МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Программной инженерии

Специальность 6-05-0612-01 «Программная инженерия»

Направление специальности 6-05-0612-01 «Программная инженерия» (программирование интернет приложений)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

по дисциплине «Объектно-ориентированные технологии программирования и стандарты проектирования»

Тема «Цветочный магазин»

Исполнитель

Студентка 2 курса группы 6 Альшевская Алина Михайловна

(Ф.И.О.)

Руководитель работы Мущук А. Н.

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

Курсовой проект защищен с оценкой

Председатель Смелов В.В.

(подпись)

Минск 2025

Оглавление

[**ВВЕДЕНИЕ** 3](#_Toc198854466)

[**1.** **Аналитический обзор литературы и постановка задачи** 4](#_Toc198854467)

[**1.2** **Постановка задачи** 8](#_Toc198854468)

[**2 Анализ требований к программному средству и разработка функциональных требований** 9](#_Toc198854469)

[**2.1 Описание средств разработки** 9](#_Toc198854470)

[**2.1.1 Язык программирования C#** 10](#_Toc198854471)

[**2.1.2 Платформа .NET Framework** 10](#_Toc198854472)

[**2.1.3 Entity Framework** 10](#_Toc198854473)

[**2.1.4 Windows Presentation Foundation** 11](#_Toc198854474)

[**2.1.5 MS SQL Server** 12](#_Toc198854475)

[**2.2 Спецификация функциональных требований к программному средству** 12](#_Toc198854476)

[**2.3 Диаграмма вариантов использования** 13](#_Toc198854477)

[**3 Проектирование программного средства** 13](#_Toc198854478)

[**3.1 Проектирование архитектуры приложения** 14](#_Toc198854479)

[**3.2** **Проектирование структуры базы данных** 16](#_Toc198854480)

[**3.3 Структура проекта** 18](#_Toc198854481)

[**3.4 Диаграмма классов** 19](#_Toc198854482)

[**4 Реализация программного средства** 19](#_Toc198854483)

[**4.1 Основные классы программного средства** 19](#_Toc198854484)

[**4.1.1 Классы для реализации паттерна MVVM** 20](#_Toc198854485)

[**4.1.2 Классы для работы с базой данных** 21](#_Toc198854486)

[**4.2 Реализация функционала приложения** 21](#_Toc198854487)

[**4.2.1 Метод для добавления товара в каталог** 22](#_Toc198854488)

[**4.2.2 Метод для удаления товара из корзины** 22](#_Toc198854490)

[**4.2.3 Метод для регистрации пользователя в приложении** 22](#_Toc198854491)

[**5. Тестирование, проверка на работоспособность, анализ полученных результатов** 23](#_Toc198854492)

[**5.1 Тестирование функций страницы регистрации** 23](#_Toc198854493)

[**5.2 Тестирование функций страницы авторизации** 24](#_Toc198854494)

[**5.3 Тестирование функции для добавления товара** 25](#_Toc198854495)

[**6 Руководство по использованию** 25](#_Toc198854496)

[**6.1 Регистрация и авторизация** 25](#_Toc198854497)

[**6.2 Использование приложения в роли клиента** 27](#_Toc198854498)

[**ПРИЛОЖЕНИЕ В** 32](#_Toc198854499)

[**ПРИЛОЖЕНИЕ Г** 35](#_Toc198854500)

[**ПРИЛОЖЕНИЕ Е** 38](#_Toc198854501)

# **ВВЕДЕНИЕ**

Тема разработки приложения «Цветочный магазин» является актуальной и востребованной в современном мире. В условиях динамичного развития рынка и увеличения спроса на цветочную продукцию эффективное управление ассортиментом, заказами и клиентскими запросами становится ключевым фактором успешной работы магазина. Автоматизация процессов учета товаров, контроля заказов и взаимодействия с клиентами способствует повышению продаж, оптимизации бизнес-процессов и достижению высоких стандартов сервиса.

Анализ конкурентов позволяет выявить сильные и слабые стороны существующих решений. В условиях растущей конкуренции и повышающихся ожиданий покупателей возникает необходимость в инновационных и удобных инструментах, способных удовлетворить потребности клиентов и помочь бизнесу развиваться.

Целью данного курсового проекта является разработка приложения для управления цветочным магазином, которое позволит владельцам эффективно организовывать работу, отслеживать заказы и взаимодействовать с клиентами. Основная задача приложения — предоставление удобного и интуитивно понятного инструмента для автоматизации ключевых бизнес-процессов, что позволит оптимизировать работу магазина и повысить качество обслуживания.

При разработке приложения учитываются несколько ключевых принципов, способствующих достижению поставленных целей. Во-первых, интерфейс должен быть интуитивно понятным и удобным, чтобы владельцы и сотрудники магазина могли быстро освоить систему, независимо от их технического опыта. Во-вторых, приложение должно обеспечивать оперативное управление ассортиментом, заказами и клиентскими запросами, позволяя оптимизировать рабочие процессы и минимизировать затраты времени.

После внедрения приложения для управления цветочным магазином владельцы смогут эффективно организовывать бизнес-процессы, управлять заказами и улучшать взаимодействие с клиентами. Это позволит оптимизировать работу магазина, повысить уровень сервиса и увеличить продажи, делая бизнес более успешным и конкурентоспособным.

1. **Аналитический обзор литературы и постановка задачи**

Сейчас в нашем распоряжении имеются онлайн-сервисы и приложения, которые помогают человеку ответственно планировать свой день, экономить свое время и ресурсы. Мы получаем новые возможности: все наши мысли можно хранить в одном приложении, забыв навсегда о бумажном носителе.

Для того, чтобы окончательно определиться с постановкой задачи курсового проекта, необходимо проанализировать прототипы программных средств выбранной темы. Рассмотрим несколькопрототипов ниже.

Posiflora – приложение, созданное участником фонда «Сколково», в 2017 году, который предназначен для упрощения работы цветочного бизнеса.

На рисунке 1.1 представлен интерфейс приложениямть «Posiflora»

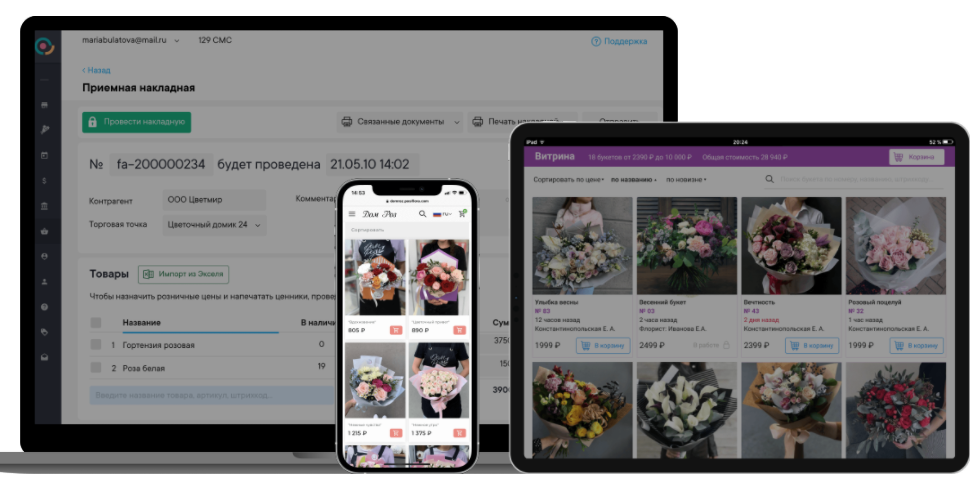


Рисунок 1.1 – Интерфейс приложения Posiflora

В цветочном магазине особенно важно удобное управление ассортиментом и заказами. Приложение обеспечивает онлайн-витрину, где продавец может демонстрировать собранные букеты, предоставляя клиентам удобный способ выбора и оформления заказов. Для продавца доступно отслеживание заказов на всех этапах, начиная с поступления нового заказа и заканчивая его доставкой. Это позволяет четко контролировать процесс выполнения заказов, исключая задержки и возможные ошибки.

Флорист получает мгновенные уведомления при поступлении новых заказов, что позволяет оперативно приступать к их выполнению. Руководитель магазина имеет доступ к детальной информации о состоянии бизнеса, включая сумму денег в кассе, общую выручку и ключевые финансовые показатели за различные периоды. Это помогает принимать обоснованные решения по развитию бизнеса и управлению ресурсами.

Система отслеживает статистику работы каждого флориста, что дает возможность оценивать эффективность их деятельности и поощрять лучших сотрудников. Анализ востребованных цветов и данных о постоянных клиентах помогает оптимизировать ассортимент, обеспечивая наличие самых популярных позиций. Кроме того, приложение позволяет отслеживать остатки цветов и других необходимых материалов на складе, предотвращая нехватку продукции.

На основе данных о текущих остатках можно планировать закупки, обеспечивая своевременное пополнение запасов. Синхронизация с интернет-магазином позволяет автоматически обновлять информацию о складе после каждой онлайн-продажи, исключая несоответствия в наличии товаров.

Плюсы такой системы очевидны: удобное управление заказами, точный контроль остатков, анализ востребованных товаров и оценка работы сотрудников. Это способствует повышению эффективности бизнеса, улучшению обслуживания клиентов и увеличению продаж.

Второй аналог - приложение «FloraPOS». Это программа для автоматизации маленького флористического салона, так и для крупной торговой сети салонов (в том числе с салонами в разных городах).

Приложение обладает рядом значительных преимуществ, которые делают его удобным и эффективным инструментом для ведения бизнеса. Оно позволяет осуществлять продажу и возврат товаров, что способствует повышению удовлетворенности клиентов. Возможность составления букетов дает гибкость в создании уникальных композиций, соответствующих запросам покупателей. Уценка товара помогает эффективно управлять запасами, минимизируя потери и привлекая больше клиентов скидками.

Дополнительным преимуществом является интеграция с интернет-магазинами и CRM-системами, что позволяет автоматически обновлять данные о товарах и заказах, улучшая взаимодействие с клиентами. Также предусмотрена интеграция с курьерской службой и логистическими сервисами, упрощая процесс доставки заказов. Важной функцией является списание товара и частичное списание букетов при его переборке, что помогает управлять складскими запасами и контролировать качество продукции.

Однако у приложения есть и недостатки, которые могут повлиять на его восприятие пользователями. Во-первых, сервис является платным, что может ограничить доступ к нему для небольших цветочных магазинов. Во-вторых, устаревший дизайн может снизить удобство использования и привлекательность интерфейса. Отсутствие поддержки форматирования может стать проблемой при внесении изменений в данные или создание более детализированных описаний товаров.

На рисунке 1.2 представлена рабочая страница приложения «FloraPOS»



Рисунок 1.2 – Приложение FloraPOS

В качестве третьего аналога рассматривается приложение «FloraPoint».

На рисунке 1.3 представлен главная страница приложаения.

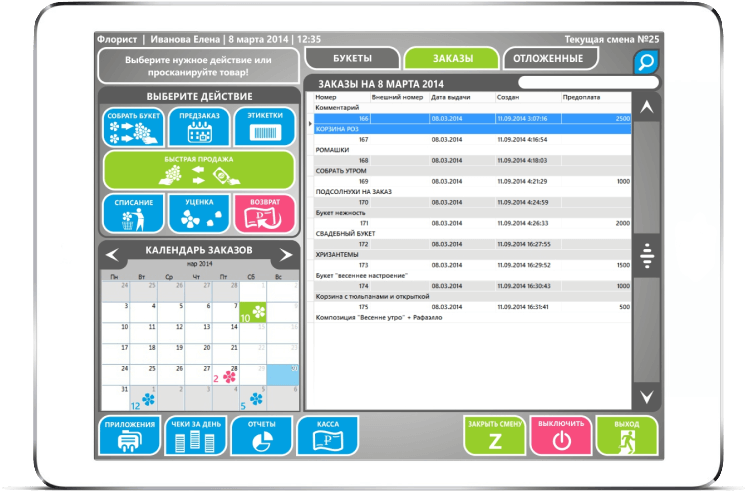


Рисунок 1.3 – Главная страница приложения «FloraPoint»

Это приложение предлагает удобные решения для организации работы. Разделенные рабочие места флористов, кассиров и администраторов позволяют каждому сотруднику комфортно выполнять свои задачи, не мешая другим. Это способствует эффективному ведению бизнеса и снижает вероятность ошибок при обработке данных.

Функция фиксирования скидок, включая скидки по группам клиентов и накопительные скидки, помогает привлекать покупателей и стимулировать повторные покупки. Гибкая система скидок делает магазин более конкурентоспособным и повышает лояльность клиентов.

Отслеживание заказов и предзаказов в режиме реального времени дает возможность контролировать выполнение заказов, минимизировать задержки и оперативно реагировать на изменения. Это особенно важно для обеспечения высокого уровня сервиса и удовлетворенности клиентов.

Приложение также позволяет оформлять поступающие товары и загружать документы из Excel, что упрощает управление складом и инвентаризацией. Автоматизация этих процессов снижает вероятность ошибок и экономит время сотрудников.

Онлайн-витрина предоставляет всю необходимую информацию о каждом букете, включая данные о флористе, дату и время сборки, а также комментарии. Это делает процесс выбора удобным для клиентов и помогает им принимать осознанные решения при покупке.

# **1.2 Постановка задачи**

Основываясь на анализе аналогичных решений, проектируемое программное средство для цветочного магазина должно включать следующие характеристики и функциональные возможности:

* **Интуитивно понятный интерфейс**, обеспечивающий удобную навигацию как для клиентов, так и для сотрудников магазина;
* **Личный кабинет пользователя и администратора**, позволяющий клиентам управлять заказами, а администраторам контролировать бизнес-процессы;
* **Оформление заказа**;
* **Написание отзывов пользователем**;
* **Модерирование отзывов администратором**, позволяющее удалять нежелательные комментарии;
* **Управление заказами администратором**, включая изменение статуса заказа и его содержания;
* **Просмотр истории заказов**;
* **Поиск и фильтрация в каталоге**;
* **Управление ассортиментом и ценами администратором**, с возможностью изменения товаров и добавления новых.

Реализация этих функций обеспечит высокую продуктивность работы и положительный пользовательский опыт, способствуя успешному выполнению заказов.

# **2 Анализ требований к программному средству и разработка функциональных требований**

Анализ требований играет важнейшую роль в процессе разработки программного обеспечения, поскольку он определяет перечень функций и возможностей, необходимых для удовлетворения запросов пользователей и заказчиков. На этом этапе осуществляется сбор, документирование, уточнение и систематизация требований, что позволяет избежать ошибок, недоразумений и несоответствий в последующих этапах разработки.

Формирование функциональных требований является ключевым аспектом, так как именно они определяют, какие возможности будут предоставлены пользователям при взаимодействии с приложением. Эти требования служат базой для проектирования архитектуры системы, ее последующего создания, а также являются основными критериями для тестирования и оценки готовности конечного продукта.

В данном разделе пояснительной записки будет представлена информация о технических решениях, используемых в процессе разработки, а также подробное описание архитектурных особенностей системы.

# **2.1 Описание средств разработки**

Для разработки приложения «Цветочный магазин» были использованы современные технологии, обеспечивающие удобство работы, высокую производительность и надежность. В качестве основного языка программирования выбран C#, а платформа .NET Framework позволила создать стабильную и функциональную систему. Для реализации графического интерфейса применена технология Windows Presentation Foundation (WPF), которая обеспечивает удобное взаимодействие пользователей с приложением.

Для управления данными используется реляционная база MS SQL Server, обеспечивающая быструю обработку информации, высокий уровень безопасности и надежность при работе с большими объемами данных. Взаимодействие приложения с базой данных реализовано с помощью фреймворка Entity Framework, который упрощает работу с данными, позволяя разработчикам использовать .NET-объекты для управления информацией. Такой подход делает процесс разработки более эффективным и удобным.

# **2.1.1 Язык программирования C#**

C# — это мощный объектно-ориентированный язык программирования, разработанный Microsoft и широко используемый для создания приложений на платформе .NET. Он подходит для разработки веб-, мобильных и настольных программ благодаря своей высокой производительности и гибкости. Компиляция в промежуточный язык позволяет оптимизировать выполнение кода, обеспечивая его эффективность.

Один из ключевых аспектов C# — простой и понятный синтаксис, который упрощает процесс написания и поддержки кода, делая язык удобным даже для начинающих разработчиков. Кроме того, он поддерживает интеграцию с различными библиотеками и фреймворками, такими как ASP.NET и Entity Framework, что ускоряет процесс разработки и расширяет функциональные возможности приложений. Это делает C# универсальным инструментом для создания современных программных решений.

# **2.1.2 Платформа .NET Framework**

.NET Framework — это программная платформа, разработанная Microsoft для создания и запуска приложений, преимущественно на Windows. Она включает в себя среду выполнения, обеспечивающую корректную работу программ, а также набор библиотек, необходимых для их разработки.

Основными компонентами .NET Framework являются общая языковая среда выполнения (Common Language Runtime, CLR), библиотека классов .NET Framework и инструменты разработки, которые помогают программистам эффективно реализовывать свои проекты. CLR отвечает за управление кодом во время выполнения, обеспечивая безопасность, сборку мусора и обработку исключений. Библиотека классов предоставляет широкий набор API для работы с данными, файловыми системами, графикой и сетевыми взаимодействиями.

# **2.1.3 Entity Framework**

Entity Framework — это мощный инструмент для работы с базами данных в среде .NET, который значительно упрощает взаимодействие с данными. Он позволяет разработчикам работать с базами данных через объектно-ориентированную модель, избавляя их от необходимости писать сложные SQL-запросы. Благодаря поддержке автоматического создания и обновления схемы базы данных, разработчики могут легко вносить изменения в структуру данных без риска потери информации. Entity Framework совместим с различными системами управления базами данных, такими как MS SQL Server, PostgreSQL и MySQL, что делает его универсальным решением для различных проектов2.

Гибкость и расширяемость фреймворка позволяют настраивать модели данных и использовать различные стратегии загрузки информации, что делает работу с ним удобной и эффективной. Кроме того, Entity Framework поддерживает несколько режимов работы с данными, включая ленивую и жадную загрузку, что позволяет оптимизировать производительность приложения. Он также предоставляет мощные инструменты для работы с запросами, позволяя использовать LINQ для удобного и гибкого взаимодействия с данными. Благодаря встроенной поддержке миграций разработчики могут легко управлять изменениями в структуре базы данных, упрощая процесс обновления приложения.

# **2.1.4 Windows Presentation Foundation**

Windows Presentation Foundation (WPF) — это современная технология для разработки графических интерфейсов приложений в среде .NET. Она создана Microsoft и предназначена для создания настольных приложений с привлекательным дизайном и интерактивными элементами.

Одним из ключевых преимуществ WPF является использование языка разметки XAML, который позволяет описывать интерфейсы декларативно, упрощая процесс разработки и отделяя визуальную часть от логики приложения. Это дает разработчикам возможность легко изменять дизайн без необходимости корректировки программного кода.

WPF поддерживает работу с 2D и 3D графикой, анимацией, мультимедиа и стилями, что делает его мощным инструментом для создания сложных пользовательских интерфейсов. Важной особенностью является привязка данных, которая позволяет динамически обновлять элементы интерфейса в зависимости от изменений в данных.

Кроме того, WPF упрощает взаимодействие между пользовательским интерфейсом и бизнес-логикой, обеспечивая удобную работу с данными. Это позволяет разработчикам создавать более динамичные и отзывчивые приложения.

В целом, WPF — это универсальная платформа, объединяющая мощные инструменты разработки и возможности для создания стильных и функциональных приложений, соответствующих современным требованиям пользователей

# **2.1.5 MS SQL Server**

MS SQL Server — это мощная реляционная система управления базами данных (СУБД), разработанная Microsoft. Она предназначена для эффективного хранения, обработки и управления данными, обеспечивая высокую производительность, надежность и безопасность.

Одним из ключевых преимуществ MS SQL Server является поддержка транзакционной обработки, что гарантирует целостность данных и предотвращает их потерю. Система предоставляет широкий набор инструментов для работы с запросами, включая язык SQL и механизмы оптимизации выполнения запросов, что ускоряет обработку информации.

Кроме того, MS SQL Server включает в себя развитые средства аналитики и управления данными, такие как хранимые процедуры, триггеры и индексы, позволяющие оптимизировать работу с базой данных. Встроенные инструменты мониторинга и управления производительностью делают систему удобной для администраторов, помогая контролировать состояние базы и обеспечивать ее стабильную работу.

# **2.2 Спецификация функциональных требований к программному средству**

Программное средство должно предоставлять следующие функциональные возможности:

Для администратора:

* + Выполнять авторизацию.
  + Просматривать каталог товаров.
  + Выполнять поисковые запросы, фильтрацию.
  + Добавление/удаление новых товаров в каталог.
  + Редактирование информации о товаре в каталоге.
  + Просматривать, изменять и удалять заказы.
  + Просматривать и удалять отзывы.
  + Добавление клиентов.

Для клиента:

* + Выполнять регистрацию и авторизацию.
  + Просматривать каталог товаров.
  + Выполнять поисковые запросы, фильтрацию.
  + Добавлять товар в корзину.
  + Удалять/изменять количество товара в корзине.
  + Оформлять заказ.
  + Просматривать информацию о статусе заказа.
  + История заказов.
  + Оставлять отзывы.
  + Редактировать профиль.

# **2.3 Диаграмма вариантов использования**

Диаграмма вариантов использования (Use Case Diagram) — это графический инструмент, предназначенный для визуализации функциональности системы с точки зрения её пользователей. Она помогает определить возможные сценарии взаимодействия, показывая связь между пользователями (актерами) и системой.

Диаграмма включает в себя несколько ключевых элементов: актеров, варианты использования и связи между ними. Актеры представляют собой пользователей или внешние системы, а варианты использования отражают доступные функции приложения. Визуально актеры изображаются в виде фигур, а варианты использования — в форме овалов, соединённых стрелками, обозначающими направление взаимодействия.

Использование диаграммы вариантов позволяет:

* наглядно представить все возможные сценарии работы системы и понять, как пользователи взаимодействуют с ней;
* определить функциональные требования на основе реальных сценариев использования;
* улучшить коммуникацию между разработчиками, заказчиками и другими заинтересованными сторонами.

Диаграмма вариантов использования для приложения «Система управления проектами» представлена в приложении А.

# **3 Проектирование программного средства**

Проектирование программного обеспечения — это ключевой этап разработки, определяющий архитектуру, структуру и функциональность будущего приложения. В рамках этого процесса создаются модели данных, интерфейсы, алгоритмы и механизмы взаимодействия компонентов системы.

Первым шагом является анализ требований, который помогает определить основные функции и возможности программного продукта. Затем разрабатываются диаграммы, визуализирующие структуру системы, включая диаграммы классов, вариантов использования и последовательностей.

Выбор архитектуры играет решающую роль, поскольку определяет принципы взаимодействия компонентов приложения. Важно учитывать такие аспекты, как масштабируемость, надежность и безопасность, чтобы обеспечить стабильную работу системы.

Кроме того, проектирование охватывает разработку пользовательского интерфейса, который должен быть удобным и интуитивно понятным. Грамотно продуманная навигация и оптимизированное взаимодействие с пользователем способствуют повышению эффективности работы с приложением. Такой подход позволяет создать функциональное и удобное программное средство, соответствующее требованиям пользователей и бизнес-задачам.

# **3.1 Проектирование архитектуры приложения**

Архитектура программного обеспечения представляет собой совокупность решений, определяющих структуру и организацию системы. Она включает выбор компонентов, их интерфейсов и способов взаимодействия, а также интеграцию в более крупные программные комплексы. Важную роль играет архитектурный стиль, который задает общую структуру системы, определяя элементы, их связи и принципы взаимодействия.

Для обеспечения высокого качества проектируемой системы используются различные архитектурные паттерны. В данном проекте применяется шаблон Model-View-ViewModel (MVVM), который состоит из трех ключевых компонентов: модели данных (Model), модели представления (ViewModel) и пользовательского интерфейса (View). Такой подход способствует четкому разделению логики приложения, упрощая процесс разработки, тестирования и поддержки.

Использование архитектурного шаблона MVVM повышает модульность системы, облегчает повторное использование кода и делает процесс разработки более удобным. Благодаря четкому разграничению бизнес-логики и пользовательского интерфейса приложение становится гибким и масштабируемым, что позволяет ему адаптироваться к изменяющимся требованиям и расширяться в будущем

На рисунке 3.1 представлена структура архитектуры MVVM.

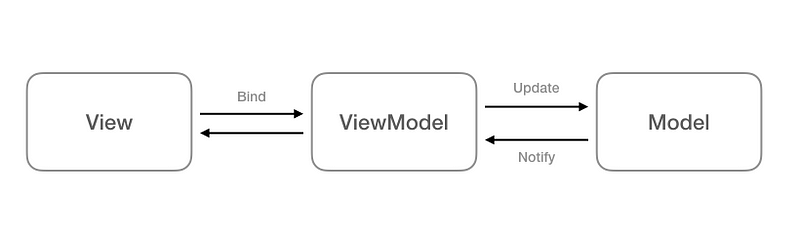


Рисунок 3.1 – Структура MVVM

Model — это компонент, отвечающий за обработку данных и реализацию бизнес-логики приложения. Он управляет информацией, взаимодействует с базой данных и выполняет вычисления, оставаясь независимым от пользовательского интерфейса. Благодаря этому Model можно использовать в различных приложениях без необходимости внесения изменений.

View представляет собой визуальную часть системы, отображающую данные пользователю. Включает элементы интерфейса, такие как кнопки, текстовые поля и списки. Хотя View реагирует на действия пользователя, оно не содержит логики обработки данных — вся обработка выполняется в ViewModel.

ViewModel играет роль посредника между Model и View. Оно управляет данными, полученными из Model, и подготавливает их для отображения в View. Кроме того, ViewModel обрабатывает пользовательские команды и обновляет данные в Model. Благодаря механизму привязки данных (Data Binding) изменения в Model автоматически отражаются в View, обеспечивая динамическое обновление интерфейса. Такой подход делает систему гибкой и удобной в использовании.

Также, в проекте применяются паттерн для работы с базой данных Repository.

Паттерн **Repository** — это архитектурный шаблон проектирования, который используется для абстрагирования доступа к данным и управления ими. Он действует как промежуточный слой между бизнес-логикой приложения и системой хранения данных, обеспечивая централизованный способ работы с информацией. Основная цель паттерна Repository — отделить логику работы с данными от бизнес-логики, что делает код более чистым, удобным для тестирования и гибким в отношении изменений в системе хранения данных. Вместо того чтобы напрямую взаимодействовать с базой данных, приложение обращается к репозиторию, который управляет операциями чтения, записи, обновления и удаления данных.

# **3.2 Проектирование структуры базы данных**

База данных представляет собой структурированное хранилище информации, предназначенное для удобного управления, обработки и систематизации данных в электронном виде. Они широко применяются в различных сферах, включая бизнес, медицину, социальные сети и онлайн-торговлю, обеспечивая быстрый доступ к необходимым сведениям и эффективное управление большими объемами информации.

Система управления базами данных (СУБД) — это специализированное программное обеспечение, которое позволяет создавать, организовывать и использовать базы данных. Она предоставляет инструменты для хранения, поиска, обновления и удаления данных, а также гарантирует их безопасность и целостность, обеспечивая надежное функционирование информационных систем.

В данном проекте для создания и управления базой данных использовалась реляционная СУБД Microsoft SQL Server, которая предоставляет мощные средства для работы с данными и обеспечивает высокую производительность и безопасность.

База данных приложения «Система управления проектами» состоит из 9 таблиц: Users, Products, Categories, Colors, ProductCategory, ProductColor Orders, Reviews, OrderDetails.

Таблица Products хранит информацию о товарах.

Таблица Users хранит информацию о пользователях.

Таблица Orders хранит информацию о заказах.

Таблица Reviews хранит информацию о отзывах.

Таблица OrderDetail хранит информацию о содержимом заказа, она нужна для того, чтобы в одном заказе можно было указать несколько товаров.

Схема базы данных FlowerShop\_Db изображена на рисунке 3.2.

|  |
| --- |
|  |

Рисунок 3.2 — Схема базы данных приложения

В разработке приложения для работы с базой данных используется библиотека Entity Framework Core 9.0.4, которая предоставляет удобные инструменты для взаимодействия с таблицами базы данных через объектно-ориентированную модель C#. В отличие от ADO.NET, разработчикам не требуется вручную писать SQL-запросы, поскольку обработка данных осуществляется с помощью LINQ, а преобразования выполняются автоматически.

Существует три основных метода проектирования базы данных: Model-First, Database-First и Code-First. В данном проекте был выбран подход Code-First, который позволяет сначала определить модели данных в коде, а затем автоматически создать соответствующую структуру базы данных. Такой метод упрощает процесс разработки, позволяя сосредоточиться на бизнес-логике приложения, а не на ручном проектировании базы данных.

Code-First также облегчает тестирование и поддержку кода, позволяя создавать тестовые данные без сложных манипуляций с базой данных. Этот подход хорошо сочетается с принципами Agile-разработки, обеспечивая гибкость и возможность быстрого внесения изменений в структуру данных. Благодаря этому разработка становится более удобной и эффективной.

# **3.3 Структура проекта**

Структура проекта представлена на рисунке 3.3.

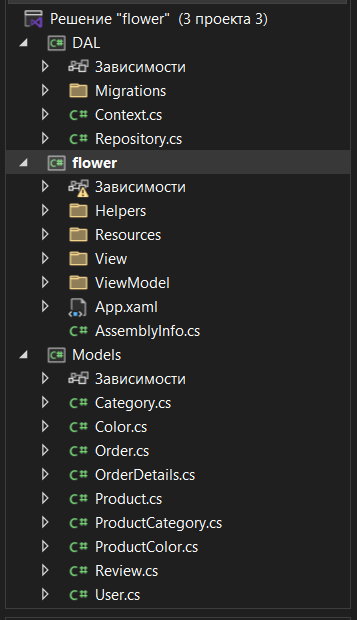


Рисунок 3.3 – Структура проекта

Папка «Migrations» хранит все классы миграций базы данных.

Папка «Views» содержит файлы с разметкой XAML, которые определяют пользовательский интерфейс (UI) приложения.

Папка «ViewModel» хранит классы, которые являются промежуточным слое между моделью и представлением, обрабатывают данные и команды.

Папка «Resource» хранит в себе вспомогательные словари для локализации приложения.

Папка «Helpers» хранит вспомогательные классы, нужные для обработки команд из ViewModel и корректной обработки исключений.

Библиотека классов «Models» содержит в себе модели, на основе которых строится приложения (по паттерну MVVM) и база данных.

Файл «App.xaml» хранит общие ресурсы проекта.

# **4 Реализация программного средства**

На данном этапе осуществляется разработка программного обеспечения, реализующего предложенное техническое решение, а также сборка готового продукта. В ходе работы создаются программные интерфейсы, обеспечивающие взаимодействие между классами, методами и функциями.

Одним из ключевых аспектов является документирование классов, их атрибутов и методов, что способствует ясности и удобству дальнейшего использования. Процесс программирования включает написание исходного кода и отладку отдельных модулей проекта.

По завершении этого этапа будет создано полнофункциональное программное средство, готовое к эксплуатации.

# **4.1 Основные классы программного средства**

В приложении создаётся большое количество классов, каждый из которых предназначен для выполнения определённых функций, что делает систему более модульной и гибкой. В этом разделе акцент будет на классах, отвечающих за реализацию паттерна MVVM, а также за работу с базой данных.

Архитектурный паттерн MVVM делит приложение на слои, обеспечивая четкое разделение между логикой представления и основной бизнес-логикой. Классы, управляющие взаимодействием с базой данных, играют ключевую роль в эффективном управлении данными. Они отвечают за такие задачи, как выполнение запросов, управление транзакциями и поддержка соединений с базой данных. Это позволяет приложению динамически загружать, обновлять и сохранять информацию, гарантируя её актуальность.

Эти компоненты важны для успешного функционирования приложения, поскольку они обеспечивают взаимодействие между интерфейсом и хранимыми данными, что способствует общей стабильности и эффективности системы.

# **4.1.1 Классы для реализации паттерна MVVM**

Пример использования паттерна MVVM рассматривается на основе страницы авторизации. В листинге 4.1 представлен код фрагмента представления (view) этой страницы, включающий XAML-разметку для создания кнопки входа. В данном примере демонстрируется механизм привязки (binding) команды LogInCommand к кнопке входа, что иллюстрирует связь представления (view) с моделью представления (viewmodel). Это позволяет элементам управления в представлении выполнять команды, определенные в модели представления, обеспечивая четкое разделение интерфейса и логики приложения.

В приложение Б будет вынесен полный код данной страницы для подробного ознакомления.

|  |
| --- |
| <Button Grid.Row="5" Content="{DynamicResource EnterLoginButtonName}" Command="{Binding LogInCommand}" FontSize="30" MinHeight="70" Margin=" 0 10 0 10" ></Button> |

Листинг 4.1 – Фрагмент кода xaml, демонстрирующий кнопку с привязкой к команде

Код модели представления (ViewModel) данной страницы будет размещен в приложении В. ViewModel играет роль связующего элемента между View и Model, предоставляя данные для отображения и обрабатывая пользовательские действия. Этот класс реализует интерфейс INotifyPropertyChanged, позволяя динамически обновлять UI посредством события PropertyChanged при изменении свойств. ViewModel получает данные из Model, преобразует их для View и выполняет логику приложения, не изменяя Model напрямую, а вызывая её методы. В данном классе обработка пользовательских действий осуществляется через команды (Commands).

Перейдем к последнему компоненту паттерна MVVM, отвечающему за работу с данными. В коде команды LoginCommand происходит обращение к данным пользователей через модель (Model) User. В сущности, модель данных представляет собой обычный класс, определяющий данные приложения. Она предоставляет интерфейс (методы и свойства) для доступа и модификации данных, может включать логику валидации и преобразования данных, а также механизмы сохранения и извлечения их из хранилища.

Основные преимущества модели данных включают возможность повторного использования в различных частях приложения или даже в разных приложениях, легкость тестирования, четкое разделение ответственности между элементами MVVM и гибкость при изменениях или расширении функционала без влияния на UI или ViewModel. Поскольку этот класс не зависит от конкретной реализации пользовательского интерфейса, его можно применять в разных приложениях и на различных страницах, а также тестировать отдельно от UI-логики, что делает процесс разработки более удобным и поддерживаемым.

Анализ реализации паттерна MVVM на примере страницы авторизации приложения показывает эффективную организацию системных компонентов. Классы, отвечающие за модель, представление и модель представления, чётко разделяют логику и интерфейс, что значительно упрощает процесс разработки и тестирования.

# **4.1.2 Классы для работы с базой данных**

Для работы с базой данных, взаимодействие с которой осуществляется с помощью библиотеки Entity Framework, необходимы специальные классы. Один из них — класс сущности, который представляет собой модель данных, а другой — класс контекста данных, предназначенный для непосредственного взаимодействия с базой. Класс контекста наследуется от DbContext, а его реализация с использованием DbSet позволяет задать перечень таблиц, создаваемых в базе данных. Кроме того, данный класс должен включать два перегруженных метода: OnModelCreating и OnConfiguring.

OnModelCreating отвечает за настройку модели данных при создании базы, позволяя определить соответствие между классами и таблицами, а также задать связи между сущностями, ограничения и индексы. OnConfiguring, в свою очередь, используется для конфигурации параметров подключения к базе данных, где можно указать строку подключения, тип провайдера и дополнительные параметры, такие как уровень ведения журнала и параметры кэширования.

Полный код реализации данного класса приведен в приложении Г.

# **4.2 Реализация функционала приложения**

В данном подразделе будут представлены некоторые методы, обеспечивающие функционал приложения: метод для добавления и удаления товара из избранного, метод для добавления и удаления товара из корзины, метод для регистрации пользователя в приложении, а также метод для формирования заказа и отправки его на рассмотрение администратору.

# **4.2.1 Метод для добавления товара в каталог**

# Код данного метода представлен в приложении Е. В данном методе реализуется процесс добавления нового продукта в базу данных. Сначала выполняется проверка переданного объекта `product` на `null`, и в случае отсутствия данных вызывается исключение `ArgumentNullException`. Далее продукт добавляется в коллекцию `Products` через контекст `\_context`, а затем сохраняются изменения в базе данных методом `SaveChanges()`. После этого проверяется количество сохраненных изменений: если они были успешно зафиксированы, метод возвращает `true`, в противном случае — `false`. Такой подход гарантирует корректную обработку данных при добавлении нового продукта.

# **4.2.2 Метод для удаления товара из корзины**

Код данного метода представлен в приложении Е. В данном методе происходит удаление товара из корзины. Сначала создается копия текущего списка Cart в виде ObservableCollection<Product> previous, после чего корзина очищается, а общая стоимость (TotalCart) сбрасывается. Далее удаляется переданный товар из previous, а оставшиеся товары добавляются обратно в Cart, при этом пересчитывается сумма стоимости товаров в корзине. Такой подход гарантирует обновление данных корзины после удаления товара и поддержание корректного отображения общей стоимости.

# **4.2.3 Метод для регистрации пользователя в приложении**

Код данного метода представлен в приложении Е. В данном методе реализуется процесс регистрации нового пользователя. Сначала извлекается список всех пользователей с помощью метода GetAllUsers() из репозитория приложения. Затем создается новый объект User, содержащий введенные пользователем данные. После этого выполняется проверка корректности введенной информации, включая валидацию имени, фамилии, пароля и электронной почты. Также проверяется, существует ли уже пользователь с таким номером телефона. Если все проверки пройдены, новый пользователь добавляется в базу данных, а его данные очищаются из формы регистрации. В случае успешного добавления открывается главное окно пользователя, а окно регистрации закрывается. Если же добавление не удалось, выводится сообщение об ошибке. Такой подход обеспечивает надежную регистрацию пользователей, предотвращая дублирование данных и гарантируя корректность введенной информации.

# **5. Тестирование, проверка на работоспособность, анализ полученных результатов**

Тестирование приложения — это процесс проверки и анализа его работы, направленный на выявление соответствия фактического поведения программы заявленным требованиям.Основные цели тестирования:

* подтверждение соответствия программного обеспечения установленным требованиям;
* обнаружение случаев некорректного или нежелательного поведения системы.

Для обеспечения стабильной работы приложения проводится обработка возможных ошибок, возникающих в процессе его функционирования. Поскольку программное средство взаимодействует с базой данных, некорректный ввод информации или её отсутствие могут привести к сбоям в работе системы. Поэтому тестирование играет ключевую роль в выявлении и устранении таких проблем, обеспечивая надежность и корректность работы приложения.

# **5.1 Тестирование функций страницы регистрации**

Первым тестом будет попытка оставить одно поле пустым, например, поле имени. Результат показан на рисунке 5.1.

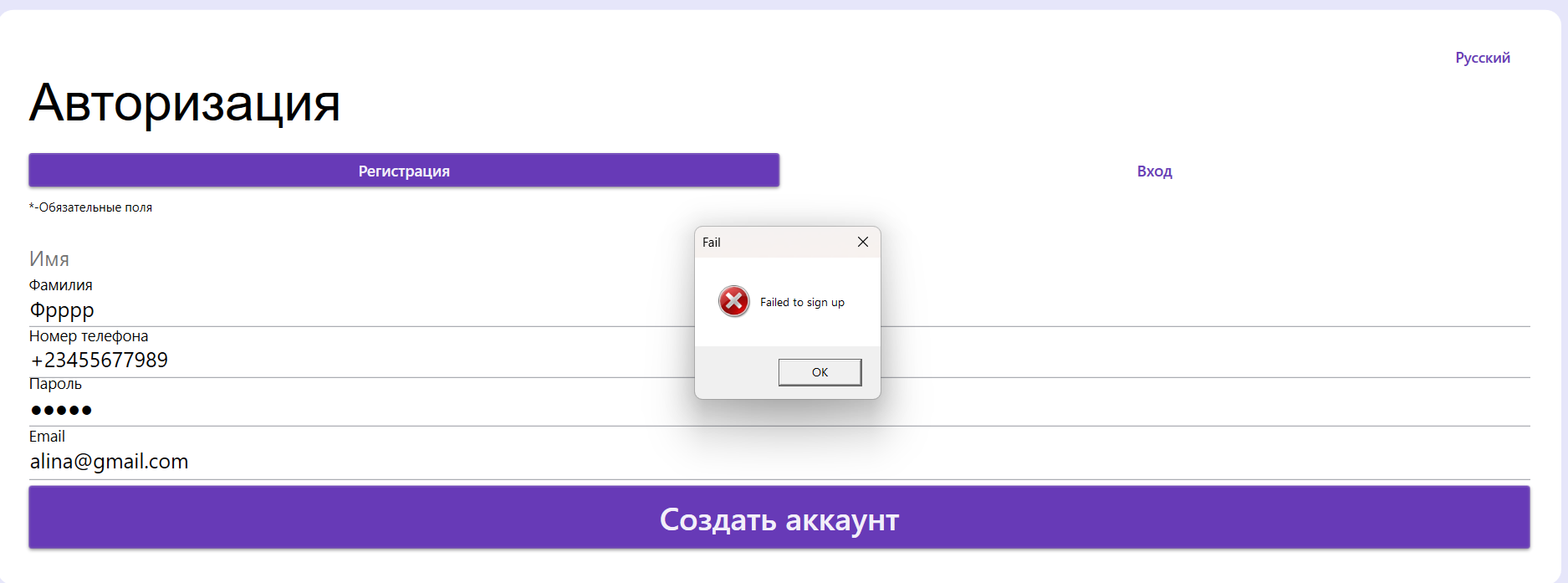


Рисунок 5.1 – Тест на пустое поле имени

Далее вторым тестом будет попытка ввести номер телефона, который уже занято. Результат отображен на рисунке 5.2.

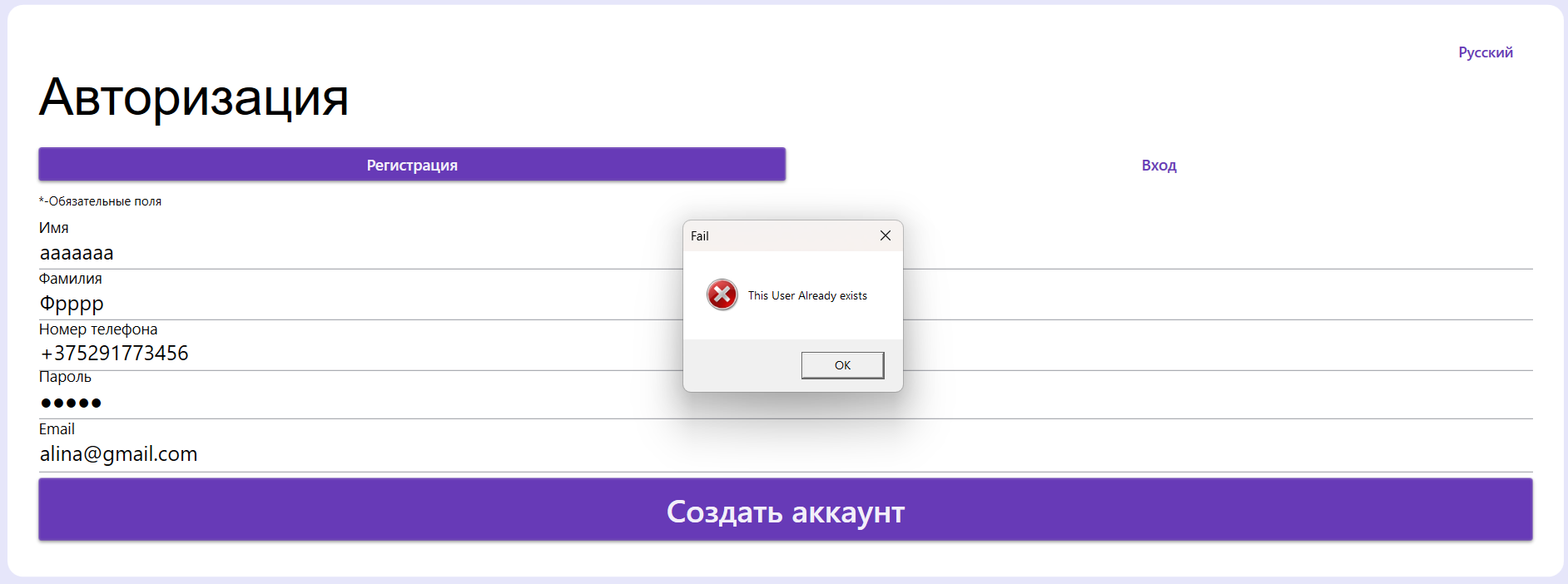


Рисунок 5.2 – Тест на уже занятое пользовательское имя

Можно сделать вывод, что в окне регистрации валидация работает корректно и информирует пользователя об совершенных ошибках.

# **5.2 Тестирование функций страницы авторизации**

Первым тестом попробуем оставить поле пароля пустым. Результат представлен на рисунке 5.3.

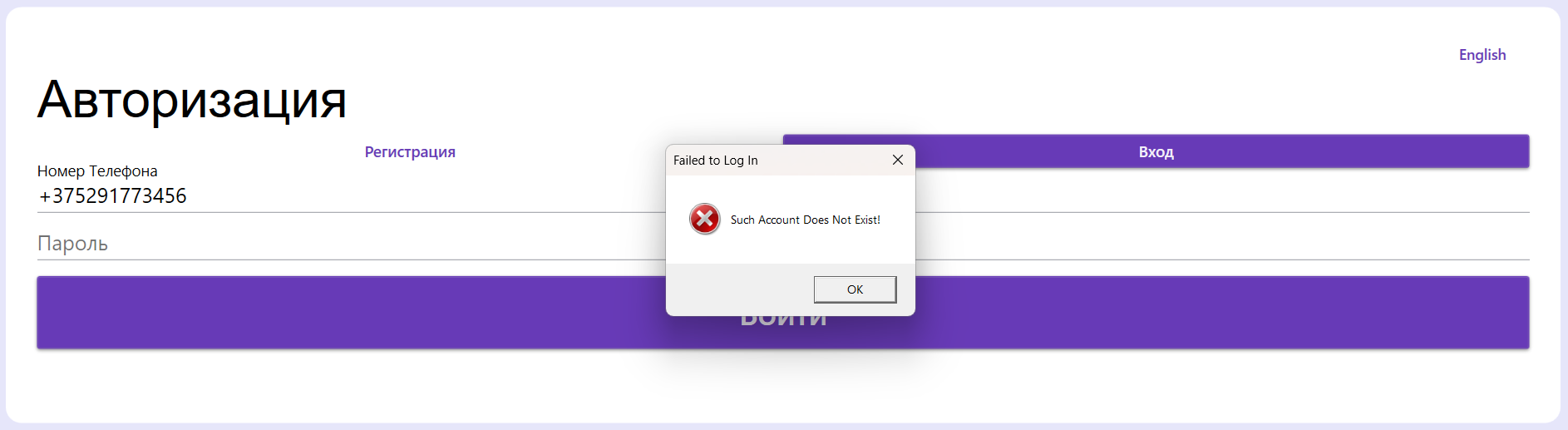


Рисунок 5.3 – Тест на пустое поле с паролем

Вторым тестом попробуем ввести данные аккаунта, которого не существует. Результат показан на рисунке 5.4.

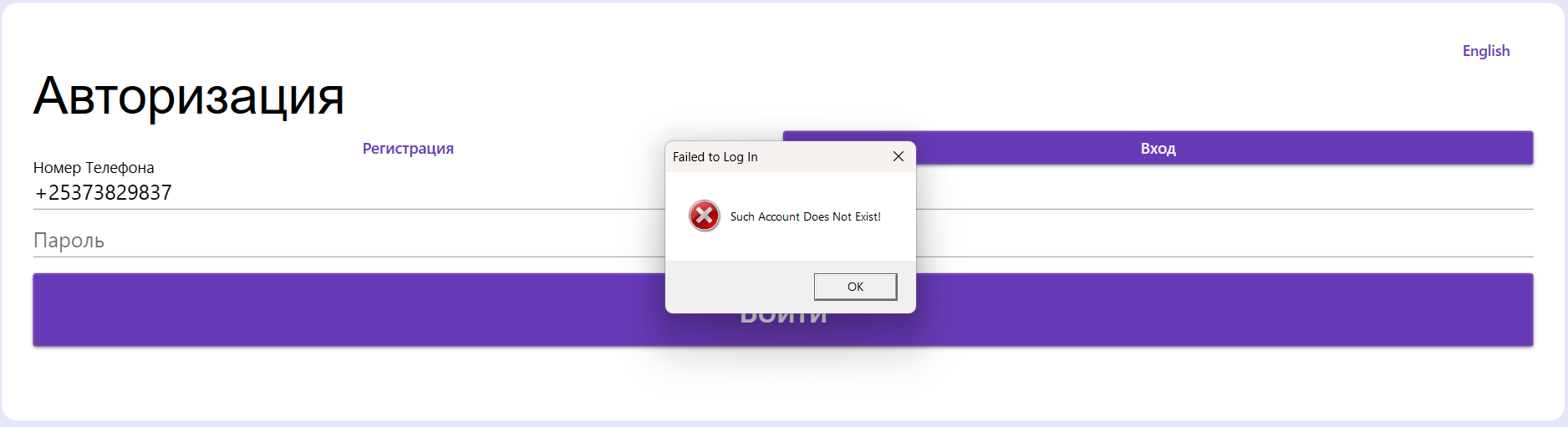


Рисунок 5.4 – Тест на некорректные данные аккаунта

Можно сделать вывод, что в окне авторизации валидация также работает корректно и информирует пользователя об совершенных ошибках.

# **5.3 Тестирование функции для добавления товара**

В данном подразделе протестируем добавление и удаление товара администратором. Первым тестом попробуем в поле количество задать не цифровое значение, а строку.

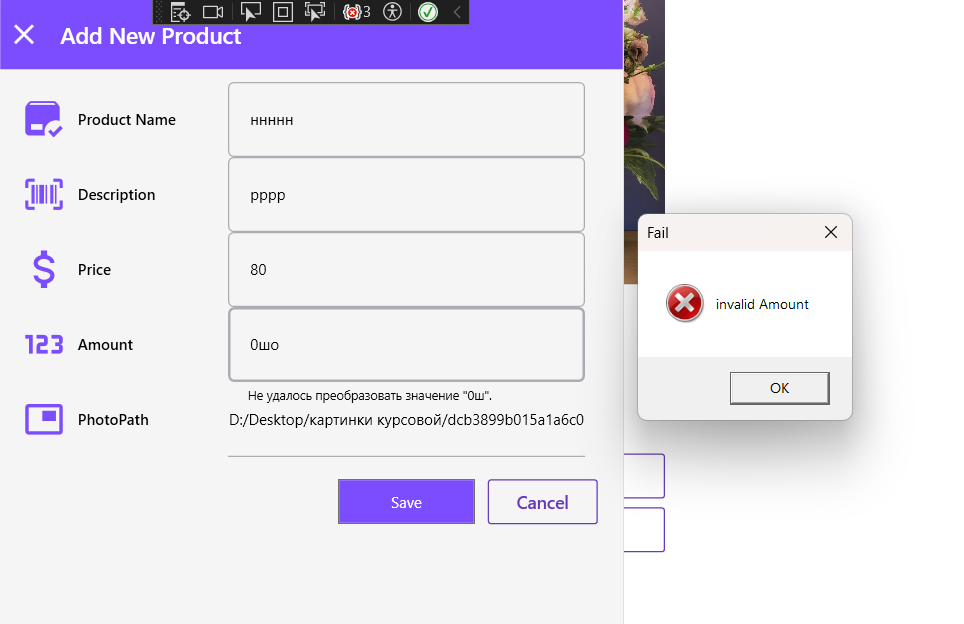


Рисунок 5.5 – Тест добавления товара

Валидация выполняется, администратор получает сообщение об ошибке.

# **6 Руководство по использованию**

# **6.1 Регистрация и авторизация**

При запуске приложения пользователь попадает на страницу авторизации, где от него требуется ввод пользовательского имени и пароля для входа (рисунок 6.1).

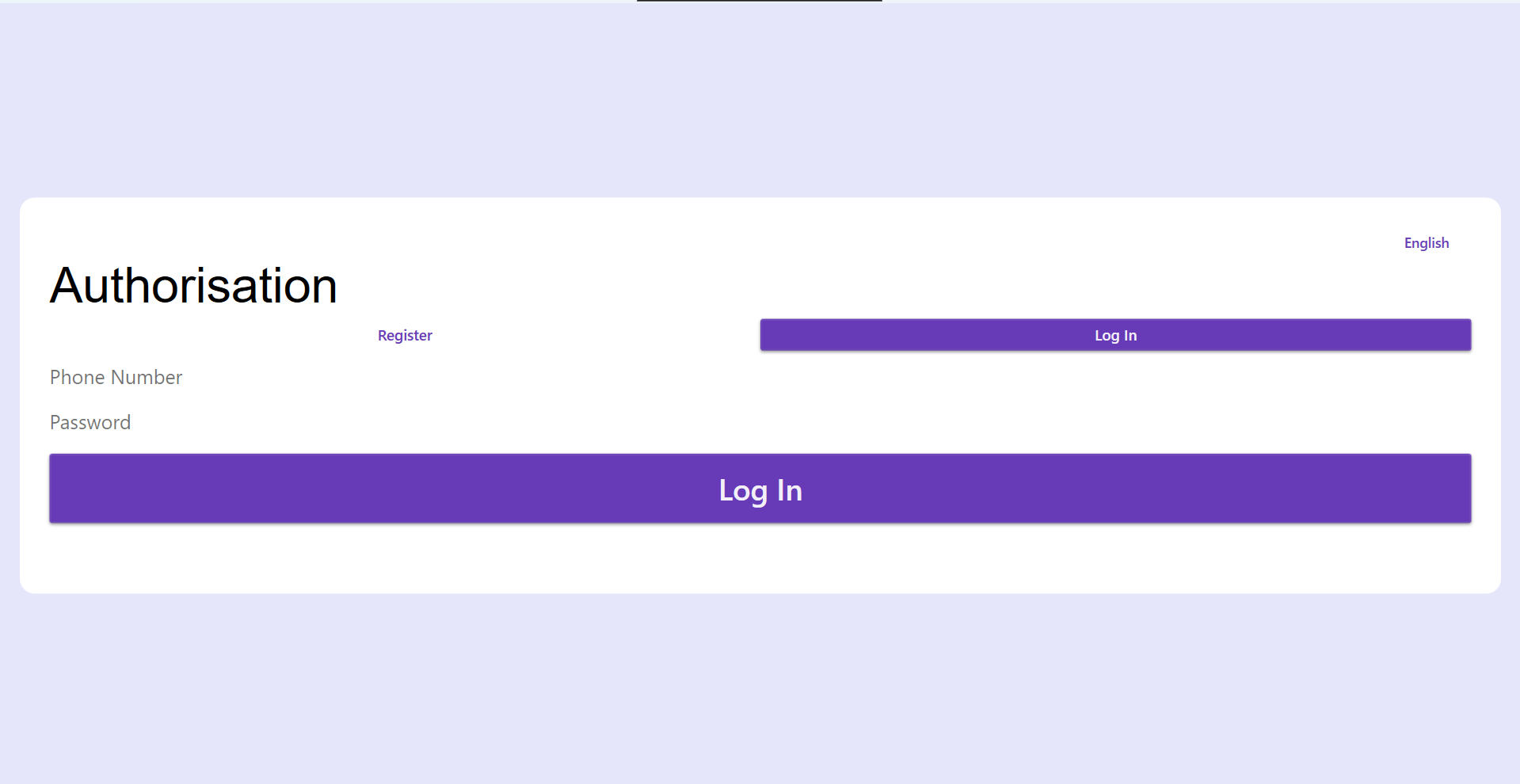


Рисунок 6.1 – Страница авторизации

Если же пользователь первый раз в приложении и у него еще нет аккаунта, то можно нажать на кнопку «Зарегистрироваться» в окне авторизации. Тогда откроется страница регистрации (рисунок 6.2), где нужно будет заполнить все поля для успешного создания аккаунта. Пока пользователь не заполнит поля корректно регистрация не пройдет успешно. Теперь можно будет зайти в только что созданный аккаунт используя соответствующие данные.

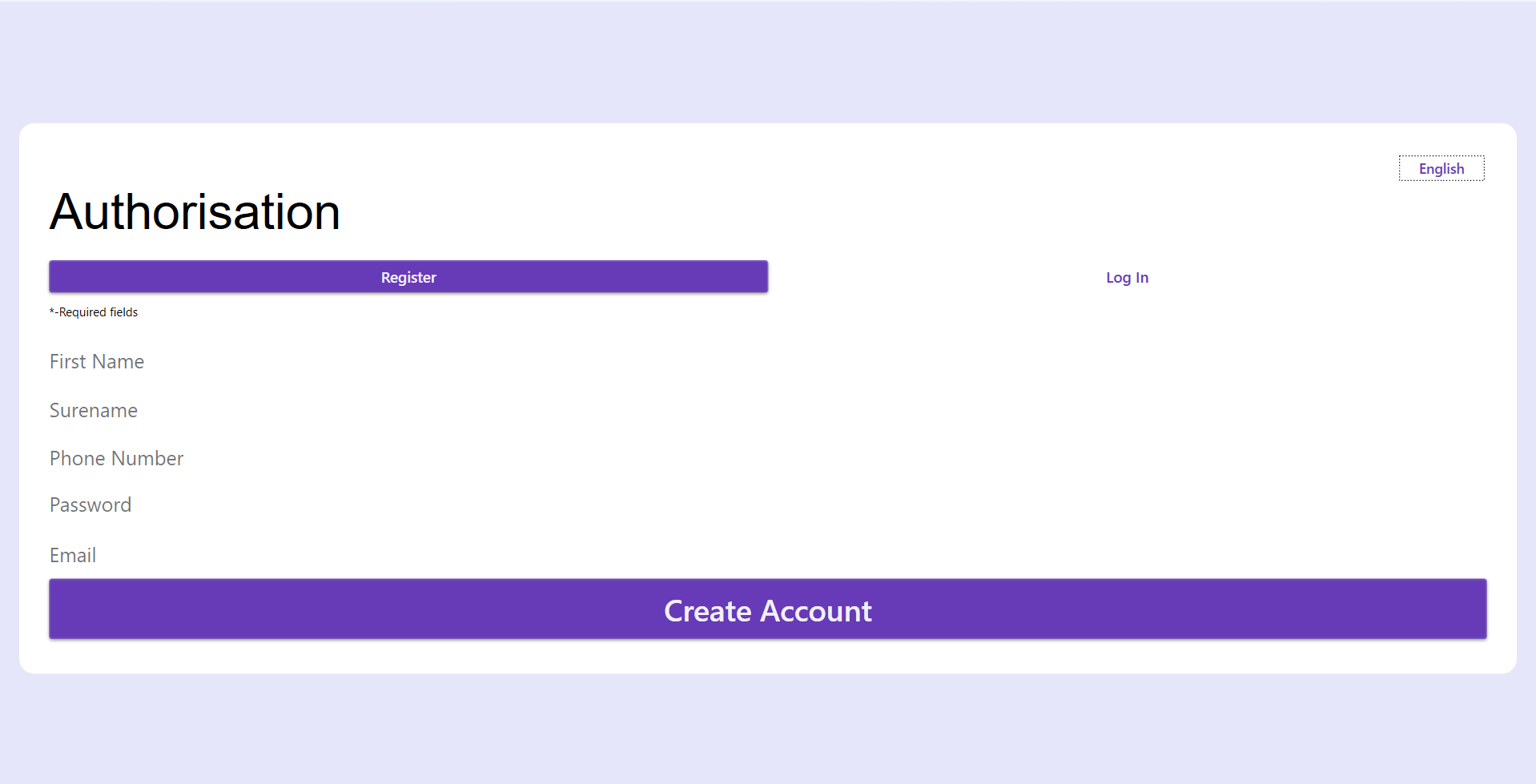


Рисунок 6.2 – Страница регистрации

# **6.2 Использование приложения в роли клиента**

После того, как пользователь вошел в аккаунт, он попадает на главную страницу приложения (на примере пользователя с ролью «клиент» которое изображено на рисунке 6.3).



Далее пользователь может перейти в свои заказы, корзину, каталог товаров, отзывы, а также при нажатии на кружочек профиля открыть форму профиля. Внизу меню панели есть кнопка для выхода назад на страницу авторизации.

# **ЗАКЛЮЧЕНИЕ**

Разработанное программное средство "Цветочный магазин" представляет собой значительный шаг вперед в сфере управления продажами и учета цветочной продукции. Данное приложение обеспечивает удобство, эффективность и простоту в организации процессов как для владельцев магазина, так и для клиентов.

Целью разработки являлось создание инновационного программного решения, удовлетворяющего современные потребности пользователей в сфере флористики. Основные принципы, заложенные в разработку, включают разнообразие функционала, а также интуитивно понятный интерфейс для легкого освоения пользователями. Это приложение может стать удобным инструментом в современной бизнес-среде, способствуя повышению эффективности работы профессионалов.

Одним из главных преимуществ программного средства является его простота использования. Дружественный пользовательский интерфейс и интуитивно понятные функции делают работу с приложением комфортной и без лишних сложностей. Пользователи смогут быстро освоиться с программой и использовать ее по назначению без необходимости дополнительного обучения.

В процессе реализации программного средства были достигнуты вышеуказанные требования и задачи. Применение паттерна MVVM обеспечило эффективное управление данными и логикой приложения, улучшенную архитектуру и обеспечило высокую отзывчивость и удобство использования пользовательского интерфейса.

Работа над проектом также позволила получить ценный опыт взаимодействия со сторонними библиотеками. Изучение и внедрение этих библиотек в проект позволило расширить его функциональность и улучшить его производительность. Разработанное программное средство "Цветочный магазин" является удобным и современным инструментом для управления цветочным бизнесом, способствующим его развитию и повышению качества обслуживания клиентов.

**Список использованных источников**

1. Онлайн-ресурс «posiflora.com» [Электронный ресурс] – Режим доступа: posiflora.com – Дата доступа: 15.03.2025
2. Онлайн-ресурс «florapos.ru» [Электронный ресурс] – Режим доступа: florapos.ru – Дата доступа: 16.03.2025
3. Онлайн-ресурс «florapoint.ru» [Электронный ресурс] – Режим доступа: florapoint – Дата доступа: 12.03.2025
4. Руководство по WPF // [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/wpf/ – Дата доступа: 20.03.2025
5. Работа с Entity Framework Core [Электронный ресурс] – Режим доступа: https://professorweb.ru/my/entity-framework/6/level1/ – Дата доступа 28.04.2025

**ПРИЛОЖЕНИЕ А**

Диаграмма вариантов использования



# **ПРИЛОЖЕНИЕ Б**

Код представления страницы авторизации

|  |
| --- |
| <Window x:Class="flower.View.LogInView"  xmlns:materialDesign="http://materialdesigninxaml.net/winfx/xaml/themes"  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  xmlns:local="clr-namespace:flower.View"  mc:Ignorable="d"  Title="{DynamicResource LogInSignUpTitle}" >  <Grid Background="Lavender">  <Border MinWidth="400" MinHeight="400" Margin="20" Padding="30" Background="White" VerticalAlignment="Center" CornerRadius="15">  <Grid>  <Grid.RowDefinitions>  <RowDefinition Height="0.5\*"></RowDefinition>  <RowDefinition Height="\*"></RowDefinition>  <RowDefinition Height="\*"></RowDefinition>  <RowDefinition Height="\*"></RowDefinition>  <RowDefinition Height="\*"></RowDefinition>  <RowDefinition Height="2\*"></RowDefinition>  <RowDefinition Height="\*"></RowDefinition>  </Grid.RowDefinitions>  <Button Grid.Row="0" x:Name="LanguageToggleButton" Content="English" Click="ChangeLangClick" Width="90" Height="30" HorizontalAlignment="Right" Style="{DynamicResource MaterialDesignFlatButton}"/>  <TextBlock Grid.Row="1" Text="{DynamicResource AuthorisationTitle}" Margin="0 0 0 5" FontSize="50" FontFamily="Arial" ></TextBlock>  <Grid Grid.Row="2" >  <Grid.ColumnDefinitions>  <ColumnDefinition Width="\*"/>  <ColumnDefinition Width="\*"/>  </Grid.ColumnDefinitions>  <Button x:Name="Register" Grid.Column="0" FontSize="15" Content="{DynamicResource RegisterButtonName}" Style="{StaticResource MaterialDesignFlatButton}" Click="Register\_Click"></Button>  <Button x:Name="Vhod" Grid.Column="1" FontSize="15" Content="{DynamicResource LogInButtonName}"></Button>  </Grid>  <TextBox Text="{Binding PhoneNumber, UpdateSourceTrigger=PropertyChanged}" x:Name="TBnumberAu" Grid.Row="3" materialDesign:HintAssist.Hint="{DynamicResource PhonrNumberTitle}" Margin="0 0 0 5" FontSize="20" Style="{DynamicResource MaterialDesignFloatingHintTextBox}" ></TextBox>  <PasswordBox PasswordChanged="PasswordBox\_PasswordChanged" x:Name="TBpassAu" Grid.Row="4" materialDesign:HintAssist.Hint="{DynamicResource PasswordTitle}" materialDesign:HintAssist.IsFloating="True" Margin="0 5 0 5" FontSize="20" ></PasswordBox>  <Button Grid.Row="5" Content="{DynamicResource EnterLoginButtonName}" Command="{Binding LogInCommand}" FontSize="30" MinHeight="70" Margin=" 0 10 0 10" ></Button>    </Grid>  </Border>  </Grid>  </Window> |

# **ПРИЛОЖЕНИЕ В**

Код модели представления страницы авторизации

|  |
| --- |
| using flower.Helpers;  using flower.View;  using Models;  using System.Collections.ObjectModel;  using System.ComponentModel;  using System.Diagnostics;  using System.Runtime.CompilerServices;  using System.Windows;  using System.Windows.Input;  namespace flower.ViewModel  {  public class LogInViewModel  {  private string \_phoneNumber;  private string \_password;  private User \_loggeduser = new User();  public User LoggedUser  {  get => \_loggeduser;  set  {  \_loggeduser = value;  }  }  private LogInView \_loginView;  public LogInViewModel(LogInView loginView)  {  \_loginView = loginView;  }  public event PropertyChangedEventHandler PropertyChanged;  protected void OnPropertyChanged([CallerMemberName] string propName = null)  => PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propName));  public string PhoneNumber  {  get => \_phoneNumber;  set  {  if (\_phoneNumber != value)  {  \_phoneNumber = value;  OnPropertyChanged(nameof(PhoneNumber));  }  }  }  public string Password  {  get => \_password;  set  {  if (\_password != value)  {  \_password = value;  OnPropertyChanged(nameof(Password));  }  }  }  private ICommand \_logIncommand;  public ICommand LogInCommand => \_logIncommand ??= new RelayCommand(Login, CanLogin);  private void Login(object parameter)  {    ObservableCollection<User> allUsers = App.repository.GetAllUsers();  bool found = false;  foreach(var user in allUsers)  {  if (user.Password==Password&&user.PhoneNumber==PhoneNumber)  {  found = true;  LoggedUser = user;  }  }  Password = string.Empty;  PhoneNumber = string.Empty;  if (!found)  {  MessageBox.Show("Such Account Does Not Exist!","Failed to Log In",MessageBoxButton.OK,MessageBoxImage.Error);  }  else  {      UserMainView userMain = new UserMainView(LoggedUser);  userMain.Show();  \_loginView.Close();  }    }    private bool CanLogin(object parameter)  {    return true;  }  }  } |

# **ПРИЛОЖЕНИЕ Г**

Код класса Context

|  |
| --- |
| using Microsoft.EntityFrameworkCore;  using Models;  namespace DAL  {  public class Context : DbContext  {  public DbSet<User> Users { get; set; }  public DbSet<Product> Products { get; set; }  public DbSet<Order> Orders { get; set; }  public DbSet<Review> Reviews { get; set; }  public DbSet<Category> Categories { get; set; }  public DbSet<Color> Colors { get; set; }  public DbSet<OrderDetails> OrderDetails { get; set; }  protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)  {  optionsBuilder.UseSqlServer("Server=DESKTOP-I\\SARVAR; Database=FlowerShop\_Db; Trusted\_Connection=true; TrustServerCertificate=Yes");  }  protected override void OnModelCreating(ModelBuilder modelBuilder)  {  // User - Order отношение  modelBuilder.Entity<Order>()  .HasOne(o => o.User)  .WithMany(u => u.Orders)  .HasForeignKey(o => o.UserId);  modelBuilder.Entity<Review>()  .HasOne(r => r.User)  .WithMany(u => u.Reviews)  .HasForeignKey(r => r.UserId).OnDelete(DeleteBehavior.Restrict);  modelBuilder.Entity<Review>()  .HasOne(r => r.Order)  .WithMany(r=>r.Reviews)  .HasForeignKey(r => r.OrderId).OnDelete(DeleteBehavior.Restrict);  modelBuilder.Entity<ProductCategory>()  .HasKey(pc => new { pc.ProductId, pc.CategoryId });  modelBuilder.Entity<ProductCategory>()  .HasOne(pc => pc.Product)  .WithMany(p => p.ProductCategories)  .HasForeignKey(pc => pc.ProductId);  modelBuilder.Entity<ProductCategory>()  .HasOne(pc => pc.Category)  .WithMany(c => c.ProductCategories)  .HasForeignKey(pc => pc.CategoryId);  modelBuilder.Entity<ProductColor>()  .HasKey(pc => new { pc.ProductId, pc.ColorId });  modelBuilder.Entity<ProductColor>()  .HasOne(pc => pc.Product)  .WithMany(p => p.ProductColors)  .HasForeignKey(pc => pc.ProductId);  modelBuilder.Entity<ProductColor>()  .HasOne(pc => pc.Color)  .WithMany(c => c.ProductColors)  .HasForeignKey(pc => pc.ColorId);  modelBuilder.Entity<OrderDetails>()  .HasOne(od => od.Order)  .WithMany(o => o.OrderDetails)  .HasForeignKey(od => od.OrderId);    modelBuilder.Entity<OrderDetails>()  .HasOne(od => od.Product)  .WithMany()  .HasForeignKey(od => od.ProductId);  }  }  } |

# **ПРИЛОЖЕНИЕ Е**

Код функции для добавления товара в каталог

|  |
| --- |
| public bool AddProduct(Product product)  {  if (product == null)  {  throw new ArgumentNullException(nameof(product), "Product cannot be null");  }  this.\_context.Products.Add(product);  int changes = this.\_context.SaveChanges();  return changes > 0;  } |

Код функции для удаления товара из корзины

|  |
| --- |
| private void RemoveProduct(Product product)  {  ObservableCollection<Product> previous = new ObservableCollection<Product>(Cart);  Cart.Clear();  TotalCart = 0;  previous.Remove(product);  foreach(var item in previous)  {  Cart.Add(item);  TotalCart+= item.Price;    }  } |

Код функции для регистрации пользователя

|  |
| --- |
| private void SignUp(object parameter)  {  ObservableCollection<User> allUsers = App.repository.GetAllUsers();  User newUser = new User(FirstName, Surename, Password, Email, PhoneNumber, false);  bool failed = false;  Debug.WriteLine($"{FirstName} - {Surename} - {Password} - {Email}");  newUser.Address = "NO";  newUser.ImagePath = "NO";  string MessageBoxMessage = "Something went wrong";  if(FirstName!=null && Surename!=null && Password != null && Email != null && PhoneNumber != null)  {  bool hasPassedChecks = true;  if (!Validator.IsValidName(FirstName) || !Validator.IsValidName(Surename))  {  hasPassedChecks = false;  MessageBoxMessage = "Invalid Name ore surename";  }  if(!Validator.IsValidEmail(Email))  {  hasPassedChecks = false;  MessageBoxMessage = "Invalid Email";  }  if (!Validator.IsValidPassword(Password))  {  hasPassedChecks = false;  MessageBoxMessage = "Invalid Password";  }  foreach(User user in allUsers)  {  if (user.PhoneNumber == newUser.PhoneNumber)  {  hasPassedChecks = false;  MessageBoxMessage = "This User Already exists";  }  }  if (hasPassedChecks)  {  if (App.repository.AddUser(newUser))  {    FirstName = string.Empty;  Surename = string.Empty;  Email = string.Empty;  Password = string.Empty;  PhoneNumber = string.Empty;  UserMainView userMain = new UserMainView(newUser);  userMain.Show();  \_signUpView.Close();  }  else  {  MessageBox.Show("Something went wrong", "Failed to sign up", MessageBoxButton.OK, MessageBoxImage.Error);  }  }  else  {  MessageBox.Show($"{MessageBoxMessage}","Fail",MessageBoxButton.OK,MessageBoxImage.Error);  }    }  else  {  MessageBox.Show("Failed to sign up","Fail",MessageBoxButton.OK, MessageBoxImage.Error);  }  } |