МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«Гродненский государственный университет имени Янки Купалы»

Факультет математики и информатики

Кафедра современных технологий программирования

МЕДВЕДЬ КИРИЛЛ МИХАЙЛОВИЧ

**Приложение «Интернет-магазин»**

Курсовой проект

по дисциплине «Конструирование программного обеспечения»

студента курса специальности

1-40 01 01 «Программное обеспечение информационных технологий» дневной формы получения образования

Научный руководитель

Банюкевич Елена Викторовна,

старший преподаватель кафедры современных

технологий программирования

Гродно, 2022

**РЕЗЮМЕ**

Медведь Кирилл Михайлович

Курсовой проект – приложение «Интернет-магазин», 29 страниц, 10 иллюстраций, 7 использованных источников.

Ключевые слова – C#, Microsoft Visual Studio, Windows Forms, ООП, финансы, магазин, покупки, расходы, регистрация аккаунта.

Цель исследования – разработка приложения для регистрации покупателей, выбора товаров для покупки и оформления заказов. Приложение позволяет выполнять следующие операции: зарегистрировать и войти в аккаунт; возможность входа в приложение в качестве администратора; в качестве пользователя пополнить свой баланс, посмотреть все свои заказы, добавить товары в корзину и оформить заказ; в качестве администратора изменить статус заказа и назначить работника к заказу.

Объект исследования – взаимодействие пользователя с приложением «Интернет-магазин».

Предмет исследования – реализация приложения «Интернет магазин» с помощью интерфейса программирования приложений Windows Form и языка программирования C#, с использованием Microsoft Visual Studio, встроенных библиотек System.

Методы исследования – анализ и формализация требований, системный анализ, анализ литературных источников, сравнительный анализ аналогичных решений, объектно-ориентированный анализ и проектирование.

**Авторская характеристика работы.**

Данное приложение разработано на языке программирования высокого уровня C# с целью предоставления возможности оформления заказов и обработки их статуса. В нём реализована работа с графическим интерфейсом при помощи Windows Form. Приложение предоставляет различные функции для редактирования заказов и просмотра информации о них.

**SUMMARY**

Medved Kirill Mikhailovich

The course work - “Online-store” application, 29 pages, 10 illustrations, 7 sources used.

Keywords - C #, Microsoft Visual Studio, Windows Forms, OOP, finance, shop, shopping, expenses, account registration.

The purpose of the study is the development of an application for registering buyers, selecting goods for purchase and placing orders. The application allows you to perform the following operations: register and log into user account; the ability to log into the application as an administrator; as a user, replenish your balance, view all your orders, add products to the basket and place an order; as an administrator, change the order status and assign a worker to the order.

The object of research is the user interaction with the application "Online-store".

The subject of the research is the implementation of the "Online-store" application using the Windows Form and the C# programming language, using Microsoft Visual Studio, the built-in System libraries.

Research methods are the study and analysis of relevant literature, information on the Internet, a systematic approach, study and analysis of existing analogues.

**The author’s description of the work.**

This application was developed in the C# high-level programming language in order to provide the possibility of placing orders and processing their status. It implements work with a graphical interface using Windows Form. The application provides various functions for editing orders and viewing information about them.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc90833278)

[Глава 1 ОБЗОР ПРОГРАММНЫХ РЕШЕНИЙ ДЛЯ УЧЕТА ФИНАНСОВ 7](#_Toc90833279)

[1.1 Описание приложения 7](#_Toc90833280)

[1.2 Обзор существующих решений 7](#_Toc90833281)

[1.2.1 Приложение «Monefy» 7](#_Toc90833282)

[1.2.2 Приложение «Money Lover» 9](#_Toc90833283)

[1.2.3 Приложение «CoinKeeper» 10](#_Toc90833284)

[1.3 Выводы по главе 1 11](#_Toc90833285)

[Глава 2 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ «МЕНЕДЖЕР ФИНАНСОВ» 12](#_Toc90833286)

[2.1 Проектирование функционала приложения 12](#_Toc90833287)

[2.2 Архитектура приложения 13](#_Toc90833288)

[2.3 Проектирование графического интерфейса 15](#_Toc90833289)

[2.4 Выводы по главе 2 17](#_Toc90833290)

Глава 3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ «МЕНЕДЖЕР ФИНАНСОВ» 19

3.1 Обзор средств технологий реализации 19

3.2 Используемые сторонние библиотеки 19

3.3 Структура проекта 20

3.4 Описание моделей программы 20

[3.5 Описание работы с базой данных 22](#_Toc90832423)

[3.6 Описание моделей представления 23](#_Toc90833292)

[3.7 Описание представлений 24](#_Toc90833293)

[3.8 Выводы по третьей главе 27](#_Toc90833294)

[ЗАКЛЮЧЕНИЕ 29](#_Toc90833295)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 30](#_Toc90833296)

# ВВЕДЕНИЕ

Всем нам приходится посещать магазин для покупки различных товаров, но это очень быстро превращается в рутину, занимающую большое количество времени. Значит очень удобно было бы, чтоб от этой рутины успешно избавиться. Отличным решением такой проблемы являются интернет-магазины. Они существенно экономят время, помогая ознакомиться с ассортиментом, узнать цены продуктов, их наличие в магазине, прочитать отзывы от других покупателей, а также подсчитать, сколько будет затрачено денег на покупку. Кроме того, за отдельную плату, а чаще всего бесплатно, можно заказать доставку на дом, где курьер доставит ваши покупки и вам останется лишь распаковать их дома. Также интернет-магазины очень удобны и для их владельцев, ведь их расходы уменьшаются засчет отсутствия арендной платы за помещения для размещения магазина.

Интернет магазины очень удобны для каталогизации всех проведенных покупок пользователями. Можно отследить каждую покупку и узнать во сколько она была проведена, какие товара, их цену и количество, кто сделал заказ, откуда и как часто он эти заказы делает. Это сильно помогает проводить учет доходов и расходов, а также обрабатывать информацию о постоянных покупателях, предлагая им выгодные предложения.

Таким образом, целью исследования было выбрано создание приложения, которое отвечает всем современным потребностям в обработки заказов. Предусматривается выполнение приложением основных функций, присущих подобным приложениям: регистрация и вход в аккаунт; возможность входа в приложение в качестве администратора; в качестве пользователя пополнить свой баланс, посмотреть все свои заказы, добавить товары в корзину и оформить заказ; в качестве администратора изменить статус заказа и назначить работника к заказу и т.д.

# ГЛАВА 1 ОБЗОР ПРОГРАММНЫХ РЕШЕНИЙ ДЛЯ ОБРАБОТКИ ЗАКАЗОВ

## Описание приложения

Приложение «Интернет-магазин» совмещать в себе одновременно и интуитивно понятный интерфейс, и вместе с этим достаточный функционал. Оно должно взаимодействовать с разнообразными пользовательскими данными: операции снятие денег с баланса, а также его пополнение, отображение информации о заказах. Задача качественного интернет-магазина – это помочь человеку быстро и понятно оформить заказ на товары, которые ему нужны. Большинство современных приложений такого типа имеют:

* Понятный интерфейс
* Наглядное отображение (картинки, схемы и т.д.)
* Информация о товарах (цена, название и т.д.)

## Обзор существующих решений

Существует огромное количество различных интернет-магазинов. Они могут быть как мобильные, так и представлять себя как веб-приложение. Рассмотрим некоторые из них.

## Веб-приложение «emall.by»

У «emall.by» большой список категорий товаров. Все товары упорядочены по категориям и по подкатегориям;

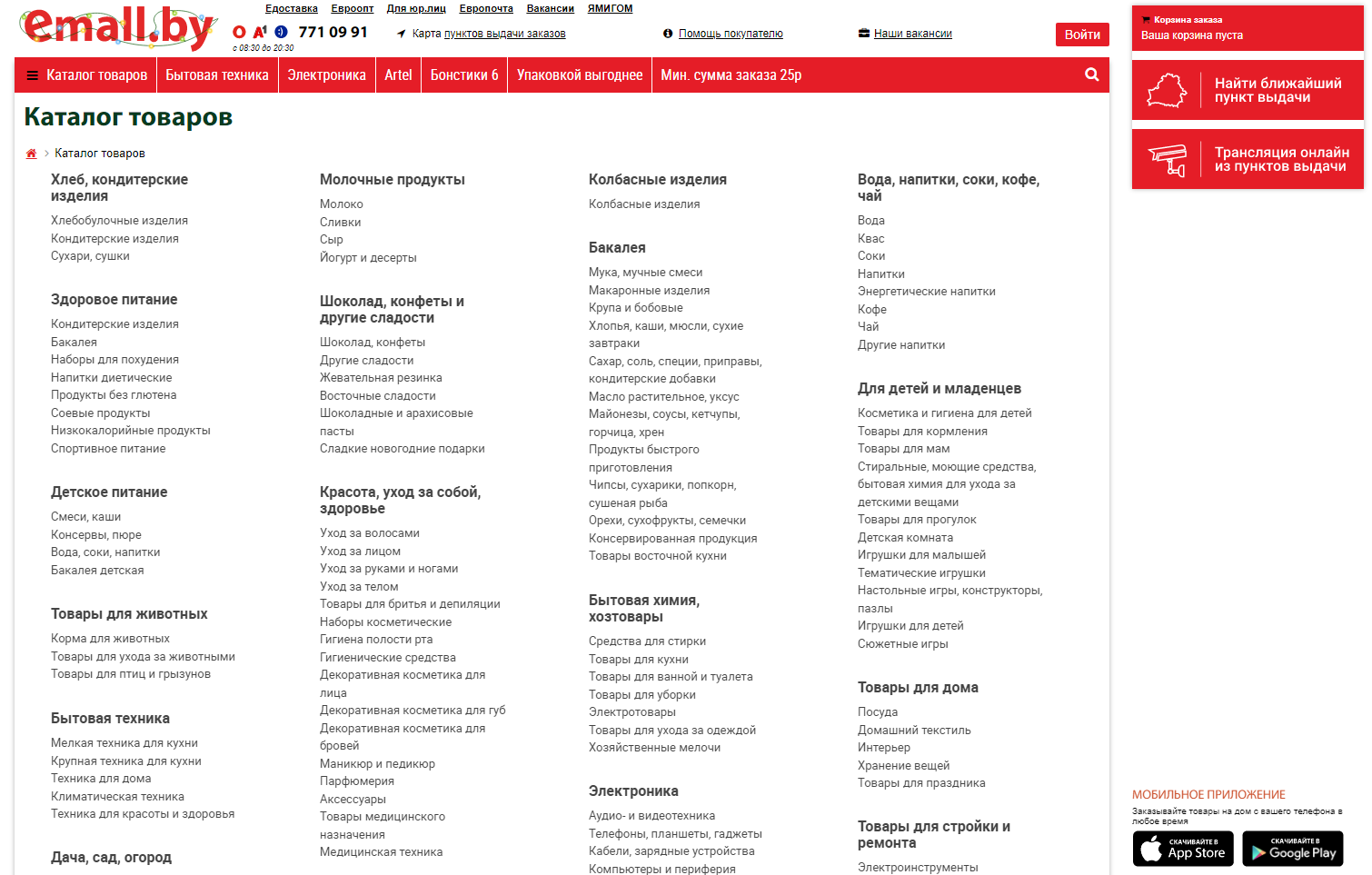


Рисунок 1.2.2 Веб-приложение «emall.by»

Плюсы:

* Большое количество товаров.
* Все товары разбиты на категории и подкатегории.
* Удобная корзина.

Минусы:

* Непривлекательный дизайн.
* Неочевидные переходы по веб-приложению.

## Веб-приложение «Wildberries»

В приложении есть множество товаров разных категорий.

Красивый дизайн, с удобным интерфейсом.

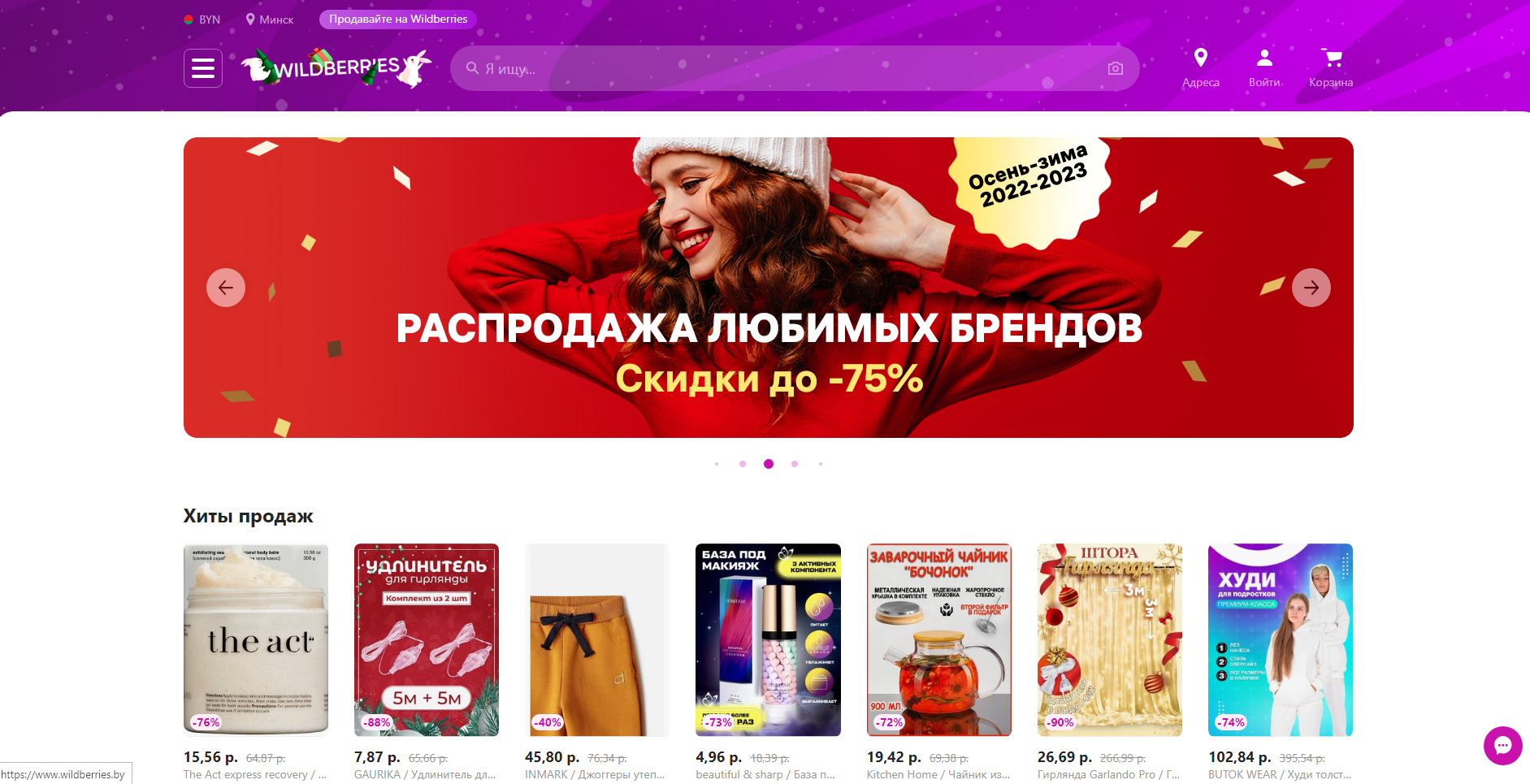


Рисунок 1.2.3 Приложение «Money Lover»

Плюсы:

* Красивый дизайн.
* Удобный интерфейс.

Минусы:

* Недружелюбный интерфейс корзины.
* Отсутствие доставки для некоторых товаров.

## Выводы по главе 1

В данной главе была рассмотрена предметная область приложения “Интернет-магазин”. Также были рассмотрены существующие приложения, кратко описан их функционал возможности.

На основе всего этого можно сделать следующие выводы: разрабатываемое приложение должно взаимодействовать с пользователем и его действиями, данными о заказах; иметь возможность редактировать статус заказов; предоставлять возможность пользователю сформировать корзину с покупками и оформить заказ; работать с регистрацией аккаунтов и входом в них.

# ГЛАВА 2 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ «ИНТЕРНЕТ-МАГАЗИН»

# Проектирование функционала приложения

Опираясь на анализ существующих приложений «Интернет магазин», а также учитывая потребности пользователя можно выделить следующий функционал, который необходимо реализовать:

* Возможность регистрации и входа в аккаунт;
* Пополнение и снятие средств с баланса пользователя;
* Просмотр всех своих заказов пользователем;
* Возможность наполнения корзины пользователем;
* Валидация вводимых пользователем данных;
* Возможность оформления заказа пользователем;
* Возможность войти в программу в качестве администратора;
* Возможность будучи администратором изменять статус заказа и закреплять работников за ним.

Теперь, после выделения базового функционала, необходимого для полной работы приложения «Интернет магазин», можно перейти к проектированию архитектуры.

# Архитектура приложения

Приложению для полной и, что главное, корректной работы необходим набор определённых связанных сущностей, а именно счета, категории и денежные транзакции. К тому же необходим удобный, интуитивно понятный интерфейс, по средствам которого пользователь будет иметь возможность взаимодействия с приложение. И, конечно же, нам обязательно нужно связать пользовательский интерфейс с сущностями.

В этом нам поможет паттерн MVVM.

Model-View-ViewModel (MVVM) — шаблон проектирования архитектуры приложения. Представлен в 2005 году Джоном Госсманом (John Gossman) как модификация шаблона Presentation Model. Ориентирован на современные платформы разработки, такие как Windows Presentation Foundation, Silverlight от компании Microsoft, ZK framework. [[2](#_https://ru.wikipedia.org/wiki/Model)]

Суть паттерна MVVM заключается в модульной разработке приложения. Графическая часть (View) и логика приложения (Model), это независящие друг от друга части приложения, но их работа связывается через модель представления (ViewModel).

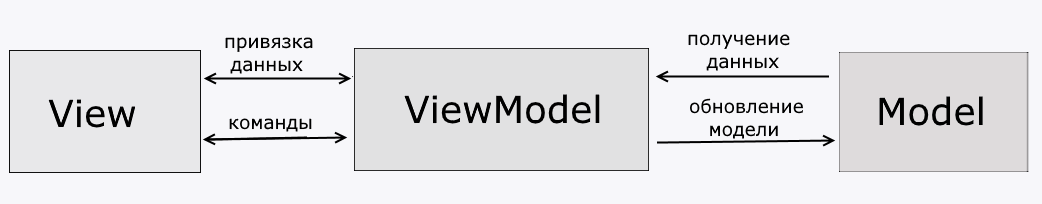


Рисунок 2.2.1 Паттерн MVVM

Модель (Model) описывает используемые в приложении данные. Модели могут содержать логику, непосредственно связанную этими данными, например, логику валидации свойств модели. В то же время модель не должна содержать никакой логики, связанной с отображением данных и взаимодействием с визуальными элементами управления.

Нередко модель реализует интерфейсы INotifyPropertyChanged или INotifyCollectionChanged, которые позволяют уведомлять систему об изменениях свойств модели. Благодаря этому облегчается привязка к представлению, хотя опять же прямое взаимодействие между моделью и представлением отсутствует.

View или представление определяет визуальный интерфейс, через который пользователь взаимодействует с приложением. Применительно к WPF представление - это код в xaml, который определяет интерфейс в виде кнопок, текстовых полей и прочих визуальных элементов.

Хотя окно (класс Window) в WPF может содержать как интерфейс в xaml, так и привязанный к нему код C#, однако в идеале код C# не должен содержать какой-то логики, кроме разве что конструктора, который вызывает метод InitializeComponent и выполняет начальную инициализацию окна. Вся же основная логика приложения выносится в компонент ViewModel.

Однако иногда в файле связанного кода все может находиться некоторая логика, которую трудно реализовать в рамках паттерна MVVM во ViewModel.

Представление не обрабатывает события за редким исключением, а выполняет действия в основном посредством команд.

ViewModel или модель представления связывает модель и представление через механизм привязки данных. Если в модели изменяются значения свойств, при реализации моделью интерфейса INotifyPropertyChanged автоматически идет изменение отображаемых данных в представлении, хотя напрямую модель и представление не связаны.

ViewModel также содержит логику по получению данных из модели, которые потом передаются в представление. И также VewModel определяет логику по обновлению данных в модели.

Поскольку элементы представления, то есть визуальные компоненты типа кнопок, не используют события, то представление взаимодействует с ViewModel посредством команд.

Например, пользователь хочет сохранить введенные в текстовое поле данные. Он нажимает на кнопку и тем самым отправляет команду во ViewModel. А ViewModel уже получает переданные данные и в соответствии с ними обновляет модель.

Итогом применения паттерна MVVM является функциональное разделение приложения на три компонента, которые проще разрабатывать и тестировать, а также в дальнейшем модифицировать и поддерживать. [[3](#_https://metanit.com/sharp/wpf/22.1.)]

Также возник вопрос, о том, где хранит пользовательскую информацию. Для хранения используется локальная база данных SQL, организация, контроль и администрирование которой осуществляется при помощи SQLite.

SQLite - это встроенный механизм базы данных SQL. В отличие от большинства других баз данных SQL, SQLite не имеет отдельного серверного процесса. SQLite читает и записывает непосредственно в обычные файлы на диске. Полная база данных SQL с несколькими таблицами, индексами, триггерами и представлениями содержится в одном файле на диске. (преведено с английского) [[4](#_https://www.sqlite.org/about.html)]

# Проектирование графического интерфейса

Одним из требований к хорошему графическому интерфейсу программной системы является концепция «делай то, что я имею в виду» или DWIM (англ. Do What I Mean). Согласно концепции требуется, чтобы система работала предсказуемо, чтобы пользователь заранее интуитивно понимал, какое действие выполнит программа после получения его команды. [[5](#_Графический_интерфейс_пользователя)]

К примеру, пользовательский интерфейс может содержать меню выбора (рис. 2.3.1)

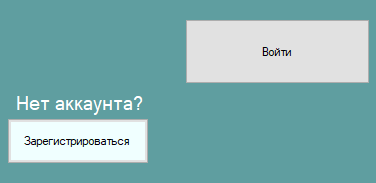


Рисунок 2.3.1 Пример меню выбора

Все кнопки интерфейса имеют надписи и (или) значки, из которых становиться понятно о их назначении (рис. 2.3.2)

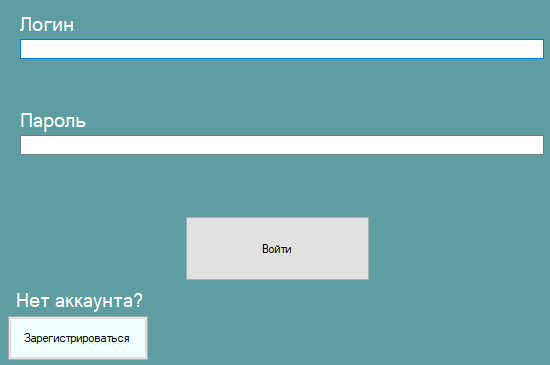


Рисунок 2.3.2 Пример кнопок

# Выводы по главе 2

В данной главе были установлены основные требования к функционалу приложения «Интернет магазин». Была установлена архитектура приложения, которая будет состоять из моделей, представлений и связывающих их моделей представления. В следствие чего был выбран паттерн программирования MVVM для реализации установленной архитектуры приложения. Также был выбран вариант реализации базы данных в виде хранения информации в .txt файлах. Также были установлены, отрисованы и функционально описаны основные разделы графического интерфейса, которые необходимо реализовать в будущем приложении.

**ГЛАВА 3  
ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ «ИНТЕРНЕТ-МАГАЗИН»**

**3.1 Обзор используемых средств**

Приложение реализовано на языке высокого уровня C#. В ходе реализации проекта, были использованы многие преимущества работы с объектно-ориентированным языком программирования, такие как полиморфизм, наследования классов и инкапсуляция, события и их обработка, и многое другое. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в то же время гибких, масштабируемых и расширяемых приложений. Для реализации графического интерфейса используется Windows Forms.

**3.2 Используемые сторонние библиотеки**

В проекте были использованные следующие сторонние библиотеки:

* Syncfusion. Библиотека использовалась для построения графиков, а также получения доступа к дополнительным, не имеющимся в стандартном наборе, элементам управления, таким как SfDatePicer;
* NIOP. Библиотека имеющая набор необходимых инструментов для работы с для чтения и записи файлов в форматах Microsoft Office, таких как Word, PowerPoint и Excel.
* Microsoft Entity Framework Core. Entity Framework Core (EF Core) представляет собой объектно-ориентированную, легковесную и расширяемую технологию от компании Microsoft для доступа к данным. EF Core позволяет абстрагироваться от самой базы данных и ее таблиц и работать с данными независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает Entity Framework, мы уже работаем с объектами.

**3.3 Структура проекта**

Проект структурно разделен на несколько частей, представляющих отдельные модули и ресурсы приложения:

1. files – папка, содержащая все .txt файлы, в которых хранятся данные, используемые программой;
2. img – папка, хранящая в себе картинки продаваемых продуктов;

**3.4 Описание моделей программы**

Основные модели программы это:

* adminPanel.cs - объект класса Form, который содержит операции для работы администратора:
* Form3\_Load() – метод, отвечающий за загрузку данных о всех заказах из файла ordersInProgress.txt в таблицу, находящуюся в форме, начинает свою работу при загрузке формы;
* assignOrder\_Click() – метод, отвечающий за запись информации об выбранном заказе в файл;
* OnFormClosed() – метод, отвечающий за корректное закрытие программы;
* chooseWorker\_Click() – метод, отвечающий за назначение работника к выбранному заказу;
* chooseStatus\_Click() – метод, отвечающий за изменение статуса выполнения выбранного заказа;
* exit\_Click() – метод, отвечающий за корректный выход из программы;
* LoginForm.cs - объект класса Form, который отвечает за вход в аккаунт:
* toRegistration\_Click () – метод, отвечающий за переход на форму регистрации аккаунта;
* toMain\_Click() – метод, отвечающий за переход на страницу профиля пользователя, в случае успешного входа;
* OnFormClosed() – метод, отвечающий за корректное закрытие программы;
* LoginForm\_Load() – метод, отвечающий за очищение файла basket.txt.
* main.cs - объект класса Form, который отвечает за отображение товаров для покупки и добавления этих товаров в корзину:
* string selectedCategory – название выбранной категории продуктов;
* Products product – объект, описывающий выбранный товар;
* bool isGood – выбран ли товар;
* listOfGoods\_SelectedIndexChanged() – метод, отвечающий за отображение информации о выбранном товаре в выпадающем списке.
* OnFormClosed() – метод, отвечающий за корректное закрытие программы;
* личныйКабинетToolStripMenuItem\_Click() - метод, отвечающий за переход в личный кабинет пользователя;
* выходToolStripMenuItem\_Click() - метод, отвечающий за корректное закрытие программы;
* chandeListGoods() - метод, отвечающий за обновление в выпадающем списке доступных товаров;
* listView1\_SelectedIndexChanged() - метод, отвечающий за выбор категории товаров;
* addBusket\_Click() - метод, отвечающий за добавление товара в корзину;
* basket\_Click() - метод, отвечающий за запись добавленных в корзину товаров в basket.txt;
* MainPage\_Load() - метод, отвечающий за добавление категорий товаров, вызывается метод при загруке формы
* addCategoryGood() - метод, отвечающий за добавление категории товаров из goodsList.txt;
* ordersForm.cs - объект класса Form, который отвечает за отображение корзины, возможность оформить заказ:
* OnFormClosed() – метод, отвечающий за корректное закрытие программы;
* toMain\_Click() - метод, отвечающий за переход к форме покупки товаров;
* ordersForm\_Load() - метод, отвечающий за загрузку информации об добавленных в корзину товаров, вызывается метод при загрузке формы;
* createId() - метод, отвечающий за генерацию id заказа;
* order\_Click() - метод, отвечающий за запись заказа в ordersInProgress.txt;
* checkBalance() - метод, отвечающий за проверку баланса пользователя, хватает ли средств для оформления заказа;
* upgateCountGood\_Click() - метод, отвечающий за изменение количества покупаемого товара;
* updatePrice() – метод, отвечающий за поlсчет общей цены заказа;
* countPrice\_Click() - метод, отвечающий за вывод цены заказа в форме;
* products.cs - класс, описывающий свойства товара:
* string pictureProduct – путь до картинки товара;
* string categoryProduct – название категории товара;
* string nameProduct – название товара;
* string[] typesProducts – список продуктов соответствующей категории;
* int price – цена товара;
* List<string> productsBusket – список продуктов, добавленных в корзину;
* selecteCategotyProduct() - метод, отвечающий за выбор списка продуктов выбранной категории и пути к картинке товаров этой категории;
* selectProduct() - метод, отвечающий за установление цены продуктам;
* addToBusket(int resultPrice) - метод, отвечающий за добавление продукта в корзину и принимающий на вход цену продукта;
* goodIsChoose() – метод, проверяющий добавлен ли продукт в корзину;
* Profile.cs - объект класса Form, который отвечает за отображение профиля пользователя:
* buttonAddBalance\_Click() – метод, отвечающий за пополнение баланса пользователя, а также проводит верификацию вводимых данных;
* StringIsDigits(string s) - метод, отвечающий за проверку, является ли строка числом, принимает на вход проверяемую строку;
* buttonToLogin\_Click() - метод, отвечающий за переход на форму авторизации;
* Profile\_Load() - метод, отвечающий за добавление логина текущего пользователя в localStorage.txt, а также загрузка данные о количестве средств на балансе пользователя из usersBalance.txt, вызывается при загрузке формы;
* buttonToBuy\_Click() - метод, отвечающий за переход на форму покупки товаров;
* OnFormClosed() – метод, отвечающий за корректное закрытие программы;
* userOrders\_Click() - метод, отвечающий за переход на форму просмотра списка заказов пользователя;
* RegistrationForm.cs - объект класса Form, который отвечает за регистрацию аккаунта:
* User user – объект класса User, который описывает свойства пользователя;
* toLogin\_Click() - метод, отвечающий за регистрацию аккаунта, вывод информации в случае провала верификации, в случае успеха – переход на форму авторизации;
* OnFormClosed() – метод, отвечающий за корректное закрытие программы;
* toLog\_Click() - метод, отвечающий за переход на форму авторизации без регистрации аккаунта;
* User.cs - класс, описывающий свойства пользователя:
* validationRegistration(string loginUser, string passwordUser) - метод, отвечающий за верификацию логина и пароля аккаунта, который хотят зарегистрировать, на вход принимает логин и пароль соответственно;
* usernameIsExists(string newUser) - метод, проверяющий наличие аккаунта с таким логином, на вход принимает логин аккаунта, который хотят зарегистрировать;
* isLow(string s) - метод, проверяющий, что передаваемая строка на вход находится в нижнем регистре;
* registrationNewUser(string loginUser, string passwordUser) - метод, отвечающий за регистрацию нового аккаунта, запись логина и пароля в users.txt, принимает на вход логин и пароль соответственно.
* isAdmin(string loginUser, string passwordUser) - метод, проверяющий, что пользователь является администратором, принимает на вход логин и пароль соответственно;
* accountLogin(string loginUser, string passwordUser) - метод, проверяющий, что аккаунт с передаваемыми на вход логином и паролем существует;
* addToLocalStorage(string inputLogin) – метод, записывающий логин пользователя, прошедшего авторизацию в localStorage.txt;
* userOrders.cs - объект класса Form, который отвечает за отображение информации о всех заказах пользователя:
* userOrders\_Load() - метод, отвечающий за загрузку данных об заказах пользователя из ordersInProgress.txt;
* toProfile\_Click() - метод, отвечающий за переход в форму профиля пользователя;

Все классы отвечающие за отображение интерфейса приложения наследуются от общего базового класса Form, необходимого для своевременного изменения информации, отображаемой графическим интерфейсом.

# 3.5 Описание работы с .txt файлами

Вся информация в .txt файлах хранится построчно, что позволяет при чтении информации из них, связывать нужную информацию между собой.

Пример имеющегося метода, работающего с .txt файлами.

Пример 1. Загрузка информации об заказах пользователя

private void userOrders\_Load(object sender, EventArgs e)

{

string[] usersOrders = File.ReadAllLines(Directory.GetCurrentDirectory() + @"\files\ordersInProgress.txt");

string[] currentUser = File.ReadAllLines(Directory.GetCurrentDirectory() + @"\files\localStorage.txt");

for (int i = 0; i < usersOrders.Length; i++)

{

usersOrders[i] = usersOrders[i].Remove(usersOrders[i].Length - 1, 1);

string[] userOder = usersOrders[i].Split('/');

if (userOder[1] == currentUser[0])

{

string idOrder = userOder[0];

string userName = userOder[1];

string status = userOder[3];

int priceOrder = 0;

for (int j = 4; j < userOder.Length; j++)

{

string[] order = userOder[j].Split('\_');

priceOrder += Convert.ToInt32(order[1]);

}

string[] row = { idOrder, userName, priceOrder.ToString(), status };

var listViewItem = new ListViewItem(row);

userOrdersList.Items.Add(listViewItem);

}

}

}

# 3.6 Описание моделей представления

Как уже было сказано выше, все модели представления наследуют базовый класс Form.

Принципы работы всех моделей представления очень схожи, поэтому не имеет особого смысла описывать в деталях каждую из них. Далее следует описание основных функций, для которых используются модели представлений:

* AccountEditViewModel.cs – управляет работой по редактированию информации о выбранном счёте;
* AccountsViewModel.cs – модель представления, предназначенная для связывания вкладки приложения “Accounts” с логикой программы;
* AccountViewModel.cs – базовая модель представления, связывающая модель Account с любой другой моделью представления или представлением;
* CategoriesViewModel.cs - связана с отображением имеющихся категорий;
* CategoryEditViewModel.cs - управляет работой по редактированию информации о выбранной категории;
* CategoryViewModel.cs - – базовая модель представления, связывающая модель Category с любой другой моделью представления или представлением;
* HomeViewModel.cs – модель представления, предназначенная для связывания вкладки приложения “Home” с логикой программы;
* InformationViewModel.cs - модель представления, предназначенная для связывания вкладки приложения “Information” с логикой программы;
* MoneyChangeEditViewModel.cs - управляет работой по редактированию информации о выбранной денежной операции;
* MoneyChangeViewModel.cs - базовая модель представления, связывающая модель MoneyChange с любой другой моделью представления или представлением;
* MainViewModel.cs – главная модель представлений, отвечающая за работу всего приложения в целом.

# 3.7 Описание представлений

Большая часть представлений работает по схожим принципам, поэтому достаточно разобрать в работе одно представление. Как пример, рассмотрим MoneyChangeEditView.xaml. По своей сути, все представления являются пользовательским элементами управления (за исключением MainWindow). Наполнять содержимым представления можно как через код xaml, так и напрямую через код C# (рис. 3.7.1, рис 3.7.2, рис 3.7.3).

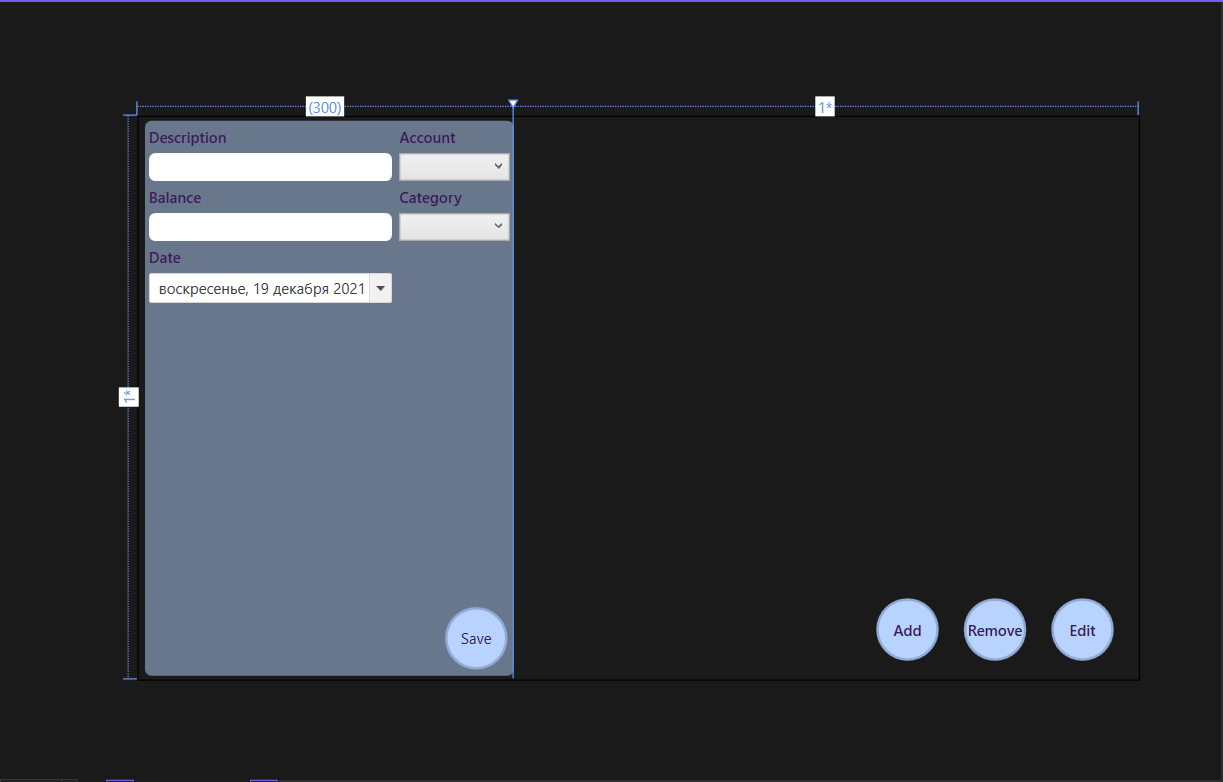


Рисунок 3.7.1 MoneyChangeEditView.xaml вид в конструкторе xaml

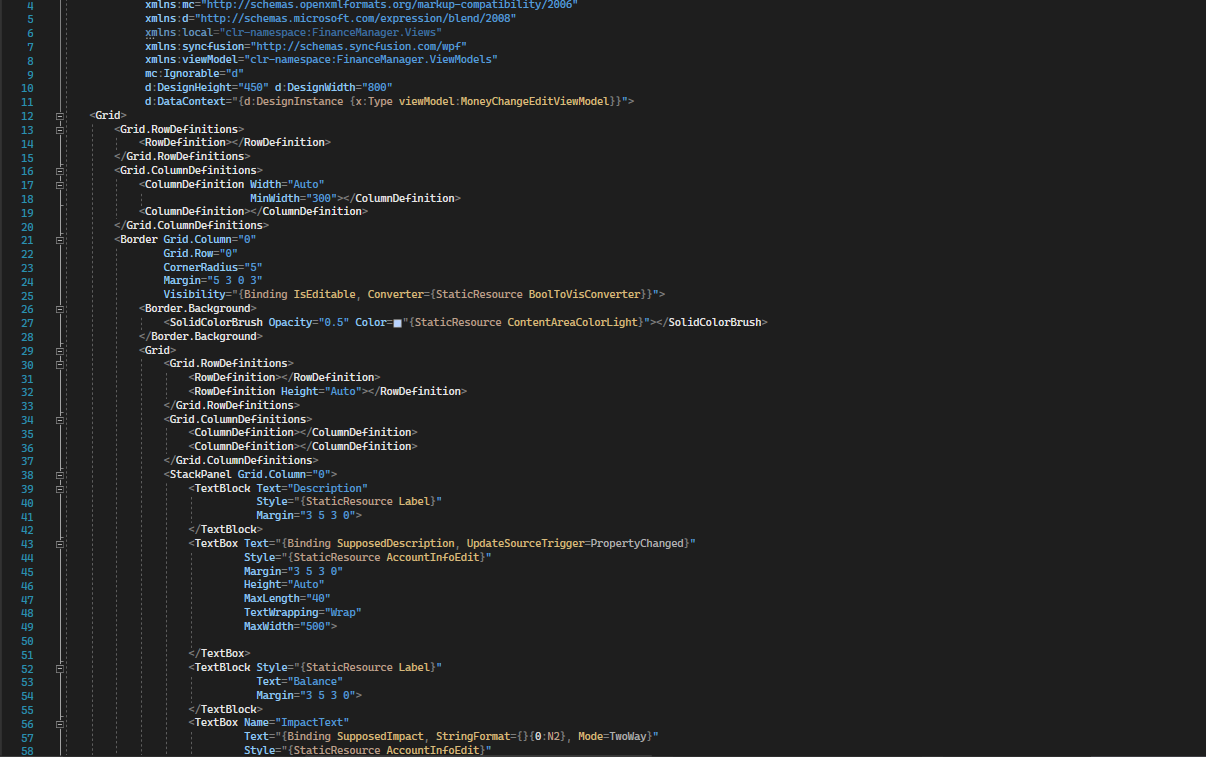


Рисунок 3.7.2 MoneyChangeEditView.xamal вид в редакторе кода xaml

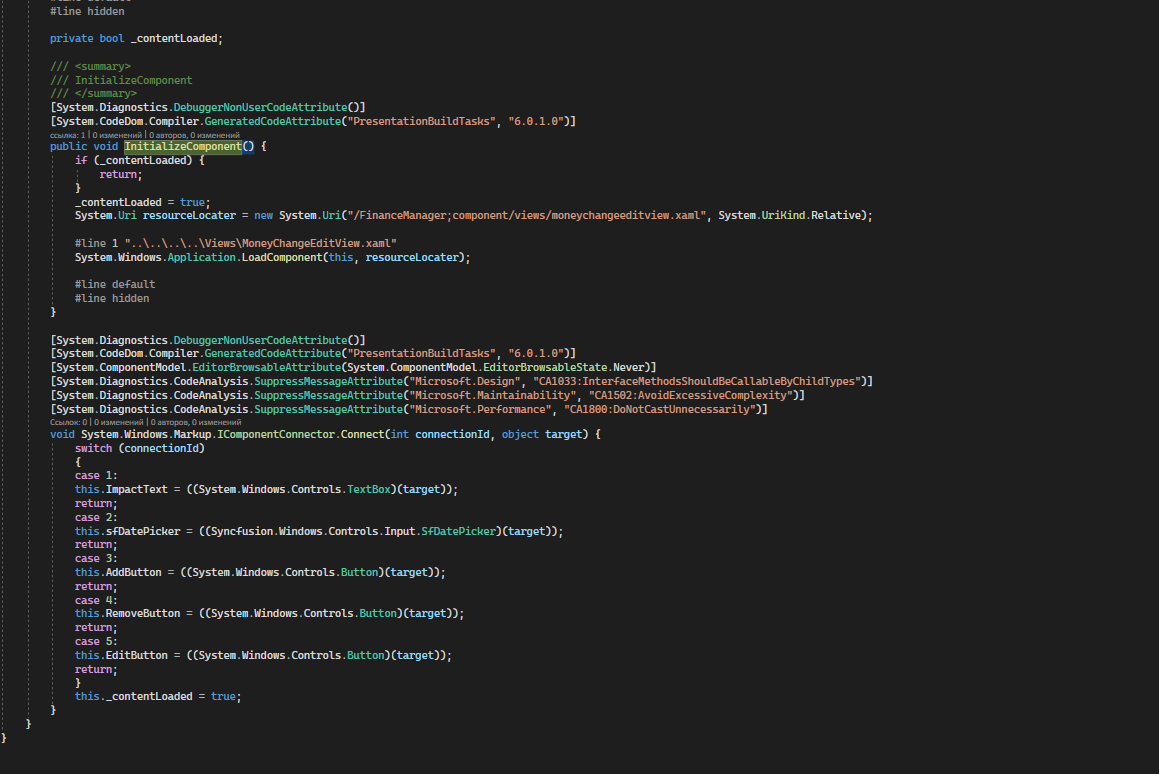


Рисунок 3.7.3 Автоматически созданный код C# для MoneyChangeEditView.xaml

В качестве примера настройки элемента управления рассмотрим ComboBox, отвечающий за редактирование счёта:

<ComboBox ItemsSource="{Binding AvailibleAccounts}"

Margin="3 5 3 0"

SelectedItem="{Binding SupposedAccount, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}">

<ComboBox.ItemTemplate>

<DataTemplate>

<TextBlock Style="{StaticResource Label}"

HorizontalAlignment="Left"

FontSize="12"

Text="{Binding Name}">

</TextBlock>

</DataTemplate>

</ComboBox.ItemTemplate>

</ComboBox>

Свойство ItemsSource связывается со свойством AvailibleAccounts объекта класса MoneyChangeEditViewModel.cs, это означает, что при изменении коллекции AvailibleAccounts, элементы, отображаемые графическим интерфейсом в ComboBox также изменятся. Аналогично и со свойством SelectedItem, если пользователь выберет какой-либо элемент из ComboBox, свойство SupposedAccount объекта класса MoneyChangeEditViewModel.cs также поменяется. Mode же отвечает за то, каким образом свойства связаны (зависят друг от друга, или только одно от другого, либо вообще получить информацию из свойства источника только при загрузке элемента управления). UpdateSourceTrigger отвечает за то, в каком случае должно произойти изменение того или иного свойства. DataTemplate определяет то, как именно будут выглядеть элементы, переданные в ComboBox из источника.

# 3.8 Выводы по третьей главе

Таким образом мы разобрались с функционалом программы и ее реализацией. В данной главе была описана структура проекта. Были рассмотрены основные примеры реализации различных функций приложения, а также разобраны основные функции используемых моделей представлений.

**ЗАКЛЮЧЕНИЕ**

Главной целью проекта была разработка приложения, содержащего базовый, минимальный функционал для реализации интернет-магазина с возможностью регистрации и авторизации аккаунта; пополнение и снятие средств с баланса пользователя; просмотр информации о заказах пользователя; добавления продуктов в корзину и последующего оформления заказа; авторизация в приложении в качестве администратора, редактирование статуса заказа и закрепление рабочего за ним.

В результате работы над курсовым проектом было разработано приложение «Интернет-магазин», которое удовлетворяет всем поставленным требованиям.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Rob Miles C# Programming Yellow Book. – Department of Computer Science University of Hull, 2014.
2. Мэтью Мак-Дональд - WPF: Windows Presentation Foundation в .NET 4.0 с примерами на C#, Вильямс, 2011.

### [Электронный ресурс] Режим доступа - <https://daily.afisha.ru/money/20691-9-luchshih-prilozheniy-dlya-ucheta-finansov/> - Дата доступа: 19.12.2021

### [Электронный ресурс] Режим доступа - <https://ru.wikipedia.org/wiki/Model-View-ViewModel> - Дата доступа: 15.12.2021

### [Электронный ресурс] Режим доступа - <https://metanit.com/sharp/wpf/22.1.php> - Дата доступа: 12.12.2021

### [Электронный ресурс] Режим доступа - <https://www.sqlite.org/about.html> - Дата доступа: 16.12.2021

### [Электронный ресурс] Режим доступа - <https://help.syncfusion.com/wpf/datepicker/formatting> - Дата доступа: 17.12.2021

### [Электронный ресурс] Режим доступа - <https://docs.microsoft.com/en-us/dotnet/api/?view=entity-framework-6.2.0> - Дата доступа: 16.12.2021