МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

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

Факультет	*Кафедра*
вычислительной техники	МОиПЭВМ

Направление подготовки: 09.03.02 «Информационные системы и технологии»

БАКАЛАВРСКАЯ РАБОТА

на тему

«Аналитическая информационная система подбора правильного питания»

Студент _____ Морозов Данила Максимович
(подпись, дата) (ФИО полностью)

Руководитель _____ Иванчуков А.Г.
(подпись, дата) (фамилия, инициалы)

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

Работа допущена к защите (протокол заседания кафедры от_____№____)

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

*Работа защищена с отметкой*_____(протокол заседания ГЭК от_____№____)

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

Пенза, 2023

Федеральное государственное бюджетное образовательное учреждение высшего образования

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
Вычислительной техники	МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____А.Ю. Козлов
«__»_____2023 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент __Морозов Данила Максимович_____гр._19ВИ1_факультета___ВТ___
<Фамилия, имя, отчество полностью>

2.Тема работы__Аналитическая информационная система подбора правильного питания__
_____
_____
_____
_____

Тема утверждена приказом ПГУ №_____от "_____"_____20_____

2. Руководитель работы доцент кафедры «МОиПЭВМ»_____
Иванчуков А.Г._____

3. Задание на работу (назначение разработки, исходные данные и т.п.)_____
*1) Разработать сервис по подбору правильного питания*_____
*2) Среда выполнения клиентской части: ОС Windows 10*_____
*Среда выполнения серверной части: ОС Windows 10*_____
*Языковые средства разработки: JavaScript, Golang, UML*_____
*Инструментальные средства разработки: Webstorm, Goland, DataGrip, Erwin Data Modeler, Visual Paradigm, Argo UML, Draw.io, Project Libre*_____
*Технологии разработки: MVC, ООП, SPA*_____
*3) Основные функции web-приложения:*_____
*а) Формирование персонального плана питания*_____
*б) Заполнение подробной анкеты пользовательских данных и предпочтений*_____
*в) Возможность редактирования плана питания*_____
*г) Просмотр рецептов блюд в интерфейсе приложения*_____
_____

4. Перечень подлежащих разработке вопросов

1) Анализ предметной области и постановка задачи

2) Проектирование

3) Реализация

4) Контроль качества программного обеспечения

5) Оформление пояснительной записки

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	12.04.2023 – 16.04.2023	
2	Проектирование программных средств	20%	17.04.2023 – 26.04.2023	
3	Реализация программных средств	30%	27.04.2023 – 11.05.2023	
4	Тестирование программных средств	20%	12.05.2023 – 21.05.2023	
5	Оформление пояснительной записки	20%	22.05.2023 – 01.06.2023	

Дата выдачи задания «22» декабря 2022г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова

Реферат

Пояснительная записка содержит 97 листов, 52 рисунка, 19 использованных источников, 24 таблицы и 5 приложений.

WEB-ПРИЛОЖЕНИЕ, АЛГОРИТМ, JAVASCRIPT, REACTJS, GOLANG, POSTGRESQL, ПРАВИЛЬНОЕ ПИТАНИЕ, АНАЛИТИКА, КЛИЕНТ-СЕРВЕР.

Объектом разработки является web-приложение подбора правильного питания с удобным и понятным интерфейсом.

Предмет разработки – создание сервиса по формированию ежедневного плана правильного питания в зависимости от характеристик пользователя.

Цель работы: реализовать web-приложение подбора правильного питания путём проведения анализа пользовательских данных.

Технологии разработки: JavaScript, ReactJs, Golang.

Результат работы: спроектировано и разработано аналитическое web-приложение подбора индивидуального плана правильного питания.

СОДЕРЖАНИЕ

Введение 7
1 Анализ требований и постановка задачи на разработку 9
1.1 Анализ предметной области 9
1.2 Анализ существующих решений 10
1.2.1 Приложение «Smart-eat» 10
1.2.2 Приложение «Neodiet» 12
1.2.3 Приложение «BeFit» 13
1.3 Сравнение рассмотренных решений 14
1.4 Постановка задачи 16
1.5 Анализ функциональных требований 17
1.6 Анализ необязательных функциональных требований 23
1.7 Анализ требований к пользовательскому интерфейсу приложения 23
1.8 Анализ требований к средствам разработки приложения 24
2 Проектирование веб-приложения по подбору правильного питания 26
2.1 Проектирование архитектуры системы 26
2.2 Проектирование базы данных 27
2.3 Проектирование программных средств 32
2.4 Проектирование пользовательского интерфейса 36
3 Реализация веб-приложения по подбору правильного питания 44
3.1 Реализация архитектуры приложения «Кушайте много» 44
3.2 Реализация базы данных 45
3.3 Реализация программных средств 46
3.4 Реализация клиентского приложения 48
4 Контроль качества веб-приложения подбора правильного питания 52
4.1 Функциональное тестирование 52
4.1.1 Определение классов эквивалентности 54
4.1.2 Анализ граничных условий 55
4.1.3 Построение функциональной диаграммы 57
4.2 Метрики программного кода проекта 60
4.3 Unit-тестирование приложения 63

5 Планирование и оценка бюджета проекта 66
Заключение 71
Список использованных источников 73
Приложение А. Глоссарий 75
Приложение Б. Исходный код клиентской части 77
Приложение В. Исходный код серверной части 83
Приложение Г. Тестирование приложения 91

Введение

Тенденции в подборе правильного питания и развитие подобных веб-приложений связаны с растущим интересом к здоровому образу жизни и питанию. Среди основных тенденций можно выделить следующие:

Персонализация рациона питания – с ростом количества доступной информации о питании и здоровом образе жизни, пользователи становятся более осведомленными и требовательными в отношении «здорового потребления». Приложения для подбора питания должны учитывать индивидуальные потребности и предпочтения пользователей, чтобы предложить наиболее подходящий рацион.

Анализ данных и использование искусственного интеллекта – с помощью анализа данных и использования искусственного интеллекта приложения для подбора питания могут предоставлять более точные и персонализированные рекомендации. Например, системы могут использовать данные о потреблении пищи и физической активности, чтобы адаптировать рацион питания и предложить более эффективные рекомендации.

Интеграция с другими приложениями и устройствами – приложения для подбора питания могут интегрироваться с другими приложениями и устройствами, такими как фитнес-трекеры и умные весы, чтобы предоставлять более полную информацию о здоровье и питании пользователей.

Удобство использования – с ростом количества приложений для подбора питания, пользователи становятся более требовательными в отношении удобства использования. Приложения должны быть просты в использовании и иметь интуитивно понятный интерфейс.

Учет различных диетических ограничений – приложения для подбора питания должны учитывать различные диетические ограничения, такие как аллергии на определенные продукты, вегетарианство, веганство и другие.

Целью данной выпускной квалификационной работы является разработка веб-приложения по подбору правильного питания.

Актуальность разработки приложения обусловлена высокой востребованностью в сфере «здорового потребления» инструмента, способного обеспечить удобное ведение плана питания. Большая часть приложений не предлагает инструментарий для того, чтобы гибко настроить ассортимент блюд и количество приёмов пищи за один день.

1 Анализ требований и постановка задачи на разработку

1.1 Анализ предметной области

Человек, задавшийся целью изменить свой вес, скорректировав питание, выходит в Интернет, будучи слабо осведомленным в сфере правильного потребления. Нормы приема пищи, подсчет калорий, нутриентность продуктов – всё это редко преподается в непрофильных учебных заведениях.

Большая часть запросов в Интернете на темы «Правильное питание», «Составление правильного питания», «Подбор правильного питания», «План питания» выдают в поиске статьи, информационные ресурсы, научные публикации, в то время как прямых инструментов взаимодействия в действительности немного.

Большинство таких приложений перегружены лишним функционалом и чаще всего сопряжены с сервисами – каталогами кухонь и рецептов со всего мира. Они предлагают огромное разнообразие функционала, например, просмотр рецептов, прочтение советов по приготовлению, интересных фактов о блюдах, вплоть до истории возникновения той или иной кухни мира. Отсутствие нормальной подачи информации по данной тематике, как правило, сопровождается негативным пользовательским опытом (рисунок 1).

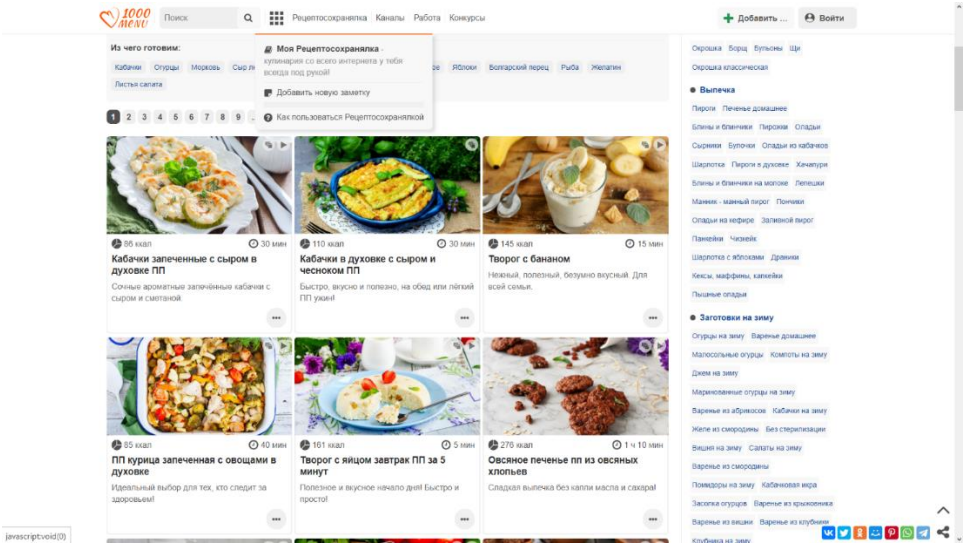


Рисунок 1 – Интерфейс сайта с рецептами

Оставшаяся меньшая часть приложений, целенаправленно предлагающих функционал формирования плана правильного питания, не страдают проблемой перегрузки пользовательского интерфейса, но сталкиваются с рядом других проблем: жесткой привязанности к указанным при регистрации продуктам питания и вытекающей из этого ограниченности развития гастрономического кругозора пользователя, решившего следовать предложенному плану питания.

Поэтому разработка приложения, которое позволит создать план правильного питания, включающий в себя диету на 7 дней недели, рецепты предлагаемых блюд, корректируемое трёх или четырёхразовое питание, может значительно упростить процесс воплощения целей по корректировке веса большого количества пользователей.

1.2 Анализ существующих решений

В рамках выполнения выпускной работы было проведено исследование наиболее популярных приложений подбора правильного питания, таких как: «Smart-eat», «Neodiet», «BeFit».

1.2.1 Приложение «Smart-eat»

Данное приложение предоставляет возможность заполнить подробную анкету пользователя, получить на основе заполненных данных планы на 7 или 28 дней, получить список продуктов к покупке.

Интерфейс приложения един для веб и мобильной версии, однако заметно ориентирование на пользователей смартфонов.

Основным минусом этого приложения можно назвать отсутствие минимальных рецептов к блюдам, доступных для прочтения в интерфейсе сервиса – их необходимо скачивать.

Интерфейс приложения представлен на рисунках 2 и 3.

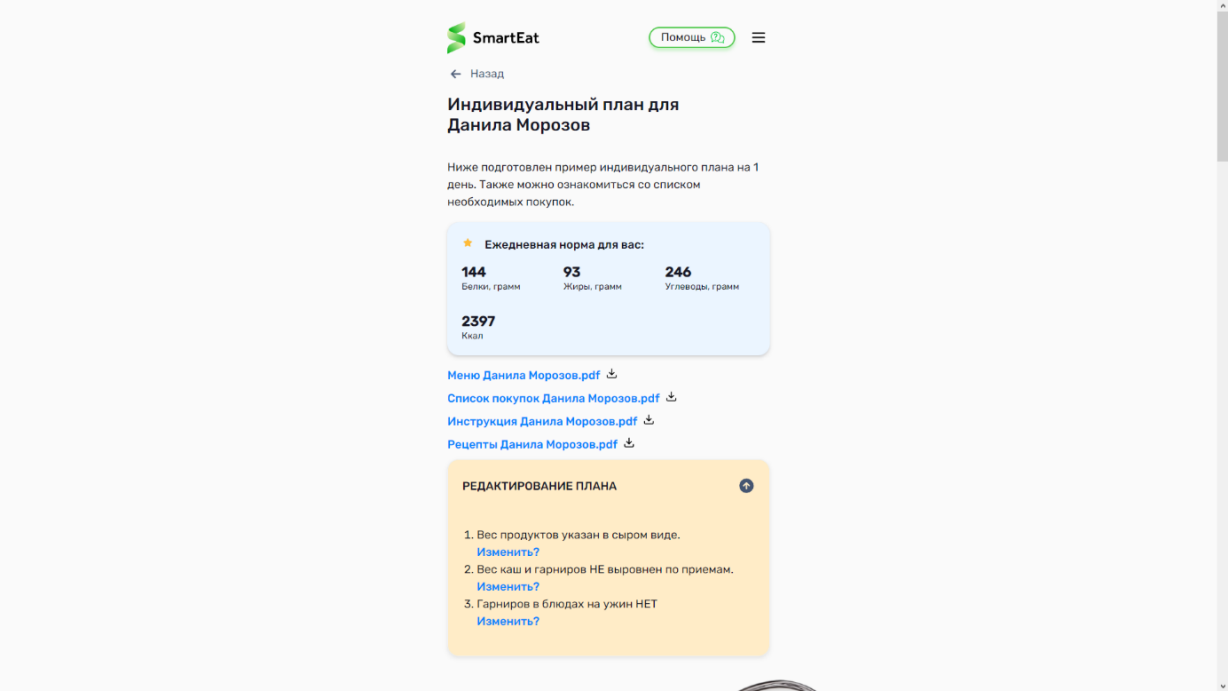


Рисунок 2 – Интерфейс приложения «Smart-eat»

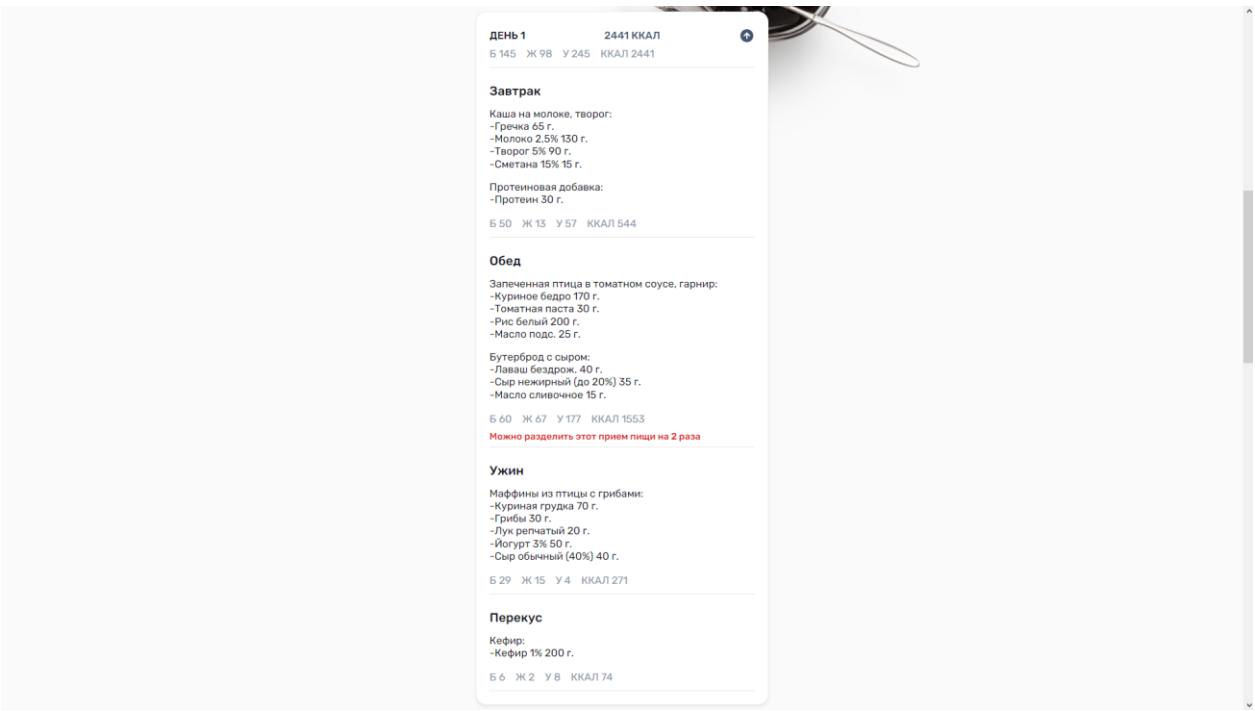


Рисунок 3 – Интерфейс плана питания в приложении «Smart-eat»

Можно сделать вывод, что при использовании этого приложения в браузере ПК устройствах, пользовательский опыт становится менее качественным.

1.2.2 Приложение «Neodiet»

Сервис «Neodiet» представлен как в виде веб-приложения, так и мобильного приложения. Данное приложение предоставляет возможность заполнить простую анкету, сформировать план трёхразового питания. Сервис требует оплаты, возможность создания пробного плана отсутствует. Однако, создатели сервиса описали основной функционал отдельно, в специальном руководстве (рисунок 4).

Рисунок 4 – Функционал сервиса «Neodiet»

Интерфейс приложения представлен на рисунке 5.

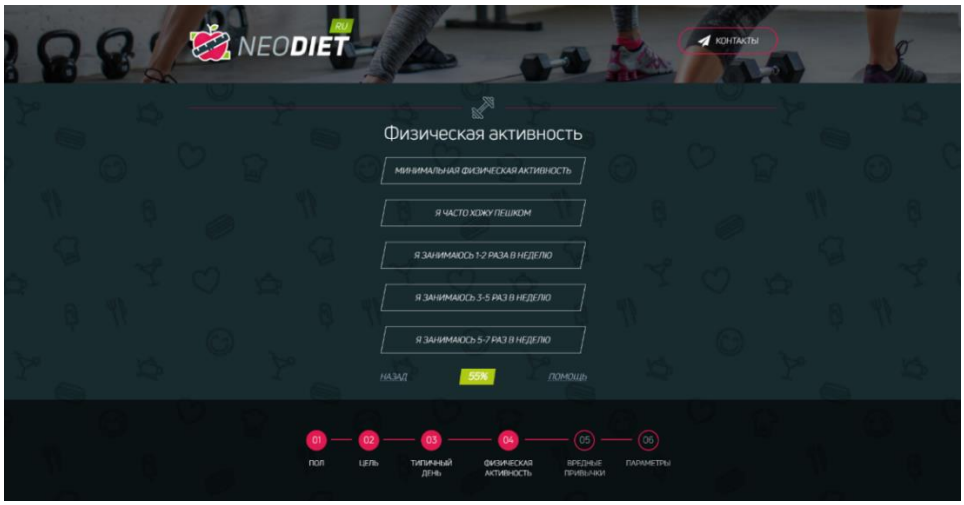


Рисунок 5 – Интерфейс приложения «Neodiet»

Основными недостатками являются отсутствие пробного периода и ограниченность функционала приложения.

1.2.3 Приложение «BeFit»

Сервис «BeFit» представлен в виде веб-приложения и мобильного приложения. Данное приложение предоставляет возможность выбрать готовый план питания, заменить отдельные блюда и ингредиенты, а также заказать выбранные блюда с доставкой на дом.

Идея данного сервиса несколько отличается от предыдущих – фокусируется на доставке готовых блюд из сформированного заранее плана. Основными недостатками являются отсутствие рецептов блюд, что лишает возможности пользователя приготовить их самостоятельно.

Интерфейс приложения представлен на рисунках 6 и 7.

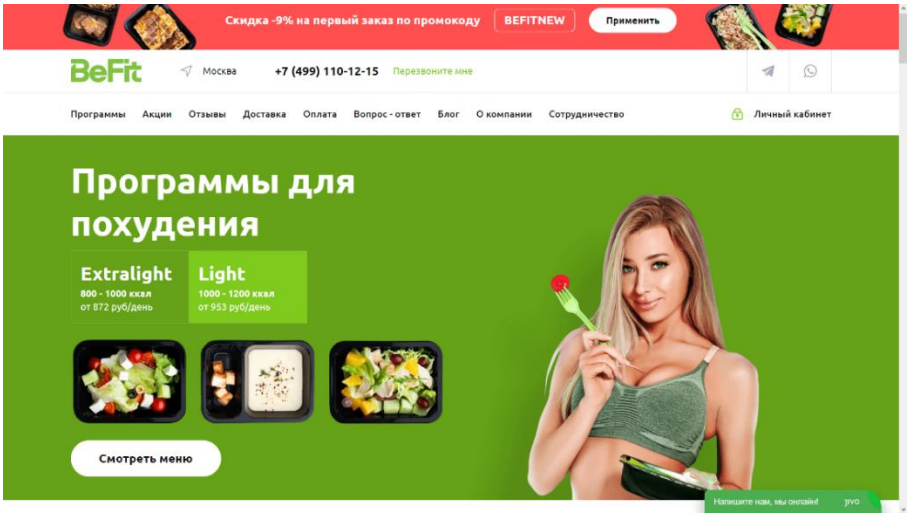


Рисунок 6 – Интерфейс приложения «BeFit»

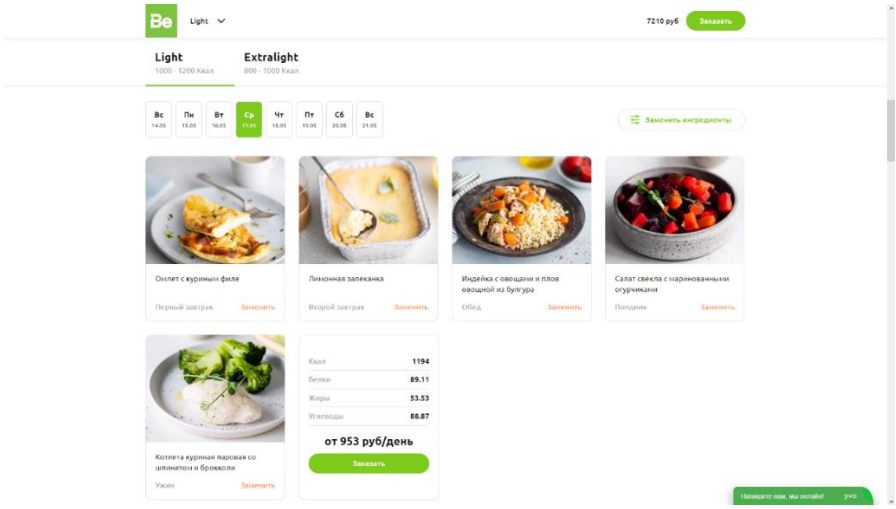


Рисунок 7 – План питания в сервисе «BeFit»

Можно сделать вывод, что сервис удобен в использовании с ПК и мобильных устройств, но не имеет минимального функционала предоставления рецептов блюд.

1.3 Сравнение рассмотренных решений

Рассмотренные сервисы обладают различным функционалом и характеристиками. В результате проведенного анализа были выделены критерии, напрямую влияющие на удобство и эффективность применения приложения:

– заполнение персональной анкеты;

– формирование плана питания с учетом характеристик и предпочтений пользователя;

– редактирование плана;

– наличие рецептов блюд;

– наличие пробного периода;

– эргономичный дизайн интерфейса.

Полученные результаты сравнения приведены в таблице 1.

Таблица 1 – Сравнение по критериям

Функциональные характеристики	«Smart-eat»	«Neodiet»	«BeFit»
Заполнение персональной анкеты	Есть, наиболее подробная анкета из всех	Есть	Нет
Формирование плана питания с учетом характеристик пользователя	Есть	Есть	Нет

Продолжение таблицы 1

Функциональные характеристики	«Smart-eat»	«Neodiet»	«BeFit»
Редактирование плана	Есть, но редактированию подвергаются лишь некоторые малозначительные параметры	Нет	Есть, редактирование очень точное, вплоть до ингредиентов блюд
Наличие рецептов блюд	Есть, в отдельном скачиваемом документе	Есть	Нет, присутствует возможность лишь заказать готовые блюда, но не приготовить самостоятельно
Наличие пробного периода	Есть, даётся бесплатный план на один день	Нет	Есть, можно просмотреть все возможные планы и все возможные блюда
Эргономичный интерфейс	Частично, интерфейс приспособлен под веб и мобильные устройства, однако в веб версии много пространства не используется	Есть	Есть

Из результатов сравнения можно сделать вывод, что каждое приложение обладает определенным функционалом, но не решает всех проблем, с которыми может столкнуться пользователь. Также, у большей части рассмотренных приложений отсутствует возможность просмотреть рецепт блюда в самом сервисе, что негативно сказывается на пользовательском опыте. Отдельно стоит отметить, что метод сбора необходимых характеристик и предпочтений пользователя у большинства сервисов не соответствует современным стандартам удобства, представлен в виде формы или вовсе отсутствует.

Таким образом, при разработке веб-приложения по подбору правильного питания будут учтены данные недостатки и реализован весь необходимый функционал.

1.4 Постановка задачи

Данная выпускная квалификационная работа посвящена разработке веб-приложения по подбору правильного питания. Приложение создается для того, чтобы упростить для пользователя путь к его идеальному телу. Приложение должно соответствовать уровню существующих аналогов или превосходить его по следующим критериям:

– заполнение удобной персональной анкеты;

– формирование плана питания с учетом характеристик и предпочтений пользователя;

– редактирование количества блюд и состава плана;

– наличие рецептов блюд и доступа к ним в интерфейсе приложения;

– наличие пробного периода, знакомящего пользователя с полным функционалом;

– эргономичный и интуитивно понятный дизайн интерфейса, ориентированный на веб-браузеры.

Сервис подбора правильного питания должен иметь следующие страницы:

– страница авторизации и регистрации;

– страница плана питания;

– страница профиля пользователя.

Также в системе должны быть реализованы следующие характеристики и функции:

– редактирование анкеты пользовательских данных;
– фиксирование прогресса прохождения плана питания;
– возможность переформировать или отредактировать план питания.

В ходе анализа аналогов была установлена потребность в упрощении процесса регистрации и сбора данных, характеристик и предпочтений пользователя. Приложение должно обладать современным инструментом, реализующим анкету регистрации, которая соответствует дизайну приложения, вовлекает и информирует пользователя по мере заполнения об его будущем опыте пользования сервисом.

1.5 Анализ функциональных требований

Анализ функциональных требований включает в себя изучение результата, которого требуется достичь, и понимание того, какие детали должны быть включены для достижения этого результата.

Этот этап разработки приложения помогает проектировать программное обеспечение более эффективно, а также способствует созданию приложения с более полным функционалом.

При разработке программного обеспечения функциональные требования определяют функции, которые должно выполнять всё приложение или только один из его компонентов.

В соответствии с проведенным анализом функциональных требований была построена диаграмма вариантов использования в нотации Unified Modeling Language (UML) [1].

Диаграмма вариантов использования представлена на рисунке 8.

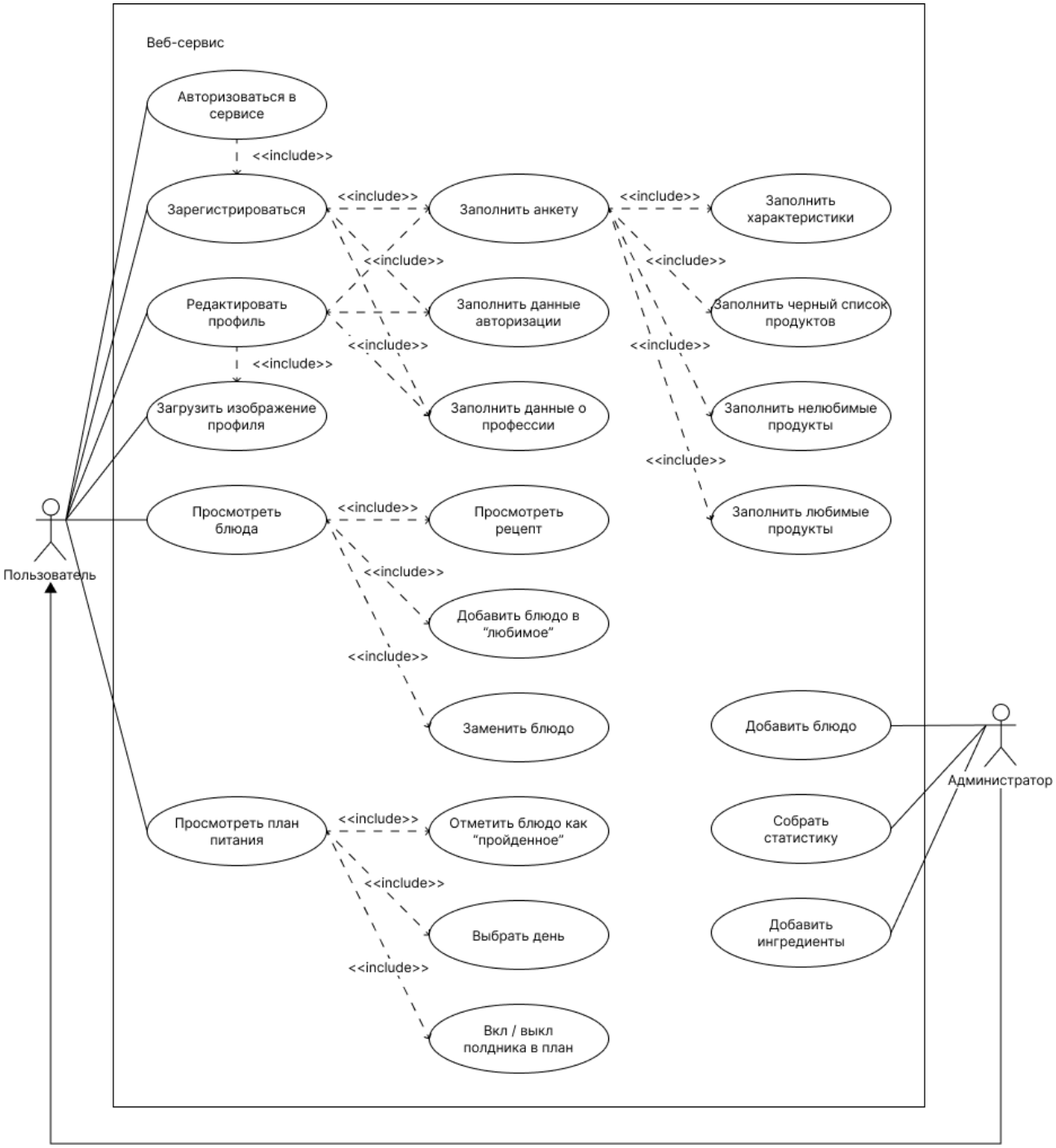


Рисунок 8 – Диаграмма вариантов использования

Актером в данной диаграмме является пользователь приложения – лицо, которому будет предоставлен план питания. Для него были выделены следующие функции:

– «Авторизоваться в сервисе» – пользователь без аккаунта может заполнить подробную анкету и после зарегистрироваться в системе, а если у него есть аккаунт, то он может войти в него;

– «Редактировать профиль» – пользователь с зарегистрированным аккаунтом может изменить изображение профиля или данные своей анкеты;

– «Просмотреть блюда» – пользователь может просмотреть список блюд в составе сформированного плана питания, а также блюд, ранее отмеченных им «любимыми»;

– «Просмотреть план питания» – основной функционал программы.

Пользователь может изучить, изменить, пройти план питания, посмотреть рецепты отдельных блюд в его составе и отметить понравившиеся, просмотреть предстоящие или пройденные дни.

Далее будут подробнее рассмотрены основные прецеденты диаграммы [2].

Описание следующего прецедента «Авторизоваться в сервисе» представлено в таблице 2.

Таблица 2 – Описание прецедента «Авторизоваться в сервисе»

Название пункта	Описание
Краткое описание	Пользователь авторизуется в приложении
Главные актеры	Пользователь
Второстепенные актеры	Нет
Предусловие	Пользователь входит в приложение
Основной поток	Пользователь авторизуется
Постусловие	Пользователь авторизован
Альтернативные потоки	1. Пользователь не авторизуется и ему не доступен стандартный функционал приложения. 2. Пользователь выбирает регистрацию, заполняет подробную анкету и авторизируется под новым профилем. 3. Пароль или логин введены неверно, и система уведомляет пользователя об ошибке.

На этапе авторизации пользователю предоставляется возможность войти в свой аккаунт или начать регистрацию нового. При выборе регистрации, пользователю необходимо заполнить подробную анкету в виде одностраничного поэтапного мастера регистрации. В процессе прохождения мастера, пользователю сообщается основная информация о работе сервиса.

Описание следующего прецедента «Редактировать профиль» представлено в таблице 3.

Таблица 3 – Описание прецедента «Редактировать профиль»

Название пункта	Описание
Краткое описание	Пользователь редактирует данные своего профиля
Главные актеры	Пользователь
Второстепенные актеры	Нет
Предусловие	Пользователь авторизован
Основной поток	Пользователь переходит на страницу своего профиля, редактирует данные своей анкеты
Постусловие	На странице отображается информация заполненной анкеты регистрации
Альтернативные потоки	Пользователь редактирует изображение профиля

Пользователь может редактировать данные заполненной ранее анкеты пользовательских данных, а также устанавливать или менять изображение профиля.

Описание следующего прецедента «Просмотреть блюда» представлено в таблице 4.

Таблица 4 – Описание прецедента «Просмотреть блюда»

Название пункта	Описание
Краткое описание	Пользователь просматривает блюда
Главные актеры	Пользователь
Второстепенные актеры	Нет
Предусловие	Пользователь авторизован
Основной поток	Пользователь просматривает информацию о блюдах в составе плана питания, их рецепты
Постусловие	Пользователь ознакомился с информацией о блюдах
Альтернативные потоки	Пользователь просматривает любимые блюда

Пользователь может просматривать блюда и их рецепты в составе плана питания, а также отмечать отдельные блюда любимыми, просматривать блюда, отмеченные ранее.

Описание следующего прецедента «Просмотреть план питания» представлено в таблице 5.

Таблица 5 – Описание прецедента «Просмотреть план питания»

Название пункта	Описание
Краткое описание	Пользователь просматривает план питания
Главные актеры	Пользователь
Второстепенные актеры	Нет
Предусловие	Пользователь авторизовался и перешёл на главную страницу
Основной поток	Пользователь изучает план сформированный план питания, выбирает дни недели, заменяет отдельные блюда в плане, добавляет и убирает полдники, а также отмечает блюда «Пройденными» и «Любимыми»
Постусловие	План питания просмотрен
Альтернативные потоки	Нет

Пользователь может изучить, изменить, пройти план питания, посмотреть рецепты отдельных блюд в его составе и отметить понравившиеся пройденные, добавить и убрать полдник, просмотреть предстоящие или пройденные дни.

В результате анализа предметной области выявлено, что система должна обеспечивать возможность:

– зарегистрироваться в системе, заполнив подробную пользовательскую анкету;
– войти в систему при наличии аккаунта;
– редактировать профиль;
– получать и изучать персонализированные планы питания;
– просматривать информацию о блюдах, их рецепты.

Диаграмма деятельности представляет собой блок-схему, показывающую деятельности и последовательность их выполнения [3]. Для уточнения взаимодействия между объектами была спроектирована диаграмма деятельности для прецедента «Просмотреть план питания» (рисунок 9).

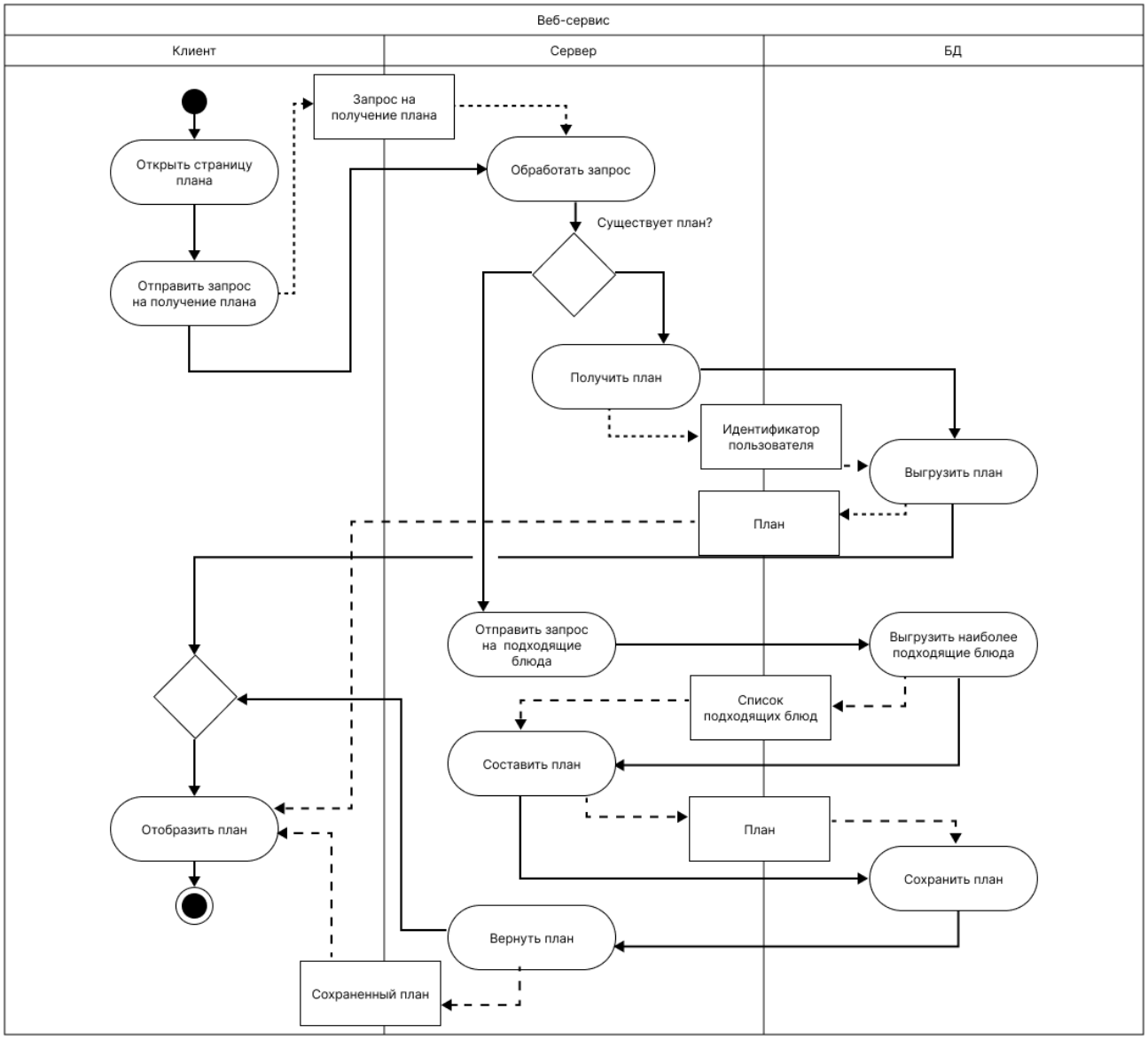


Рисунок 9 – Диаграмма деятельности для прецедента «Просмотреть план питания»

Данный вариант затрагивает основные функции системы, а также наибольшее количество объектов и связей между ними.

Пользователь открывает главную страницу, содержащую план питания. Отправляется запрос на получение плана, после чего происходит проверка на существование экземпляра. Если план существует, он запрашивается из БД на основе идентификатора пользователя, форматируется и отправляется клиенту. Если план ещё не был сформирован, отправляется запрос в БД на вычисление перечня подходящих блюд. Составляется список наиболее подходящих блюд, из

которых формируется новый план питания. Отправляется запрос на сохранение плана питания, в то же время в ответе клиенту отправляется экземпляр. Клиент отображает полученный план в интерфейсе сервиса.

1.6 Анализ необязательных функциональных требований

Также желательно, чтобы программный продукт обладал следующими дополнительными характеристиками и функциями:

- возможность сохранить любимые блюда;
- возможность просмотреть все любимые блюда;
- загрузка изображения профиля.

Необязательные функциональные требования приводят к улучшению пользовательского опыта [4].

1.7 Анализ требований к пользовательскому интерфейсу приложения

Требования к пользовательскому интерфейсу являются одними из основных для определения функционала и дизайна приложения. Позитивный пользовательский опыт по большей мере зависит от наличия отзывчивого, интуитивно понятного и лаконичного интерфейса [5]. При анализе принципов материального дизайна были выведены следующие требования к интерфейсу приложения:

- простота и понятность;
- плавность анимаций;
- высокая скорость работы;
- обратная связь на действия пользователя;
- однородность элементов, выполняющих схожие функции;
- схожий графический стиль всех элементов;
- возможность вернуться к предыдущему состоянию.

Анализ популярных сервисов в предметной области показал, что дизайнеры подобных приложений чаще всего отдают предпочтение

пользователям мобильных устройств, в следствие чего веб-интерфейс адаптируется, теряя в практичности и удобстве для пользователей. Для приложения по подбору правильного питания «Кушайте много» была выбрана стратегия, ориентированная на пользователей веб-приложений.

1.8 Анализ требований к средствам разработки приложения

На основе изучения различных средств для разработки приложения был выведен ряд требований:

– поддержка языка JavaScript;

– удобная работа с клиентской и серверной частью приложения, а также с БД;

– удобное подключение и работа с различными сторонними фреймворками.

Исходя из вышеперечисленных требований в качестве среды разработки была выделена группа решений от компании JetBrains [6], продукты которой в полном объеме отвечают заданным требованиям:

– Webstorm – один из лучших инструментов frontend-разработчика на данный момент, предлагает обширную, но полностью настраиваемую базу функций, упрощающих работу с JavaScript кодом;

– Goland – IDE для работы с языком Golang, не уступает по удобству другим продуктам компании;

– Datagrip – инструмент для работы с различными базами данных.

В качестве системы управления базой данных (СУБД) была выбрана PostgreSQL, которая является объектно-реляционной и имеет открытый исходный код. Она поддерживает большую часть стандарта SQL и предлагает множество современных функций: сложные запросы, внешние ключи, триггеры, изменяемые представления, транзакционная целостность, а также многоверсионность [7].

Также были выбраны фреймворки и библиотеки:

– React.js – библиотека для создания пользовательских интерфейсов, которая помогает определить пользовательский интерфейс один раз и при последующем изменении состояния приложения, пользовательский интерфейс перестраивается для реагирования на изменения [8];

– React-router-dom – упрощает создание маршрутизации в приложении;

– Axios – обеспечивает удобную работу с REST API [9];

– Mobx.js – фреймворк, облегчающий менеджмент состояния клиентского приложения, отвечает за хранение и преобразование данных пользовательской сессии.

Mobx связывает все компоненты приложения одной сетью данных без необходимости передачи данных сверху-вниз, решая тем самым проблему «prop-drilling» – передачи данных через несколько вложенных дочерних компонентов в попытке доставить их глубоко вложенному компоненту. Проблема с этим подходом заключается в том, что большинству компонентов, через которые передаются данные – они не нужны. Транзитные компоненты просто используются в качестве среды для транспортировки этих данных к компоненту назначения [10].

Перечисленные технологии полностью удовлетворяют потребностям, выделенным для реализации программного продукта.

2 Проектирование веб-приложения по подбору правильного питания

2.1 Проектирование архитектуры системы

Разрабатываемое приложение по подбору правильного питания «Кушайте много» имеет двухуровневую архитектуру клиент-сервер. Архитектура системы представлена на рисунке 10.

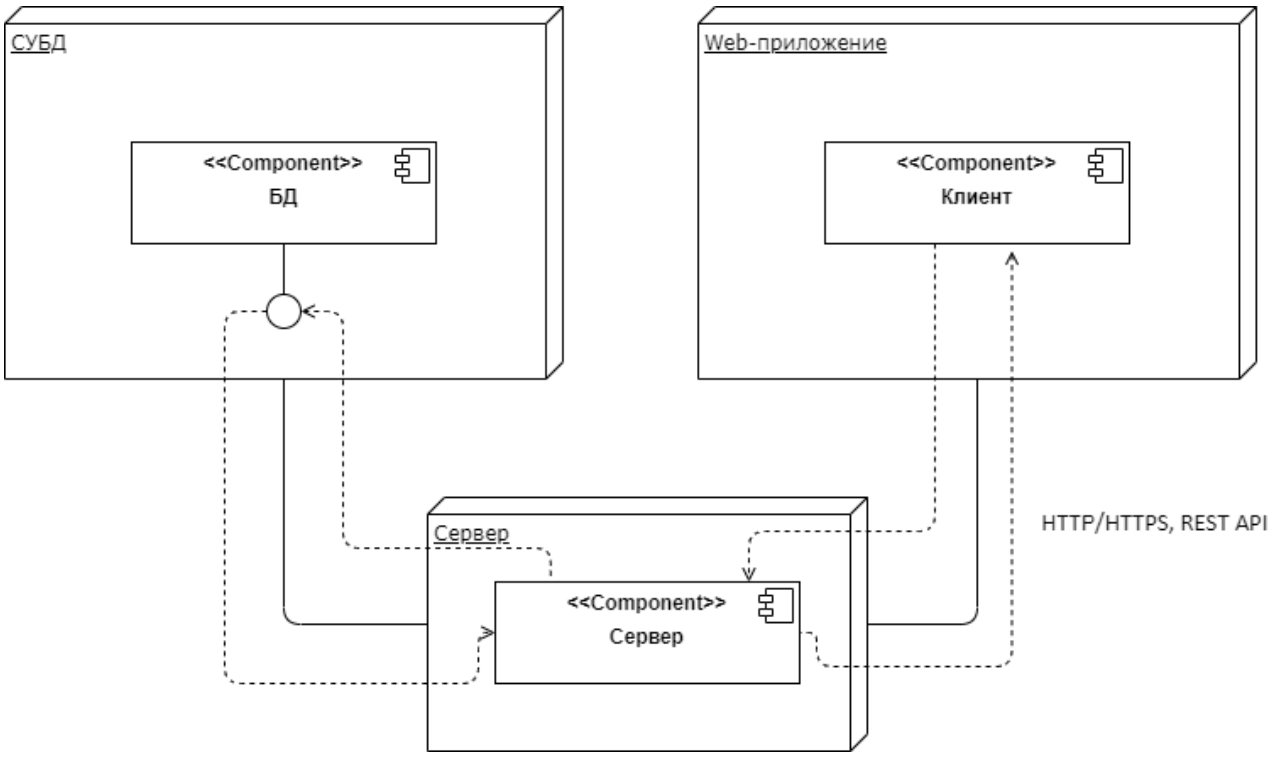


Рисунок 10 – Клиент-серверная архитектура приложения

Данный вид архитектуры включает в себя три основных компонента:

– «Клиент» – frontend-приложение, которое представляет из себя окно установленного в системе пользователя браузера, через которое он форматирует запросы пользователя и отправляет их на сервер, а также принимает ответы, преобразует и отображает их в понятной форме;

– «Сервер» – backend-приложение, обрабатывающее входные запросы от приложений-клиентов, обычно располагается на удалённой машине или распределённой группе машин;

– «БД» – база данных, в которой хранится информация о пользователе, персонально сформированные планы питания и данные блюд, содержащихся в них.

Перечисленные компоненты архитектуры составляют необходимый минимум для функционирования клиент-серверного приложения [11].

2.2 Проектирование базы данных

На основе проанализированной предметной области была построена логическая модель БД. Логическая модель не привязана к какой-либо СУБД и представляет из себя абстрактный взгляд на данные. Построенная логическая модель представлена на рисунке 11.

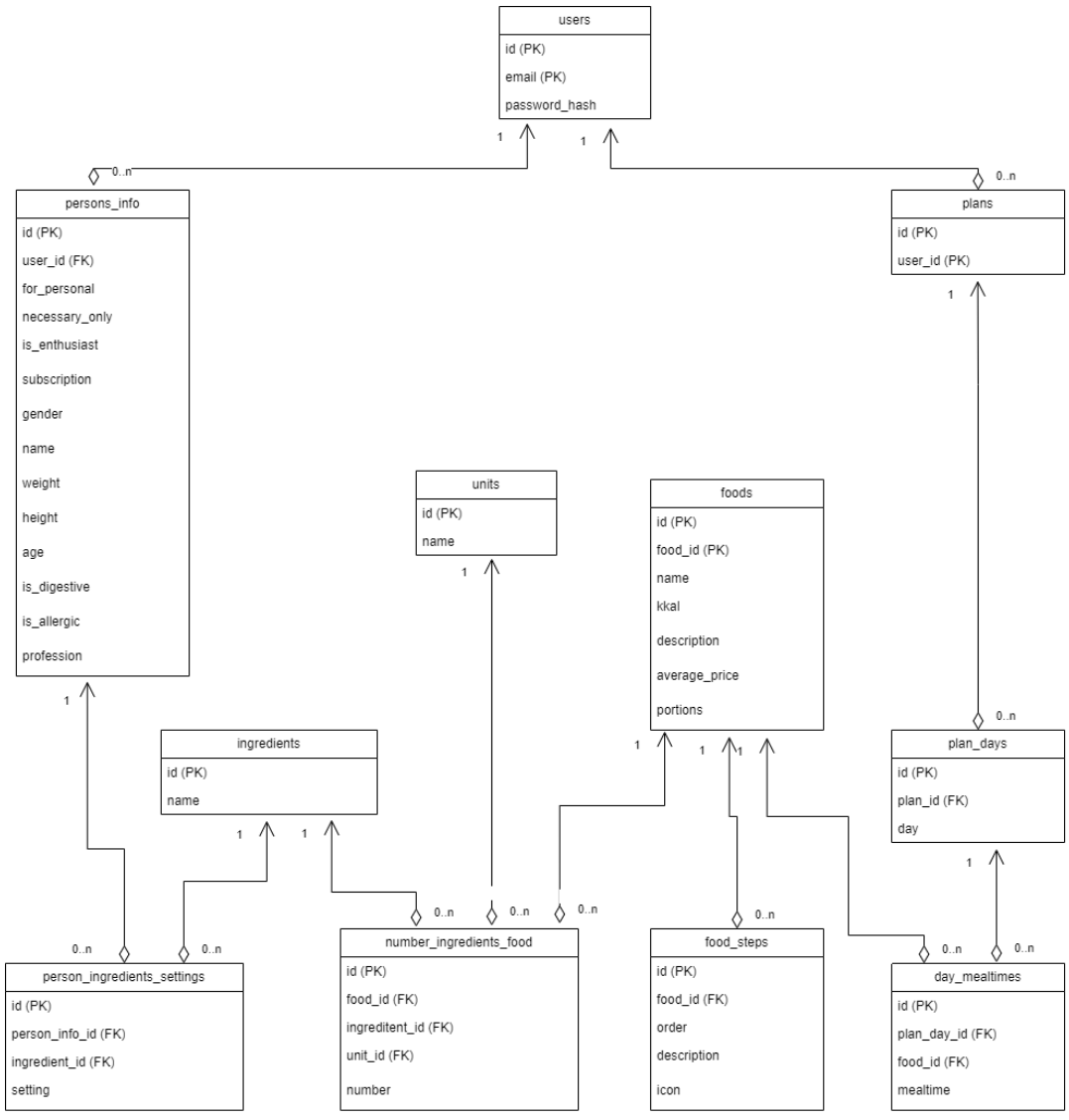


Рисунок 11 – Логическая схема БД

После того, как были выделены сущности и определены отношения между ними, необходимо нормализовать структуру БД.

Спроектированная логическая модель БД не включает в себя неявные сущности, атрибуты, зависящие только от части уникального ключа, и транзитивных зависимостей, следовательно, она соответствует третьей НФ [12].

Далее будут подробнее рассмотрены спецификации сущностей логической модели.

Спецификация сущности «persons_info», представляющей из себя пользовательские данные приведена в таблице 6.

Таблица 6 – Спецификация сущности «persons_info»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Идентификатор пользовательских данных	Целый	10	‘0-9’	Нет	PK	1736
user_id	Идентификатор пользователя	Целый	10	‘0-9’	Да	FK	3123
for_personal	Факт использования для личных целей	Логический	1	0, 1	Да		1
necessary_only	Факт сбора только необходимых данных	Логический	1	0, 1	Да		0
is_enthusiast	Факт, сообщающий, что пользователь не против расширения своего гастрономического кругозора	Логический	1	0, 1	Да		0
is_digestive	Факт наличия у пользователя пищевых расстройств	Логический	1	0, 1	Да		0
is_allergic	Факт наличия у пользователя аллергии	Логический	1	0, 1	Да		0
subscription	Факт подписи на рассылку уведомлений	Логический	1	0, 1	Да		0
gender	Пол, где 0 = мужской, 1 = женский	Логический	1	0,1	Да		0
name	Имя пользователя	Символьный	64	‘А-я’, ‘A-z’	Нет		Данила

Продолжение таблицы 6

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
weight	Вес пользователя, кг	Целый	3	`0-9`	Нет		102
height	Рост пользователя, см	Целый	3	`0-9`	Нет		192
age	Возраст пользователя	Целый	3	`0-9`	Нет		20
profession	Профессия пользователя	Символьный	64	‘А-я’, ‘A-z’	Нет		Археолог

Спецификация сущности «person_ingredients_settings», содержащей отношение пользователя к некоторым ингредиентам приведена в таблице 7.

Таблица 7 – Спецификация сущности «person_ingredients_settings»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Идентификатор пользовательских настроек ингредиентов	Целый	10	‘0-9’	Нет	PK	1736
Person_info_id	Идентификатор пользовательских данных	Целый	10	‘0-9’	Нет	FK	1736
Ingredient_id	Идентификатор ингредиента	Целый	10	‘0-9’	Нет	FK	1736
setting	Пользовательская настройка	Целый	3	0, 1, 2	Нет		2

Спецификация сущности «ingredients», содержащей ингредиенты приведена в таблице 8.

Таблица 8 – Спецификация сущности «ingredients»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Идентификатор ингредиента	Целый	10	‘0-9’	Нет	PK	1736
name	Наименование ингредиента	Символьный	64	‘А-я’, ‘A-z’	Нет		Яйцо

Спецификация сущности «foods», содержащей информацию о блюдах приведена в таблице 9.

Таблица 9 – Спецификация сущности «foods»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Приме р
id	Идентификатор блюда	Целый	10	‘0-9’	Нет	PK	1736
name	Наименование блюда	Символьный	128	‘0-9’, ‘А-я’, ‘A-z’	Да		Тост с авокадо
description	Краткое описание блюда	Символьный	512	‘0-9’, ‘А-я’, ‘A-z’	Да		Вкусный, быстрый завтрак
kkal	Количество калорий	Целый	10	‘0-9’	Нет		1921
Average_price	Средняя цена блюда	Целый	10	‘0-9’	Нет		1921
portions	Количество порций	Целый	2	‘0-9’	Нет		11

Спецификация сущности «number_ingredients_food», представляющей из себя позицию ингредиента в рецепте блюда приведена в таблице 10.

Таблица 10 – Спецификация сущности «number_ingredients_food»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Идентификатор позиции ингредиента	Целый	10	‘0-9’	Нет	PK	1736
Food_id	Идентификатор блюда	Целый	10	‘0-9’	Нет	FK	1736
Ingredient_id	Идентификатор ингредиента	Целый	10	‘0-9’	Нет	FK	1736
Unit_id	Идентификатор единицы измерения	Целый	10	‘0-9’	Нет	FK	1736
number	Количество используемых ингредиентов в е.и.	Целый	10	‘0-9’	Нет		1941

Спецификация сущности «day_mealtimes», представляющей приём пищи конкретного дня плана питания приведена в таблице 11.

Таблица 11 – Спецификация сущности «day_mealtimes»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Идентификатор приёма пищи	Целый	10	‘0-9’	Нет	PK	1736
Food_id	Идентификатор блюда	Целый	10	‘0-9’	Нет	FK	1736
Plan_day_id	Идентификатор дня плана питания	Целый	10	‘0-9’	Нет	FK	1736
mealtime	Прием пищи	Символьный	32	‘0-9’, ‘A-z’	Нет		breakfast

Спецификация сущности «units», содержащей единицы измерения ингредиентов (шт, г, стакан, ч.л.) приведена в таблице 12.

Таблица 12 – Спецификация сущности «units»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Идентификатор единицы измерения	Целый	10	‘0-9’	Нет	PK	1736
name	Наименование е.и.	Символьный	32	‘0-9’, ‘A-z’	Нет		шт

Спецификация сущности «food_steps», представляющей шаги рецепта приготовления блюда приведена в таблице 13.

Таблица 13 – Спецификация сущности «food_steps»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Идентификатор шага рецепта	Целый	10	‘0-9’	Нет	PK	1736
Food_id	Идентификатор блюда	Целый	10	‘0-9’	Нет	FK	1736
order	Порядковый номер шага	Целый	3	‘0-9’	Нет		3
description	Описание шага	Символьный	512	‘0-9’, ‘A-z’	Нет		Посолить да поперчить
icon	Идентификатор иконки шага	Целый	10	‘0-9’	Нет		23

Спецификация сущности «plans», содержащей план питания приведена в таблице 14.

Таблица 14 – Спецификация сущности «plans»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Идентификатор плана питания	Целый	10	‘0-9’	Нет	PK	1736
user_id	Идентификатор пользователя	Целый	10	‘0-9’	Нет	FK	1736

Спецификация сущности «users», содержащей пользовательские данные авторизации приведена в таблице 15.

Таблица 15 – Спецификация сущности «users»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Идентификатор пользователя	Целый	10	‘0-9’	Нет	PK	1736
email	Почтовый ящик пользователя	Строковый	256	‘0-9’ , ‘A-z’, ‘@.’	Нет		e@mail.ru
Password_hash	Зашифрованый пароль	Строковый	256	‘0-9’ , ‘A-z’, ‘_.!@&*^%^’	Нет		1736sdsf@

Описанные сущности соответствуют необходимым нормальным формам для построения оптимальной архитектуры БД, что соответствует выделенным требованиям.

2.3 Проектирование программных средств

Во время проектирования программных средств целесообразно использовать диаграмму классов, предназначенную для представления внутренней структуры программы в виде классов и связей между ними [13].

Для клиентского части приложения планирования и формирования правильного питания была спроектирована диаграмма классов, представленная на рисунке 12.

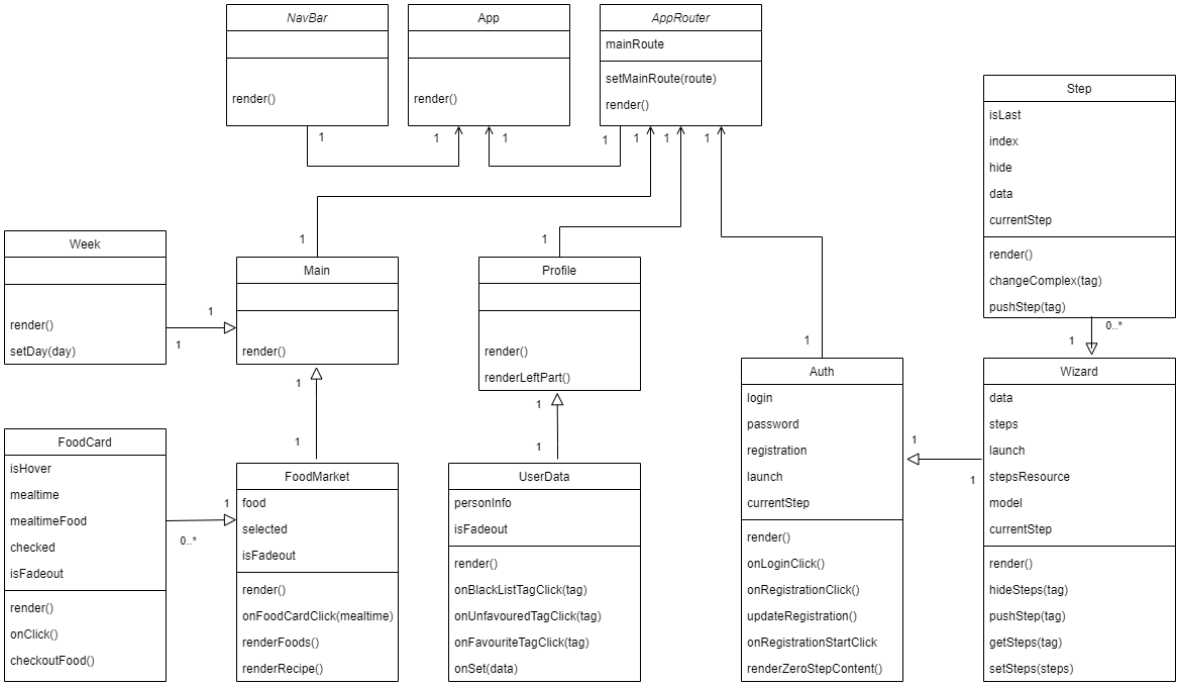


Рисунок 12 – Диаграмма классов клиентского приложения подбора правильного питания

На диаграмме представлены классы:

– App – родительский класс;
– AppRouter – класс, содержащий маршрутизацию;
– NavBar – класс, реализующий навигацию по сервису;
– Main – класс главной страницы сервиса, содержащей план питания;
– Week – класс, реализующий переключение дней плана;
– FoodMarket – класс, реализующий вывод плана и рецепта блюд;
– FoodCard – класс блюда;
– Profile – класс профиля пользователя;
– UserInfo – класс, реализующий вывод редактируемой части анкеты;
– Auth – класс страницы авторизации;
– Wizard – класс мастера, реализующий отрисовку и компоновку сложных шагов, в данном случае, регистрации;
– Step – класс шага мастера регистрации.

В рамках проектирования серверной части было выделено два основных сервиса:

– сервис питания – реализует формирование персональных планов питания, отвечает за блюда, их рецепты, а также за анализ и обработку пользовательских данных в процессе подбора позиций плана;

– сервис авторизации – обеспечивает работу системы пользовательских профилей, отвечает за регистрацию, авторизацию, редактирование пользовательских данных, поддержку сессий работы с приложением.

Диаграмма классов для сервиса питания представлена на рисунке 13.

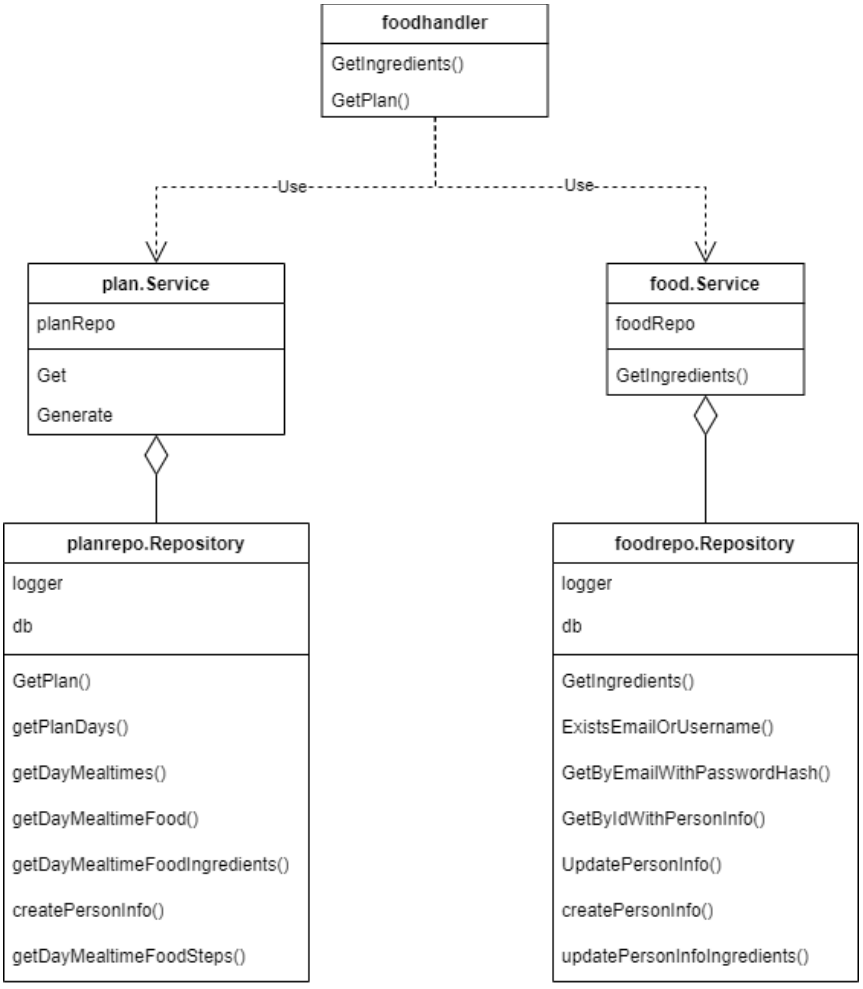


Рисунок 13 – Сервис питания

На диаграмме представлены классы:

– foodHandler – пакет, содержащий функции обработчики для входящих запросов связанных с блюдами;

– food.Service – сервис реализует бизнес-логику работы с блюдами;

– plan.Service – сервис реализует бизнес-логику работы с планом питания;

– planrepo.Repository – репозиторий, реализующий основные операции для работы с БД, связанных с планом питания;

– foodrepo.Repository – репозиторий, реализующий основные операции для работы с БД, связанных с блюдами.

Диаграмма классов для сервиса авторизации представлена на рисунке 14.

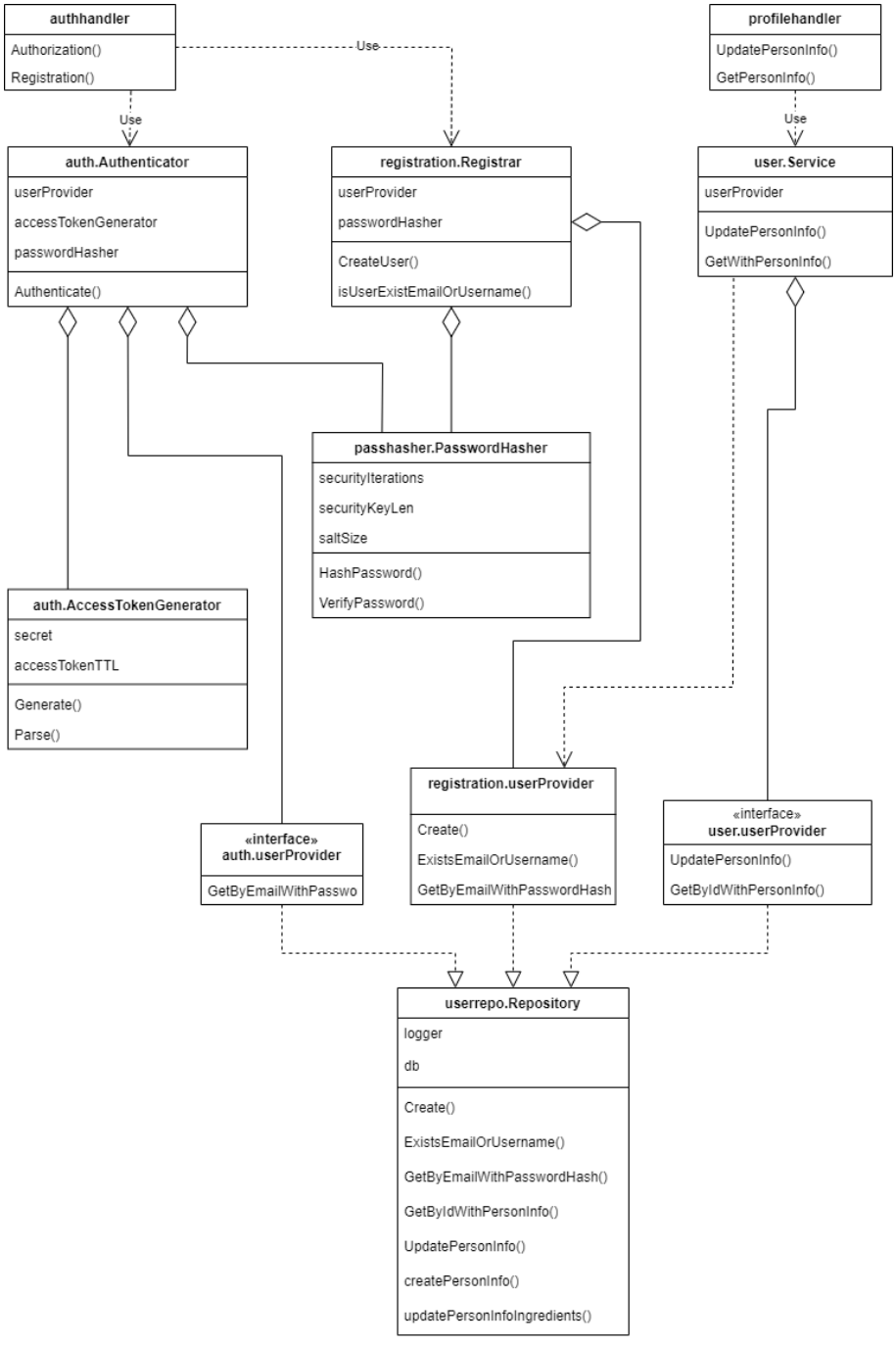


Рисунок 14 – Сервис авторизации

На диаграмме представлены классы:

– authHandler – пакет, содержащий функции обработчики для входящих запросов, связанных с авторизацией и регистрацией пользователей;

– auth.Authenticator – сервис реализует бизнес-логику авторизации пользователей;

– registration.Registrar – сервис реализует бизнес-логику регистрации пользователей;

– profileHandler – пакет, содержащий функции обработчики для входящих запросов, связанных с профилем пользователей;

– auth.AccessTokenGenerator – сервис реализует бизнес-логику работы с JWT-токенами;

– user.Service – сервис реализует бизнес-логику работы с пользователями;

– passhasher.PasswordHasher – сервис, реализующий бизнес-логику безопасности хеширования и верификации пароля;

– auth.userProvider – интерфейс, обязывающий реализовать методы работы с пользователями во время авторизации;

– user.userProvider – интерфейс, обязывающий реализовать методы работы с пользователями;

– registration.userProvider – интерфейс, обязывающий реализовать методы работы с пользователями во время регистрации;

– userrepo.Repository – репозиторий, реализующий основные операции для работы с БД, связанных с пользователями.

Представленные сервисы отражают архитектуру серверной части приложения.

2.4 Проектирование пользовательского интерфейса

Для проектирования графического интерфейса используется несколько общих правил:

– интерфейс должен быть интуитивно понятным;

– возможность возвращения пользователя в то место, где он закончил работу в прошлый раз;

– программа должна взаимодействовать с пользователем на основе наименьшей значимой единицы ввода;

– разработка интерфейса исходя из принципа наименьшего возможного количества действий со стороны пользователя [14].

Страница авторизации и регистрации по умолчанию находится в состоянии с одним открытым блоком ввода логина и пароля (рисунок 15).

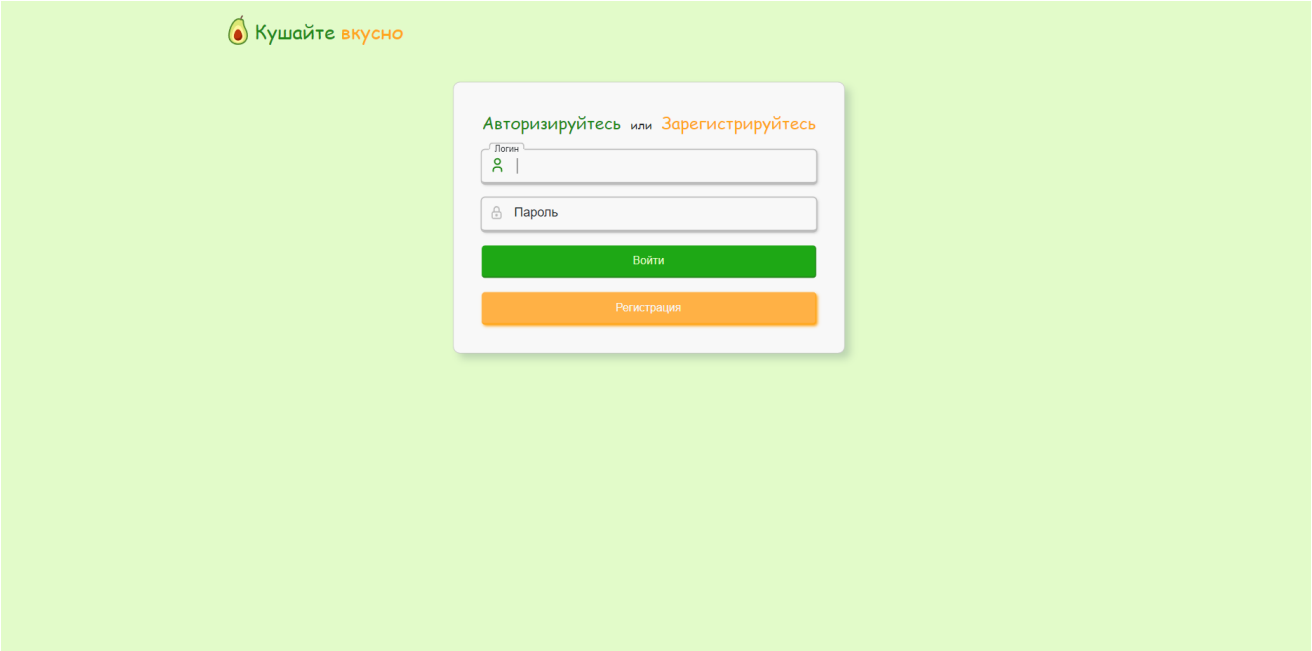


Рисунок 15 – Страница авторизации и регистрации

При нажатии на кнопку «Регистрация», не переходя на новую страничку инициируется заполнение анкеты данных, необходимых для создания профиля пользователя.

Существует несколько общепринятых способов сбора информации посредством элементов интерфейса, самые популярные из них: форма и мастер.

Форма – это стандартный способ сбора данных, когда пользователь заполняет поля ввода информации (например, имя, фамилия, адрес электронной почты) и отправляет их на сервер. Формы располагаются чаще всего во «всплывающих окнах», могут быть различной сложности и содержать различные

типы полей, такие как текстовые поля, поле выбора, флажки, радиокнопки и прочие интерактивные элементы.

Мастер – это более интерактивный способ сбора данных, который направляет пользователя через последовательность шагов, задавая вопросы и предоставляя опции для выбора. Мастеры могут быть более удобными и легкими в использовании для пользователей, поскольку они обычно предоставляют более простой и понятный интерфейс, чем формы и располагаются на переключаемых страницах-шагах. Мастеры могут быть особенно полезны при сборе сложных данных, таких как сведения о покупке товара или создание профиля пользователя.

Оба способа имеют свои преимущества и недостатки, и выбор между ними зависит от целей и потребностей конкретного проекта. Для этого проекта был выбран и модифицирован под особенности дизайна метод «Мастера».

Сбор информации начинается с ознакомительного шага (рисунок 16).

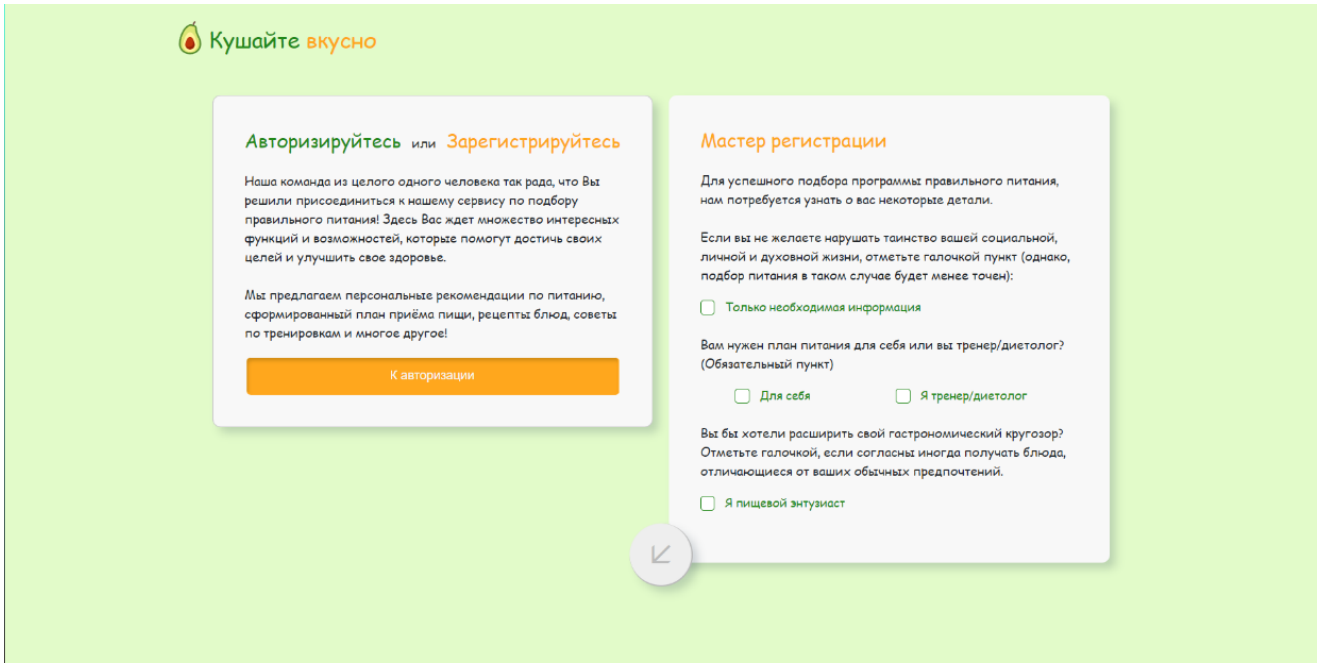


Рисунок 16 – Первый шаг мастера регистрации

Переключение шагов осуществляется по нажатию на кнопку с иконкой стрелки, располагающейся в нижней части блока. Пока не будут заполнены и

провалидированы обязательные поля шага, кнопка будет находиться в заблокированном состоянии.

При переходе на новый шаг, соответствующий блок будет отрисован в шахматном порядке по направлению вниз. Интерфейс пользователя автоматически будет пролистан до уровня нового шага.

Подобный подход позволяет держать всю заполняемую пользователем информацию у него в прямом доступе, при этом не перегружая интерфейс и оптимально используя центральное пространство страницы.

Поля ввода данных мастера сообщают об ошибках по мере заполнения шага (рисунок 17).

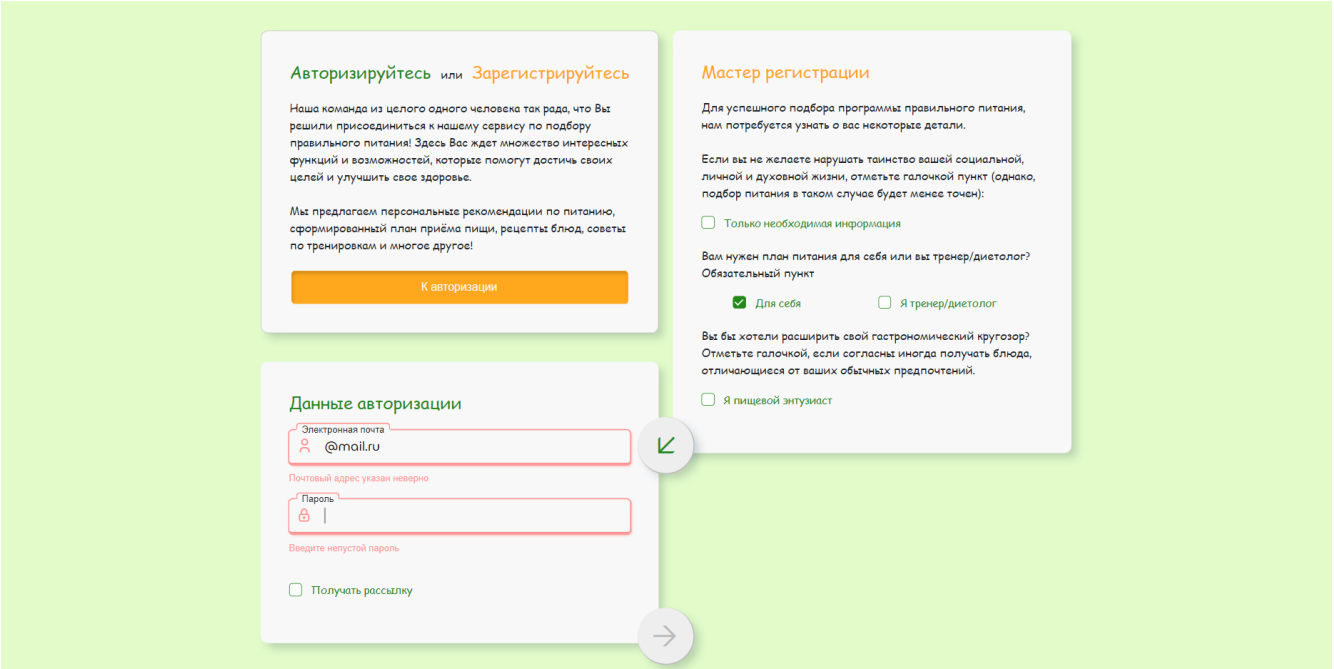


Рисунок 17 – Отображение ошибок

После заполнения последнего шага мастера, появляется благодарность пользователю за заполнение анкеты и кнопка «Завершить», по нажатию которой происходит регистрация и переход в сервис при её успешном завершении (рисунок 18).

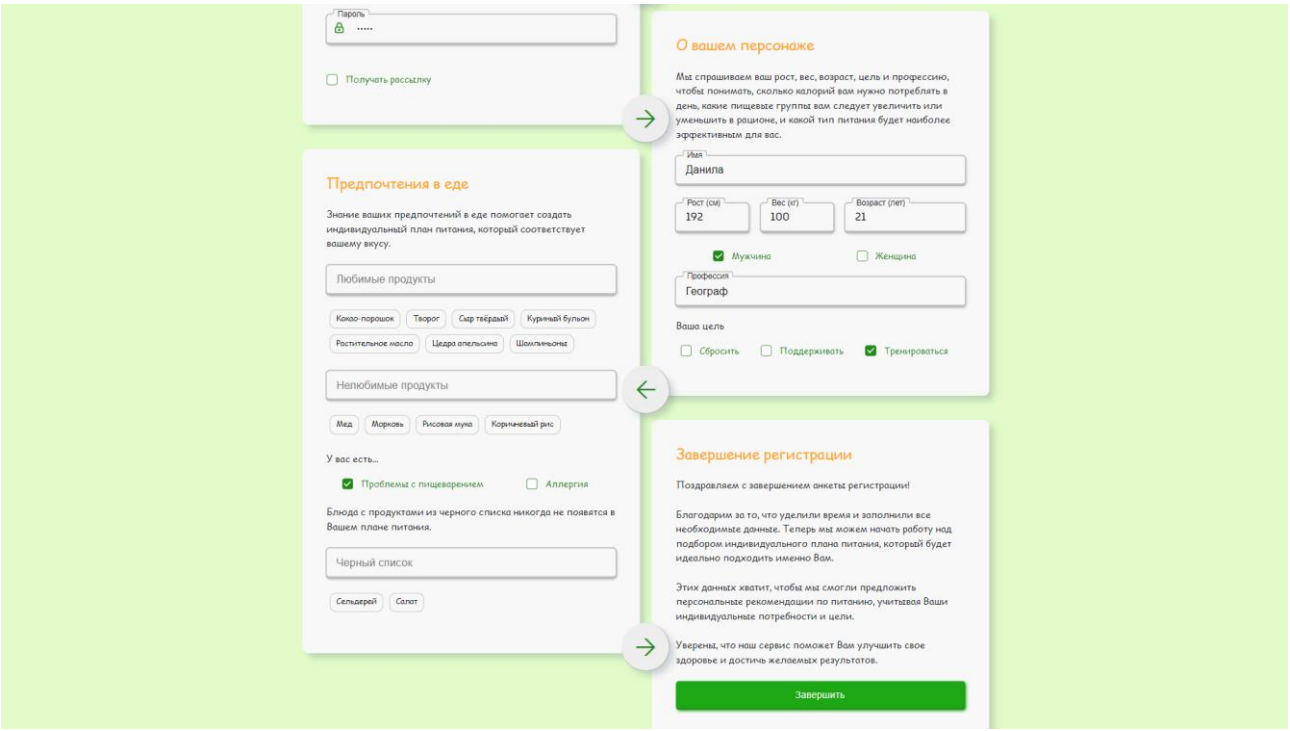


Рисунок 18 – Оставшиеся шаги и завершение мастера

Самые основные элементы должны располагаться на главной странице, в данном случае – план питания и сами блюда. Все остальные – интуитивно понятны для пользователя по расположению (рисунок 19).

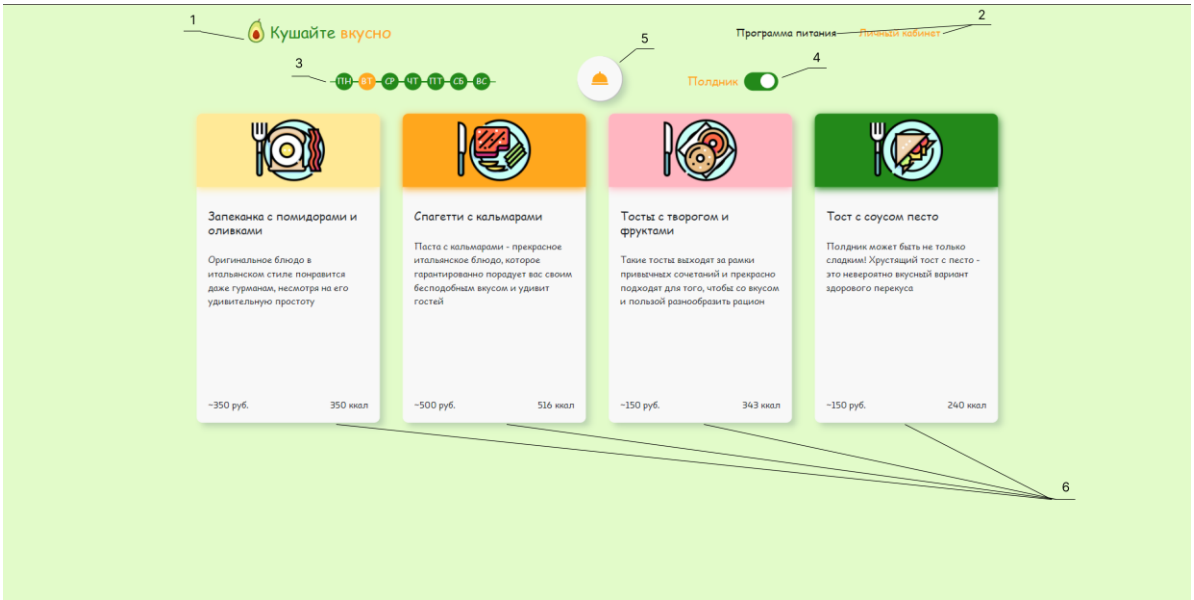


Рисунок 19 – Главная страница

На рисунке 19 изображены элементы интерфейса:

1 – логотип и название сервиса «Кушайте много» – «Кушайте вкусно», отвечает за перемещение на основную страницу, доступную по роуту «/main»;

2 – навигация для доступа к любой странице из любого места приложения;

3 – слайдер дней недели, осуществляет переключение по частям плана питания, соответственно дням;

4 – переключатель «Полдник» добавляет и убирает полдники в выбранный день плана питания;

5 – кнопка создания плана питания, позволяет переформировать план с учётом обновлённых пользовательских данных;

6 – карточка блюд, которые состоят из иконки приёма пищи (завтрак, обед, полдник, ужин) и краткой информации: названии, количество калорий, приблизительных стоимости и времени приготовления.

По нажатию на блюдо открывается рецепт его приготовления. Вариант страницы с открытым рецептом представлен на рисунке 20.

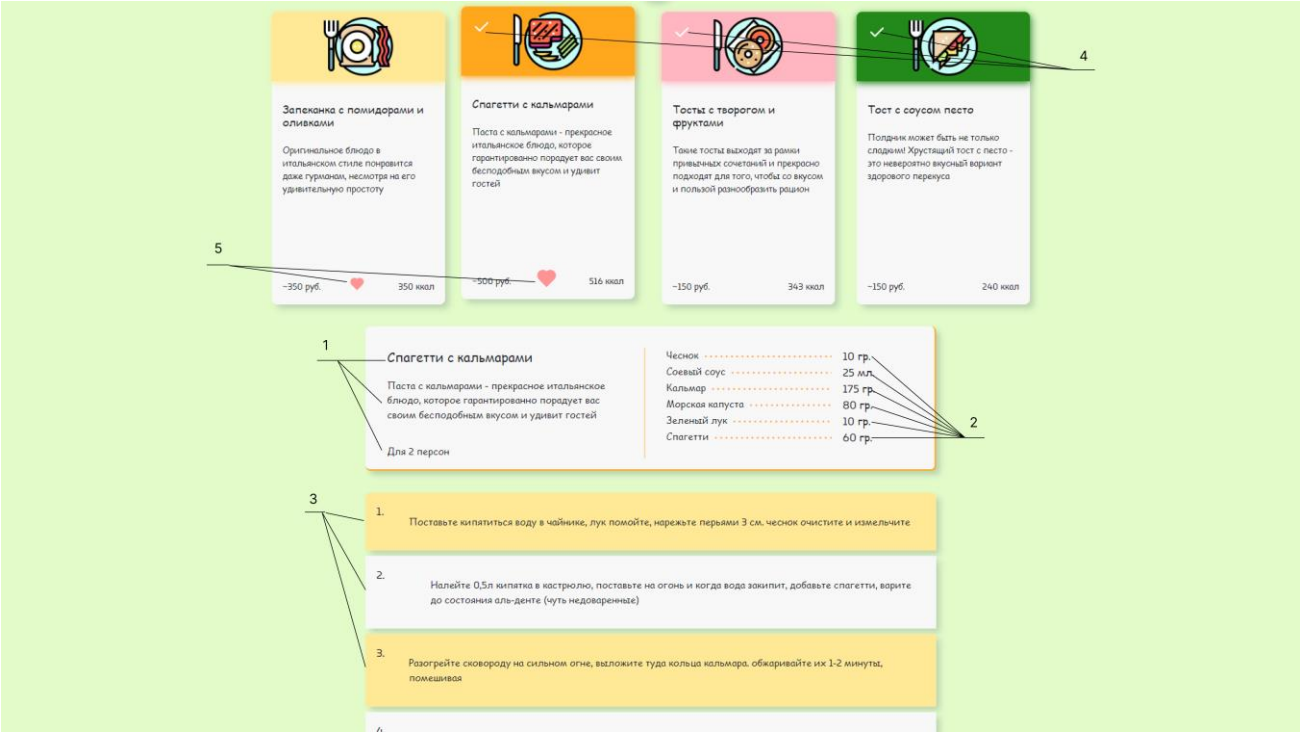


Рисунок 20 – Главная страница с выбранным рецептом блюда

На рисунке 20 изображены элементы интерфейса:

1 – информация о рецепте, содержит название, краткое описание, количество порций;

2 – ингредиенты, используемые для приготовления блюда и их необходимое количество;

3 – шаги рецепта приготовления блюда;

4 – иконки отметки пройденных блюд;

5 – иконки отметки любимых блюд.

Страница профиля доступна авторизованным пользователям по клику на ссылку «Личный кабинет» в шапке сервиса (рисунок 21).

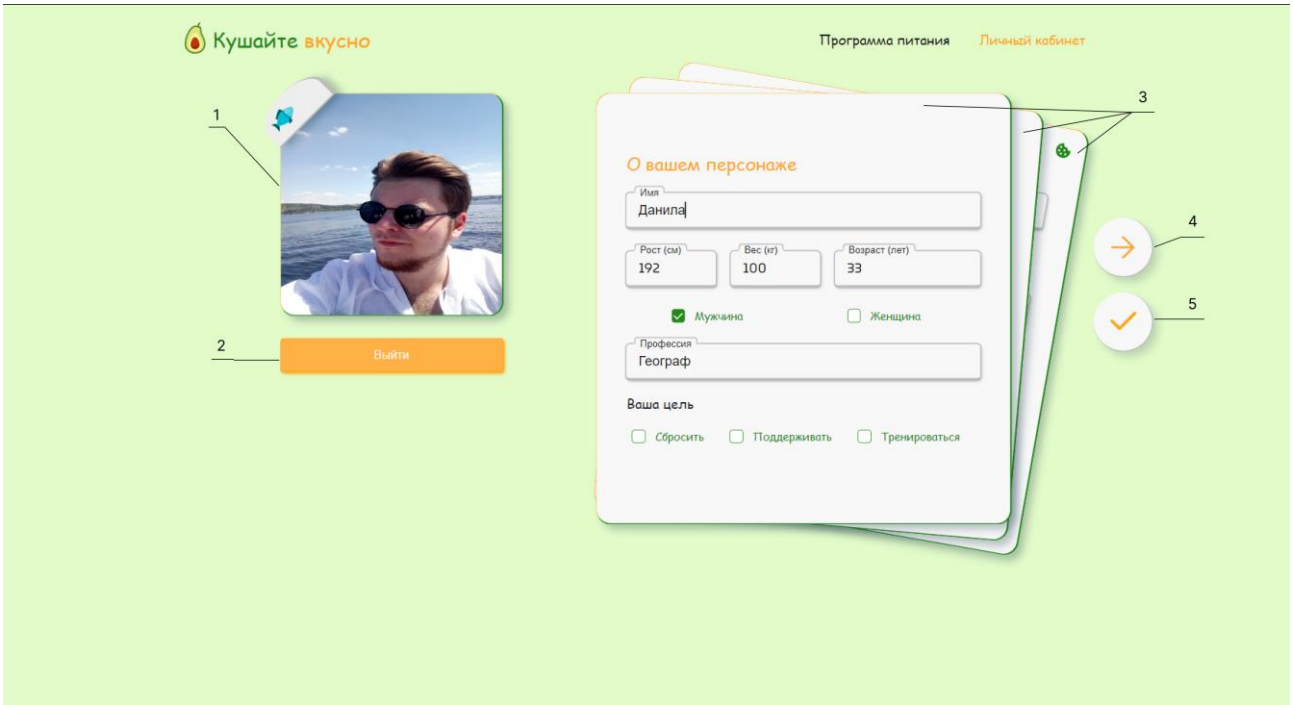


Рисунок 21 – Страница профиля

На рисунке 21 изображены элементы интерфейса:

1 – аватар профиля, при наведении и клике на картинку, загружается новое изображение;

2 – кнопка выхода из аккаунта пользователя;

3 – раздел редактирования пользовательских данных, представляет из себя набор карточек, соответствующих шагам мастера регистрации.

4 – кнопка переключения карточек редактирования;

5 – кнопка подтверждения сохранения отредактированных данных пользователя, появляется при изменении хотя бы одного поля.

Переключение карточек пользовательских данных происходит также по клику на отдельные карточки, находящиеся позади главной.

Таким образом, был спроектирован логичный и простой в использовании интерфейс, а также был сформирован новый взгляд на сбор пользовательских данных в виде вертикального одностраничного интерактивного мастера. В приложении были реализованы такие принципы материального дизайна, как: схожий графический стиль элементов, обратная связь на действия пользователя, и однородность элементов, выполняющих схожие функции.

3 Реализация веб-приложения по подбору правильного питания

3.1 Реализация архитектуры приложения «Кушайте много»

Диаграмма развертывания показывает, как будет выглядеть система при ее развертывании на физических узлах. Диаграмма развёртывания содержит графические изображения устройств, процессов и связей между ними [15].

Система состоит из ПК пользователя, с помощью которого он осуществляет взаимодействие с ней через веб-браузер, сервера, написанного на Golang, позволяющего принимать запросы пользователя, отправлять REST запросы на сервер БД, получать ответ и передавать его пользователю, СУБД PostgreSQL, позволяющей принимать запросы сервера, обрабатывать их и возвращать ответ.

Диаграмма развертывания приложения планирования путешествий представлена на рисунке 22.

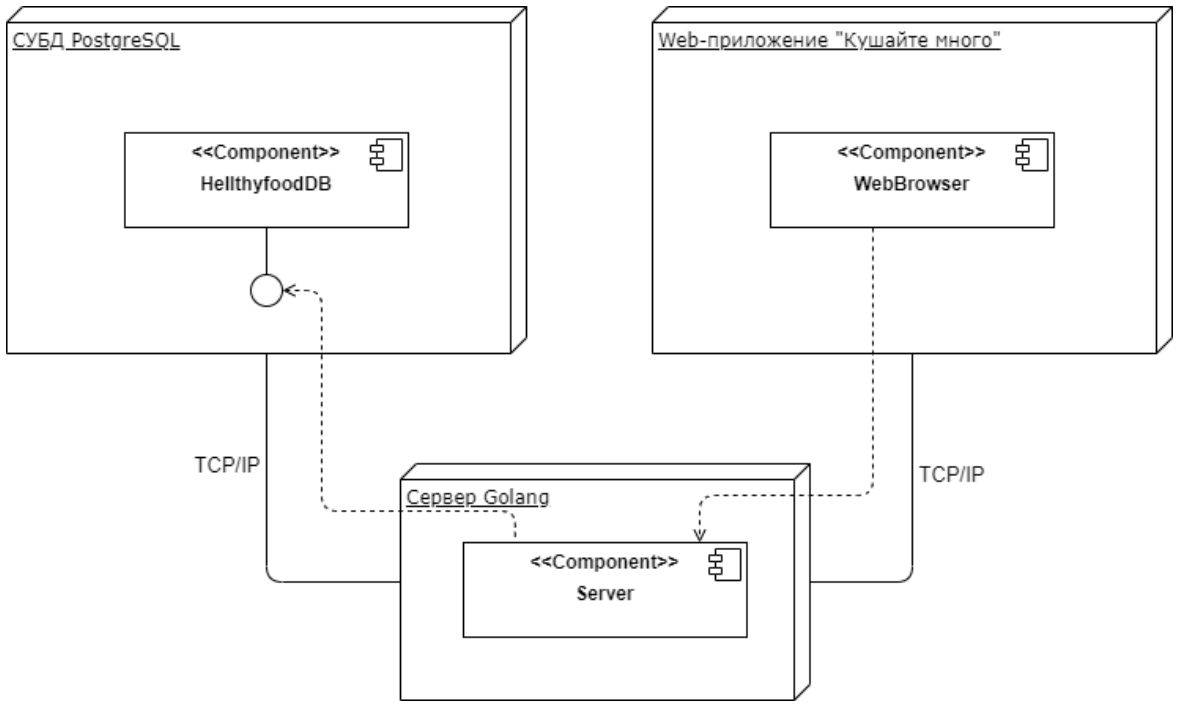


Рисунок 22 – Диаграмма развертывания

Взаимодействие СУБД и сервера происходит через интерфейс БД HellthyfoodDB. Взаимодействие приложения в веб-браузере пользователя с сервером и с сервером БД происходит через TCP/IP соединение.

3.2 Реализация базы данных

На основе логической модели БД была спроектирована и реализована база данных «HellthyFoodDB». Инструментами PostgreSQL была создана физическая модель, которая отличается от логической тем, что учитывает особенности той СУБД, в которой реализована система.

Построенная физическая модель представлена на рисунке 23.

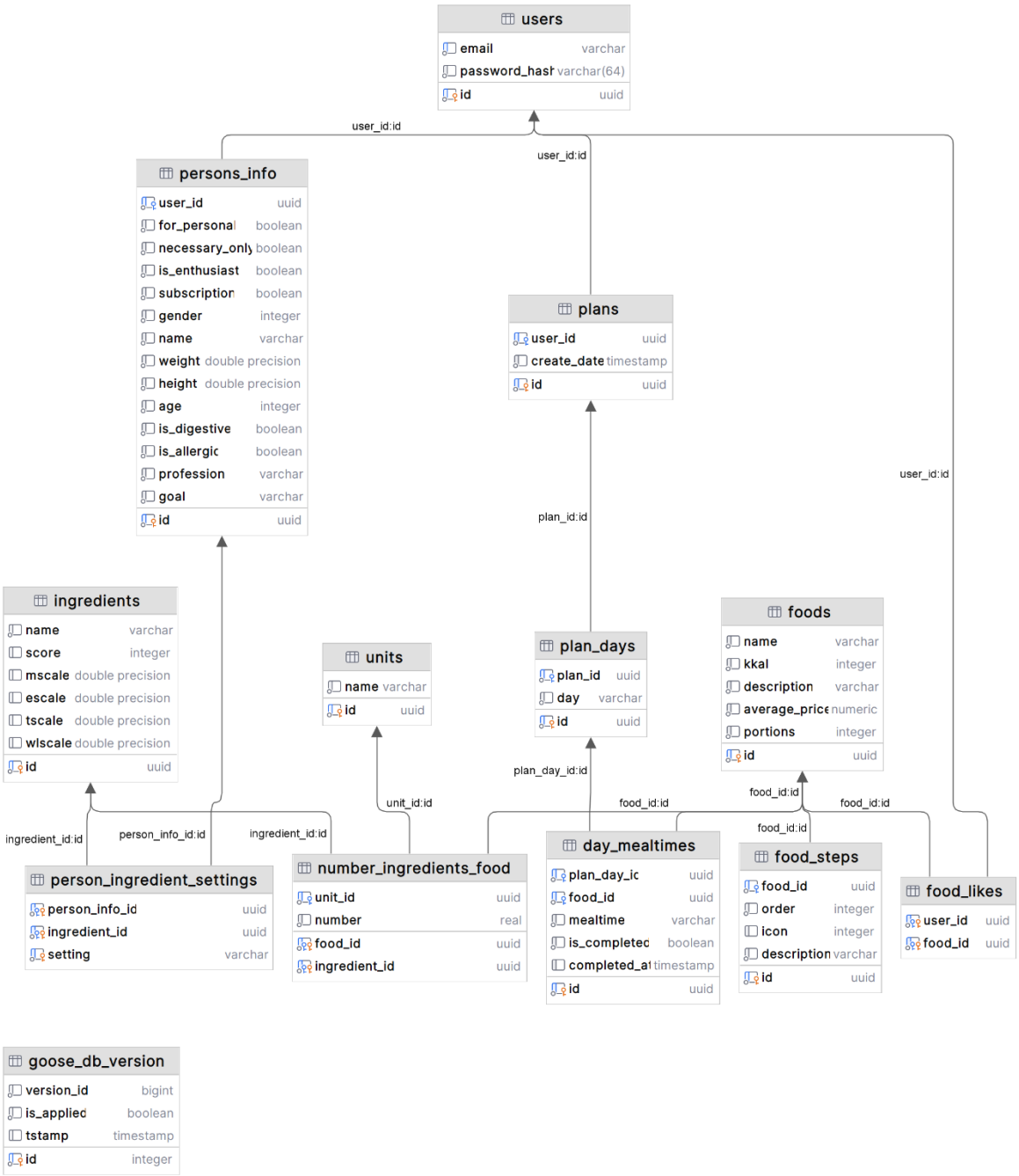


Рисунок 23 – Физическая схема БД

После проектирования, данная модель была реализована в СУБД PostgreSQL. В процессе реализации сущности модели стали таблицами, атрибуты – столбцами, идентификаторы таблиц – первичными ключами, а связи между таблицами были реализованы с помощью вторичных ключей.

3.3 Реализация программных средств

Для отображения отношений между объектами и последовательности этих отношений, на примере прецедента «Авторизация» была разработана диаграмма кооперации, представленная на рисунке 24.

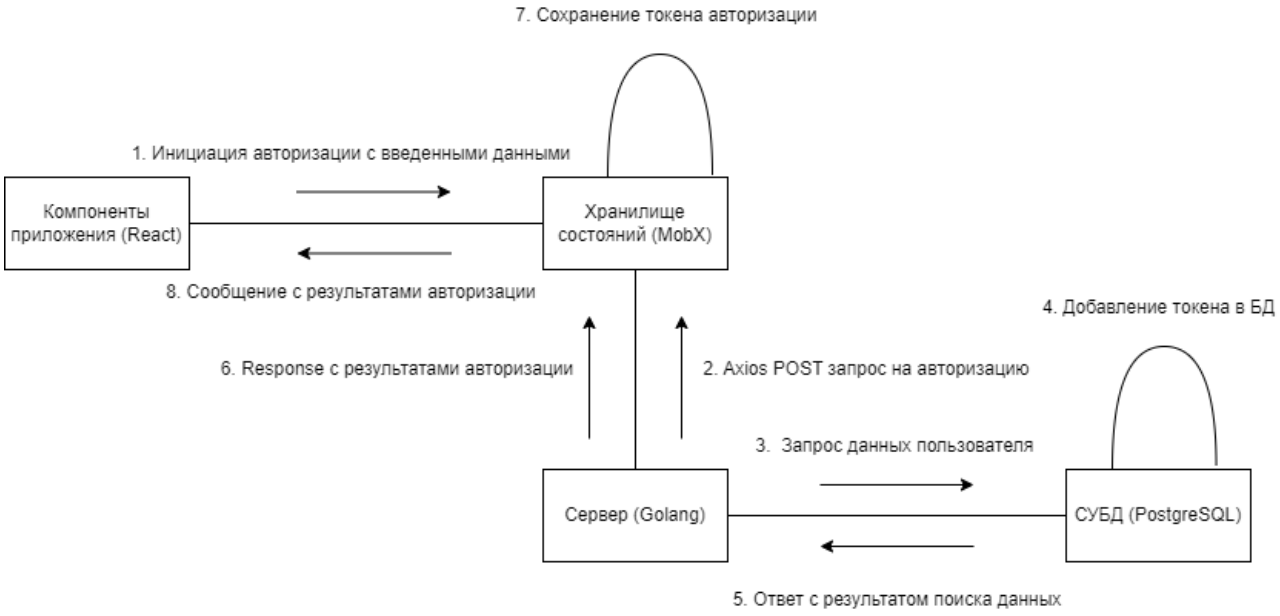


Рисунок 24 – Диаграмма кооперации прецедента «Авторизация»

Диаграмма описывает процесс авторизации пользователя в сервисе. Она включает в себя четыре объекта: «Компоненты приложения (React)», «Хранилище состояний (MobX)», «Сервер (Golang)» и «СУБД (PostgreSQL)», которые обмениваются сообщениями.

После ввода данных авторизации, Компонент приложения через Хранилище состояний с помощью Axios POST запроса, отправляет запрос на сервер. Запрос принимает сервис авторизации. Данные валидируются, после чего формируется запрос к базе данных на существование профиля с подходящими данными авторизации. СУБД проверяет данные и генерирует токен, который

приходит в ответе на сервер. На сервере формируется response с результатами авторизации и токеном в БД, после чего через Хранилище состояний токен сохраняется, отправляется сообщение об успехе или неудаче в Компонент приложения.

Для отображения состояний системы во время выполнения сценария прецедента «Авторизация» в режиме регистрации была разработана диаграмма состояний, представленная на рисунке 25.

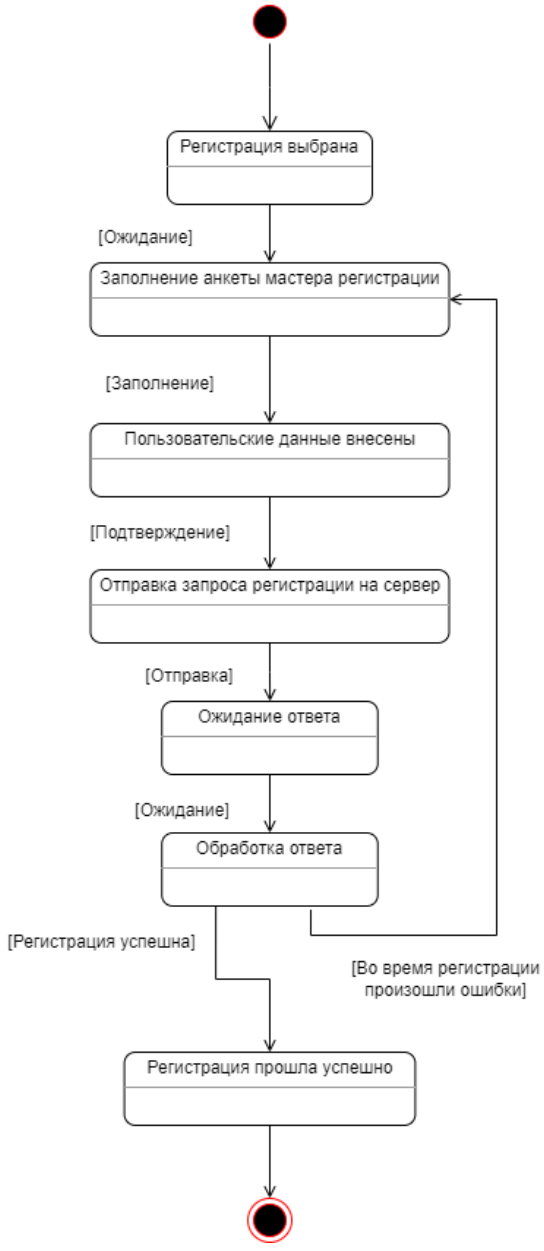


Рисунок 25 – Диаграмма состояний варианта использования «Авторизация»

Диаграмма описывает состояния, в которых бывает система при регистрации нового пользователя. Ее начало совпадает выбором регистрации на странице авторизации, а конец – с успешной регистрацией в сервисе.

Описание состояний системы:

– «Регистрация» обозначает, что пользователь инициировал начало новой регистрации и раскрытие мастера нажатием на кнопку «Регистрация»;

– «Заполнение анкеты мастера регистрации» – состояние обозначающее, что система находится в ожидании того, когда пользователь заполнит все шаги мастера регистрации;

– «Пользовательские данные внесены» говорит о том, что пользователь внес в систему данные, необходимые для регистрации и формирования плана питания;

– «Отправка запроса регистрации на сервер» обозначает, что клиент нажал на кнопку «Завершить регистрацию» последнего шага мастера и отправил данные на сервер;

– «Ожидание ответа» – клиент ожидает ответа от сервера, содержащего сведения об успешности сохранения данных в БД и регистрации нового профиля;

– «Обработка ответа» означает, что клиенту пришел ответ от сервера, и система перешла в состояние обработки результатов;

– «Регистрация прошла успешно» говорит об успешной регистрации в сервисе.

Если зарегистрировать новый профиль не удалось, пользователю предлагается возможность изменить вносимые данные. При успешном добавлении данных пользователь регистрируется и авторизуется в сервисе.

3.4 Реализация клиентского приложения

Функциональное назначение разрабатываемого веб-приложения – предоставление персонализированных планов правильного питания, формируемых на основе характеристик и предпочтений пользователя.

Аналитическая информационная система имеет следующее основные функции:

– заполнение пользовательской анкеты данных;

– создание и просмотр сформированного персонального плана правильного питания;

– сохранение и просмотр любимых блюд и их рецептов приготовления.

Для реализации программного продукта используется язык программирования JavaScript, желательно версии стандарта ES6+. Данная версия работает быстрее и стабильнее, а также имеет обширный набор встроенных функций высокоуровневого программирования [16].

Использованная система управления базами данных – PostgreSQL версии 15.1 и выше для хранения данных системы. Данная СУБД способна справляться с одновременной обработкой нескольких задач, поддерживает большое количество типов данных и вспомогательных инструментов для работы.

В основе веб-приложения используется шаблон программирования MVC. При таком подходе, модель данных, представление и бизнес-логика, разделены на отдельные компоненты [17].

Хранение и обработка данных осуществляется посредством общего класса-хранилища данных с использованием фреймворка MobX. Доступ к хранилищу для всех компонентов приложения, вне зависимости от их уровня иерархии, предоставляется проводником контекста библиотеки React.js.

Для отображения взаимосвязей между компонентами создана диаграмма компонентов, которая описывает физическое представление системы, ее архитектуру. Также диаграмма компонентов детально описывает структурные принципы и архитектуру приложения и позволяет установить направление потока данных и как объекты взаимодействуют друг с другом [18].

Данная диаграмма для клиентского приложения сервиса «Кушайте много» представлена на рисунке 26.

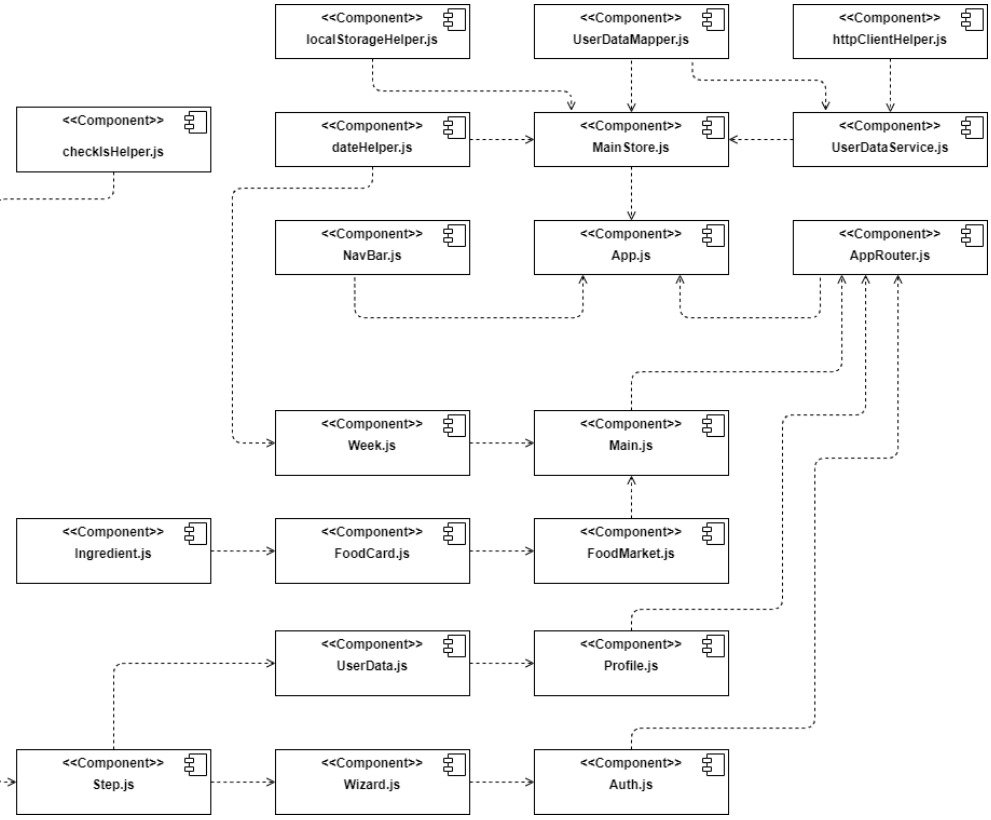


Рисунок 26 – Диаграмма компонентов сервиса «Кушайте много»

На рисунке 26 изображены следующие компоненты:

– App.js – родительский компонент приложения, предоставляет контекст дочерним компонентам;

– NavBar.js – шапка сервиса, обеспечивающая навигацию по страницам;

– AppRouter.js – компонент, обеспечивающий маршрутизацию между страницами;

– MainStore.js – класс-хранилище данных, менеджер состояния приложения, обеспечивающий единую сеть данных между всеми компонентами;

– UserDataService.js – сервис, содержащий функции, отправляющие запросы данных на сервер;

– httpClientHelper.js – вспомогательный компонент, реализующий методы GET, POST, PUT, DELETE Rest-запросов на основе Axios;

– UserDataMapper.js – вспомогательный компонент, форматирующий объекты данных;

– localStorageHelper.js – вспомогательный компонент, обеспечивающий работу с локальным хранилищем браузера;

– dateHelper.js – вспомогательный компонент, обеспечивающий работу с датами внутри приложения;

– checkIsHelper.js – вспомогательный компонент, выполняющий проверки данных на соответствие определённым типам или структурам данных;

– Main.js – компонент главной страницы приложения, имеющий состояния загрузки плана питания и его просмотра;

– Week.js – компонент-переключатель дней недели, позволяющий просматривать дни плана питания;

– FoodMarket.js – компонент плана питания, обеспечивающий отображения блюд в составе плана, а также рецепта конкретного выбранного блюда;

– FoodCard.js – компонент блюда, содержащий иконку приёма пищи, название блюда и краткую информацию о нём;

– Ingredient.js – компонент ингредиента блюда в составе рецепта;

– Profile.js – компонент профиля пользователя, представляющего личный кабинет, в котором можно просмотреть и изменить пользовательские данные;

– UserData.js – компонент, отображающий данные пользователя в виде карточек, соответствующих шагам мастера регистрации;

– Auth.js – компонент страницы авторизации, содержащий форму ввода данных, используемых для входа в аккаунт, и мастера регистрации нового профиля;

– Wizard.js – компонент-мастер регистрации, обеспечивающий отображение и переключение шагов мастера;

– Step.js – компонент шаг мастера, отображающий поля для ввода сгруппированных данных.

Перечисленные компоненты образуют оптимизированную и расширяемую архитектуру клиентского приложения.

4 Контроль качества веб-приложения подбора правильного питания

Тестирование программного обеспечения – процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определённым образом.

В ходе анализа способов тестирования программного обеспечения было определено, что более подходящими из них для данного приложения являются: тестирование методом черного ящика, тестирование метрики кода, а также автоматизированное тестирование.

Для определения сложности программного кода обработки пользовательских данных при авторизации были использованы метрики Холстеда.

4.1 Функциональное тестирование

Тестирование чёрным ящиком – это метод тестирования программного обеспечения, который основан на проверке функциональности системы, не учитывая её внутреннюю структуру и реализацию. Этот метод тестирования позволяет оценить работоспособность системы, проверив её на входных и выходных данных, а также на соответствие её функциональным требованиям. Тестирование чёрным ящиком может быть использовано на любом этапе разработки ПО, и позволяет выявить ошибки и дефекты в работе системы.

Проанализировав варианты использования и функциональные требования, был подготовлен список наиболее возможных сценариев использования сайта и системы управления содержимым сайта, который реализует требуемые от программного продукта функции (таблица 16).

Таблица 16 – Сценарии тестирования приложения

№	Название тестового случая	Тест	Результат
1	Отображение страницы авторизации, если пользователь не авторизован	Страница авторизации будет отображена	Страница авторизации отображена, переход на другие страницы приводит к переадресации (приложение Г.1)
2	Отображение ошибки при вводе неправильных данных авторизации	Поле ввода перейдёт в состояние ошибки	Поле ввода переведено в состояние ошибки, отображено сообщение (приложение Г.2)
3	Отображение мастера регистрации при выборе регистрации	Мастер отобразит первый шаг	Первый шаг мастера отображен (приложение Г.3)
4	Отображение ошибки в вводе неправильных данных в поля мастера регистрации	Поле ввода шага мастера регистрации перейдёт в состояние ошибки	Поле ввода переведено в состояние ошибки, отображено сообщение (приложение Г.4)
5	Отображение корректных данных зарегистрированного пользователя	Профиль пользователя будет содержать правильные данные	Карточки профиля заполнены правильными данными пользователя (приложения Г.5 – Г.7)
6	Изменение пользовательских данных в профиле	Данные будут корректно изменены	Пользовательские данные изменены успешно (приложение Г.8)
7	Корректное формирование персонального плана питания	Будет сформирован план питания, не содержащий продуктов из черного списка	Персональный план питания успешно сформирован (приложение Г.9)
8	Корректное изменение состава плана питания, путём удаления полдника	Полдник будет убран из плана конкретного дня	Полдник успешно убран из плана питания (приложение Г.10)
9	Просмотр рецепта блюда	Рецепт отобразится в интерфейсе для прочтения	Рецепт выбранного блюда отображен в сервисе (приложение Г.11)
10	Пометка блюда «пройденным»	Блюдо будет помечено как «пройденное»	Блюдо успешно помечено (приложение Г.12)
11	Пометка блюда «любимым»	Блюдо будет помечено как «любимое»	Блюдо успешно помечено (приложение Г.13)
12	Загрузка аватара в профиль пользователя	Изображение будет загружено и отобразится в профиле	Аватар успешно загружен (приложение Г.14)

Функциональное тестирование более подробным образом может быть рассмотрено на примере прецедента «Авторизация». В данном прецеденте для

успешного входа в приложение нужно ввести корректный логин и пароль, которые зарегистрированы в системе.

Для реализации тестирования методом черного ящика были использованы функциональная диаграмма, классы эквивалентности и таблица решений [19].

4.1.1 Определение классов эквивалентности

Разработка тестов методом эквивалентного разбиения осуществляется в два этапа: выделение классов эквивалентности и построение тестов.

Для тестирования поля ввода «Логин» были выделены следующие классы эквивалентности, представленные в таблице 17.

Таблица 17 – Классы эквивалентности

	Правильные КЭ	Неправильные КЭ
Длина логина	3 - 30 (1)	0-2 (2) 31- ∞ (3)
Структура	Начинается с буквы (4)	Начинается не с буквы (5)
Содержание	Буквы Латинские (6) Цифры (7) Нижнее подчеркивание (8)	Остальное (9)

На основании классов эквивалентности были сформированы возможные ситуации, которые представлены в таблице 18.

Таблица 18 – Функциональное тестирование поля ввода «Логин»

№	КЭ	Входные данные	Выходные данные
1	Все правильные 1, 4, 6, 7, 8	Omegadro	Успешно
2	2, 4, 6, 7	XD	Ошибка: меньше 3х символов
3	3, 4, 6, 7, 8	Gzhegosch_bzhentschischtschikevitsch	Ошибка: больше 30 символов

Продолжение таблицы 18

№	КЭ	Входные данные	Выходные данные
4	1, 5, 6, 7, 8	_X_Killer_2008_X_	Ошибка: начинается не с буквы
5	1, 4, 9	@nime	Ошибка: имеет неразрешенный символ

Результаты тестирования приведены в приложении Г.

4.1.2 Анализ граничных условий

Анализ граничных условий может позволить найти ошибки, находящиеся на границах классов эквивалентности, а также может включать тесты на данных, выходящих за допустимый диапазон значений. В качестве исходного поля для тестирования было выбрано поле ввода «Логин». Для него были выделены следующие границы:

– физическая граница;
– произвольная граница;
– логическая граница.

Физическая граница – количество введенных символов равно нулю.

Произвольная граница – количество введенных символов должно быть в промежутке от 3 до 30 включительно.

Логическая граница – введённое значение должно состоять из латинских букв, цифр, а также может включать специальный символ «_».

Произвольная граница представлена на рисунке 27.

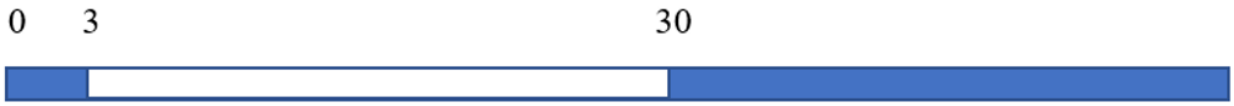


Рисунок 27 – Произвольная граница ввода поля «Логин»

Логическая граница представлена на рисунке 28.

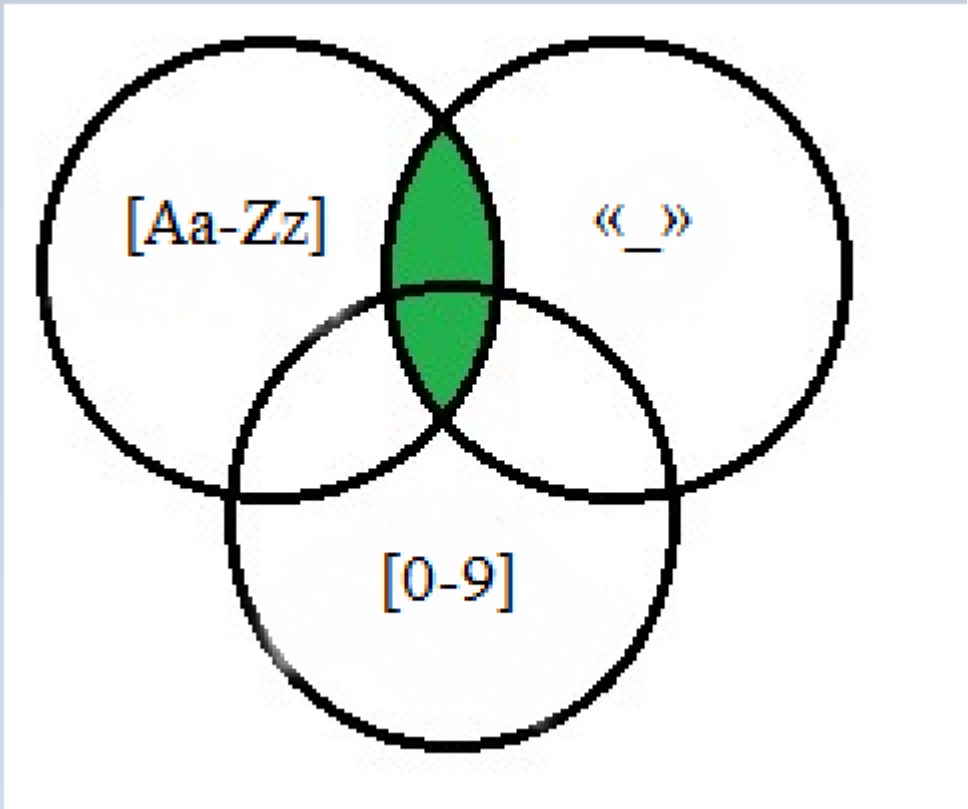


Рисунок 28 – Логическая граница ввода поля «Логин»

Тестовые наборы для проверки логических границ приведены в таблице 19.

Таблица 19 – Граничные условия

Граничное условие	Значение	Результат
Пустая строка		Ошибка (рисунок Д.7)
Количество символов в строке меньше 3	XD	Ошибка (рисунок Д.8)
Количество символов в строке равно 3	XDX	Успех (рисунок Д.9)
Количество символов в строке равно 30	sweet_explorer_adventurer_2021	Успех (рисунок Д.10)
Количество символов в строке больше 30	sweet_explorer_adventurer_2021_2024	Ошибка (рисунок Д.11)
Строка содержит только допустимые символы	omegadro_2	Успех (рисунок Д.12)
Строка содержит недопустимые символы	omegadro-2	Ошибка (рисунок Д.13)
Строка содержит русские буквы	логин	Ошибка (рисунок Д.14)

На основе выделенных пограничных условий была построена функциональная диаграмма.

4.1.3 Построение функциональной диаграммы

Метод функциональных диаграмм или диаграмм причинно-следственных связей помогает систематически выбирать высокорезультативные тесты. Кроме этого, метод функциональных диаграмм дает полезный побочный эффект, так как позволяет обнаруживать неполноту и неоднозначность исходных спецификаций.

Для тестирования были выбраны поля ввода «Логин» и «Пароль» формы авторизации.

Таблица причин и следствий, построенная для тестирования формы регистрации нового пользователя, представлена в таблице 20.

Таблица 20 – Причины и следствия

Причины	
П1	Корректная длина логина
П2	Корректная длина пароля
П3	Логин состоит из корректных символов
П4	Пароль состоит из корректных символов
П5	Логин свободен
Следствия	
С1	Сообщение: «Некорректная длина логина»
С2	Сообщение: «Некорректная длина пароля»
С3	Сообщение: «Некорректные символы в составе логина»
С4	Сообщение: «Некорректные символы в составе пароля»
С5	Сообщение: «Логин занят»
С6	Сообщение: «Регистрация прошла успешно»

Для графического представления причин и следствий была построена функциональная диаграмма, отражающая связь между входными и выходными данными.

Функциональная диаграмма представлена на рисунке 29.

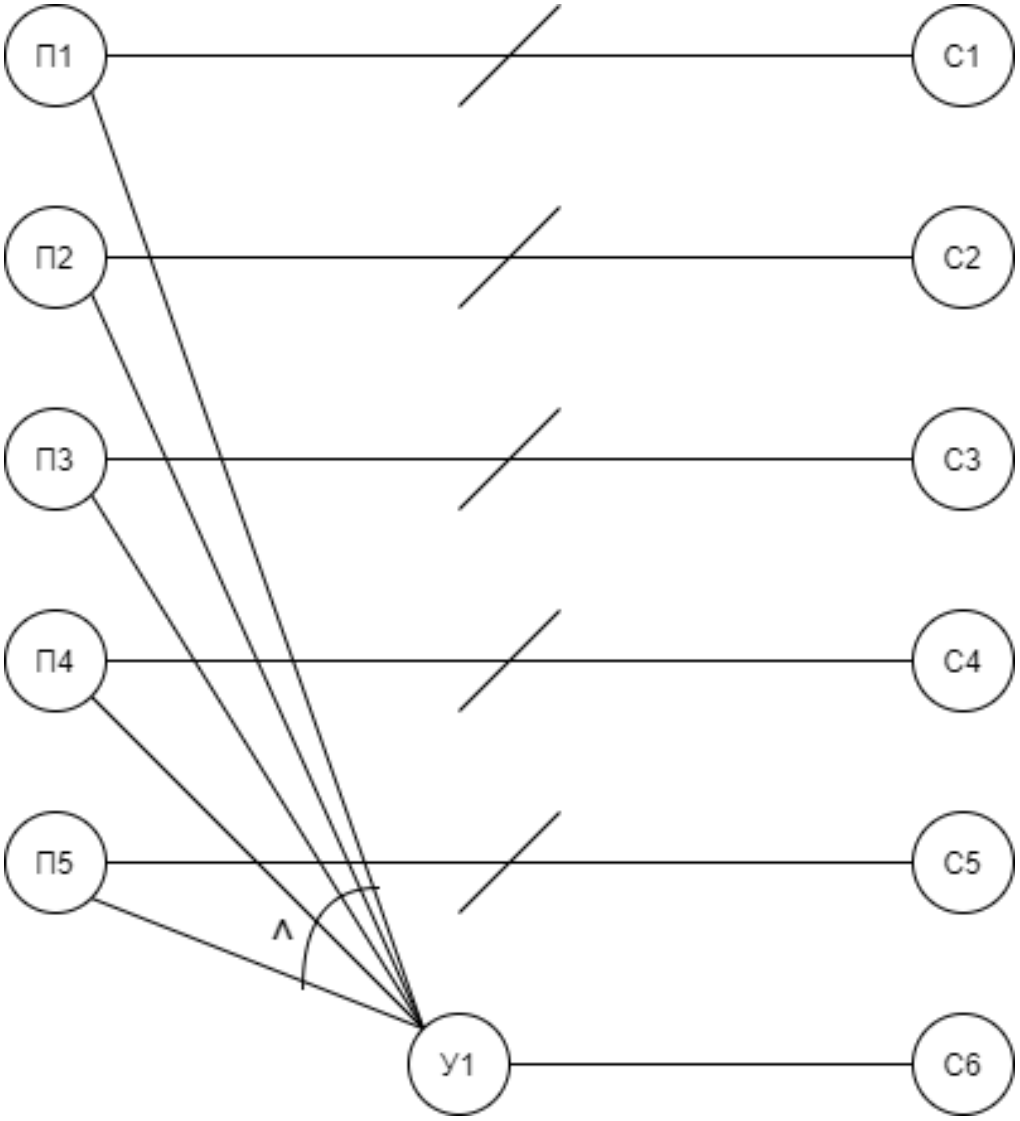


Рисунок 29 – Функциональная диаграмма причин и следствий

На диаграмме показано, что если первая причина не выполняется, то выполняется первое следствие. Таким же образом, если последующие причины не выполняется, то выполняется соответствующие следствия.

В случае, если все причины успешно выполнены, то выполняется шестое следствие, которое сообщает о том, что заказ оформлен успешно.

На основе таблицы причин и следствий была построена таблица решений, отражающая, как зависят друг от друга следствия и причины на разных комбинациях.

Таблица решений представлена в таблице 21.

Таблица 21 – Таблица решений

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
П1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1
П2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	1
П3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	1
П4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
П5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
С1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0
С2	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0
С3	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
С4	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
С5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
С6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Далее были составлены тест-кейсы, представленные в таблице 22.

Таблица 22 – Таблица решений: тест-кейсы

№	Входные данные	Ожидаемый результат
1	Логин: log Пароль: psw	Некорректная длина логина, Некорректная длина пароля
2	Логин: Sigurd Пароль: 123	Некорректная длина пароля
3	Логин: @r Пароль: password	Некорректная длина логина, Некорректные символы в составе логина, Некорректные символы в составе пароля
4	Логин: Gotlib? Пароль: 1234ab#c	Некорректные символы в составе логина, Некорректные символы в составе пароля
5	Логин: slightly___morethen30simbolsinlogin Пароль: morethen20s!mbols!np@ssword	Некорректная длина логина, Некорректные символы в составе пароля
6	Логин: Bjorn Пароль: ^)(=|/	Некорректная длина пароля, Некорректные символы в составе пароля
7	Логин: Br Пароль: ftgf5?,t	Некорректная длина логина, Некорректные символы в составе пароля
8	Логин: Ivar Пароль: SummerB!rd	Некорректные символы в составе пароля
9	Логин: ThrowTheFireAndFlameAndIronTubes% Пароль: 1234	Некорректная длина логина, Некорректная длина пароля, Некорректные символы в составе логина

Продолжение таблицы 22

№	Входные данные	Ожидаемый результат
10	Логин: Ulfberht& Пароль: anotherveryveryverylongpassword	Некорректная длина пароля, Некорректные символы в составе логина
11	Логин: @notherveryveryv@rylonglonglogin Пароль: pass sword	Некорректная длина логина, Некорректные символы в составе логина
12	Логин: gud-mund Пароль: 1q2w3e4r5t	Некорректные символы в составе логина
13	Логин: correctbatveryveryverylonglogin Пароль: correctbatveryveryverylongpassword	Некорректная длина логина, Некорректная длина пароля
14	Логин: Toke Пароль: veryveryveryverylongpassword123	Некорректная длина пароля
15	Логин: lg Пароль: drowssap	Некорректная длина логина
16	Логин: (занятый логин) Пароль: correctpassword	Логин занят
17	Логин: (свободный логин) Пароль: correctpassword	Регистрация прошла успешно

В таблице перечислены входные и соответствующие выходные данные.

4.2 Метрики программного кода проекта

Для оценки сложности программного кода обработки регистрации пользователей была использована метрика Холстеда.

Основу метрики Холстеда составляют четыре измеряемых характеристики программы:

– число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов);

– число уникальных операндов программы (словарь операндов);

– общее число операторов в программе;

– общее число операндов в программе.

Для анализа метрик был выбран наиболее часто встречающийся в архитектуре сервиса компонент – поле ввода данных. Программный код обработки полей ввода данных авторизации пользователей представлен в листинге 1.

Листинг 1 – Функция для валидации введенного значения

```
export const validateInput = (value, type, maxLength) => {
      if (!value) {
            return true;
      }
      if (value.length > maxLength) {
            return false;
      }
            if (type === InputTypeEnum.numbers && !numbersOnly(value, type)) {
            return false;
      }
            return ! (type === InputTypeEnum.password &&
      passwordDenySpaces(value, type));
};
```

Данный код является функцией validateInput, которая принимает три параметра: value, type и maxLength. Функция возвращает результат проверки на валидность входных данных.

Первый if-блок проверяет, что значение value не является пустым. Если значение пустое, то функция возвращает true, что означает, что входные данные являются валидными.

Второй if-блок проверяет, что длина value не превышает максимально допустимой длины maxLength. Если длина value больше, чем maxLength, то функция возвращает false, что означает, что входные данные не являются валидными.

Третий if-блок проверяет тип вводимых данных type. Если type равен InputTypeEnum.numbers, то функция вызывает функцию numbersOnly(value, type), которая проверяет, что значение value содержит только числа. Если значение value не содержит только числа, то функция возвращает false, что означает, что входные данные не являются валидными.

В последней строке функция проверяет, что тип вводимых данных равен InputTypeEnum.password, и вызывает функцию passwordDenySpaces(value, type), которая проверяет, что значение value не содержит пробелы. Если значение value содержит пробелы, то функция возвращает true, что означает, что входные данные не являются валидными.

Если все проверки пройдены успешно, то функция возвращает true, что означает, что входные данные являются валидными.

Для использования метрики Холстеда требуется подсчитать количество операторов и операндом. Искомые данные об операторах функции представлены в таблице 23.

Таблица 23 – Операторы

Операторы	Число вхождений
=	1
!	3
.	3
if	3
===	2
return	3
;	4
,	2
{	3
}	3
>	1
passwordDenySpaces	1
numbersOnly	1
&&	2
true	1
false	2
n1 = 16	N1 = 35

Данные об операндах функции представлены в таблице 24.

Таблица 24 – Операнды

Операнды	Число вхождений
value	4
type	4
InputTypeEnum	2
maxLength	1
n2 = 4	N2 = 11

Исходя из этих данных, можно рассчитать промежуточные значения, используя которые можно получить искомую сложность функции.

Длина программы рассчитывается по формуле:

$$N = N1 + N2 = 35 + 11 = 46.$$

Словарь программы рассчитывается по формуле:

$$n = n1 + n2 = 16 + 4 = 20.$$

Объем программы рассчитывается по формуле:

$$HV = N * \log_2 n,$$

$$\mathrm{H}V = 46 * 4{,}248 = 198{,}8.$$

Потенциальный объем программы:

$$V^* = \left(2 + n`\right) * \log 2\left(2 + n`\right),$$

$$V^* = 4 * \log 2(4) = 4{,}75489.$$

Теоретический уровень программы рассчитывается по формуле:

$$L` = \frac{V^*}{HV},$$

$$L` = \frac{4{,}75489}{198{,}8} = 0{,}023917.$$

Сложность программы рассчитывается по формуле:

$$HD = \left(\frac{n1}{2}\right) * \left(\frac{N2}{n2}\right),$$

$$HD = \left(\frac{16}{2}\right) * \left(\frac{11}{4}\right) = 22.$$

Исходя из расчетов, было определено, что сложность функции validateInput равна 22, что является показателем простой по сложности функции.

4.3 Unit-тестирование приложения

Клиентскую часть приложения, написанную на JavaScript с применением фреймворка-библиотеки React.js эффективнее всего тестировать с использованием Jest – фреймворком для тестирования JavaScript. Он позволяет писать unit-тесты с приемлемым, знакомым и функциональным API, и быстро достигать желаемых результатов. Jest хорошо документирован, требует

минимальной настройки и может быть расширен, чтобы соответствовать любым требованиям.

Для Unit-тестирования выбран фрагмент программного кода обработки авторизации из предыдущего раздела.

Реализованные Unit-тесты представлены в листинге 2.

Листинг 2 – Unit-тесты обработки авторизации

```
describe(`validateInput function tests`, () => {
      it(`should validate text type input correctly`, () => {
            const result = validateInput(`xddddddd`, InputTypeEnum.text, 10);
            const expected = true;

            expect(result).toEqual(expected);
      });

      it(`should validate invalid text type input correctly`, () => {
            const result = validateInput(`xddddddd`, InputTypeEnum.text, 6);
            const expected = false;

            expect(result).toEqual(expected);
      });

      it(`should validate number type input correctly`, () => {
            const result = validateInput(`1.01`, InputTypeEnum.number, 6);
            const expected = true;

            expect(result).toEqual(expected);
      });

      it(`should validate invalid number type input correctly`, () => {
            const result = validateInput(`1.a01`, InputTypeEnum.number, 6);
            const expected = false;

            expect(result).toEqual(expected);
      });

      it(`should validate password type input correctly`, () => {
            const result = validateInput(`password`, InputTypeEnum.password,10);
            const expected = true;

            expect(result).toEqual(expected);
      });

      it(`should validate invalid password type input correctly`, () => {
            const result = validateInput(`pass word`,
      InputTypeEnum.password,10);
            const expected = false;

            expect(result).toEqual(expected);
      });
});
```

Представленные Unit-тесты для выбранного сегмента кода:

– тестирование успешного ввода в текстовое поле;
– тестирование неуспешного ввода в текстовое поле;
– тестирование успешного ввода в числовое поле;
– тестирование неуспешного ввода в числовое поле;
– тестирование успешного ввода в поле с типом «пароль»;
– тестирование неуспешного ввода в поле с типом «пароль».

Тестирование производится отдельно запуском скрипта npm test в терминале IDE Webstorm, а также при каждом запуске приложения, проверяя работоспособность покрытого тестами кода.

Результат работы Unit-тестов показан на рисунке 30.

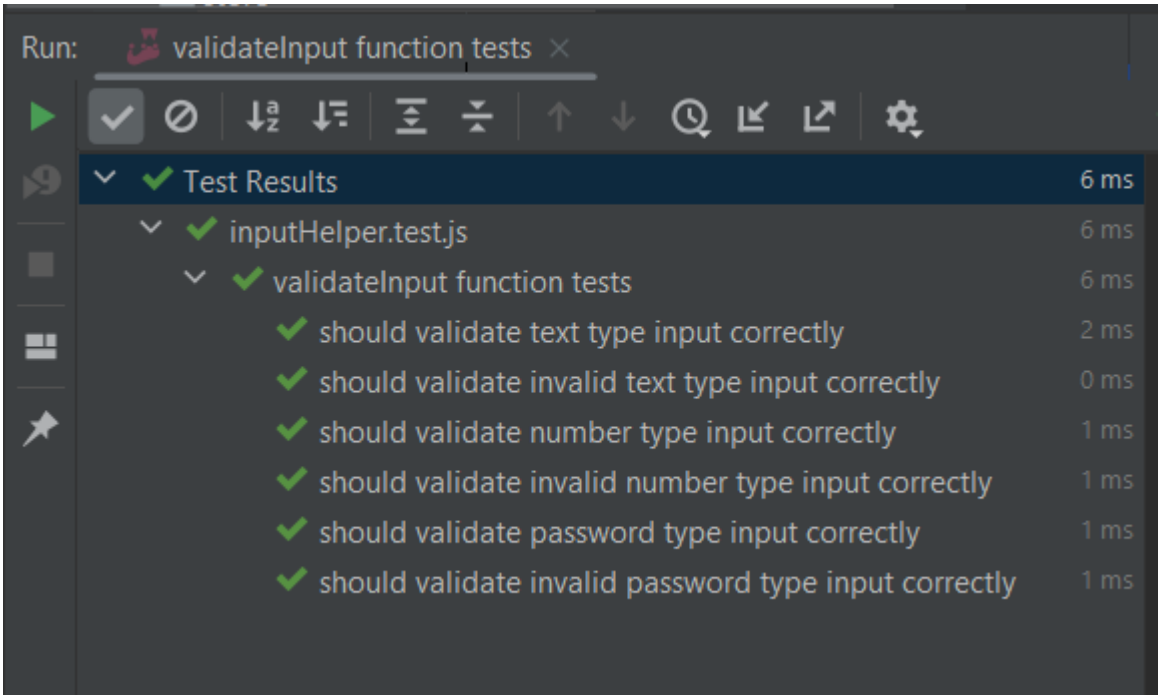


Рисунок 30 – Результат Unit-тестирования

Проверка Unit-тестированием выбранного фрагмента кода показывает положительный результат. Покрытый тестами код работает правильно.

5 Планирование и оценка бюджета проекта

Управление проектами – это деятельность, в ходе которой достигаются поставленные задачи и цели проекта, с учетом различных рисков, ограничений на ресурсы, качества, трудозатрат. Ключевым фактором в организации эффективного управления проектом является наличие заранее определенного плана и минимизация отклонений от него в процессе управления.

Для планирования и оценки стоимости построим диаграммы бизнес-процессов для выполнения бакалаврской работы, в нотации IDE в программной среде AllFusion Process Modeler 7. На рисунке 31 представлена контекстная диаграмма бизнес-процесса «Бакалаврская работа»

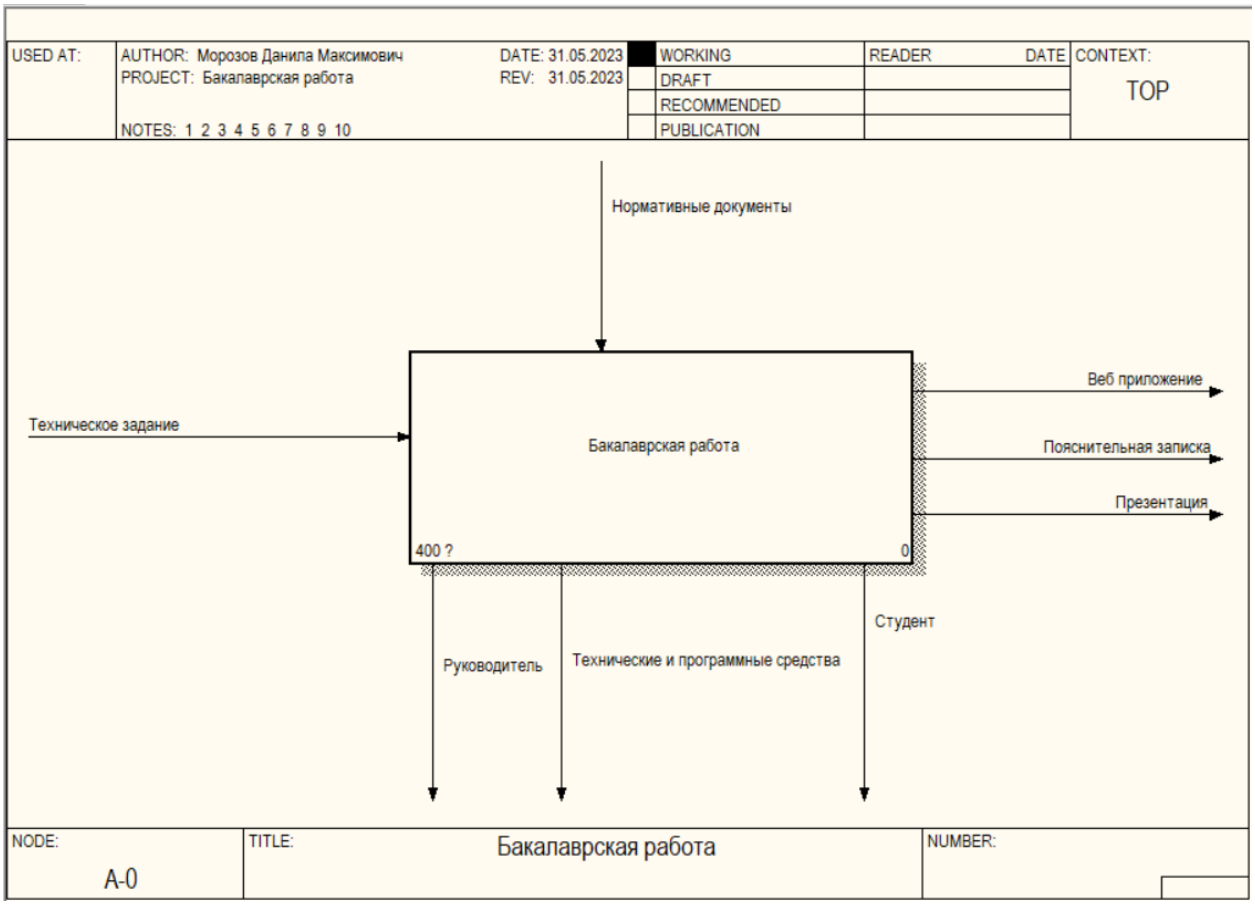


Рисунок 31 – Бизнес-процесс «Бакалаврская работа»

Основной бизнес-процесс был подвергнут формализации и декомпозиции. Результат декомпозиции можно увидеть на рисунке 32.

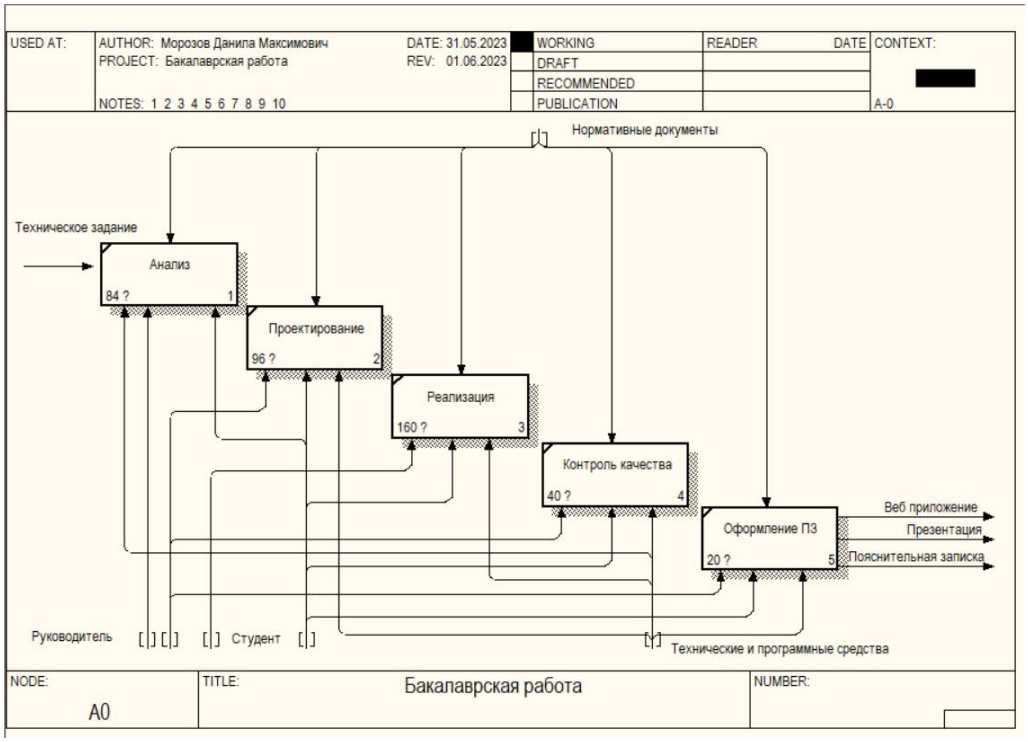


Рисунок 32 – Декомпозиция бизнес-процесса

При декомпозиции контекстной диаграммы процесс выполнения был разбит на 5 составляющих процессов: «Анализ предметной области и постановка задачи», «Проектирование», «Реализация», «Контроль качества», «Оформление пояснительной записки».

Для более удобного восприятия сроков работ по проекту, продолжительности этапов, а также отражения расхода ресурсов был использован простой и полезный инструмент календарного планирования проектов – диаграмма Ганта.

Для формирования диаграммы необходимо задать ресурсы проекта. Можно выделить несколько видов фундаментальных ресурсов:

– трудовые – работники, получающие зарплату и выполняющее интеллектуальную и физическую работу;

– материальные – расходные материальные ресурсы, используемые для производства и поддерживания процесса деятельности;

– невозобновляемые – энергоносители и прочие природные и техногенные ресурсы, возобновить запас которых невозможно.

Каждый ресурс имеет свои особенности, влияющие на их использование. Затраты на расходование этих ресурсов и составляют бюджет проекта. Информация о ресурсах, используемых в рамках разработки сервиса «Кушайте много» изображена на рисунке 33.

	Название	Тип	Максимальные единицы	Стандартная ставка	Ставка сверхурочных	Начислять
	Аналитик	Работа	100%	600 руб./час	1200 руб./час	Пропорционально
	Фронтенд	Работа	100%	400 руб./час	800 руб./час	Пропорционально
	Бэкенд	Работа	100%	400 руб./час	800 руб./час	Пропорционально
	Дизайнер	Работа	100%	300 руб./час	600 руб./час	Пропорционально
	Тестировщик	Работа	100%	250 руб./час	500 руб./час	Пропорционально
	Бумага	Материал		250 руб.		Пропорционально
	Интернет трафик	Материал		0 руб.		Пропорционально

Рисунок 33 – Ресурсы проекта

Далее выделенные ресурсы должны быть соотнесены с задачами проекта. Распределение ресурсов отображено на рисунке 34.

Название	Стоимость	Продолжи...	Начало	Окончание	Название ресурса
Бакалаврская работа	**252279,5 руб.**	**47,5 дней?**	**03.04.23 8:00**	**07.06.23 13:00**	
Анализ	**66029,5 руб.**	**10,5 дней?**	**03.04.23 8:00**	**17.04.23 13:00**	
Анализ предметной области	14400 руб.	3 дней?	03.04.23 8:00	05.04.23 17:00	Аналитик
Описание предметной области	4800 руб.	1 день?	06.04.23 8:00	06.04.23 17:00	Аналитик;Интернет трафик
Сравнительный анализ аналогов	22800 руб.	3 дней?	07.04.23 8:00	11.04.23 17:00	Аналитик;Дизайнер[50%];Фронтенд [50%];Интернет трафик
Анализ функциональных требований	5600 руб.	1 день?	12.04.23 8:00	12.04.23 17:00	Аналитик[50%];Фронтенд [50%];Бэкенд[50%];Интернет трафик
Анализ требований к пользовательскому интерфейсу	15229,5 руб.	2 дней?	13.04.23 8:00	14.04.23 17:00	Аналитик;Дизайнер;Фронтенд [70%]
Анализ требований к средствам разработки приложения	3200 руб.	0,5 дней?	17.04.23 8:00	17.04.23 13:00	Фронтенд ;Бэкенд
Проектирование	**76800 руб.**	**12 дней?**	**17.04.23 13:00**	**03.05.23 13:00**	
Описание базовой архитектуры системы	6400 руб.	1 день?	17.04.23 13:00	18.04.23 13:00	Аналитик[50%];Фронтенд ;Бэкенд
Проектирование базы данных	9600 руб.	3 дней?	18.04.23 13:00	21.04.23 13:00	Бэкенд
Проектирование программных средств	12800 руб.	2 дней?	21.04.23 13:00	25.04.23 13:00	Фронтенд ;Бэкенд
Проектирование пользовательского интерфейса	48000 руб.	6 дней?	25.04.23 13:00	03.05.23 13:00	Фронтенд ;Дизайнер;Аналитик[50%]
Реализация	**78400 руб.**	**20 дней?**	**03.05.23 13:00**	**31.05.23 13:00**	
Реализация базы данных	9600 руб.	3 дней?	03.05.23 13:00	08.05.23 13:00	Бэкенд
Реализация программных средств	12800 руб.	2 дней?	08.05.23 13:00	10.05.23 13:00	Фронтенд ;Бэкенд
Реализация пользовательского интерфейса	22400 руб.	7 дней?	10.05.23 13:00	19.05.23 13:00	Фронтенд
Отладка серверной части	16800 руб.	4 дней?	19.05.23 13:00	25.05.23 13:00	Тестировщик[50%];Бэкенд
Отладка клиентской части	16800 руб.	4 дней?	25.05.23 13:00	31.05.23 13:00	Тестировщик[50%];Фронтенд
Контроль качества	**31050 руб.**	**5 дней?**	**31.05.23 13:00**	**07.06.23 13:00**	
Ручное тестирование	13200 руб.	3 дней?	31.05.23 13:00	05.06.23 13:00	Тестировщик;Аналитик[50%]
Функциональное тестирование	2000 руб.	1 день?	05.06.23 13:00	06.06.23 13:00	Тестировщик
Составление программной документации	15850 руб.	1 день?	06.06.23 13:00	07.06.23 13:00	Фронтенд ;Бэкенд;Аналитик;Дизайнер;Тестировщик;Бумага
Сдача в эксплуатацию	13200 руб.	1 день?	07.06.23 13:00	08.06.23 13:00	Интернет трафик;Тестировщик;Бэкенд;Фронтенд ;Аналитик

Рисунок 34 – Распределение ресурсов

В Project Libre для планирования процесса разработки программного обеспечения был создан проект, дата начала которого 03.04.23.

На основе внесенных данных в проект была построена диаграмма Ганта, которая является визуальным представлением календарного плана (рисунок 35).

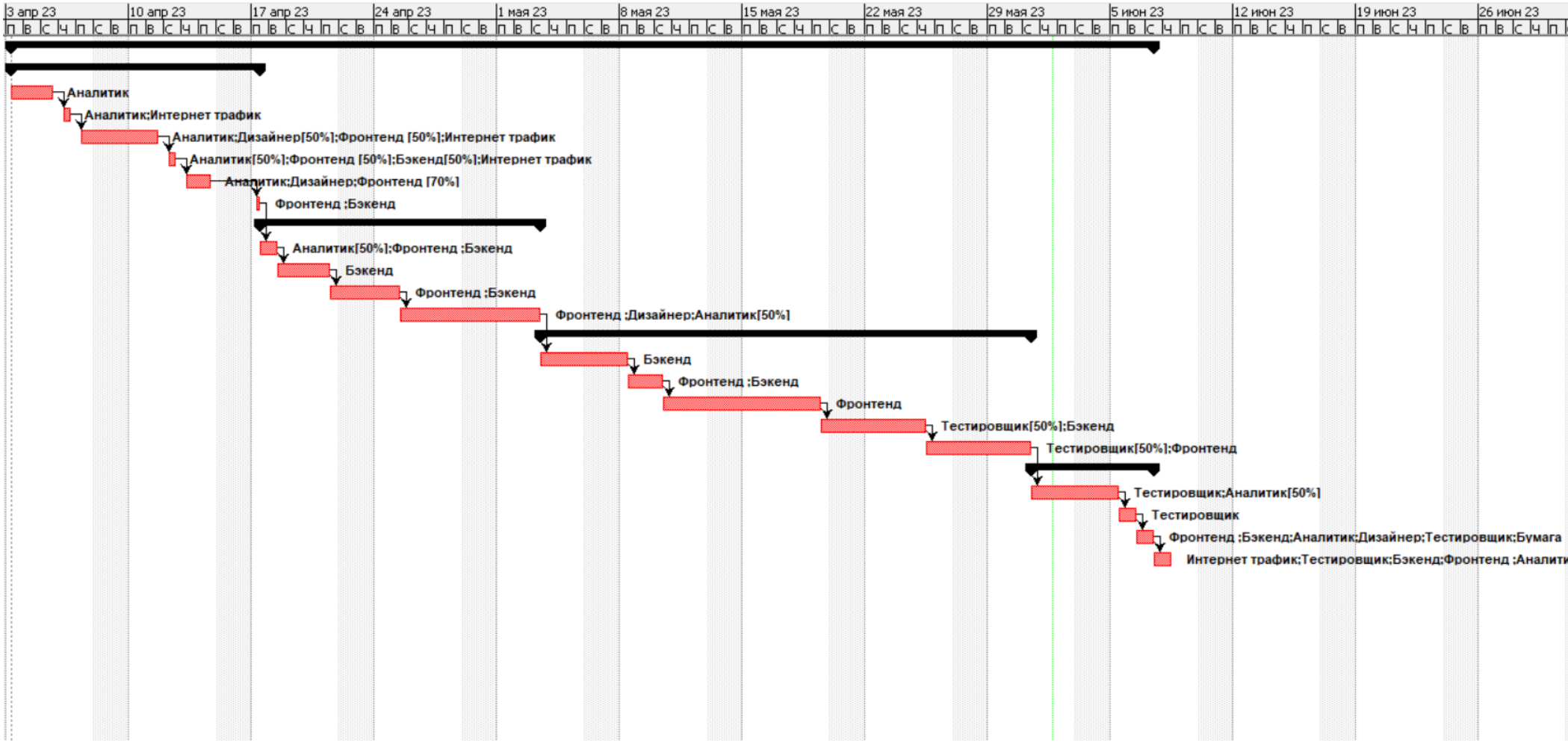


Рисунок 35 – Диаграмма Ганта разработанного проекта

С помощью инструмента планирования проектной деятельности Project Libre был получен обзорный отчет проекта планирования разработки приложения по подбору правильного питания, из которого видно, что срок выполнения проекта составил 47,5 дней, а бюджет проекта – ~252300 рублей. При помощи оценки сроков и бюджета проекта можно сделать вывод о том, что процесс разработки приложения является экономически выгодным.

Заключение

В результате выполнения выпускной квалификационной работы была разработана аналитическая информационная система подбора правильного питания.

На этапе анализа требований и предметной области были подобраны и проанализированы современные сервисы-аналоги в сфере формирования питания. В результате анализа была проведена сравнительная характеристика, в ходе которой были выявлены основные преимущества и недостатки рассмотренных сервисов, на основе которых были определены функциональные и дополнительные требования, а также составлено техническое задание на разработку веб-приложения.

Было проведено планирование проекта, определена длительность работ и их взаимосвязь. Составлена контекстная диаграмма бизнес-процессов и проведена их декомпозиция, после чего была составлена диаграмма Ганта. В ходе анализа этапов и их продолжительности было установлено, что на проектирование и разработку программного продукта потребуется 47,5 дней и 252300 рублей.

На основе технического задания было проведено проектирование программных средства и структур хранения данных для разрабатываемой системы. Были описаны логическая и физическая структура базы данных и составлена диаграмма классов базы данных.

На основе спроектированной модели было реализовано веб-приложение, учитывающее специфику работы системы подбора правильного питания. Описана архитектура приложения, составлена спецификация основных классов и методов. Клиентская часть приложения разработана на языке JavaScript, серверная часть на языке Golang.

Проведено функциональное и структурное тестирование, а также определены метрики программного кода проекта. Все функциональные

требования выполнены успешно, о чем свидетельствует отсутствие ошибок при функциональном тестировании и покрытие кода тестами.

Таким образом, в результате выполнения выпускной квалификационной работы разработано веб-приложение подбора правильного питания, отвечающее всем требованиям технического задания.

На данный момент система развёрнута на сервере и предоставляет открытый доступ в сети Интернет, имеет небольшую, но растущую группу активных пользователей. Сервис имеет большой потенциал для создания дополнительных модулей и дальнейшего развития.

Список использованных источников

1. Простое руководство по UML-диаграммам и моделированию баз данных // Microsoft [Электронный ресурс]. – URL: https://www.microsoft.com/ru-ru/microsoft-365/business-insights-ideas/resources/guide-to-uml-diagramming-and-database-modeling (дата обращения: 16.05.2023).

2. Диаграмма прецедентов // Creately [Электронный ресурс]. – URL: https://creately.com/blog/ru/диаграмма/учебное-пособие-по-диаграмма-прецеде/ (дата обращения: 16.05.2023).

3. Диаграмма активностей // Ensi [Электронный ресурс]. – URL: https://docs.ensi.tech/analyst-guides/tools/diagrams/uml/activity-diagram (дата обращения: 16.05.2023).

4. Формирование требований и классификация требований // Business Analysis [Электронный ресурс]. – URL: https://analytics.infozone.pro/formation-requirements-and-classification-requirements/ (дата обращения: 17.05.2023).

5. Что нужно знать о UX и UI // ItAndDigital [Электронный ресурс]. – URL: https://itanddigital.ru/bloghrconsulting/tpost/pd29e17zz1-polzovatelskii-opit-i-polzovatelskii-int (дата обращения: 18.05.2023).

6. 8 лучших IDE JetBrains для разработчиков // Intuit [Электронный ресурс]. – URL: https://bestprogrammer.ru/izuchenie/8-luchshih-ide-jetbrains-dlya-razrabotchikov (дата обращения: 19.05.2023).

7. СУБД PostgreSQL // Yandex Практикум [Электронный ресурс]. – URL: https://practicum.yandex.ru/blog/chto-takoe-subd-postgresql/ (дата обращения: 20.05.2023).

8. Стефанов, С. React. Быстрый старт / С. Стефанов – С.: Питер, 2023. – 304 с.

9. Использование Axios с React // Timeweb [Электронный ресурс]. – URL: https://timeweb.cloud/tutorials/react/ispolzovanie-axios-s-react (дата обращения: 21.05.2023).

10. Mobx // Habr [Электронный ресурс]. – URL: https://habr.com/ru/articles/282578/ (дата обращения: 21.05.2023).

11. Басс, Л. Архитектура программного обеспечения на практике / Л. Басс – С.: Питер, 2012. – 624 с.

12. Дейт, К. Введение в системы баз данных / К. Дейт – М.: Диалектика, 2019. – 1328 с.

13. Полное руководство по диаграмме классов UML // Cybermedian [Электронный ресурс]. – URL: https://www.cybermedian.com/ru/a-comprehensive-guide-to-uml-class-diagram/ (дата обращения: 27.05.2023).

14. Унгер, Р. UX-дизайн. Практическое руководство по проектированию опыта взаимодействия / Р. Унгер, К. Чендлер – М.: Символ, 2017. – 327 с.

15. Простое руководство к диаграммам развертывания UML // Creately [Электронный ресурс]. – URL: https://creately.com/blog/ru/uncategorized-ru/учебное-пособие-по-диаграмме-развёрт/ (дата обращения: 27.05.2023).

16. Симпсон, К. ES6 и не только / К. Симпсон – С.: Питер, 2018. – 336 с.

17. Что такое MVC // Skillbox [Электронный ресурс]. – URL: https://skillbox.ru/media/code/chto_takoe_mvc_bazovye_kontseptsii_i_primer_prilozheniya/ (дата обращения: 29.05.2023).

18. Простое руководство к диаграммам компонентов UML // Creately [Электронный ресурс]. – URL: https://creately.com/blog/ru/uncategorized-ru/учебное-пособие-по-компонентной-диаг/ (дата обращения: 29.05.2023).

19. Бейзер, Б. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем / Б. Бейзер – С.: Питер, 2004. – 320 с.

Приложение А
ГЛОССАРИЙ
(справочное)

SPA – это web-приложение, использующее единственный HTML-документ, как контейнер для всех web-страниц, организующее навигацию между ними и взаимодействие с пользователем через динамически подгружаемые HTML, CSS, JavaScript без необходимости перезагрузки.

MVC – модель-представление-контроллер (от англ. model-view-controller). Это способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения.

IDE – это программное приложение, представляющее из себя среду разработки, которое помогает программистам эффективно писать программный код.

Приложение Б
ИСХОДНЫЙ КОД КЛИЕНТСКОЙ ЧАСТИ
(рекомендуемое)

Листинг Б.2 – Компонент MainStore

```
import {makeAutoObservable, reaction, runInAction} from 'mobx';
import localStorageHelper from "../helpers/localStorageHelper";
import UserModel from "../models/UserModel";
import DateHelper from "../helpers/dateHelper";
import DaysEnum from "../enums/DaysEnum";
import BrunchEnum from "../enums/BrunchEnum";
import { mapIngredients, mapUserModelToSave } from
'../mappers/userDataMapper';
import {getIngredients, getFoodPlan, loginUser, registrateUser}
from "../services/userDataService";

export default class MainStore {
    userModel = { ...UserModel };
    _isAuth = false;
    token = null;
    day = DateHelper.getDayOfWeek();
    activeTab = null;
    isSnacksDisabled = true;
    recipesCache = [];
    ingredients = [];
    food = [];
    validationState = {
        auth: ``
    };
    pendingState = {
        auth: true,
        plan: true,
        ingredients: true,
    };
    constructor() {
        this.init();
        makeAutoObservable(this);
    }
    // ACTION //
    init = async () => {
        this.initAuth();
        this.updateSnacks();
        await this.initIngredients();
        await this.loadPlan();
    }
    initAuth = () => {
        const localToken = localStorageHelper.getLocalToken();
        if (!localToken) {
            this.setIsAuth(false);
            return null;
        }
        this.setIsAuth(true);
        this.setToken(localToken);
        reaction(() => this.isAuth, () => {
            this.loadPlan();
        });
```

```
    }
        setFoodChecked = (value, id) => {
        const currentMeal = this.foodByDay.find(meal => meal.id
=== id);
        if (!currentMeal) {
            return;
        }
        currentMeal.checked = value;
    }
    updateSnacks = () => {
        this.isSnacksDisabled =
localStorage.getItem(this.snacksStorageKey) === `true`;
    }
    updateUserData = (field, value) => {
        if (this.userModel.hasOwnProperty(field)) {
            this.userModel[field] = value;
        }
    }
    //  SETS  //
    setFood = food => {
        this.food = food || [];
    }
    setDay = day => {
        this.day = Object.values(DaysEnum).includes(day) ? day :
DaysEnum.monday;
        this.updateSnacks();
    }
        clear = () => {
        this.recipesCache = [];
        this.setFood();
    }
    unauthorise = () => {
        this.setToken(null);
        this.setIsAuth(false);
        this.clear();
        localStorageHelper.deleteLocalToken();
    }
    // COMPUTED //
    get snacksStorageKey() {
        return `snacks_disabled_${this.day}`;
    }
    get isAuth() {
        return this._isAuth;
    }
    get foodByDay() {
        const { data = [] } = this.food.find(food => food.day ===
this.day) || {};
        return this.isSnacksDisabled ? data.filter(food =>
food.brunch !== BrunchEnum.sweetSnack) : data;
    }
    }
```

Листинг Б.4 – Страница Main

```
import React, {useContext} from 'react';
import {observer} from "mobx-react-lite";
import {Context} from "../index";
import Container from "../components/common/Container/Container";
import {getFoodPlan} from "../services/userDataService";
import Week from "../components/Week/Week";
import FoodMarket from "../components/FoodMarket/FoodMarket";
import Switch, { Color } from "../components/common/Switch";
import './style.m.css';

const Main = observer(() => {
    const {main} = useContext(Context);
    return <React.Fragment>
        <Container isFlex>
            <Week />
            <div className="snacks">
                Полдник
                <Switch
                    value={!main.isSnacksDisabled}
                    onChange={() =>
main.setSnacks(!main.isSnacksDisabled)}
                    color={Color.green}
                />
            </div>
        </Container>
        {!main.pendingState.ingredients && <FoodMarket />}
    </React.Fragment>;
});
export default Main;
```

Листинг Б.5 – Страница Profile

```
import React from 'react';
import {observer} from "mobx-react-lite";
import cn from "classnames";
import {Context} from "../../index";
import Container from
"../../components/common/Container/Container";
import Button, { Color } from
"../../components/common/buttons/Button";
import UserData from "./components/UserData";
import './style.css';

const Profile = observer(() => {
    const {main} = React.useContext(Context);
    const inputFileRef = React.useRef();
    return <Container>
        <input ref={inputFileRef} style={{ display: `none` }}
onChange={() => {}} type="file"/>
        <div className="profile__container">
```

```
            {renderLeftPart()}
            <UserData />
        </div>
    </Container>
});
export default Profile;
```

Листинг Б.6 – Страница Auth

```
import React from 'react';
import {useHistory} from "react-router-dom";
import {observer} from "mobx-react-lite";
import cn from "classnames";
import {MAIN_ROUTE} from "../../resources/consts";
import {Context} from "../../index";
import Container from
"../../components/common/Container/Container";
import Button, { Color } from
"../../components/common/buttons/Button";
import Input, {InputIcon, InputType} from
"../../components/common/Input";
import Wizard from "../../components/Wizard/Wizard";
import UserModel from "../../models/UserModel";
import StepsResource from "./resources/RegistrationStepsResource";
import { ReactComponent as UserIcon } from
"../../icons/common/user.m.svg";
import { ReactComponent as LockIcon } from
"../../icons/common/lock.m.svg";
import localStorageHelper from "../../helpers/localStorageHelper";
import JsonParser from "../../helpers/jsonParser";
import '../style.m.css';

const Auth = observer(() => {
    const {main} = React.useContext(Context);
    const history = useHistory();
    const [login, setLogin] = React.useState(``);
    const [password, setPassword] = React.useState(``);
    const [registration, setRegistration] = React.useState({
...UserModel });
    const [currentStep, setCurrentStep] = React.useState(null);
    const [launch, setLaunch] = React.useState(null);
    React.useEffect(() => {
        const localProgress =
JsonParser.parse(localStorageHelper.getLocalRegistrationProgress()
);
        if (localProgress) {
            setLaunch(localProgress.launch);
            setCurrentStep(localProgress.currentStep);
            setRegistration({ ...registration, ...localProgress
});
        }
    }, []);
    React.useEffect(() => {
```

```
        main.setValidationError(`auth`, ``);
    }, [login, password, registration]);
    const onLoginClick = async () => {
        localStorageHelper.deleteLocalRegistrationProgress();
        const result = await main.login(login, password);
        if (result) {
            history.push(MAIN_ROUTE);
        }
    };
    const onRegistrationClick = async () => {
        const result = await main.registrate(registration);
        if (result) {
            localStorageHelper.deleteLocalRegistrationProgress();
            history.push(MAIN_ROUTE);
        }
    };
    const updateRegistration = (updatedData, clear = false) => {
        setRegistration({ ...registration, ...updatedData });
        if (!clear) {
localStorageHelper.setLocalRegistrationProgress(JSON.stringify({
                ...registration,
                ...updatedData,
                currentStep,
                launch
            }));
        }
    };
    const onRegistrationStartClick = () => {
        setLaunch(!launch);
        localStorageHelper.deleteLocalRegistrationProgress();
    };
        if (main.pendingState.ingredients) {
        return null;
    }
    return <Container>
        <Wizard
            data={registration}
            model={UserModel}
            stepsResource={StepsResource}
            updateData={updateRegistration}
            onComplete={onRegistrationClick}
            setCurrentStep={setCurrentStep}
            currentStep={currentStep}
            launch={launch}
        />
    </Container>
});
export default Auth;
```

Приложение В
ИСХОДНЫЙ КОД СЕРВЕРНОЙ ЧАСТИ
(рекомендуемое)

Листинг В.1 – Контроллер Auth Service

```
package auth
import (
    "account-service/entities"
    "account-service/service/passhasher"
    "context"
    "database/sql"
    "errors"
)
type userProvider interface {
    GetByEmailWithPasswordHash(ctx context.Context, email string)
(*entities.User, string, error)
}
type Authenticator struct {
    userProvider         userProvider
    accessTokenGenerator AccessTokenGenerator
    passwordHasher       passhasher.PasswordHasher
}
func NewAuthenticator(userProvider userProvider,
accessTokenGenerator AccessTokenGenerator, passwordHasher
passhasher.PasswordHasher) *Authenticator {
    return &Authenticator{
       userProvider:         userProvider,
       accessTokenGenerator: accessTokenGenerator,
       passwordHasher:       passwordHasher,
    }
}
func (a Authenticator) Authenticate(ctx context.Context, request
AuthorizeRequest) (*Access, error) {
    user, passwordHash, err :=
a.userProvider.GetByEmailWithPasswordHash(ctx,
request.Credentials.Login)
    switch {
    case err == nil:
    case errors.Is(err, sql.ErrNoRows):
       return nil, InvalidCredentialsError
    default:
       return nil, err
    }
    if
!a.passwordHasher.VerifyPassword(request.Credentials.Password,
passwordHash) {
       return nil, InvalidCredentialsError
    }
    accessToken, err := a.accessTokenGenerator.Generate(user)
    if err != nil {
       return nil, err
    }
```

```
    return &Access{
       AccessToken: accessToken,
    }, nil
}
```

Листинг В.2 – Контроллер Registration Service

```
package registration

import (

     "account-service/entities"

     "account-service/service/passhasher"

     "context"

     "github.com/google/uuid"

)

type userProvider interface {

     Create(ctx context.Context, user entities.User, passwordHash
string) (uuid.UUID, error)

     ExistsEmailOrUsername(ctx context.Context, username string,
email string) (bool, error)

     GetByEmailWithPasswordHash(ctx context.Context, email string)
(*entities.User, string, error)

}

type Registrar struct {

     userProvider   userProvider

     passwordHasher passhasher.PasswordHasher

}

func NewRegistrar(userProvider userProvider, passwordHasher
passhasher.PasswordHasher) *Registrar {

     return &Registrar{

          userProvider:   userProvider,

          passwordHasher: passwordHasher,

     }

}

func (r Registrar) CreateUser(ctx context.Context, user
entities.User, password string) (uuid.UUID, error) {

     isUserExists, err := r.isUserExistEmailOrUsername(ctx,
user.Username, user.Email)
```

```
	if err != nil {
		return uuid.Nil, err
	}
	if isUserExists {
		return uuid.Nil, UserIsAlreadyRegisteredError
	}
	userId, err := r.userProvider.Create(ctx, user,
r.passwordHasher.HashPassword(password))
	if err != nil {
		return uuid.Nil, err
	}
	return userId, nil
}
func (r Registrar) isUserExistEmailOrUsername(ctx context.Context,
username string, email string) (bool, error) {
	return r.userProvider.ExistsEmailOrUsername(ctx, username,
email)
}
```

Листинг В.3 – Контроллер User Service

```
package user
import (
	"account-service/entities"
	"context"
	"github.com/google/uuid"
)
type userProvider interface {
	UpdatePersonInfo(ctx context.Context, userId uuid.UUID,
personInfo entities.PersonInfo) error
	GetByIdWithPersonInfo(ctx context.Context, id uuid.UUID)
(*entities.User, error)
}
type Service struct {
	userProvider userProvider
```

```
}

func NewService(userProvider userProvider) *Service {

     return &Service{userProvider: userProvider}

}

func (s Service) UpdatePersonInfo(ctx context.Context, userId
uuid.UUID, personInfo entities.PersonInfo) error {

     return s.userProvider.UpdatePersonInfo(ctx, userId,
personInfo)

}

func (s Service) GetWithPersonInfo(ctx context.Context, id
uuid.UUID) (*entities.User, error) {

     return s.userProvider.GetByIdWithPersonInfo(ctx, id)

}
```

Листинг В.7 – Пакет Registration handler

```
package authhandler
import (
    "account-service/entities"
    "account-service/pkg/plhttp/plrouter"
    "account-service/pkg/pllogger"
    "account-service/service/registration"
    "errors"
    "github.com/google/uuid"
    "net/http"
    "regexp"
)
var (
type personInfo struct {
    Name                  string      `json:"name"
example:"Павел"`
    Age                   int         `json:"age" example:"35"`
    Gender                int         `json:"gender" example:"1"`
    Weight                float32     `json:"weight" example:"93"`
    Height                float32     `json:"height"
example:"184"`
    Profession            string      `json:"profession"
example:"Пекарь"`
    ForPersonal           bool        `json:"forPersonal"
example:"false"`
    NecessaryOnly         bool        `json:"necessaryOnly"
example:"false"`
    IsEnthusiast          bool        `json:"isEnthusiast"
example:"false"`
```

```
    Subscription          bool        `json:"subscription"
example:"false"`
    IsDigestive           bool        `json:"isDigestive"
example:"false"`
    IsAllergic            bool        `json:"isAllergic"
example:"false"`
    FavouriteIngredients  []uuid.UUID
`json:"favouriteIngredients"`
    UnfavouredIngredients []uuid.UUID
`json:"unfavouredIngredients"`
    BlacklistIngredients  []uuid.UUID
`json:"blacklistIngredients"`
}
type registrationRequest struct {
    Username   string     `json:"username" binding:"required"
example:"topaz321"`
    Email      string     `json:"email" binding:"required"
example:"example@server.ru"`
    Password   string     `json:"password" binding:"required"
example:"testPassword"`
    PersonInfo personInfo `json:"personInfo"`
}
func (r registrationRequest) Validate() bool {
    if !_emailRegex.Match([]byte(r.Email)) {
       return false
    }
    if len(r.Username) == 0 {
       return false
    }
    if len(r.Password) == 0 {
       return false
    }
    return true
}

// Registration godoc
// @Summary     Регистрация пользователя
// @Param       request body registrationRequest true
"Регистрационные данные пользователя"
// @Router      /account-service/registration [post]
func Registration(l pllogger.Logger, registrar
*registration.Registrar) plrouter.HandlerFunc {
    return func(c *plrouter.RouteContext) {
       var req registrationRequest
       if err := c.ShouldBindJSON(&req); err != nil {
          l.Error(err.Error())
          c.ErrorResponse(http.StatusBadRequest, "bad body")
          return
       }
```

```
        if !req.Validate() {
            c.ErrorResponse(http.StatusBadRequest, "validate error")
            return
        }
        }
        _, err := registrar.CreateUser(c.AuthContext, user,
req.Password)
        switch {
        case err == nil:
        case errors.Is(err,
registration.UserIsAlreadyRegisteredError):
            c.ErrorResponse(http.StatusUnauthorized, "Пользователь с
таким email уже существует")
            return
        default:
            l.Error(err.Error())
            c.ErrorResponse(http.StatusInternalServerError,
"internal server error")
            return
        }
        c.OkResponse()
        return
    }
}
func ingredientIdToEntity(ingredients []uuid.UUID)
[]entities.Ingredient {
    result := make([]entities.Ingredient, 0, len(ingredients))
    for _, id := range ingredients {
        result = append(result, entities.Ingredient{
            Id: id,
        })
    }
    return result
}
```

Листинг В.8 – Пакет Authorization handler

```
package authhandler
import (
    "account-service/pkg/plhttp/plrouter"
    "account-service/pkg/pllogger"
    "account-service/service/auth"
    "errors"
    "net/http"
)
type authorizationRequest struct {
    Login    string `json:"login" binding:"required"
example:"example@server.ru"`
    Password string `json:"password" binding:"required"
```

```
example:"testPassword"`
}
type authorizationResponse struct {
    Token string `json:"token"`
}


// Authorization godoc
// @Summary     Авторизация пользователя
// @Param       request body authorizationRequest true "Данные для
авторизации пользователя"
// @Success     200 {object} authorizationResponse
// @Router      /account-service/authorization [post]
func Authorization(l pllogger.Logger, authenticator
*auth.Authenticator) plrouter.HandlerFunc {
    return func(c *plrouter.RouteContext) {
       var req authorizationRequest
       if err := c.ShouldBindJSON(&req); err != nil {
          l.Error(err.Error())
          c.ErrorResponse(http.StatusBadRequest, "bad body")
          return
       }
       authorizeRequest := auth.AuthorizeRequest{
          Credentials: auth.Credentials{
             Login:    req.Login,
             Password: req.Password,
          },
       }
       access, err := authenticator.Authenticate(c.AuthContext,
authorizeRequest)
       switch {
       case err == nil:
       case errors.Is(err, auth.InvalidCredentialsError):
          c.ErrorResponse(http.StatusUnauthorized, "Неверный логин
или пароль")
          return
       default:
          l.Error(err.Error())
          c.ErrorResponse(http.StatusInternalServerError,
"internal server error")
          return
       }
       var resp = authorizationResponse{
          Token: access.AccessToken.Token,
       }
       c.OkResponseWithResult(resp)
       return
    }
}
```

Приложение Г

ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

(рекомендуемое)

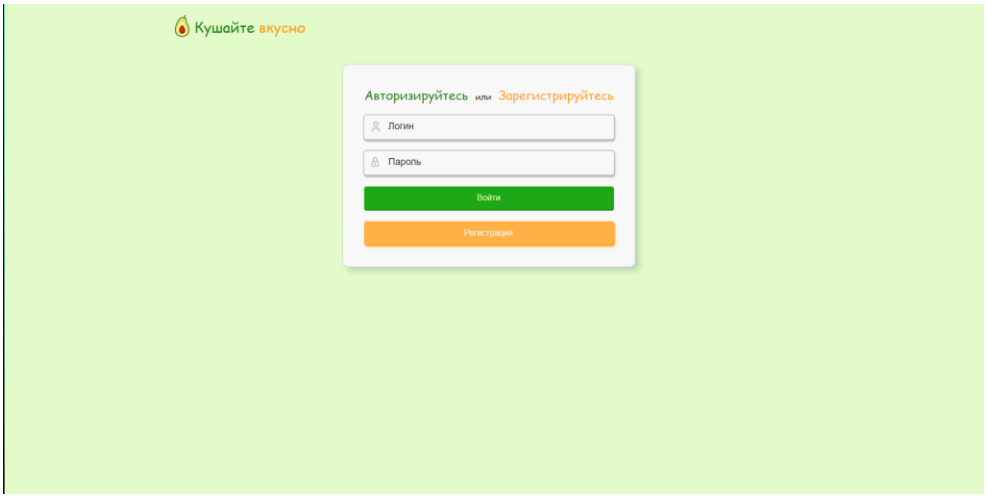


Рисунок Г.1 – Отображение страницы авторизации

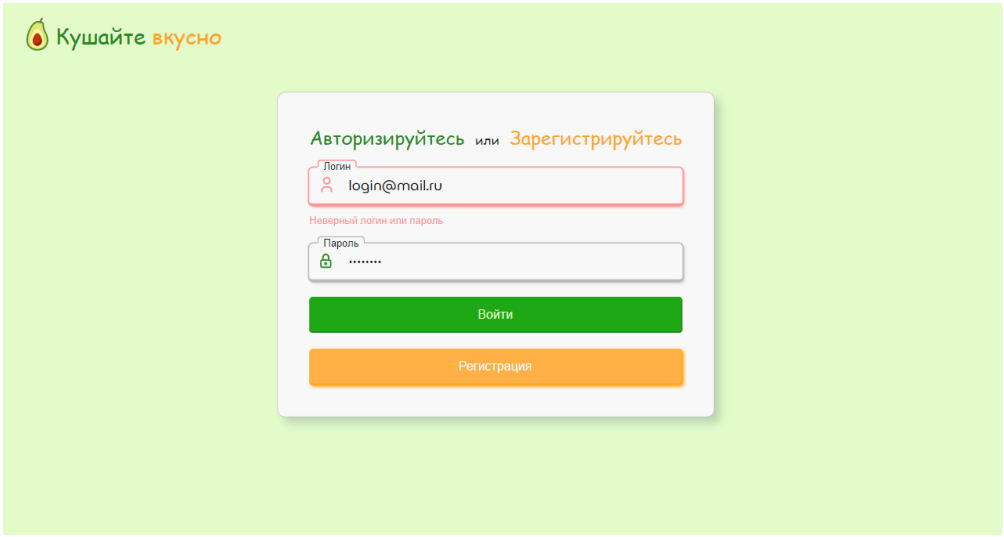


Рисунок Г.2 – Отображение ошибки ввода данных

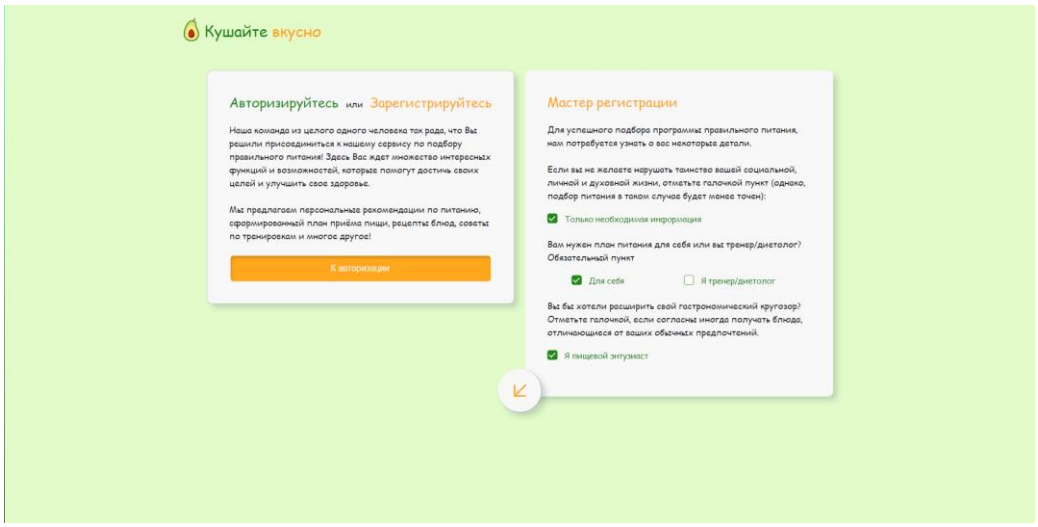


Рисунок Г.3 – Отображение мастера регистрации

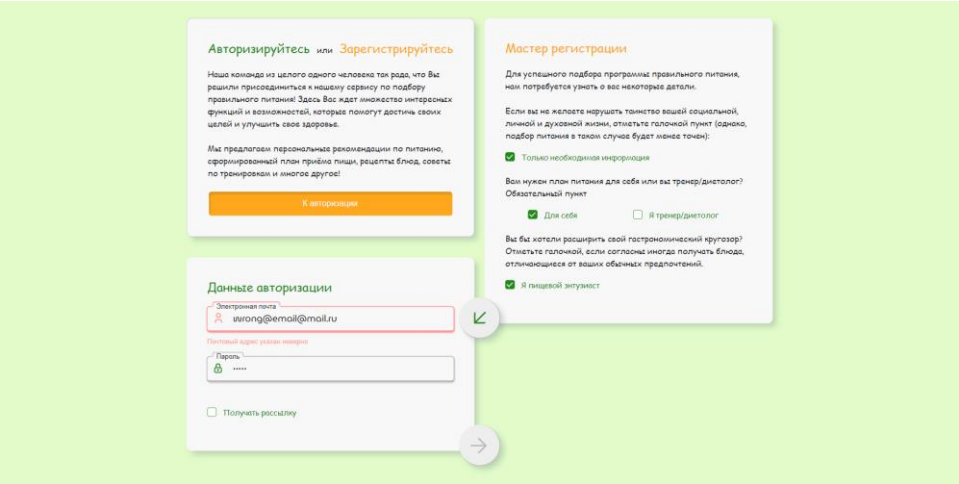


Рисунок Г.4 – Отображение ошибки ввода поля мастера регистрации

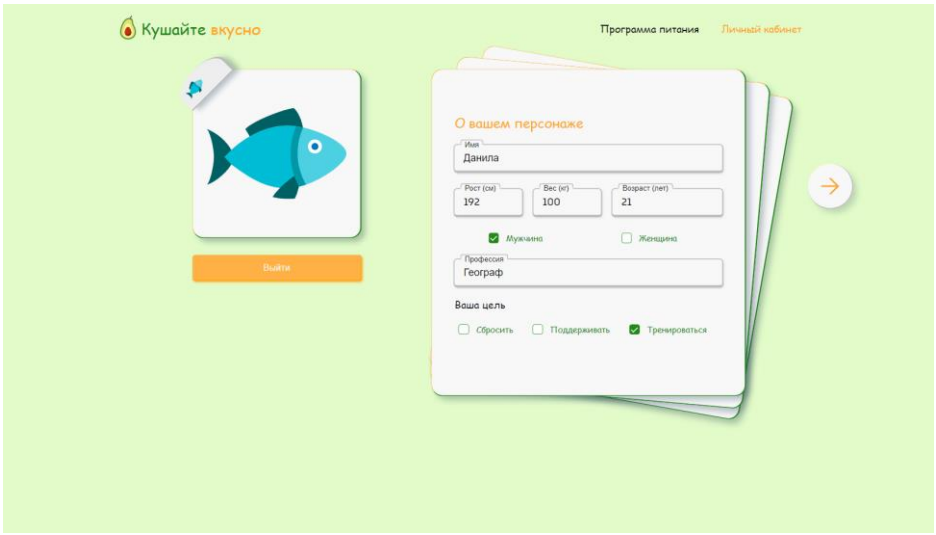


Рисунок Г.5 – Отображение карточки характеристик в профиле

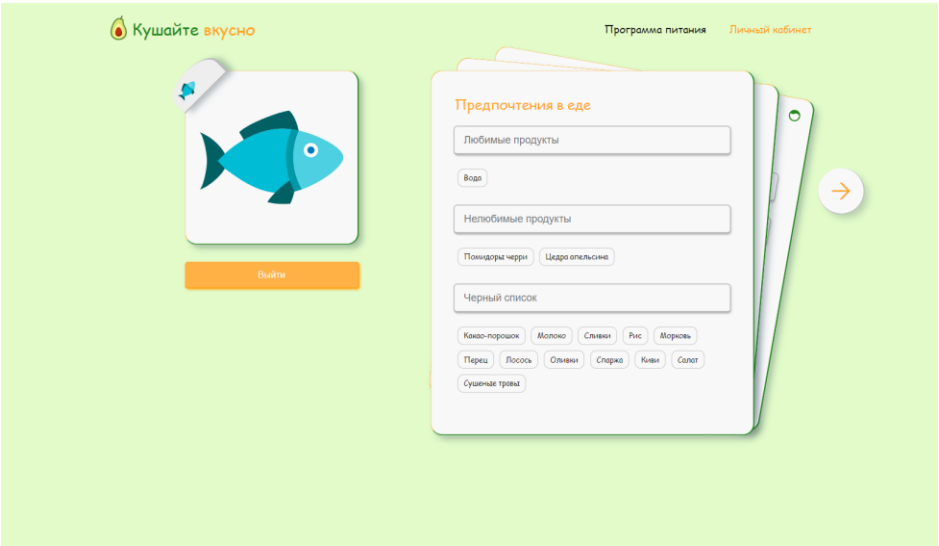


Рисунок Г.6 – Отображение карточки предпочтений в профиле

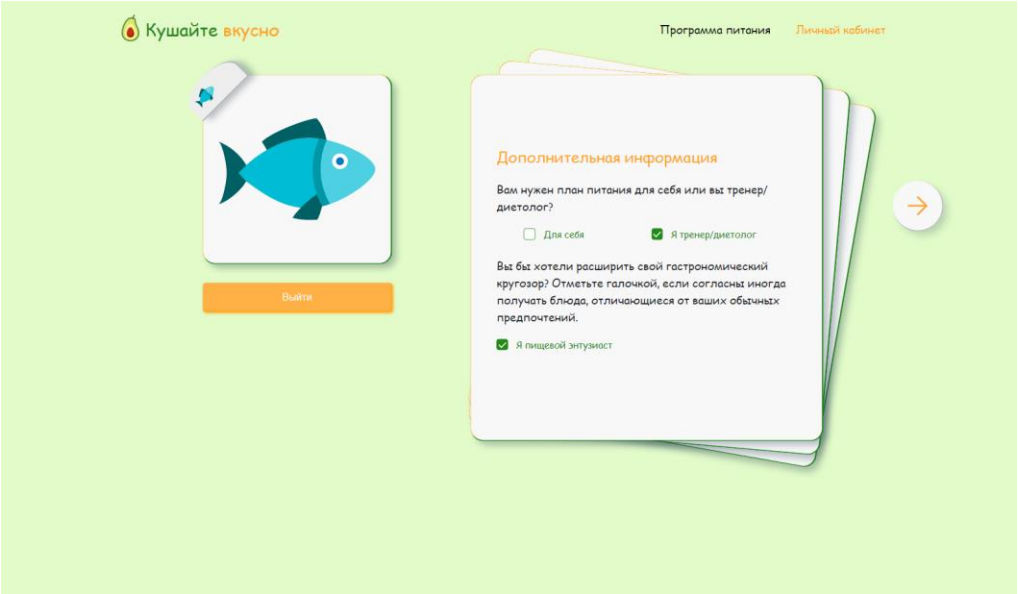
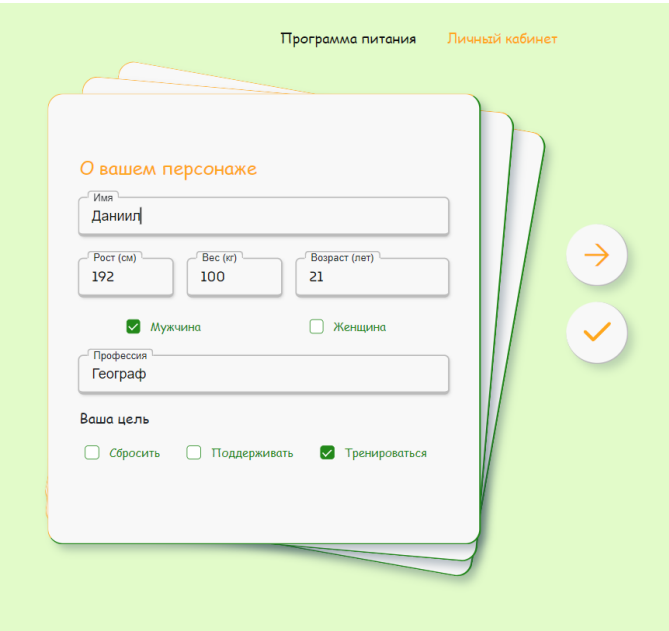
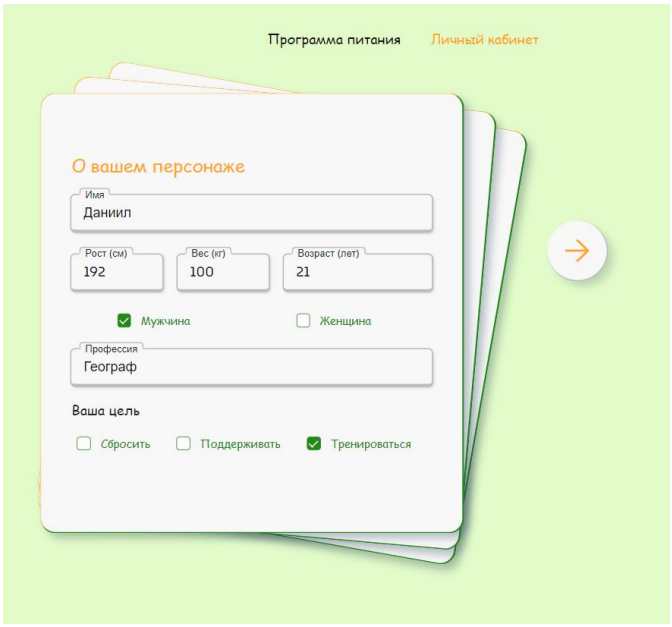


Рисунок Г.7 – Отображение карточки дополнительной информации в профиле



а) Ввод новых данных



б) Сохранение новых данных

Рисунок Г.8 – Ввод новых пользовательских данных

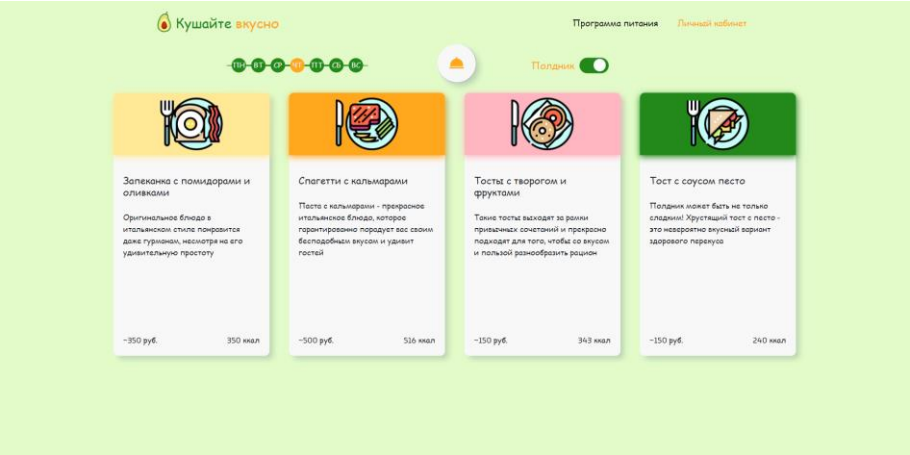


Рисунок Г.9 – Сформированный план питания

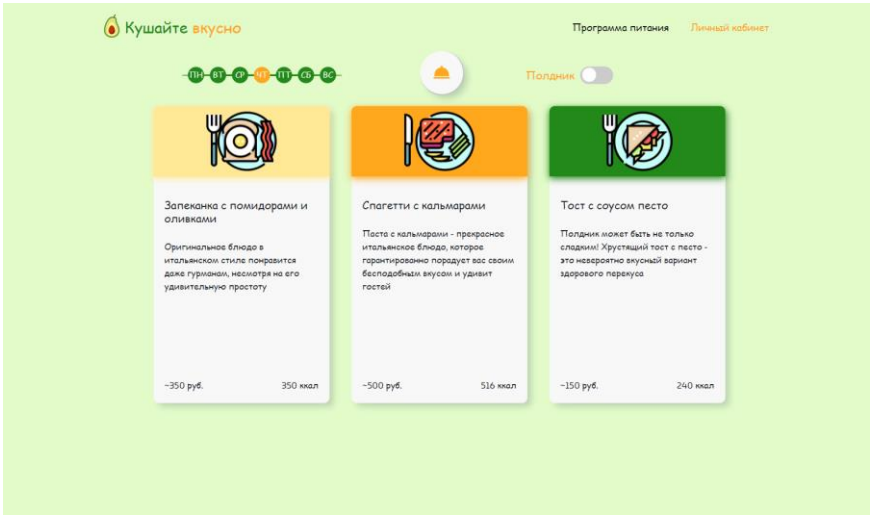


Рисунок Г.10 – План питания без полдника

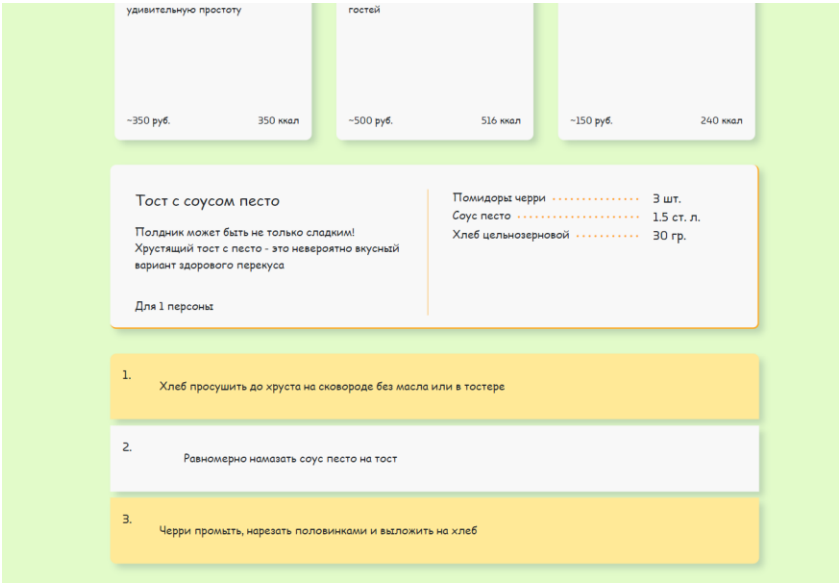
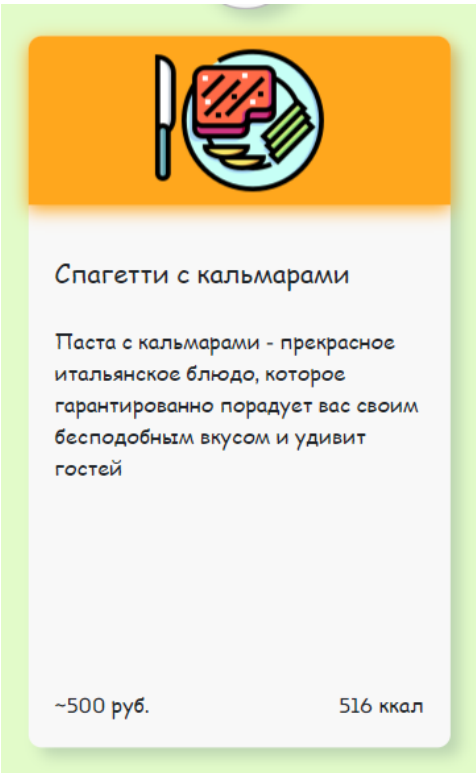
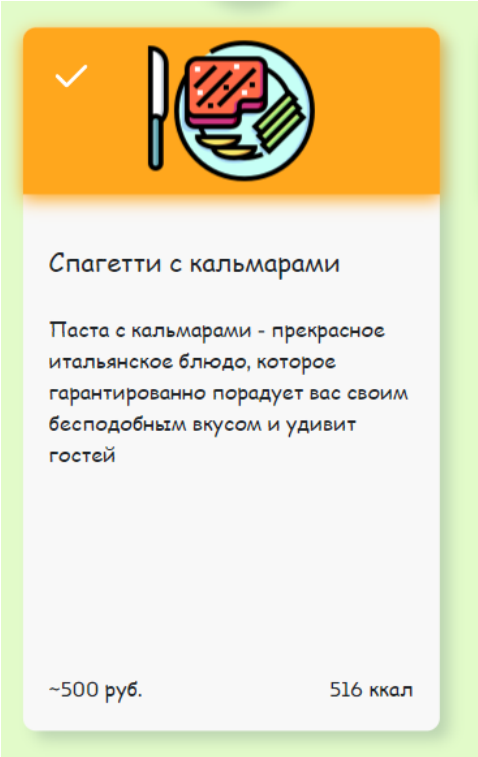


Рисунок Г.11 – Просмотр рецепта блюда

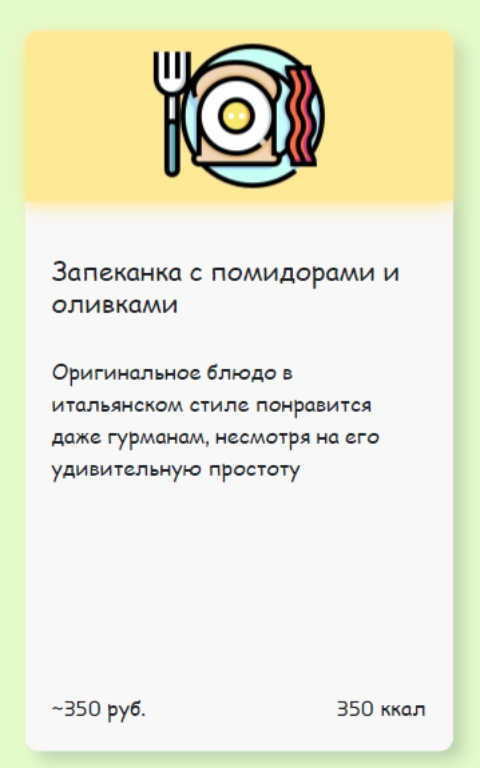


а) Непройденное блюдо

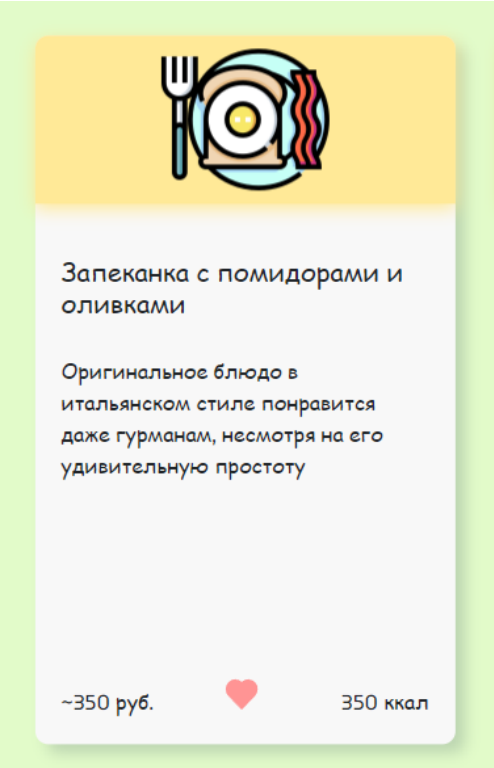


б) Пройденное блюдо

Рисунок Г.12 – Пометка блюда пройденным



а) Блюдо без пометки



б) Любимое блюдо

Рисунок Г.13 – Пометка любимого блюда

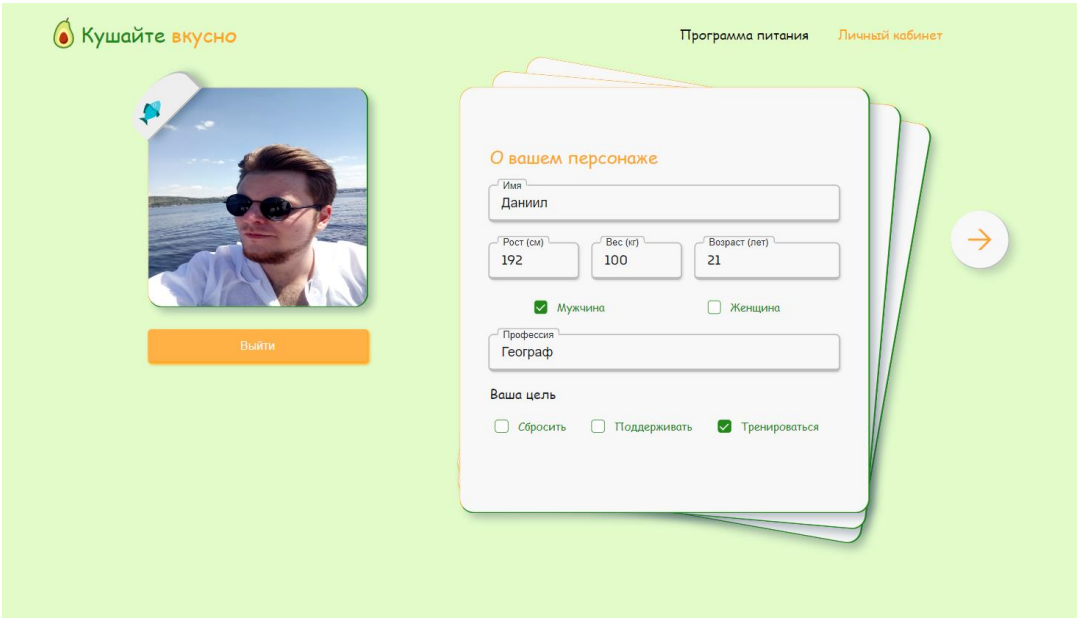


Рисунок Г.14 – Загруженный аватар профиля