МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Направление специальности 1-40 01 01 Программное обеспечение информационных технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

по дисциплине «Объектно-ориентированное программирование»

Тема Программное средство «Пищевой дневник Evva»

Исполнитель

студент (ка) 2 курса группы 6 Авдеева Вера Дмитриевна

(Ф.И.О.)

Руководитель работы преп.-стажер Север А.С.

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

Курсовой проект защищен с оценкой

Председатель Пацей Н.В.

(подпись)

Минск 2023

**Содержание**

Введение…………………………………………………………………………….....3

1. Аналитический обзор прототипов и литературных источников……………….4
   1. Аналитический обзор источников…………………………………………4
   2. Обзор аналогов……………………………………………………………...4
2. Анализ требований к программному средству и разработка функциональных требований…………………………………………………………………………7
3. Проектирование программного средства………………………………………...9
   1. Проектирование архитектуры приложения………………………………....9
   2. Проектирование базы данных………………………………………………12
4. Реализация программного средства.……………………………………………14
   1. Реализация архитектуры MVVM…………………………………………14
   2. Реализация паттерна Command…………………………………………..14
   3. Реализация паттерна Repository и UnitOfWork………………………….15
   4. Реализация классов Models и ViewModels…………………………….…15
   5. Реализация вспомогательных классов…………………………………...16
   6. Реализация Views …………………………………………………………16
5. Тестирование, проверка работоспособности и анализ полученных результатов……………………………………………………………………….17
6. Руководство по использованию………………………………………………...23

Заключение……………………………………………………………………….….28

Список литературы………………………………………………………………….39

Приложение А……………………………………………………………………….30

Приложение Б………………………………………………………………………..32

Приложение В…………………………………………………………………….….33

**Введение**

Курсовой проект посвящён разработке программного средства «Пищевой дневник Evva», основной целью которого является помощь пользователям в контроле своего рациона питания и подсчёте потребляемых калорий. В настоящее время существует малое количество аналогов, реализованных в качестве десктопных приложений. Большинство таких сервисов представлено в виде мобильных приложений.

Приложение «Пищевой дневник Evva» – это инструмент, который позволяет пользователям отслеживать, сколько калорий они потребляют в течение дня и контролировать свой рацион питания. С помощью этого приложения пользователи могут вносить информацию о потребляемых продуктах, получать рекомендации и статистику по калорийности различных блюд, а также следить за содержанием белков, жиров и углеводов в каждой порции. «Пищевой дневник Evva» помогает пользователю достигнуть своих целей. Приложение имеет простой интерфейс, что делает его легким в использовании для пользователей всех уровней.

Для создания приложения будет использован современный, минималистичный и привлекательный дизайн, который будет удобен для пользователей. Важно создать легкую и приятную атмосферу в приложении, которая поможет пользователям поддерживать мотивацию на протяжении всего процесса подсчета калорий. В приложении будут присутствовать графики, диаграммы и другие элементы дизайна, которые продемонстрируют прогресс пользователей.

В качестве интерфейса приклaдного 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.

Хранение данных осуществляется в Microsoft SQL Server.

1. Аналитический обзор прототипов и литературных источников
   1. **Аналитический обзор источников**

В ходе подготовки пояснительной записки была изучена специальная техническая, учебно-методическая и справочная литература, статьи и материалы, опубликованные в сети интернет.

Принцип создания окон для регистрации и авторизации был взят из статьи «WPF – система авторизации и регистрации». В статье были рассмотрены алгоритм работы системы авторизации и регистрации и пример создания окон регистрации и авторизации.

Алгоритм работы окна авторизации:

1. выполняется проверка заполняемости всех полей;
2. выполняется проверка существования записи в базе данных;
3. если запись существует, пользователь авторизуется.

Алгоритм работы окна регистрации:

1. выполняется проверка заполняемости всех полей;
2. выполняется проверка существования записи с таким же логином в базе данных;
3. если записи не существует, добавляется новая запись.

Принцип работы с SQL были получены из статьи «Подключение к базе данных». В статье было рассмотрено подключение необходимых библиотек, работа с SQL.

Концепция создания отчета была получена из статьи «Report-Driven Design». В статье была рассмотрена общая информация о создании отчёта и пример создания отчёта.

Дополнительная информации о принципах работы с WPF была получена из интернет-источника «Metanit», содержащего практические советы по работе с технологией.

* 1. **Обзор аналогов**

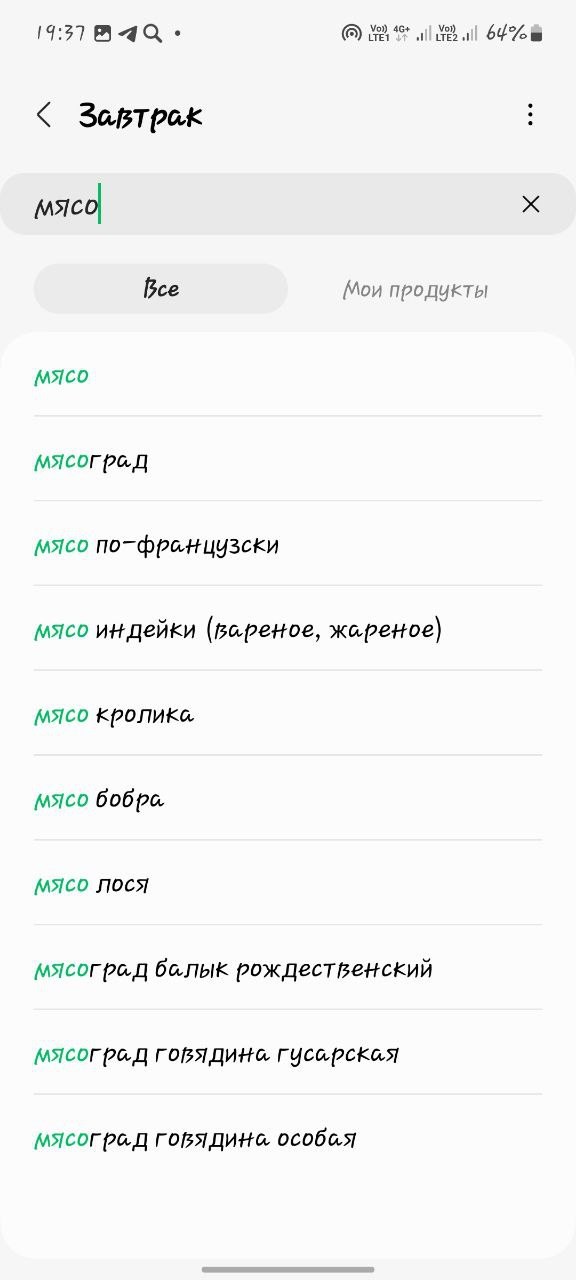
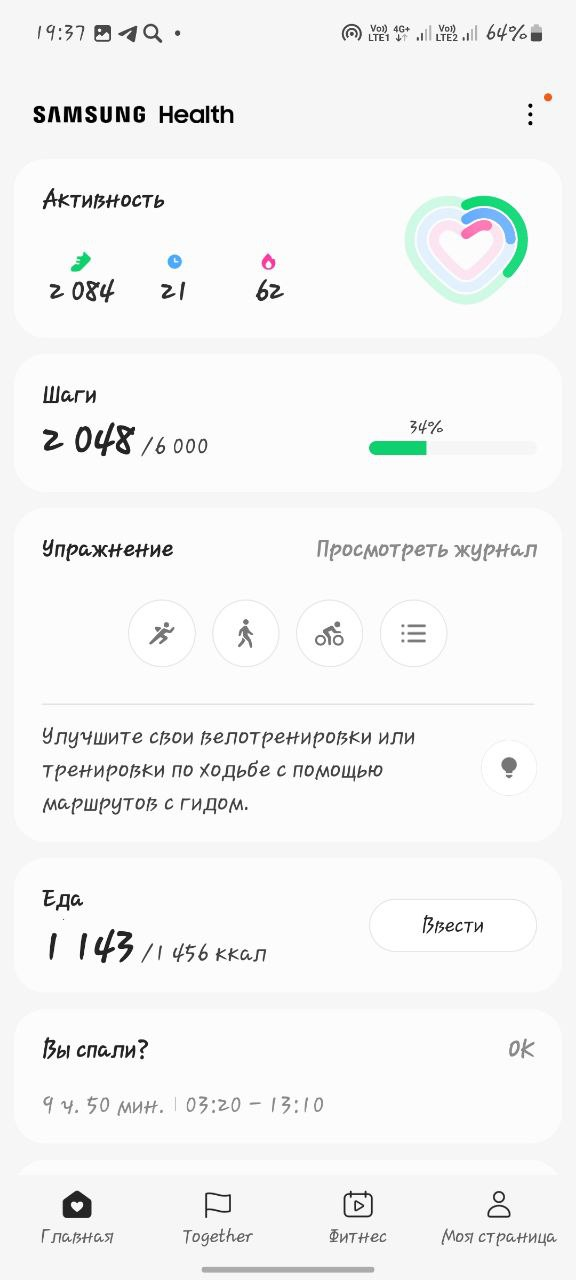
Для создания принципиально нового решения в виде программного продукта, используемого для контроля рациона пользователя, необходимо проанализировать уже существующее программные средства в данной сфере. Анализ достоинств и недостатков этих аналогов позволит сформировать требования к проектируемому программному средству, учитывающие опыт существующих разработок и внести в них улучшения или изменения.

В качестве исследуемых аналогов были выбраны программные продукты, которые можно отнести к группе электронных ежедневников, блокнотов, «todo-листов» и приложений, предназначенных для непосредственного для трекинга пищевых привычек, как наиболее близкие по области применения к разрабатываемому программному средству. Источником информации послужили электронные базы в сети Интернет.

В результате поиска были обнаружены следующие ресурсы:

* «Samsung Health» – это программа, которая изначально создавалась для работы с фитнес-трекерами и смарт-часами от Samsung. Но в дальнейшем в неё был внедрён целый комплекс функций для отслеживания состояния здоровья и предоставления рекомендации по соблюдению ЗОЖ.

Приложение имеет простой и интуитивно понятный интерфейс, что делает приложение привлекательным для пользователя. Интерфейс приложения представлен на рисунке 1.1

Рисунок 1.1 – Интерфейс «Samsung Health»

Функционал данного программного средства включает в себя:

1. Подсчет количества пройденных шагов. Информация берётся из носимого гаджета Samsung или из встроенного гироскопа/акселерометра. При расчёте применяются и GPS-данные, чтобы подсчёт шагов не выполнялся, например, во время поездки на авто или велосипеде.
2. Отслеживание изменений веса. Данные вводятся либо самим пользователем, либо берутся из весов с поддержкой Bluetooth или Wi-Fi-соединения.
3. Отслеживание режима сна. Также программа предоставляет статистику физической активности, при наличии фитнес-трекера или смарт-часов добавляет функцию «умного» будильника (включается, когда пользователь находится в стадии неглубокого сна).
4. Расчёт количества употреблённых и сожжённых килокалорий. Для анализа берутся данные из персонально составленного меню.
5. Контроль за уровнем сахара в крови. Данные также нужно вводить самостоятельно или использовать совместимый носимый гаджет.
6. Контроль за изменением кровяного давления. Работает только при подключении умных часов, поддерживающих измерение АД. Показатели можно вводить и самостоятельно (измеряя давление тонометром).

Следующим рассмотренным аналогом является десктопное приложение «Fit Diary» – приложение от Microsoft, позволяющее вести подсчёт калорий. Есть возможность добавлять в базу свои продукты и упражнения, вести учёт потребления воды и изменения веса. Также программное средство предоставляет подробные графики учёта потребления БЖУ и калорий, воды, изменения веса и физических параметров. У приложения слишком простой и неброский интерфейс, что может сыграть в минус, так как большинство пользователей принимают во внимание внешнее составляющее приложения. Пример интерфейса представлен на рисунке 1.2

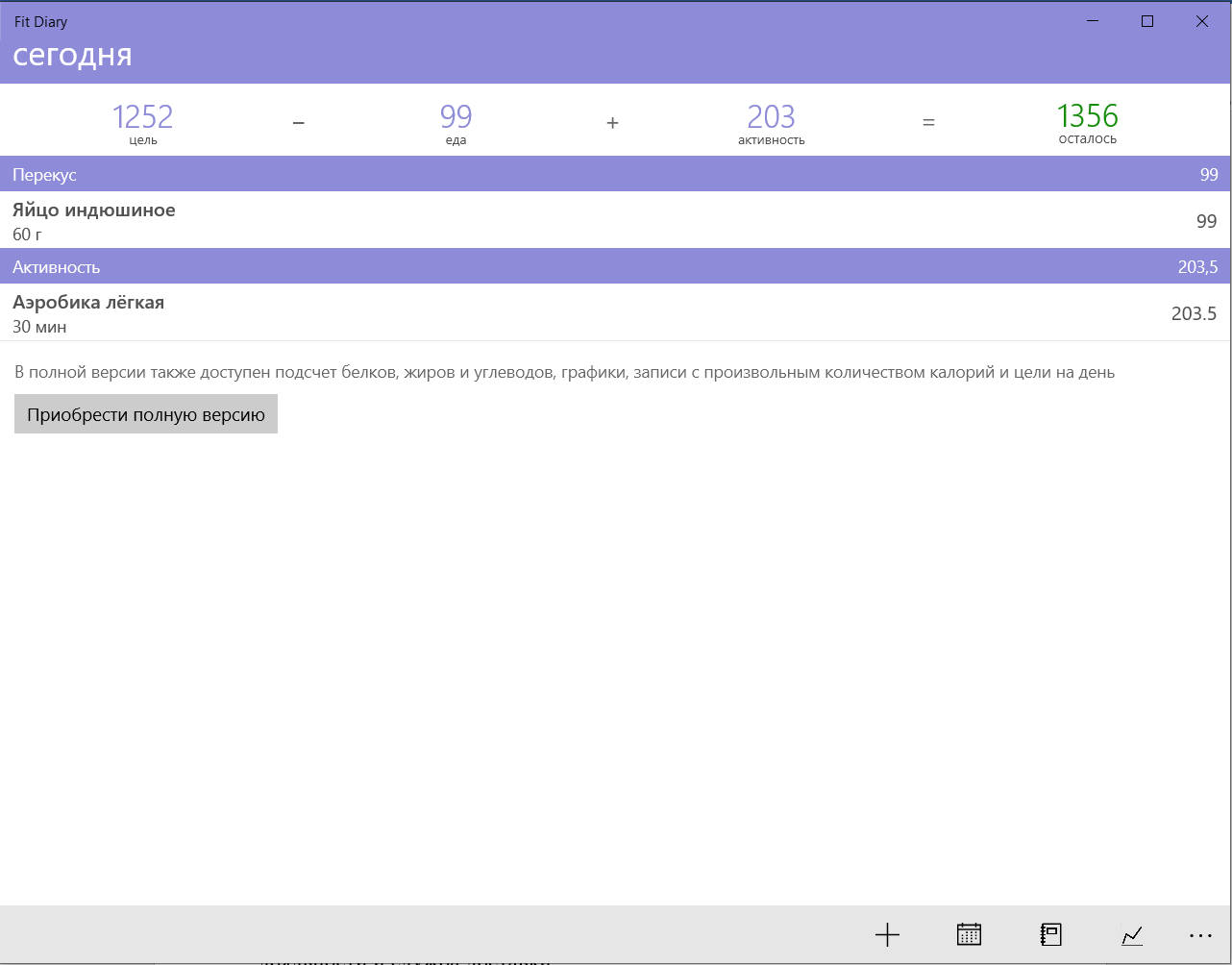


Рисунок 1.2 – Интерфейс «Fit Diary»

Таким образом были изучены аналоги приложения, сделаны выводы по разработке необходимого функционала, а также приняты решения по внешнему оформлению программного средства.

1. Анализ требований к программному средству и разработка функциональных требований

Анализ требований — это процесс сбора требований к программному обеспечению, их систематизации, документирования, анализа, выявления противоречий, неполноты, разрешения конфликтов в процессе разработки программного обеспечения.

Цель анализа требований в проектах — получить максимум информации о заказчике и специфике его задач, уточнить рамки проекта, оценить возможные риски. На этом этапе происходит идентификация принципиальных требований методологического и технологического характера, формулируются цели и задачи проекта, а также определяются критические факторы успеха, которые впоследствии будут использоваться для оценки результатов внедрения. Определение и описание требований — шаги, которые во многом определяют успех всего проекта, поскольку именно они влияют на все остальные этапы.

Различают три уровня требований к проекту:

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

Бизнес-требования содержат высокоуровневые цели организации или заказчиков системы. Как правило, их высказывают те, кто финансируют проект, покупатели системы, менеджер реальных пользователей, отдел маркетинга. Курсовой проект не подразумевает наличие заказчика, который мог бы выдвинуть бизнес-требования, поэтому в качестве таких высокоуровневых требований можно рассматривать общие требования к разрабатываемому средству. К их числу относятся:

* простота и лёгкость интерфейса;
* использование принципов объектно-ориентированного программирования;
* использование архитектурных шаблонов проектирования;
* использование системы управления базами данных (СУБД);

Весь дальнейший процесс проектирования и разработки программного средства должен находиться в очерченных бизнес-требованиями границах.

Следующими требованиями являются требования пользователей. Данные требования описывают цели и задачи, которые пользователям позволит решить система. Таким образом, в пользовательских требованиях указано, что клиенты смогут делать с помощью системы.

* Пользователь данного программного решения должен иметь возможность:
* регистрировать себя в системе;
* входить в приложение, после ввода данных, необходимых для аутентификации;
* просматривать свой ежедневный прогресс на главной;
* просматривать продукты в списке;
* добавлять продукты в список потребленных;
* удалять продукты из списка потребленных;
* изменение роста и веса;
* возможность добавление отсутствующего съеденного продукта;

Администратор имеет возможность:

* выполнять действия пользователя;
* вносить новую информацию о продуктах;
* удалять информацию о продуктах;
* просматривать информацию о пользователях.

После проведения анализа были выявлены следующие функциональные требования:

* архитектура приложения должна соответствовать шаблонам проектирования, таким как MVVM, Command;
* вся информация должна храниться в базе данных;
* приложение должно производить валидацию вводимых пользователем данных;
* приложение должно корректным образом обрабатывать возникающие исключительные ситуации: отображать понятное для пользователя сообщение о возникшей ошибке;
* приложение должно предоставлять пользователям возможность создания нового аккаунта в виде регистрационной формы;
* приложение должно предоставлять возможность пользователям проходить аутентификацию и входить в систему под соответствующим введенным данным пользовательским именем;
* приложение должно предоставлять возможность поиска продуктов по следующим критериям: название.

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.

1. **Проектирование программного средства**

**3.1 Проектирование архитектуры приложения**

Архитектура программного обеспечения — совокупность вaжнейших решений об организации программной системы. Архитектура включает:

* выбор структурных элементов и их интерфейсов, с помощью которых составлена система, а также их поведения в рамках сотрудничества структурных элементов;
* соединение выбрaнных элементов структуры и поведения во всё более крупные системы;
* архитектурный стиль, который направляет всю организацию — все элементы, их интерфейсы, их сотрудничество и их соединение.

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 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. Поэтому для хорошего проектирования View необходимо понять, как пользователь будет взаимодействовать с приложением. Для этого была составлена схема на рисунке 3.2, на которой представлен принцип работы приложения с точки зрения пользователя.



Рисунок 3.2 – Диаграмма использования

На рисунке 3.3 была отображена диаграмма последовательности для авторизации. На диаграмме последовательности отображаются только те объекты, которые непосредственно принимают участие во взаимодействии.



Рисунок 3.3 – Диаграмма последовательности для авторизации

На рисунке 3.4 изображена диаграмма компонентов приложения.



Рисунок 3.4 – Диаграмма компонентов

На рисунке 3.5 представлена блок-схема для алгоритма регистрации пользователя.



Рисунок 3.5 – Схема алгоритма регистрации

Логика перехода между страницами реализована в классе MainWindow. Для получения и записи данных в БД используются классы DietManagerContext, DietManagerDBRepository и UnitOfWork, которые создают достаточный уровень абстракции для быстро доступа к БД. Для создания подключения и работы с БД используется технология Entity Framework Core. Для администрирования приложения путём доступа к БД был создан пользователь с логином admin, обладающий правами администратора.

3.2. Проектирование базы данных

Для данного курсового проекта была спроектирована база данных. Для управления базой данных используется ПО Microsoft SQ Server 2022. Логическая схема данных представлена на рисунке 3.6.



Рисунок 3.6 – Логическая схема данных

В базе данных находится 6 таблиц. В таблице Users хранятся данные для идентификации пользователей, зарегистрированные в приложении, UsersData – личный данные зарегистрированных пользователей, UsersParams – список отчётов о физических параметрах пользователей, Reports – список отчётов употребления продуктов пользователями, Products – список продуктов, добавленных в коллекцию продуктов пользователями, и FoodCategory, которая содержит все возможные категории продуктов.

Таблица Users состоит из следующих столбцов:

* Id;
* IsAdmin;
* UserLogin;
* UserPassword;
* Salt.

Столбец UserLogin содержит логин пользователя, UserPassword – хешированный алгоритмом SHA1 пароль, Salt – соль, используемая при хешировании пароля, IsAdmin – логическое указание о том, является ли пользователь администратором.

Таблица UsersData состоит из столбцов:

1. Id;
2. IdData;
3. FullName;
4. Birthday.

Столбец IdData содержит идентификатор пользователя, которому соответствует запись в таблице, FullName – ФИО пользователя, Birthday – дату дня рождения пользователя.

Таблица UsersParams состоит из столбцов:

1. Id;
2. IdParams;
3. ParamsDate;
4. UserWeight;
5. UserHeight.

В столбце IdParams хранится идентификатор пользователя, которому соответствует запись в таблице, ParamsDate – дата занесения отчёта физических характеристиках пользователя, UserWeight – масса тела пользователя, UserHeight – рост пользователя.

Таблица Products включает столбцы:

1. Id;
2. IdAdded;
3. ProductName;
4. CaloriesGram;
5. ProteinsGram;
6. FatsGram;
7. CarbohydratesGram;
8. FoodCategory.

Столбец IdAdded содержит идентификатор пользователя, добавившего продукт в коллекцию, ProductName – название продукта, CaloriesGram ­­– количество калорий на сто грамм продукта, ProteinsGram – количество грамм белков на сто грамм продукта, FatsGram – количество грамм жиров на сто грамм продукта, CarbohydratesGram – количество грамм углеводов на сто грамм продукта, FoodCategory – категория продукта.

Таблица Reports включает столбцы:

1. Id;
2. IdReport;
3. ProductName;
4. ReportDate;
5. EatPeriod;
6. DayGram;
7. DayCalories;
8. DayProteins;
9. DayFats;
10. DayCarbohydrates;
11. MostCategory.

Столбец IdReport содержит идентификатор пользователя, добавившего отчёт о потреблённом в пищу продукте, ProductName – название потреблённого пользователем продукта в пищу, ReportDate – время и дата внесения отчёта пользователем, EatPeriod – название периода приёма пищи, DayGram – количество грамм потреблённого продукта, DayCalories – количество калорий на количество грамм потреблённого продукта, DayProteins – количество грамм белков на количество грамм потреблённого продукта, DayFats – количество грамм жиров на количество грамм потреблённого продукта, DayCarbohydrates – количество грамм углеводов на количество грамм потреблённого продукта, MostCategory – категория употреблённого продукта.

Скрипты создания базы данных приложения представлены в Приложении А.

1. **Реализация программного средства**

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

4.1 Реализация архитектуры MVVM

Для реализации паттерна MVVM файлы программы были распределены по соответствующим директориям и реализовали соответствующие функции. Разделение проекта на логические модули представлено на рисунке 4.1.

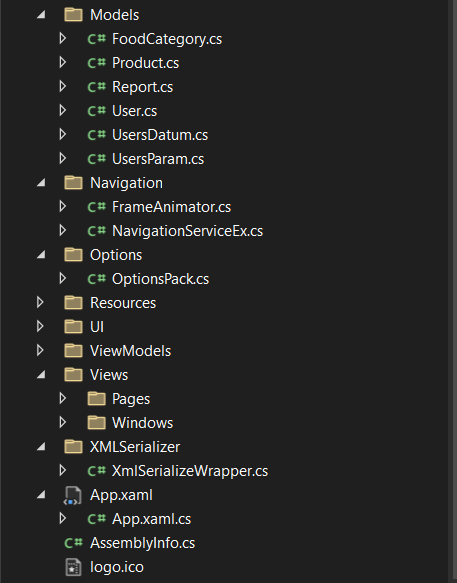
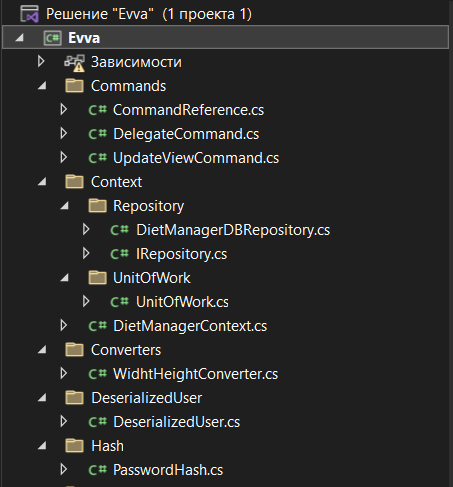


Рисунок 4.1 – Логические модули проекта

Архитектура MVVM (Model-View-ViewModel) является одним из популярных паттернов архитектуры для проектов, связанных с разработкой пользовательских интерфейсов. Модель представляет данные в приложении, Представление отображает данные пользователю, а ViewModel является связующим звеном между Моделью и Представлением, обрабатывая логику, связанную с UI и взаимодействуя с Моделью для запроса и обновления данных.

4.2 Реализация паттерна Command

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 трех классов: CommandReference, DelegateCommand и UpdateViewCommand.

Класс DelegateCommand позволяет делегировать командную логику методами, передаваемыми в качестве параметров, и позволяет View связывать команды с объектами, которые не являются частью дерева элементов. Также содержит в себе класс CommandManagerHelper, содержащий методы для CommandManager, которые помогают избежать утечки памяти, используя слабые ссылки.

Класс CommandReference облегчает привязку ключа в разметке XAML к команде. Определяется в модели представления путем предоставления свойства зависимости от команды. Класс является производным от Freezable, чтобы обойти ограничение в WPF, когда привязка данных из XAML.

Класс UpdateViewCommand используется для привязки команд ViewModel к кнопкам на XAML документах, которые используются для изменения ViewModel окна.

4.3 Реализация паттерна Repository и UnitOfWork

В папке Context расположены файлы, необходимые для работы с базой данных. Реализовано 2 паттерна для работы с базой данных. В папке Repository расположен класс DietManagerDBRepositiry и интерфейс IRepository. В папке UnitOfWork расположен одноименный класс.

Класс DietManagerDBRepositiry является реализацией паттерна репозиторий. Репозиторий 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. Листинг класса представлен в Приложении В.

Класс UnitOfWork является реализацией паттерна UnitOfWork для приложения. Паттерн UnitofWork позволяет упростить работу с различными репозиториями и дает уверенность, что все репозитории будут использовать один и тот же контекст данных. Листинг класса представлен в Приложении Б.

В любом приложении, работающим с БД через Entity Framework Core, необходимо использовать контекст (класс производный от DbContext) и набор данных DbSet, через который можно взаимодействовать с таблицами из БД. В данном случае таким контекстом является класс DietManagerContext.

4.4 Реализация классов Models и ViewModels

В папке Models расположены сущностные классы, которые используются для создания БД. Этими классами являются FoodCategory, Product, Report, User, UsersParam, UsersDatum

Классы с суффиксом ViewModel содержат: закрытые поля, открытые свойства и команды, привязываемые к элементам управления на страницах View. ViewModel также содержит логику по получению данных из модели, которые потом передаются в представление. И также VewModel определяет логику по обновлению данных в модели.

4.5 Реализация вспомогательных классов

В проекте было реализовано несколько дополнительных вспомогательных классов для расширения функционала приложения.

Класс PasswordHash содержит методы для генерации соли, определения, является ли пароль, введённым пользователем, тем, который хранится в базе данных приложения, и для хеширования паролей. Листинг класса представлен в Приложении Б.

Классы DeserializedUser и XMLSerialiseWrapper содержат в себе методы, позволяющие сериализовать текущего пользователя и его настройки для сохранения текущей сессии. Реализовано для облегчения использования.

Классы из папки Navigation с названиями FrameAnimator и NavigationServiceEx являются библиотечными классами из MahApps.Merto.

Класс FrameAnimator используется для подключения анимации при перемещении между страницами главного окна.

Класс NavigationServiceEx инкапсулирует возможность загрузки содержимого в контексте навигации в стиле браузера.

**4.6 Реализация Views**

Для разработки графической части приложения была выбрана технология WPF.

Windows Presentation Foundation (WPF) — это библиотека для создания пользовательских интерфейсов для интеллектуальных клиентских приложений

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 реализовано три окна:

1. стартовое окно (регистрация и вход в систему);
2. главное окно (содержит различные страницы);
3. окно изменения продукта.

А также девять страниц:

1. страница для авторизации;
2. страница для регистрации;
3. страница с настройками;
4. главная страница с общей информацией на тему здоровья;
5. страница для просмотра потребленных калорий;
6. страница для добавления съеденного продукта;
7. страница для отслеживания веса;
8. страница для просмотра и изменения употребленного за день;
9. страница администратора (для администратора).

В результате выполнения данного этапа было создано функционирующее программное средство.

1. Тестирование, проверка работоспособности и анализ полученных результатов

Основной целью тестирования приложения было стремление доказать невозможно введения пользователем данных, которые бы могли привести приложение в неработоспособное состояние. Были проведены:

1. тесты на валидацию данных при авторизации и регистрации;
2. тесты на невозможность включения продуктов в список продуктов при вводе неверной информации;
3. тесты на невозможность добавления отчёта об употреблённом продукте при вводе неверной информации;
4. тесты на невозможность удаления всех отчётов о физических показателях пользователя;
5. тесты на невозможность добавления отчёта о физических показателях пользователя при вводе неверной информации;
6. тесты на невозможность изменения пароля при вводе неверных данных.

В момент регистрации возможна ситуация, когда пользователь вводит уже существующий логин. Обработка данного исключения продемонстрирована на рисунке 5.1.

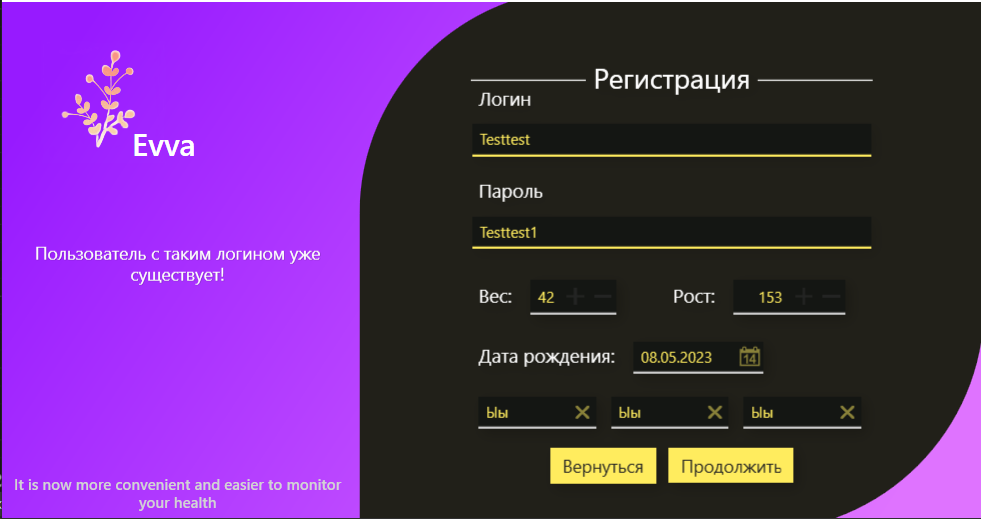


Рисунок 5.1 ­– Пользователь с таким логином уже существует

Также предусмотрены границы допустимых значений, вводимых в числовые поля. Таким образом, при вводе значения, ниже или выше предполагаемых границ, оно устанавливается в наиболее близкое из доступных.

До того момента, пока пользователь не заполнит верно все данные, кнопка подтверждения регистрации не будет доступна. Подобное ограничение реализовано в листинге 5.1.

private bool CanRegister()

{

if(UserLogin == "" || UserPassword == "" || UserWeight == 0 || UserHeight == 0 || UserName == "" || UserLastname == "" || UserSurname == "")

{

return false;

}

else if(DateToChoose.CompareTo(DateTime.Today.Date) >= 0 )

{

return false;

}

else if (!Regex.IsMatch(UserLogin, "^([a-z]|[A-Z]|[0-9]){5,20}$") ||

!Regex.IsMatch(UserPassword, "^([a-z]|[A-Z]|[0-9]){8,20}$")||

!Regex.IsMatch(UserName, "^[А-Я]{1}[а-я]{1,99}$") ||

!Regex.IsMatch(UserLastname, "^[А-Я]{1}[а-я]{1,99}$") ||

!Regex.IsMatch(UserSurname, "^[А-Я]{1}[а-я]{1,99}$"))

{

return false;

}

else

{

return true;

}}

Листинг 5.1 – метод, ограничивающий доступ к команде, привязанной к кнопке

Предупреждение о неверно введённом пароле или логине продемонстрировано на рисунке 5.2.

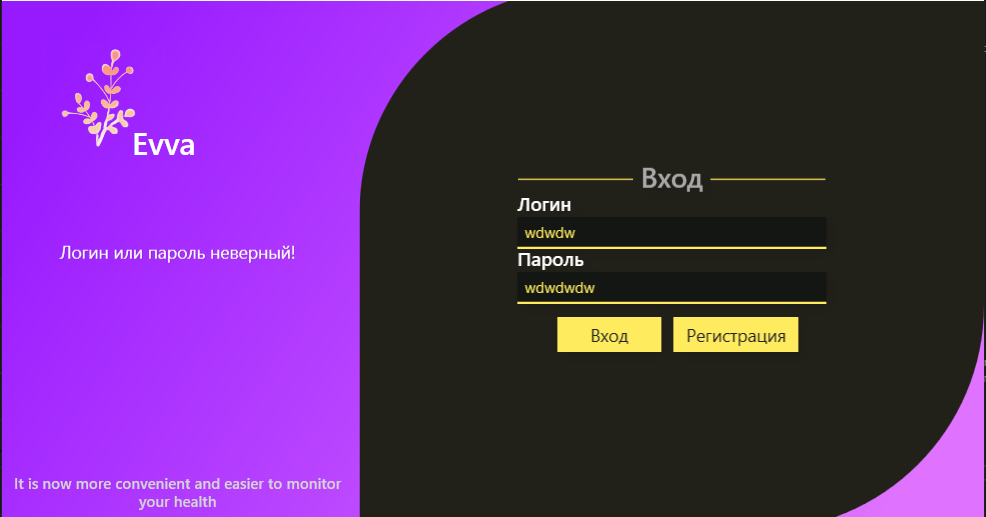


Рисунок 5.2 – Логин или пароль неверный

После авторизации пользователь попадает на домашнюю страницу, которая представляет из себя страницу c общей информацией о здоровом образе жизни

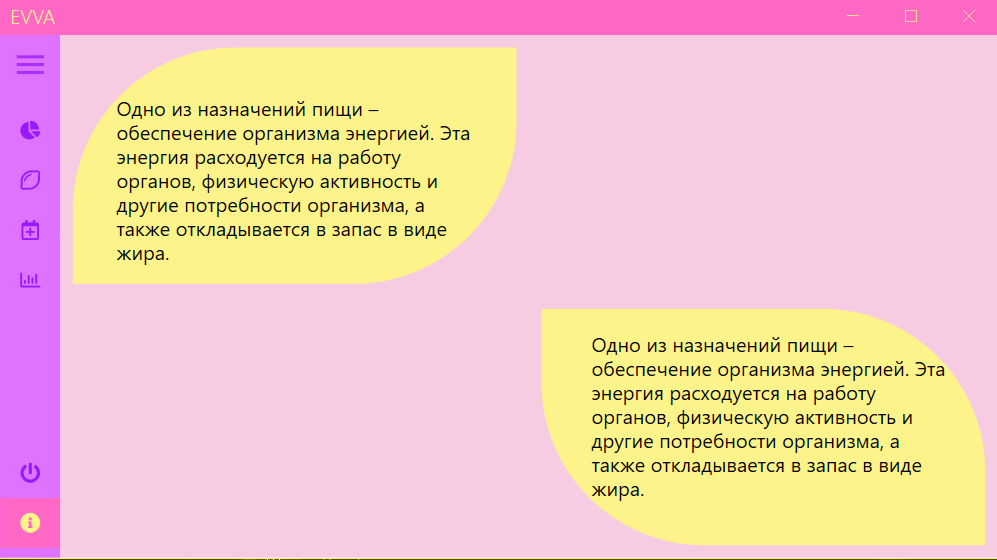


Рисунок 5.3 – Окно с информацией

При попытке добавить продукт в список употреблённых продуктов при дате, установленной неверно, либо при массе продукта, меньше либо равно одному грамму, кнопка добавления блокируется, что не даёт пользователю сделать неверный шаг. Результат метода проверки на рисунке 5.4. При добавлении отчётов о физических параметрах пользователя и продуктов в коллекцию продуктов предусмотрены ограничений, накладываемые на текстовые поля, которые ограничивают доступ к кнопкам на странице. Накладываются ограничение на значения, вводимые в поля, и их отсутствие хотя бы в одном из полей. Данное ограничение представлено на рисунке 5.4.

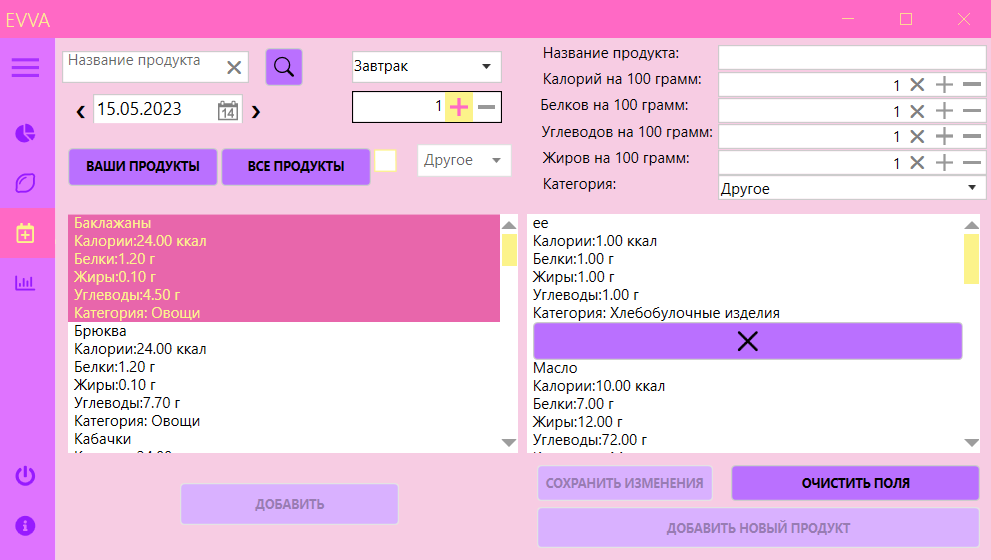


Рисунок 5.4 – Добавление продукта, добавление нового продукта

На листинге 5.2 представлен метод, накладывающий одно из ограничений.

private bool canAddProductToCollection()

{

bool productExist = false;

using (UnitOfWork unit = new UnitOfWork())

{

List<Product> productsList = unit.ProductRepository.Get().ToList();

foreach (Product x in productsList)

{

if (x.ProductName.ToLower() == LastSelected.ProductName.ToLower())

{

productExist = true;

break;

}

}

}

if (ProductName == "" || Calories == 0 || Proteins == 0 || Fats == 0 || Carbohydrates == 0 || SelectedCategory == "" || productExist || LastSelected.Id != 0)

{

return false;

}

else

{

return true;

}

}

Листинг 5.2 – метод, ограничивающий доступ к команде, привязанной к кнопке добавления продукта в коллекцию

Подобное ограничение реализовано в листинге 5.3.

private bool canAddProduct()

{

if (SelectedProduct == null || GramValue == 0 || DateToChoose.Date.CompareTo(DateTime.Now.Date) > 0 )

{

return false;

}

Else {

return true;}

}

Листинг 5.3 – метод, ограничивающий доступ к команде, привязанной к кнопке добавления продукта

Предусмотрена возможность того, что пользователь захочет удалить единственный оставшийся отчёт о его физических параметрах. В таком случае, кнопка удаления отчёта блокируется. На рисунке 5.5 отображено окно статистики.

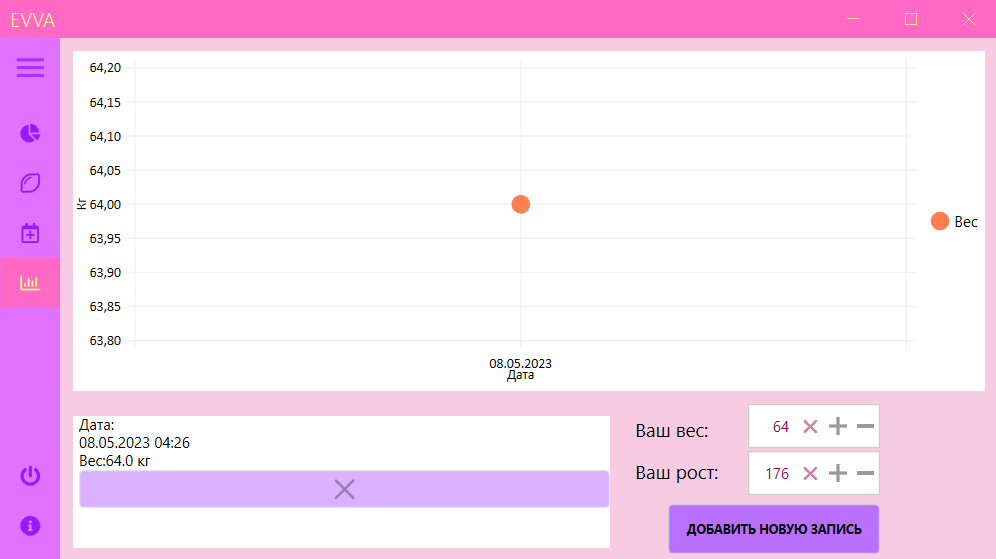


Рисунок 5.5 – Невозможно удалить последний отчёт

На листинге 5.4 представлен метод, накладывающий данное ограничение. В коллекции отчётов должно находится не менее одного элемента.

private bool canDeleteWeightRow()

{

if(StatisticCollection.Count() >1)

{

return true;

}

else

{

return false;

}}

Листинг 5.4 – метод, ограничивающий доступ к команде, привязанной к кнопке удаления отчёта

В окне настроек у пользователя есть возможность сменить старый пароль на новый. На новый пароль накладываются ограничения по количеству символов и символьному содержанию. Валидационное сообщение выводится в окне, расположенном непосредственно на данной странице, что представлено на рисунке 5.7.

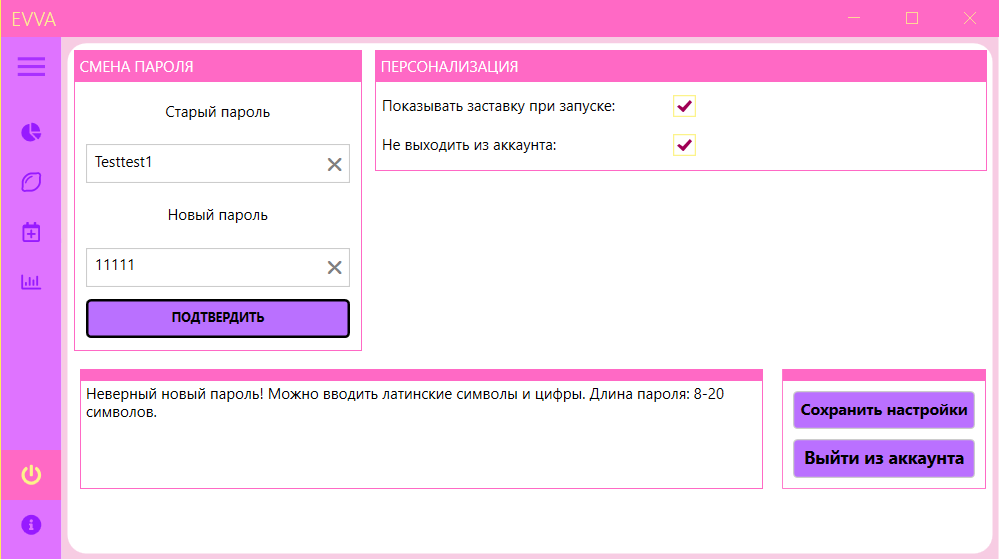
****

Рисунок 5.7 – Введён неверный новый пароль

Также предусмотрено валидационное сообщение в случае неверного ввода старого пароля, что представлено на рисунке 5.8.

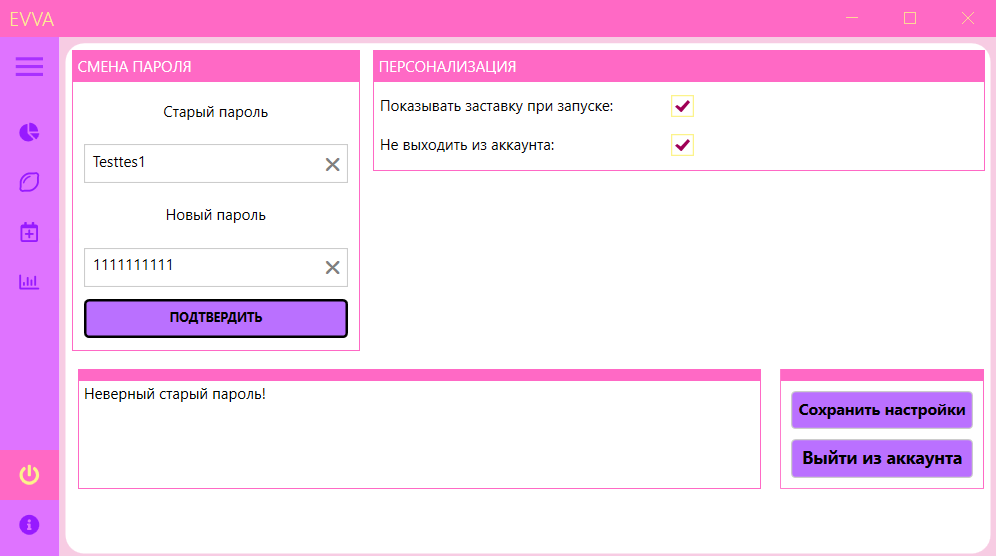


Рисунок 5.8 – Введён неверный старый пароль

Таким образом была реализована валидация во всех местах ввода пользовательских данных.

# **Руководство по использованию**

Окно входа содержит два поля для ввода логина и пароля. После ввода верных данных, необходимо нажать на кнопку вход. Окно входа в аккаунт представлено на рисунке 6.1.

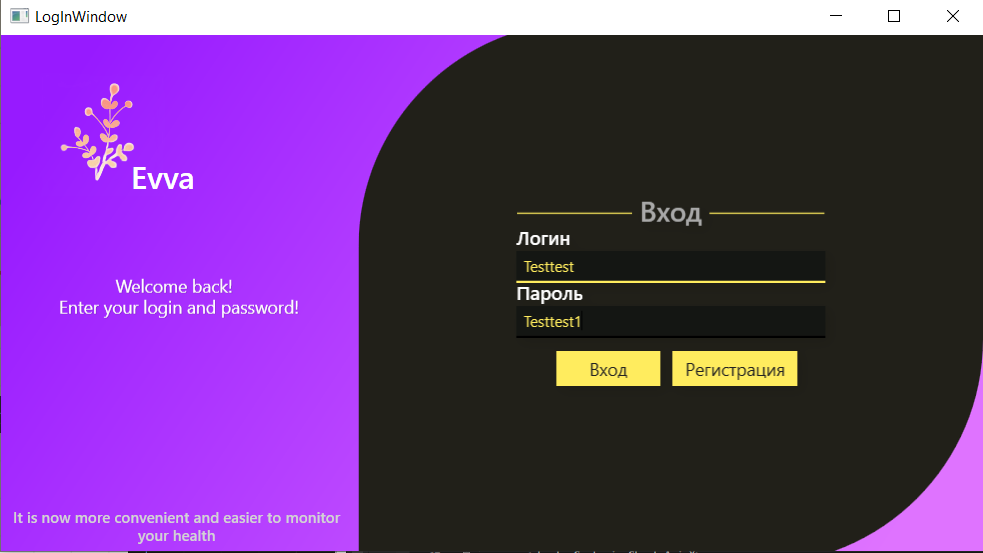


Рисунок 6.1 – Окно входа в аккаунт

Окно регистрации содержит поля для ввода логина, пароля, даты Дня рождения, массы и роста пользователя (для формирования первого отчёта о физических параметрах), имени, фамилии и отчества. Данная информацию в дальнейшем используется для расчёта нормы потребления питательных веществ пользователем. Окно регистрации представлено на рисунке 6.2.

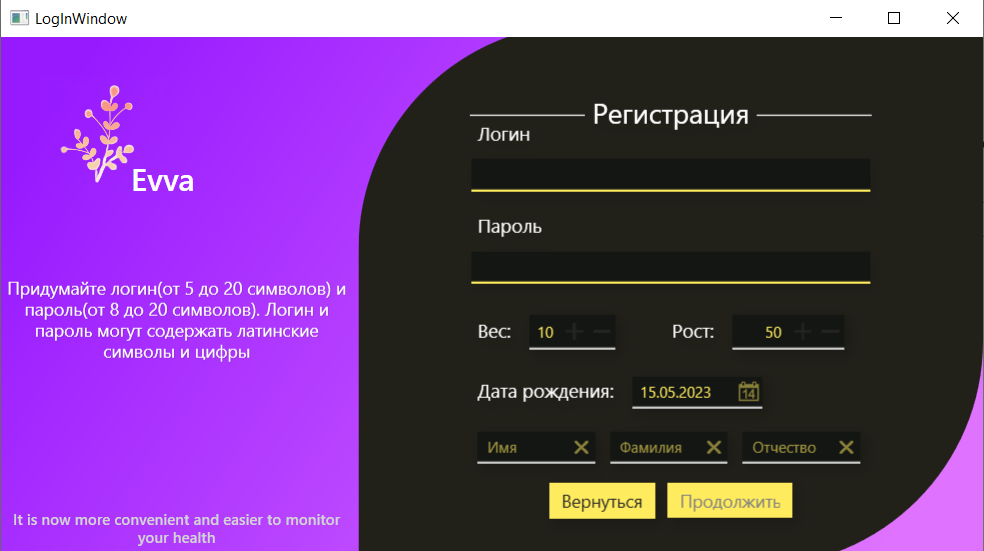


Рисунок 6.2 – Окно регистрации

Первая страница - страница результатов за последний день. Данная страница содержит информацию о количестве уже потреблённых и необходимых для потребления питательных веществ, время последнего изменения потреблённых продуктов, основную категорию потреблённых продуктов за день, информацию о росте и массе пользователя. Присутствует возможность просматривать информацию за предыдущие дни, изменяя дату при помощи элементов управления. Данная страница представлена на рисунке 6.3.

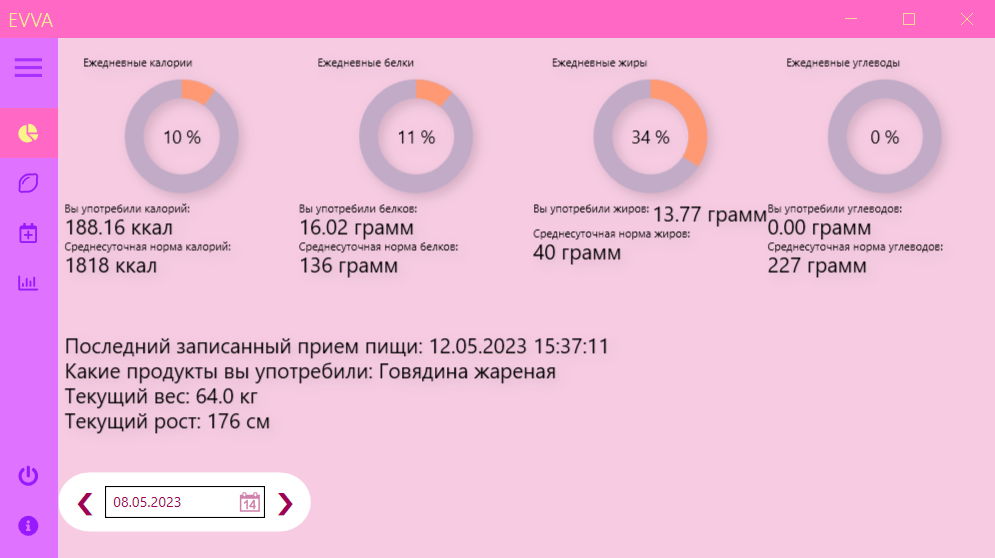


Рисунок 6.3 – Страница результатов за последний день

На странице питания распложена информация о последних употреблённых за день продуктах. Страница питания представлена на рисунке 6.4.

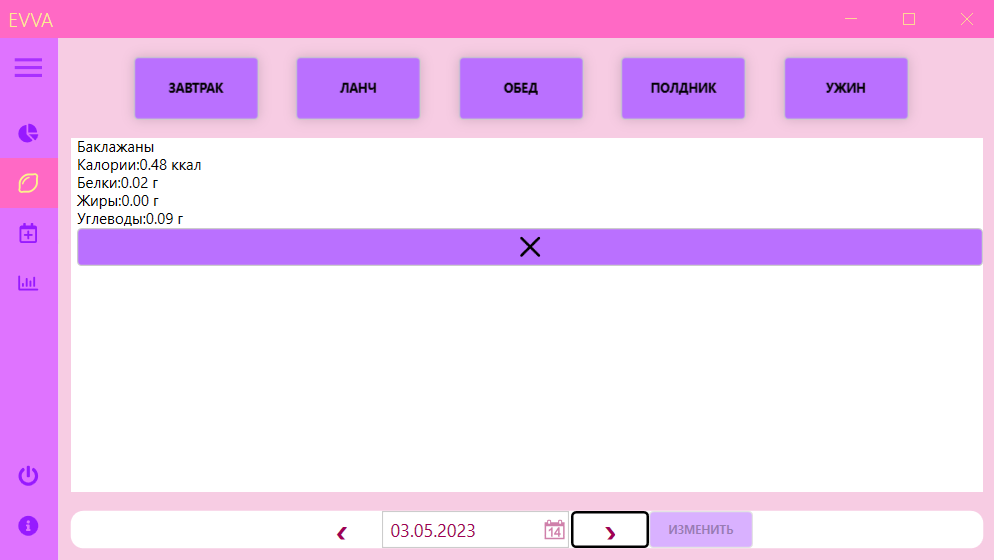


Рисунок 6.4 – Страница питания

Присутствует возможность удаления продуктов из списка и просмотра информации за предыдущие дни. Информация о приёмах пищи разделена на пять разделов, соответствующих различным периодам приёма пищи.

При выборе продукта и нажатии на кнопку «Изменить» появляется модальное окно, позволяющее пользователю изменить данные, внесённые в таблицу. На протяжении всей работы с данным окном содержимое основного окна блокируется. Данное окно представлено на рисунке 6.5.

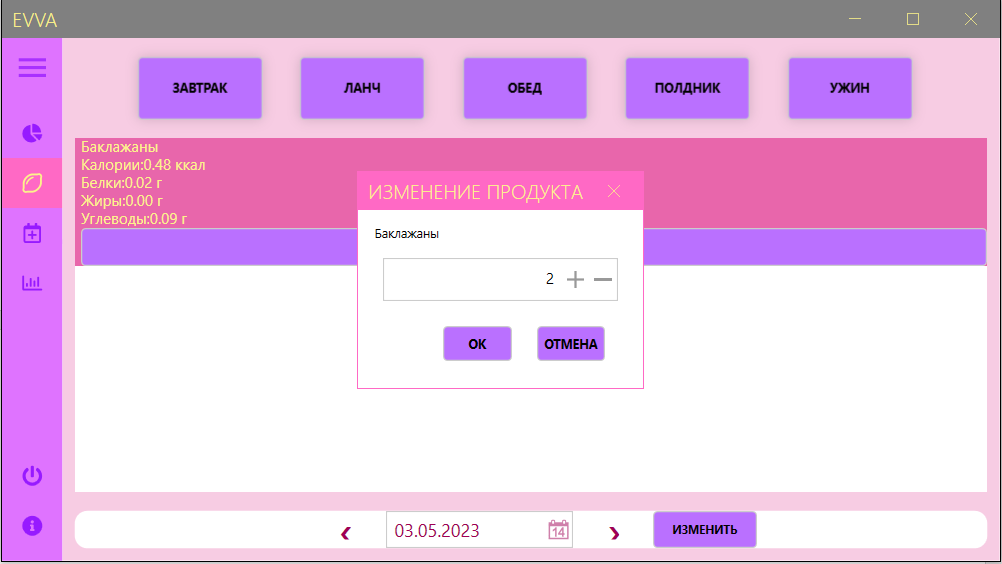


Рисунок 6.5 – Окно изменения внесённой информации

Страница добавления продукта необходима для добавления отчётов об употреблённых пользователем продуктах. Можно осуществлять поиск продуктов по названию, категории, просматривать всю коллекцию продуктов либо личную коллекцию пользователя. Можно выбрать день и период приёма пищи, указать количество грамм употреблённого продукта при добавлении отчёта. Также страница содержит интерфейс для добавления, удаления и редактирования внесённых продуктов. Данная страница представлена на рисунке 6.6.

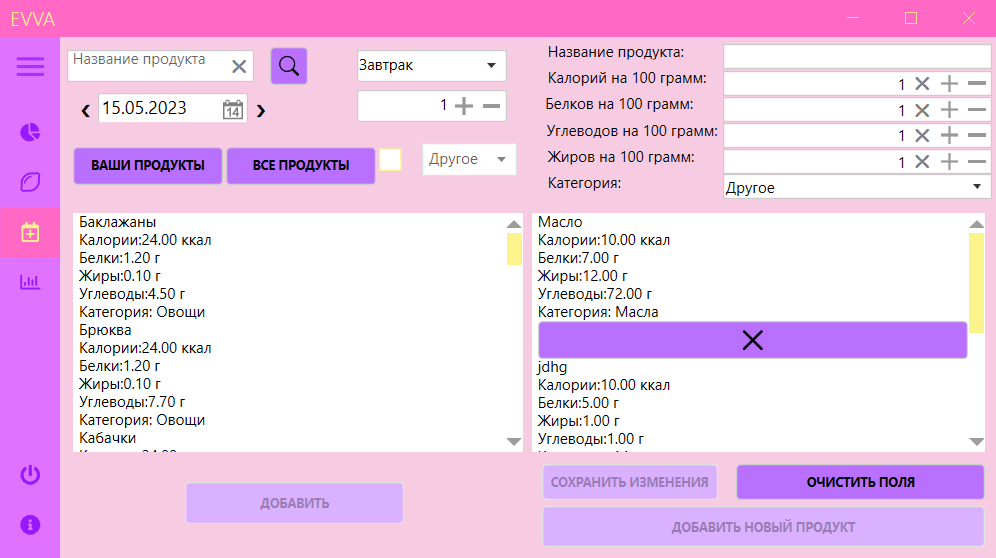
****

Рисунок 6.6 – Страница добавления продукта и добавление нового продукта

Страница статистики содержит часть информации, располагаемой на странице результатов за текущий день, график динамики изменения массы пользователя, таблицу, содержащую информацию о каждом из внесённых пользователем отчёта о его физических параметрах, а также используется для внесения отчётов о физических параметрах пользователя. Каждый из внесённых отчётов можно удалить, кроме одного, необходимого для расчёта требуемого пользователю количества питательных веществ. Данная страница используется для более чёткого представления пользователя о его успехах, а также позволяет проследить динамику изменения его физических параметров. Страница статистики представлена на рисунке 6.7.

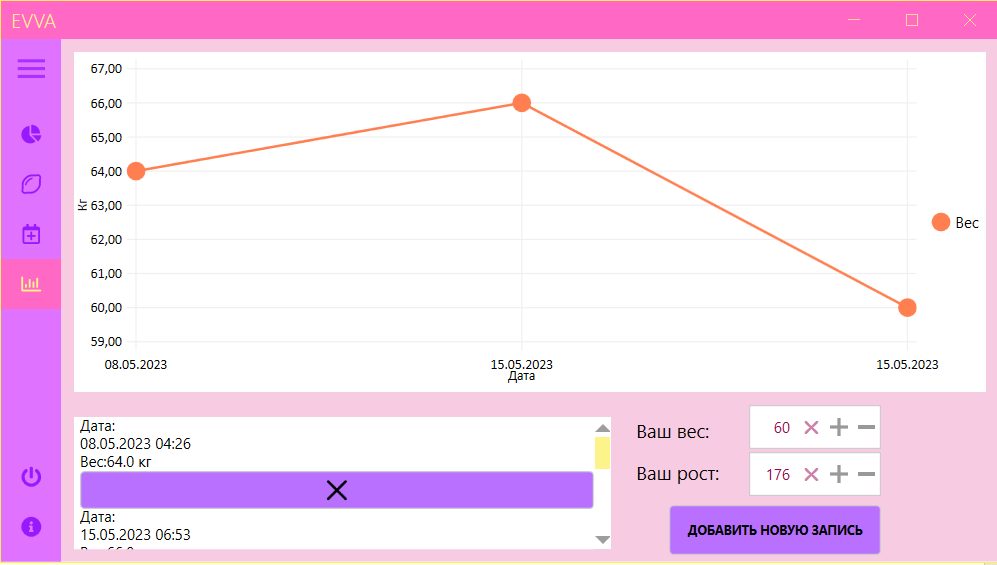


Рисунок 6.7 – Страница статистики

Страница «О приложении» является стартовой и содержит общую информацию о здоровом образе жизни. Данная страница представлена на рисунке 6.8.

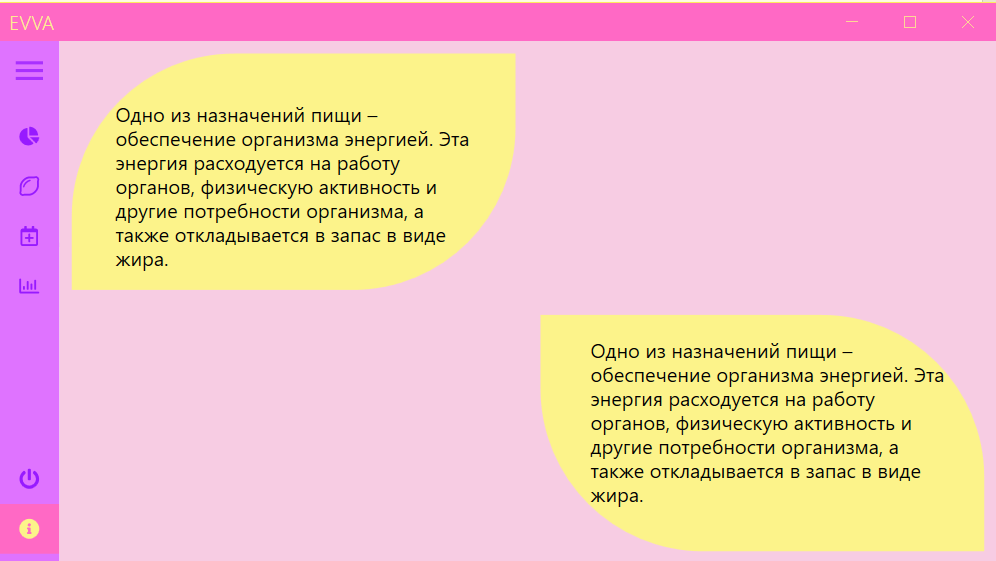


Рисунок 6.8 – Страница с информацией о приложении

Страница настроек предоставляет пользователю интерфейс для редактирования значений параметров работы приложения. Также на данной странице присутствует кнопка выхода из аккаунта и окно изменения пароля пользовательского аккаунта. Данная страница представлена на рисунке 6.9.

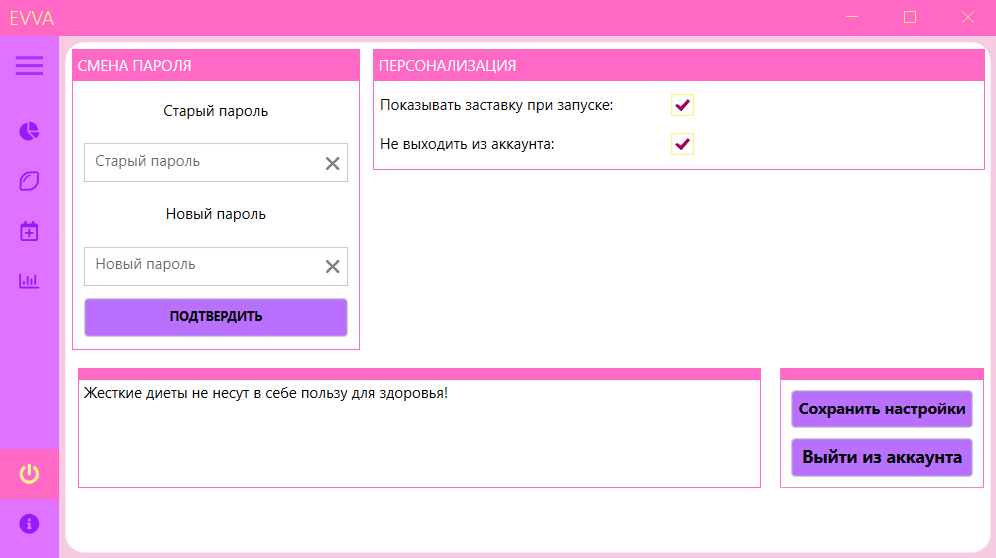


Рисунок 6.9 – Страница настроек

При входе в приложение под аккаунтом администратора присутствует возможность редактирования всех таблиц базы данных приложения, в чём и заключается особенность работы с приложением в роли администратора. На странице расположен элемент управления, позволяющий перемещаться между всеми таблицами базы данных, а также текстовые поля, используемые для изменения, удаления или добавления записей. Страница администрирования представлена на рисунке 6.10.

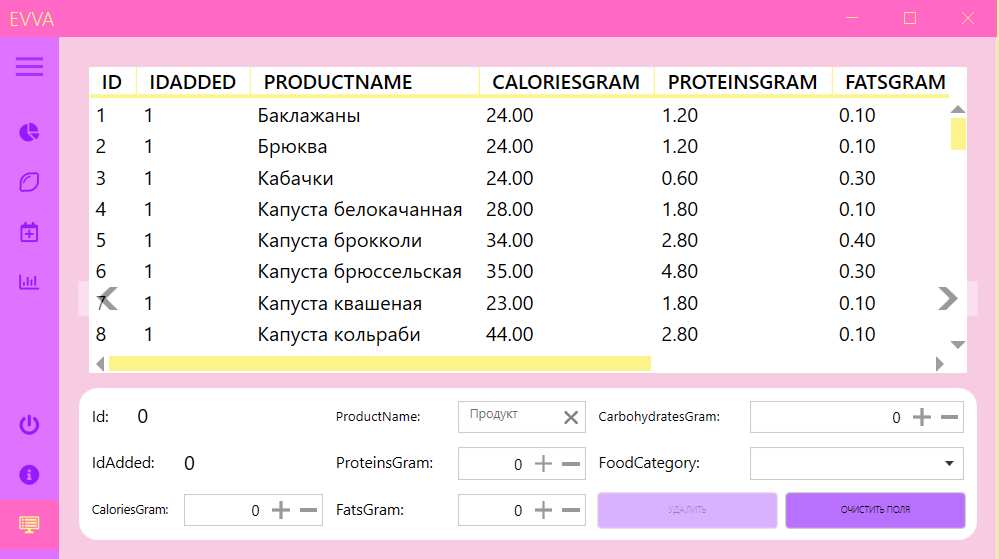
****

Рисунок 6.10 – Страница администрирования приложения

Стоит отметить, что при работе с данной страницей имеет место каскадное удаление записей, что позволяет не нарушать ограничений целостности базы данных.

ЗАКЛЮЧЕНИЕ

Основной задачей данного проекта было усвоение основных технологий разработки десктопных приложений. Разработанное программное средство послужило практической реализацией знаний, полученных в результате изучения теоретических знаний работы с базами данных, технологиями WPF и Entity Framework Core.

В процессе решения поставленной задачи была достигнута поставленная цель по созданию приложения «Evva». Данное приложение может служить не только примером практического применения полученных знаний, но также представляет из себя готовый продукт, который может иметь практическую пользу при использовании его по назначению.

В программном средстве были реализованы следующие функции:

1. авторизация, регистрация и смена пароля;
2. вывод подробной статистике об употреблённых пользователем питательных веществ;
3. возможность добавления, редактирования и удаления продуктов, употреблённых пользователем в определённый период времени;
4. трекинг и редактирование информации об изменении физических характеристик пользователя;
5. добавление, изменение и удаление продуктов в коллекцию продуктов для употребления;
6. удобный процесс поиска необходимых для добавления в рацион продуктов;
7. изменение внешнего вида приложения и настройка приложения под предпочтения пользователя;
8. администрирование приложения.

Разработанная программа имеет завершённый вид, работает верно и готова к использованию.

СПИСОК ЛИТЕРАТУРЫ

1. Пацей, Н.В. Курс лекций по языку программирования С# / Н.В. Пацей. – Минск: БГТУ, 2016. – 175 с.

2. MSDN сеть разработчиков в Microsoft [Электронный ресурс] / Режим доступа: <http://msdn.microsoft.com/library/rus/> . Дата доступа: 20.04.2023

3. METANIT.COM Сайт о программировании [Электронный ресурс] / Режим доступа: <https://metanit.com> . Дата доступа: 20.04.2023

4. ProfessorWeb .NET & Web Programming [Электронный ресурс] / Режим доступа: <https://professorweb.ru> Дата доступа: 13.04.2023

5 Microsoft Docs Archived Content [Электронный ресурс]/ Режим доступа: <https://docs.microsoft.com/en-us/archive/> Дата доступа: 26.04.2023

6 Форум для программистов или разработчиков [Электронный ресурс] – <https://stackoverflow.com/> – Дата доступа: 5.05.2023

**Приложение А**

**Диаграмма классов**







**Приложение Б**

**Листинг для создания таблиц базы данных приложения**

use master;

CREATE DATABASE DietManager;

use DietManager;

CREATE TABLE Users

(

Id int constraint PK\_USERS primary key (Id) identity(1,1),

IsAdmin bit NOT NULL default '0',

UserLogin nvarchar(25) NOT NULL UNIQUE,

UserPassword varbinary(20) NOT NULL UNIQUE,

Salt nvarchar(50) NOT NULL

)

CREATE TABLE UsersParams

(

Id int constraint PK\_USERS\_PARAMS primary key(Id) identity(1,1),

IdParams int NOT NULL constraint FK\_USERS\_PARAMS\_USERS foreign key (IdParams) references Users(Id),

ParamsDate datetime NOT NULL,

UserWeight decimal(4,1) default '0' NOT NULL,

UserHeight int default '0' NOT NULL

)

CREATE TABLE UsersData

(Id int constraint PK\_USERS\_DATA primary key(Id) identity(1,1),

IdData int NOT NULL constraint FK\_USERS\_DATA\_USERS foreign key (IdData) references Users(Id),

FullName nvarchar(300) NOT NULL,

Birthday date NOT NULL)

CREATE TABLE FoodCategories

(

Id int constraint PK\_FCATEGORIES primary key (Id) identity(1,1),

CategoryName nvarchar(50) NOT NULL UNIQUE

)

CREATE TABLE Products

(

Id int constraint PK\_PRODUCTS primary key(Id) identity(1,1),

IdAdded int NOT NULL constraint FK\_PRODUCTS\_USERS foreign key(IdAdded) references Users(Id),

ProductName varchar(200) NOT NULL UNIQUE,

CaloriesGram decimal(7,2) NOT NULL default '0',

ProteinsGram decimal(7,2) NOT NULL default '0',

FatsGram decimal(7,2) NOT NULL default '0',

CarbohydratesGram decimal(7,2) NOT NULL default '0',

FoodCategory nvarchar(50) constraint FK\_PRODUCTS\_FCATEGORY foreign key (FoodCategory) references FoodCategories(CategoryName)

)

CREATE TABLE Reports

(

Id int constraint PK\_REPORTS primary key(Id) identity(1,1),

IdReport int NOT NULL constraint FK\_REPORTS\_USERS foreign key (IdReport) references Users(Id) ,

ProductName varchar(200) NOT NULL constraint FK\_REPORTS\_PRODUCTS foreign key (ProductName) references Products(ProductName),

ReportDate datetime NOT NULL,

EatPeriod varchar(8) NOT NULL,

DayGram decimal(7,2) NOT NULL default '0',

DayCalories decimal(7,2) NOT NULL default '0',

DayProteins decimal(7,2) NOT NULL default '0',

DayFats decimal(7,2) NOT NULL default '0',

DayCarbohydrates decimal(7,2) NOT NULL default '0',

MostCategory nvarchar(50) constraint FK\_REPORTS\_FCATEGORY foreign key (MostCategory) references FoodCategories(CategoryName))

**Приложение В**

**Класс UnitOfWork**

public class UnitOfWork : IDisposable

{ private DietManagerContext db = new DietManagerContext();

private DietManagerDBRepository<User> userRepository;

private DietManagerDBRepository<Report> reportRepository;

private DietManagerDBRepository<Product> productRepository;

private DietManagerDBRepository<FoodCategory> foodCategoryRepository;

private DietManagerDBRepository<UsersDatum> userDatumRepository;

private DietManagerDBRepository<UsersParam> userParamRepository;

public DietManagerDBRepository<UsersParam> UserParamRepository

{ get

{ if (userParamRepository == null)

userParamRepository = new DietManagerDBRepository<UsersParam>(db);

return userParamRepository;

}

}

public DietManagerDBRepository<UsersDatum> UserDatumRepository

{ get { if (userDatumRepository == null)

userDatumRepository = new DietManagerDBRepository<UsersDatum>(db);

return userDatumRepository;

}

}

public DietManagerDBRepository<FoodCategory> FoodCategoryRepository

{ get { if (foodCategoryRepository == null)

foodCategoryRepository = new DietManagerDBRepository<FoodCategory>(db);

return foodCategoryRepository;

}

}

public DietManagerDBRepository<Product> ProductRepository

{ get { if (productRepository == null)

productRepository = new DietManagerDBRepository<Product>(db);

return productRepository;

}

}

public DietManagerDBRepository<Report> ReportRepository

{ get

{ if (reportRepository == null)

reportRepository = new DietManagerDBRepository<Report>(db);

return reportRepository;

}

}

public DietManagerDBRepository<User> UserRepository

{ get

{ if (userRepository == null)

userRepository = new DietManagerDBRepository<User>(db);

return userRepository;

}

}

public void Save()

{ db.SaveChanges(); }

private bool disposed = false;

public virtual void Dispose(bool disposing)

{ if (!this.disposed)

{ if (disposing)

{

db.Dispose();

}

this.disposed = true;

}

}

public void Dispose()

{ Dispose(true);

GC.SuppressFinalize(this);} }

**Класс PasswordHash**

public static class PasswordHash

{

public static int GenerateSaltForPassword()

{

RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider();

byte[] saltBytes = new byte[4];

rng.GetNonZeroBytes(saltBytes);

return (((int)saltBytes[0]) << 24) + (((int)saltBytes[1]) << 16) + (((int)saltBytes[2]) << 8) + ((int)saltBytes[3]);

}

public static byte[] ComputePasswordHash(string password, int salt)

{

byte[] saltBytes = new byte[4];

saltBytes[0] = (byte)(salt >> 24);

saltBytes[1] = (byte)(salt >> 16);

saltBytes[2] = (byte)(salt >> 8);

saltBytes[3] = (byte)(salt);

byte[] passwordBytes = UTF8Encoding.UTF8.GetBytes(password);

byte[] preHashed = new byte[saltBytes.Length + passwordBytes.Length];

System.Buffer.BlockCopy(passwordBytes, 0, preHashed, 0, passwordBytes.Length);

System.Buffer.BlockCopy(saltBytes, 0, preHashed, passwordBytes.Length, saltBytes.Length);

SHA1 sha1 = SHA1.Create();

return sha1.ComputeHash(preHashed);

}

public static bool IsPasswordValid(string passwordToValidate, int salt, byte[] correctPasswordHash)

{

byte[] hashedPassword = ComputePasswordHash(passwordToValidate, salt);

return hashedPassword.SequenceEqual(correctPasswordHash);

}

}

**Класс DietManagerDBRepository<TEntity>**

public class DietManagerDBRepository<TEntity> : IRepository<TEntity> where TEntity : class

{

DbContext \_context;

DbSet<TEntity> \_dbSet;

public DietManagerDBRepository(DbContext context)

{

\_context = context;

\_dbSet = context.Set<TEntity>();

}

public IEnumerable<TEntity> Get()

{

return \_dbSet.AsNoTracking().ToList();

}

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

{

return \_dbSet.AsNoTracking().Where(predicate).ToList();

}

public TEntity FindById(int id)

{

return \_dbSet.Find(id);

}

public void Create(TEntity item)

{

\_dbSet.Add(item);

}

public void Update(TEntity item)

{

\_context.Entry(item).State = EntityState.Modified;

}

public void Remove(TEntity item)

{

\_dbSet.Remove(item);

}

public IEnumerable<TEntity> GetWithInclude(params Expression<Func<TEntity, object>>[] includeProperties)

{

return Include(includeProperties).ToList();

}

public IEnumerable<TEntity> GetWithInclude(Func<TEntity, bool> predicate,

params Expression<Func<TEntity, object>>[] includeProperties)

{

var query = Include(includeProperties);

return query.Where(predicate).ToList();

}

private IQueryable<TEntity> Include(params Expression<Func<TEntity, object>>[] includeProperties)

{

IQueryable<TEntity> query = \_dbSet.AsNoTracking();

return includeProperties

.Aggregate(query, (current, includeProperty) => current.Include(includeProperty));

}

public void RemoveRange(IEnumerable<TEntity> entities)

{

\_dbSet.RemoveRange(entities);

}

}