

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Вычислительные машины, комплексы, системы и сети»

БАКАЛАВРСКАЯ РАБОТА

на тему

**МОБИЛЬНОЕ ПРИЛОЖЕНИЕ ДЛЯ
АНАЛИЗА И КОРРЕКТИРОВКИ ПИТАНИЯ**

Студент 31.05.22 Коротов В.О.
(подпись, дата)

Руководитель 31.05.22 Митрохин М.А.
(подпись, дата)

Нормоконтролёр 31.05.22 Бычков А.С.
(подпись, дата)

Работа допущена к защите (протокол зас. кафедры от 02.06.22 № 13)

Заведующий кафедрой

(подпись)

М.А. Митрохин

Работа защищена с отметкой Отлично (протокол заседания ГЭК от 15.06.2022 № 7)

Секретарь ГЭК

(подпись)

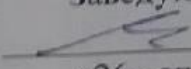
Е.И. Гурин

Пенза 2022

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

Факультет
вычислительной техники

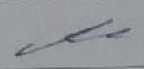
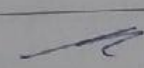
Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
«26» апреля 2022 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Коротов Владислав Олегович, гр.18ВВ1 факультета вычислительной техники.
2. Тема работы: «Мобильное приложение для анализа и корректировки питания»
Тема утверждена приказом ПГУ №0357/о от "26" апреля 2022 г.
3. Руководитель бакалаврской работы д.т.н., профессор, зав. каф. ВТ ПГУ Митрохин М.А.
4. Задание на работу (назначение разработки, исходные данные и т.п.)
Разработать мобильное приложение для анализа и корректировки питания с применением клиент-серверной архитектуры. Приложение должно вычислять примерное содержание калорий в блюде по фотографии, предоставлять пользователю возможность отслеживать свой вес, подбирать рецепты и помогать скорректировать питание.
5. Перечень подлежащих разработке вопросов
Требуется провести анализ предметной области, разработать архитектуру приложения, выбрать инструменты для разработки, реализовать клиент-серверное приложение.

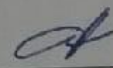
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	10%	26.04.2022 – 30.04.2022	
2.	Выбор инструментов для разработки	10%	01.05.2022 – 03.05.2022	
3.	Разработка серверной части	30%	04.05.2022 – 18.05.2022	
4.	Разработка клиентской части	30%	19.05.2022 – 22.05.2022	
5.	Выполнение задания по БЖД	10%	23.05.2022 – 26.05.2022	
6.	Оформление выпускной работы	10%	27.05.2022 – 31.05.2022	

Дата выдачи задания "26" апреля 2022 г.

Руководитель бакалаврской работы  Митрохин М.А.
(подпись, дата) 26.04.22 г.

Задание к исполнению принял студент  Коротов В.О.
(подпись, дата) 26.04.22 г.

Работу к защите допустить
Декан факультета  14.06.2022 Фионова Л.Р.
(подпись, дата)

Реферат

Работа содержит 93 листа, 23 рисунка, 5 таблиц, 7 источников, 3 приложения.

ANDROID, KOTLIN, КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ, PYTHON, PYTORCH, НЕЙРОННАЯ СЕТЬ, FLASK, БАЗА ДАННЫХ.

Цель ВКР – разработка приложения для анализа и корректировки питания пользователя. Это достигается путем анализа фотографий с едой, которые пользователь пересылает на сервер, расчета необходимого ежедневного количества калорий для поддержания веса. Также есть возможность отслеживать свой вес, читать статьи и подбирать рецепты по введенному списку ингредиентов.

В ходе выполнения ВКР было разработано клиент-серверное приложение, которое удовлетворяет поставленной задаче.

					ПГУ 1.090301.20.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб		Коротков В.О.					
Пров		Митрохин М.А.					
Н. Контр.		Бычков А.С.					
Утв		Митрохин М.А.					
					Лит.	Лист	Листов
						2	93
					ФВТ, каф. ВТ гр. 18ВВ1		

Мобильное приложение для анализа и корректировки питания

Пояснительная записка

Содержание

Введение.....	5
1 Мобильное приложение для анализа и корректировки питания	6
1.1 Анализ предметной области для решения поставленной задачи.....	6
1.1.1 Анализ темы.....	6
1.1.2 Анализ аналогов	6
1.1.3 Постановка задачи.....	10
1.2 Архитектура приложения.....	11
1.2.1 Выбор архитектуры приложения.....	11
1.2.2 Общая структура проекта.....	12
1.2.1 Разработка <i>UML</i> -диаграмм.....	13
1.3 Выбор инструментов разработки	15
1.3.1 Выбор языков программирования.....	15
1.3.2 Выбор фреймворка машинного обучения	16
1.3.3 Инструмент для аннотации картинок	16
1.3.4 Выбор среды программирования	17
1.4 Разработка программы.....	22
1.4.1 Анализ фотографии и расчет калорий	22
1.4.2 Нейронная сеть	23
1.4.3 Разработка сервера	30
1.4.4 Разработка клиента	32
2 Безопасность жизнедеятельности.....	50
2.1 Характеристика опасных факторов при работе с ПЭВМ	50
2.2 Классификация и категорирование помещения	51

Изм.	Лист	№ докум	Подпись	Дата

ПГУ 1.090301.20.001 ПЗ

Лист

3

2.3 Санитарно-гигиеническая характеристика.....	53
2.4 Электробезопасность	54
2.5 Пожарная безопасность	55
2.6 Расчет количества светильников в помещении	56
Заключение	58
Список использованных источников	59
Приложение А – <i>UML</i> -диаграммы	60
Приложение Б – Листинг программы	67
Приложение В – Презентация.....	87

Введение

В настоящее время многие люди страдают от проблем, вызванных неправильным питанием. Оно становится причиной смерти около 11 миллионов людей ежегодно. Люди злоупотребляют вредной, жирной пищей и иногда совсем не включают в свой рацион фрукты и овощи, которые содержат полезные для организма элементы и необходимые витамины. Проблемы с сердцем, инсульт, рак, болезни желудка и кишечника могут быть вызваны именно неправильным питанием. Также важно следить за своим весом. Исследования показывают, что люди с избыточным весом склонны к появлению таких серьезных заболеваний, как диабет или рак. Недостаток веса может привести к ослаблению иммунитета. Поэтому необходимо следить за собственным весом и заботиться о своем здоровье.

					<i>ПГУ 1.090301.20.001 ПЗ</i>	Лист
Изм.	Лист	№ докум	Подпись	Дата		5

1 Мобильное приложение для анализа и корректировки питания

1.1 Анализ предметной области для решения поставленной задачи

1.1.1 Анализ темы

В современном мире очень важно следить за своим здоровьем, а в частности, за питанием. Люди все чаще посещают рестораны быстрого питания, в следствии чего могут набирать лишний вес. Неправильное и несбалансированное питание может привести к серьезным нарушениям и сбоям в организме человека. Лишний вес может привести к целому ряду опасных для здоровья заболеваний, таких как гипертония или ишемическая болезнь сердца. Все это в итоге приводит к ранней смерти человека. Именно поэтому так важно выдерживать правильную диету и следить за своим весом.

Сейчас большинство людей имеют под рукой смартфон, поэтому было решено разработать мобильное приложение, с помощью которого пользователи смогли бы анализировать свое питание. Люди с лишним весом могли бы отслеживать изменение своего веса, спортсмены соблюдали бы диету и следили бы за потреблением калорий. Люди с проблемами со здоровьем могли бы подобрать себе рецепт, исключив продукты питания, которые им не следует употреблять.

1.1.2 Анализ аналогов

Проанализировав рынок мобильных приложений, было найдено большое количество приложений, которые могут помочь пользователю следить за своим питанием. Ниже представлен анализ и подробное описание некоторых из них.

Приложение *Calorie Mama* – одна из немногих программ, которая может определить по фотографии, какие из продуктов на ней представлены. Есть большая база данных с информацией о калорийности продуктов. В

приложении представлено большое количество рецептов, которые понравятся широкому кругу пользователей. Также есть возможность отслеживания изменения веса. Слабые стороны приложения – после установки программа занимает очень большое количество свободного места на устройстве. Некоторые пользователи отмечают, что в приложении довольно сложно ориентироваться. Функции сканирования штрих кода и идентификации еды по фото представлены отдельно друг от друга, хотя было бы логичнее и удобнее сделать их вместе. Интерфейс приложения представлен на рисунке 1.1.

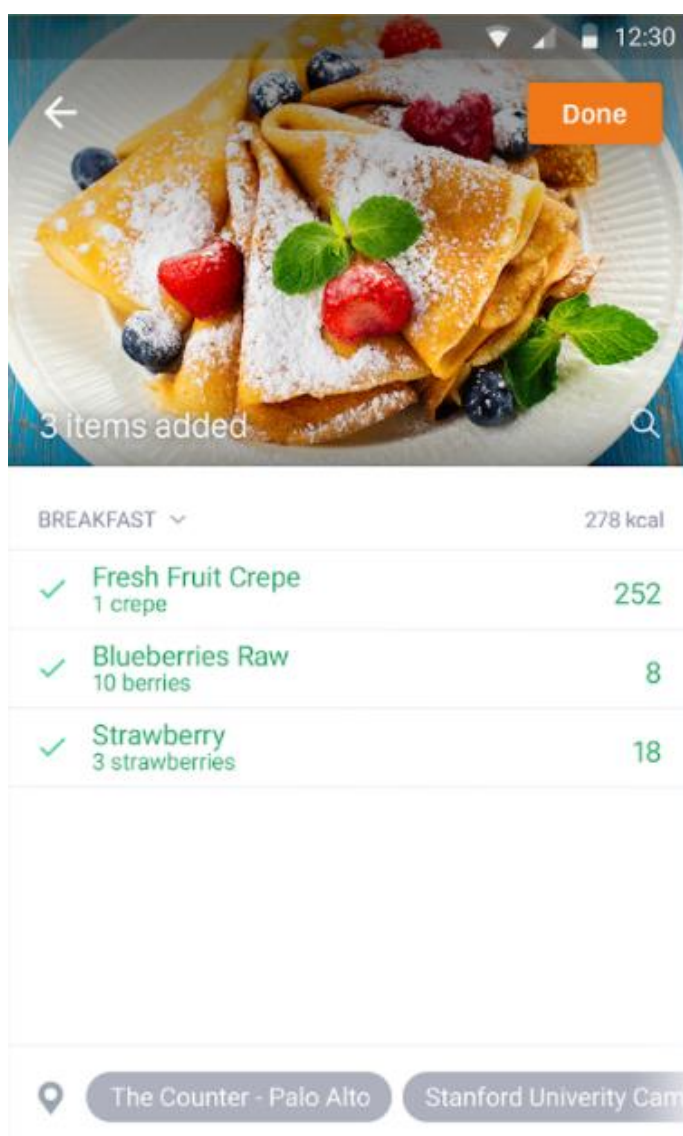


Рисунок 1.1 - Приложение *Calorie Mama*

Одно из самых популярных приложений для подсчета калорий является приложение *FatSecret*. Оно помогает пользователям с легкостью находить информацию о том, сколько содержится белков, жиров и углеводов в продукте, составлять отчеты о потребляемых калориях, а также имеет интеграцию с различными спортивными приложениями (например, с приложениями *Google Fit* или *Samsung Health*). Можно отслеживать изменения веса. Программа имеет достаточно богатый функционал, но не обошлось и без минусов. Некоторые пользователи жалуются на слишком сложный интерфейс и непонятную навигацию по приложению. Для того, чтобы пользоваться приложением, необходима регистрация аккаунта, что может оттолкнуть некоторых пользователей. Интерфейс приложения представлен на рисунке 1.2.

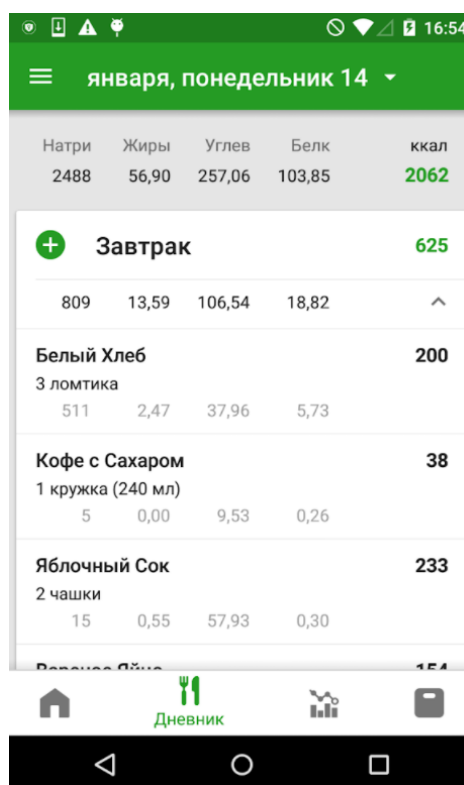


Рисунок 1.2 - Приложение *FatSecret*

Приложение *MyNetDiary* обладает очень приятным и информативным интерфейсом, что положительно сказывается на опыте пользователей. Можно отсканировать штрих код продукта и узнать калорийность. Также

предоставлен большой выбор диет, в зависимости от того, что необходимо пользователю (похудеть, набрать массу, вегетарианская диета и т.д.). Пользователю открывается доступ к широкому выбору рецептов, которые удовлетворят любые вкусовые предпочтения. Как и в предыдущем примере, присутствует интеграция со многими популярными спортивными приложениями и отслеживание веса. Присутствуют советы по улучшению питания. Из недостатков можно отметить, что в приложении отсутствует идентификация продукта по фото и расчет его калорийности. Также в одном из обновлений часть функционала была перенесена в раздел, доступный только за деньги, что негативно сказалось на отзывах некоторых пользователей. Интерфейс приложения представлен на рисунке 1.3.

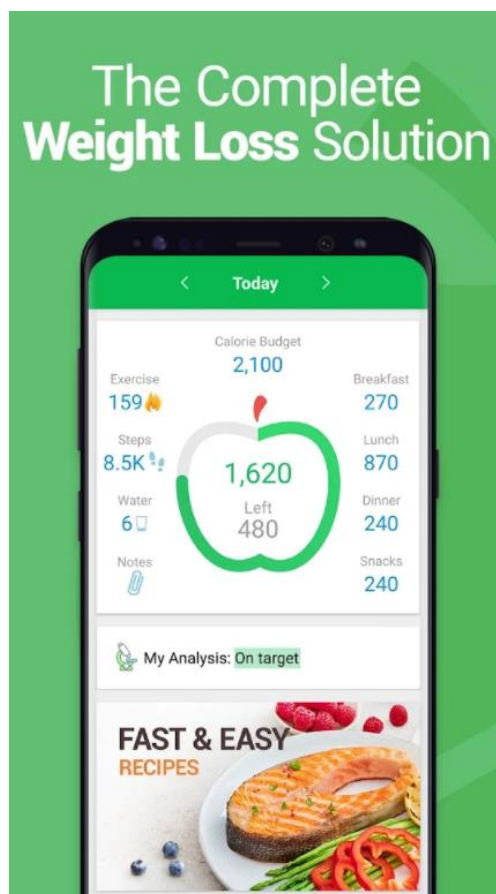


Рисунок 1.3 - Приложение MyNetDiary

Сравнение функционала вышеперечисленных приложений представлено в таблице 1.1.

Таблица 1.1 - Результат сравнительного анализа приложений

Функция	Название приложения		
	<i>Calorie Mama</i>	<i>FatSecret</i>	<i>MyNetDiary</i>
Идентификация по фото	+	+	-
Отслеживание потребляемых калорий	+	+	+
Подсчет калорий по фото	-	-	-
Отслеживание веса	+	+	+
Интеграция со спортивными приложениями	-	+	+
Интуитивный интерфейс	-	-	+
Поиск рецептов	+	-	+
Советы по питанию	-	-	+

В ходе анализа было выявлено, что многие приложения имеют сложный интерфейс, что следует из отрицательных отзывов пользователей. Также приложения не могут выполнить подсчет калорий по фото, поэтому приходится вводить вручную вес продукта. Почти во всех приложениях отсутствуют советы, которые могли бы улучшить питание человека. Из этого вытекает необходимость разработки приложения, которое покрыло бы эти недостатки.

1.1.3 Постановка задачи

Если верить статистике, операционная система *Android* лидирует на мировом рынке и занимает около 40%. Если брать в расчет только

мобильные устройства, то этот показатель колеблется в районе 70%. Именно поэтому было решено выбрать данную операционную систему для разработки программного продукта.

Возможности и функционал приложения:

- определение по фото состава блюда и его калорийность;
- возможность отслеживать количество потребляемых калорий;
- подбор советов по улучшению качества питания;
- возможность отслеживать изменение веса;
- возможность подобрать рецепт по введенному набору продуктов.

1.2 Архитектура приложения

1.2.1 Выбор архитектуры приложения

Клиент-серверная архитектура позволяет обрабатывать приходящие запросы, не используя при этом ресурсы другой машины. Такое распределение обязанностей необходимо для того, чтобы эффективно обрабатывать запросы, ведь в некоторых случаях для выполнения операции может потребоваться довольно большое количество времени и вычислительных ресурсов, которых у клиента может быть недостаточно. Именно поэтому, такие операции перекладываются на сервер, который обычно обладает достаточной вычислительной мощностью для выполнения различного рода задач.

Клиентская программа предназначена для пользователя. Она будет отправлять различные запросы к серверу и ожидать ответа.

Сервер выполняет прием запросов, которые отправляются со стороны клиента и обрабатывает их, отправляя обратно результат.

Для решения поставленной задачи потребуется нейронная сеть, которая сможет определять по фото состав блюда. Модель нейронной сети может много весить, поэтому ее хранение напрямую на устройстве пользователя сильно увеличит объем занимаемой памяти. Этот факт может оттолкнуть

многих пользователей, что видно по негативным комментариям одного из проанализированных приложений – *Calorie Mama*. Для решения этой проблемы можно создать сервер, на котором будет храниться модель нейронной сети, статьи, список рецептов и информация о калорийности различных продуктов. Также сервер может использоваться для хранения фотографий блюд пользователей, которые позднее могут использоваться для увеличения объема данных для обучения нейронной сети. В роли клиента будет выступать мобильное приложение. Клиент будет отправлять запросы для получения списка рецептов, а также будут посылаться фотографии блюд для анализа состава и получения информации о калорийности продуктов.

1.2.2 Общая структура проекта

Прежде, чем разрабатывать приложение, необходимо спроектировать его и определить, каким функционалом оно будет обладать, как будет происходить взаимодействие с сервером и т.д. Клиентское приложение будет отправлять различные запросы к серверу и ожидать ответа от него. На самом сервере будет храниться модели нейронных сетей, которые будут использоваться для анализа фотографии от пользователя. Также сервер будет хранить различные данные – статьи, рецепты, ингредиенты и диеты. По запросу пользователя сервер будет отправлять эти данные на клиент. Общая структура приложения представлена на рисунке 1.4.

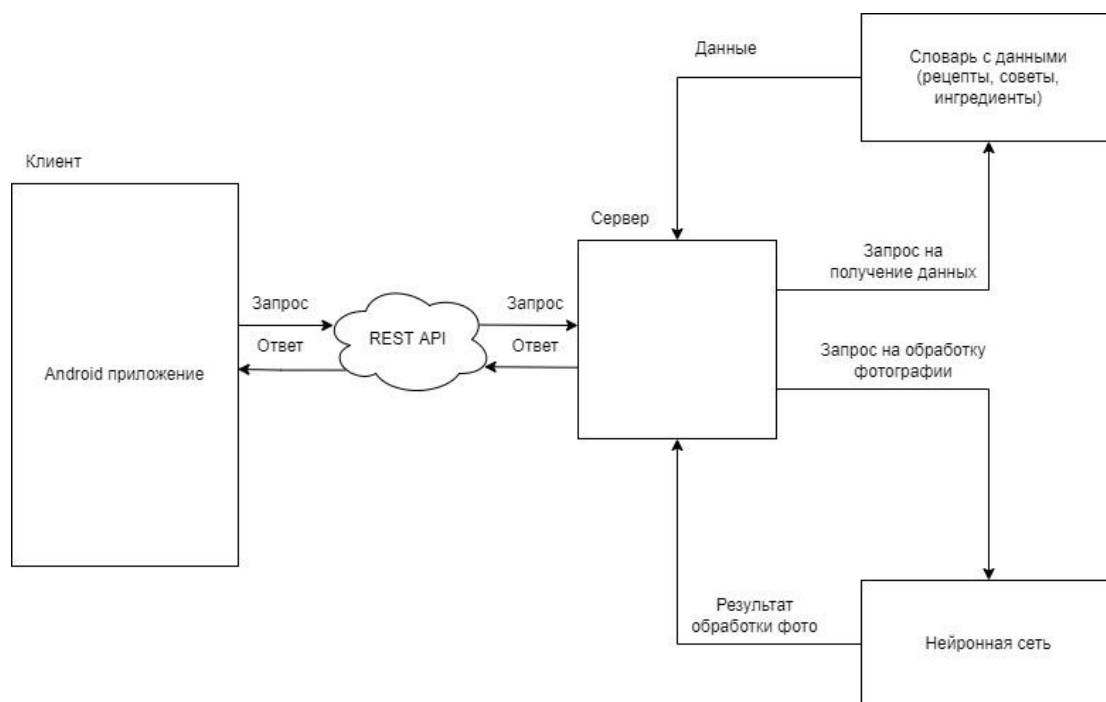


Рисунок 1.4 - Общая структура проекта

1.2.1 Разработка *UML*-диаграмм

Все разработанные диаграммы представлены в приложении Б. Исходя из выбранной архитектуры приложения была спроектирована диаграмма развертывания, которая показывает связь клиента и сервера. Далее была разработана диаграмма прецедентов для определения функционала клиента. К основным функциям относятся: вычисление калорий по фото, просмотр статей и рецептов, отслеживание веса.

Одна из главных функций приложения – определение калорий по фотографии. После того, как клиент загрузил картинку и ввел все необходимые данные, фотография отправляется на сервер, где происходит анализ изображения. В процессе анализа отправленной фотографии определяются ингредиенты, которые входят в блюдо и рассчитываются калории для каждого составляющего ингредиента. После этого результаты отправляются клиенту, который отображает их на интерфейсе. Далее пользователь может сохранить результаты в базу данных. Процесс

взаимодействия клиента и сервера был спроектирован с помощью диаграммы последовательности.

На сервере происходит анализ изображения, который позволяет определить количество калорий ингредиентов, которые представлены на фотографии. Это достаточно сложный процесс, который лучше всего заранее спроектировать с помощью диаграммы деятельности. После получения фотографии и диаметра тарелки, нейронные сети начинают анализ изображения. В ходе работы нейронных сетей на фотографии находится тарелка и создаются маски для всех ингредиентов, входящих в блюдо. Далее, зная диаметр тарелки, можно определить вес продукта, а затем и количество калорий. После этого, формируется результат и отправляется назад клиенту. Процесс представлен на диаграмме деятельности.

Для определения поведения клиента в процессе анализа фотографии была спроектирована диаграмма деятельности.

Перед разработкой также необходимо определить, какие классы будут в приложении и за какие функции они будут отвечать. Класс *CalorieCalculator* будет использоваться для расчета необходимого количества калорий для поддержания веса. Этот класс будет использоваться во фрагменте *HomeFragment*, в котором пользователь сможет изменить информацию о себе. В нем также будет описываться взаимодействия пользователя с интерфейсом приложения. Все совершаемые действия будут вызывать соответствующие функции из класса *HomeViewModel*. Также в классе *HomeViewModel* будут определены переменные, значения которых будет отслеживаться во фрагменте. Переменные хранят информацию о росте, весе, возрасте и т.д. Как только значение переменной изменится, интерфейс пользователя обновится. Аналогично будут взаимодействовать классы *FoodAnalysisViewModel* и *FoodAnalysisActivity*. *DishDetailing* будет хранить информацию о блюде, фотографию которого добавил пользователь. *Ingredient* отвечает за хранение одного продукта, который входит в блюдо.

1.3 Выбор инструментов разработки

1.3.1 Выбор языков программирования

Одним из самых популярных языков для создания *Android* приложений остается язык *Java*. На этом языке было написано огромное количество проектов под данную операционную систему, в том числе и с открытым исходным кодом, который можно свободно изучать. За такое большое время использования языка было создано огромное количество библиотек и готовых решений. Можно легко найти информацию о том, как решить ту или иную проблему. Однако в последнее время стал набирать популярность другой язык программирования – *Kotlin*, разработанный российской компанией *JetBrains*. С 2017 года данный язык является официальным языком для разработки *Android* приложений. Несмотря на то, что язык достаточно молодой, он уже обрел большую популярность. *Kotlin* обладает рядом преимуществ перед *Java*, вот некоторые из них:

- *null*-безопасность позволяет сократить количество *NullPointerException* до минимума;

- функции-расширения в *Kotlin* позволяет расширить функциональность любого класса без использования наследования;

- умное приведение типов;

- простой синтаксис и компактность программного кода;

- наличие классов для хранения данных (*data class*), которые автоматически генерируют функции *equals*, *hashCode*, *toString*.

Kotlin позволяет написать абсолютно то же самое, что и на *Java*, но гораздо компактнее, удобнее и проще. В настоящее время лучше всего для разработки *Android* приложения подойдет именно этот язык. Исходя из вышеперечисленного, было принято решение создавать приложение на нем.

Для создания нейронной сети будет использоваться язык программирования *Python*. Он подходит для быстрого создания и обучения

моделей нейронной сети. Также для *Python* существует большой выбор фреймворков машинного обучения.

Так как нейронная сеть будет писаться и тестироваться на языке *Python*, для удобного взаимодействия с ней будет использоваться такой же язык для написания сервера.

1.3.2 Выбор фреймворка машинного обучения

Существует большое количество фреймворков для машинного обучения, такие как *TensorFlow*, *Keras*, *Pytorch*, *MXNet* и т.д. В последнее время становится все более популярным фреймворк *Pytorch* от компании *Facebook*. Этот фреймворк базируется на библиотеке *Torch* и имеет открытый исходный код. Он отлично подходит для решения множества задач машинного обучения, таких как классификация и сегментация изображения, поиск объектов на изображении, анализ аудио- и текстовой информации. Также присутствует большое количество обученных моделей, которые можно натренировать на собственных данных.

1.3.3 Инструмент для аннотации картинок

Любая нейронная сеть работает с данными. В поставленной задаче необходимо определить по фото, какие продукты включает в себя блюдо. Для этого необходимо собрать набор из картинок и написать нейронную сеть, которая будет заниматься поиском объектов на фотографии и сегментацией изображения. Самый простой путь решения задачи – создать аннотацию для каждой картинке, в которой будет содержаться вся необходимая информация об ограничивающей рамке изображения или сегменте, который относится к тому или иному продукту.

Make Sense – сервис, позволяющий создавать аннотации к картинкам. Данный продукт имеет полностью открытый исходный код, интуитивно понятный интерфейс и обладает всем необходимым функционалом для простого и быстрого создания аннотаций к изображениям. Возможно создание сегментов для изображения, а также добавление ограничивающих

рамок. Для аннотаций, включающих в себя сегменты, поддерживается экспорт в двух форматах – *VGG* и *COCO*. Для ограничивающих рамок поддерживается экспорт в форматы *CSV*, *YOLO* и *VOC XML*. Формат *COCO* включает в себя набор пар ключ:значение. Каждому изображению соответствуют данные, включающие в себя сегменты изображения, координаты ограничивающей рамки, название файла и др. Интерфейс инструмента *Make Sense* представлен на рисунке 1.5.

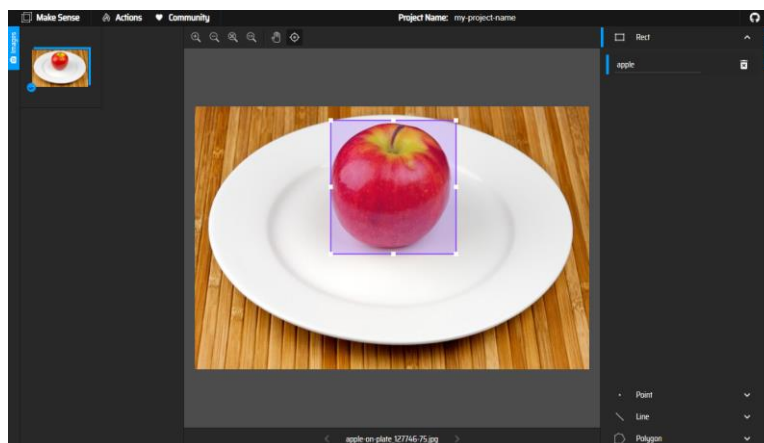


Рисунок 1.5 - Инструмент *Make Sense*

1.3.4 Выбор среды программирования

Среди всех сред для создания *Android* приложений наиболее популярными являются следующие:

- *Eclipse*;
- *IntelliJIDEA*;
- *Android Studio*.

Рассмотрим каждую из них по отдельности и выберем наиболее подходящую для создания проекта.

Eclipse – среда для создания кроссплатформенных приложений от *Eclipse Foundation*. До появления официальной среды *Android Studio* от *Google* она являлась основным инструментом для создания приложений. Данная среда показывает отличную производительность даже на слабых машинах и обладает широким набором инструментов для создания

приложений. Однако с появлением *Android Studio* компания *Google* прекратила поддержку данной среды как основной для разработки *Android* приложений.

IntelliJIDEA – среда от российской компании *JetBrains*. Удобная среда с встроенной поддержкой *Android*. Обладает большим набором инструментов для разработки проектов. Также имеет большое количество плагинов, которые помогут облегчить разработку, улучшить удобство пользования программой и добавить большое количества новых функций, позволяющих ускорить разработку проекта. Есть возможность разрабатывать проект вместе с другими людьми в режиме реального времени. Присутствует многофункциональный отладчик, который позволяет во время прерывания программы выполнять код, что является отличным инструментом для тестирования и отладки написанного кода. Из недостатков можно отметить требования к наличию производительной системы для комфортной работы в программе.

Android Studio – официальная среда для разработки *Android* приложений от компании *Google*. В настоящее время это лучший выбор для разработки проекта. Эта бесплатная среда от компании *Google* базируется на *IntelliJIDEA* компании *JetBrains*, обладает всеми вышеописанными преимуществами этой среды и отлично подходит для создания мобильных приложений. В среде присутствует встроенный эмулятор *Android*, который удобно использовать для тестирования и проверки работоспособности приложения. *IDE* регулярно обновляется и привносит большое количество полезных изменений. Также присутствуют специальные инструменты для просмотра базы данных и измерения производительности. Одним из недостатков среды является высокие требования к системе.

Для выбора среды из вышеперечисленных была составлена таблица 1.2, которая показывает достоинства и недостатки программ.

Таблица 1.2 - Достоинства и недостатки различных сред разработки

Среда	Достоинства	Недостатки
<i>Eclipse</i>	Обладает хорошей производительностью и отлично показывает себя на слабых компьютерах. Так как среда существует много лет, в интернете можно найти много информации по ее использованию и настройке, а также большое количество документации	Данная среда является неактуальной, так как компания <i>Google</i> прекратила поддержку <i>Eclipse</i> и плагинов для нее.
<i>IntelliJIDEA</i>	Обладает встроенной поддержкой <i>Android</i> . Есть широкий набор инструментов для быстрой и эффективной разработки проекта. В среде есть многофункциональный отладчик, который позволяет отслеживать значения переменных и выполнять код во время того, как программа была прервана, дойдя до точки останова. Большой выбор плагинов, с помощью которых можно настроить среду под себя.	Для того, чтобы было удобно пользоваться программой, необходима производительная система. Некоторые функции среды недоступны в бесплатной версии.
<i>Android Studio</i>	Встроенный эмулятор, с помощью которого удобно проверять работоспособность приложения, не подключая реальное устройство. Наличие инструментов для измерения производительности приложения, а также просмотра базы данных в режиме реального времени. Частые обновления, которые всячески улучшают среду и облегчают разработку.	Для комфортной работы в среде необходима производительная система.

Исходя из вышеперечисленных особенностей и преимуществ различных сред, было принято решение использовать *Android Studio* для разработки проекта.

Создание нейронной сети будет происходить с использованием языка *Python*. *Jupyter Notebook* – веб-приложение, которое позволяет выполнять программы на *Python*. Одной из особенностей этой среды является то, что весь код можно разбить на отдельные части и после выполнять и тестировать их в любом удобном порядке. Однако для обучения нейронной сети может потребоваться большие вычислительные мощности. Поэтому было принято решение воспользоваться сервисом *Google Colab*, который может запускать *jupyter*-ноутбук напрямую в браузере, а также предоставляет доступ к графическим ускорителям и нейронным процессорам, которые могут быть полезны при обучении нейронной сети. Главным преимуществом этого сервиса является то, что все вышеперечисленные функции предоставляются бесплатно. Предоставляемый графический процессор представлен на рисунке 1.6.

NVIDIA-SMI 460.32.03			Driver Version: 460.32.03			CUDA Version: 11.2						

GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile Uncorr. ECC							
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M.	MIG M.					
=====												
0	Tesla K80	Off	00000000:00:04:0	Off	0							
N/A	52C	P8	29W / 149W	0MiB / 11441MiB	0%	Default	N/A					

Processes:												
GPU	GI	CI	PID	Type	Process name	GPU Memory						
	ID	ID				Usage						
=====												
No running processes found												

Рисунок 1.6 - Параметры графического процессора в *Google Colab*

Для создания сервера также необходимо выбрать среду. *Jupyter notebook*, описанный выше, для таких задач не подходит, т.к. не обладает всеми инструментами характерными для большинства *IDE*, например, у него отсутствует встроенный отладчик и возможность рефакторинга кода. По

причине отсутствия встроенного отладчика не подходят и редакторы кода, такие как *Visual Studio Code*, *Atom* или *Sublime Text*.

Spyder – среда для *Python*, ориентированная на научных расчетов. Данная среда обладает подсветкой синтаксиса, автоматическое дополнением кода и возможностью просмотра значений переменных. Особенность данной среды заключается в том, что она ориентирована на определенную задачу – анализ данных.

Другая *IDE* – *PyCharm*, разработанная российской компанией *JetBrains*. Она является кроссплатформенной и предоставляет широкий набор инструментов для удобной разработки на языке программирования *Python*. Также *IDE* обладает всеми преимуществами других сред для разработки от компании *JetBrains* – есть удобный отладчик, рефакторинг, автоматическое дополнение кода, наличие большого количества плагинов, ускоряющих работу и многое другое.

Для выбора среды была составлена таблица 1.3, показывающая достоинства и недостатки вышеперечисленных программ.

Таблица 1.3 - Достоинства и недостатки сред для разработки

Среда	Достоинства	Недостатки
<i>Jupyter notebook</i>	Быстрое выполнение и тестирование частей кода в любом удобном порядке.	Отсутствует встроенный отладчик, нет возможности рефакторинга. По этим причинам не подходит для разработки сервера.
<i>Spyder</i>	Подсветка синтаксиса, автоматическое дополнение кода, многофункциональный отладчик.	В основном предназначен для научных расчетов. Сложная настройка среды, устаревший интерфейс.
<i>PyCharm</i>	Большой выбор плагинов, многофункциональный отладчик, удобное представление проекта в программе.	Для комфортной работы требуется производительная система. Некоторые функции недоступны в бесплатной версии программы.

Исходя из анализа вышеперечисленных достоинства и недостатков различных сред для разработки, было принято решение остановиться на среде *PyCharm*.

1.4 Разработка программы

1.4.1 Анализ фотографии и расчет калорий

На стороне сервера будет происходить анализ фотографии, присланной пользователем. Вместе с фотографией пользователь отправляет диаметр тарелки, на которой находится еда. Для расчета калорий будет использоваться метод приближения веса объекта. Заключается этот метод в нахождении веса, используя среднее отношения веса продукта к занимаемой площади. Таблица со значениями была получена путем определения площади с помощью нейронной сети и дальнейшего взвешивания объекта на весах. Используя эту таблицу, можно определить примерный вес любого продукта, зная площадь.

Процесс анализа разделен на несколько этапов. Рассмотрим их подробнее. На первом этапе фотография пропускается через нейронную сеть, которая найдет тарелку на изображении, а именно – определит ограничивающую рамку для объекта. Таким образом, можно узнать диаметр тарелки в пикселях. Расстояние одного пикселя *pixel_width_cm* будет найдено по следующей формуле:

$$pixel_width_cm = plate_width / plate_width_px,$$

где *plate_width* – диаметр тарелки в см;

plate_width_px – диаметр тарелки в пикселях.

Дальше легко посчитать общую площадь изображения *full_area*:

$$full_area = width * pixel_width_cm * height * pixel_width_cm,$$

где *width* – ширина изображения в пикселях;

height – высота изображения в пикселях.

Найденное значение площади понадобится позже для определения веса продукта.

Второй этап – анализ состава блюда. Изображение пропускается через вторую нейронную сеть, которая проводит сегментацию изображения. Сегментация изображения – процесс, в котором каждый пиксель изображения помечается классом, в которому он относится. В ходе этого процесса для каждого составляющего блюда создается маска. По маске можно найти площадь, занимаемую продуктом. Для начала определим отношение занимаемой площади объекта к общей площади:

$$ratio = food_pixels / image_pixels,$$

где *food_pixels* – количество пикселей в маске изображения;

image_pixels – количество пикселей в изображении.

Зная это отношение, легко вычислить реальное значение площади занимаемого объекта *food_area_cm2*:

$$food_area_cm2 = full_area * ratio,$$

где *full_area* – площадь изображения, определенная на первом этапе анализа.

Вес продукта в граммах *food_weight_gram*, находится по формуле:

$$food_weight_gram = food_area_cm2 * food_gram_per_area,$$

где *food_gram_per_area* – значение из таблицы, показывающее среднее отношение веса к площади продукта.

Остается только посчитать количество калорий *food_calories*:

$$food_calories = food_weight_gram * calorie_capacity / 100,$$

где *calorie_capacity* – количество калорий в продукте на 100 грамм.

1.4.2 Нейронная сеть

Для того, чтобы определить состав блюда на изображении и рассчитать количество калорий, необходимо разработать две нейронные сети. Первая будет осуществлять поиск тарелки на фото, вторая – создавать маски для объектов еды на фотографии.

Для определения состава блюда и создания масок была разработана нейронная сеть, выполняющая сегментацию изображения. Разработка велась с использованием фреймворка *Pytorch* [1]. В совокупности для тренировки и тестирования нейронной сети, было собрано около 100 картинок на каждый тип еды. Далее для каждой картинке была выполнена сегментация с помощью инструмента *Make Sense*. В итоге был получен файл с аннотациями в формате *JSON*, который содержит информацию о том, какие объекты находятся на фотографии, координаты ограничивающей рамки, данные о сегментах и др. Формат представлен на рисунке 1.7.

```
"annotations": [{
  "id": 0,
  "iscrowd": 0,
  "image_id": 1,
  "category_id": 1,
  "segmentation": [
    [248.8943488943489, 310
  ],
  "bbox": [214.004914004914,
  "area": 78218.8845088108
```

Рисунок 1.7 - Файл с аннотациями для картинок

Далее был разработан класс *FoodDataset*, представляющий набор данных. Код представлен ниже.

```
class FoodDataset(torch.utils.data.Dataset):
    def __init__(self, root, annotation, w, h,
transforms=None):
        self.root = root
        self.transforms = transforms
        self.coco = COCO(annotation)
        self.ids = list(sorted(self.coco.imgs.keys()))
        self.width = w
        self.height = h

    def __getitem__(self, index):
        coco = self.coco
        img_id = self.ids[index]
```

```

ann_ids = coco.getAnnIds(imgIds=img_id)
coco_annotation = coco.loadAnns(ann_ids)
path = coco.loadImgs(img_id)[0]['file_name']

img = Image.open(os.path.join(self.root,
path)).convert("RGB")

masks = coco.annToMask(coco_annotation[0])
num_objs = len(coco_annotation)

boxes = []
categories = []
for i in range(num_objs):
    x_min = coco_annotation[i]['bbox'][0]
    y_min = coco_annotation[i]['bbox'][1]
    x_max = x_min + coco_annotation[i]['bbox'][2]
    y_max = y_min + coco_annotation[i]['bbox'][3]
    boxes.append([x_min, y_min, x_max, y_max])
    category_id = coco_annotation[i]['category_id']
    categories.append(category_id)

boxes = torch.as_tensor(boxes, dtype=torch.float32)
masks = torch.as_tensor(masks, dtype=torch.float32)
categories = torch.as_tensor(categories,
dtype=torch.int64)

img_id = torch.tensor([img_id])
areas = []
for i in range(num_objs):
    areas.append(coco_annotation[i]['area'])
areas = torch.as_tensor(areas, dtype=torch.float32)
iscrowd = torch.zeros((num_objs,),
dtype=torch.int64)

my_annotation = {}
my_annotation["boxes"] = boxes
my_annotation["labels"] = categories

```

```

my_annotation["image_id"] = img_id
my_annotation["area"] = areas
my_annotation["iscrowd"] = iscrowd

if self.transforms is not None:
    img, my_annotation = self.transforms(img,
my_annotation)

return img, my_annotation

def __len__(self):
    return len(self.ids)

```

Для увеличения набора данных были созданы трансформации, которые использовались при его создании [2]. Далее были определены загрузчики данных, которые представляют обертку для набора данных. Данные в наше нейронную сеть будут передаваться не все целиком, а по частям, например по 3 картинки за раз. Для этого определяем *batch_size* равный 3. Код представлен ниже.

```

train_transforms = T.Compose([T.ToTensor(),
T.RandomHorizontalFlip(0.5)])
test_transforms = T.Compose([T.ToTensor()])

dataset = FoodDataset(train_dir, train_annotation, 600, 400,
train_transforms)
dataset_test = FoodDataset(test_dir, test_annotation, 600,
400, test_transforms)

data_loader = torch.utils.data.DataLoader(
    dataset,
    batch_size=3,
    shuffle=True
)

```

```

data_loader_test = torch.utils.data.DataLoader(
    dataset_test,
    batch_size=3,
    shuffle=False
)

```

Одним из несомненных преимуществ *Pytorch* является большое количество обученных моделей. Для поставленной задачи взята готовая обученная модель *Mask R-CNN*, которая используется для сегментации изображения. В качестве метода оптимизации был использован стохастический градиентный спуск (*SGD*). Для настройки скорости обучения используется *lr_scheduler*, а именно *StepLR*, который будет изменять скорость обучения сети с равными интервалами. Тренировка проходила с использованием технологии *CUDA*, которая позволяет ускорить процесс обучения с помощью графических процессоров [3]. Код представлен ниже.

```

def get_model_instance_segmentation(num_classes):
    model =
torchvision.models.detection.maskrcnn_resnet50_fpn(pretrained=True)

    in_features =
model.roi_heads.box_predictor.cls_score.in_features
    model.roi_heads.box_predictor =
FastRCNNPredictor(in_features, num_classes)

    in_features_mask =
model.roi_heads.mask_predictor.conv5_mask.in_channels
    hidden_layer = 256
    model.roi_heads.mask_predictor =
MaskRCNNPredictor(in_features_mask,
hidden_layer,
num_classes)

```

```

return model

device = torch.device('cuda') if torch.cuda.is_available()
else torch.device('cpu')
print(device)

num_classes = 6
model = get_object_instance_segmentation_model(num_classes)
model.to(device)
params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(params, lr=0.005, momentum=0.9,
weight_decay=0.0005)

lr_scheduler = torch.optim.lr_scheduler.StepLR(
    optimizer,
    step_size=3,
    gamma=0.1
)
num_epochs = 15

for epoch in range(num_epochs):
    train_one_epoch(model, optimizer, data_loader,
device, epoch, print_freq=10)
    lr_scheduler.step()
    evaluate(model, data_loader_test, device=device)

```

После 15 эпох обучения модели значение потери стало равно 0,0336. В результате применения нейронной сети было получен список категорий, которая она нашла на картинке, координаты ограничивающей рамки и маска для элемента изображения. Нейронная сеть распознает на изображении яблоки, яичницу-глазунью, помидор, хлеб и рис. Результат работы нейронной сети представлен на рисунке 1.8.

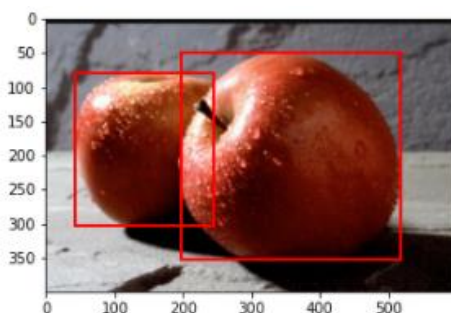


Рисунок 1.8 - Результат поиска объектов на фото

Результат сегментации изображения представлен на рисунке 1.9.

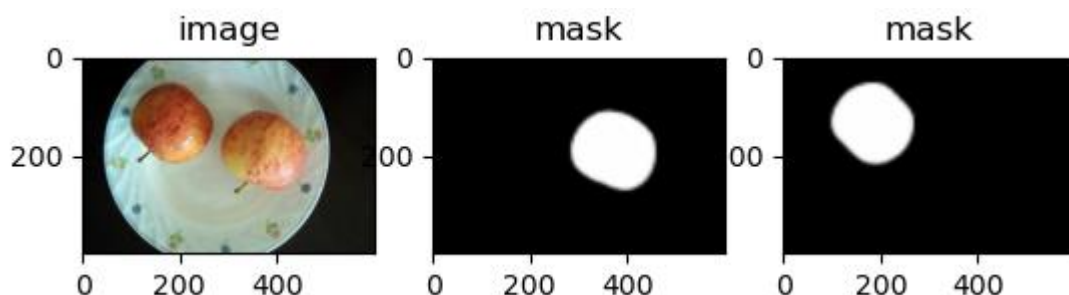


Рисунок 1.9 - Результат сегментации изображения

Для поиска тарелки на изображении была разработана еще одна нейронная сеть. За основу взята готовая модель *Faster R-CNN*, предназначенная для поиска объектов на изображении. Разработка сети аналогична предыдущей, однако при аннотации изображения использовалась только ограничивающая рамка. Результат работы представлен на рисунке 1.10.



Рисунок 1.10 - Результат нахождения тарелки на изображении

1.4.3 Разработка сервера

Создание сервера происходила на языке *Python* с помощью фреймворка *Flask* [4]. С помощью этого инструмента был разработан *REST API* для получения необходимых данных от сервера. *REST API* – архитектура взаимодействия для распределенного приложения с помощью протокола *HTTP*. Данная архитектура очень популярна для построения клиент-серверных приложений. Данные, которые будет получать клиент, будут в формате *JSON*. Этот формат представляет данные в виде пар ключ: значение и в виде упорядоченного набора. Было создано 4 файла *JSON*, которые хранят в себе список рецептов, небольших статей с советами по питанию, диетами, ингредиентов для блюд с полной информацией о калорийности и количестве белков, жиров и углеводов. Формат всех этих данных очень схож, для рецепта представлен на рисунке 1.11.

```
"receipts": [{
  "id": 1,
  "title": "Яичница",
  "previewUrl": "receipts/omelette.png",
  "ingredients": ["Яйца", "Растительное масло", "Соль", "Зелень"],
  "description": "Положить/вылить на раскалённую сковороду масло, н",
  "timeMin": 3,
  "calories": 243
```

Рисунок 1.11 - Формат данных для рецептов

Ниже приведен фрагмент кода для запроса на получения всех рецептов. Для других запросов методы аналогичны, отличие заключается лишь в названии и передаваемых данных.

```
receipts = {}
with open('receipts.json', 'r', encoding='utf-8') as jsf:
    receipts = json.load(jsf)

@app.route('/receipts', methods=['GET'])
def get_receipts():
    public_receipts = {'receipts': list(map(set_item_url_,
```

```

receipts['receipts']))}

    return jsonify(public_receipts)

@app.route('/search', methods=['GET'])
def search():
    title = request.args.get('title', default='',
type=str).lower()
    ingredient_list =
request.args.to_dict(flat=False).get('ingredients')
    print(ingredient_list)
    filtered = list(filter(lambda x: title in x['title'],
receipts['receipts']))
    if ingredient_list is not None:
        filtered = list(filter(lambda x:
set(ingredient_list).issubset(set(x['ingredients'])), filtered))
    public_receipts = {'receipts': list(map(set_item_url,
filtered))}

    return jsonify(public_receipts)

def set_item_url(item, url_key='previewUrl'):
    public_items = {}
    for key in item:
        if key == url_key:
            public_items[key] = url_for('static',
filename=item[key], _external=True)
            print(public_items[key])
        else:
            public_items[key] = item[key]
    return public_items

app = Flask(__name__)
if __name__ == '__main__':
    app.run(debug=True, port=PORT)

```

Также был написан метод для загрузки картинки на сервер для дальнейшего анализа. Код представлен ниже.

```
@app.route('/upload', methods=['POST'])
def upload():
    img = request.json
    data = base64.b64decode(img ['img'])
    plate_diameter = img['diameter']
    with open('static/uploads/'+str(uuid.uuid4())+'.jpg', 'wb')
as f:
    f.write(data)
    f.close()
    result = food_analysis(f.name, plate_diameter)

    return jsonify(result)
```

Функция *food_analysis* предназначена для анализа фотографии пользователя. В таблице 1.4 представлены результаты вычисления веса некоторых продуктов по фотографии и сравнение с реальным весом, который был определен путем взвешивания на весах. Отклонение показывает, насколько вычисленный вес продукта отличается от реального.

Таблица 1.4 – Результат вычисления веса продуктов

Еда	Реальный вес, г	Вычисленный вес, г	Отклонение, %
Яблоко	156	170	8,9
Яичница-глазунья	48	51	6,25
Белый рис	112	126	12,5
Черный хлеб	39	40	2,5
Помидор	152	139	8,5

1.4.4 Разработка клиента

В качестве клиента было разработано мобильное приложение. Для взаимодействия с сервером была использована библиотека *Retrofit*. Был создан интерфейс *FoodAPI*, который включает в себя все функции для

получения данных от сервера и отправку фотографий. Ниже представлен фрагмент кода для отправки картинки на сервер и получения списка всех рецептов. Для получения остальных данных (список ингредиентов, диет, статей) использовались аналогичные методы.

```
Interface FoodAPI {  
    ...  
    @POST("upload")  
    fun uploadPhoto(  
        @Body img: String  
    ): Call<DetectedImageResponse>  
  
    @GET("receipts")  
    suspend fun getAllReceipts(): ReceiptsResponse  
  
    data class ReceiptsResponse(  
        val receipts: List<ReceiptResponse>  
    )  
  
    @Parcelize  
    data class ReceiptResponse(  
        val id: Int,  
        val title: String,  
        val previewUrl: String,  
        val ingredients: List<String>,  
        val description: String,  
        val timeMin: Int,  
        val calories: Int  
    ) : Parcelable
```

При запуске приложения, пользователю требуется заполнить небольшую анкету, указав свой рост, вес, возраст, желаемый вес, уровень физической активности и пол. Введенные данные будут в дальнейшем использоваться для подсчета калорий, которые человек должен ежедневно

потреблять в день. Анкета отображается только один раз, когда пользователь впервые открыл приложение. Интерфейс экрана показан на рисунке 1.12.

4:21

MyCalories

Прежде, чем приступить к работе с приложением, заполните небольшую анкету, чтобы мы могли подобрать вам персональную программу

Вес(кг)
78

Рост(см)
195

Цель (в кг)
85

Возраст
25

Низкая активность

Пол: Мужской

ПРОДОЛЖИТЬ

Рисунок 1.12 - Анкета для пользователя

Основной интерфейс состоит из 4 доступных вкладок: главная, рецепты, калории и вес. На первой вкладке показывается количество калорий, которые человек потребил за сегодняшний день, число калорий, которое он должен потребить. Эти данные отображаются на круговом элементе, чтобы пользователь мог визуально оценить количество калорий,

которое он потребил за день. Также отображаются данные, которые были получены из анкеты. Эти данные можно в любой момент изменить, кликнув по соответствующему элементу на интерфейсе [5]. Снизу расположен небольшой список с дополнительными опциями. Пользователь может ознакомиться с диетами, подробной информацией о продуктах, советах, а также посчитать индекс массы тела. Интерфейс представлен на рисунке 1.13.

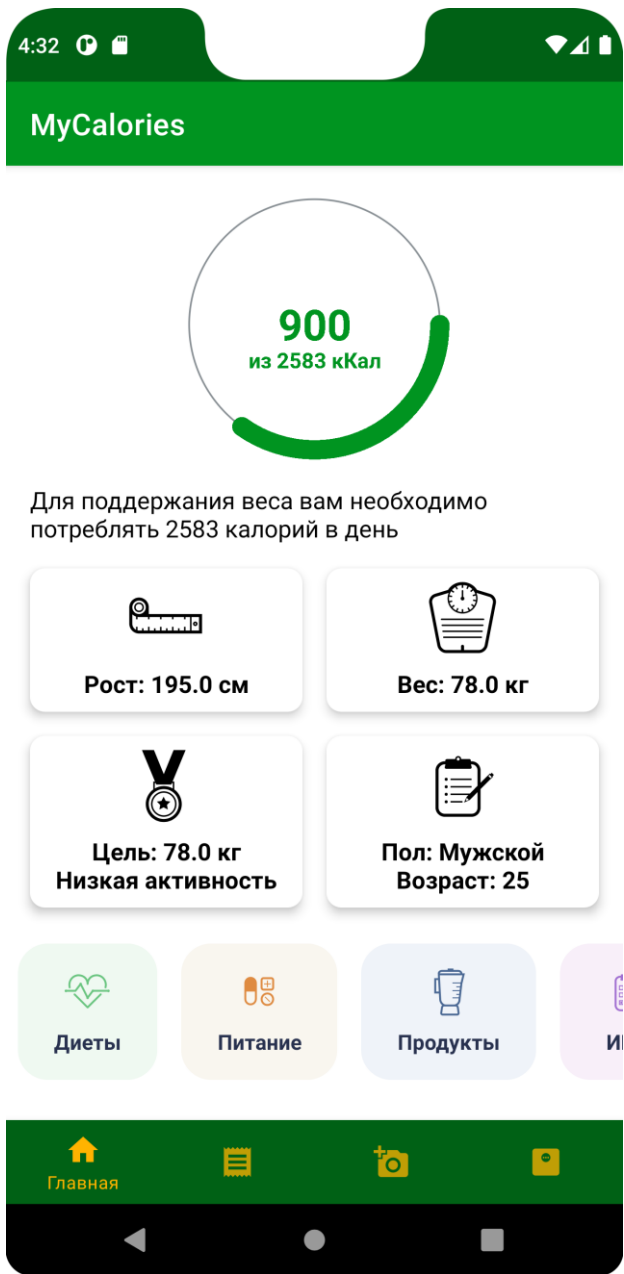


Рисунок 1.13 - Вкладка «Главная»

Расчет необходимых калорий идет с помощью формулы Миффлина Сан-Жеора, которая была разработана диетологами для вычисления оптимального количества калорий для поддержания веса. Выглядит данная формула следующим образом для мужчин и женщин:

$$\begin{aligned} \text{calories} &= 10 * \text{weight} + 6.25 * \text{height} - 5 * \text{age} + 5, \\ \text{calories} &= 10 * \text{weight} + 6.25 * \text{height} - 5 * \text{age} - 161, \end{aligned}$$

где *calories* – количество калорий для поддержаний веса;

weight – вес человека в килограммах;

height – рост человека в сантиметрах;

age – возраст человека.

Данная формула хороша тем, что учитывает различные характеристики человека, однако позднее данная формула была доработана для того, чтобы получить более точное значение калорий, исходя из физической активности человека. Формула стала выглядеть следующим образом для мужчин и женщин:

$$\begin{aligned} \text{calories} &= (10 * \text{weight} + 6.25 * \text{height} - 5 * \text{age} + 5) * A, \\ \text{calories} &= (10 * \text{weight} + 6.25 * \text{height} - 5 * \text{age} - 161) * A, \end{aligned}$$

где *A* – коэффициент физической активности человека.

Всего различают 5 ступеней, которые представлены в таблице 1.5.

Таблица 1.5 - Ступени физической активности

Ступень активности	Значение коэффициента <i>A</i>
Минимальная активность	1.2
Слабая активность	1.375
Средняя активность	1.55
Высокая активность	1.725
Очень высокая активность	1.9

Последняя ступень – очень высокая активность – в основном относится к людям, которые профессионально занимаются спортом. С помощью данной

таблицы и формулы можно определить примерное количество калорий, которое необходимо человеку для поддержания его веса. На языке *Kotlin* функция, которая определяет необходимое количество калорий выглядит следующим образом:

```
fun calculateRequiredCalories(weight: Float, height: Float,
age: Int, gender: Gender, activity: PersonActivity): Int =
    if (gender == Gender.Male) {
        ((10 * weight + 6.25f * height - 5 * age + 5) *
activity.getRatio()).toInt()
    } else {
        ((10 * weight + 6.25f * height - 5 * age - 161) *
activity.getRatio()).toInt()
    }
```

Первая опция – «Диеты». После нажатия на нее, откроется новый экран со списком диет. Изучив список, пользователь может подобрать подходящую диету для себя. Для каждой диеты отображается название и примерное время, которое необходимо для того, чтобы прочитать материал. Время *time* рассчитывается по формуле:

$$time = length / speed,$$

где *length* – длина текста в символах;

speed – средняя скорость чтения человека в минуту, равная 1400 знаков/минуту.

Опции «Питание» и «Продукты» визуально похожи на опцию «Диеты», но отображают другие данные. В разделе питание можно найти полезные советы, которые помогут поддерживать здоровый образ жизни, а также расскажут многое о вреде некоторых веществ, которые мы ежедневно употребляем в пищу. В разделе «Продукты» можно найти больше информации о конкретном продукте питания, в котором описываются его полезные свойства, плюсы и минусы употребления, возможный вред для здоровья, а также противопоказания к употреблению. Изучив эти небольшие

статьи, пользователь сможет скорректировать свое питание. Интерфейс представлен на рисунке 1.14.

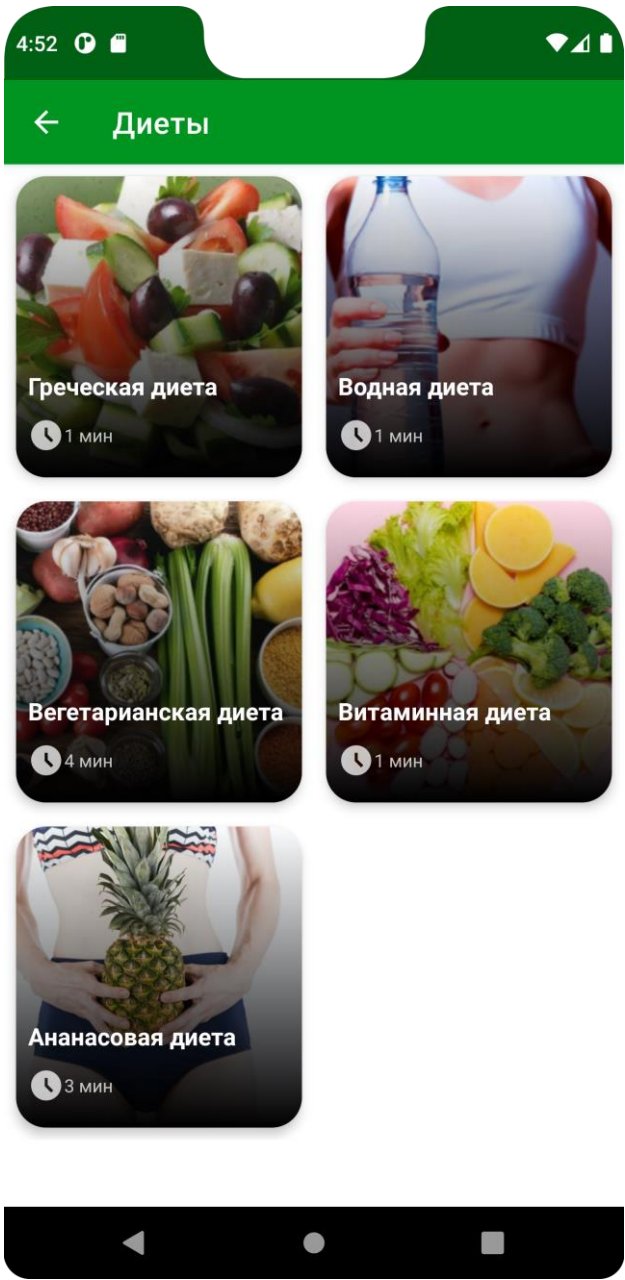


Рисунок 1.14 - Список диет

Интерфейс для списка статей о питании представлен на рисунке 1.15.

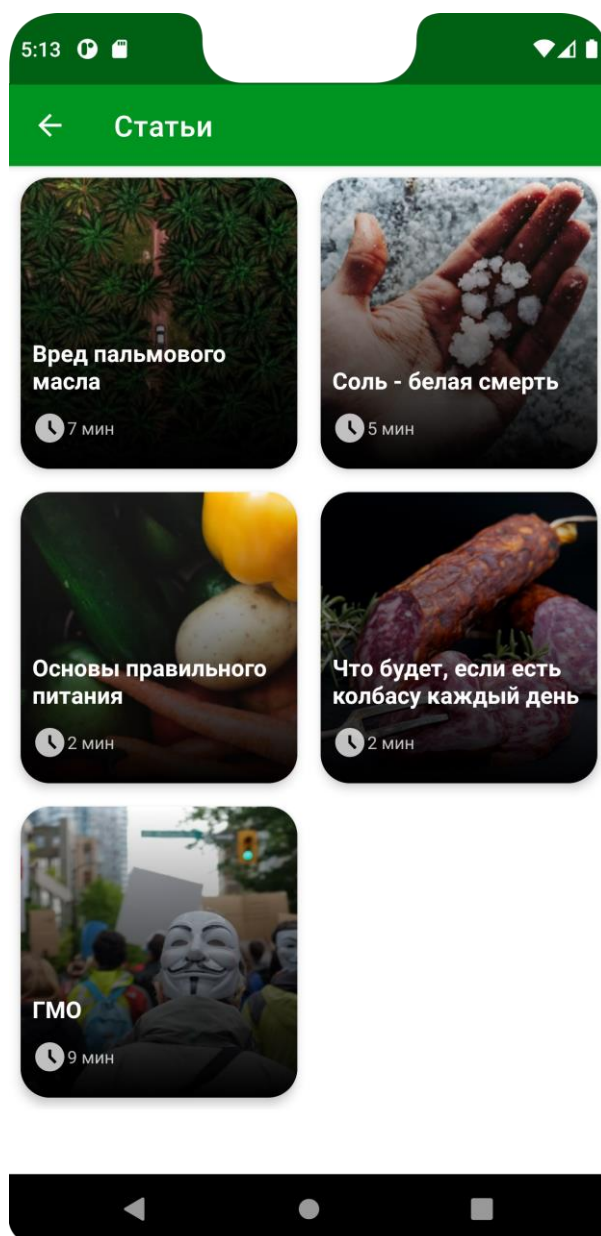


Рисунок 1.15 - Список статей о питании

Последняя опция вкладки «Главная» – «Индекс массы тела». На этой странице можно указать свой рост и вес для определения индекса массы тела, которая позволит человеку оценить собственный показатель соответствия веса к своему росту. Индекс *bmi* рассчитывается по формуле:

$$bmi = weight / (height * height),$$

где *weight* – вес человека в килограммах;

height – рост человека в метрах.

Изм.	Лист	№ докум	Подпись	Дата

После ввода необходимых данных на интерфейс отображается вычисленное значение. С помощью этого показателя, человек может узнать, в какой категории он относится, в соответствии с интерпретацией этого индекса всемирной организацией здравоохранения. Значение индекса массы тела не является идеальным, так как не учитывает возраст человека, пол, объем грудной клетки и другие показатели. Также он может неверно отображать результат для людей с развитой мускулатурой, однако для подавляющего большинства людей с обычным телосложением, такой показатель можно интерпретировать для обнаружения отклонений от нормы и оценки собственного веса. Интерфейс показан на рисунке 1.16.



Рисунок 1.16 - Индекс массы тела

Следующая вкладка – «Рецепты», включает в себя список с рецептами, которые пользователь может изучить. Для каждого рецепта отображается название, количество калорий в нем, а также время приготовления. Для удобства пользователя была добавлена функция поиска, которую можно активировать, нажав на иконку с лупой в верхней части экрана. Кроме того, список можно отфильтровать [6], нажав на иконку с тремя полосками и выбрав ингредиенты, которые обязательно должны присутствовать в блюде. Нажав на иконку с сердцем, покажутся все рецепты, которые пользователь добавил в избранное. Интерфейс вкладки представлен на рисунке 1.17.

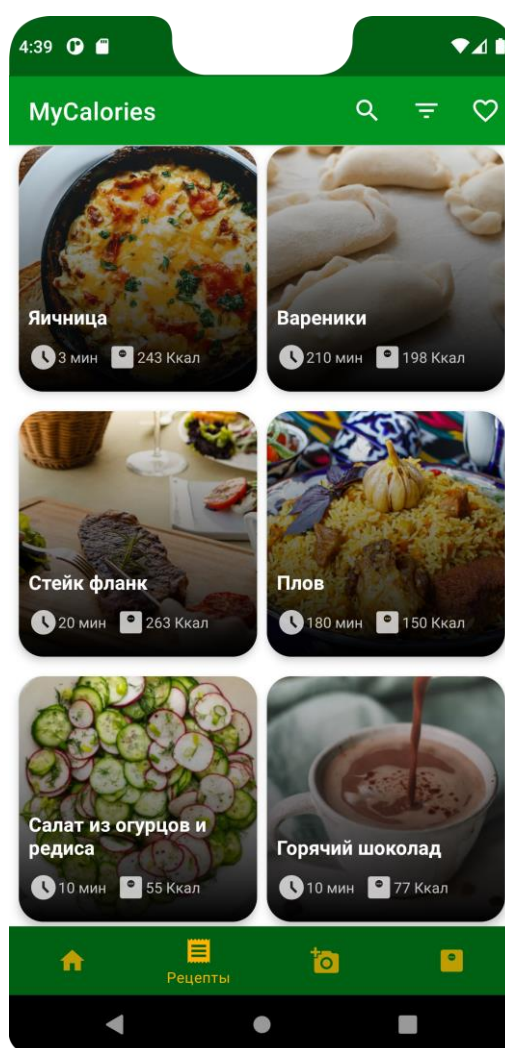


Рисунок 1.17 - Вкладка «Рецепты»

Для детального просмотра рецепта, необходимо нажать на него. После чего откроется новый экран, в котором будет отображаться информация,

которая поможет пользователю приготовить блюдо. Отображается список ингредиентов и способ приготовления. Каждый понравившийся рецепт можно добавить в избранное, нажав на иконку с сердцем [7]. Интерфейс экрана представлен на рисунке 1.18.

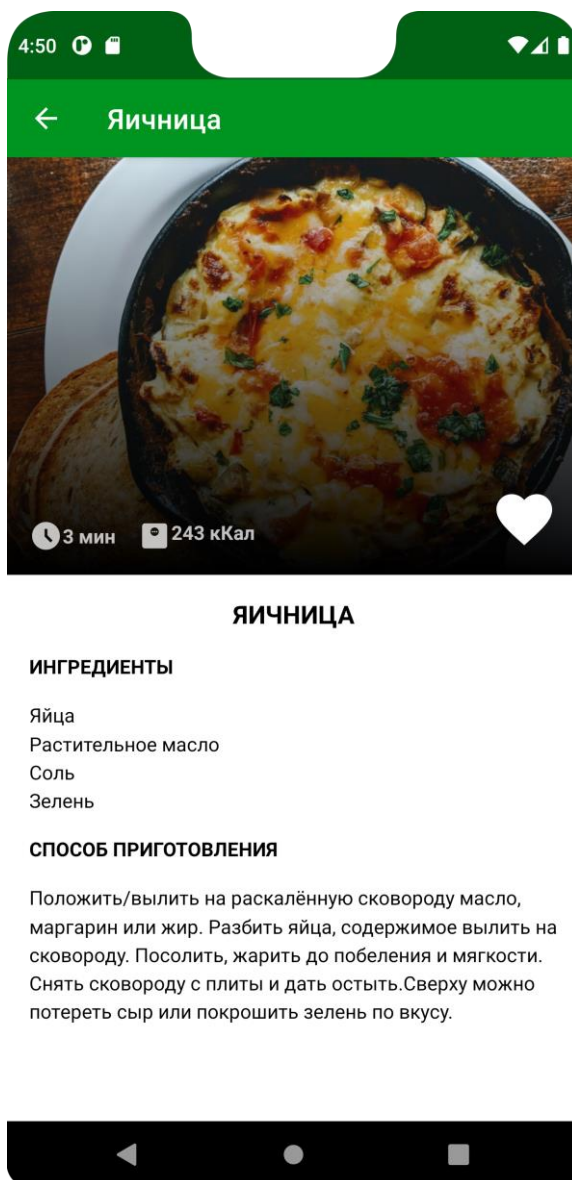


Рисунок 1.18 - Рецепт блюда

Вкладка «Калории» отображает список с проанализированными фотографиями, которые добавил пользователь. Каждый элемент списка относится к отдельному дню для удобства пользователя. Каждый элемент включает в себя дату, список с фотографиями пользователя, сделанный в день добавления, общее количество калорий для всех проанализированных

фото. В любой момент можно добавить очередную фотографию для анализа. Для этого служит большая круглая кнопка в правом нижнем углу экрана. Интерфейс вкладки представлен на рисунке 1.19.

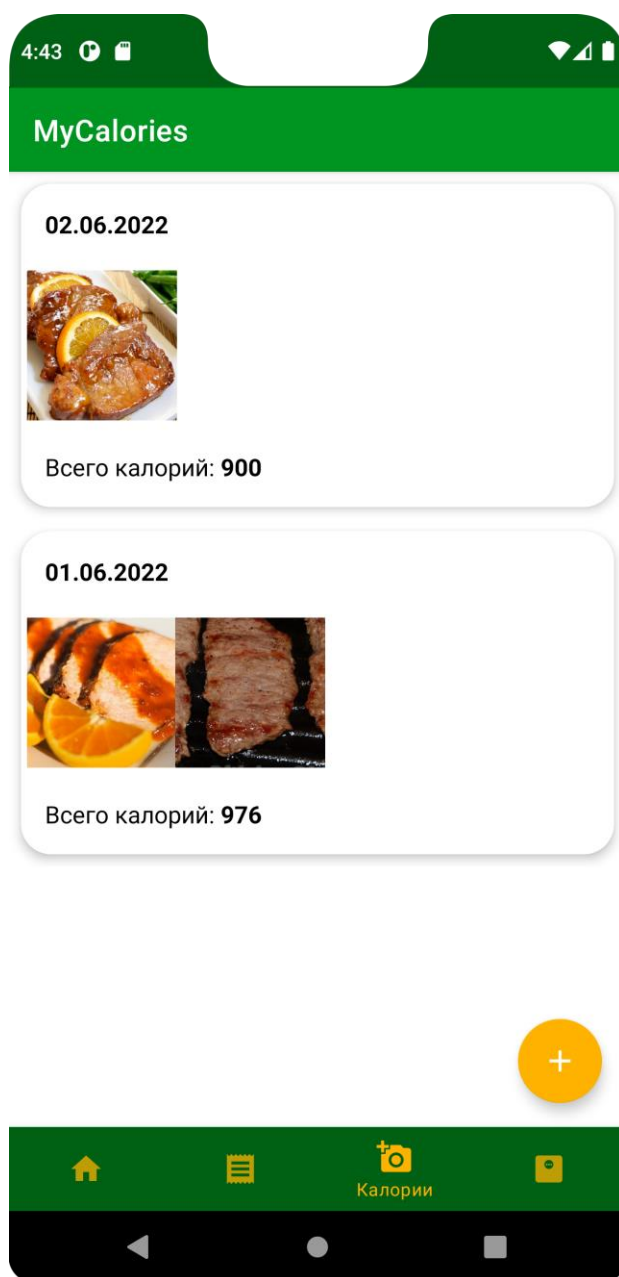


Рисунок 1.19 - Вкладка «Калории»

Также каждый элемент списка можно детально просмотреть, нажав на него. После выбора элемента, отобразятся все сделанные фотографии пользователем в определенный день, а также рядом с каждым изображением будет показано значение калорий, которое находится в блюде. В любой

момент можно добавить новый элемент, загрузив изображение с устройства, либо сделав фотографию. Интерфейс показан на рисунке 1.20.

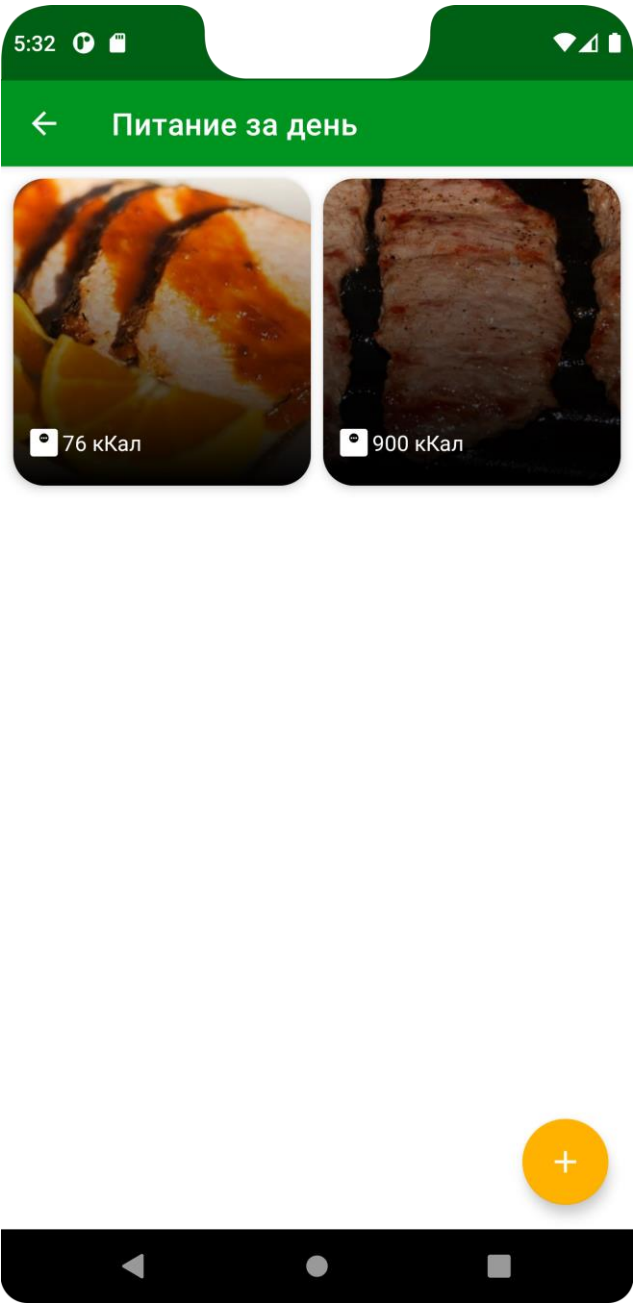


Рисунок 1.20 - Список с фотографиями за определенный день

Каждый элемент можно отредактировать, нажав на него. При этом отобразится загруженная фотография, состав продукта и диаметр тарелки, которую ввел пользователь. Диаметр и фотографию можно изменить. При нажатии на кнопку «Проанализировать» загруженное изображение и

введенный диаметр отправляются на сервер. Эти данные используются для определения состава блюда и калорийности каждого его составляющего. При анализе изображение определяется его примерный вес, однако такое значение не может быть стопроцентно точным. Поэтому у пользователя есть возможность ввести собственное значение веса в граммах. Для этого достаточно нажать на нужный продукт, что вызовет открытие диалогового окна, в котором нужно прописать значение. Как только диалоговое окно будет закрыто, количество калорий будет пересчитано. Интерфейс показан на рисунке 1.21.

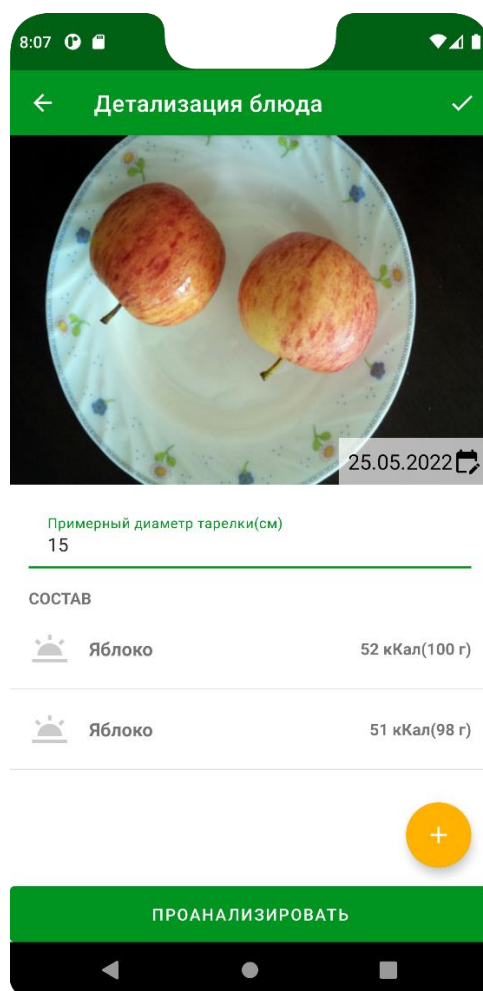


Рисунок 1.21 - Детализация блюда

После нажатия на кнопку с галочкой, все сделанные изменения сохраняются в базу данных. Сущности показаны на рисунке 1.22.

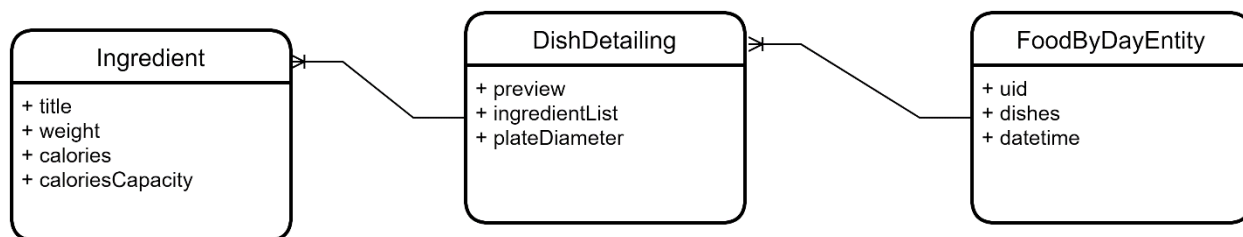


Рисунок 1.22 - Сущности базы данных

Так как представленные сущности являются сложными объектами, они сохраняются в базу данных в виде строки с помощью специальных классов-преобразователей. Эти классы представлены ниже:

```

class FoodByDayConverter {

    @TypeConverter
    fun dishDetailingListToJson(value:
MutableList<DishDetailing>): String = Gson().toJson(value)

    @TypeConverter
    fun jsonToDishDetailingList(value: String):
MutableList<DishDetailing> =
        Gson().fromJson(value,
Array<DishDetailing>::class.java).toMutableList()

}
  
```

Данные переводятся в строку формата *JSON*, а затем сохраняются в базу данных. При считывании данных происходит обратный процесс и строковые данные вновь переводятся в объект.

Вкладка «Вес» необходима для того, чтобы пользователь мог отслеживать изменения своего веса. Кнопка в правом углу экрана служит для добавления новой записи. В списке отображается вес в определенную дату, и его изменение по сравнению с предыдущим замером. В зависимости от поставленной цели, изменение веса будет отображаться разных цветом. Если пользователь хочет похудеть, любое уменьшение веса будет отображаться

зеленых цветом, а набор массы – красным. Если пользователь хочет набрать массу, то наоборот. В случае, когда пользователь не планирует менять свой вес, изменение будет отображаться желтым цветом. Интерфейс показан на рисунке 1.23.

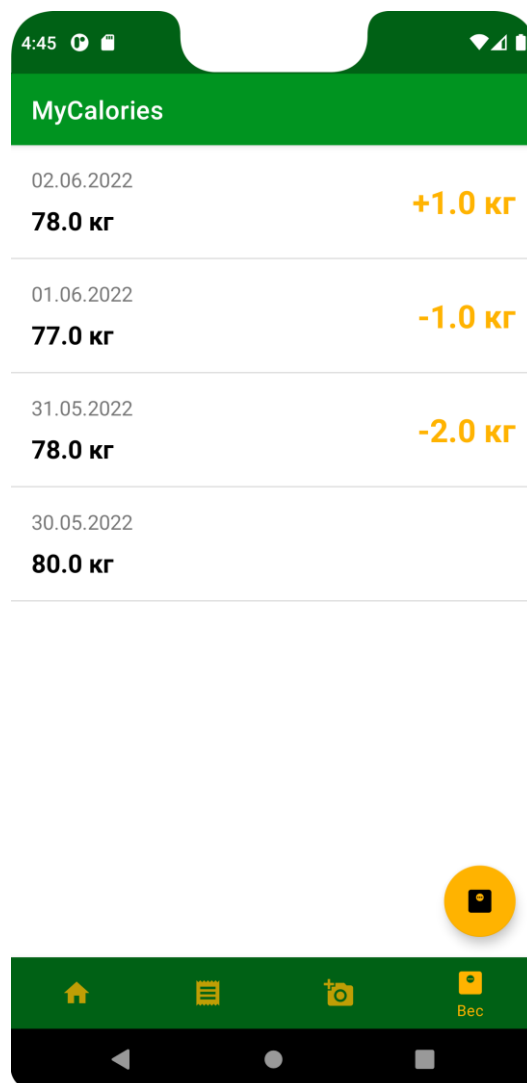


Рисунок 1.23 - Вкладка «Вес»

В приложении используется база данных *Room*. Для этого была описана сама сущность *WeightEntity* и интерфейс, описывающий операции для получения, удаления и добавления данных. Для хранения сложных типов данных, таких как класс *Date*, был создан специальный класс для преобразования сложного объекта в более простой тип, который можно хранить в базе данных. Код представлен ниже.

```

@Entity
data class WeightEntity(
    @ColumnInfo(name = "weight") val weight: Float,
    @ColumnInfo(name = "datetime") val datetime: Date
) {
    @PrimaryKey(autoGenerate = true) var uid: Int = 0
}

@Dao
interface WeightDao {

    @Query("SELECT * FROM weightEntity ORDER BY datetime DESC")
    fun getAll(): LiveData<List<WeightEntity>>

    fun insertItem(item: WeightEntity) {
        val result = getByDate(item.datetime)
        if(null != result) {
            item.uid = result.uid
            update(item)
        }
        else {
            insert(item)
        }
    }

    @Insert
    fun insert(item: WeightEntity)

    @Update
    fun update(item: WeightEntity)

    @Delete
    fun delete(item: WeightEntity)

    @Query("DELETE FROM weightEntity")
    fun deleteAll()

```

```

        @Query("SELECT * FROM weightEntity WHERE datetime = :date")
        fun getByDate(date: Date): WeightEntity?
    }

    class DateConverter {

        @TypeConverter
        fun fromTimestamp(value: Long?): Date? {
            return value?.let { Date(it) }
        }

        @TypeConverter
        fun dateToTimestamp(date: Date?): Long? {
            return date?.time
        }
    }

```

2 Безопасность жизнедеятельности

2.1 Характеристика опасных факторов при работе с ПЭВМ

При разработке проекта необходимо учитывать факторы, которые могут негативно сказаться на работе и производительности труда, поэтому одним из главных условий является создание и обеспечение комфортных условий при работе с ЭВМ.

Для того, чтобы эти условия обеспечить, нужно определить, какие факторы могут повлиять на разработку. Условия, в которых работает человек, определяются организацией рабочего пространства и условиями рабочей среды, куда можно отнести микроклимат, шум и другие вредные факторы.

При работе за компьютером на человека могут действовать следующие типы факторов:

- физические;
- психофизиологические;
- химические.

Наиболее часто возникают первые два типа факторов. Физические включают в себя:

- повышенные уровни рентгеновского, инфракрасного, а также ультрафиолетового излучения;
- повышенные уровни электромагнитных и электростатических полей;
- повышенная температура воздуха зоны, в которой работает человек;
- пониженная температура воздуха зоны, в которой работает человек;
- повышенная температура поверхностей компьютера;
- чрезмерная запыленность рабочей зоны;
- повышенная яркость монитора или света;
- повышенное напряжение в сети;
- отсутствие хорошей освещенности рабочего места или ее недостаток.

К психофизиологическим можно отнести следующим:

- напряжение зрения по время работы с компьютером;
- эмоциональные нагрузки;
- интеллектуальные нагрузки;
- умственное напряжение;
- перенапряжение мышц.

Очень часто во время работы с ЭВМ человек читает много текста с монитора и перенапрягает свои глаза. Также из-за длительного нахождения за компьютером в одной и той же позе может возникнуть статическое напряжение мышц, что может привести к ухудшению мозгового кровообращения или кровообращения в области пальцев рук и запястья. Во время работы также могут возникать головные боли, боли в области шеи, появляется усталость и слезоточивость глаз. Также усиленная интеллектуальная деятельность может приводить к нервно-психическим перегрузкам.

К химическим факторам можно отнести повышенное содержание различных токсичных и вредных веществ для организма человека, сюда можно отнести углекислый газ, аммиак, озон и другие вещества.

2.2 Классификация и категорирование помещения

Местом, в котором работает программист является офис. В ходе определения категории помещения по пожарной опасности, было выявлено, что рабочее место программиста относится к пожароопасной категории не предусматривающие возможность взрыва В1-В4 – в помещении находятся твердые горючие и тяжело воспламеняющиеся вещества и материалы, не относящиеся к категории А и Б. Категория В разделяется на 4 подгруппы по уровню удельной пожарной нагрузки q :

- В1, где q выше 2200 МДж/м²;
- В2, где q в пределах 1401-2200 МДж/м²;

- В3, где q в пределах 181-1400 МДж/м²;
- В4, где q в пределах 1-180 МДж/м².

Деятельность программиста относится к категории работ IA – легкие физические работы. В соответствии с установленными нормами, температура на рабочем месте должна быть в пределах от 22,9 до 25,1 °С в летнее время и 22-24 °С в зимнее. Относительная влажность воздуха должна быть в пределах 15-75%, температура поверхностей – 19-26 °С.

Обеспечение и поддержание необходимого уровня температуры является одной из обязанностей работодателя. Сотрудники в соответствии с трудовым кодексом Российской Федерации имеют право на комфортные условия труда.

Работодатель обязан использовать различные приборы для поддержания комфортных для работы температурных условий. Для этого используются кондиционеры и обогреватели.

Кондиционер – устройство, предназначенное для поддержания оптимальных температурных условий в помещении. В летнее время может использоваться для охлаждения и регулирования температуры.

Обогреватель – устройство, предназначенное для нагрева помещения и поддержания температуры воздуха. Используется в зимнее время года для обогрева помещения.

Кроме поддержания комфортных температурных условий, работодатель обязан выполнять следующие требования:

- обеспечение безопасности и условий труда;
- обеспечение работников необходимым оборудованием, инструментами, технической документацией;
- соблюдение правил пожарной безопасности;
- осуществление мер по обеспечению пожарной безопасности;
- поддержка различных приборов и систем в пригодном и исправном состоянии;

- поддержка систем и средств противопожарной безопасности;
- проводить обучение сотрудников правилам пожарной безопасности.

В соответствии с классификацией помещений по опасности поражения электрическим током рабочее место программиста относится к помещениям без повышенной опасности по следующим признакам:

- отсутствие токопроводящей пыли и полов;
- относительная влажность воздуха не превышает отметку в 75%;
- температура не превышает 30 °С.

2.3 Санитарно-гигиеническая характеристика

К основным показателям, которые характеризуют микроклимат относят следующие:

- температура воздуха;
- скорость движения воздуха;
- относительная влажность.

Работа программиста категории IА включает в себя следующие оптимальные требования и нормы:

- оптимальная температура 22-24 °С;
- относительная влажность 40-60%;
- скорость движения воздуха 0,1 м/с.

При работе с ЭВМ человек длительное время находится в неподвижном положении и часто напрягает глаза. Работа за компьютером происходит в основном с применением искусственного освещения, поэтому необходимо обеспечить сотрудников комфортными условиями с целью уменьшения напряжения на глаза.

Отсутствие или недостаток освещения в помещении может отрицательно сказаться на здоровье глаз, снижает работоспособность, ухудшает кровообращение, вызывает сильное напряжение, и может доставлять человеку сильный дискомфорт. Также это может служить

причиной сильной головной боли. После долгого рабочего дня может возникать чувство усталости и слезоточивости глаз.

При работе за компьютером необходимо соблюдать следующие требования:

- использование ламп для освещения рабочего места;
- сочетание локального освещения и общего;
- не допускать прямых и ярких попаданий солнечных лучей на монитор, так как это может повлиять на отображаемую картинку и вызвать искажения;
- не работать за компьютером в темном помещении;
- помещения в ЭВМ не должны находиться рядом с помещениями, в которых уровень шума и вибраций превышает нормальные значения;
- уровень шума на рабочем месте не должен превышать 50 дБА.

2.4 Электробезопасность

Электрические приборы, находящиеся в помещении, могут представлять для человека большую опасность, если с ними неправильно обращаться и не соблюдать правила электробезопасности – система мероприятий, которые могут обеспечить безопасность человека от губительного воздействия электричества.

Электробезопасность направлена на предотвращение различных угроз и вредного воздействия со стороны электрического тока на сотрудников. Воздействие электричества на человека может привести к серьезным травмам, к которым относятся ожоги, повреждение кровеносных сосудов и мышц, к остановке сердца.

Для обеспечения безопасности необходимо правильно обращаться с компьютером и устройствами, которые установлены на рабочем месте программиста (принтер, сканер, лампы для освещения, монитор). Необходимо выполнять следующие требования:

- не допускать попадания влаги на монитор, клавиатуру и другие электрические приборы;
- держать устройства подальше от нагревателей и других источников тепла;
- всегда проверять исправность проводов и разъемов;
- не ставить электрические устройства и приборы в места с повышенным уровнем пыли и влажности;
- не устанавливать приборы на поверхности, которые могут накапливать статическое электричество (органическое стекло);
- не нарушать температурные нормы и нормы относительной влажности воздуха.

2.5 Пожарная безопасность

При работе за компьютером необходимо проявлять осторожность и следовать правилам, которые помогут избежать возникновения опасных ситуаций и угроз. Пожарная безопасность – комплекс мер, которые направлены на предотвращение угроз возникновения пожара.

Выделяют следующие правила, которые необходимо соблюдать:

- не пользоваться приборами и проводами, которые имеют поврежденный вид;
- не курить в помещениях, а использовать для этого специально отведенные места;
- не устанавливать приборы рядом с нагревателями и другими источниками тепла;
- обучать сотрудников правилам пожарной безопасности.

В случае возникновения пожара необходимо предпринять следующие действия:

- по возможности отключить оборудование от сети;
- по возможности эвакуировать материально-ценные вещи;

- сообщить о произошедшем соответствующим лицам (руководителю, пожарным).

Для тушения пожара необходимо иметь огнетушители, пожарные краны и пожарные щиты, которые необходимо использовать для предотвращения и ликвидации пожара. При использовании огнетушителя необходимо, чтобы его тип соответствовал типу веществу или материалу, которое подверглось возгоранию.

2.6 Расчет количества светильников в помещении

Освещение – неотъемлемая часть при работе с компьютером. Правильно подобранное освещение помогает работать за компьютером с комфортом для глаз. Работа при недостаточном освещении или его отсутствии может негативно сказаться на зрении человека, вызвать головные боли, нарушить кровообращение и вызывать повышенную утомляемость глаз, а также слезоточивость.

Для того, чтобы рассчитать необходимое количество светильников в помещении, воспользуемся формулой:

$$N = E_n * S * K_z * Z / (\eta * \Phi),$$

где η – коэффициент использования светового потока

Z – коэффициент неравномерности освещения, для люминесцентных ламп равен 1.1;

K_z – коэффициент запаса, принимается равным 1.4 для сухих помещений;

E_n – нормируемая минимальная освещенность, примем значение равное 300 лк;

S – площадь помещения;

Φ – световой поток.

Площадь помещения при длине $A = 4$ м, $B = 4$ м, найдем по формуле:

$$S = A * B = 4 * 4 = 16 \text{ м}^2.$$

Коэффициент использования светового потока зависит от индекса помещения, который можно вычислить по следующей формуле:

$$i = A \cdot B / (h_p \cdot (A + B)),$$

где h_p – высота светильников над рабочей поверхностью, в данном случае это значение равно 2 м.

Найдем значение коэффициента использования светового потока:

$$i = 4 \cdot 4 / (2 \cdot (4 + 4)) = 1.$$

Коэффициента отражения потолка $P_{\text{п}}$ и стен $P_{\text{с}}$ выбраны равными 70% и 50%, так как являются белыми и светлыми поверхностями соответственно. Исходя из этих данных находим коэффициент использования светового потока $\eta = 0,34$.

Для освещения выбираем лампы ЛБ40, световой поток которых равен $\Phi = 2800$ лм.

Подсчитаем количество необходимых светильников для ламп ЛПО с двумя лампами:

$$N = 300 \cdot 16 \cdot 1.4 \cdot 1.1 / (0,34 \cdot 2800 \cdot 2) = 4.$$

Таким образом, необходимое количество светильников равно 4.

Заключение

В результате проделанной работы было разработано клиент-серверное приложение, с помощью которого пользователи могут по одной фотографии узнать количество калорий, которое содержится в еде, могут отслеживать и контролировать изменения собственного веса, получать информацию о необходимом ежедневном количестве калорий. Также пользователи могут узнать больше о еде, которую они каждый день употребляют в пищу, выявить все плюсы и минусы, а также узнать, какие витамины содержатся в том или ином продукте. При необходимости пользователь может изучить диеты, которые могут послужить для сброса веса. В приложении также есть статьи, которые помогут пользователю в ведении здорового образа жизни и изучения полезных свойств продуктов. С помощью индекса массы тела, пользователь может оценить свое соотношение веса к росту, выявив при этом у себя возможные отклонения от нормы.

В ходе работы был изучен принцип разработки мобильных приложений, *REST API*. Были закреплены навыки работы с языками программирования *Kotlin* и *Python*, а также навыки работы со средой *Android Studio* и *PyCharm*. Были изучены принципы создания нейронных сетей для сегментации и поиска объектов на изображении.

Разработанное приложение можно в дальнейшем улучшить, добавив большее количество статей, рецептов и материала для изучения и улучшения питания. Можно улучшить нейронную сеть, обучив ее определять большее количество различных продуктов. Также можно улучшить точность системы по определению количества калорий. Можно добавить интеграцию с различными приложениями, синхронизацию данных с сервером, а также сканирование штрих кодов и *QR*-кодов.

Список использованных источников

1. Ян Пойнтер. Программируем с *PyTorch*: Создание приложений глубокого обучения. – СПб.: Питер, 2020 г. – 256 с.
2. Майкл Авенди. *PyTorch Computer Vision Cookbook*. – Packt Publishing, 2020 г. – 364 с.
3. Лука Антига, Томас Виманн, и Эли Стивенс. *Deep Learning with PyTorch*. – Manning Publications, 2020 г. – 520 с.
4. Мигель Гринберг. Разработка веб-приложений с использованием *Flask* на языке *Python*. – М.: ДМК Пресс, 2016 г. – 272 с.
5. Дейтел П., Дейтел Х., Уолд Э. – *Android* для разработчиков. – СПб.: Питер, 2016 г. – 512 с.
6. Колисниченко Д.Н. - Программирование для *Android* 5 – СПб.: БХВ-Петербург, 2020 г. – 288 с.
7. Харди Б., Филлипс Б., Стюарт К., Марсикано К. *Android*. Программирование для профессионалов. – СПб.: Питер, 2016 г. – 640 с.

Приложение А

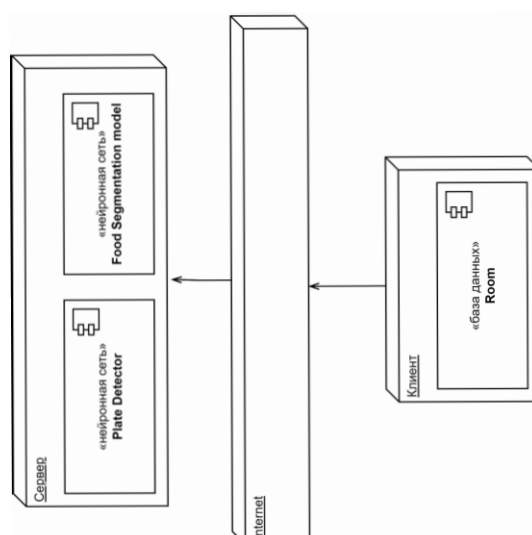
(Обязательное)

Мобильное приложение для анализа и корректировки питания

UML-диаграммы

					<i>ПГУ 1.090301.20.001 ПЗ</i>	Лист
						60
Изм.	Лист	№ докум	Подпись	Дата		

Приложение А.1 - Диаграмма развертывания



ПГУ 1.090301.20.002 Д19									
Изм.	Лист	№ докум	Подпись	Дата	Модель, разработанная для системы информационного дистрибуции				
1	1	1	1	1	ПГУ, апр. 97, р. 18881				

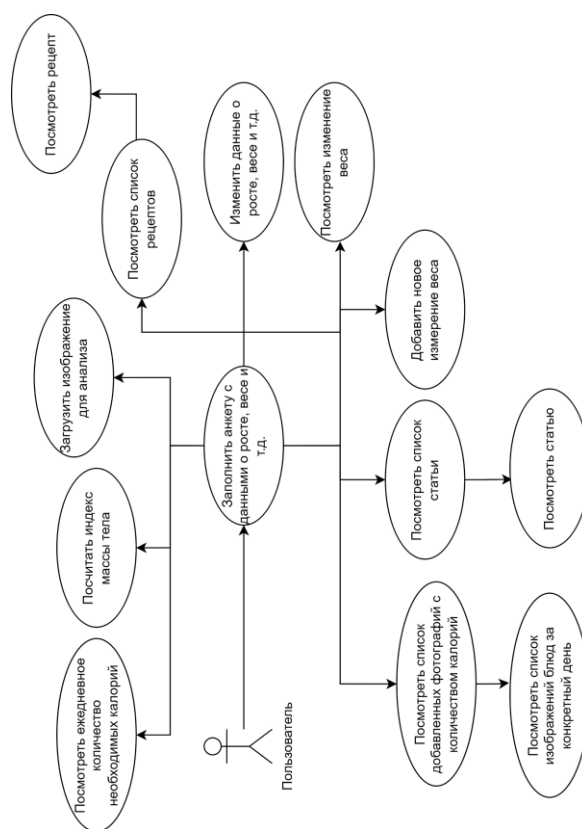
Изм.	Лист	№ докум	Подпись	Дата

ПГУ 1.090301.20.001 ПЗ

Лист

61

Приложение А.2 - Диаграмма прецедентов



ПГУ 1.090301.20.003 Д13									
Модель, разработанная для анализа и документирования системы									
Диаграмма прецедентов									
ПГУ код БТ:									
г.р. 18881									

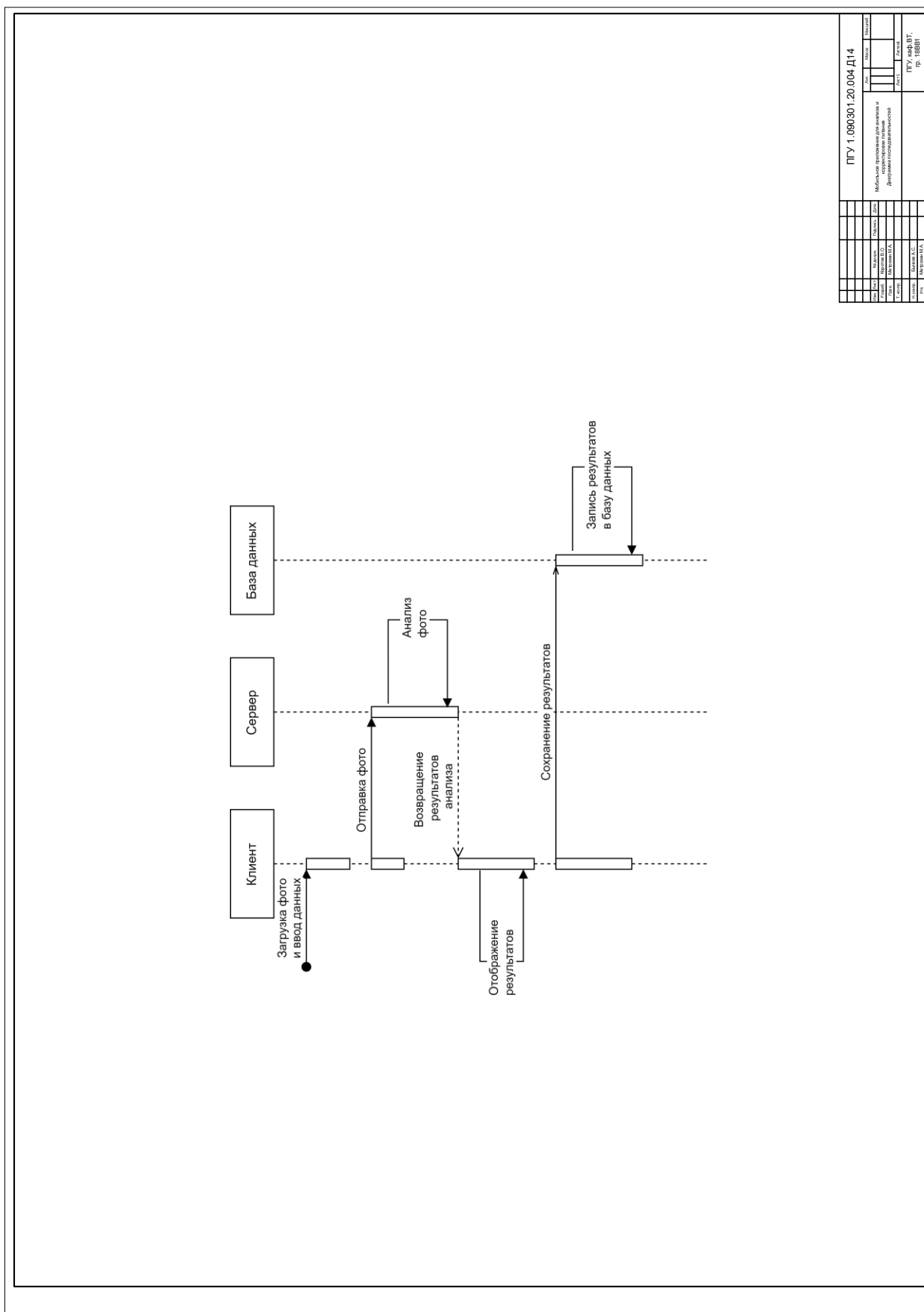
Изм.	Лист	№ докум	Подпись	Дата

ПГУ 1.090301.20.001 ПЗ

Лист

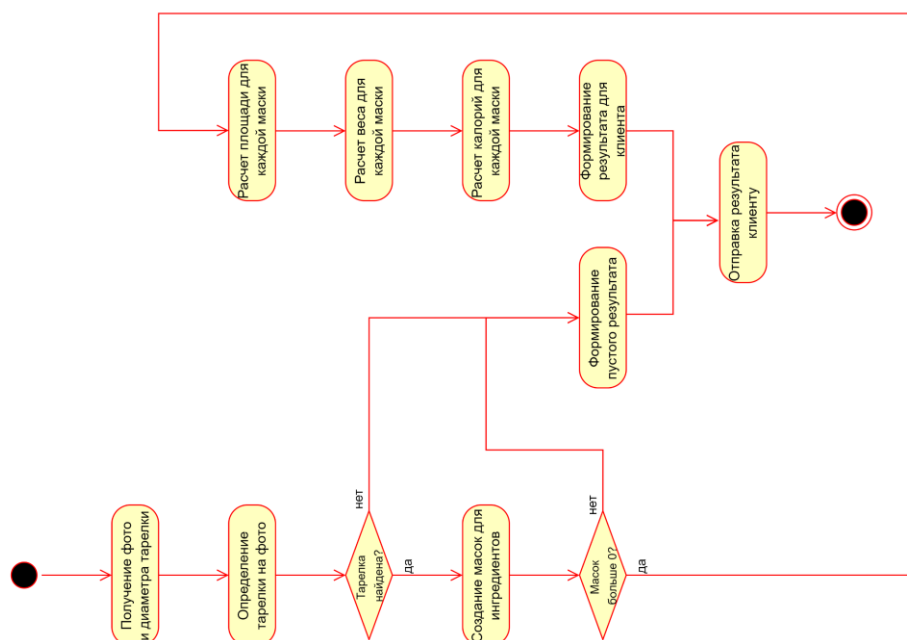
62

Приложение А.3 - Диаграмма последовательностей

[illegible]

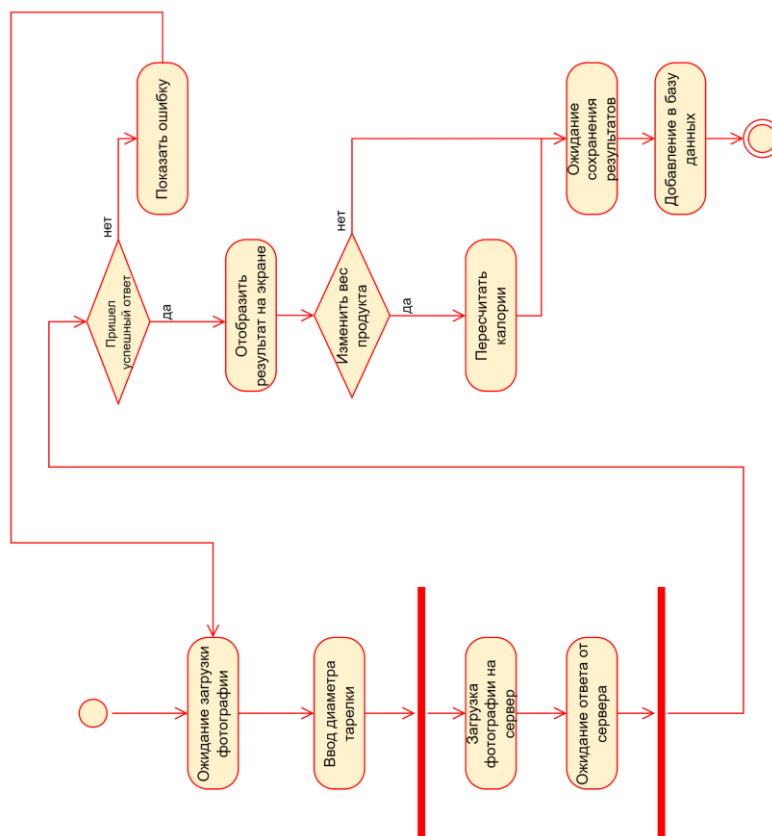
Изм.	Лист	№ докум	Подпись	Дата

Приложение А.4 - Диаграмма деятельности сервера

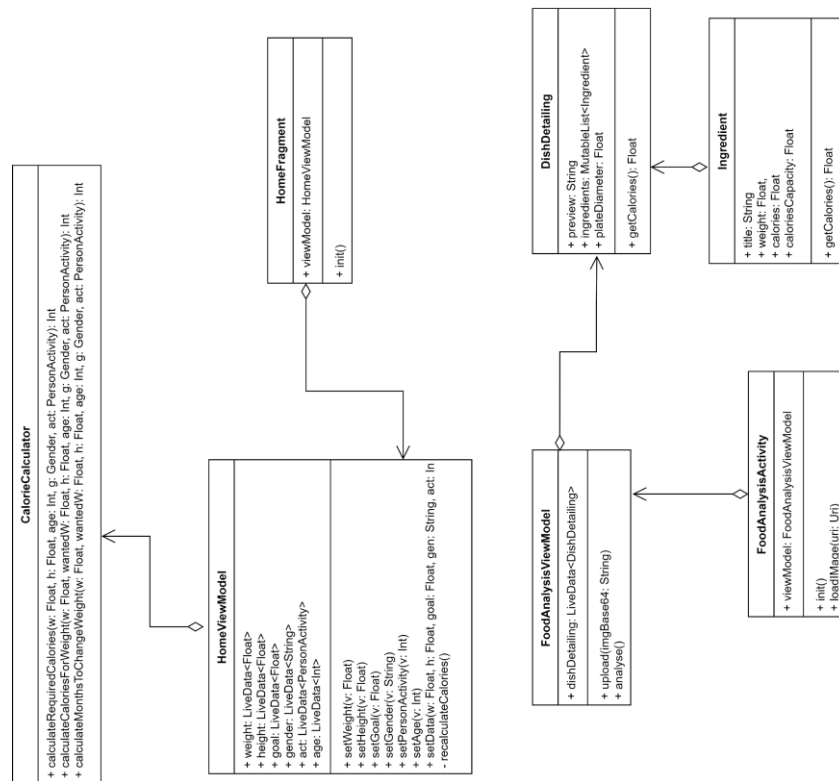
[illegible]

Изм.	Лист	№ докум	Подпись	Дата

Приложение А.5 - Диаграмма деятельности клиента

[illegible]

Приложение А.6 - Диаграмма классов



ПГУ 1.090301.20.007 Д11									
Изм.	Лист	№ докум	Подпись	Дата	Исполн	Провер	Утверд	Подпись	Дата
Модель, разработанная и утвержденная в соответствии с требованиями									
ПГУ, апр. 87, г. 1988									

Приложение Б

(Обязательное)

Мобильное приложение для анализа и корректировки питания

Листинг программы

					ПГУ 1.090301.20.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		67

Файл MainActivity.kt

<...>

@AndroidEntryPoint

```
class MainActivity : AppCompatActivity(R.layout.activity_main) {
```

```
    override fun onSaveInstanceState(outState: Bundle) {
```

```
        super.onSaveInstanceState(outState)
```

```
    }
```

```
    private val binding by
```

```
viewBinding(ActivityMainBinding::bind)
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
```

```
        super.onCreate(savedInstanceState)
```

```
        val navController =
```

```
(supportFragmentManager.findFragmentById(R.id.fragmentContainerV  
iew) as
```

```
NavHostFragment).navController
```

```
binding.bottomNavigationView.setupWithNavController(navControlle  
r)
```

```
binding.bottomNavigationView.setOnItemSelectedListener {  
item ->
```

```
    if (item.itemId !=
```

```
binding.bottomNavigationView.selectedItemId)
```

Изм.	Лист	№ докум	Подпись	Дата

ПГУ 1.090301.20.001 ПЗ

Лист

68

```

        NavigationUI.onNavDestinationSelected(item,
navController)

        true

    }

}

}

```

Файл FoodAPI.kt

<...>

```

interface FoodAPI {

    companion object {

        const val BASE_URL = "http://10.0.2.2:15666/"

    }

    @GET("search")

    suspend fun search(

        @Query("title") title: String = "",
@Query("ingredients") ingredients: List<String> = listOf()

    ): ReceiptsResponse

    @GET("receipt")

    suspend fun getReceipt(

        @Query("receipt_id") receiptId: Int

    ): ReceiptResponse

```

Изм.	Лист	№ докум	Подпись	Дата

ПГУ 1.090301.20.001 ПЗ

Лист

69

```
@GET("receipts")
```

```
suspend fun getAllReceipts(): ReceiptsResponse
```

```
@GET("diets")
```

```
suspend fun getAllDiets(): DietsResponse
```

```
@GET("advices")
```

```
suspend fun getAllAdvices(): AdvicesResponse
```

```
@GET("ingredients")
```

```
suspend fun getIngredients(
```

```
    @Query("title") title: String = ""
```

```
): IngredientsResponse
```

```
@Headers("Content-Type: application/json")
```

```
@POST("upload")
```

```
fun uploadPhoto(
```

```
    @Body imgBody: UploadImageBody,
```

```
): Call<FoodAnalysisResponse>
```

```
data class UploadImageBody(val img: String, val diameter:
Float)
```

```

data class ReceiptsResponse(

    val receipts: List<ReceiptResponse>

)

@Parcelize

data class ReceiptResponse(

    val id: Int,

    val title: String,

    val previewUrl: String,

    val ingredients: List<String>,

    val description: String,

    val timeMin: Int,

    val calories: Int

) : Parcelable

@Parcelize

data class DietResponse(

    val id: Int, val title: String, val previewUrl: String,
val description: String

) : Parcelable

data class DietsResponse(

    val diets: List<DietResponse>

)

```

```

@Parcelize

data class AdviceResponse(

    val id: Int, val title: String, val previewUrl: String,
val description: String

) : Parcelable

```

```

data class AdvicesResponse(

    val advices: List<AdviceResponse>

)

```

```

@Parcelize

data class IngredientResponse(

    val id: Int,

    val title: String,

    val previewUrl: String,

    val description: String,

    val calories: Float,

    val proteins: Float,

    val fats: Float,

    val carbohydrates: Float

) : Parcelable

```

```

data class IngredientsResponse(

```

```

        val ingredients: List<IngredientResponse>

    )

    @Parcelize

    data class FoodAnalysisResponse(

        val result: List<WeightResponse>

    ): Parcelable

    @Parcelize

    data class WeightResponse(

        val title: String,

        val weight: Float,

        val calories: Float,

        val box: List<Float>,

        val ingredient: IngredientResponse

    ): Parcelable

}

```

Файл CalorieCalculator.kt

<...>

```

object CalorieCalculator {

    enum class Gender {

        Male, Female

    }

}

```

Изм.	Лист	№ докум	Подпись	Дата

ПГУ 1.090301.20.001 ПЗ

Лист

73

```

enum class PersonActivity {

    MINIMAL_ACTIVITY, WEAK_ACTIVITY, AVERAGE_ACTIVITY,
    HIGH_ACTIVITY, EXTRA_ACTIVITY;

fun getRatio(): Float {

    return when (this) {

        MINIMAL_ACTIVITY -> 1.2f

        WEAK_ACTIVITY -> 1.375f

        AVERAGE_ACTIVITY -> 1.55f

        HIGH_ACTIVITY -> 1.725f

        else -> 1.9f

    }

}

companion object {

    fun getActivity(index: Int) = values()[index]

    fun toInfo(index: Int): String {

        return when (getActivity(index)) {

            MINIMAL_ACTIVITY -> "Минимальная активность"

            WEAK_ACTIVITY -> "Низкая активность"

            AVERAGE_ACTIVITY -> "Средняя активность"

            HIGH_ACTIVITY -> "Высокая активность"

            else -> "Экстра активность"

        }

    }

}

```

```

        }

    }

}

const val MAX_WEIGHT_CHANGE = 4

const val MIN_MALE_CALORIES = 1800

const val MIN_FEMALE_CALORIES = 1200

const val OPTIMAL_CALORIE_CHANGE_PER_KG = 257


fun getMinCalories(gender: Gender): Int = when (gender) {

    Gender.Male -> MIN_MALE_CALORIES

    else -> MIN_FEMALE_CALORIES

}


private fun calculateRequiredCalories(weight: Float, height:
Float, age: Int, gender: Gender, activity: PersonActivity): Int
=

    if (gender == Gender.Male) {

        ((10 * weight + 6.25f * height - 5 * age + 5) *
activity.getRatio()).toInt()

    } else {

        ((10 * weight + 6.25f * height - 5 * age - 161) *
activity.getRatio()).toInt()

    }


fun calculateCaloriesForWeight(

```

```

        weight: Float, wantedWeight: Float, height: Float, age:
Int, gender: Gender, activity: PersonActivity

    ): Int {

        val requireCalories = calculateRequiredCalories(weight,
height, age, gender, activity)

        return (requireCalories + (wantedWeight -
weight).coerceIn(

            -MAX_WEIGHT_CHANGE.toFloat(),

            MAX_WEIGHT_CHANGE.toFloat()

        ) *
OPTIMAL_CALORIE_CHANGE_PER_KG).coerceAtLeast(getMinCalories(gend
er).toFloat()).toInt()

    }

fun calculateMonthsToChangeWeight(

    weight: Float, wantedWeight: Float, height: Float, age:
Int, gender: Gender, activity: PersonActivity

    ): Int {

        if (wantedWeight < weight) {

            val requireCalories =
calculateRequiredCalories(weight, height, age, gender, activity)

            val maxCalorieLoss = requireCalories -
getMinCalories(gender)

            val maxKgLoss = maxCalorieLoss /
OPTIMAL_CALORIE_CHANGE_PER_KG.toFloat()

            val kgToLoss = min((weight - wantedWeight).coerceIn(

```

```

        0f,

        MAX_WEIGHT_CHANGE.toFloat()

    ), maxKgLoss)

    return ceil(abs(wantedWeight - weight) /
kgToLoss).toInt()

}

    return (abs(wantedWeight - weight) / MAX_WEIGHT_CHANGE +
1).toInt()

}

}

```

Файл AppModule.kt

<...>

```
@Module
```

```
@InstallIn(SingletonComponent::class)
```

```
object AppModule {
```

```
    @Singleton
```

```
    @Provides
```

```
    fun provideWeightDao(@ApplicationContext context: Context):
WeightDao =

        (context as MyApplication).database.weightDao

```

```

@Singleton

@Provides

fun provideReceiptDao(@ApplicationContext context: Context):
ReceiptDao =

    (context as MyApplication).database.receiptDao


@Singleton

@Provides

fun provideCalorieDao(@ApplicationContext context: Context):
FoodByDayDao =

    (context as MyApplication).database.foodByDayDao


@Singleton

@Provides

fun provideFoodApi(@ApplicationContext context: Context):
FoodAPI = (context as MyApplication).foodAPI(

    context.provideUnsplashRetrofit()

)

}

```

Файл HomeFragment.kt

<...>

```
@AndroidEntryPoint
```

```
class HomeFragment : Fragment(R.layout.fragment_home) {
```

Изм.	Лист	№ докум	Подпись	Дата

ПГУ 1.090301.20.001 ПЗ

Лист

78

```

        private val binding by
viewBinding(FragmentHomeBinding::bind)

        private val viewModel by viewModels<HomeViewModel>()

        private val adapter = OptionAdapter().apply {

            onItemClick = { item, position ->

                val intent = Intent(requireContext(),
item.intentCls)

                startActivity(intent)

            }

        }

        private val sPrefs: SharedPreferences by lazy {

            requireContext().getSharedPreferences(

                MyApplication.PREFERENCE_FILE_KEY,
Context.MODE_PRIVATE

            )

        }

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

            super.onViewCreated(view, savedInstanceState)

            init()

        }

        private fun init() {

```

```

        val height = sPrefs.getFloat(MyApplication.PREFS_HEIGHT,
-1f)

        val weight = sPrefs.getFloat(MyApplication.PREFS_WEIGHT,
-1f)

        val age = sPrefs.getInt(MyApplication.PREFS_AGE, 25)

        val gender =
sPrefs.getString(MyApplication.PREFS_GENDER, "") ?: ""

        val goal = sPrefs.getFloat(MyApplication.PREFS_GOAL, -
1f)

        val personActivity =
sPrefs.getInt(MyApplication.PREFS_PERSON_ACTIVITY, 0)

        viewModel.setData(weight, goal, height, age, gender,
personActivity)


        binding.textMyHeight.text = "Рост: $height см"

        binding.textMyWeight.text = "Вес: $weight кг"

        binding.textGoal.text = "Цель: ${goal}
кг\n${CalorieCalculator.PersonActivity.toInfo(personActivity)}"

        binding.textGenderAge.text = "Пол: $gender\nВозраст:
$age"


binding.recyclerViewReportsAndAdvices.addItemDecoration(ItemOffs
etDecorator(8f))

        binding.recyclerViewReportsAndAdvices.layoutManager =

                LinearLayoutManager(requireContext(),
LinearLayoutManager.HORIZONTAL, false)

```

```

binding.recyclerViewReportsAndAdvices.adapter = adapter

viewModel.currentCalories.observe(viewLifecycleOwner) {

    it?.let {

        binding.calorieViewer.calories = it

    }

}

viewModel.requiredCalories.observe(viewLifecycleOwner) {

    it?.let {

        binding.calorieViewer.requiredCalories = it

    }

}

viewModel.options.observe(viewLifecycleOwner) { newList
->

    adapter.setData(newList)

}

viewModel.calorieInfo.observe(viewLifecycleOwner) { info
->

    info?.let {

        binding.textCalorieInfo.text = info

    }
}

```

```

    }

    binding.rowWeight.setOnClickListener {

        showInputDialog("Bec", MyApplication.PREFS_WEIGHT) {

            binding.textMyWeight.text = "Bec: $it кг"

            viewModel.setWeight(it)

        }

    }

    binding.rowHeight.setOnClickListener {

        showInputDialog("Рост", MyApplication.PREFS_HEIGHT)

    {

        binding.textMyHeight.text = "Рост: $it см"

        viewModel.setHeight(it)

    }

    }

    binding.rowGoal.setOnClickListener {

        val initialGoal = viewModel.goal.value!!

        val initialActivityIndex =
viewModel.personActivity.value!!

        val dialog = DialogGoalPersonActivity(initialGoal,
initialActivityIndex) { newGoal, newPersonActivityIndex ->

```

```

        if (null == newGoal || newGoal.toFloat() <= 0f)
        {

            Toast.makeText(requireContext(), "Ошибка
ввода!", Toast.LENGTH_SHORT).show()

            return@DialogGoalPersonActivity

        }

        sPrefs.edit().putFloat(MyApplication.PREFS_GOAL,
newGoal)

        .putInt(MyApplication.PREFS_PERSON_ACTIVITY,
newPersonActivityIndex).apply()

        viewModel.setGoal(newGoal)

viewModel.setPersonActivity(newPersonActivityIndex)

        binding.textGoal.text = "Цель: ${newGoal}
кк\r\n${CalorieCalculator.PersonActivity.toInfo(newPersonActivityI
ndex)}"

    }

    dialog.show(childFragmentManager, null)

}

binding.rowGenderAge.setOnClickListener {

    val initialAge = viewModel.age.value!!

    val initialGender = viewModel.gender.value!!

    val dialog = GenderAgeDialog(initialGender,
initialAge) { gender, age ->

```

```

        if (gender.isEmpty() || age == null ||
age.toInt() < 0) {

            Toast.makeText(requireContext(), "Ошибка
ввода!", Toast.LENGTH_SHORT).show()

            return@GenderAgeDialog

        }

sPrefs.edit().putString(MyApplication.PREFS_GENDER,
gender).putInt(MyApplication.PREFS_AGE, age).apply()

viewModel.setGender(gender)

viewModel.setAge(age)

binding.textGenderAge.text = if
(gender.isNotEmpty()) "Пол: $gender\nВозраст: $age" else "Данные
не указаны"

    }

    dialog.show(childFragmentManager, null)

}

}

private fun showInputDialog(title: String, key: String,
event: ((value: Float) -> Unit)) {

    val editText = EditText(requireContext())

    val container = FrameLayout(requireContext())

```

```

        val params =
            FrameLayout.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT,
            ViewGroup.LayoutParams.WRAP_CONTENT)

        val dp = TypedValue.applyDimension(
            TypedValue.COMPLEX_UNIT_DIP, 16f,
            requireContext().resources.displayMetrics

        ).toInt()

        params.marginEnd = dp

        params.marginStart = dp

        params.topMargin = dp

        params.bottomMargin = dp

        editText.layoutParams = params

        container.addView(editText)

        AlertDialog.Builder(requireContext()).apply {

            setView(container)

            setTitle(title)

            setPositiveButton("OK") { dialog, _ ->

                val inputValue = editText.text.toString()

                if (null == inputValue.toFloatOrNull() ||
                inputValue.toFloat() <= 0f) {

                    Toast.makeText(requireContext(), "Ошибка
                    ввода!", Toast.LENGTH_SHORT).show()

                    return@setPositiveButton

                }
            }
        }

```

```

        sPrefs.edit().putFloat(key,
inputValue.toFloat()).apply()

        event.invoke(inputValue.toFloat())

    }

    setNeutralButton("Отмена", null)

    show()

}

}

}

```

Приложение В

(Обязательное)

Мобильное приложение для анализа и корректировки питания

Презентация

					ПГУ 1.090301.20.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		87

Мобильное приложение для анализа и корректировки питания

Выполнил: студент группы 18ВВ1

Коротов В.О.

Руководитель: Митрохин М.А.

Задачи проекта

- Расчет калорий блюда по фотографии
- Подбор рецептов по ингредиентам
- Рекомендации для улучшения питания
- Удобное отслеживание веса



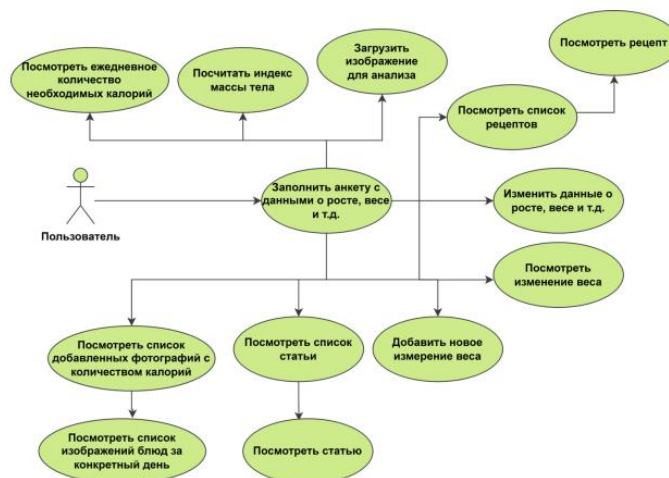
Изм.	Лист	№ докум	Подпись	Дата

ПГУ 1.090301.20.001 ПЗ

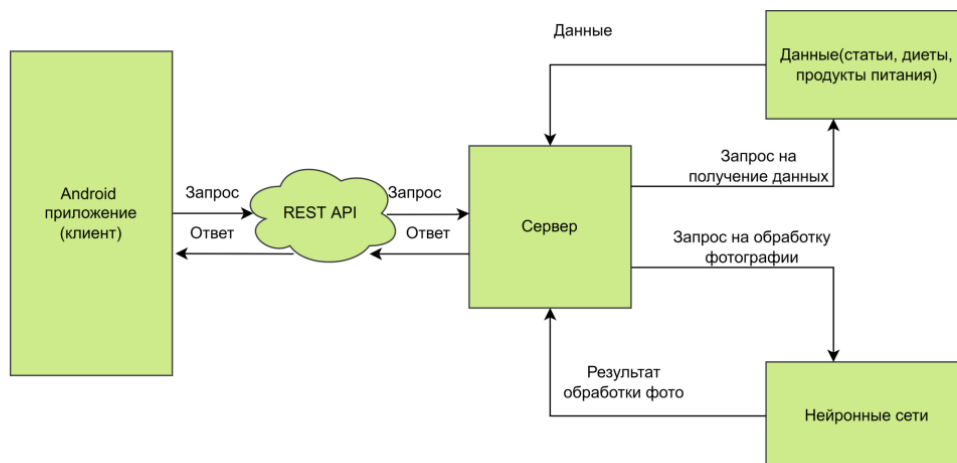
Лист

88

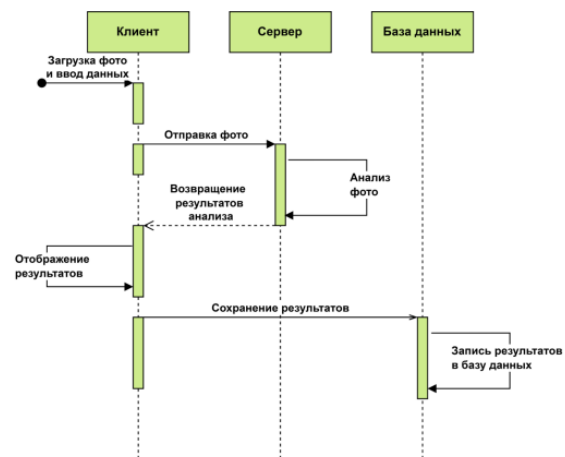
UML-диаграмма вариантов использования



Структура проекта



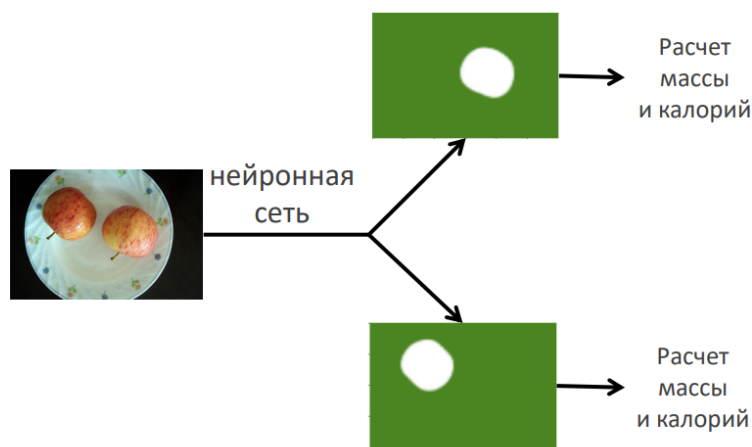
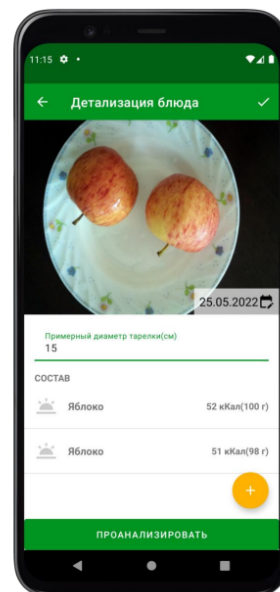
UML-диаграмма последовательностей



Определение калорий



Определение калорий



Формула для расчета калорий

$$Calories = \frac{w * h * d^2 * ingredientPixels * avg * capacity}{plateWidth^2 * imagePixels * 100}$$

w – ширина изображения в пикселях;

h – высота изображения в пикселях;

plateWidth – ширина тарелки в пикселях;

d – диаметр тарелки в см;

ingredientPixels – количество пикселей в маске ингредиента;

imagePixels – общее количество пикселей в изображении;

capacity – калорийность продукта (на 100 г).

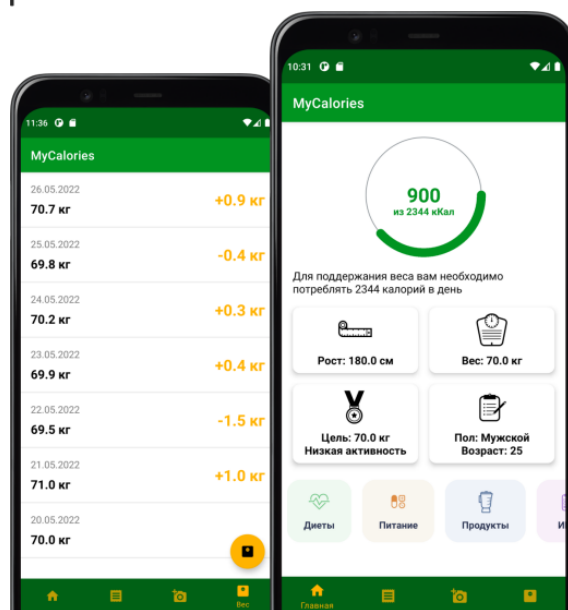
avg – среднее отношения массы продукта к занимаемой площади (г/см²)

Отслеживание калорий и веса

По формуле Миффлина Сан-Жеора вычисляется ежедневная норма калорий

Приложение позволяет человеку вводить данные и следить за изменением своего веса

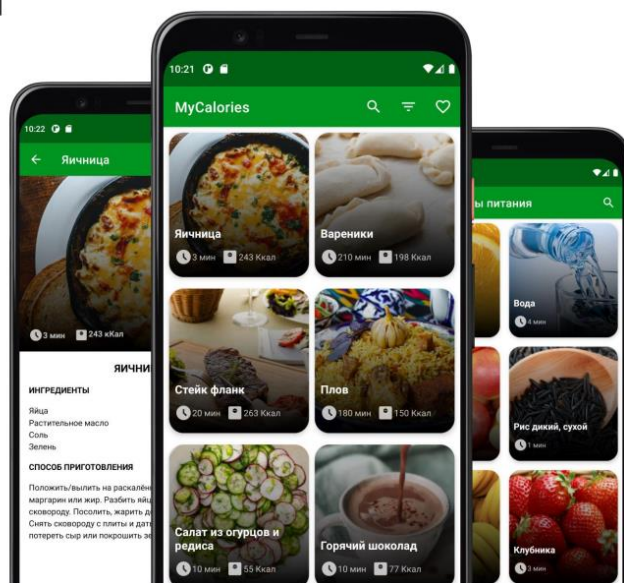
Система адаптируется под потребности пользователя и наглядно демонстрирует результат на экране



Улучшение питания

Доступны статьи, которые описывают принципы здорового питания

Есть возможность подобрать рецепт по ингредиентам



Индекс массы тела

ИМТ позволяет определить соотношение массы человека к его росту. Это значение позволит узнать, является ли масса недостаточной, избыточной или нормальной

Отклонение от нормы предупредит пользователя о возможных проблемах со здоровьем



Заключение

Разработанное мобильное приложение позволяет людям отслеживать потребление калорий, подбирать рецепты, следить за весом и корректировать питание

В дальнейшем можно добавить большее количество классов для нейронной сети, улучшить метод определения калорий

