МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Учреждения 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 10 Программное обеспечение информационных технологий (программирование интернет приложений)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
КУРСОВОГО ПРОЕКТ:

по дисциплине «Объектно-ориентированные технологии программирования и стандарты проектирования»
Тема Программное средство «Coffee Shop»

Исполнитель
студент (ка) 2 курса группы 6 Пузиков Алексей Алексеевич
(Ф.И.О.)

Руководитель работы преподаватель-стажер Север А.С.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой
Председатель Пацей Н.В.
(подпись)

Минск 2023

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПЕУБЛИКИ БЕЛАРУСЬ
Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»
Факультет информационных технологий
Кафедра программной инженерии

Утверждаю
Заведующий кафедрой ПИ
_____ __Н.В. Пацей
подпись инициалы и фамилия
“___”_____2023г

ЗАДАНИЕ
к курсовому проектированию
по дисциплине "Объектно-ориентированные технологии программирования и стандарты проектирования"

Специальность: 1-40 01 01 Программное обеспечение информационных технологий Группа: __6__
Студент: Пузиков Алексей Алексеевич
Тема: Программное средство «Кофейня»

1. Срок сдачи студентом законченной работы: "20 мая 2023 г."

2. Исходные данные к проекту:

2.1. Функционально ПС поддерживает:

- Функции администратора кофейни:
  - Добавлять товары в каталог;
  - Удалять товары из каталога;
  - Доступ ко всем пользователям.
- Функции клиента:
  - Выполнять регистрацию и авторизацию;
  - Просматривать каталог;
  - Сортировать товары по заданным категориям;
  - Выполнять заказ на определенные товары;
  - Редактирование профиля;
  - Поиск товаров.

2.2. При выполнении курсового проекта необходимо использовать принципы проектирования ООП. Приложение разрабатывается под ОС Windows и представляет собой настольное приложение (desktop). Отображение, бизнес логика должны быть максимально независимы друг от друга для возможности расширения. Диаграммы вариантов использования, классов реализации задачи, взаимодействия разработать на основе UML. Язык разработки проекта – C#. Управление программой должно быть интуитивно понятным и удобным. При разработке использовать несколько наиболее подходящих шаблонов проектирования ПО.

3. Содержание расчетно-пояснительной записки
(перечень вопросов подлежащих разработке)

- Введение

- Постановка задачи и обзор литературы (алгоритмы решения, обзор прототипов, актуальность задачи)
- Проектирование архитектуры проекта (структура модулей, классов).
- Разработка функциональной модели и модели данных ПС (выполняемые функции)
- Тестирование
- Заключение
- Список используемых источников
- Приложения

4. Форма представления выполненной курсовой работы:

- Теоретическая часть курсового проекта должны быть представлены в формате docx. Оформление записки должно быть согласно выданным правилам.
- Листинги программы представляются в приложении.
- Пояснительную записку, листинги, проект (инсталляцию проекта) необходимо загрузить на диск, указанный преподавателем.

Календарный план

№ п/п	Наименование этапов курсового проекта	Срок выполнения этапов проекта	Примечание
1	Введение	19.02.2023	
2	Аналитический обзор литературы по теме проекта. Изучение требований, определение вариантов использования	12.03.2023	
3	Анализ и проектирование архитектуры приложения (построение диаграмм, проектирование бизнес-слоя, представления и данных)	26.03.2023	
4	Проектирование структуры базы данных. Разработка дизайна пользовательского интерфейса	02.04.2023	
5	Кодирование программного средства	23.04.2023	
6	Тестирования и отладка программного средства	30.04.2023	
7	Оформление пояснительной записки	07.05.2023	
8	Защита проекта	20.05.2023	

5. Дата выдачи задания ____12.02.2023____

Руководитель_____ Север А.С.
(подпись)

Задание принял к исполнению _____ Пузиков А.А.
(дата и подпись студента)

Содержание

Введение

В настоящее время существует немало языков программирования и фреймворков, которые дают возможность разработать приложения для различных платформ. Один из таких языков – C#. Он широко используется как для создания серверных приложений, так и для разработки приложений на платформе Windows.

В данном курсовом проекте рассматривается процесс разработки программного средства «Кофейня Coffee Shop», представляющего собой удобный интерфейс для решения всего спектра задач, связанного с оформлением заказов, просмотром и добавлением пунктов меню, новостей кофейни и многим другим. Исходя из этого, повышается производительность труда, скорость оформления заказов и общая прибыль фирмы. Большинство таких сервисов представлено в виде веб-сайтов. Также будут описаны основные этапы создания приложения, начиная с проектирования интерфейса, и заканчивая тестированием и оптимизацией кода.

В качестве интерфейса прикладного программирования был выбран обширный API-интерфейс — Windows Presentation Foundation (WPF), позволяющий создавать красивые и производительные приложения с разнообразными элементами управления и реагировать на различные действия пользователя.

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

В качестве сервиса для хранения данных была выбрана объектно-реляционная система управления базами данных PostgreSQL.

В работе будут рассмотрены основные этапы создания приложения – от проектирования интерфейса до оптимизации кода.

1. Аналитический обзор литературы

1.1 Обзор прототипов

Основной задачей курсового проекта является разработка системы для комплексной автоматизации приема заказов кофейни, а также для решения всего спектра задач, связанного с оформлением заказов, просмотром и добавлением пунктов меню, новостей кофейни и многим другим.

Перед проектированием UI, необходимо выделить ряд критериев, которые позволят спроектировать нужный продукт конечному пользователю. Прежде всего необходимо описать цели и задачи пользователя, которые проект будет решать, после выделить ряд критериев, которые будут рассмотрены у аналогов.

- Форма регистрации и входа.
- Дизайн домашней страницы
- Профиль пользователя
- Панель навигации
- Различные уникальные компоненты

«Presto» - облачная система для автоматизации небольших кафе, столовых, ресторанов и целых сетей. В одном окне: работа зала и кухни, учет продуктов, калькуляция блюд и себестоимость, графики смен и мотивация персонала, система лояльности и служба доставки.

Интерфейс «Presto» представлен на рисунке 1.1.

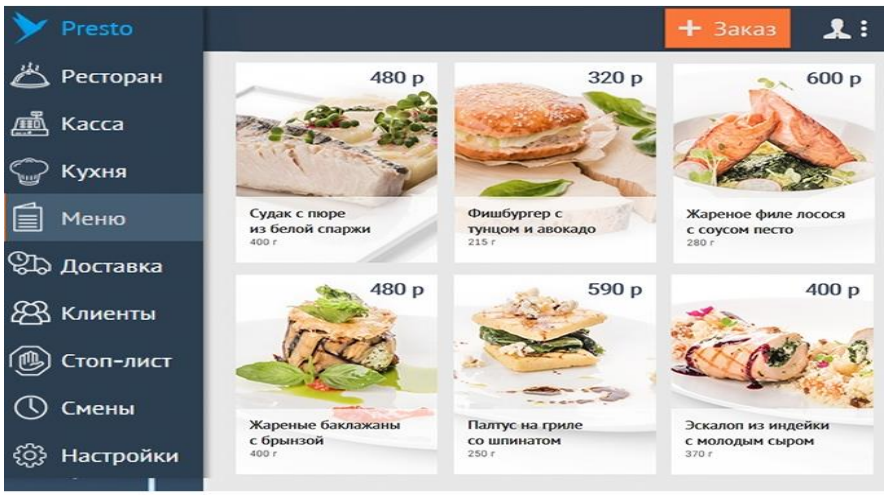


Рисунок 1.1 – Интерфейс «Presto»

К минусам данного программного средства можно отнести: загруженный интерфейс, отсутствие возможности работы со стороны клиента, стоимость. К плюсам: поддержка 1С, неплохой UI, надежность.

Продукт «Restoplace» - сервис бронирования столов для сайтов и социальных сетей ресторанов и кафе. Онлайн приём броней, депозиты,

статистика, база гостей, банкеты. Настраивается за 15 минут, работает в облаке. Стоит от 0 руб. в месяц.

Интерфейс «Restoplace» представлен на рисунке 1.2.

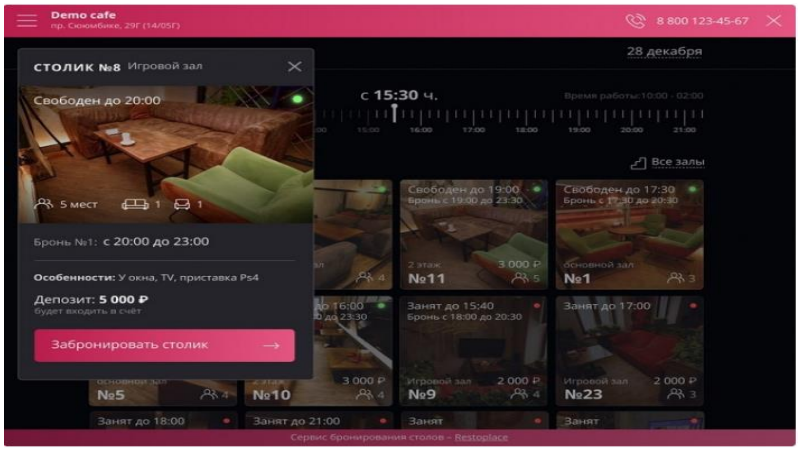


Рисунок 1.2 – Интерфейс «Restoplace»

К минусам данного программного средства можно отнести: примитивный интерфейс и плохие цвета, маленький функционал, отсутствие поиска. К плюсам: наличие мобильного приложения, простота в использовании, фильтрация.

«Fusion POS» - Облачное решение для автоматизации ресторана, кафе, магазина или кальянной. Рабочее место кассира и официанта. Управление меню, складом, лояльностью, отчеты. Интеграция с принтерами чеков и фискальными регистраторами.

Интерфейс «Fusion POS» представлен на рисунке 1.3.

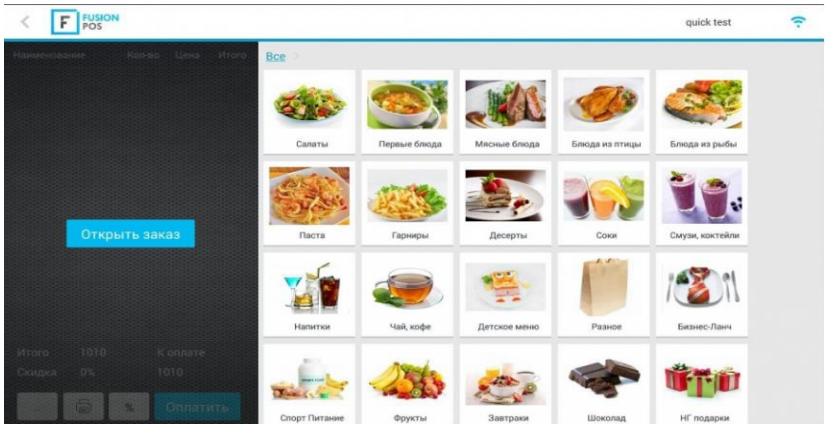


Рисунок 1.3 – Интерфейс «Fusion POS»

К минусам данного программного средства можно отнести: возможность работы только со стороны администратора, излишек категорий. К плюсам: возможность интеграции с кассовой системой, вывод чека на печать.

1.2 Постановка задачи

После анализа прототипов и аналогичных решений, их преимуществ и недостатков, можно выделить ряд основных задач, которые должно решать программное средство.

Функционально ПС должно выполнять следующие задачи:

Функции администратора кофейни:

- Добавлять товары в каталог;
- Редактировать информацию о товарах;
- Удалять товары из каталога;
- Доступ ко всем пользователям.

Функции клиента:

- Выполнять регистрацию и авторизацию;
- Просматривать каталог;
- Сортировать товары по заданным категориям;
- Выполнять заказ на определенные товары;
- Редактирование профиля;
- Поиск товаров.

Описанный функционал позволит повысить производительность труда, скорость оформления заказов и общую прибыль фирмы.

1.3 Средства разработки

При разработке приложения будут использованы

- интегрированная среда разработки Microsoft Visual Studio 2022;
- язык программирования C#;
- расширяемый язык разметки XAML;
- технологии WPF, MailKit;
- База данных PostgreSQL.

Использование данных технологий имеет несколько преимуществ при разработке приложения для управления образовательным процессом.

Во-первых, интегрированная среда разработки Microsoft Visual Studio 2022 обеспечивает высокую производительность и удобство при разработке приложений.

Во-вторых, язык программирования C# — это высокопроизводительный и эффективный язык, который позволяет быстро и легко разрабатывать приложения.

В-третьих, использование расширяемого языка разметки XAML позволяет создавать богатые пользовательские интерфейсы.

База данных PostgreSQL позволяет создавать надежные и быстрые хранилища данных, которые используются для хранения и обработки информации о пользователях, группах, курсах и других сущностях, необходимых для управления образовательным процессом.

1.4 Разработка функциональных требований

Разработка функциональных требований в курсовом проекте предназначен для описания основных функций и возможностей разрабатываемого продукта или системы. В этом разделе необходимо сформулировать требования к функциональности продукта, которые должны быть реализованы в процессе его разработки.

Необходимо начать с описания общих целей и задач, которые решает продукт или система. Затем следует перечислить основные функции, которые должны быть реализованы в продукте, и описать их подробно.

Далее необходимо определить требования к каждой из функций, такие как входные и выходные данные, методы работы и ограничения на использование. Кроме того, следует определить требования к интерфейсу продукта, его удобству использования и интуитивности.

Важным элементом является описание сценариев использования продукта, которые помогут понять, каким образом пользователи будут взаимодействовать с продуктом и какие функции будут использоваться в различных ситуациях, а также должен включать описание тестовых сценариев, которые позволят проверить соответствие продукта или системы заданным требованиям.

В целом, раздел, Разработка функциональных требований является ключевым для успешной разработки продукта или системы, так как он определяет основные функции и возможности продукта и помогает согласовать ожидания разработчиков и пользователей.

Диаграмма вариантов использования представлена на рисунке 1.4.

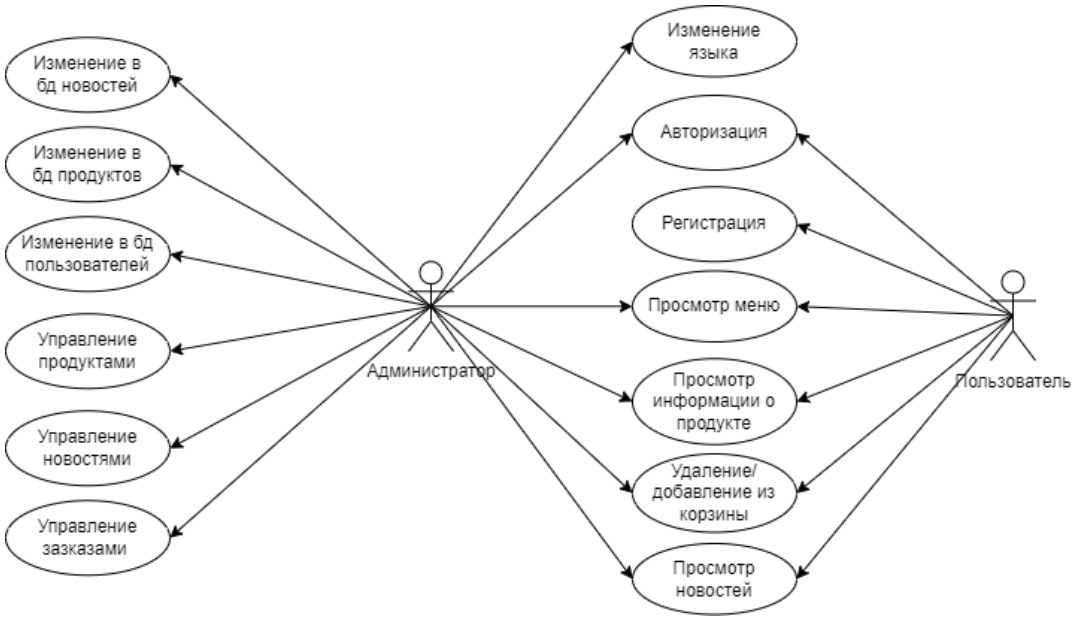


Рисунок 1.4 – Диаграмма вариантов использования

Программное средство должно предоставлять следующие функциональные возможности:

Для пользователя:

- регистрация;
- авторизация;
- просмотр каталога;
- просмотр новостей;
- сортировка и поиск товаров;
- добавление товара в корзину;
- заказ товара;
- получение подтверждающего заказ e-mail письма;
- просмотр информации о данных пользователя;
- просмотр информации о товарах в корзине;
- просмотр информации о товаре;
- возможность поменять язык и тему приложения.

Для администратора:

- авторизация;
- просмотр каталога;
- сортировка и поиск товаров;
- просмотр заказов пользователей;
- принятие заказов;
- отклонение заказов;
- добавление товаров;
- добавление новостей;
- загрузка изображения при добавлении товара;
- удаление товаров;
- возможность поменять язык и тему приложения.

2. Проектирование программного обеспечения

2.1 Обобщённая структура

Архитектура программного обеспечения определяет основные решения, связанные с организацией программной системы. Она включает в себя выбор структурных элементов и интерфейсов, определение их поведения в рамках взаимодействия, объединение элементов в более крупные системы и определение архитектурного стиля, который влияет на всю организацию элементов, их интерфейсы, взаимодействие и объединение.

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 2.1 представлена диаграмма, которая показывает общую структуру приложения в рамках шаблона MVVM.

Рисунок 2.1 – 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. Поэтому для хорошего проектирования View необходимо понять, как пользователь будет взаимодействовать с приложением. Для этого была составлена схема, на которой представлен принцип работы приложения с точки зрения пользователя.

На рисунке 2.2 видно, что при запуске приложения пользователь будет направлен на страницу входа. Для того, чтобы продолжить работу, он должен будет ввести корректные учетные данные или перейти на страницу регистрации и заполнить соответствующие поля.

После входа в приложение пользователю откроется окно приложения с 5 вкладками:

– Главная;

– Меню;
– Профиль;
– О компании;
– Карта;
– Выход.

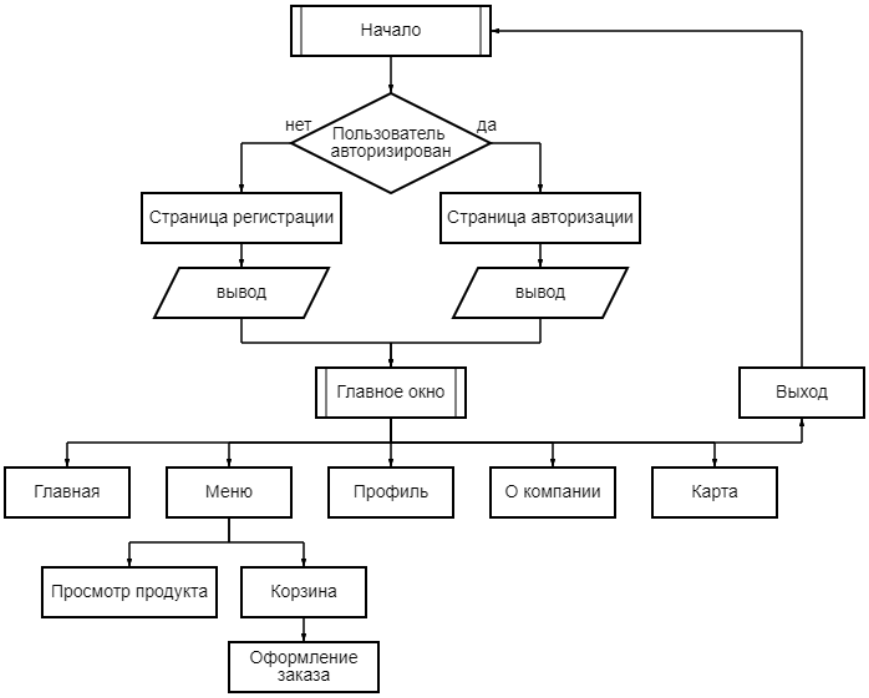


Рисунок 2.2 – Схема навигации по страницам

На странице «Главная» будет отображаться сетка с добавленными администратором новостями. На этой странице пользователь сможет просмотреть последние новости о ресторане.

В разделе «Меню» отображаются карточки с блюдами из меню, на которых указано название, описание и цена. Можно осуществлять поиск по названию, а также фильтрацию по соответствующим критериям.

В разделе «Корзина» пользователь сможет найти блюда, которые он добавил в корзину на странице «Меню» и перейти к оформлению заказа.

При авторизации пользователя с ролью «Администратор» появляется еще один соответствующий пункт меню, при переходе на который появляется доступ к страницам «Добавить новость», «Добавить блюдо» и «Добавить пользователя».

2.2 Модель данных

Для разработки приложения в качестве сервиса для хранения данных была выбрана объектно-реляционная система управления базами данных PostgreSQL.

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 (инфологическое) и логическое (даталогическое) проектирование.

Концептуальное (инфологическое) проектирование — построение семантической модели предметной области, то есть информационной модели наиболее высокого уровня абстракции. В результате этого этапа создаётся ER-модель. Такая модель создаётся без ориентации на какую-либо конкретную СУБД и модель данных.

Основными понятиями ER-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 деталь, которая служит для уточнения, идентификации, классификации, числовой характеристики или выражения состояния сущности.

В рамках этого этапа была создана ER-модель, которая включает 9 сущностей:

– пользователь;

– описание продукта;

– продукт;

– продукт из корзины;

– тип продукта;

– социальные сети;

– банковские карты;

– уведомления;

– новости.

Также в ER-модели были определены необходимые связи. Например, между сущностями тип продукта и продукт была установлена связь один-ко-многим. Для каждой сущности были выделены атрибуты.

Логическое (даталогическое) 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 в приложении Б.

На рисунке 2.3 проиллюстрирована структура таблицы Users, которая содержит информацию о пользователях. В данной таблице поле Id, представленное 16-битным идентификатором, является первичным ключом. Поле UserName хранит логин пользователя, Password – пароль в хешированном виде, Picture – фото профиля, Email – адрес электронной почты, IsAdmin – является ли администратором, BankCardId хранит идентификатор строки в таблице с банковскими картами, SocialNetwork – идентификатор строки в таблице социальных сетей.

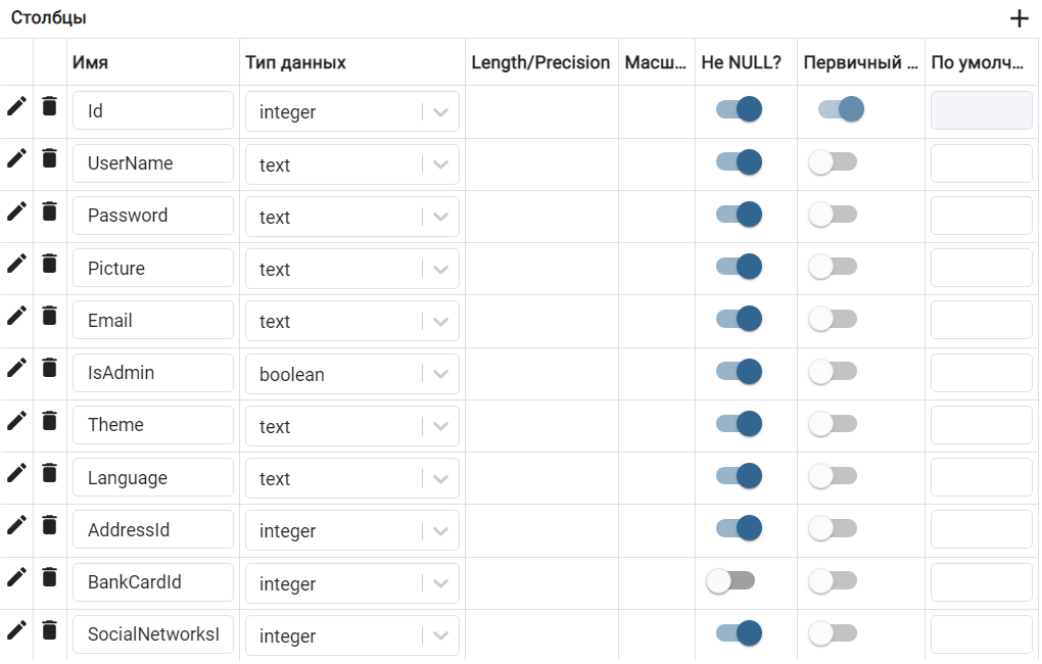


Рисунок 2.3 – Структура таблицы Users

На рисунке 2.4 приведена схема таблицы Products, содержащая информацию о товарах.

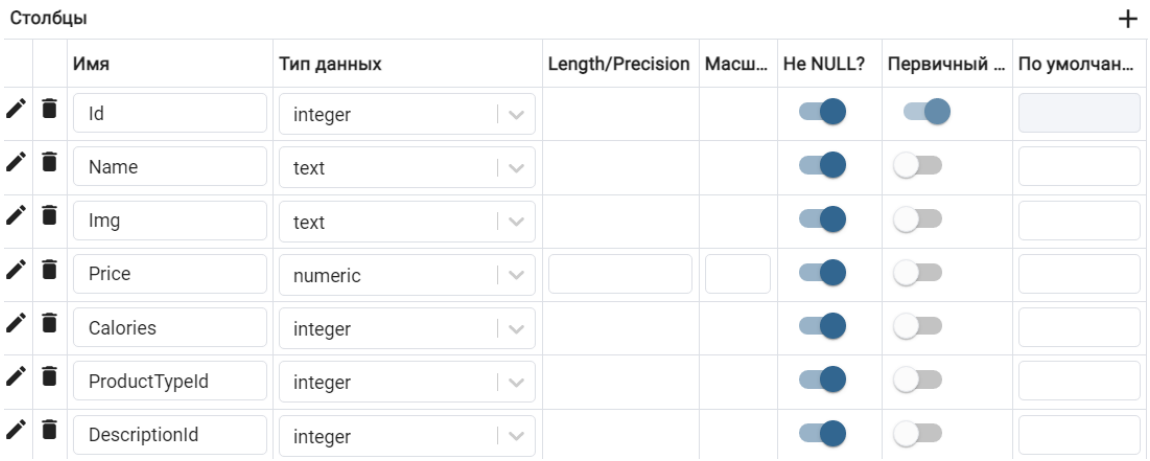


Рисунок 2.4 – Структура таблицы Products

Первичный ключ Id хранит уникальный идентификатор товара, Name – название товара, Image – путь к изображению товара, Price – цена за единицу

товара, Calories – каличество калорий на еденицу товара, ProductTypeId – хранит идентификатор строки в таблице типов продуктов, DescriptionId – хранит идентификатор строки в таблице с описанием товаров.

На рисунке 2.5 изображена структура таблицы ProductTypes, которая хранит информацию о существующих типах продуктов.

Рисунок 2.5 – Структура таблицы ProductTypes

Первичный ключ Id хранит уникальный идентификатор типа продукта, Name – название типа продутка.

3. Реализация программного обеспечения

3.1 Реализация структуры проекта

Для реализации паттерна MVVM файлы программы были распределены по соответствующим директориям и реализовали соответствующие функции. Разделение проекта на логические модули представлено на рисунке 3.1.

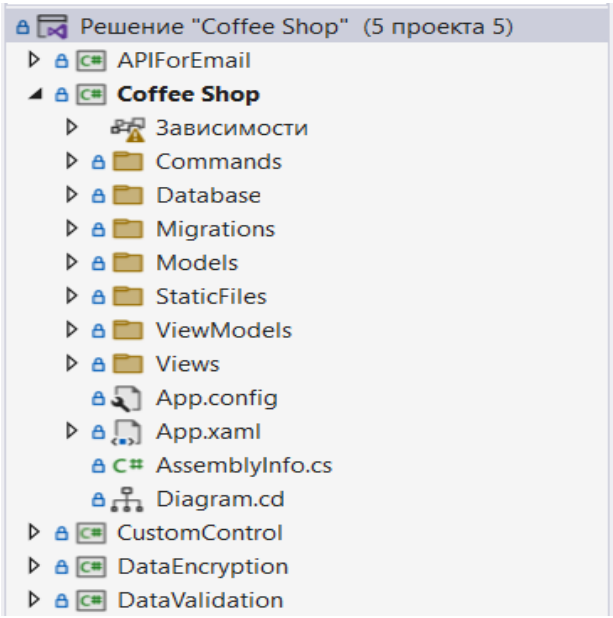


Рисунок 3.1 – Структура проекта

Описание структуры проекта приведено в таблице 3.1.

Таблица 3.1 – Структура проекта

Имя пакета	Описание
Commands	Содержит классы для реализации команд
Database	Содержит класс контекста данных, а также классы-репозитории, которые являются промежуточным звеном между классами-моделями, непосредственно взаимодействующими с данными, и остальной программой. Также содержит класс, реализующий паттерн UnitOfWork.
Migrations	Содержит сгенерированные Entity Framework'ком классы, для взаимодействия с базой данных
Models	Классы сущностей
StaticFiles	Содержит ресурсы, используемые в приложении, такие как: картинки, языковые ресурсы, стили
ViewModels	Содержит логику, которая позволяет получить данные при помощи View, обработать их, используя при этом Model, после чего передать на сервер.
Views	Содержит представления View

Таким образом, сформированная таблица помогает понять общую структуру проектируемого программного средства.

3.2 Реализация шаблона MVVM и другие паттерны

При создании приложения использовался паттерн проектирования MVVM. Он заключается в разделении представления от бизнес-логики. Это достигается за счёт ввода новой логической конструкции ViewModel. Она связывает представление и бизнес-логику приложения.

Все бизнес-классы в данном курсовом проекте наследовались от класса ViewModelBase, который в свою очередь является реализацией интерфейса INotifyPropertyChanged. Данный класс предоставляет метод OnPropertyChanged, которые могут непосредственно вызываться в методе set свойств, принадлежащих ViewModel. Он предоставляют модель оповещения об изменении свойства, а также позволяют ViewModel обойтись без огромного количества приватных полей.

Реализация класса ViewModelBase представлена в листиге 3.1.

```
internal class ViewModelBase : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler? PropertyChanged;

    protected void OnPropertyChanged(string propertyName)
    {
        PropertyChangedEventHandler? handler = PropertyChanged;

        if (handler != null)
        {
            handler(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

Листинг 3.1 – Реализация класса ViewModelBase

В приложении также 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 класса DelegateCommand код которого представлен на листинге 3.2 в приложении В. Класс реализует два метода:

– CanExecute: определяет, может ли команда выполняться;

– Execute: собственно, выполняет логику команды.

В данном программном средстве также реализованы шаблоны Repository и Unit Of Work.

Для реализации паттерна Repository был написан интерфейс IRepository, структура которого представлена на рисунке 4.3. В этом интерфейсе описаны методы для взаимодействия с контекстом базы данных, структура которого представлена в листинге 3.2. В этом интерфейсе описаны методы для взаимодействия с контекстом базы данных.

```
    interface IRepository<T> where T : class
    {
```

```
        List<T> GetIEnumerable();
        T Get(int id);
        void Add(T item);
        void Update(T item);
        void Delete(int id);
    }
```

Листинг 3.2 – Структура интерфейса IRepository

На примере с сущностью Users продемонстрирована реализация интерфейса IRepository (листинг 3.3). В каждой из нужных в данном программном средстве репозиториев будет передаваться контекст базы данных.

```
class UserRepositoryPosgreSQL : IRepository<User>
{
    private ApplicationContext db { get; set; } = ApplicationContext.GetContext();
    public IEnumerable<User> GetIEnumerable()
    {
        return db.Users.Include(x => x.BankCard)
            .Include(x => x.ProductsFromBasket)
            .Include(x => x.SocialNetworks)
            .Include(x => x.Notifications);
    }
    public User? Get(int id) => GetIEnumerable().ToList().Find(x => x.Id == id);
    public void Create(User user)
    {
        user.Id = db.Users.Count() + 1;
        db.Users.Add(user);
    }
    public void Update(User user) => db.Entry(user).State = EntityState.Modified;
    public void Delete(int id)
    {
        User? user = db.Users.Find(id);
        if (user != null)
            db.Users.Remove(user);
    }
    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);
    }
}
```

Листинг 3.3 – Имплементация интерфейса IRepository в классе UserRepository

Далее все репозитории объединяются в общий класс, реализующий шаблон Unit Of Work, структура которого представлена в приложении В. Unit Of Work предоставляет доступ к репозиториям через свойства, а также некоторые методы, такие как Save для сохранения изменений в базе данных

3.3 Реализация хеширования паролей

Хранение паролей в базе данных является небезопасным, поскольку злоумышленники могут получить несанкционированный доступ у учетным записям пользователей. Поэтому было принято решение использовать алгоритм RSA для хеширования паролей.

3.4 Взаимоотношения между классами

Для представления внутренней структуры программы в виде классов и связей между ними используется диаграмма классов. Приложение имеет многослойную архитектуру, поэтому классы разделяются по группам (слоям). Диаграмма классов представлена в Приложении А.

3.5 Дополнительная функциональность

В программном средстве было принято решение реализовать несколько библиотек для расширения функциональности.

Библиотека DataValidation – это универсальная библиотека, предназначенная для валидации полей для ввода, использующихся в основном проекте. Она содержит в себе единственный класс Validator. Он содержит все необходимые регулярные выражения и методы для их проверки, используемые при валидации данных. Реализация класса представлена в виде листинга в приложении В.

Библиотека DataEncryption – это библиотека, предназначенная для хеширования передаваемых в неё данных на основе алгоритма RSA. Она содержит в себе класс Crypter и класс CryptographerBuilder, в которых содержаться методы для кодирования и декодирования данных. Реализация класса Crypter приведена в приложении В.

Библиотека CustomControl – это библиотека, содержащая в себе набор пользовательских элементов управления, разработанных для использования в основном проекте WPF. Она может включает в себя такие элементы, как кнопки, текстовые поля, списки, диаграммы, графики и т.д.

Библиотека EmailService – это библиотека, содержащая в себе весь необходимый функционал для отправки сообщений на почту. Она содержит в себе класс MailBuilder. Отправка сообщений на почту осуществляется при помощи сторонней библиотеки MailKit с открытым исходным кодом для IMAP, POP3 и SMTP.

4. Тестирование и анализ полученных результатов

4.1 Тестирование

Подготовка тестовых данных: важно подготовить тестовые данные, которые будут использоваться для проверки всего программного средства со всех сторон. Эти данные должны воспроизводить реальные сценарии использования и содержать данные, которые могут быть неожиданными или неправильными.

Анализ результатов: результаты тестов должны быть анализированы для выявления ошибок, проблем или улучшений, которые могут быть сделаны в проекте. В случае обнаружения проблем необходимо выполнять отладку и тестирование до тех пор, пока все проблемы не будут решены.

Регулярное тестирование: тестирование должно быть выполнено регулярно, чтобы убедиться, что проект по-прежнему работает корректно и эффективно. Необходимо выполнять тестирование при каждом изменении проекта или его окружения.

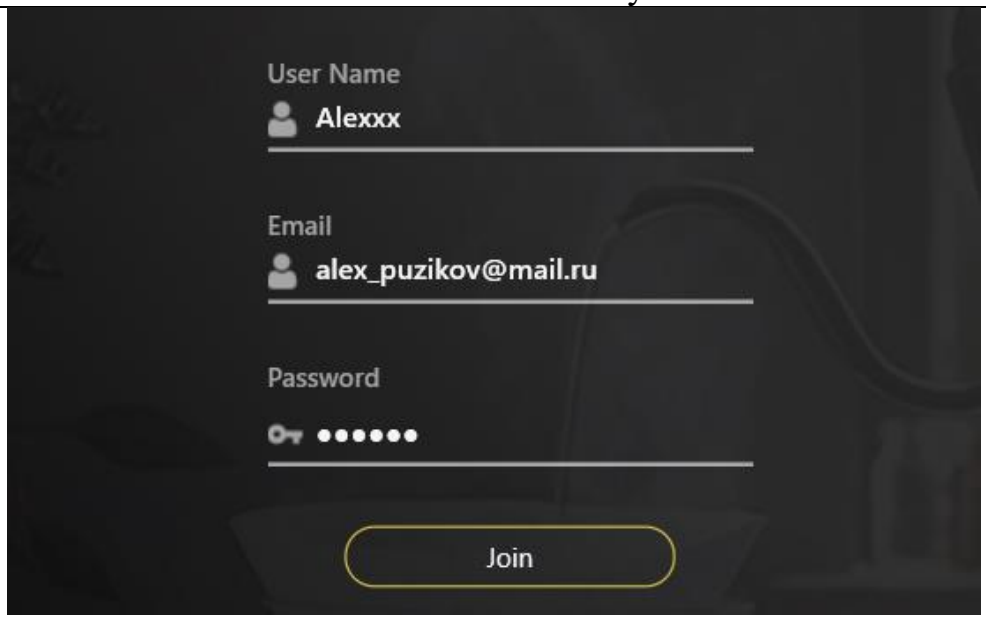
Исходя из данного списка был проведен ряд тестов на реальных данных, в которые входит проверка всех этапов инициализации пользователя, отправка сообщений на почту, оформления заказа и добавление новых позиций меню.

Для тестирования инициализации пользователя можно провести следующие шаги.

Проверка регистрации нового пользователя:

A. Создание нового пользователя в тестируемом приложении
B. Попытка сохранения данных и проверка наличия пользователя в базе данных

Таблица 4.1 – Описание тестов по инициализации пользователя

Описание тестовых случаев

A

Продолжение таблицы 4.1

	Id [PK] integer	UserName text	Password text	Picture text	Email text	IsAdmin boolean	BankCardId integer	SocialNetworksId integer
4	4	Alexxx	lRopBnm8vsl2t...	\StaticFile...	alex_puzikov@mail.ru	false	[null]	4

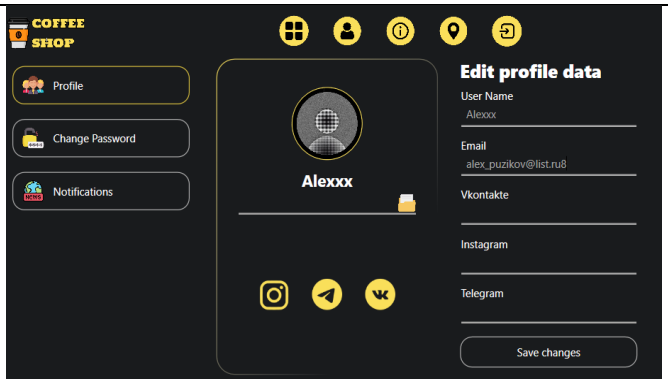
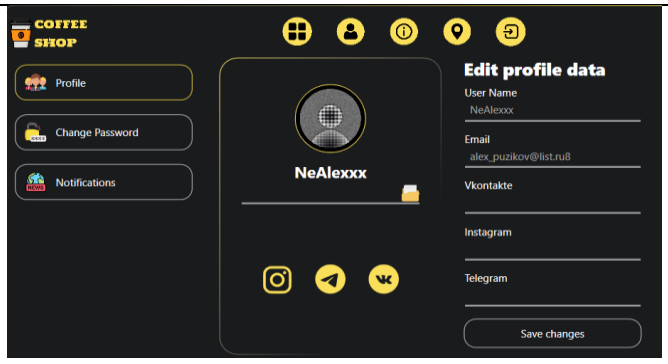
В

Проверка входа существующего пользователя:

A. Попытка входа с использованием существующих учетных данных и проверка успешности авторизации
B. Проверка корректности отображения данных пользователя на главном экране
C. Проверка возможности изменения данных пользователя

Таблица 4.2 – Описание тестов по входу и получению информации

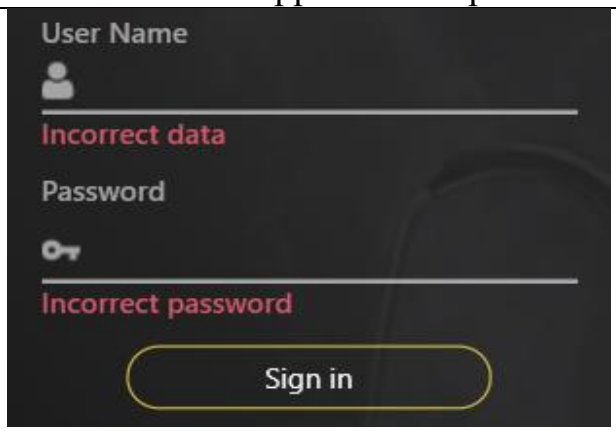
Описание тестовых случаев

A

Б

C

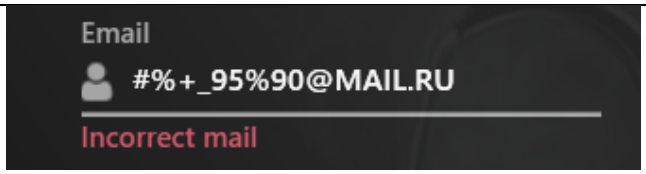
Проверка сценариев обработки ошибок:

A. Попытка входа с пустыми полями и проверка корректности вывода ошибки
B. Попытка регистрации с некорректным email и проверка корректности вывода ошибки
C. Попытка регистрации под именем уже существующего пользователя

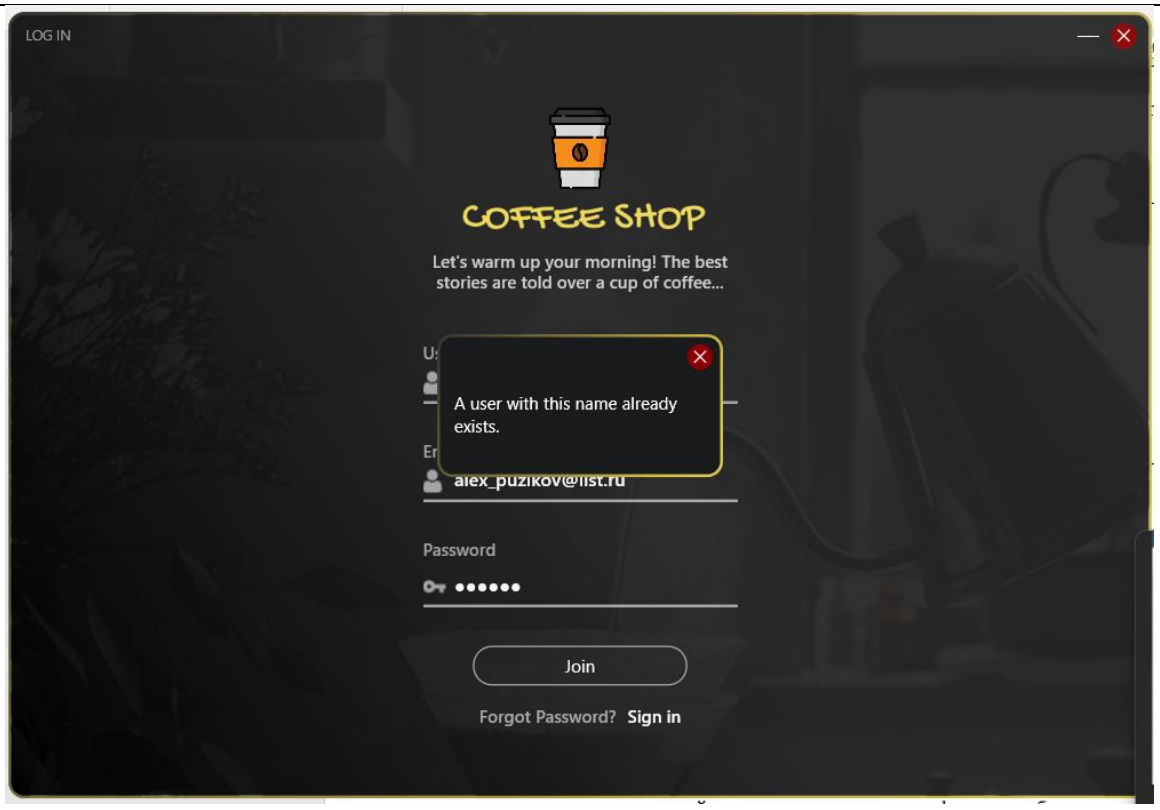
Таблица 4.3 – Описание тестов по корректной обработке ошибок



A



B



C

В процессе тестирования следует удостовериться, что все этапы инициализации пользователя (регистрация, вход в систему, изменение данных и выход из учетной записи) работают корректно и без ошибок. Также важно

проверить корректность обработки ошибок и соответствие отображаемых сообщений действительности.

В панели администратора возможно генерирование многих исключительных ситуаций, таких как: отправка формы добавления или редактирования товара при наличии незаполненных полей, ввод нецелочисленных либо выходящих за допустимые границы значений в поля «Цена», «Вес», «Количество» или «Проба металла», отсутствие прикрепленного изображения товара или прикрепление повторяющегося изображения.

Примеры обработки таких ситуаций представлены на рисунке 4.1.

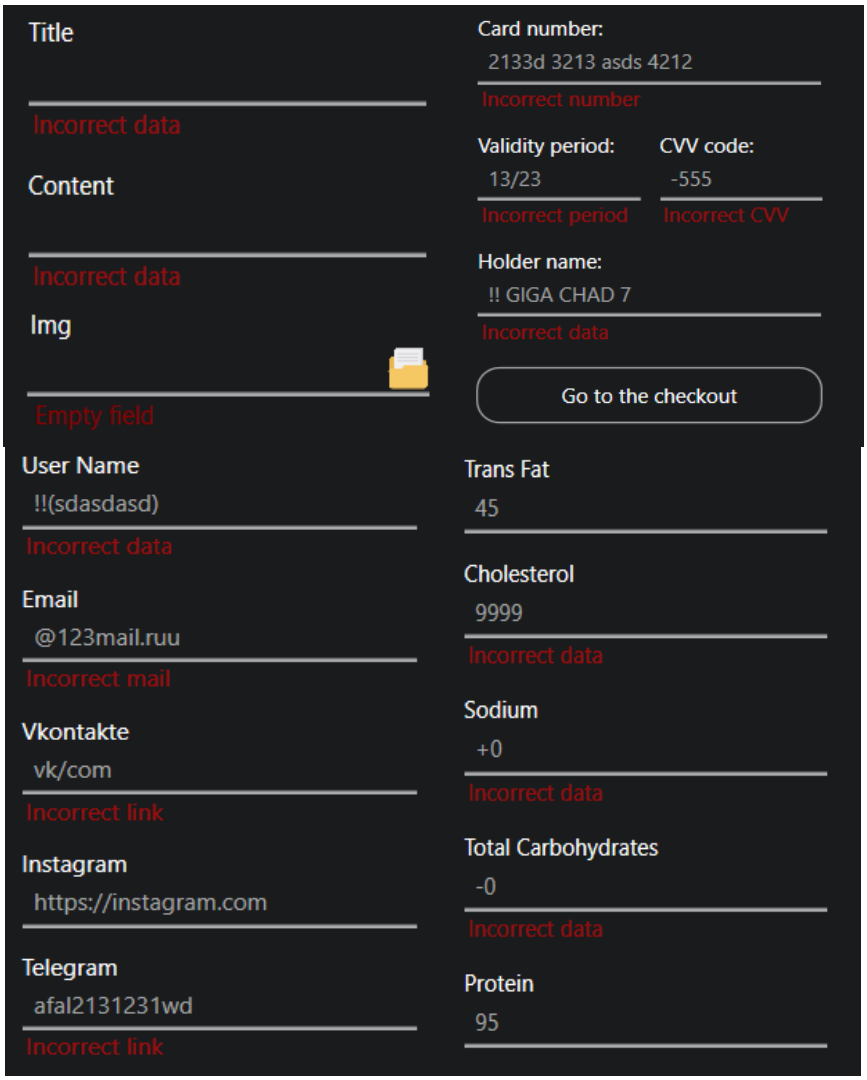


Рисунок 4.1 – Примеры обработки ошибок

Таким образом была реализована валидация во всех полях для ввода, доступных в приложении.

5. Руководство по установке и использованию программного обеспечения

5.1 Использование приложения

При запуске данного программного средства пользователь попадает на страницу авторизации, содержащую формы входа, регистрации и восстановления пароля. Если у пользователя еще нет аккаунта, ему следует нажать на кнопку «Регистрация», которая его перенаправит на форму для регистрации. Представление стартового окна представлено на рисунке 5.1.

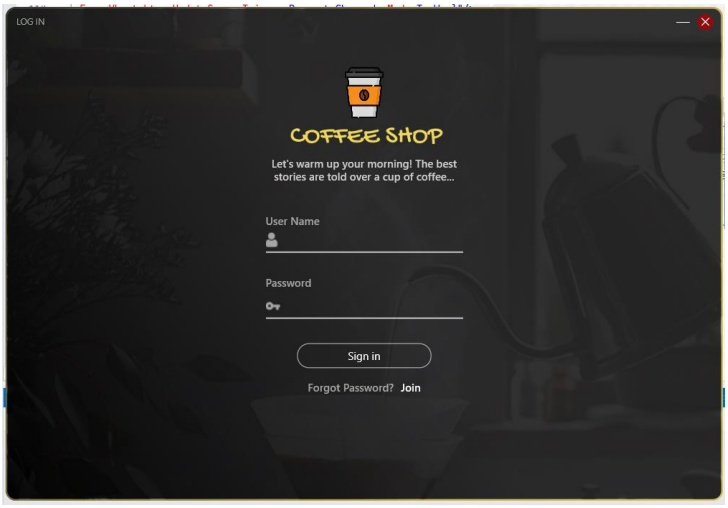


Рисунок 5.1 – Стартовое окно

Если была нажата кнопка «Регистрация», в открывшейся форме следует ввести все данные о регистрируемом пользователе. В случае, если кнопка была нажата ошибочно, форму можно закрыть. Форма регистрации 5.2.

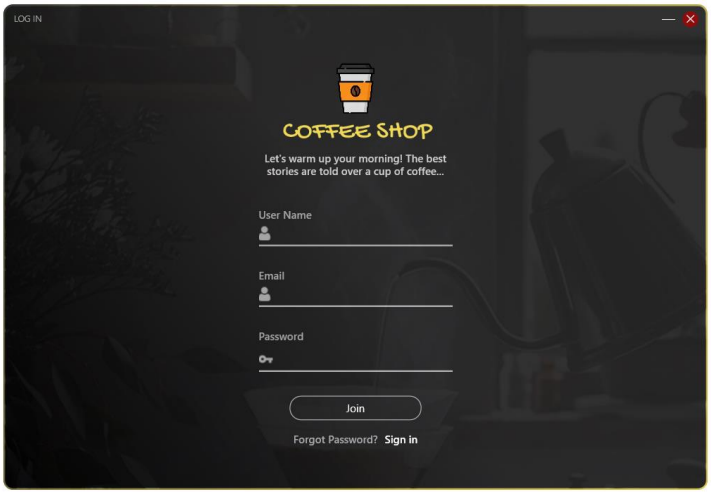


Рисунок 5.2 – Стартовое окно

После успешного входа пользователь попадает на главную страницу. В шапке приложения есть логотип, при нажатии на который происходит переход

на главную страницу, в середине расположено меню приложения, состоящее из 7 кнопок для навигации между страницами. Главная страница представлена на рисунке 5.3.

Рисунок 5.3 – Главная страница

На странице меню слева представлена панель для поиска и фильтрации товаров по названию, категории, количеству калорий и по диапазону цен. По центру представлен список товаров. При нажатии на отдельную единицу товара происходит переход на страницу с его подробным описанием. В панели справа отображается корзина товаров. Страница меню представлена на рисунке 5.4.

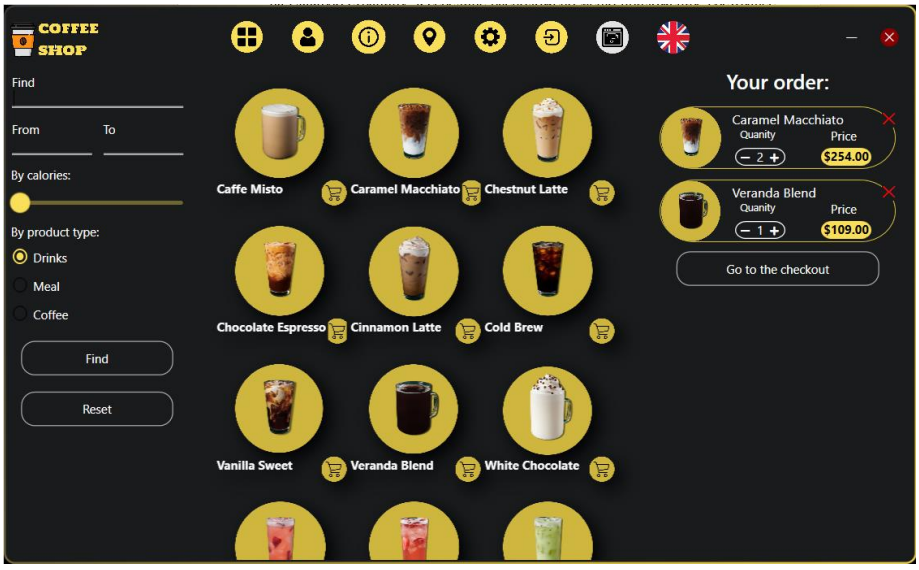


Рисунок 5.4 – Страница меню

При переходе на страницу профиля у пользователя появляется возможность изменить свои личные данные: имя, почта, изображение профиля, а также социальные сети. Страница профиля представлена на

рисунке 5.5. При переходе на вкладу с изменением пароля, расположенной в левой части окна, пользователь имеет возможность сменить пароль. При переходе на вкладку уведомления отображаются все входящие уведомления.

Рисунок 5.5 – Страница профиля

При переходе на страницу «О компании» пользователь имеет возможность ознакомиться с информацией, непосредственно связанной с компанией и ее историей. Пример страницы «О компании» представлена на рисунке 5.6.

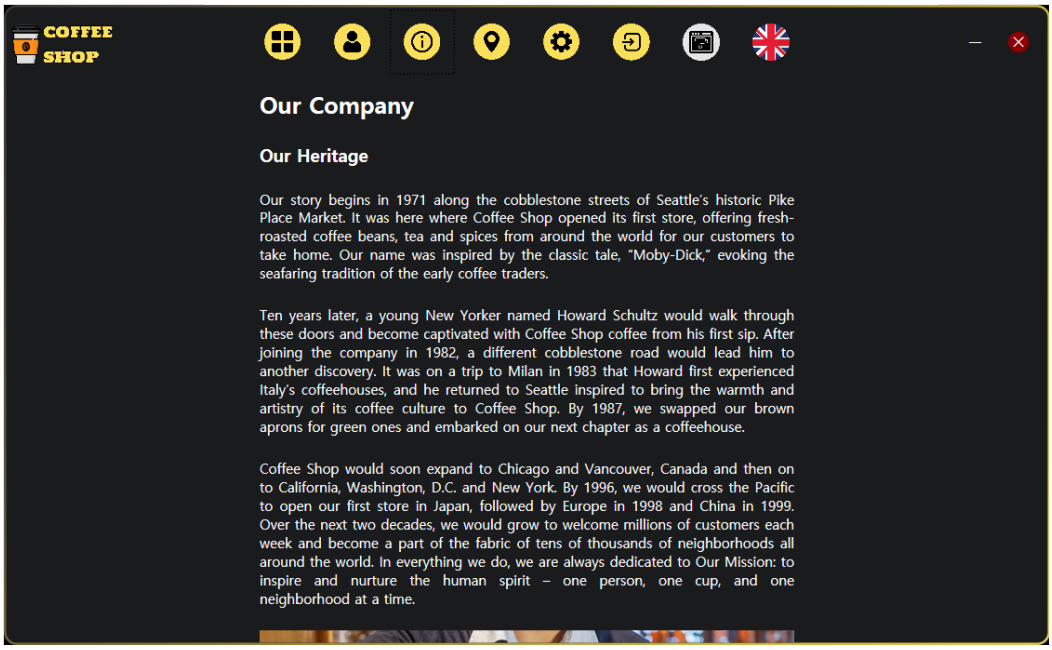


Рисунок 5.6 – Страница «О компании»

При входе под учетной записью администратора появляется дополнительная кнопка, для перехода на страницу администратора. На ней администратор может управлять базой данных пользователей, новостей и

продуктов. Также он может добавлять новости, продукты, пользователей и управлять их заказами. Страница администратора представлена на рисунке 5.7.

Рисунок 5.7 – Страница администратора

При переходе на страницу с картой пользователь имеет возможность ознакомиться с расположением кофеен по всему миру, а также с временем их работы. Пример страницы с картой представлен на рисунке 5.8.

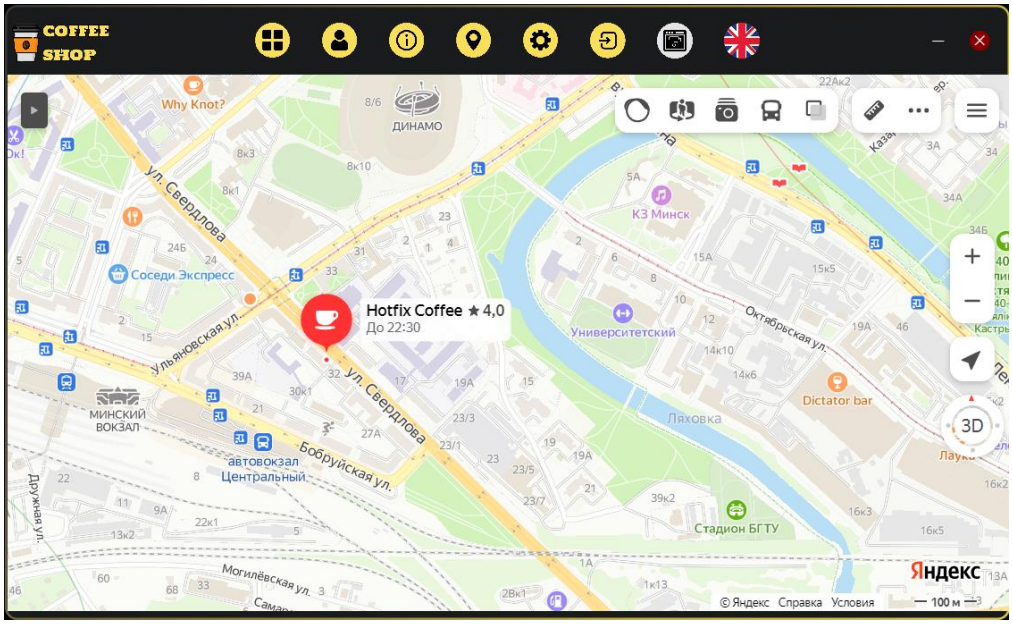


Рисунок 5.7 – Страница с картой

При нажатии на кнопку выхода из аккаунта происходит завершение текущей сессии и открывается страница для входа. При нажатии на кнопку смены темы происходит изменение цветового оформления. При нажатии на кнопку для смены языка происходит смена локализации приложения.

Заключение

В процессе решения поставленной задачи была достигнута поставленная цель по созданию программного средства «Кофейня Coffee Shop». При разработке программного средства были выполнены все пункты из указанного списка предполагаемого основного функционала приложения, а именно:

Функции администратора кофейни:

- Добавлять товары в каталог;
- Редактировать информацию о товарах;
- Удалять товары из каталога;
- Доступ ко всем пользователям.

Функции клиента:

- Выполнять регистрацию и авторизацию;
- Просматривать каталог;
- Сортировать товары по заданным категориям;
- Выполнять заказ на определенные товары;
- Редактирование профиля;
- Поиск товаров.

Также в рамках проекта было разработано Руководство по установке и использованию программного обеспечения. Оно содержит подробные инструкции по установке и настройке приложения, а также описание основных функциональных возможностей и действий пользователя при работе с приложением.

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 источников

1. Пацей, Н.В. Курс лекций по языку программирования C# / 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: 10.03.2023

3. METANIT.COM Сайт о программировании [Электронный ресурс] / Режим доступа: https://metanit.com . Дата доступа: 10.03.2023

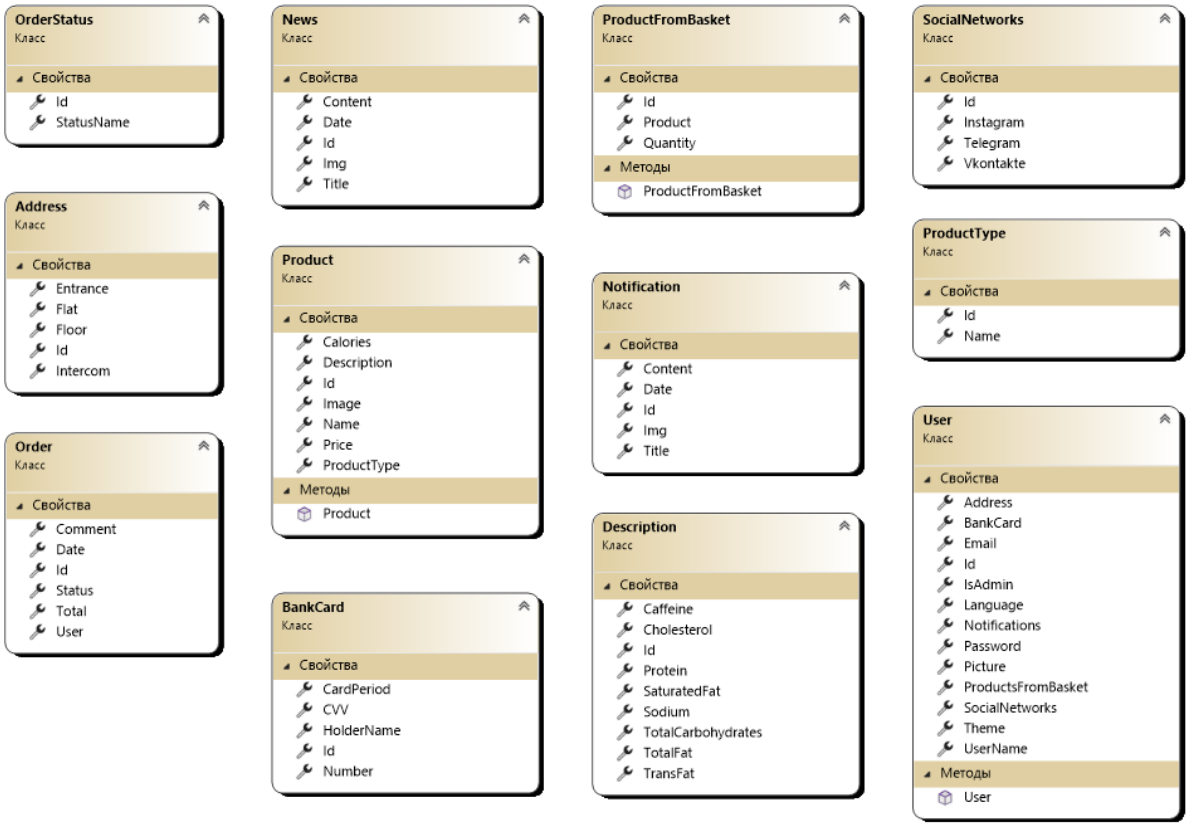
4. ProfessorWeb .NET & Web Programming [Электронный ресурс] / Режим доступа: https://professorweb.ru Дата доступа: 15.03.2023

5 Microsoft Docs Archived Content [Электронный ресурс]/ Режим доступа: https://docs.microsoft.com/en-us/archive/ Дата доступа: 23.03.2023

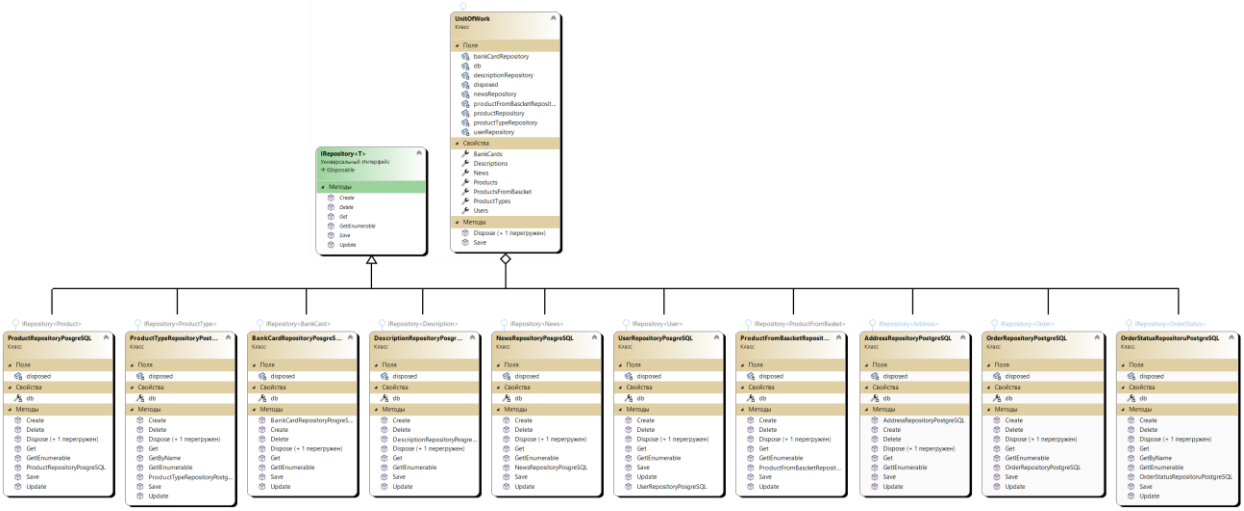
6 Форум для программистов или разработчиков [Электронный ресурс] – https://stackoverflow.com/ – Дата доступа: 27.03.2023

7 Регулярные выражения [Электронный ресурс] – Метод доступа: https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expressions Дата доступа: 25.04.2023.

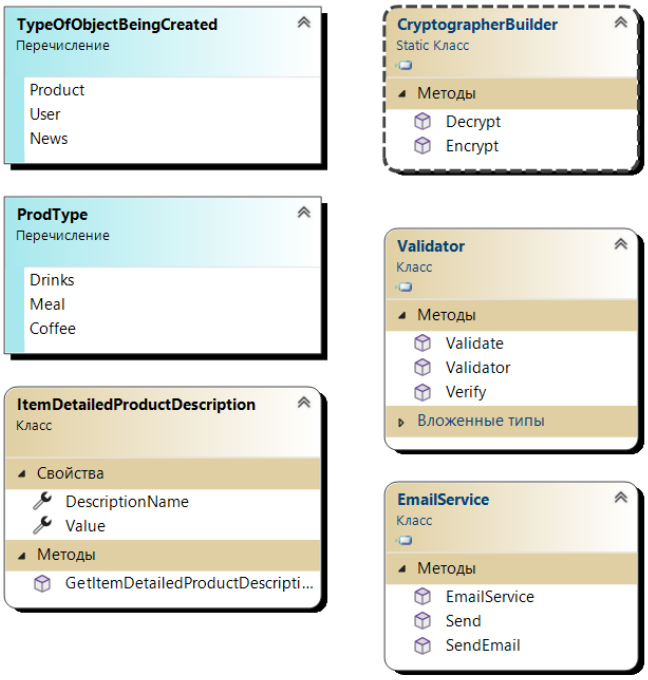
Приложение А



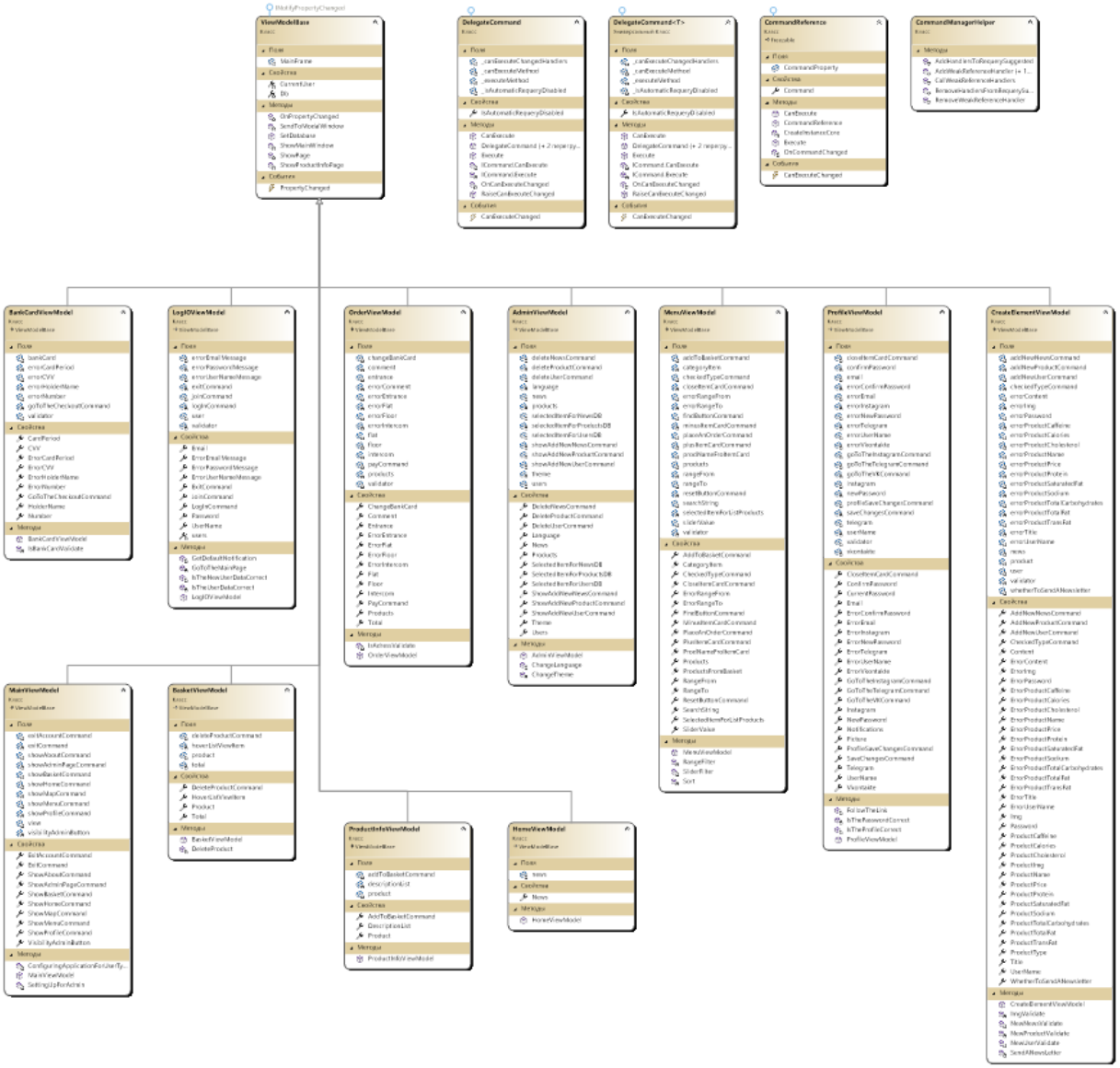
Классы модели



Классы репозитории

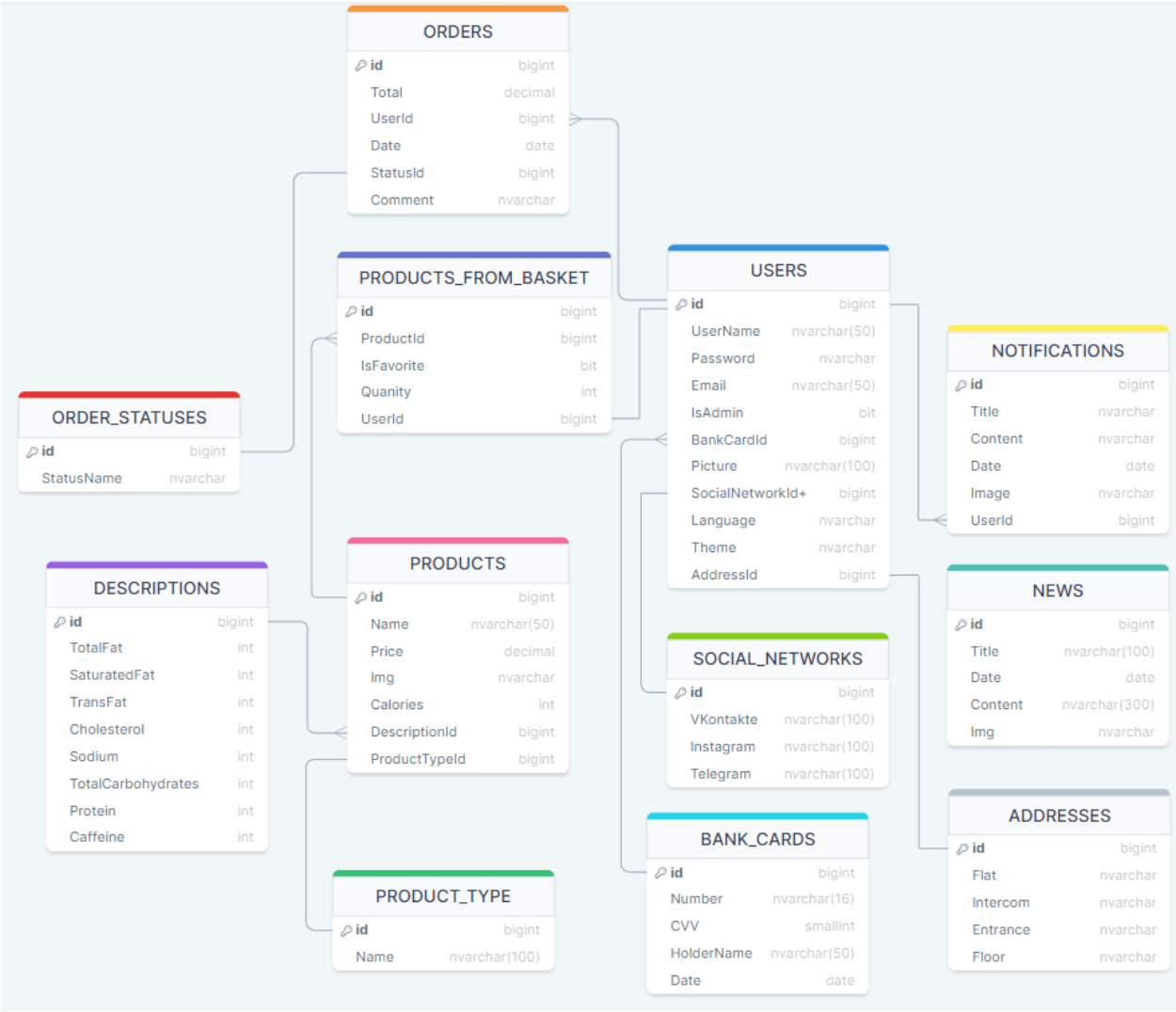


Классы утилиты



Классы ViewModel

Приложение Б



Логическая модель базы данных

Приложение В

Реализация класса UnitOfWork

```
internal class UnitOfWork : IDisposable
{
    private ApplicationContext db = ApplicationContext.GetContext();
    private NewsRepositoryPosgreSQL? newsRepository;
    private UserRepositoryPosgreSQL? userRepository;
    private ProductRepositoryPosgreSQL? productRepository;
    private BankCardRepositoryPosgreSQL? bankCardRepository;
    private DescriptionRepositoryPosgreSQL? descriptionRepository;
    private ProductTypeRepositoryPostgreSQL? productTypeRepository;
    private ProductFromBascketRepositoryPosgreSQL? productFromBascketRepository;

    public NewsRepositoryPosgreSQL News
    {
        get
        {
            if (newsRepository == null)
            {
```

```
                newsRepository = new NewsRepositoryPosgreSQL();
            }
            return newsRepository;
        }
    }
    public UserRepositoryPosgreSQL Users
    {
        get
        {
            if (userRepository == null)
            {
                userRepository = new UserRepositoryPosgreSQL();
            }
            return userRepository;
        }
    }
    public ProductRepositoryPosgreSQL Products
    {
        get
        {
            if (productRepository == null)
            {
                productRepository = new ProductRepositoryPosgreSQL();
            }
            return productRepository;
        }
    }
    public BankCardRepositoryPosgreSQL BankCards
    {
        get
        {
            if (bankCardRepository == null)
            {
                bankCardRepository = new BankCardRepositoryPosgreSQL();
            }
            return bankCardRepository;
        }
    }
    public DescriptionRepositoryPosgreSQL Descriptions
    {
        get
        {
            if (descriptionRepository == null)
            {
                descriptionRepository = new DescriptionRepositoryPosgreSQL();
            }
            return descriptionRepository;
        }
    }
    public ProductTypeRepositoryPostgreSQL ProductTypes
    {
        get
        {
            if (productTypeRepository == null)
            {
                productTypeRepository = new ProductTypeRepositoryPostgreSQL();
            }
            return productTypeRepository;
        }
    }
    public ProductFromBascketRepositoryPosgreSQL ProductsFromBascket
    {
        get
        {
            if (productFromBascketRepository == null)
```

```
            {
                productFromBascketRepository = new
ProductFromBascketRepositoryPosgreSQL();
            }
            return productFromBascketRepository;
        }
    }
    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);
    }
}
```

Реализация класса DelegateCommand

```
public class DelegateCommand : ICommand
    {
        public DelegateCommand(Action executeMethod)
            : this(executeMethod, null, false) { }
        public DelegateCommand(Action executeMethod, Func<bool> canExecuteMethod)
            : this(executeMethod, canExecuteMethod, false) { }

        public DelegateCommand(Action executeMethod, Func<bool> canExecuteMethod,
bool isAutomaticRequeryDisabled)
        {
            if (executeMethod == null)
            {
                throw new ArgumentNullException("executeMethod");
            }

            _executeMethod = executeMethod;
            _canExecuteMethod = canExecuteMethod;
            _isAutomaticRequeryDisabled = isAutomaticRequeryDisabled;
        }
        public bool CanExecute()
        {
            if (_canExecuteMethod != null)
                return _canExecuteMethod();
            return true;
        }

        public void Execute()
        {
            if (_executeMethod != null)
                _executeMethod();
        }
```

```
        public bool IsAutomaticRequeryDisabled
        {
            get => _isAutomaticRequeryDisabled;
            set
            {
                if (_isAutomaticRequeryDisabled != value)
                {
                    if (value)
CommandManagerHelper.RemoveHandlersFromRequerySuggested(_canExecuteChangedHandlers
);
                    else
CommandManagerHelper.AddHandlersToRequerySuggested(_canExecuteChangedHandlers);
                    _isAutomaticRequeryDisabled = value;
                }
            }
        }

        public void RaiseCanExecuteChanged()
        ⇨  OnCanExecuteChanged();

        protected virtual void OnCanExecuteChanged()
        =>
   CommandManagerHelper.CallWeakReferenceHandlers(_canExecuteChangedHandlers);

        public event EventHandler CanExecuteChanged
        {
            add
            {
                if (!_isAutomaticRequeryDisabled)
                    CommandManager.RequerySuggested += value;
                CommandManagerHelper.AddWeakReferenceHandler(ref
_canExecuteChangedHandlers, value, 2);
            }
            remove
            {
                if (!_isAutomaticRequeryDisabled)
                    CommandManager.RequerySuggested -= value;
CommandManagerHelper.RemoveWeakReferenceHandler(_canExecuteChangedHandlers,
value);
            }
        }

        bool ICommand.CanExecute(object parameter) => CanExecute();
        void ICommand.Execute(object parameter) => Execute();

        private readonly Action _executeMethod = null;
        private readonly Func<bool> _canExecuteMethod = null;
        private bool _isAutomaticRequeryDisabled = false;
        private List<WeakReference> _canExecuteChangedHandlers;
    }
```

Реализация класса Validator

```
public class Validator
{
    public Validator(object instance)
    {
        this.Instance = instance;
    }

    private object Instance;

    public enum ValidationBased
```

```
    {
        Email,
        Password,
        TextTo,
        OrdinaryDigits,
        Links,
        BankCVV,
        BankPeriod,
        BankNumber,
        Image,
        Date
    }

    public bool Verify(ValidationBased validationBased, string value, string
property)
    {
        string regex = string.Empty;
        string errorMessage = string.Empty;

        switch (validationBased)
        {
            case ValidationBased.Email:
                regex = "^((\\w+([-+.]\\w+)*@\\w+([-.]\\w+)*\\.\\w+([-
.]\\w+)*)\\s*[;]{0,1}\\s*)+$";
                errorMessage = "Incorrect mail";
                break;
            case ValidationBased.Password:
                regex = @"^.{6,}$";
                errorMessage = "Incorrect password";
                break;
            case ValidationBased.TextTo:
                regex = @"^[a-zA-Z\s]{5,}$";
                errorMessage = "Incorrect data";
                break;
            case ValidationBased.OrdinaryDigits:
                regex = @"^[1-9]\d{0,2}$";
                errorMessage = "Incorrect data";
                break;
            case ValidationBased.Links:
                regex = @"^(http|https|ftp|)\://|[a-zA-Z0-9\-\.]+\.[a-zA-Z](:[a-
zA-Z0-9]*)?/?([a-zA-Z0-9\-\._\?\,\'/\\\+&amp;%\$#\=~])*[^\.\,\)\(\s]$";
                errorMessage = "Incorrect link";
                break;
            case ValidationBased.BankCVV:
                regex = @"^[0-9]{1,3}$";
                errorMessage = "Incorrect CVV";
                break;
            case ValidationBased.BankPeriod:
                regex = @"^(0?[1-9]|1[0-2])\/[0-9]{2}$";
                errorMessage = "Incorrect period";
                break;
            case ValidationBased.BankNumber:
                regex = @"^\d{4}-\d{4}-\d{4}-\d{4}$";
                errorMessage = "Incorrect number";
                break;
            case ValidationBased.Image:
                regex = @"^([a-zA-Z]:)?\\(\w+\\)+\w+(\.\w+)?$";
                errorMessage = "Incorrect path";
                break;
            case ValidationBased.Date:
                regex = @"^(0[1-9]|[12][0-9]|3[01])\.(0[1-
9]|1[012])\.(19|20)\d\d$";
                errorMessage = "Incorrect date";
                break;
        }
```

```
        return Validate(regex, errorMessage, value, property);
    }

    public bool Validate(string regex, string errorMessage, string value, string
property)
    {
        Regex reg = new Regex(regex);
        if (!reg.IsMatch(value))
        {
            SetPropertyValue(Instance, property, errorMessage);
            return false;
        }
        SetPropertyValue(Instance, property, "");
        return true;
    }

    private static void SetPropertyValue(object obj, string propertyName, object
value)
    {
        PropertyInfo property = obj.GetType().GetProperty(propertyName);

        if (property != null)
        {
            property.SetValue(obj, value, null);
        }
        else
        {
            throw new ArgumentException($"Свойство {propertyName} не найдено в
типе {obj.GetType()}");
        }
    }
}
```

Реализация класса Crypter

```
public class Crypter
{
    private string Key { get; } =
"<RSAKeyValue><Modulus>q4Y9Y7WFChl64pI4jJluzF6C/93My2XCUYLO76hwJv9PbGyjMT9dmUw/m5C
289Y0j/XaLI0jgO4I6NX1VOmybKnO+rZ95NppH54VT6VfATjdrym+6CpcoergFOsRb08leqINVCTl+yodf
q6eHr1mQ9y/YdF6t6FOX/q1tihcxSU=</Modulus><Exponent>AQAB</Exponent><P>45YekSe+h43s5
y8QG8JkOjX6sOgZ7IYhqgg5hoXC5riTrIN2IAJdh6fp2WvetIADw/KlvkK4XRSKZg5Blpp7Yw==</P><Q>
wPBS7pTJjUYpXB92gFbuV8w1HYr7H37vAslOsge9IJd3rUdzj6mEKuH29EnUc19HdkiB/hZuV0OP53ZpJr
XX1w==</Q><DP>PQgaBGU7JBD8cfbeBAO6ax3kr6JeqV5DEt0HyDqAzOy8tWu/ts/Lk0CFZsgVviQCXn7o
0cAEvvluL/Yswp2E7w==</DP><DQ>l8Cuyh7XBLpJr77DeyBk6UOiB3GYIWa6YWuq7RZvKGJabD1F5JpFb
WE711r2siQf1iYjsJE+Cn8Ggdy9yge/Ew==</DQ><InverseQ>2o6HxugEikRddLxVV58JZUdDN7ErPZmx
EXBk2g7XqkTQ1yov8dq/KtLXUMfLbl6cWfe4GIdpefvDaVhAkS/Q2A==</InverseQ><D>da2EwrrPys0O
bRHKsFO4G4igMbFXhxiKh+fJ18zlHSw+rnGeSPRjYABbB3zyuDn3F+mhxL0UZalp/WyFg7tOCHS4fkNNyD
tac++OwHCXGSsrHqZhUDfZ9jDwRIFZWTocadvWjTleIgjTAxdB/kW5My40r7C9tbONhuI66s2H3UU=</D>
</RSAKeyValue>";
    private RSACryptoServiceProvider rsa { get; } = new();
    public string? Decrypt(string value)
    {
        rsa.FromXmlString(Key);
        return ToString(rsa.Decrypt(Convert.FromBase64String(value), true));
    }

    public string Encrypt(string value)
    {
        rsa.FromXmlString(Key);
        return Convert.ToBase64String(rsa.Encrypt(ToByte(value), true));
    }

    private byte[] ToByte(string value)
    {
```

```
        return Encoding.UTF8.GetBytes(value);
    }

    private string ToString(byte[] value)
    {
        return Encoding.UTF8.GetString(value);
    }
}
```