МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 1-98 01 03 «Программное обеспечение информационной безопасности мобильных систем»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА**

по дисциплине «Современные технологии программирования мобильных систем»

Тема «Personal\_Nutritionist»

Исполнитель

студент 2 курса группы 5 Шкода Кристина Михайловна

(Ф.И.О.)

Руководитель работы ассистент Чистякова Ю. А.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Председатель Пацей Н.В.

(подпись)

Минск 2021

# Содержание

[Содержание 2](#_Toc103831193)

[**ВВЕДЕНИЕ** 3](#_Toc103831194)

[1 Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству 4](#_Toc103831195)

[2 Анализ требований к программному средству и разработка функциональных требований 6](#_Toc103831196)

[3 Проектирование программного средства 8](#_Toc103831197)

[3.1 Проектирование архитектуры приложения 8](#_Toc103831198)

[3.2 Проектирование базы данных 9](#_Toc103831199)

[3.3 Проектирование доступа к базе данных 12](#_Toc103831200)

[3.4 Схема и описание авторизации 12](#_Toc103831201)

[3.5 Проектирование логики сценариев использования 14](#_Toc103831202)

[3.6 Структура проекта 14](#_Toc103831203)

[4 Реализация программного средства 16](#_Toc103831204)

[4.1 Реализация сущностей 16](#_Toc103831205)

[4.2 Реализация уровня доступа к данным 16](#_Toc103831206)

[4.4 Реализация представления 17](#_Toc103831207)

[5 Тестирование, проверка работоспособности и анализ полученных результатов 19](#_Toc103831208)

[6 Методика использования программного средства 26](#_Toc103831209)

[ЗАКЛЮЧЕНИЕ 35](#_Toc103831210)

[Список использованных литературных источников 36](#_Toc103831211)

[ПРИЛОЖЕНИЕ А 37](#_Toc103831212)

[ПРИЛОЖЕНИЕ Б 38](#_Toc103831213)

[ПРИЛОЖЕНИЕ В 39](#_Toc103831214)

[ПРИЛОЖЕНИЕ Г 40](#_Toc103831215)

[ПРИЛОЖЕНИЕ Д 41](#_Toc103831216)

[ПРИЛОЖЕНИЕ Е 42](#_Toc103831217)

[ПРИЛОЖЕНИЕ Ж 43](#_Toc103831218)

[ПРИЛОЖЕНИЕ З 44](#_Toc103831219)

# **ВВЕДЕНИЕ**

Тема моего приложения: «Personal\_Nutritionist», что в переводе «Персональный Диетолог».

В современном мире информационные технологии развиваются с невероятной скоростью. Так, за относительно небольшой промежуток времени истории человечества они стали играть немаловажную роль во всех сферах жизни общества. С развитием технологий человеку становятся доступны все новые и новые возможности, о которых ранее приходилось только мечтать.

Стоит отметить, среди трендов последних лет очень четко выделяется направление самоорганизации. Приложения для повышения продуктивности занимают все более высокие позиции в рейтингах по количеству скачиваний в PlayMarket и AppStore. Тренд очевиден: современный человек сегодня – это человек, стремящийся контролировать и анализировать свою жизнь. И, безусловно, такой человек не откажется от удобного приложения, которое позволит контролировать дневной рацион питания и норму калорий.

Для чего нужно знать, сколько энергии человек получает с пищей и сколько расходует его организм. Количество энергии исчисляется в калориях. Так устроен мир, что люди в нем разные. Профессии некоторых людей предполагают постоянный контроль массы тела: спортсмены, танцоры и т.д. Это одни из немногих причин, по которым люди сталкиваются с необходимостью подсчитывать количество калорий, потребляемых за день.

Раньше для этого пользовались специальными таблицами, где искали продукты, смотрели количество калорий продукта и пересчитывали под нужный вес. На сегодняшний день данный процесс заметно упрощается с помощью калькулятора калорий. Не нужно самостоятельно пересчитывать калорийность под вес, считать их суммарное количество.

При написании курсового проекта одной из задач являлся простой и доступный дизайн как для пользователей, владеющих техническим образованием, так и нет.

В качестве интерфейса прикладного программирования был выбран обширный API-интерфейс — Windows Presentation Foundation (WPF), предназначенный для создания настольных программ с графически насыщенным пользовательским интерфейсом.

Для работы с WPF использовался объектно-ориентированный язык программирования с С-подобным синтаксисом — С#, разработанный для создания приложений на платформе Microsoft .NET Framework.

Для хранения данных была использована облачная платформа Microsoft SQL Server Management Studio.

1 Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству

Немаловажным этапом в разработке программного продукта является аналитический обзор прототипов. Аналогов десктопного программного средства «Персональный диетолог» мной найдено не было. Но были найдены сайты и мобильные приложения, осуществляющие ту же функцию, что и мое программное средство.

Например, сайт https://allcalc.ru помогает рассчитать суточную норму калорий. Интерфейс « Allcalc» представлен на рисунке 1.1.

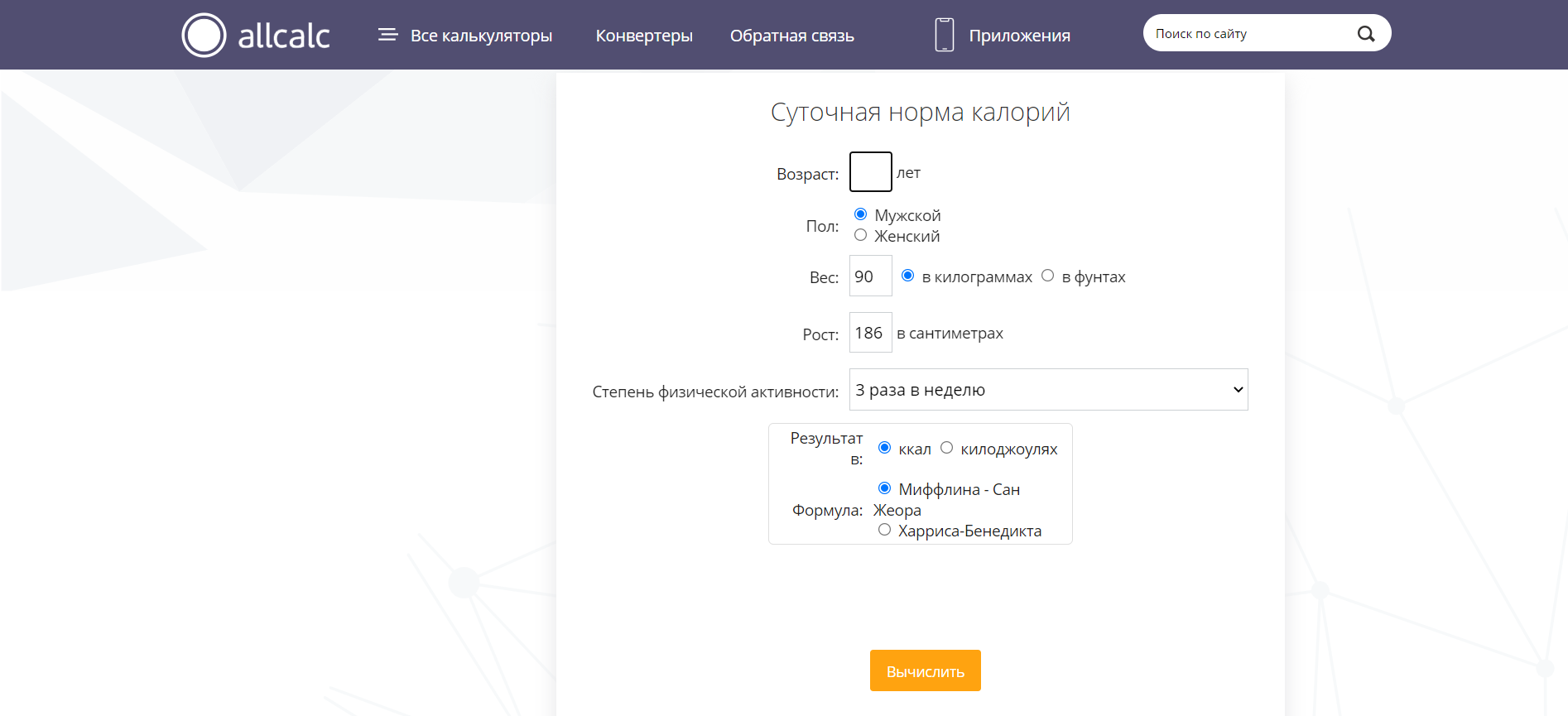


Рисунок 1.1 − Интерфейс «Allcalc»

На этом сайте можно рассчитать по введенным пользователем данным (возраст, пол, вес, рост, степень физической нагрузки) необходимое количество калорий на день для разных целей: остаться в том же весе или похудеть. Так же можно выбрать по какой формуле пользователь хочет, чтобы были рассчитаны калории, и в какой единице измерения были выведены.

Проанализировав «Allcalc», можно выделить некоторые плюсы и минусы программного средства.

Основные плюсы:

* возможность выбрать желаемую формулу;
* возможность ввести степень физической активности;
* возможность рассчитать калории для быстрого похудения;

Основные минусы:

* повсеместная реклама;
* нет возможности посчитать количество калорий для набора веса;

«Lifesum» — приложение, которое помогает следить за питанием.

Основные плюсы приложения:

* возможность подсчета съеденных калорий;
* возможность добавлять объем выпитой воды;
* возможность добавлять завтрак, обед и ужин;

Основные минусы:

* постоянно предлагает перейти на премиум;
* недоступны многие функции без покупки премиум-аккаунта;

Интерфейс «Lifesum» представлен на рисунке 1.2.

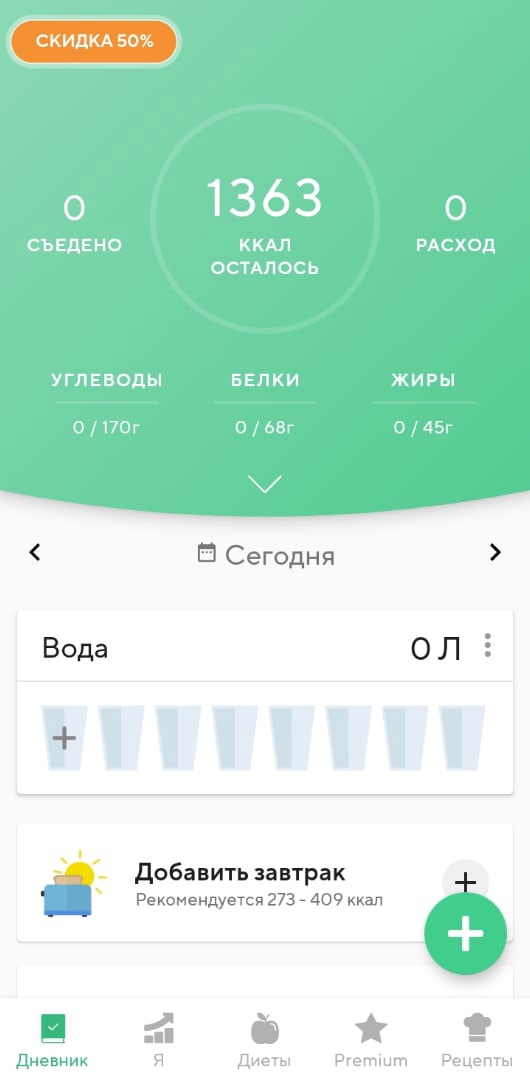


Рисунок 1.2 − Интерфейс «Lifesum»

2 Анализ требований к программному средству и разработка функциональных требований

Анализ требований — это сбор требований к программному обеспечению, их систематизации, документирования, анализа, выявления противоречий, незавершенности и разрешения конфликтов в процессе разработки программного обеспечения.

Различают три уровня требований к проекту:

* бизнес-требования;
* пользовательские требования;
* функциональные требования.

Бизнес-требования содержат высокоуровневые цели организации или заказчиков системы. К их числу относятся:

* простота интерфейса;
* использование принципов объектно-ориентированного программирования;
* использование системы управления базами данных.

Следующими требованиями являются требования пользователей. Данные требования описывают цели и задачи, которые пользователям позволит решить система. В данном приложение пользователи разделены на три группы: пользователь и администратор.

Пользователь должен иметь возможность:

* регистрировать себя в системе;
* входить в приложение, после ввода данных, необходимых для аутентификации;
* добавлять продукты;
* добавлять рецепты;
* добавлять понравившееся в избранное;
* заполнять завтрак, обед, ужин;
* удалять из избранного;
* выходить из профиля.

Администратор должен иметь возможность:

* выходить из профиля;
* смотреть список пользователей;
* доступ к данным пользователя;
* удалять рецепты/продукты;
* редактировать съеденное количество калорий пользователя.

Данные требования обобщены в виде диаграммы вариантов использования разрабатываемого программного средства, которая приведена в приложении А. Она отражает функциональность программного средства с точки зрения получения значимого результата для пользователя.

Функциональные требования определяют функциональность ПО, которую разработчики должны построить, чтобы пользователи смогли выполнить свои задачи в рамках бизнес-требований. После проведения анализа были выявлены следующие функциональные требования:

* архитектура приложения должна соответствовать шаблонам проектирования, таким как MVVM и Repository.
* вся информация должна храниться в базе данных;
* приложение должно предоставлять пользователям возможность создания нового аккаунта в виде формы регистрации;
* приложение должно предоставлять возможность пользователям проходить аутентификацию и входить в систему под соответствующим логином;
* приложение должно производить валидацию вводимых данных при регистрации, авторизации;
* приложение должно корректным образом обрабатывать возникающие исключительные ситуации;
* приложение должно предоставлять возможность пользователю добавлять/изменять завтрак, обед и ужин;
* приложение должно предоставлять возможность пользователю просматривать список рецептов и продуктов;
* приложение должно предоставлять возможность пользователю просматривать описание рецепта;
* приложение должно предоставлять возможность пользователю просматривать историю питания;
* приложение должно предоставлять возможность пользователю добавлять рецепт/продукт;
* приложение должно предоставлять возможность пользователю добавлять рецепты в избранное или же удалять из избранного;
* приложение должно предоставлять возможность администратору просматривать список пользователей;
* приложение должно предоставлять возможность администратору просматривать данные о пользователе;
* приложение должно предоставлять возможность администратору добавлять рецепты и продукты;
* приложение должно предоставлять возможность администратору удалять рецепты и продукты;
* приложение должно предоставлять возможность администратору просматривать описание рецепта;

Таким образом, был проведен тщательный анализ требований к программному средству, который позволил разработать список функциональных требований. Разработка данной программной системы должна проводиться в соответствии с сформированными списком.

3 Проектирование программного средства

Проектирование программного средства — этап жизненного цикла программного обеспечения, во время которого исследуется структура и взаимосвязи элементов разрабатываемой системы. На этапе проектирования уточняется функциональная спецификация системы: прорабатывается архитектура системы, определяются требования к аппаратному обеспечению.

## **3.1 Проектирование архитектуры приложения**

Архитектура программного обеспечения — совокупность важнейших решений об организации программной системы.

Для удовлетворения проектируемой системы различным атрибутам качества применяются различные архитектурные шаблоны (паттерны). В разрабатываемом приложении используется архитектурный шаблон Model-View-ViewModel (MVVM).

Шаблон MVVM имеет три основных слоя: модель, которая представляет бизнес-логику приложения, представление пользовательского интерфейса, и представление-модель, в котором содержится вся логика построения графического интерфейса и ссылка на модель, поэтому он выступает в качестве модели для представления.

На рисунке 3.1 представлена диаграмма, которая показывает общую структуру приложения в рамках шаблона MVVM.



Рисунок 3.1 – Структура шаблона MVVM

View или представление определяет визуальный интерфейс, через который пользователь взаимодействует с приложением. Так как пользовательский интерфейс и качество его реализации играет далеко не последнее место в конечном результате, разработка эффективного интерфейса, приятного и удобного для конечного пользователя, является важной задачей.

ViewModel или модель представления связывает модель и представление через механизм привязки данных. Она содержит модель, преобразованную к представлению, а также команды, которыми может пользоваться представление, чтобы влиять на модель.

Model или модель представляет собой логику работы с данными и описание фундаментальных данных, необходимых для работы приложения.

## **3.2 Проектирование базы данных**

Проектирование баз данных — процесс создания схемы базы данных и определения необходимых ограничений целостности.

Основные задачи проектирования базы данных:

* обеспечение хранения в БД всей необходимой информации;
* обеспечение возможности получения данных по всем необходимым запросам;
* сокращение избыточности и дублирования данных;
* обеспечение целостности базы данных.

Проектирование базы данных проводится в два этапа: концептуальное и логическое проектирование.

Концептуальное проектирование — построение семантической модели предметной области, то есть информационной модели наиболее высокого уровня абстракции. Такая модель создаётся без ориентации на какую-либо конкретную СУБД и модель данных. Основными понятиями модели являются: сущность, связь и атрибут

Сущность – это реальный или представляемый объект, информация о котором должна сохраняться и быть доступна.

Связь – это графически изображаемая ассоциация, устанавливаемая между двумя сущностями. Эта ассоциация обычно является бинарной и может существовать между двумя разными сущностями или между сущностью и ей же самой (рекурсивная связь).

Атрибут сущности − это любая детaль, которая служит для уточнения, идентификации, классификации, числовой характеристики или выражения состояния сущности.

В рамках этого этапа была создана модель, которая включает 8 сущностей:

* пользователь;
* избранное;
* история питания;
* продукты;
* рецепты;
* прием пищи;
* роли;
* подсчет калорий админом.

Также в модели были определены необходимые связи. Например, между сущностями роли и пользователь была установлена связь один-ко-многим. Для каждой сущности были выделены атрибуты. Например, для пользователя качестве атрибутов были выделены такие характеристики, как идентификатор и имя.

Логическое проектирование — создание схемы базы данных на основе конкретной модели данных, например, реляционной модели данных. Для реляционной модели данных логическая модель — набор схем отношений, обычно с указанием первичных ключей, а также «связей» между отношениями, представляющих собой внешние ключи. На этапе логического проектирования учитывается специфика конкретной модели данных, но может не учитываться специфика конкретной СУБД.

Логическая модель базы данных представлена на рисунке 3.3

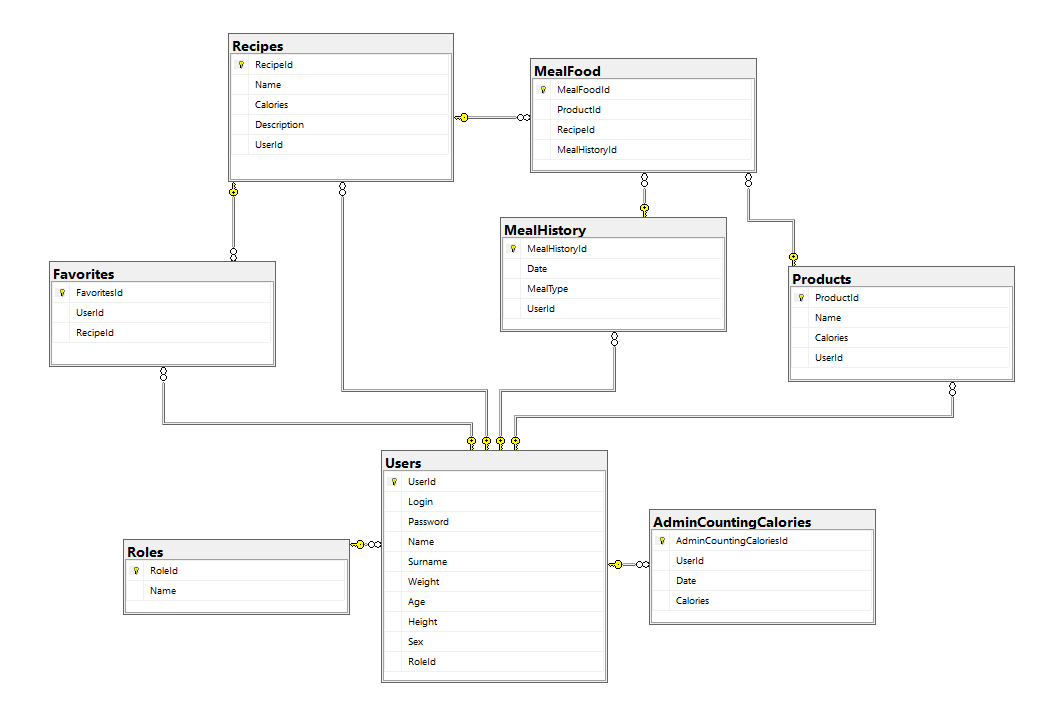


Рисунок 3.3 − Логическая модель базы данных

Всего в базе данных содержится 8 таблиц. В таблице Users хранятся все пользователи, зарегистрированные в приложении. Таблица Roles содержит роли, которые могут иметь пользователи. Таблица AdminCountingCalories хранит посчитанное администратором количество калорий за данный день. Таблица Products хранит продукты, созданные либо администратором, либо пользователем. Таблица Recipes хранит рецепты, добавленные либо администратором, либо пользователем. Таблица Favorites содержит рецепты, которые понравились пользователю. Таблица MealHistory хранит историю приема пищи. В таблице MealFood хранится еда, которую пользователь потребил на завтрак, обед и ужин.

Таблица Users состоит из 10 столбцов:

* UserId;
* Login;
* Password;
* Name;
* Surname;
* Weight;
* Age;
* Height;
* Sex;
* RoleId.

В столбцах Name и Surname хранится информация о имени и фамилии зарегистрированного пользователя соответственно, в столбце Weight – вес пользователя, Age – возраст, Height – рост, Sex – пол. UserId – идентификатор пользователя, RoleId – идентификатор роли пользователя. Login – логин пользователя, Password – пароль пользователя.

Таблица Roles состоит из 2 столбцов:

* RoleId;
* Name;

В столбце Name хранится информация о названии роли пользователя, в столбце RoleId – идентификатор.

Таблица AdminCountingCalories состоит из 4 столбцов:

* AdminCountingCaloriesId;
* UserId;
* Date;
* Calories.

В столбце Calories хранится количество калорий, посчитанные администратором, в столбце Date – даты, когда было съедено данное количество калорий. UserId – идентификатор пользователя, который съел столько калорий. AdminCountingCaloriesId – идентификатор числа калорий, посчитанных администратором.

Таблица Products состоит из 4 столбцов:

* ProdictId;
* Name;
* Calories;
* UserId.

В столбце ProdictId хранится идентификатор продукта питания, UserId – идентификатор пользователя, который создал данный продукт. Name – название продукта питания. Calories – это количество калорий в данном продукте питания.

Таблица Recipes состоит из 5 столбцов:

* RecipeId;
* Name;
* Calories;
* Description;
* UserId.

В столбце RecipeId – идентификатор рецепта, UserId – идентификатор пользователя, который создал данный рецепт. Name – название рецепта. Calories – количество калорий в данном рецепте, Description – описание рецепта.

Таблица Favorites состоит из 3 столбцов:

* FavoriteId;
* UserId;
* RecipeId.

В столбце FavoriteId хранится идентификатор рецепта, находящегося в избранном. UserId –идентификатор пользователя, которому понравился рецепт. RecipeId – идентификатор рецепта из таблицы рецептов.

Таблица MealFood состоит из 4 столбцов:

* MealFoodId;
* ProductId;
* RecipeId;
* MealHistoryId.

В столбце MealFoodId хранится идентификатор приема пищи, характеризующий потребление продукта питания или же блюда по рецепту, в столбце ProductId – идентификатор продукта питания, который был потреблен за данный прием пищи, столбце RecipeId – идентификатор рецепта, который был потреблен за данный прием пищи. MealHistoryId – идентификатор истории приема пищи.

Таблица MealHistory состоит из 4 столбцов:

* MealHistoryId;
* Date;
* MaelType;
* UserId.

В столбце MealHistoryId хранится идентификатор истории приема пищи. В столбце Date – дата, когда был прием пищи, в столбце MaelType – тип приема пищи (0 – завтрак, 1 – обед, 2 – ужин). UserId – идентификатор пользователя, который ел.

## **3.3 Проектирование доступа к базе данных**

Для инкапсулирования логики работы с источниками данных в программе используется паттерн Repository. Класс, реализующий данный паттерн, не содержит бизнес-логику, не управляет бизнес-процессами, он только содержит операции над данными. Как правило, репозиторий реализует CRUD-интерфейс, то есть представляет операции по извлечению, добавлению, редактированию и удалению данных.

## **3.4 Схема и описание авторизации**

Схема авторизации представлена на рисунке 3.4. Сразу после запуска программы пользователь попадает в окно авторизации. Если поля логина и пароля будут пустыми, то кнопка авторизации будет не доступна. После корректного ввода данных программа обращается к базе данных и проверяет существует ли пользователь с соответствующим логином и паролем. Если пользователь найден, будет выполнен вход в приложение, в противном случае выведено сообщение об ошибке.

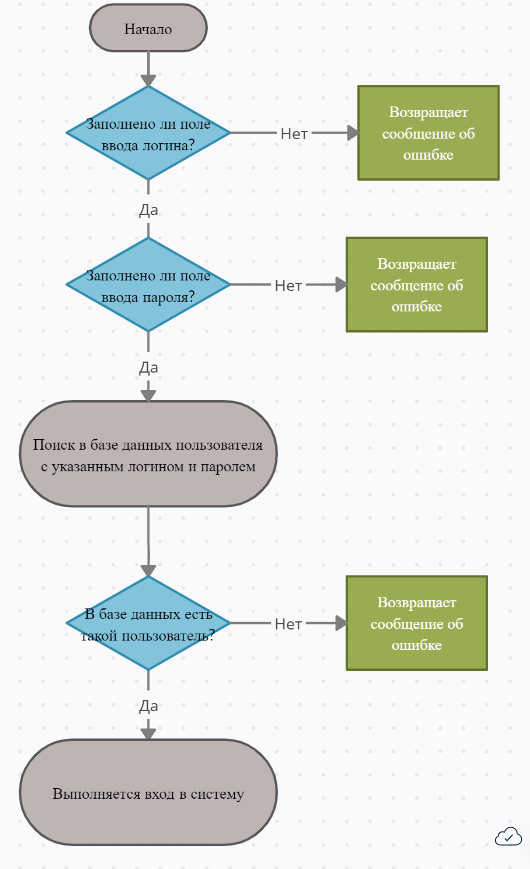


Рисунок 3.4 − Блок-схема авторизации

## **3.5 Проектирование логики сценариев использования**

При проектировании приложения были рассмотрены различные сценарии использования. На рисунке 3.5 представлена диаграмма последовательности добавление продукта в завтрак.

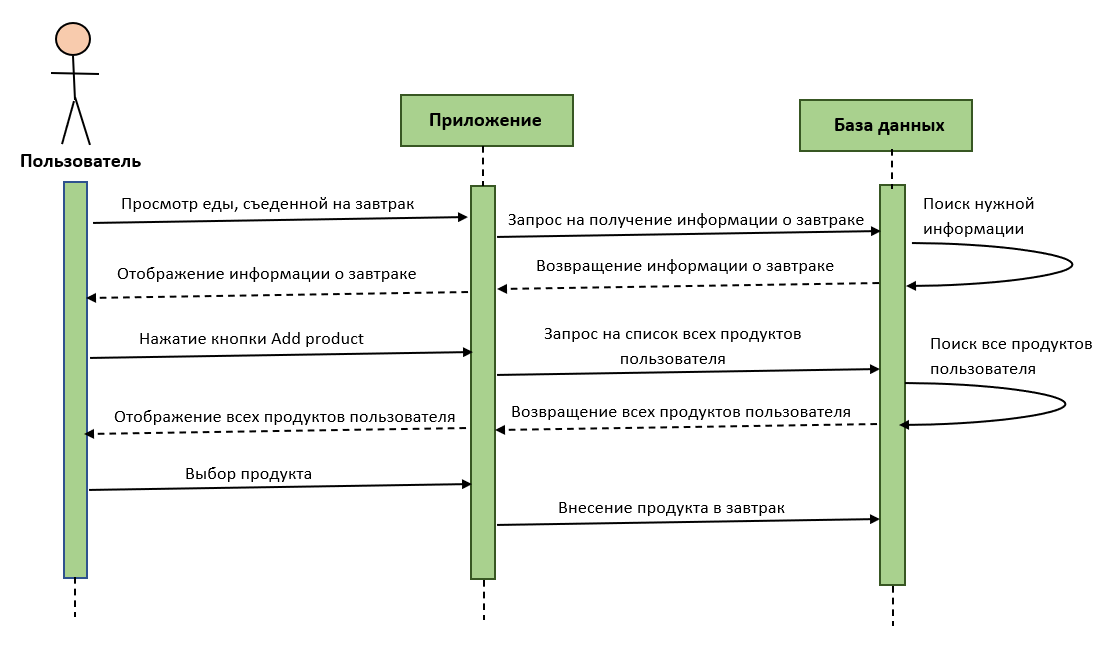


Рисунок 3.5 − Диаграмма последовательности добавление продукта в завтрак

Из диаграммы видно, что сначала пользователю необходимо просмотреть всю съеденную им еду на завтрак. Затем приложение отправит запрос в базу данных на получение информации о завтраке. Дальше, когда база данных вернет информацию о завтраке, эта информация будет показана пользователю. После того, как пользователь нажмет кнопку добавления продукта, приложение отправит запрос на список всех продуктов пользователя. Далее, когда база данных вернет список продуктов, он будет показан пользователю. Потом пользователь выберет необходимый продукт и в конце приложение внесет продукт в список съеденной еды на завтрак.

## **3.6 Структура проекта**

Структура проекта представлена на рисунке 3.6.

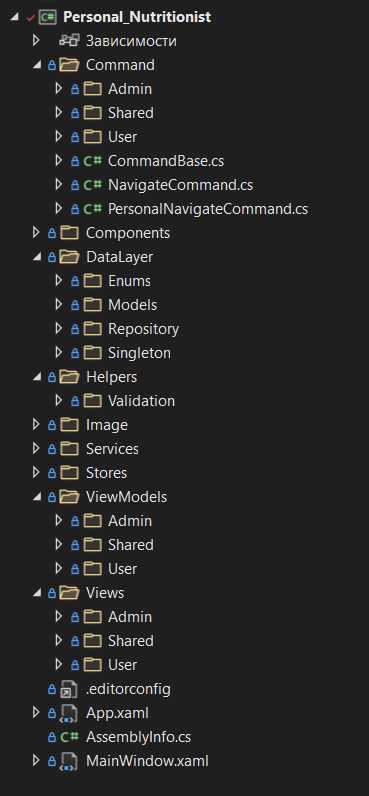


Рисунок 3.6 – Структура проекта

Папка «Commands» содержит команды. В ней находятся три папки, в которых команды определены отдельно для двух ролей и совпадающие команды. Папка «Components» содержит все необходимые графические элементы, необходимые для корректного отображения интерфейса программы, к примеру иконки. Папка «DataLayer» содержит классы для работы с данными. Папка «Enums» содержит все перечисления. Папка «Models» содержит все классы моделей. Папка «Repository» содержит классы для работы с базой данных. Папка «Singleton» содержит класс для получения текущего аккаунта пользователя. Папка «Helpers» содержит классы для валидации различных полей. Папка «Services» содержит классы для переключения UserControls. Папка «Stores» содержит классы хранения текущего UserControl. Папка «ViewModels» содержит все классы моделей представлений. Папка «Views» содержит все классы представлений.

4 Реализация программного средства

Важным этапом разработки приложения является непосредственная реализация программного решения в соответствии с уже сформированными требованиями и шаблонами.

**4.1 Реализация сущностей**

В соответствии с требованиями в качестве хранилища данных программного средства должна быть база данных, поэтому первым шагом в реализации программы является выбор технологии, позволяющей это осуществить. Выбор остановился на ORM технологии Entity Framework. Она предоставляет три подхода по проектированию базы данных. В данном программном решении был использован подход Code-First. Создание базы данных происходит из созданной вручную модели объектов C#. Созданные модели объектов совпадают с сущностями, которые были сформированы раннее в разделе 3.2.

Диаграмма классов UML для сущностных классов представлена в приложении Б.

На диаграмме классов видно, как связаны между собой сущности. Сущность User связана с классами Favorites, Recipe, MealHistory, Product, AdminCountingCalories отношением композиции т.е. при уничтожении объекта User в области памяти вместе с ним будут уничтожены и объекты Favorites, Recipe, MealHistory, Product, AdminCountingCalories связанные с ним. И в этом плане объект User является главным, а объекты Favorites, Recipe, MealHistory, Product, AdminCountingCalories – зависимыми. Также на диаграмме можно заметить, что класс Role и User также связаны отношением композиции. Recipe связана с сущностями Favorite и MealFood отношением композиции. MealHistory связана с MealFood отношением композиции. Product связана с MealFood отношением композиции.

**4.2 Реализация уровня доступа к данным**

В программном решении доступ к данным организован с помощью паттерна Repository. Паттерн Repository позволяет абстрагироваться от конкретных подключений к источникам данных, с которыми работает программа, и является промежуточным звеном между классами, непосредственно взаимодействующими с данными, и остальной программой. В соответствии с паттерном Repository был создан обобщенный интерфейс IRepository<T>. Код интерфейса IRepository <T> представлен в приложении В.

**4.3 Реализация архитектуры MVVM**

Для реализации паттерна MVVM файлы программы были распределены по соответствующим директориям и реализовали соответствующие функции. Разделение проекта на логические модули описано в разделе 3.5 «Структура проекта».

Навигация по страницам представлена классами NavigationStore и NavigationService, который реализует интерфейс INotifyPropertyChanged. Интерфейс INotifyPropertyChanged позволяет уведомить систему об изменениях свойств модели. Эти классы позволяют организовать переход между UserControls, которые отвечают за вход пользователя в систему, регистрацию и главную страницу пользователя. На главной странице пользователя навигация происходит по такому же принципу и представлена классами PersonalNavigationStore и PersonalNavigationService. В классах PersonalNavigationStore и NavigationStore хранится текущий UserControl, то есть тот, на котором находится пользователь в данный момент. Эти классы реализуют метод OnCurrentViewModelChanged, который уведомляет об изменении текущего UserControl. Классы PersonalNavigationService и NavigationService реализуют метод Navigate, который и осуществляет непосредственно само переключение UserControls. Код класса навигации NavigationService представлен в приложении Е, а код класса навигации для страниц конкретного пользователя PersonalNavigationService представлен в приложении Д.

В приложении используется паттерн Command который позволяет инкапсулировать запрос на выполнение определенного действия в виде отдельного объекта. В WPF команды представлены интерфейсом ICommand. В приложении он представлен в виде абстрактного класса CommandBase, который наследуют все классы команд собственной команды RelayCommandParametr код которой приведен в приложении Ж.

Класс определяет два метода:

* CanExecute: определяет, может ли команда выполняться
* Execute: собственно, выполняет логику команды

А также реализует событие OnCanExecuteChanged вызывается при изменении условий, указывающий, может ли команда выполняться.

**4.4 Реализация представления**

Для разработки графической части приложения была выбрана технология WPF.

Windows Presentation Foundation (WPF) — это библиотека для создания пользовательских интерфейсов для интеллектуальных клиентских приложений. В основе WPF лежит мощная инфраструктура, основанная на DirectX — API-интерфейсе графики с аппаратным ускорением, который обычно используется в современных компьютерных играх. Это означает, что применение развитых графических эффектов не приведёт к снижению производительности.

Одной из важных особенностей WPF является использование языка декларативной разметки интерфейса XAML, основанного на XML. Разработка с использованием XAML позволяет отделить графический интерфейс от логики приложения, а также создавать насыщенный интерфейс, используя или декларативное объявление интерфейса, или код на управляемых языках C#.

В конечном итоге в приложении было реализованы следующие пользовательские элементы управления.

* стартовая для регистрации;
* страница для авторизации;
* страница, содержащая меню, для пользователя;
* страница, содержащая меню, для администратора;

На главных страницах пользователя и администратора имеются подстраницы, которые располагаются на главной в зависимости от выбранного пункта меню. Для пользователя сделаны следующие страницы:

* страница для добавления приема пищи;
* страница для изменения приема пищи;
* страница для добавления рецепта в прием пищи;
* страница для добавления продукта в прием пищи;
* страница для отображения рецептов;
* страница для добавления рецепта;
* страница для просмотра описания рецепта;
* страница для отображения продуктов питания;
* страница для добавления продуктов питания;
* страница для отображения избранных рецептов.

Для администратора сделаны следующие страницы:

* страница для просмотра списка пользователей;
* страница для просмотра приема пищи пользователя;
* страница для просмотра еды на завтрак, обед или ужин;
* страница для отображения и редактирования списка продуктов питания;
* страница для добавления продукта питания;
* страница для добавления рецепта;
* страница для просмотра описания рецепта;
* страница для отображения и редактирования рецептов.

В результате выполнения данного этапа было создано функционирующее программное средство.

5 Тестирование, проверка работоспособности и анализ полученных результатов

Прежде всего были проведены тесты авторизации: проверка на пустые и неверные данные. Валидация выполнена таким образом, что кнопка для входа или регистрации будет недоступна до тех пор, пока поля не будут пустыми, а введённые данные не будут корректными. При таком способе валидации шанс ввести неверные данные минимален. На рисунке 5.1 видно валидация на странице регистрации не допускает ввода некорректных значений в поля, а также все поля являются обязательными для заполнения.

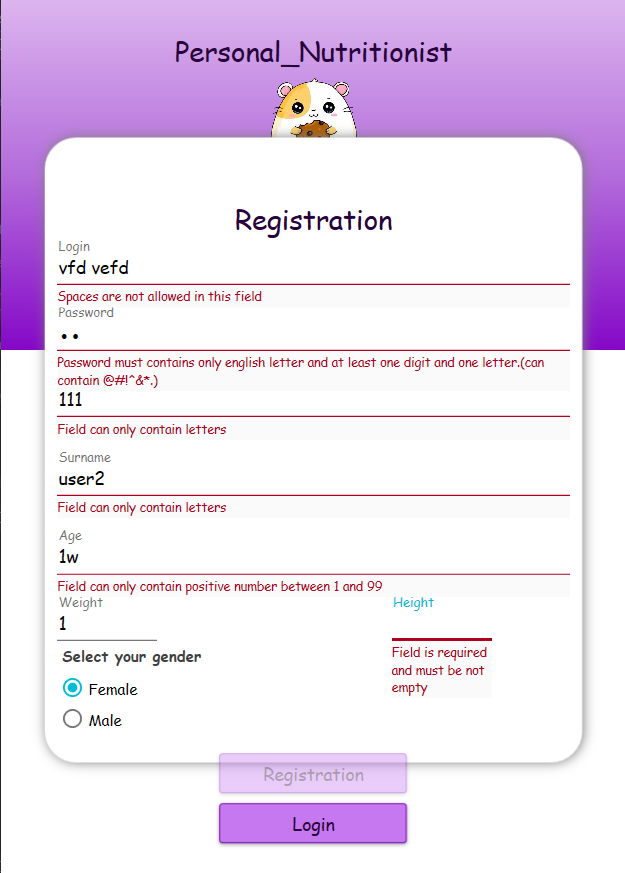


Рисунок 5.1 – Валидация формы регистрации

На рисунке 5.2 представлена валидация формы входа

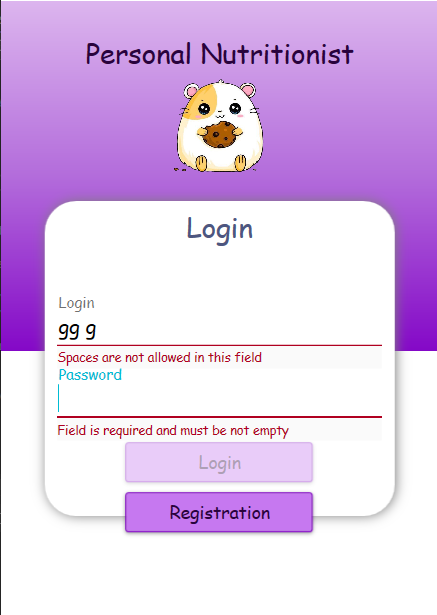


Рисунок 5.2 – Валидация формы входа

Если пользователь при регистрации вводит логин, который уже зарегистрирован в системе, то получает уведомление об этом как показано на рисунке 5.3:

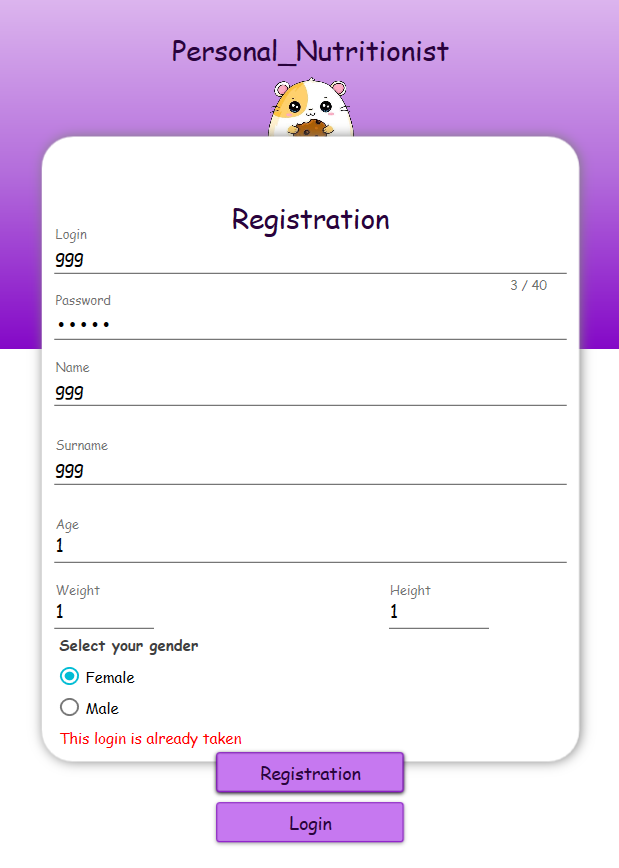
****

Рисунок 5.3 – Регистрация уже существующего пользователя

Если пользователь пытается зайти в свой профиль с неверными данными, то так же получает сообщение об этом, чтобы правильно скорректировать данные. При неправильном пароле выводится сообщение как показано на рисунке 5.4:

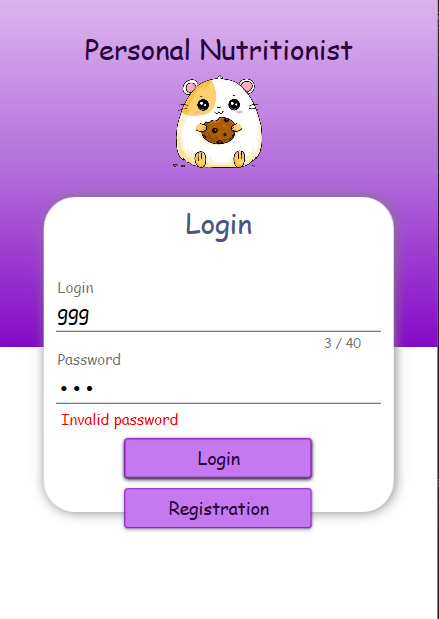


Рисунок 5.4 – Вход с неправильным паролем

При попытке входа незарегистрированного пользователя приложение отобразит сообщение о том, что такой пользователь не зарегистрирован в системе, как это показано на рисунке 5.5:

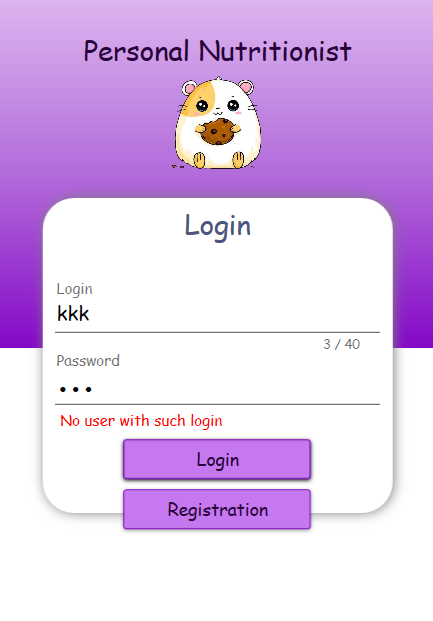


Рисунок 5.5 – Вход незарегистрированного пользователя

Также были проведены тесты на работу валидации внутри приложения. Наличие валидации является обязательным в связи с тем, что в приложении постоянно ведётся работа с базой данных.

Валидация организованна таким же образом, как и на форме авторизации, т.е. кнопки добавить и обновить будут недоступны до тех пор, пока не будут введены верные данные в поля. Все ошибки, возникшие при валидации, доступны и отображены под соответствующим полем, что позволяет пользователю легко понять какие данные не валидны и быстро исправить их.

Пример результата валидации данных и обработки различных экстремальных ситуаций внутри приложения приведены на рисунках 5.6-5.8

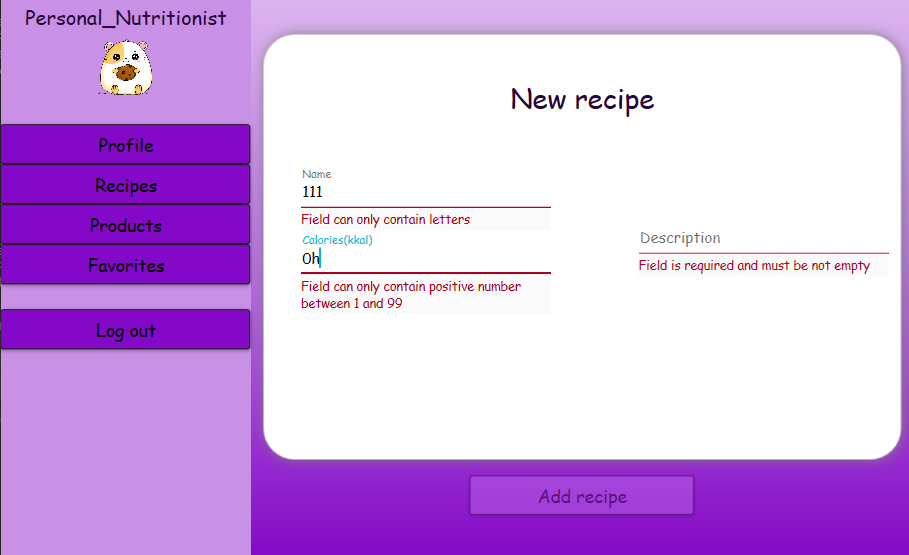


Рисунок 5.6 – Валидация при добавлении рецепта

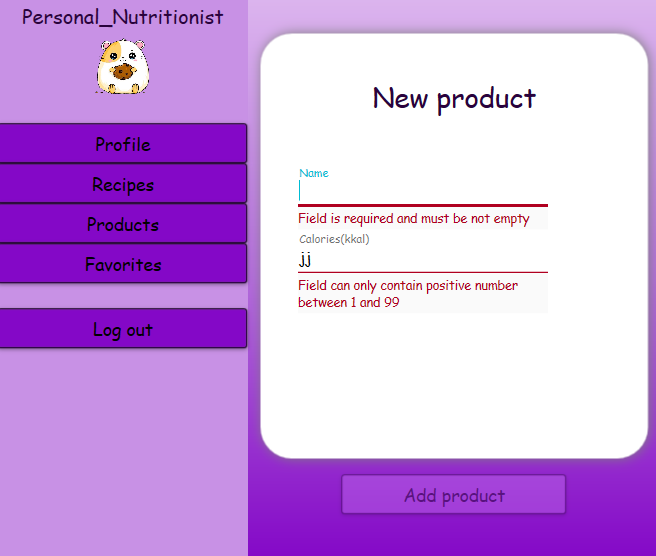


Рисунок 5.7 – Валидация при добавлении продукта

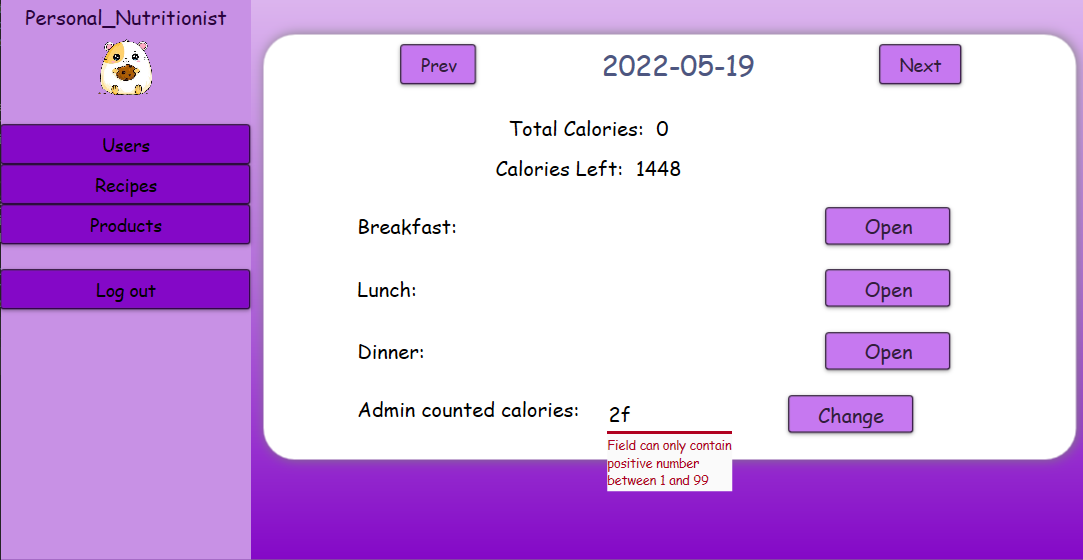


Рисунок 5.8 – Валидация изменении админом количества калорий

На данном этапе были выполнены тесты на проверку работоспособности приложения, а именно на проверку валидации. Были проведены анализы результатов, которые показали, что валидация в приложении работает в соответствии с тем, как она задумывалась.

6 Методика использования программного средства

После установки и регистрации в зависимости от роли пользователя предоставляются различные функции. Далее будут перечислены возможности обычного пользователя на рисунках 6.1 – 6.10.

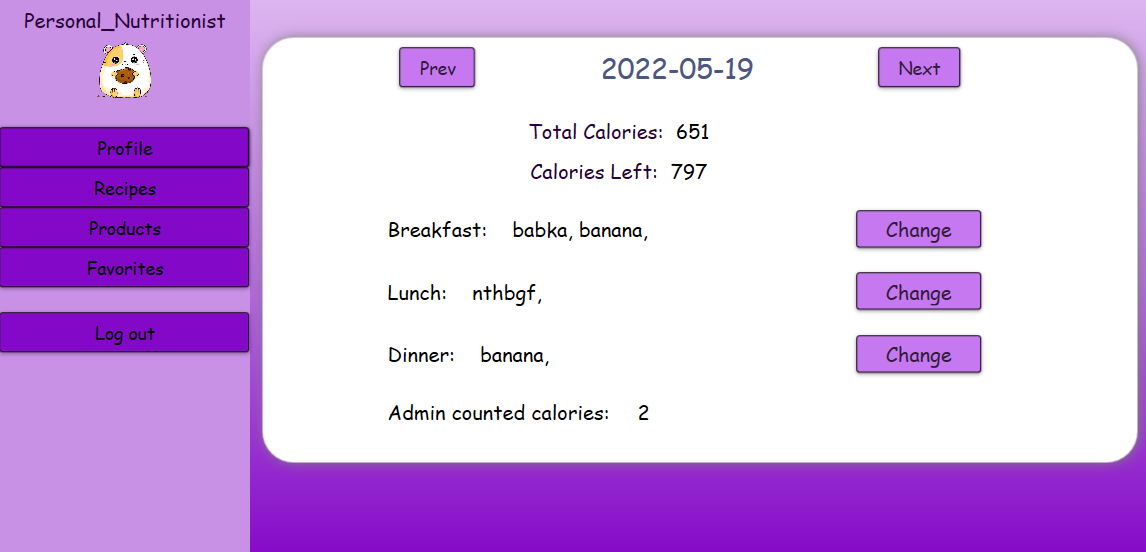


Рисунок 6.1 – Просмотр приемов пищи

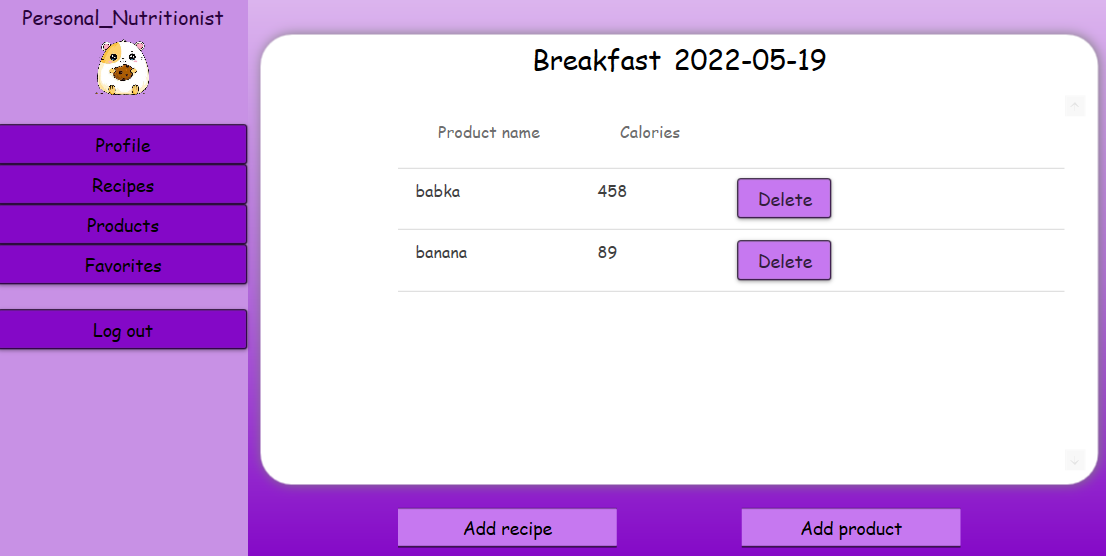


Рисунок 6.2 – Добавление и удаление еды

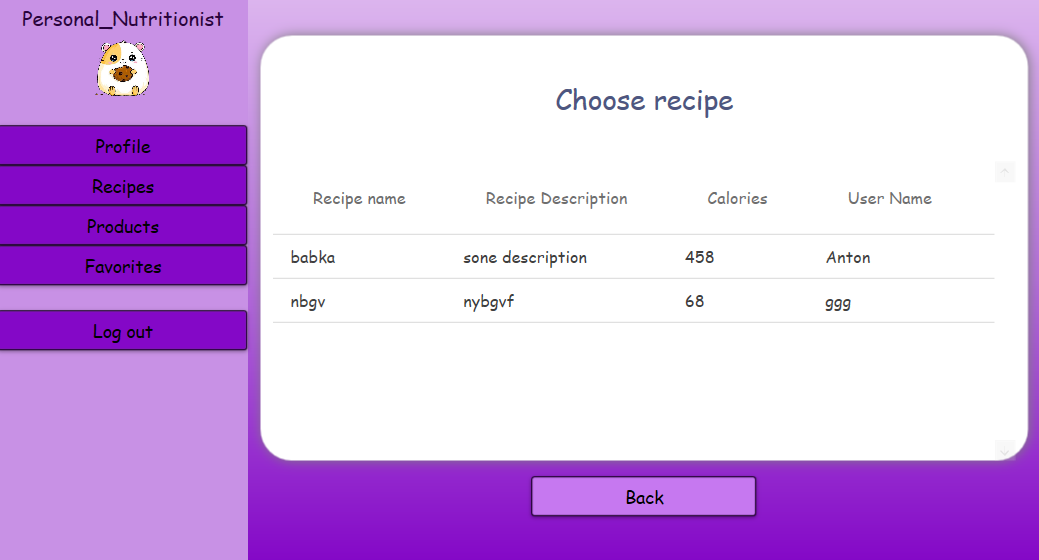


Рисунок 6.3 – Выбор рецепта при добавлении в рацион

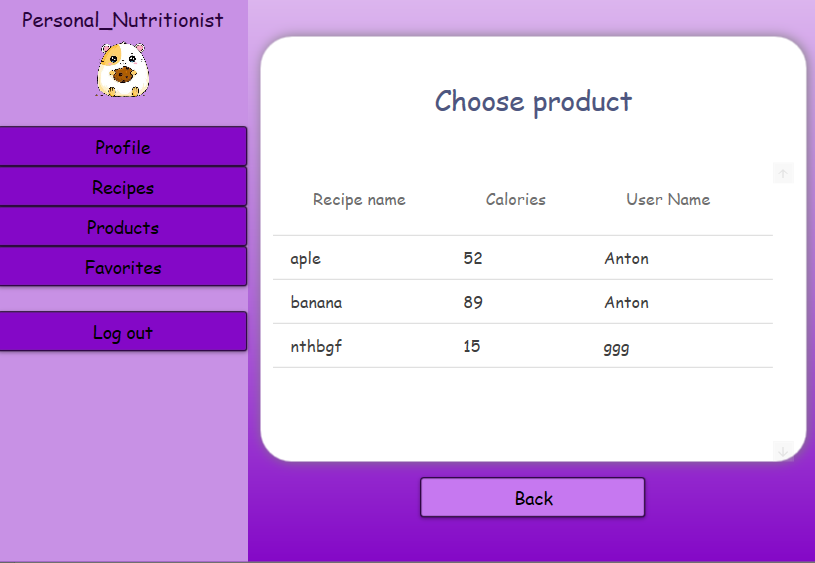


Рисунок 6.4 – Выбор продукта при добавлении в рацион

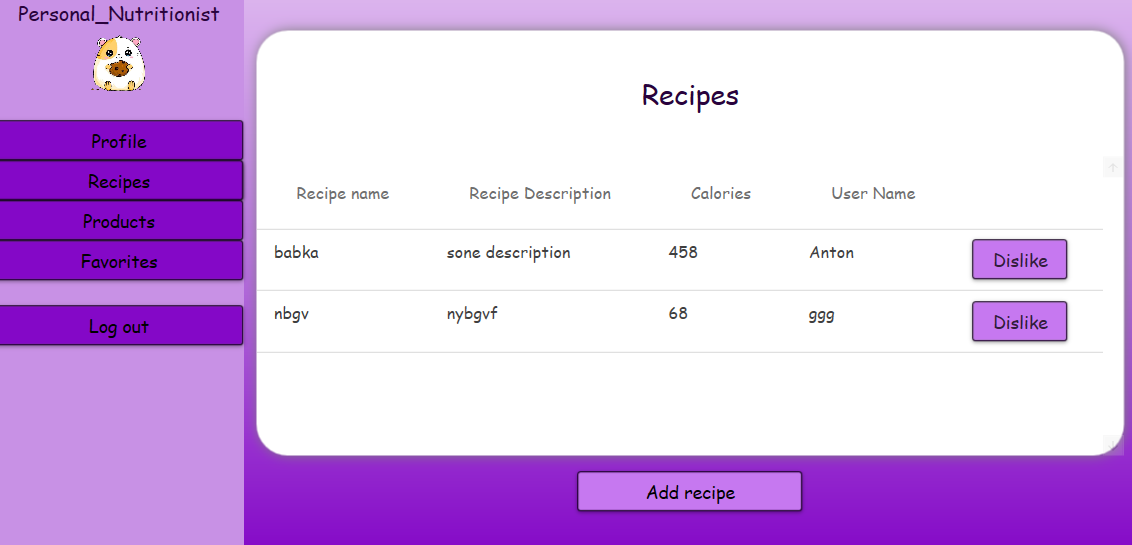


Рисунок 6.5 – Просмотр рецептов

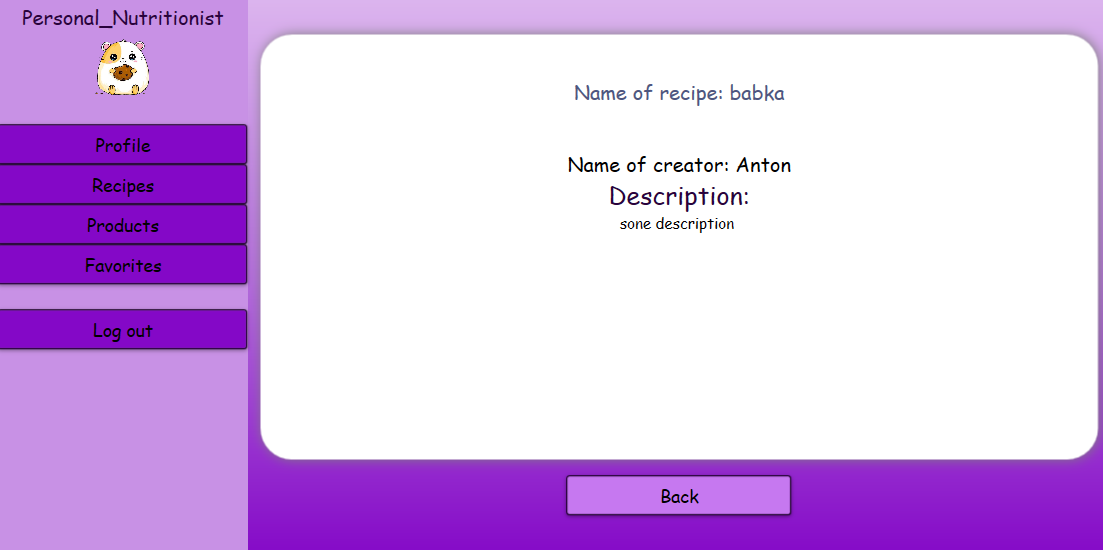


Рисунок 6.6 – Просмотр описания рецепта

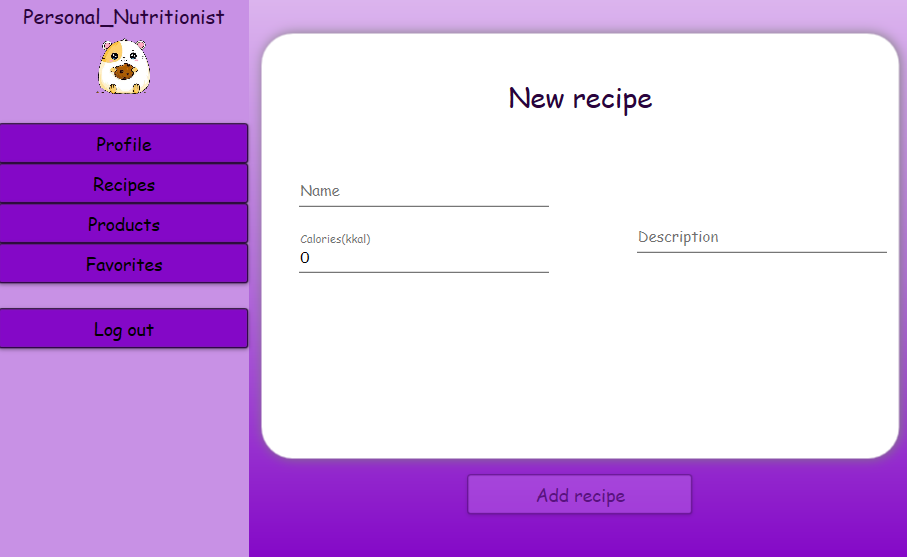


Рисунок 6.7 – Добавление рецепта

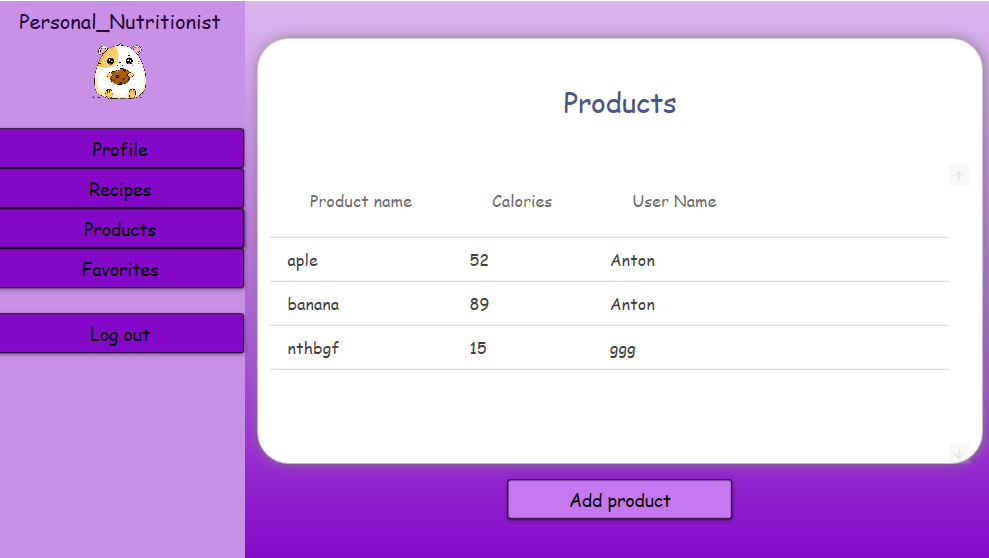


Рисунок 6.8 – Просмотр продуктов питания

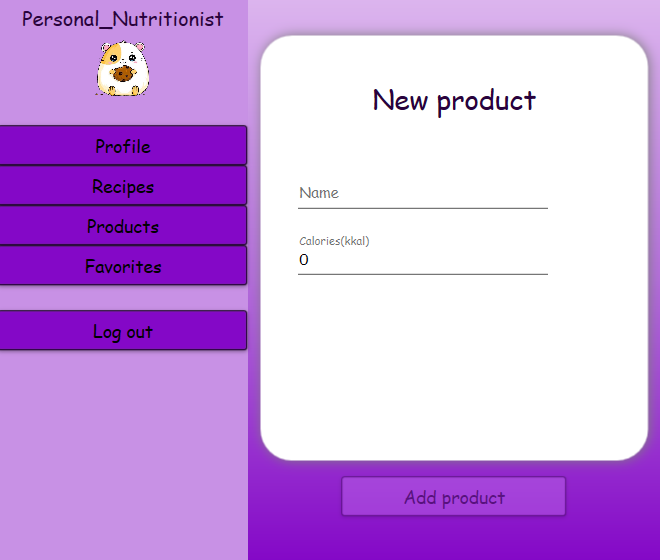


Рисунок 6.9 – Добавление продуктов питания

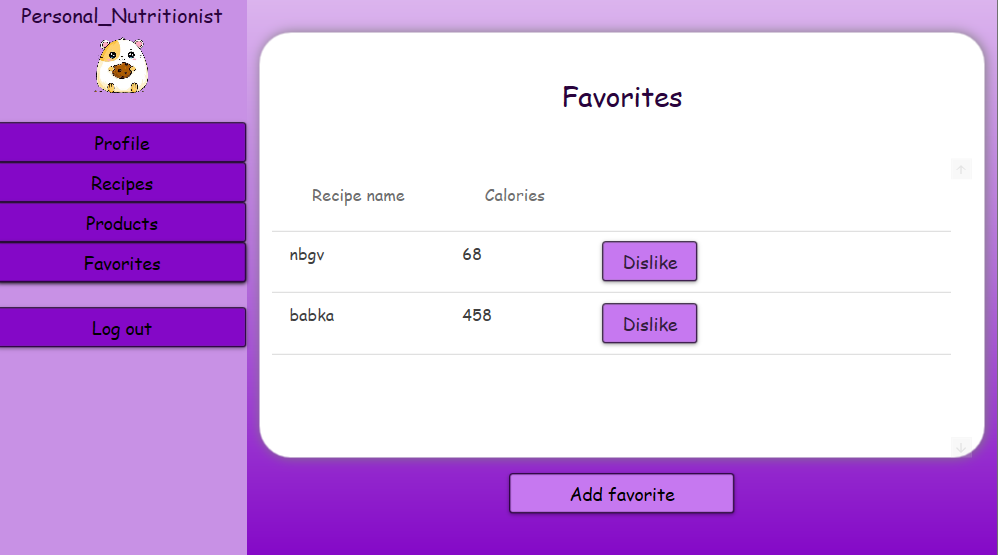


Рисунок 6.10 – Просмотр избранного

На рисунках 6.11-6.24 показаны возможности администратора.

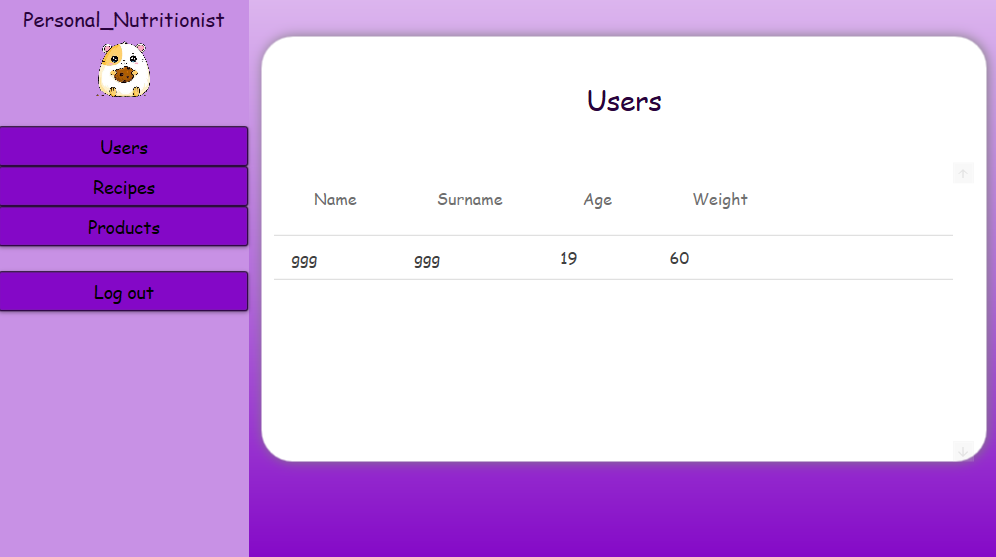


Рисунок 6.23 – Просмотр списка зарегистрированных пользователей

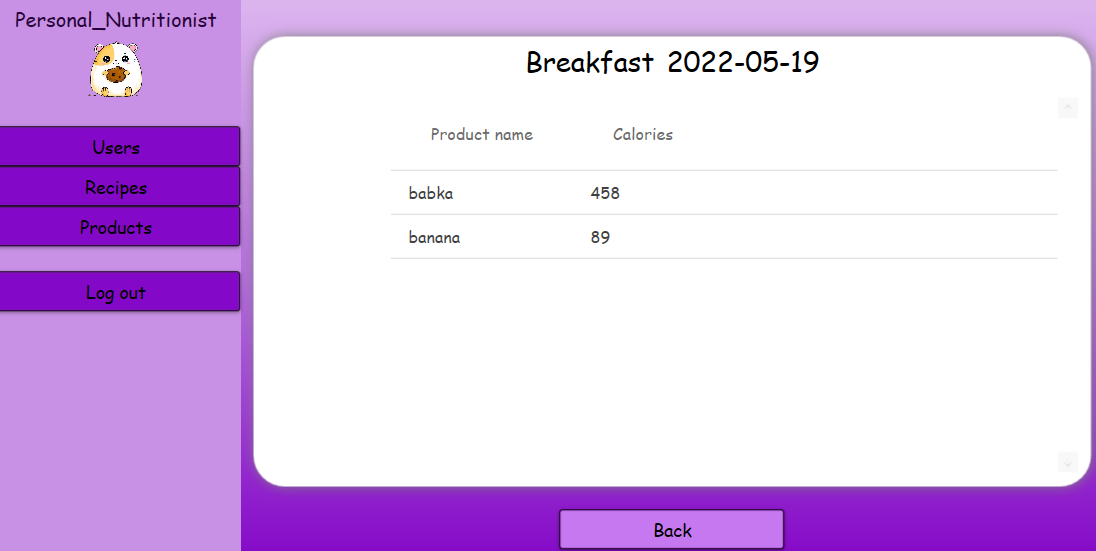


Рисунок 6.7 – Просмотр рациона

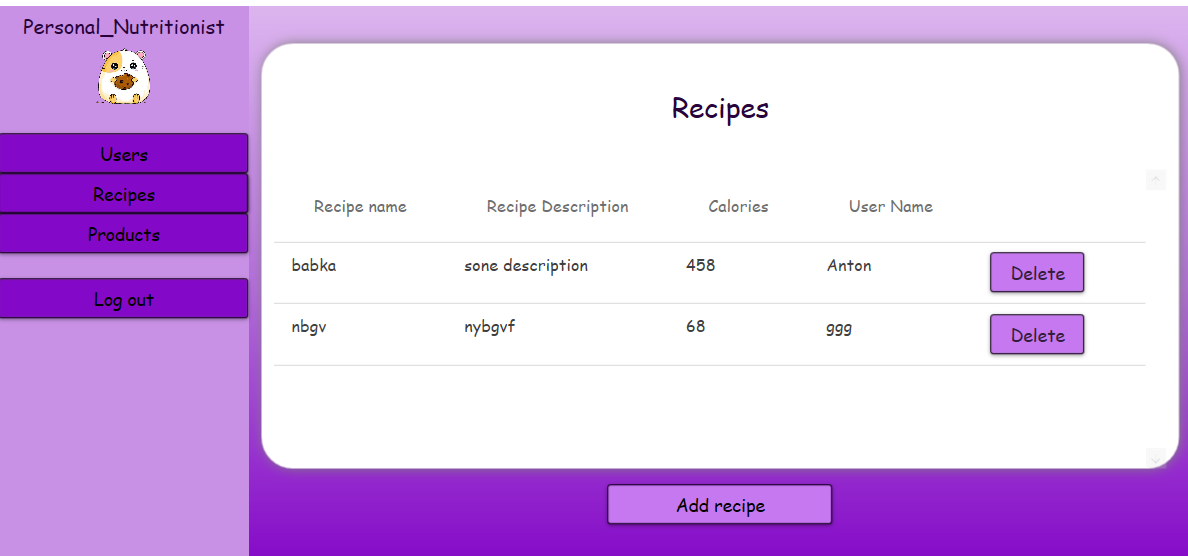


Рисунок 6.8 – Просмотр/ удаление рецептов

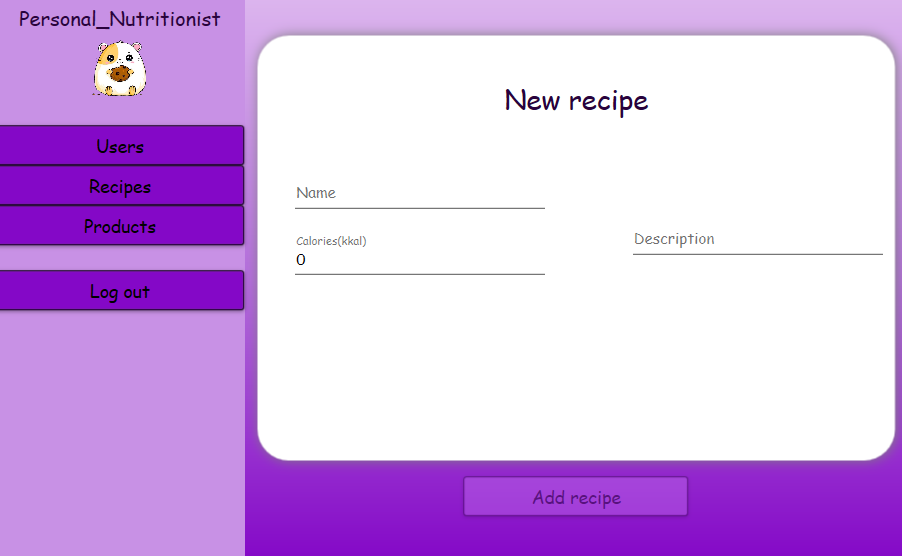


Рисунок 6.9 – Добавление рецепта

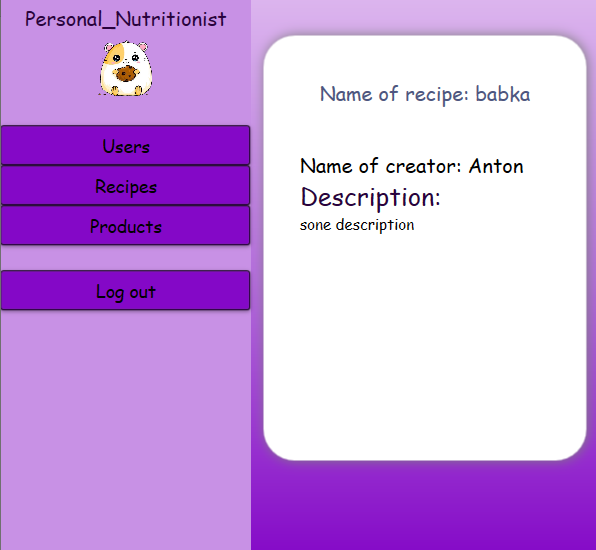


Рисунок 6.10 – Просмотр описания рецепта

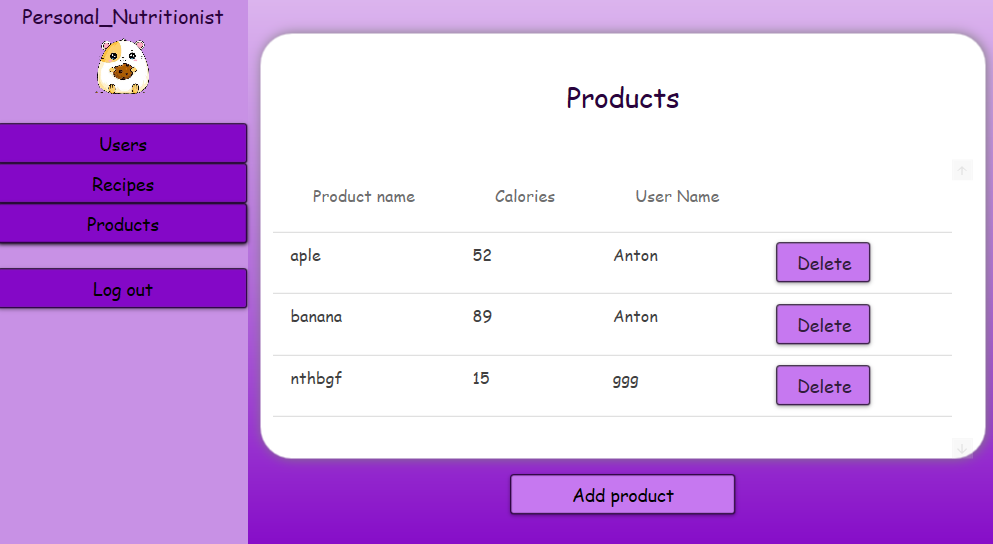


Рисунок 6.23 – Просмотр/удаление продуктов питания

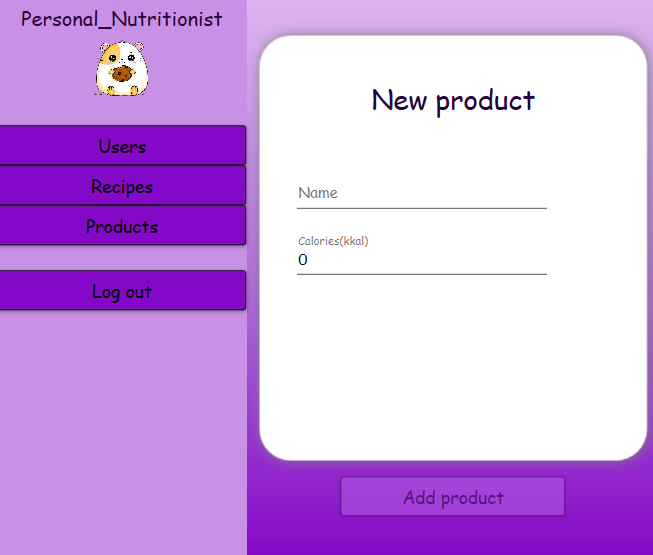


Рисунок 6.24 – Добавление продукта питания

Исходя из приведённых рисунков видно, как именно нужно использовать программное средство всех ролей.

ЗАКЛЮЧЕНИЕ

В процессе решения поставленной задачи была достигнута поставленная цель по созданию программного средства «Персональный диетолог». При разработке были выполнены все пункты из указанного списка предполагаемого основного функционала приложения.

Также были изучены основные технологии, которые применялись в разработке данного программного средства. В основе структуры приложения лежит паттерн MVVM. В качестве системы управления базой данных был выбран и использован Microsoft SQL Server 2018.

Данный проект был разработан для тех пользователей, которые хотели бы следить за своим питанием и весом. Главной задачей было сделать простое и удобное приложение. Для этого были разработаны следующие страницы: «Login», «Registration», «Home» для всех ролей. Для администратора были созданы страницы: «User», «User Meal History», «User Info», «Recipe», «Recipe Info», «Add Recipe», «Product», «Add Product». Для пользователя были созданы следующие страницы «Recipe», «Recipe Info», «Add Recipe», «Profile», «Change Meal», «Add Meal Recipe», «Add Meal Product», «Product», «Add Product», «Favorites». Все эти страницы были созданы для максимально удобного и эффективного использования приложения.

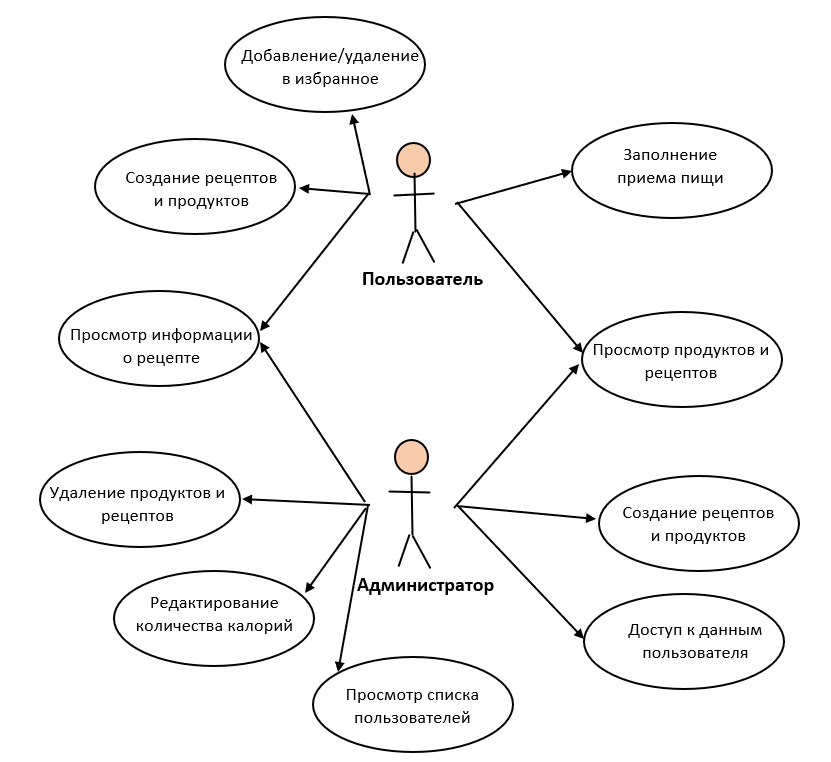
В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

Список использованных литературных источников

1. Пацей, Н.В. Курс лекций по языку программирования С# / Н.В. Пацей. – Минск: БГТУ, 2016. – 175 с.
2. **Preply** Enterprise [Электронный ресурс] / Режим доступа: <https://preply.com/>. Дата доступа: 20.04.2021
3. MSDN сеть разработчиков в Microsoft [Электронный ресурс] / Режим доступа: <http://msdn.microsoft.com/library/rus/> . Дата доступа: 21.04.2021
4. METANIT.COM Сайт о программировании [Электронный ресурс] / Режим доступа: <https://metanit.com> . Дата доступа: 21.04.2021
5. ProfessorWeb .NET & Web Programming [Электронный ресурс] / Режим доступа: <https://professorweb.ru> Дата доступа: 11.04.2021
6. Microsoft Azure [Электронный ресурс] / Режим доступа: <https://docs.microsoft.com/ru-ru/azure/> – Дата доступа: 15.04.2021
7. Форум для программистов или разработчиков [Электронный ресурс] – <https://stackoverflow.com/> – Дата доступа: 29.04.2021
8. Microsoft Docs Archived Content [Электронный ресурс]/ Режим доступа: <https://docs.microsoft.com/en-us/archive/> Дата доступа: 23.04.2021

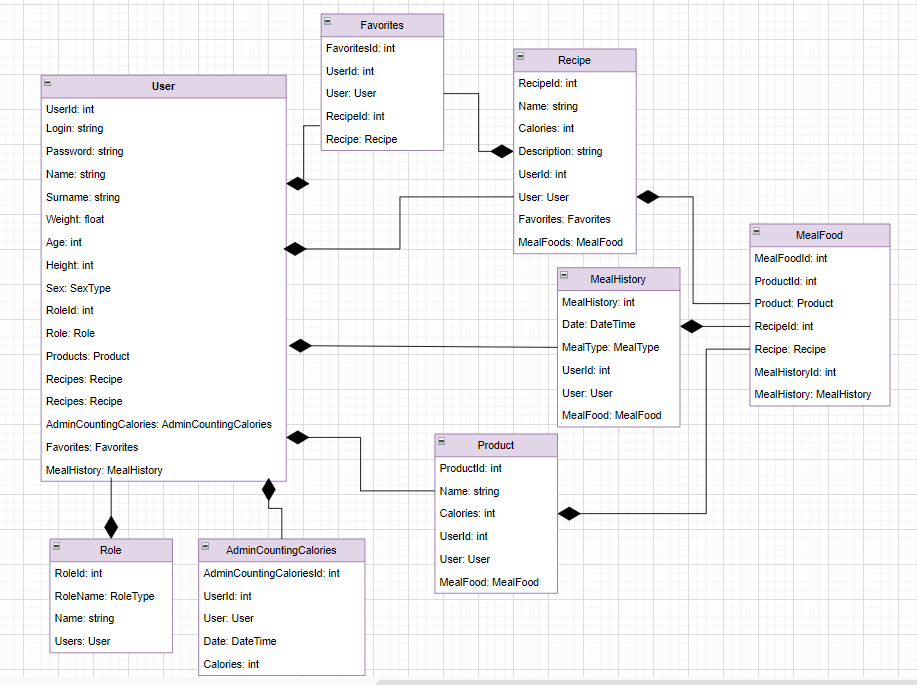
ПРИЛОЖЕНИЕ А

Диаграмма использования



ПРИЛОЖЕНИЕ Б

Диаграмма классов



ПРИЛОЖЕНИЕ В

Листинг 1. Код интерфейса IRepository<T>

using System;

using System.Collections.Generic;

namespace Personal\_Nutritionist.DataLayer.Repository

{

public interface IRepository<TEntity> where TEntity : class

{

void Create(TEntity item);

TEntity FindById(int id);

IEnumerable<TEntity> Get();

IEnumerable<TEntity> Get(Func<TEntity, bool> predicate);

void Remove(TEntity item);

void Update(TEntity item);

}

}

ПРИЛОЖЕНИЕ Г

Листинг 3. Код класса PersonalNavigationService

using Personal\_Nutritionist.Stores;

using Personal\_Nutritionist.ViewModels;

using System;

using System.Windows;

namespace Personal\_Nutritionist.Services

{

public class PersonalNavigationService<TViewModel> where TViewModel : ViewModelBase

{

private readonly PersonalNavigationStore \_personalNavigationStore;

private readonly Func<TViewModel> \_createViewModel;

public PersonalNavigationService(PersonalNavigationStore personalNavigationStore, Func<TViewModel> createViewModel)

{

\_personalNavigationStore = personalNavigationStore;

\_createViewModel = createViewModel;

}

public void Navigate()

{

try

{

var CurrentPersonalViewModel = \_createViewModel();

if (CurrentPersonalViewModel == null)

return;

\_personalNavigationStore.CurrentPersonalViewModel = CurrentPersonalViewModel;

}

catch

{

MessageBox.Show("Can't switch pages");

}

}

}

}

ПРИЛОЖЕНИЕ Д

Листинг 4. Код класса NavigationService

using Personal\_Nutritionist.Stores;

using Personal\_Nutritionist.ViewModels;

using System;

using System.Windows;

namespace Personal\_Nutritionist.Services

{

public class NavigationService<TViewModel> where TViewModel : ViewModelBase

{

private readonly NavigationStore \_navigationStore;

private readonly Func<TViewModel> \_createViewModel;

public NavigationService(NavigationStore navigationStore, Func<TViewModel> createViewModel)

{

\_navigationStore = navigationStore;

\_createViewModel = createViewModel;

}

public void Navigate()

{

try

{

\_navigationStore.CurrentViewModel = \_createViewModel();

}

catch

{

MessageBox.Show("Can't switch pages");

}

}

}

}

ПРИЛОЖЕНИЕ Е

Листинг 5. Код класса CommandBase

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Input;

namespace Personal\_Nutritionist.Command

{

public abstract class CommandBase : ICommand

{

public event EventHandler CanExecuteChanged

{

add { CommandManager.RequerySuggested += value; }

remove { CommandManager.RequerySuggested -= value; }

}

public virtual bool CanExecute(object parameter) => true;

public abstract void Execute(object parameter);

}

}

ПРИЛОЖЕНИЕ Ж

Листинг 6. Код класса NavigationStore

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using Personal\_Nutritionist.ViewModels;

namespace Personal\_Nutritionist.Stores

{

public class NavigationStore

{

public event Action CurrentViewModelChanged;

private ViewModelBase \_currentViewModel;

public ViewModelBase CurrentViewModel

{

get => \_currentViewModel;

set

{

\_currentViewModel?.Dispose();

\_currentViewModel = value;

OnCurrentViewModelChanged();

}

}

private void OnCurrentViewModelChanged()

{

try

{

CurrentViewModelChanged?.Invoke();

}

catch

{

MessageBox.Show("Can't change page");

}

}

}

}

ПРИЛОЖЕНИЕ З

Листинг 7. Код класса PersonalNavigationStore

using Personal\_Nutritionist.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

namespace Personal\_Nutritionist.Stores

{

public class PersonalNavigationStore

{

public event Action CurrentPersonalViewModelChanged;

private ViewModelBase \_currentPersonalViewModel;

public ViewModelBase CurrentPersonalViewModel

{

get => \_currentPersonalViewModel;

set

{

\_currentPersonalViewModel?.Dispose();

\_currentPersonalViewModel = value;

OnCurrentViewModelChanged();

}

}

private void OnCurrentViewModelChanged()

{

try

{

CurrentPersonalViewModelChanged?.Invoke();

}

catch

{

MessageBox.Show("Can't change page");

}

}

}

}