**4.1. Анализ исходных данных на курсовое проектирование.**

**4.1.1. Анализ исходных данных к курсовому проекту.**

**Исходные данные к проекту:**

**Тема проекта:** *«Программное средство «Магазин музыкальных инструментов»».*

**Описание системы** – *Программа предоставляет возможность приобрести музыкальные инструменты и аксессуары. Есть возможность создать аккаунт. Администратор способен добавлять, удалять, редактировать товары, устанавливать цену и скидки, назначать других администраторов. Пользователь способен выбирать товары, добавлять в корзину и оформлять заказ. Есть возможность оставить отзыв.*

**Назначение системы** – *Цифровизация деятельности магазина музыкальных инструментов. Мобильное приложение.*

**Язык и среда программирования** – С#, SQLite, Microsoft Visual Studio 2022

Программное средство «*Магазин музыкальных инструментов*» является мобильным приложением, предоставляющим функциональность магазина музыкальных инструментов в цифровом виде. Пользовательские роли представлены администратором, зарегистрированным и незарегистрированным пользователем/покупателем. В возможности администратора входит добавление товаров, их редактирование и удаление, а также возможность устанавливать цену и скидку на товар, подтверждать других администраторов, отвечать на отзывы. Пользователь способен просматривать товары, добавлять их в корзину и оформлять заказ, а также оставлять отзывы. Информация о товарах и отзывы хранятся в базе данных и подгружается при запуске программы.

Основным языком разработки приложения является С#. Этим обуславливается выбор среды разработки, а именно Microsoft Visual Studio. Данная среда предоставляет полный инструментарий для работы с проектами на C#, как с консольными, так и с проектами с визуальным интерфейсом, которым и будет обладать данное программное средство.

**4.1.2. Обоснование и описание выбора языка программирования, средств разработки, используемых технологий и сторонних библиотек.**

На сегодняшний момент объектно-ориентированный язык программирования C# - один из самых мощных, быстро развивающихся и востребованных языков в ИТ-отрасли. В настоящий момент на нем пишутся самые различные приложения: от небольших десктопных программ до крупных веб-порталов и веб-сервисов, обслуживающих ежедневно миллионы пользователей.

C# уже не молодой язык и, как и вся платформа .NET уже прошёл большой путь. Первая версия языка вышла вместе с релизом Microsoft Visual Studio .NET в феврале 2002 года. Текущей версией языка является версия C# 10.0, которая вышла 8 ноября 2021 года вместе с релизом .NET 6. За это время язык значительно развился и собрал вокруг себя большое сообщество разработчиков. C# является языком с Си-подобным синтаксисом и близок в этом отношении к C++ и Java. Поэтому, если имеется опыт программирования на C++ и/или Java, то овладеть C# становится не настолько сложной задачей.

C# является объектно-ориентированным и в этом плане много перенял у Java и С++. Например, C# поддерживает полиморфизм, наследование, перегрузку операторов, статическую типизацию. [Типы, допускающие значение null,](https://learn.microsoft.com/ru-ru/dotnet/csharp/nullable-references) обеспечивают защиту от переменных, которые не ссылаются на выделенные объекты. [Обработка исключений](https://learn.microsoft.com/ru-ru/dotnet/csharp/fundamentals/exceptions/) предоставляет структурированный и расширяемый подход к обнаружению ошибок и восстановлению после них. [Лямбда-выражения](https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/lambda-expressions) поддерживают приемы функционального программирования. [Синтаксис LINQ](https://learn.microsoft.com/ru-ru/dotnet/csharp/linq/) создает общий шаблон для работы с данными из любого источника. Поддержка языков для [асинхронных операций](https://learn.microsoft.com/ru-ru/dotnet/csharp/programming-guide/concepts/async/) предоставляет синтаксис для создания распределенных систем. В C# имеется [Единая система типов](https://learn.microsoft.com/ru-ru/dotnet/csharp/fundamentals/types/). Все типы C#, включая типы-примитивы, такие как int и double, наследуют от одного корневого типа object. Все типы используют общий набор операций, а значения любого типа можно хранить, передавать и обрабатывать схожим образом. Более того, C# поддерживает как определяемые пользователями [ссылочные типы](https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/builtin-types/reference-types), так и [типы значений](https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/builtin-types/value-types). C# позволяет динамически выделять объекты и хранить упрощенные структуры в стеке. C# поддерживает универсальные методы и типы, обеспечивающие повышенную безопасность типов и производительность. C# предоставляет итераторы, которые позволяют разработчикам классов коллекций определять пользовательские варианты поведения для клиентского кода.

C# поддерживает Управление версиями, чтобы обеспечить совместимость программ и библиотек с течением времени. Вопросы управления версиями существенно повлияли на такие аспекты разработки C#, как раздельные модификаторы virtual и override, правила разрешения перегрузки методов и поддержка явного объявления членов интерфейса.

Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений. И C# продолжает активно развиваться, и с каждой новой версией появляется все больше интересной функциональности.

Язык C# был создан специально для работы и неразрывно связан с фреймворком .NET. Фреймворк .NET представляет мощную платформу для создания приложений. Можно выделить следующие ее основные черты:

Поддержка нескольких языков.

Основой платформы является общеязыковая среда исполнения Common Language Runtime (CLR), благодаря чему .NET поддерживает несколько языков: наряду с C# это также VB.NET, C++, F#, а также различные диалекты других языков, привязанные к .NET, например, Delphi.NET. При компиляции код на любом из этих языков компилируется в сборку на общем языке CIL (Common Intermediate Language) - своего рода ассемблер платформы .NET. Поэтому при определенных условиях мы можем сделать отдельные модули одного приложения на отдельных языках.

.NET является переносимой платформой (с некоторыми ограничениями). Например, последняя версия платформы на данный момент - .NET 6 поддерживается на большинстве современных ОС Windows, MacOS, Linux. Используя различные технологии на платформе .NET, можно разрабатывать приложения на языке C# для самых разных платформ - Windows, MacOS, Linux, Android, iOS, Tizen.

.NET представляет единую для всех поддерживаемых языков библиотеку классов. И вне зависимости от создаваемого приложения, будь то текстовый редактор, чат или сложный веб-сайт - так или иначе задействуется библиотека классов .NET. Эти библиотеки поддерживают множество различных рабочих нагрузок. Они упорядочены по пространствам имен, которые предоставляют разные полезные возможности: от операций файлового ввода и вывода до управления строками и синтаксического анализа XML, от платформ веб-приложений до элементов управления Windows Forms. Обычно приложение C# активно используют библиотеку классов .NET для решения типовых задач.

Общеязыковая среда исполнения CLR и базовая библиотека классов являются основой для целого стека технологий, которые разработчики могут задействовать при построении тех или иных приложений. Например, для работы с базами данных в этом стеке технологий предназначена технология ADO.NET и Entity Framework Core. Для построения графических приложений с богатым насыщенным интерфейсом - технология WPF и WinUI, для создания более простых графических приложений - Windows Forms. Для разработки кроссплатформенных мобильных и десктопных приложений - Xamarin/MAUI. Для создания веб-сайтов и веб-приложений - ASP.NET и т.д.

К этому стоит добавить активной развивающийся и набирающий популяность Blazor - фреймворк, который работает поверх .NET и который позволяет создавать веб-приложения как на стороне сервера, так и на стороне клиента. А в будущем будет поддерживать создание мобильных приложений и, возможно, десктоп-приложений.

Согласно ряду тестов веб-приложения на .NET 6 в ряде категорий сильно опережают веб-приложения, построенные с помощью других технологий. Приложения на .NET 6 в принципе отличаются высокой производительностью.

Также еще следует отметить такую особенность языка C# и фреймворка .NET, как автоматическая сборка мусора. А это значит, что в большинстве случаев не приходится, в отличие от С++, заботиться об освобождении памяти. Вышеупомянутая общеязыковая среда CLR сама вызовет сборщик мусора и очистит память.

Стоит отметить, что .NET долгое время развивался преимущественно как платформа для Windows под названием .NET Framework. В 2019 вышла последняя версия этой платформы - .NET Framework 4.8. Она больше не развивается. С 2014 Microsoft стал развивать альтернативную платформу - .NET Core, которая уже предназначалась для разных платформ и должна была вобрать в себя все возможности устаревшего .NET Framework и добавить новую функциональность. Текущей версией является .NET 6.

Нередко приложение, созданное на C#, называют управляемым кодом (managed code). Это значит, что данное приложение создано на основе платформы .NET и поэтому управляется общеязыковой средой CLR, которая загружает приложение и при необходимости очищает память. Но есть также приложения, например, созданные на языке С++, которые компилируются не в общий язык CIL, как C#, VB.NET или F#, а в обычный машинный код. В этом случае .NET не управляет приложением. В то же время платформа .NET предоставляет возможности для взаимодействия с неуправляемым кодом.

Как писалось выше, код на C# компилируется в приложения или сборки с расширениями exe или dll на языке CIL. Далее при запуске на выполнение подобного приложения происходит JIT-компиляция (Just-In-Time) в машинный код, который затем выполняется. При этом, поскольку приложение может быть большим и содержать множество инструкций, в текущий момент времени будет компилироваться лишь та часть приложения, к которой непосредственно идет обращение. Если мы обратимся к другой части кода, то она будет скомпилирована из CIL в машинный код. При том уже скомпилированная часть приложения сохраняется до завершения работы программы. В итоге это повышает производительность.

Технология WPF.

Windows Presentation Foundation является частью экосистемы платформы .NET и представляет собой подсистему для построения графических интерфейсов.

Если при создании традиционных приложений на основе WinForms за отрисовку элементов управления и графики отвечали такие части ОС Windows, как User32 и GDI+, то приложения WPF основаны на DirectX. В этом состоит ключевая особенность рендеринга графики в WPF: используя WPF, значительная часть работы по отрисовке графики, как простейших кнопок, так и сложных 3D-моделей, ложиться на графический процессор на видеокарте, что также позволяет воспользоваться аппаратным ускорением графики.

Одной из важных особенностей является использование языка декларативной разметки интерфейса XAML, основанного на XML: вы можете создавать насыщенный графический интерфейс, используя или декларативное объявление интерфейса, или код на управляемых языках C#, VB.NET и F#, либо совмещать и то, и другое.

Преимущества WPF

Использование традиционных языков .NET-платформы - C#, F# и VB.NET для создания логики приложения

Возможность декларативного определения графического интерфейса с помощью специального языка разметки XAML, основанном на xml и представляющем альтернативу программному созданию графики и элементов управления, а также возможность комбинировать XAML и C#/VB.NET.

Независимость от разрешения экрана: поскольку в WPF все элементы измеряются в независимых от устройства единицах, приложения на WPF легко масштабируются под разные экраны с разным разрешением.

Богатые возможности по созданию различных приложений: это и мультимедиа, и двухмерная и трехмерная графика, и богатый набор встроенных элементов управления, а также возможность самим создавать новые элементы, создание анимаций, привязка данных, стили, шаблоны, темы и многое другое

Аппаратное ускорение графики - вне зависимости от того, работаете ли вы с 2D или 3D, графикой или текстом, все компоненты приложения транслируются в объекты, понятные Direct3D, и затем визуализируются с помощью процессора на видеокарте, что повышает производительность, делает графику более плавной.

Создание приложений под множество ОС семейства Windows