
        targetCal: Int,
        explanation: String
    ): Boolean
private fun resolveTargetCalories(
    userMessage: String,
    currentCalories: Int
): Int
private suspend fun refineMealPlan(userMessage: String)
private fun buildRefinePrompt(
    userMessage: String,
    oldPlan: MealPlan,
    targetCal: Int,
    dishes: List<DishEntity>
): String

```

Листинг Г.9 – Сохранение ИИ-рациона в дневник питания (файл DietologistViewModel.kt)

```

private fun getTodayStartMillis(): Long =
    Calendar.getInstance().apply {
        set(Calendar.HOUR_OF_DAY, 0)
        set(Calendar.MINUTE, 0)
        set(Calendar.SECOND, 0)
        set(Calendar.MILLISECOND, 0)
    }.timeInMillis
private suspend fun addMealItemsToDiary(
    items: List<MealItem>,
    mealType: String,
    dayStart: Long
)

fun saveToDiary() {
    val plan = currentMealPlan ?: return

    viewModelScope.launch {
        val today = LocalDate.now().toString()
        val dayStart = getTodayStartMillis()

        val existingPlan = aiMealPlanDao.getByDate(today,
        userId)

        val diaryEntriesCount =
        diaryDao.getEntriesCountForDay(dayStart, userId)

        if (existingPlan?.isSavedToDiary == true &&
        diaryEntriesCount > 0) {
            chatDao.insert(
                ChatMessageEntity(
                    userId = userId,
                    role = "assistant",
                    content = "Этот рацион уже добавлен в
        дневник."
                )
            )
        }
    }
}

```

```

        )
        return@launch
    }

    // Если карточка ИИ-рациона осталась, но в дневнике
уже нет продуктов,
    // считаем ее устаревшей и перезаписываем новым
рационом.
    aiMealPlanDao.deleteByDate(today, userId)

    val entity = AiMealPlanEntity(
        userId = userId,
        date = today,
        jsonContent = Gson().toJson(plan),
        totalCalories = plan.totalCalories(),
        totalProtein = plan.totalProtein(),
        totalFat = plan.totalFat(),
        totalCarbs = plan.totalCarbs(),
        isSavedToDiary = true
    )

    aiMealPlanDao.insert(entity)

    addMealItemsToDiary(plan.breakfast, "BREAKFAST",
dayStart)
    addMealItemsToDiary(plan.lunch, "LUNCH", dayStart)
    addMealItemsToDiary(plan.dinner, "DINNER", dayStart)
    addMealItemsToDiary(plan.snacks, "SNACK", dayStart)

    chatDao.insert(
        ChatMessageEntity(
            userId = userId,
            role = "assistant",
            content = "✅ Рацион добавлен в дневник!"
        )
    )
}
}

```

Приложение Д
(рекомендуемое)

Результаты функционального тестирования

Давайте настроим ваш профиль

Ваше имя

Пол

☒ Мужской ☐ Женский

Возраст

Рост (см)

Вес (кг)

Уровень активности

☐ Сидячий образ жизни

☒ Лёгкая активность (1-3 раза в неделю)

☐ Умеренная активность (3-5 раза в неделю)

☐ Высокая активность (6-7 раза в неделю)

☐ Экстремальная активность

Цель

☐ Похудение

☒ Поддержание веса

☐ Набор мышечной массы

Аллергии (выберите при наличии)

☐ Молоко

☐ Орехи

☐ Глютен

☐ Яйца

☐ Рыба

☐ Соевые продукты

Имя должно содержать хотя бы 2 символа

Рассчитать норму калорий

Рисунок Д.1 – Результат тест-кейса № 1

Давайте настроим ваш профиль

Ваше имя
Антон#

Пол

☒ Мужской ☐ Женский

Возраст
25

Рост (см)
170

Вес (кг)
70

Уровень активности

☐ Сидячий образ жизни

☒ Лёгкая активность (1-3 раза в неделю)

☐ Умеренная активность (3-5 раза в неделю)

☐ Высокая активность (6-7 раза в неделю)

☐ Экстремальная активность

Цель

☐ Похудение

☒ Поддержание веса

☐ Набор мышечной массы

Аллергии (выберите при наличии)

☐ Молоко

☐ Орехи

☐ Глютен

☐ Яйца

☐ Рыба

☐ Соевые продукты

Имя содержит недопустимые символы

Рассчитать норму калорий

Рисунок Д.2 – Результат тест-кейса № 2

Давайте настроим ваш профиль

Ваше имя

Антон

Пол

☒ Мужской ☐ Женский

Возраст

0

Рост (см)

170

Вес (кг)

70

Уровень активности

☐ Сидячий образ жизни
☒ Лёгкая активность (1-3 раза в неделю)
☐ Умеренная активность (3-5 раза в неделю)
☐ Высокая активность (6-7 раза в неделю)
☐ Экстремальная активность

Цель

☐ Похудение
☒ Поддержание веса
☐ Набор мышечной массы

Аллергии (выберите при наличии)

☐ Молоко

☐ Орехи

☐ Глютен

☐ Яйца

☐ Рыба

☐ Соевые продукты

Возраст должен быть от 1 до 120 лет

Рассчитать норму калорий

Рисунок Д.3 – Результат тест-кейса № 3

Давайте настроим ваш профиль

Ваше имя

АНТОН

Пол

☒ Мужской ☐ Женский

Возраст

25

Рост (см)

40

Вес (кг)

70

Уровень активности

- ☐ Сидячий образ жизни
- ☒ Лёгкая активность (1-3 раза в неделю)
- ☐ Умеренная активность (3-5 раза в неделю)
- ☐ Высокая активность (6-7 раза в неделю)
- ☐ Экстремальная активность

Цель

- ☐ Похудение
- ☒ Поддержание веса
- ☐ Набор мышечной массы

Аллергии (выберите при наличии)

- ☐ Молоко
- ☐ Орехи
- ☐ Глютен
- ☐ Яйца
- ☐ Рыба
- ☐ Соевые продукты

Рост должен быть от 50 до 250 см

Рассчитать норму калорий

Рисунок Д.4 – Результат тест-кейса № 4

Давайте настроим ваш профиль

Ваше имя
Антон

Пол
☒ Мужской ☐ Женский

Возраст
25

Рост (см)
300

Вес (кг)
70

Уровень активности

☐ Сидячий образ жизни

☒ Лёгкая активность (1-3 раза в неделю)

☐ Умеренная активность (3-5 раза в неделю)

☐ Высокая активность (6-7 раза в неделю)

☐ Экстремальная активность

Цель

☐ Похудение

☒ Поддержание веса

☐ Набор мышечной массы

Аллергии (выберите при наличии)

☐ Молоко

☐ Орехи

☐ Глютен

☐ Яйца

☐ Рыба

☐ Соевые продукты

Рост должен быть от 50 до 250 см

Рассчитать норму калорий

Рисунок Д.5 – Результат тест-кейса № 5

Давайте настроим ваш профиль

Ваше имя
Антон

Пол
☒ Мужской ☐ Женский

Возраст
25

Рост (см)
170

Вес (кг)
5

Уровень активности
☐ Сидячий образ жизни
☒ Лёгкая активность (1-3 раза в неделю)
☐ Умеренная активность (3-5 раза в неделю)
☐ Высокая активность (6-7 раза в неделю)
☐ Экстремальная активность

Цель
☐ Похудение
☒ Поддержание веса
☐ Набор мышечной массы

Аллергии (выберите при наличии)
☐ Молоко
☐ Орехи
☐ Глютен
☐ Яйца
☐ Рыба
☐ Соевые продукты

Вес должен быть от 10 до 300 кг

Рассчитать норму калорий

Рисунок Д.6 – Результат тест-кейса № 6

Давайте настроим ваш профиль

Ваше имя

Антон

Пол

☒ Мужской ☐ Женский

Возраст

25

Рост (см)

170

Вес (кг)

350

Уровень активности

☐ Сидячий образ жизни

☒ Лёгкая активность (1-3 раза в неделю)

☐ Умеренная активность (3-5 раза в неделю)

☐ Высокая активность (6-7 раза в неделю)

☐ Экстремальная активность

Цель

☐ Похудение

☒ Поддержание веса

☐ Набор мышечной массы

Аллергии (выберите при наличии)

☐ Молоко

☐ Орехи

☐ Глютен

☐ Яйца

☐ Рыба

☐ Соевые продукты

Вес должен быть от 10 до 300 кг

Рассчитать норму калорий

Рисунок Д.7 – Результат тест-кейса № 7

Давайте настроим ваш профиль

Ваше имя

Антон

Пол

☐ Мужской ☐ Женский

Возраст

25

Рост (см)

170

Вес (кг)

70

Уровень активности

☐ Сидячий образ жизни

☒ Лёгкая активность (1-3 раза в неделю)

☐ Умеренная активность (3-5 раза в неделю)

☐ Высокая активность (6-7 раза в неделю)

☐ Экстремальная активность

Цель

☐ Похудение

☒ Поддержание веса

☐ Набор мышечной массы

Аллергии (выберите при наличии)

☐ Молоко

☐ Орехи

☐ Глютен

☐ Яйца

☐ Рыба

☐ Соевые продукты

Выберите пол

Рассчитать норму калорий

Рисунок Д.8 – Результат тест-кейса № 8

Давайте настроим ваш профиль

Ваше имя

Антон

Пол

☒ Мужской ☐ Женский

Возраст

25

Рост (см)

170

Вес (кг)

70

Уровень активности

☐ Сидячий образ жизни

☐ Лёгкая активность (1-3 раза в неделю)

☐ Умеренная активность (3-5 раза в неделю)

☐ Высокая активность (6-7 раза в неделю)

☐ Экстремальная активность

Цель

☐ Похудение

☒ Поддержание веса

☐ Набор мышечной массы

Аллергии (выберите при наличии)

☐ Молоко

☐ Орехи

☐ Глютен

☐ Яйца

☐ Рыба

☐ Соевые продукты

Выберите уровень активности

Рассчитать норму калорий

Рисунок Д.9 – Результат тест-кейса № 9

Давайте настроим ваш профиль

Ваше имя

Антон

Пол

☒ Мужской ☐ Женский

Возраст

25

Рост (см)

170

Вес (кг)

70

Уровень активности

Уровень активности

☐ Сидячий образ жизни

☒ Лёгкая активность (1-3 раза в неделю)

☐ Умеренная активность (3-5 раза в неделю)

☐ Высокая активность (6-7 раза в неделю)

☐ Экстремальная активность

Цель

☐ Похудение

☐ Поддержание веса

☐ Набор мышечной массы

☐ Молоко

☐ Орехи

☐ Глютен

☐ Яйца

☐ Рыба

☐ Соевые продукты

Выберите цель

Рассчитать норму калорий

Рисунок Д.10 – Результат тест-кейса № 10

Приложение Е
(рекомендуемое)

Листинг модульных тестов приложения

Листинг Е.1 – Файл ProfileValidationServiceTest.kt

```

package com.app.nutritiondiary.domain.validation

import com.app.nutritiondiary.domain.model.ActivityLevel
import com.app.nutritiondiary.domain.model.Gender
import com.app.nutritiondiary.domain.model.Goal
import org.junit.Assert.*
import org.junit.Test

class ProfileValidationServiceTest {

    private val service = ProfileValidationService()

    @Test
    fun `validateName - empty name returns error`() {
        val result = service.validateName("")
        assertTrue(result is ValidationResult.Error)
        assertEquals("Введите имя", (result as
ValidationResult.Error).message)
    }

    @Test
    fun `validateName - too short name returns error`() {
        val result = service.validateName("A")
        assertTrue(result is ValidationResult.Error)
    }

    @Test
    fun `validateName - name with digits returns error`() {
        val result = service.validateName("Иван1")
        assertTrue(result is ValidationResult.Error)
    }

    @Test
    fun `validateAge - out of range returns error`() {
        val result = service.validateAge("0")
        assertTrue(result is ValidationResult.Error)
        assertEquals("Возраст должен быть от 1 до 120 лет",
(result as ValidationResult.Error).message)
    }

    @Test
    fun `validateAge - valid age returns ok`() {
        val result = service.validateAge("25")
        assertEquals(ValidationResult.Ok, result)
    }

    @Test
    fun `validateHeight - out of range returns error`() {
        val result = service.validateHeight("300")
        assertTrue(result is ValidationResult.Error)
    }
}

```

```

@Test
fun `validateWeight - out of range returns error`() {
    val result = service.validateWeight("5.0")
    assertTrue(result is ValidationResult.Error)
}
@Test
fun `checkSelections - missing gender returns error`() {
    val result = service.checkSelections(null,
ActivityLevel.LIGHT, Goal.MAINTAIN)
    assertTrue(result is ValidationResult.Error)
}
@Test
fun `checkSelections - all valid returns ok`() {
    val result = service.checkSelections(Gender.MALE,
ActivityLevel.LIGHT, Goal.MAINTAIN)
    assertEquals(ValidationResult.Ok, result)
}
}

```