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 «Приложения для покупки и сборки ПК «PC Builder»

Исполнитель

студент (ка) 2 курса группы 6 Окулич Дмитрий Юрьевич

(Ф.И.О.)

Руководитель работы преп.-стажёр Чистякова Ю. А.

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

Курсовой проект защищен с оценкой

Председатель Пацей Н. В.

(подпись)

Минск 2023

**Содержание**

[ВВЕДЕНИЕ 3](#_Toc134923374)

[1. Аналитический обзор литературы 4](#_Toc134923375)

[1.1 Постановка задач 4](#_Toc134923376)

[1.2 Средства разработки 4](#_Toc134923377)

[1.3 Обзор прототипов 5](#_Toc134923378)

[1.3.1 Интернет-магазин Ram.by 5](#_Toc134923379)

[1.3.2 Интернет-магазин X-core.by 7](#_Toc134923380)

[1.3.3 Интернет-магазин “Фабрика компьютеров” fk.by 9](#_Toc134923381)

[1.3.4 Интернет-магазин 21vek.by 11](#_Toc134923382)

[1.3.5 Интернет-каталог Onliner.by 12](#_Toc134923383)

[1.3.6 Итог 12](#_Toc134923384)

[1.4 Функциональные требования 13](#_Toc134923385)

[2. Проектирование программного средства 14](#_Toc134923386)

[2.1 Модель базы данных 14](#_Toc134923387)

[2.2 Структура проекта 17](#_Toc134923388)

[2.3 Диаграмма UML и блок-схема алгоритма 17](#_Toc134923389)

[3. Реализация программного обеспечения 20](#_Toc134923390)

[3.1 Реализация моделей 20](#_Toc134923391)

[3.2 Реализация Репозитория и UnitOfWork 22](#_Toc134923392)

[3.3 Реализация ViewModel 24](#_Toc134923393)

[3.4 Вспомогательные классы 27](#_Toc134923394)

[4. Тестирование и анализ полученных результатов 30](#_Toc134923395)

[4.1 Тестирование 30](#_Toc134923396)

[5. Руководство по установке и использованию программного обеспечения 31](#_Toc134923397)

[5.1 Установка приложения 31](#_Toc134923398)

[5.2 Использование приложения 31](#_Toc134923399)

[Заключение 32](#_Toc134923400)

[Список использованных источников 33](#_Toc134923401)

[ПРИЛОЖЕНИЕ А 34](#_Toc134923402)

[ПРИЛОЖЕНИЕ Б 35](#_Toc134923403)

[ПРИЛОЖЕНИЕ В 36](#_Toc134923404)

[ПРИЛОЖЕНИЕ Г 37](#_Toc134923405)

[ПРИЛОЖЕНИЕ Д 38](#_Toc134923406)

[ПРИЛОЖЕНИЕ Е 39](#_Toc134923407)

[ПРИЛОЖЕНИЕ Ё 40](#_Toc134923408)

[ПРИЛОЖЕНИЕ Ж 47](#_Toc134923409)

## ВВЕДЕНИЕ

Сегодня, в связи с большой распространённостью компьютерного железа, много людей хотят сами собирать собственные сборки ПК. Так же это тенденция связана с тем, что покупка комплектующих по отдельности выходит порой значительно дешевле нежели покупать уже готовые сборки. Однако всё ещё много людей не могут найти времени на подбор комплектующих и проверку их совместимости. В таком случае люди прибегают к помощи специального ПО и различных сайтов. В данном курсовом проекте будет рассмотрено создание программного средства «Приложения для покупки и сборки ПК PC Builder», которое позволит пользователям, как и покупать комплектующие по отдельности так и создавать собственные сборки.

Цель данного курсового проекта – создать приложение для сборки и покупки ПК «PC Builder», которое будет предоставлять пользователям возможность поиска и покупки интересующих их комплектующих, создавать собственные сборки. В рамках проекта будут рассмотрены все этапы разработки программного средства, начиная от анализа требований и выбора технологий, заканчивая тестированием и документированием результатов.

Программное средство «Приложения для покупки и сборки ПК PC Builder» будет предоставлять возможность выбора комплектующих из каталога с последующим помещением оного в корзину. Так же будет дана возможность пользователям создавать свои сборки при помощи специального конфигуратора. Сборка может быть помещена в корзину для последующей покупки. Приложение будет иметь удобный и интуитивно понятный интерфейс, который позволит пользователям быстро и легко находить необходимую информацию. Результатом данной работы будет создание полноценного программного средства, которое поможет людям сильно ускорить сборку, а также будет развивать сообщество людей, которые интересуются сборкой ПК.

Главная задача данного курсового проектирования – это разработка программного средства, которое реализует все вышеперечисленные функции и решает поставленные задачи. Язык разработки проекта – C#. При выполнении курсового проекта будут использованы принципы и приемы ООП, база данных MS SQL Server, и технология Windows Presentation Foundation (WPF).

## Аналитический обзор литературы

## Постановка задач

Основной задачей курсового проекта сделать приложение, в котором пользователи смогут искать комплектующие для ПК и покупать их. Так же приложение будет давать возможность создавать свои сборки ПК и так же покупать их.

Функционально ПС должно выполнять следующие задачи:

* Функции администратора:
  + Управление базой данных;
  + Управление каталогом комплектующих ПК;
* Функции клиента:
  + Выполнять регистрацию и авторизацию;
  + Просмотр каталога комплектующих;
  + Заполнять форму заказа по заданным формам;
  + Создание/Удаление шаблонов сборок ПК;
  + Просмотр каталога шаблонов сборок ПК;

Описанный функционал позволит пользователю сделать вышеописанные действия.

## Средства разработки

В ходе разработки данного программного средства используются следующие инструменты:

* объектно-ориентированный язык программирования C#;
* платформа для кроссплатформенной разработки с открытым исходным

кодом .NET Framework 4.9;

* расширяемый язык разметки XAML;
* система управления базами данных Microsoft SQL Server;
* шаблон проектирования MVVM;
* фреймворк для работы с базой данных Entity Framework 6;
* интегрированная среда разработки Microsoft Visual Studio2022.

C# — это язык программирования, предназначенный для разработки самых разнообразных приложений, предназначенных для выполнения в среде .NET Framework. Visual C# — это реализация языка C# корпорацией Майкрософт. Поддержка Visual C# в Visual Studio обеспечивается с помощью полнофункционального редактора кода, компилятора, шаблонов проектов, конструкторов, мастеров кода, мощного и удобного отладчика и многих других средств. Библиотека классов .NET Framework предоставляет доступ ко многим службам операционной системы и другим полезным, правильным классам, что существенно ускоряет цикл разработки.

XAML — это декларативный язык разметки. С точки зрения модели программирования .NET Core язык XAML упрощает создание пользовательского интерфейса для приложения .NET Core. Можно создать видимые элементы пользовательского интерфейса в декларативной XAML-разметке, а затем отделить определение пользовательского интерфейса от логики времени выполнения, используя файлы кода программной части, присоединенные к разметке с помощью определений разделяемых классов. Язык XAML напрямую представляет создание экземпляров объектов в конкретном наборе резервных типов, определенных в сборках.

Microsoft SQL Server — система управления реляционными базами данных (РСУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов — Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка.

SQL Server — это основа платформы обработки данных Майкрософт, которая предоставляет надежную и устойчивую производительность (в том числе благодаря технологиям обработки данных в памяти) и помогает быстрее извлечь ценную информацию из любых данных, расположенных как в локальной среде, так и в облаке.

MVVM — шаблон проектирования архитектуры приложения, который позволяет отделить логику приложения от визуальной части, что упрощает тестирование и поддержку приложения.

## Обзор прототипов

Основными критериями сравнения будут каталог и конфигуратор ПК. Будут учитывается, как и их функционал так и их удобство.

Список аналогов:

1. Интернет-магазин [Ram.by](https://ram.by/fastconfig)
2. Интернет-магазин [X-core.by](https://x-core.by/onlayn-konfigurator)
3. Интернет-магазин “Фабрика компьютеров” [fk.by](https://fk.by/configurator)
4. Интернет-магазин [21vek.by](https://www.21vek.by/hardware/)
5. Интернет-каталог [Onliner.by](https://catalog.onliner.by)

## Интернет-магазин Ram.by

Ram.by имеет два разных по сложности сборщика ПК. Первый, он же упрощенный, показанный на рисунке 1.1 представляет собой простые ползунки, где чем дальше ползунок, тем лучше комплектующие конкретного типа, так же даёт возможность в некоторых комплектующих менять производителя. Второй, он же детальный, показанный на рисунке 1.3 представляет собой список комплектующих в котором пользователь выбирает нужный себе элемент, справа от списка отображается список выборных комплектующих, а также их совместимость между собой.

Каталог, показанный на рисунке 1.1, представляет панель фильтров слева, список комплектующих по середине, комплектующие отображены в виде фото и названия ниже.

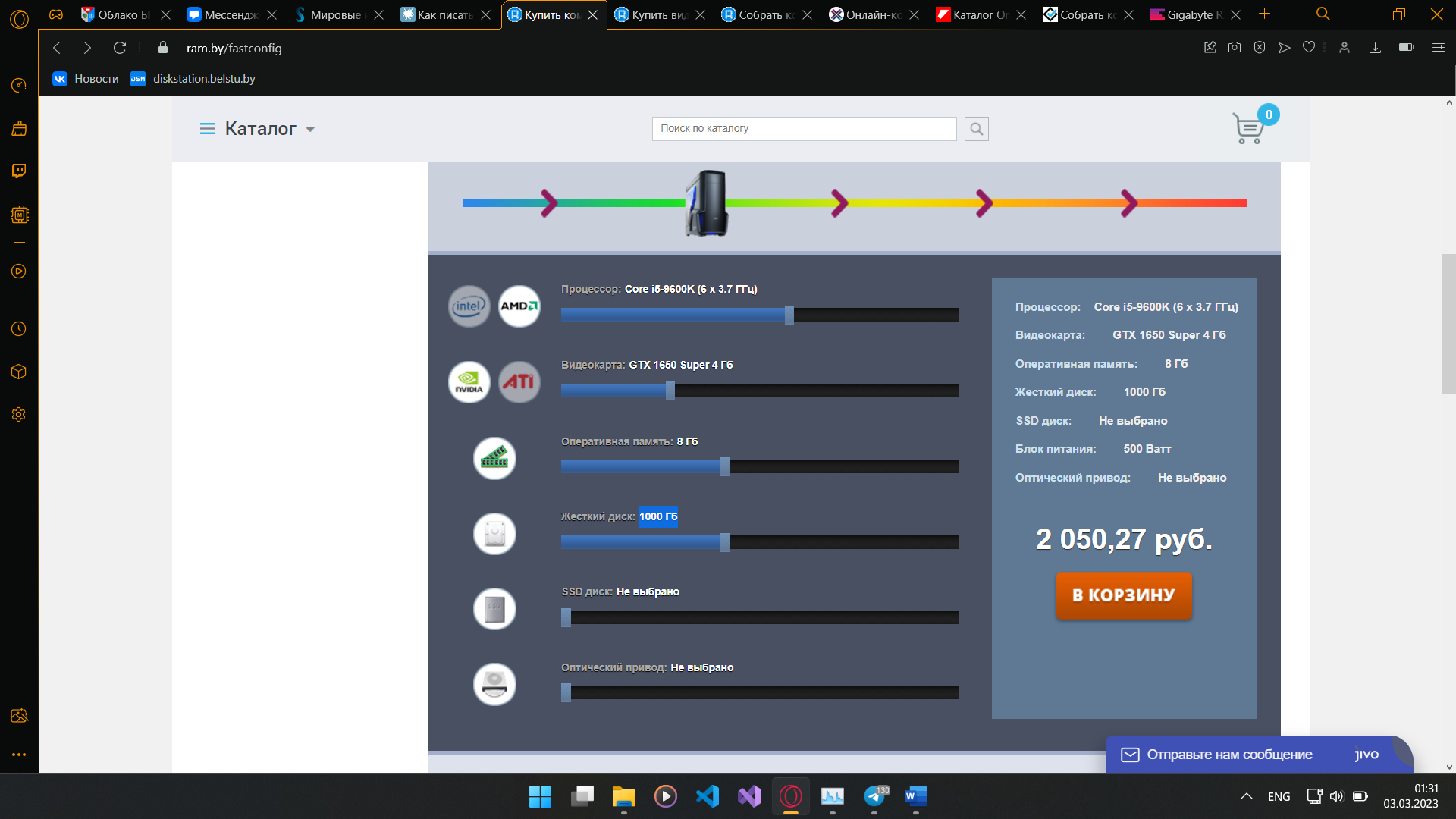


Рисунок 1.1 Упрощённый сборщик ПК

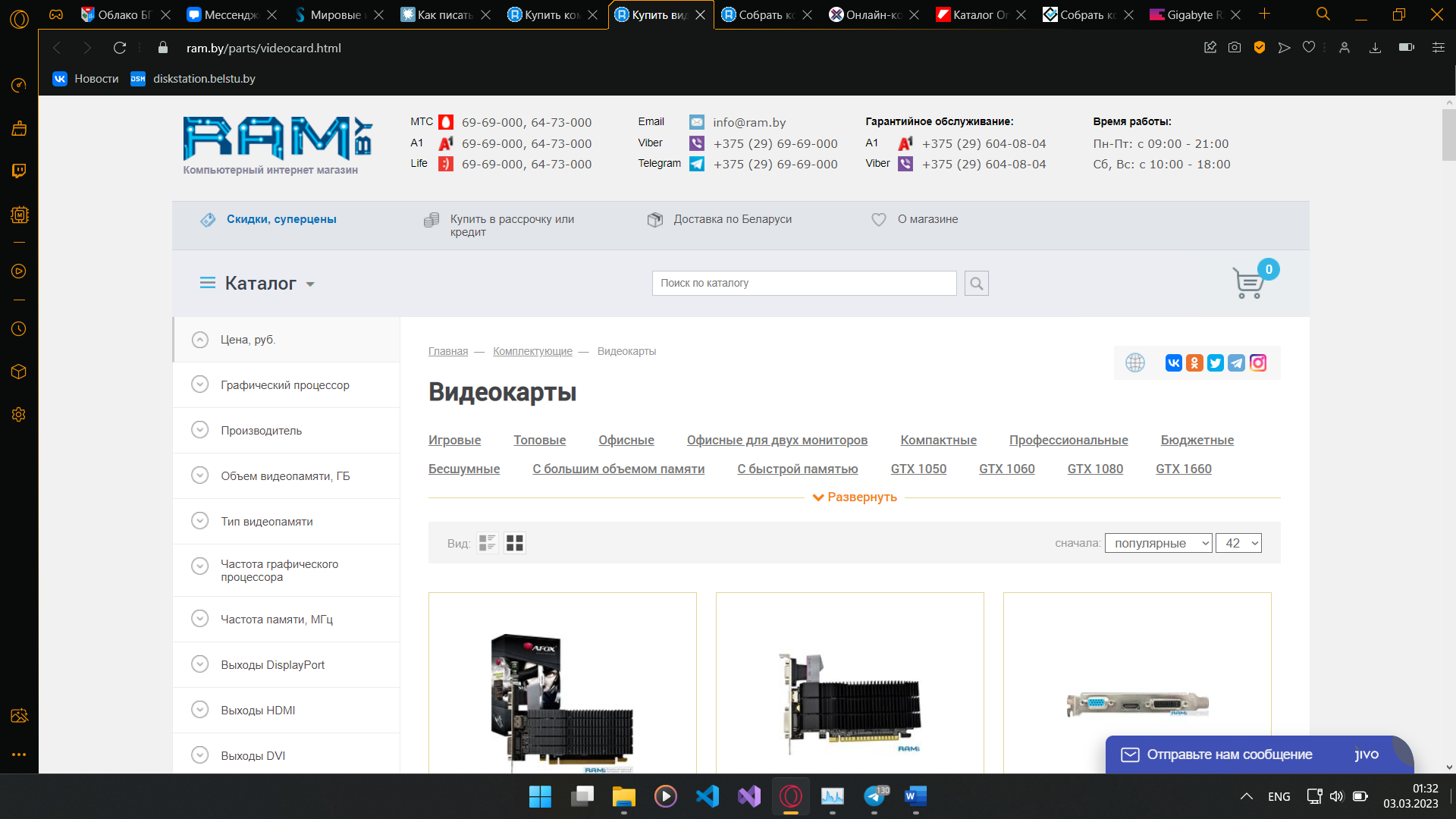


Рисунок 1.2 Каталог

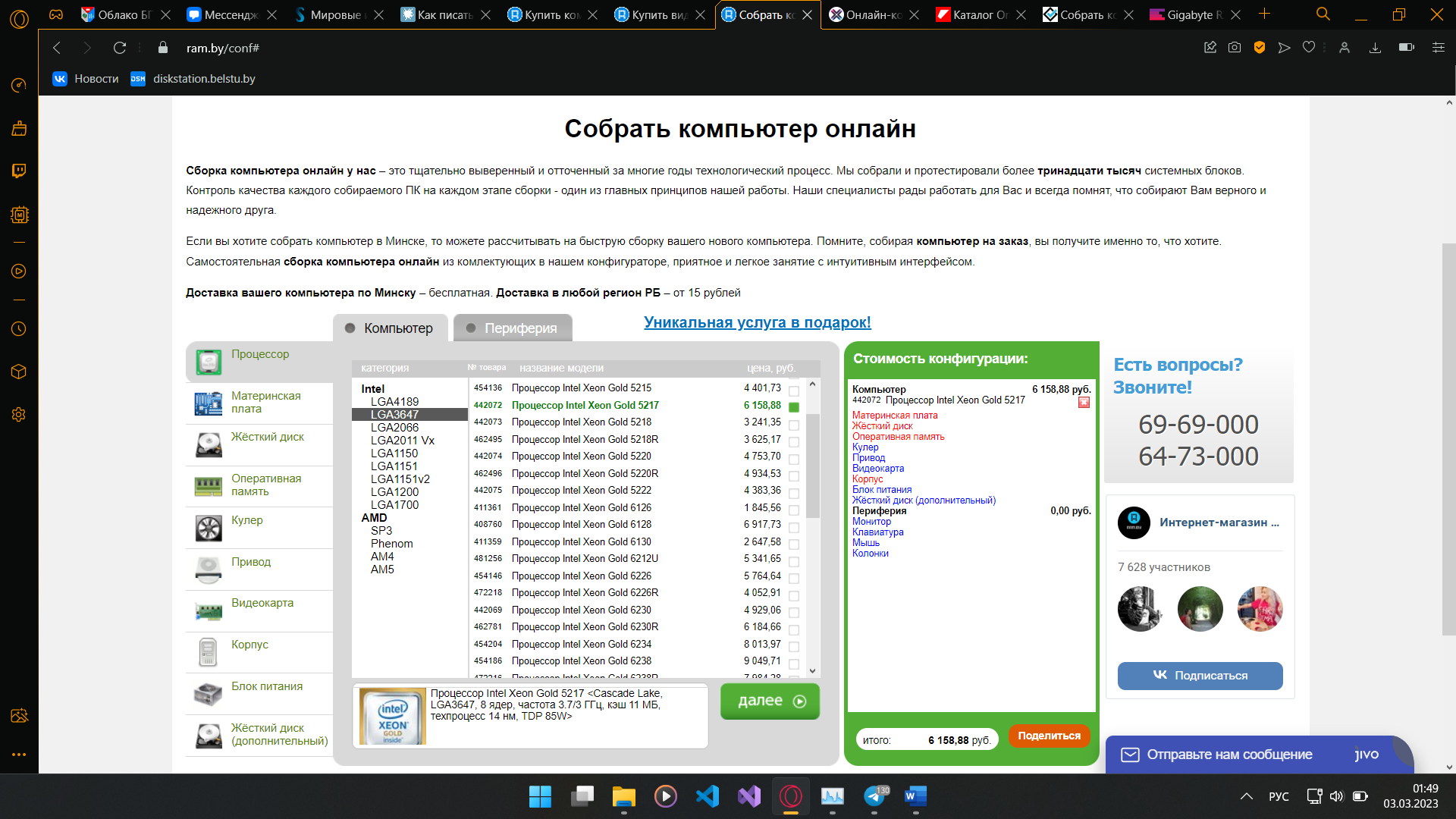


Рисунок 1.3 Детальный сборщик ПК

Плюсы:

* Есть два редактора что увеличивает потенциальную аудиторию;
* Простой редактор позволяет собрать ПК в кротчайшее время.

Минусы:

* Простому редактору не достаёт комплектующих таких как: материнская плата, блок питания;
* Устаревший дизайн на страницах редакторов;
* Все страницы магазина перегружены часто не нужной пользователю информацией;
* Детальный редактор крайне сложный и не удобны в использовании.

## 1.3.2 Интернет-магазин X-core.by

Каталог X-core, показанный на рисунке 1.4, в большинстве своём не отличается от других аналогов кроме наличия ползунков в нескорых фильтрах как на пример ползунок, ограничивающий цену.

Редактор, или же конфигуратор, показанный на рисунке 1.5, представляет из себя список комплектующих разделенный на два подсписка: системный блок, периферийные устройства. Системный блок — это сборка самого ПК, а периферийные устройства — это то, что к комплектующим ПК не относится, однако, являются частью ПК, например клавиатура. Для выбора детали на сайте открывается окно, которое является обычным каталогом.

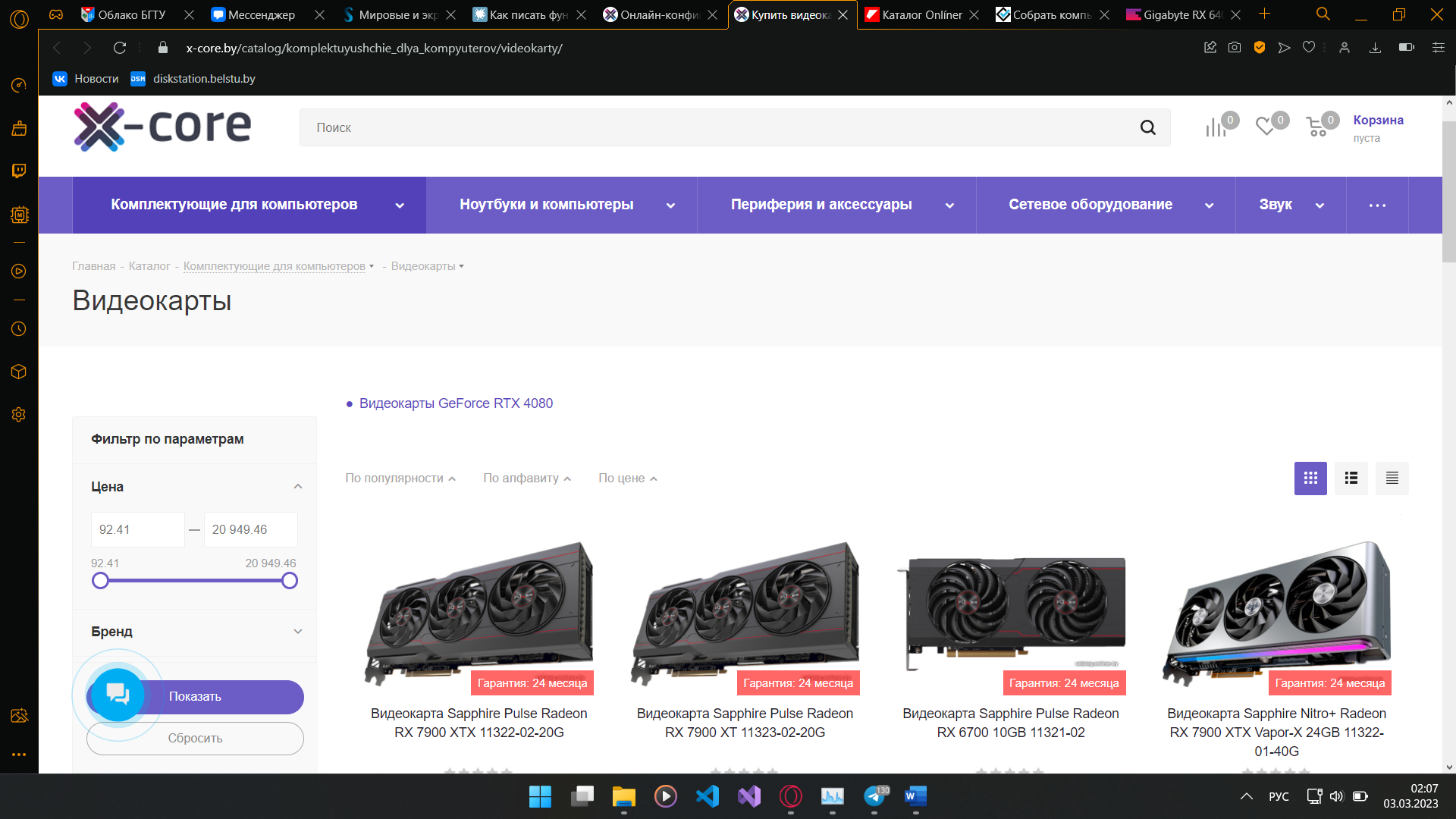


Рисунок 1.4 Каталог

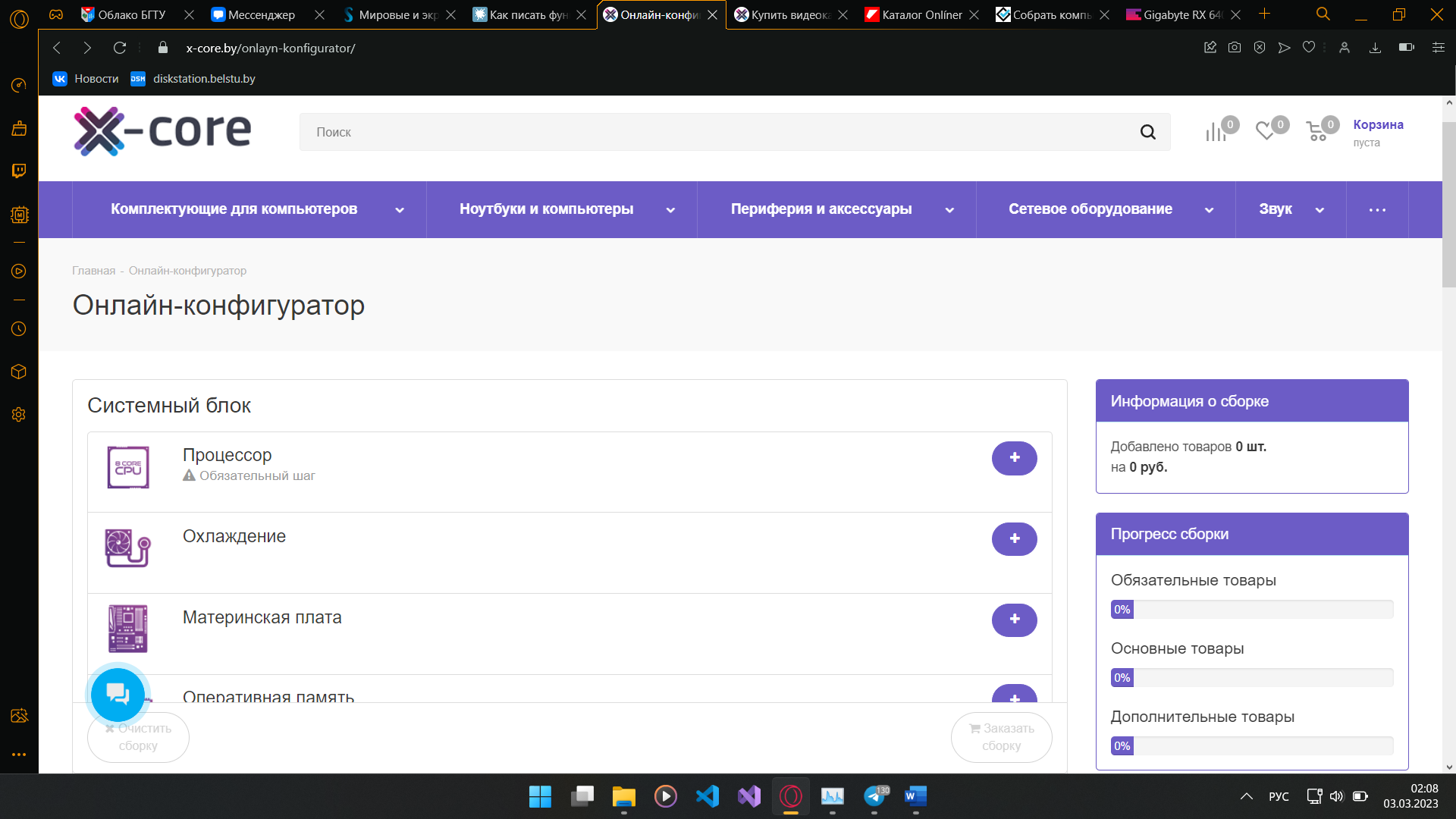


Рисунок 1.5 Конфигуратор

Плюсы:

* Хороший дизайн;
* Удобный конфигуратор;
* Есть автоматический фильтр по совместимости комплектующих.

Минусы не были обнаружены или были не совсем значительны.

## 1.3.3 Интернет-магазин “Фабрика компьютеров” fk.by

Каталог в Fk.by, показанный на рисунках 1.6 и 1.7, выделяется более полный описанием элемента под его названием.

Конфигуратор, показанный на рисунке 1.8, похож на тот который был Ram.by, однако в нем больше пунктов, а также есть возможность скопировать ссылку на конфигурацию.

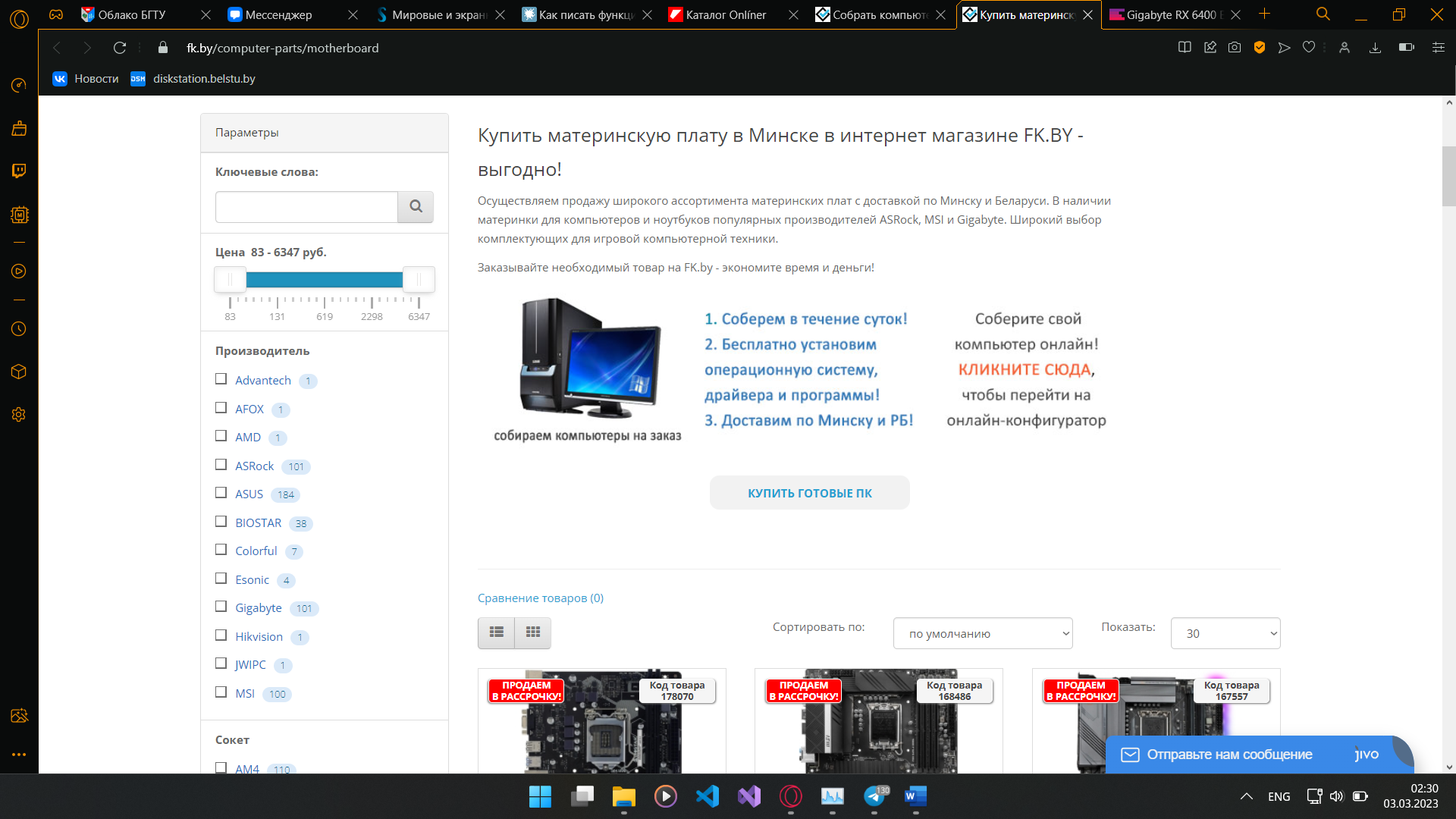


Рисунок 1.6 Каталог 1

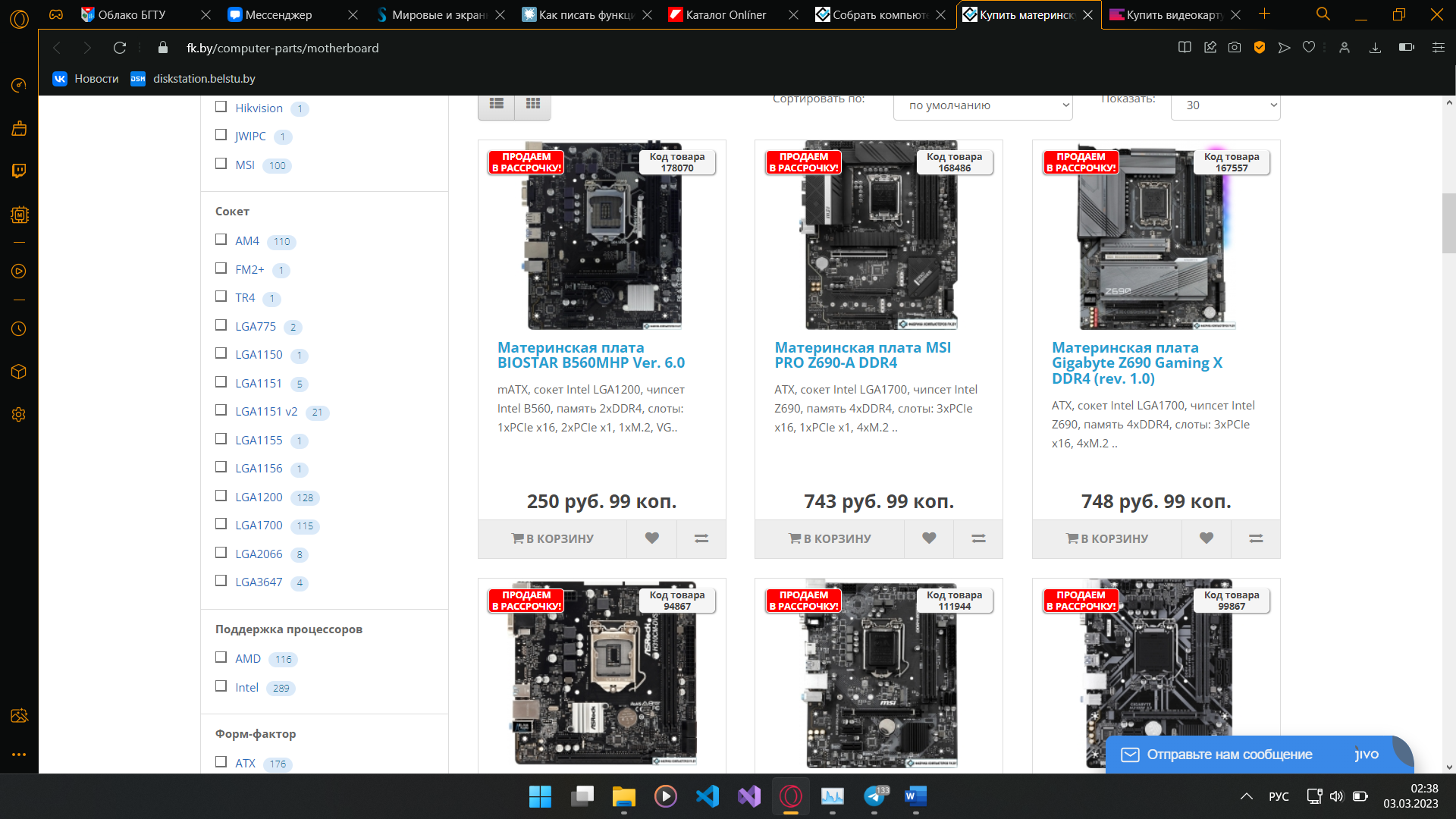


Рисунок 1.7 Каталог 2

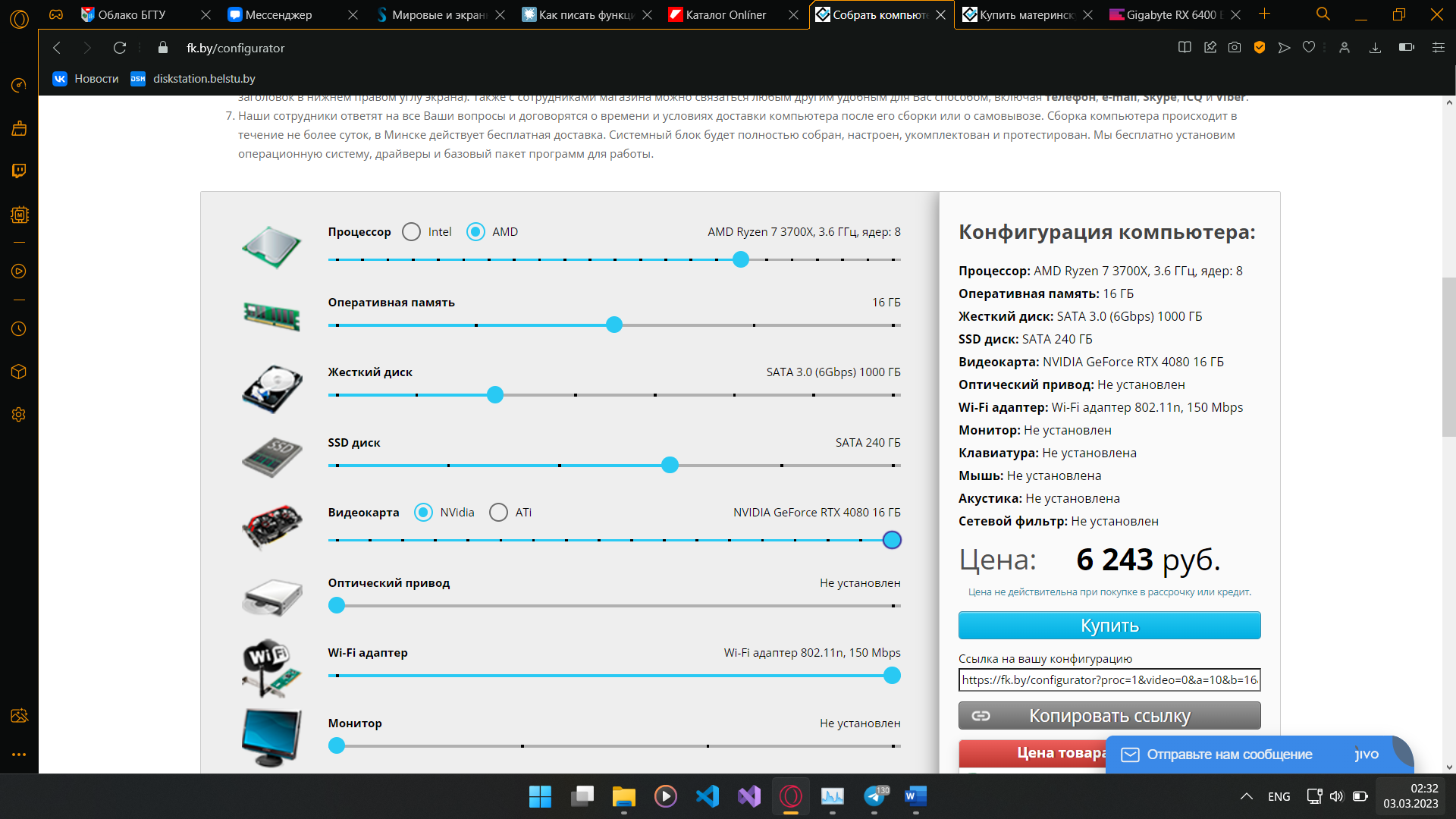


Рисунок 1.8 Конфигуратор

Плюсы:

* Детально описание элементов в каталоге;
* Много пунктов в конфигураторе;
* Возможность скопировать ссылку на конфигурацию.

Минусы:

* Все страницы магазина перегружены часто не нужной пользователю информацией;
* Немного устаревший дизайн.

## 1.3.4 Интернет-магазин 21vek.by

21vek.by является вторичным аналогом. Хоть он и не предоставляет возможности создавать конфигурацию ПК, однако у него есть интересный каталог, показанный на рисунке 1.9, предоставляющий более обширный каталог комплектующих, а также имеет большой набор фильтров.

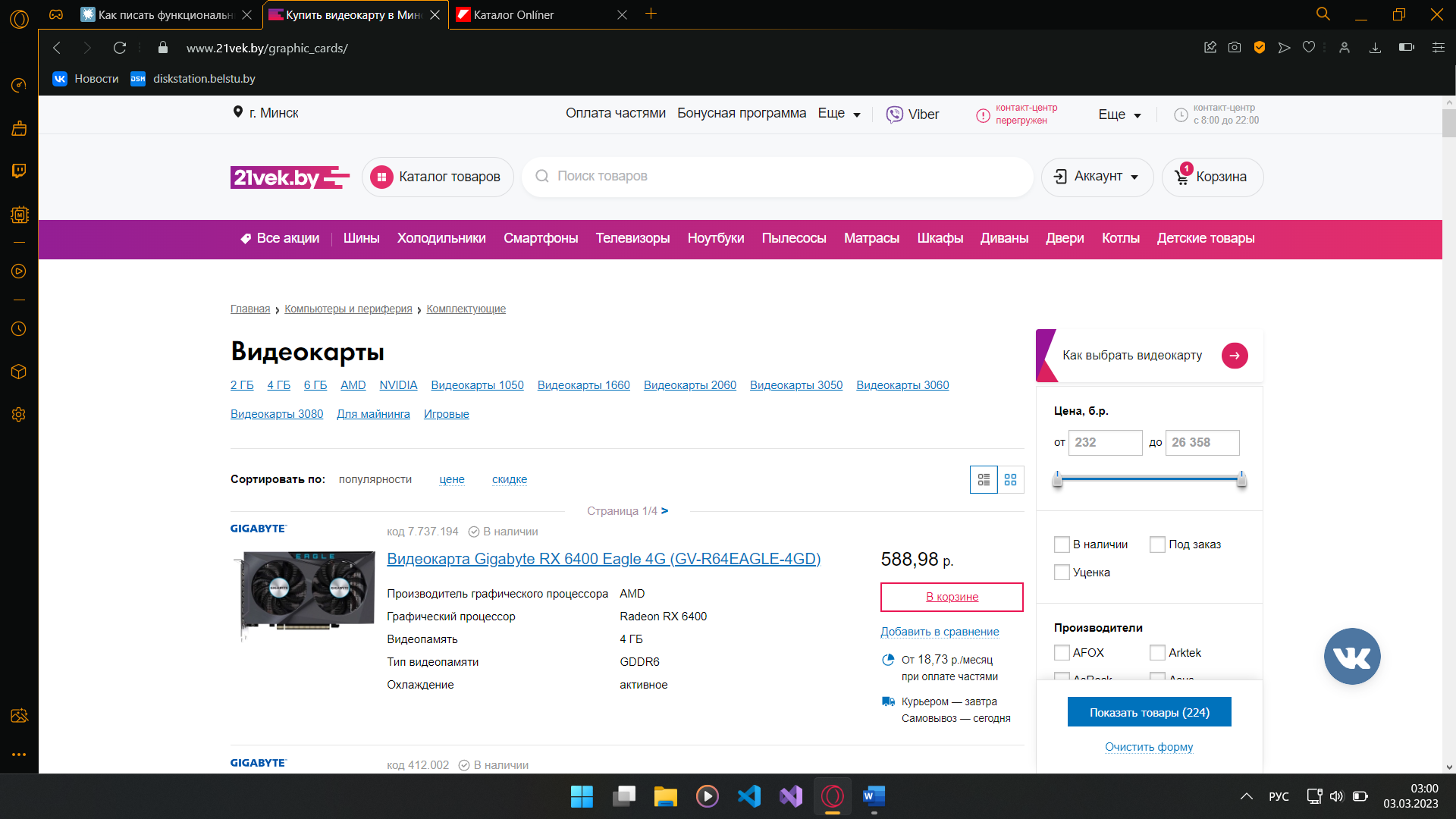


Рисунок 1.9 Каталог

Плюсы:

* Дизайн
* Каталог выдаёт много информации
* Много фильтров

Минусы не были обнаружены.

## 1.3.5 Интернет-каталог Onliner.by

Onliner.by является непрямым аналогом. Его каталог, показанный на рисунке 1.10, по функционалу почти такой же как и у 21vek.by, однако он предостовляет возможноть выбора поставщика товара.

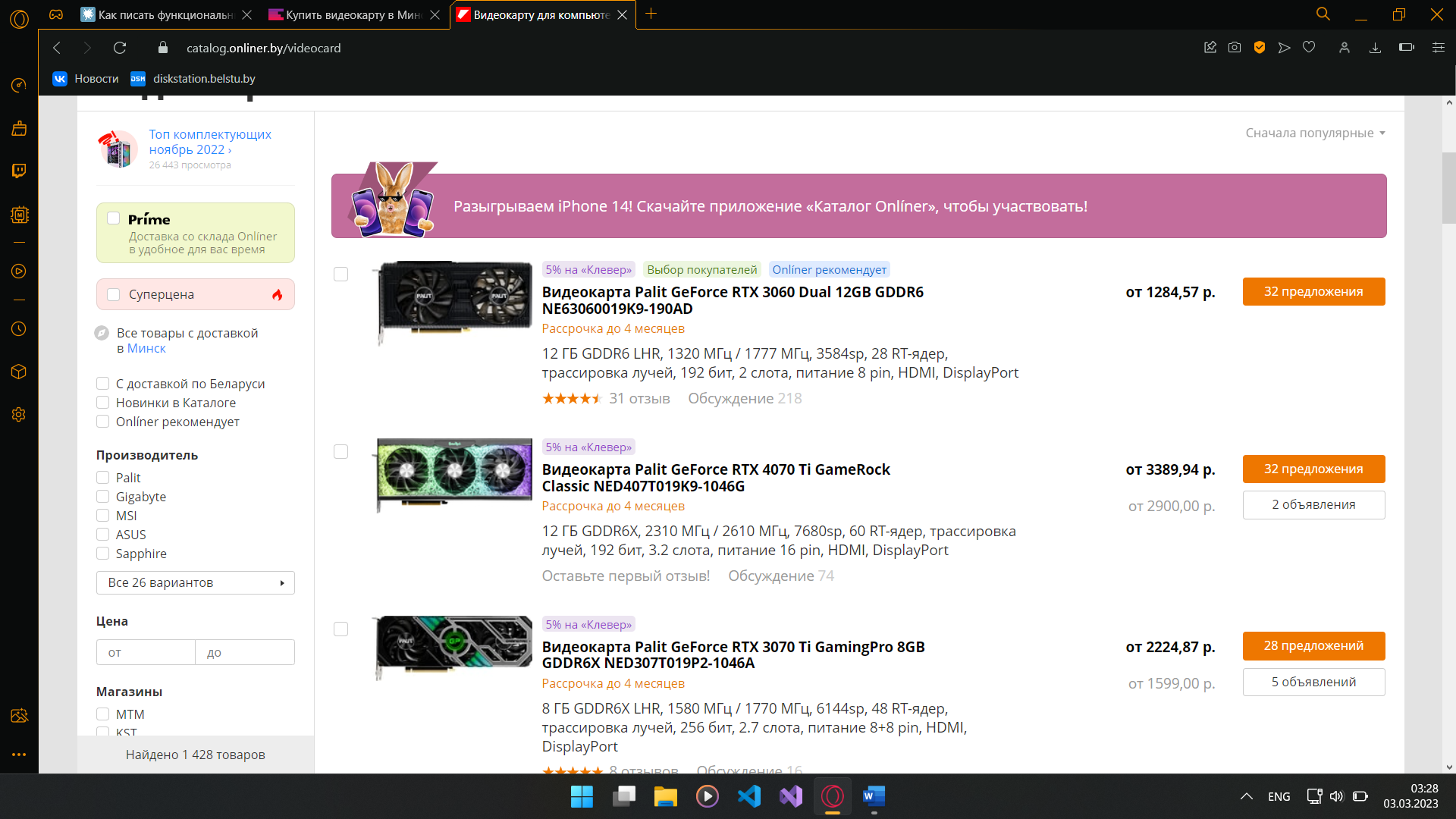


Рисунок 1.10 Каталог

Плюсы совпадают с 21vek.by.

## 1.3.6 Итог

По итогу можно сказать, что конфигуратор будет похож на тот который в X-core.by, однако с возможностью выкладывать конфиги в общий доступ что бы другие пользователи могли их скопировать или прокомментировать.

Каталог будет что-то средне между 21vek.by и Onliner.by.

Список рекомендаций:

* Не нагромождать окна лишней информацией;
* Конфигуратор должен быть детальный, но при этом достаточно удобным;
* Связать окно конфигуратора с окном каталога;
* В предпросмотре комплектующего можно отобразить больше информации.

## Функциональные требования

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 в Приложении A.

Система состоит из следующих блоков: «Регистрация», «Авторизация», «Управление своей корзиной», «Использование каталога комплектующих», «Создание сборки ПК», «Удаление сборки ПК», «Использование каталога сборок ПК», «Включить комплектующие в корзину», «Включить сборку ПК в корзину», «Отправка заказа на обработку», «Управление каталогом сборок ПК», «Управление базой данных», «Управление каталогом комплектующих». Некоторые блоки имеют зависимости так на пример «Отправка заказа на обработку» может быть сделана только если было сделано «Включить комплектующие в корзину» или «Включить сборку ПК в корзину». Так же при «Создание сборки ПК» в обязательном порядке будет использоваться блок «Использование каталога комплектующих».

## Проектирование программного средства

## Модель базы данных

База данных состоит из семи таблиц. Диаграмма базы данных со всеми зависимостями предоставлена в приложении Б.

Таблица Users необходима для хранения информации о пользователях.

Таблица 2.1 – Таблица Users

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| Id | Int | Содержит идентификатор пользователя |
| FirstName | nvarchar(MAX) | Содержит имя пользователя |
| LastName | nvarchar(MAX) | Содержит фамилию пользователя |
| Email | nvarchar(MAX) | Содержит электронную почту пользователя |
| Password | nvarchar(MAX) | Содержит пароль пользователя |
| ContactNumber | nvarchar(MAX) | Содержит контактный номер пользователя |
| Address | nvarchar(MAX) | Содержит адрес пользователя |
| Rights | nvarchar(MAX) | Содержит права пользователя |
| LastAuthDate | Datetime | Содержит дату последней авторизации |

Таблица Products необходима для хранения информации о товарах.

Таблица 2.2 – Таблица Products

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| Id | Int | Содержит идентификатор товара |
| Manufacturer | nvarchar(MAX) | Содержит название производителя |
| Name | nvarchar(MAX) | Содержит название товара |
| ShortDescription | nvarchar(MAX) | Содержит краткое описание товара |
| FullDescription | nvarchar(MAX) | Содержит полное описание товара |
| ImageUrl | nvarchar(MAX) | Содержит название изображения товара с расширением |
| Price | int | Содержит цену товара |
| Type | nvarchar(MAX) | Содержит тип товара |

Таблица Performances необходима для хранения свойств товаров.

Таблица 2.3 – Таблица Performances

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| Id | Int | Содержит идентификатор свойства |
| ProductId | int | Содержит идентификатор товара, которому принадлежит свойство |
| Name | nvarchar(MAX) | Содержит название свойства |
| Value | nvarchar(MAX) | Содержит значение свойства |
| Tag | nvarchar(MAX) | Содержит символьный тег свойства |
| DependencyType | nvarchar(MAX) | Содержит тип отношения свойства относительно других свойств |

Таблица Templates необходима для хранения сборок ПК.

Таблица 2.4 – Таблица Templates

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| Id | Int | Содержит идентификатор сборки |
| CreatorId | int | Содержит идентификатор пользователя, создавшего сборку |
| Name | nvarchar(MAX) | Содержит название свойства |
| Description | nvarchar(MAX) | Содержит описание сборки |

Таблица TemplateItems необходима для создания связи «многие ко многим» между таблицами Products и Templates.

Таблица 2.5 – Таблица TemplateItems

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| Id | Int | Содержит идентификатор предмета сборки |
| TemplateId | int | Содержит идентификатор сборки |
| ProductId | int | Содержит идентификатор продукта |

Таблица Orders необходима для хранения заказов пользователей.

Таблица 2.6 – Таблица Orders

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| Id | Int | Содержит идентификатор заказа |
| UserId | int | Содержит идентификатор пользователя |
| Status | nvarchar(MAX) | Содержит статус заказа |
| Note | nvarchar(MAX) | Содержит заметку пользователя к заказу |
| OpenOrderDate | Datetime | Содержит дату открытия заказа |
| CloseOrderDate | Datetime | Содержит дату закрытия заказа |

Таблица OrderItems необходима для создания связи «многие ко многим» между таблицами Orders, Products и Tempate.

Таблица 2.7 – Таблица Orders

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| Id | Int | Содержит идентификатор предмета заказа |
| OrderId | int | Содержит идентификатор заказа |
| ProductId | int? | Содержит идентификатор продукта |
| TemplateId | int? | Содержит идентификатор сборки ПК |

## Структура проекта

Проект будет состоять из 11 папок описание который предоставленной на таблице 2.8.

Таблица 2.8 – описание папок проекта

|  |  |
| --- | --- |
| Элемент | Описание |
| Commands | Содержит файлы для паттерна command |
| CustomUI | Содержит файлы самодельных UI элементов |
| Filters | Содержит файлы фильтров для каталога комплектующих |
| Localization | Содержит файлы локализации |
| Resources | Содержит файлы ресурсов (фото, иконки и тд.) |
| Utilities | Содержит файлы специальных классов необходимых для работы ПС |
| Styles | Содержит файлы стилей для UI элементов |
| Repositories | Содержит репозитории для моделей |
| Model | Содержит модели ПС |
| ViewModel | Содержит файлы viewmodel для отображения моделей |
| View | Содержит файлы окон и страниц |

Большое внимание уделяется качеству организации архитектуры приложения по этому было принято решение об использование архитектурного паттерна MVVM, а так же необходимого для этого паттерна Command. Для более удобной работы с базой данных было принято решение об использовании двух паттернов Repository и UnitOfWork.

## Диаграмма UML и блок-схема алгоритма

Диаграмма UML представляет собой графическое изображение структуры системы и взаимосвязей между ее элементами. Для курсового проекта необходимо построить диаграмму, которая будет отражать все основные компоненты системы и связи между ними.

Для создания диаграммы UML можно использовать различные виды диаграмм, такие как диаграмма вариантов использования (Use Case), диаграмма классов (Class Diagram), диаграмма последовательности (Sequence Diagram), диаграмма компонентов (Component Diagram) и другие.

Диаграмма классов в рамках паттерна MVVM (Model-View-ViewModel) включает три основных класса: Model, View и ViewModel, а также дополнительные классы, используемые в приложении.

Класс Model отвечает за представление данных и содержит всю логику, необходимую для работы с данными. Он может включать свойства, методы и события, и представляет данные, которые будут отображаться в пользовательском интерфейсе.

Класс View представляет пользовательский интерфейс и отвечает за отображение данных из модели. Он включает элементы управления, такие как кнопки, поля ввода и метки, которые позволяют пользователю взаимодействовать с приложением.

Класс ViewModel является посредником между View и Model. Он предоставляет интерфейс для взаимодействия между View и Model, обеспечивая связь между ними. ViewModel может содержать свойства, методы и события, которые позволяют View получать данные из Model и отображать их, а также обрабатывать события пользователя и отправлять их в Model для обработки.

Дополнительные классы, используемые в приложении, могут включать классы, представляющие отдельные элементы пользовательского интерфейса, классы, содержащие логику обработки данных и классы, представляющие сервисы и взаимодействующие с внешними источниками данных. Основные зависимости между классами представлены Приложение В и Приложение Г. В данных приложениях продемонстрированы диаграммы для классов моделей и репозитория.

Для моделирования взаимодействия объектов в языке UML используются соответствующие диаграммы последовательности представленная в Приложении Д.

Данная диаграмма описывает взаимодействие пользователя и приложения во время регистрации и авторизации.

Сценарий начинается с того что пользователь подключается к базе данных через приложения. При удачном поключения пользователь взаимодействует с окном авторизации и отправляет запрос на переход в окно регистрации после чего приложение выполняего его запрос. В окно регестрации пользователь после запольнения данных отправляет запрос на регистрацию, далее окно регистрации создаёт новый экземляр класса User и отправляет его в базу данных. Если всё прошло удачно и пользователь был добавлен в базу данных, то окну вернётся соответствующий результат и он сообщит пользователь об успехе операции, а так же вернёт обратно в окно авторизации. Далее пользователи выпольняет вход отправляя окну авторизации свои данные, после чего окно отправит запрос на проверку существования пользователя и если всё прошло удачно, то окну вернётся предпологаемый пользователь и после проверки на пароль вход будет выпольнен.

Надо уточнить, что в виде окон предпологаются их ViewModel классы которые и буду обрабатывать запросы от пользователя.

Так же стоит знать в каких состояних находится приложение во время авторизации и регистрации для этого была составлена диаграмма состояний продемострированна в Приложении Е.

Данная диаграмма явно показывает какие действия проводит программа и пользователь для достижения необходимого результата. Так например видно что во время регистрации позьватель вводит данные, а окно регистрации проверяет его данные на валидность, и если валидация подтверждена, то идёт поиск существуюего такого пользователя в базе данных, и если такого нет, то только тогда регистрация подствеждается, а иначе пользователь потвторно вводит данные.

## Реализация программного обеспечения

## Реализация моделей

Модели — классы описывающие данные с которыми приложению нужно будет работать. Модели так же удобно использовать для того, чтобы описывать таблицы для базы данных.

Так как используется Entity Framework c подходом Code first необходимо было сделать класс, который будет наследовать класс DbContext, такой класс продемонстрирован на рисунке 3.1.

Рисунок 3.1 Контекст базы данных

public class DataBase : DbContext

{

public DbSet<User> Users { get; set; }

public DbSet<Product> Products { get; set; }

public DbSet<Performance> Performances { get; set; }

public DbSet<Template> Templates { get; set; }

public DbSet<TemplateItems> TemplateItems { get; set; }

public DbSet<Order> Orders { get; set; }

public DbSet<OrderItem> OrderItems { get; set; }

public DataBase() : base("default0") { }

}

Класс DataBase помимо наследования DbContext ещё создаёт экземпляры класса DbSet<T> который представляет коллекцию объектов, которые могут быть запрошены, добавлены, изменены и удалены из базы данных. Каждый DbSet представляет таблицу в базе данных и предоставляет методы для выполнения операций CRUD с данными.

Так же необходимо вызвать базовый конструктор, в который передаётся название параметра в конфигураторе, который обозначает строку подключения.

Но для работы с базой данных требуется создать классы моделей. Следующие классы были созданы: User, Product, Performance, Template, TemplateItem, Order, OrderItem. Для примера следует разобрать класс модели User код которого предоставлен на рисунке 3.2.

public class User

{

[NotMapped]

public static User Current { get; set; }

[Key]

public int Id { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Email { get; set; }

public string Address { get; set; }

public string ContactNumber { get; set; }

public string Password { get; set; }

public string Rights { get; set; }

public DateTime LastAuthDate { get; set; }

}

Рисунок 3.2 класс модели User

Класс содержит свойства для хранения информации о пользователе, такие как имя, фамилия, электронная почта, адрес, контактный номер, пароль, права доступа и дата последней аутентификации.

Свойство "Id" помечено атрибутом [Key], что означает, что это свойство будет использоваться в качестве первичного ключа в базе данных.

Свойство "Current" помечено атрибутом [NotMapped], что означает, что это свойство не будет отображаться в базе данных при создании таблицы для класса "User".

Этот код использует аннотации из пространства имен "System.ComponentModel.DataAnnotations", которые могут использоваться для настройки маппинга классов на таблицы базы данных в Entity Framework.

## Реализация Репозитория и UnitOfWork

Приложение реализует паттерны Репозиторий и UnitOfWork для это были сделаны следующие классы и интерфейсы: IRepository<T>, RepositoryBase<T>, DataBaseManager.

IRepository представляет собой обобщенный интерфейс и используется для доступа к данным. Код продемонстрирован на рисунке 3.3.

public interface IRepository<T>

{

T Get(int id);

List<T> GetAll();

void Add(T item);

void Update(T item);

void Delete(int id);

}

Рисунок 3.3 интерфейс репозитория

Он содержит 6 методов. Get(int id) - получает объект T по заданному идентификатору. GetAll() - получает все объекты T из источника данных. Add(T item) - добавляет новый объект T в источник данных. Update(T item) - обновляет существующий объект T в источнике данных. Delete(int id) - удаляет объект T с заданным идентификатором из источника данных.

Репозиторий позволяет разделить бизнес-логику и логику доступа к данным, что делает код более чистым и поддерживаемым. Конкретные реализации репозитория могут использовать любой источник данных, такой как базу данных, веб-сервис или файловую систему. Интерфейс IRepository позволяет установить общий интерфейс для доступа к данным независимо от конкретной реализации.

RepositoryBase<T> - базовый абстрактный класс, который реализует интерфейс IRepository<T>. Код RepositoryBase<T> продемонстрирован на рисунке 3.4.

public abstract class RepositoryBase<T> : IRepository<T>

{

protected DataBase \_dataBase;

public RepositoryBase(DataBase dataBase)

{

\_dataBase = dataBase;

}

public abstract void Add(T item);

public abstract void Delete(int id);

public abstract T Get(int id);

public abstract List<T> GetAll();

public abstract void Update(T item);

}

Рисунок 3.4 абстрактный класс репозитория

Класс имеет защищенное поле \_dataBase, которое используется для хранения экземпляра класса DataBase, отвечающего за управление доступом к данным. Конструктор класса принимает параметр dataBase, который инициализирует поле \_dataBase.

Класс определяет абстрактную реализацию все методов, объявленных в интерфейсе IRepository<T>. Таким образом, класс наследник должен реализовать эти методы.

Базовый класс позволяет сократить дублирование кода и повысить уровень абстракции для работы с различными типами данных. Конкретная реализация репозитория может наследоваться от базового класса и реализовать методы в соответствии с конкретными требованиями.

Примером наследования класса RepositoryBase<T> является класс UserRepository листинг которого продемонстрирован в Приложении Ё Листинг 1.

После создания всех репозиториев нужно бы собрать из в одном месте для удобного доступа для это и служи паттерн UnitOfWork. Однако, помимо этого, он позволит управлять соединением. Реализацией этого паттерна в проекте является класс DataBaseManager листинг которого предоставлен в Приложении Ё Листинг 2.

Это класс управляет подключением к базе данных и предоставляет доступ к репозиториям, через которые можно взаимодействовать с данными.

В классе определены следующие поля и методы:

* Instance: статическое поле, которое хранит единственный экземпляр класса DataBaseManager.
* \_locktemp0 и \_locktemp1: объекты для синхронизации потоков при создании единственного экземпляра класса.
* OnConnected: событие, которое вызывается при успешном подключении к базе данных.
* OnError: событие, которое вызывается при возникновении ошибки при работе с базой данных.
* OnSuccess: событие, которое вызывается при успешном выполнении операции с базой данных.
* \_dataBase: объект класса DataBase, который предоставляет доступ к базе данных.
* IsConnected: флаг, который указывает, подключен ли объект DataBaseManager к базе данных.
* Users, Orders, OrderItems, Products, Performances, Templates, TemplateItems: объекты репозиториев для работы с соответствующими таблицами базы данных.
* CreateInstance(): статический метод, который создает единственный экземпляр класса DataBaseManager, если он еще не создан.
* Connect(): метод, который устанавливает соединение с базой данных, инициализирует ее, добавляет запись "ADMIN" в таблицу Users, если такой записи еще нет, и создает объекты репозиториев.
* ConnectAsync(): асинхронный метод, который выполняет аналогичные действия, что и метод Connect(), но выполняет их асинхронно с использованием Task.
* Dispose(): метод, который освобождает ресурсы, используемые объектом DataBaseManager.
* DropError(): метод, который вызывается в случае возникновения ошибки при работе с базой данных.
* DropSuccess(): метод, который вызывается при успешном выполнении операции с базой данных.

## Реализация ViewModel

ViewModel является посредником между View и Model и предоставляет данные, которые View может использовать для отображения информации и выполнения пользовательских действий. ViewModel также обрабатывает пользовательский ввод из View и обновляет Model соответствующим образом.

Для реализации ViewModel в проекте существует три базовых класса: BaseViewModel, WindowViewModel<T>, FrameViewModel<F, W>.

BaseViewModel представляет собой самый базовый класс для ViewModel который реализует интерфейс INotifyPropertyChanged и IDisposable. Код класса предоставлен на рисунке 3.5.

public abstract class BaseViewModel : INotifyPropertyChanged, IDisposable

{

public event PropertyChangedEventHandler PropertyChanged;

public void OnPropertyChanged([CallerMemberName] string prop = "")

{

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs(prop));

}

public virtual void Dispose() { }

}

Рисунок 3.5 базовая ViewModel

Интерфейс INotifyPropertyChanged предназначен для уведомления об изменениях свойств объекта. Когда свойство объекта изменяется, вызывается метод OnPropertyChanged, который вызывает событие PropertyChanged, чтобы уведомить подписчиков об изменении свойства.

Интерфейс IDisposable предназначен для освобождения неуправляемых ресурсов, используемых объектом. Когда объект больше не нужен, его можно утилизировать, вызвав метод Dispose.

В данном коде метод Dispose определен как виртуальный, поэтому его можно переопределить в производных классах. Он позволяет освободить ресурсы, используемые объектом BaseViewModel. Метод OnPropertyChanged вызывает событие PropertyChanged, которое указывает, какое свойство было изменено.

Так как в проекте используются, как и окна так и страницы было принято решения разбить BaseViewModel на два более узкоспециализированных класса. WindowViewModel предназначен для управления окна в приложении. Код класса предоставлен на рисунке 3.6.

public abstract class WindowViewModel<T> : BaseViewModel

where T : Window

{

public T Owner { get; private set; }

public WindowViewModel(T owner)

{

Owner = owner;

}

}

Рисунок 3.6 ViewModel для окон

Этот класс представляет базовый класс модели представления для окон приложения и параметризован типом окна T, который ограничивается наследованием от класса Window.

У этого класса есть свойство Owner, которое представляет владельца ViewModel в виде окна. Он устанавливается через конструктор, где объект окна передается как параметр.

FrameViewModel предназначен для страниц используемых в окне. Код этого класс предоставлен на рисунке 3.7.

public abstract class FrameViewModel<F, W> : BaseViewModel

where F : Page

where W : Window

{

public F Owner { get; private set; }

public W OwnerWindow { get; private set; }

public FrameViewModel(F owner, W window)

{

Owner = owner;

OwnerWindow = window;

}

protected void AnimateAwake(FrameworkElement block)

{

ThicknessAnimation marginAnim = new ThicknessAnimation();

marginAnim.From = new Thickness(0,15,0,0);

marginAnim.To = new Thickness(0);

marginAnim.EasingFunction = new QuadraticEase() { EasingMode =

EasingMode.EaseOut };

marginAnim.Duration = TimeSpan.FromSeconds(0.4);

block.BeginAnimation(FrameworkElement.MarginProperty, marginAnim);

}

}

Рисунок 3.7 ViewModel для страниц

Класс FrameViewModel<F, W> является абстрактным и унаследован от BaseViewModel. Он имеет два параметра типа F и W, которые ограничены наследованием от классов Page и Window соответственно.

У класса есть два свойства: Owner, тип которого соответствует параметру F, и OwnerWindow, тип которого соответствует параметру W. Оба свойства имеют приватный сеттер и задаются в конструкторе. Они обозначают страницу этого ViewModel и окно, в котором страница находится.

У класса также есть метод AnimateAwake, который принимает FrameworkElement. Внутри метода происходит создание анимации ThicknessAnimation, которая задает начальное значение для отступа свойства Margin элемента FrameworkElement и конечное значение. Происходит также задание функции смягчения анимации и продолжительности. Затем метод запускает анимацию для свойства Margin элемента.

## Вспомогательные классы

Впомогательные классы – классы необходимые для реализации некоторо функционала приложения. Всего таких классов 5, и это: BasketManager, ImageManager, LocalizationManager, PasswordHasher, TemplateValidator.

Класс BasketManager реализует управление корзиной товаров и наследуется от интерфейса IDisposable. Он содержит поле items, которое представляет список элементов корзины товаров (BasketItem). Также есть методы для добавления, удаления и очистки товаров в корзине, проверки и сохранения корзины в файл. Листинг класса предоставлен в Приложении Ж Листинг 1.

Каждый элемент корзины BasketItem может представлять либо обычный товар (Product), либо шаблон (Template). Он содержит соответствующий объект Product или Template и свойство IsTemplate, которое указывает, является ли элемент шаблоном.

Класс BasketManager также имеет статическое поле Instance, которое позволяет получить доступ к единственному экземпляру класса. Класс содержит метод CreateInstance, который создает экземпляр класса, если он еще не был создан.

Методы LoadBasket и SaveBasket используются для загрузки и сохранения данных корзины в файл, используя JSON-сериализацию.

В методе CheckItem проверяется, существуют ли все товары и шаблоны, находящиеся в корзине. Если какой-либо товар или шаблон был удален из базы данных, он также удаляется из корзины.

Структура BasketItem содержит информацию об элементе корзины, его типе (IsTemplate) и соответствующем товаре или шаблоне. Она также имеет два конструктора, которые используются для создания новых элементов корзины.

Класс ImageManager содержит методы для сохранения, загрузки и получения изображений. Листинг класса предосталвел в Приложении Ж Листинг 4.

Конструктор этого класса проверяет, существует ли папка "Images" и, если она не существует, создает ее.

Метод SaveImage сохраняет изображение из указанного пути в папку "Images", если оно там еще не существует. Метод возвращает имя файла.

Метод LoadImage загружает изображение с указанным именем из папки "Images" и возвращает объект ImageSource. Если изображение не существует, метод возвращает null.

Метод GetNoImage возвращает изображение "no-photos.png", находящееся в папке "Resources". Если изображение не удается загрузить, метод возвращает null.

Класс LocalizationManager отвечает за локализацию приложения. Листинг класса предосталвел в Приложении Ж Листинг 2.

В нем определены статический конструктор и конструктор экземпляра, свойства Instance и Current, а также методы SetLang и CreateInstance.

Статический конструктор инициализирует два объекта-блокировщика. Конструктор экземпляра получает объект Application, перебирает все свойства классов Eng и Rus, начинающиеся с "Loc", и добавляет их в ресурсы приложения с инициализирующим значением "INIT". Затем определяется текущий язык (английский или русский) и устанавливается соответствующий набор строк ресурсов.

Метод SetLang принимает объект перечисления Language, устанавливает его в качестве текущего языка и обновляет соответствующие ресурсы приложения.

Метод CreateInstance создает единственный экземпляр класса LocalizationManager при помощи двойной блокировки и инициализирует его при передаче объекта Application.

PasswordHasher представляет статический класс, который содержит два метода: GetHash и Compare для работы с хешированием паролей. Код предоставлен на рисунке 3.8.

public static class PasswordHasher

{

public const string Salt = "PCBUILDER";

public static string GetHash(string password)

{

using (var sha256 = SHA256.Create())

{

byte[] bytes = Encoding.UTF8.GetBytes(Salt + password);

byte[] hash = sha256.ComputeHash(bytes);

return Convert.ToBase64String(hash);

}

}

public static bool Compare(string hash, string password)

{

string passHash = GetHash(password);

return passHash == hash;

}

}

Рисунок 3.8 класс PasswordHasher

Константа Salt представляет строку-соль, которая будет добавлена к паролю при хешировании для увеличения безопасности.

Метод GetHash получает на вход строку с паролем, добавляет к нему строку-соль и применяет SHA256-алгоритм хеширования. Полученный хеш преобразуется в строку в формате Base64 и возвращается.

Метод Compare получает на вход хеш и пароль. Внутри метода вызывается метод GetHash с переданным паролем. Полученный хеш сравнивается со входным хешем и результат сравнения возвращается в виде булевого значения. Если хеши совпадают, то пароль считается верным, иначе - неверным.

Класс TemplateValidator используется для проверки совместимости компонентов в сборке ПК. Листинг класса предосталвел в Приложении Ж Листинг 3.

Метод Validate() итерирует список products и для каждого продукта извлекает из базы данных все компоненты производительности, относящиеся к этому продукту. Затем он разделяет эти компоненты на две группы: главные и зависимые. Главные компоненты являются теми, которые не зависят от других компонентов, тогда как зависимые компоненты зависят от главных.

Далее, метод проходится по каждому главному компоненту и проверяет, есть ли у него зависимые компонент с тем же тегом. Если такой компонент существует, то проверяется, имеют ли оба компонента одно и то же значение производительности. Если значения различаются, то создается экземпляр TemplateValidateError, содержащий сообщение об ошибке и тип продукта, к которому относится компонент, и этот экземпляр добавляется в список ошибок. Если после проверки список ошибок пуст, значит сборка совместима, и метод возвращает пустой список ошибок.

Структура TemplateValidateError используется для представления ошибок проверки. Она содержит сообщение об ошибке и тип продукта, к которому относится компонент, вызвавший ошибку.

## Тестирование и анализ полученных результатов

## Тестирование

Тестирование является очень важной частью разработки приложения так как без тестирования не возможно понять будет ли хорошо работать приложение во время эксплуатации каким-либо пользователем. Поэтому тестирование должно быть тщательно спланировано и организовано, чтобы обеспечить полное и точное тестирование каждого взаимодействия пользователя с приложением. Вот некоторые из ключевых компонентов, которые должны быть включены в такой раздел:

Подготовка тестовых данных: важно подготовить тестовые данные, которые будут использоваться для проверки каждого взаимодействия. 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 я провел ряд тестов на реальных данных, в которое входит регистрация пользователя, выбор товара и покупка, создания сборки ПК.

Описание тестовых случаев для регистрации:

1. Пользователь уже существует.
2. Не корректная электронная почта.
3. Не корректный контактный номер.

Таблица 4.1 описание тестов для регистрации

|  |
| --- |
| Описание тестовых случаев |
| a |
| b |
| c |

Описание тестовых случаев для выбора товара и покупки:

1. Добавление товара в корзину и удаление его из базы данных.
2. Оформление заказа и удаление товара из заказа из базы данных.

Таблица 4.2 описание тестов для выбора и покупки товара

|  |
| --- |
| Описание тестовых случаев |
| a |
| b |

Описание тестовых случаев для создания сборки ПК:

1. Оставить поле названия сборки пустым.
2. Оставить одно или более ячеек пустыми.
3. Поставить не совместимые комплектующие.

Таблица 4.3 описание тестов для создания сборки ПК

|  |
| --- |
| Описание тестовых случаев |
| a |
| b |
| c |

Проверка сценариев ошибок:

Для регистрации:

Таблица 4.4 результат тестов для регистрации

|  |
| --- |
| a |
| b |
| c |

Для выбора товара и покупки:

Таблица 4.5 результат тестов для выбора и покупки товара

|  |
| --- |
| a |
| b |

Для создания сборки ПК:

Таблица 4.6 результат тестов для создания сборки ПК

|  |
| --- |
| a |
| b |
| c |

На основе проведенных тестов можно сделать следующие выводы:

Тестирование регистрация пользователя показало, что процесс работает корректно и пользователь успешно может зарегистрироваться в системе.

Тестирование выбор товара и покупки показало, что если товар будет удалён из каталога, то у всех пользователей товар пропадёт из корзины и из истории заказов во избежание ошибок.

Тестирование создания сборки ПК показало, что процесс работает корректно и пользователь успешно может создать свою сборку.

## Руководство по установке и использованию программного обеспечения

## Установка приложения

Для установки и использования программного обеспечения на технологии .NET Framework, WPF, MSSQL.

Для установки и использования программного обеспечения вам потребуется:

* Операционная система Windows.
* .NET Framework 4.9 SDK или выше.
* MSSQL Server

Таблица 5.1 – Шаги по запуску приложения

|  |  |
| --- | --- |
| Действия | Шаги |
| Склонируйте репозиторий приложения | 1 |
| Настройте строку подключения в  PCBuilder.exe.config под себя | 2 |
| Запустите MSSQL server | 3 |
| Запустите приложение | 4 |

В файле PCBuilder.exe.config надо будет изменить строчку «connectionString» вы должны будите в пункте «Data Source» установить название вашего sql сервера, а также убедитесь, что ваш sql сервер имеет пользователя с такими данными как «User ID=ADMIN; Password=1234».

Следуя это инструкциии, вы сможете установить и запустить приложение, а также использовать его для своих целей.

## Использование приложения

Ниже представлено подробное руководство пользователя для работы с приложением.

После входа в приложение и после подключения к sql серверу вы попадёте вы окно авторизации. Если у вас есть аккаунт, то вам потребуется ввести почту и пароль. Если же у вас нету аккаунта, то следует нажать текстовую кнопку «регистрация». Окно авторизации продемонстрированно на рисунке 5.1. Так же вы можете войти в систему как администратор, написав в качестве почты и пароля «ADMIN».



Рисунок 5.1 окно авторизации

В окне регистрации вам необходимо ввести свою электронную почту, пароль, имя, фамилию, адрес доставки, контактный номер. Электронная почта должна быть заполнена в формате «(текст)@(почта).(домен)». Контактный телефон должен быть заполнен в формате «+(код)(номер)». После чего вам нужно нажать кнопку подтвердить, если всё хорошо выскочит сообщение об успехе операции, иначе высветятся ошибки валидации. Так же вы можете вернуться на окно авторизации через тактовую кнопку. Окно авторизации продемонстрированно на рисунке 5.2.

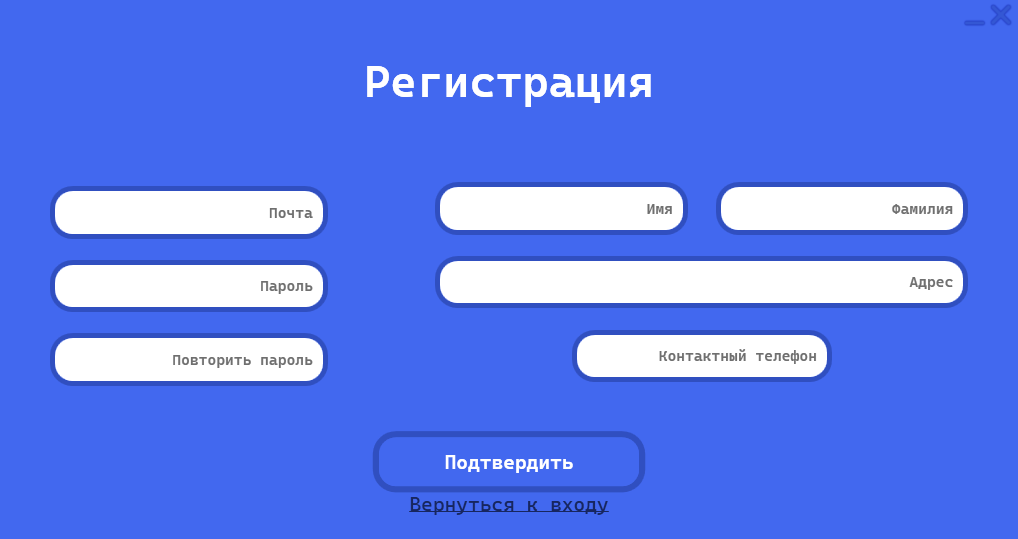


Рисунок 5.2 окно регистрации

После авторизации вы попадаете на главное окно на страницу каталога. Далее идёт функционал любого интернет-магазина. Далее будет описываться функционал страниц.

На странице «Каталог» вы можете устанавливать фильтры, сбрасывать фильтры, а также искать интересующий вас товар по названию через текстовый поисковик. Существует два фильтра, по цене, где нужно указать диапазон цен, и по типу товара. Вы можете нажать на карточку товара что бы попасть на страницу с его детальным описанием. Так же вы можете добавить товар в корзину нажав на соответствующую кнопку. Страница продемонстрирована на рисунке 5.3.



Рисунок 5.3 страница «каталог»

На странице «Корзина» вам показан список ваших товаров. Вы можете удалить товар из корзины нажав на красную кнопку или же открыть страницу этого товара нажав на синюю кнопку. Так же вы можете оформить заказ написав к нему заметку и нажав на кнопку подтвердить. Страница продемонстрирована на рисунке 5.4.

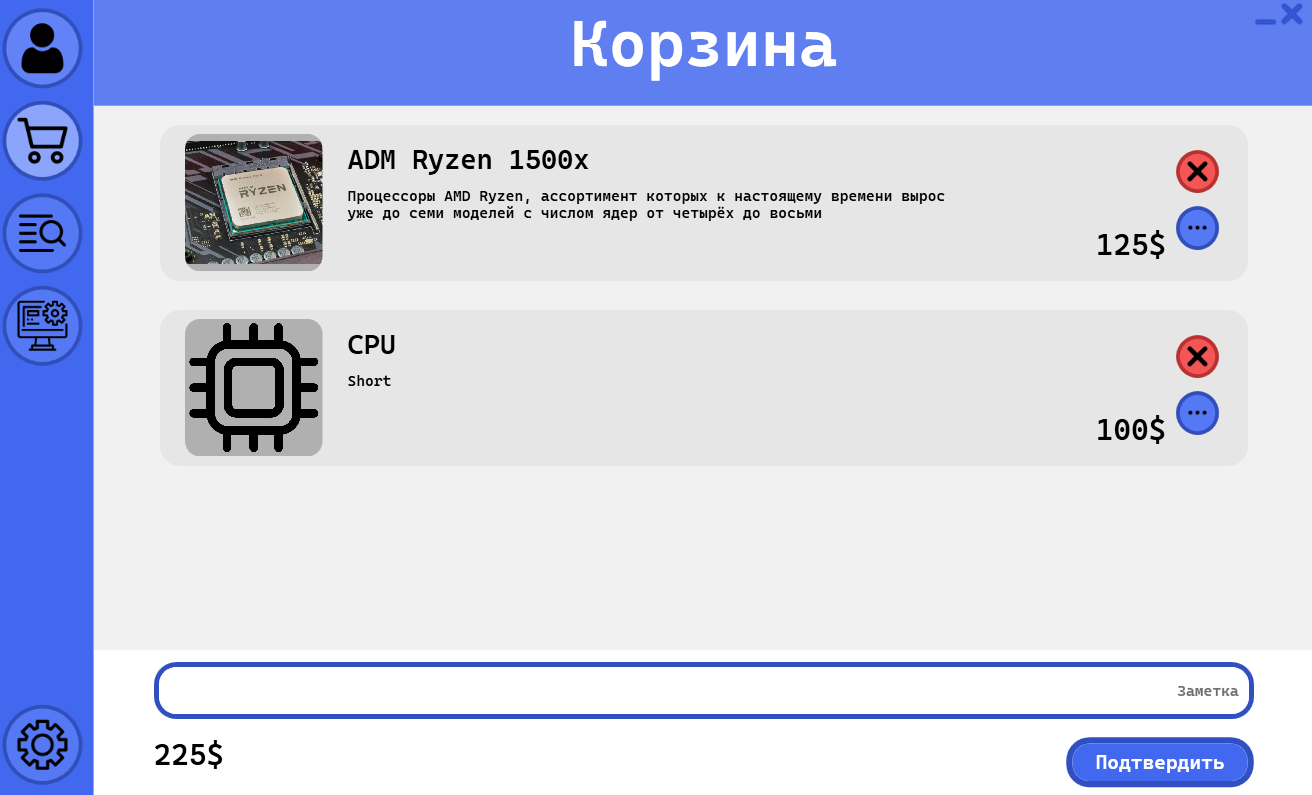


Рисунок 5.4 страница «Корзина»

На странице «Пользователь» вы лишь можете просматривать информацию о вашем аккаунте, а также просматривать историю заказов. Страница продемонстрирована на рисунке 5.5.

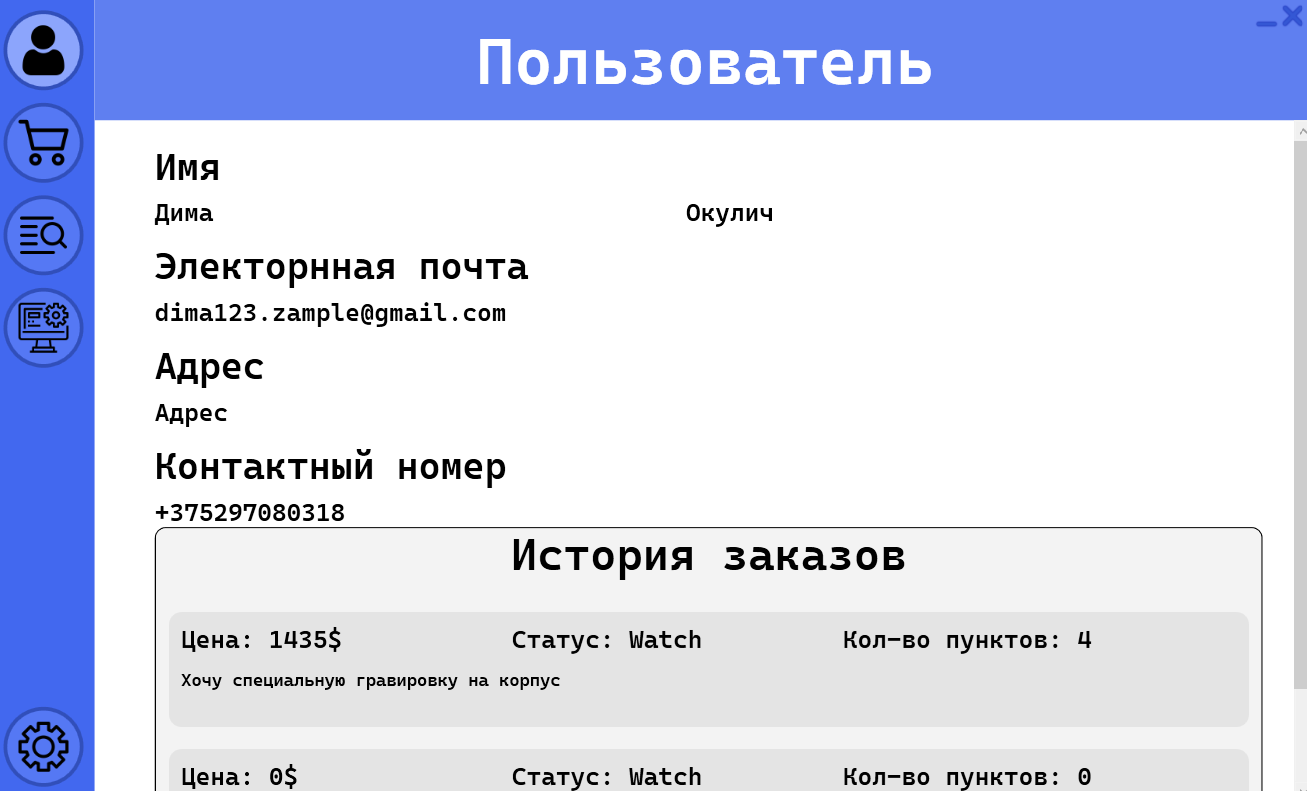


Рисунок 5.5 страница «Пользователь»

На странице «Настройки» вы лишь сможете поменять язык интерфейса приложения на Русский или Английский. Страница продемонстрирована на рисунке 5.6.

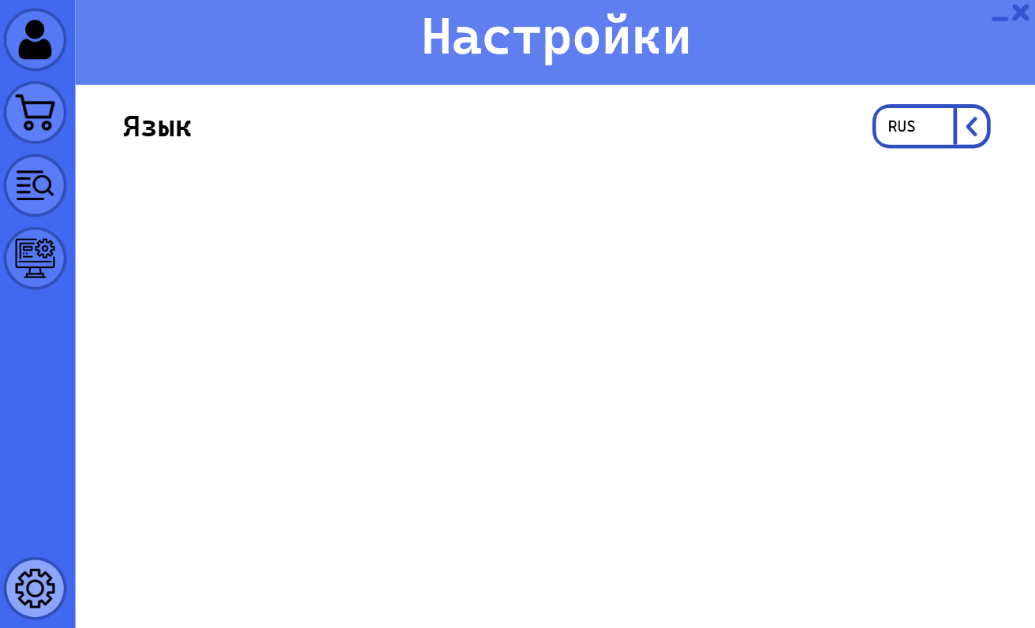


Рисунок 5.6 страница «Настройки»

На странице «Сборки» вы можете просматривать сборки ПК, которые сделали другие люди вы можете увидеть из цену, а также кто их сделал. Вы можете, как и в каталоге искать сборки по названию, а также вы можете выбрать что бы отображались только ваши или же еще и чужие сборки. Вы можете добавить сборку в корзину или посмотреть из каких компонентов она состоит. А также вы можете создать свою сборку ПК нажав кнопку «создать» тогда вы перейдёте на страницу «Редактор сборок» где вы и будите создавать сборку. Страница продемонстрирована на рисунке 5.7.

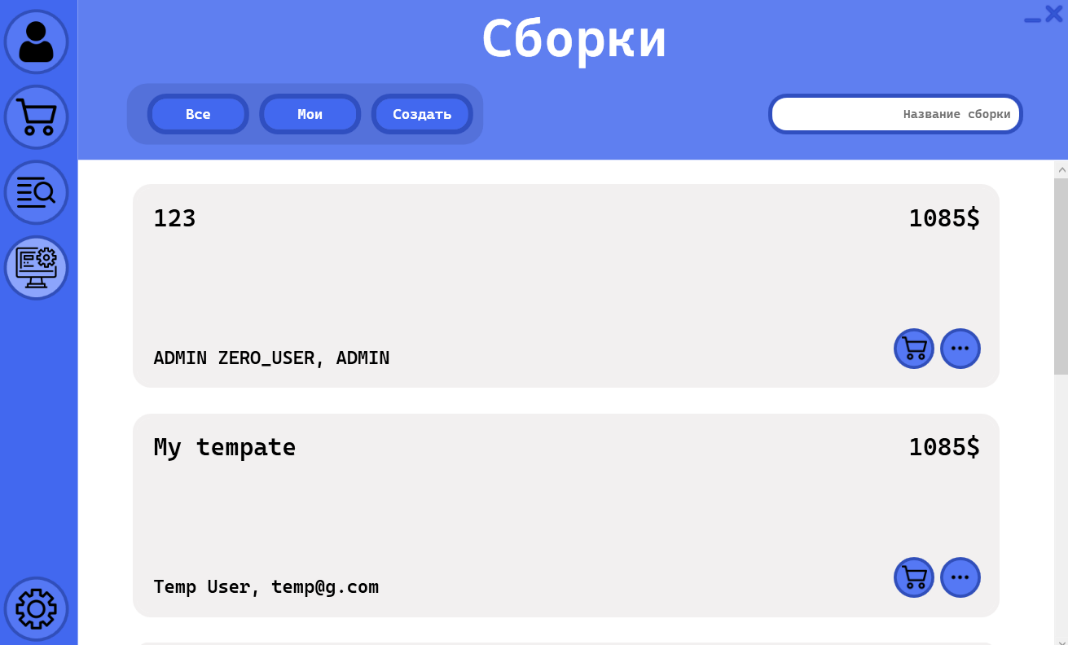


Рисунок 5.7 страница «Сборки»

На странице «Редактор сборок» вы можете создать собственную сборку ПК из комплектующих которые есть в каталоге. Для того что бы создать свою сборку надо заполнить поле имя, описание. Далее требуется заполнять ячейки комплектующих иконка ячейки символизирует под какой тип ячейка. У ячейки есть две кнопки одна отвечает за открытие страницы товара, положенного в эту ячейку, а другая открывает каталог с предустановленным фильтром по типу. При подборе комплектующих будьте внимательны все зависимые свойства комплектующих должны быть одинаковые (например: сокет, формфактор и т.д.). Однако если вы выберете не совместимый компонент, то при подтверждении приложение вас уведомит об этом. Для подтверждения нужно нажать на соответствующую кнопку. Страница продемонстрирована на рисунке 5.8.

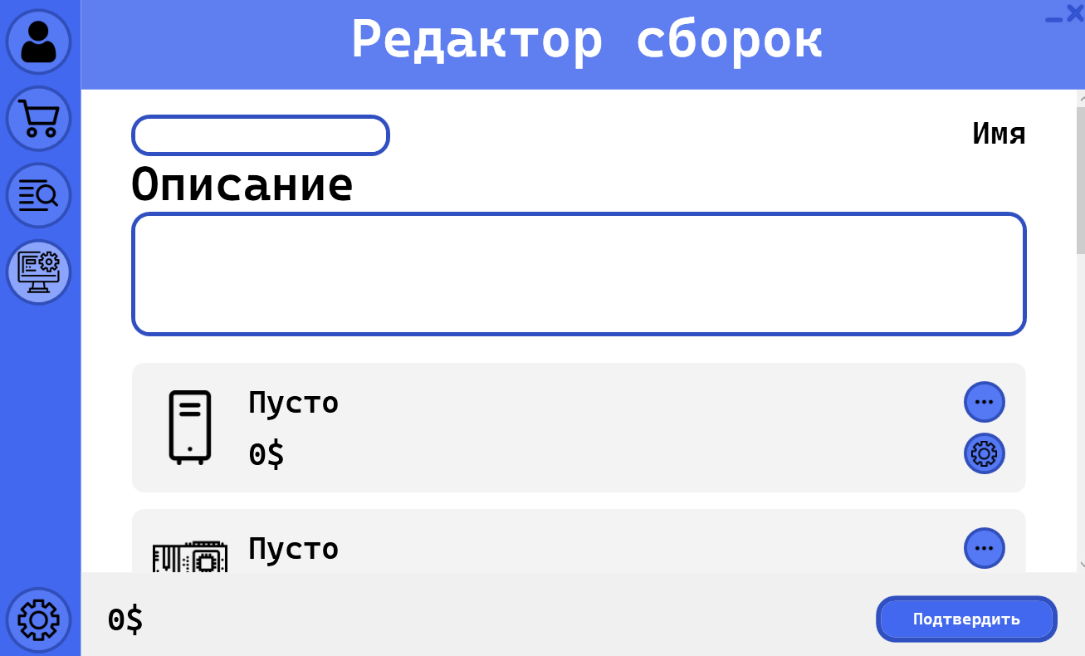


Рисунок 5.8 страница «Редактор сборок»

Для успешного использования приложения рекомендуется использовать современные версии всех технологий и библиотек для повышения безопасности и стабильности работы системы. Необходимо следить за актуальностью обновлений и патчей для используемых технологий и библиотек.

## Заключение

В заключении хочется отметить, что приложений дающих возможность создавать свои сборки ПК и выкладывать их в сеть мало. Однако это очень облегчает сборку свое ПК, а люди, которые об этом мало осведомлены могут найти себе сборку по желанию.

В данном проекте были использованы следующие технологии: .NET Framework 4.9, MSSQL server, WPF. Эти технологии позволили реализовать множество функциональных возможностей, таких как аутентификация и авторизация пользователей, использование каталог, корзины, создание сборок ПК.

При разработке программного средства были выполнены все пункты из указанного списка предполагаемого основного функционала приложения, а именно:

* Функции администратора:
  + Управление базой данных;
  + Управление каталогом комплектующих ПК;
* Функции клиента:
  + Выполнять регистрацию и авторизацию;
  + Просмотр каталога комплектующих;
  + Заполнять форму заказа по заданным формам;
  + Создание/Удаление шаблонов сборок ПК;
  + Просмотр каталога шаблонов сборок ПК;

По результатам тестирования было выявлено, что функционал работает корректно и взаимодействует друг с другом без каких-либо проблем.

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 целом, разработка приложения на базе .NET Framework WPF является сложным и многогранным процессом, но в результате позволяет создавать мощные и гибкие приложения, которые могут эффективно решать широкий спектр задач.

## Список использованных источников

1. LikedIn [Электронный ресурс] Метод доступа: [https://www.linkedin.com](https://www.linkedin.comleclick.ru) Дата доступа: 14.05.2023

2. Руководство по WPF [Электронный ресурс] – Режим доступа: <https://metanit.com/sharp/wpf/#google_vignette> – Дата доступа: 14.05.2023.

3. Руководство по Entity Framework [Электронный ресурс] – Режим доступа: <https://metanit.com/sharp/efcore/> – Дата доступа: 14.05.2023.

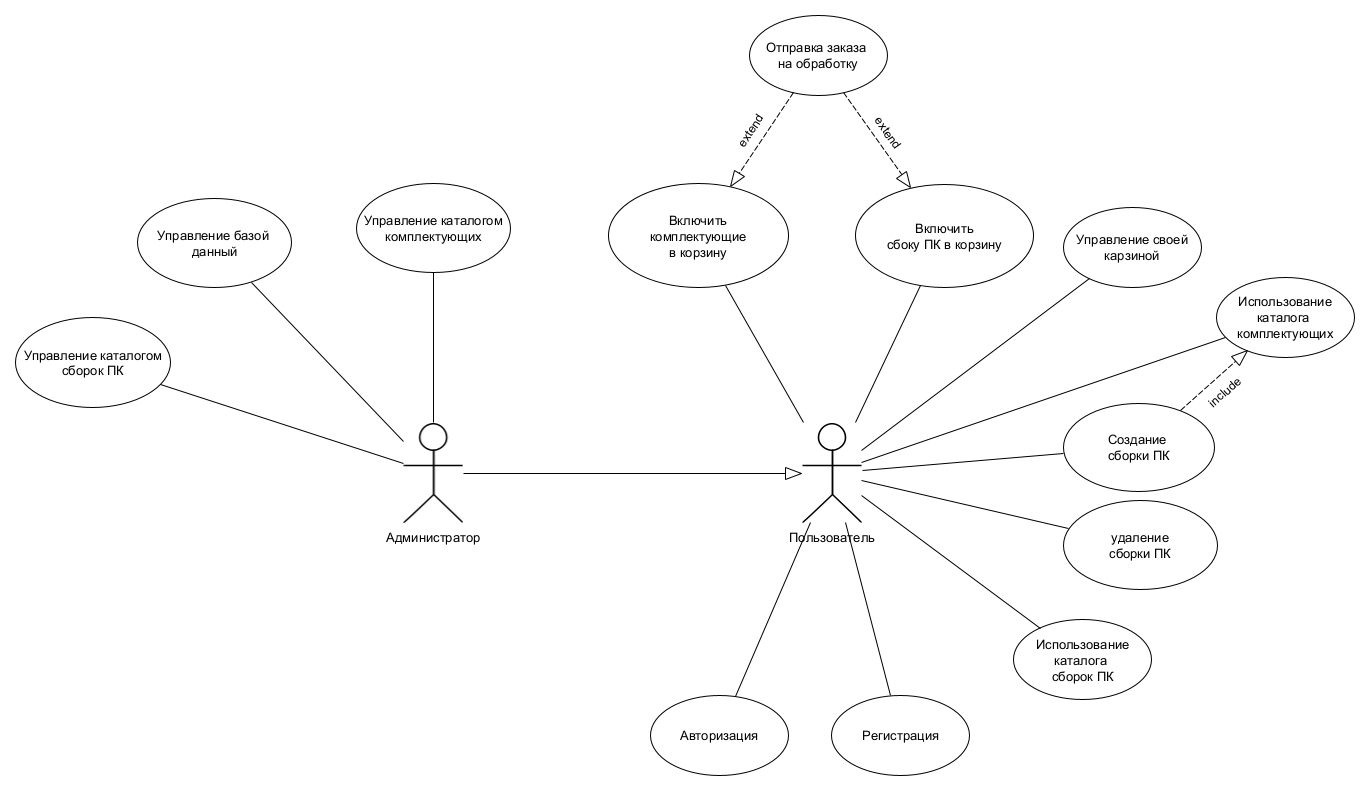
4. Связи между таблицами базы данных [Электронный ресурс] – Режим доступа: <https://habr.com/ru/post/488054/> – Дата доступа: 18.04.2023.

5. DataBase First [Электронный ресурс] – Режим доступа: <https://docs.microsoft.com/en-us/ef/ef6/modeling/designer/workflows/database-first> – Дата доступа: 14.05.2023.

6. Регулярные выражения [Электронный ресурс] – Метод доступа: <https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expressions> Дата доступа: 14.05.2023.

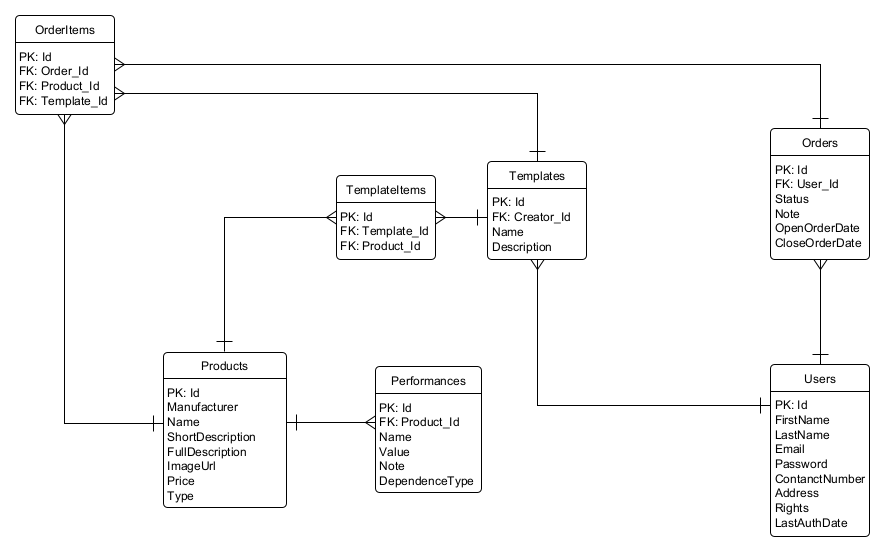
## ПРИЛОЖЕНИЕ А

**Use-case диаграмма**



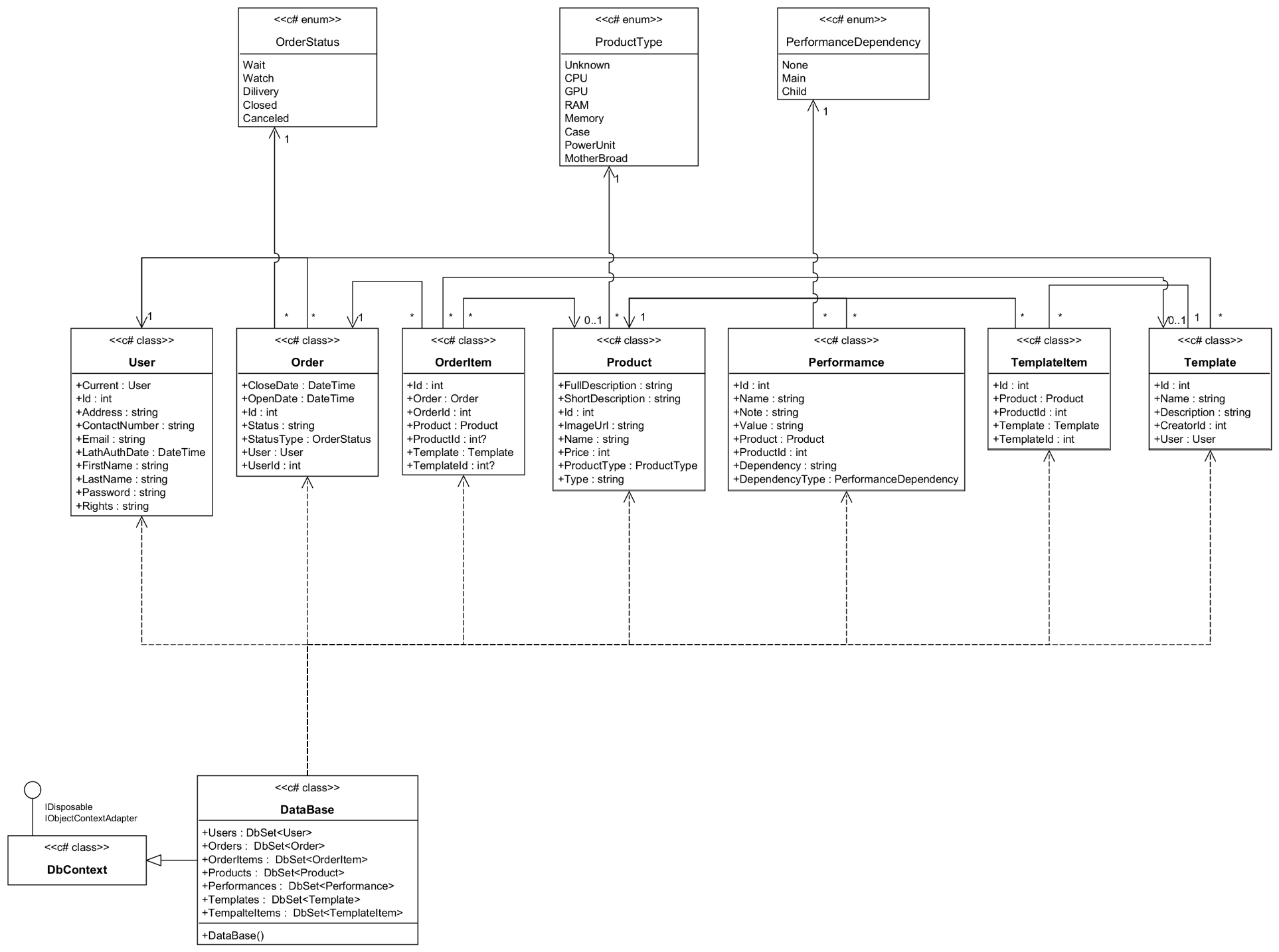
## ПРИЛОЖЕНИЕ Б

**Диаграмма базы данных**



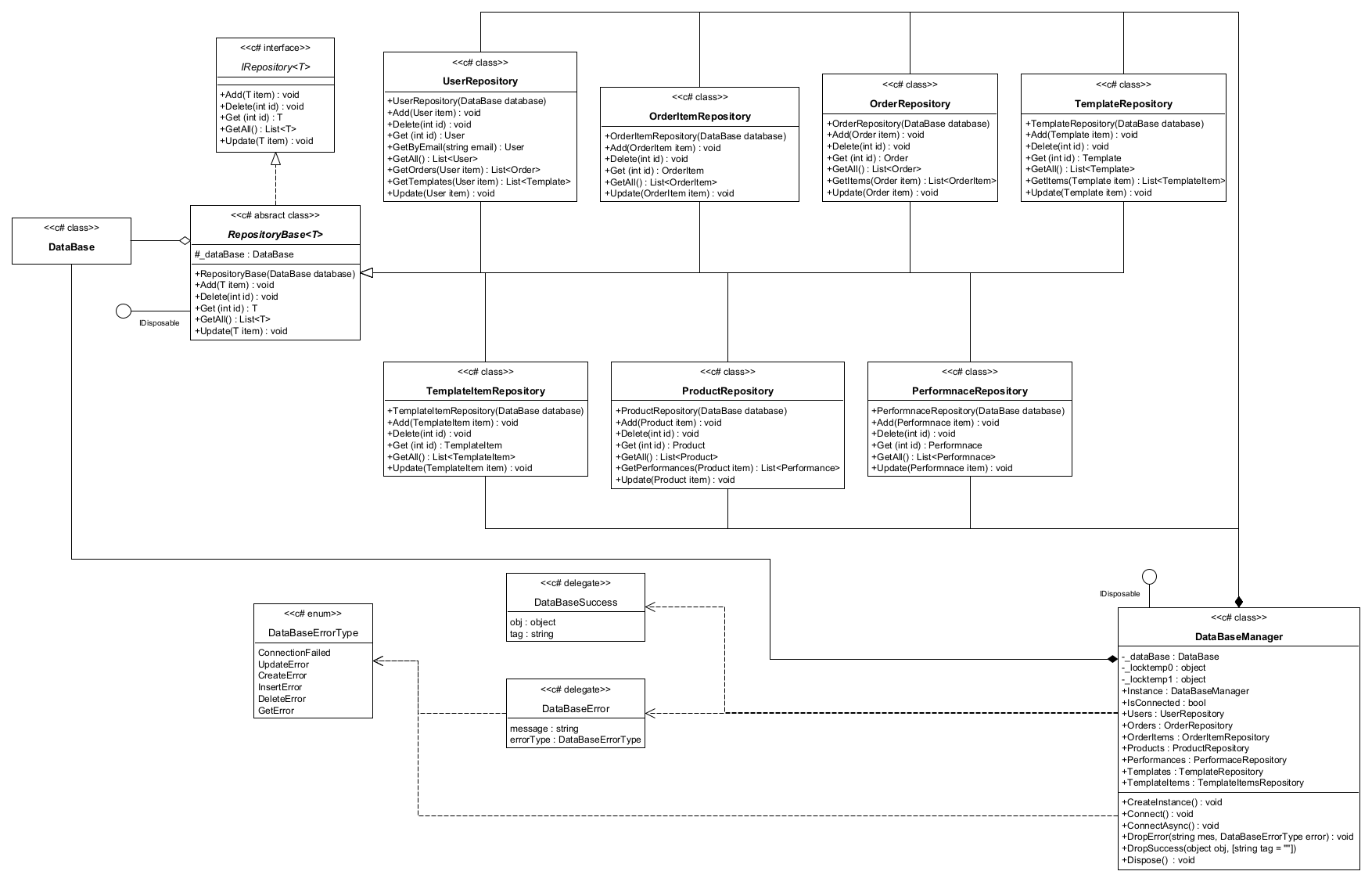
## ПРИЛОЖЕНИЕ В

**Диаграмма классов для моделей**



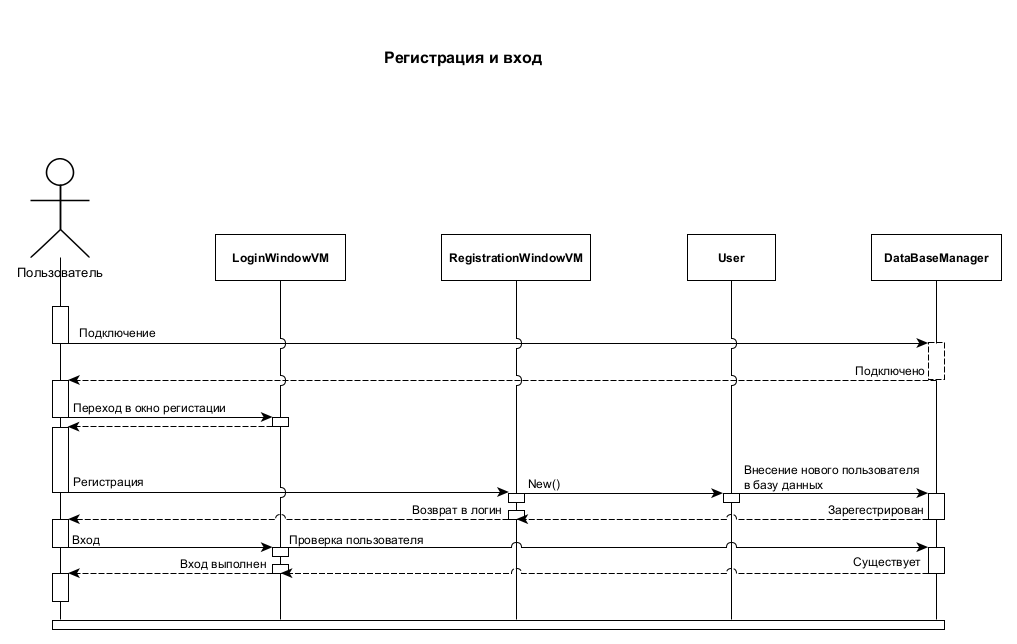
## ПРИЛОЖЕНИЕ Г

**Диаграмма классов репозитория**



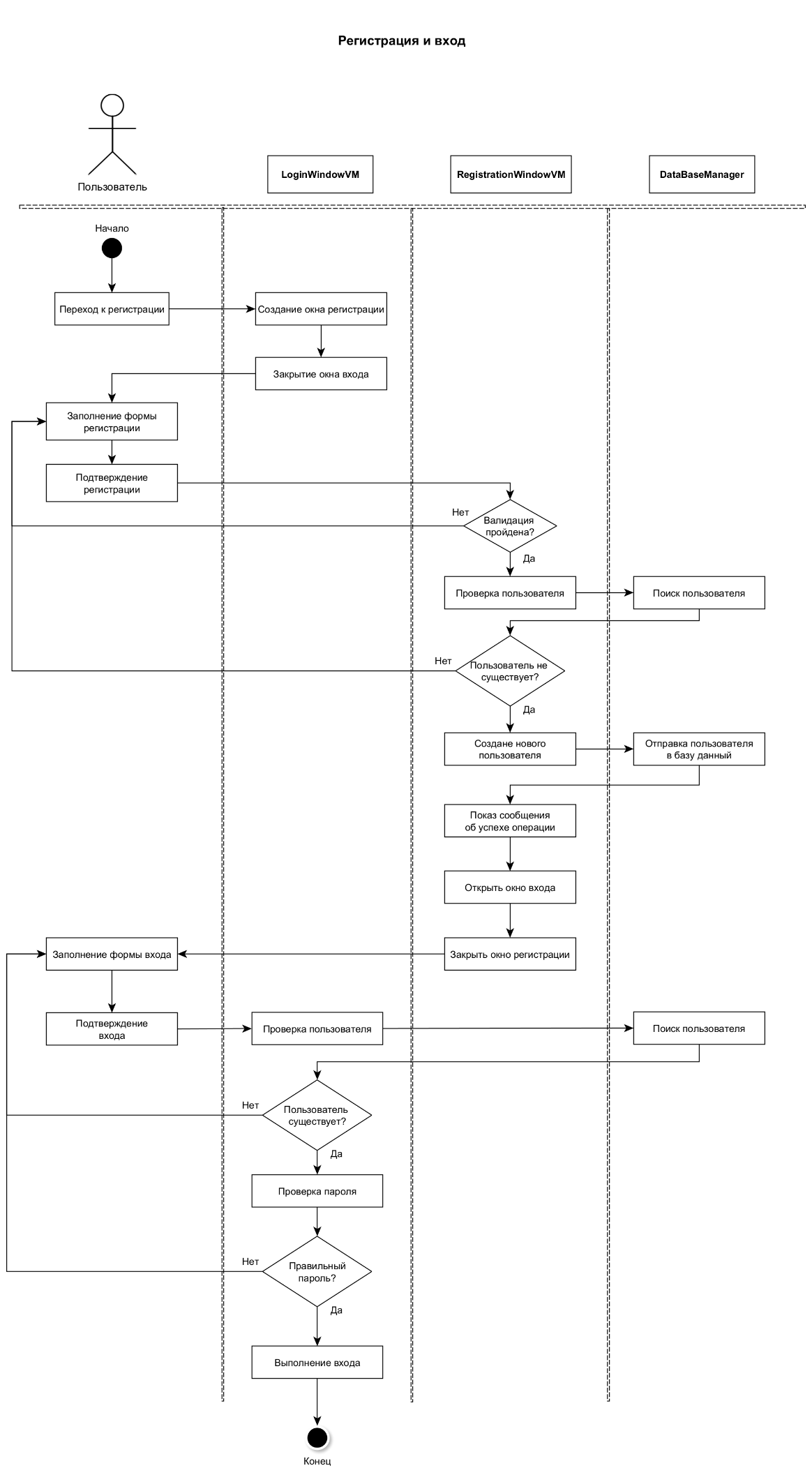
## ПРИЛОЖЕНИЕ Д

**Диаграмма последовательности для регистрации и авторизации**



## ПРИЛОЖЕНИЕ Е

**Диаграмма состояний для регистрации и авторизации**



## ПРИЛОЖЕНИЕ Ё

**Листинг 1 - С# - код репозитория пользователей**

public class UserRepository : RepositoryBase<User>

{

public UserRepository(DataBase dataBase) : base(dataBase)

{

}

public override void Add(User item)

{

try

{

\_dataBase.Users.Add(item);

\_dataBase.SaveChanges();

DataBaseManager.Instance.DropSuccess(null, "User\_Add");

}

catch (SystemException Error)

{

DataBaseManager.Instance.DropError(Error.Message, DataBaseErrorType.CreateError);

}

}

public override void Delete(int id)

{

try

{

\_dataBase.Users.Remove(\_dataBase.Users.First(i => i.Id == id));

\_dataBase.SaveChanges();

DataBaseManager.Instance.DropSuccess(null, "User\_Delete");

}

catch (SystemException Error)

{

DataBaseManager.Instance.DropError(Error.Message, DataBaseErrorType.CreateError);

}

}

public override User Get(int id)

{

User result = null;

try

{

result = \_dataBase.Users.Find(id);

DataBaseManager.Instance.DropSuccess(null, "User\_Get");

}

catch (SystemException Error)

{

DataBaseManager.Instance.DropError(Error.Message, DataBaseErrorType.CreateError);

}

return result;

}

public override List<User> GetAll()

{

List<User> result = null;

try

{

result = \_dataBase.Users.ToList();

DataBaseManager.Instance.DropSuccess(null, "User\_GetAll");

}

catch (SystemException Error)

{

DataBaseManager.Instance.DropError(Error.Message, DataBaseErrorType.CreateError);

}

return result;

}

public List<Order> GetOrders(User user)

{

List<Order> result = null;

try

{

result = \_dataBase.Orders.Where(i => i.UserId == user.Id).ToList();

DataBaseManager.Instance.DropSuccess(null, "User\_GetOrders");

}

catch (SystemException Error)

{

DataBaseManager.Instance.DropError(Error.Message, DataBaseErrorType.CreateError);

}

return result;

}

public List<Template> GetTemplates(User user)

{

List<Template> result = null;

try

{

result = \_dataBase.Templates.Where(i => i.CreatorId == user.Id).ToList();

DataBaseManager.Instance.DropSuccess(null, "User\_GetTemplates");

}

catch (SystemException Error)

{

DataBaseManager.Instance.DropError(Error.Message, DataBaseErrorType.CreateError);

}

return result;

}

public override void Update(User item)

{

try

{

User edit = \_dataBase.Users.Find(item.Id);

edit.Address = item.Address;

edit.Password = item.Password;

edit.Email = item.Email;

edit.FirstName = item.FirstName;

edit.LastName = item.LastName;

edit.LastAuthDate = item.LastAuthDate;

\_dataBase.SaveChanges();

DataBaseManager.Instance.DropSuccess(null, "User\_Update");

}

catch (SystemException Error)

{

DataBaseManager.Instance.DropError(Error.Message, DataBaseErrorType.CreateError);

}

}

}

**Листинг 2 – С# - код менеджера базы данных**

public delegate void DataBaseError(string message, DataBaseErrorType errorType);

public delegate void DataBaseSuccess(object obj, string tag);

public class DataBaseManager : IDisposable

{

public static DataBaseManager Instance { get; private set; }

private static object \_locktemp0, \_locktemp1;

public event Action OnConnected;

public event DataBaseError OnError;

public event DataBaseSuccess OnSuccess;

private DataBase \_dataBase;

public bool IsConnected { get; private set; }

public UserRepository Users { get; private set; }

public OrderRepository Orders { get; private set; }

public OrderItemRepository OrderItems { get; private set; }

public ProductRepository Products { get; private set; }

public PerformanceRepository Performances { get; private set; }

public TemplateRepository Templates { get; private set; }

public TemplateItemRepository TemplateItems { get; private set; }

public static void CreateInstance()

{

\_locktemp0 = new object();

\_locktemp1 = new object();

lock (\_locktemp0)

{

if (Instance == null)

{

lock (\_locktemp1)

{

Instance = new DataBaseManager();

}

}

}

}

public void Connect()

{

try

{

\_dataBase = new DataBase();

\_dataBase.Database.Initialize(false);

if (\_dataBase.Users.FirstOrDefault(i => i.FirstName == "ADMIN") == null)

{

User user = new User()

{

Id = -1,

FirstName = "ADMIN",

LastName = "ZERO\_USER",

LastAuthDate = DateTime.Now,

Email = "ADMIN",

Address = "ADMIN",

Rights = "ADMIN",

Password = PasswordHasher.GetHash("ADMIN")

};

\_dataBase.Users.Add(user);

\_dataBase.SaveChanges();

}

Users = new UserRepository(\_dataBase);

Orders = new OrderRepository(\_dataBase);

OrderItems = new OrderItemRepository(\_dataBase);

Products = new ProductRepository(\_dataBase);

Performances = new PerformanceRepository(\_dataBase);

Templates = new TemplateRepository(\_dataBase);

TemplateItems = new TemplateItemRepository(\_dataBase);

IsConnected = true;

OnConnected?.Invoke();

}

catch (SystemException Error)

{

OnError?.Invoke(Error.Message, DataBaseErrorType.ConnectionFailed);

}

}

public async void ConnectAsync()

{

try

{

\_dataBase = new DataBase();

// Initialize of database

string result = await Task.Run(() =>

{

try

{

\_dataBase.Database.Initialize(false);

return "Ok";

}

catch (SystemException Error)

{

return Error.Message;

}

});

if (result != "Ok")

throw new SystemException(result);

if (\_dataBase.Users.FirstOrDefault(i => i.FirstName == "ADMIN") == null)

{

User user = new User()

{

Id = -1,

FirstName = "ADMIN",

LastName = "ZERO\_USER",

LastAuthDate = DateTime.Now,

Email = "ADMIN",

Address = "ADMIN",

Rights = "ADMIN",

Password = PasswordHasher.GetHash("ADMIN")

};

\_dataBase.Users.Add(user);

await \_dataBase.SaveChangesAsync();

}

Users = new UserRepository(\_dataBase);

Orders = new OrderRepository(\_dataBase);

OrderItems = new OrderItemRepository(\_dataBase);

Products = new ProductRepository(\_dataBase);

Performances = new PerformanceRepository(\_dataBase);

Templates = new TemplateRepository(\_dataBase);

TemplateItems = new TemplateItemRepository(\_dataBase);

IsConnected = true;

OnConnected?.Invoke();

}

catch (SystemException Error)

{

OnError?.Invoke(Error.Message, DataBaseErrorType.ConnectionFailed);

}

}

public void Dispose()

{

\_dataBase? Dispose();

}

public void DropError(string message, DataBaseErrorType errorType)

{

if (errorType == DataBaseErrorType.ConnectionFailed)

IsConnected = false;

OnError?.Invoke(message, errorType);

}

public void DropSuccess(object obj, string note = "")

{

OnSuccess?.Invoke(obj, note);

}

}

public enum DataBaseErrorType

{

ConnectionFailed, UpdateError, CreateError, InsertError, DeleteError, GetError

}

## ПРИЛОЖЕНИЕ Ж

**Листинг 1 – С# - код класса BasketManager.**

public class BasketManager : IDisposable

{

public const string BasketsFolder = "Baskets";

private static object \_temp0;

private static object \_temp1;

public static BasketManager Instance;

private JsonSerializerOptions \_serializer;

private List<BasketItem> items;

public List<BasketItem> Items { get { return items; } }

public string BasketPath { get => Path.Combine(BasketsFolder,

$"{OwnerName}.json"); }

public string OwnerName { get; private set; }

public static void CreateInstance(string userName)

{

\_temp0 = new object();

\_temp1 = new object();

lock (\_temp0)

{

if (Instance == null)

{

lock (\_temp1)

{

Instance = new BasketManager(userName);

}

}

}

}

public BasketManager(string userName)

{

if (!Directory.Exists(BasketsFolder))

Directory.CreateDirectory(BasketsFolder);

items = new List<BasketItem>();

\_serializer = new JsonSerializerOptions()

{

IncludeFields = true,

WriteIndented = true,

};

OwnerName = userName.Replace('.', 'D');

LoadBasket();

}

private void LoadBasket()

{

if (!File.Exists(BasketPath))

return;

string json = File.ReadAllText(BasketPath);

items = JsonSerializer.Deserialize<List<BasketItem>>(json, \_serializer);

CheckItem();

}

private void SaveBasket()

{

if (File.Exists(BasketPath))

File.Delete(BasketPath);

string json = JsonSerializer.Serialize(items, typeof(List<BasketItem>),

\_serializer);

File.WriteAllText(BasketPath, json);

}

public void CheckItem()

{

for (int i = 0; i < items.Count; i++)

{

if (items[i].IsTemplate)

{

if (DataBaseManager.Instance.Templates.Get(items[i].Template.Id)

== null)

{

items.Remove(items[i]);

i--;

}

}

else

{

if (DataBaseManager.Instance.Products.Get(items[i].Product.Id)

== null)

{

items.Remove(items[i]);

i--;

}

}

}

}

public void Clear() => items.Clear();

public void Add(Product product) => items.Add(new BasketItem(product));

public void Add(BasketItem item) => items.Add(item);

public void Add(Template template) => items.Add(new BasketItem(template));

public void Remove(Product product) => items.Remove(items.First(

i => i.Product == product));

public void Remove(Template template) => items.Remove(items.First(

i => i.Template == template));

public void Remove(BasketItem item) => items.Remove(item);

public void Dispose()

{

SaveBasket();

}

}

[Serializable]

public struct BasketItem

{

public bool IsTemplate { get; set; }

public Product Product { get; set; }

public Template Template { get; set; }

public BasketItem(Product product)

{

Product = product;

Template = null;

IsTemplate = false;

}

public BasketItem(Template template)

{

Product = null;

Template = template;

IsTemplate = true;

}

}

**Листинг 2 – С# - код LocalizationManager**

public class LocalizationManager

{

private static object \_temp0;

private static object \_temp1;

public static LocalizationManager Instance { get; private set; }

private Application \_app;

public Language Current { get; private set; }

static LocalizationManager()

{

\_temp0 = new object();

\_temp1 = new object();

}

public LocalizationManager(Application app)

{

\_app = app;

foreach (var item in typeof(Eng).GetProperties())

{

if (item.Name.StartsWith("Loc"))

{

\_app.Resources.Add(item.Name, "INIT");

}

}

string culture = CultureInfo.CurrentCulture.Name;

Language cur = culture == "ru-RU" || culture == "ru-BY" ? Language.RUS : Language.ENG;

Current = cur;

SetLang(cur);

}

public void SetLang(Language language)

{

Current = language;

PropertyInfo[] properties;

switch (language)

{

case Language.ENG:

properties = typeof(Eng).GetProperties();

break;

case Language.RUS:

properties = typeof(Rus).GetProperties();

break;

default:

properties = typeof(Eng).GetProperties();

break;

}

foreach (var item in properties)

{

if (item.Name.StartsWith("Loc"))

{

\_app.Resources[item.Name] = (string)item.GetValue(typeof(Eng));

}

}

}

public static void CreateInstance(Application app)

{

lock (\_temp0)

{

if (Instance == null)

{

lock (\_temp1)

{

Instance = new LocalizationManager(app);

}

}

}

}

}

public enum Language

{

ENG, RUS

}

**Листинг 3 – C# - код TemplateValidator**

public class TemplateValidator

{

private List<Product> products;

public TemplateValidator(List<Product> products)

{

this.products = products;

}

public List<TemplateValidateError> Validate()

{

List<Performance> mainPerformances = new List<Performance>();

List<Performance> childPerformances = new List<Performance>();

List<TemplateValidateError> errors = new List<TemplateValidateError>();

foreach (Product product in products)

{

foreach (var item in

DataBaseManager.Instance.Products.GetPerformances(product))

{

switch (item.DependencyType)

{

case PerformanceDependency.Main:

mainPerformances.Add(item);

break;

case PerformanceDependency.Child:

childPerformances.Add(item);

break;

}

}

}

foreach (var item in mainPerformances)

{

foreach (var item2 in childPerformances)

{

if (item.Tag == item2.Tag)

{

if (item.Value != item2.Value)

errors.Add(

new TemplateValidateError($"{Application.Current.

Resources["Loc\_TempEdit\_ValidateErrMesTemp"]} \"{item.Name}\"!",

item2.Product.ProductType));

}

}

}

return errors;

}

}

public struct TemplateValidateError

{

public string Message { get; private set; }

public ProductType ProductType { get; private set; }

public TemplateValidateError(string message, ProductType productType)

{

Message = message;

ProductType = productType;

}

}

**Листинг 4 – C# - код класса ImageManager**

public static class ImageManager

{

static ImageManager()

{

if (!Directory.Exists("Images"))

Directory.CreateDirectory("Images");

}

public static string SaveImage(string path)

{

string name = Path.GetFileName(path);

if (File.Exists(Path.Combine("Images", name)))

return name;

File.Copy(path, Path.Combine("Images", name));

return name;

}

public static ImageSource LoadImage(string name)

{

string exePath = System.Reflection.Assembly.GetExecutingAssembly().

Location;

string imageUri = Path.Combine(Path.GetDirectoryName(exePath), "Images",

name);

if (!File.Exists(imageUri))

return null;

BitmapImage image = new BitmapImage();

image.BeginInit();

image.UriSource = new Uri(imageUri, UriKind.Absolute);

image.EndInit();

return image;

}

public static ImageSource GetNoImage()

{

BitmapImage image = new BitmapImage();

image.BeginInit();

image.UriSource = new Uri("/Resources/no-photos.png", UriKind.Relative);

image.EndInit();

return image;

}

}