

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

**Мобильное приложение для планирования
и контроля режима питания**

Студент

_____ Дибин Андрей Юрьевич
(подпись, дата) (ФИО полностью)

Руководитель

_____ Князев В.Н.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

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

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

Заведующий кафедрой

_____ Макарычев П.П.
(подпись) (фамилия, инициалы)

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

Секретарь ГЭК

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

Пенза, 2022

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

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
П.П. Макарычев
«__» _____ 202 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Дибин Андрей Юрьевич гр. 18ВП1 факультета ВТ

<Фамилия, имя, отчество полностью>

2. Тема работы Мобильное приложение для планирования и контроля режима питания

Тема утверждена приказом ПГУ № 370/О от “12” апреля 2022

3. Руководитель работы к.т.н., доцент кафедры «МОиПЭВМ» Князев В.Н.

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

1) Разработать приложение для мобильных устройств, обеспечивающее поддержку планирования и контроля режима питания.

2) Среда выполнения приложения: операционная система Android 7.1 и выше

Среда выполнения сервера: операционная система Windows 10

Языковые средства разработки: C#, UML, XAML, SQL

Инструментальные средства разработки: Microsoft SQL Server Management Studio 2018, Microsoft Visual Studio 2022 preview, Microsoft Project, ArgoUML

Технологии разработки: RUP, ООП, Entity Framework

3) Основные функции мобильного приложения:

а) регистрация и авторизация пользователей;

б) просмотр списка имеющихся блюд;

в) добавление и удаление блюд в приемы пищи пользователя;

г) подбор персонального меню;

д) расчет потребности в калориях и калорийности рациона

5. Перечень подлежащих разработке вопросов

1) Анализ предметной области и постановка задачи

2) Проектирование

3) Реализация

4) Контроль качества программного обеспечения

5) Оформление пояснительной записки

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области постановка задачи	15%	13.04.2022– 24.04.2022	
2	Проектирование	25%	25.04.2022– 08.05.2022	
3	Реализация	40%	09.05.2022– 2.06.2022	
4	Контроль качества программного обеспечения	5%	3.06.2022– 05.06.2022	
5	Оформление пояснительной записки	15%	06.06.2022– 17.06.2022	

Дата выдачи задания "22" 12 2021

Руководитель бакалаврской работы _____

(подпись, дата)

Задание к исполнению принял студент _____

(подпись, дата)

Работу к защите допустить

Декан факультета _____

(подпись, дата)

Реферат

Пояснительная записка содержит 113 листов, 52 рисунка, 13 таблиц, 25 использованных источников, 7 приложений.

ПЛАНИРОВАНИЕ РЕЖИМА ПИТАНИЯ, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА, MS SQL SERVER, ANDROID, C#, XAML, UML, RUP

Объектом разработки является мобильное приложение для планирования и контроля режима питания.

Цель работы – разработка мобильного приложения для планирования и контроля режима питания, серверной части приложения и базы данных.

Разработка базы данных проводилась с помощью СУБД MS SQL Server. Разработка клиентского мобильного приложения, а также серверного приложения проводилась на языке C# в среде программирования Microsoft Visual Studio 2022 preview. Разработка проведена с использованием операционной системы Windows 10.

Клиентское приложение предназначено для использования под управлением ОС Android версии 7.1 и выше, а серверная часть работает под управлением операционной системы Windows 10 и выше.

					ПГУ 09.03.04 – 08ВР181.03 ПЗ			
Изм.	Лист	№ докум.	Подп.	Дата	Мобильное приложение для планирования и контроля режима питания. Пояснительная записка	Лит.	Лист	Листов
Разраб.		Дибин А.Ю.					4	113
Пров.		Князев В.Н				Группа 18ВП1		
Реценз.								
Н. контр.		Такташкин Д.В						
Утв.		Макарычев П.П						

Содержание

Введение.....	7
1 Анализ требований и предметной области.....	9
1.1 Анализ предметной области и постановка задачи.....	9
1.2 Анализ аналогичных программных продуктов.....	9
1.2.1 Мобильное приложение «YAZIO Трекер питания»	10
1.2.2 Мобильное приложение «Хики – калькулятор калорий»	11
1.2.3 Мобильное приложение «FatSecret»	12
1.2.4 Сравнение аналогов	14
1.3 Анализ целевой аудитории.....	14
1.4 Анализ функциональных требований	17
1.5 Анализ нефункциональных требований	20
1.6 Разработка концептуальной модели базы данных.....	20
1.7 Планирование разработки и оценка бюджета	21
2 Проектирование мобильного приложения	29
2.1 Выбор технологий и инструментальных средств разработки	29
2.2 Проектирование архитектуры программного обеспечения.....	30
2.3 Проектирование логической модели базы данных.....	31
2.4 Проектирование интерфейса пользователя	33
3 Реализация мобильного приложения	39
3.1 Реализация базы данных.....	39
3.2 Реализация серверной части	40
3.3 Реализация клиентской части	43
4 Тестирование и оценка качества приложения	46
4.1 Тестирование серверной части	46

4.1.1 Функциональное тестирование.....	46
4.1.2 Модульное тестирование.....	47
4.2 Тестирование клиентской части	48
4.3 Метрики кода.....	50
4.3.1 Метрика Мак-Кейба.....	51
4.3.2 Метрика Холстеда.....	52
Заключение	54
Список использованных источников	55
Приложение А. Глоссарий	58
Приложение Б. Листинг кода создания базы данных	60
Приложение В. Исходный код серверной части.....	64
Приложение Г. Исходный код клиентской части	79
Приложение Д. Результаты тестирования серверной части приложения.....	99
Приложение Е. Исходный код модульного тестирования серверной части приложения	104
Приложение Ж. Результаты тестирования клиентской части приложения..	108

Введение

В сегодняшние дни информационные технологии все шире распространяются на разнообразные сферы человеческой деятельности.

9 мая 2017 года была утверждена «Стратегия развития информационного общества в Российской Федерации на 2017 – 2030 годы». В ней были поставлены основные цели и задачи политики Российской Федерации в сфере информационных технологий, а также объявлен курс на формирование национальной цифровой экономики [1]. С этой же целью в 2017 году была принята государственная программа «Цифровая экономика». Она призвана обеспечить «развитие цифровой экономики Российской Федерации, в которой данные в цифровой форме являются ключевым фактором производства во всех сферах экономической деятельности, что повышает конкурентоспособность страны, качество жизни граждан, обеспечивает экономический рост и экономический суверенитет» [2].

Практически у каждого современного человека при себе есть смартфон, который давно перестал быть лишь средством связи. Большинство тарифных планов для смартфонов включает в себя некий объем мобильного трафика. Кроме того, мобильный интернет оттесняет на второй план традиционный доступ к сети с использованием стационарных компьютеров по причине большей мобильности и удобства. Это отмечено в государственной программе «Цифровая экономика», в 2016 году в нашей стране «на 100 человек приходилось 159,95 мобильного телефона и из 100 человек 71,29 человека использовали мобильный доступ к сети Интернет» [2]. Со временем эта тенденция только усиливается.

Роль пользователей мобильных устройств усиливается по сравнению с пользователями персонального компьютера. Из-за этого разработчики веб-сервисов стараются адаптироваться под изменяющийся рынок и выпускать версии своих продуктов в формате мобильных приложений. Не обошла эта тенденция и сферу здорового образа жизни и правильного питания.

В 2020 году, по данным международной Global Market Insights, объем мирового рынка мобильных технологий в области здравоохранения – mHealth достиг \$56 млрд и до 2030 года будет расти на 30% ежегодно. К 2027 году он превысит \$300 млрд, прогнозирует американская Reports and Data [3].

Если человеку необходимо следить за своим весом, то ему нужно регулярно записывать свои приемы пищи и отслеживать калорийность потребленных блюд, чтобы понимать превысил ли человек свою энергетическую потребность или же нет. Для этого обычно использовались блокноты и тетради, или в крайнем случае, таблицы в Microsoft Excel. Но у блокнотов есть существенный недостаток, нужно вручную считать сумму килокалорий и знать калорийность каждого блюда. А использование электронных таблиц усложнено их первоначальной настройкой и пониженной мобильностью, ведь нужно часто обращаться к компьютеру. Таким образом, на смену этим инструментам пришли мобильные приложения, которые удобны своей доступностью, их можно быстро открыть, воспользоваться и быстро отключить.

Таким образом, учитывая современные тенденции, можно сделать вывод о том, что разработка мобильного приложения для планирования и контроля режима питания актуальна и перспективна в условиях нашего общества.

1 Анализ требований и предметной области

1.1 Анализ предметной области и постановка задачи

Разрабатываемое мобильное приложение должно обеспечить пользователям простой способ отслеживания индивидуального рациона питания, легко получать информацию о питательной ценности продуктов, а также получать рекомендацию по составу индивидуального дневного рациона питания. Благодаря приложению пользователю не потребуются дорогостоящие советы диетологов, ведь функционал подбора персонального меню может в большинстве случаев заменить прием такого специалиста.

При регистрации пользователь вводит данные о себе (пол, возраст, вес), на основе которых формируется его суточная потребность энергетической ценности еды, выраженная в калорийности рациона. Приложение должно обеспечивать всем пользователям доступ к информации о блюдах, информация о которых загружена в систему. У каждого пользователя должна быть возможность ввода информации о уже совершенных приемах пищи.

Таким образом, приложение имеет следующие основные функции:

- регистрация и авторизация пользователей;
- подбор персонального меню;
- просмотр списка блюд;
- добавление и удаление блюд в приемы пищи;
- расчет потребности в калориях и калорийности рациона.

Таким образом, задача работы – это разработка мобильного приложения на основе архитектуры, состоящей из серверного приложения, базы данных и клиентского мобильного приложения.

1.2 Анализ аналогичных программных продуктов

Сейчас существует ряд мобильных приложений для контроля режима питания. Рассмотрим наиболее известные приложения такого рода и произведем для них сравнительный анализ.

1.2.1 Мобильное приложение «YAZIO трекер питания»

Мобильное приложение YAZIO – бесплатное приложение «счетчик калорий», в функционале приложения есть возможность вести дневник питания, отслеживать физическую активность.

Внешний вид (интерфейс) приложения представлен на рисунке 1.

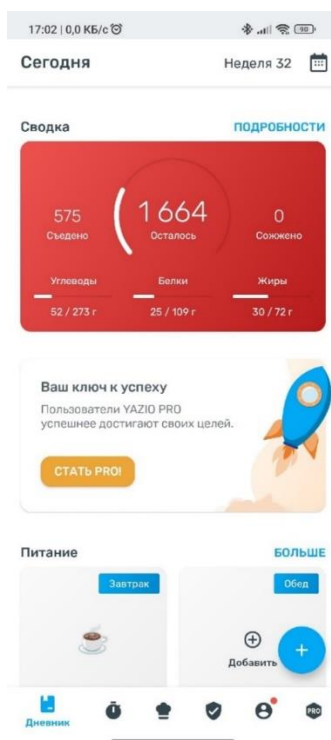


Рисунок 1 – Интерфейс приложения YAZIO

В верхней части на главной странице приложения располагается виджет, который показывает 3 количества калорий: «съедено» – это сумма энергетической ценности всех приемов пищи, которые занесены в приложение как съеденные; «осталось» – это сумма калорий, которые необходимо употребить до конца дня, «сожжено» – это количество калорий, которые вы «сожгли» с помощью физических упражнений.

После виджета, чуть ниже располагается рекламный баннер, рекламирующий платную подписку на это приложение. Далее идет блок с приемами пищи, в которые можно добавлять продукты, которые вы употребили в пищу. Далее, в приложении есть баннер «активность», с помощью которого можно подключить отслеживание шагов. Следующий

баннер – «измерения тела», там можно подредактировать значение своего веса. Далее идет «отслеживание выпитой воды», «заметки», раздел «узнать подробнее» со статьями на тему здорового образа жизни (ЗОЖ) [4].

К недостаткам приложения можно отнести: навязчивую рекламу дорогостоящей платной подписки (многочисленные баннеры почти на каждой странице приложения), недоступность многих функций без платной подписки. Например, без платной подписки не работают такие функции приложения как: отображение графика белков, жиров и углеводов (БЖУ); отображение минерального состава блюд; рецепты блюд; планы питания; запись информации о теле, кроме веса и другие, перегруженный интерфейс, излишне много функций (заметки, рецепты, трекер голодания).

1.2.2 Мобильное приложение «Хики – калькулятор калорий»

Программа рассчитает нужную норму калорий и БЖУ в зависимости от ваших параметров и активности. Интерфейс программы представлена на рисунке 2.

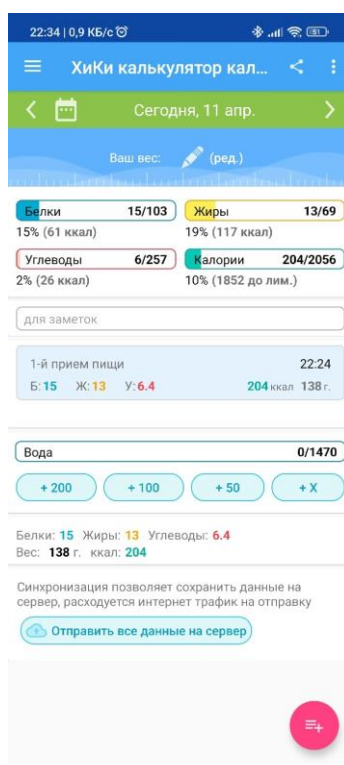


Рисунок 2 – Интерфейс ХиКи Калькулятор калорий

Навигация в приложении устроена с помощью боковой панели, в которой размещены вкладки приложения.

Первая вкладка – это профиль, в нем можно заполнить информацию о поле, возрасте, росте и весе, а также указать уровень физической активности и цель. При вводе этих данных будет рассчитана норма чистой воды, индекс массы тела и лимит калорий.

На вкладке приемы пищи, можно добавлять приемы пищи, состоящие из продуктов, а также корректировать данные о весе и указывать количество выпитой воды. Во вкладке счетчик воды также можно указывать количество выпитой воды, а во вкладке продукты есть список продуктов, с которым можно ознакомиться.

Во вкладке статистика можно посмотреть некоторые статистические данные и графики изменения веса и потребления калорий. Во вкладке микшер или рецепты можно создать блюдо из доступных продуктов [5].

К недостаткам приложения можно отнести: отсутствие функции подбора персонального меню, дублирование функций в разных разделах (например, в приеме пищи можно указать актуальное потребление воды, но есть еще и отдельная вкладка для контроля за потреблением воды, список продуктов возможно посмотреть отдельно, но добавить продукт в прием пищи можно только со страницы «приемы пищи»).

1.2.3 Мобильное приложение «FatSecret»

При первом запуске приложения необходимо указать свой возраст, пол, рост, образ жизни (например, активный или сидячий), а также фактический и желанный показатели веса. На основе этих данных сервис высчитывает показатель рекомендуемое суточное количество калорий (РСК), которого вы должны придерживаться.

Для постоянной фиксации потреблённых и потраченных калорий предназначена вкладка «Дневник». Открыв её, вы сможете добавить продукты и порции, которые вы употребили в пищу в течение суток. Или упражнения и

другие активности, которыми вы занимались. Система сама переведёт все в калории.

База данных приложения содержит сведения о калориях в пищевых товарах известных производителей, а также во многих блюдах. Эта информация используется FatSecret при подсчётах и позволяет не вносить лишние данные вручную. Найти нужный продукт, товар или блюдо можно с помощью встроенного каталога или формы поиска.

Для большего контроля можно периодически вносить результаты взвешиваний на вкладке «Вес», на которой также отображается график изменения веса. Следить за динамикой калорий можно на вкладке «Отчёты» [6].

Интерфейс программы представлен на рисунке 3.

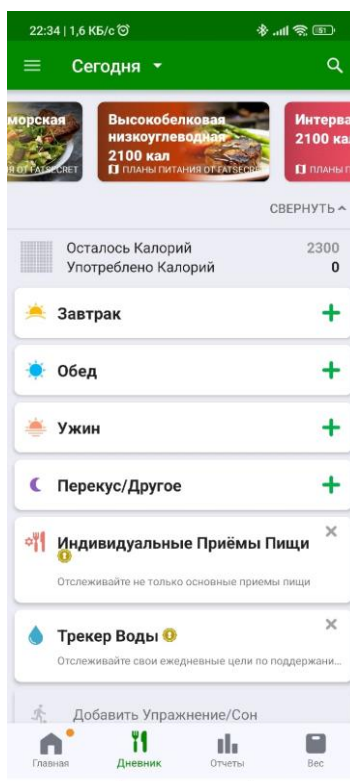


Рисунок 3 – Интерфейс FatSecret

Недостатком мобильного приложения является отсутствие функции подбора персонального меню. У программы перегруженный и запутанный интерфейс, не интуитивно понятное расположение всех функций.

1.2.4 Сравнение аналогов

Был произведен сравнительный анализ рассмотренных выше мобильных приложений. Результат анализа занесен в таблицу 1.

Таблица 1 – Сравнение аналогов

Мобильное приложение	Платная подписка	Функция «подбор персонального меню»	Интерфейс
YAZIO	Есть, дорогостоящая	Нет	Перегруженный
ХиКи	Есть	Нет	Простой
FatSecret	Есть	Нет	Запутанный
Разрабатываемое приложение	Нет, все функции бесплатны	Есть	Простой

Из результатов сравнения можно сделать вывод о том, что большинство приложений используют механизм платных подписок, ограничивающих функциональность этих приложений. Также, можно заметить, что функции подбора меню нет в рассмотренных выше программах. Кроме того, во многих рассмотренных программах перегруженный или запутанный интерфейс.

Таким образом, в результате анализа программ аналогичного назначения было принято решение о разработке собственного приложения, которое будет лишено недостатков, выявленных у рассмотренных выше мобильных приложений.

1.3 Анализ целевой аудитории

Целевой аудиторией проекта являются пользователи мобильных устройств, интересующиеся правильным питанием и здоровым образом жизни. В настоящее время спрос на мобильные приложения в категории здоровья и фитнеса растет, о чем свидетельствует последний отчет аналитической компании Sensor Tower [7]. График, на котором показана динамика роста загрузок приложений этой категории представлен на рисунке 4.

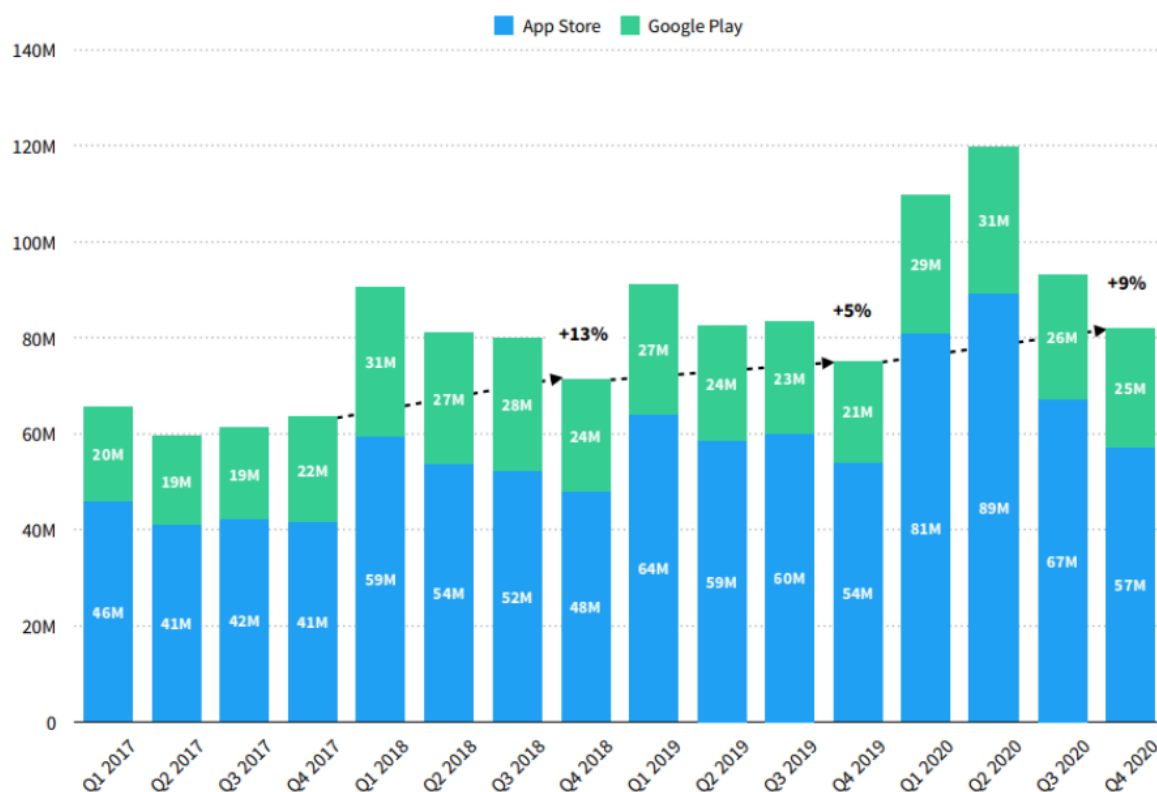


Рисунок 4 – Количество загрузок приложений в категории здоровье и фитнес

Как видно по графику, прослеживается устойчивая тенденция к росту числа загрузок с каждым годом. Среди всех подкатегорий категории здоровье и фитнес по полученному доходу лидирует подкатегория диеты, это видно на графике, представленном на рисунке 5.

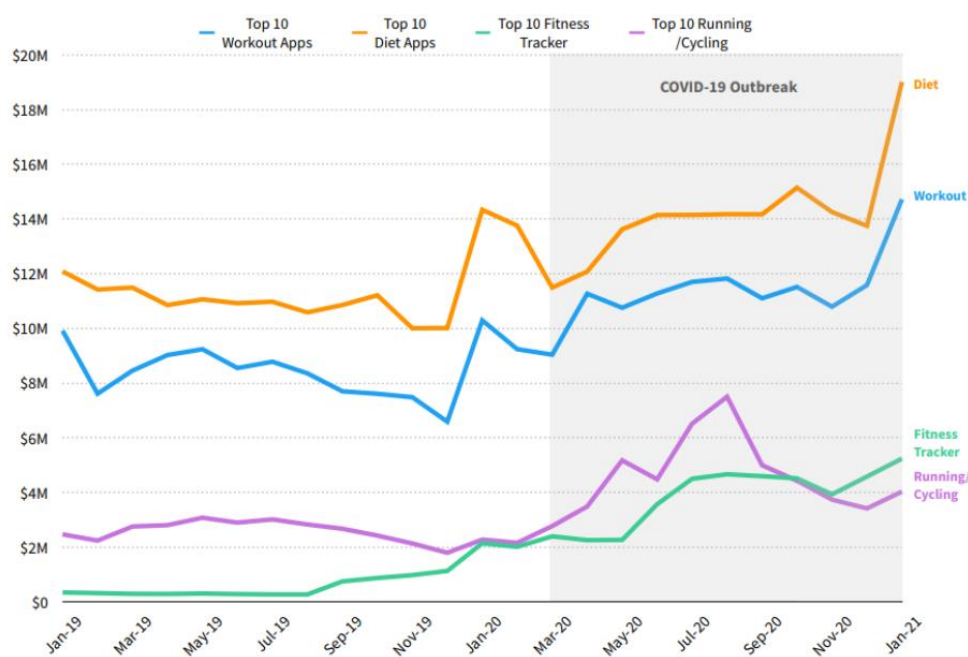


Рисунок 5 – Доходы топ 10 приложений по подкатегориям

На рисунке 6 представлены доли основных производителей смартфонов на мировом рынке.

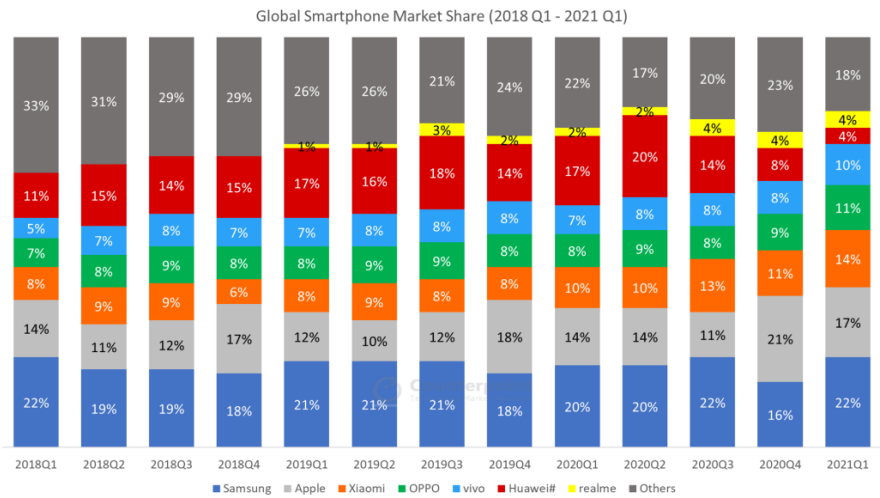


Рисунок 6 – Доли производителей смартфонов на мировом рынке

На графике видно, что доля компании Apple составляет не более 20% от всех остальных производителей, это значит, что на операционную систему IOS приходится примерно в 5 раз меньше всех устройств, так как остальные смартфоны имеют операционную систему Android. Программный продукт будет доступен большему числу пользователей, если разрабатывать его под ОС Android.

Статистика распространения разных версий системы Android представлена на рисунке 7.

ANDROID PLATFORM VERSION	API LEVEL	CUMULATIVE DISTRIBUTION
4.1 Jelly Bean	16	
4.2 Jelly Bean	17	99,8%
4.3 Jelly Bean	18	99,5%
4.4 KitKat	19	99,4%
5.0 Lollipop	21	98,0%
5.1 Lollipop	22	97,3%
6.0 Marshmallow	23	94,1%
7.0 Nougat	24	89,0%
7.1 Nougat	25	85,6%
8.0 Oreo	26	82,7%
8.1 Oreo	27	78,7%
9.0 Pie	28	69,0%
10. Q	29	50,8%
11. R	30	24,3%

Рисунок 7 – Статистика распространения разных версий Android

Можно увидеть, что 85,6 % пользователей системы Android использует версии не старше 7.1. Поэтому приложение будет разрабатываться для Android 7.1 и более новых версий системы [8].

1.4 Анализ функциональных требований

По результатам постановки задачи были выявлены следующие функции приложения:

- регистрация пользователей;
- авторизация пользователей;
- добавление блюд в прием пищи;
- удаление блюд из приема пищи;
- просмотр текущей и рекомендуемой калорийности рациона;
- подбор персонального меню.

На основе вышеперечисленных функций была создана диаграмма вариантов использования (рисунок 8) [9, 10].



Рисунок 8 – Диаграмма вариантов использования мобильного приложения

На рисунке 9 представлена спецификация варианта использования «Подобрать персональное меню».

Прецедент: подобрать персональное меню.
Краткое описание: подбор персонального меню.
Главные актеры: пользователь.
Второстепенные актеры: БД.
Предусловие: пользователь зарегистрирован и авторизован.
Основной поток: <ol style="list-style-type: none"> 1. Пользователь выбрал дату, на которую формируется меню. 2. Пользователь нажал на кнопку «подобрать персональное меню». 3. Система, учитывая рекомендуемую калорийность, подбирает рацион для пользователя, используя блюда из БД. 4. Система отображает на экране выбранное меню.
Постусловие: приложение отображает на экране персональное меню.

Рисунок 9 – Спецификация варианта использования «Подобрать персональное меню»

На рисунке 10 представлена спецификация варианта использования «Добавить блюдо в прием пищи».

Прецедент: добавить блюдо в прием пищи.
Краткое описание: пользователь выбирает блюдо и добавляет его в прием пищи.
Главные актеры: пользователь.
Второстепенные актеры: БД.
Предусловие: пользователь зарегистрирован и авторизован.
Основной поток: <ol style="list-style-type: none"> 1. Пользователь переходит на главную страницу. 2. Пользователь выбирает дату меню 3. Система загружает меню по выбранной дате, на экране отображается список приемов пищи и блюда, к ним относящиеся. 4. Пользователь нажимает кнопку «добавить» у выбранного приема пищи. 5. Система отображает на экране список блюд, их БЖУ состав и калорийность. 6. Пользователь выбирает блюдо из списка и вводит его вес. 7. Пользователь нажимает на кнопку добавить. 8. Система добавляет это блюдо в меню к выбранному приему пищи в базе данных.
Постусловие: блюдо добавлено.

Рисунок 10 – Спецификация варианта использования «Добавить блюдо в прием пищи»

Рассмотрим подробнее вариант использования «авторизоваться». Для этого варианта использования составлена диаграмма деятельности, она представлена на рисунке 11.

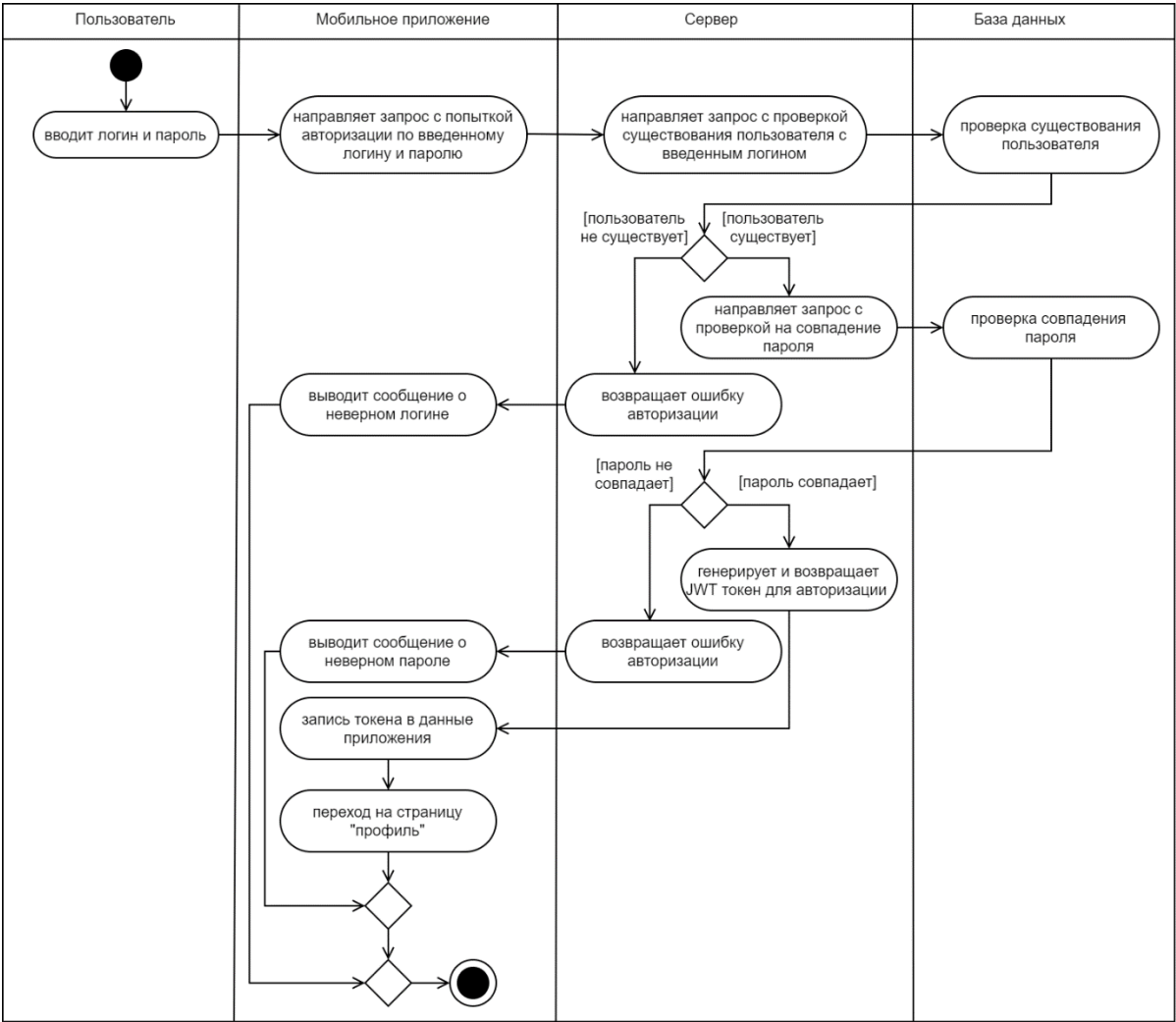


Рисунок 11 – Диаграмма деятельности для варианта использования «Авторизоваться»

В результате анализа функциональных требований были выявлены основные функции разрабатываемого приложения, представленные на диаграмме вариантов использования. Также, для некоторых вариантов использования были составлены спецификации.

1.5 Анализ нефункциональных требований

Разработка серверной части приложения должна быть выполнена на языке программирования C#, в среде программирования Microsoft Visual Studio 2022 preview. Она должна работать под управлением операционной системы Windows 10 и выше [11].

Клиентская часть приложения должна быть реализована на языке программирования C# с использованием фреймворка MAUI в среде программирования Microsoft Visual Studio 2022 preview [12]. Она должна представлять собой мобильное приложение с удобным и интуитивно понятным пользовательским интерфейсом. Кроме того, программное средство должно показывать стабильную работу при любых входных данных и действиях пользователя. Клиентское приложение должно успешно работать под управлением ОС Android 7.1 и выше.

1.6 Разработка концептуальной модели базы данных

На основе анализа предметной области построена концептуальная модель базы данных. Без учета деталей возможной реализации базы данных были выделены отдельные сущности. Составлена концептуальная модель связей между сущностями в нотации «сущность-связь». Данная диаграмма представлена на рисунке 12.

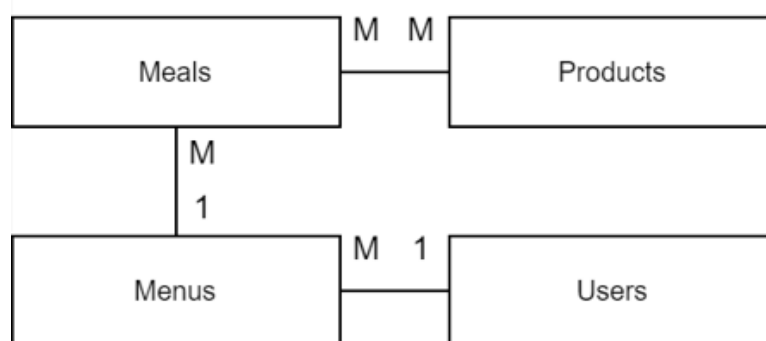


Рисунок 12 – Диаграмма «сущность-связь» для базы данных мобильного приложения

На данном уровне моделирования были спроектированы только атрибуты, несущие информационную нагрузку для каждой сущности, не рассматривая фиктивные атрибуты и внешние ключи.

1.7 Планирование разработки и оценка бюджета

Качественное выполнение работ по любому проекту невозможно при отсутствии грамотного планирования.

Планирование разработки и оценка бюджета была выполнена при помощи программы для управления проектами Microsoft Project [13, 14]. Проект был начат 1 февраля 2022 года. При планировании разработки было выделено 23 задачи. Список задач и их описание представлено на таблице 2.

Таблица 2 – Перечень задач проекта

№	Название задачи	Длительность	Предшественники
1	Начало проекта	-	-
2	Анализ требований	-	-
3	Анализ предметной области	3 дня	1
4	Анализ приложений-аналогов	4 дня	3
5	Анализ функциональных требований	2 дня	4
6	Постановка задачи	1 день	5
7	Анализ требований завершен	-	6
8	Проектирование приложения	-	-
9	Проектирование базы данных	3 дня	7
10	Проектирование архитектуры приложения	3 дня	9
11	Проектирование интерфейса пользователя	4 дня	10
12	Проектирование приложения завершено	-	11
13	Реализация приложения	-	-
14	Реализация базы данных	8 дней	9
15	Реализация серверной части приложения	14 дней	12;14
16	Реализация клиентской части приложения	18 дней	12;14
17	Реализация приложения завершена	-	15;16
18	Отладка	-	-
19	Тестирование работы серверной части приложения	3 дня	17
20	Тестирование работы клиентской части приложения	4 дня	17
21	Исправление ошибок	8 дней	19; 20

Продолжение таблицы 2

№	Название задачи	Длительность	Предшественники
22	Отладка завершена	-	21
23	Составление и оформление документации	7 дней	22
24	Конец проекта	-	23

Таким образом, для реализации проекта нужно выполнить 24 задачи. В таблице 3 представлено описание затрачиваемых ресурсов.

Таблица 3 – Список ресурсов проекта

Название ресурса	Тип	Стандартная ставка	Ставка сверхурочных
Программист	Трудовой	200 Р/ч	400 Р/ч
Аналитик	Трудовой	200 Р/ч	400 Р/ч
Тестировщик	Трудовой	150 Р/ч	300 Р/ч
Дизайнер	Трудовой	150 Р/ч	300 Р/ч
Интернет	Затраты	-	-
Компьютер	Материальный	38000 руб.	-
Мобильное устройство	Материальный	15000 руб.	-

Распределение ресурсов по задачам представлено в таблице 4.

Таблица 4 – Назначение ресурсов

№	Название задачи	Ресурс	Единицы
1	Начало проекта		
2	Анализ требований		
3	Анализ предметной области	Компьютер Аналитик Интернет	1 шт. 50% 100 руб.
4	Анализ приложений-аналогов	Аналитик Интернет	50% 130 руб.
5	Анализ функциональных требований	Аналитик Интернет	50% 70 руб.
6	Постановка задачи	Аналитик Программист	100% 100%
7	Анализ требований завершен		

Продолжение таблицы 4

№	Название задачи	Ресурс	Единицы
8	Проектирование приложения		
9	Проектирование базы данных	Аналитик	100%
		Программист	100%
10	Проектирование архитектуры приложения	Аналитик	50%
		Программист	100%
11	Проектирование интерфейса пользователя	Аналитик	50%
		Программист	100%
		Дизайнер	100%
12	Проектирование приложения завершено		
13	Реализация приложения		
14	Реализация базы данных	Аналитик	50%
		Программист	100%
		Интернет	200 руб.
15	Реализация серверной части приложения	Программист	100%
		Интернет	500 руб.
16	Реализация клиентской части приложения	Программист	100%
		Интернет	500 руб.
17	Реализация приложения завершена		
18	Отладка		
19	Тестирование работы серверной части приложения	Программист	50%
		Тестировщик	100%
20	Тестирование работы клиентской части приложения	Программист	50%
		Тестировщик	100%
		Мобильное устройство	1 шт.
21	Исправление ошибок		
22	Отладка завершена		
23	Составление и оформление документации	Аналитик	50%
		Программист	50%
24	Конец проекта		

Таким образом, реализацией проекта будут заниматься программист, аналитик и дизайнер, которым для работы нужен компьютер и мобильное устройство, а также доступ к интернету. Построен сетевой график (рисунок 13).

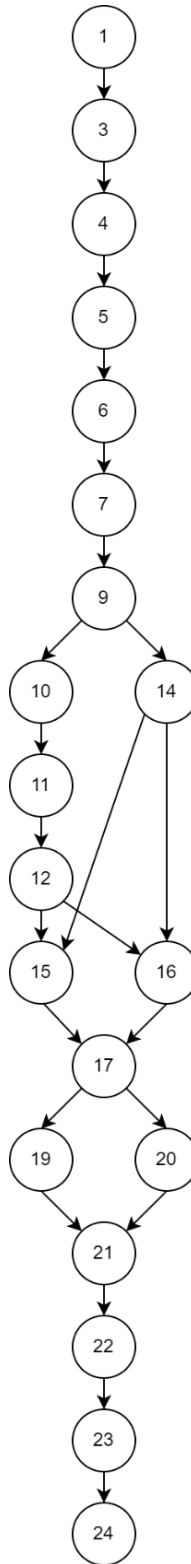


Рисунок 13 – Сетевой график проекта

Перечень задач в Microsoft Project представлен на рисунке 14.

	Название задачи	Длительность	Начало	Окончание	Предшественники	Названия ресурсов
1	Начало проекта	0 дней	Вт 01.02.22	Вт 01.02.22		
2	Анализ требований	10 дней	Вт 01.02.22	Пн 14.02.22		
3	Анализ предметной области	3 дней	Вт 01.02.22	Чт 03.02.22	1	Компьютер[1];Аналитик[50%];Интернет[100,00 Р]
4	Анализ приложений-аналогов	4 дней	Пт 04.02.22	Ср 09.02.22	3	Аналитик[50%];Интернет[130,00 Р]
5	Анализ функциональных требований	2 дней	Чт 10.02.22	Пт 11.02.22	4	Аналитик[50%];Интернет[70,00 Р]
6	Постановка задачи	1 день	Пн 14.02.22	Пн 14.02.22	5	Аналитик;Программист
7	Анализ требований завершен	0 дней	Пн 14.02.22	Пн 14.02.22	6	
8	Проектирование приложения	10 дней	Вт 15.02.22	Пн 28.02.22		
9	Проектирование базы данных	3 дней	Вт 15.02.22	Чт 17.02.22	7	Аналитик;Программист
10	Проектирование архитектуры приложения	3 дней	Пт 18.02.22	Вт 22.02.22	9	Программист;Аналитик[50%]
11	Проектирование интерфейса пользователя	4 дней	Ср 23.02.22	Пн 28.02.22	10	Программист;Дизайнер;Аналитик[50%]
12	Проектирование приложения завершенно	0 дней	Пн 28.02.22	Пн 28.02.22	11	
13	Реализация приложения	26 дней	Пт 18.02.22	Пт 25.03.22		
14	Реализация базы данных	8 дней	Пт 18.02.22	Вт 01.03.22	9	Аналитик[50%];Программист;Интернет[200,00 Р]
15	Реализация серверной части приложения	14 дней	Ср 02.03.22	Пн 21.03.22	12;14	Программист;Интернет[500,00 Р]
16	Реализация клиентской части приложения	18 дней	Ср 02.03.22	Пт 25.03.22	12;14	Программист;Интернет[500,00 Р]
17	Реализация приложения завершена	0 дней	Пт 25.03.22	Пт 25.03.22	15;16	
18	Отладка	12 дней	Пн 28.03.22	Вт 12.04.22		
19	Тестирование работы серверной части приложения	3 дней	Пн 28.03.22	Ср 30.03.22	17	Программист[50%];Тестировщик
20	Тестирование работы клиентской части приложения	4 дней	Пн 28.03.22	Чт 31.03.22	17	Программист[50%];Тестировщик;Мобильное устройство[1]
21	Исправление ошибок	8 дней	Пт 01.04.22	Вт 12.04.22	19;20	Программист;Тестировщик[25%]
22	Отладка завершена	0 дней	Вт 12.04.22	Вт 12.04.22	21	
23	Составление и оформление документации	7 дней	Ср 13.04.22	Чт 21.04.22	22	Аналитик[50%];Программист[50%]
24	Конец проекта	0 дней	Чт 21.04.22	Чт 21.04.22	23	

Рисунок 14 – Список задач проекта

Диаграмма Ганта, спроектированная в Microsoft Project представлена на рисунке 15.

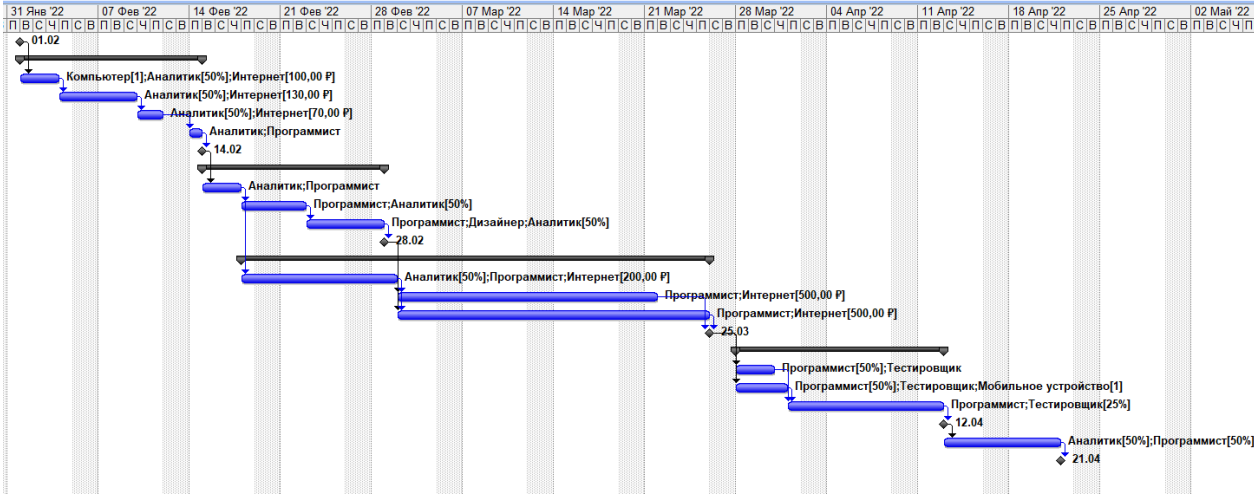


Рисунок 15 – Диаграмма Ганта

На рисунках 16, 17 представлены графики трудовых ресурсов «Аналитик», «Программист».

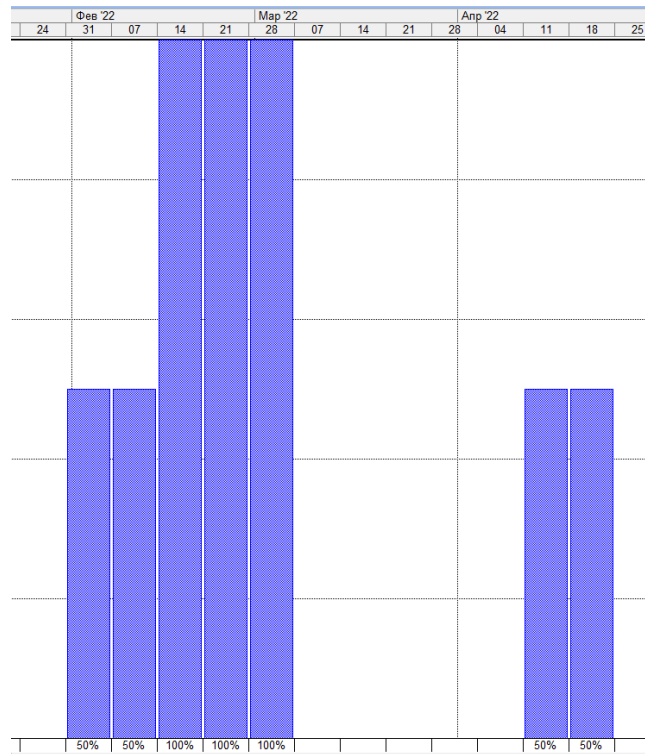


Рисунок 16 – График загрузки ресурса «Аналитик»

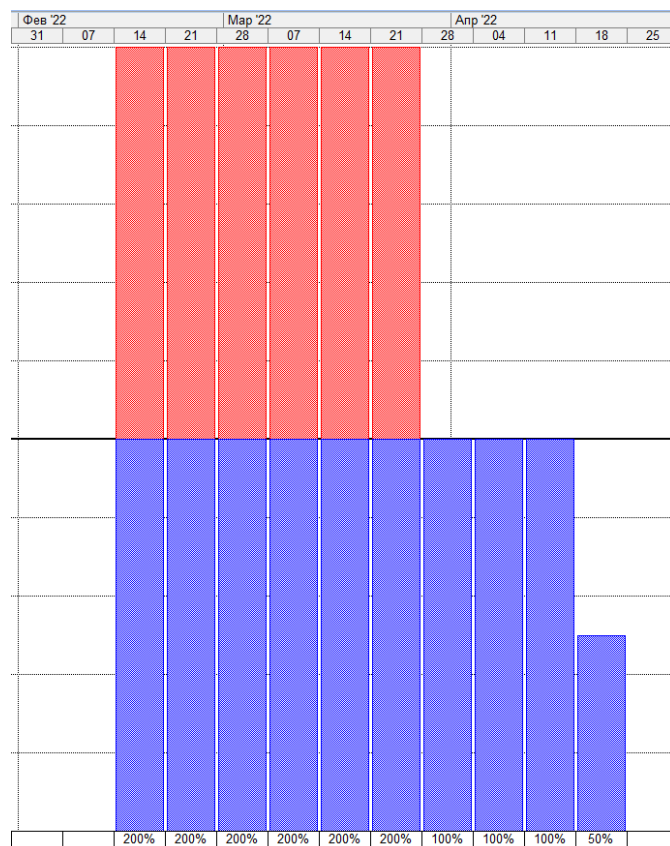


Рисунок 17 – График загрузки ресурса «Программист»

На рисунках 18, 19 представлены графики трудовых ресурсов «Дизайнер», «Тестировщик».

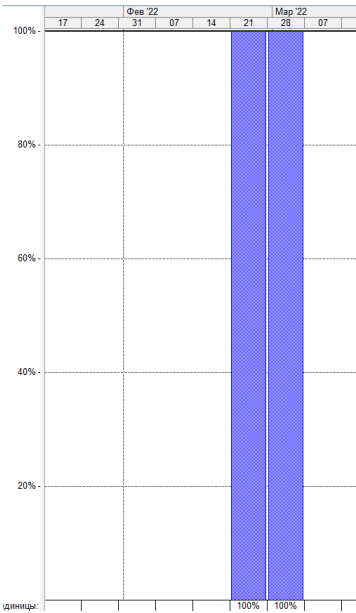


Рисунок 18 – График загруженности ресурса «Дизайнер»

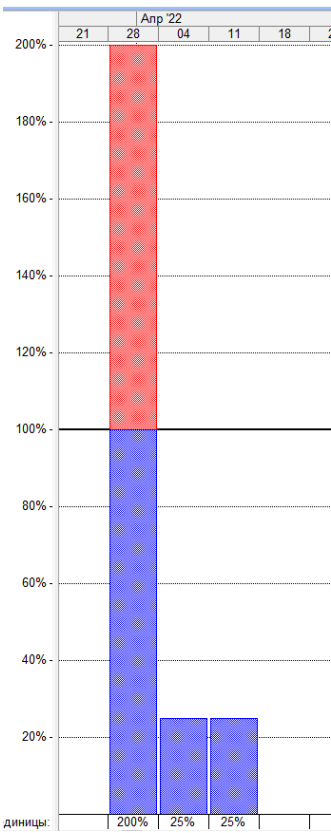


Рисунок 19 – График загруженности ресурса «Тестировщик»

На рисунке 20 представлен обзорный отчет проекта. Реализация проекта займет 58 дней, а предполагаемые затраты 206900 рублей.

Даты			
Начало:	Вт 01.02.22	Окончание:	Чт 21.04.22
Баз. начало:	НД	Баз. окончание:	НД
Факт. начало:	НД	Факт. окончание:	НД
Отклон. начала:	0 дней	Отклон. окончания:	0 дней
Длительность			
Текущий план:	58 дней	Оставшаяся:	58 дней
Базовый план:	0 дней?	Фактическая:	0 дней
Отклонение:	58 дней	Процент завершения:	0%
Трудозатраты			
Текущий план:	788 ч	Оставшиеся:	788 ч
Базовый план:	0 ч	Фактические:	0 ч
Отклонение:	788 ч	Процент завершения:	0%
Затраты			
Текущий план:	206 900,00 Р	Оставшиеся:	206 900,00 Р
Базовый план:	0,00 Р	Фактические:	0,00 Р
Отклонение:	206 900,00 Р		
Состояние задачи		Состояние ресурса	
Еще не начатые задачи	24	Трудовые ресурсы:	3
Выполняющиеся задачи	0	Трудовые ресурсы с прев. дост	2
Завершенные задачи:	0	Материальные ресурсы:	2
Итого задач:	24	Итого ресурсов:	7

Рисунок 20 – Обзорный отчет проекта

На основе произведенных оценок стоимости и сроков проекта можно сделать вывод о его экономической целесообразности.

2 Проектирование мобильного приложения

2.1 Выбор технологий и инструментальных средств разработки

При разработке программного продукта было решено придерживаться методологии RUP. Особенности данной методологии является раннее обнаружение и устранение основных рисков, акцент на выполнении требований заказчиков, а также контроль качества на этапах разработки программного проекта [15].

Методология RUP предполагает итеративную модель разработки. В конце очередной итерации необходимо получить промежуточную версию разрабатываемого продукта. Цикл разработки содержит в себе 4 этапа.

На начальном этапе формируется видение проекта, создается экономическое обоснование, а также оцениваются риски и определяется основная функциональность продукта. На этапе уточнения анализируется предметная область, документируются требования. Также происходит более точная оценка сроков и суммы проекта. На этапе построения происходит основная реализация функциональной части продукта. На этапе внедрения создается конечная версия.

Выбор данной методологии обусловлен такими достоинствами как гибкость разработки (например, в случае выявления ошибок или изменения требований).

Для документирования требований и моделирования проекта будет использован язык моделирования UML. Для построения диаграмм – средство UML-моделирования ArgoUML.

Для планирования и оценки бюджета проекта будет использовано приложение Microsoft Project. С его помощью будет составлен календарный план проекта, а также рассчитаны затраты для разработки приложения.

Для создания клиентской части, выполненной в качестве мобильного приложения, была выбрана среда программирования Visual Studio 2022

preview. Приложение будет разрабатываться на языке C# с использованием фреймворка MAUI, позволяющего создавать мобильные приложения для ОС Android. Для хранения данных выбрана СУБД Microsoft SQL Server. Реализация базы данных будет производиться с помощью Entity Framework, который позволяет, на основе написанных классов на языке C#, генерировать таблицы в базе данных, а также удобно обращаться к данным, используя C# Linq [16, 17].

2.2 Проектирование архитектуры программного обеспечения

Было принято решение использовать в приложении трехуровневую архитектуру «клиент-сервер». Ее схема представлена на рисунке 21.

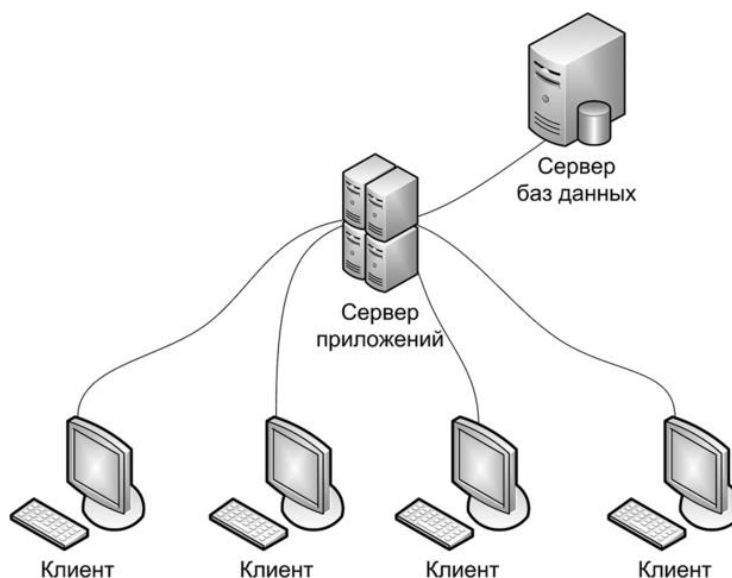


Рисунок 21 – Трехуровневая архитектура «клиент-сервер»

На нижнем уровне на смартфонах пользователей расположены клиентские приложения, которые реализуют логику представлений. Клиентская часть разрабатываемого программного обеспечения (ПО) обеспечивает пользователю удобный интерфейс, позволяющий просматривать необходимые ему данные из базы данных (БД) и взаимодействовать с ними, а также реализует возможность удаленно обращаться к серверу приложений.

На среднем уровне располагается сервер приложений, который обеспечивает обмен данными между БД и клиентскими приложениями.

На верхнем уровне располагается сервер БД, который принимает информацию от сервера приложений. Сервер БД выделен для обработки и хранения данных.

Достоинством архитектуры является то, что все данные хранятся на стороне сервера БД. Это позволяет минимизировать объем клиентского приложения. А также сервер БД не нагружен операциями, совершаемыми сервером приложений [18].

2.3 Проектирование логической модели базы данных

На основе анализа предметной области и функциональных требований была построена логическая модель базы данных. Она представлена в нотации «сущность-связь» на рисунке 22.

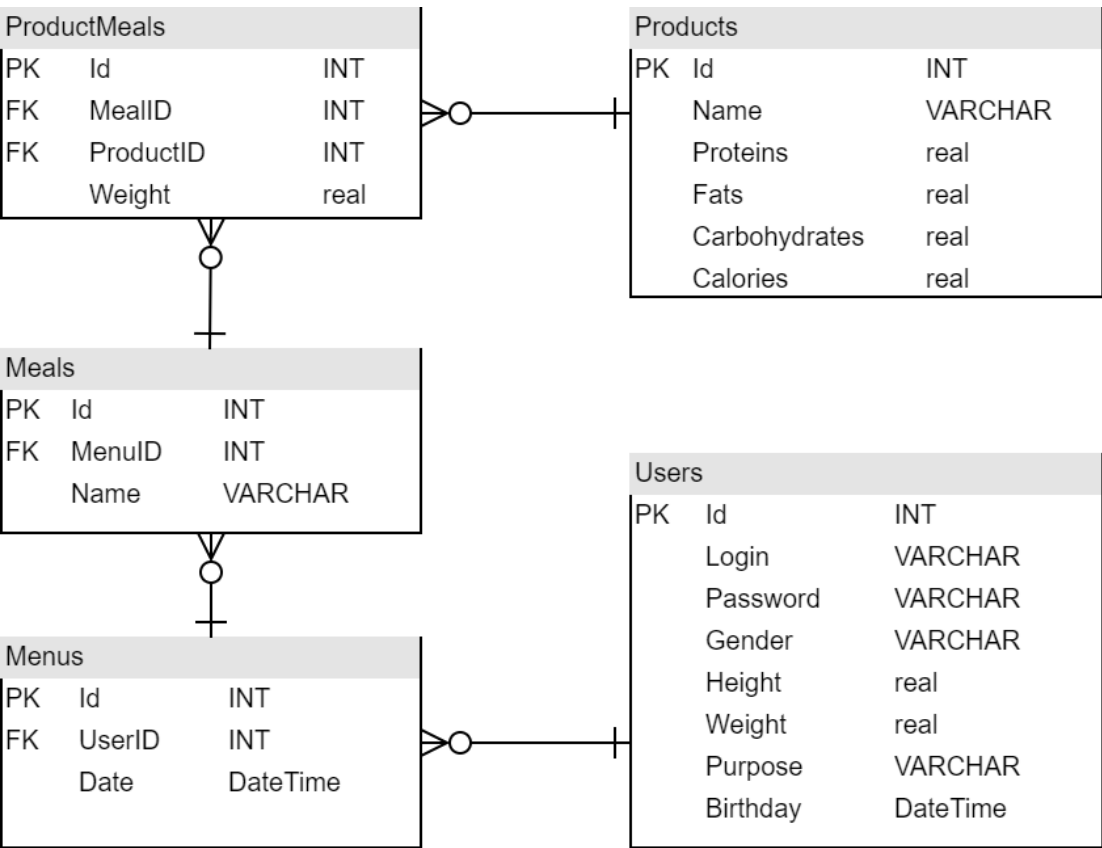


Рисунок 22 – Логическая модель данных

Сущности логической модели данных описаны в таблице 5.

Таблица 5 – Описание сущностей

№	Название	Описание	Количество экземпляров	Режим обновления
1	Users	Перечень пользователей системы	1000	Новые записи добавляются при регистрации пользователя, при изменении данных пользователем данные обновляются
2	Menus	Перечень меню для каждого пользователя	100000	Новые записи добавляются при выборе новой даты для отображения меню
3	Meals	Перечень приемов пищи	400000	Новые записи добавляются при выборе новой даты для отображения меню
4	Products	Перечень продуктов и блюд	10000	Новые записи добавляются при необходимости обновить информацию о продуктах (примерно раз в месяц)
5	ProductMeals	Перечень продуктов	1000000	Новые записи добавляет пользователь при добавлении продукта в меню или при подборе меню

В таблице 6 представлено описание связей между сущностями.

Таблица 6 – Описание связей

№	Тип сущности 1	Тип сущности 2	Описание	Мощность	Обязательность
1	Users	Menus	У каждого пользователя может быть список меню за каждый день, на который они составлены	1:M	Не обязательная
2	Menus	Meals	В каждом меню содержатся приемы пищи	1:M	Обязательная
3	Meals	ProductMeals	В каждом приеме пищи может быть список объектов ProductMeal, которые содержат в себе вес и Id продукта	1:M	Не обязательная
4	ProductMeals	Products	В каждом ProductMeal содержится Id продукта	M:1	Обязательная

Разработанная логическая модель БД не отражает деталей реализации под конкретную базу данных. Таким образом, данная модель является универсальной.

2.4 Проектирование интерфейса пользователя

С учетом функциональных требований, описанных в разделе анализа требований, разработан интерфейс пользователя для клиентской части. Он содержит такие страницы как стартовая страница, страницы регистрации и авторизации, вкладка профиль, вкладка главное, страница изменения информации о пользователе, страница добавления продукта.

Стартовая страница представлена на рисунке 23.

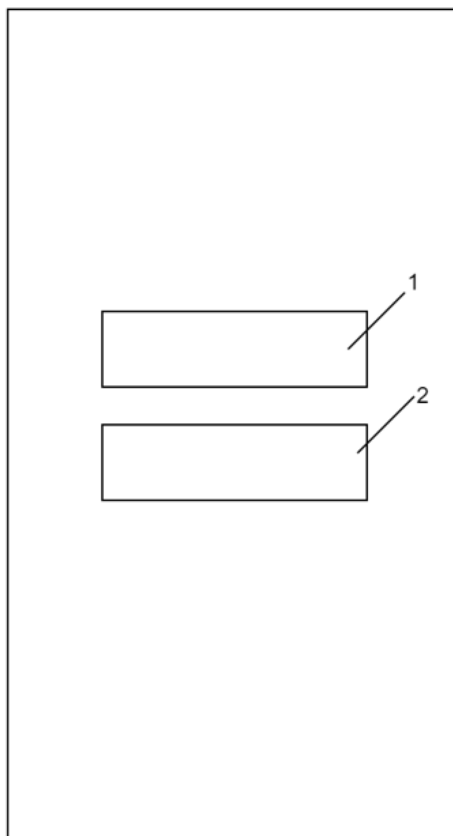


Рисунок 23 – Макет стартовой страницы

Рассмотрим элементы стартовой страницы, представленной на рисунке 23:

1 – кнопка вход;

2 – кнопка регистрация.

При нажатии на кнопку вход пользователь перейдет на страницу авторизации. При нажатии на кнопку регистрация пользователь перейдет на страницу регистрации. Страница авторизации представлена на рисунке 24.

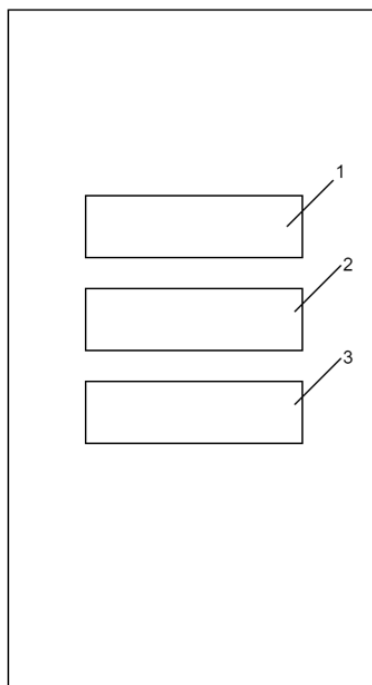


Рисунок 24 – Макет страницы авторизации

Рассмотрим элементы страницы авторизации, представленной на рисунке 24:

- 1 – поле для ввода логина;
- 2 – поле для ввода пароля;
- 3 – кнопка войти.

Для авторизации пользователю необходимо ввести свой логин и пароль, затем нажать на кнопку войти. Если данные верны, то пользователь попадет на вкладку профиль, если же нет, то система выведет ошибку. Для успешной авторизации пользователю необходимо сначала зарегистрироваться. Макет страницы регистрации аналогичен макету страницы регистрации, только там пользователь вводит не существующую пару логин и пароль, а придумывает новый уникальный логин и пароль определенной длины. Для завершения регистрации пользователь со страницы регистрации попадает на страницу ввода данных о пользователе, представленной на рисунке 25.

Рисунок 25 – Макет страницы ввода данных о пользователе

Рассмотрим элементы страницы ввода данных о пользователе, представленной на рисунке 25:

- 1 – кнопка сохранить;
- 2 – элемент с надписью о вводе роста;
- 3 – поле ввода роста;
- 4 – элемент с надписью о вводе веса;
- 5 – поле ввода веса;
- 6 – элемент с надписью о вводе информации о поле;
- 7 – элемент выбора пола;
- 8 – элемент с надписью о вводе информации о дне рождении;
- 9 – элемент выбора даты, предназначенный для ввода дня рождения;
- 10 – элемент с надписью о выборе цели;
- 11 – элемент выбора цели из предложенных вариантов.

При успешном вводе данных о пользователе, приложение перейдёт во вкладку профиль, макет которой представлен на рисунке 26. А в случае неудачи система выдаст ошибку и можно будет повторить попытку.

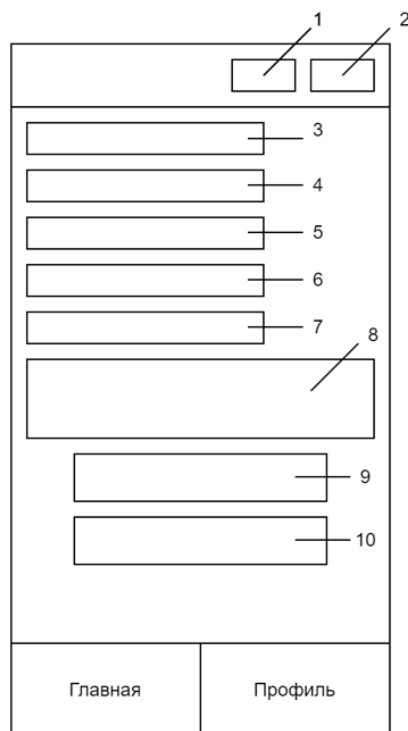


Рисунок 26 – Макет вкладки профиль

Рассмотрим элементы вкладки профиль, представленной на рисунке 26:

- 1 – кнопка редактировать;
- 2 – кнопка выйти;
- 3 – элемент с информацией о росте пользователя;
- 4 – элемент с информацией о весе пользователя;
- 5 – элемент с информацией о возрасте пользователя;
- 6 – элемент с информацией о поле пользователя;
- 7 – элемент с информацией о ИМТ пользователя;
- 8 – элемент с информацией о норме килокалорий для пользователя и рекомендации по выбору цели;
- 9 – элемент выбора даты, на какое число сформировать персональное меню для пользователя;
- 10 – кнопка сформировать персональное меню.

На вкладке профиль пользователь может ознакомиться с той информацией, которую он вводил при регистрации и посмотреть свою норму калорий. Также пользователь при нажатии кнопки редактировать может

изменить свои данные, введенные при регистрации. Еще он может выйти из своего аккаунта с помощью соответствующей кнопки. Кроме вкладки профиль в приложении существует еще и вкладка главная, которая показана на рисунке 27.

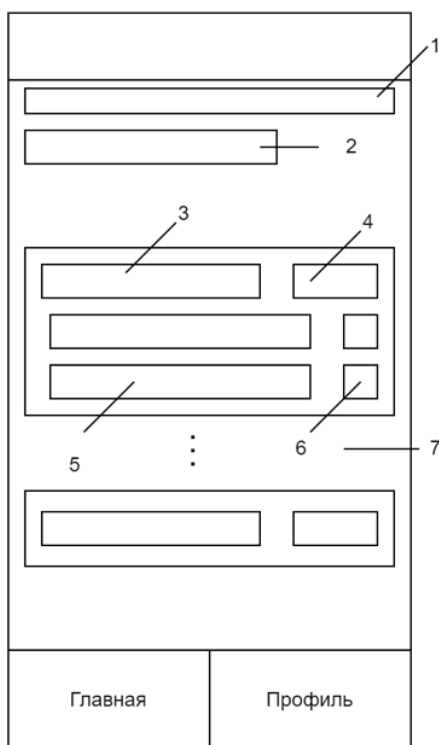


Рисунок 27 – Макет вкладки главная

Рассмотрим элементы вкладки главная, представленной на рисунке 27:

- 1 – текстовый элемент, который показывает уже потребленное количество калорий и рекомендованное программой количество калорий;
- 2 – элемент выбора даты;
- 3 – текстовый элемент, отображающий название приема пищи;
- 4 – кнопка добавления блюд или продуктов в прием пищи;
- 5 – текстовый элемент, отображающий название блюда или продукта, а также его массу и калорийность;
- 6 – кнопка удаления блюда или продукта из приема пищи;
- 7 – список приемов пищи.

При переходе на главную вкладку пользователю показывается меню на сегодняшний день, если пользователь ничего не добавлял или не выполнял

подбор персонального меню, то в этом меню будут только приемы пищи, такие как «завтрак», «обед», «ужин» и «перекусы». В каждый из которых пользователь может добавить новый продукт с помощью кнопки, расположенной у соответствующего приема пищи. Так же просто и удалить продукт, достаточно нажать кнопку, расположенную рядом с соответствующим продуктом. Макет страницы добавления блюд и продуктов представлен на рисунке 28.

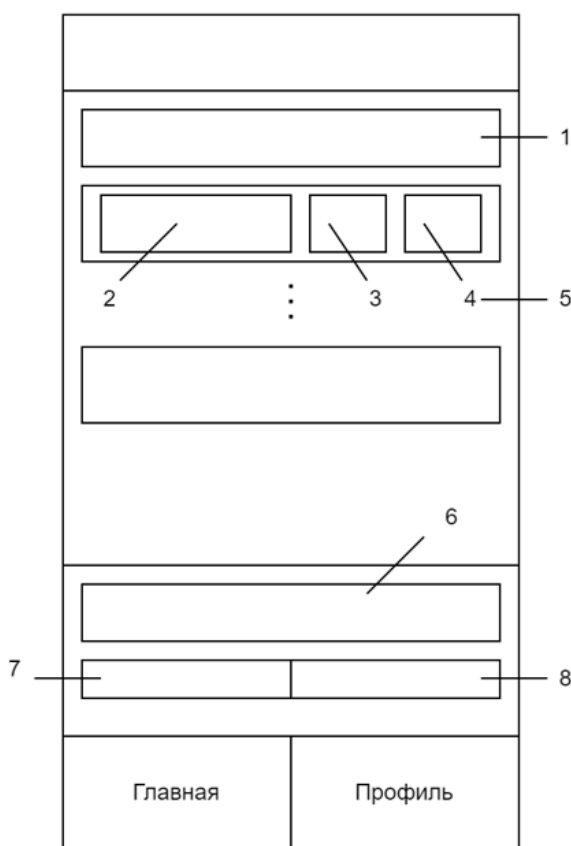


Рисунок 28 – Макет страницы добавления продукта

Рассмотрим элементы страницы добавления продукта, представленной на рисунке 28:

- 1 – элемент поиска;
- 2 – текстовый элемент, отображающий название продукта;
- 3 – текстовый элемент, отображающий БЖУ состав продукта;
- 4 – текстовый элемент, отображающий калорийность продукта;
- 5 – список продуктов;

- 6 – текстовый элемент, отображает выбранный продукт;
- 7 – поле для ввода веса продукта;
- 8 – кнопка добавить.

На странице добавления продуктов, можно найти нужный продукт листая список, а можно воспользоваться поиском по названию. Когда нужный продукт высветился на экране достаточно нажать на него, чтобы выбрать его, внизу страницы необходимо указать вес и нажать кнопку добавить, в таком случае продукт добавится к выбранному ранее приему пищи.

3 Реализация мобильного приложения

3.1 Реализация базы данных

На основе логической модели базы данных были разработаны классы на языке C#, на основе которых, с использованием Entity Framework будет сформирована БД. Диаграмма классов представлена на рисунке 29.

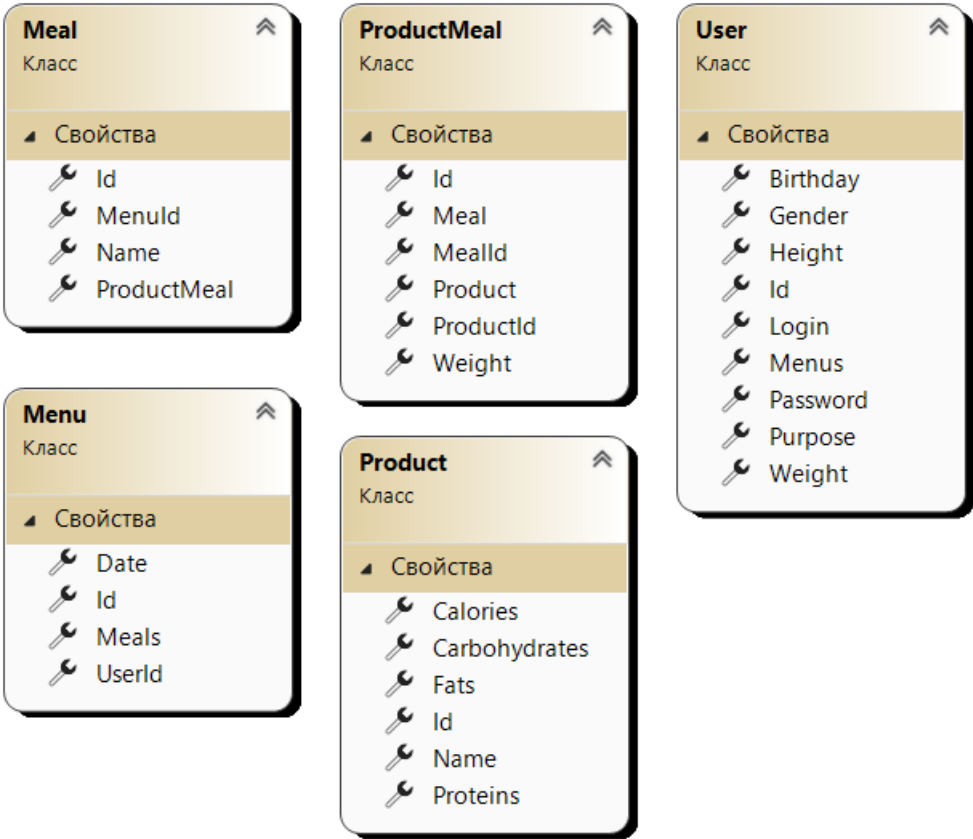


Рисунок 29 – Диаграмма классов для создания БД

Описание классов для создания базы данных вместе с назначениями представлено в таблице 7.

Таблица 7 – Описание классов для создания базы данных

Класс	Назначение
User	Отвечает за представление данных пользователя, содержит в себе информацию о логине, пароле, росте, весе, цели, поле и дне рождения пользователя.
Product	Отвечает за представление данных продукта, содержит в себе информацию о названии, количестве белков, жиров и углеводов, а также о калорийности продукта.
Menu	Отвечает за представление данных о меню пользователя, содержит в себе информацию о дате, а также идентификатор пользователя и приема пищи.
Meal	Отвечает за представление данных о приемах пищи в меню, содержит в себе информацию о названии приема пищи и идентификаторы меню и продукта в приеме пищи.
ProductMeal	Отвечает за представление данных о продуктах в приеме пищи, содержит в себе информацию о весе продукта и идентификатор продукта.

На основе написанных классов, используя Entity Framework, была сгенерирована миграция, исходный код которой представлен в приложении Б. В результате исполнения этой миграции создана база данных под управлением СУБД Microsoft SQL Server.

3.2 Реализация серверной части

Разработка серверной части программного обеспечения производилась в Microsoft Visual Studio 2022 preview на языке C# на платформе ASP.NET [19, 20]. Приложение разработано с использованием паттерна MVC, значит классы можно условно разделить на слой модели и слой контроллеров [21]. Слой контроллеров предназначен для обработки HTTP запросов. Диаграмма классов слоя контроллеров представлена на рисунке 30.

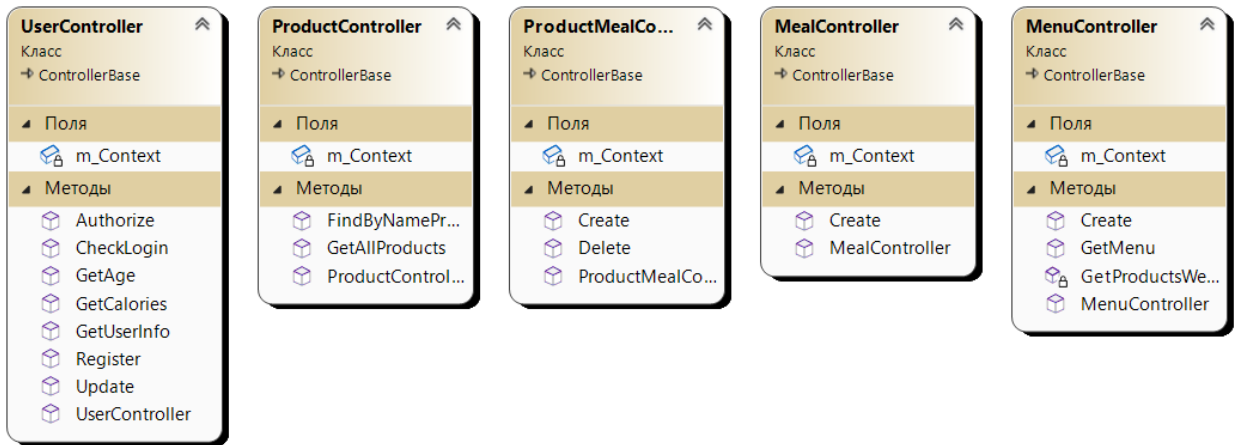


Рисунок 30 – Диаграмма классов слоя контроллеров

Описание классов слоя контроллеров представлено в таблице 8.

Таблица 8 – Описание классов слоя контроллеров

Класс	Назначение
UserController	Отвечает за обработку запросов на добавление новых пользователей в систему, авторизацию пользователей, получение информации о пользователе и обновление данных пользователя.
ProductController	Отвечает за обработку запросов на получение списка всех продуктов, и на получение списка продуктов, содержащих в названии переданную строку.
MenuController	Отвечает за обработку запросов на создание нового меню и получение меню.
MealController	Отвечает за обработку запросов на создание приемов пищи.
ProductMealController	Отвечает за обработку запросов на добавление и удаление продуктов из приемов пищи.

Рассмотрим HTTP методы, реализованные в классах слоя контроллеров более подробно. Спецификация этих методов представлена в таблице 9.

Таблица 9 – Спецификация HTTP методов

Метод	Параметры	Возвращаемые данные	Краткое описание
POST User/register	Тело запроса: объект DTO в формате JSON, представляющий информацию о пользователе	Успех или неудача добавления пользователя	Позволяет добавить нового пользователя

Продолжение таблицы 9

Метод	Параметры	Возвращаемые данные	Краткое описание
POST User/authorize	Тело запроса: объект DTO в формате JSON, содержащий логин и пароль	JWT токен для авторизации	Позволяет выполнить авторизацию пользователя
PUT User/update	Заголовок запроса: JWT токен Тело запроса: объект DTO в формате JSON, представляющий информацию о пользователе	Успех или неудача обновления информации о пользователе	Позволяет выполнить обновление данных о пользователе
GET User	Заголовок запроса: JWT токен	Объект DTO в формате JSON, представляющий информацию о пользователе	Позволяет получить информацию о пользователе
POST ProductMeal/ create	Заголовок запроса: JWT токен Тело запроса: объект DTO в формате JSON, представляющий информацию о продукте в приеме пищи	Успех или неудача создания продукта в приеме пищи	Позволяет добавить продукт в приеме пищи
POST ProductMeal/ delete	Заголовок запроса: JWT токен Параметр: id продукта в приеме пищи	Успех или неудача удаления продукта в приеме пищи	Позволяет удалить продукт в приеме пищи
GET Products/ AllProducts	Заголовок запроса: JWT токен	Объект DTO в формате JSON, содержащий список всех продуктов	Позволяет получить список всех продуктов
GET Products/ FindByName	Заголовок запроса: JWT токен Параметр: строка поиска	Объект DTO в формате JSON, содержащий список продуктов	Позволяет получить список продуктов, подходящий по
GET Menu/ Create	Заголовок запроса: JWT токен Параметр: дата	Объект DTO в формате JSON, содержащий информацию о меню, приемах пищи и их содержанием	Позволяет выполнить подбор персонального меню на определенную дату, при этом информация о старом меню удаляется
GET Menu	Заголовок запроса: JWT токен Параметр: дата	Объект DTO в формате JSON, содержащий информацию о меню	Позволяет получить информацию о меню на определенную дату, если на эту дату нет меню, создает его

Листинг реализации серверной части приведен в приложении В.

3.3 Реализация клиентской части

Разработка мобильного приложения производилась на языке C# в Microsoft Visual Studio 2022 preview, с использованием фреймворка MAUI, который позволяет разрабатывать мобильные приложения. Для клиентской части было решено использовать паттерн проектирования Model-View-ViewModel [22, 23]. Страницы мобильного приложения написаны на XAML. Структура программы в виде диаграммы компонентов представлена на рисунке 31.

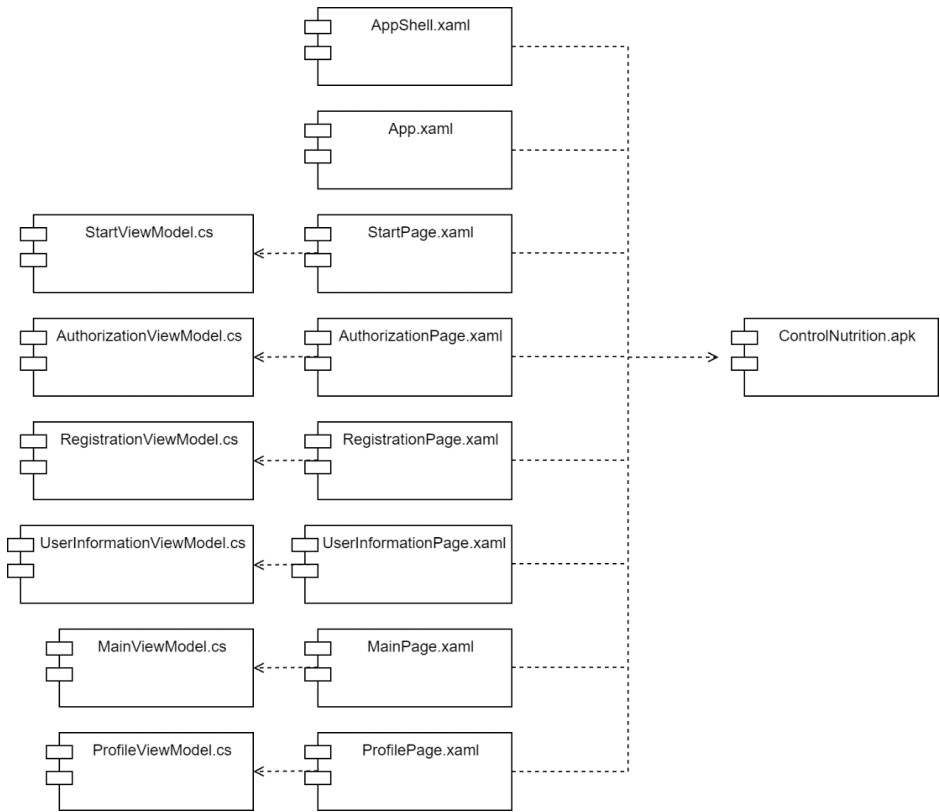


Рисунок 31 – Диаграмма компонентов клиентской части приложения

Описание компонентов представлено в таблице 10.

Таблица 10 – Описание компонентов клиентской части приложения

Компонент	Назначение
AppShell.xaml	XAML-файл, в котором регистрируется стартовая страница и вкладки приложения

Продолжение таблицы 10

Компонент	Назначение
App.xaml	ХАМЛ-файл, в котором подключаются цвета и стили приложения
SartPage.xaml	ХАМЛ-файл, в котором описывается представление стартовой страницы
StartViewModel.cs	Файл класса, описывает логику работы стартовой страницы
AuthorizationPage.xaml	ХАМЛ-файл, в котором описывается представление страницы авторизации
AuthorizationViewModel.cs	Файл класса, описывает логику работы страницы авторизации
RegistrationPage.xaml	ХАМЛ-файл, в котором описывается представление страницы регистрации
RegistrationViewModel.cs	Файл класса, описывает логику работы страницы регистрации
UserInfoPage.xaml	ХАМЛ-файл, в котором описывается представление страницы добавления информации о пользователе
UserInfoViewModel.cs	Файл класса, описывает логику работы страницы добавления информации о пользователе
MainPage.xaml	ХАМЛ-файл, в котором описывается представление главной страницы
MainViewModel.cs	Файл класса, описывает логику работы главной страницы
ProfilePage.xaml	ХАМЛ-файл, в котором описывается представление вкладки профиль
ProfileViewModel.cs	Файл класса, описывает логику работы вкладки профиль
ControlNutrition.apk	Исполняемый файл приложения

Физическое расположение компонентов системы показано на диаграмме развертывания, которая представлена на рисунке 32.

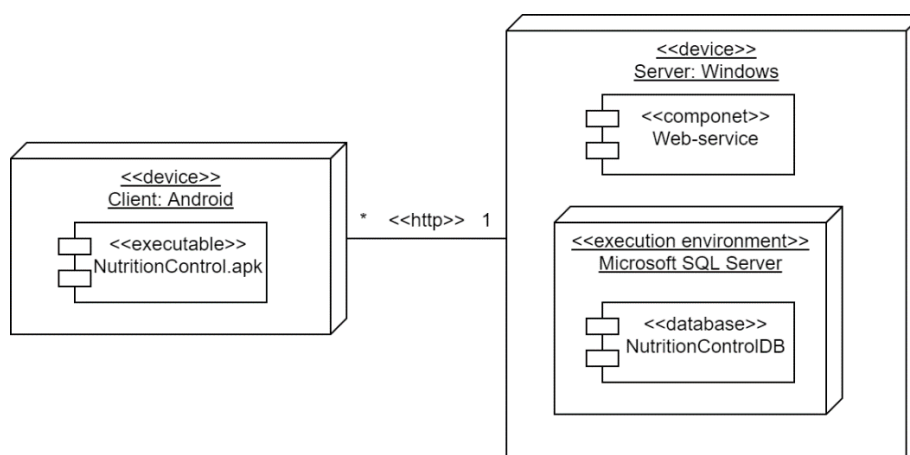


Рисунок 32 – Диаграмма развертывания приложения

Как видно на диаграмме развертывания, взаимодействие между сервером и клиентом происходит посредством протокола HTTP. Клиент представлен мобильным приложением, работающим на устройстве под управлением ОС Android, а сервер – компьютером под управлением ОС Windows. На сервере располагается БД и API. Листинг реализации клиентской части приведен в приложении Г.

4 Тестирование и оценка качества приложения

4.1 Тестирование серверной части

4.1.1 Функциональное тестирование

Тестирование веб-приложения проводилось при помощи программного средства Swagger. Результаты тестирования функционала серверной части представлены в таблице 11. Скриншоты тестов представлены в приложении Д.

Таблица 11 – Результаты тестирования серверной части

№	Описание теста	Запрос		Результат
		Метод	Параметры	
1	Добавление пользователя в систему	POST	логин=newuser1, пароль= pass1234, цель=Похудеть, пол=Мужской, рост=180, вес=80, день рождения= 01.01.2000	Тест пройден. Пользователь добавлен в систему. Код ответа HTTP – 200 (рисунок Д.1).
2	Авторизация пользователя	POST	логин=newuser1, пароль= pass1234	Тест пройден. Получен токен авторизации. Код ответа HTTP – 200 (рисунок Д.2).
3	Изменение данных о пользователе	PUT	цель= Поддерживать вес, пол=Мужской, рост=180, вес=80, день рождения= 01.01.2000	Тест пройден. Пользователь изменен. Код ответа HTTP – 200 (рисунок Д.3).
4	Получение данных о пользователе	GET	-	Тест пройден. Данные получены. Код ответа HTTP – 200 (рисунок Д.4).
5	Подбор персонального меню	GET	Дата=10.06.22	Тест пройден. Персональное меню подобрано. Код ответа HTTP – 200 (рисунок Д.5).
6	Получение меню пользователя	GET	Дата=10.06.22	Тест пройден. Меню получено. Код ответа HTTP – 200 (рисунок Д.6).

Продолжение таблицы 11

№	Описание теста	Запрос		Результат
		Метод	Параметры	
7	Получение списка всех продуктов	GET	-	Тест пройден. Список продуктов получен. Код ответа HTTP – 200 (рисунок Д.7).
8	Поиск списка продуктов по совпадению подстроки	GET	sustr=мол	Тест пройден. Список продуктов получен. Код ответа HTTP – 200 (рисунок Д.8).

Таким образом, функциональное тестирование показало корректность работы функций сервера.

4.1.2 Модульное тестирование

Модульные тесты – автоматизированные тесты, позволяющие оценить корректность работы некоторой автономной единицы программного кода путем сравнения фактического результата работы с ожидаемым. Основное преимущество модульных тестов состоит в том, что они позволяют быстро выявить и устранить ошибки, возникающие при модификации кода [24].

В рамках данного проекта было решено покрыть модульными тестами методы классов UserController и ProductController.

Класс UserController отвечает за работу с данными пользователя. Он содержит в себе логику авторизации и регистрации пользователей, реализованную с помощью методов Authorize и Register, для которых были составлены модульные тесты проверяющие корректность поведения методов при поступлении в качестве аргументов определенных данных.

Класс ProductController отвечает за работу с данными о продуктах питания. Он содержит в себе логику выдачи списка всех продуктов, реализованную с помощью метода GetAllProducts, и логику выдачи списка продуктов по определенному содержанию в названии продукта,

реализованную с помощью метода FindByNameProducts. Для этих методов были разработаны модульные тесты, проверяющие корректность их работы.

Скриншот выполнения модульных тестов представлен на рисунке 33.

Тестирование	Длительн...	Приз
TestProject (6)	1,1 с	
TestProject (6)	1,1 с	
UnitTests (6)	1,1 с	
Authorize_ReturnsJWT_WithCorrectData	10 мс	
Authorize_ReturnsNotFound_WithBadLogin	905 мс	
GetAllProducts_ReturnsOk	3 мс	
GetProductsByName_ReturnsOk	35 мс	
Register_ReturnsBadRequest_WithExistUser	185 мс	
Register_ReturnsOk_WithGoodData	1 мс	

Рисунок 33 – Результаты выполнения модульных тестов

Для проведения модульных тестов был разработан тестовый класс UnitTests, код которого представлен в приложении Е.

4.2 Тестирование клиентской части

Проведено функциональное тестирование для клиентской части. Результаты тестирования представлены в таблице 12.

Таблица 12 – Результаты тестирования клиентской части

Вариант использования	Тест	Параметры	Результат
Зарегистрироваться	Ввести данные для регистрации, попытка регистрации	логин=newuser1, пароль= pass1234, цель=Похудеть, пол=Мужской, рост=180, вес=80, день рождения= 01.01.2000	Тест пройден. Пользователь зарегистрирован в системе (рисунок Ж.1).
	Попытаться повторно использовать тот же логин для регистрации.	логин=newuser1, пароль= pass1234, цель=Похудеть, пол=Мужской, рост=180, вес=80, день рождения= 01.01.2000	Тест пройден. На экране сообщение о занятом логине. (рисунок Ж.2).

Продолжение таблицы 12

Вариант использования	Тест	Параметры	Результат
Авторизоваться	Попытаться ввести несуществующий логин	логин=notexist, пароль= pass1234	Тест пройден. На экране сообщение о несуществующем логине. (рисунок Ж.3).
	Попытаться ввести неверный пароль	логин=newuser1, пароль= 1q2w3e4r	Тест пройден. На экране сообщение о неправильном пароле. (рисунок Ж.4).
	Ввести верные данные для авторизации	логин=newuser1, пароль= pass1234	Тест пройден. Пользователь авторизован (рисунок Ж.5).
Просмотреть список блюд	Пользователь переходит на главную, затем нажимает кнопку +	-	Тест пройден. На экране отображается список блюд. (рисунок Ж.6)
Добавить блюдо в прием пищи	Пользователь пытается добавить блюдо с отрицательным весом	блюдо = Айсберг, вес = -1 г	Тест пройден. На экране сообщение о неправильном пароле. (рисунок Е.7).
	Пользователь пытается добавить блюдо	блюдо = Айсберг, вес = 100 г	Тест пройден. Блюдо добавлено (рисунок Ж.8).
Удалить блюдо из приема пищи	Пользователь пытается удалить блюдо из приема пищи	блюдо = Айсберг	Тест пройден. Блюдо удалено (рисунок Ж.9).
Подобрать персональное меню	Пользователь пытается подобрать персональное меню	Дата = 10.06.22	Тест пройден. Меню подобрано (рисунок Ж.10).
Посмотреть текущую калорийность, посмотреть рекомендованную калорийность	Пользователь переходит на страницу главная и просматривает текущую и рекомендованную калорийность	-	Тест пройден. Текущая и рекомендованная калорийности отображаются на экране (рисунок Ж.10).

Функциональное тестирование клиентской части показало полное соответствие поставленным функциональным требованиям. Все тесты пройдены без ошибок. Результаты тестирования клиентской части приложения представлены в приложении Ж.

4.3 Метрики кода

Метрики кода – это какие-либо количественные величины, которые позволяют оценить качество кода [25]. Рассмотрим код метода Register из класса UserController.cs. Листинг этого метода представлен в листинге 1.

Листинг 1 – Код метода Register

```
[HttpPost("register")]
public async Task<ActionResult> Register(RegUserDTO data)
{
    try
    {
        var foundUser = await m_Context.Users.FirstOrDefaultAsync(x
=>
            string.Equals(x.Login, data.Login));
        if (foundUser is not null)
        {
            return BadRequest("Пользователь с таким логином уже
существует!");
        }

        var user = new User {
            Login = data.Login,
            Password = data.Password,
            Purpose = data.Purpose,
            Gender = data.Gender,
            Height = data.Height,
            Weight = data.Weight,
            Birthday = data.Birthday
        };

        await m_Context.Users.AddAsync(user);
        await m_Context.SaveChangesAsync();
    }
    catch (Exception ex)
    {
        return BadRequest(ex.Message);
    }

    return Ok();
}
```

Приведенный метод взят из серверной части приложения. Он отвечает за обработку запроса на регистрацию пользователя. Если пользователь с

данным логином обнаружен в БД, то этот метод вернет ошибку, иначе пользователь успешно создается.

Вычислим для данного метода метрику Мак-Кейба и метрику Холстеда.

4.3.1 Метрика Мак-Кейба

Метрика Мак-Кейба или цикломатическая метрика, позволяет оценить структурную сложность программы на основе управляющего графа программы. Для этого создадим управляющий граф. Он представлен на рисунке 34.

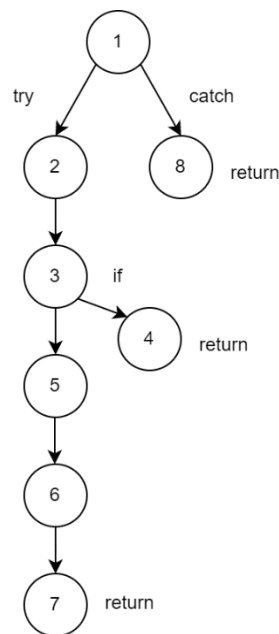


Рисунок 34 – Управляющий граф

Формулой для вычисления метрики Мак-Кейба является:

$$C = e - v + 2p,$$

где p – число компонент связности графа;

v – количество вершин;

e – количество дуг.

Для вычисления метрики необходимо подсчитать количество вершин и дуг данного графа. Количество вершин $v = 8$, количество дуг $e = 7$.

Число компонент связности графа – это количество таких подмножеств вершин графа, что каждые две вершины в этом подмножестве соединены хотя бы одним путем. В данном графе:

$$p = 1.$$

Получили следующее значение метрики Мак-Кейба:

$$C = e - v + 2p = 1.$$

Полученное значение меньше 10, значит анализируемый программный код обладает очень простой структурой потока управления и легок для восприятия.

4.3.2 Метрика Холстеда

Довольно важной метрикой кода является метрика Холстеда. Этот показатель составляется на основе синтаксической структуры кода. Он позволяет оценить сложность написания программы и ее потенциальный объем. Проанализируем рассматриваемый метод с использованием метрики Холстеда.

Первым шагом необходимо составить словарь операторов и операндов исследуемого кода. Словарь представлен в таблице 12.

Таблица 12 – Словарь операторов и операндов

Оператор	Число операторов	Операнд	Число операндов
var	1	ex	1
is	1	foundUser	2
await	3	x	2
new	1	m_Context	3
not	1	data	9
;	6	null	1
=>	1	BadRequest()	2
{	5	user	2
}	5	“”	1
try	1	Login	1
catch	1	Password	1
,	6	Purpose	1

Продолжение таблицы 12

Оператор	Число операторов	Операнд	Число операндов
.	12	Gender	1
(	9	Height	1
)	9	Weight	1
“	2	Birthday	1
return	1	Ok()	1
=	8		

По словарю мы вычисляем такие значения, как: Общее число операторов $N_1 = 73$, число уникальных операторов $n_1 = 18$, общее число операндов $N_2 = 31$, число уникальных операндов $n_2 = 17$.

Объем словаря программы может быть рассчитан, как:

$$n = n_1 + n_2 = 35.$$

Длина кода будет равна:

$$N = N_1 + N_2 = 104.$$

Теоретический объем программы (V) в условных единицах:

$$V = N \log_2 n \approx 533.$$

Мы можем оценить теоретическую длину программы:

$$C = n_1 \log_2(n_1) + n_2 \log_2(n_2) \approx 145.$$

Сложность программы рассчитывается как:

$$D = \frac{n_1}{2} \times \frac{N_2}{n_2} \approx 16,4.$$

Оценка интеллектуальных усилий будет равна:

$$E = D \times V \approx 8741,2.$$

Полученные характеристики свидетельствуют о том, что программный код имеет достаточно высокий уровень качества и удобства для дальнейшей поддержки.

Заключение

В ходе выполнения выпускной квалификационной работы было разработано мобильное приложение для планирования и контроля режима питания.

На этапе анализа требований и предметной области был выполнен анализ аналогичных приложений, в результате которого были выявлены их достоинства и недостатки, определены функциональные и нефункциональные требования к создаваемому программному средству и произведен анализ целевой аудитории.

Было осуществлено планирование разработки приложения и оценка бюджета. Результаты планирования показали, что длительность проекта составляет 58 дней, а предполагаемые затраты на реализацию – 206.900 рублей. Таким образом, это свидетельствует об экономической целесообразности разработки такого программного проекта.

На этапе проектирования была составлена логическая модель базы данных и разработана архитектура мобильного приложения. База данных была создана с использованием Entity Framework и развернута под управлением СУБД Microsoft SQL Server 2018. Серверная часть приложения была разработана на языке C# в среде программирования Microsoft Visual Studio 2022 preview. Клиентская часть была разработана на языке C# с использованием фреймворка MAUI и среды программирования Microsoft Visual Studio 2022 preview.

Было выполнено тестирование серверной и клиентской частей разрабатываемого приложения. Оно показало корректность его работы. Также рассчитаны метрики программного кода для оценки его качества.

В дальнейшем планируется опубликовать разработанное приложение в магазине мобильных приложений для системы Android Google Play и развернуть его серверную часть на облачном сервисе.

Таким образом, в ходе выполнения выпускной квалификационной работы были успешно выполнены все поставленные задачи.

Список использованных источников

1. Распоряжение Президента Российской Федерации от 9 мая 2017 г. № 203 «Стратегия развития информационного общества в Российской Федерации на 2017 – 2030 годы». [Электронный ресурс]. URL: http://zmedu.hostedu.ru/files/ykaz_7668.pdf (дата обращения: 14.04.2022).
2. Программа «Цифровая экономика Российской Федерации». Утверждена распоряжением Правительства Российской Федерации от 28 июля 2017 г. № 1632-р. [Электронный ресурс]. URL: <http://static.government.ru/media/files/9gFM4FHj4PsB79I5v7yLVuPgu4bvR7M0.pdf> (дата обращения: 14.04.2022).
3. Дарья Б. COVID-19 стал драйвером для «здоровых» приложений и гаджетов // Инновации [Электронный ресурс]. – URL: <https://plus.rbc.ru/news/6086cc8f7a8aa91c9dc7e8fe> (дата обращения: 16.04.2022)
4. Материалы с сайта «www.yazio.com» [Электронный ресурс]. – URL: <https://www.yazio.com/en> (дата обращения: 17.04.2022)
5. Материалы с сайта «hiki-soft.ru» [Электронный ресурс]. – URL: <https://hiki-soft.ru/> (дата обращения: 18.04.2022)
6. Обзор сервиса для контроля веса FatSecret [Электронный ресурс]. – URL: <https://lifel hacker.ru/fatsecret/> (дата обращения: 18.04.2022)
7. Мобильное здоровье 2021: отчет Sensor Tower [Электронный ресурс]. – URL: <https://apptractor.ru/measure/app-store-analytics/otchet-sostojanie-rynkaprilozhenij-dlja-fitnesa-i-zdorovja-2021.html> (дата обращения: 18.04.2022)
8. Статистика распространения разных версий Android [Электронный ресурс]. – URL: <https://3dnews.ru/1054313/desyatka-ostayotsya-samoy-populyarnoy-versiey-os-android> (дата обращения: 23.04.2022)
9. Прошкина Е. Н. Язык моделирования информационных процессов UML 2.0 (учебное пособие) / Прошкина Е. Н., Балашова И. Ю. – Пенза, Издательство ПГУ, 2018. – 112 с.

10. Арлоу, Д. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование, 2-е издание / Д. Арлоу, А. Нейштадт – СПб: Символ Плюс, 2007. – 624 с.
11. Вячеслав Черников Разработка мобильных приложений на C# для ios и Android. – Изд-во ДМК-пресс, 2020 – 188 с
12. Что такое .NET MAUI // Официальная документация Microsoft [Электронный ресурс]. – URL: <https://docs.microsoft.com/ru-ru/dotnet/maui/what-is-maui> (дата обращения: 20.04.2022)
13. Скороход, С.В. Управление проектами средствами Microsoft Project. Курс лекций. / С.В. Скороход. – М.: Интуит НОУ, 2016. – 319 стр.
14. Акатова, Н.А. Планирование и контроль выполнения проектов с использованием MS Project / Н.А. Акатова, М.Г. Коротких. – М.: МИСиС, 2019. – 74 с.
15. RUP (Rational Unified Process) – методология разработки ПО [Электронный ресурс]. – URL: <https://qaevolution.ru/metodologiya-menedzhment/rup/> (дата обращения: 21.04.2022)
16. Введение в Entity Framework [Электронный ресурс]. – URL: <https://metanit.com/sharp/entityframework/1.1.php> (дата обращения: 25.04.2022)
17. Entity Framework // Официальная документация Microsoft [Электронный ресурс]. – URL: <https://docs.microsoft.com/ru-ru/ef/> (дата обращения: 26.04.2022)
18. Архитектура «Клиент-Сервер» [Электронный ресурс]. – URL: <https://itelon.ru/blog/arkhitektura-klient-server/> (дата обращения: 2.05.2022)
19. Обзор ASP.NET // Официальная документация Microsoft [Электронный ресурс]. – URL: <https://docs.microsoft.com/ru-ru/aspnet/overview> (дата обращения: 10.05.2022)
20. Пример приложения Web API // Руководство по ASP.NET Core 6 [Электронный ресурс]. – URL: <https://metanit.com/sharp/aspnet6/11.1.php> (дата обращения: 16.05.2022)

21. MVC – модель-представление-контроллер // Методологии разработки [Электронный ресурс]. – URL: <https://web-creator.ru/articles/mvc> (дата обращения: 18.05.2022)
22. Шаблон model-view-ViewModel // Официальная документация Microsoft [Электронный ресурс]. – URL: <https://docs.microsoft.com/ru-ru/aspnet/overview> (дата обращения: 24.05.2022)
23. Привязка данных и MVVM // Официальная документация Microsoft [Электронный ресурс]. – URL: <https://docs.microsoft.com/ru-ru/aspnet/overview> (дата обращения: 26.05.2022)
24. Модульное тестирование: что это? Типы, инструменты [Электронный ресурс]. – URL: https://logrocon.ru/news/unit_testing (дата обращения: 1.06.2022)
25. Новичков А., Шамрай А., Черников А. Метрики кода и практическая реализация по их сбору и анализу [Электронный ресурс]. – URL: https://club.cnews.ru/blogs/entry/metriki_koda_i_prakticheskaya_realizatsiya_po_ih_sboru_i_analizu_chast_1_ (дата обращения: 3.06.2022)

Приложение А
ГЛОССАРИЙ
(справочное)

Android – мобильная операционная система для мобильных устройств, разрабатываемая и выпускаемая американской компанией Google и используемая на большинстве современных мобильных устройств.

СУБД (Система управления базой данных) – специализированное программное обеспечение, предоставляющее программный интерфейс, который дает возможность создавать, модифицировать и использовать базу данных.

API (Application Programming Interface) – набор классов, функций, процедур, констант и структур, предоставляемых приложением или операционной системой для использования во внешних программных продуктах.

RUP (Rational Unified Process) – методология разработки программного обеспечения, в основе которой лежит итеративная модель разработки.

DTO (Data Transfer Object) – лишенный функционала класс, предназначенный для хранения и передачи информации.

HTTP (Hyper Text Transfer Protocol) – протокол прикладного уровня, предназначенный для передачи текстовой информации, построенный на основе протока транспортного уровня TCP.

JSON – текстовый вид представления данных, преобразующийся в сущности любого языка или технологии, также легок для чтения человеком.

UML (Unified Modeling Language) – язык графического описания для объектного моделирования в области разработки программного обеспечения, моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

Приложение Б
ЛИСТИНГ КОДА СОЗДАНИЯ БАЗЫ ДАННЫХ
(рекомендуемое)

Листинг Б.1 – Исходный код миграции для создания базы данных

```

using System;
using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace WebNutritionControl.Migrations
{
    public partial class InitialCreate : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Products",
                columns: table => new
                {
                    Id = table.Column<int>(type: "int", nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    Name = table.Column<string>(type: "nvarchar(max)",
nullable: false),
                    Proteins = table.Column<double>(type: "float", nullable:
false),
                    Fats = table.Column<double>(type: "float", nullable:
false),
                    Carbohydrates = table.Column<double>(type: "float",
nullable: false),
                    Calories = table.Column<double>(type: "float", nullable:
false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Products", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "Users",
                columns: table => new
                {
                    Id = table.Column<int>(type: "int", nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    Login = table.Column<string>(type: "nvarchar(max)",
nullable: false),
                    Password = table.Column<string>(type: "nvarchar(max)",
nullable: false),
                    Gender = table.Column<string>(type: "nvarchar(max)",
nullable: false),
                    Height = table.Column<double>(type: "float", nullable:
false),
                    Weight = table.Column<double>(type: "float", nullable:
false),
                    Purpose = table.Column<string>(type: "nvarchar(max)",
nullable: false),
                    Birthday = table.Column<DateTime>(type: "datetime2",
nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Users", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "Menus",

```

```

        columns: table => new
        {
            Id = table.Column<int>(type: "int", nullable: false)
                .Annotation("SqlServer:Identity", "1, 1"),
            Name = table.Column<string>(type: "nvarchar(max)",
nullable: false),
            Date = table.Column<DateTime>(type: "datetime2",
nullable: false),
            UserId = table.Column<int>(type: "int", nullable: true)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_Menus", x => x.Id);
            table.ForeignKey(
                name: "FK_Menus_Users_UserId",
                column: x => x.UserId,
                principalTable: "Users",
                principalColumn: "Id");
        });

migrationBuilder.CreateTable(
    name: "Meals",
    columns: table => new
    {
        Id = table.Column<int>(type: "int", nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        Name = table.Column<string>(type: "nvarchar(max)",
nullable: false),
        MenuId = table.Column<int>(type: "int", nullable: true)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Meals", x => x.Id);
        table.ForeignKey(
            name: "FK_Meals_Menus_MenuId",
            column: x => x.MenuId,
            principalTable: "Menus",
            principalColumn: "Id");
    });

migrationBuilder.CreateTable(
    name: "ProductMeals",
    columns: table => new
    {
        Id = table.Column<int>(type: "int", nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        Weight = table.Column<int>(type: "int", nullable: false),
        MealId = table.Column<int>(type: "int", nullable: false),
        ProductId = table.Column<int>(type: "int", nullable:
false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_ProductMeals", x => x.Id);
        table.ForeignKey(
            name: "FK_ProductMeals_Meals_MealId",
            column: x => x.MealId,
            principalTable: "Meals",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
        table.ForeignKey(
            name: "FK_ProductMeals_Products_ProductId",
            column: x => x.ProductId,
            principalTable: "Products",

```

```

        principalColumn: "Id",
        onDelete: ReferentialAction.Cascade);
    });

    migrationBuilder.CreateIndex(
        name: "IX_Meals_MenuId",
        table: "Meals",
        column: "MenuId");

    migrationBuilder.CreateIndex(
        name: "IX_Menus_UserId",
        table: "Menus",
        column: "UserId");

    migrationBuilder.CreateIndex(
        name: "IX_ProductMeals_MealId",
        table: "ProductMeals",
        column: "MealId");

    migrationBuilder.CreateIndex(
        name: "IX_ProductMeals_ProductId",
        table: "ProductMeals",
        column: "ProductId");
}

protected override void Down(MigrationBuilder migrationBuilder)
{
    migrationBuilder.DropTable(
        name: "ProductMeals");

    migrationBuilder.DropTable(
        name: "Meals");

    migrationBuilder.DropTable(
        name: "Products");

    migrationBuilder.DropTable(
        name: "Menus");

    migrationBuilder.DropTable(
        name: "Users");
}
}
}

```

Приложение В
ИСХОДНЫЙ КОД СЕРВЕРНОЙ ЧАСТИ
(рекомендуемое)

Листинг В.1 – Исходный код класса MealController.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Security.Claims;
using WebNutritionControl.DTO;
using WebNutritionControl.Model;

namespace WebNutritionControl.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class MealController : ControllerBase
    {
        private readonly NutritionContext m_Context;

        public MealController(NutritionContext context)
        {
            m_Context = context;
        }

        [HttpPost("Create")]
        public async Task<ActionResult> Create (CreateMealDTO createMealDTO)
        {
            var userId =
                Int32.Parse(User.FindFirstValue(ClaimTypes.NameIdentifier));
            var userMenus = await m_Context.Menus.Where(x => (x.Date ==
                createMealDTO.Date && x.UserId == userId)).ToListAsync();
            try
            {
                var newMeal = new Meal()
                {
                    MenuId = userId,
                    Name = createMealDTO.Name
                };
                await m_Context.Meals.AddAsync(newMeal);
                await m_Context.SaveChangesAsync();
            }
            catch (Exception ex)
            {
                return BadRequest($"{ex.Message}");
            }

            return Ok();
        }
    }
}
```

Листинг В.2 – Исходный код класса MenuController.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Security.Claims;
using WebNutritionControl.DTO;
using WebNutritionControl.Model;

namespace WebNutritionControl.Controllers
```

```

{
    [ApiController]
    [Route("[controller]")]
    public class MenuController : ControllerBase
    {

        private readonly NutritionContext m_Context;

        public MenuController(NutritionContext context)
        {
            m_Context = context;
        }

        [HttpGet("Create")]
        [Authorize]
        public async Task<ActionResult> Create(string dateStr)
        {
            DateTime date = DateTime.Parse(dateStr);
            var userId =
Int32.Parse(User.FindFirstValue(ClaimTypes.NameIdentifier));
            var userMenu = await m_Context.Menus.FirstOrDefaultAsync(x =>
(x.Date.Equals(date) && x.UserId == userId));
            if (userMenu != null)
            {
                m_Context.Menus.Remove(userMenu);
                await m_Context.SaveChangesAsync();
            }
            try
            {
                var newMenu = new Menu()
                {
                    UserId = userId,
                    Date = date
                };
                var res = await m_Context.Menus.AddAsync(newMenu);
                await m_Context.SaveChangesAsync();

                var MealNames = new List<string>()
                {
                    "Завтрак",
                    "Обед",
                    "Ужин",
                    "Перекус (ы)"
                };

                foreach (var name in MealNames)
                {
                    var newMeal = new Meal()
                    {
                        MenuId = res.Entity.Id,
                        Name = name
                    };
                    await m_Context.Meals.AddAsync(newMeal);
                }
                await m_Context.SaveChangesAsync();
                var foundUser = await m_Context.Users.FirstOrDefaultAsync(x
=> x.Id.Equals(userId));
                var calories = UserController.GetCalories(foundUser);
                var productsWeights = await GetProductsWeights(calories);
                foreach (var productWeight in productsWeights)
                {
                    (var prodId, var mealName, var weight) = productWeight;
                    var foundMeal = await
m_Context.Meals.FirstOrDefaultAsync(x => x.MenuId.Equals(res.Entity.Id) &&

```

```

        x.Name.Equals(mealName));
var productMeal = new ProductMeal()
{
    MealId = foundMeal.Id,
    ProductId = prodId,
    Weight = (int)weight
};
await m_Context.ProductMeals.AddAsync(productMeal);
await m_Context.SaveChangesAsync();
    }

    }
catch (Exception ex)
{
    return BadRequest($"{ex.Message}");
}

return Ok();
}

private async Task<List<(int, string, double)>>
GetProductsWeights(double calories)
{
    List<(int, string, double)> productWeights = new List<(int,
string, double)>();
    Random random = new Random();
    var MealNames = new List<string>()
    {
        "Завтрак",
        "Обед",
        "Ужин"
    };

    foreach (var mealName in MealNames)
    {
        if (mealName == "Завтрак")
        {
            var needCal = calories * 0.3;
            var calProd1 = needCal * 0.6;
            var calProd2 = needCal * 0.4;
            int[] firstProd = { 227, 151, 258 };
            int[] secondProd = { 235, 236, 247 };

            var firstId = firstProd[random.Next(firstProd.Length)];
            var secondId =
secondProd[random.Next(secondProd.Length)];

            var foundProd1 = await
m_Context.Products.FirstOrDefaultAsync(x => x.Id.Equals(firstId));
            var foundProd2 = await
m_Context.Products.FirstOrDefaultAsync(x => x.Id.Equals(secondId));

            var weight1 = Math.Round((calProd1 / foundProd1.Calories
* 10)) * 10;
            var weight2 = Math.Round((calProd2 / foundProd2.Calories
* 10)) * 10;

            productWeights.Add((firstId, mealName, weight1));
            productWeights.Add((secondId, mealName, weight2));
        }
        else if (mealName == "Обед")
        {
            var needCal = calories * 0.4;
            var calProd1 = needCal * 0.15;

```

```

        var calProd2 = needCal * 0.5;
        var calProd3 = needCal * 0.35;

        int[] firstProd = { 22, 225 };
        int[] secondProd = { 265, 266, 114 };
        int[] thirdProd = { 355, 269, 42 };

        var firstId = firstProd[random.Next(firstProd.Length)];
        var secondId =
secondProd[random.Next(secondProd.Length)];
        var thirdId = thirdProd[random.Next(thirdProd.Length)];

        var foundProd1 = await
m_Context.Products.FirstOrDefaultAsync(x => x.Id.Equals(firstId));
        var foundProd2 = await
m_Context.Products.FirstOrDefaultAsync(x => x.Id.Equals(secondId));
        var foundProd3 = await
m_Context.Products.FirstOrDefaultAsync(x => x.Id.Equals(thirdId));

        var weight1 = Math.Round((calProd1 / foundProd1.Calories
* 10)) * 10;
        var weight2 = Math.Round((calProd2 / foundProd2.Calories
* 10)) * 10;
        var weight3 = Math.Round((calProd3 / foundProd3.Calories
* 10)) * 10;

        productWeights.Add((firstId, mealName, weight1));
        productWeights.Add((secondId, mealName, weight2));
        productWeights.Add((thirdId, mealName, weight3));
    }
    else if (mealName == "УЖИН")
    {
        var needCal = calories * 0.3;
        var calProd1 = needCal * 0.6;
        var calProd2 = needCal * 0.4;
        int[] firstProd = { 149, 153, 49 };
        int[] secondProd = { 328, 166, 163 };
        var firstId = firstProd[random.Next(firstProd.Length)];
        var secondId =
secondProd[random.Next(secondProd.Length)];

        var foundProd1 = await
m_Context.Products.FirstOrDefaultAsync(x => x.Id.Equals(firstId));
        var foundProd2 = await
m_Context.Products.FirstOrDefaultAsync(x => x.Id.Equals(secondId));

        var weight1 = Math.Round((calProd1 / foundProd1.Calories
* 10)) * 10;
        var weight2 = Math.Round((calProd2 / foundProd2.Calories
* 10)) * 10;

        productWeights.Add((firstId, mealName, weight1));
        productWeights.Add((secondId, mealName, weight2));
    }
}

return productWeights;
}

[HttpGet]
[Authorize]
public async Task<ActionResult<GetMenuDTO>> GetMenu(string dateStr)
{

```

```

        DateTime date = DateTime.Parse(dateStr);
        GetMenuDTO result = new GetMenuDTO();
        var userId =
Int32.Parse(User.FindFirstValue(ClaimTypes.NameIdentifier));
        var userMenu = await m_Context.Menus.FirstOrDefaultAsync(x =>
(x.Date.Equals(date) && x.UserId == userId));
        if (userMenu != null)
        {
            double userCalories = 0d;
            var meals = await m_Context.Meals.Where(x => (x.MenuId ==
userMenu.Id)).ToListAsync();
            foreach (var meal in meals)
            {
                var productMeals = await m_Context.ProductMeals.Where(x
=> x.MealId.Equals(meal.Id)).ToListAsync();

                List<ProductMealDTO> productMealsDTO = new ();
                foreach(var productMeal in productMeals)
                {
                    var product = await
m_Context.Products.FirstOrDefaultAsync(x =>
x.Id.Equals(productMeal.ProductId));
                    var calories = product.Calories * productMeal.Weight
* 0.01d;

                    userCalories += calories;
                    var productMealDTO = new ProductMealDTO()
                    {
                        Id = productMeal.Id,
                        Name = product.Name,
                        Weight = productMeal.Weight,
                        Calories = calories
                    };
                    productMealsDTO.Add(productMealDTO);
                }

                result.MealNamesProductMeals.Add(meal.Name,
productMealsDTO);
            }
            result.UserCalories = userCalories;
        }
        else
        {
            try
            {
                var newMenu = new Menu()
                {
                    UserId = userId,
                    Date = date
                };
                var res = await m_Context.Menus.AddAsync(newMenu);
                await m_Context.SaveChangesAsync();

                var MealNames = new List<string>
                {
                    "Завтрак",
                    "Обед",
                    "Ужин",
                    "Перекус(ы)"
                };

                foreach (var name in MealNames)
                {
                    result.MealNamesProductMeals.Add(name, new());
                    var newMeal = new Meal()

```

```

        {
            MenuId = res.Entity.Id,
            Name = name
        };
        await m_Context.Meals.AddAsync(newMeal);
    }
    await m_Context.SaveChangesAsync();

}
catch (Exception ex)
{
    return BadRequest($"{ex.Message}");
}
}
return Ok(result);
}
}
}

```

Листинг В.3 – Исходный код класса ProductController.cs

```

using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using WebNutritionControl.DTO;
using WebNutritionControl.Model;

namespace WebNutritionControl.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class ProductController : ControllerBase
    {
        private readonly NutritionContext m_Context;

        public ProductController(NutritionContext context)
        {
            m_Context = context;
        }

        [HttpGet("AllProducts")]
        //[Authorize]
        public async Task<ActionResult<List<ProductDTO>>> GetAllProducts()
        {
            try
            {
                var products = m_Context.Products;
                List<ProductDTO> newProducts = new List<ProductDTO>();
                foreach (var product in products)
                {
                    newProducts.Add(new ProductDTO()
                    {
                        Id = product.Id,
                        Name = product.Name,
                        Proteins = product.Proteins,
                        Fats = product.Fats,
                        Carbohydrates = product.Carbohydrates,
                        Calories = product.Calories
                    });
                }

                return Ok(newProducts);
            }
            catch { }
        }
    }
}

```

```

        }
        catch (Exception ex)
        {
            return BadRequest(ex.Message);
        }
    }

    [HttpGet("FindByName")]
    // [Authorize]
    public async Task<ActionResult<List<ProductDTO>>>
FindByNameProducts(string substr)
    {
        try
        {
            var products = m_Context.Products;
            List<ProductDTO> newProducts = new List<ProductDTO>();
            foreach (var product in products)
            {
                if (product.Name.ToLower().Contains(substr.ToLower()))
                {
                    newProducts.Add(new ProductDTO()
                    {
                        Id = product.Id,
                        Name = product.Name,
                        Proteins = product.Proteins,
                        Fats = product.Fats,
                        Carbohydrates = product.Carbohydrates,
                        Calories = product.Calories
                    });
                }
            }

            return Ok(newProducts);
        }
        catch (Exception ex)
        {
            return BadRequest(ex.Message);
        }
    }
}

```

Листинг В.4 – Исходный код класса ProductMealController.cs

```

using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Security.Claims;
using WebNutritionControl.DTO;
using WebNutritionControl.Model;

namespace WebNutritionControl.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class ProductMealController : ControllerBase
    {
        private readonly NutritionContext m_Context;
    }
}

```

```

public ProductMealController(NutritionContext context)
{
    m_Context = context;
}

[HttpPost("Create")]
[Authorize]
public async Task<ActionResult> Create(CreateProductMealDTO
createProductMealDTO)
{
    try
    {
        if (createProductMealDTO.Weight < 1) return BadRequest($"Вес
продукта должен быть больше 0");
        var userId =
Int32.Parse(User.FindFirstValue(ClaimTypes.NameIdentifier));
        var foundMenu = await m_Context.Menus.FirstOrDefaultAsync(x
=>
            string.Equals(x.UserId, userId) &&
x.Date.Equals(createProductMealDTO.Date.Date));
        if (foundMenu is null) return NotFound("Меню не
существует!");
        var foundMeal = await m_Context.Meals.FirstOrDefaultAsync(x
=>
            string.Equals(x.Name, createProductMealDTO.MealName) &&
x.MenuId.Equals(foundMenu.Id));
        if (foundMeal is null) return NotFound("Приема пищи не
существует!");

        var newProductMeal = new ProductMeal
        {
            ProductId = createProductMealDTO.ProductId,
            MealId = foundMeal.Id,
            Weight = createProductMealDTO.Weight
        };
        await m_Context.ProductMeals.AddAsync(newProductMeal);
        await m_Context.SaveChangesAsync();
    }
    catch (Exception ex)
    {
        return BadRequest($" {ex.Message}");
    }

    return Ok();
}

[HttpDelete]
[Authorize]
public async Task<ActionResult> Delete(int id)
{
    try
    {
        var found = await m_Context.ProductMeals.FirstAsync(x =>
x.Id.Equals(id));
        m_Context.ProductMeals.Remove(found);
        await m_Context.SaveChangesAsync();
    }
    catch (Exception ex)
    {
        return BadRequest($" {ex.Message}");
    }

    return Ok();
}

```

```

    }
}
}

```

Листинг В.5 – Исходный код класса UserController.cs

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text.Json;
using WebNutritionControl.Model;
using Microsoft.IdentityModel.Tokens;
using System.Globalization;
using WebNutritionControl.DTO;
using Microsoft.AspNetCore.Authorization;

namespace WebNutritionControl.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class UserController : ControllerBase
    {
        private readonly NutritionContext m_Context;

        public UserController (NutritionContext context)
        {
            m_Context = context;
        }

        [HttpPost("checkLogin")]
        public async Task<ActionResult> CheckLogin (string login)
        {
            try
            {
                var foundUser = await m_Context.Users.FirstOrDefaultAsync(x
=>
                string.Equals(x.Login, login));
                if(foundUser is not null)
                {
                    return BadRequest("Пользователь с таким логином уже
существует!");
                }
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }

            return Ok();
        }

        [HttpPost("register")]
        public async Task<ActionResult> Register(RegUserDTO data)
        {
            try
            {
                var foundUser = await m_Context.Users.FirstOrDefaultAsync(x
=>
                string.Equals(x.Login, data.Login));
                if (foundUser is not null)
                {

```

```

        return BadRequest("Пользователь с таким логином уже
существует!");
    }

    var user = new User {
        Login = data.Login,
        Password = data.Password,
        Purpose = data.Purpose,
        Gender = data.Gender,
        Height = data.Height,
        Weight = data.Weight,
        Birthday = data.Birthday
    };

    await m_Context.Users.AddAsync(user);
    await m_Context.SaveChangesAsync();
}
catch (Exception ex)
{
    return BadRequest(ex.Message);
}

return Ok();
}

[HttpPost("authorize")]
public async Task<ActionResult<string>> Authorize(AuthUserDTO data)
{
    const int NumberDayValidToken = 30;

    try
    {
        var foundUser = await m_Context.Users.FirstOrDefault(x
=>
        string.Equals(x.Login, data.Login));
        if (foundUser is null) return NotFound("Пользователя с таким
логином не существует!");
        if (foundUser.Password != data.Password) return
BadRequest("Неверный пароль!");
        var claims = new List<Claim>
        {
            new(ClaimTypes.NameIdentifier, $"{foundUser.Id}"),
            new(ClaimTypes.Name, foundUser.Login)
        };
        var jwt = new JwtSecurityToken(
            issuer: AuthOptions.ISSUER,
            audience: AuthOptions.AUDIENCE,
            claims: claims,
            expires:
DateTime.UtcNow.Add(TimeSpan.FromDays(NumberDayValidToken)),
            signingCredentials: new
SigningCredentials(AuthOptions.GetSymmetricSecurityKey(),
SecurityAlgorithms.HmacSha256)
        );

        string? encodedJwt = new
JwtSecurityTokenHandler().WriteToken(jwt);
        return Ok(encodedJwt);
    }
    catch (Exception ex)
    {
        return BadRequest(ex.Message);
    }
}

```

```

    }

    [HttpPut]
    [Authorize]
    public async Task<ActionResult> Update(UpdateUserDto upUserDTO)
    {
        try
        {
            var userId =
Int32.Parse(User.FindFirstValue(ClaimTypes.NameIdentifier));
            var foundUser = await m_Context.Users.FirstOrDefaultAsync(x
=> string.Equals(x.Id, userId));
            foundUser.Purpose = upUserDTO.Purpose;
            foundUser.Birthday = upUserDTO.Birthday;
            foundUser.Gender = upUserDTO.Gender;
            foundUser.Height = upUserDTO.Height;
            foundUser.Weight = upUserDTO.Weight;
            await m_Context.SaveChangesAsync();
            return Ok();
        }
        catch (Exception ex)
        {
            return BadRequest(ex.Message);
        }
    }

    [HttpGet]
    [Authorize]
    public async Task<ActionResult<GetUserDto>> GetUserInfo()
    {
        try
        {
            var userId =
Int32.Parse(User.FindFirstValue(ClaimTypes.NameIdentifier));
            var foundUser = await m_Context.Users.FirstOrDefaultAsync(x
=>
                string.Equals(x.Id, userId));
            if (foundUser is null) return NotFound("Пользователя не
существует!");
            var age = GetAge(foundUser.Birthday);
            var calories = GetCalories(foundUser);
            var imt = (foundUser.Weight / (foundUser.Height *
foundUser.Height)) * 10000;
            string str;
            if (imt < 18.5)
            {
                str = "У вас недостаток веса, вам рекомендовано выбрать
цель - Набрать массу.";
            }
            else if (imt < 25)
            {
                str = "У вас нормальный вес, вам рекомендовано выбрать
цель - Поддерживать вес.";
            }
            else
            {
                str = "У вас избыток веса, вам рекомендовано выбрать цель
- Похудеть.";
            }
            GetUserDto getUserDto = new()
            {
                Gender = foundUser.Gender,

```

```

        Height = foundUser.Height,
        Purpose = foundUser.Purpose,
        Weigth = foundUser.Weight,
        Birthday = foundUser.Birthday,
        Calories = calories,
        Age = age,
        Imt = imt,
        Message = str +
        $"\\nДля достижения Вашей цели ({foundUser.Purpose}) Вам
необходимо " +
        $"потреблять {calories} ккал в сутки"

    };
    return Ok(getUserDto);
}
catch (Exception ex)
{
    return BadRequest(ex.Message);
}
}

public static int GetAge(DateTime birthDate)
{
    var now = DateTime.Today;
    return now.Year - birthDate.Year - 1 +
        ((now.Month > birthDate.Month || now.Month == birthDate.Month
        && now.Day >= birthDate.Day) ? 1 : 0);
}

public static double GetCalories (User foundUser)
{
    int kf1;
    double kf2;
    if (foundUser.Gender == "Мужской")
    {
        kf1 = 5;
    }
    else
    {
        kf1 = -161;
    }

    if (foundUser.Purpose == "Похудеть")
    {
        kf2 = 0.75d;
    }
    else if (foundUser.Purpose == "Набрать массу")
    {
        kf2 = 1.25d;
    }
    else kf2 = 1;
    var age = GetAge(foundUser.Birthday);
    return ((10 * foundUser.Weight) + (6.25 * foundUser.Height) - (5
    * age) + kf1) * kf2 * 1.2;
}
}
}

```

Листинг В.6 – Исходный код класса Meal.cs

```
namespace WebNutritionControl.Model
{
    public class Meal
    {
        public int Id { get; set; }
        public int MenuId { get; set; }
        public string Name { get; set; }
        public IList<ProductMeal> ProductMeal { get; set; } = new
List<ProductMeal>();
    }
}
```

Листинг В.7 – Исходный код класса Menu.cs

```
namespace WebNutritionControl.Model
{
    public class Menu
    {
        public int Id { get; set; }
        public int UserId { get; set; }
        public DateTime Date { get; set; }
        public IList<Meal> Meals { get; set; } = new List<Meal>();
    }
}
```

Листинг В.8 – Исходный код класса NutritionContext.cs

```
using Microsoft.EntityFrameworkCore;

namespace WebNutritionControl.Model
{
    public class NutritionContext : DbContext
    {
        public DbSet<User> Users { get; set; }
        public DbSet<Menu> Menus { get; set; }
        public DbSet<Meal> Meals { get; set; }
        public DbSet<ProductMeal> ProductMeals { get; set; }
        public DbSet<Product> Products { get; set; }

        public NutritionContext (DbContextOptions<NutritionContext> options)
        : base(options)
        {
        }
    }
}
```

Листинг В.9 – Исходный код класса Product.cs

```
namespace WebNutritionControl.Model
{
    public class Product
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public double Proteins { get; set; }
        public double Fats { get; set; }
        public double Carbohydrates { get; set; }
    }
}
```

```

        public double Calories { get; set; }
    }
}

```

Листинг В.10 – Исходный код класса ProductMeal.cs

```

namespace WebNutritionControl.Model
{
    public class ProductMeal
    {
        public int Id { get; set; }
        public int Weight { get; set; }
        public int MealId { get; set; }
        public Meal Meal { get; set; }
        public int ProductId { get; set; }
        public Product Product { get; set; }
    }
}

```

Листинг В.11 – Исходный код класса User.cs

```

namespace WebNutritionControl.Model
{
    public class User
    {
        public int Id { get; set; }
        public string Login { get; set; }
        public string Password { get; set; }
        public string Gender { get; set; }
        public double Height { get; set; }
        public double Weight { get; set; }
        public string Purpose { get; set; }
        public DateTime Birthday { get; set; }
        public IList<Menu> Menus { get; set; } = new List<Menu>();
    }
}

```

Приложение Г
ИСХОДНЫЙ КОД КЛИЕНТСКОЙ ЧАСТИ
(рекомендуемое)

Листинг Г.1 – Исходный код класса AuthService.cs

```

using NutritionControlMAUI;
using NutritionControlMAUI.DTO;
using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Net.Http.Json;
using System.Text;
using System.Text.Json;
using System.Threading;
using System.Threading.Tasks;

namespace NutritionControlDTO.Services
{
    internal class AuthService : HttpService
    {
        private const string c_RegisterPath = "/User/register";
        private const string c_AuthorizePath = "/User/authorize";

        public async Task RegisterAsync(RegUserDTO regUserDTO)
        {
            var response = await Client.PostAsJsonAsync(new Uri(BaseUrl,
c_RegisterPath), regUserDTO);

            if (!response.IsSuccessStatusCode) await
HttpErrorsHandler(response);
        }

        public async Task AuthorizeAsync(string login, string password)
        {
            var data = new AuthUserDTO { Login = login, Password = password
};

            var response = await Client.PostAsJsonAsync(new Uri(BaseUrl,
c_AuthorizePath), data);

            if (!response.IsSuccessStatusCode) await
HttpErrorsHandler(response);

            string token = await response.Content.ReadAsStringAsync();

            App.SetToken(token);
        }
    }
}

```

Листинг Г.2 – Исходный код класса HttpService.cs

```

using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;

namespace NutritionControlDTO.Services
{
    internal abstract class HttpService
    {

```

```

        protected static readonly Uri HttpUrl = new
Uri("http://192.168.1.231:5059");
        protected readonly HttpClientHandler ClientHandler;
        protected readonly HttpClient Client;

        public Uri BaseUrl => HttpUrl;

        protected HttpService()
        {
            ClientHandler = new HttpClientHandler();
            Client = new HttpClient(ClientHandler);
            Client.Timeout = TimeSpan.FromSeconds(20);
        }

        protected void AddTokenToHeaders()
        {
            const string auth = "Authorization";
            string value = $"Bearer {Preferences.Get("token",
string.Empty)}";

            Client.DefaultRequestHeaders.Remove(auth);
            Client.DefaultRequestHeaders.Add(auth, value);
        }

        protected async Task HttpErrorHandler(HttpResponseMessage message)
        {
            try
            {
                var errString = await message.Content.ReadAsStringAsync();
                throw new HttpRequestException(errString);
            }
            catch (HttpRequestException)
            {
                throw;
            }
            catch (Exception ex)
            {
                throw new HttpRequestException("Ошибка! ", ex);
            }
        }
    }
}

```

Листинг Г.3 – Исходный код класса MenuService.cs

```

using NutritionControlDTO.Services;
using NutritionControlMAUI.DTO;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http.Json;
using System.Text;
using System.Threading.Tasks;

namespace NutritionControlMAUI.Services
{
    internal class MenuService : HttpService

```

```

{
    private const string c_GetMenuPath = "/Menu?dateStr=";
    private const string c_CreateMenuPath = "/Menu/Create?dateStr=";

    public async Task<GetMenuDTO> GetAsync(DateTime day)
    {
        AddTokenToHeaders();
        var response = await Client.GetAsync(new Uri(BaseUrl,
c_GetMenuPath + day.ToShortDateString().Replace(".", "%2F")));

        if (!response.IsSuccessStatusCode) await
HttpErrorsHandler(response);

        return await response.Content.ReadFromJsonAsync<GetMenuDTO>();
    }

    public async Task CreateAsync(DateTime day)
    {
        AddTokenToHeaders();
        var response = await Client.GetAsync(new Uri(BaseUrl,
c_CreateMenuPath + day.ToShortDateString().Replace(".", "%2F")));

        if (!response.IsSuccessStatusCode) await
HttpErrorsHandler(response);
    }
}
}

```

Листинг Г.4 – Исходный код класса ProductMealService.cs

```

using NutritionControlDTO.Services;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http.Json;
using System.Text;
using System.Threading.Tasks;
using NutritionControlMAUI.DTO;

namespace NutritionControlMAUI.Services
{
    internal class ProductMealService : HttpService
    {
        private const string c_CreatePath = "/ProductMeal/Create";

        private const string c_DeletePath = "/ProductMeal?id=";

        public async Task CreateAsync(CreateProductMealDTO
createProductMealDTO)
        {
            AddTokenToHeaders();
            var response = await Client.PostAsJsonAsync(new Uri(BaseUrl,
c_CreatePath), createProductMealDTO);

            if (!response.IsSuccessStatusCode) await
HttpErrorsHandler(response);
        }

        public async Task DelAsync(int id)
        {

```

```

        AddTokenToHeaders();
        var response = await Client.DeleteAsync(new Uri(BaseUrl,
c_DeletePath + id.ToString()));

        if (!response.IsSuccessStatusCode) await
HttpErrorsHandler(response);

        var r = response.Content.ReadAsStringAsync();
    }
}
}

```

Листинг Г.5 – Исходный код класса ProductService.cs

```

using NutritionControlDTO.Services;
using NutritionControlMAUI.DTO;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http.Json;
using System.Text;
using System.Text.Json;
using System.Threading.Tasks;

namespace NutritionControl.Services
{
    internal class ProductService : HttpService
    {
        private const string c_AllProducts = "/Product/AllProducts";
        private const string c_FindByNameProducts =
"/Product/FindByName?substr=";

        public async Task<List<ProductDTO>> GetAllAsync()
        {
            var response = await Client.GetAsync(new Uri(BaseUrl,
c_AllProducts));

            if (!response.IsSuccessStatusCode) await
HttpErrorsHandler(response);

            var result = await
response.Content.ReadFromJsonAsync<List<ProductDTO>>();

            return result ?? new();
        }

        public async Task<List<ProductDTO>> GetByNameAsync(string substr)
        {
            var response = await Client.GetAsync(new Uri(BaseUrl,
c_FindByNameProducts + substr));

            if (!response.IsSuccessStatusCode) await
HttpErrorsHandler(response);

            var result = await
response.Content.ReadFromJsonAsync<List<ProductDTO>>();

            return result ?? new();
        }
    }
}

```

Листинг Г.6 – Исходный код класса UserService.cs

```
using NutritionControlDTO.Services;
using NutritionControlMAUI.DTO;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http.Json;
using System.Text;
using System.Threading.Tasks;

namespace NutritionControlMAUI.Services
{
    internal class UserService : HttpService
    {
        private const string c_GetUserPath = "/User";

        public async Task<GetUserDto> GetAsync()
        {
            AddTokenToHeaders();
            var response = await Client.GetAsync(new Uri(BaseUrl,
c_GetUserPath));

            if (!response.IsSuccessStatusCode) await
HttpErrorsHandler(response);
            var res = await response.Content.ReadFromJsonAsync<GetUserDto>();

            return res;
        }

        public async Task UpdateAsync(UpdateUserDto upUserDTO)
        {
            AddTokenToHeaders();
            var response = await Client.PutAsJsonAsync(new Uri(BaseUrl,
c_GetUserPath), upUserDTO);

            if (!response.IsSuccessStatusCode) await
HttpErrorsHandler(response);
        }
    }
}
```

Листинг Г.7 – Исходный код класса AuthorizationViewModel.cs

```
using NutritionControlDTO.Services;
using NutritionControlMAUI.Views;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NutritionControlMAUI.ViewModels
{
    internal class AuthorizationViewModel : BaseViewModel
    {
        public string Login { get; set; }
        public string Password { get; set; }

        private readonly AuthService authService;

        public Command AuthCommand { get; }
    }
}
```

```

public AuthorizationViewModel()
{
    AuthCommand = new Command(OnAuthClicked);
    authService = new AuthService();
}

private async void OnAuthClicked()
{
    try
    {
        await authService.AuthorizeAsync(Login, Password);
        await Shell.Current.GoToAsync("//" + nameof(ProfilePage));
    }
    catch (Exception ex)
    {
        await Shell.Current.DisplayAlert("Ошибка", ex.Message, "OK");
    }
}
}
}

```

Листинг Г.8 – Исходный код класса BaseViewModel.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Text;
using System.Threading.Tasks;

namespace NutritionControlMAUI.ViewModels
{
    internal class BaseViewModel : INotifyPropertyChanged
    {
        private bool m_IsBusy = false;
        private string m_Title = string.Empty;

        protected readonly object m_Lock = new object();

        public bool IsBusy
        {
            get => m_IsBusy;
            set => SetProperty(ref m_IsBusy, value);
        }

        public string Title
        {
            get => m_Title;
            set => SetProperty(ref m_Title, value);
        }

        public event PropertyChangedEventHandler PropertyChanged;

        protected bool SetProperty<T>(
            ref T field, T value,
            [CallerMemberName] string propertyName = "",
            Action raiseChanged = null)

```

```

        {
            if (EqualityComparer<T>.Default.Equals(field, value))
                return false;

            field = value;
            raiseChanged?.Invoke();
            RaisePropertyChanged(propertyName);

            return true;
        }

        protected void RaisePropertyChanged([CallerMemberName] string
propertyName = "")
        {
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
        }
    }
}

```

Листинг Г.9 – Исходный код класса MainViewModel.cs

```

using NutritionControl.Services;
using NutritionControlMAUI.DTO;
using NutritionControlMAUI.Services;
using NutritionControlMAUI.Views;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NutritionControlMAUI.ViewModels
{
    record Meal (string Name, List<ProductMealDTO> ProductMeals);
    internal class MainViewModel : BaseViewModel
    {
        public GetMenuDTO GetMenuDTO { get; set; }
        public GetUserDto User { get; set; }

        public ObservableCollection<Meal> Meals { get; } = new
ObservableCollection<Meal>();

        private DateTime _day = DateTime.Now.Date;
        public DateTime Day
        {
            get => _day;
            set
            {
                _day = value;
                GetMenu();
            }
        }

        private readonly MenuService menuService;
        private readonly ProductMealService productMealService;
        private readonly UserService userService;

        public Command ReloadCommand { get; }
        public Command AddClickedCommand { get; }
        public Command DelClickedCommand { get; }
    }
}

```

```

public MainViewModel()
{
    menuService = new();
    ReloadCommand = new Command(Reload);
    productMealService = new ProductMealService();
    userService = new();
    AddClickedCommand = new Command<string>(AddClicked);
    DelClickedCommand = new Command<int>(DelClicked);
}

private async void DelClicked(int id)
{
    try
    {
        await productMealService.DelAsync(id);
        GetMenu();
    }
    catch (Exception ex)
    {
        await Shell.Current.DisplayAlert("Ошибка", ex.Message, "OK");
    }
}

private async void AddClicked(string mealName)
{
    await Shell.Current.GoToAsync(nameof(ProductsPage), new
Dictionary<string, object>
    {
        ["MealName"] = mealName,
        ["Day"] = Day
    });
}

private void Reload()
{
    Day = DateTime.Now.Date;
    RaisePropertyChanged(nameof(Day));
    GetMenu();
}

private async void GetMenu()
{
    try
    {
        User = await userService.GetAsync();
        RaisePropertyChanged(nameof(User));
        GetMenuDTO = await menuService.GetAsync(Day);
        RaisePropertyChanged(nameof(GetMenuDTO));
        var list = GetMenuDTO.MealNamesProductMeals.Select(x => new
Meal(x.Key, x.Value)).ToList();
        Meals.Clear();
        foreach(var item in list)
        {
            Meals.Add(item);
        }
    }
    catch (Exception ex)
    {
        await Shell.Current.DisplayAlert("Ошибка", ex.Message, "OK");
    }
}

```

```
    }
}
```

Листинг Г.10 – Исходный код класса ProductsViewModel.cs

```
using NutritionControl.Services;
using NutritionControlMAUI.DTO;
using NutritionControlMAUI.Services;
using NutritionControlMAUI.Views;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;

namespace NutritionControlMAUI.ViewModels
{
    internal class ProductsViewModel : BaseViewModel, IQueryAttributable
    {
        public ObservableCollection<ProductDTO> Products { get; set; }

        public ProductDTO Product { get; set; }

        public CreateProductMealDTO CreateProductMealDTO { get; set; } =
new();

        private readonly ProductService productService;
        private readonly ProductMealService productMealService;

        public Command TapProductCommand { get; }
        public Command AddCommand { get; }

        public ICommand PerformSearch => new Command<string>(async (string
query) =>
        {
            try
            {
                var products = await productService.GetByNameAsync(query);
                Products = new(products);
                RaisePropertyChanged(nameof(Products));
            }
            catch (Exception ex)
            {
                await Shell.Current.DisplayAlert("Ошибка", ex.Message, "OK");
            }
        });

        public ProductsViewModel()
        {
            Products = new ObservableCollection<ProductDTO>();
            productService = new ProductService();
            productMealService = new ProductMealService();
            TapProductCommand = new Command<ProductDTO>(OnTapProduct);
            AddCommand = new Command(OnClickAdd);
            CreateProductMealDTO.Weight = 100;
            GetAllProducts();
        }

        public void OnTapProduct(ProductDTO product)
```

```

    {
        Product = product;
        RaisePropertyChanged(nameof(Product));
    }

    public async void OnClickAdd()
    {
        if (Product == null)
        {
            await Shell.Current.DisplayAlert("Ошибка", "Вы не выбрали
продукт", "OK");
        }
        else
        {
            try
            {
                CreateProductMealDTO.ProductId = Product.Id;
                await
productMealService.CreateAsync(CreateProductMealDTO);
                await Shell.Current.GoToAsync("//" + nameof(MainPage));
            }
            catch (Exception ex)
            {
                await Shell.Current.DisplayAlert("Ошибка", ex.Message,
"OK");
            }
        }
    }

    public async void GetAllProducts()
    {
        try
        {
            var products = await productService.GetAllAsync();
            Products = new(products);
            RaisePropertyChanged(nameof(Products));
        }
        catch (Exception ex)
        {
            await Shell.Current.DisplayAlert("Ошибка", ex.Message, "OK");
        }
    }

    public void ApplyQueryAttributes(IDictionary<string, object> query)
    {
        CreateProductMealDTO.MealName = (string)query["MealName"];
        CreateProductMealDTO.Date = ((DateTime)query["Day"]).Date;
    }
}

```

Листинг Г.11 – Исходный код класса ProfileViewModel.cs

```

using NutritionControlMAUI.DTO;
using NutritionControlMAUI.Services;
using NutritionControlMAUI.Views;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;

```

```

namespace NutritionControlMAUI.ViewModels
{
    internal class ProfileViewModel : BaseViewModel
    {
        public GetUserDto User { get; set; }
        public DateTime Date { get; set; }

        private readonly UserService userService;
        private readonly MenuService menuService;

        public ICommand ChangeClick { get; }
        public ICommand ExitClick { get; }
        public ICommand ReloadCommand { get; }

        public Command MenuClick { get; }

        public ProfileViewModel()
        {
            userService = new();
            menuService = new();
            ChangeClick = new Command(OnChangeClick);
            ExitClick = new Command(OnExitClick);
            ReloadCommand = new Command(Reload);
            MenuClick = new Command(OnMenuClick);
        }

        private async void OnMenuClick()
        {
            try
            {
                bool answer = await Shell.Current.DisplayAlert("Вы уверены  
что хотите подобрать меню?", "" +
                    "В случае подбора меню, Ваше текущее меню удалится, новое  
меню появится на его месте. ", "Да", "Нет");
                if (answer)
                {
                    await menuService.CreateAsync(Date);
                    await Shell.Current.GoToAsync("//" + nameof(MainPage));
                }
            }
            catch (Exception ex)
            {
                await Shell.Current.DisplayAlert("Ошибка", ex.Message, "OK");
            }
        }

        private void Reload()
        {
            GetUser();
        }

        private async void OnChangeClick()
        {
            await Shell.Current.GoToAsync(nameof(UserInfoPage), new
            Dictionary<string, object>
            {
                ["IsChange"] = true
            });
        }

        private async void OnExitClick()
        {

```

```

        bool answer = await Shell.Current.DisplayAlert("Вы уверены что  
хотите выйти?", "", "Да", "Нет");
        if (answer)
        {
            Preferences.Clear();
            await Shell.Current.GoToAsync("//" + nameof(StartPage));
        }

        public async void GetUser()
        {
            try
            {
                Date = DateTime.Now.Date;
                RaisePropertyChanged(nameof(Date));
                User = await userService.GetAsync();
                RaisePropertyChanged(nameof(User));
            }
            catch (Exception ex)
            {
                await Shell.Current.DisplayAlert("Ошибка", ex.Message, "OK");
            }
        }
    }
}

```

Листинг Г.12 – Исходный код класса RegistrationViewModel.cs

```

using NutritionControlDTO.Services;
using NutritionControlMAUI.Views;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NutritionControlMAUI.ViewModels
{
    internal class RegistrationViewModel : BaseViewModel
    {
        public string Login { get; set; }
        public string Password { get; set; }
        private readonly AuthService authService;

        public Command RegCommand { get; }
        public RegistrationViewModel()
        {
            authService = new AuthService();
            RegCommand = new Command(OnRegClicked);
        }

        private async void OnRegClicked()
        {
            try
            {
                await Shell.Current.GoToAsync(nameof(UserInfoPage), new
Dictionary<string, object>
                {
                    ["Login"] = Login,

```

```

        ["Password"] = Password,
    });
}
catch (Exception ex)
{
    await Shell.Current.DisplayAlert("Ошибка", ex.Message, "OK");
}
}
}
}

```

Листинг Г.13 – Исходный код класса StartViewModel.cs

```

using NutritionControlMAUI.Views;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NutritionControlMAUI.ViewModels
{
    internal class StartViewModel : BaseViewModel
    {
        public Command LoginCommand { get; }
        public Command RegistrCommand { get; }

        public StartViewModel()
        {
            LoginCommand = new Command(OnLoginClicked);
            RegistrCommand = new Command(OnRegClicked);
        }
        private async void OnLoginClicked(object obj)
        {
            await Shell.Current.GoToAsync(nameof(AuthorizationPage));
        }
        private async void OnRegClicked()
        {
            await Shell.Current.GoToAsync(nameof(RegistrationPage));
        }
    }
}

```

Листинг Г.14 – Исходный код класса UserInfoViewModel.cs

```

using NutritionControlDTO.Services;
using NutritionControlMAUI.DTO;
using NutritionControlMAUI.Services;
using NutritionControlMAUI.Views;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NutritionControlMAUI.ViewModels
{
    internal class UserInfoViewModel : BaseViewModel, IQueryAttributable
    {
        private static readonly List<string> s_Genders;
    }
}

```

```

private static readonly List<string> s_Purposes;

public List<string> Genders => s_Genders;
public List<string> Purposes => s_Purposes;

public bool IsChange { get; set; } = false;
public RegUserDTO RegUserDTO { get; set; } = new();

public Command SaveClick { get; }
public Command ReloadCommand { get; }

private readonly AuthService authService;
private readonly UserService userService;

public UserInfoViewModel()
{
    authService = new AuthService();
    userService = new UserService();
    SaveClick = new Command(OnSaveClicked);
}

static UserInfoViewModel()
{
    s_Genders = new List<string>
    {
        "Мужской",
        "Женский"
    };
    s_Purposes = new List<string>
    {
        "Похудеть",
        "Поддерживать вес",
        "Набрать массу"
    };
}

public async void InitPage()
{
    if (IsChange)
    {
        GetUserDto getUserDTO = await userService.GetAsync();
        RegUserDTO.Height = getUserDTO.Height;
        RegUserDTO.Weigth = getUserDTO.Weigth;
        RegUserDTO.Gender = getUserDTO.Gender;
        RegUserDTO.Purpose = getUserDTO.Purpose;
        RegUserDTO.Birthday = getUserDTO.Birthday;
        RaisePropertyChanged(nameof(RegUserDTO));
    }
    else
    {
        RegUserDTO.Height = 180;
        RegUserDTO.Weigth = 70;
        RegUserDTO.Gender = "Мужской";
        RegUserDTO.Purpose = "Поддерживать вес";
        RegUserDTO.Birthday = DateTime.Parse("01/01/2000");
    }
}

public void ApplyQueryAttributes(IDictionary<string, object> query)
{
    IsChange = query.TryGetValue("IsChange", out _);
    if (!IsChange)
    {

```

```

        RegUserDTO.Login = (string)query["Login"];
        RegUserDTO.Password = (string)query["Password"];
    }
    InitPage();
}

public async void OnSaveClicked()
{
    try
    {
        if (IsChange)
        {
            UpdateUserDto updateUserDto = new UpdateUserDto()
            {
                Birthday = RegUserDTO.Birthday,
                Gender = RegUserDTO.Gender,
                Height = RegUserDTO.Height,
                Purpose = RegUserDTO.Purpose,
                Weigth = RegUserDTO.Weigth
            };
            await userService.UpdateAsync(updateUserDto);
            await Shell.Current.DisplayAlert("Успешно", "Новые данные
применены ", "OK");
            await Shell.Current.GoToAsync("//" +
nameof(ProfilePage));
        }
        else
        {
            await authService.RegisterAsync(RegUserDTO);
            await Shell.Current.DisplayAlert("Успешная регистрация",
"Поздравляем! Вы зарегистрировались, " +
"теперь вы перейдете на страницу для авторизации.",
"OK");
            await Shell.Current.GoToAsync(nameof(AuthorizationPage));
        }
    }
    catch (Exception ex)
    {
        await Shell.Current.DisplayAlert("Ошибка", ex.Message, "OK");
    }
}
}

```

Листинг Г.15 – Исходный код страницы MainPage.xaml

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:viewmodels="clr-namespace:NutritionControlMAUI.ViewModels"
    xmlns:dto="clr-namespace:NutritionControlMAUI.DTO"
    xmlns:behaviors="clr-namespace:NutritionControlMAUI"
    x:DataType="viewmodels:MainViewModel"
    x:Class="NutritionControlMAUI.Views.MainPage">
    <ContentPage.Behaviors>
        <behaviors:ContentPageAppendingCommandBehavior
AppearingCommand="{Binding ReloadCommand}" />
    </ContentPage.Behaviors>
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="auto"/>
            <RowDefinition Height="auto"/>
            <RowDefinition/>

```

```

        </Grid.RowDefinitions>
        <Label>
            <Label.Text>
                <MultiBinding StringFormat="{0}Потреблено: {0:F1} ккал из
{1:F1} ккал">
                    <Binding Path="GetMenuDTO.UserCalories" />
                    <Binding Path="User.Calories"/>
                </MultiBinding>
            </Label.Text>
        </Label>
        <DatePicker Date="{Binding Day}" BackgroundColor="WhiteSmoke"
Grid.Row="1" Margin="5,5,5,5"/>
        <Grid Grid.Row="2" >
            <CollectionView ItemsSource="{Binding Meals}" >
                <CollectionView.ItemTemplate>
                    <DataTemplate x:DataType="viewmodels:Meal">
                        <Border StrokeShape="Rectangle" Margin="5,5,5,5"
BackgroundColor="WhiteSmoke">
                            <Grid>
                                <Grid.ColumnDefinitions>
                                    <ColumnDefinition Width="3*" />
                                    <ColumnDefinition Width="1*" />
                                </Grid.ColumnDefinitions>
                                <Grid.RowDefinitions>
                                    <RowDefinition Height="auto"/>
                                    <RowDefinition Height="auto"/>
                                </Grid.RowDefinitions>
                                <Label Grid.Column="0" Text="{Binding Name}"
FontSize="Medium"
                                    LineBreakMode="CharacterWrap"
HeightRequest="30"
                                    Margin="10, 10, 0, 0"/>
                                <Button Grid.Column="1" Text="+"
FontSize="Large"
                                    HeightRequest="30" Padding="0"
Command="{Binding AddClickedCommand, Source={RelativeSource
AncestorType={x:Type viewmodels:MainViewModel}}}"
                                    Margin="5, 20, 5, 0"
CommandParameter="{Binding Name}"/>
                            <Grid Grid.Row="1" Grid.ColumnSpan="2" >
                                <CollectionView ItemsSource="{Binding
ProductMeals}">
                                    <CollectionView.ItemTemplate>
                                        <DataTemplate
x:DataType="dto:ProductMealDTO">
                                            <Border
StrokeShape="Rectangle" Margin="5,5,5,5" BackgroundColor="LightGray">
                                                <Grid>
                                                    <Grid.ColumnDefinitions>
                                                        <ColumnDefinition
Width="4*" />
                                                        <ColumnDefinition
Width="1*" />
                                                        <ColumnDefinition
Width="1*" />
                                                        <ColumnDefinition
Width="1*" />
                                                    </Grid.ColumnDefinitions>
                                                    <Label
Grid.Column="0" Text="{Binding Name}" FontSize="Medium"
LineBreakMode="CharacterWrap" HeightRequest="30"

```

```

Margin="5, 0, 0, 0"/>
<Label
Grid.Column="1" Text="{Binding Weight, StringFormat='{0} г'}" Margin="5, 15,
0, 0"/>
<Label
Grid.Column="2" Text="{Binding Calories, StringFormat='{0:0} к'}" Margin="5,
15, 0, 0"/>
<Button
Grid.Column="3" Padding="0" Text="-" FontSize="Large" Margin="5, 0, 0, 0"

Command="{Binding DelClickedCommand, Source={RelativeSource
AncestorType={x:Type viewmodels:MainViewModel}}}"

CommandParameter="{Binding Id}" />
</Grid>
</Border>
</DataTemplate>
</CollectionView.ItemTemplate>
</CollectionView>
</Grid>

</Grid>
</Border>
</DataTemplate>
</CollectionView.ItemTemplate>
</CollectionView>

</Grid>
</Grid>
</ContentPage>

```

Листинг Г.16 – Исходный код страницы ProductsPage.xaml

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="NutritionControlMAUI.Views.ProductsPage"
xmlns:viewmodels="clr-namespace:NutritionControlMAUI.ViewModels"
xmlns:dto="clr-namespace:NutritionControlMAUI.DTO"
x:DataType="viewmodels:ProductsViewModel"
Title="Продукты">
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="auto"/>
<RowDefinition Height="4*"/>
<RowDefinition Height="*/"/>
</Grid.RowDefinitions>
<SearchBar x:Name="searchBar" HorizontalOptions="Center"
WidthRequest="420"
SearchCommand="{Binding PerformSearch}"
SearchCommandParameter="{Binding Text, Source={x:Reference
searchBar}}"/>
<CollectionView ItemsSource="{Binding Products}" Grid.Row="1">
<CollectionView.ItemTemplate>
<DataTemplate x:DataType="dto:ProductDTO">
<Border StrokeShape="Rectangle" Margin="5,5,5,5"
BackgroundColor="WhiteSmoke">
<Grid>
<Grid.GestureRecognizers>
<TapGestureRecognizer Command="{Binding
TapProductCommand, Source={RelativeSource AncestorType={x:Type
viewmodels:ProductsViewModel}}}"
CommandParameter="{Binding .}" />

```

```

        </Grid.GestureRecognizers>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="3*" />
            <ColumnDefinition Width="2*" />
            <ColumnDefinition Width="1*" />
        </Grid.ColumnDefinitions>
        <Label Grid.Column="0" Text="{Binding Name}"
FontSize="Medium"
LineBreakMode="CharacterWrap"
HeightRequest="55"
Margin="5, 0, 0, 0"/>
        <Label Grid.Column="1" Margin="5, 15, 0, 0">
            <Label.Text>
                <MultiBinding StringFormat="B: {0} Ж: {1}
Y: {2}">
                    <Binding Path="Proteins" />
                    <Binding Path="Fats"/>
                    <Binding Path="Carbohydrates"/>
                </MultiBinding>
            </Label.Text>
        </Label>
        <Label Grid.Column="2" Text="{Binding Calories,
StringFormat='{0} ккал'}" Margin="5, 15, 0, 0"/>
    </Grid>
</Border>
</DataTemplate>
</CollectionView.ItemTemplate>
</CollectionView>
<StackLayout Grid.Row="2">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="*" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <Grid Grid.Row="0" Grid.ColumnSpan="2" >
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="3*" />
                <ColumnDefinition Width="2*" />
                <ColumnDefinition Width="1*" />
            </Grid.ColumnDefinitions>
            <Label Grid.Column="0" Text="{Binding Product.Name}"
FontSize="Medium"
LineBreakMode="CharacterWrap"
HeightRequest="55"
Margin="5, 0, 0, 0"/>
            <Label Grid.Column="1" Margin="5, 15, 0, 0">
                <Label.Text>
                    <MultiBinding StringFormat="{0} B: {0} Ж: {1} Y:
{2}">
                        <Binding Path="Product.Proteins" />
                        <Binding Path="Product.Fats"/>
                        <Binding Path="Product.Carbohydrates"/>
                    </MultiBinding>
                </Label.Text>
            </Label>
            <Label Grid.Column="2" Text="{Binding Product.Calories,
StringFormat='{0} ккал'}" Margin="5, 15, 0, 0"/>
        </Grid>

```

```

        <Entry Grid.Row="1" Grid.Column="0" Text="{Binding
CreateProductMealDTO.Weight}" Keyboard="Numeric"
BackgroundColor="WhiteSmoke"/>
        <Button Grid.Row="1" Grid.Column="1" Text="Добавить"
Command="{Binding AddCommand}"/>
    </Grid>

</StackLayout>
</Grid>
</ContentPage>

```

Листинг Г.17 – Исходный код страницы UserInformationPage.xaml

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="NutritionControlMAUI.Views.UserInfoPage"
xmlns:dto="clr-namespace:NutritionControlMAUI.DTO"
xmlns:behaviors="clr-namespace:NutritionControlMAUI"
xmlns:viewmodels="clr-namespace:NutritionControlMAUI.ViewModels"
x:DataType="viewmodels:UserInfoViewModel">
    <ContentPage.Behaviors>
        <behaviors:ContentPageAppendingCommandBehavior
AppearingCommand="{Binding ReloadCommand}" />
    </ContentPage.Behaviors>
    <ContentPage.ToolbarItems>
        <ToolbarItem Text="Сохранить" Command="{Binding SaveClick}"/>
    </ContentPage.ToolbarItems>
    <StackLayout Padding="20" Spacing="10" WidthRequest="380"
VerticalOptions="Start" HorizontalOptions="CenterAndExpand">
        <Label Text="Ваш рост (см):"/>
        <Entry x:Name="EntryHeight"
Text="{Binding RegUserDTO.Height}"
Keyboard="Numeric"
Placeholder="Рост"
BackgroundColor="WhiteSmoke" />
        <Label Text="Ваш вес (кг):"/>
        <Entry x:Name="EntryWeight"
Text="{Binding RegUserDTO.Weighth}"
Keyboard="Numeric"
Placeholder="Вес"
BackgroundColor="WhiteSmoke"/>
        <Label Text="Ваш пол:"/>
        <Picker x:Name="PickerGender"
ItemsSource="{Binding Genders}"
SelectedItem="{Binding RegUserDTO.Gender}"
BackgroundColor="WhiteSmoke"/>
        <Label Text="Ваш день рождения:"/>
        <DatePicker x:Name="DatePickerBirthday"
BackgroundColor="WhiteSmoke"
Date="{Binding RegUserDTO.Birthday}"/>
        <Label Text="Ваша цель:"/>
        <Picker x:Name="PickerPurpose"
ItemsSource="{Binding Purposes}"
SelectedItem="{Binding RegUserDTO.Purpose}"
BackgroundColor="WhiteSmoke"/>
    </StackLayout>
</ContentPage>

```

Приложение Д
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ СЕРВЕРНОЙ ЧАСТИ
ПРИЛОЖЕНИЯ
(рекомендуемое)

<h2>Responses</h2>	
<h3>Curl</h3>	
<pre>curl -X 'POST' \ 'http://192.168.1.231:5059/User/register' \ -H 'accept: */*' \ -H 'Content-Type: application/json' \ -d '{ "login": "newuser1", "password": "pass1234", "purpose": "Похудеть", "gender": "Мужской", "height": 180, "weight": 80, "birthday": "2000-01-01T10:00:27.610Z" }'</pre>	
<h3>Request URL</h3>	
<pre>http://192.168.1.231:5059/User/register</pre>	
<h3>Server response</h3>	
Code	Details
200	<div><h4>Response headers</h4><pre>content-length: 0 date: Wed, 15 Jun 2022 10:06:46 GMT server: Kestrel</pre></div>
<h3>Responses</h3>	
Code	Description
200	Success

Рисунок Д.1 – Добавление нового пользователя

The screenshot displays a web browser window with a REST client interface. The interface is divided into several sections:

- Curl**: A text area containing a curl command:

```
curl -X 'POST' \
'http://192.168.1.231:5059/User/authorize' \
-H 'accept: text/plain' \
-H 'Content-Type: application/json' \
-d '{
  "login": "newuser1",
  "password": "pass1234"
}'
```
- Request URL**: A text area containing the URL: `http://192.168.1.231:5059/User/authorize`
- Server response**: A section with a table header containing **Code** and **Details**.
- 200**: A section containing the **Response body**, which is a long JSON string. To the right of the JSON string is a **Download** button with a download icon.
- Response headers**: A section containing a list of headers: `content-type: text/plain; charset=utf-8`, `date: Wed, 15 Jun 2022 10:18:37 GMT`, `server: Kestrel`, and `transfer-encoding: chunked`.

Рисунок Д.2 – Авторизация пользователя

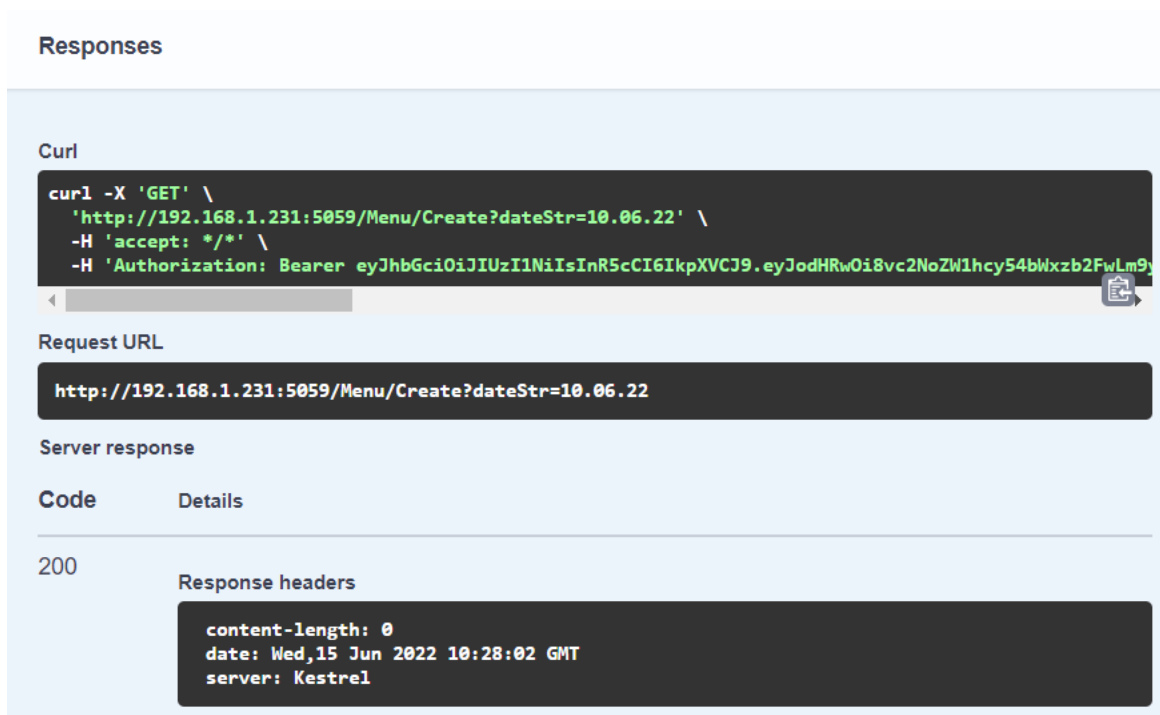


Рисунок Д.5 – Подбор персонального меню

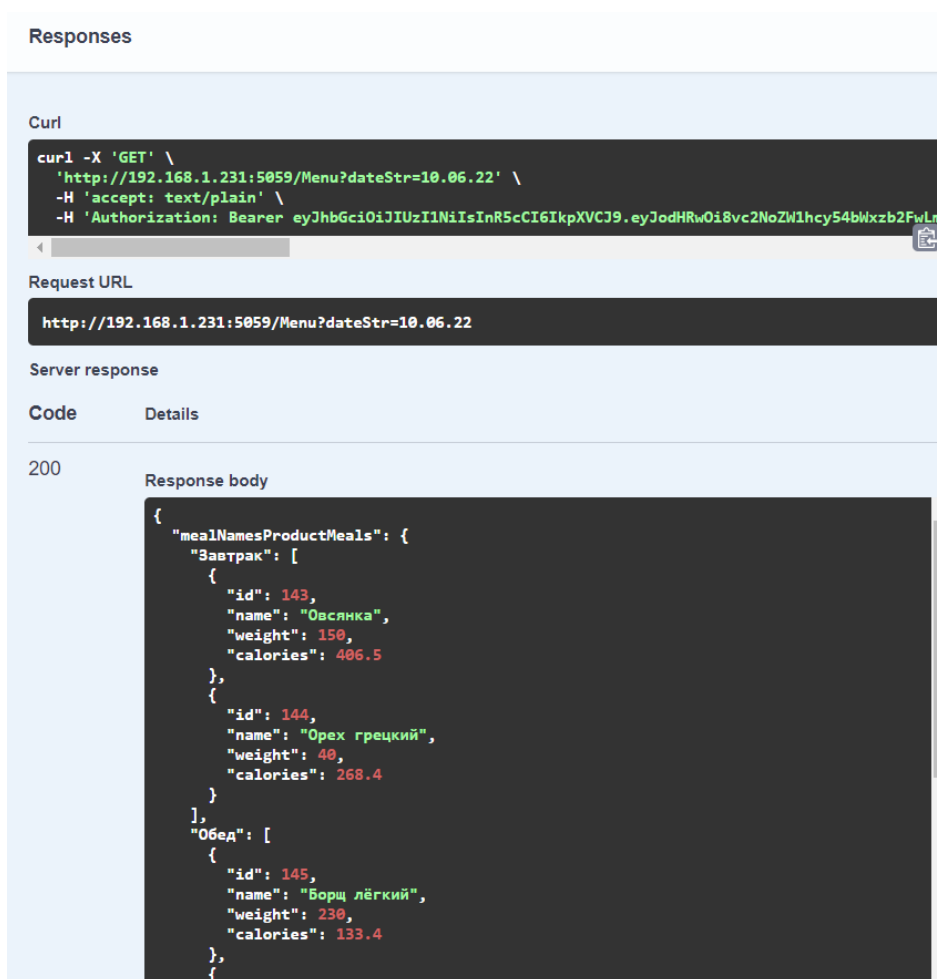


Рисунок Д.6 – Получение меню пользователя

Responses

Curl

```
curl -X 'GET' \
  'http://192.168.1.231:5059/Product/AllProducts' \
  -H 'accept: text/plain' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzbnZwLm9y' \
  >
```

Request URL

```
http://192.168.1.231:5059/Product/AllProducts
```

Server response

Code	Details
200	<p>Response body</p> <pre>[{ "id": 1, "name": "Абрикосы", "proteins": 1, "fats": 0, "carbohydrates": 10, "calories": 45 }, { "id": 2, "name": "Авокадо", "proteins": 2, "fats": 19, "carbohydrates": 7, "calories": 197 },]</pre>

Рисунок Д.7 – Получение списка всех продуктов

Responses

Curl

```
curl -X 'GET' \
  'http://192.168.1.231:5059/Product/FindByName?substr=%D0%BC%D0%BE%D0%BB' \
  -H 'accept: text/plain' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzbnZwLm9y' \
  >
```

Request URL

```
http://192.168.1.231:5059/Product/FindByName?substr=%D0%BC%D0%BE%D0%BB
```

Server response

Code	Details
200	<p>Response body</p> <pre>[{ "id": 112, "name": "Картофель молодой", "proteins": 2, "fats": 0, "carbohydrates": 16, "calories": 61 }, { "id": 138, "name": "Кофе молотый Арабика", "proteins": 12, "fats": 14, "carbohydrates": 3, "calories": 193 },]</pre>

Рисунок Д.8 – Поиск списка продуктов по совпадению подстроки

Приложение Е
ИСХОДНЫЙ КОД МОДУЛЬНОГО ТЕСТИРОВАНИЯ СЕРВЕРНОЙ
ЧАСТИ ПРИЛОЖЕНИЯ
(рекомендуемое)

Листинг Е.1 – Исходный код модульного тестирования приложения

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Moq;
using WebNutritionControl.DTO;
using WebNutritionControl.Model;

namespace TestProject
{
    public class UnitTests
    {
        [Fact]
        public async Task Authorize_ReturnsJWT_WithCorrectData()
        {
            var dto = new AuthUserDTO()
            {
                Login = "andrey",
                Password = "qwertyui",
            };

            var options = new DbContextOptionsBuilder<NutritionContext>()
                .UseInMemoryDatabase(databaseName: "database")
                .Options;

            var m_Context = new NutritionContext(options);
            var controller = new UserController(m_Context);
            var result = await controller.Authorize(dto);
            Assert.IsType<ActionResult<string>>(result);
        }

        [Fact]
        public async Task Authorize_ReturnsNotFound_WithBadLogin()
        {
            var dto = new AuthUserDTO()
            {
                Login = "thrgfevd",
                Password = "qwertyui"
            };

            var options = new DbContextOptionsBuilder<NutritionContext>()
                .UseInMemoryDatabase(databaseName: "database")
                .Options;

            var m_Context = new NutritionContext(options);
            var controller = new UserController(m_Context);
            var result = await controller.Authorize(dto);
            var expected = "Пользователя с таким логином не существует!";

            Assert.Equal(expected, ((ObjectResult)result.Result).Value);
        }

        [Fact]
        public async Task Register_ReturnsOk_WithGoodData()
        {
            var dto = new RegUserDTO()
            {
                Login = "TestUser",
            }
        }
    }
}

```

```

        Password = "testpass",
        Birthday = DateTime.Now,
        Gender = "Мужской",
        Height = 180,
        Weigth = 70,
        Purpose = "Поддерживать вес"
    };

    var options = new DbContextOptionsBuilder<NutritionContext>()
        .UseInMemoryDatabase(databaseName: "database")
        .Options;

    var m_Context = new NutritionContext(options);
    var controller = new UserController(m_Context);

    var result = await controller.Register(dto);
    var expected = controller.Ok();

    Assert.Equal(expected.StatusCode,
        ((StatusCodeResult)result).StatusCode);
}

[Fact]
public async Task Register_ReturnsBadRequest_WithExistUser()
{
    var dto = new RegUserDTO()
    {
        Login = "TestUser1",
        Password = "testpass",
        Birthday = DateTime.Now,
        Gender = "Мужской",
        Height = 180,
        Weigth = 70,
        Purpose = "Поддерживать вес"
    };

    var options = new DbContextOptionsBuilder<NutritionContext>()
        .UseInMemoryDatabase(databaseName: "database")
        .Options;

    var m_Context = new NutritionContext(options);
    var controller = new UserController(m_Context);
    await controller.Register(dto);
    var result = await controller.Register(dto);
    var expected = "Пользователь с таким логином уже существует!";

    Assert.Equal(expected, ((ObjectResult)result).Value);
}

[Fact]
public async Task GetAllProducts_ReturnsOk()
{
    var options = new DbContextOptionsBuilder<NutritionContext>()
        .UseInMemoryDatabase(databaseName: "database")
        .Options;

    var m_Context = new NutritionContext(options);
    var controller = new ProductController(m_Context);

    for (int i = 0; i < 10; i++)
    {
        Product product = new()

```

```

        {
            Calories = 100,
            Carbohydrates = 10,
            Fats = 1,
            Proteins = 11,
            Name = "product" + "i"
        };
        await m_Context.Products.AddAsync(product);
        m_Context.SaveChanges();
    }

    var result = await controller.GetAllProducts();
    var count =
        ((List<ProductDTO>) ((ObjectResult) result.Result).Value).Count;
    var expected = 10;

    Assert.InRange(count, expected, int.MaxValue);
}

[Fact]
public async Task GetProductsByName_ReturnsOk()
{
    var options = new DbContextOptionsBuilder<NutritionContext>()
        .UseInMemoryDatabase(databaseName: "database")
        .Options;

    var m_Context = new NutritionContext(options);
    var controller = new ProductController(m_Context);

    for (int i = 0; i < 5; i++)
    {
        Product product = new()
        {
            Calories = 100,
            Carbohydrates = 10,
            Fats = 1,
            Proteins = 11,
            Name = "someproduct" + "i"
        };
        await m_Context.Products.AddAsync(product);
        m_Context.SaveChanges();
    }

    var result = await controller.FindByNameProducts("some");
    var count =
        ((List<ProductDTO>) ((ObjectResult) result.Result).Value).Count;
    var expected = 5;

    Assert.Equal(expected, count);
}

}
}

```

Приложение Ж
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ КЛИЕНТСКОЙ ЧАСТИ
ПРИЛОЖЕНИЯ
(рекомендуемое)

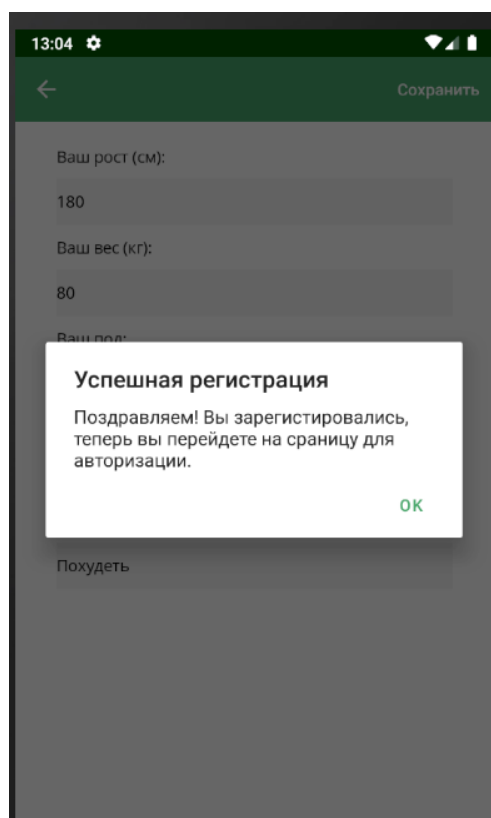


Рисунок Ж.1 – Ввести данные для регистрации

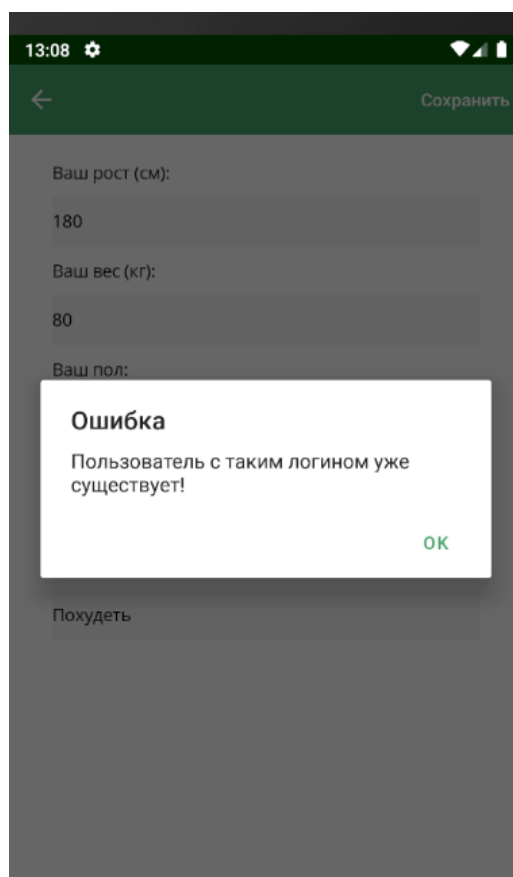


Рисунок Ж.2 – Попытаться использовать занятый логин

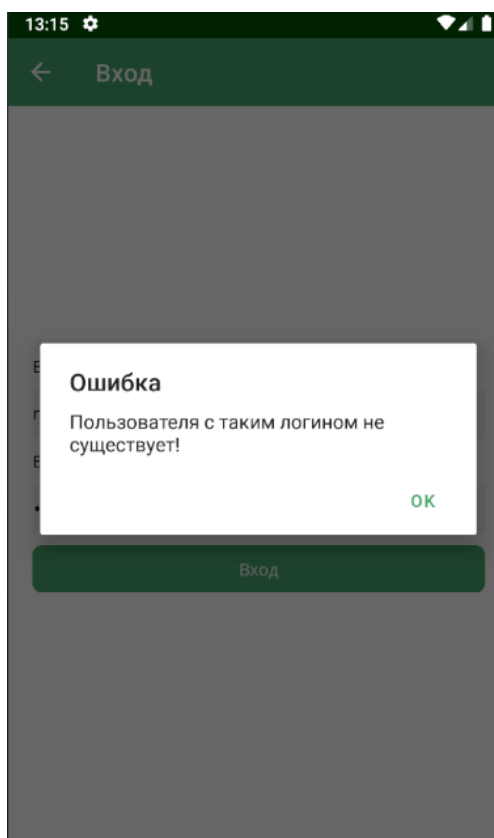


Рисунок Ж.3 – Попытаться ввести несуществующий логин

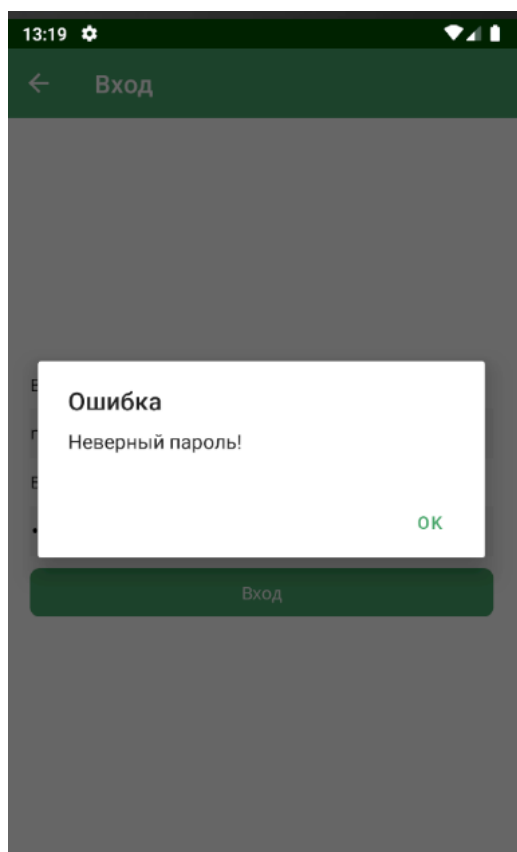
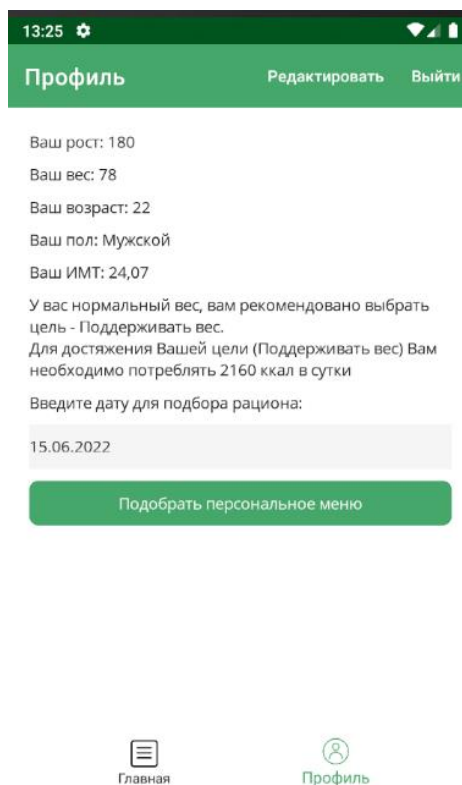


Рисунок Ж.4 – Попытаться ввести неправильный пароль



13:25

Профиль Редактировать Выйти

Ваш рост: 180
Ваш вес: 78
Ваш возраст: 22
Ваш пол: Мужской
Ваш ИМТ: 24,07

У вас нормальный вес, вам рекомендовано выбрать цель - Поддерживать вес.
Для достижения Вашей цели (Поддерживать вес) Вам необходимо потреблять 2160 ккал в сутки

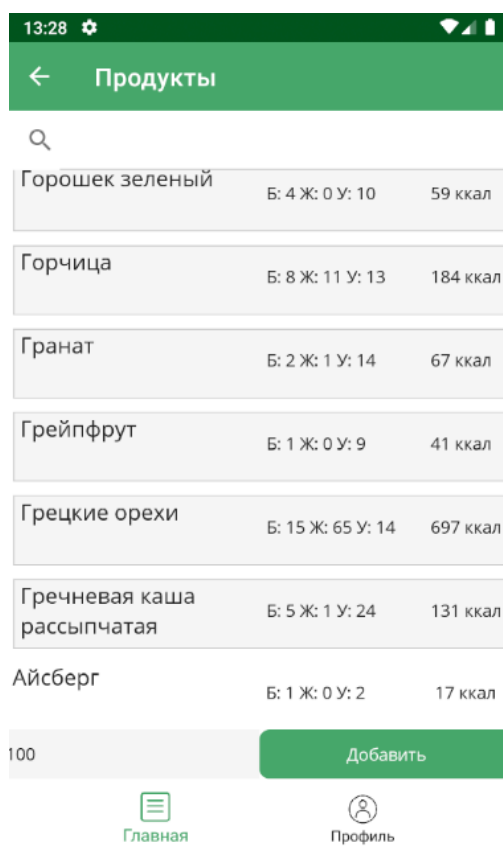
Введите дату для подбора рациона:

15.06.2022

Подобрать персональное меню

Главная Профиль

Рисунок Ж.5 – Ввести верные данные для авторизации



13:28

← Продукты

Горошек зеленый Б: 4 Ж: 0 У: 10 59 ккал

Горчица Б: 8 Ж: 11 У: 13 184 ккал

Гранат Б: 2 Ж: 1 У: 14 67 ккал

Грейпфрут Б: 1 Ж: 0 У: 9 41 ккал

Грецкие орехи Б: 15 Ж: 65 У: 14 697 ккал

Гречневая каша рассыпчатая Б: 5 Ж: 1 У: 24 131 ккал

Айсберг Б: 1 Ж: 0 У: 2 17 ккал

100

Добавить

Главная Профиль

Рисунок Ж.6 – Просмотр списка блюд

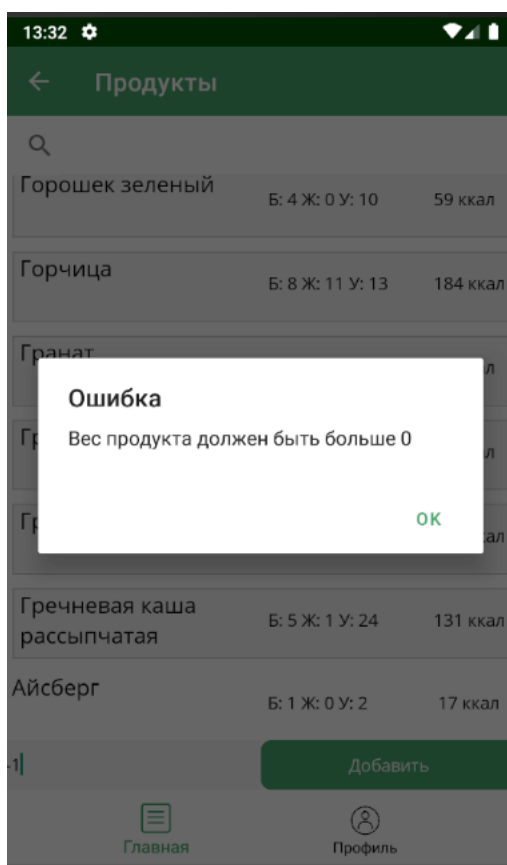


Рисунок Ж.7 – Попытка добавить блюдо с отрицательным весом

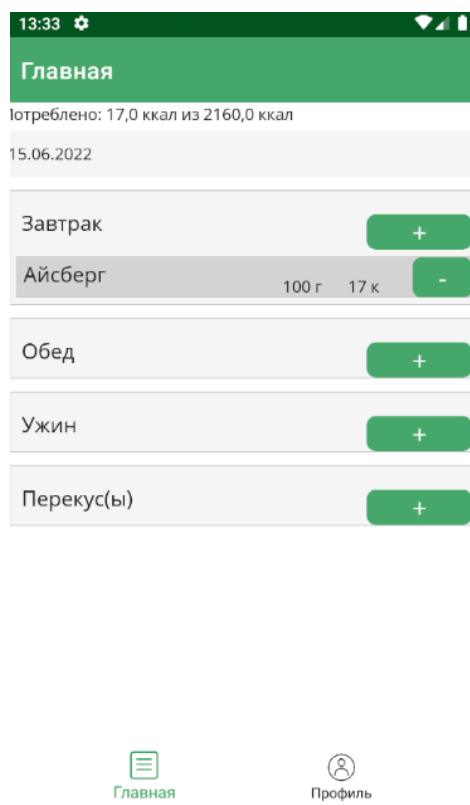


Рисунок Ж.8 – Добавление блюда в прием пищи

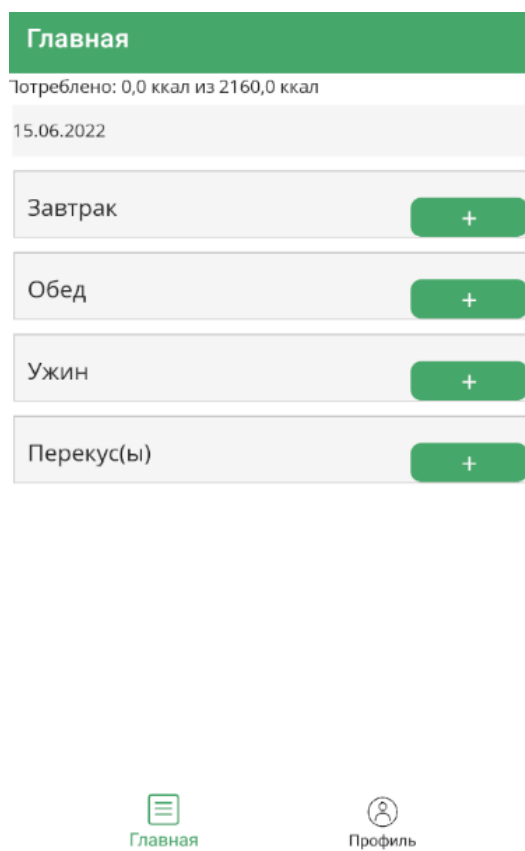


Рисунок Ж.9 – Удаление блюда из приема пищи

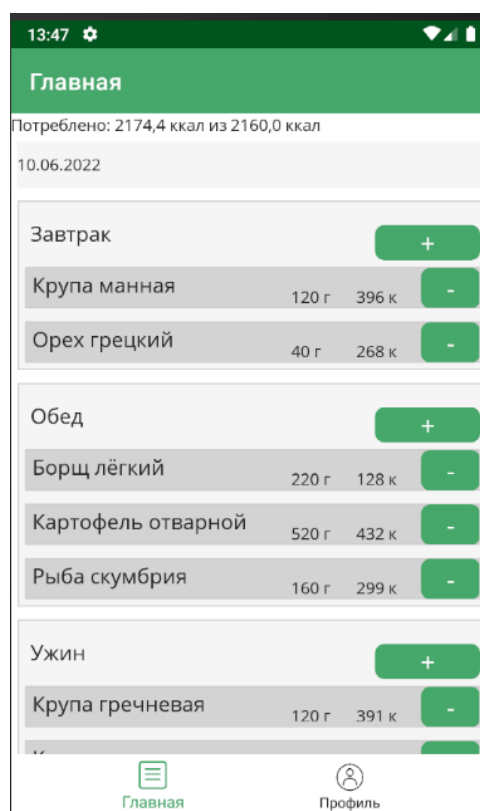


Рисунок Ж.10 – Результат подбора персонального меню