

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждения образования «БЕЛОРУССКИЙ
ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий
Кафедра Информационных систем и технологий
Специальность 1-40 05 01 Информационные системы и технологии
Специализация 1-40 05 01-03 Информационные системы и технологии (издательско-полиграфический комплекс)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту на тему:**

Веб-приложение для формирования списка покупок на основе рецептов

Дипломник Гончаревич Евгений Витальевич
(Ф.И.О.)

Руководитель проекта асс. Д. В. Сазонова
(учен. степень, звание, должность, подпись, Ф.И.О.)

И. о. заведующего кафедрой ст. преп. Е. А. Блинова
(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультант ст. преп. А. С. Соболевский
(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер ст. преп. О. А. Нистюк
(учен. степень, звание, должность, подпись, Ф.И.О.)

Дипломный проект защищен с оценкой _____

Председатель ГЭК к.т.н., доц. В. К. Дюбков
(учен. степень, звание, должность, подпись, Ф.И.О.)

ЗАДАНИЕ не вставлять.

Реферат

Пояснительная записка дипломного проекта содержит 61 страницу пояснительной записки, 20 таблиц, 8 листингов, 15 формул, 45 иллюстраций, 7 источников литературы, 10 приложений.

ВЕБ-ПРИЛОЖЕНИЕ, BROWSER, JAVASCRIPT, TYPESCRIPT, REACT, BUN, DOCKER, ELYSIA, PRISMA ORM

Целью дипломного проекта являлось веб-приложение для формирования списка покупок на основе рецептов. Пояснительная записка состоит из введения, шести разделов, заключения и источников литературы.

В первой главе проводится аналитический обзор аналогичных решений и постановка целей и задач проекта.

Вторая глава посвящена проектированию системы и содержит структурную схему базы данных, диаграмму вариантов использования, а также схему алгоритмов, задействованных в дипломном проекте.

В третьей главе описаны процесс разработки, принципы функционирования и назначение созданных компонентов проекта.

В четвертой главе описано тестирования разработанного приложения.

В пятой главе приведено руководство пользователя, позволяющее понять моменты развертывания частей проекта.

В шестой главе производится подробный расчет экономических параметров, срока окупаемости приложения.

Графическая часть включает в себя 4 диаграммы и 2 плаката, указанных в приложениях записки, суммарно имея 6 графических материалов.

				БГТУ 00.00.ПЗ			
	ФИО	Подпись	Дата				
Разраб.	Гончаревич Е. В.			Реферат	Лит.	Лист	Листов
Пров.	Сазонова Д. В.				У	1	1
					74218008, 2024		
Н. контр.	Нистюк О. А.						
Утв.	Блинова Е.А.						

Abstract

The explanatory note of the diploma project contains 61 pages of an explanatory note, 20 tables, 8 listings, 15 formulas, 45 illustrations, 7 sources of literature, 10 appendices.

WEB-APPLICATION, BROWSER, JAVASCRIPT, TYPESCRIPT, REACT, BUN, DOCKER, ELYSIA, PRISMA ORM

The purpose of the diploma project was a web application for the formation of a shopping list based on recipes. The explanatory note consists of an introduction, six sections, conclusion and sources of literature.

The first chapter conducts an analytical review of similar solutions and sets the goals and objectives of the project.

The second chapter is devoted to the design of the system and contains a structural diagram of the database, a diagram of the use cases, and a diagram of the algorithms involved in the project.

The third chapter describes the development process, principles of functioning and purpose of the created project components.

Chapter four describes the testing of the developed application.

The fifth chapter provides a full user's guide to understand the moments of deployment of the project parts.

The sixth chapter provides a detailed calculation of economic parameters, pay-back period of the application.

The graphic part includes 4 diagrams and 2 posters specified in the appendices of the note, having a total of 6 graphic materials.

				БГТУ 00.00.ПЗ			
	ФИО	Подпись	Дата				
Разраб.	Гончаревич Е. В.			Abstract	Лит.	Лист	Листов
Пров.	Сазонова Д. В.				У	1	1
					74218008, 2024		
Н. контр.	Нистюк О.А.						
УТВ.	Блинова Е. А.						

Содержание

Введение	7
1 Постановка задачи и обзор аналогов	8
1.1 Актуальность исследуемой области.....	8
1.2 Обзор аналогов	8
1.3 Приложение «Bring!».....	9
1.4 Веб-приложение «Listonic»	10
1.5 Веб-приложение «Russian food»	12
1.6 Постановка задачи.....	14
1.7 Выводы по разделу.....	14
2 Проектирование приложения	15
2.1 Архитектура приложения	15
2.2 Проектирование базы данных.....	16
2.3 Описание таблиц	17
2.4 Описание перечислений	20
2.5 Диаграмма вариантов использования	21
2.6 Проектирование основных алгоритмов	23
2.7 Выбор средств реализации	23
2.8 Серверная часть	24
2.9 Клиентская часть	25
2.10 Централизованная база данных	25
2.11 Выводы по разделу.....	25
3 Разработка приложения.....	26
3.1 Реализация серверной части.....	26
3.2 Разработка клиентской части	30
3.3 Контейнеризация	33
3.4 Выводы по разделу.....	36
4 Тестирование приложения	38
4.1 Выводы по разделу.....	44
5 Руководство пользователя	45
5.1 Руководство по развертыванию приложения.....	45
5.2 Руководство для не авторизованного пользователя.....	46
5.3 Руководство для авторизованного пользователя	46
5.4 Руководство для администратора/модератора	49
5.5 Выводы по разделу.....	50
6 Техничко-экономическое обоснование проекта.....	51
6.1 Общая характеристика разрабатываемого программного средства	51

				БГТУ 00.00.ПЗ						
	ФИО	Подпись	Дата							
Разраб.	Гончаревич Е. В.			Содержание			Лит.	Лист	Листов	
Пров.	Сазонова Д. В.						У	1	2	
Н. контр.	Нистюк О.А.						74218008, 2024			
Утв.	Блинова Е. А.									

6.2 Исходные данные для проведения расчетов	52
6.3 Методика обоснования цены	52
6.3.1 Объем программного средства	53
6.3.2 Основная заработная плата	53
6.3.3 Дополнительная заработная плата	54
6.3.4 Отчисления в Фонд социальной защиты населения	54
6.3.5 Расходы на материалы.....	55
6.3.6 Прочие прямые затраты.....	55
6.3.7 Накладные расходы	55
6.3.8 Сумма расходов на разработку программного средства	56
6.3.9 Расходы на сопровождение и адаптацию	56
6.3.10 Полная себестоимость	56
6.3.11 Определение цены, оценка эффективности	57
6.4 Выводы по разделу	59
Заключение	60
Список использованных источников	61
ПРИЛОЖЕНИЕ А Диаграмма развертывания	62
ПРИЛОЖЕНИЕ Б Логическая схема базы данных.....	63
ПРИЛОЖЕНИЕ В Диаграмма вариантов использования	64
ПРИЛОЖЕНИЕ Г Блок-схема алгоритма добавления продукта в хранилище.....	65
ПРИЛОЖЕНИЕ Д Блок-схема алгоритма формирования списка покупок.....	66
ПРИЛОЖЕНИЕ Е Листинг моделей базы данных.....	67
ПРИЛОЖЕНИЕ Ж Листинг маршрутизатора	73
ПРИЛОЖЕНИЕ И Листинг сервера раздачи картинок	86
ПРИЛОЖЕНИЕ К Листинг списка зависимостей.....	88
ПРИЛОЖЕНИЕ Л Таблица экономических показателей.....	90

Введение

Схемы автоматизации и использования информационных систем могут быть успешно применены к составлению и управлению списками покупок. В современном мире, когда ритм жизни ускоряется, а выбор товаров в магазинах огромен, эффективное управление списком покупок становится важным аспектом организации покупок. Пользователи могут создавать списки покупок в специальных приложениях или веб-сервисах, где они могут добавлять необходимые продукты, следить за количеством и планировать покупки на основе своих потребностей.

Информационные системы могут предоставлять подробную информацию о продуктах, включая цены, сроки годности, пищевую ценность и другие характеристики. Это помогает покупателям принимать более информированные решения. Системы могут предоставлять рекомендации по покупкам, исходя из предыдущих покупок, предпочтений и даже диетических ограничений.

Более технологичные системы могут автоматически обновлять список покупок на основе имеющихся запасов дома, учитывая, что нужно пополнить. Информационные системы могут оптимизировать маршрут по магазину, чтобы сэкономить время и энергию покупателей. Могут предоставлять возможность сканировать товары и оплачивать их через мобильное приложение, что упрощает процесс покупок.

Основной аудиторией данного приложения будут семьи, активные люди, студенты и все, кто ценит свое время и стремится к оптимизации процесса покупок и приготовления пищи. С помощью приложения они смогут легко составлять списки покупок на основе рецептов, управлять своими запасами и экономить как время, так и энергию, необходимые для планирования и совершения покупок.

Целью данного дипломного проекта является разработка приложения для просмотра списка имеющихся продуктов, просмотра идей для рецептов, а также составление списка покупок на основании составов рецептов.

Задачи дипломного проекта:

- изучить аналоги разрабатываемого приложения на рынке;
- изучить и выбрать подходящие технологии для реализации проекта,
- спроектировать структуру базы данных,
- разработать серверную часть;
- разработать клиентскую часть;
- протестировать разработанное приложение;
- разработать руководство пользователя по разворачиванию приложения;
- выполнить экономическое обоснование проекта.

				БГТУ 00.00.ПЗ						
	ФИО	Подпись	Дата							
Разраб.	Гончаревич Е. В.			Введение			Лит.	Лист	Листов	
Пров.	Сазонова Д. В.						У	1	1	
Н. контр.	Нистюк О. А.						74218008, 2024			
Утв.	Блинова Е. А.									

1 Постановка задачи и обзор аналогов

1.1 Актуальность исследуемой области

В современном мире, где ритм жизни постоянно нарастает, эффективное планирование питания и управление покупками становятся все более актуальными задачами. Существует постоянная потребность в инструментах, способных помочь людям сократить время на походы по магазинам и упростить процесс составления списка покупок. Приложения, основанные на рецептах, представляют собой потенциальное решение этой проблемы, позволяя пользователям создавать списки покупок на основе ингредиентов, необходимых для приготовления конкретных блюд.

С разнообразием продуктов и услуг на рынке, выбор подходящих продуктов для покупки становится все более сложным. Приложения, предлагающие функциональность создания списков покупок на основе рецептов, могут значительно облегчить этот процесс, предоставляя пользователям точные списки ингредиентов, необходимых для приготовления конкретных блюд. Такие приложения также могут предложить альтернативные продукты или варианты блюд, учитывая диетические предпочтения или аллергии, что делает их еще более ценными в современном мире.

С развитием технологий и повышением доступности смартфонов и интернета, приложения для управления покупками становятся неотъемлемой частью повседневной жизни. Приложения, основанные на рецептах, представляют собой логичное дополнение к этому тренду, объединяя функциональность планирования питания с возможностью удобного составления списков покупок.

Исследование и разработка таких приложений имеет большое значение в контексте современного образа жизни, подчеркивая важность инноваций в области управления ресурсами и оптимизации повседневных процессов.

1.2 Обзор аналогов

Для создания web-приложения с привлекательным пользовательским интерфейсом и удобной функциональностью необходимо проанализировать аналоги. Анализ конкурентов способствует не только выявлению достоинств и недостатков, но также помогает понять, к чему стоит стремиться при разработке собственного web-приложения и выявить основные потребности пользователей.

В результате поиска аналогичных решений с похожей тематикой было выбрано два web-приложения. Анализ будет осуществляться путем оценки приложений по различным критериям, таким как: интерфейс, юзабилити, функциональность, наличие нестандартных подходов реализации и т. д.

				БГТУ 01.00.ПЗ						
	ФИО	Подпись	Дата							
Разраб.	Гончаревич Е. В.			1 Постановка задачи и обзор аналогов	Лит.		Лист		Листов	
Пров.	Сазонова Д. В.				У		1	7		
					74218008, 2024					
Н. контр.	Нистюк О. А.									
Утв.	Блинова Е. А.									

1.3 Приложение «Bring!»

В качестве одного из аналогов было рассмотрено приложение «Bring!» [1].

«Bring» [1] – это web-приложение, которое предназначено конкретно для создания списков покупок, – оно делает подготовку и сам поход в супермаркет максимально простыми и удобными. Не в последнюю очередь это связано с контекстным указателем в поисковой строке, а также большими пиктограммами записей.

Отдельного внимания также заслуживает отдельный раздел приложения с идеями блюд на завтрак, обед и ужин. Они включают как рецепты, так и списки покупок. Разработчики программы не забыли про возможность создания нескольких каталогов, а также совместного доступа к ним.

Главная страница приложения представлена на рисунке 1.1.



Рисунок 1.1 – Главная страница «Bring!»

Программа предоставляет множество действий, например, добавление и удаление продуктов из списка «Дом», который отражает имеющиеся продукты. На рисунке 1.2 изображена вкладка «Идеи». Вкладка предлагает на выбор множество рецептов для просмотра и приготовления.

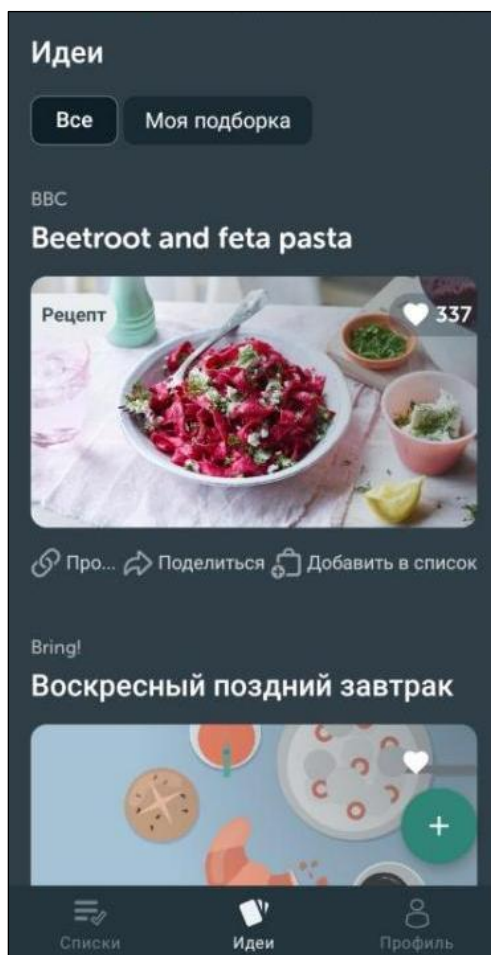


Рисунок 1.2 – Вкладка «Идеи»

Приложение имеет богатый функционал, начиная от выбора рецепта и завершая выбором иконки для продукта. Однако, в приложении отсутствует возможность указывать единицы измерения продуктов (штуки, килограммы и т.д.), что следует учесть при разработке своего web-приложения. Так же отсутствует возможность суммировать цену продуктов.

1.4 Веб-приложение «Listonic»

«Listonic» [2] – web-приложение с минимальным интерфейсом, но богатым функционалом. В отличие от предыдущего аналога, обладает возможностью указания стоимости продуктов (но только в долларах, что ограничивает круг пользователей приложения), добавлять свои продукты в список, указывать им иконку, сортировать список по алфавиту. Также есть возможность подсчитать итоговую стоимость продуктов.

Глядя на интерфейс веб-приложения «Listonic» ощущается философия минимализма. У пользователя есть минимальный набор кнопок и, соответственно действий, что исключает возможность запутаться в них. В связи с тем, что интерфейс не перегружен, пользователь быстро справляется со своей задачей и не проявляет негативных эмоций в сторону приложения. Однако, такое по вкусу не каждому.

Главная страница показана на рисунке 1.3.

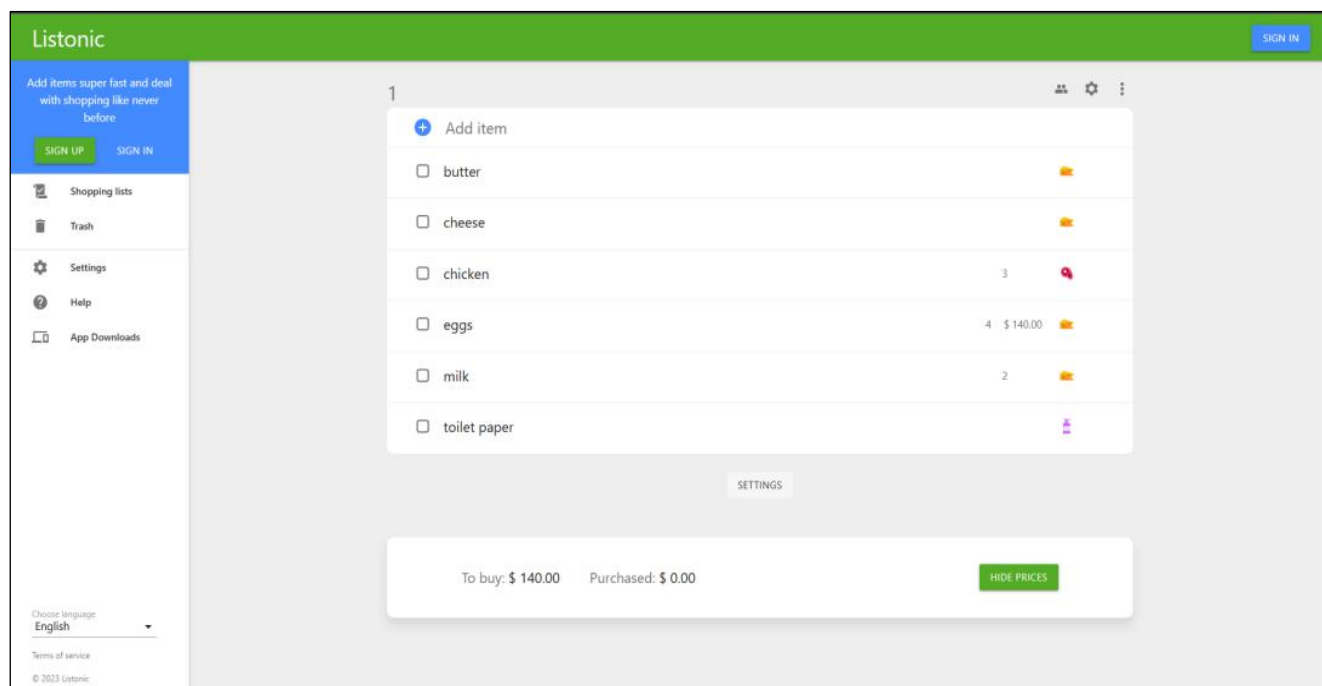


Рисунок 1.3 – Главная страница «Listonic»

Дополнительно в приложении присутствует функция «Корзины». Список после удаления попадает в этот раздел. Корзина изображена на рисунке 1.4.

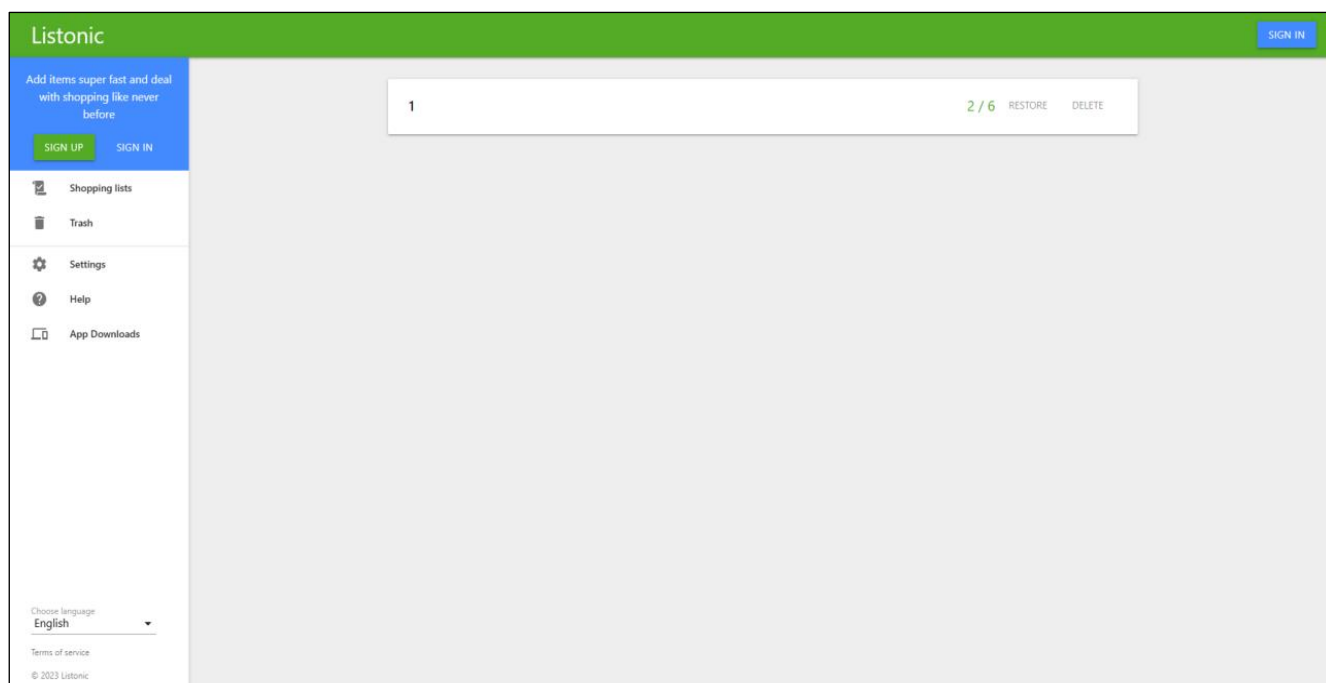


Рисунок 1.4 – Страница корзины приложения «Listonic»

Удаленные списки покупок можно восстановить при необходимости, или же удалить окончательно. Очень удобно, если пользователь передумает удалять список. Из минусов можно отметить, что списки покупок не содержат суммарную стоимость продуктов, из которых они состоят, а та же отсутствует дата подтверждения покупки, что исключает возможность сортировки по датам. Это стоит учесть при создании собственного веб-приложения.

1.5 Веб-приложение «Russian food»

«Russian food» [3] – приложение-склад рецептов. Здесь любой пользователь может посмотреть неисчисляемое количество различных рецептов на любую тематику, будь то вегетарианские блюда, закуски, горячие блюда, супы и т.д. Несмотря на название, рецепты здесь от различных народов и времен. Часть главного экрана продемонстрирована на рисунке 1.5.

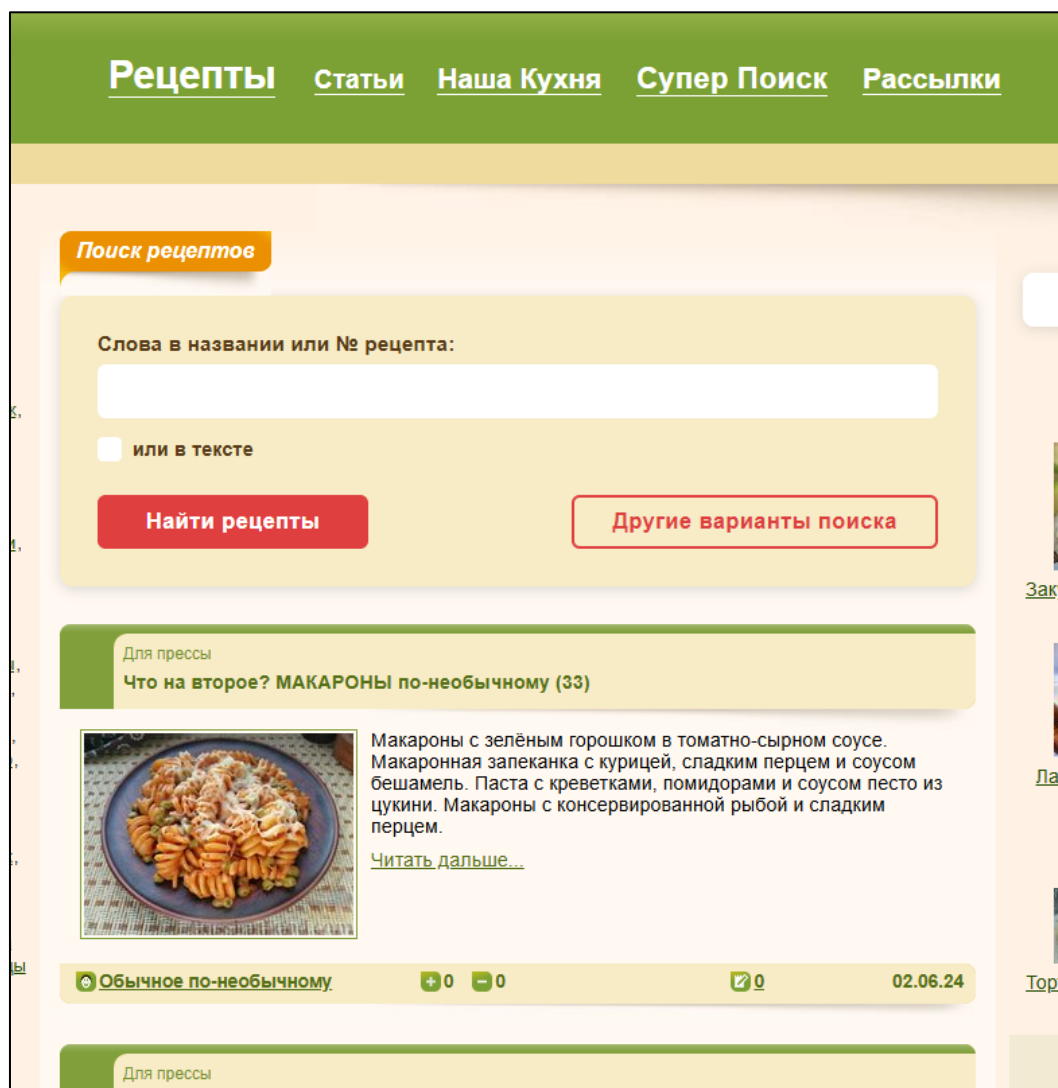


Рисунок 1.5 – Главная страница «Russian food»

На главной странице нас приветствуют сразу несколько областей. Первая – полоса с пунктами меню. В ней выделяется пункт «Супер-поиск», изображенный на рисунке 1.6. В отличие от поиска на главной странице, этот может искать рецепты по большему числу параметров, таких как ингредиенты, типы блюд, национальность и т. д. Это может стать очень удобным критерием для интернационализации приложения. Чем больше народов привлечено в приложение – тем больше различных блюд и способов их приготовления. Так же поддерживается опция выбора вегетарианских блюд, что очень удобно для людей, которые не могут или не хотят употреблять животное мясо. Так же поддерживается исключение продуктов, если пользователь испытывает непереносимость одного или нескольких компонентов.

Рисунок 1.6 – «Супер-поиск»

Однако, самая интересная область приложения – это область добавления собственных рецептов. Здесь авторизованный пользователь волен создавать собственные рецепты. Часть интерфейса создания показана на рисунке 1.7.

Рисунок 1.7 – Создание собственного рецепта

Здесь очень много удобных опций, таких как выбор темы блюда, в которой это блюдо появится при поиске, загрузка фотографий блюда (в т. ч. и для пошагового приготовления) и прочее.

В целом, приложение «Russian food» очень удобное и практичное, но не имеет возможности отслеживания продуктов, имеющих у пользователя.

1.6 Постановка задачи

Целью данного дипломного проекта является разработка web-приложения для составления списка покупок. Для реализации проекта необходимо установить список задач, выполнение которых будет отображать прогресс:

- предоставлять возможность регистрации и авторизации по логину и паролю;
- предоставлять роль администратора, модератора и стандартного пользователя;
- предоставлять возможность пользователю управлять содержимым своего списка продуктов: добавлять, изменять, удалять продукты;
- предоставлять пользователю возможность работы с динамическим чек-листом покупок: автоматическое добавление и удаление выбранных из рецептов продуктов с учетом имеющихся в личном списке продуктов;
- предоставлять возможность пользователю добавлять и удалять продукты в чек-листе;
- предоставлять пользователю возможность подтверждения покупки для автоматического пополнения списка имеющихся продуктов;
- предоставлять возможность администратору управлять базой рецептов и продуктов;
- предоставлять возможность администратору управлять аккаунтами пользователей: добавлять, удалять, восстанавливать.

1.7 Выводы по разделу

Перед тем, как приступить к реализации самого проекта, необходимо определиться с целями, задачами и актуальностью самого проекта. Данные разделы позволяют здраво оценить на какие элементы необходимо обратить внимание при реализации той или иной функциональности, удостовериться в современности и конкурентоспособности среди других представителей данного сегмента.

В ходе анализа существующих аналогов, была выявлена целесообразность разработки и определен основной спектр функциональных возможностей, которые необходимо реализовать при разработке данного программного продукта. Также, на основании использования приложений-аналогов были выработаны основные критерии для построения дизайна приложения, позволяющего осуществлять наиболее простое и интуитивное его использование.

В разделах с постановкой и актуальностью задачи были вынесены основные цели, которые необходимо реализовать в приложении, определены основные роли пользователей, использующих приложение, а также обозначены пункты, подтверждающие актуальность разработки приложения.

2 Проектирование приложения

2.1 Архитектура приложения

Основная задача дипломного проекта – создание клиент-серверного приложения, предоставляющего возможности для формирования списка покупок на основе подобранных пользователем рецептов.

Web-приложение представляется клиентской стороной, делегирующей серверу основное выполнение функционала. Таким образом, делая систему разбитой на логический блоки, тем самым упрощая формирование точек связи. Диаграмма развёртывания приложения представлена на рисунке 2.1 и в Приложении А.

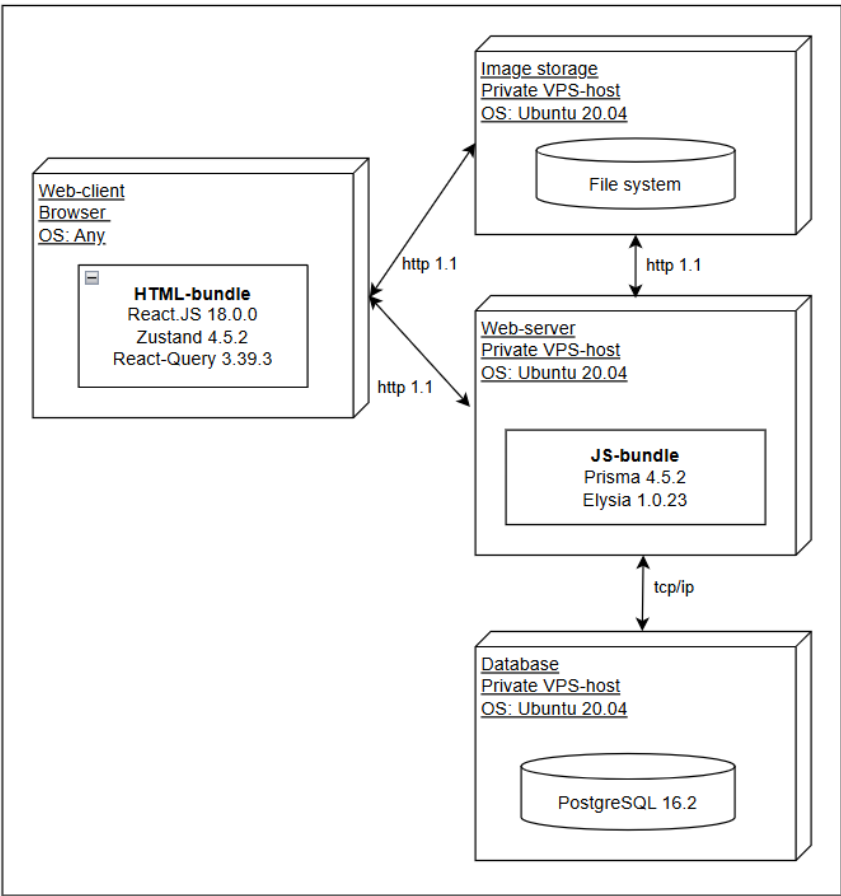


Рисунок 2.1 – Диаграмма развёртывания приложения

Сервером является приложение, расположенное на хостинге для круглосуточного доступа пользователей к данным. Сами данные хранятся в СУБД, с которым связывается сервер для работы с информацией о пользователях. Отдельно хранятся загруженные изображения.

				БГТУ 02.00.ПЗ									
	ФИО	Подпись	Дата										
Разраб.	Гончаревич Е. В.			2 Проектирование приложения				Лит.		Лист	Листов		
Пров.	Сазонова Д. В.										1	11	
								74218008, 2024					
Н. контр.	Нистюк О. А.												
Утв.	Блинова Е. А.												

2.2 Проектирование базы данных

Как отмечалось в предыдущих разделах, одним из ключевых моментов при проектировании и создании базы данных является грамотный анализ предметной области приложения. Как следствие – составление такой модели данных, которая будет правильно отражать то, как с этими данными в общем, и этой моделью, в частности, подразумевается взаимодействовать.

Основой инфраструктуры базы данных является грамотно спроектированная модель, которая отображает связь пользовательских таблиц. Правильное и корректное взаимодействие их друг с другом как раз и заключается в схеме базы данных со связями, верно отображающими их положение.

Перед началом разработки приложения была спроектирована база данных для сервера с 11 таблицами. Логическая схема базы данных, используемой в проекте, приведена ниже на рисунке 2.2.

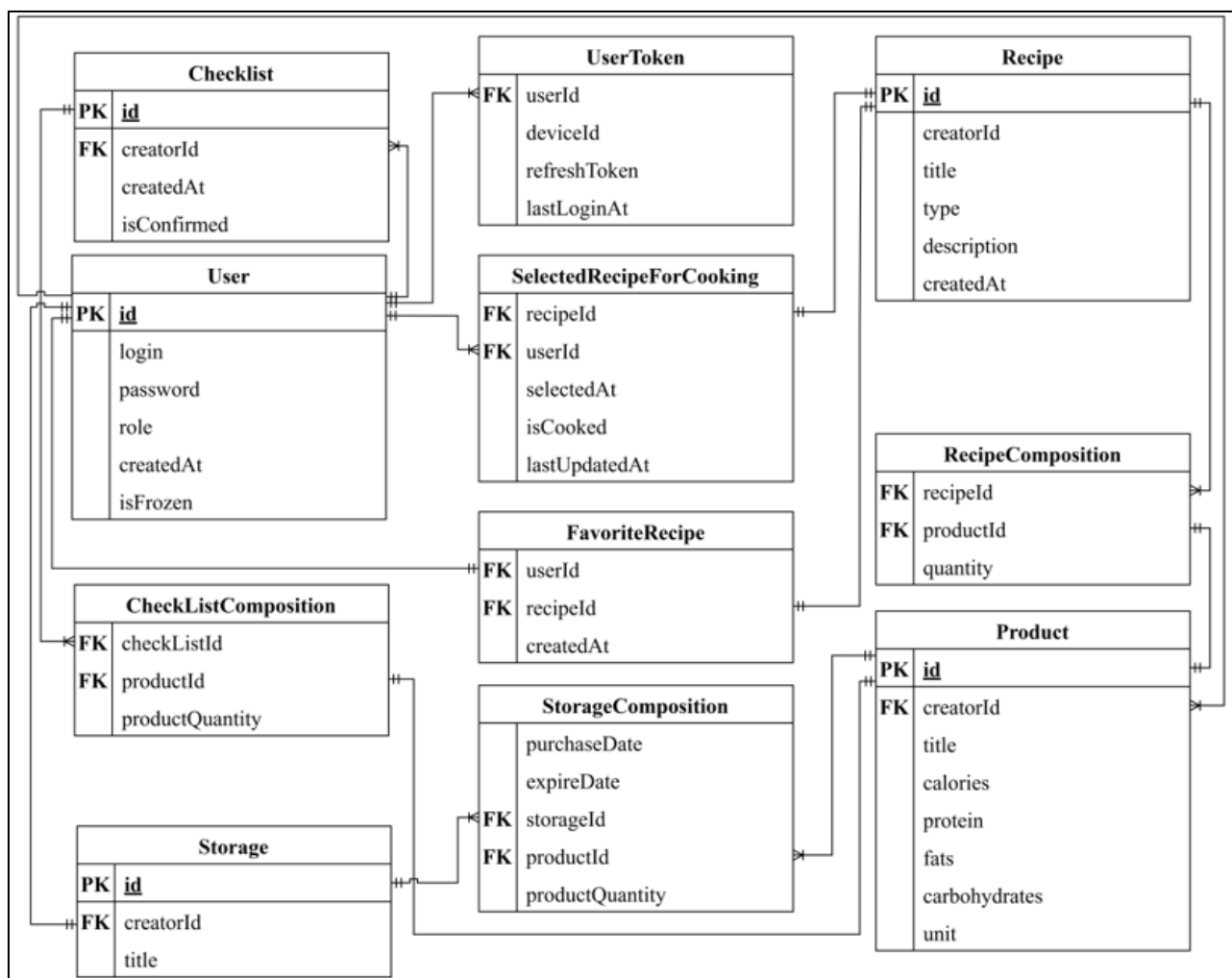


Рисунок 2.2 – Логическая схема базы данных

Обобщенное описание направленности таблиц дает первичное представление о созданной системе связи данных и является мини-презентацией особенностей структуры. Также данная структура приведена в Приложении Б.

Стоит отметить, что проектирование данной БД осуществлялось с расчетом, что в качестве СУБД будет использоваться PostgreSQL, благодаря чему количество

таблиц удалось несколько сократить, однако этого достаточно для полной реализации функционала разрабатываемого веб-приложения.

Структура базы данных и краткое описание представлены в таблице 2.1.

Таблица 2.1 – Описание таблиц

Таблица	Описание
Checklist	Список предстоящих покупок
UserToken	Информация о сессии пользователя
Recipe	Информация о рецепте
User	Информация о пользователе
SelectedRecipeForCooking	Блюда, которые предстоит приготовить
ChecklistComposition	Состав списка покупок
FavoriteRecipe	Избранные пользователем рецепты
Product	Информация о продукте
StorageComposition	Состав пользовательского списка имеющихся продуктов
Storage	Информация о пользовательском списке имеющихся продуктов
RecipeComposition	Состав рецепта

Подробное описание содержимого основных таблиц приведено ниже.

2.3 Описание таблиц

«Ядро» всей схемы базы данных – таблица User. Она содержит всю информацию об аккаунте пользователя, а также несколько дополнительных характеристик. Описание полей в таблице 2.2.

Таблица 2.2 – Описание таблицы «User»

Поле	Описание
login	Логин пользователя
password	Хэш пароля пользователя
role	Роль пользователя в системе (администратор, модератор, стандартный)
createdAt	Дата регистрации пользователя
isFrozen	Показывает удален ли аккаунт
id	Идентификатор пользователя

Таблица Checklist характеризует список предстоящих покупок пользователя. Описание таблицы Checklist находится в таблице 2.3.

Таблица 2.3 – Описание таблицы «Checklist»

Поле	Описание
creatorId	Идентификатор создателя списка
createdAt	Дата создания
isConfirmed	Подтверждена ли покупка
lastUpdatedAt	Дата последнего обновления списка в базе

Таблица ChecklistComposition является продолжением таблицы Checklist. Она характеризует состав списка предстоящих покупок. Описание таблицы ChecklistComposition находится в таблице 2.4.

Таблица 2.4 – Описание таблицы «ChecklistComposition»

Поле	Описание
checklistId	Идентификатор списка покупок
productId	Идентификатор продукта
productQuantity	Количество продукта

Таблица Storage описывает информацию о хранилище пользователя. У одного пользователя может быть только одно хранилище. Хранилище может быть только у обычного пользователя, администрация не может иметь хранилище в своем аккаунте. Описание таблицы Storage находится в таблице 2.5.

Таблица 2.5 – Описание таблицы «Storage»

Поле	Описание
id	Идентификатор хранилища
creatorId	Идентификатор пользователя, к которому привязано хранилище
title	Название хранилище (опционально)

Таблица StorageComposition является продолжение Storage. Она описывает все продукты и их характеристики, имеющиеся на счету у пользователя. Описание таблицы StorageComposition находится в таблице 2.6.

Таблица 2.6 – Описание таблицы «StorageComposition»

Поле	Описание
purchaseDate	Дата приобретения продукта
expireDate	Срок годности продукта (дата до порчи)
storageId	Идентификатор хранилища
productId	Идентификатор продукта
productQuantity	Количество продукта (грамм, штук и т.д.)

Таблица FavoriteRecipe описывает рецепты, которые пользователь посчитал достойными того, чтобы иметь к ним быстрый доступ и готовить чаще, чем другие рецепты. Описание таблицы FavoriteRecipe находится в таблице 2.7.

Таблица 2.7 – Описание таблицы «FavoriteRecipe»

Поле	Описание
userId	Идентификатор пользователя
recipeId	Идентификатор рецепта
createdAt	Дата создания

Таблица SelectedRecipeForCooking хранит информацию о рецептах, которые пользователь выбрал для приготовления. Описание таблицы SelectedRecipeForCooking находится в таблице 2.8.

Таблица 2.8 – Описание таблицы «SelectedRecipeForCooking»

Поле	Описание
recipeId	Идентификатор рецепта
userId	Идентификатор пользователя
selectedAt	Дата создания
lastUpdatedAt	Дата обновления
isCooked	Приготовлен ли рецепт

Таблица UserToken хранит информацию о сессии пользователя. Благодаря этой таблице поддерживается вход с различных пользовательских устройств. Описание таблицы UserToken находится в таблице 2.9.

Таблица 2.9 – Описание таблицы «UserToken»

Поле	Описание
userId	Идентификатор пользователя
deviceId	Идентификатор устройства
refreshToken	Токен обновления
lastLoginAt	Дата последнего обновления записи

Таблица Recipe содержит общую информацию о рецепте, такую как название, тип, описание и прочее. Описание таблицы Recipe находится в таблице 2.10.

Таблица 2.10 – Описание таблицы «Recipe»

Поле	Описание
id	Идентификатор рецепта
creatorId	Идентификатор создателя
title	Название рецепта
type	Тип рецепта
description	Описание рецепта
createdAt	Дата создания

Таблица RecipeComposition характеризует состав рецепта. Описание таблицы RecipeComposition находится в таблице 2.11.

Таблица 2.11 – Описание таблицы «RecipeComposition»

Поле	Описание
recipeId	Идентификатор рецепта.
productId	Идентификатор продукта
quantity	Количество продукта

Таблица Product содержит полную информацию о продукте. Описание таблицы Product находится в таблице 2.12.

Таблица 2.12 – Описание таблицы «Product»

Поле	Описание
1	2
id	Идентификатор продукта

Продолжение таблицы 2.12

1	2
creatorId	Идентификатор создателя
title	Название продукта
calories	Количество калорий
protein	Количество белка
fats	Количество жиров
carbohydrates	Количество углеводов
unit	Размерность продукта (граммы, штуки и т.д.)

Данный набор таблиц достаточен для приложения и легко масштабируется.

2.4 Описание перечислений

Так как в проекте используется база данных PostgreSQL, есть возможность использовать перечисления вместо создания дополнительных таблиц.

В таблице 2.13 приведены все типы перечислений, использующиеся в проекте, а также краткое пояснение их роли.

Таблица 2.13 – Перечисления в приложении

Перечисление	Описание
Roles	Роль пользователя в системе
RecipeTypes	Типы рецептов
Units	Единицы измерения продуктов

Перечисление Roles используется при создании или регистрации пользователей. В таблице 2.14 пояснение всех значений.

Таблица 2.14 – Перечисление Roles

Роль	Описание
Admin	Модератор
Default	Рядовой пользователь приложения
God	Администратор. Максимальный уровень привилегий

Перечисление Units используется при создании продуктов и редактирований продуктов. Определяет какие единицы измерения будут использоваться при использовании продуктов в рецептах и хранилище. В таблице 2.15 пояснение всех значений.

Таблица 2.15 – Перечисление Units

Единица	Описание
1	2
Grams	Грамм
Liter	Литры
Pinch	Пинты
Pieces	Куски
Kilograms	Килограммы

Продолжение таблицы 2.15

1	2
Milliliter	Миллилитры
Tablespoon	Столовые ложки
Teaspoon	Чайные ложки
Items	Штуки
Portion	Порции

Перечисление RecipeTypes используется при создании рецептов и редактировании рецептов. Определяет какие типы блюд существуют в приложении. В таблице 2.16 пояснение всех значений.

Таблица 2.16 – Перечисление RecipeTypes

Тип	Описание
VEGETARIAN_DISHES	Вегетарианское
BAKING	Выпечка
GARNISHES	Гарниры
HOT_DISHES	Горячие блюда
HOT_APPETIZERS	Горячие закуски
DESSERTS	Десерты
HOME_MADE_FASTFOOD	Домашний фастфуд
BREAKFAST	Завтрак
PRESERVES	Заготовки
CHARCOAL	На углях
BEVERAGES	Напитки
SALADS	Салаты
SAUCES_PASTAS_AND_DRESSINGS	Соусы, пасты, приправы
SOUPS_AND_BROTHS	Супы и бульоны
DOUGH	Тесто
BREAD	Хлеб
COLD_APPETIZERS	Холодные закуски
SHISHKEBABS	Шашлыки
OTHER	Другое

Использование перечислений для типов рецептов и единиц измерений продуктов позволяет свести вероятность появления непредсказуемых пользовательских значений к нулю. Расширение перечислений стоит минимальных затрат разработчика: не нужно писать дополнительный функционал, а просто добавить значение.

2.5 Диаграмма вариантов использования

Диаграмма вариантов использования показывает все возможные варианты использования приложения и взаимодействия пользователей с различными ролями. Данная диаграмма для проекта представлена на рисунках 2.3 и 2.4.

В приложении присутствуют 3 основные роли:

- администратор приложения;
- обычный пользователь;
- модератор приложения.

Жизнь приложения начинается с одного встроенного администратора, которые, впоследствии, может создать других администраторов, модераторов и обычных пользователей. Администратор вправе управлять пользователями (удалять, восстанавливать, создавать, назначать роль). Также администратор имеет возможность создавать и удалять продукты и рецепты. Все изменения, внесенные администратором, будут отображаться у всех остальных пользователей.

Модератор имеет те же привилегии, что и администратор. Однако, модератор лишен способности создавать и удалять пользователей. Для манипуляции с пользователями модератор обязан связаться с одним из администраторов. Модератор способен изменять информацию о продуктах и рецептах.

У роли модератора и администратора своя экосистема в приложении. Две эти роли имеют больше привилегий, чем обычные пользователи, однако, в отличие от последних, не могут иметь и заполнять свои «хранилища».

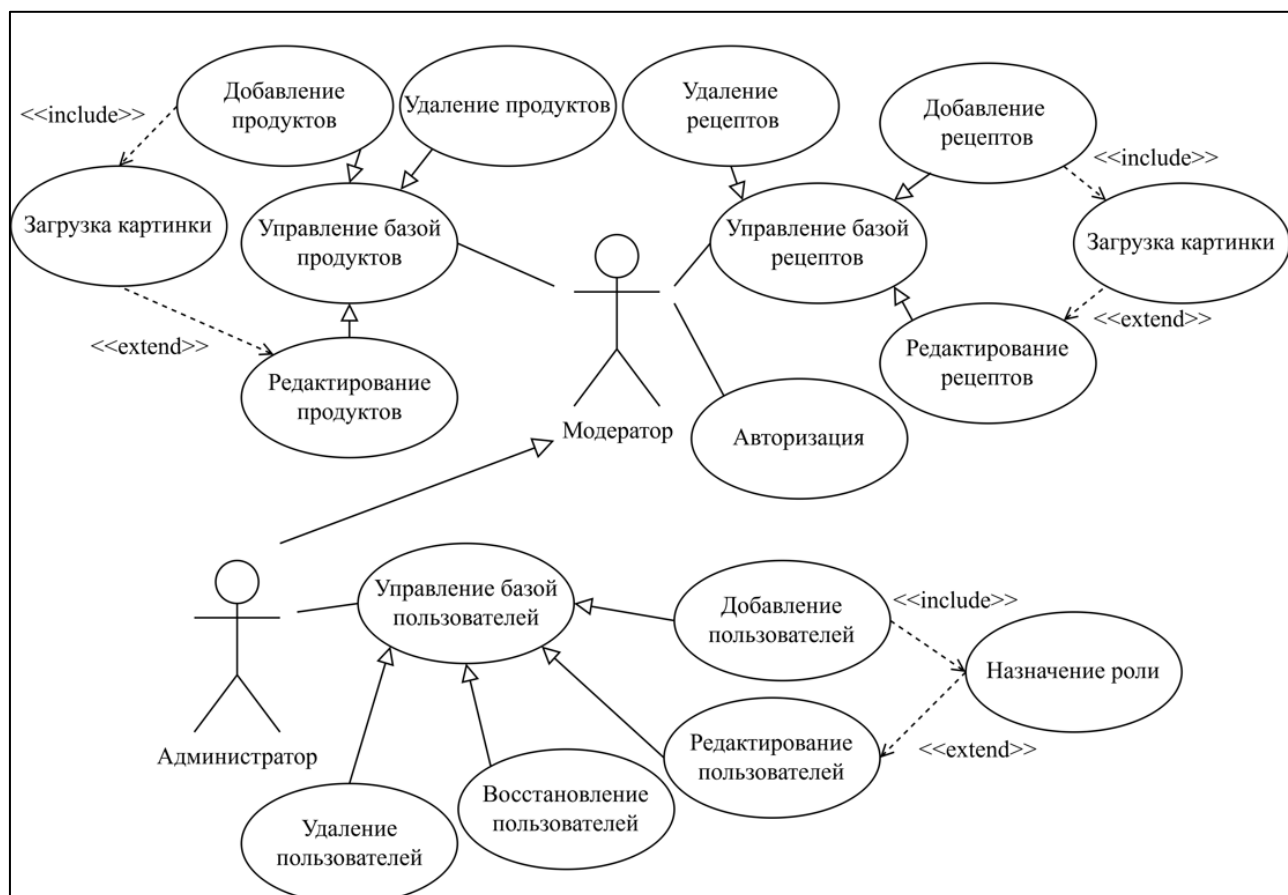


Рисунок 2.3 – Диаграмма вариантов использования роли администратор/модератор

Пользователь вправе зарегистрироваться в приложении или быть созданным администратором. Пользователь имеет доступ к рецептам и продуктам, а также к своему хранилищу продуктов. Пользователь имеет возможность выбирать рецепты, после чего продукты из выбранных рецептов попадут в чек-лист пользователя.

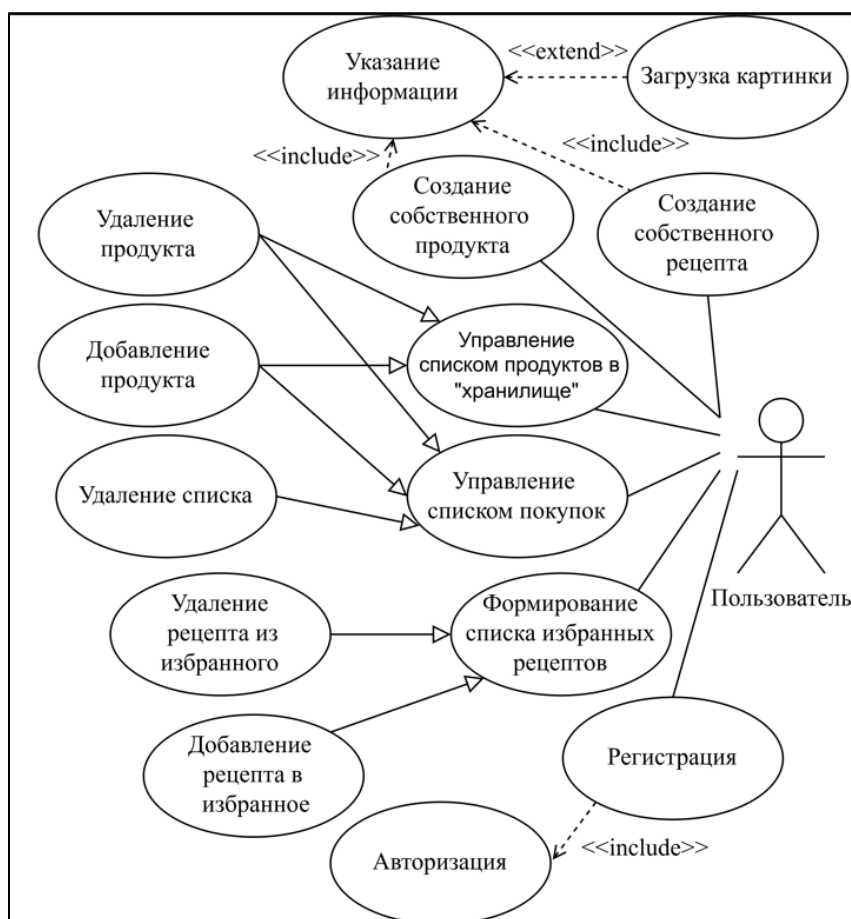


Рисунок 2.4 – Диаграмма вариантов использования обычного пользователя

Таким образом установив определенную иерархию ролей, станет проще разделять обязанности пользователей и интерфейсы. Стандартный пользователь не сможет сломать структуру приложения и повредить целостность данных, а администрация не сможет повредить записи пользовательских аккаунтов. Полная диаграмма вариантов использования находится в Приложении В.

2.6 Проектирование основных алгоритмов

Два основных алгоритма в приложении – добавление продукта в хранилище и формирование списка покупок. Они находятся в Приложениях Г и Д.

Алгоритм добавления продукта в хранилище описывает то, как продукт попадает в хранилище пользователю.

Алгоритм формирования списка покупок учитывает те продукты, которые уже имеются в хранилище пользователя. То есть, если продукта нету в хранилище – он добавляется в указанном количестве. Иначе, указанное количество вычитается из имеющегося, и разница между ними записывается в базу данных.

2.7 Выбор средств реализации

Для реализации проекта необходимо, первым делом, построить полную структуру проекта и выбрать платформу разработки. Для оптимизации временных затрат,

в качестве экосистемы и языка программирования был выбран TypeScript (надмножество языка JavaScript), в силу строгой типизаций.

Структура проекта стоит из трех компонентов:

- серверная часть (бизнес-логика);
- клиентская часть (UI);
- централизованная база данных.

2.8 Серверная часть

Для реализации серверной части был выбран Bun.js с использованием фреймворка Elysia для упрощения разработки. Она отвечает за обработку запросов, работу с базой данных и логику приложения

Bun.js (Bun) – это платформа с открытым исходным кодом для работы с языком JavaScript. Она позволяет писать серверный код для веб-приложений и динамических веб-страниц, а также программ командной строки. В основе платформы – событийно-управляемая модель с неблокирующими операциями ввода-вывода, что делает ее эффективной и легкой.

Тест производительность Bun показан на рисунке 2.5.

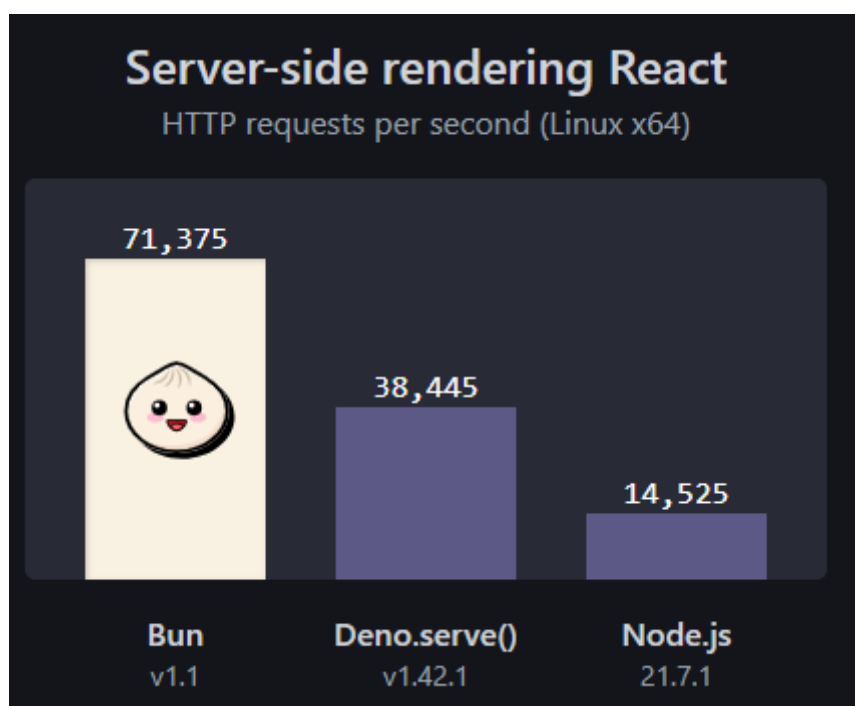


Рисунок 2.5 – Сравнение производительности Bun с аналогами

Веб-фреймворк Elysia (Bun.js/TypeScript) представляет собой популярный веб-фреймворк, написанный на JavaScript и работающий внутри среды исполнения Bun.js. Этот модуль освещает некоторые ключевые преимущества этого быстрого фреймворка, установку среды разработки и выполнение основных задач веб-разработки и развертывания.

Elysia – альтернатива популярному фреймворку ExpressJS. Так как это надстройка над API среды Bun, Elysia работает намного быстрее аналогов на NodeJS. Разработчики обещают прирост в скорости ответа от сервера в десятки раз,

по сравнению с конкурентом. Так же строгая типизация и отсутствие нужды описывать каждую конечную точку для клиента сильно облегчает работу разработчиков. Однако, стоит отметить минус, что для этого требуется так называемый моно репозиторий или отдельная генерация типов.

Во многих тестах среда выполнения Bun показывает себя в разы лучше, чем устаревший NodeJS. Это связано с тем, что Bun использует новый подход и инструменты, а также некоторые системные API, которые ускоряют работу с вводом-выводом файлов. При этом, разработчикам удалось сохранить практически полную обратную совместимость со средой выполнения NodeJS, что кратно увеличивает рейтинг Bun среди других сред исполнения.

Хостом для процесса сервера станет система на операционной системе GNU/Linux, так как среда Bun максимально оптимизирована под использование API именно систем на базе ядра Linux.

2.9 Клиентская часть

Для реализации клиентской части выбрана библиотека React, отвечающая за создание пользовательского интерфейса.

React (React.js или ReactJS) – это бесплатная библиотека JavaScript с открытым исходным кодом и огромным сообществом.

К преимуществам этой библиотеки относится компонентный подход, Virtual DOM, мемоизация и ленивая загрузка компонентов. Несмотря на свой богатый функционал, React все еще остается библиотекой, что позволяет его использовать в качестве основы для различных фреймворков.

2.10 Централизованная база данных

Хранение данных будет реализовываться с помощью PostgreSQL. Так как она является одним из нескольких бесплатных популярных вариантов СУБД, часто используется для ведения баз данных веб-сайтов. Это была одна из первых разработанных систем управления базами данных, поэтому в настоящее время она хорошо развита, и позволяет пользователям управлять как структурированными, так и неструктурированными данными. Может быть использован на большинстве основных платформ, включая Linux. Прекрасно справляется с задачами импорта информации из других типов баз данных с помощью собственного инструментария.

2.11 Выводы по разделу

В данной главе была разработана диаграмма вариантов использования мобильного приложения, которая позволила составить более целостное понимание всех функциональных возможностей проекта.

Была спроектирована база данных, где определены необходимые таблицы, поля для каждой из них, типы данных и ограничения целостности. Также были установлены связи между таблицами, которые описаны на схеме базы данных.

Помимо этого, были представлены основные диаграммы, полно определяющие задумку проекта, что позволяет быстро и наглядно вникнуть в суть продукта.

3 Разработка приложения

3.1 Реализация серверной части

Построение правильной архитектуры проекта – важная часть разработки. Правильная структура ускоряет процесс разработки, что, в свою очередь, обеспечивает наименьшую энтропию функций и файлов (меньше ненужных слоев абстракции в проекте), наилучший уровень абстракции и максимальную производительность.

При разработке серверной части дипломного проекта была создана структура, представленная на рисунке 3.1.

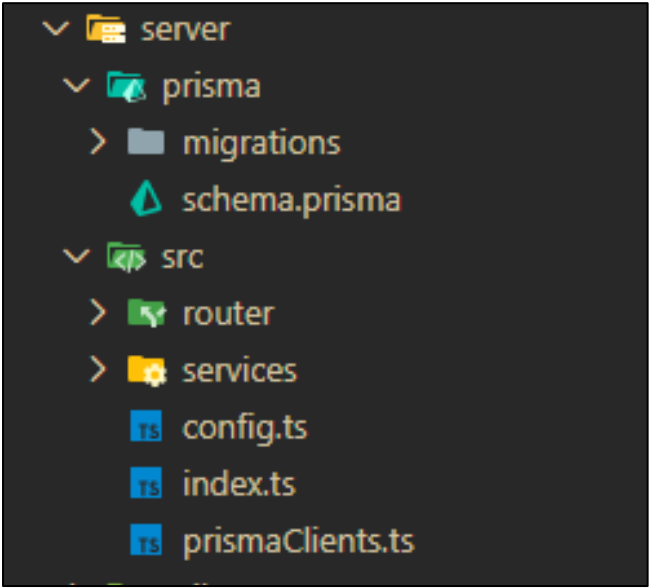


Рисунок 3.1 – Структура сервера

Данная структура достаточна для реализации всех нужд проекта. Ниже приведено подробное описание всей структуры.

Каждая из указанных папок играет ключевую роль в структуре и функционировании серверной части проекта, обеспечивая определенные функциональные возможности и обработку данных, необходимых для корректной работы приложения.

Логически, архитектура сервера распределена на две основные части: взаимодействие с базой данных и основная бизнес-логика сервера.

Так как в проекте решено использовать технологию ORM и ее реализацию в виде Prisma ORM (ORM для JavaScript), для нее используется отдельная папка prisma. В ней находятся два основных компонента: файл схемы и папка с миграциями. Для создания таблиц в системе Prisma используется свой особый язык, слабо похожий на типичный JavaScript.

				БГТУ 03.00.ПЗ				
	ФИО	Подпись	Дата					
Разраб.	Гончаревич Е. В.			3 Разработка приложения	Лит.		Лист	Листов
Пров.	Сазонова Д. В.					У	1	12
					74218008, 2024			
Н. контр.	Нистюк О. А.							
Утв.	Блинова Е. А.							

Пример описания модели приведен на листинге 3.1. Полный листинг описания моделей находится в Приложении Е.

```
model Storage {
  id          String          @id @default(cuid())
  creatorId   String          @unique
  title       String?         @db.VarChar(30)
  user User @relation(fields: [creatorId], references: [id],
onDelete: Cascade)
  StorageComposition StorageComposition[]

  @@index([creatorId], type: Hash)
}
```

Листинг 3.1 – Листинг модели Storage

Синтаксис удобен и достаточно понятен. Возможности движка Prisma позволяют задать все необходимые ограничения для базы данных и таблиц, такие как уникальные поля, отношения, поведения при удалении, индексация и прочее.

В папке с миграциями находятся датированные миграции, что позволяет читать историю и сохранять последовательность миграций. Так же, в случае ошибки, можно вернуться к предыдущей точке сохранения. Каждая миграция представляет собой файл типа SQL, в котором описаны все изменения. Пример на рисунке 3.2.

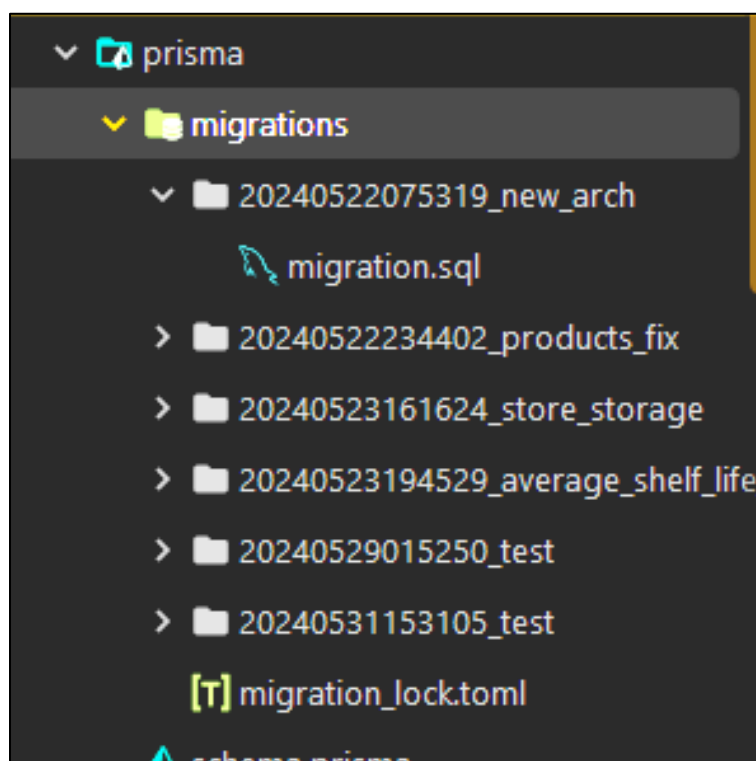


Рисунок 3.2 – История миграций проекта

Файл migration-lock.toml хранит информацию о провайдере базы данных.

Основная логика сервера находится в папке src. Именно здесь находятся маршрутизаторы и сервисы. Для каждого сервиса существует свой файл, описывающий функционал и логику взаимодействия с базой. Структура на рисунке 3.3.

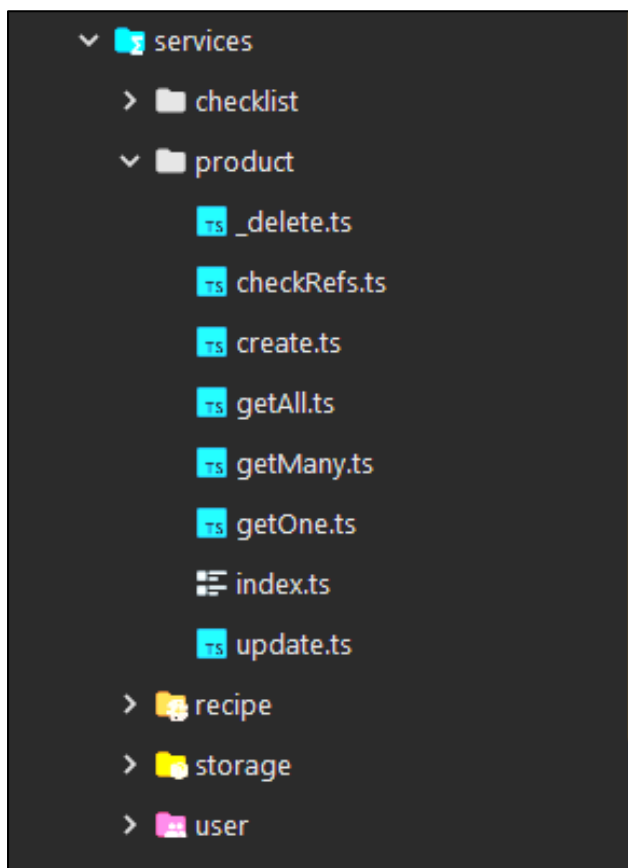


Рисунок 3.3 – Структура сервисов

Весь маршрутизатор реализован в папке router. Здесь определены все конечные точки приложения, а также настраивается валидация параметров запроса. Структура маршрутизатора показана на рисунке 3.4.

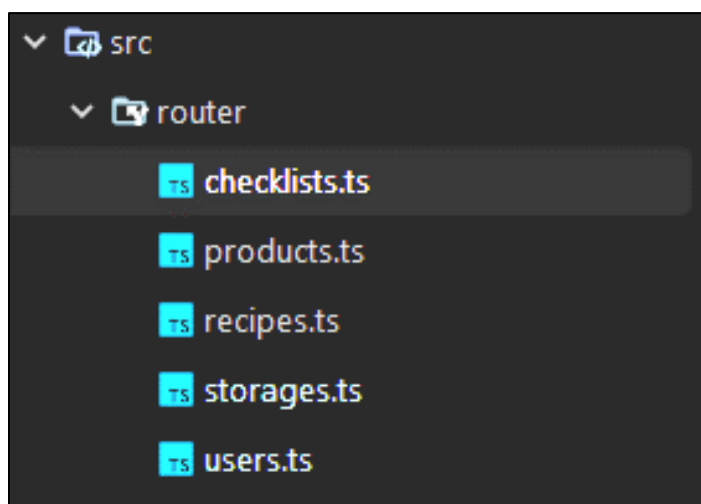


Рисунок 3.4 – Структура маршрутизатора

Под каждую сущность в приложении также отведен свой собственный файл с названием, соответствующим сущности.

Маршрутизатор распределен по различным файлам. Для объединения конечных точек используется файл index. Листинг полный маршрутизатора рецептов можно посмотреть в Приложении Ж.

Точка входа показана на листинге 3.2.

```
const app = new Elysia()
  .onError(e => console.log(e))
  .use(cors({ credentials: true }))
  .use(cookie())
  .group('/users', users)
  .group('/products', products)
  .group('/recipes', recipes)
  .group('/checklists', checklists)
  .group('/storages', storages)
  .use(swagger())

await publicDBClient.$connect()
app.listen(CONFIG.PORT, () =>
  console.log(`Server started on port ${CONFIG.PORT}`)
)
export type VF_API_ROUTER_TYPES = typeof app
process.on('SIGINT', async () => {
  await publicDBClient.$disconnect()
  console.log('SIGINT received')
  process.exit(0)
})
```

Листинг 3.2 – Листинг точки входа в приложение

Контроллеров в проекте нету как таковых, так как эту задачу берет на себя фреймворк. Его реализация позволяет абстрагироваться от их ручного описания.

Файл конфигурации и переменные среды сервера находится в файле `config`. Конфигурация показана на листинге 3.3.

```
export const CONFIG = {
  PORT: Bun.env.PORT || 3000,
  JWT_ACCESS: Bun.env.JWT_ACCESS || 'AcCeSs',
  JWT_REFRESH: Bun.env.JWT_REFRESH || 'ReFrEsH',
  CLIENT_URL: Bun.env.CLIENT_URL || 'http://localhost:3001',
}
```

Листинг 3.3 – Листинг конфигурации

Настройка клиента базы данных находится в файле `prismaClient`. Создание клиента показано на листинге 3.4.

```
import { PrismaClient } from 'prisma/prisma-client'

export const publicDBClient = new PrismaClient({
  errorFormat: 'minimal',
  log: ['info', 'warn', 'error'],
})
```

Листинг 3.4 – Листинг `prismaClient`

В опциях указаны параметры вывода ошибок и логирование.

PrismaClient является мостом между базой данных и бизнес-логикой. JavaScript обращается к API Prisma через мост, что гарантирует безопасность и оптимизацию запросов к базе данных. Благодаря типизации разработчик сразу видит ожидаемый результат, что снижает вероятность ошибки доступа к несуществующим данным к минимуму. Клиент автоматически конвертирует методы JavaScript в SQL.

Для хранения картинок в приложении создан отдельный сервис. Листинг кода сервиса можно посмотреть в Приложении И.

3.2 Разработка клиентской части

Как уже описывалось ранее, для реализации клиентской части была выбрана прогрессивная библиотека React.js. React – это JavaScript-библиотека для создания пользовательских интерфейсов.

Так как для разработки клиента было принято решение использовать React, основное разграничение по пакетам происходило по принципу принадлежности к определенной библиотеке или паттерну проектирование веб-приложения. Проектирование с помощью React подразумевает создание компонентов, взаимодействующих друг с другом. Главным компонентом, с которого начинается работа приложения, является App.tsx. Структуру проекта можно увидеть на рисунке 3.5.

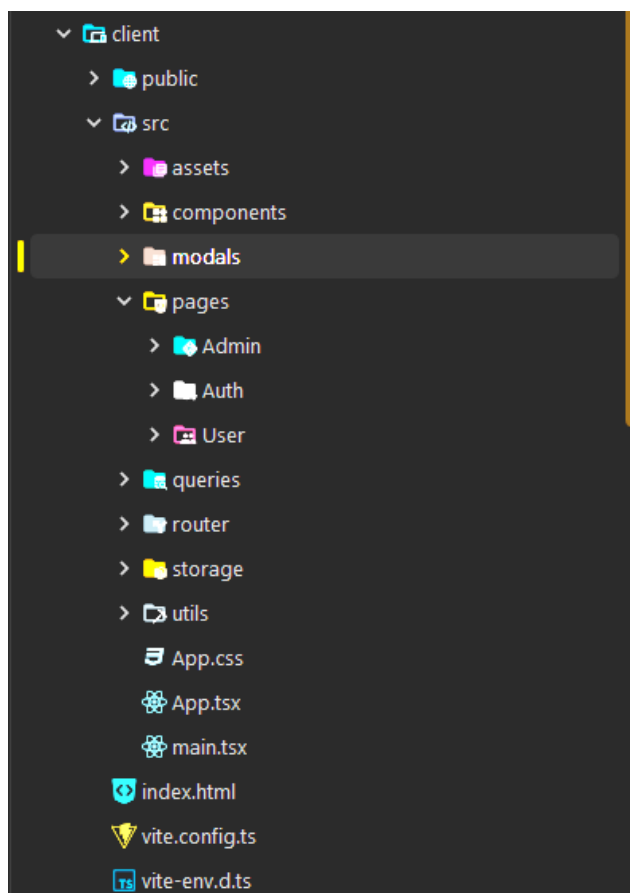


Рисунок 3.5 – Структура клиентской части

Папка components содержит внутри компоненты, которые представляют собой маленькие части пользовательского интерфейса приложения. Компоненты могут

быть переиспользуемыми и содержать в себе логику и представление для отдельных элементов интерфейса. В компонентах имеется разделение на две папки, в которых содержатся модальные окна как для админа, так и для клиента.

Структура папки components изображена на рисунке 3.6.

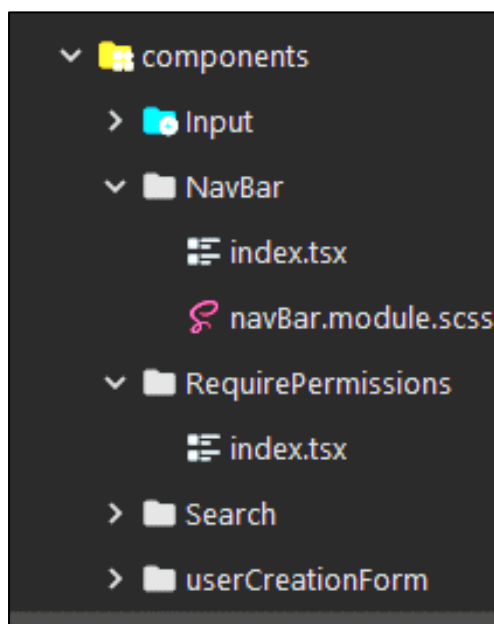


Рисунок 3.6 – Структура components

Пример компонента из папки components показан на листинге 3.5.

```
export function Input(props: IInputProps) {
  const [isFocused, setIsFocused] = useState(false)
  return (
    <div className={s.container}>
      <p className={s.label}>{props.label}</p>
      <input
        onFocus={() => setIsFocused(true)}
        onBlur={() => setIsFocused(false)}
        type={props.type}
        placeholder={props.placeholder}
        maxLength={props.maxLength}
        value={props.value ?? ''}
        onChange={props.onChange}
        className={` ${s.input} ${isFocused ? s.activeInput : ''} ${props.hasError ? s.errorInput : ''}`}
        min={0}
      />
      <div className={s.errorText}>
        {props.hasError ? props.errorText : ''}
      </div>
    </div>
  )
}
```

Листинг 3.5 – Листинг компонента Input

К каждому компоненту привязан свой файл-модуль со стилями. Концепция модульности стилей позволяет забыть про архитектуру БЭМ и подобные ей, так как сборщик автоматически генерирует уникальные стили для каждого файла компонентов, что исключает повторяемость и конфликт стилей.

Папка `queries` содержит объект `treaty` (аналог `Fetch API`) и дополнительные объекты для обращения к серверу. Структура папки на рисунке 3.7.

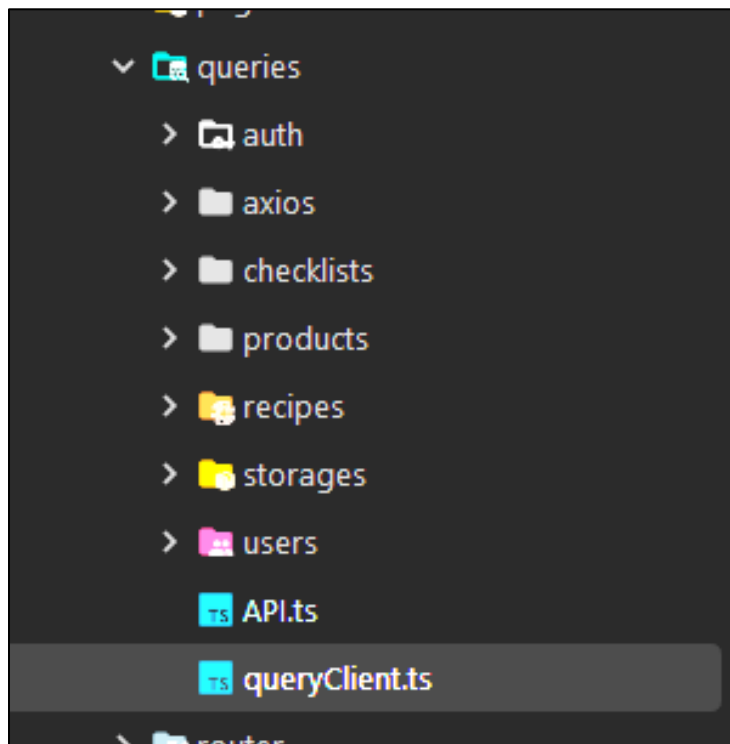


Рисунок 3.7 – Структура `queries`

`Pages` – папка, содержащая страницы приложения. Страницы собираются из компонентов. Структура `pages` показана на рисунке 3.8.

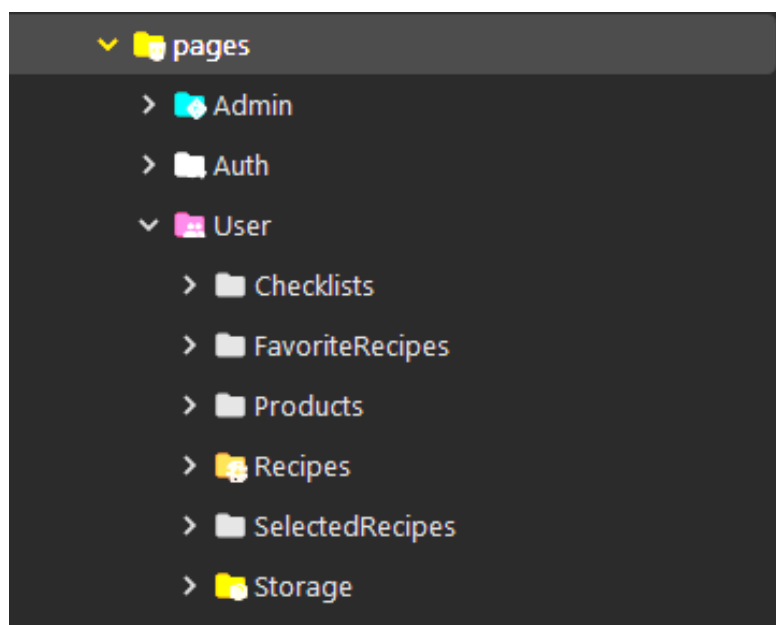


Рисунок 3.8 – Структура `pages`

Папка storage содержит логику по управлению глобальным состоянием приложения. Структура хранилища состояний показана на рисунке 3.9.

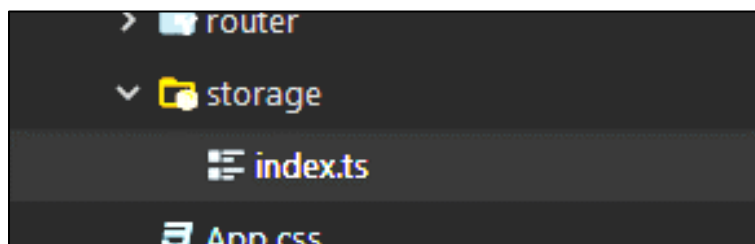


Рисунок 3.9 – Структура storage

Страницы разделены на интерфейс администратора и модератора, пользователя и страницу авторизации.

Assets хранит файлы, такие как картинки, которые динамически импортируются. В отличие assets, файлы в папке public являются общедоступными из любой точки приложения и при компиляции проекта не изменяют своей структуры и названия. Так же к assets нету прямого доступа через браузер.

3.3 Контейнеризация

Самым удобным способом, благодаря которому можно осуществить развертывание приложения – это Docker. Docker позволяет создавать контейнеры, в которых приложения могут существовать независимо от других приложений.

Чтобы развернуть клиент-серверное приложение на React и Node.js в Docker, требуется создать Docker-контейнеры для каждой части приложения (клиентской и серверной) и настроить их работу вместе. Добавленный Dockerfile для клиентской части и серверной представлен на рисунке 3.10.

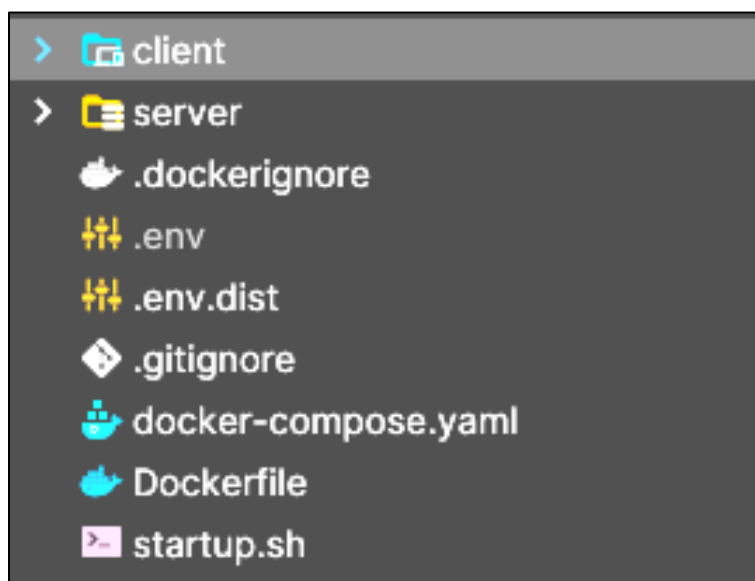


Рисунок 3.10 – Добавление Dockerfile для клиента и сервера

Код Dockerfile представлен в листинге 3.6.

```
FROM node:21-alpine3.18
WORKDIR /app
COPY . .
WORKDIR /app/server
RUN npm i
RUN npx prisma generate
RUN npm run compile
WORKDIR /app/client
RUN npm i
WORKDIR /app
RUN chmod +x startup.sh
CMD ["/bin/sh", "startup.sh" ]
```

Листинг 3.6 – Dockerfile клиентской и серверной части проекта

Затем необходимо создать Docker-образ из Dockerfile в текущем рабочем каталоге и присваивания ему тег.

Создание образа представлено на рисунке 3.11.

```
D:\4\library\client>docker build -t client-app .
[+] Building 348.3s (7/15)                                docker:default
=> [internal] load build definition from Dockerfile        0.1s
=> => transferring dockerfile: 1.16kB                      0.0s
=> [internal] load .dockerignore                          0.0s
=> => transferring context: 2B                              0.0s
=> [internal] load metadata for docker.io/library/nginx:stable-alpine 2.3s
=> [internal] load metadata for docker.io/library/node:latest 2.2s
=> [auth] library/nginx:pull token for registry-1.docker.io 0.0s
=> [auth] library/node:pull token for registry-1.docker.io 0.0s
=> [build 1/6] FROM docker.io/library/node:latest@sha256:db2672e3c200b85e0b813cdb294fac16764711d7a66b41315e626 345.9s
=> => resolve docker.io/library/node:latest@sha256:db2672e3c200b85e0b813cdb294fac16764711d7a66b41315e6261f2231f2 0.0s
=> => sha256:db2672e3c200b85e0b813cdb294fac16764711d7a66b41315e6261f2231f2331 1.21kB / 1.21kB 0.0s
=> => sha256:27e1a8ca91d35598fbae8dee7f1c211f0f93cec529f6804a60e9301c53a604d0 14.68MB / 24.05MB 345.9s
=> => sha256:5758f6b5d1790ea1cb16223ec92e964bb3222d55787725e889c5c39e8d40f50f 2.00kB / 2.00kB 0.0s
=> => sha256:b866e35a0dc4df85e168524b368567023eb22b06fe16f2237094e937fcd24d96 7.53kB / 7.53kB 0.0s
=> => sha256:90e5e7d8b87a34877f61c2b86d053db1c4f440b9054cf49573e3be5d6a674a47 14.68MB / 49.58MB 345.9s
=> => sha256:d3a767d1d12e57724b9f254794e359f3b04d4d5ad966006e5b5cda78cc382762 14.68MB / 64.13MB 345.9s
=> [stage-1 1/2] FROM docker.io/library/nginx:stable-alpine@sha256:0571deea2f7c6779607b66ee64e270922bb28984 345.9s
=> => resolve docker.io/library/nginx:stable-alpine@sha256:0571deea2f7c6779607b66ee64e270922bb289849f764d9edc 0.0s
=> => sha256:0571deea2f7c6779607b66ee64e270922bb289849f764d9edc15645d132f5 1.65kB / 1.65kB 0.0s
=> => sha256:762ac2dbbd669a6e1e84cd189b85b95dde160a7262540fb6a9cf805e04c5b902 1.78kB / 1.78kB 0.0s
=> => sha256:bbb1abad84c039801cc6140102d0a481000131699380dec7b332264072e8be07 16.32kB / 16.32kB 0.0s
=> [internal] load build context                          237.8s
=> => transferring context: 1.34GB                          237.5s
```

Рисунок 3.11 – Создание образа из Dockerfile

В листинге 3.7 приведен код startup.sh. Этот файл, выполняющийся после запуска контейнера, производит определенные пост-настройки.

```
#!/bin/sh
cd /app/server || exit
npm run apply_migrations
npm start &
cd /app/client || exit
npm run build
npm run preview
```

Листинг 3.7 – Листинг startup.sh

На данном этапе были созданы все необходимые docker-файлы, которые нужны для создания и запуска контейнеров Docker. Что бы не запускать все части проекта отдельно, был создан файл `docker-compose.yaml`. В нем будет происходить процесс создания сервисов, настройка их окружения, конфигурационных файлов и так далее. Как правило, к `docker-compose` файлу привязаны переменные среды, например, находящиеся в файле `.env` (environment). Указание этих переменных позволяет запускать экземпляры контейнеров с различными параметрами. Правильно написанный `docker-compose` исключает его последующее редактирование, так как управление может производиться лишь сменой переменных среды. Содержимое данного файла представлено в листинге 3.8.

```
version: '3.8'
services:
  db:
    image: postgres
    container_name: VF_DB
    restart: always
    environment:
      - POSTGRES_USER
      - POSTGRES_PASSWORD
      - POSTGRES_DB
    ports:
      - '5432:5432'
    volumes:
      - ./data/postgres:/var/lib/postgresql/data
    networks:
      - VF_NET
  adminer:
    container_name: Adminer
    image: adminer
    restart: always
    ports:
      - '8080:8080'
    networks:
      - VF_NET
volumes:
  db_data:
networks:
  VF_NET:
    driver: bridge
```

Листинг 3.8 – Файл «`docker-compose.yaml`»

Затем `docker-compose` читает конфигурацию из файла `docker-compose.yaml`, создает и запускает все контейнеры для сервисов, указанных в этом файле, и после этого отдает управление командной строке обратно, если указан соответствующий параметр. После создания контейнера доступна вся сводка о нем, журнал действий, данные о системе и прочее. Так же присутствует возможность интерактивного взаимодействия с контейнером, если необходимого функционала недостаточно или требуется ручное управление.

Результат сборки контейнеров показан на рисунке 3.12.

```

→ Virtual-Fridge git:(migrate-to-bun) X SIGINT received
→ Virtual-Fridge git:(migrate-to-bun) docker-compose up -d
WARN[0000] /home/eugene/Virtual-Fridge/docker-compose.yaml:
[+] Running 2/0
  ✓ Container VF_DB      Running
  ✓ Container Adminer    Running
→ Virtual-Fridge git:(migrate-to-bun) |

```

Рисунок 3.12 – Контейнеры

Созданные таким образом контейнеры можно легко транспортировать на любую систему в любую сеть, так как платформа абстрагируется от системы, в которой она работает. Так же имеется возможность легкой настройки оркестратора для балансировки нагрузки контейнеров, настройки общей сети и так далее.

В отличие от виртуальной машины, платформа Docker не требует большого числа ресурсов для развертки системы.

Созданные контейнеры представлены на рисунке 3.13.

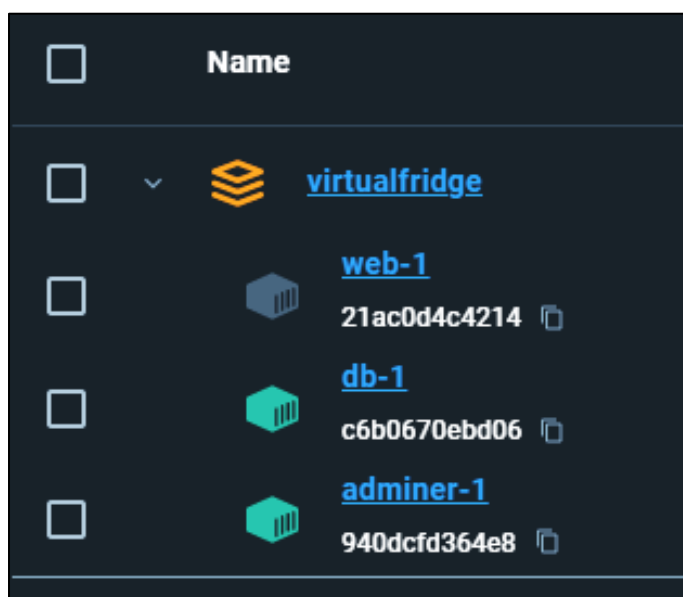


Рисунок 3.13 – Контейнеры

Использование Docker позволило эффективно изолировать приложение и его зависимости, обеспечивая повышенную масштабируемость и упрощая процесс разработки. Применение инструментов Bun.js и React в контейнерах дало возможность создать динамический пользовательский интерфейс.

3.4 Выводы по разделу

В данном разделе были подробно описаны этапы проектирования программного средства. Сперва были выбраны средства для реализации программного средства, далее создана структура папок и файлов, произведена настройка зависимостей.

Для реализации серверной части приложения был использован Bun.js вместе с фреймворком Elysia.js для создания веб-приложения. Показана архитектура приложения с пояснением каждой детали. Произведена настройка параметров сервера, а также создан клиент ORM Prisma для взаимодействия с базой данных по принципу Code First и настроено журналирование операций.

Для разработки клиентской части приложения была выбрана библиотека React. Этот инструмент позволяет строить динамичные интерфейсы, обладает высокой гибкостью и удобством разработки. Сделан упор на использование ленивой загрузки компонентов, что дает несколько преимуществ, такие как уменьшение времени до первого взаимодействия (TTI) и, соответственно, меньший размер конечной сборки, так как «ленивые» модули загружаются в приложение по мере надобности. Сборщик Vite позволил использовать все преимущества разработки с его использованием, такие как горячая замена модулей (HMR), удобное разбиение файлов CSS на модули и генерацию уникальных имен классов.

Для хранения данных была выбрана СУБД PostgreSQL, так как известна своей надежностью и стабильностью. Она имеет долгую историю разработки, активное сообщество разработчиков и пользователей, а также широко используется в различных проектах, что говорит о ее надежности. Из преимуществ, из-за которых выбор пал именно на эту систему, можно привести тип данных ENUM, устанавливающий ограничение на определенные значения в полях и, соответственно, уменьшающий набор таблиц в схеме базы данных. Ещё одним неоспоримым преимуществом является наилучшая интеграция с ORM Prisma.

Также была сформулирована архитектура взаимодействия всех 3 частей программного средства на основе трехуровневой архитектуры.

После описания технологий была сформирована база данных и таблицы, из которых она состоит. Таблицы связаны в основном между собой связями один-ко-многим и один-к-одному.

Дополнительно были изучены варианты использования и построена соответствующая диаграмма, а также построены блок-схемы алгоритма вычета продуктов из списка покупок и составления общего списка покупок.

Для удобства разработки и развертывания приложения использовалась платформа для контейнеризации Docker, которая позволяет поместить приложение со всеми зависимостями в контейнер, обеспечивая легкую переносимость и удобство управления окружением приложения.

Для оркестрации созданных контейнеров задействована утилита Docker-Compose, позволяющая управлять связкой контейнеров как единым целым. Конфигурация описана в файле docker-compose.yml. В этом файле указываются имена контейнеров, описываются переменные среды и настраивается общая локальная сеть.

Список всех установленных зависимостей находится в Приложении К.

4 Тестирование приложения

Тестирование веб-приложений – это комплекс определенных услуг, который может включать в себя различные виды тестирования программного обеспечения.

Основная цель любого тестирования, в том числе и тестирования веб-приложений, – обнаружить все ошибки в программном обеспечении и разработать рекомендации по их предотвращению в будущем.

Одним из видов тестирования, которое затрагивает все возможные этапы при использовании ПО, является регрессионное тестирование. Данное тестирование предполагает тестирование любых способов взаимодействия с приложением любыми возможными методами.

Сперва необходимо протестировать форму регистрации и авторизации. Форма авторизации представлена на рисунке 4.1.

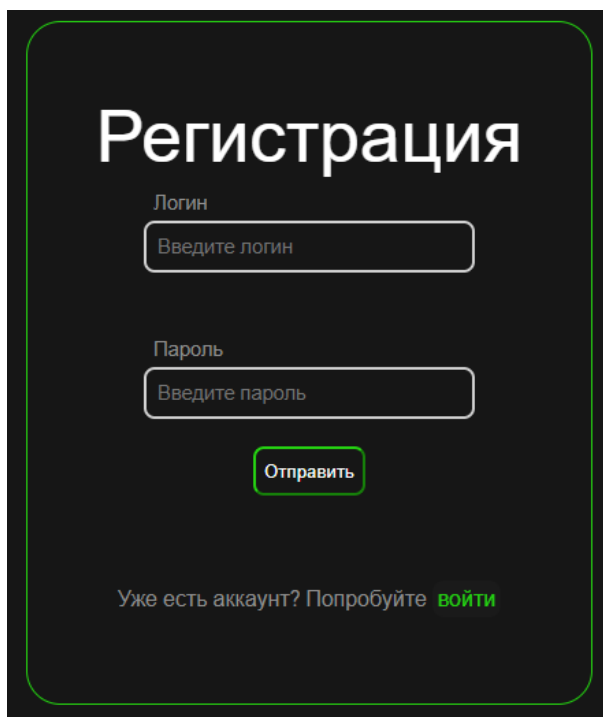
Рисунок 4.1 – Форма авторизации

В данном случае, форма регистрации находится в том же месте и на той же странице, что и форма авторизации. Для переключения режима необходимо нажать соответствующую кнопку.

Для каждой типа входа предусмотрена валидация полей как на клиентской части, так и на серверной. Даже при использовании стороннего клиента, сервер не пропусти не валидные данные.

Форма регистрации представлена на рисунке 4.2.

				БГТУ 04.00.ПЗ			
	ФИО	Подпись	Дата				
Разраб.	Гончаревич Е. В.			4 Тестирование приложения		Лит.	Лист
Пров.	Сазонова Д. В.						Листов
							1
Н. контр.	Нистюк О. А.					74218008, 2024	
Утв.	Блинова Е. А.						



Регистрация

Логин

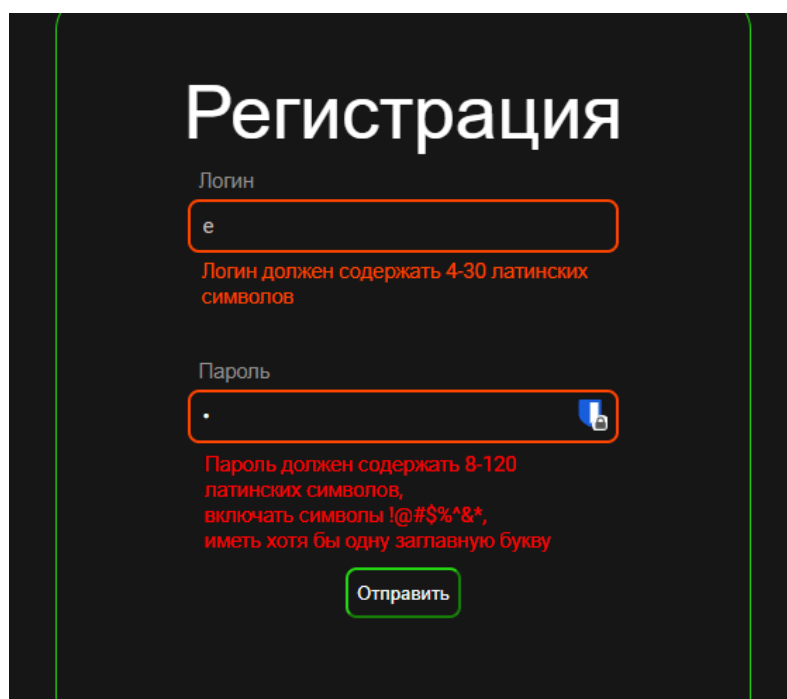
Пароль

Отправить

Уже есть аккаунт? Попробуйте [войти](#)

Рисунок 4.2 – Форма регистрации

Формы проверяют введенные в них данные при попытке отправки формы. По полям Login и Password стоит правило обязательного поля (рисунок 4.3).



Регистрация

Логин

Логин должен содержать 4-30 латинских символов

Пароль

Пароль должен содержать 8-120 латинских символов, включать символы !@#%&*, иметь хотя бы одну заглавную букву

Отправить

Рисунок 4.3 – Валидация обязательных полей

Для того, чтобы пользователь однозначно понимал, что не так с полями, предусмотрены подписи под каждым пользователем. Подпись содержит все необходимые требования к тексту в полях ввода. При попытке входа пользователя, который не был зарегистрирован, также появится предупреждающее сообщение (рисунок 4.4). Это однозначно указывает пользователю именно на его ошибку.

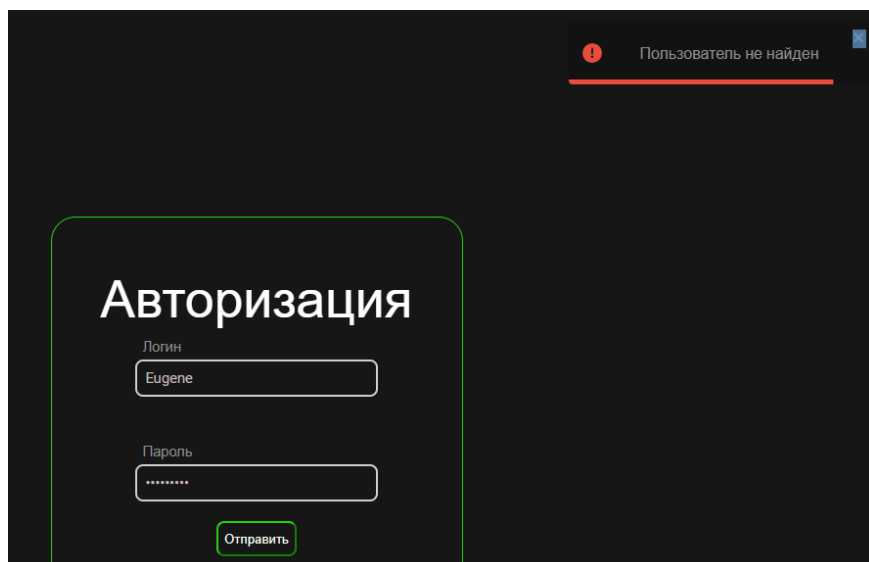


Рисунок 4.4 – Сообщение о незарегистрированном пользователе

Протестируем создание пользователя с учетной записи администратора. При попытке ввести некорректные данные поля горят красным цветом, а при наведении на них можно наблюдать всплывающее сообщение с подсказкой. Пример валидации данных изображен на рисунке 4.5.

Валидация логина и пароля при создании пользователя позволяет избежать ситуации, когда пользователь создан администратором некорректно, а сам пользователь не имеет возможности войти в систему из-за ошибки в форме авторизации.

A screenshot of a web application's user creation interface. The title 'Создание пользователя' is centered at the top. Below it are four input fields: 'Название' (Name) with placeholder 'Введите логин', 'Пароль' (Password) with placeholder 'Введите пароль', 'Архивировать пользователя' (Archive user) checkbox, and 'Роль' (Role) dropdown menu with 'Обычный пользователь' (Regular user) selected. At the bottom are two buttons: a green 'Сохранить' (Save) button and an orange 'Отмена' (Cancel) button.

Рисунок 4.5 – Форма создания пользователя

При попытке создать пользователя, который существует, появится уведомление о том, что логин уже занят. Уникальность логина важна для однозначной идентификации пользователя. Ограничение прописано на уровне базы данных так, что, даже обойдя валидацию на клиенте, ограничение целостности не будет нарушено.

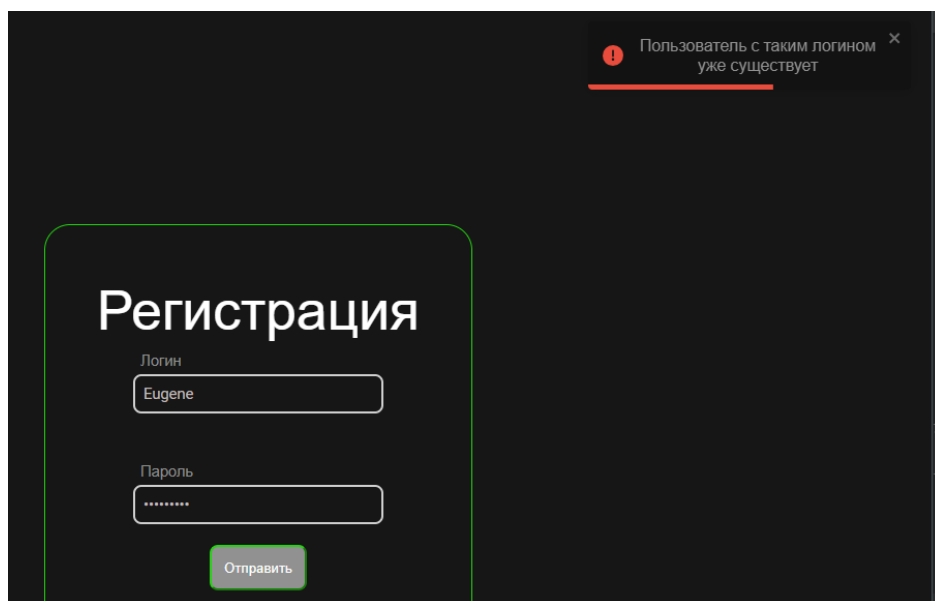


Рисунок 4.6 – Уведомление о занятом логине

Во время работы с приложением пользователям придется столкнуться с созданием различных рецептов или продуктов. Для каждого вида сущности существуют свои ограничения, например, недопустимые символы в имени продукта, числовые поля и дублирование названий сущностей. На рисунке 4.7 показана проверка недопустимых символов в создании/редактировании продукта. Аналогичная валидация находится и в создании рецептов.

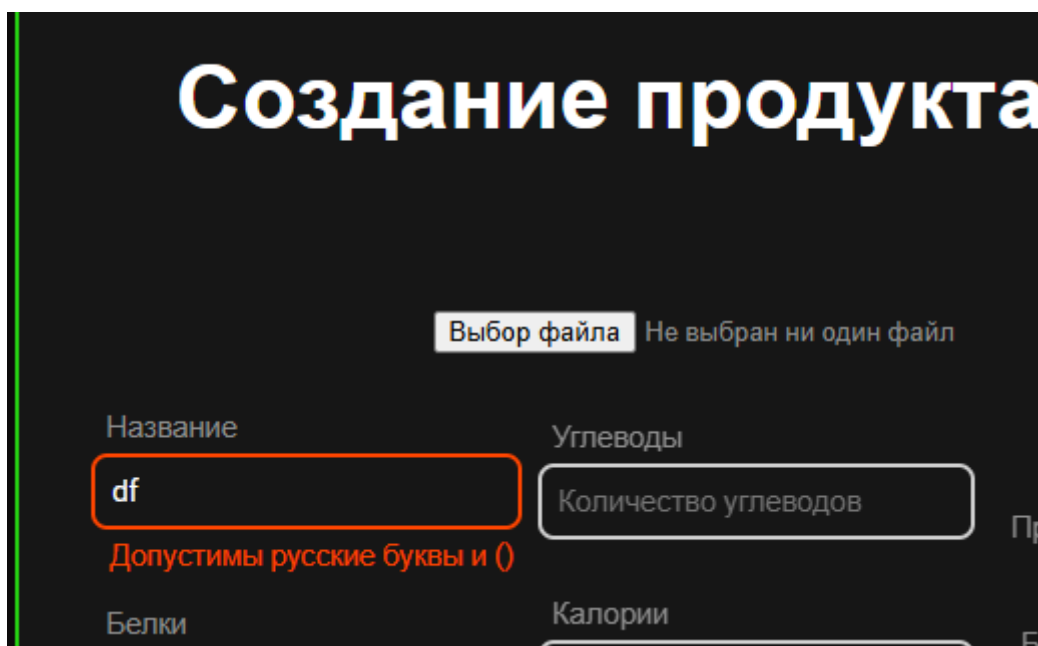


Рисунок 4.7 – Валидация названия в создании продукта

Данное ограничение необходимо для того, чтобы исключить некорректные названия из символов и цифр.

Так как в создании есть числовые значения, то необходимо исключить попадание строковых значений туда, где это недопустимо. Рисунок 4.8 демонстрирует поле для ввода только числовых значений не меньше нуля.

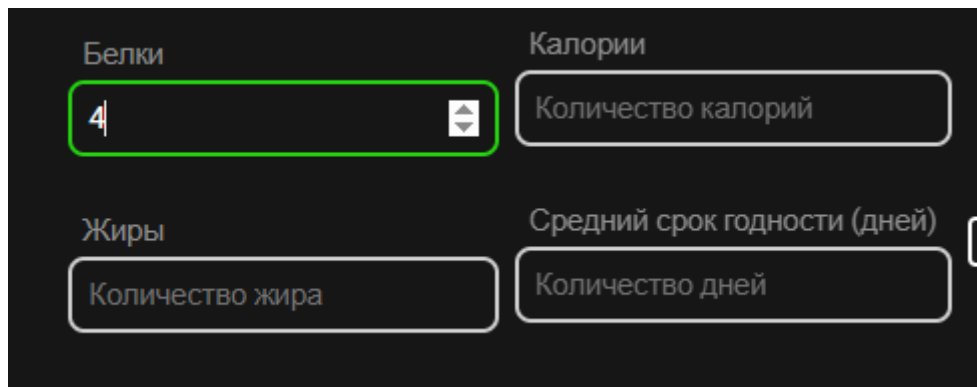
The image shows a dark-themed user interface for creating a product. It features four input fields arranged in a 2x2 grid. The top-left field is labeled 'Белки' (Proteins) and contains the number '4'. The top-right field is labeled 'Калории' (Calories) and contains the text 'Количество калорий'. The bottom-left field is labeled 'Жиры' (Fats) and contains the text 'Количество жира'. The bottom-right field is labeled 'Средний срок годности (дней)' (Average shelf life (days)) and contains the text 'Количество дней'. A green rectangular box highlights the 'Белки' input field, indicating a validation check for numerical values.

Рисунок 4.8 – Валидация названия в создании продукта

При создании продукта можно указать его единицы измерения, такие как порции, граммы, килограммы и так далее. В схеме базы эти значения прописаны и являются конечными, поэтому нужно ограничить пользователя в выборе единиц измерения. Рисунок 4.9 демонстрирует выпадающий список с допустимыми значениями единиц измерения продукта.

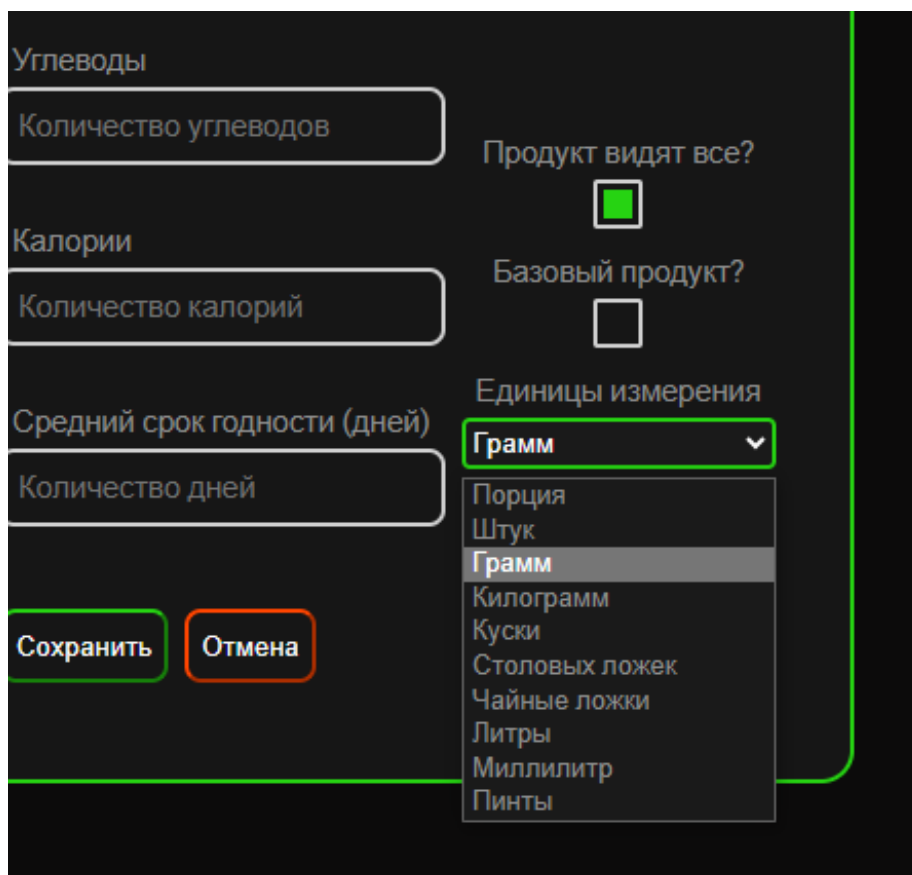
The image shows a dark-themed user interface for creating a product. It features four input fields arranged in a 2x2 grid. The top-left field is labeled 'Углеводы' (Carbohydrates) and contains the text 'Количество углеводов'. The top-right field is labeled 'Калории' (Calories) and contains the text 'Количество калорий'. The bottom-left field is labeled 'Средний срок годности (дней)' (Average shelf life (days)) and contains the text 'Количество дней'. The bottom-right field is labeled 'Единицы измерения' (Units of measurement) and has a dropdown menu open. The dropdown menu lists several units: 'Порция' (Portion), 'Штук' (Pieces), 'Грамм' (Grams), 'Килограмм' (Kilograms), 'Куски' (Pieces), 'Столовых ложек' (Tablespoons), 'Чайные ложки' (Teaspoons), 'Литры' (Liters), 'Миллилитр' (Milliliters), and 'Пинты' (Pints). The 'Грамм' option is currently selected and highlighted. At the bottom of the form, there are two buttons: 'Сохранить' (Save) and 'Отмена' (Cancel). A green line highlights the 'Единицы измерения' dropdown menu and the 'Сохранить' button.

Рисунок 4.9 – Ограничение единиц измерения

Данное ограничение не позволяет создать продукты с одинаковым названием и разными единицами измерения.

В системе предусмотрены уведомления об успешном завершении какой-либо операции, будь то создание рецепта, создание продукта, добавление продукта в хранилище и прочее. Рисунок 4.10 демонстрирует уведомление пользователя.

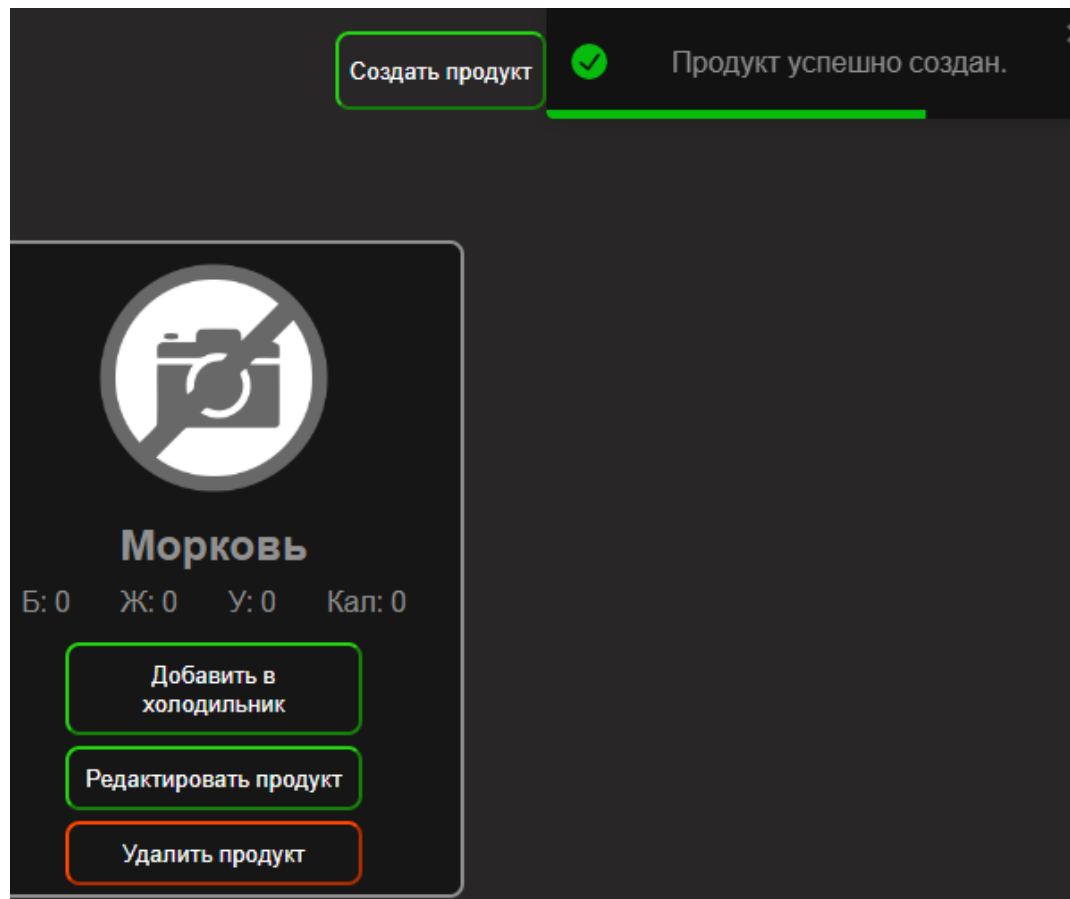


Рисунок 4.10 – Успешное завершение операции

Соответственно, при возникновении ошибки пользователь получит уведомление с причиной возникновения ошибки. Рисунок 4.11 показывает пример ошибки.

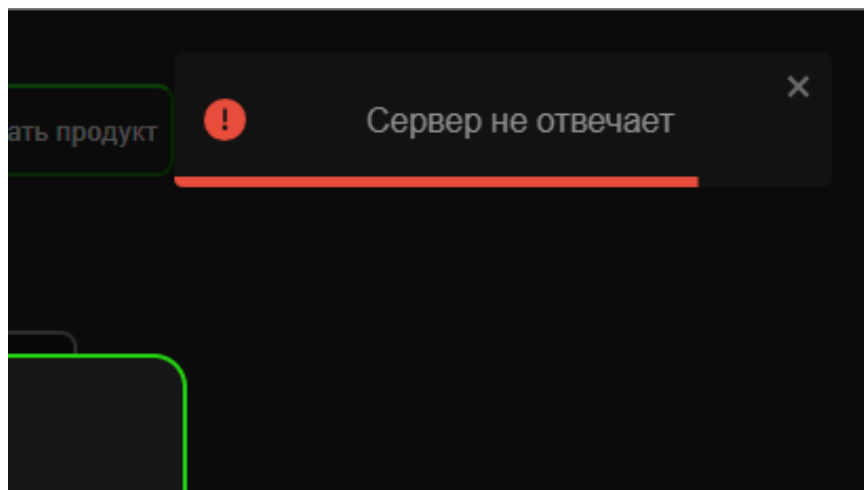


Рисунок 4.11 –Пример уведомления об ошибке

В приложении может возникнуть ситуация, когда рецептов/продуктов в приложении очень много, что затрудняет поиск. Для решения этой проблемы на каждой странице со списками добавлен поиск по названиям. На рисунке 4.12 пример поискового запроса для продуктов.

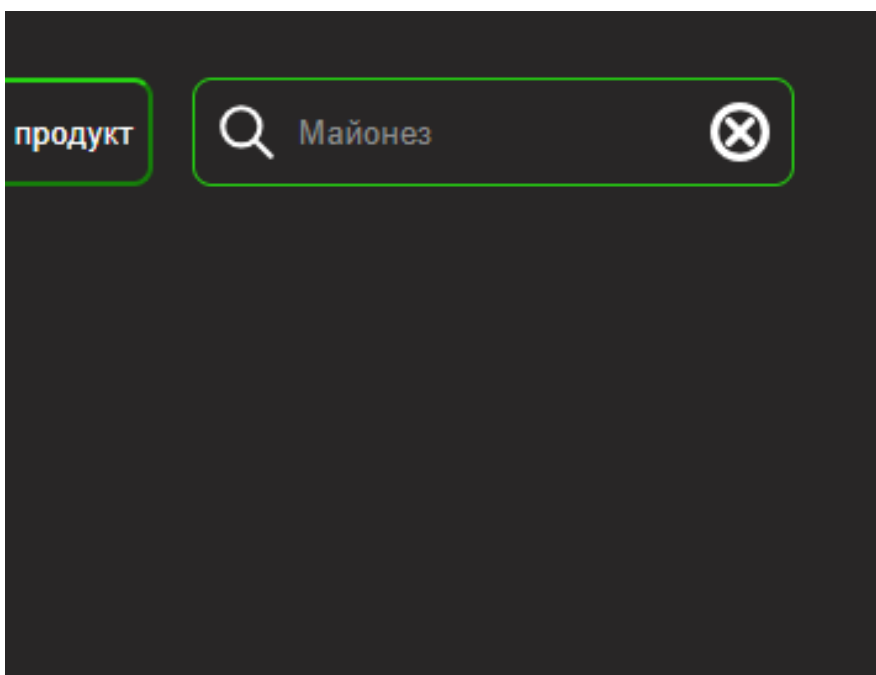


Рисунок 4.12 –Пример поискового запроса

Из-за наличия соответствующей иконки в поле поиска пользователь сразу же поймет, что это поле для поиска.

4.1 Выводы по разделу

В разделе, посвященном тестированию, были рассмотрены основные процессы приложения, на которых пользователь может чаще всего совершать ошибки, и протестирована правильность работы приложения. Во всех необходимых местах добавлена валидация и проверка значений, а также уведомления о завершении операции.

После проведения тестирования можно сделать вывод, что покрываемый функционал работает правильно и никаких ошибок в ходе исполнения замечено не было. Все функции приложения отрабатывают корректно, а в случае, если возникает ошибка пользователь получает соответствующее уведомление. В свою очередь это говорит о грамотном проектировании приложения.

5 Руководство пользователя

Пользователям, использующим приложение впервые, очень часто непонятны некоторые аспекты использования приложения. Руководство предназначено для обеспечения подробной информации о функциональности и способах использования данного проекта. В нем находятся инструкции по развертыванию и использованию приложения от разных ролей.

5.1 Руководство по развертыванию приложения

Для успешного развертывания приложения необходимо зайти в директорию с файлом «docker-compose.yaml», открыть терминал или PowerShell в этой директории и прописать команду для развертывания контейнеров в системе. При этом, если выполнение идет в среде Windows, то требуется дополнительно установить WSL и Docker Desktop. Через некоторое время приложение развернется в докере. Результат развертывания можно увидеть на рисунке 5.1.

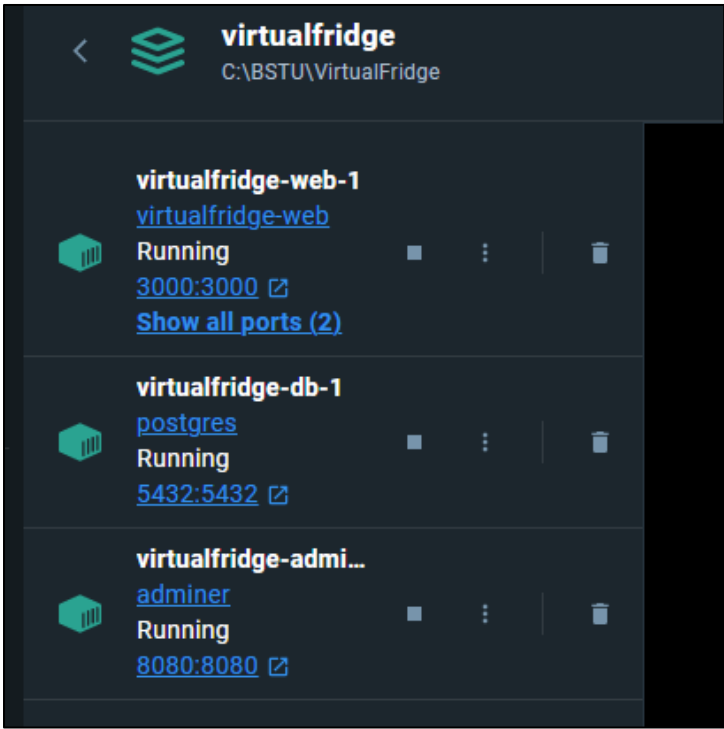


Рисунок 5.1 – Контейнер в Docker Desktop

Как видно, все 3 контейнера запущены, критических ошибок при запуске обнаружено не было, значит развертывание прошло успешно.

				БГТУ 05.00.ПЗ			
	ФИО	Подпись	Дата				
Разраб.	Гончаревич Е. В.			5 Руководство пользователя		Лит.	Лист
Пров.	Сазонова Д. В.					У	1
							6
Н. контр.	Нистюк О. А.					74218008, 2024	
Утв.	Блинова Е. А.						

5.2 Руководство для не авторизованного пользователя

У неавторизованного пользователя только две опции: зарегистрироваться или авторизоваться. При этом, для доступа к некоторым точкам сервера требуется обязательная авторизация. Рисунок 5.2 демонстрирует приватность методов на сервере.

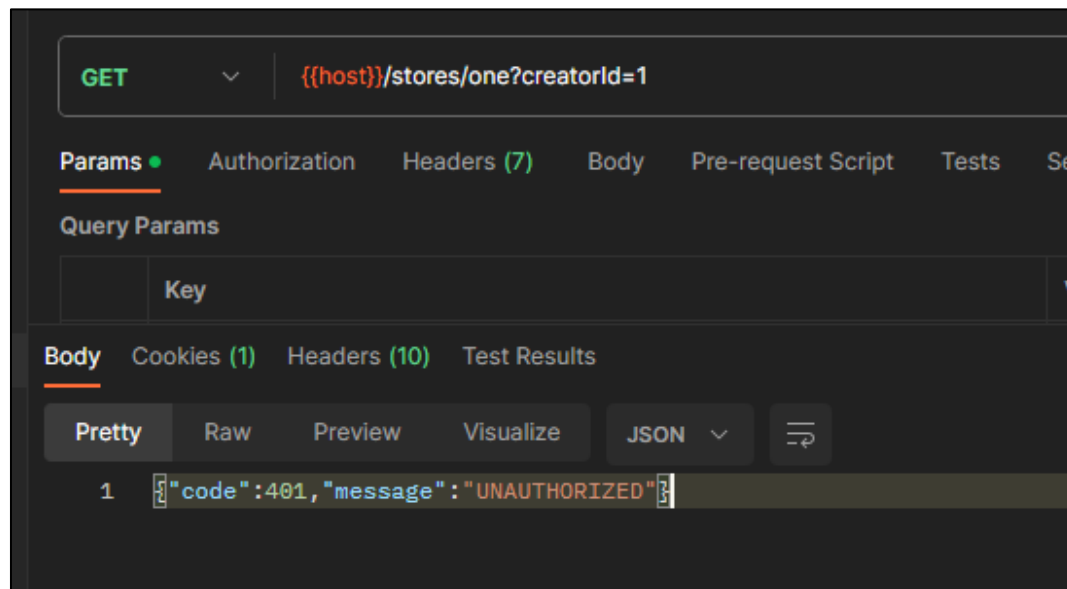


Рисунок 5.2 – Запрос, требующий авторизации

Как видно на рисунке 5.2, запрос на получение хранилища не прошел, так как в заголовках не указан токен авторизации.

Не все методы являются приватными. Например, регистрация нового пользователя или просмотр рецептов. Ведь не все хотят быть активными участниками приложения. Многие хотят просто посмотреть, что бы приготовить сегодня.

Приватными же методами являются создание/удаление рецептов/продуктов, или же создание хранилища и отслеживание продуктов в нем. Создание рецептов не может быть общедоступно, так как возможно столкновение с проблемой спама, когда база данных обрастет безобразными записями. В случае авторизованного доступа злоумышленника можно быстро вычислить и заблокировать доступ к приложению. Это один из базовых способов обеспечения безопасности приложения.

5.3 Руководство для авторизованного пользователя

Авторизованному пользователю при входе доступно три вкладки: рецепты, чек-листы и хранилище. На вкладке «Рецепты» пользователю доступны те рецепты, которые администраторы сделали видимыми. Пользователь способен с помощью поиска найти рецепт, который ему угоден. Гибкая система поиска по названию, типу блюда, калориям покрывает все нужды пользователя, предлагая быстрый поиск.

Сам рецепт представляет собой карточки с кратким описанием и его содержанием. Карточку рецепта можно развернуть и посмотреть подробнее. При раскрытии карточки пользователь может видеть пошаговое приготовление, нужные продукты, количество БЖУ продуктов.

Пользователь может нажать рецепт, после чего на том появится зеленая рамка. После этого станет активной кнопка «Сформировать список покупок», которая перенесет все продукты из рецепта в чек-лист, за вычетом тех, что уже имеются у пользователя, или без вычета, если требуется.

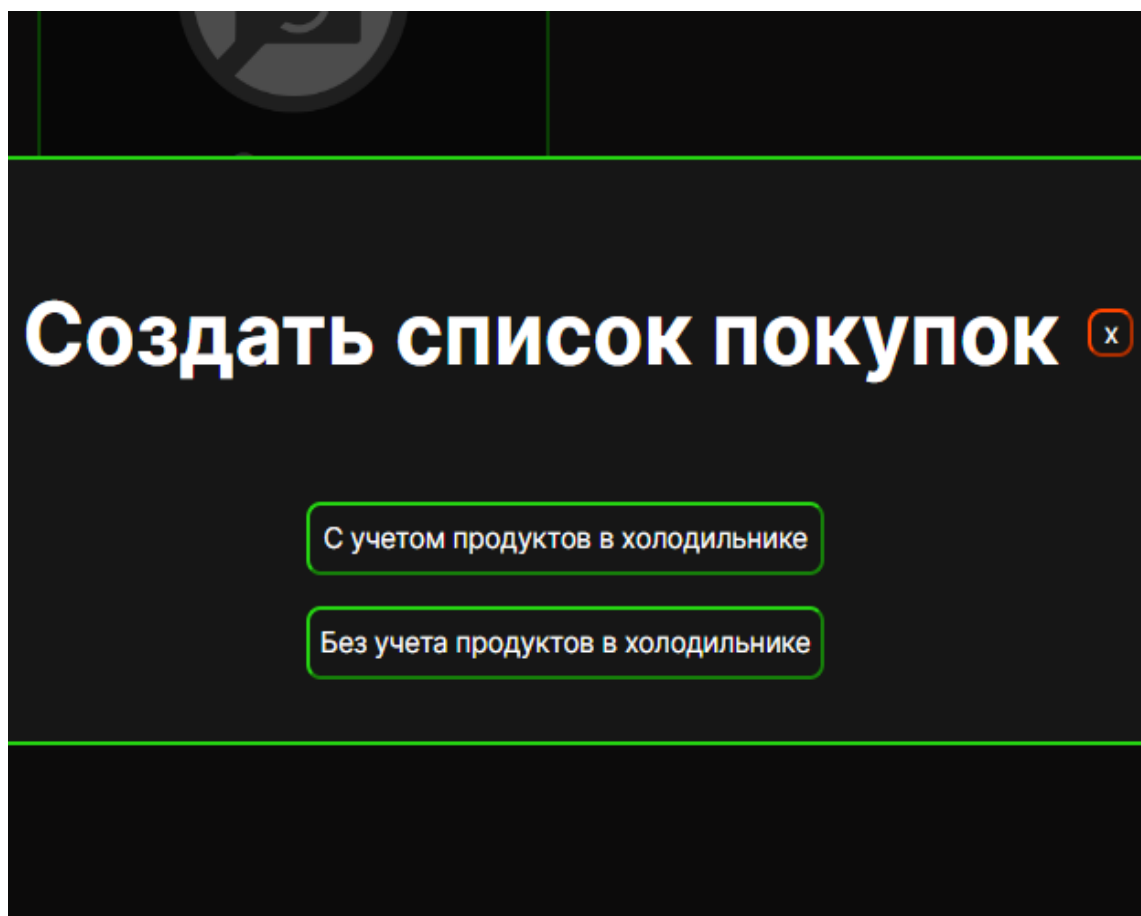


Рисунок 5.3 – Пример формирования списка продуктов

На вкладке «Хранилище» пользователю показываются все его продукты, которые он получил тем или иным образом.

В каждой карточке с продуктом содержится название продукта, его количество, единицы измерения, дата, когда продукт испортится (если пользователь указал ее или если указан средний срок хранения), а также калории, белки, жиры, углеводы.

В самой карточке есть кнопки редактировать и удалить для соответствующих действий. В режиме редактирования есть возможность разделять один и тот же продукт на несколько частей и указывать, сколько может храниться та или иная часть. Это покрывает ситуацию, когда один тип продукта был приобретен в разное время и/или с разным сроком годности.

Помимо карточек с продуктами, по левой стороне есть всплывающая форма для добавления нового продукта. В форме присутствует поле для поиска продуктов по базе и после его выбора, последний появляется внизу формы, где можно отредактировать его количество, срок хранения и т. д.

Так же над карточками присутствует поле для поиска продуктов в хранилище по названию. Вид страницы «Хранилище» показан на рисунке 5.4.

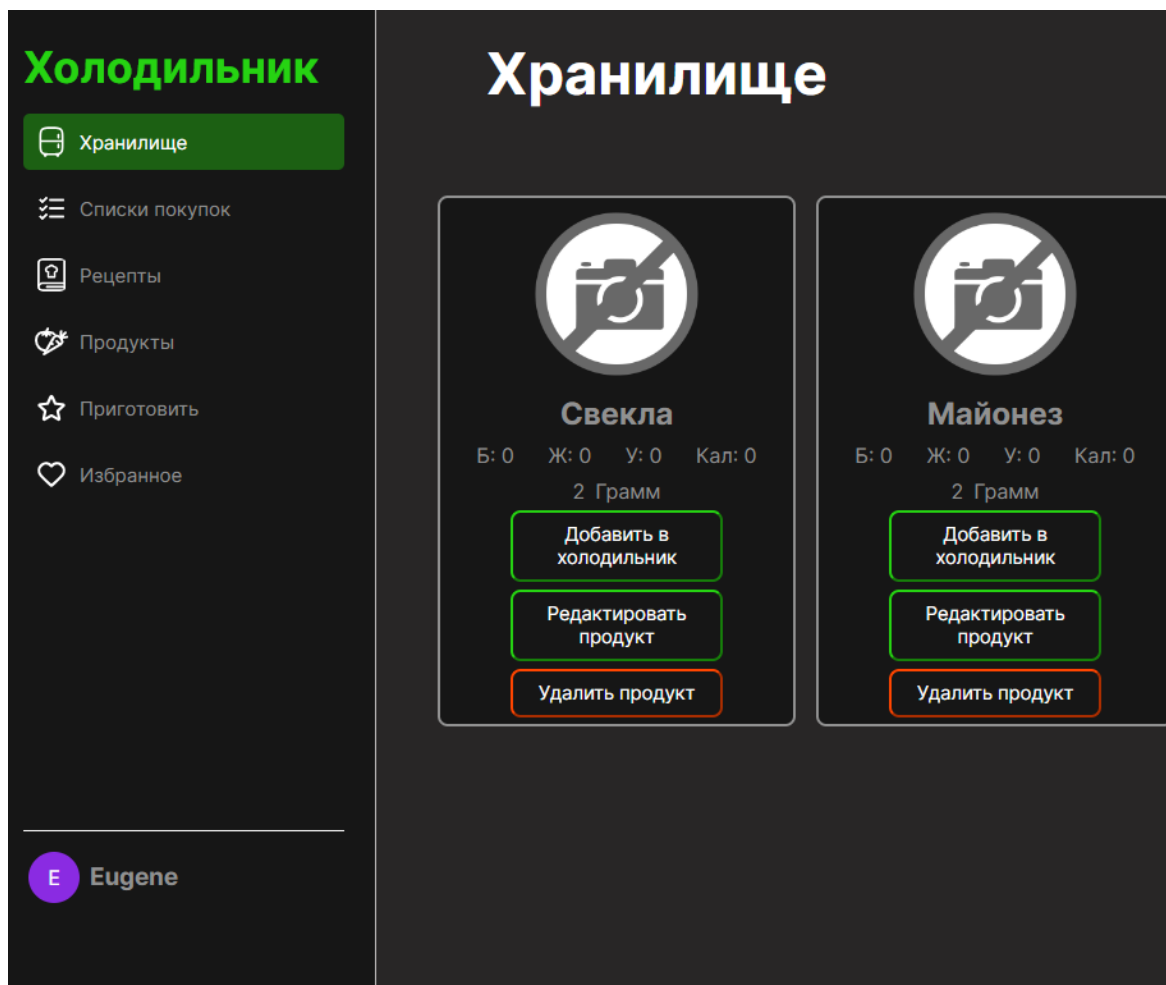


Рисунок 5.4 – Вид страницы «Хранилище»

На странице «Чек-листы» пользователь видит свои чек-листы. Каждый из них отображает дату создания и статус покупки. Так же есть две кнопки, которые дают посмотреть информацию о чек-листе и подтвердить покупку.

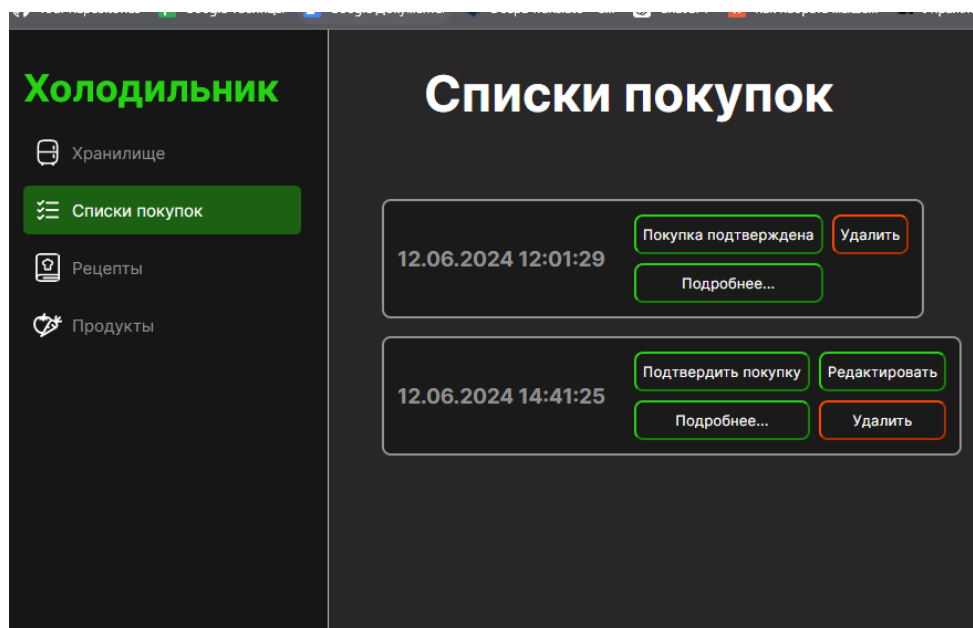


Рисунок 5.5 – Страница «Список покупок»

После подтверждения покупки все содержимое добавляется в хранилище.

5.4 Руководство для администратора/модератора

Администратору при входе в приложение доступны следующие вкладки: «Пользователи», «Продукты» и «Рецепты». Все они сводятся к одному принципу: показать весь список сущностей, соответствующих названию вкладки, которые можно редактировать или удалить. И показать форму для создания сущности. На рисунке 5.6 показана страница «Пользователи».

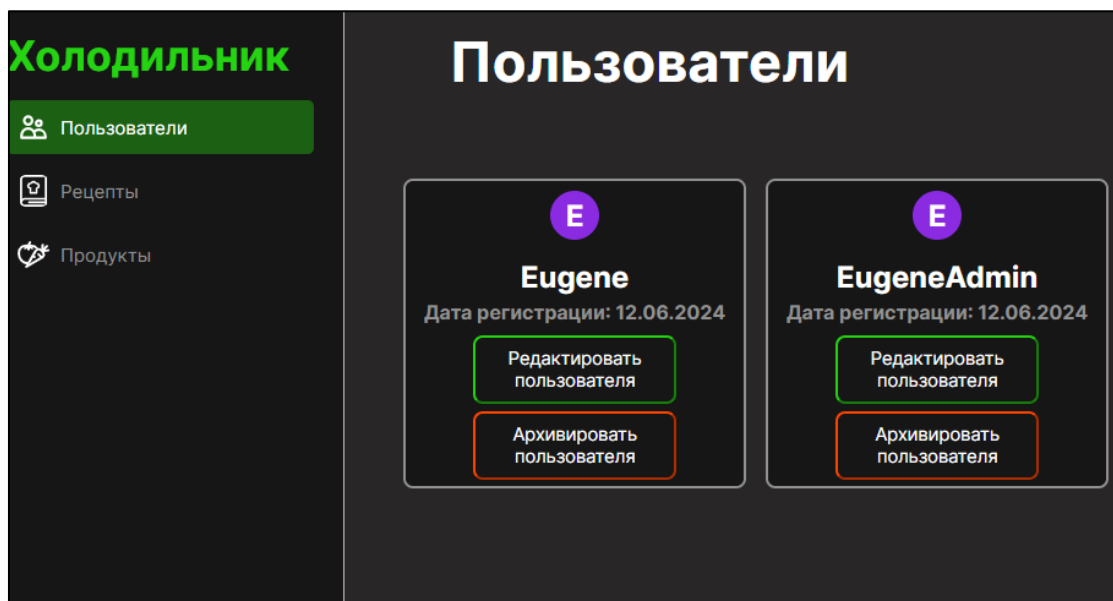


Рисунок 5.6 – Страница «Пользователи»

На этой странице присутствует специальный фильтр «Показать архив». По нажатию на него появляется список архивированных пользователей.

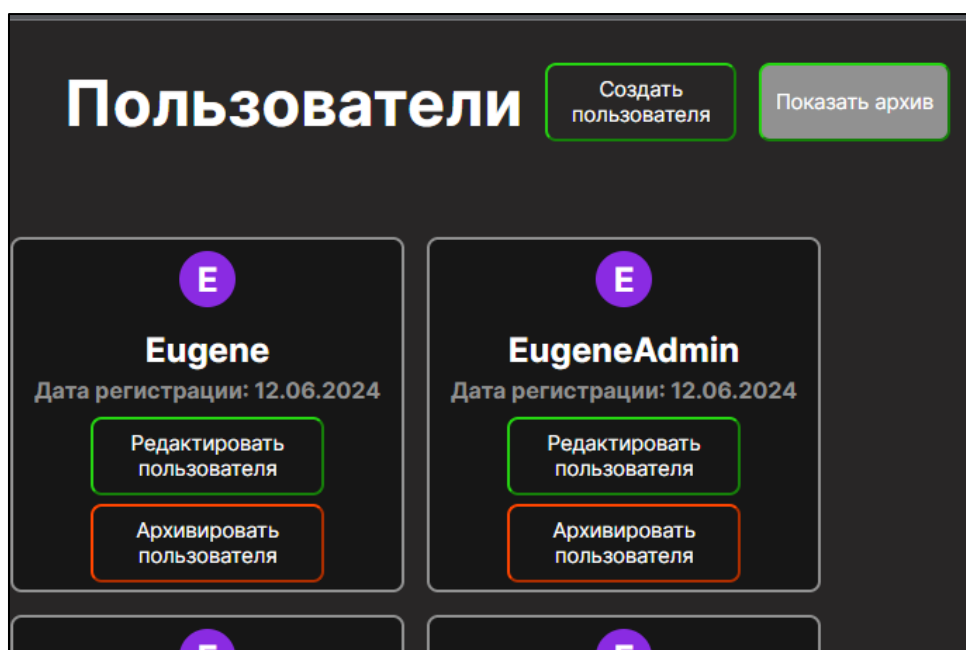


Рисунок 5.7 – Режим «Показать архив?»

На сервере не реализовано удаление записей, так как злоумышленник, получивший доступ, может удалить все данные и сломать все приложение. Ущерб станет критическим. У модератора, в отличие от администратора, отсутствует возможность влияния на базу пользователей. В связи с этим, в интерфейсе у модератора отсутствует вкладка «Пользователи», что представлено на рисунке 5.8.

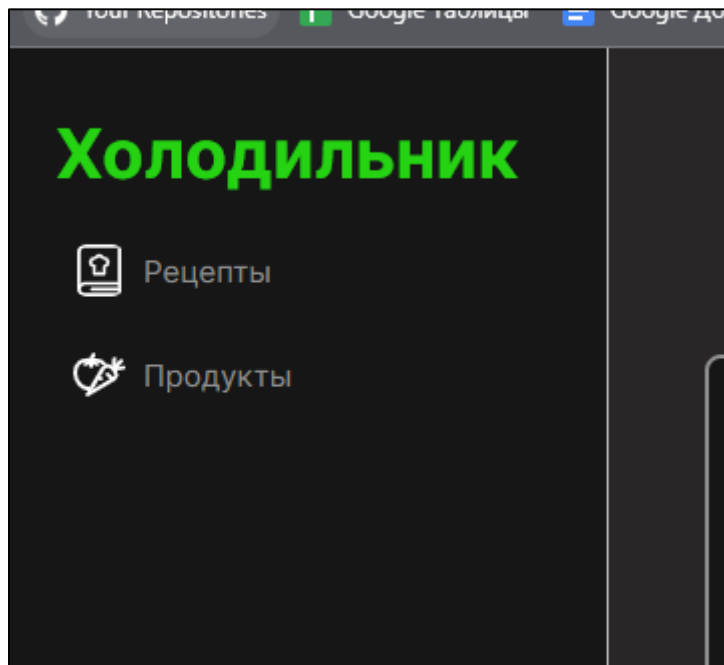


Рисунок 5.8 – Панель навигации модератора

Таким образом было описано руководство для роли администратора.

5.5 Выводы по разделу

В данном разделе были представлены руководства для всех пользователей приложений: неавторизованный пользователь, авторизованный пользователь, администратор/модератор. В самом начале описан способ развертывания приложения на различных системах, таких как Linux и Windows. В разделе с развертыванием подробно описаны действия для запуска приложения в любой системе.

Для авторизованного пользователя были подробно описаны возможности пополнения и редактирования своего хранилища, поиск продуктов и рецептов, а также добавление собственных рецептов.

Для неавторизованного пользователя описано управление приложением с ограниченными функционалом, а именно регистрация и авторизация на начальном экране.

Для администратора и модератора приложения были расписаны возможности по управлению приложением, такие как создание, редактирование и архивирование пользователей. Так же подробно описана работа с рецептами и продуктами, возможности создания и удаления.

6 Технико-экономическое обоснование проекта

Основной целью экономического раздела является экономическое обоснование целесообразности разработки программного средства, представленного в дипломном проекте. В рамках данного раздела необходимо определить затраты, произведенные на всех стадиях разработки описанного программного модуля, а также полную стоимость программного средства.

6.1 Общая характеристика разрабатываемого программного средства

При выполнении дипломного проекта было разработано приложение, предназначенное для формирования списка покупок на основе рецептов, а также система выбора рецептов и продуктов.

Цель приложения – предоставить пользователю возможность отслеживать состояния своего хранилища продуктов (так называемого виртуального холодильника), а также выбор и подбор рецептов, и автоматический подсчет продуктов для приобретения. После того, как пользователь зашел в свою учетную запись посредством авторизации, он попадает на главную страницу и ему становятся доступны все имеющиеся функции приложения.

Средство позволяет просматривать базу самых различных рецептов, как созданных администраторами системы, так и от обычных пользователей.

В приложении присутствует возможность создания и редактирования своих собственных рецептов и продуктов.

Также в системе имеется возможность добавления рецептов в избранное для более быстрого доступа к понравившимся рецептам или просто для того, чтобы не потерять любимый рецепт среди других.

Во время разработки приложения использовались технологии JavaScript, Bun, Docker-compose, Elysia, React, MUI. Непосредственно в разработке продукта использовались библиотека React Query.

Стратегия монетизации предполагает платную месячную подписку на дополнительный функционал в приложении. В рамках данного раздела необходимо определить затраты на всех стадиях разработки программного средства.

Необходимо выполнить расчет экономии основных видов ресурсов в связи с использованием разработанного программного средства на основе успеха и монетизации приложений аналогов.

					БГТУ 06.00.ПЗ у		
		ФИО	Подпись	Дата	6 Технико-экономическое обоснование проекта		
Разраб.		Гончаревич Е. В.					
Провер.		Сазонова Д. В.					
Консульт.		Соболевский А.С.					
Н. контр.		Нистюк О. А.					
Утв.		Блинова Е. А.					
					Лит.	Лист	Листов
						1	9
					74218008, 2024		

6.2 Исходные данные для проведения расчетов

Исходные данные для расчета приведены в таблице 6.1.

Таблица 6.1 – Исходные данные для расчета

Наименование показателя	Условные обозначения	Норматив
Численность разработчиков, чел	$Ч_p$	1
Коэффициент изменения скорости обработки информации, ед.	$K_{ск}$	0,6
Норматив дополнительной заработной платы, %	$H_{дз}$	14
Норматив общепроизводственных и общехозяйственных расходов, %	$H_{обп, обх}$	20
Ставка отчислений в Фонд социальной защиты населения, %	$H_{фсзн}$	34
Ставка отчислений по обязательному страхованию в БРУСП «Белгосстрах», %	$H_{бгс}$	0,6
Цена одного машино-часа, руб.	$C_{мч}$	0,06
Норматив прочих прямых затрат, %	$H_{пз}$	20
Норматив расходов на сопровождение и адаптацию, %	$H_{рса}$	17,0
Ставка НДС, %	$H_{ндс}$	0
Налог на прибыль, %	$H_{п}$	0

Источниками исходных данных для данных расчетов выступают действующие законы и нормативно-правовые акты. Для ПВД НДС и налог на прибыль 0%.

6.3 Методика обоснования цены

В современных рыночных экономических условиях программное средство (ПС) выступает преимущественно в виде продукции организаций, представляющей собой функционально завершенные и имеющие товарный вид, реализуемые покупателям по рыночным отпускным ценам. Все завершенные разработки являются научно-технической продукцией.

Широкое применение вычислительных технологий требует постоянного обновления и совершенствования программных средств. Выбор эффективных проектов программных средств связан с их экономической оценкой и расчетом экономического эффекта, который может определяться как у разработчика, так и у обычного пользователя системы.

У разработчика экономический эффект выступает в виде чистой прибыли от реализации программных средств, остающейся в распоряжении организации, а у пользователя – в виде экономии трудовых, материальных и финансовых ресурсов, получаемой за счет:

- снижения трудоемкости расчетов и алгоритмизации программирования и отладки программ;

- сокращения расходов на оплату машинного времени и других ресурсов на диагностику и отладку программ;
- снижения расходов на материалы;
- ускорение ввода в эксплуатацию новых систем;
- улучшения показателей основной деятельности в результате использования передовых программных средств.

Стоимостная оценка программных средств у разработчиков предполагает определение затрат, что включает следующие статьи:

- заработная плата исполнителей – основная и дополнительная;
- отчисления в фонд социальной защиты населения;
- отчисления по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний;
- расходы на оплату машинного времени;
- прочие прямые затраты;
- все накладные расходы.

На основании затрат рассчитывается себестоимость и отпускная цена конечного программного средства.

6.3.1 Объем программного средства

В таблице 6.2 указаны в укрупненном виде все работы, реально выполненные для создания, указанного в дипломной работе программного средства и количество рабочих дней, реально потраченных для выполнения этих работ.

Таблица 6.2 – Затраты рабочего времени на разработку ПС

Содержание работ	Затраты рабочего времени, дней
Построение диаграмм для проектирования дипломного проекта	6
Создание базы данных	3
Разработка серверной части программного средства	15
Развертывание сервера	1
Разработка клиентской части программного средства	21
Тестирование работоспособности средства	5
Написание руководства пользователя	2
Всего	53

Результат по данной таблице будет использовано далее для расчетов.

6.3.2 Основная заработная плата

Для определения величины основной заработной платы, было проведено исследование величин заработных плат для специалистов в программировании на JavaScript. В итоге было установлено, что средняя месячная заработная плата на позиции junior составляет 1600 рублей.

Согласно таблице 6.1, проект разрабатывался одним человеком на протяжении 53 дней, что соответствует 2,5 месяца, поскольку в среднем рабочих дней в месяце 21, отсюда суммарное количество дней разработки делим на усредненное количество рабочих дней в месяце. Таким образом, основная заработная плата будет рассчитываться по формуле 6.1.

$$C_{\text{оз}} = T_{\text{разл}} \cdot C_{\text{зп1}}, \quad (6.1)$$

где $C_{\text{оз}}$ – основная заработная плата, руб.;

$T_{\text{разл}}$ – время разработки, месяцев;

$C_{\text{зп1}}$ – средняя месячная заработная плата JavaScript разработчика.

$$C_{\text{оз}} = 2,5 \cdot 1600 = 4000 \text{ руб.}$$

В дальнейшем для других расчетов используется основная заработная плата, рассчитанная по указанной выше методике.

6.3.3 Дополнительная заработная плата

Дополнительная заработная плата на конкретное программное средство включает выплаты, предусмотренные законодательством о труде, и определяется по нормативу в процентах к основной заработной плате по формуле 6.2.

$$C_{\text{дз}} = \frac{C_{\text{оз}} \cdot N_{\text{дз}}}{100}, \quad (6.2)$$

где $C_{\text{оз}}$ – основная заработная плата, руб.;

$N_{\text{дз}}$ – норматив дополнительной заработной платы, %.

$$C_{\text{дз}} = 4000 \cdot 14 / 100 = 560 \text{ руб.}$$

6.3.4 Отчисления в Фонд социальной защиты населения

Отчисления в Фонд социальной защиты населения (ФСЗН) и по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний определяются в соответствии с действующими законодательными актами по нормативу в процентном отношении к фонду основной и дополнительной зарплаты исполнителей.

Отчисления в Фонд социальной защиты населения вычисляются по формуле 6.3.

$$C_{\text{фсзн}} = \frac{(C_{\text{оз}} + C_{\text{дз}}) \cdot N_{\text{фсзн}}}{100}, \quad (6.3)$$

где $C_{\text{оз}}$ – основная заработная плата, руб.;

$C_{\text{дз}}$ – дополнительная заработная плата на конкретное ПС, руб.;

$N_{\text{фсзн}}$ – норматив отчислений в Фонд социальной защиты населения, %.

Отчисления в БРУСП «Белгосстрах» вычисляются по формуле 6.4.

$$C_{\text{бгс}} = \frac{(C_{\text{оз}} + C_{\text{дз}}) \cdot H_{\text{бгс}}}{100}, \quad (6.4)$$

$$C_{\text{фсзн}} = \frac{(4000 + 560) \cdot 34}{100} = 1550,40 \text{ руб.}$$

$$C_{\text{бгс}} = \frac{(4000 + 560) \cdot 0,6}{100} = 27,36 \text{ руб.}$$

Таким образом, общие отчисления в БРУСП «Белгосстрах» составили 27,36 руб., а в фонд социальной защиты населения – 1550,4 руб.

6.3.5 Расходы на материалы

Расходов на материалы не было.

6.3.6 Прочие прямые затраты

Сумма прочих затрат $C_{\text{пз}}$ определяется как произведение основной заработной платы исполнителей на конкретное программное средство $C_{\text{оз}}$ на норматив прочих затрат в целом по организации $H_{\text{пз}}$, и находится по формуле 6.5.

$$C_{\text{пз}} = \frac{C_{\text{оз}} \cdot H_{\text{пз}}}{100}, \quad (6.5)$$

Все данные необходимые для вычисления есть, поэтому можно определить сумму полную прочих затрат.

$$C_{\text{пз}} = 4000 \cdot 20 / 100 = 800 \text{ руб.}$$

6.3.7 Накладные расходы

Сумма накладных расходов $C_{\text{обп,обх}}$ – произведение основной заработной платы исполнителей на конкретное программное средство $C_{\text{оз}}$ на норматив накладных расходов в целом по организации $H_{\text{обп,обх}}$, по формуле 6.6.

$$C_{\text{обп,обх}} = \frac{C_{\text{оз}} \cdot H_{\text{обп,обх}}}{100}. \quad (6.6)$$

Все данные необходимые для вычисления есть, поэтому можно определить полную сумму накладных расходов.

$$C_{\text{обп,обх}} = 4000 \cdot 20 / 100 = 800 \text{ руб.}$$

6.3.8 Сумма расходов на разработку программного средства

Сумма расходов на разработку программного средства C_p определяется как сумма основной и дополнительной заработной платы исполнителей на конкретное программное средство, отчислений на социальные нужды, расходов на материалы, расходов на оплату машинного времени, полной суммы прочих затрат и суммы накладных расходов, по формуле 6.7.

$$C_p = C_{оз} + C_{дз} + C_{фсзн} + C_{бгс} + C_{пз} + C_{обп,обх}. \quad (6.7)$$

Все данные необходимые для вычисления есть, поэтому можно определить сумму расходов на разработку программного средства.

$$C_p = 4000 + 560 + 1550,40 + 27,36 + 800 + 800 = 7\,737,76 \text{ руб.}$$

Сумма расходов на разработку программного средства была вычислена на основе данных, рассчитанных ранее в данном разделе, и составила 7 737,76 рублей.

6.3.9 Расходы на сопровождение и адаптацию

Сумма расходов на сопровождение и адаптацию программного средства $C_{рса}$ определяется как произведение суммы расходов на разработки на норматив расходов на сопровождение и адаптацию $H_{рса}$, и находится по формуле 6.8.

$$C_{рса} = \frac{C_p \cdot H_{рса}}{100}, \quad (6.8)$$

$$C_{рса} = 7\,737,76 \cdot 17 / 100 = 1315,42 \text{ руб.}$$

Сумма расходов на сопровождение и адаптацию была вычислена на основе данных, рассчитанных ранее в данном разделе. Все проведенные выше расчеты необходимы для вычисления полной себестоимости проекта.

6.3.10 Полная себестоимость

Полная себестоимость $C_{п}$ определяется как сумма двух элементов: суммы расходов на разработку приложения C_p и суммы расходов на сопровождение и адаптацию программного средства $C_{рса}$.

Полная себестоимость $C_{п}$ вычисляется по формуле 6.9.

$$C_{п} = C_p + C_{рса}, \quad (6.9)$$

$$C_{п} = 7737,76 + 1315,42 = 9053,18 \text{ руб.}$$

Полная себестоимость программного средства была вычислена на основе данных, рассчитанных ранее в данном разделе.

6.3.11 Определение цены, оценка эффективности

Рассматриваемое программное средство разрабатывается для потребления другими лицами. После анализа способа монетизации продуктов-аналогов, был выбран способ монетизации «расширенная подписка на месяц».

Продукты-аналоги, наиболее близкие теме дипломного проекта:

– «Bring» – веб-приложение, в большинстве своем выполняет функцию отслеживания содержимого холодильника [1];

– «Listonic» – приложение, перенявшее характеристики предыдущего, но с возможностью поиска и подбора рецептов [3].

По балловому методу цены рассчитываются на основе оценок значимости различных параметров качества программного продукта для потребителей. Сначала каждому из тех параметров, по которым судят о качестве продукции, присваивается значение удельного веса, которое характеризует, насколько данный параметр важен для потребителя по сравнению с другими. Далее каждому параметру базового программного продукта присваивается определенное число баллов, суммирование которых с учетом весовых коэффициентов дает интегральный показатель качества продукции конкурента ИК (формула 6.10).

$$ИК = \Sigma (K_i \cdot ПК_i) / \Sigma K_i \text{ при } \Sigma K_i = 1, \quad (6.10)$$

где K_i – весовой коэффициент, отражающий значимость i -го показателя;

$ПК_i$ – число баллов, присвоенное i -му показателю продукта конкурента.

Показатель качества продукта ИР рассчитывается по формуле 6.11.

$$ИР = \Sigma (K_i \cdot ПР_i) / \Sigma K_i \text{ при } \Sigma K_i = 1, \quad (6.11)$$

где $ПР_i$ – число баллов, присвоенное i -му показателю качества рассматриваемого программного продукта.

Были выбраны следующие характеристики для показателей качества рассматриваемого программного продукта и программного продукта конкурента:

– дизайн – то, как приложение выглядит внешне, привлекательность, очень важна для любого современного приложения;

– юзабилити – насколько приложение удобно в использовании;

– функциональность – количество инноваций, внедряемых в приложение;

– отсутствие багов – наличие несущественных ошибок в работе.

Расчет показателей качества базового и нового продуктов, согласно балловому методу, приводится в таблицах 6.3.

Таблица 6.3 – Показатели качества рассматриваемого программного продукта и программного продукта конкурента

Показатель качества	Весовой коэффициент	Мое приложение	Bring	Listonic
Дизайн	0,3	8	6	4
Юзабилити	0,4	7	7	6

Продолжение таблицы 6.3

Показатель качества	Весовой коэффициент	Мое приложение	Bring	Listonic
Функциональность	0,2	7	7	5
Отсутствие багов	0,1	8	7	7
Всего	1	7,4	6,7	5,3

Расчет прогнозного количества регистраций в системе K_i при монетизации методом расширенной месячной подписки на дополнительный функционал приложения, рассчитывается по формуле 6.12.

$$K_i = (K_0 / T_0 \cdot \text{ИР}) / \text{ИК} \quad (6.12)$$

где K_0 – количество регистраций в ПС конкурента;

T_0 – количество лет существования приложения;

ИР – показатель рассматриваемого программного продукта;

ИК – показатель программного продукта конкурента.

$$K_1 = (500\,000 / 6 \cdot 7,4) / 6,7 = 92\,039,80 \text{ (регистраций в год)},$$

$$K_2 = (200\,000 / 4 \cdot 7,4) / 5,3 = 69\,811,32 \text{ (регистраций в год)},$$

$$K = (92\,039,80 + 69\,811,32) / 2 = 80\,925,56 \text{ (регистраций в год)}.$$

Определение цены месячной расширенной подписки нового продукта Π_1 будет осуществляться по следующей формуле 6.13.

$$\Pi_1 = (\Pi_0 \cdot \text{ИР}) / \text{ИК}, \quad (6.13)$$

где Π_0 – цена подписки программного продукта конкурента,

ИР – показатель рассматриваемого программного продукта,

ИК – показатель программного продукта конкурента.

У приложения Bring ежемесячная подписка – 7\$ (20,51).

У Listonic – 5\$ (14,65).

$$\Pi_1 = (20,51 \cdot 7,4) / 6,7 = 22,65 \text{ рублей},$$

$$\Pi_2 = (14,65 \cdot 7,4) / 5,3 = 20,4 \text{ рублей},$$

$$\Pi = (22,65 + 20,45) / 2 = 21,55 \text{ рублей}.$$

Пользователи получают базовый функционал бесплатно, а за дополнительные опции платят. При таком раскладе обычно только 0,5 процент пользователей покупает расширенную подписку. В таком случае при среднем количестве регистраций в год, равном 80 925,56, денежные поступления от покупки подписки будут составлять $\Pi_{\text{ост.в год}} = 8\,719,73$ рублей за год.

Количество покупателей продукта необходимых для окупаемости расширения $\Pi_{\text{п}}$ вычисляется по формуле 6.14.

$$П_{\Pi} = C_{\Pi} / Ц, \quad (6.14)$$

где C_{Π} – полная себестоимость, руб.;

$П_{\text{ост.в год}}$ – денежные поступления от продажи подписки за год, руб.

$$T_{\text{кп}} = 9053,18 / 21,55 = 420 \text{ покупателей.}$$

Срок окупаемости приложения $T_{\text{ок}}$ вычисляется по формуле 6.15.

$$T_{\text{ок}} = C_{\Pi} / П_{\text{ост.в год}}, \quad (6.15)$$

где C_{Π} – полная себестоимость, руб.;

$П_{\text{ост.в год}}$ – денежные поступления от месячной подписки на дополнительный функционал приложения за год, руб.

$$T_{\text{ок}} = 9053,18 / 8719,73 = 1,03 \text{ года.}$$

6.4 Выводы по разделу

В таблице 6.4 представлены результаты расчетов для основных показателей данной главы в краткой форме.

Таблица 6.4 – Таблица экономических показателей

Наименование показателя	Значение
Время разработки, мес.	2,5
Количество программистов, чел.	1
Основная заработная плата, руб.	4000
Дополнительная заработная плата, руб.	560
Отчисления в Фонд социальной защиты населения и по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний, руб.	1550,4
Прочие прямые затраты, руб.	800
Накладные расходы, руб.	800
Себестоимость разработки программного средства, руб.	7737,42
Расходы на сопровождение и адаптацию, руб.	1 315,42
Полная себестоимость, руб.	9053,18
Годовые денежные поступления от продажи подписки, руб.	8 719,73
Срок окупаемости, лет	1,03
Количество покупателей для окупаемости, чел.	420

Разработка программного средства, осуществляемая одним программистом в течение 2,5 месяцев, при заданных условиях обойдется в 9053,18 руб. Реализации данного программного средства будет приносить годовые денежные поступления в размере 8 719,73 рублей и окупиться при покупке 420 подписок через 1,05 года. Данная таблица находится в Приложении Л.

Заключение

В ходе разработки программного продукта – веб-приложение для формирования списка покупок на основе рецептов, был проведен анализ существующих платформ и сервисов, предоставляющих данные услуги. Изучены основные аспекты интерфейса, функциональности, и бизнес-моделей, выделены положительные аспекты и решения, которые могли бы быть успешно интегрированы в создаваемое программное обеспечение. Были сформулированы основные цели и актуальность разработки сервиса.

Проектирование программного продукта предполагало выбор основных технологий, фреймворков и библиотек, разработку архитектуры системы, включая клиентскую и серверную части, а также проектирование базы данных. На этапе проектирования были определены основные алгоритмы работы программы, их визуализация в виде блок-схем, а также создана диаграмма использования приложения.

Реализация программного продукта началась с создания серверной части. Была изучена структура проекта, аутентификации и авторизации пользователей. Архитектура была основана на использовании Bun.js, а также использовании базы данных PostgreSQL для хранения информации о рецептах, пользователях и продуктах.

Клиентская часть программного продукта также была разработана. Была определена структура проекта, осуществлена реализация HTTP клиентов для общения с серверной частью, а также выбран React для построения пользовательского интерфейса и управления глобальным состоянием приложения.

Тестирование программного продукта проводилось с целью выявления и устранения возможных ошибок. Все ключевые функции приложения были протестированы на различных уровнях доступа, чтобы гарантировать корректную работу системы для всех типов пользователей.

Было составлено подробное руководство для пользователей программного продукта. Руководство содержит информацию о функциональности приложения для каждой из ролей пользователей.

В заключение была экономически обоснована стоимость разработанного дипломного проекта, а также его срок окупаемости.

				БГТУ 00.00.ПЗ			
	ФИО	Подпись	Дата				
Разраб.	Гончаревич Е.В.			Заключение	Лит.	Лист	Листов
Пров.	Сазонова Д.В.				У	1	1
					74218008, 2024		
Н. контр.	Нистюк О. А.						
Утв.	Блинова Е. А.						

Список использованных источников

1 Bring [Электронный ресурс]/Читатель– Режим доступа: <https://apps.apple.com/ru/app/bring/> – Дата доступа: 03.06.2024.

2 Listonic [Электронный ресурс]/Читатель – Режим доступа: <https://new.app.listonic.com/en> – Дата доступа: 03.06.2024.

3 Russian food [Электронный ресурс]/Читатель – Режим доступа: <https://www.russianfood.com/> – Дата доступа: 03.06.2024.

4 PostgreSQL [Электронный ресурс]/ PostgreSQL – Режим доступа: <https://www.postgresql.org/> – Дата доступа: 03.06.2024

5 React – Getting Started [Электронный ресурс] – Режим доступа: <https://reactjs.org/docs/getting-started.html> – Дата доступа: 03.06.2024

6 Понимая Docker [Электронный ресурс] – Режим доступа: <https://habr.com/ru/post/253877/> – Дата доступа: 03.06.2024

7 Axios [Электронный ресурс] – Режим доступа: <https://axios-http.com/ru/docs/intro> – Дата доступа: 03.06.2024

				БГТУ 00.00.ПЗ						
	ФИО	Подпись	Дата							
Разраб.	Гончаревич Е.В.			Список использованных источников	Лит.		Лист		Листов	
Пров.	Сазонова Д.В.				У		1	1		
					74218008, 2024					
Н. контр.	Нистюк О. А.									
Утв.	Блинова Е. А.									

ПРИЛОЖЕНИЕ А Диаграмма развертывания

ПРИЛОЖЕНИЕ Б Логическая схема базы данных

ПРИЛОЖЕНИЕ В Диаграмма вариантов использования

ПРИЛОЖЕНИЕ Г Блок-схема алгоритма добавления продукта в хранилище

ПРИЛОЖЕНИЕ Д Блок-схема алгоритма формирования списка покупок

ПРИЛОЖЕНИЕ Е Листинг моделей базы данных

```

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model User {
  id            String           @id @default(cuid())
  login         String           @unique @db.VarChar(30)
  password      String           @db.VarChar()
  role          Roles            @default(DEFAULT)
  createdAt     DateTime         @default(now())
  @db.Timestamp(6)
  isFrozen      Boolean?         @default(false)
  isBlocked     Boolean?         @default(false)
  UserToken     UserToken[]
  Storage       Storage?
  Product       Product[]
  Recipe        Recipe[]
  ChosenRecipe  SelectedRecipeForCooking[]
  FavoriteRecipe FavoriteRecipe[]
  RecipePullRequest RecipePullRequest[]
  ProductPullRequest ProductPullRequest[]
  Checklist     Checklist[]
  UserGrade     UserGrade[]

  @@index([login])
}

model UserToken {
  userId      String
  deviceId    String
  refreshToken String
  lastLoginAt DateTime? @updatedAt @db.Timestamp(6)
  user        User      @relation(fields: [userId], references: [id],
onDelete: Cascade)

  @@id([userId, deviceId])
  @@index([lastLoginAt])
}

model Storage {
  id            String           @id @default(cuid())
  creatorId     String           @unique
  title         String?         @db.VarChar(30)
  user          User             @relation(fields: [creatorId],
references: [id], onDelete: Cascade)
  StorageComposition StorageComposition[]

```

```

    @@index([creatorId], type: Hash)
}

model StorageComposition {
  purchaseDate      DateTime? @db.Timestamp(6)
  expireDate        DateTime   @default("1970-01-01T00:00:00.00Z")
  @db.Timestamp(6)
  storageId         String
  productId         String
  productQuantity   Int         @db.SmallInt
  storage           Storage     @relation(fields: [storageId], references:
[id], onDelete: Cascade)
  product           Product     @relation(fields: [productId], references:
[id], onDelete: Cascade)

  @@unique([expireDate, storageId, productId])
  @@index([storageId], type: Hash)
}

model Checklist {
  id                String                @id @default(cuid())
  creatorId         String
  createdAt         DateTime              @default(now())
  @db.Timestamp(6)
  isConfirmed       Boolean              @default(false)
  lastUpdatedAt     DateTime              @default(now()) @updatedAt
  ChecklistComposition ChecklistComposition[]
  user              User                 @relation(fields:
[creatorId], references: [id], onDelete: Cascade)

  @@index([creatorId], type: Hash)
}

model ChecklistComposition {
  checklistId       String
  productId         String
  productQuantity   Int         @db.SmallInt
  price            Decimal       @default(0) @db.Money
  currency          Currencies   @default(BYN)
  checklist         Checklist    @relation(fields: [checklistId], refer-
ences: [id], onDelete: Cascade)
  product          Product      @relation(fields: [productId], references:
[id], onDelete: Cascade)

  @@id([checklistId, productId])
}

model Product {
  id                String                @id @default(cuid())
  creatorId         String?
  title            String                @unique @db.VarChar(50)
  calories          Int                 @db.SmallInt
  protein           Int                 @db.SmallInt
  fats             Int                 @db.SmallInt

```

```

    carbohydrates      Int                @db.SmallInt
    unit                Units              @default (GRAMS)
    isOfficial          Boolean            @default (false)
    isFrozen            Boolean            @default (false)
    isRecipePossible    Boolean            @default (false)
    averageShelfLifeInDays Int?
    user                User?              @relation(fields:
[creatorId], references: [id], onDelete: SetNull)
    StorageComposition  StorageComposition[]
    ChecklistComposition ChecklistComposition[]
    RecipeComposition   RecipeComposition[]
    ProductPullRequest  ProductPullRequest[]
    Recipe              Recipe[]

    @@index([title])
}

model Recipe {
    id                String                @id @default(cuid())
    creatorId         String?
    title             String                @unique @db.VarChar(100)
    type              RecipeTypes
    description        String?              @db.VarChar(10000)
    createdAt          DateTime              @default(now())
    isPrivate          Boolean              @default(true)
    isOfficial         Boolean              @default(false)
    isFrozen           Boolean              @default(false)
    productRef         String?
    user              User?                @relation(fields:
[creatorId], references: [id], onDelete: SetNull)
    Product            Product?            @relation(fields:
[productRef], references: [id], onDelete: SetNull)
    RecipeComposition  RecipeComposition[]
    ChosenRecipe       SelectedRecipeForCooking[]
    FavoriteRecipe     FavoriteRecipe[]
    RecipePullRequest  RecipePullRequest[]
    RecipeRating       RecipeRating[]
    UserGrade          UserGrade[]

    @@index([isOfficial, type, title])
}

model RecipeComposition {
    recipeId String
    productId String
    quantity Int @db.SmallInt
    recipe      Recipe @relation(fields: [recipeId], references: [id],
onDelete: Cascade)
    product     Product @relation(fields: [productId], references: [id],
onDelete: Cascade)

    @@id([productId, recipeId])
}

```

```

model RecipeRating {
    recipeId String @id @default(cuid())
    rating    Float  @default(0)
    recipe     Recipe @relation(fields: [recipeId], references: [id],
onDelete: Cascade)

    @@index([rating])
}

model UserGrade {
    recipeId String
    userId    String
    grade     Int
    recipe     Recipe @relation(fields: [recipeId], references: [id],
onDelete: Cascade)
    user       User    @relation(fields: [userId], references: [id],
onDelete: Cascade)

    @@id([recipeId, userId])
}

model FavoriteRecipe {
    userId     String
    recipeId   String
    createdAt  DateTime @default(now())
    user       User     @relation(fields: [userId], references: [id],
onDelete: Cascade)
    recipe     Recipe   @relation(fields: [recipeId], references: [id],
onDelete: Cascade)

    @@id([recipeId, userId])
}

model SelectedRecipeForCooking {
    userId      String
    recipeId    String
    selectedAt   DateTime @default(now()) @db.Timestamp(6)
    isCooked    Boolean  @default(false)
    lastUpdatedAt DateTime @default(now()) @updatedAt
    user        User     @relation(fields: [userId], references: [id],
onDelete: Cascade)
    recipe      Recipe   @relation(fields: [recipeId], references:
[id], onDelete: Cascade)

    @@id([userId, recipeId])
}

model RecipePullRequest {
    creatorId String
    recipeId  String @unique
    status    Status @default(PENDING)
    comment   String? @db.Text
    user      User    @relation(fields: [creatorId], references: [id],
onDelete: Cascade)

```

```

    product Recipe @relation(fields: [recipeId], references: [id],
onDelete: Cascade)

    @@index([creatorId], type: Hash)
}

model ProductPullRequest {
    creatorId String
    productId String @unique
    status Status @default(PENDING)
    comment String? @db.Text
    user User @relation(fields: [creatorId], references: [id],
onDelete: Cascade)
    product Product @relation(fields: [productId], references: [id],
onDelete: Cascade)

    @@index([creatorId], type: Hash)
}

enum Roles {
    ADMIN
    DEFAULT
    GOD
}

enum Currencies {
    USD
    BYN
    RUB
}

enum RecipeTypes {
    VEGETARIAN_DISHES
    BAKING
    GARNISHES
    HOT_DISHES
    HOT_APPETIZERS
    DESSERTS
    HOME_MADE_FASTFOOD
    BREAKFAST
    PRESERVES
    CHARCOAL
    BEVERAGES
    SALADS
    SAUCES_PASTAS_AND_DRESSINGS
    SOUPS_AND_BROTHS
    DOUGH
    BREAD
    COLD_APPETIZERS
    SHISHKEBABS
    OTHER
}

enum Units {

```

```
    PORTION
    ITEMS
    GRAMS
    KILOGRAMS
    PIECES
    TABLESPOON
    TEASPOON
    LITER
    MILLILITER
    PINCH
}

enum Status {
    PENDING
    REJECTED
    ACCEPTED
}
```


ПРИЛОЖЕНИЕ Ж Листинг маршрутизатора

```

import { RecipeTypes } from '@prisma/client'
import cookie from '@elysiajs/cookie'
import { cors } from '@elysiajs/cors'
import swagger from '@elysiajs/swagger'
import { CONFIG } from '@server/config'
import { publicDBClient } from '@server/prismaClients'
import { checklists } from '@server/router/checklists'
import { products } from '@server/router/products'
import { recipes } from '@server/router/recipes'
import { storages } from '@server/router/storages'
import { users } from '@server/router/users'
import { Elysia } from 'elysia'
import { RecipeTypes } from '@prisma/client'
import {
  _delete,
  addToFavorite,
  addToSelected,
  checkRefs,
  create,
  getAll,
  getAllFavorites,
  getAllSelected,
  getMany,
  getOne,
  removeFromFavorite,
  removeFromSelected,
  update,
  updateSelected,
} from '@server/services/recipe'
import { Elysia, t } from 'elysia'
import {
  _delete,
  addToFavorite,
  addToSelected,
  checkRefs,
  create,
  getAll,
  getAllFavorites,
  getAllSelected,
  getMany,
  getOne,
  removeFromFavorite,
  removeFromSelected,
  update,
  updateSelected,
} from '@server/services/recipe'
import { Elysia, t } from 'elysia'
export const recipes = (app: RecipeRouterType) =>
  app
    .get('/one/:id', ({ params }) => getOne(params.id), {
      params: t.Object({ id: t.String() }),
    })

```

```

.get('/many', ({ query }) => getMany(query.ids), {
  query: t.Object({ ids: t.Array(t.String()) }),
})
.get('/', ({ query }) => getAll(query), {
  query: t.Object({
    cursor: t.Optional(t.String()),
    title: t.Optional(t.String()),
    take: t.Optional(t.Numeric()),
  }),
})
.post('/', ({ body }) => create(body), {
  body: t.Object({
    info: t.Object({
      creatorId: t.String(),
      title: t.String(),
      type: t.Enum(RecipeTypes),
      description: t.Optional(t.String()),
      isPrivate: t.Optional(t.Boolean()),
      isOfficial: t.Optional(t.Boolean()),
      isFrozen: t.Optional(t.Boolean()),
    }),
    composition: t.Array(
      t.Object({
        productId: t.String(),
        quantity: t.Number(),
      }), { minItems: 1 }
    ),
  }),
})
.patch('/', ({ body }) => update(body), {
  body: t.Object({
    info: t.Object({
      id: t.String(),
      title: t.String(),
      type: t.Enum(RecipeTypes),
      description: t.String(),
      isPrivate: t.Optional(t.Boolean()),
      isOfficial: t.Optional(t.Boolean()),
      isFrozen: t.Optional(t.Boolean()),
    }),
    composition: t.Optional(
      t.Array(
        t.Object({
          productId: t.String(),
          quantity: t.Number(),
        }),
        { minItems: 1 }
      )
    ),
  }),
})
.delete('/:id', ({ params }) => _delete(params.id), {
  params: t.Object({ id: t.String() }),
})

```

```

.get('/refs', ({ query }) => checkRefs(query.id), {
  query: t.Object({ id: t.String() }),
})
.group('/favorites', app =>
  app
    .get('/', ({ query }) => getAllFavorites(query), {
      query: t.Object({
        title: t.Optional(t.String()),
        userId: t.String(),
        cursor: t.Optional(
          t.Object({
            userId: t.String(),
            recipeId: t.String(),
          })
        ),
        take: t.Optional(t.Numeric()),
      }),
    })
    .post('/', ({ query }) => addToFavorite(query), {
      query: t.Object({ userId: t.String(), recipeId: t.String() }),
    })
    .delete('/', ({ query }) => removeFromFavorite(query), {
      query: t.Object({ userId: t.String(), recipeId: t.String() }),
    })
)
.group('/selected', app =>
  app
    .get('/', ({ query }) => getAllSelected(query), {
      query: t.Object({
        cursor: t.Optional(t.String()),
        userId: t.String(),
        take: t.Optional(t.Numeric()),
      }),
    })
    .post('/', ({ body }) => addToSelected(body), {
      body: t.Object({
        userId: t.String(),
        recipeId: t.String(),
        isCooked: t.Optional(t.Boolean({ default: false })),
      }),
    })
    .put('/', ({ body }) => updateSelected(body), {
      body: t.Object({
        userId: t.String(),
        recipeId: t.String(),
        isCooked: t.Boolean(),
      }),
    })
    .delete('/', ({ body }) => removeFromSelected(body),
  {

```

```

        body: t.Object({ userId: t.String(), recipeId:
t.String() })),
    ))
  ).get('/:id', ({ params }) => getOne(params.id), {
    params: t.Object({ id: t.String() }),
  })
  .get('/many', ({ query }) => getMany(query.ids), {
    query: t.Object({ ids: t.Array(t.String()) }),
  })
  .get('/', ({ query }) => getAll(query), {
    query: t.Object({
      cursor: t.Optional(t.String()),
      createdAt: t.Optional(t.Date()),
      take: t.Optional(t.Numeric()),
    }),
  })
  .post('/', ({ body }) => create(body), {
    body: t.Object({
      userId: t.String(),
      subtractStorage: t.Boolean(),
      recipesId: t.Array(t.String()),
    }),
  })
  .patch('/', ({ body }) => update(body), {
    body: t.Object({
      userId: t.String(),
      checklistId: t.String(),
      products: t.Array(
        t.Object({
          id: t.String(),
          quantity: t.Number(),
        })
      ),
    }),
  })
  .patch('/confirm', ({ body }) => confirm(body), {
    body: t.Object({
      userId: t.String(),
      checklistId: t.String(),
    }),
  })
  .delete('/drop/:id', ({ params }) => _delete(params.id), {
    params: t.Object({ id: t.String() }),
  })
  .get('/:id', ({ params }) => getOne(params.id), {
    params: t.Object({ id: t.String() }),
  })
  .get('/many', ({ query }) => getMany(query.ids), {
    query: t.Object({ ids: t.Array(t.String()) }),
  })
  .get('/', ({ query }) => getAll(query), {
    query: t.Object({
      cursor: t.Optional(t.Nullable(t.String())),
      title: t.Optional(t.String()),
      take: t.Optional(t.Numeric()),
    }),
  })

```

```

    }),
  })
  .post('/', ({ body }) => create(body), {
    body: t.Object({
      creatorId: t.String(),
      title: t.String(),
      calories: t.Number(),
      protein: t.Number(),
      fats: t.Number(),
      carbohydrates: t.Number(),
      isOfficial: t.Boolean(),
      isFrozen: t.Boolean(),
      isRecipePossible: t.Boolean(),
      averageShelfLifeInDays: t.Optional(t.Number()),
      unit: t.Enum(Units, { default: Units.GRAMS }),
    }),
  })
  .patch('/', ({ body }) => update(body), {
    body: t.Object({
      id: t.String(),
      title: t.Optional(t.String()),
      calories: t.Optional(t.Number()),
      protein: t.Optional(t.Number()),
      fats: t.Optional(t.Number()),
      carbohydrates: t.Optional(t.Number()),
      isOfficial: t.Optional(t.Boolean()),
      isFrozen: t.Optional(t.Boolean()),
      isRecipePossible: t.Optional(t.Boolean()),
      averageShelfLifeInDays: t.Optional(t.Number()),
      unit: t.Optional(t.Enum(Units, { default:
Units.GRAMS })),
    }),
  })
  .delete('/:id', ({ params }) => _delete(params.id), {
    params: t.Object({ id: t.String() }),
  })
  .get('/refs', ({ query }) => checkRefs(query.id), {
    query: t.Object({ id: t.String() }),
  })
  .get('/', ({ query }) => getInfo(query.creatorId), {
    query: t.Object({ creatorId: t.String() }),
  })
  .get('/composition', ({ query }) => getComposition(query), {
    query: t.Object({
      storageId: t.String(),
      // cursor: t.Optional(
      //   t.Object({
      //     productId: t.String(),
      //     expireDate: t.Date(),
      //     storageId: t.String(),
      //   })
      // ),
      take: t.Optional(t.Numeric()),
    }),
  })
})

```

```

        .patch('/', ({ query }) => setInfo(query), {
            query: t.Object({ storageId: t.String(), title:
t.String() })),
        })
        .patch('/composition', ({ body }) => setComposition(body), {
            body: t.Array(
                t.Object({
                    expireDate: t.Optional(t.Date({ examples: new
Date() })),
                    storageId: t.String(),
                    productId: t.String(),
                    purchaseDate: t.Optional(t.Date({ examples:
new Date() })),
                    productQuantity: t.Number(),
                })
            ),
        })
        .put('/add', ({ body }) => addProductToStorage(body), {
            body: t.Object({
                storageId: t.String(),
                productId: t.String(),
                productQuantity: t.Numeric({ minimum: 0 }),
                expireDate: t.Optional(
                    t.Date({
                        default: new Date('1970-01-
01T00:00:00.00Z'),
                    })
                ),
            }),
        })
        .delete('/remove', ({ body }) => removeProductFromStorage(body), {
            body: t.Object({
                storageId: t.String(),
                productId: t.String(),
            }),
        })
        .get('/one/:id', ({ params }) => getOne(params.id), {
            params: t.Object({ id: t.String() }),
        })
        .get('/many', ({ query }) => getMany(query.ids), {
            query: t.Object({ ids: t.Array(t.String()) }),
        })
        .get('/', ({ query }) => getAll(query), {
            query: t.Object({
                cursor: t.Optional(t.String()),
                title: t.Optional(t.String()),
                take: t.Optional(t.Numeric()),
            }),
        })
        .post('/', ({ body }) => create(body), {
            body: t.Object({
                info: t.Object({
                    creatorId: t.String(),
                    title: t.String(),
                })
            })
        })

```

```

        type: t.Enum(RecipeTypes),
        description: t.Optional(t.String()),
        isPrivate: t.Optional(t.Boolean()),
        isOfficial: t.Optional(t.Boolean()),
        isFrozen: t.Optional(t.Boolean()),
    )),
    composition: t.Array(
        t.Object({
            productId: t.String(),
            quantity: t.Number(),
        })),
        { minItems: 1 }
    ),
)),
))
.patch('/', ({ body }) => update(body), {
    body: t.Object({
        info: t.Object({
            id: t.String(),
            title: t.String(),
            type: t.Enum(RecipeTypes),
            description: t.String(),
            isPrivate: t.Optional(t.Boolean()),
            isOfficial: t.Optional(t.Boolean()),
            isFrozen: t.Optional(t.Boolean()),
        })),
        composition: t.Optional(
            t.Array(
                t.Object({
                    productId: t.String(),
                    quantity: t.Number(),
                })),
                { minItems: 1 }
            )
        ),
    )),
))
.delete('/:id', ({ params }) => _delete(params.id), {
    params: t.Object({ id: t.String() }),
})
.get('/refs', ({ query }) => checkRefs(query.id), {
    query: t.Object({ id: t.String() }),
})
.group('/favorites', app =>
    app
        .get('/', ({ query }) => getAllFavorites(query), {
            query: t.Object({
                title: t.Optional(t.String()),
                userId: t.String(),
                cursor: t.Optional(
                    t.Object({
                        userId: t.String(),
                        recipeId: t.String(),
                    })
                ),
            })
        })
    )
)

```

```

        ),
        take: t.Optional(t.Numeric()),
    )),
    ))
    .post('/', ({ query }) => addToFavorite(query), {
        query: t.Object({ userId: t.String(), recipeId: t.String() }),
    })
    .delete('/', ({ query }) => removeFromFavorite(query), {
        query: t.Object({ userId: t.String(), recipeId: t.String() }),
    })
)
.group('/selected', app =>
    app
        .get('/', ({ query }) => getAllSelected(query), {
            query: t.Object({
                cursor: t.Optional(t.String()),
                userId: t.String(),
                take: t.Optional(t.Numeric()),
            }),
        })
        .post('/', ({ body }) => addToSelected(body), {
            body: t.Object({
                userId: t.String(),
                recipeId: t.String(),
                isCooked: t.Optional(t.Boolean({ default: false })),
            }),
        })
        .put('/', ({ body }) => updateSelected(body), {
            body: t.Object({
                userId: t.String(),
                recipeId: t.String(),
                isCooked: t.Boolean(),
            }),
        })
        .delete('/', ({ body }) => removeFromSelected(body), {
            body: t.Object({ userId: t.String(), recipeId: t.String() }),
        })
    )
.onError(e => console.log(e))
.use(cors({ credentials: true, origin: 'localhost:5173' }))
.use(cookie())
.get('/one', ({ query }) => getOne(query.login), {
    query: t.Object({
        login: t.String(),
    }),
})
.get('/', ({ query }) => getAll(query), {
    query: t.Object({
        take: t.Optional(t.Numeric({ minimum: 1 })),
    }),
})

```



```

        cursor: t.Optional(t.Nullable(t.String())),
        login: t.Optional(t.String()),
    })),
    })
    .post('/create', ({ body }) => create(body), {
        body: t.Object({
            login: t.String(),
            password: t.String(),
            role: t.Enum(Roles, { default: Roles.DEFAULT }),
            isBlocked: t.Optional(t.Boolean({ default:
false })),
            isFrozen: t.Optional(t.Boolean({ default: false })),
            deviceId: t.String(),
        })),
    })
    .patch('/update', ({ body }) => update(body), {
        body: t.Object({
            id: t.String(),
            login: t.Optional(t.String()),
            password: t.Optional(t.String()),
            role: t.Optional(t.Enum(Roles, { default:
Roles.DEFAULT })),
            isBlocked: t.Optional(t.Boolean({ default:
false })),
            isFrozen: t.Optional(t.Boolean({ default: false })),
        })),
    })
    .delete('/:id', ({ params }) => _delete(params.id), {
        params: t.Object({ id: t.String() }),
    })
    .post(
        '/signin',
        async ({ body, cookie: { refreshToken } }) => {
            const { user, ...tokens } = await signin(body)
            refreshToken.set({
                value: tokens.accessToken,
                httpOnly: true,
            })
            return { user, refreshToken: tokens.refreshToken }
        },
        {
            body: t.Object({
                password: t.String(),
                login: t.String(),
                deviceId: t.String(),
            }),
        }
    )
    .post(
        '/signup',
        async ({ body, cookie: { refreshToken } }) => {
            const { user, ...tokens } = await signup(body)
            refreshToken.set({
                value: tokens.accessToken,

```

```

        httpOnly: true,
    })
    return { user, refreshToken: tokens.refreshToken }
},
{
    body: t.Object({
        password: t.String(),
        login: t.String(),
        deviceId: t.String(),
        role: t.Optional(t.Enum(Roles, { default:
Roles.DEFAULT })),
        isBlocked: t.Optional(t.Boolean({ default:
false })),
        isFrozen: t.Optional(t.Boolean({ default:
false })),
    }),
})
)
.delete(
    '/signout',
    async ({ body, cookie: { refreshToken } }) => {
        await signout(body)
        refreshToken.remove()
    },
    {
        body: t.Object({
            userId: t.String(),
            deviceId: t.String(),
        }),
    }
)

type RecipeRouterType = Elysia<
    '/recipes',
    false,
    {
        decorator: NonNullable<unknown>
        store: NonNullable<unknown>
        derive: NonNullable<unknown>
        resolve: NonNullable<unknown>
    },
    { type: NonNullable<unknown>; error: NonNullable<unknown> },
    { schema: NonNullable<unknown>; macro: NonNullable<unknown> },
    NonNullable<unknown>,
    {
        derive: NonNullable<unknown>
        resolve: NonNullable<unknown>
        schema: NonNullable<unknown>
    },
    {
        derive: NonNullable<unknown>
        resolve: NonNullable<unknown>
        schema: NonNullable<unknown>
        decorator: NonNullable<unknown>
    }

```

```

        store: NonNullable<unknown>
    }>

type UserRouterType = Elysia<
  '/users',
  false,
  {
    decorator: NonNullable<unknown>
    store: NonNullable<unknown>
    derive: NonNullable<unknown>
    resolve: NonNullable<unknown>
  },
  { type: NonNullable<unknown>; error: NonNullable<unknown> },
  { schema: NonNullable<unknown>; macro: NonNullable<unknown> },
  NonNullable<unknown>,
  {
    derive: NonNullable<unknown>
    resolve: NonNullable<unknown>
    schema: NonNullable<unknown>
  },
  {
    derive: NonNullable<unknown>
    resolve: NonNullable<unknown>
    schema: NonNullable<unknown>
    decorator: NonNullable<unknown>
    store: NonNullable<unknown>
  }
>

type StorageRouterType = Elysia<
  '/storages',
  false,
  {
    decorator: NonNullable<unknown>
    store: NonNullable<unknown>
    derive: NonNullable<unknown>
    resolve: NonNullable<unknown>
  },
  { type: NonNullable<unknown>; error: NonNullable<unknown> },
  { schema: NonNullable<unknown>; macro: NonNullable<unknown> },
  NonNullable<unknown>,
  {
    derive: NonNullable<unknown>
    resolve: NonNullable<unknown>
    schema: NonNullable<unknown>
  },
  {
    derive: NonNullable<unknown>
    resolve: NonNullable<unknown>
    schema: NonNullable<unknown>
    decorator: NonNullable<unknown>
    store: NonNullable<unknown>
  }
>

```

```

type RecipeRouterType = Elysia<
  '/recipes',
  false,
  {
    decorator: NonNullable<unknown>
    store: NonNullable<unknown>
    derive: NonNullable<unknown>
    resolve: NonNullable<unknown>
  },
  { type: NonNullable<unknown>; error: NonNullable<unknown> },
  { schema: NonNullable<unknown>; macro: NonNullable<unknown> },
  NonNullable<unknown>,
  {
    derive: NonNullable<unknown>
    resolve: NonNullable<unknown>
    schema: NonNullable<unknown>
  },
  {
    derive: NonNullable<unknown>
    resolve: NonNullable<unknown>
    schema: NonNullable<unknown>
    decorator: NonNullable<unknown>
    store: NonNullable<unknown>
  }
>
type ProductRouterType = Elysia<
  '/products',
  false,
  {
    decorator: NonNullable<unknown>
    store: NonNullable<unknown>
    derive: NonNullable<unknown>
    resolve: NonNullable<unknown>
  },
  { type: NonNullable<unknown>; error: NonNullable<unknown> },
  { schema: NonNullable<unknown>; macro: NonNullable<unknown> },
  NonNullable<unknown>,
  {
    derive: NonNullable<unknown>
    resolve: NonNullable<unknown>
    schema: NonNullable<unknown>
  },
  {
    derive: NonNullable<unknown>
    resolve: NonNullable<unknown>
    schema: NonNullable<unknown>
    decorator: NonNullable<unknown>
    store: NonNullable<unknown>
  }
>
type ChecklistRouterType = Elysia<
  '/checklists',

```

```

false,
{
    decorator: NonNullable<unknown>
    store: NonNullable<unknown>
    derive: NonNullable<unknown>
    resolve: NonNullable<unknown>
},
{ type: NonNullable<unknown>; error: NonNullable<unknown> },
{ schema: NonNullable<unknown>; macro: NonNullable<unknown> },
NonNullable<unknown>,
{
    derive: NonNullable<unknown>
    resolve: NonNullable<unknown>
    schema: NonNullable<unknown>
},
{
    derive: NonNullable<unknown>
    resolve: NonNullable<unknown>
    schema: NonNullable<unknown>
    decorator: NonNullable<unknown>
    store: NonNullable<unknown>
}
>

```

ПРИЛОЖЕНИЕ И Листинг сервера раздачи картинок

```

import { cors } from '@elysiajs/cors'
import { EntityType } from '@static/types'
import { Elysia, InternalServerError, t } from 'elysia'

const app = new Elysia()
  .use(cors({ credentials: true }))
  .post(
    '/',
    async ({ body, query, set }) => {
      const image = body.image
      if (!(image instanceof File)) {
        set.status = 422
        throw new Error('Not a file type')
      }

      if (!image.name) {
        set.status = 400
        throw new Error('No image found in FormData', {
          cause: 'formData image is undefined',
        })
      }

      const fileExtension = image.name.split('.').pop()
      if (fileExtension !== 'png') {
        set.status = 422
        throw new Error('Not a png format')
      }

      const result = await Bun.write(
        `${__dirname}/data/${query.type}/${query.entityId}.png`,
        image,
        { createPath: true }
      )
      if (!result) return new InternalServerError('Can not write file')
      return new Response()
    },
    {
      query: t.Object({
        type: t.Enum(EntityType),
        entityId: t.String(),
      }),
      body: t.Object({
        image: t.File(),
      }),
    }
  )
  .get(
    ':type/:entityId',
    async ({ params }) =>

```

```

        Bun.file(`${__dir-
name}/data/${params.type}/${params.entityId}.png`),
        {
            params: t.Object({
                type: t.Enum(EntityType),
                entityId: t.String(),
            }),
        }
    )
    .listen(3005, () => console.log('Static server listening 3005
port'))

export type STATIC_SERVER_TYPE = typeof app

```

ПРИЛОЖЕНИЕ К Листинг списка зависимостей

```

{
  "name": "virtual-fridge",
  "version": "1.0.0",
  "description": "",
  "type": "module",
  "scripts": {
    "dev:server": "bun run --watch ./server/core/index.ts",
    "dev:static": "bun run --watch ./server/static/index.ts",
    "dev:client": "bun --bun vite ./client/",
    "build:client": "tsc && vite build",
    "apply_migrations": "bunx prisma migrate dev --
schema=./server/prisma/schema.prisma",
    "prisma_gen": "bunx prisma generate --
schema=./server/prisma/schema.prisma"
  },
  "author": "EugeneAlWin",
  "license": "MIT",
  "devDependencies": {
    "@types/bun": "^1.1.2",
    "@types/cors": "^2.8.17",
    "@types/jsonwebtoken": "^9.0.6",
    "@types/react": "^18.2.37",
    "@types/react-dom": "^18.2.15",
    "@typescript-eslint/eslint-plugin": "^7.7.0",
    "@typescript-eslint/parser": "^6.10.0",
    "@vitejs/plugin-react": "^4.2.1",
    "@vitejs/plugin-react-swc": "^3.5.0",
    "bun-types": "^1.1.8",
    "eslint": "^8.53.0",
    "eslint-config-prettier": "^9.1.0",
    "eslint-plugin-prettier": "^5.1.3",
    "eslint-plugin-react-hooks": "^4.6.0",
    "eslint-plugin-react-refresh": "^0.4.6",
    "prettier": "^3.0.3",
    "prisma": "^5.15.0",
    "sass": "^1.69.5",
    "typescript": "^5.4.5",
    "vite": "^5.2.11",
    "vite-plugin-mkcert": "^1.17.1"
  },
  "dependencies": {
    "@elysiajs/cookie": "^0.8.0",
    "@elysiajs/cors": "^1.0.2",
    "@elysiajs/eden": "^1.0.8",
    "@elysiajs/swagger": "^1.0.3",
    "@paralleldrive/cuid2": "^2.2.2",
    "@prisma/client": "5.15.0",
    "@tanstack/react-query": "^5.29.2",
    "@tanstack/react-query-devtools": "^5.8.7",
    "axios": "^1.6.2",
    "bcrypt": "^5.1.1",
    "dotenv": "^16.3.1",

```



```
"elysia": "^1.0.23",  
"jsonwebtoken": "^9.0.2",  
"react": "^18.2.0",  
"react-dom": "^18.2.0",  
"react-router-dom": "^6.20.0",  
"react-toastify": "^9.1.3",  
"remeda": "^1.61.0",  
"zustand": "^4.4.6"  
}  
}
```

ПРИЛОЖЕНИЕ Л Таблица экономических показателей