

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

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

БАКАЛАВРСКАЯ РАБОТА

на тему

Приложение для составления рациона
питания и расчета калорийности

Студент

(подпись, дата)

Морозов Артем Сергеевич

(ФИО полностью)

Руководитель

(подпись, дата)

Макарычев П. П.

(фамилия, инициалы)

Нормоконтролёр

(подпись, дата)

Такташкин Д.В.

(фамилия, инициалы)

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

Заведующий кафедрой

(подпись)

Козлов А.Ю.

(фамилия, инициалы)

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

Секретарь ГЭК

(подпись)

Попова Н.А.

(фамилия, инициалы)

Пенза, 2025

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

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«__» _____ 2025 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА**

1. Студент Морозов Артем Сергеевич гр. 21ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Приложение для составления рациона питания и расчета калорийности

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

3. Руководитель работы к.т.н., профессор кафедры «МОиПЭВМ» Макарычев П. П.

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

1) Требуется разработать приложение, предоставляющее рекомендации рационов питания в соответствии с параметрами и целями пользователя, а также инструмент для расчета калорийности съеденных за день блюд.

2) Среда выполнения приложения: операционная система Windows 10

3) Языковые средства разработки: Golang, SQL, UML.

4) Инструментальные средства разработки: JetBrains Goland, Microsoft Visual Studio Code, Docker, Microsoft Office 2020, Visual Paradigm 17.0

5) Технологии разработки: REST API, микросервисная архитектура, миграции, контейнеризация

6) Основные функции приложения:

а) регистрация и авторизация пользователей;

б) ввод, редактирование, просмотр параметров пользователя (рост, вес, пол, возраст, цель);

в) расчет и отображение индекса массы тела (ИМТ) и дневной нормы КБЖУ;

г) просмотр рекомендаций по питанию: дневная норма КБЖУ и подходящие рационы питания;

д) ведение дневника питания с возможностью ввода съеденных блюд (калоризатор);

е) заполнение и просмотр графика изменения веса пользователя.

5. Перечень подлежащих разработке вопросов

1) Постановка задачи и анализ предметной области

2) Проектирование

3) Реализация

4) Контроль качества программного обеспечения

5) План разработки и оценка бюджета

6) Оформление пояснительной записки

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	12.04.2025 – 16.04.2025	
2	Проектирование программных средств	20%	17.04.2025 – 26.04.2025	
3	Реализация программных средств	30%	27.04.2025 – 11.05.2025	
4	Тестирование программных средств	20%	12.05.2025 – 21.05.2025	
5	Оформление пояснительной записки	20%	22.05.2025 – 01.06.2025	

Дата выдачи задания «22» декабря 2025г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

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

Реферат

Пояснительная записка содержит 149 листов, 35 рисунков, 36 таблиц, 22 использованных источника, 4 приложения.

GOLANG, WEB, HTML, JS CORE, БАЗА ДАННЫХ, POSTGRES, UML, GOLAND, MICROSOFT VISUAL STUDIO CODE, DOCKER

Объект разработки – информационная система, предназначенная для автоматизации процесса составления дневной нормы макронутриентов и подбора питания, а также ведения записей в дневник питания.

Цель проекта – создание веб-приложения, объединяющего функциональность по расчету суточной нормы КБЖУ (калорий, белков, жиров, углеводов), подбору рекомендаций по питанию на основе индивидуальных параметров пользователя, а также ведению пищевого дневника с возможностью анализа статистики.

Среда выполнения: операционная система Windows 10. Язык программирования – Golang. Язык запросов – SQL. СУБД – PostgreSQL.

Приложение реализуется по клиент-серверной архитектуре. Серверная логика обрабатывается на языке Golang. Клиентская часть выполнена с применением JS React.

					ПГУ 09.03.02 – 08BP211.17 ПЗ			
Изм.	Лист	№ докум.	Подп.	Дата	Приложение для составления рациона питания и расчета калорийности. Пояснительная записка	Лит.	Лист	Листов
Разраб.		Морозов А. С.						
Пров.		Макарычев П. П.					4	149
Реценз.						Группа 21ВИ1		
Н. контр.		Такташкин Д.В						
Утверд.		Козлов А.Ю.						

Содержание

Введение	7
1 Анализ сервисов для подбора питания	8
1.1 Анализ предметной области и постановка задачи	8
1.2 Обзор и анализ сервисов для подбора питания	9
1.2.1 Сервис «FatSecret».....	9
1.2.2 Сервис «MyNetDiary»	10
1.2.3 Сервис «Cronometer».....	11
1.2.4 Сервис «MyFitnessPal».....	11
1.2.5 Сервис «LifeSum».....	12
1.3 Сравнение сервисов составления рациона питания и расчета калорийности	14
1.4 Анализ функциональных требований	21
1.5 Анализ нефункциональных требований	25
2 Проектирование сервиса для подбора питания	26
2.1 Архитектура приложения	26
2.2 Проектирование структур данных	26
2.2.1 Концептуальная модель базы данных.....	26
2.3 Проектирование программных средств сервиса подбора питания	27
2.4 Проектирование пользовательского интерфейса	30
3 Реализация сервиса подбора питания	33
3.1 Реализация базы данных	33
3.1.1 Физическая модель базы данных	33
3.1.2 Заполнение базы данных тестовыми данными.....	38
3.2 Реализация серверной части приложения.....	39
3.3 Реализация клиентской части приложения.....	40
4 Контроль качества программного обеспечения	46
4.1 Выбор режима тестирования	46
4.2 Функциональное тестирование	46
4.3 Метрики кода.....	49
4.3.1 Метрики Холстеда.....	49
4.3.2 Метрики сопровождаемости.....	52

4.4 Планирование разработки и оценка бюджета	52
Заключение	54
Список использованных источников	56
Приложение А (обязательное) Листинг приложения.....	58
Приложение Б (рекомендуемое) Листинг SQL-запросов	112
Приложение В (рекомендуемое) Листинг Docker-файлов.....	124
Приложение Г (рекомендуемое) Диаграммы классов.....	128
Приложение Д (рекомендуемое) Результаты функционального тестирования	131

Введение

Рациональное питание занимает ключевую роль в поддержании здоровья человека, профилактике заболеваний и достижении личных целей. В условиях современного ритма жизни, когда большинство людей испытывают нехватку времени и перегружены информацией, системный подход к планированию питания становится очень важным.

Актуальность разработки программных средств, автоматизирующих процессы составления рациона и контроля калорийности, обусловлена растущей осведомлённостью населения о роли питания в формировании качества жизни. Всё больше людей стремятся следить за своим здоровьем, учитывать индивидуальные физиологические параметры и научно обоснованные нормы потребления КБЖУ (калорий, белков, жиров и углеводов). Однако ручной расчёт показателей и подбор подходящих блюд требует времени, внимания и знания основ нутрициологии, что затруднительно для большинства пользователей.

Цифровые инструменты, позволяющие в интерактивной форме формировать рацион, отслеживать динамику веса и вносить данные о приёмах пищи, становятся важным элементом в формировании осознанного отношения к питанию и здоровому образу жизни.

Целью выпускной квалификационной работы является разработка веб-приложения для составления рационов питания и расчета калорийности, включающего:

- регистрацию и настройку профиля пользователя с учетом индивидуальных параметров;
- расчет индекса массы тела (ИМТ) и суточной нормы КБЖУ;
- подбор сбалансированных программ питания;
- ведение пищевого дневника и учёт потреблённых блюд;
- отображение статистики и динамики веса пользователя в виде графика.

1 Анализ сервисов для подбора питания

1.1 Анализ предметной области и постановка задачи

В условиях стремительного ритма жизни и роста осведомлённости о важности здорового образа жизни всё большее внимание уделяется вопросам правильного и сбалансированного питания. Правильное питание способствует поддержанию нормальной массы тела, улучшению общего самочувствия, профилактике хронических заболеваний и достижению индивидуальных целей – будь то снижение веса, набор мышечной массы или удержание текущего физического состояния.

Подбор рациона требует учета множества факторов: пола, возраста, роста, веса и цели пользователя. Самостоятельный расчет дневной нормы калорий и нутриентов (белков, жиров, углеводов) может быть сложным и трудоёмким, особенно для пользователей без специальной подготовки. Это создаёт потребность в удобных цифровых решениях, способных автоматизировать процесс подбора рациона и обеспечить наглядный контроль за питанием.

Актуальность разработки программного продукта для составления рациона и расчета калорийности обусловлена необходимостью предоставления пользователю персонализированного инструмента, позволяющего учитывать его физиологические параметры, предоставлять полноценные программы питания, вести пищевой дневник и отслеживать динамику изменений массы тела.

Для эффективной реализации данной задачи требуется приложение, сочетающее функции расчета ИМТ, определения суточной нормы КБЖУ, подбора программ питания на день и учета фактически потреблённых продуктов. Дополнительную значимость приобретает возможность визуального отображения прогресса пользователя в виде графиков изменения массы тела.

1.2 Обзор и анализ сервисов для подбора питания

1.2.1 Сервис «FatSecret»

Бесплатное приложение для подсчёта калорий с базой продуктов, дневником питания и возможностью отслеживания веса. Является самым популярным сервисом среди всех описанных. Интерфейс сервиса указан на рисунке 1.

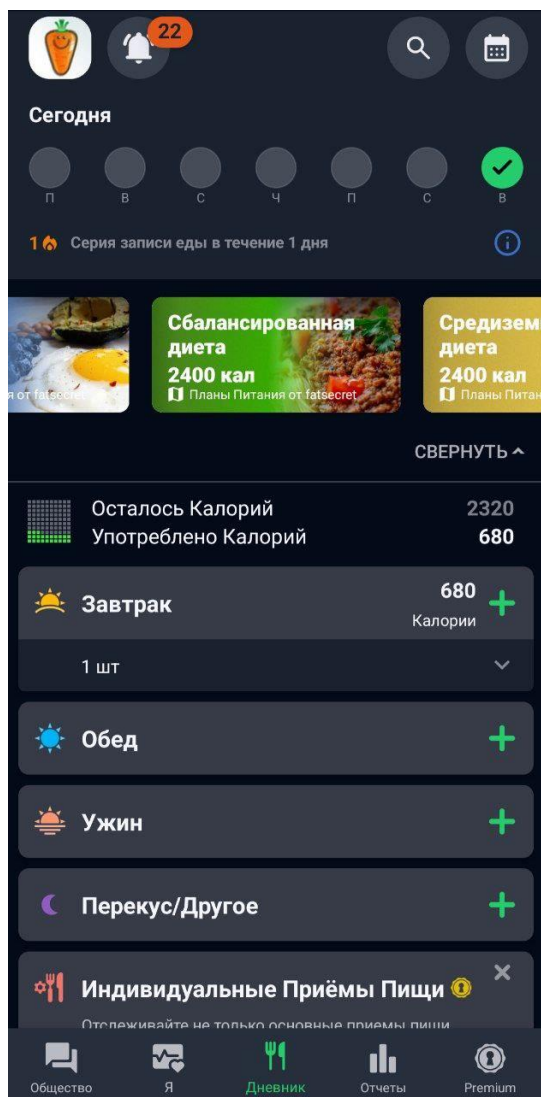


Рисунок 1 – Интерфейс сервиса «FatSecret»

На основе отзывов пользователей [2] были выделены достоинства и недостатки сервиса.

Достоинства:

- удобный ввод продуктов и блюд, импорт рецептов;

- сообщество пользователей и возможность делиться результатами;
- полноценный функционал без подписки.

Недостатки: минимальный набор инструментов отслеживания динамики веса пользователя.

1.2.2 Сервис «MyNetDiary»

Инструмент для контроля питания и веса, включающий дневник питания, учёт воды, активности и медицинских показателей. Интерфейс сервиса указан на рисунке 2.

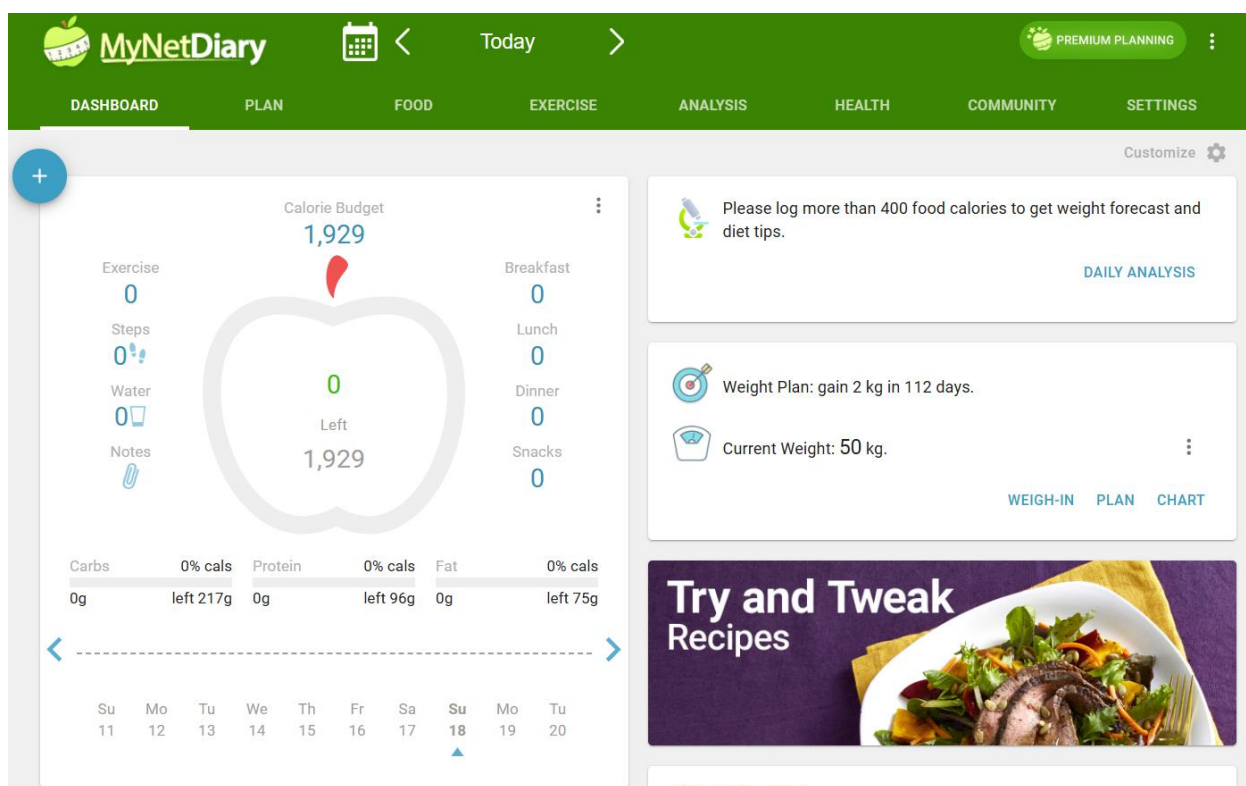


Рисунок 2 – Интерфейс сервиса «MyNetDiary»

На основе отзывов пользователей [3] были выделены достоинства и недостатки сервиса.

Достоинства:

- удобное отслеживание цели по набору/снижению веса;
- поддержка персональных целей (например, диабет).

Недостатки:

- часть функций только в премиум-версии;

- устаревший интерфейс.

1.2.3 Сервис «Cronometer»

Приложение для точного контроля питания с научным подходом – отслеживает не только КБЖУ, но и витамины, минералы и микроэлементы. Интерфейс сервиса указан на рисунке 3.

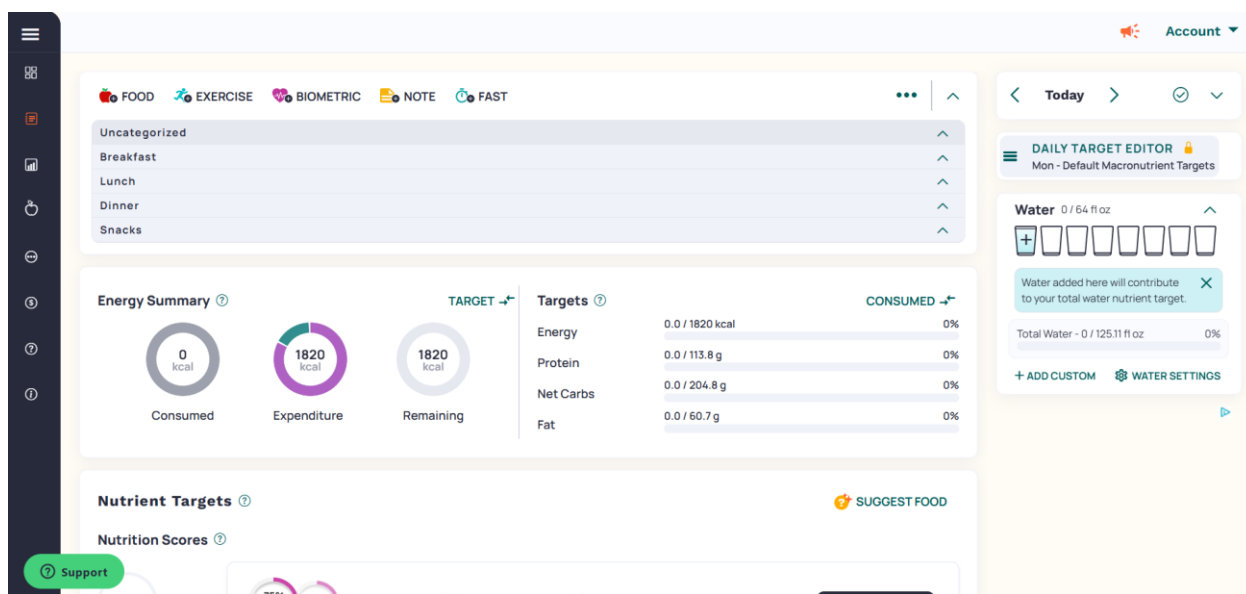


Рисунок 3 – Интерфейс сервиса «Cronometer»

На основе отзывов пользователей [4] были выделены достоинства и недостатки сервиса.

Достоинства:

- богатая база нутриентов;
- подходит для продвинутых пользователей и специалистов.

Недостатки:

- часть функций только в премиум-версии;
- сложный для новичков интерфейс.

1.2.4 Сервис «MyFitnessPal»

Сервис предлагает обширную базу данных продуктов, возможность сканирования штрих-кодов и интеграцию с различными фитнес-трекерами. Интерфейс сервиса указан на рисунке 4.

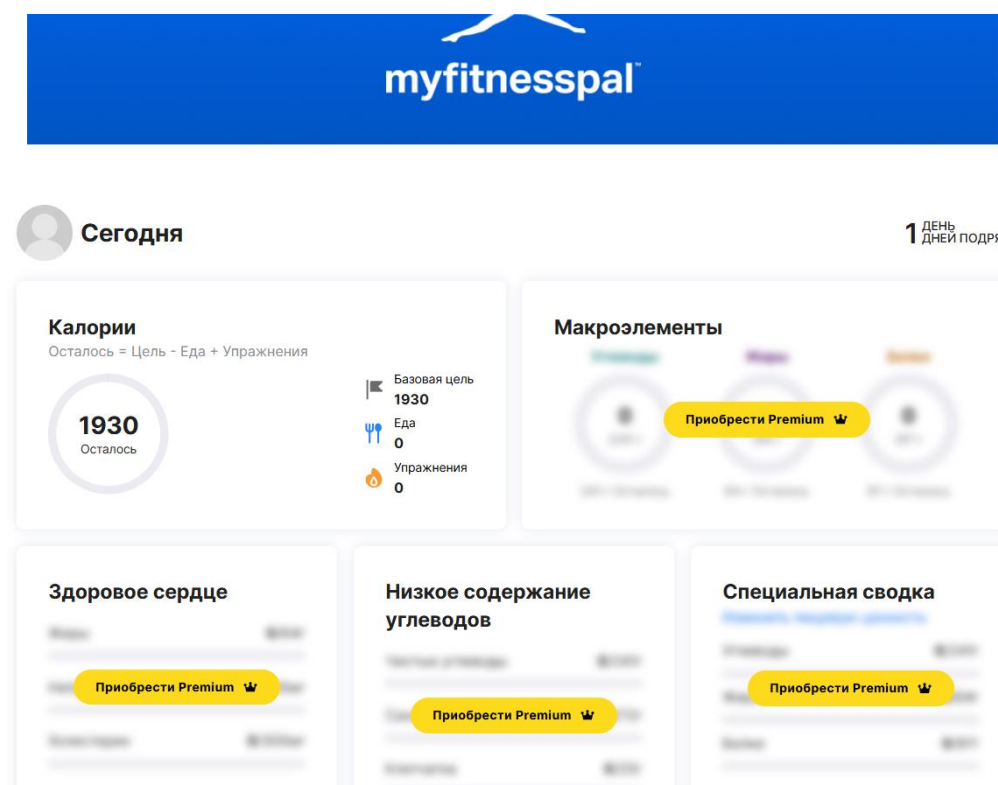


Рисунок 4 – Интерфейс сервиса «MyFitnessPal»

На основе отзывов пользователей [5] были выделены достоинства и недостатки сервиса.

Достоинства:

- обширная база данных продуктов, включая блюда из ресторанов;
- интеграция с популярными фитнес-приложениями и устройствами;
- возможность отслеживания макро и микронутриентов.

Недостатки:

- большинство функций недоступны в бесплатной версии;
- интерфейс перегружен рекламой в бесплатной версии.

1.2.5 Сервис «LifeSum»

Приложение с акцентом на здоровый образ жизни, предлагающее разнообразные диетические планы, рецепты и инструменты для формирования полезных привычек. Интерфейс сервиса указан на рисунке 5.

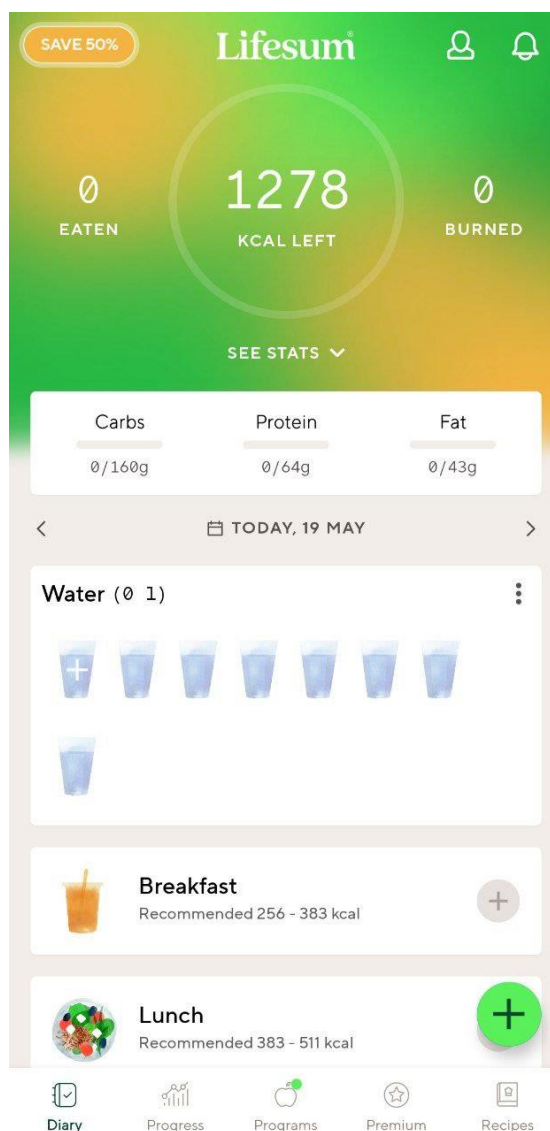


Рисунок 5 – Интерфейс сервиса «LifeSum»

На основе отзывов пользователей [6] были выделены достоинства и недостатки сервиса.

Достоинства:

- разнообразные планы питания (например, кето, веганский) с готовыми рецептами и списками покупок;
- интеграция с популярными фитнес-приложениями и устройствами.

Недостатки:

- большинство функций недоступны в бесплатной версии;
- некоторые пользователи отмечают неточности в базе данных продуктов и ограниченные возможности по корректировке информации.

1.3 Сравнение сервисов составления рациона питания и расчета калорийности

Для проведения сравнительного анализа популярных сервисов для составления рациона питания и расчета калорийности были определены ключевые критерии, для которых выставлены субъективные оценки важности, представленные в таблице 1.

Таблица 1 – Критерии оценки сервисов

№	Критерий	Описание	Оценка
1	Функциональность	Объём доступных функций	7
2	База данных продуктов	Объём и точность информации о продуктах питания, включая бренды и блюда ресторанов	6
3	Интерфейс и удобство использования	Интуитивность интерфейса и визуальная привлекательность	5
4	Планирование питания	Наличие готовых планов питания и рецептов	3

Для визуализации отношений весов критериев оценки приложений составлена матрица сравнений (таблица 2).

Таблица 2 – Матрица сравнений

Критерий \ критерий	1	2	3	4
1	1	1.2	1.4	2.3
2	0.83	1	1.2	2
3	0.71	0.83	1	1.6
4	0.43	0.5	0.625	1

Для произведения дальнейшей нормализации матрицы высчитаны суммы столбцов:

- функциональность = $1 + 0.83 + 0.71 + 0.43 = 2.97$;
- БД продуктов = $1.2 + 1 + 0.83 + 0.5 = 3.53$;
- интерфейс = $1.4 + 1.2 + 1 + 0.625 = 4.23$;
- планирование питания = $2.3 + 2 + 1.6 + 1 = 6.9$.

Нормализованная матрица (каждая ячейка разделена на сумму столбца) указана в таблице 3.

Таблица 3 – Нормализованная матрица сравнений

Критерий \ критерий	1	2	3	4
1	0.34	0.34	0.33	0.33
2	0.28	0.28	0.28	0.29
3	0.24	0.23	0.24	0.23
4	0.14	0.14	0.15	0.14

Итоговые веса критериев (таблица 4) рассчитывались путём нахождения среднего числа каждой строки таблицы 4.

Таблица 4 – Веса критериев

Критерий	Вес
Функциональность	0.34
БД продуктов	0.28
Интерфейс и удобство	0.24
Планирование питания	0.14

Для каждого критерия выставлены оценки аналогов. Оценки происходили по шкале Саати (таблица 5).

Таблица 5 – Шкала Саати

Интенсивность важности	Определение
1	Одинаковая важность
3	Умеренное превосходство
5	Значительное превосходство
7	Очевидное превосходство
9	Абсолютное превосходство
2, 4, 6, 8	Промежуточные значения

Сравнение оценок по критерию функциональности указаны в таблице 6.

Таблица 6 – Оценка сервисов по критерию функциональности

Сервис \ сервис	FatSecret	MyNetDiary	Cronometer	MyFitnessPal	Lifesum
FatSecret	1	1/3	1/3	1	1
MyNetDiary	3	1	1	3	3
Cronometer	3	1	1	3	3
MyFitnessPal	1	1/3	1/3	1	1
Lifesum	1	1/3	1/3	1	1

Для произведения дальнейшей нормализации матрицы высчитаны суммы столбцов:

- FatSecret = 9;
- MyNetDiary = 3;
- Cronometer = 3;
- MyFitnessPal = 9;
- Lifesum = 9.

Нормализованная матрица описана в таблице 7.

Таблица 7 – Нормализованная матрица по критерию функциональности

Сервис \ сервис	FatSecret	MyNetDiary	Cronometer	MyFitnessPal	Lifesum
FatSecret	0.11	0.11	0.11	0.11	0.11
MyNetDiary	0.33	0.33	0.33	0.33	0.33
Cronometer	0.33	0.33	0.33	0.33	0.33
MyFitnessPal	0.11	0.11	0.11	0.11	0.11
Lifesum	0.11	0.11	0.11	0.11	0.11

Локальные веса функциональности (среднее по строкам):

- FatSecret = 0.11;
- MyNetDiary = 0.33;
- Cronometer = 0.33;

- MyFitnessPal = 0.11;
- Lifesum = 0.11.

Сравнение оценок по критерию база данных указаны в таблице 8.

Таблица 8 – Оценка сервисов по критерию база данных

Сервис \ сервис	FatSecret	MyNetDiary	Cronometer	MyFitnessPal	Lifesum
FatSecret	1	1	1	1	3
MyNetDiary	1	1	1	1	3
Cronometer	1	1	1	1	3
MyFitnessPal	1	1	1	1	3
Lifesum	0.33	0.33	0.33	0.33	1

Для проведения дальнейшей нормализации матрицы рассчитаны суммы столбцов:

- FatSecret = 4.33;
- MyNetDiary = 4.33;
- Cronometer = 4.33;
- MyFitnessPal = 4.33;
- Lifesum = 13.

Нормализованная матрица описана в таблице 9.

Таблица 9 – Нормализованная матрица по критерию база данных

Сервис \ сервис	FatSecret	MyNetDiary	Cronometer	MyFitnessPal	Lifesum
FatSecret	0.23	0.23	0.23	0.23	0.23
MyNetDiary	0.23	0.23	0.23	0.23	0.23
Cronometer	0.23	0.23	0.23	0.23	0.23
MyFitnessPal	0.23	0.23	0.23	0.23	0.23
Lifesum	0.08	0.08	0.08	0.08	0.08

Локальные веса критерия база данных (среднее по строкам):

- FatSecret = 0.23;
- MyNetDiary = 0.23;
- Cronometer = 0.23;
- MyFitnessPal = 0.23;
- Lifesum = 0.08.

Сравнение оценок по критерию интерфейс указаны в таблице 10.

Таблица 10 – Оценка сервисов по критерию интерфейс

Сервис \ сервис	FatSecret	MyNetDiary	Cronometer	MyFitnessPal	Lifesum
FatSecret	1	5	3	3	1
MyNetDiary	0.2	1	0.33	0.33	0.2
Cronometer	0.33	3	1	1	0.33
MyFitnessPal	0.33	3	1	1	0.33
Lifesum	1	5	3	3	1

Для проведения дальнейшей нормализации матрицы рассчитаны суммы столбцов:

- FatSecret = 2.86;
- MyNetDiary = 17;
- Cronometer = 8.33;
- MyFitnessPal = 8.33;
- Lifesum = 2.86.

Нормализованная матрица описана в таблице 11.

Таблица 11 – Нормализованная матрица по критерию интерфейс

Сервис \ сервис	FatSecret	MyNetDiary	Cronometer	MyFitnessPal	Lifesum
FatSecret	0.35	0.29	0.36	0.36	0.35
MyNetDiary	0.07	0.06	0.04	0.04	0.07
Cronometer	0.12	0.18	0.12	0.12	0.12

Продолжение таблицы 11

Сервис \ сервис	FatSecret	MyNetDiary	Cronometer	MyFitnessPal	Lifesum
MyFitnessPal	0.12	0.18	0.12	0.12	0.12
Lifesum	0.35	0.29	0.36	0.36	0.35

Локальные веса критерия интерфейс (среднее по строкам):

- FatSecret = 0.34;
- MyNetDiary = 0.06;
- Cronometer = 0.13;
- MyFitnessPal = 0.13;
- Lifesum = 0.34.

Сравнение оценок по критерию функциональности указаны в таблице 12.

Таблица 12 – Оценка сервисов по критерию планирование питания

Сервис \ сервис	FatSecret	MyNetDiary	Cronometer	MyFitnessPal	Lifesum
FatSecret	1	3	3	3	1
MyNetDiary	0.33	1	1	1	0.33
Cronometer	0.33	1	1	1	0.33
MyFitnessPal	0.33	1	1	1	0.33
Lifesum	1	3	3	3	1

Для проведения дальнейшей нормализации матрицы рассчитаны суммы столбцов:

- FatSecret = 3;
- MyNetDiary = 9;
- Cronometer = 9;
- MyFitnessPal = 9;
- Lifesum = 3.

Нормализованная матрица описана в таблице 13.

Таблица 13 – Нормализованная матрица по критерию планирование питания

Сервис \ сервис	FatSecret	MyNetDiary	Cronometer	MyFitnessPal	Lifesum
FatSecret	0.33	0.33	0.33	0.33	0.33
MyNetDiary	0.11	0.11	0.11	0.11	0.11
Cronometer	0.11	0.11	0.11	0.11	0.11
MyFitnessPal	0.11	0.11	0.11	0.11	0.11
Lifesum	0.33	0.33	0.33	0.33	0.33

Локальные веса критерия планирование питания (среднее по строкам):

- FatSecret = 0.33;
- MyNetDiary = 0.11;
- Cronometer = 0.11;
- MyFitnessPal = 0.11;
- Lifesum = 0.33.

Учитывая оценки сервисов по критериям и веса, рассчитаны суммы весов (S):

- $S(\text{FatSecret}) = 0.33 * 0.34 + 0.34 * 0.28 + 0.34 * 0.24 + 0.33 * 0.14 = 0.68 + 0.84 + 0.72 + 0.42 = 0.34;$
- $S(\text{MyNetDiary}) = 0.11 * 0.34 + 0.06 * 0.28 + 0.06 * 0.24 + 0.11 * 0.14 = 1.02 + 0.84 + 0.24 + 0.28 = 0.9;$
- $S(\text{Cronometer}) = 0.11 * 0.34 + 0.13 * 0.28 + 0.13 * 0.24 + 0.11 * 0.14 = 1.02 + 0.84 + 0.48 + 0.28 = 0.12;$
- $S(\text{MyFitnessPal}) = 0.11 * 0.34 + 0.13 * 0.28 + 0.13 * 0.24 + 0.11 * 0.14 = 0.68 + 0.84 + 0.48 + 0.28 = 0.12;$
- $S(\text{Lifesum}) = 0.32 * 0.34 + 0.34 * 0.28 + 0.34 * 0.24 + 0.33 * 0.14 = 0.68 + 0.56 + 0.72 + 0.14 = 0.32.$

По результатам расчетов, наибольшую сумму весов набрал сервис «FatSecret», который был выбран прототипом для разрабатываемого приложения.

На основе анализа сервисов по планированию питания и контроля калорийности, были выделены главные аспекты, которые максимально соответствуют ежедневным задачам пользователей и будут учитываться при разработке приложения:

- сбор параметров пользователя (рост, вес, возраст, пол, цель), возможность их редактирования, автоматический расчёт индекса массы тела (ИМТ);
- расчёт дневной нормы КБЖУ (калории, белки, жиры, углеводы) и вывод готовых программ питания на день (завтрак, обед, ужин);
- график изменения веса – визуальное представление прогресса, построенное на основе введённых пользователем данных;
- калоризатор – инструмент, позволяющий пользователю вручную вносить съеденные блюда и автоматически учитывать их в статистике дневного рациона.

Реализация данных функций обеспечивает сбалансированный минимум, необходимый для контроля питания и массы тела без перегрузки лишним функционалом. Такой подход делает приложение простым в использовании и доступным для большой аудитории пользователей, заинтересованных в улучшении своего образа жизни.

1.4 Анализ функциональных требований

В результате анализа предметной области и существующих решений были сформулированы ключевые функциональные требования к разрабатываемому приложению:

- регистрация и авторизация – создание и вход в личный аккаунт пользователя;
- работа с профилем пользователя – ввод, редактирование и отображение параметров (рост, вес, возраст, пол, цель), автоматический расчет индекса массы тела (ИМТ);

- расчет и отображение дневной нормы КБЖУ – автоматическое определение суточной нормы калорий, белков, жиров и углеводов на основе введенных данных и цели;
- рекомендации по питанию – формирование программ питания с подробным описанием завтрака, обеда, ужина и других приемов пищи, просмотр информации о каждом блюде;
- график изменения веса – визуализация прогресса пользователя в виде графика изменения массы тела;
- калоризатор / пищевой дневник – ввод съеденных блюд вручную, автоматическое добавление информации в дневной рацион и статистику.

На основе перечисленных функций была построена диаграмма вариантов использования (Use Case Diagram), отражающая взаимодействие пользователя с системой и охватывающая все ключевые сценарии работы с приложением, указанная на рисунке 6.

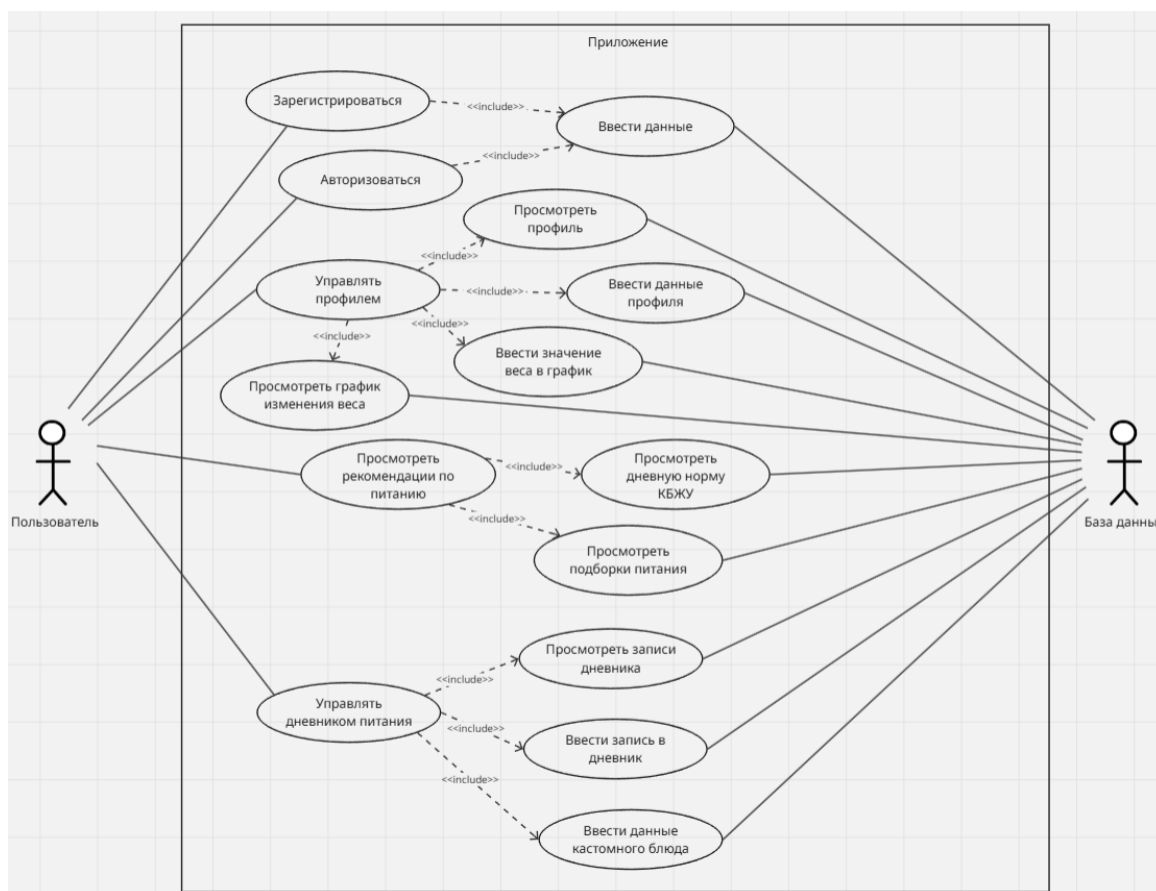


Рисунок 6 – Диаграмма вариантов использования

На рисунке 7 приведены спецификации варианта использования «Зарегистрироваться».

Прецедент: зарегистрироваться
ID: 1
Краткое описание: пользователь регистрируется в системе
Главные актеры: пользователь
Второстепенные актеры: нет
Предусловие: нет
Основной поток: 1. Пользователь вводит некорректные данные (некорректные символы, почта, длина пароля) 1.1. Система выводит предупреждение о некорректном вводе 2. Пользователь с введенной почтой существует 2.2. Система выводит предупреждение об уже существующей почте 3. Пользователь добавляется в систему
Постусловие: пользователь добавлен в базу данных
Альтернативные потоки: нет

Рисунок 7 – Сценарий выполнения варианта использования «Зарегистрироваться»

На рисунке 8 приведены спецификации варианта использования «Управлять профилем».

Прецедент: управлять профилем
ID: 2
Краткое описание: пользователь использует страницу профиля для записи/просмотра параметров, истории веса
Главные актеры: пользователь
Второстепенные актеры: нет
Предусловие: пользователь авторизован
Основной поток: 1. Пользователь вводит данные профиля 1.1. Система выводит индекс массы тела, основываясь на введенных данных 2. Пользователь нажимает кнопку “история веса” 2.2. Система выводит график истории веса 3. Пользователь записывает значение веса за указанную дату 2.2. Система выводит обновленный график
Постусловие: отображены данные, новые данные добавлены в базу данных
Альтернативные потоки: нет

Рисунок 8 – Сценарий выполнения варианта использования «Управлять профилем»

На рисунке 9 приведены спецификации варианта использования «Просмотреть рекомендации по питанию».

Прецедент: просмотреть рекомендации по питанию
ID: 3
Краткое описание: пользователь просматривает предложенные рационы питания и дневную норму КБЖУ
Главные актеры: пользователь
Второстепенные актеры: нет
Предусловие: пользователь авторизован
Основной поток: 1. Пользователь просматривает норму КБЖУ 1.1. Система по параметрам пользователя выводит дневную норму КБЖУ 2. Пользователь просматривает рационы питания 2.1. Система по параметрам пользователя выводит список подходящих рационов
Постусловие: отображены данные
Альтернативные потоки: параметры пользователя не введены, система предложит ввести параметры

Рисунок 9 – Сценарий выполнения варианта использования «Просмотреть рекомендации по питанию»

На рисунке 10 приведены спецификации варианта использования «Управлять дневником питания».

Прецедент: управлять дневником питания
ID: 4
Краткое описание: пользователь использует страницу дневника для просмотра/ввода записей о съеденных блюдах
Главные актеры: пользователь
Второстепенные актеры: нет
Предусловие: пользователь авторизован
Основной поток: 1. Пользователь просматривает записи дневника за указанную дату 1.1. Система выводит записи дневника 2. Пользователь нажимает кнопку “+” и вводит съеденное блюдо 2.1. Система выводит обновленные записи дневника 3. Пользователь нажимает кнопку “-” и выбирает блюдо на удаление 3.1. Система выводит обновленные записи дневника 4. Пользователь нажимает кнопку “Работать с личными блюдами” 4.1. Система выводит ранее записанные личные блюда пользователя 5. Пользователь нажимает кнопку “+” и вводит данные блюда 5.1. Система выводит обновленные записи личных блюд
Постусловие: отображены данные, новые данные добавлены в базу данных
Альтернативные потоки: нет

Рисунок 10 – Сценарий выполнения варианта использования «Просмотреть рекомендации по питанию»

Таким образом, были проанализированы функции будущего приложения.

1.5 Анализ нефункциональных требований

В качестве языка серверной части выбран Golang. Данный язык отличается высокой производительностью, простотой синтаксиса и отличной поддержкой параллелизма, что делает его особенно подходящим для построения легковесных и надежных веб-приложений. Также Go хорошо масштабируется и предоставляет множество готовых инструментов для разработки API.

Для хранения данных используется PostgreSQL – популярная объектно-реляционная СУБД, которая используется в большинстве современных проектов.

В качестве основной среды разработки для backend-части выбрана JetBrains Goland – специализированная IDE для Go, предоставляющая мощные инструменты для написания, отладки и тестирования кода.

Клиентская часть приложения разрабатывается с использованием HTML, JavaScript и библиотеки React, которая позволяет создавать динамичные и отзывчивые пользовательские интерфейсы, обеспечивая высокую скорость работы и удобство в сопровождении кода.

Такой стек технологий позволяет реализовать современное веб-приложение с интуитивно понятным интерфейсом.

2 Проектирование сервиса для подбора питания

2.1 Архитектура приложения

Разрабатываемое приложение построено на клиент-серверной архитектуре, обозначенной на рисунке 11.

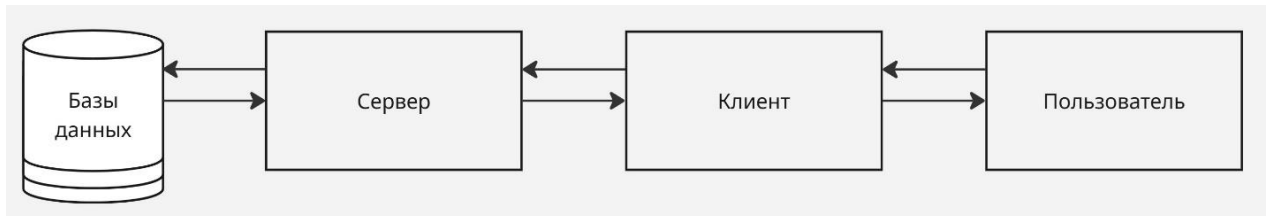


Рисунок 11 – Архитектура «клиент-сервер»

Пользователь взаимодействует только с клиентским интерфейсом, который, в свою очередь, обращается к серверу. Сервер обрабатывает запросы и обращается к базе данных, возвращая клиенту нужные данные в формате, пригодном для отображения.

2.2 Проектирование структур данных

2.2.1 Концептуальная модель базы данных

В соответствии с функциональными возможностями разрабатываемого приложения была составлена концептуальная модель базы данных, обозначенная на рисунке 12.

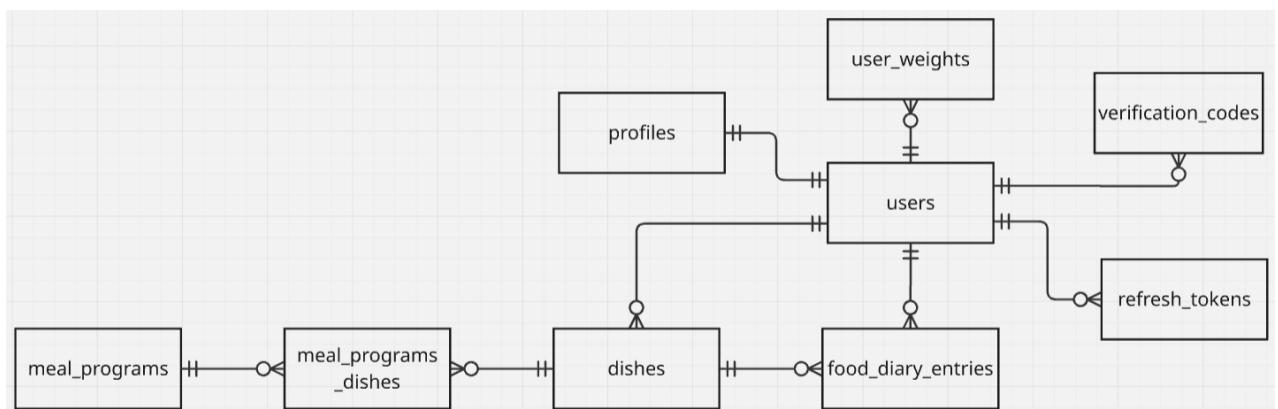


Рисунок 12 – Концептуальная модель базы данных

В таблице 14 указаны сущности, которые были обозначены в концептуальной модели базы данных.

Таблица 14 – Сущности базы данных

Название	Назначение
users	Данные пользователей
refresh_tokens	Обновляющие токены, используемые в механизме авторизации и аутентификации пользователей
profiles	Параметры и цели пользователей
user_weights	Отметки весов в заданные точки времени
meal_programs	Программы питания, состоящие из блюд
dishes	Сведения и описания блюд
meal_programs_dishes	Связующая таблица между программами питания и блюдами
food_diary_entries	Записи в пищевой дневник
verification_codes	Коды для подтверждения почты при регистрации

Таким образом была составлена начальная структура будущей базы данных сервиса.

2.3 Проектирование программных средств сервиса подбора питания

Спроектированы программные средства каждого микросервиса и составлены диаграммы классов.

Классы сервиса авторизации изображены на диаграмме классов в приложении Г.1.

Классы сервиса авторизации описаны в таблице 15.

Таблица 15 – Компоненты диаграммы классов сервиса авторизации

Класс	Назначение
App	Управляет жизненным циклом приложения
User, LoginInput, TokenPair, Claims, User, RefreshToken, VerifyEmailRequest, VerificationCode	Структуры для работы с данными сервиса

Продолжение таблицы 15

Класс	Назначение
MyContext	Обеспечивает единое хранилище для контекста приложения и логгера
Config	Хранит конфигурационные параметры приложения
Server	Отвечает за настройку и управление HTTP-сервером
ImplAuthorizationService	Отвечает за обработку бизнес-логики, связанной с регистрацией, авторизацией и обновлением токенов
AuthorizationPostgres	Отвечает за работу с базой данных
AuthorizationService	Интерфейс для обозначения методов сервиса
AuthorizationRepository	Интерфейс для обозначения методов репозитория

Классы сервиса профиля изображены на диаграмме классов в приложении Г.2.

Классы сервиса питания описаны в таблице 16.

Таблица 16 – Компоненты диаграммы классов сервиса питания

Класс	Назначение
App	Управляет жизненным циклом приложения
Profile, CreateUserWeightRequest, UserWeight	Структуры для работы с данными сервиса
MyContext	Обеспечивает единое хранилище для контекста приложения и логгера
Config	Хранит конфигурационные параметры приложения
Server	Отвечает за настройку и управление HTTP-сервером
ImplProfileService	Отвечает за обработку бизнес-логики, связанной с созданием, получением, обновлением профилей
ProfilePostgres	Отвечает за работу с базой данных
ProfileService	Интерфейс для обозначения методов сервиса
ProfileRepository	Интерфейс для обозначения методов репозитория

Классы сервиса питания изображены на диаграмме классов в приложении Г.3.

Классы сервиса питания описаны в таблице 17.

Таблица 17 – Компоненты диаграммы классов сервиса питания

Класс	Назначение
App	Управляет жизненным циклом приложения
Profile, Plan, Dish, MealProgram, ProgramDish, DishSearchResult, UserDish	Структуры для работы с данными сервиса
MyContext	Обеспечивает единое хранилище для контекста приложения и логгера
Config	Отвечает за хранение конфигурационных параметров приложения
Server	Отвечает за настройку и управление HTTP-сервером
ImplNutritionService	Отвечает за обработку бизнес-логики, связанной с созданием рекомендаций по питанию
NutritionPostgres	Отвечает за работу с базой данных
NutritionService	Интерфейс для обозначения методов сервиса
NutritionRepository	Интерфейс для обозначения методов репозитория

Классы сервиса дневника изображены на диаграмме классов в приложении Г.4.

Классы сервиса дневника описаны в таблице 18.

Таблица 18 – Компоненты диаграммы классов сервиса дневника

Класс	Назначение
App	Управляет жизненным циклом приложения
DiaryEntry, DiaryEntryWithDish, Dish	Структуры для работы с данными сервиса
MyContext	Обеспечивает единое хранилище для контекста приложения и логгера
Config	Отвечает за хранение конфигурационных параметров приложения
Server	Отвечает за настройку и управление HTTP-сервером

Продолжение таблицы 18

ImplDiaryService	Отвечает за обработку бизнес-логики, связанной с созданием рекомендаций по питанию
DiaryPostgres	Отвечает за работу с базой данных
DiaryService	Интерфейс для обозначения методов сервиса
DiaryRepository	Интерфейс для обозначения методов репозитория

Благодаря представленным диаграммам можно просмотреть все детали структур приложения.

2.4 Проектирование пользовательского интерфейса

В соответствии с функциональными возможностями приложения были спроектированы макеты страниц пользовательского интерфейса, обеспечивающие удобную и наглядную работу с системой. Интерфейс включает следующие основные страницы: авторизация, профиль, подборки питания и дневник питания.

Страница авторизации, где реализованы формы регистрации и входа в систему. Поля для ввода: электронная почта, юзернейм и пароль. Кнопки: регистрация и вход.

Макет страницы авторизации продемонстрирован на рисунке 13.

Логин/Регистрация

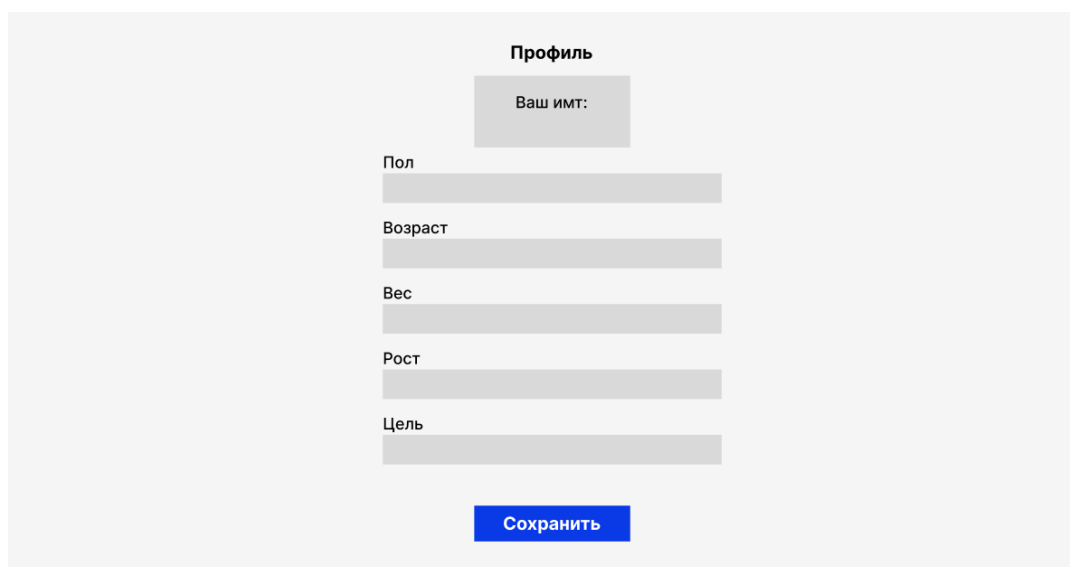
Зарегистрироваться
Войти

Рисунок 13 – Макет страницы авторизации

Страница профиля, на которой пользователь может вводить и редактировать персональные данные (рост, вес, возраст, пол, цель),

просматривать индекс массы тела (ИМТ) и график изменения веса. Поля для ввода: пол, возраст, вес, рост, цель. Кнопка сохранения профиля.

Макет продемонстрирован на рисунке 14.

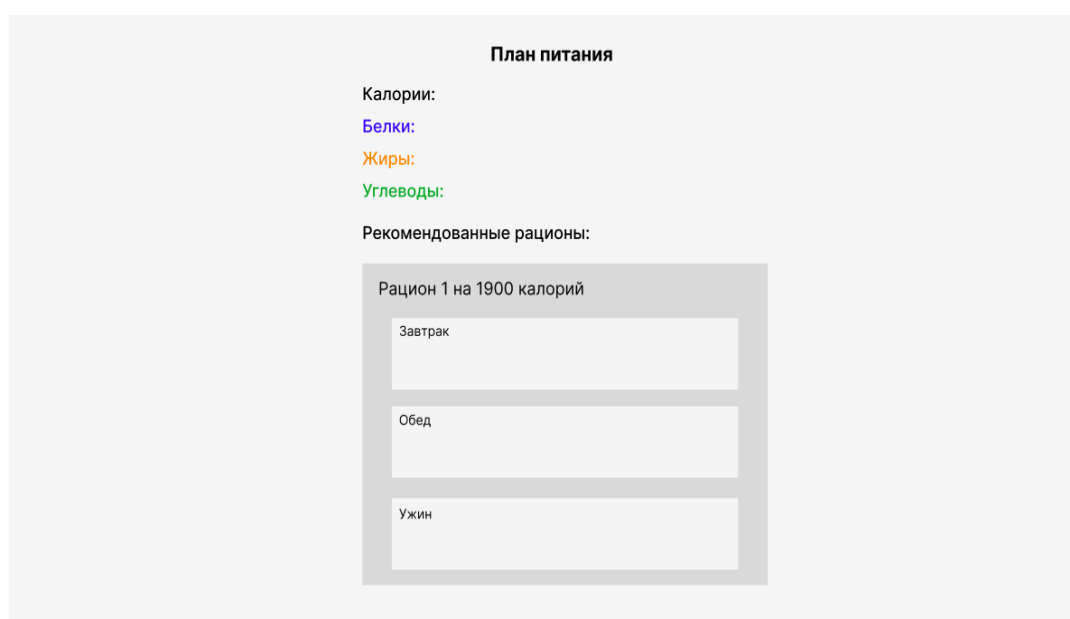


The screenshot shows a web form titled "Профиль" (Profile). At the top, there is a label "Ваш имт:" (Your BMI) next to a light gray input field. Below this, there are five labels with corresponding input fields: "Пол" (Gender), "Возраст" (Age), "Вес" (Weight), "Рост" (Height), and "Цель" (Goal). At the bottom of the form is a blue button with the text "Сохранить" (Save).

Рисунок 14 – Макет страницы профиля

Страница питания, отображающая рассчитанную дневную норму КБЖУ и подборки рекомендованных программ питания с разбивкой по приёмам пищи (завтрак, обед, ужин и т. д.). Информация на странице: дневная норма КБЖУ, список программ питания.

Макет продемонстрирован на рисунке 15.



The screenshot shows a web page titled "План питания" (Nutrition Plan). It lists the following information: "Калории:" (Calories), "Белки:" (Proteins), "Жиры:" (Fats), and "Углеводы:" (Carbohydrates). Below this, it says "Рекомендованные рационы:" (Recommended rations). A box titled "Рацион 1 на 1900 калорий" (Ration 1 on 1900 calories) contains three input fields labeled "Завтрак" (Breakfast), "Обед" (Lunch), and "Ужин" (Dinner).

Рисунок 15 – Макет страницы питания

Страница дневника питания, содержащая калоризатор – инструмент для ввода съеденных блюд, с возможностью сохранения данных и просмотра статистики по рациону. Кнопки: добавить блюдо, удалить блюдо, ввести свое блюдо.

Макет продемонстрирован на рисунке 16.

The mockup shows a web interface for a nutrition diary. At the top, it says 'Дневник питания' (Nutrition Diary) and '< 10 may >'. Below this are four labels: 'Калории:' (Calories), 'Белки:' (Proteins), 'Жиры:' (Fats), and 'Углеводы:' (Carbohydrates). The main area contains three input fields labeled 'Завтрак' (Breakfast), 'Обед' (Lunch), and 'Ужин' (Dinner). To the right of these fields are two buttons: a red minus button and a green plus button. Below the input fields is a section titled 'Список ваших блюд' (List of your dishes) containing two items: 'Блюдо 1' (Dish 1) and 'Блюдо 2' (Dish 2), each with a red minus button. At the bottom is a green button labeled 'СОЗДАТЬ' (Create).

Рисунок 16 – Макет страницы дневника питания

Таким образом был смоделирован удобный и понятный интерфейс.

3 Реализация сервиса подбора питания

3.1 Реализация базы данных

3.1.1 Физическая модель базы данных

На основе логической модели базы данных, была создана физическая модель, в которой подробно описаны все таблицы, их атрибуты и соответствующие типы данных. Модель указана на рисунке 17.

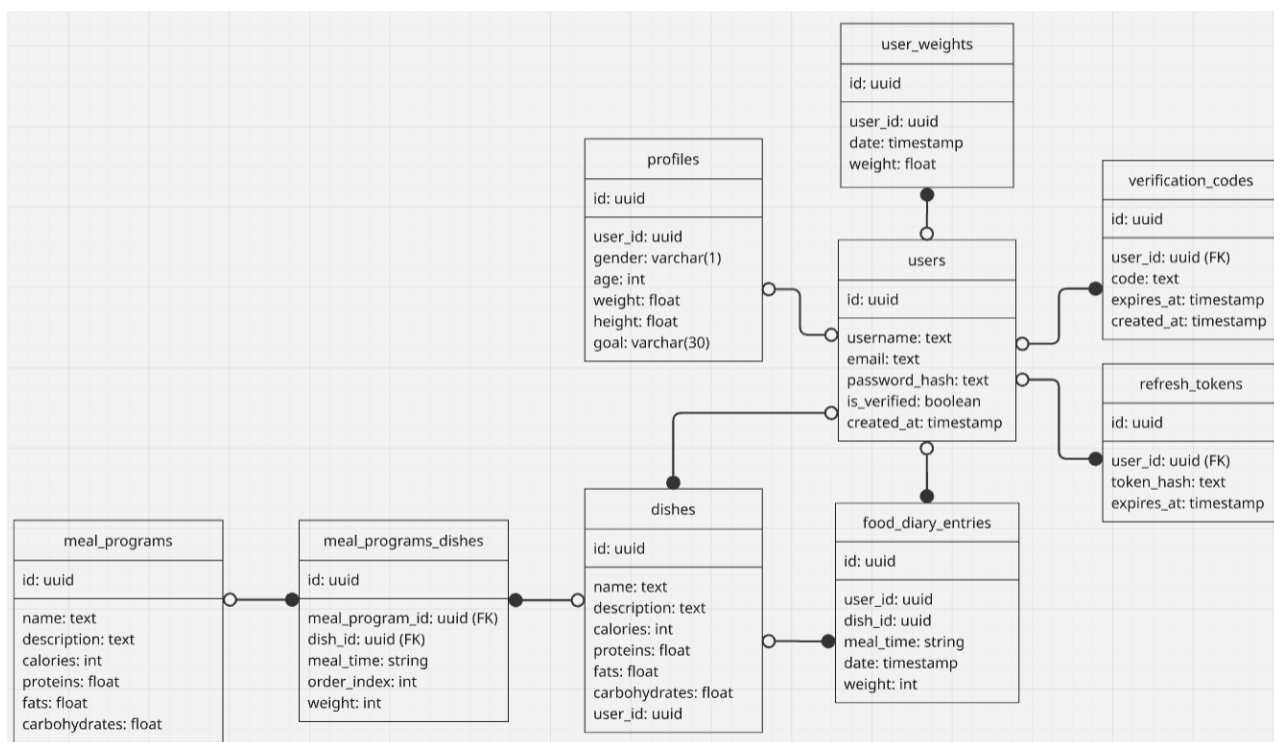


Рисунок 17 – Физическая модель базы данных

Описание атрибутов сущности «Users» представлено в таблице 19.

Таблица 19 – Описание атрибутов сущности «users»

Название	Описание	Тип данных	Возможность NULL	Значение по умолчанию	Ключ
id	Идентификатор	uuid	Нет	—	PK
username	Юзернейм пользователя	text	Нет	—	—
email	Эл. почта пользователя	text	Нет	—	—
password_hash	Хэш пароля пользователя	text	Нет	—	—

Продолжение таблицы 19

Название	Описание	Тип данных	Возможность NULL	Значение по умолчанию	Ключ
is_verified	Верифицированность пользователя	boolean	Нет	—	—
created_at	Время создания записи	timestamp	Нет	current_timestamp	—

Описание атрибутов сущности «user_weights» представлено в таблице 20.

Таблица 20 – Описание атрибутов сущности «user_weights»

Название	Описание	Тип данных	Возможность NULL	Значение по умолчанию	Ключ
id	Идентификатор	uuid	Нет	—	PK
user_id	Идентификатор пользователя	uuid	Нет	—	FK
date	Дата взвешивания	timestamp	Нет	—	—
weight	Значение веса	float	Нет	—	—

Описание атрибутов сущности «profiles» представлено в таблице 21.

Таблица 21 – Описание атрибутов сущности «profiles»

Название	Описание	Тип данных	Возможность NULL	Значение по умолчанию	Ключ
id	Идентификатор	uuid	Нет	—	PK
user_id	Идентификатор пользователя	uuid	Нет	—	—
gender	Гендер пользователя	varchar(1)	Нет	—	—
age	Возраст пользователя	int	Нет	—	—
weight	Вес пользователя	float	Нет	—	—
height	Рост пользователя	float	Нет	—	—
goal	Цель пользователя	varchar(30)	Нет	—	—

Описание атрибутов сущности «verification_codes» представлено в таблице 22.

Таблица 22 – Описание атрибутов сущности «verification_codes»

Название	Описание	Тип данных	Возможность NULL	Значение по умолчанию	Ключ
id	Идентификатор	uuid	Нет	—	PK
user_id	Идентификатор пользователя	uuid	Нет	—	FK
code	Значение кода подтверждения	text	Нет	—	—
expires_at	Время, когда код станет неактуальным	timestamp	Нет	—	—
created_at	Время создания кода	timestamp	Нет	current_timestamp	—

Описание атрибутов сущности «refresh_tokens» представлено в таблице 23.

Таблица 23 – Описание атрибутов сущности «refresh_tokens»

Название	Описание	Тип данных	Возможность NULL	Значение по умолчанию	Ключ
id	Идентификатор	uuid	Нет	—	PK
user_id	Идентификатор пользователя	uuid	Нет	—	FK
token_hash	Хэш токена обновления	text	Нет	—	—
expires_at	Время, когда токен станет неактуальным	timestamp	Нет	—	—

Описание атрибутов сущности «dishes» представлено в таблице 24.

Таблица 24 – Описание атрибутов сущности «dishes»

Название	Описание	Тип данных	Возможность NULL	Значение по умолчанию	Ключ
id	Идентификатор	uuid	Нет	—	PK
name	Название блюда	text	Нет	—	—
description	Описание блюда	text	Нет	—	—
calories	Калорийность блюда	int	Нет	—	—
proteins	Содержание белков в блюде	float	Нет	—	—

Продолжение таблицы 24

Название	Описание	Тип данных	Возможность NULL	Значение по умолчанию	Ключ
fats	Содержание жиров в блюде	float	Нет	—	—
carbohydrates	Содержание углеводов в блюде	float	Нет	—	—
user_id	Идентификатор пользователя	uuid	Да	—	—

Описание атрибутов сущности «food_diary_entries» представлено в таблице 25.

Таблица 25 – Описание атрибутов сущности «food_diary_entries»

Название	Описание	Тип данных	Возможность NULL	Значение по умолчанию	Ключ
id	Идентификатор	uuid	Нет	—	PK
user_id	Юзернейм пользователя	text	Нет	—	—
dish_id	Идентификатор блюда	text	Нет	—	—
meal_time	Время приема пищи	text	Нет	—	—
date	Дата приема блюда	boolean	Нет	—	—
weight	Вес порции	timestamp	Нет	—	—

Описание атрибутов сущности «meal_programs» представлено в таблице 26.

Таблица 26 – Описание атрибутов сущности «meal_programs»

Название	Описание	Тип данных	Возможность NULL	Значение по умолчанию	Ключ
id	Идентификатор	uuid	Нет	—	PK
name	Название рациона	text	Нет	—	—
description	Описание рациона	text	Нет	—	—
calories	Общая калорийность рациона	int	Нет	—	—

Продолжение таблицы 26

Название	Описание	Тип данных	Возможность NULL	Значение по умолчанию	Ключ
proteins	Общее содержание белков рациона	float	Нет	—	—
fats	Общее содержание жиров рациона	float	Нет	—	—
carbohydrates	Общее содержание углеводов рациона	float	Нет	—	—

Описание атрибутов сущности «meal_programs_dishes» представлено в таблице 27.

Таблица 27 – Описание атрибутов сущности «meal_programs_dishes»

Название	Описание	Тип данных	Возможность NULL	Значение по умолчанию	Ключ
id	Идентификатор	uuid	Нет	—	PK
meal_program_id	Идентификатор рациона	uuid	Нет	—	FK
dish_id	Идентификатор блюда	uuid	Нет	—	FK
meal_time	Время приема пищи	string	Нет	—	—
order_index	Место в списке	int	Нет	—	—
weight	Вес порции	int	Нет	—	—

На основе физической модели была построена база данных, используя СУБД PostgreSQL.

Для управления версиями базы данных, используется механизм миграций. Каждая миграция – это SQL-скрипт, который описывает изменения структуры БД или вставку начальных данных.

В данном проекте миграции используются для каждого микросервиса отдельно, скрипты описаны в приложении С.

Для взаимодействия с базами данных, были написаны SQL-запросы, прикрепленные в приложении D.

3.1.2 Заполнение базы данных тестовыми данными

Для заполнения тестовыми данными были написаны миграции для базы сервиса питания, введен обширный набор данных продуктов, блюд и рационов питания.

В таблицу «dishes» записаны данные более 800 продуктов и блюд (рисунок 18).

```
dishes=# select count(*) from dishes;
count
-----
    805
(1 row)
```

```
dishes=# select * from dishes limit 20;
```

id	name	description	calories	proteins	fats	carbohydrates	user_id
dd89157f-fd90-497e-b486-13c5b72b9c3	Гречка отварная	Гречневая каша на воде без масла	101	3.6	0.7	20	
de478a38-a259-48a1-a3e3-fb785c56b887	Рис белый отварной	Белый рис вареный без добавок	130	2.7	0.3	28	
2241cf7f-361d-41f6-9549-a3d8196e5834	Рис бурый отварной	Бурый рис вареный без масла	111	2.6	0.9	23	
7892a0e4-ef45-4024-8a09-cfe2f00a0485	Рис басмати отварной	Рис басмати вареный	121	3	0.4	25.2	
a97b081a-bb93-4856-a975-2bdee6c33735	Овсянка на воде	Овсяная каша без сахара и масла	88	3	1.7	15	
fea5b87c-bb9c-4e0b-86f9-1bfab93e65b2	Герловка отварная	Герловая крупа на воде	109	3.1	0.4	22.2	
08fd47d0-1f6a-4e12-9111-419b3439b2a1	Пшеница каша	Пшено вареное на воде	90	3	1	17	
112878f2-26bc-4d49-8a70-c51b1fc1f2b5	Кукурузная каша	Каша из кукурузной крупы	86	2.3	0.4	19.6	
a94981ac-b202-475c-8ab9-9a747f70f075	Киноа отварная	Киноа приготовленная на воде	120	4.4	1.9	21.3	
9b68100f-1b74-432f-a60d-214f411ab012	Булгур отварной	Булгур вареный без масла	83	3.1	0.2	18.6	
554fdd1b-526c-40f7-b225-338a0d86f861	Янчевая каша	Янчевая крупа на воде	76	2.3	0.3	15.7	
b64d98cc-c47c-492d-b4fe-2f2c8a45a4a7	Полба отварная	Каша из полбы	127	5.5	0.9	26.4	
3019ce10-d8f6-4047-9753-311f43621f19	Чечевича отварная	Чечевича вареная на воде	116	9	0.4	20.1	
fe4be9b3-13bb-4ddb-bc42-11a62b312784	Нут отварной	Нут вареный без добавок	164	8.9	2.6	27.4	
68938dd-e145-4fcf-92b2-f08567fd9f09	Фасоль отварная	Фасоль вареная без масла	127	8.7	0.5	22.8	
fb586b9a-bfff-41c3-91e7-9ac2b4db4fe1	Макарон отварные	Макарон из твердых сортов пшеницы	160	5.5	1.1	30	
131ce1e7-8b37-4a3c-87be-c8f9ded594bc	Спагетти аль денте	Спагетти вареные до состояния аль денте	158	5.8	1.1	30.5	
02c220fb-74cb-497f-b195-805fcae6088b	Лапша гречневая	Гречневая лапша (соба)	99	5	0.1	21.4	

Рисунок 18 – Заполненная таблица «dishes»

В таблицу «meal_programs» записаны данные более 70 рационов питания (рисунок 19).

```
dishes=# select count(*) from meal_programs;
count
-----
    74
(1 row)
```

```
dishes=# select name, calories, proteins, fats, carbohydrates from meal_programs limit 20;
```

name	calories	proteins	fats	carbohydrates
Рацион 3 на 2800 ккал	2898	209.11999999999998	136.32	190.22
Вегетарианский рацион	1037	46.47	14.280000000000001	176.45
Рацион 2 на 1500 ккал	1523	150.1	58.5	87.75
Рацион 1 на 3600 ккал	3648	245.94	144.03000000000003	314.92
Рацион 4 на 1800 ккал	1841	156.7	30.049999999999997	219.8
Рацион 5 на 1400 ккал	1400	128.15	45.05	112.85
Средиземноморский рацион	840	84.46000000000001	29.240000000000002	55.040000000000006
Рацион 2 на 3100 ккал	3105	182.89000000000001	143.58000000000004	258.64
Рацион 2 на 3200 ккал	3233	216.34	151.73000000000002	232.42000000000002
Рацион 2 на 2000 ккал	2075	150.45000000000002	59.99999999999999	224.70000000000002
Рацион 3 на 2500 ккал	2512	200.95	108.85	153.8
Рацион 1 на 2700 ккал	2775	169.95000000000002	120.55000000000001	237.85000000000002
Классический сбалансированный	725	83.13	15.200000000000001	59.705000000000005
Сытный рацион	1930	144.85	76.25	153.5

Рисунок 19 – Заполненная таблица «meal_programs»

Введенных данных достаточно для представления работы и проведения тестов приложения.

3.2 Реализация серверной части приложения

Серверная часть приложения реализована на основе микросервисной архитектуры и написана на языке Golang, используя среду программирования JetBrains Goland. Для взаимодействия с базой данных использовалась библиотека SQLX, расширяющая функционал стандартной библиотеки. Компоненты сервиса описаны в диаграмме компонентов (рисунок 20).

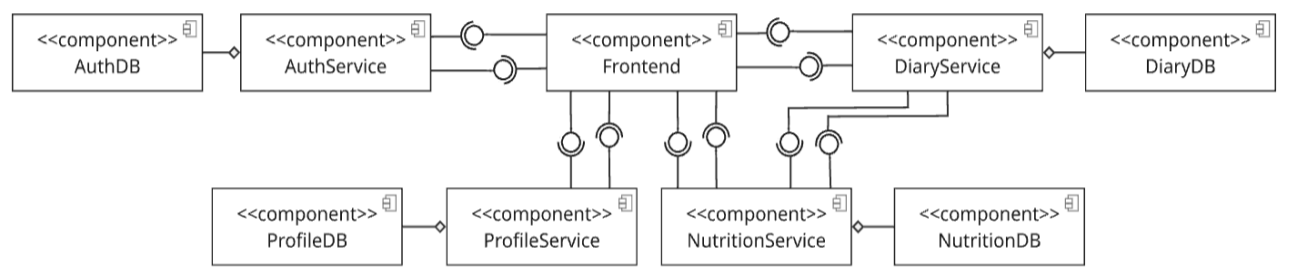


Рисунок 20 – Диаграмма компонентов

Взаимодействие между клиентской частью приложения и микросервисами, а также между самими микросервисами, осуществляется с помощью REST API, обеспечивая стандартизированный подход к обмену данными с помощью HTTP-запросов (GET, POST, PUT, DELETE).

Эндпоинты, предоставляющие доступ к серверной части приложения, описаны в таблице 28.

Таблица 28 – Описание эндпоинтов сервиса

№	URI	Тип HTTP-запроса	Описание
1	/api/register/	POST	Регистрация пользователя
2	/api/login/	POST	Авторизация пользователя
3	/api/refresh/	POST	Выдача новой пары JWT-токенов
4	/api/profile/	POST	Создание нового профиля пользователя

Продолжение таблицы 28

№	URI	Тип HTTP-запроса	Описание
5	/api/profile/	GET	Получение данных профиля пользователя
6	/api/profile/	PUT	Изменение данных профиля пользователя
7	/api/nutrition/	POST	Составление дневной нормы КБЖУ пользователя и подбор рационов питания.
8	/api/nutrition/dishes/query-by-ids/	POST	Получение данных блюд по списку их идентификаторов
9	/api/nutrition/dishes/search/	GET	Получение списка блюд, подходящих по названию
10	/api/nutrition/dishes/	GET	Получение списка блюд пользователя
11	/api/nutrition/dishes/	POST	Создание нового блюда
12	/api/nutrition/dishes/	DELETE	Удаление блюда пользователя
13	/api/diary/entries/	GET	Получение записей дневника пользователя за указанную дату
14	/api/diary/entries/	POST	Создание новой записи дневника
15	/api/diary/entries/	DELETE	Удаление записи из дневника

Каждый микросервис в приложении разработан с использованием многослойной архитектуры для разделения зон ответственности обработчиков, бизнес-логики и работы с БД.

Сервисы и базы данных были развернуты в изолированных окружениях с помощью Docker [15, 19]. Основным преимуществом Docker является его способность упаковывать приложения и все их зависимости в легковесные переносимые контейнеры. Это гарантирует, что приложение будет работать стабильно, независимо от того, на какой машине оно запущено.

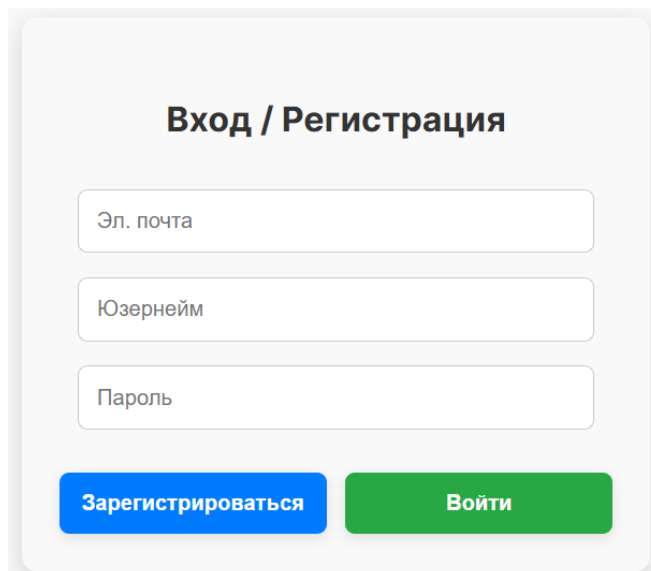
Все конфигурации Docker-файлов и DockerCompose-файла прикреплены в приложении Е.

3.3 Реализация клиентской части приложения

На основе макетов интерфейса приложения были написаны страницы WEB-приложения, обеспечивающие взаимодействие с серверной частью.

Клиентская часть реализована с использованием библиотеки React, позволяющей создавать интерактивные интерфейсы. Приложение состоит из четырех ключевых страниц.

Интерфейс страницы авторизации изображен на рисунке 21.



Вход / Регистрация

Эл. почта

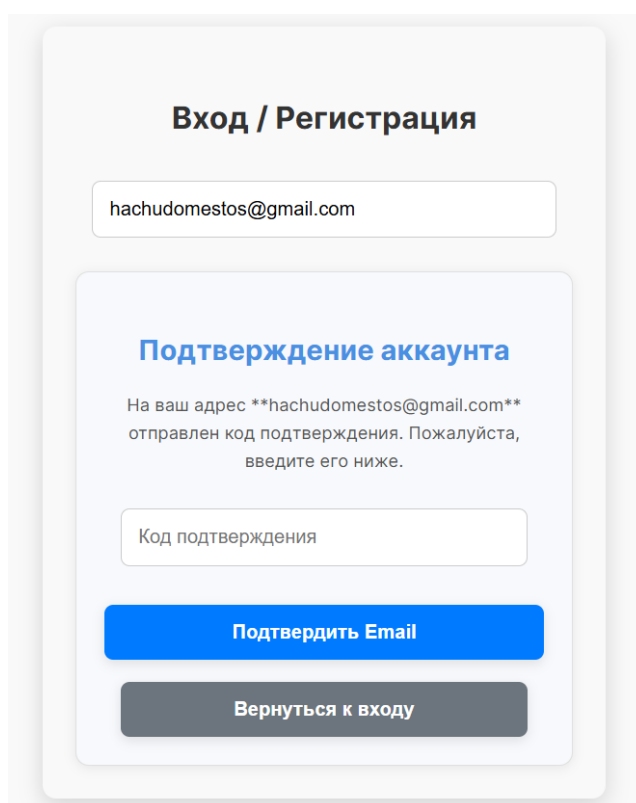
Юзернейм

Пароль

Зарегистрироваться **Войти**

Рисунок 21 – Интерфейс страницы авторизации

Интерфейс формы подтверждения почты изображен на рисунке 22.



Вход / Регистрация

hachudomestos@gmail.com

Подтверждение аккаунта

На ваш адрес **hachudomestos@gmail.com**
отправлен код подтверждения. Пожалуйста,
введите его ниже.

Код подтверждения

Подтвердить Email

Вернуться к входу

Рисунок 22 – Интерфейс формы подтверждения почты

Интерфейс страницы профиля изображен на рисунке 23.

The screenshot shows a user profile page with a header containing three buttons: "Профиль" (Profile), "Питание" (Nutrition), and "Дневник" (Journal). Below the header is the title "Страница профиля" (Profile page). The main content area contains a box with "ИМТ: 24.7" and a green bar indicating "Нормальный вес" (Normal weight). Below this is a form with fields for "Пол:" (Gender) set to "Мужчина" (Male), "Возраст:" (Age) set to "20", "Вес (кг):" (Weight in kg) set to "80", "Рост (см):" (Height in cm) set to "180", and "Цель:" (Goal) set to "Поддержание веса" (Maintain weight). A blue button "Сохранить профиль" (Save profile) is at the bottom of the form. Below the form is a blue button "История моего веса" (My weight history).

Рисунок 23 – Интерфейс страницы профиля

Интерфейс формы истории веса пользователя изображен на рисунке 24.

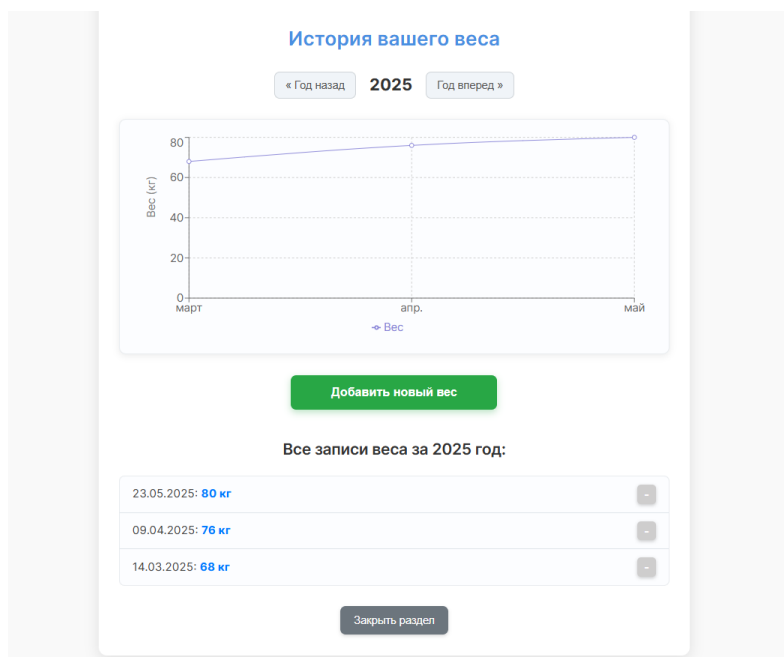


Рисунок 24 – Интерфейс формы истории веса пользователя

Интерфейс страницы питания изображен на рисунке 25.

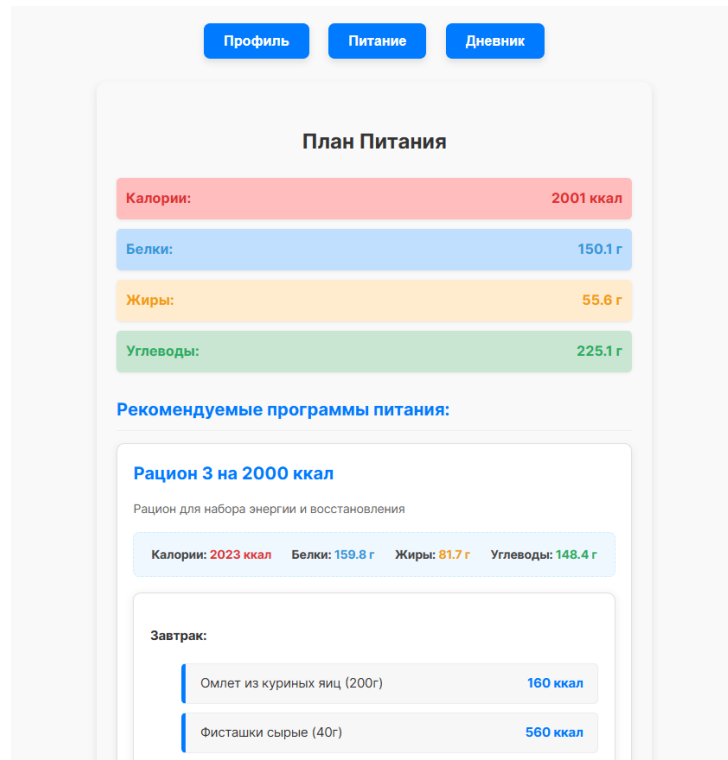


Рисунок 25 – Интерфейс страницы питания

Интерфейс страницы дневника изображен на рисунке 26.

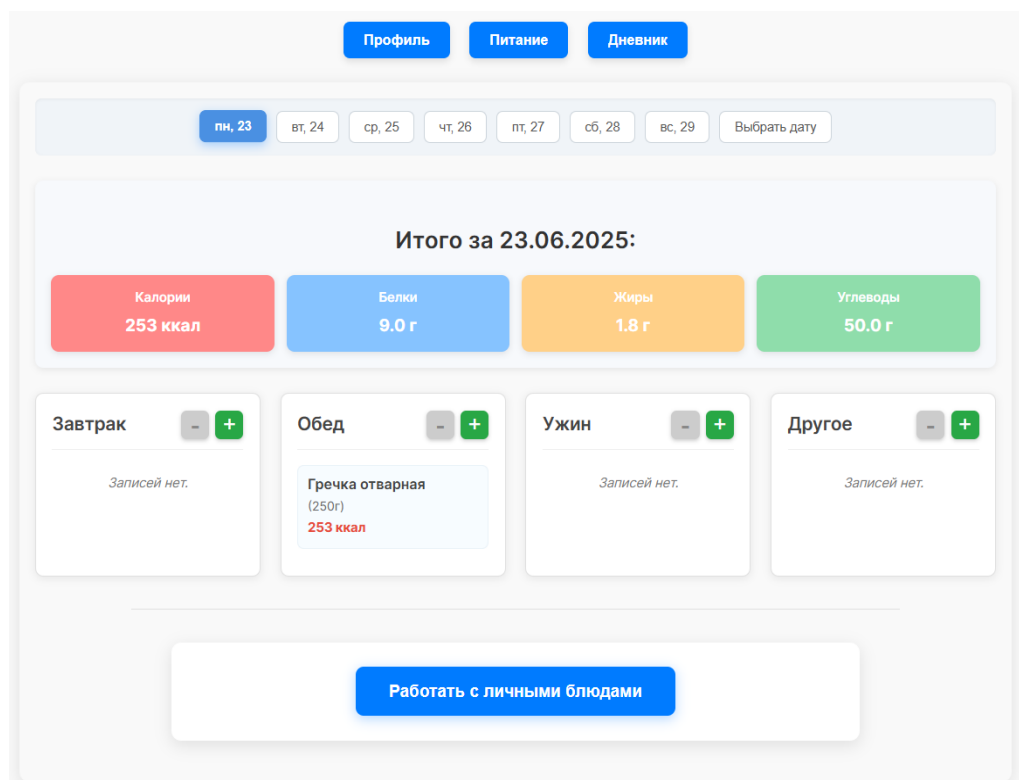


Рисунок 26 – Интерфейс страницы дневника

Интерфейс формы ввода съеденного блюда изображен на рисунке 27.

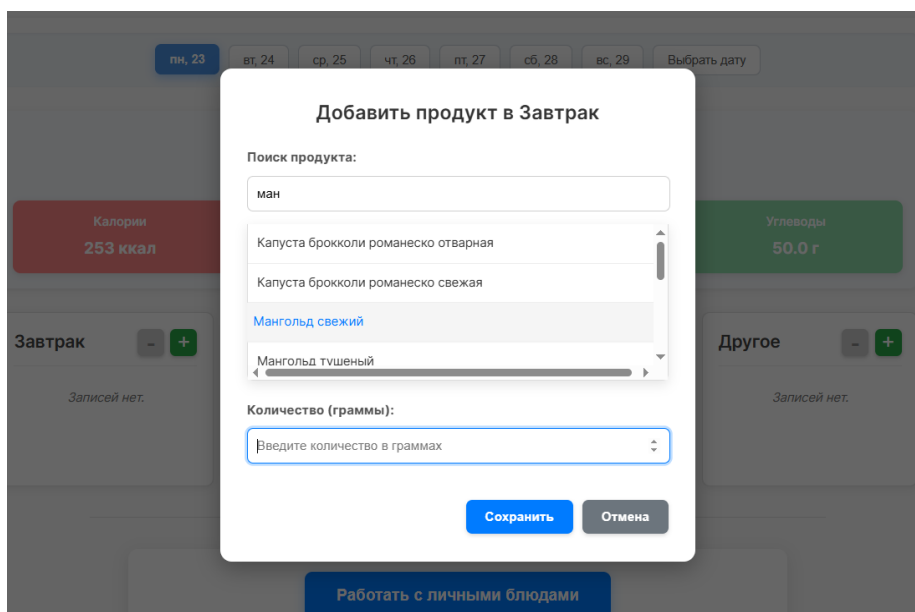


Рисунок 27 – Интерфейс формы ввода съеденного блюда

Интерфейс формы работы с личными блюдами пользователя изображен на рисунке 28.

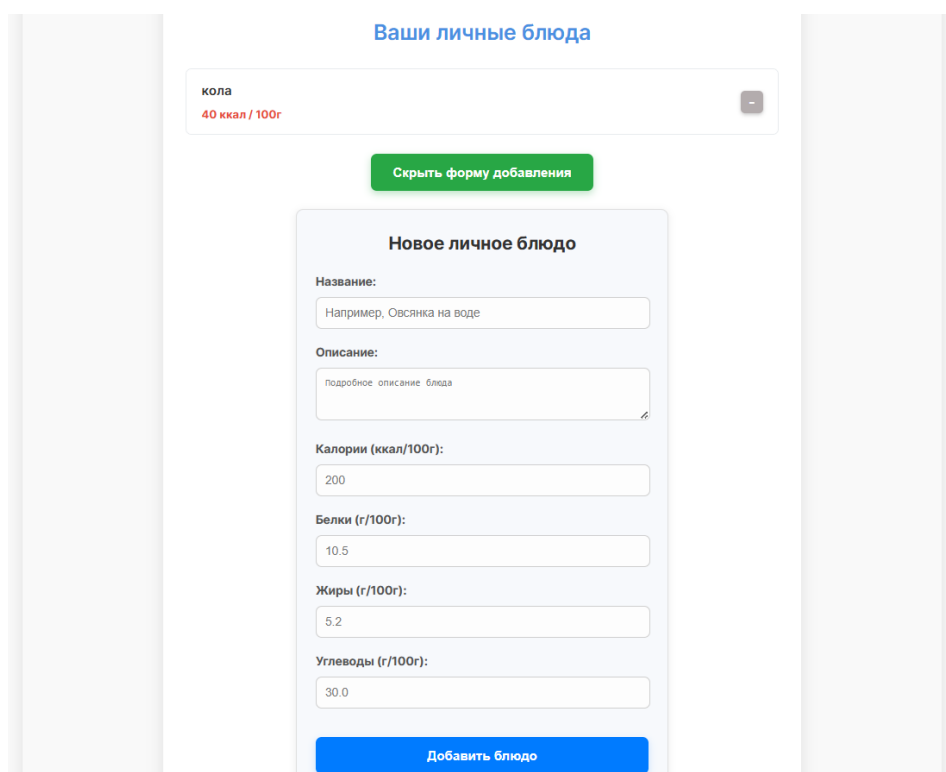


Рисунок 28 – Интерфейс формы работы с личными блюдами пользователя

Таким образом был разработан современный и удобный интерфейс, который обеспечивает интуитивно понятное взаимодействие пользователя с интерфейсом.

4 Контроль качества программного обеспечения

4.1 Выбор режима тестирования

Для проверки корректности работы системы было выбрано функциональное тестирование. Этот подход используется для проверки соответствия разработанного приложения ранее определенным требованиям. То есть было проверено, что каждая функция приложения работает так, как было задумано.

Главным преимуществом этого метода является то, что он имитирует реальное использование приложения конечным пользователем, что помогает выявить проблемы, которые напрямую влияют на пользовательский опыт взаимодействия с приложением.

4.2 Функциональное тестирование

Было произведено функциональное тестирование процесса регистрации пользователя, а именно – валидация электронной почты, юзернейма и пароля.

Адрес электронной почты должен быть корректного формата и содержать:

- имя почты, состоящее из латинских букв (до символа @);
- символ @;
- доменное имя (например, mail.ru).

Юзернейм должен быть длиной от 3 до 20 символов и состоять из латинских букв, цифр или символа «_».

Пароль должен быть длиной от 6 до 30 символов и состоять из:

- латинских букв, цифр или специальных символов;
- хотя бы одной заглавной буквы и хотя бы одной цифры.

В таблице 29 выписаны причины, действия пользователя при регистрации.

Таблица 29 – Причины

Обозначение	Описание
П1	Корректный формат электронной почты
П2	Длина юзернейма от 3 до 20 символов

Продолжение таблицы 29

Обозначение	Описание
ПЗ	Юзернейм состоит из латинских букв (a – z, A – Z), цифр (0 – 9) или символа «_»
П4	Длина пароля от 6 до 30 символов
П5	Пароль состоит из латинских букв (a – z, A – Z), цифр (0 – 9) или символов (!, *, +, # и т. д.)
П6	Пароль содержит хотя бы одну заглавную букву (A – Z) и хотя бы одну цифру (0 – 9)

В таблице 30 выписаны следствия, ожидаемые поведения системы, если причины выполнены.

Таблица 30 – Следствия

Обозначение	Описание
С1	Сообщение «некорректный формат электронной почты»
С2	Сообщение «длина юзернейма от 3 символов»
С3	Сообщение «недопустимые символы в юзернейме»
С4	Сообщение «длина пароля от 6 символов»
С5	Сообщение «недопустимые символы в пароле»
С6	Сообщение «пароль должен содержать заглавную букву и цифру»
С7	Пользователь зарегистрирован в системе

Для демонстрации процесса авторизации в сервисе, была построена функциональная диаграмма (рисунок 29), где наглядно отображено взаимодействие пользователя с системой.

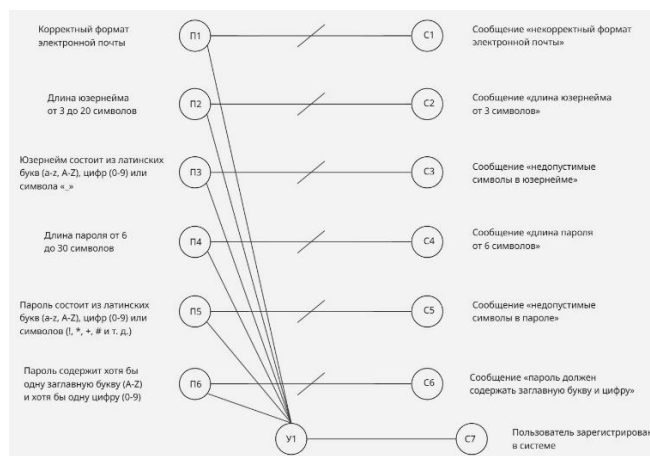


Рисунок 29 – Функциональная диаграмма регистрации пользователя

Для проверки корректности формы регистрации в приложении были составлены функциональные тесты в таблице 31.

Таблица 31 – Функциональные тесты регистрации пользователя

№	Эл. почта	Юзер нейм	Пароль	Ожидаемый результат	Фактический результат
1	mail@mail.ru	user	Qwerty1+	Пользователь зарегистрирован	Пользователь зарегистрирован
2	mailmail.ru	user	Qwerty1+	Сообщение «некорректный формат электронной почты»	Сообщение «некорректный формат электронной почты»
3	@mail.ru	user	Qwerty1+	Сообщение «некорректный формат электронной почты»	Сообщение «некорректный формат электронной почты»
4	mail@	user	Qwerty1+	Сообщение «некорректный формат электронной почты»	Сообщение «некорректный формат электронной почты»
5	mail@mail.ru	u	Qwerty1+	Сообщение «длина юзернейма от 3 символов»	Сообщение «длина юзернейма от 3 символов»
6	mail@mail.ru	user+	Qwerty1+	Сообщение «недопустимые символы в юзернейме»	Сообщение «недопустимые символы в юзернейме»
7	mail@mail.ru	user	Qwe1+	Сообщение «длина пароля от 6 символов»	Сообщение «длина пароля от 6 символов»
8	mail@mail.ru	user	Qwepw1+	Сообщение «недопустимые символы в пароле»	Сообщение «недопустимые символы в пароле»
9	mail@mail.ru	user	Qwerty	Сообщение «пароль должен содержать заглавную букву и цифру»	Сообщение «пароль должен содержать заглавную букву и цифру»

Также, были проведены тесты всех остальных функций приложения, которые описаны в таблице 32.

Таблица 32 – Функциональные тесты основных функций приложения

№	Действие	Данные запроса	Данные ответа	Ожидаемый результат	Фактический результат
1	Запись параметров пользователя	Параметры пользователя: пол, возраст, вес, рост, цель	Статус ОК	Данные пользователя записаны в базу данных, сервер вернул статус 200	Данные пользователя записаны в базу данных, сервер вернул статус 200 (приложения Д.1, Д.2, Д.3)

Продолжение таблицы 32

№	Действие	Данные запроса	Данные ответа	Ожидаемый результат	Фактический результат
2	Запись веса пользователя за указанную дату	Дата, значение веса	Статус ОК	Значение веса записано в базу данных, сервер вернул статус 200	Значение веса записано в базу данных, сервер вернул статус 200 (приложения Д.4, Д.5, Д.6)
3	Удаление значения веса пользователя	Идентификатор записи веса	Статус ОК	Значение веса удалено из базы данных, сервер вернул статус 200	Значение веса удалено из базы данных, сервер вернул статус 200 (приложения Д.7, Д.8, Д.9)
4	Запись в дневник	Идентификатор блюда, граммовки порции	Статус ОК	Запись дневника введена в базу данных, сервер вернул статус 200	Запись дневника введена в базу данных, сервер вернул статус 200 (приложения Д.10, Д.11, Д.12)
5	Удаление записи из дневника	Идентификатор записи	Статус ОК	Запись дневника удалена из базы данных, сервер вернул статус 200	Запись дневника удалена из базы данных, сервер вернул статус 200 (приложения Д.13, Д.14, Д.15)

Таким образом, были протестированы основные функции приложения, и ошибок в ходе проверки не выявлено.

4.3 Метрики кода

4.3.1 Метрики Холстеда

Метрики Холстеда используются для оценки объёма кода и сложности реализации [1].

Основные характеристики:

- количество уникальных операторов;
- количество уникальных операндов;
- общее количество операторов;
- общее количество операндов.

С помощью этих характеристик рассчитываются:

- объем программы;
- сложность;

– интеллектуальные усилия.

Для анализа был выбран метод программы для подсчёта дневной нормы нутриентов. Операторы метода указаны в таблице 33.

Таблица 33 – Операторы метода

Оператор	Количество
=	5
+	4
-	3
*	7
==	1
:=	4
()	11
{}	5
case	3
switch	1
if	1
return	1

Уникальных операторов (n1) = 12, всего операторов (N1) = 46.

Операнды метода указаны в таблице 34.

Таблица 34 – Операнды метода

Операнд	Количество
profile.Gender	2
«m»	1
88.36	1
13.4	1
profile.Weight	2
4.8	1
profile.Height	2
5.7	1
float32(profile.Age)	2
447.6	1
9.2	1

Продолжение таблицы 34

Операнд	Количество
3.1	1
4.3	1
multiplier	4
profile.Goal	1
«gain»	1
1.2	1
«lose»	1
0.8	1
1.0	1
totalCalories	4
proteins	1
fats	1
carbohydrates	1
0.3	1
4	2
0.25	1
9	1
0.45	1

Уникальных операндов (n_2) = 28, всего операторов (N_2) = 41.

Словарь программы (n) = $n_1 + n_2 = 12 + 28 = 40$.

Длина программы (N) = $N_1 + N_2 = 46 + 41 = 87$.

Объем программы (V) = $N * \log_2(n) = 87 * \log_2(40) = 463$.

Трудоёмкость (E) = $V * n^{1/2} = 463 * 12^{1/2} = 2778$.

Время понимания (T) = $E/18 = 2778/18 = 154.33$.

Предполагаемое количество ошибок (B) = $V/3000 = 463/3000 = 0.15$.

Идеальный объем (V') = $(n_1 + n_2) * \log_2(n_1 + n_2) = 40 * \log_2(40) = 213$.

Уровень программы (L) = $V'/V = 213/463 = 0.46$.

Уровень идеальной программы = 1, в данном случае уровень = 0.46, что близко к среднему значению.

4.3.2 Метрики сопровождаемости

Метрики сопровождаемости оценивают, насколько легко поддерживать или модифицировать код.

Количество логических строк (LOC) = 19.

Цикломатическая сложность ($V(G)$) определяется по количеству точек принятия решений (if, else, switch, case) = 5.

Сопровождаемость кода (MI) = $171 - 5.2 * \ln(V) - 0.23 * V(G) - 16.2 * \ln(\text{LOC})$
 $= 171 - 5.2 * \ln(463) - 0.23 * 5 - 16.2 * \ln(19) = 90$

Цикломатическая сложность = 90 говорит о высокой сопровождаемости, что означает, что код легко понимать и поддерживать.

4.4 Планирование разработки и оценка бюджета

При помощи приложения Project Libre были составлены план разработки и оценка бюджета приложения [11, 21]. Были учтены все процессы разработки, выделены ресурсы и рассчитано время выполнения, начиная с 8 мая 2025 года. План разработки указан на рисунке 34.

1		☐ Анализ предметной области	9 days	4/8/25, 8:00 AM	4/18/25, 5:00 PM
2		Изучение предметной области	4 days	4/8/25, 8:00 AM	4/11/25, 5:00 PM
3		Изучение существующих аналогов приложений	2 days	4/12/25, 8:00 AM	4/15/25, 5:00 PM
4		Определение функций и требований к системе	3 days	4/16/25, 8:00 AM	4/18/25, 5:00 PM
5		☐ Проектирование системы	16 days	4/19/25, 8:00 AM	5/12/25, 5:00 PM
6		Моделирование бизнес-процессов	4 days	4/19/25, 8:00 AM	4/24/25, 5:00 PM
7		Проектирование архитектуры клиент-серверного взаимодействия	2 days	4/25/25, 8:00 AM	4/28/25, 5:00 PM
8		Проектирование базы данных	4 days	4/29/25, 8:00 AM	5/2/25, 5:00 PM
9		Проектирование API	3 days	5/3/25, 8:00 AM	5/7/25, 5:00 PM
10		Проектирование интерфейса	3 days	5/8/25, 8:00 AM	5/12/25, 5:00 PM
11		☐ Разработка приложения	24 days	5/13/25, 8:00 AM	6/13/25, 5:00 PM
12		Реализация базы данных	6 days	5/13/25, 8:00 AM	5/20/25, 5:00 PM
13		Реализация взаимодействия приложения с базой данных	5 days	5/21/25, 8:00 AM	5/27/25, 5:00 PM
14		Реализация точек входа в приложение	2 days	5/28/25, 8:00 AM	5/29/25, 5:00 PM
15		Реализация бизнес-логики	7 days	5/30/25, 8:00 AM	6/9/25, 5:00 PM
16		Реализация интерфейса	4 days	6/10/25, 8:00 AM	6/13/25, 5:00 PM
17		☐ Тестирование	2 days	6/14/25, 8:00 AM	6/17/25, 5:00 PM
18		Тестирование API	1 day	6/14/25, 8:00 AM	6/16/25, 5:00 PM
19		Тестирование интерфейса	1 day	6/17/25, 8:00 AM	6/17/25, 5:00 PM
20		Конец проекта	0 days	6/17/25, 5:00 PM	6/17/25, 5:00 PM

Рисунок 34 – План разработки

Разработка была проведена при помощи ресурсов, описанных в таблице 35.

Таблица 35 – Ресурсы, используемые в разработке

Ресурс	Тип	Стандартная ставка	Сверхурочные
Бэкенд-разработчик 1	Трудовой	800 руб/час	1600 руб/час
Бэкенд-разработчик 2	Трудовой	800 руб/час	1600 руб/час
Фронтенд-разработчик	Трудовой	700 руб/час	1400 руб/час

На основе плана разработки и распределенных ресурсов, была построена диаграмма Ганта, размещенная на рисунке 35.

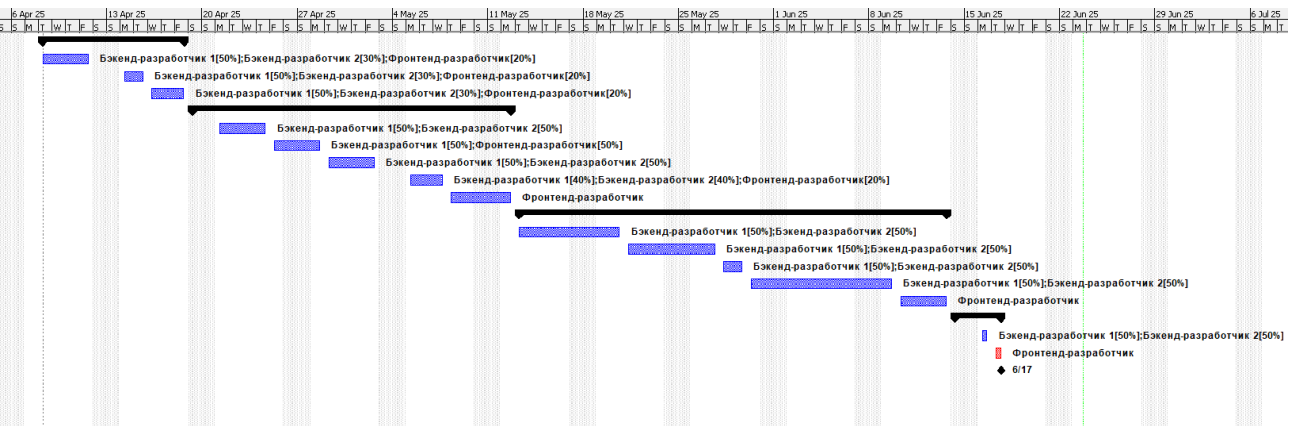


Рисунок 35 – Диаграмма Ганта

В таблице 36 представлены затраты на разработку проекта.

Таблица 36 – Затраты на разработку

Критерий	Значение
Длительность разработки	51 день
Оплата бэкенд-разработчика 1	124400 руб.
Оплата бэкенд-разработчика 2	116400 руб.
Оплата фронтенд-разработчика	65600 руб.
Общая стоимость	306400 руб.

По таблице затрат можно сделать вывод, что длительность разработки составляет 51 день, а затраты составляют 306400 руб.

Заключение

В результате выполнения выпускной квалификационной работы было разработано и протестировано веб-приложение для персонализированного подбора рационов питания и ведение пищевого дневника.

На стадии анализа требований была обоснована актуальность разработки приложения, а также выявлены ключевые функциональные и нефункциональные требования к системе, базирующиеся на потребностях пользователей.

В рамках проектирования была разработана логическая модель данных, а также спроектирована микросервисная архитектура приложения. Это позволило разделить функциональные домены и обеспечить масштабируемость и поддерживаемость системы.

Реализация серверной части осуществлялась на языке Golang с использованием микросервисной архитектуры и REST API. Каждый микросервис был структурирован согласно многослойной архитектуре (обработчики, сервисы, репозитории). В качестве системы управления базами данных для каждого микросервиса была выбрана PostgreSQL, а для взаимодействия с ней применена библиотека sqlx. Схемы базы данных развертывались с помощью миграций, а база данных заполнена тестовыми данными.

Клиентская часть разрабатывалась с использованием библиотеки React, что позволило создать современный, удобный и адаптивный интерфейс. Приложение включает в себя четыре основные страницы: авторизации, профиля, питания и дневника.

Развертывание серверной части и БД проводилось в изолированных Docker-контейнерах, что обеспечило скорость и воспроизводимость окружения на разных машинах.

Тестирование приложения производилось с использованием функционального тестирования, в ходе которого были проверены основные

функции приложения. По результатам тестирования ошибок выявлено не было, что подтверждает работоспособность приложения.

Таким образом, задачи по проектированию и разработке сервиса составления рациона питания и расчета калорий выполнены в полном объеме. Разработано эффективное приложение для поддержки здорового образа жизни и достижения индивидуальных целей пользователей.

Список использованных источников

1. Применение метрик Холстеда для количественного оценивания характеристик программ ЭВМ / А.В. Аверьянов, И.Н. Кошель, В.В. Кузнецов // Известия высших учебных заведений. Приборостроение. – 2019. – №11. – С. 970–975.
2. Отзывы о приложении FatSecret // Отзовик [Электронный ресурс]. URL: https://otzovik.com/reviews/schetchik_kaloriy_fatsecret/.
3. Отзывы о приложении MyNetDiary // PlayMarket [Электронный ресурс]. URL: <https://play.google.com/store/apps/details?id=com.fourtechnologies.mynetdiary.ad>.
4. Отзывы о приложении Cronometer // PlayMarket [Электронный ресурс]. URL: <https://play.google.com/store/apps/details?id=com.cronometer.android.gold>.
5. Отзывы о приложении MyFitnessPal // IRecommend [Электронный ресурс]. URL: <https://irecommend.ru/content/myfitnesspal>.
6. Отзывы о приложении LifeSum // IRecommend [Электронный ресурс]. URL: <https://irecommend.ru/content/kompyuternaya-programma-lifesum-vash-fitness-pomoshchnik-i-schetchik-kalorii>.
7. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование / Арлоу Д., Нейштадт А. // СПб.: Символ-Плюс, 2007. – 624 с.
8. Учимся разрабатывать REST API на Go на примере сокращателя ссылок // Habr [Электронный ресурс]. URL: <https://habr.com/ru/companies/selectel/articles/747738/>;
9. Оптимизация запросов PostgreSQL / Домбровская Г., Новиков Б., Артемьев А. // М.: ДМК Пресс, 2018. – 400 с.
10. Язык Go для начинающих / Жашкевич М. // Самиздат, 2020. – 109 с.
11. Управление проектами: Учебное пособие / Иванов П.В. // Ростов н/Д: Феникс, 2018. – 102 с.

12. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений / Канер К., Фолк Д., Нгуен Х. // Киев: ДиаСофт, 2001. – 544 с.
13. PostgreSQL. Основы языка SQL / Моргунов Е.П. // М.: ДМК Пресс, 2017. – 320 с.
14. Web-дизайн: удобство использования Web-сайтов / Нильсен Я., Лоранжер, Х. // М.: Вильямс, 2007. – 368 с.
15. Руководство по Docker Compose для начинающих // Habr [Электронный ресурс]. URL: <https://habr.com/ru/companies/ruvds/articles/450312/>
16. Изучаем PostgreSQL 10 / Салахалдин Д., Волков А. // М.: ДМК Пресс, 2018. – 400 с.
17. Чистая архитектура приложения на Go // Habr [Электронный ресурс]. URL: <https://habr.com/ru/companies/slurm/news/670786/>.
18. Объектный подход в проектировании баз данных // Портал научно-практических публикаций // Portalnp [Электронный ресурс]. URL: <http://portalnp.ru/2014/06/1987>.
19. Путеводитель по Docker. От основ контейнеризации до создания собственного докера // Habr [Электронный ресурс]. URL: <https://habr.com/ru/articles/810777/>
20. React JS. Подробное руководство / Шварц М. // М.: Питер, 2019. – 350 с.
21. Разработка REST-серверов на Go. Часть 2: применение маршрутизатора gorilla/mux // Habr [Электронный ресурс]. URL: <https://habr.com/ru/companies/ruvds/articles/561108/>
22. Управление проектами в среде ProjectLibre // Cyberleninka [Электронный ресурс]. URL: <https://cyberleninka.ru/article/n/upravlenie-proektami-v-srede-projectlibre>.

Приложение А
(обязательное)
Листинг приложения

Листинг A.1 – Код сервиса авторизации

```

app.go:

package app

import (
    "context"
    "fmt"
    "time"

    "github.com/jmoiron/sqlx"
    _ "github.com/lib/pq"
    "github.com/pressly/goose/v3"
    "go.uber.org/zap"

    "auth/internal/api"
    "auth/internal/config"
    "auth/internal/database"
    "auth/internal/service"
    "auth/internal/utils"
)

type App struct {
    ctx      utils.MyContext
    server *api.Server
    db       *sqlx.DB
    config config.Config
}

func NewApp(ctx context.Context, logger *zap.SugaredLogger, config
config.Config) *App {
    return &App{
        ctx:      utils.NewMyContext(ctx, logger),
        config: config,
    }
}

func (a *App) InitDatabase() error {
    a.ctx.Logger.Infof("connecting to DB with: %s",
a.config.DBConnectionString)

    var err error

    for i := 0; i < 10; i++ {
        a.db, err = sqlx.Open("postgres",
a.config.DBConnectionString)
        if err != nil {
            a.ctx.Logger.Warnf("failed to connect to DB, attempt %d:
%v", i+1, err)
            time.Sleep(5 * time.Second)
            continue
        }
    }
}

```

```

        if err = a.db.Ping(); err == nil {
            a.ctx.Logger.Infof("successfully connected to DB")
            break
        }

        a.ctx.Logger.Warnf("DB not ready, attempt %d: %v\n", i+1,
err)
            time.Sleep(5 * time.Second)
        }

        return nil
    }

func (a *App) RunMigrations() error {
    a.ctx.Logger.Info("running database migrations")

    if err := goose.Up(a.db.DB, "./migrations"); err != nil {
        return fmt.Errorf("failed to apply migrations: %w", err)
    }

    a.ctx.Logger.Info("migrations applied successfully")
    return nil
}

func (a *App) InitService() {
    s := service.NewAuthorizationService(
        database.NewAuthorizationPostgres(a.db),
        a.config.JWTSecret,
        a.config.AccessTokenTTL,
        a.config.RefreshTokenTTL,
    )

    a.server = api.NewServer(a.ctx, a.config)
    a.server.HandleAuth(a.ctx, s)
}

func (a *App) Run() error {
    go func() {
        if err := a.server.Run(); err != nil {
            a.ctx.Logger.Fatalf("error occurred while running http
server: %s", err.Error())
        }
    }()

    a.ctx.Logger.Info("run server")
    return nil
}

func (a *App) Shutdown() error {
    err := a.server.Shutdown(a.ctx)
    if err != nil {
        a.ctx.Logger.Errorf("Failed to disconnect from server %v",

```

```

err)
    return err
}

err = a.db.Close()
if err != nil {
    a.ctx.Logger.Errorf("failed to disconnect from bd %v", err)
}

a.ctx.Logger.Info("server shut down successfully")
return nil
}
models.go:

package models

import (
    "github.com/golang-jwt/jwt/v5"
    "time"
)

type User struct {
    ID          string `json:"id" db:"id"`
    Email       string `json:"email" db:"email"`
    Username    string `json:"username" db:"username"`
    Password    string `json:"password" db:"password_hash"`
    IsVerified bool   `db:"is_verified"`
}

type LoginInput struct {
    Email    string `json:"email"`
    Password string `json:"password"`
}

type TokenPair struct {
    AccessToken string `json:"access_token"`
    RefreshToken string `json:"refresh_token"`
}

type Claims struct {
    RefreshTokenID string `json:"refresh_token_id"`
    UserID          string `json:"user_id"`
    UserIP          string `json:"user_ip"`
    jwt.RegisteredClaims
}

type VerifyEmailRequest struct {
    Email string `json:"email"`
    Code  string `json:"code"`
}

type VerificationCode struct {
    ID string `db:"id"`
}

```

```

        UserID    string    `db:"user_id"`
        Code      string    `db:"code"`
        ExpiresAt time.Time  `db:"expires_at"`
        CreatedAt time.Time  `db:"created_at"`
    }
server.go:

package api

import (
    "net/http"
    "time"

    "github.com/gorilla/handlers"
    "github.com/gorilla/mux"

    "auth/internal/api/handler"
    "auth/internal/api/middleware"
    "auth/internal/config"
    "auth/internal/service"
    "auth/internal/utils"
)

const (
    maxHeaderBytes = 1 << 20
    readTimeout    = 10 * time.Second
    writeTimeout   = 10 * time.Second
)

type Server struct {
    httpServer *http.Server
    router     *mux.Router
}

func NewServer(ctx utils.MyContext, config config.Config) *Server
{
    router := mux.NewRouter()

    allowedOrigins :=
handlers.AllowedOrigins([]string{"http://localhost:3000"})
    allowedMethods := handlers.AllowedMethods([]string{"GET",
"POST", "OPTIONS"})
    allowedHeaders := handlers.AllowedHeaders([]string{"Content-
Type", "Authorization"})

    corsRouter := handlers.CORS(allowedOrigins, allowedMethods,
allowedHeaders)(router)

    recoveredRouter := middleware.RecoveryMiddleware(ctx,
corsRouter)

    return &Server{
        httpServer: &http.Server{

```

```

        Addr:            config.ServerPort,
        MaxHeaderBytes:  maxHeaderBytes,
        ReadTimeout:     readTimeout,
        WriteTimeout:    writeTimeout,
        Handler:         recoveredRouter,
    },
    router: router,
}
}

func (s *Server) Run() error {
    return s.httpServer.ListenAndServe()
}

func (s *Server) Shutdown(ctx utils.MyContext) error {
    return s.httpServer.Shutdown(ctx.Ctx)
}

func (s *Server) HandleAuth(ctx utils.MyContext, service
service.AuthorizationService) {
    s.router.HandleFunc("/api/register/", handler.Register(ctx,
service)).Methods(http.MethodPost)
    s.router.HandleFunc("/api/login/", handler.Login(ctx,
service)).Methods(http.MethodPost)
    s.router.HandleFunc("/api/refresh/", handler.Refresh(ctx,
service)).Methods(http.MethodPost)
    s.router.HandleFunc("/api/verify/", handler.VerifyEmail(ctx,
service)).Methods(http.MethodPost)
}
auth.go:
package handler

import (
    "encoding/json"
    "net/http"

    "auth/internal/models"
    "auth/internal/service"
    "auth/internal/utils"
)

func Register(ctx utils.MyContext, service
service.AuthorizationService) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        var user models.User

        if err := json.NewDecoder(r.Body).Decode(&user); err != nil
{
            utils.NewErrorResponse(ctx, w, "invalid JSON payload",
http.StatusBadRequest)
            return
        }
    }
}

```

```

        id, err := service.CreateUser(ctx, user)
        if err != nil {
            utils.NewErrorResponse(ctx, w, err.Error(),
http.StatusInternalServerError)
            return
        }

        if err := utils.WriteResponse(w, http.StatusOK,
map[string]interface{}{"id": id}); err != nil {
            utils.NewErrorResponse(ctx, w, err.Error(),
http.StatusInternalServerError)
            return
        }
    }
}

func Login(ctx utils.MyContext, service
service.AuthorizationService) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        var input models.LoginInput

        if err := json.NewDecoder(r.Body).Decode(&input); err !=
nil {
            utils.NewErrorResponse(ctx, w, "invalid JSON payload",
http.StatusBadRequest)
            return
        }

        userIP := utils.GetClientIP(r)
        if userIP == "" {
            utils.NewErrorResponse(ctx, w, "unable to determine user
IP", http.StatusBadRequest)
            return
        }

        tokenPair, err := service.Login(ctx, input, userIP)
        if err != nil {
            utils.NewErrorResponse(ctx, w, err.Error(),
http.StatusInternalServerError)
            return
        }

        if err = utils.WriteResponse(w, http.StatusOK, tokenPair);
err != nil {
            utils.NewErrorResponse(ctx, w, "internal server error",
http.StatusInternalServerError)
            return
        }
    }
}

func Refresh(ctx utils.MyContext, service

```

```

service.AuthorizationService) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        var tokenPair models.TokenPair
        if err := json.NewDecoder(r.Body).Decode(&tokenPair); err
        != nil {
            utils.NewErrorResponse(ctx, w, "invalid JSON payload",
            http.StatusBadRequest)
            return
        }

        if tokenPair.AccessToken == "" || tokenPair.RefreshToken ==
        "" {
            utils.NewErrorResponse(ctx, w, "both access and refresh
            tokens are required", http.StatusBadRequest)
            return
        }

        userIP := utils.GetClientIP(r)
        if userIP == "" {
            utils.NewErrorResponse(ctx, w, "unable to determine user
            IP", http.StatusBadRequest)
            return
        }

        newTokenPair, err := service.RefreshTokens(ctx,
        tokenPair.AccessToken, tokenPair.RefreshToken, userIP)
        if err != nil {
            if err.Error() == "hash and token do not match" ||
            err.Error() == "invalid access token" {
                utils.NewErrorResponse(ctx, w, err.Error(),
                http.StatusUnauthorized)
            } else {
                utils.NewErrorResponse(ctx, w, err.Error(),
                http.StatusInternalServerError)
            }
            return
        }

        if err = utils.WriteResponse(w, http.StatusOK,
        newTokenPair); err != nil {
            utils.NewErrorResponse(ctx, w, "internal server error",
            http.StatusInternalServerError)
            return
        }
    }
}

func VerifyEmail(ctx utils.MyContext, service
service.AuthorizationService) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        var req models.VerifyEmailRequest
        if err := json.NewDecoder(r.Body).Decode(&req); err != nil
        {

```

```

        utils.NewErrorResponse(utils.MyContext{Ctx:
r.Context()}, w, "invalid request format", http.StatusBadRequest)
        return
    }

    if req.Email == "" || req.Code == "" {
        utils.NewErrorResponse(utils.MyContext{Ctx:
r.Context()}, w, "email and confirmation code are required",
http.StatusBadRequest)
        return
    }

    err := service.VerifyEmail(ctx, req.Email, req.Code)
    if err != nil {
        if err.Error() == "invalid verification code" {
            utils.NewErrorResponse(utils.MyContext{Ctx:
r.Context()}, w, "invalid verification code",
http.StatusBadRequest)
            return
        }
        utils.NewErrorResponse(utils.MyContext{Ctx:
r.Context()}, w, "email verification failed",
http.StatusBadRequest)
        return
    }
}

if err = utils.WriteResponse(w, http.StatusOK,
utils.StatusResponse{Status: "ok"}); err != nil {
    utils.NewErrorResponse(ctx, w, "internal server error",
http.StatusInternalServerError)
    return
}
}
}

service.go:

package service

import (
    "crypto/rand"
    "database/sql"
    "encoding/base64"
    "errors"
    "fmt"
    "log"
    "math/big"
    "time"

    "github.com/golang-jwt/jwt/v5"
    "github.com/google/uuid"
    "golang.org/x/crypto/bcrypt"

    "auth/internal/database"

```

```

        dbmodels "auth/internal/database/models"
        "auth/internal/models"
        "auth/internal/utils"
    )

type AuthorizationService interface {
    CreateUser(ctx utils.MyContext, user models.User) (string,
error)
    Login(ctx utils.MyContext, input models.LoginInput, userIP
string) (models.TokenPair, error)
    RefreshTokens(ctx utils.MyContext, accessToken, refreshToken,
userIP string) (models.TokenPair, error)
    VerifyEmail(ctx utils.MyContext, email, code string) error
    generateTokens(ctx utils.MyContext, userID, userIP string)
(models.TokenPair, error)
}

type ImplAuthorizationService struct {
    repo          database.AuthorizationRepository
    jwtSecret      []byte
    accessTokenTTL time.Duration
    refreshTokenTTL time.Duration
}

func NewAuthorizationService(repo
database.AuthorizationRepository, jwtSecret []byte,
accessTokenTTL, refreshTokenTTL time.Duration)
*ImplAuthorizationService {
    return &ImplAuthorizationService{
        repo:          repo,
        jwtSecret:      jwtSecret,
        accessTokenTTL: accessTokenTTL,
        refreshTokenTTL: refreshTokenTTL,
    }
}

func (s *ImplAuthorizationService) CreateUser(ctx utils.MyContext,
req models.User) (string, error) {
    existingUser, err := s.repo.GetUser(ctx, req.Email)
    if err != nil && !errors.Is(err, sql.ErrNoRows) {
        return "", fmt.Errorf("database error while checking email:
%w", err)
    }

    hashedPassword, err := generatePasswordHash(req.Password)
    if err != nil {
        return "", err
    }

    var userToSave models.User

    if existingUser.ID != "" {
        if existingUser.IsVerified {

```

```

        return "", fmt.Errorf("user with this email is already
registered and confirmed")
    } else {
        userToSave = existingUser
        userToSave.Username = req.Username
        userToSave.Password = hashedPassword

        fmt.Println(userToSave)

        err = s.repo.UpdateUser(ctx, userToSave)
        if err != nil {
            return "", fmt.Errorf("failed to update unverified
user: %w", err)
        }
        log.Printf("unverified account for email %s (ID: %s)
updated. sending new verification code", userToSave.Email,
userToSave.ID)
    }
    } else {
        userToSave = models.User{
            Username:    req.Username,
            Email:        req.Email,
            Password:     hashedPassword,
            IsVerified:    false,
        }
        var newUserID string
        newUserID, err = s.repo.CreateUser(ctx, userToSave)
        if err != nil {
            return "", fmt.Errorf("failed to create new user: %w",
err)
        }
        userToSave.ID = newUserID
        log.Printf("new user %s (ID: %s) created. sending
verification code", userToSave.Email, userToSave.ID)
    }

    err = s.repo.DeleteVerificationCodesForUser(ctx,
userToSave.ID)
    if err != nil {
        log.Printf("warning: failed to delete old verification
codes for user %s: %v", userToSave.ID, err)
    }

    code, _ := generateVerificationCode()
    expiresAt := time.Now().Add(15 * time.Minute)

    verificationCode := models.VerificationCode{
        UserID:        userToSave.ID,
        Code:          code,
        ExpiresAt:     expiresAt,
    }

    err = s.repo.CreateVerificationCode(ctx, verificationCode)

```

```

        if err != nil {
            return "", fmt.Errorf("failed to save verification code:
%w", err)
        }

        go func() {
            fmt.Println(userToSave, verificationCode)
            if emailErr :=
utils.SendVerificationEmail(userToSave.Email, code); emailErr !=
nil {
                log.Printf("critical error sending email for %s: %v",
userToSave.Email, emailErr)
            }
        }()

        return userToSave.ID, nil
    }

func (s *ImplAuthorizationService) Login(ctx utils.MyContext,
input models.LoginInput, userIP string) (models.TokenPair, error)
{
    user, err := s.repo.GetUser(ctx, input.Email)
    if err != nil {
        return models.TokenPair{}, err
    }

    if !user.IsVerified {
        return models.TokenPair{}, fmt.Errorf("user is not
verified")
    }

    err = bcrypt.CompareHashAndPassword([]byte(user.Password),
[]byte(input.Password))
    if err != nil {
        return models.TokenPair{}, errors.New("invalid password")
    }

    return s.generateTokens(ctx, user.ID, userIP)
}

func (s *ImplAuthorizationService) VerifyEmail(ctx
utils.MyContext, email, code string) error {
    user, err := s.repo.GetUser(ctx, email)
    if err != nil {
        if errors.Is(err, sql.ErrNoRows) {
            return fmt.Errorf("user email address not found")
        }
        return fmt.Errorf("database error while searching for user:
%w", err)
    }

    if user.IsVerified {
        return fmt.Errorf("account already verified")
    }

```

```

    }

    verificationCodeEntry, err :=
s.repo.GetLatestVerificationCode(ctx, user.ID)
    if err != nil {
        if err.Error() == fmt.Sprintf("no active verification code
found for user %s: %v", user.ID, sql.ErrNoRows) {
            return fmt.Errorf("verification code not found or
expired")
        }
        return fmt.Errorf("database error while receiving
verification code: %w", err)
    }

    if verificationCodeEntry.Code != code {
        return fmt.Errorf("invalid verification code")
    }

    if time.Now().After(verificationCodeEntry.ExpiresAt) {
        err = s.repo.DeleteVerificationCodesForUser(ctx, user.ID)
        return fmt.Errorf("the verification code has expired")
    }

    err = s.repo.UpdateUserVerificationStatus(ctx, user.ID, true)
    if err != nil {
        return fmt.Errorf("failed to update user verification
status: %w", err)
    }

    err = s.repo.DeleteVerificationCodesForUser(ctx, user.ID)
    if err != nil {
        log.Printf("warning: failed to delete used verification
codes for user %s: %v", user.ID, err)
    }

    return nil
}

func (s *ImplAuthorizationService) RefreshTokens(ctx
utils.MyContext, accessToken, refreshToken, userIP string)
(models.TokenPair, error) {
    claims, err := parseAccessToken(accessToken, s.jwtSecret)
    if err != nil {
        return models.TokenPair{}, fmt.Errorf("invalid access
token: %w", err)
    }

    refreshTokenID := claims.RefreshTokenID
    storedIP := claims.UserIP
    storedUserID := claims.UserID

    if storedIP != userIP {
        go utils.SendWarningEmail(ctx, storedUserID, storedIP,

```

```

userIP)
    }

    storedTokenHash, err := s.repo.GetRefreshToken(ctx,
refreshTokenID)
    if err != nil {
        return models.TokenPair{}, err
    }

    if err = compareHashes(storedTokenHash, refreshToken); err !=
nil {
        return models.TokenPair{}, err
    }

    return s.generateTokens(ctx, storedUserID, userIP)
}

func (s *ImplAuthorizationService) generateTokens(ctx
utils.MyContext, userID, userIP string) (models.TokenPair, error)
{
    refreshTokenStr, err := generateRefreshToken()
    if err != nil {
        return models.TokenPair{}, err
    }

    refreshTokenHashByte, err :=
bcrypt.GenerateFromPassword([]byte(refreshTokenStr),
bcrypt.DefaultCost)
    if err != nil {
        return models.TokenPair{}, errors.New("failed to hash
refresh token")
    }
    refreshTokenHash := string(refreshTokenHashByte)

    refreshTokenID := uuid.New().String()

    err = s.repo.SaveRefreshToken(
        ctx,
        dbmodels.RefreshToken{
            ID:          refreshTokenID,
            UserID:      userID,
            TokenHash:   refreshTokenHash,
            ExpiresAt:   time.Now().Add(s.refreshTokenTTL),
        },
    )
    if err != nil {
        return models.TokenPair{}, err
    }

    accessToken, err := generateAccessToken(refreshTokenID,
userID, userIP, s.jwtSecret, s.accessTokenTTL)
    if err != nil {
        return models.TokenPair{}, errors.New("failed to generate

```

```

access token")
    }

    return models.TokenPair{
        AccessToken:  accessToken,
        RefreshToken: refreshTokenStr,
    }, nil
}

func generateAccessToken(refreshTokenID, userID, userIP string,
    jwtSecret []byte, tokenTTL time.Duration) (string, error) {
    claims := models.Claims{
        RefreshTokenID: refreshTokenID,
        UserID:         userID,
        UserIP:         userIP,
        RegisteredClaims: jwt.RegisteredClaims{
            ExpiresAt: jwt.NewNumericDate(time.Now().Add(tokenTTL)),
            IssuedAt:  jwt.NewNumericDate(time.Now()),
        },
    }

    token := jwt.NewWithClaims(jwt.SigningMethodHS512, claims)

    return token.SignedString(jwtSecret)
}

func generateRefreshToken() (string, error) {
    randomBytes := make([]byte, 32)
    if _, err := rand.Read(randomBytes); err != nil {
        return "", errors.New("failed to generate random bytes for
refresh token")
    }

    refreshToken := base64.StdEncoding.EncodeToString(randomBytes)

    return refreshToken, nil
}

func parseAccessToken(accessToken string, jwtSecret []byte)
(*models.Claims, error) {
    token, err := jwt.ParseWithClaims(accessToken,
&models.Claims{}, func(token *jwt.Token) (interface{}, error) {
        if _, ok := token.Method.(*jwt.SigningMethodHMAC); !ok {
            return nil, errors.New("unexpected signing method")
        }
        return jwtSecret, nil
    })
    if err != nil {
        if errors.Is(err, jwt.ErrTokenExpired) {
            claims, ok := token.Claims.(*models.Claims)
            if !ok {
                return nil, errors.New("invalid token claims")
            }
        }
    }
}

```

```

        return claims, nil
    }
    return nil, errors.New("error parsing token: " +
err.Error())
}

claims, ok := token.Claims.(*models.Claims) // !
if !ok || !token.Valid {
    return nil, errors.New("error parsing token")
}

return claims, nil
}

func compareHashes(hash string, token string) error {
    err := bcrypt.CompareHashAndPassword([]byte(hash),
[]byte(token))
    if err != nil {
        if errors.Is(err, bcrypt.ErrMismatchedHashAndPassword) {
            return errors.New("hash and token do not match")
        }

        return err
    }

    return nil
}

func generatePasswordHash(password string) (string, error) {
    hash, err := bcrypt.GenerateFromPassword([]byte(password),
bcrypt.DefaultCost)
    if err != nil {
        return "", err
    }

    return string(hash), nil
}

func generateVerificationCode() (string, error) {
    n, err := rand.Int(rand.Reader, big.NewInt(1000000))
    if err != nil {
        return "", fmt.Errorf("failed to generate random code: %w",
err)
    }
    return fmt.Sprintf("%06d", n.Int64()), nil
}
}

postgres.go:

package database

import (
    "auth/internal/models"
    _ "embed"

```

```

    "errors"
    "fmt"
    "time"

    "github.com/google/uuid"
    "github.com/jmoiron/sqlx"

    dbmodels "auth/internal/database/models"
    "auth/internal/utls"
    "database/sql"
)

type AuthorizationRepository interface {
    CreateUser(ctx utils.MyContext, user models.User) (string,
error)
    UpdateUser(ctx utils.MyContext, user models.User) error
    GetUser(ctx utils.MyContext, email string) (models.User,
error)
    SaveRefreshToken(ctx utils.MyContext, refreshToken
dbmodels.RefreshToken) error
    GetRefreshToken(ctx utils.MyContext, tokenID string) (string,
error)
    CreateVerificationCode(ctx utils.MyContext, code
models.VerificationCode) error
    GetLatestVerificationCode(ctx utils.MyContext, userID string)
(models.VerificationCode, error)
    DeleteVerificationCodesForUser(ctx utils.MyContext, userID
string) error
    UpdateUserVerificationStatus(ctx utils.MyContext, userID
string, isVerified bool) error
}

type AuthorizationPostgres struct {
    db *sqlx.DB
}

func NewAuthorizationPostgres(db *sqlx.DB) *AuthorizationPostgres
{
    return &AuthorizationPostgres{db: db}
}

//go:embed sql/CreateUser.sql
var sqlCreateUser string

func (r *AuthorizationPostgres) CreateUser(ctx utils.MyContext,
user models.User) (string, error) {
    user.ID = uuid.New().String()

    result, err := r.db.ExecContext(ctx.Ctx, sqlCreateUser,
user.ID, user.Username, user.Email, user.Password)
    if err != nil {
        return "", err
    }
}

```

```

    rowsAffected, err := result.RowsAffected()
    if err != nil {
        return "", err
    }
    if rowsAffected == 0 {
        return "", errors.New("user was not created")
    }

    return user.ID, nil
}

//go:embed sql/UpdateUser.sql
var sqlUpdateUser string

func (r *AuthorizationPostgres) UpdateUser(ctx utils.MyContext,
user models.User) error {
    createdAt := time.Now()
    _, err := r.db.ExecContext(ctx.Ctx, sqlUpdateUser, user.ID,
user.Username, user.Password, createdAt)
    if err != nil {
        return fmt.Errorf("failed to update user: %w", err)
    }
    return nil
}

//go:embed sql/GetUser.sql
var sqlGetUser string

func (r *AuthorizationPostgres) GetUser(ctx utils.MyContext, email
string) (models.User, error) {
    var user models.User

    err := r.db.GetContext(ctx.Ctx, &user, sqlGetUser, email)

    return user, err
}

//go:embed sql/DeleteRefreshToken.sql
var sqlDeleteRefreshToken string

//go:embed sql/SaveRefreshToken.sql
var sqlSaveRefreshToken string

func (r *AuthorizationPostgres) SaveRefreshToken(ctx
utils.MyContext, token dbmodels.RefreshToken) error {
    tx, err := r.db.Beginx()
    if err != nil {
        return errors.New("internal server error")
    }

    _, err = tx.ExecContext(ctx.Ctx, sqlDeleteRefreshToken,
token.UserID)

```

```

    if err != nil {
        tx.Rollback()
        ctx.Logger.Error("failed to delete token: ", err)
        return errors.New("internal server error")
    }

    _, err = tx.ExecContext(ctx.Ctx, sqlSaveRefreshToken,
token.ID, token.UserID, token.TokenHash, token.ExpiresAt)
    if err != nil {
        tx.Rollback()
        ctx.Logger.Error("failed to insert refresh token: ", err)
        return errors.New("internal server error")
    }

    err = tx.Commit()
    if err != nil {
        return errors.New("internal server error")
    }

    return nil
}

//go:embed sql/GetRefreshToken.sql
var getRefreshToken string

func (r *AuthorizationPostgres) GetRefreshToken(ctx
utils.MyContext, tokenID string) (string, error) {
    var tokenHash string

    err := r.db.GetContext(ctx.Ctx, &tokenHash, getRefreshToken,
tokenID)
    if err != nil {
        if errors.Is(err, sql.ErrNoRows) {
            return "", errors.New("invalid access token")
        }

        ctx.Logger.Error("failed to get refresh token: ", err)
        return "", errors.New("internal server error")
    }

    return tokenHash, nil
}

//go:embed sql/CreateVerificationCode.sql
var sqlCreateVerificationCode string

func (r *AuthorizationPostgres) CreateVerificationCode(ctx
utils.MyContext, code models.VerificationCode) error {
    code.ID = uuid.New().String()
    code.CreatedAt = time.Now()
    _, err := r.db.ExecContext(ctx.Ctx, sqlCreateVerificationCode,
code.ID, code.UserID, code.Code, code.ExpiresAt, code.CreatedAt)
    if err != nil {

```

```

        return fmt.Errorf("failed to create verification code: %w",
err)
    }
    return nil
}

//go:embed sql/GetLatestVerificationCode.sql
var sqlGetLatestVerificationCode string

func (r *AuthorizationPostgres) GetLatestVerificationCode(ctx
utils.MyContext, userID string) (models.VerificationCode, error) {
    var code models.VerificationCode
    err := r.db.GetContext(ctx.Ctx, &code,
sqlGetLatestVerificationCode, userID)
    if err != nil {
        return models.VerificationCode{}, fmt.Errorf("no active
verification code found for user %s: %w", userID, err)
    }
    return code, nil
}

//go:embed sql/DeleteVerificationCodesForUser.sql
var sqlDeleteVerificationCodesForUser string

func (r *AuthorizationPostgres) DeleteVerificationCodesForUser(ctx
utils.MyContext, userID string) error {
    _, err := r.db.ExecContext(ctx.Ctx,
sqlDeleteVerificationCodesForUser, userID)
    if err != nil {
        return fmt.Errorf("failed to delete verification codes for
user %s: %w", userID, err)
    }
    return nil
}

//go:embed sql/UpdateUserVerificationStatus.sql
var sqlUpdateUserVerificationStatus string

func (r *AuthorizationPostgres) UpdateUserVerificationStatus(ctx
utils.MyContext, userID string, isVerified bool) error {
    _, err := r.db.ExecContext(ctx.Ctx,
sqlUpdateUserVerificationStatus, userID, isVerified)
    if err != nil {
        return fmt.Errorf("failed to update user verification
status for user %s: %w", userID, err)
    }
    return nil
}
}
main.go:
package main

import (

```

```

"context"
"os"
"os/signal"
"syscall"

"github.com/joho/godotenv"
"go.uber.org/zap"

"auth/internal/app"
"auth/internal/config"
)

func main() {
    prdLogger, _ := zap.NewProduction()
    defer prdLogger.Sync()
    logger := prdLogger.Sugar()

    mainCtx := context.Background()
    ctx, cancel := context.WithCancel(mainCtx)
    defer cancel()

    err := godotenv.Load()
    if err != nil {
        logger.Warnf(".env file not found: %v", err)
    }

    config, err := config.NewConfig()
    if err != nil {
        logger.Fatalf("failed to read config: %s", err.Error())
    }

    newApp := app.NewApp(ctx, logger, config)

    if err := newApp.InitDatabase(); err != nil {
        logger.Fatalf("failed to initialize db: %s", err.Error())
    }

    if err = newApp.RunMigrations(); err != nil {
        logger.Fatalf("failed to run migrations: %s", err.Error())
    }

    newApp.InitService()

    if err = newApp.Run(); err != nil {
        logger.Errorf(err.Error())
        return
    }

    sigChan := make(chan os.Signal, 1)
    signal.Notify(sigChan, syscall.SIGTERM, syscall.SIGINT)
    <-sigChan

    if err = newApp.Shutdown(); err != nil {

```

```

        logger.Errorf(err.Error())
        return
    }
}

```

Листинг А.2 – Код сервиса профиля

main.go:

```

package main

import (
    "context"
    "os"
    "os/signal"
    "syscall"

    "github.com/joho/godotenv"
    "go.uber.org/zap"

    "tube-profile/internal/app"
    "tube-profile/internal/config"
)

func main() {
    prdLogger, _ := zap.NewProduction()
    defer prdLogger.Sync()
    logger := prdLogger.Sugar()

    mainCtx := context.Background()
    ctx, cancel := context.WithCancel(mainCtx)
    defer cancel()

    err := godotenv.Load()
    if err != nil {
        logger.Warnf(".env file not found: %v", err)
    }

    config, err := config.NewConfig()
    if err != nil {
        logger.Fatalf("failed to read config: %s", err.Error())
    }

    newApp := app.NewApp(ctx, logger, config)

    if err := newApp.InitDatabase(); err != nil {
        logger.Fatalf("failed to initialize db: %s", err.Error())
    }

    if err := newApp.RunMigrations(); err != nil {
        logger.Fatalf("failed to run migrations: %s", err.Error())
    }
}

```

```

newApp.InitService()

if err = newApp.Run(); err != nil {
    logger.Errorf(err.Error())
    return
}

sigChan := make(chan os.Signal, 1)
signal.Notify(sigChan, syscall.SIGTERM, syscall.SIGINT)
<-sigChan

if err = newApp.Shutdown(); err != nil {
    logger.Errorf(err.Error())
    return
}
}
app.go:
package app

import (
    "context"
    "fmt"
    "time"

    "github.com/jmoiron/sqlx"
    _ "github.com/lib/pq"
    "github.com/pressly/goose/v3"
    "go.uber.org/zap"

    "tube-profile/internal/api"
    "tube-profile/internal/config"
    "tube-profile/internal/database"
    "tube-profile/internal/service"
    "tube-profile/internal/utils"
)

type App struct {
    ctx      utils.MyContext
    server   *api.Server
    db       *sqlx.DB
    config   config.Config
}

func NewApp(ctx context.Context, logger *zap.SugaredLogger, config
config.Config) *App {
    return &App{
        ctx:      utils.NewMyContext(ctx, logger),
        config:   config,
    }
}

func (a *App) InitDatabase() error {

```

```

    a.ctx.Logger.Infof("connecting to DB with: %s",
a.config.DBConnectionString)

    var err error

    for i := 0; i < 10; i++ {
        a.db, err = sqlx.Open("postgres",
a.config.DBConnectionString)
        if err != nil {
            a.ctx.Logger.Warnf("failed to connect to DB, attempt %d:
%v", i+1, err)
            time.Sleep(5 * time.Second)
            continue
        }

        if err = a.db.Ping(); err == nil {
            a.ctx.Logger.Infof("successfully connected to DB")
            break
        }

        a.ctx.Logger.Warnf("DB not ready, attempt %d: %v\n", i+1,
err)
        time.Sleep(5 * time.Second)
    }

    return nil
}

func (a *App) RunMigrations() error {
    a.ctx.Logger.Info("running database migrations")

    if err := goose.Up(a.db.DB, "./migration"); err != nil {
        return fmt.Errorf("failed to apply migrations: %w", err)
    }

    a.ctx.Logger.Info("migrations applied successfully")
    return nil
}

func (a *App) InitService() {
    s := service.NewProfileService(
        database.NewProfilePostgres(a.db),
        a.config.JWTSecret,
    )

    a.server = api.NewServer(a.ctx, a.config)
    a.server.HandleAuth(a.ctx, s)
}

func (a *App) Run() error {
    go func() {
        if err := a.server.Run(); err != nil {
            a.ctx.Logger.Fatalf("error occurred while running http

```

```

server: %s", err.Error())
    }
    }()

    a.ctx.Logger.Info("run server")
    return nil
}

func (a *App) Shutdown() error {
    err := a.server.Shutdown(a.ctx)
    if err != nil {
        a.ctx.Logger.Errorf("Failed to disconnect from server %v",
err)
        return err
    }

    err = a.db.Close()
    if err != nil {
        a.ctx.Logger.Errorf("failed to disconnect from bd %v", err)
    }

    a.ctx.Logger.Info("server shut down successfully")
    return nil
}
service.go:
package service

import (
    "time"
    "tube-profile/internal/database"
    "tube-profile/internal/database/mapper"
    "tube-profile/internal/model"
    "tube-profile/internal/utils"
)

type ProfileService interface {
    CreateProfile(ctx utils.MyContext, profile model.Profile)
error
    GetProfile(ctx utils.MyContext, userID string) (model.Profile,
error)
    UpdateProfile(ctx utils.MyContext, profile model.Profile)
error
    GetUserWeightHistory(ctx utils.MyContext, userID string, date
time.Time) ([]model.UserWeight, error)
    AddUserWeight(ctx utils.MyContext, weight model.UserWeight)
error
    DeleteUserWeight(ctx utils.MyContext, userID, weightID string)
error
}

type ImplProfileService struct {
    repo          database.ProfileRepository

```

```

    jwtSecret []byte
}

func NewProfileService(repo database.ProfileRepository, jwtSecret
[]byte) *ImplProfileService {
    return &ImplProfileService{
        repo:      repo,
        jwtSecret: jwtSecret,
    }
}

func (s *ImplProfileService) CreateProfile(ctx utils.MyContext,
profile model.Profile) error {
    err := s.repo.Create(ctx, mapper.ToDBProfile(profile))
    if err != nil {
        return err
    }

    return nil
}

func (s *ImplProfileService) GetProfile(ctx utils.MyContext,
userID string) (model.Profile, error) {
    profile, err := s.repo.GetByUserID(ctx, userID)
    if err != nil {
        return model.Profile{}, err
    }

    return mapper.FromDBProfile(profile), nil
}

func (s *ImplProfileService) UpdateProfile(ctx utils.MyContext,
profile model.Profile) error {
    err := s.repo.Update(ctx, mapper.ToDBProfile(profile))
    if err != nil {
        return err
    }

    return nil
}

func (s *ImplProfileService) GetUserWeightHistory(ctx
utils.MyContext, userID string, date time.Time)
([]model.UserWeight, error) {
    weights, err := s.repo.GetUserWeightsByYear(ctx, userID, date)
    if err != nil {
        return []model.UserWeight{}, err
    }

    return weights, nil
}

func (s *ImplProfileService) AddUserWeight(ctx utils.MyContext,

```

```

weight model.UserWeight) error {
    err := s.repo.CreateUserWeight(ctx, weight)
    if err != nil {
        return err
    }

    return nil
}

func (s *ImplProfileService) DeleteUserWeight(ctx utils.MyContext,
userID, weightID string) error {
    err := s.repo.DeleteUserWeight(ctx, userID, weightID)
    if err != nil {
        return err
    }

    return nil
}
postgres.go:
package database

import (
    "database/sql"
    "embed"
    "errors"
    "fmt"
    "github.com/google/uuid"
    "github.com/jmoiron/sqlx"
    "log"
    "strings"
    "time"
    "tube-profile/internal/model"
    "tube-profile/internal/utils"
)

type ProfileRepository interface {
    Create(ctx utils.MyContext, profile Profile) error
    GetByUserID(ctx utils.MyContext, userID string) (Profile,
error)
    Update(ctx utils.MyContext, input Profile) error
    GetUserWeightsByYear(ctx utils.MyContext, userID string, date
time.Time) ([]model.UserWeight, error)
    CreateUserWeight(ctx utils.MyContext, weight model.UserWeight)
error
    DeleteUserWeight(ctx utils.MyContext, userID, weightID string)
error
}

type ProfilePostgres struct {
    db *sqlx.DB
}

```

```

func NewProfilePostgres(db *sqlx.DB) *ProfilePostgres {
    return &ProfilePostgres{db: db}
}

//go:embed sql/CreateProfile.sql
var createProfile string

func (p *ProfilePostgres) Create(ctx utils.MyContext, profile
Profile) error {
    profile.ID = uuid.New().String()

    log.Println("Repository userID", profile.UserID)

    _, err := p.db.ExecContext(ctx.Ctx, createProfile, profile.ID,
profile.UserID, profile.Gender, profile.Age, profile.Weight,
profile.Height, profile.Goal)
    if err != nil {
        return fmt.Errorf("failed to insert profile: %w", err)
    }

    return nil
}

//go:embed sql/GetProfile.sql
var getProfile string

func (p *ProfilePostgres) GetByUserID(ctx utils.MyContext, userID
string) (Profile, error) {
    var profile Profile

    err := p.db.GetContext(ctx.Ctx, &profile, getProfile, userID)
    if err != nil {
        if errors.Is(err, sql.ErrNoRows) {
            return Profile{}, fmt.Errorf("profile not found for
user_id: %s", userID)
        }
        return Profile{}, err
    }

    return profile, nil
}

func (p *ProfilePostgres) Update(ctx utils.MyContext, input
Profile) error {
    var (
        queryBuilder strings.Builder
        args          []interface{}
        argIndex      int
    )

    queryBuilder.WriteString("UPDATE profiles SET ")

    if input.Gender != "" {

```

```

        argIndex++
        queryBuilder.WriteString(fmt.Sprintf("gender = %d, ",
argIndex))
        args = append(args, input.Gender)
    }
    if input.Age != 0 {
        argIndex++
        queryBuilder.WriteString(fmt.Sprintf("age = %d, ",
argIndex))
        args = append(args, input.Age)
    }
    if input.Weight != 0 {
        argIndex++
        queryBuilder.WriteString(fmt.Sprintf("weight = %d, ",
argIndex))
        args = append(args, input.Weight)
    }
    if input.Height != 0 {
        argIndex++
        queryBuilder.WriteString(fmt.Sprintf("height = %d, ",
argIndex))
        args = append(args, input.Height)
    }
    if input.Goal != "" {
        argIndex++
        queryBuilder.WriteString(fmt.Sprintf("goal = %d, ",
argIndex))
        args = append(args, input.Goal)
    }

    queryStr := queryBuilder.String()
    queryStr = queryStr[:len(queryStr)-2]

    argIndex++
    queryStr += fmt.Sprintf(" WHERE user_id = %d;", argIndex)
    args = append(args, input.UserID)

    _, err := p.db.ExecContext(ctx.Ctx, queryStr, args...)
    if err != nil {
        if errors.Is(err, sql.ErrNoRows) {
            return fmt.Errorf("profile with user_id %s not found",
input.UserID)
        }
        return fmt.Errorf("failed to update profile: %w", err)
    }

    return nil
}

//go:embed sql/GetUserWeightsByYear.sql
var sqlGetUserWeightsByYear string

func (p *ProfilePostgres) GetUserWeightsByYear(ctx

```

```

utils.MyContext, userID string, date time.Time)
([]model.UserWeight, error) {
    var weights []model.UserWeight

    err := p.db.SelectContext(ctx.Ctx, &weights,
sqlGetUserWeightsByYear, userID, date)
    if err != nil {
        return nil, err
    }

    return weights, nil
}

//go:embed sql/CreateUserWeight.sql
var sqlCreateUserWeight string

func (p *ProfilePostgres) CreateUserWeight(ctx utils.MyContext,
weight model.UserWeight) error {
    weight.ID = uuid.New().String()

    _, err := p.db.ExecContext(ctx.Ctx, sqlCreateUserWeight,
weight.ID, weight.UserID, weight.Date, weight.Weight)
    if err != nil {
        return fmt.Errorf("failed to insert weight: %w", err)
    }

    return nil
}

//go:embed sql/DeleteUserWeight.sql
var sqlDeleteUserWeight string

func (p *ProfilePostgres) DeleteUserWeight(ctx utils.MyContext,
userID, weightID string) error {
    _, err := p.db.ExecContext(ctx.Ctx, sqlDeleteUserWeight,
userID, weightID)
    if err != nil {
        return fmt.Errorf("failed to delete weight: %w", err)
    }

    return nil
}
models.go:

package model

import "time"

type Profile struct {
    UserID string `json:"user_id"`
    Gender string `json:"gender"`
    Age    int    `json:"age"`
    Weight float32 `json:"weight"`
}

```

```

    Height float32 `json:"height"`
    Goal    string  `json:"goal"`
}

type CreateUserWeightRequest struct {
    Date    string  `json:"date"`
    Weight  float32 `json:"weight"`
}

type UserWeight struct {
    ID       string    `json:"id" db:"id"`
    UserID   string    `json:"user_id" db:"user_id"`
    Date     time.Time `json:"date" db:"date"`
    Weight   float32   `json:"weight" db:"weight"`
}

```

Листинг А.3 – Код сервиса питания

main.go:

```

package main

import (
    "context"
    "os"
    "os/signal"
    "syscall"

    "github.com/joho/godotenv"
    "go.uber.org/zap"

    "tube-nutrition/internal/app"
    "tube-nutrition/internal/config"
)

func main() {
    prdLogger, _ := zap.NewProduction()
    defer prdLogger.Sync()
    logger := prdLogger.Sugar()

    mainCtx := context.Background()
    ctx, cancel := context.WithCancel(mainCtx)
    defer cancel()

    err := godotenv.Load()
    if err != nil {
        logger.Warnf(".env file not found: %v", err)
    }

    config, err := config.NewConfig()
    if err != nil {
        logger.Fatalf("failed to read config: %s", err.Error())
    }
}

```

```

newApp := app.NewApp(ctx, logger, config)

if err := newApp.InitDatabase(); err != nil {
    logger.Fatalf("failed to initialize db: %s", err.Error())
}

if err = newApp.RunMigrations(); err != nil {
    logger.Fatalf("failed to run migrations: %s", err.Error())
}

newApp.InitService()

if err = newApp.Run(); err != nil {
    logger.Errorf(err.Error())
    return
}

sigChan := make(chan os.Signal, 1)
signal.Notify(sigChan, syscall.SIGTERM, syscall.SIGINT)
<-sigChan

if err = newApp.Shutdown(); err != nil {
    logger.Errorf(err.Error())
    return
}
}
server.go:
package api

import (
    "net/http"
    "time"
    "tube-nutrition/internal/api/handler"
    "tube-nutrition/internal/config"
    "tube-nutrition/internal/service"

    "github.com/gorilla/handlers"
    "github.com/gorilla/mux"

    "tube-nutrition/internal/api/middleware"
    "tube-nutrition/internal/utils"
)

const (
    maxHeaderBytes = 1 << 20
    readTimeout    = 10 * time.Second
    writeTimeout   = 10 * time.Second
)

type Server struct {
    httpServer *http.Server

```

```

    router      *mux.Router
}

func NewServer(ctx utils.MyContext, config config.Config) *Server
{
    router := mux.NewRouter()

    allowedOrigins :=
handlers.AllowedOrigins([]string{"http://localhost:3000"})
    allowedMethods := handlers.AllowedMethods([]string{"GET",
"POST", "PUT", "DELETE", "OPTIONS"})
    allowedHeaders := handlers.AllowedHeaders([]string{"Content-
Type", "Authorization"})

    corsRouter := handlers.CORS(allowedOrigins, allowedMethods,
allowedHeaders)(router)

    recoveredRouter := middleware.RecoveryMiddleware(ctx,
corsRouter)

    authenticatedRouter := middleware.UserIdentity(ctx,
recoveredRouter)

    return &Server{
        httpServer: &http.Server{
            Addr:            config.ServerPort,
            MaxHeaderBytes: maxHeaderBytes,
            ReadTimeout:    readTimeout,
            WriteTimeout:   writeTimeout,
            Handler:        authenticatedRouter,
        },
        router: router,
    }
}

func (s *Server) Run() error {
    return s.httpServer.ListenAndServe()
}

func (s *Server) Shutdown(ctx utils.MyContext) error {
    return s.httpServer.Shutdown(ctx.Ctx)
}

func (s *Server) HandleAuth(ctx utils.MyContext, service
service.NutritionService) {
    s.router.HandleFunc("/api/nutrition/",
handler.NutritionPlan(ctx, service)).Methods(http.MethodPost)
    s.router.HandleFunc("/api/nutrition/dishes/query-by-ids/",
handler.QueryDishesByIDs(ctx, service)).Methods(http.MethodPost)
    s.router.HandleFunc("/api/nutrition/dishes/search/",
handler.SearchDishes(ctx, service)).Methods(http.MethodGet)
    s.router.HandleFunc("/api/nutrition/dishes/",
handler.GetUserDishes(ctx, service)).Methods(http.MethodGet)
}

```

```

        s.router.HandleFunc("/api/nutrition/dishes/",
handler.CreateUserDish(ctx, service)).Methods(http.MethodPost)
        s.router.HandleFunc("/api/nutrition/dishes/",
handler.DeleteUserDish(ctx, service)).Methods(http.MethodDelete)
    }
models.go:

package api

import (
    "net/http"
    "time"
    "tube-nutrition/internal/api/handler"
    "tube-nutrition/internal/config"
    "tube-nutrition/internal/service"

    "github.com/gorilla/handlers"
    "github.com/gorilla/mux"

    "tube-nutrition/internal/api/middleware"
    "tube-nutrition/internal/utlis"
)

const (
    maxHeaderBytes = 1 << 20
    readTimeout    = 10 * time.Second
    writeTimeout   = 10 * time.Second
)

type Server struct {
    httpServer *http.Server
    router     *mux.Router
}

func NewServer(ctx utils.MyContext, config config.Config) *Server
{
    router := mux.NewRouter()

    allowedOrigins :=
handlers.AllowedOrigins([]string{"http://localhost:3000"})
    allowedMethods := handlers.AllowedMethods([]string{"GET",
"POST", "PUT", "DELETE", "OPTIONS"})
    allowedHeaders := handlers.AllowedHeaders([]string{"Content-
Type", "Authorization"})

    corsRouter := handlers.CORS(allowedOrigins, allowedMethods,
allowedHeaders)(router)

    recoveredRouter := middleware.RecoveryMiddleware(ctx,
corsRouter)

    authenticatedRouter := middleware.UserIdentity(ctx,
recoveredRouter)

```

```

    return &Server{
        httpServer: &http.Server{
            Addr:            config.ServerPort,
            MaxHeaderBytes:  maxHeaderBytes,
            ReadTimeout:     readTimeout,
            WriteTimeout:    writeTimeout,
            Handler:         authenticatedRouter,
        },
        router: router,
    }
}

func (s *Server) Run() error {
    return s.httpServer.ListenAndServe()
}

func (s *Server) Shutdown(ctx utils.MyContext) error {
    return s.httpServer.Shutdown(ctx.Ctx)
}

func (s *Server) HandleAuth(ctx utils.MyContext, service
service.NutritionService) {
    s.router.HandleFunc("/api/nutrition/",
handler.NutritionPlan(ctx, service)).Methods(http.MethodPost)
    s.router.HandleFunc("/api/nutrition/dishes/query-by-ids/",
handler.QueryDishesByIDs(ctx, service)).Methods(http.MethodPost)
    s.router.HandleFunc("/api/nutrition/dishes/search/",
handler.SearchDishes(ctx, service)).Methods(http.MethodGet)
    s.router.HandleFunc("/api/nutrition/dishes/",
handler.GetUserDishes(ctx, service)).Methods(http.MethodGet)
    s.router.HandleFunc("/api/nutrition/dishes/",
handler.CreateUserDish(ctx, service)).Methods(http.MethodPost)
    s.router.HandleFunc("/api/nutrition/dishes/",
handler.DeleteUserDish(ctx, service)).Methods(http.MethodDelete)
}
handler.go:

package handler

import (
    "encoding/json"
    "fmt"
    "github.com/google/uuid"
    "net/http"
    "strconv"
    "strings"
    "tube-nutrition/internal/model"
    nutritionservice "tube-nutrition/internal/service"
    "tube-nutrition/internal/utils"
)

type QueryDishesByIDsRequestBody struct {

```

```

        DishIDs []string `json:"dish_ids"`
    }

    func NutritionPlan(ctx utils.MyContext, service
nutritionservice.NutritionService) http.HandlerFunc {
        return func(w http.ResponseWriter, r *http.Request) {
            authHeader := r.Header.Get("Authorization")
            headerParts := strings.Split(authHeader, " ")
            accessToken := headerParts[1]

            nutritionPlan, err := service.NutritionPlan(ctx,
accessToken)
            if err != nil {
                utils.NewErrorResponse(ctx, w, err.Error(),
http.StatusInternalServerError)
                return
            }

            if err = utils.WriteResponse(w, http.StatusOK,
nutritionPlan); err != nil {
                utils.NewErrorResponse(ctx, w, "internal server error",
http.StatusInternalServerError)
                return
            }
        }
    }

    func QueryDishesByIDs(ctx utils.MyContext, service
nutritionservice.NutritionService) http.HandlerFunc {
        return func(w http.ResponseWriter, r *http.Request) {
            var reqBody QueryDishesByIDsRequestBody
            err := json.NewDecoder(r.Body).Decode(&reqBody)
            if err != nil {
                utils.NewErrorResponse(ctx, w, "invalid request body:
"+err.Error(), http.StatusBadRequest)
                return
            }

            if len(reqBody.DishIDs) == 0 {
                utils.NewErrorResponse(ctx, w, "the list of dish IDs
cannot be empty", http.StatusBadRequest)
                return
            }

            dishes, err := service.GetDishesByIDs(ctx, reqBody.DishIDs)
            if err != nil {
                fmt.Printf("error in the service when receiving dishes
by ID: %v\n", err)
                utils.NewErrorResponse(ctx, w, err.Error(),
http.StatusInternalServerError)
                return
            }
        }
    }

```

```

        if err = utils.WriteResponse(w, http.StatusOK, dishes); err
        != nil {
            utils.NewErrorResponse(ctx, w, "internal server error",
            http.StatusInternalServerError)
            return
        }
    }
}

func SearchDishes(ctx utils.MyContext, service
nutritionservice.NutritionService) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        userID := r.Context().Value("UserID").(string)

        query := r.URL.Query().Get("query")
        if query == "" {
            utils.NewErrorResponse(ctx, w, "the 'query' parameter is
            required", http.StatusInternalServerError)
            return
        }

        limitStr := r.URL.Query().Get("limit")
        limit := 10
        if limitStr != "" {
            parsedLimit, err := strconv.Atoi(limitStr)
            if err == nil && parsedLimit > 0 {
                limit = parsedLimit
            }
        }

        dishes, err := service.SearchDishes(ctx, query, limit,
        userID)
        if err != nil {
            utils.NewErrorResponse(ctx, w, "internal server error",
            http.StatusInternalServerError)
            return
        }

        if err = utils.WriteResponse(w, http.StatusOK, dishes); err
        != nil {
            utils.NewErrorResponse(ctx, w, "internal server error",
            http.StatusInternalServerError)
            return
        }
    }
}

func GetUserDishes(ctx utils.MyContext, service
nutritionservice.NutritionService) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        userID := r.Context().Value("UserID").(string)

        diaryEntries, err := service.GetUserDishes(ctx, userID)

```

```

        if err != nil {
            utils.NewErrorResponse(ctx, w, err.Error(),
http.StatusInternalServerError)
            return
        }

        if err = utils.WriteResponse(w, http.StatusOK,
diaryEntries); err != nil {
            utils.NewErrorResponse(ctx, w, "internal server error",
http.StatusInternalServerError)
            return
        }
    }
}

func CreateUserDish(ctx utils.MyContext, service
nutritionservice.NutritionService) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        var dish model.UserDish
        err := json.NewDecoder(r.Body).Decode(&dish)
        if err != nil {
            utils.NewErrorResponse(ctx, w, "invalid request body:
"+err.Error(), http.StatusBadRequest)
            return
        }

        userID := r.Context().Value("UserID").(string)
        dish.UserID = userID

        err = service.CreateUserDish(ctx, dish)
        if err != nil {
            utils.NewErrorResponse(ctx, w, err.Error(),
http.StatusInternalServerError)
            return
        }

        if err = utils.WriteResponse(w, http.StatusOK,
utils.StatusResponse{Status: "ok"}); err != nil {
            utils.NewErrorResponse(ctx, w, "internal server error",
http.StatusInternalServerError)
            return
        }
    }
}

func DeleteUserDish(ctx utils.MyContext, service
nutritionservice.NutritionService) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        userID := r.Context().Value("UserID").(string)

        dishID := r.URL.Query().Get("id")
        if dishID == "" {
            utils.NewErrorResponse(ctx, w, "the 'id' parameter is

```

```

required", http.StatusInternalServerError)
    return
}

    if dishID == "" {
        utils.NewErrorResponse(ctx, w, "the dishID is missing
from the request", http.StatusBadRequest)
        return
    }

    _, err := uuid.Parse(dishID)
    if err != nil {
        http.Error(w, "invalid dishID format. Expected UUID",
http.StatusBadRequest)
        return
    }

    err = service.DeleteUserDish(ctx, userID, dishID)
    if err != nil {
        utils.NewErrorResponse(ctx, w, err.Error(),
http.StatusInternalServerError)
        return
    }

    if err = utils.WriteResponse(w, http.StatusOK,
utils.StatusResponse{Status: "ok"}); err != nil {
        utils.NewErrorResponse(ctx, w, "internal server error",
http.StatusInternalServerError)
        return
    }
}
}
service.go:

package service

import (
    "encoding/json"
    "fmt"
    "net/http"
    "time"
    "tube-nutrition/internal/database"
    "tube-nutrition/internal/model"
    "tube-nutrition/internal/utils"
)

type NutritionService interface {
    NutritionPlan(ctx utils.MyContext, accessToken string)
(model.Plan, error)
    GetDishesByIDs(ctx utils.MyContext, dishIDs []string)
([]model.Dish, error)
    SearchDishes(ctx utils.MyContext, query string, limit int,
userID string) ([]model.DishSearchResult, error)

```

```

    GetUserDishes(ctx utils.MyContext, userID string)
    ([]model.Dish, error)
    CreateUserDish(ctx utils.MyContext, dish model.UserDish) error
    DeleteUserDish(ctx utils.MyContext, userID, dishID string)
error
}

type ImplNutritionService struct {
    repo      database.NutritionRepository
    jwtSecret []byte
}

func NewNutritionService(repo database.NutritionRepository,
    jwtSecret []byte) *ImplNutritionService {
    return &ImplNutritionService{
        repo:      repo,
        jwtSecret: jwtSecret,
    }
}

func (s *ImplNutritionService) NutritionPlan(ctx utils.MyContext,
    accessToken string) (model.Plan, error) {
    profile, err := FetchUserProfile(accessToken)
    if err != nil {
        return model.Plan{}, fmt.Errorf("failed to fetch user
profile: %w", err)
    }

    totalCalories, proteins, fats, carbohydrates :=
    calculateDailyNeeds(profile)

    minCalories, maxCalories :=
    GetCalorieRangeByHundreds(totalCalories)

    programs, err := s.repo.GetMealPrograms(ctx, minCalories,
    maxCalories)
    if err != nil {
        return model.Plan{}, fmt.Errorf("failed to fetch programs:
%w", err)
    }

    // programs := mapper.FromDBPrograms(dbPrograms)

    plan := model.Plan{
        Calories:      totalCalories,
        Proteins:      proteins,
        Fats:          fats,
        Carbohydrates: carbohydrates,
        Programs:      programs,
    }

    return plan, nil
}

```

```

func (s *ImplNutritionService) GetDishesByIDs(ctx utils.MyContext,
dishIDs []string) ([]model.Dish, error) {
    dishes, err := s.repo.GetDishesByIDs(ctx, dishIDs)
    if err != nil {
        return nil, fmt.Errorf("failed to fetch dishes: %w", err)
    }

    return dishes, nil
}

func (s *ImplNutritionService) SearchDishes(ctx utils.MyContext,
query string, limit int, userID string) ([]model.DishSearchResult,
error) {
    if len(query) < 2 {
        return []model.DishSearchResult{}, nil
    }
    if limit == 0 {
        limit = 10
    }

    dishes, err := s.repo.SearchDishes(ctx, query, limit, userID)
    if err != nil {
        return nil, fmt.Errorf("error in the service when searching
for dishes: %w", err)
    }
    return dishes, nil
}

func (s *ImplNutritionService) GetUserDishes(ctx utils.MyContext,
userID string) ([]model.Dish, error) {
    dishes, err := s.repo.GetUserDishes(ctx, userID)
    if err != nil {
        return nil, fmt.Errorf("failed to fetch user dishes: %w",
err)
    }

    return dishes, nil
}

func (s *ImplNutritionService) CreateUserDish(ctx utils.MyContext,
dish model.UserDish) error {
    err := s.repo.CreateUserDish(ctx, dish)
    if err != nil {
        return fmt.Errorf("failed to create dish: %w", err)
    }

    return nil
}

func (s *ImplNutritionService) DeleteUserDish(ctx utils.MyContext,
userID, dishID string) error {
    err := s.repo.DeleteUserDish(ctx, userID, dishID)

```

```

    if err != nil {
        return fmt.Errorf("failed to delete dish: %w", err)
    }

    return nil
}

func FetchUserProfile(accessToken string) (model.Profile, error) {
    client := &http.Client{
        Timeout: 5 * time.Second,
    }

    req, err := http.NewRequest("GET", fmt.Sprintf("http://tube-
profile:82/api/profile/"), nil)
    if err != nil {
        return model.Profile{}, err
    }

    req.Header.Set("Authorization", "Bearer "+accessToken)

    resp, err := client.Do(req)
    if err != nil {
        return model.Profile{}, err
    }
    defer resp.Body.Close()

    if resp.StatusCode != http.StatusOK {
        return model.Profile{}, fmt.Errorf("failed to fetch user
profile, status code: %d", resp.StatusCode)
    }

    var profile model.Profile
    if err := json.NewDecoder(resp.Body).Decode(&profile); err !=
nil {
        return model.Profile{}, err
    }

    return profile, nil
}

func calculateDailyNeeds(profile model.Profile) (int, float32,
float32, float32) {
    var bmr float32

    if profile.Gender == "m" {
        bmr = 88.36 + (13.4 * profile.Weight) + (4.8 *
profile.Height) - (5.7 * float32(profile.Age))
    } else {
        bmr = 447.6 + (9.2 * profile.Weight) + (3.1 *
profile.Height) - (4.3 * float32(profile.Age))
    }

    var multiplier float32

```

```

switch profile.Goal {
case "gain":
    multiplier = 1.15
case "lose":
    multiplier = 0.85
default:
    multiplier = 1.0
}

totalCalories := int(bmr * multiplier)
proteins := float32(0.3 * float32(totalCalories) / 4)
fats := float32(0.25 * float32(totalCalories) / 9)
carbohydrates := float32(0.45 * float32(totalCalories) / 4)

return totalCalories, proteins, fats, carbohydrates
}

func GetCalorieRangeByHundreds(calories int) (minCalories int,
maxCalories int) {
    minCalories = (calories / 100) * 100

    maxCalories = minCalories + 100

    return minCalories, maxCalories
}
postgres.go:
package database

import (
    "database/sql"
    "embed"
    "errors"
    "fmt"
    "github.com/google/uuid"
    "github.com/jmoiron/sqlx"
    "github.com/lib/pq"
    "tube-nutrition/internal/model"
    "tube-nutrition/internal/utils"
)

type NutritionRepository interface {
    GetMealPrograms(ctx utils.MyContext, minCalories, maxCalories
int) ([]model.MealProgram, error)
    GetDishesByIDs(ctx utils.MyContext, dishIDs []string)
([]model.Dish, error)
    SearchDishes(ctx utils.MyContext, query string, limit int,
userID string) ([]model.DishSearchResult, error)
    GetUserDishes(ctx utils.MyContext, userID string)
([]model.Dish, error)
    CreateUserDish(ctx utils.MyContext, dish model.UserDish) error
    DeleteUserDish(ctx utils.MyContext, userID, dishID string)
error

```

```

}

type NutritionPostgres struct {
    db *sqlx.DB
}

func NewNutritionPostgres(db *sqlx.DB) *NutritionPostgres {
    return &NutritionPostgres{db: db}
}

//go:embed sql/GetMealPrograms.sql
var sqlGetMealPrograms string

func (p *NutritionPostgres) GetMealPrograms(ctx utils.MyContext,
    minCalories, maxCalories int) ([]model.MealProgram, error) {
    var mealPrograms []model.MealProgram

    err := p.db.SelectContext(ctx.Ctx, &mealPrograms,
        sqlGetMealPrograms, minCalories, maxCalories)
    if err != nil {
        return nil, err
    }

    return mealPrograms, nil
}

//go:embed sql/GetDishesByIDs.sql
var sqlGetDishesByIDs string

func (p *NutritionPostgres) GetDishesByIDs(ctx utils.MyContext,
    dishIDs []string) ([]model.Dish, error) {
    var dishes []model.Dish

    err := p.db.SelectContext(ctx.Ctx, &dishes, sqlGetDishesByIDs,
        pq.Array(dishIDs))
    if err != nil {
        if errors.Is(err, sql.ErrNoRows) {
            return []model.Dish{}, nil
        }
        return nil, fmt.Errorf("error when getting dishes by ID:
        %w", err)
    }

    return dishes, nil
}

//go:embed sql/SearchDishes.sql
var sqlSearchDishes string

func (p *NutritionPostgres) SearchDishes(ctx utils.MyContext,
    query string, limit int, userID string) ([]model.DishSearchResult,
    error) {
    var dishes []model.DishSearchResult

```

```

    err := p.db.SelectContext(ctx.Ctx, &dishes, sqlSearchDishes,
query, limit, userID)
    if err != nil {
        if errors.Is(err, sql.ErrNoRows) {
            return []model.DishSearchResult{}, nil
        }
        return nil, fmt.Errorf("error when searching for dishes in
repositories: %w", err)
    }

    return dishes, nil
}

//go:embed sql/GetUserDishes.sql
var sqlGetUserDishes string

func (p *NutritionPostgres) GetUserDishes(ctx utils.MyContext,
userID string) ([]model.Dish, error) {
    var dishes []model.Dish

    err := p.db.SelectContext(ctx.Ctx, &dishes, sqlGetUserDishes,
userID)
    if err != nil {
        if errors.Is(err, sql.ErrNoRows) {
            return []model.Dish{}, nil
        }
        return nil, fmt.Errorf("error when getting user dishes:
%w", err)
    }

    return dishes, nil
}

//go:embed sql/CreateUserDish.sql
var sqlCreateUserDish string

func (p *NutritionPostgres) CreateUserDish(ctx utils.MyContext,
dish model.UserDish) error {
    dish.ID = uuid.New().String()

    _, err := p.db.ExecContext(ctx.Ctx, sqlCreateUserDish,
dish.ID, dish.Name, dish.Description, dish.Calories,
dish.Proteins, dish.Fats, dish.Carbohydrates, dish.UserID)
    if err != nil {
        return fmt.Errorf("failed to insert user dish: %w", err)
    }

    return nil
}

//go:embed sql/DeleteUserDish.sql
var sqlDeleteUserDish string

```

```

func (p *NutritionPostgres) DeleteUserDish(ctx utils.MyContext,
userID, dishID string) error {
    _, err := p.db.ExecContext(ctx.Ctx, sqlDeleteUserDish, userID,
dishID)
    if err != nil {
        return err
    }

    return nil
}

```

Листинг А.4 – Код сервиса дневника

main.go:

```

package main

import (
    "context"
    "os"
    "os/signal"
    "syscall"

    "github.com/joho/godotenv"
    "go.uber.org/zap"

    "tube-diary/internal/app"
    "tube-diary/internal/config"
)

func main() {
    prdLogger, _ := zap.NewProduction()
    defer prdLogger.Sync()
    logger := prdLogger.Sugar()

    mainCtx := context.Background()
    ctx, cancel := context.WithCancel(mainCtx)
    defer cancel()

    err := godotenv.Load()
    if err != nil {
        logger.Warnf(".env file not found: %v", err)
    }

    config, err := config.NewConfig()
    if err != nil {
        logger.Fatalf("failed to read config: %s", err.Error())
    }

    newApp := app.NewApp(ctx, logger, config)

    if err := newApp.InitDatabase(); err != nil {
        logger.Fatalf("failed to initialize db: %s", err.Error())
    }
}

```

```

    }

    if err = newApp.RunMigrations(); err != nil {
        logger.Fatalf("failed to run migrations: %s", err.Error())
    }

    newApp.InitService()

    if err = newApp.Run(); err != nil {
        logger.Errorf(err.Error())
        return
    }

    sigChan := make(chan os.Signal, 1)
    signal.Notify(sigChan, syscall.SIGTERM, syscall.SIGINT)
    <-sigChan

    if err = newApp.Shutdown(); err != nil {
        logger.Errorf(err.Error())
        return
    }
}
server.go:
package api

import (
    "net/http"
    "time"
    "tube-diary/internal/api/handler"
    "tube-diary/internal/config"
    "tube-diary/internal/service"

    "github.com/gorilla/handlers"
    "github.com/gorilla/mux"

    "tube-diary/internal/api/middleware"
    "tube-diary/internal/utils"
)

const (
    maxHeaderBytes = 1 << 20
    readTimeout    = 10 * time.Second
    writeTimeout   = 10 * time.Second
)

type Server struct {
    httpServer *http.Server
    router     *mux.Router
}

func NewServer(ctx utils.MyContext, config config.Config) *Server
{

```

```

    router := mux.NewRouter()

    allowedOrigins :=
handlers.AllowedOrigins([]string{"http://localhost:3000"})
    allowedMethods := handlers.AllowedMethods([]string{"GET",
"POST", "PUT", "DELETE", "OPTIONS"})
    allowedHeaders := handlers.AllowedHeaders([]string{"Content-
Type", "Authorization"})

    corsRouter := handlers.CORS(allowedOrigins, allowedMethods,
allowedHeaders)(router)

    recoveredRouter := middleware.RecoveryMiddleware(ctx,
corsRouter)

    authenticatedRouter := middleware.UserIdentity(ctx,
recoveredRouter)

    return &Server{
        httpServer: &http.Server{
            Addr:            config.ServerPort,
            MaxHeaderBytes: maxHeaderBytes,
            ReadTimeout:     readTimeout,
            WriteTimeout:    writeTimeout,
            Handler:         authenticatedRouter,
        },
        router: router,
    }
}

func (s *Server) Run() error {
    return s.httpServer.ListenAndServe()
}

func (s *Server) Shutdown(ctx utils.MyContext) error {
    return s.httpServer.Shutdown(ctx.Ctx)
}

func (s *Server) HandleAuth(ctx utils.MyContext, service
service.DiaryService) {
    s.router.HandleFunc("/api/diary/entries/",
handler.GetUserDiaryEntriesByDate(ctx,
service)).Methods(http.MethodGet)
    s.router.HandleFunc("/api/diary/entries/",
handler.CreateDiaryEntry(ctx, service)).Methods(http.MethodPost)
    s.router.HandleFunc("/api/diary/entries/",
handler.DeleteDiaryEntry(ctx, service)).Methods(http.MethodDelete)
}
models.go:
package model

type DiaryEntry struct {

```

```

    ID      string `json:"id" db:"id"`
    UserID   string `json:"user_id" db:"user_id"`
    DishID   string `json:"dish_id" db:"dish_id"`
    MealTime string `json:"meal_time" db:"meal_time"`
    Date     string `json:"date" db:"date"`
    Weight   int    `json:"weight" db:"weight"`
}

type DiaryEntryWithDish struct {
    ID      string `json:"id"`
    UserID   string `json:"user_id"`
    Dish     Dish   `json:"dish"`
    MealTime string `json:"meal_time"`
    Date     string `json:"date"`
    Weight   int    `json:"weight"`
}

type Dish struct {
    ID            string `json:"id"`
    Name          string `json:"name"`
    Description    string `json:"description"`
    Calories      int    `json:"calories"`
    Proteins      float64 `json:"proteins"`
    Fats          float64 `json:"fats"`
    Carbohydrates float64 `json:"carbohydrates"`
}

service.go:

package service

import (
    "bytes"
    "encoding/json"
    "fmt"
    "net/http"
    "time"
    "tube-diary/internal/database"
    "tube-diary/internal/model"
    "tube-diary/internal/utils"
)

type DiaryService interface {
    GetUserDiaryEntriesByDate(ctx utils.MyContext, accessToken
string, userID string, date time.Time)
    ([]model.DiaryEntryWithDish, error)
    CreateDiaryEntry(ctx utils.MyContext, diaryEntry
model.DiaryEntry) error
    DeleteDiaryEntry(ctx utils.MyContext, userID, entryID string)
error
}

type ImplDiaryService struct {
    repo      database.DiaryRepository

```

```

    jwtSecret []byte
}

func NewDiaryService(repo database.DiaryRepository, jwtSecret
[]byte) *ImplDiaryService {
    return &ImplDiaryService{
        repo:      repo,
        jwtSecret: jwtSecret,
    }
}

func (s *ImplDiaryService) GetUserDiaryEntriesByDate(ctx
utils.MyContext, accessToken string, userID string, date
time.Time) ([]model.DiaryEntryWithDish, error) {
    diaryEntries, err := s.repo.GetUserDiaryEntriesByDate(ctx,
userID, date)
    if err != nil {
        return nil, fmt.Errorf("failed to fetch diary entries: %w",
err)
    }

    diaryEntriesWithDishes, err := FetchDishDetails(ctx,
accessToken, diaryEntries)
    if err != nil {
        return nil, fmt.Errorf("failed to fetch diary entries: %w",
err)
    }

    return diaryEntriesWithDishes, nil
}

func (s *ImplDiaryService) CreateDiaryEntry(ctx utils.MyContext,
diaryEntry model.DiaryEntry) error {
    err := s.repo.CreateDiaryEntry(ctx, diaryEntry)
    if err != nil {
        return fmt.Errorf("failed to create diary entry: %w", err)
    }

    return nil
}

func (s *ImplDiaryService) DeleteDiaryEntry(ctx utils.MyContext,
userID, entryID string) error {
    err := s.repo.DeleteDiaryEntry(ctx, userID, entryID)
    if err != nil {
        return fmt.Errorf("failed to delete diary entry: %w", err)
    }

    return nil
}

func FetchDishDetails(ctx utils.MyContext, accessToken string,
entries []model.DiaryEntry) ([]model.DiaryEntryWithDish, error) {

```

```

if len(entries) == 0 {
    return []model.DiaryEntryWithDish{}, nil
}

uniqueDishIDsMap := make(map[string]struct{})
for _, entry := range entries {
    uniqueDishIDsMap[entry.DishID] = struct{}{}
}

dishIDsToFetch := make([]string, 0, len(uniqueDishIDsMap))
for id := range uniqueDishIDsMap {
    dishIDsToFetch = append(dishIDsToFetch, id)
}

requestBodyData := map[string][]string{"dish_ids":
dishIDsToFetch}
requestBodyBytes, err := json.Marshal(requestBodyData)
if err != nil {
    return nil, fmt.Errorf("error serializing dish ID for
request Nutrition: %w", err)
}

req, err := http.NewRequestWithContext(ctx.Ctx, "POST",
fmt.Sprintf("http://tube-nutrition:83/api/nutrition/dishes/query-
by-ids/"), bytes.NewBuffer(requestBodyBytes))
if err != nil {
    return nil, fmt.Errorf("error creating request to
Nutrition: %w", err)
}
req.Header.Set("Authorization", "Bearer "+accessToken)

client := &http.Client{Timeout: 5 * time.Second}
resp, err := client.Do(req)
if err != nil {
    return nil, fmt.Errorf("error while making request to
Nutrition: %w", err)
}
defer resp.Body.Close()

if resp.StatusCode != http.StatusOK {
    return nil, fmt.Errorf("nutrition service returned error:
status %d", resp.StatusCode)
}

var dishesFromNutrition []model.Dish
if err :=
json.NewDecoder(resp.Body).Decode(&dishesFromNutrition); err !=
nil {
    return nil, fmt.Errorf("error decoding response from
Nutrition: %w", err)
}

dishesMap := make(map[string]model.Dish)

```

```

    for _, dish := range dishesFromNutrition {
        dishesMap[dish.ID] = dish
    }

    enrichedEntries := make([]model.DiaryEntryWithDish, 0,
len(entries))
    for _, entry := range entries {
        dishDetails, found := dishesMap[entry.DishID]
        if !found {
            fmt.Printf("Warning: Food item with ID '%s' for diary
entry not found in Nutrition service.\n", entry.DishID)
            continue
        }

        enrichedEntries = append(enrichedEntries,
model.DiaryEntryWithDish{
            ID:        entry.ID,
            UserID:    entry.UserID,
            Dish:      dishDetails,
            MealTime:  entry.MealTime,
            Date:      entry.Date,
            Weight:    entry.Weight,
        })
    }

    return enrichedEntries, nil
}

postgres.go:
package database

import (
    _ "embed"
    "fmt"
    "github.com/google/uuid"
    "github.com/jmoiron/sqlx"
    "time"
    "tube-diary/internal/model"
    "tube-diary/internal/utils"
)

type DiaryRepository interface {
    GetUserDiaryEntriesByDate(ctx utils.MyContext, userID string,
date time.Time) ([]model.DiaryEntry, error)
    CreateDiaryEntry(ctx utils.MyContext, entry model.DiaryEntry)
error
    DeleteDiaryEntry(ctx utils.MyContext, userID, entryID string)
error
}

type DiaryPostgres struct {
    db *sqlx.DB
}

```

```

func NewDiaryPostgres(db *sqlx.DB) *DiaryPostgres {
    return &DiaryPostgres{db: db}
}

//go:embed sql/GetUserDiaryEntriesByDate.sql
var sqlGetUserDiaryEntriesByDate string

func (p *DiaryPostgres) GetUserDiaryEntriesByDate(ctx
utils.MyContext, userID string, date time.Time)
([]model.DiaryEntry, error) {
    var diaryEntries []model.DiaryEntry

    err := p.db.SelectContext(ctx.Ctx, &diaryEntries,
sqlGetUserDiaryEntriesByDate, userID, date)
    if err != nil {
        return nil, err
    }

    return diaryEntries, nil
}

//go:embed sql/CreateDiaryEntry.sql
var sqlCreateDiaryEntry string

func (p *DiaryPostgres) CreateDiaryEntry(ctx utils.MyContext,
entry model.DiaryEntry) error {
    entry.ID = uuid.New().String()

    _, err := p.db.ExecContext(ctx.Ctx, sqlCreateDiaryEntry,
entry.ID, entry.UserID, entry.DishID, entry.MealTime, entry.Date,
entry.Weight)
    if err != nil {
        return fmt.Errorf("failed to insert diary entry: %w", err)
    }

    return nil
}

//go:embed sql/DeleteDiaryEntry.sql
var sqlDeleteDiaryEntry string

func (p *DiaryPostgres) DeleteDiaryEntry(ctx utils.MyContext,
userID, entryID string) error {
    _, err := p.db.ExecContext(ctx.Ctx, sqlDeleteDiaryEntry,
userID, entryID)
    if err != nil {
        return err
    }

    return nil
}

```


Приложение Б
(рекомендуемое)
Листинг SQL-запросов

Листинг Б.1 – Миграции инициализация таблиц базы данных сервиса авторизации

```
-- +goose Up
-- +goose StatementBegin
CREATE TABLE users (
    id UUID PRIMARY KEY,
    username TEXT NOT NULL,
    email TEXT NOT NULL UNIQUE,
    password_hash TEXT NOT NULL,
    is_verified BOOLEAN DEFAULT FALSE NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE refresh_tokens (
    id UUID PRIMARY KEY,
    user_id UUID NOT NULL REFERENCES users(id),
    token_hash TEXT NOT NULL,
    expires_at TIMESTAMP NOT NULL
);

CREATE TABLE verification_codes (
    id UUID PRIMARY KEY,
    user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    code TEXT NOT NULL,
    expires_at TIMESTAMP NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
-- +goose StatementEnd

-- +goose Down
-- +goose StatementBegin
DROP TABLE refresh_tokens;
DROP TABLE users;
-- +goose StatementEnd
```

Листинг Б.2 – Миграции инициализации таблиц базы данных сервиса профиля

```
-- +goose Up
-- +goose StatementBegin
CREATE TABLE profiles (
    id UUID PRIMARY KEY,
    user_id UUID UNIQUE NOT NULL,
    gender VARCHAR(1) NOT NULL,
    age INT NOT NULL,
    weight FLOAT NOT NULL,
    height FLOAT NOT NULL,
    goal VARCHAR(30) NOT NULL -- (gain, maintain, lose)
);

CREATE TABLE user_weights(
    id UUID PRIMARY KEY,
    user_id UUID NOT NULL,
```

```

        date TIMESTAMP NOT NULL,
        weight FLOAT NOT NULL
    );
-- +goose StatementEnd

-- +goose Down
-- +goose StatementBegin
DROP TABLE profiles;
-- +goose StatementEnd

```

Листинг Б.3 – Миграции инициализации таблиц базы данных сервиса питания

```

-- +goose Up
-- +goose StatementBegin
CREATE EXTENSION IF NOT EXISTS "uuid-osspl";

CREATE TABLE dishes (
    id UUID PRIMARY KEY,
    name TEXT NOT NULL,
    description TEXT,
    calories INT NOT NULL,
    proteins FLOAT NOT NULL,
    fats FLOAT NOT NULL,
    carbohydrates FLOAT NOT NULL,
    user_id UUID
);

CREATE TABLE meal_programs (
    id UUID PRIMARY KEY,
    name TEXT NOT NULL,
    description TEXT NOT NULL,
    calories INT NOT NULL DEFAULT 0,
    proteins FLOAT NOT NULL DEFAULT 0,
    fats FLOAT NOT NULL DEFAULT 0,
    carbohydrates FLOAT NOT NULL DEFAULT 0
);

CREATE TABLE meal_programs_dishes (
    id UUID PRIMARY KEY,
    meal_program_id UUID NOT NULL REFERENCES meal_programs(id),
    dish_id UUID NOT NULL REFERENCES dishes(id),
    meal_time TEXT NOT NULL,
    order_index INT NOT NULL,
    weight INT NOT NULL
);
-- +goose StatementEnd

-- +goose Down
-- +goose StatementBegin
DROP TABLE meal_programs_dishes;

DROP TABLE dishes;

```

```
DROP TABLE meal_programs;
```

```
DROP EXTENSION IF EXISTS "uuid-ossf";
-- +goose StatementEnd
```

Листинг Б.3 – Миграции инициализации таблиц базы данных сервиса питания

```
-- +goose Up
-- +goose StatementBegin
CREATE EXTENSION IF NOT EXISTS "uuid-ossf";

CREATE TABLE dishes (
    id UUID PRIMARY KEY,
    name TEXT NOT NULL,
    description TEXT,
    calories INT NOT NULL,
    proteins FLOAT NOT NULL,
    fats FLOAT NOT NULL,
    carbohydrates FLOAT NOT NULL,
    user_id UUID
);

CREATE TABLE meal_programs (
    id UUID PRIMARY KEY,
    name TEXT NOT NULL,
    description TEXT NOT NULL,
    calories INT NOT NULL DEFAULT 0,
    proteins FLOAT NOT NULL DEFAULT 0,
    fats FLOAT NOT NULL DEFAULT 0,
    carbohydrates FLOAT NOT NULL DEFAULT 0
);

CREATE TABLE meal_programs_dishes (
    id UUID PRIMARY KEY,
    meal_program_id UUID NOT NULL REFERENCES meal_programs(id),
    dish_id UUID NOT NULL REFERENCES dishes(id),
    meal_time TEXT NOT NULL,
    order_index INT NOT NULL,
    weight INT NOT NULL
);
-- +goose StatementEnd

-- +goose Down
-- +goose StatementBegin
DROP TABLE meal_programs_dishes;

DROP TABLE dishes;

DROP TABLE meal_programs;

DROP EXTENSION IF EXISTS "uuid-ossf";
-- +goose StatementEnd
```

Листинг Б.4 – Фрагмент миграции заполнения таблицы dishes начальными данными

```

INSERT INTO dishes (id, name, description, calories, proteins,
fats, carbohydrates)
VALUES
-- Крупы и каши
    (uuid_generate_v4(), 'Гречка отварная', 'Гречневая каша на
воде без масла', 101, 3.6, 0.7, 20.0),
    (uuid_generate_v4(), 'Рис белый отварной', 'Белый рис вареный
без добавок', 130, 2.7, 0.3, 28.0),
    (uuid_generate_v4(), 'Рис бурый отварной', 'Бурый рис вареный
без масла', 111, 2.6, 0.9, 23.0),
    (uuid_generate_v4(), 'Рис басмати отварной', 'Рис басмати
вареный', 121, 3.0, 0.4, 25.2),
    (uuid_generate_v4(), 'Овсянка на воде', 'Овсяная каша без
сахара и масла', 88, 3.0, 1.7, 15.0),
    (uuid_generate_v4(), 'Перловка отварная', 'Перловая крупа на
воде', 109, 3.1, 0.4, 22.2),
    (uuid_generate_v4(), 'Пшенная каша', 'Пшено вареное на воде',
90, 3.0, 1.0, 17.0),
    (uuid_generate_v4(), 'Кукурузная каша', 'Каша из кукурузной
крупы', 86, 2.3, 0.4, 19.6),
    (uuid_generate_v4(), 'Киноа отварная', 'Киноа приготовленная
на воде', 120, 4.4, 1.9, 21.3),
    (uuid_generate_v4(), 'Булгур отварной', 'Булгур вареный без
масла', 83, 3.1, 0.2, 18.6),
    (uuid_generate_v4(), 'Ячневая каша', 'Ячневая крупа на воде',
76, 2.3, 0.3, 15.7),
    (uuid_generate_v4(), 'Полба отварная', 'Каша из полбы', 127,
5.5, 0.9, 26.4),
    (uuid_generate_v4(), 'Чечевица отварная', 'Чечевица вареная на
воде', 116, 9.0, 0.4, 20.1),
    (uuid_generate_v4(), 'Нут отварной', 'Нут вареный без
добавок', 164, 8.9, 2.6, 27.4),
    (uuid_generate_v4(), 'Фасоль отварная', 'Фасоль вареная без
масла', 127, 8.7, 0.5, 22.8),

-- Макароны изделия
    (uuid_generate_v4(), 'Макароны отварные', 'Макароны из твердых
сортов пшеницы', 160, 5.5, 1.1, 30.0),
    (uuid_generate_v4(), 'Спагетти аль денте', 'Спагетти вареные
до состояния аль денте', 158, 5.8, 1.1, 30.5),
    (uuid_generate_v4(), 'Лапша гречневая', 'Гречневая лапша
(соба)', 99, 5.0, 0.1, 21.4),
    (uuid_generate_v4(), 'Лапша рисовая', 'Рисовая лапша вареная',
109, 0.9, 0.2, 25.2),
    (uuid_generate_v4(), 'Фарфалле отварные', 'Бантики из твердых
сортов пшеницы', 158, 5.8, 1.2, 30.0),

-- Картофель и корнеплоды
    (uuid_generate_v4(), 'Картофель отварной', 'Картофель вареный

```

```

в мундире', 82, 2.0, 0.1, 17.0),
  (uuid_generate_v4(), 'Картофельное пюре', 'Пюре на воде без
масла', 88, 2.1, 0.2, 19.0),
  (uuid_generate_v4(), 'Картофель запеченный', 'Картофель
запеченный без масла', 93, 2.5, 0.1, 21.0),
  (uuid_generate_v4(), 'Батат запеченный', 'Запеченный сладкий
картофель', 90, 2.0, 0.2, 20.7),
  (uuid_generate_v4(), 'Морковь тушеная', 'Морковь тушеная без
масла', 54, 1.2, 0.3, 12.0),
  (uuid_generate_v4(), 'Свекла отварная', 'Свекла вареная без
добавок', 44, 1.7, 0.2, 10.0),
  (uuid_generate_v4(), 'Тыква запеченная', 'Тыква запеченная без
сахара', 26, 1.0, 0.1, 6.5),

-- Овощные гарниры
  (uuid_generate_v4(), 'Брокколи на пару', 'Брокколи
приготовленная на пару', 35, 2.4, 0.4, 6.6),
  (uuid_generate_v4(), 'Цветная капуста на пару', 'Цветная
капуста приготовленная на пару', 25, 2.0, 0.3, 4.1),
  (uuid_generate_v4(), 'Кабачки тушеные', 'Кабачки тушеные без
масла', 24, 1.1, 0.3, 4.6),
  (uuid_generate_v4(), 'Баклажаны запеченные', 'Баклажаны
запеченные без масла', 25, 1.0, 0.2, 6.0),
  (uuid_generate_v4(), 'Спаржа на пару', 'Спаржа зеленая
приготовленная на пару', 22, 2.4, 0.2, 3.9),
  (uuid_generate_v4(), 'Шпинат тушеный', 'Шпинат тушеный без
масла', 23, 2.9, 0.4, 3.6),
  (uuid_generate_v4(), 'Стручковая фасоль на пару', 'Зеленая
стручковая фасоль', 31, 1.8, 0.2, 7.0),
  (uuid_generate_v4(), 'Брюссельская капуста тушеная',
'Брюссельская капуста без масла', 43, 3.4, 0.3, 9.0),
  (uuid_generate_v4(), 'Морковь по-корейски', 'Острая морковь с
специями', 112, 1.2, 8.5, 9.2),
  (uuid_generate_v4(), 'Капуста тушеная', 'Белокочанная капуста
тушеная', 42, 1.8, 1.4, 6.4),

```

Листинг Б.5 – Фрагмент миграции заполнения таблицы meal_programs

начальными данными

```

-- +goose Up
-- +goose StatementBegin
WITH
  meal_plan_data (
    program_name,
    program_description,
    meal_time_name,
    dish_name,
    order_index,
    weight
  ) AS (
    VALUES
      -- Рацион 1: Классический сбалансированный

```

```

        ('Классический сбалансированный', 'Полноценное питание
с основными приемами пищи', 'Завтрак', 'Овсянка на воде', 1, 150),
        ('Классический сбалансированный', 'Полноценное питание
с основными приемами пищи', 'Завтрак', 'Яйцо куриное вареное
вкрутую', 2, 55),
        ('Классический сбалансированный', 'Полноценное питание
с основными приемами пищи', 'Обед', 'Гречка отварная', 1, 150),
        ('Классический сбалансированный', 'Полноценное питание
с основными приемами пищи', 'Обед', 'Куриная грудка отварная', 2,
120),
        ('Классический сбалансированный', 'Полноценное питание
с основными приемами пищи', 'Обед', 'Брокколи на пару', 3, 100),
        ('Классический сбалансированный', 'Полноценное питание
с основными приемами пищи', 'Ужин', 'Треска запеченная', 1, 150),
        ('Классический сбалансированный', 'Полноценное питание
с основными приемами пищи', 'Ужин', 'Салат из свежих овощей', 2,
150),
    ),
    calculated_program_nutrients AS (
        SELECT
            mpd.program_name,
            mpd.program_description,
            SUM(d.calories * mpd.weight / 100.0) AS
total_calories,
            SUM(d.proteins * mpd.weight / 100.0) AS
total_proteins,
            SUM(d.fats * mpd.weight / 100.0) AS total_fats,
            SUM(d.carbohydrates * mpd.weight / 100.0) AS
total_carbohydrates
        FROM
            meal_plan_data AS mpd
            JOIN dishes d ON mpd.dish_name = d.name
        GROUP BY
            mpd.program_name,
            mpd.program_description
    ),
    inserted_programs AS (
INSERT INTO meal_programs (id, name, description, calories,
proteins, fats, carbohydrates)
SELECT
    uuid_generate_v4 (),
    cpn.program_name,
    cpn.program_description,
    cpn.total_calories,
    cpn.total_proteins,
    cpn.total_fats,
    cpn.total_carbohydrates
FROM
    calculated_program_nutrients AS cpn
GROUP BY
    cpn.program_name,
    cpn.program_description,
    cpn.total_calories,

```

```

        cpn.total_proteins,
        cpn.total_fats,
        cpn.total_carbohydrates
    RETURNING
        id,
        name
    )
INSERT INTO
    meal_programs_dishes (
        id,
        meal_program_id,
        dish_id,
        meal_time,
        order_index,
        weight
    )
SELECT
    uuid_generate_v4(),
    ip.id,
    d.id,
    mpd.meal_time_name,
    mpd.order_index,
    mpd.weight
FROM
    meal_plan_data AS mpd
    JOIN inserted_programs ip ON mpd.program_name = ip.name
    JOIN dishes d ON mpd.dish_name = d.name;
-- +goose StatementEnd

```

Листинг Б.6 – Миграции инициализации таблиц базы данных сервиса дневника

```

-- +goose Up
-- +goose StatementBegin
CREATE TABLE diary_entries (
    id UUID PRIMARY KEY,
    user_id UUID NOT NULL,
    dish_id UUID NOT NULL,
    meal_time TEXT NOT NULL,
    date TIMESTAMP NOT NULL,
    weight INT NOT NULL
);
-- +goose StatementEnd

```

Листинг Б.7 – SQL-запросы сервиса авторизации

```

INSERT INTO users (id, username, email, password_hash) VALUES ($1,
    $2, $3, $4)

INSERT INTO verification_codes (id, user_id, code, expires_at,
    created_at)
VALUES ($1, $2, $3, $4, $5);

DELETE FROM refresh_tokens
WHERE user_id = $1

```

```
DELETE FROM verification_codes
WHERE user_id = $1;
```

```
SELECT
    id,
    user_id,
    code,
    expires_at,
    created_at
FROM
    verification_codes
WHERE
    user_id = $1 AND expires_at > NOW()
ORDER BY
    created_at DESC
LIMIT 1;
```

```
SELECT token_hash
FROM refresh_tokens
WHERE id = $1;
```

```
SELECT id, username, email, password_hash, is_verified FROM users
WHERE email=$1
```

```
INSERT INTO refresh_tokens (id, user_id, token_hash, expires_at)
VALUES ($1, $2, $3, $4)
```

```
UPDATE users
SET
    username = $2,
    password_hash = $3,
    created_at = $4
WHERE
    id = $1;
```

```
UPDATE users
SET
    is_verified = $2
WHERE
    id = $1;
```

Листинг Б.8 – SQL-запросы сервиса профиля

```
INSERT INTO profiles (id, user_id, gender, age, weight, height,
goal)
VALUES ($1, $2, $3, $4, $5, $6, $7)
```

```
INSERT INTO user_weights (id, user_id, date, weight)
VALUES ($1, $2, $3, $4);
```

```
DELETE FROM user_weights
WHERE id = $2 AND user_id = $1;
```

```
SELECT user_id, gender, age, weight, height, goal
FROM profiles
WHERE user_id = $1
```

```
SELECT
    id,
    date,
    weight,
    user_id
FROM
    user_weights
WHERE
    user_id = $1 AND EXTRACT(YEAR FROM date) = EXTRACT(YEAR FROM
    $2::timestamp)
ORDER BY

    date ASC;
```

Листинг Б.9 – SQL-запросы сервиса питания

```
INSERT INTO dishes (id, name, description, calories, proteins,
fats, carbohydrates, user_id)
VALUES ($1, $2, $3, $4, $5, $6, $7, $8);
```

```
DELETE FROM dishes
WHERE user_id = $1 AND id = $2;
```

```
SELECT
    id,
    name,
    description,
    calories,
    proteins,
    fats,
    carbohydrates
FROM
    dishes
WHERE
    id = ANY($1::uuid[]);
```

```
SELECT
    mp.id,
    mp.name,
    mp.description,
    mp.calories,
    mp.proteins,
    mp.fats,
    mp.carbohydrates,
    -- COALESCE гарантирует, что если у программы нет блюд,
    вернется пустой массив '[]', а не NULL.
    COALESCE(
        json_agg(
```

```

        json_build_object(
            'dish_id', d.id,
            'dish_name', d.name,
            'meal_time', mpd.meal_time,
            'weight', mpd.weight,
            'order_index', mpd.order_index,
            'calories', d.calories,
            'proteins', d.proteins,
            'fats', d.fats,
            'carbohydrates', d.carbohydrates
        )
        ORDER BY
            mpd.order_index
    ) FILTER (
        WHERE d.id IS NOT NULL
    ),
    '[]' :: json
) AS dishes
FROM
    meal_programs mp
    LEFT JOIN meal_programs_dishes mpd ON mp.id =
mpd.meal_program_id
    LEFT JOIN dishes d ON mpd.dish_id = d.id
WHERE
    mp.calories BETWEEN $1 AND $2
GROUP BY
    mp.id;

SELECT
    id,
    name,
    description,
    calories,
    proteins,
    fats,
    carbohydrates
FROM
    dishes
WHERE
    user_id = $1;

SELECT
    id,
    name
FROM
    dishes
WHERE
    name ILIKE '%' || $1 || '%' -- ILIKE для регистронезависимого
поиска, '%' с обеих сторон для поиска фрагмента в любой части
    AND (user_id IS NULL OR user_id = $3)
ORDER BY
    name
LIMIT $2;

```

Листинг Б.10 – SQL-запросы сервиса дневника

```
INSERT INTO diary_entries (id, user_id, dish_id, meal_time, date,  
weight)  
VALUES ($1, $2, $3, $4, $5, $6)
```

```
DELETE FROM diary_entries  
WHERE user_id = $1 AND id = $2;
```

```
SELECT  
    id,  
    user_id,  
    dish_id,  
    meal_time,  
    date,  
    weight  
FROM  
    diary_entries  
WHERE  
    user_id = $1 AND date = $2;
```

Приложение В
(рекомендуемое)
Листинг Docker-файлов

Листинг В.1 – Листинг Docker-файла для контейнеризации отдельных сервисов

```
FROM golang:1.23.4-alpine

RUN go version

ENV GOPATH=/

COPY . .

RUN go mod download

RUN go build -o app ./cmd

CMD ["/app"]
```

Листинг В.2 – Листинг Docker-Compose файла управления контейнерами сервиса

```
services:
  tube-auth:
    container_name: tube-auth
    build:
      context: ./tube-auth
    ports:
      - "81:81"
    depends_on:
      auth-db:
        condition: service_healthy

  auth-db:
    container_name: auth-db
    image: postgres:16
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: qwerty
      POSTGRES_DB: users
    volumes:
      - ./tube-
auth/.database/postgres/data:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    healthcheck:
      test: [ "CMD-SHELL", "pg_isready -U user -d users" ]
      interval: 10s
      timeout: 5s
      retries: 5

  tube-profile:
    container_name: tube-profile
    build:
      context: ./tube-profile
```

```

    ports:
      - "82:82"
    depends_on:
      profile-db:
        condition: service_healthy
profile-db:
  container_name: profile-db
  image: postgres:16
  environment:
    POSTGRES_USER: user
    POSTGRES_PASSWORD: qwerty
    POSTGRES_DB: profiles
  volumes:
    - ./tube-
profile/.database/postgres/data:/var/lib/postgresql/data
  ports:
    - "5433:5432"
  healthcheck:
    test: [ "CMD-SHELL", "pg_isready -U user -d profiles" ]
    interval: 10s
    timeout: 5s
    retries: 5

tube-nutrition:
  container_name: tube-nutrition
  build:
    context: ./tube-nutrition
  ports:
    - "83:83"
  depends_on:
    nutrition-db:
      condition: service_healthy

nutrition-db:
  container_name: nutrition-db
  image: postgres:16
  environment:
    POSTGRES_USER: user
    POSTGRES_PASSWORD: qwerty
    POSTGRES_DB: dishes
  volumes:
    - ./tube-
nutrition/.database/postgres/data:/var/lib/postgresql/data
  ports:
    - "5434:5432"
  healthcheck:
    test: [ "CMD-SHELL", "pg_isready -U user -d dishes" ]
    interval: 10s
    timeout: 5s
    retries: 5
tube-diary:
  container_name: tube-diary
  build:
    context: ./tube-diary

```

```
ports:
  - "84:84"
depends_on:
  diary-db:
    condition: service_healthy

diary-db:
  container_name: diary-db
  image: postgres:16
  environment:
    POSTGRES_USER: user
    POSTGRES_PASSWORD: qwerty
    POSTGRES_DB: diaries
  volumes:
    - ./tube-
diary/.database/postgres/data:/var/lib/postgresql/data
  ports:
    - "5435:5432"
  healthcheck:
    test: [ "CMD-SHELL", "pg_isready -U user -d diaries" ]
    interval: 10s
    timeout: 5s
    retries: 5
```

Приложение Г
(рекомендуемое)
Диаграммы классов

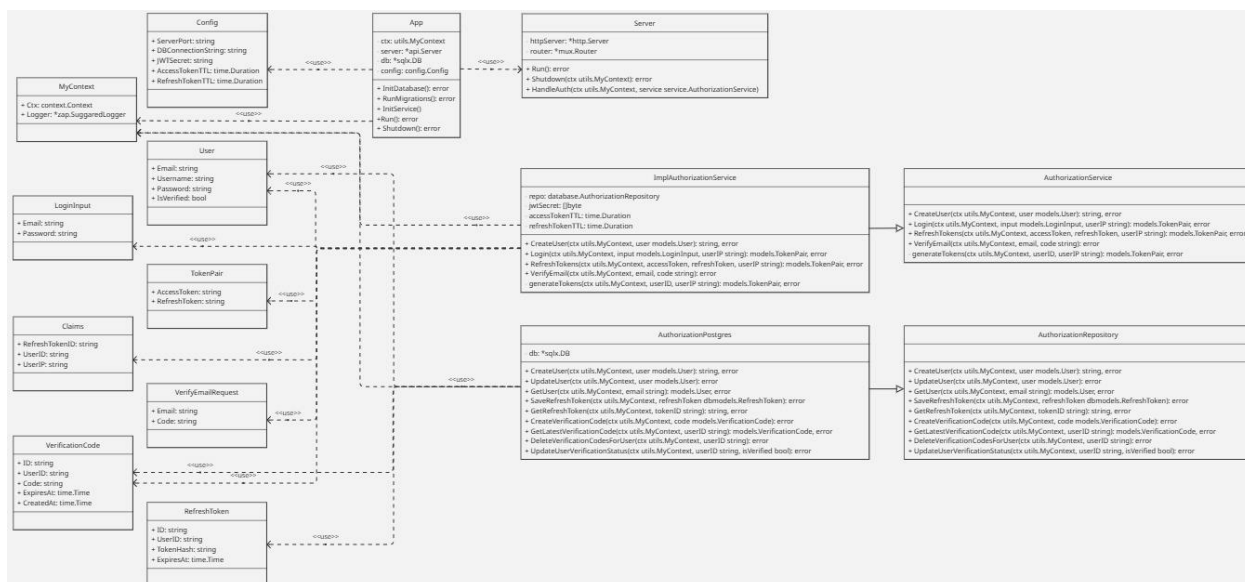


Рисунок Г.1 – Диаграмма классов сервиса авторизации

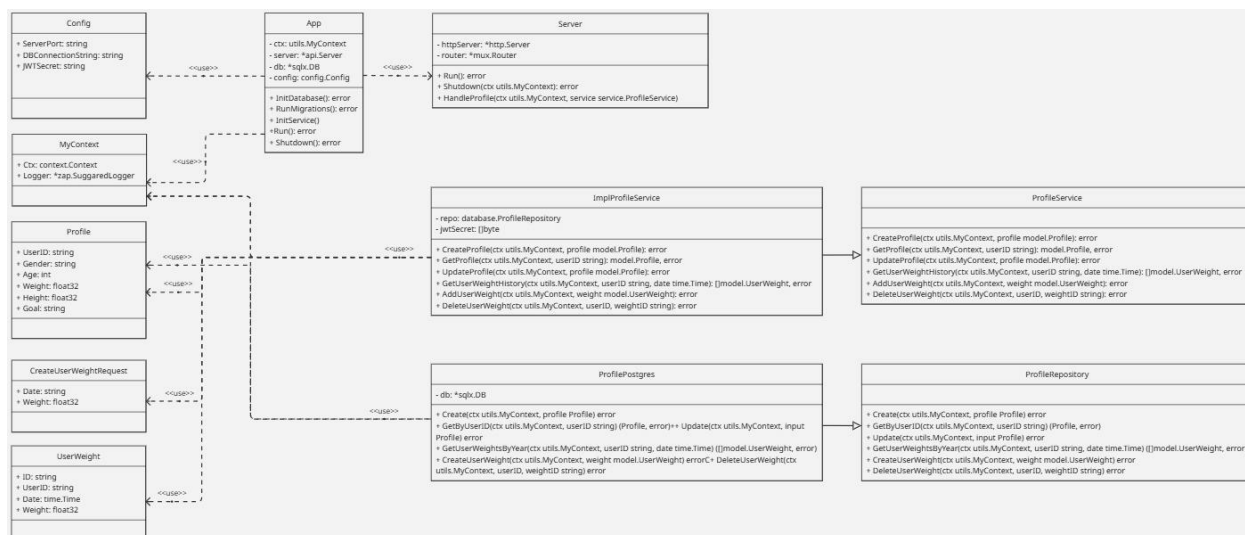


Рисунок Г.2 – Диаграмма классов сервиса профиля

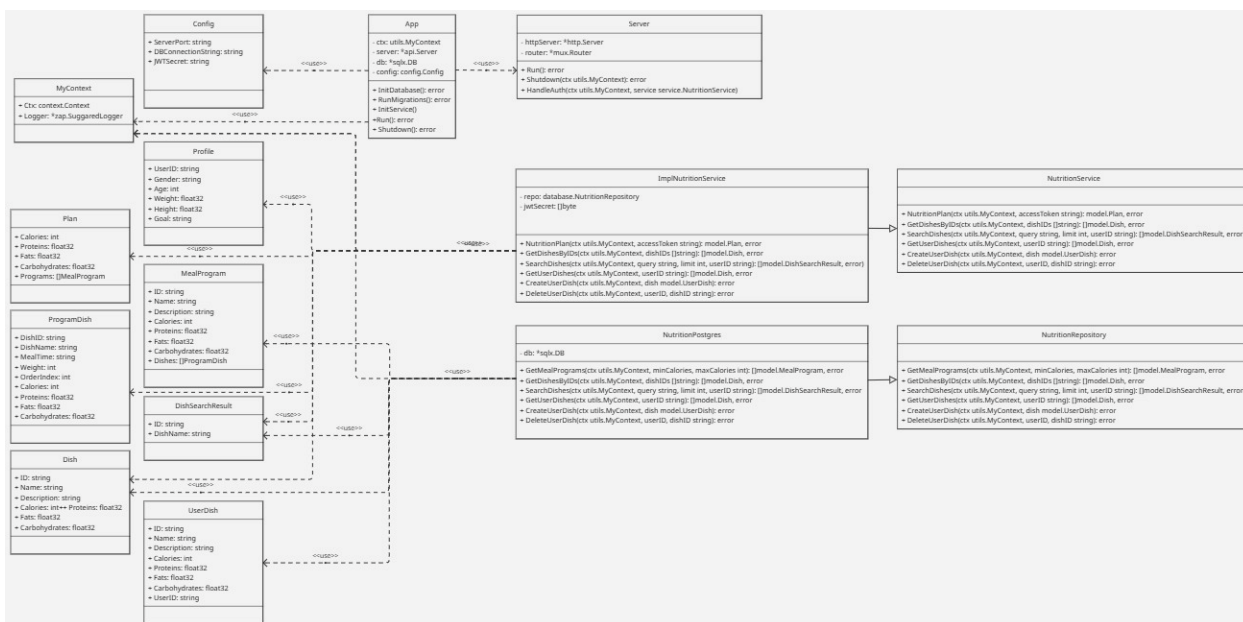


Рисунок Г.3 – Диаграмма классов сервиса питания

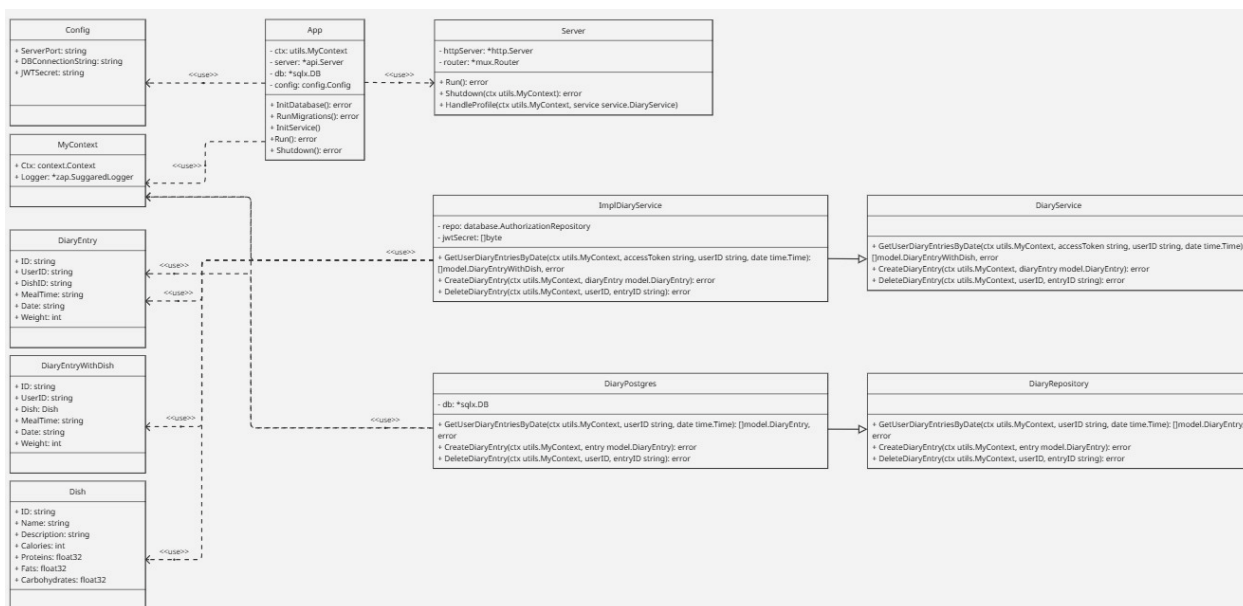


Рисунок Г.4 – Диаграмма классов сервиса дневника

Приложение Д
(рекомендуемое)
Результаты функционального тестирования

Нормальный вес

Пол:

Мужчина ▼

Возраст:

20

Вес (кг):

80

Рост (см):

185

Цель:

Поддержание веса ▼

Сохранить профиль

Рисунок Д.1 – Список тестируемых данных заполнения профиля

id	user_id	gender	age	weight	height	goal
dda9e62a-f6b7-4ade-a44a-7acd91a7e420	7664de5f-c114-45bd-b4ed-abc802bb5905	m	20	85	185	maintain

Рисунок Д.2 – Результат записи профиля в базу данных

Name	×	Headers	Payload	Preview	Response	Initiator	Timing
☐ profile/	▼	General					
☒ profile/		Request URL http://localhost:82/api/profile/ Request Method POST Status Code 200 OK Remote Address [::1]:82 Referrer Policy strict-origin-when-cross-origin					
☐ weight/		▼ Response Headers ☐ Raw					
☒ weight/		Access-Control-Allow-Headers * Access-Control-Allow-Methods PUT, GET, HEAD, POST, DELETE, OPTIONS Access-Control-Allow-Origin http://localhost:3000 Content-Length 16 Content-Type application/json					
☐ weight/?date=2025-01-01							
☒ weight/?date=2025-01-01							

Рисунок Д.3 – Ответ сервера на создание профиля

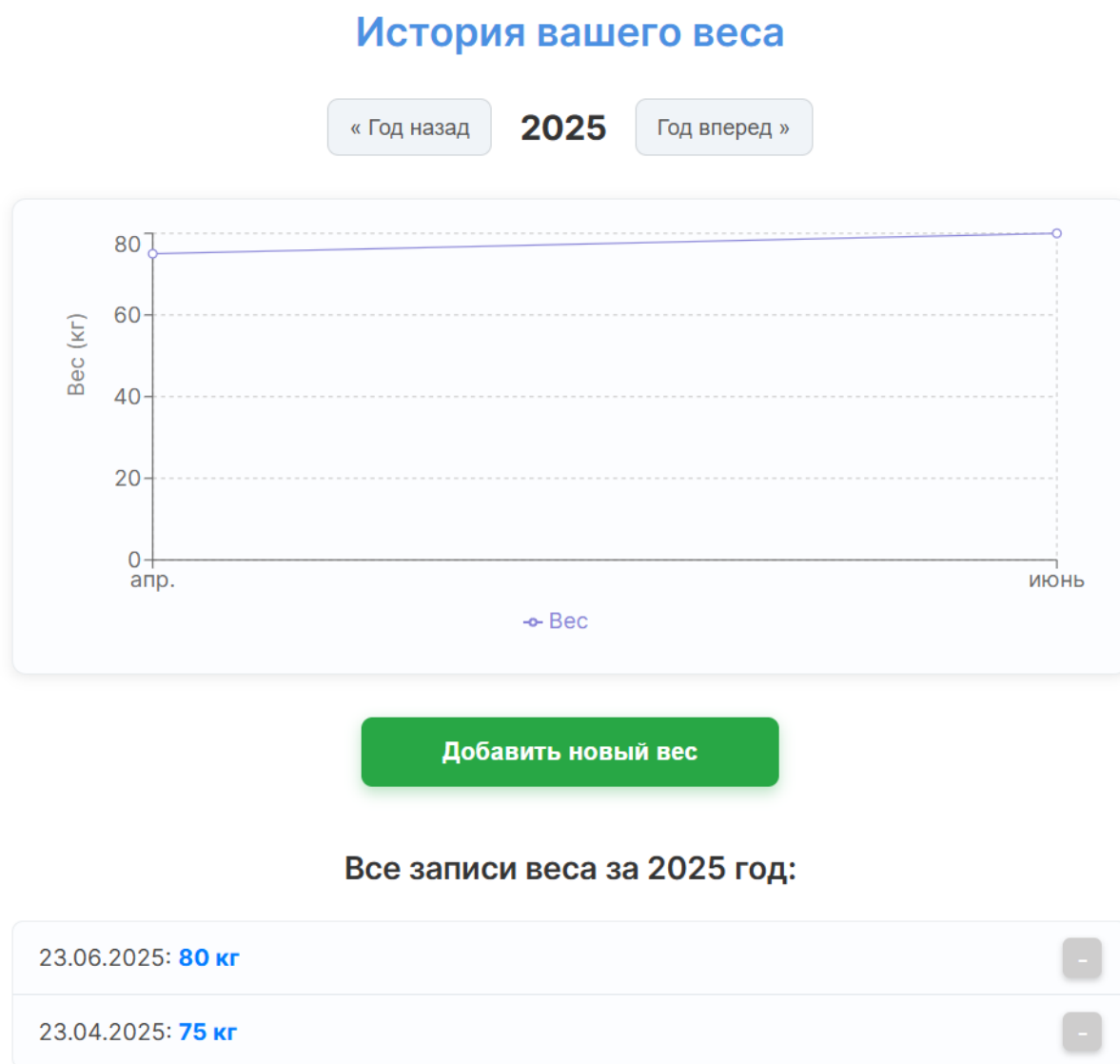


Рисунок Д.4 – Значение на запись в историю весов

Request URL	http://localhost:82/api/profile/weight/
Request Method	POST
Status Code	● 200 OK
Remote Address	[::1]:82
Referrer Policy	strict-origin-when-cross-origin
▼ Response Headers	☐ Raw
Access-Control-Allow-Headers	*
Access-Control-Allow-Methods	PUT, GET, HEAD, POST, DELETE, OPTIONS
Access-Control-Allow-Origin	http://localhost:3000
Content-Length	16
Content-Type	application/json

Рисунок Д.5 – Ответ сервера на создание записи веса

```
profiles=# select * from user_weights where user_id = '17004964-9e7c-4bc9-b561-be7f1f71ca82';
```

id	user_id	date	weight
cd9d418a-cf06-4b02-8b0b-93ca329eb3db	17004964-9e7c-4bc9-b561-be7f1f71ca82	2025-06-23 00:00:00	80
fcfc7482-1471-4b38-acfc-6dcffe73e599	17004964-9e7c-4bc9-b561-be7f1f71ca82	2025-04-23 00:00:00	75

(2 rows)

Рисунок Д.6 – Результат записи значения веса в базу данных

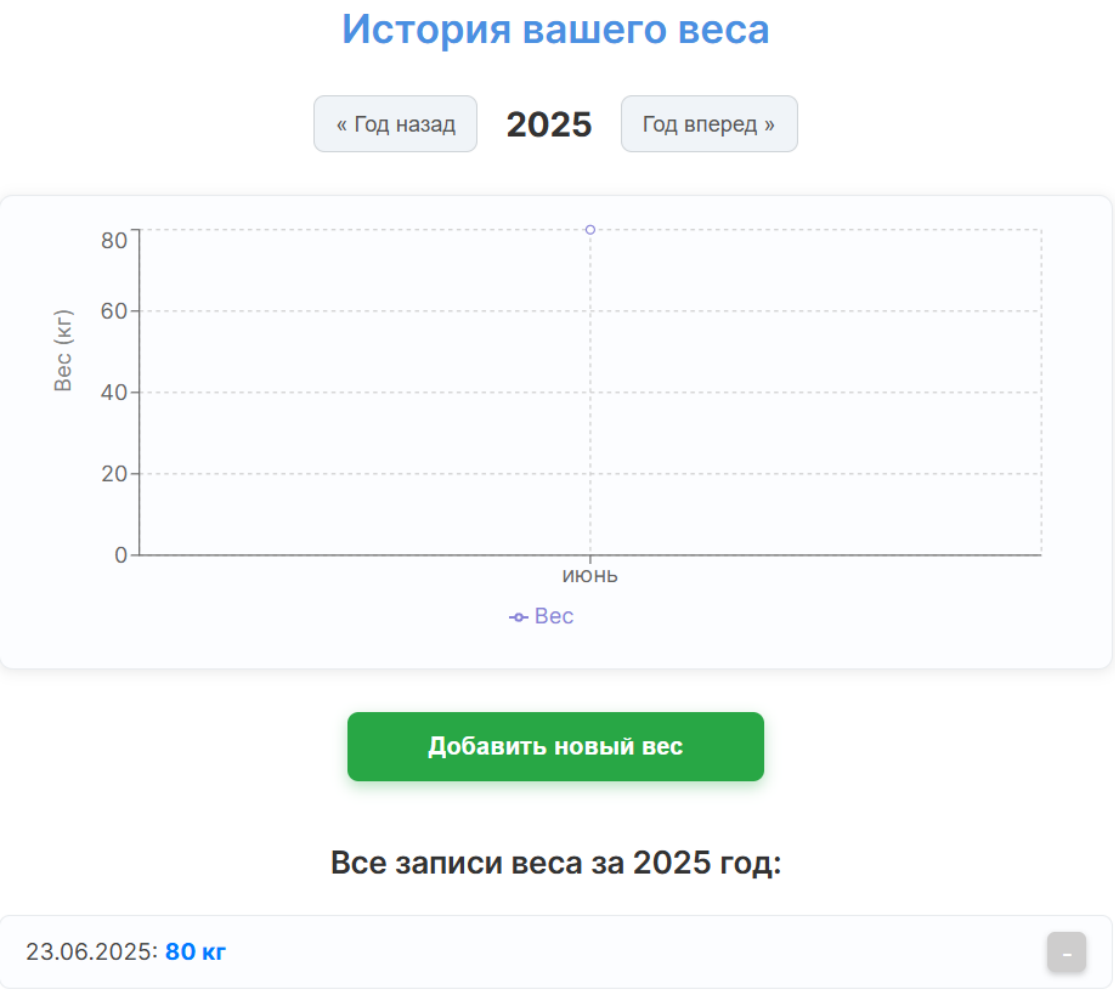


Рисунок Д.7 – Удаление значения из истории весов

Request URL	http://localhost:82/api/profile/weight?id=fcfc7482-1471-4b38-acfc-6dcffe73e599
Request Method	DELETE
Status Code	200 OK
Remote Address	::1:82
Referrer Policy	strict-origin-when-cross-origin
▼ Response Headers	☐ Raw
Access-Control-Allow-Headers	*
Access-Control-Allow-Methods	PUT, GET, HEAD, POST, DELETE, OPTIONS
Access-Control-Allow-Origin	http://localhost:3000
Content-Length	16
Content-Type	application/json

Рисунок Д.8 – Ответ сервера на удаление значение веса

```
profiles=# select * from user_weights where user_id = '17004964-9e7c-4bc9-b561-be7f1f71ca82';
      id      |      user_id      |      date      | weight
-----+-----+-----+-----
cd9d418a-cf06-4b02-8b0b-93ca329eb3db | 17004964-9e7c-4bc9-b561-be7f1f71ca82 | 2025-06-23 00:00:00 |      80
(1 row)
```

Рисунок Д.9 – Результат удаления значения веса из базы данных

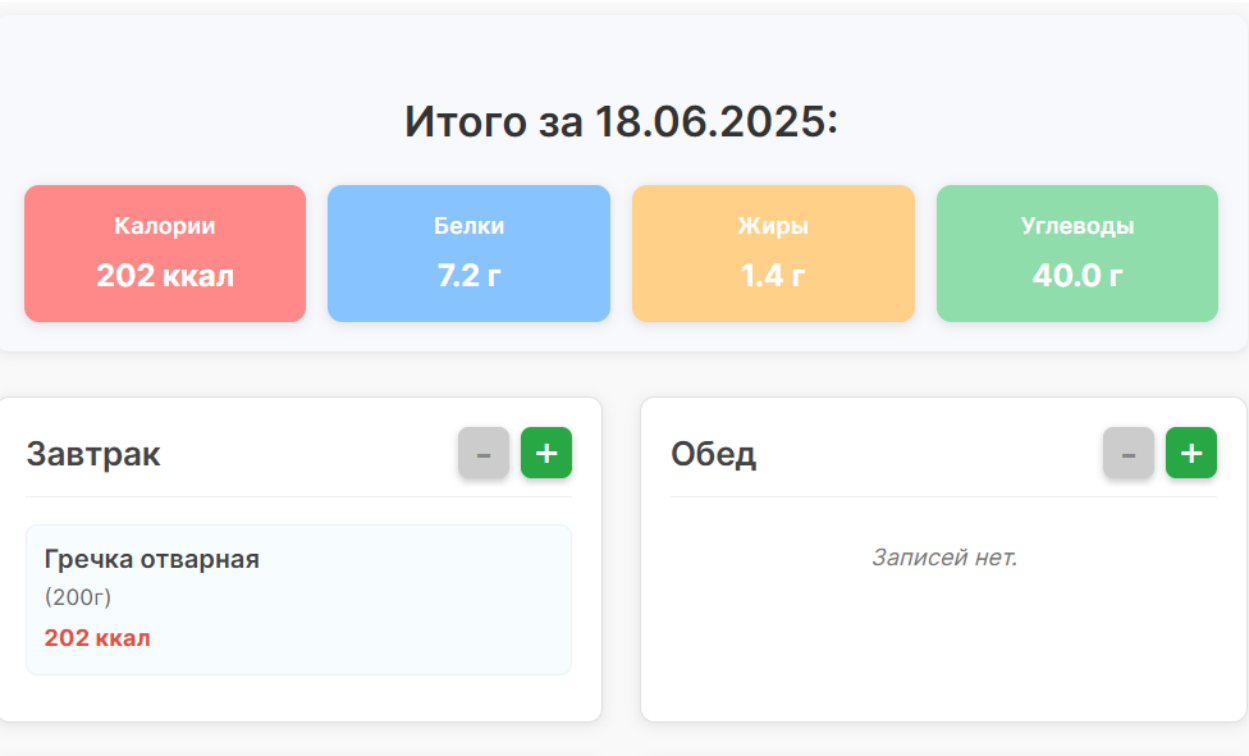


Рисунок Д.10 – Добавление записи в дневник питания

Request URL	http://localhost:84/api/diary/entries/
Request Method	POST
Status Code	200 OK
Remote Address	:::1]:84
Referrer Policy	strict-origin-when-cross-origin
▼ Response Headers	☐ Raw
Access-Control-Allow-Headers	*
Access-Control-Allow-Methods	PUT, GET, HEAD, POST, DELETE, OPTIONS
Access-Control-Allow-Origin	http://localhost:3000
Content-Length	16
Content-Type	application/json

Рисунок Д.11 – Ответ сервера на добавление записи в дневник питания

```
diaries=# select * from diary_entries where user_id = '17004964-9e7c-4bc9-b561-be7f1f71ca82';
 id | user_id | dish_id | meal_time | date | weight
-----+-----+-----+-----+-----+-----
 29b10e3d-278a-4aea-b94d-b740cc3a1351 | 17004964-9e7c-4bc9-b561-be7f1f71ca82 | dd09157f-fd90-497e-b486-13c5b72b9c3 | Завтрак | 2025-06-18 00:00:00 | 200
(1 row)
```

Рисунок Д.12 – Результат добавления записи дневника питания в базу данных

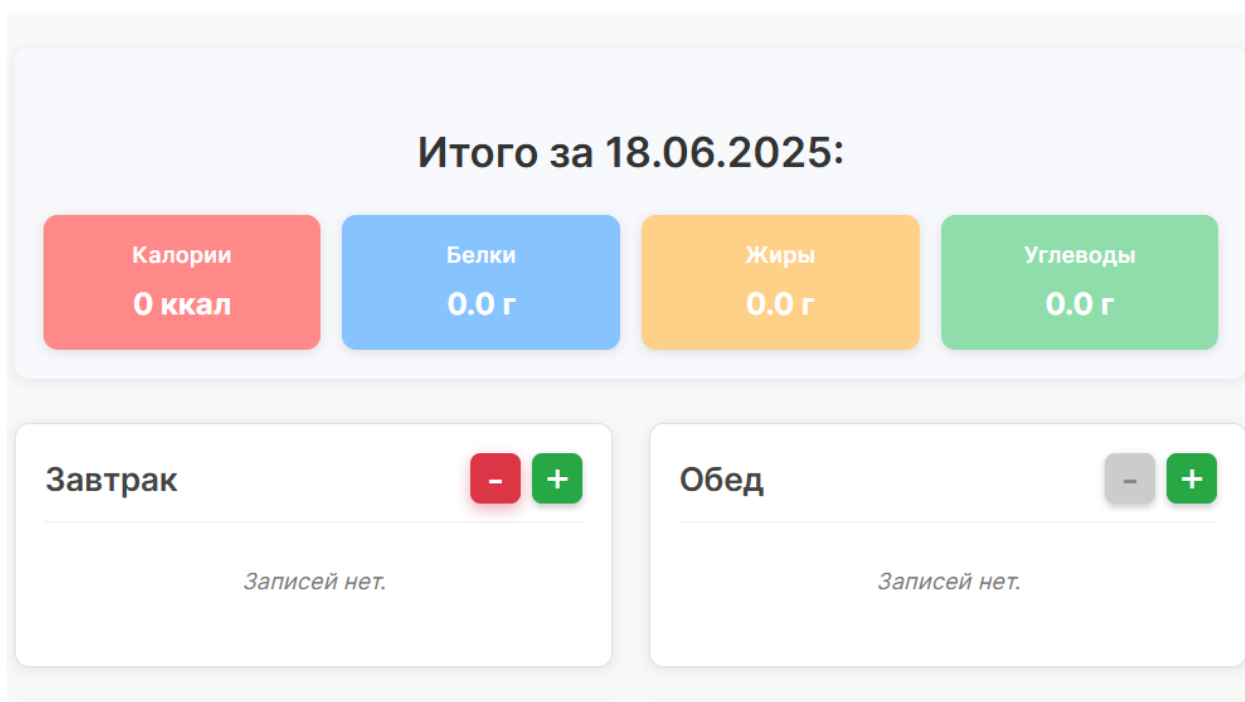


Рисунок Д.13 – Удаление записи из дневника питания

Request URL	http://localhost:84/api/diary/entries/?id=29b10e3d-278a-4aea-b94d-b740cc3a1351
Request Method	DELETE
Status Code	200 OK
Remote Address	[::1]:84
Referrer Policy	strict-origin-when-cross-origin
▼ Response Headers	☐ Raw
Access-Control-Allow-Headers	*
Access-Control-Allow-Methods	PUT, GET, HEAD, POST, DELETE, OPTIONS
Access-Control-Allow-Origin	http://localhost:3000
Content-Length	16
Content-Type	application/json

Рисунок Д.14 – Ответ сервера на удаление записи из дневника питания

```
diaries=# select * from diary_entries where user_id = '17004964-9e7c-4bc9-b561-be7f1f71ca82';
 id | user_id | dish_id | meal_time | date | weight
-----+-----+-----+-----+-----+-----
(0 rows)
```

Рисунок Д.15 – Результат удаления записи дневника питания из базы данных