





## Реферат

Курсовая работа: 72 страниц, 23 рисунка, 2 листинга, 7 источников, 2 приложения

WPF, MVVM, XML, C#, ООП, OBJ, MTL, UV-РАЗВЕРТКА, DIRECT X, VISUAL STUDIO, 3D МОДЕЛЬ, SOLID, ПРЕДМЕТ ГАРДЕРОБА

В данном курсовом проекте рассмотрены вопросы разработки desktop-приложения для подбора образа одежды. Программное обеспечение создано на базе Microsoft Visual Studio. Реализация системы выполнена при помощи языка программирования C# и языка разметки XML. Реализованы классы, определены поля и основные операции классов. Описана внутренняя структура программы и взаимосвязи между классами. Определены наборы тестовых данных, на базе которых были написаны модульные тесты.

## Содержание

Содержание .....	4
Введение .....	5
2 Техническое задание .....	7
2.1 Наименование программы .....	7
2.2 Краткая характеристика области применения .....	7
2.3 Основания для разработки .....	7
2.4 Требования к составу выполняемых функций.....	7
3 Теоретическое описание решения .....	8
4 Описание архитектуры системы.....	11
5 Используемые библиотеки и методы программирования .....	16
6 Реализация программного продукта .....	17
6.1 Описание реализации модулей, классов, методов и алгоритмов, используемых при решении задачи. ....	17
6.2 Описание тестовых планов и наборов тестов .....	23
6.3 Результаты тестирования программы.....	28
Заключение .....	31
Список использованных источников .....	32
Приложение А .....	33
Приложение Б .....	72

## **Введение**

Подбор образа одежды — это важный этап при выборе стиля и создании своего уникального образа. Однако, не всегда возможно примерить на себя все интересующие нас варианты одежды.

Цель данной курсовой работы - разработка программного продукта, который позволит пользователям создавать свои образы с использованием 3D графики.

Основные задачи курсовой работы:

1. Изучение и описание методов создания виртуальных моделей и примерки одежды с использованием 3D графики.
2. Проектирование приложения, создание UML диаграмм.
3. Написание тестов для проверки работы созданной библиотеки.
4. Разработка интерфейса программы для удобного пользования библиотекой.

Программный продукт, разработанный в рамках данной курсовой работы, может быть полезен для онлайн-магазинов, которые продают одежду и хотят предоставить своим клиентам возможность примерки виртуально, а также для дизайнеров одежды, которые хотят создавать виртуальные модели своих коллекций.

## **1 Нормативные ссылки**

В настоящих методических указаниях использованы ссылки на следующие нормативные документы:

ГОСТ 7.82-2001 СИБИД. Библиографическая запись. Библиографическое описание электронных ресурсов. Общие требования и правила составления

ГОСТ 7.9-95 СИБИД. Реферат и аннотация. Общие требования

ГОСТ 19.105-78 ЕСПД. Общие требования к программным документам

ГОСТ 19.401-78 ЕСПД. Текст программы. Требования к содержанию и оформлению

ГОСТ 19.402-78 ЕСПД. Описание программы

ГОСТ 19.404-79 ЕСПД. Пояснительная записка. Требования к содержанию и оформлению

## **2 Техническое задание**

### **2.1 Наименование программы**

Наименование – «Your Own Stylist»

### **2.2 Краткая характеристика области применения**

Программа позволяет создавать и подбирать образ с помощью библиотеки моделей предметов гардероба, материалов и цветов формата RGBA.

### **2.3 Основания для разработки**

Функциональным назначением программы является предоставление пользователю возможности подбирать виртуальный образ максимально интуитивным и понятным способом.

### **2.4 Требования к составу выполняемых функций**

Программа должна обеспечивать возможность выполнения перечисленных ниже функций:

- перемещение виртуальной камеры внутри окон просмотра моделей;
- выбор позы и пола манекена;
- переключение отображения манекена;
- смена освещения, окружения и поверхности;
- выбор моделей одежды для просмотра образа;
- установка параметров для подбора наряда;
- выбор одежды в главном окне;
- изменение материала выбранного предмета;
- выбор цвета и прозрачности материала выбранного предмета;
- применение настроек материала.

### 3 Теоретическое описание решения

Для достижения максимальной наглядности в отображении моделей одежды, было принято решение использовать в приложении 3D-графику. В данной работе будут рассмотрены вопросы, связанные с использованием формата OBJ, MTL, кубических карт и UV-развертки в Direct X.

Формат OBJ является одним из наиболее распространенных форматов 3D-моделей, который поддерживается многими программами для создания и редактирования 3D-графики. Он содержит информацию о геометрии модели, текстурах, материалах и других свойствах (Рисунок 1).

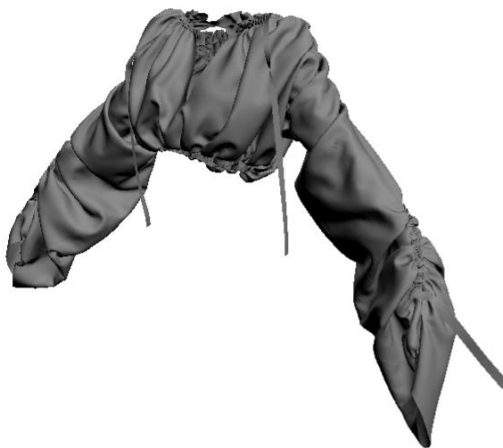


Рисунок 1 – Пример объекта формата .obj

Файлы формата MTL содержат информацию о материалах, используемых в 3D-модели. Они могут содержать информацию о цвете, текстурах, прозрачности и других свойствах материала. Файлы MTL обычно используются вместе с файлами OBJ для определения внешнего вида модели.

Кубические карты (Cube Maps) являются специальным типом текстур, которые используются для создания реалистичных отражений и окружающей среды в 3D-графике. Они состоят из шести текстур, каждая из которых представляет собой изображение, снятое с разных сторон объекта (Рисунки 2 и 3).



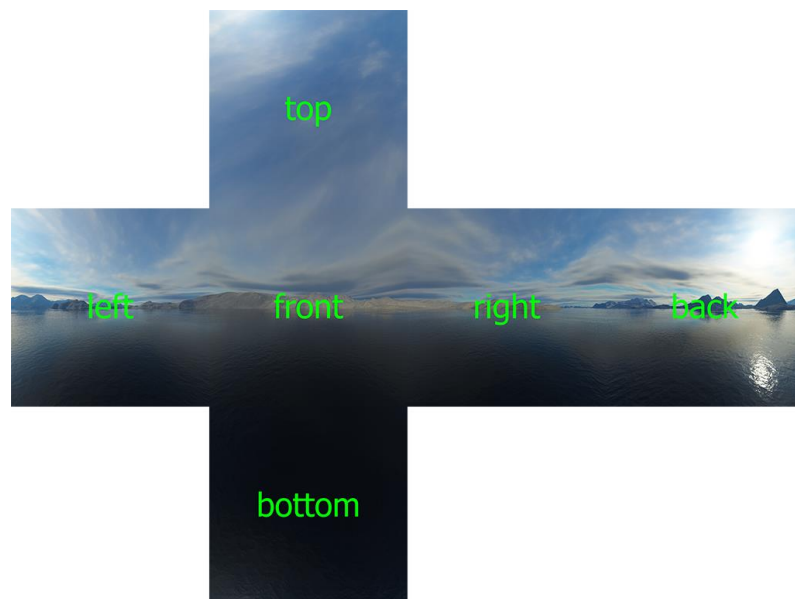


Рисунок 2 – Пример кубической карты

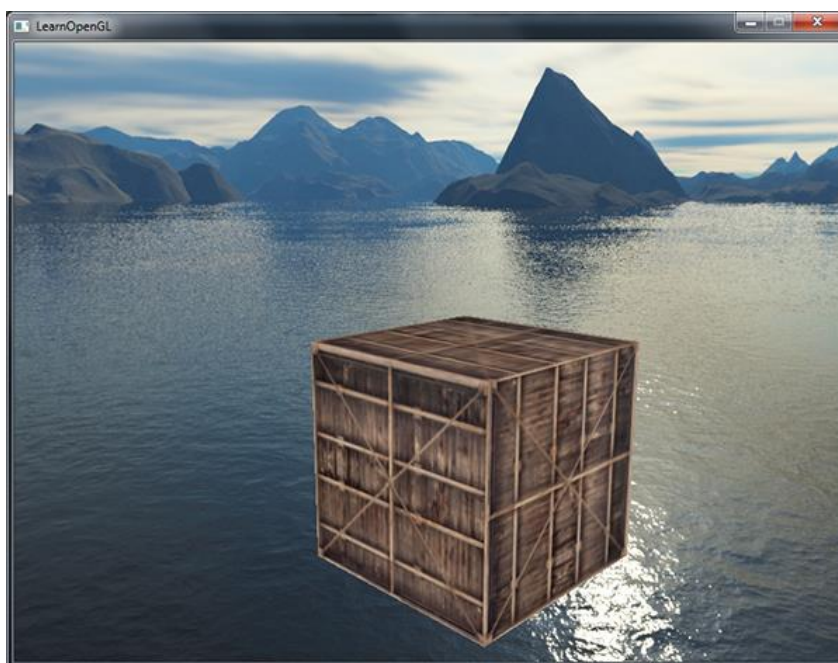


Рисунок 3 – Пример кубической карты

UV-развертка (UV Mapping) является процессом присвоения текстурным координатам вершинам 3D-модели. Она позволяет разместить текстуру на поверхности модели таким образом, чтобы она выглядела

реалистично и соответствовала форме объекта. UV-развертка может быть выполнена в различных программах для создания и редактирования 3D-графики, и она является важной частью процесса создания реалистичных 3D-моделей (Рисунок 4).

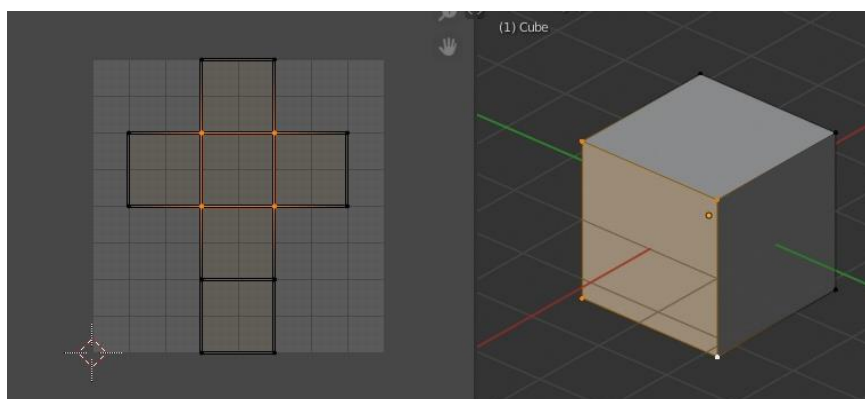


Рисунок 4 – Пример UV-развертки

DirectX представляет собой набор из нескольких API (application programming interface — интерфейс программирования приложений), позволяющих разработчикам игр и других интерактивных приложений получать доступ к специфическим функциям аппаратного обеспечения без необходимости написания аппаратнозависимого программного кода. DirectX основан на наборе интерфейсов COM или классов. COM расшифровывается, как Component Object Model (Компонентная Модель Объектов), и представляет собой спецификацию интерфейса, в котором функции вызываются через указатели. Таким образом, объекты COM могут описываться такими языками программирования, как C/C++, Delphi или даже Basic.

## 4 Описание архитектуры системы

В качестве архитектурного шаблона для приложения был выбран паттерн MVVM. Для его реализации был выбран фреймворк Windows Presentation Foundation, так как он наиболее подходит для реализации данного шаблона.

Паттерн Model-View-ViewModel (MVVM) — это один из паттернов проектирования семейства MVx, основанный на разделении приложения на три основных компонента: Model (Модель), View (Представление) и ViewModel (Модель-Представление). Каждый из этих компонентов отвечает за свою часть логики приложения и взаимодействует с другими компонентами через строго определенные интерфейсы.

Model (Модель) - компонент, который отвечает за управление данными и бизнес-логикой приложения. Он содержит данные, методы для работы с этими данными и правила, которые определяют, как эти данные могут быть изменены.

View (Представление) - компонент, который отвечает за отображение данных пользователю. Он содержит элементы управления, которые позволяют пользователю взаимодействовать с приложением.

ViewModel (Модель-Представление) - компонент, который связывает Model и View с помощью привязки данных (Binding), которая является встроенной в технологии WPF. Этот компонент содержит логику, которая позволяет отображать данные из Model в View и обрабатывать действия пользователя, которые влияют на данные в Model.

Вместе эти компоненты образуют архитектуру приложения, которая позволяет разделить логику приложения на отдельные компоненты и упростить его разработку и поддержку (Рисунок 5).

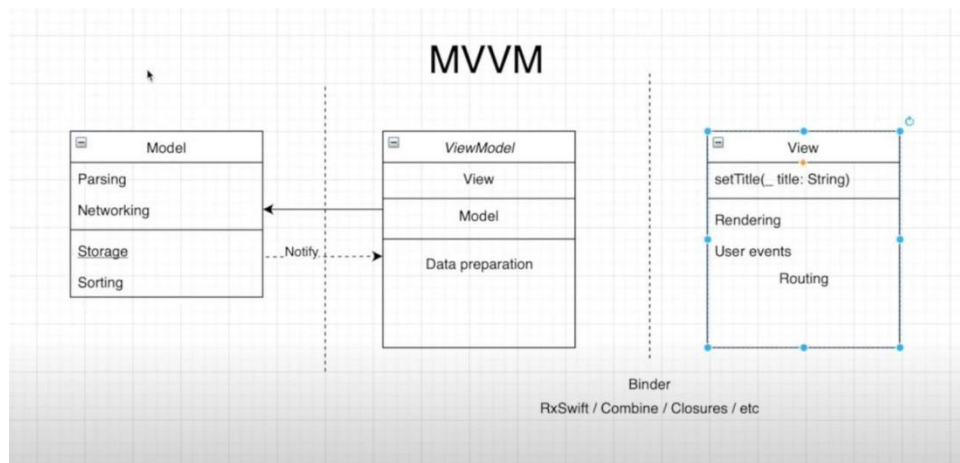


Рисунок 5 – Схема архитектурного паттерна MVVM

Вариант использования (use case) представляет собой последовательность действий, выполняемых системой в ответ на событие, инициализируемое некоторым внешним объектом.

Диаграмма вариантов использования для пользователя отображена на рисунке 6, она отражает все возможные взаимодействия пользователя с программой, ниже перечислены основные из них:

- собрать образ из выбранной одежды – выбор различных типов одежды из стандартной библиотеки;
- подобрать образ по указанным параметрам;
- сменить пол или гендер манекена;
- переключить видимость пола;
- сменить освещение и окружение сцены;
- выбрать модель в окне вьюпорта;
- осмотреть модель в окне предпросмотра;
- выбрать материал или цвет;
- применить настройки к модели в основном окне.

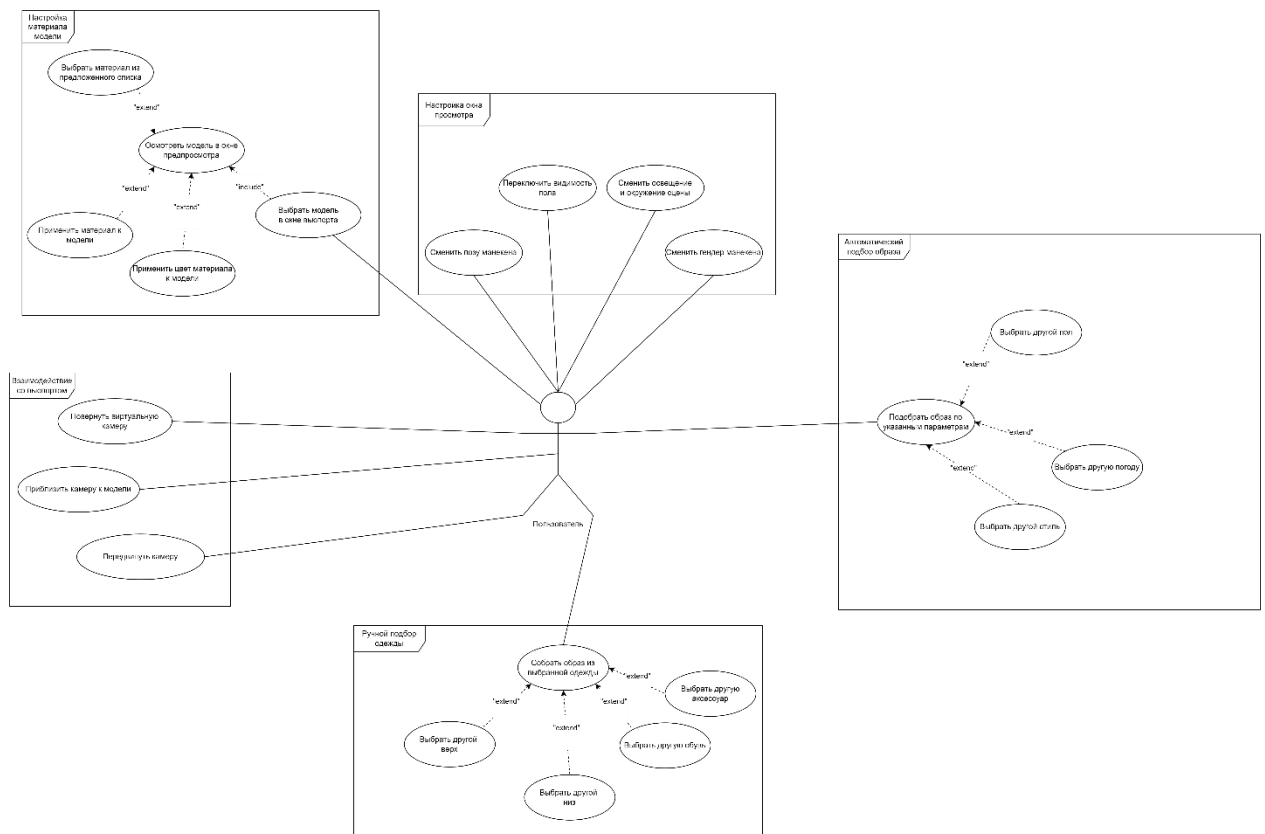


Рисунок 6 – Диаграмма вариантов использования

Диаграмма классов отображает общую структуру иерархии классов в системе, показывая их взаимодействие, атрибуты, методы, интерфейсы и взаимосвязи. Она демонстрирует общую организацию структуры классов, отображая различные классы в системе, их свойства и функции, а также взаимосвязь между классами.

Для проектирования приложения были применены паттерны «Многопоточный одиночка» и «Фабричный метод».

Фабричный метод был использован для реализации пяти фабрик, наследующих общий абстрактный класс. Фабрики создают различные типы предметов гардероба, реализующих общий интерфейс `IClosetItemModel` для доступа к ним без определения конкретного типа. Диаграмма классов для предметов гардероба представлена на рисунке 7, а для фабрик на рисунке 8.

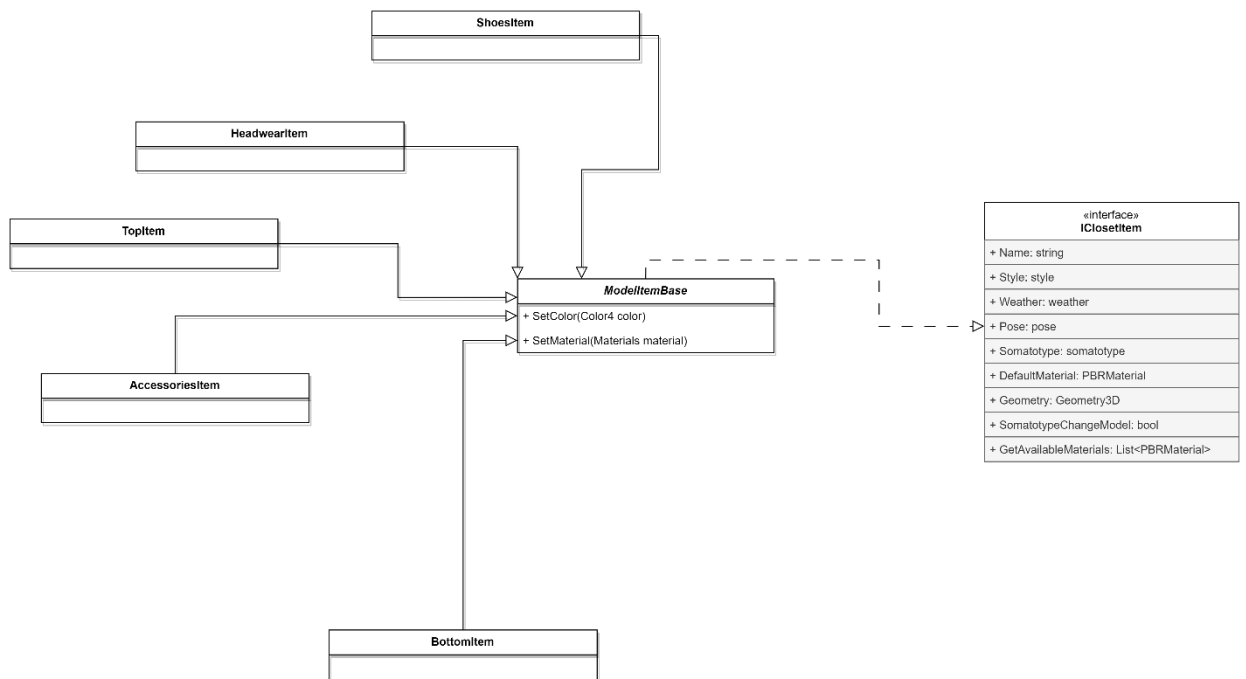


Рисунок 7 – Диаграмма классов предметов гардероба

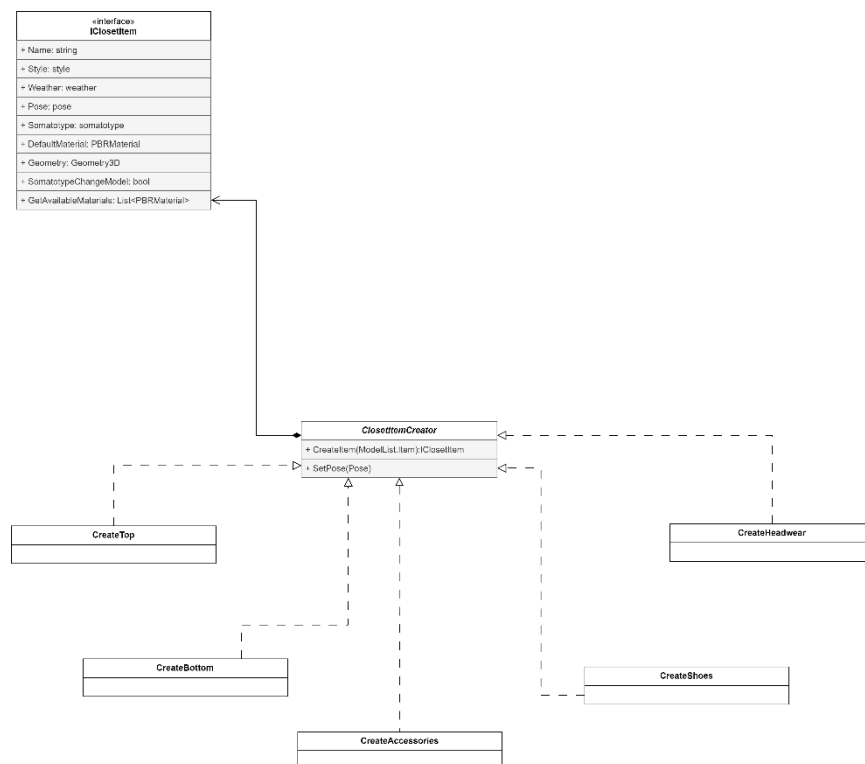


Рисунок 8 – Диаграмма классов фабрик

Многопоточный одиночка был использован для создания доступа к классам, которые должны существовать в единичном экземпляре. К ним относятся:

1. **MannequinSettings** – класс, который хранит в себе манекен и текущие предметы гардероба, а также осуществляет частичное управление над ними;
2. **ViewportSettings** – класс, хранящий в себе основные настройки текущей сцены главного окна просмотра (Рисунок 9).

Именно через них подразумевается взаимодействие пользователя с основной логикой приложения.

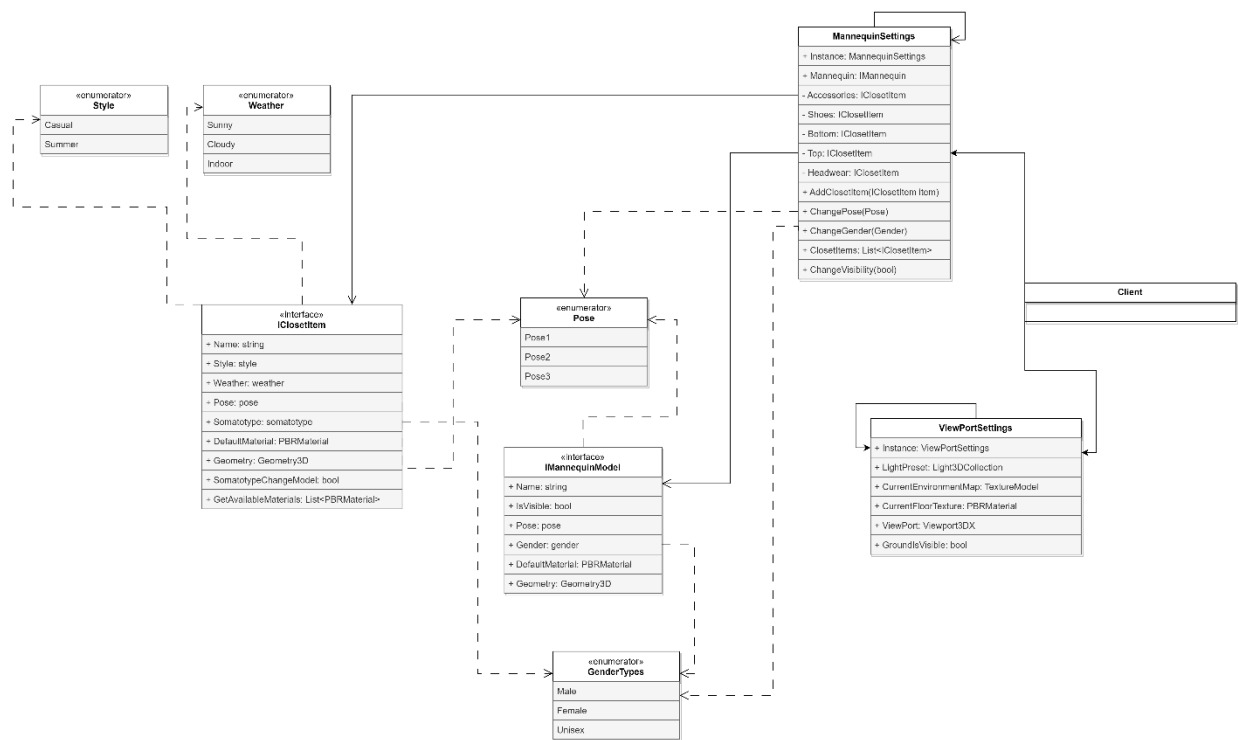


Рисунок 9 – Диаграмма классов настроек, реализующих паттерн одиночки

## **5 Используемые библиотеки и методы программирования**

В данной работе были использованы такие библиотеки, как:

1. Helix Toolkit WPF Sharp DX – это библиотека WPF элементов для создания 2D и 3D графики с использованием SharpDX. Она предоставляет 3D просмотрщики, 2D элементы, UI элементы и инструменты для 3D моделирования. Благодаря Helix Toolkit можно легко добавлять 3D и 2D функциональность в WPF приложения с применением SharpDX для ускорения рендеринга. Отображение и хранение всех 3D объектов приложения происходило через данную библиотеку.

2. Haley WPF – это библиотека UI компонентов для WPF, которая позволяет ускорить разработку интерфейса за счет использования готовых компонентов, поддержки MVVM, тем оформления и визуального дизайнера.

3. Haley Icons предоставляет набор стандартных, материальных и кастомных иконок для использования в компонентах Haley UI. Это включает в себя иконки Material Design, глифы FontAwesome и полностью кастомные SVG иконки.

4. SciChart.Wpf.UI.Transitionz — это библиотека для создания анимированных переходов между элементами в WPF приложениях, а также для упрощения создания сложных анимаций.

5. MSTest – это библиотека для написания юнит-тестов, которая предоставляет необходимую инфраструктуру в виде атрибутов, ассертов и отчетов, позволяя эффективно тестировать методы на уровне отдельных функций.



## 6 Реализация программного продукта

### 6.1 Описание реализации модулей, классов, методов и алгоритмов, используемых при решении задачи.

Для реализации базовых параметров были разработаны основные виды перечислений, такие как: гендер, позы, базовые материалы, тип одежды, стиль и погода (Рисунок 10).

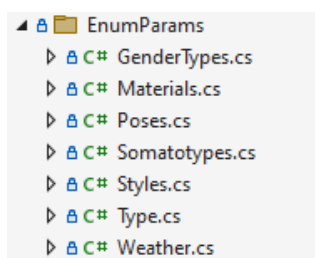


Рисунок 10 – Основные перечисляемые параметры

Фабрики, создающие различные типы предметов унаследовали общий абстрактный класс ClosetItemCreator (Рисунок 11). Фабрики также хранят в себе позу и гендер для создаваемого предмета, а также содержат метод, позволяющий задавать их.

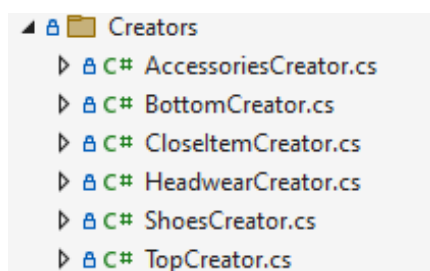


Рисунок 11 – Фабрики для создания предметов

Каждый тип предметов гардероба наследует общий абстрактный класс, реализующий базовую логику загрузки геометрии, настройки доступных материалов, а также описанные интерфейсом IClosetItemModel основные

свойства и методы (Рисунок 12).

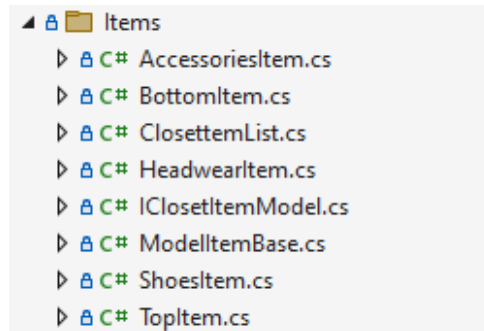


Рисунок 12 – Виды предметов

Для упрощения взаимодействия со системой был составлен статический класс `ClosetItemList`, хранящий в себе словарь предметов стандартной библиотеки 3D моделей (Рисунок 13).

```
Ссылка 18
public static class ClosetItemList
{
    private static Dictionary<string, Item> items = new()
    {
        { "Tshirt",
            new("Tshirt", GenderTypes.Unisex, Type.Top, Styles.Casual | Styles.Summer,
                Weather.Sunny | Weather.Cloudy | Weather.Indoor, "Футболка") },
        { "Blouse",
            new("Blouse", GenderTypes.Female, Type.Top, Styles.Casual | Styles.Summer | Styles.Formal,
                Weather.Sunny | Weather.Cloudy | Weather.Indoor, "Блузка") },
        { "Shirt",
            new("Shirt", GenderTypes.Male, Type.Top, Styles.Casual,
                Weather.Sunny | Weather.Cloudy | Weather.Indoor, "Рубашка") },
        { "RoundGlasses",
            new("RoundGlasses", GenderTypes.Unisex, Type.Accessories, Styles.Casual | Styles.Formal,
                Weather.Sunny | Weather.Cloudy | Weather.Indoor, "Круглые очки") },
        { "WomenGlasses",
            new("WomenGlasses", GenderTypes.Female, Type.Accessories, Styles.Casual | Styles.Formal,
                Weather.Sunny | Weather.Cloudy | Weather.Indoor, "Женские очки") },
        { "AviatorGlasses",
            new("AviatorGlasses", GenderTypes.Male, Type.Accessories, Styles.Casual | Styles.Summer,
                Weather.Sunny | Weather.Cloudy | Weather.Indoor, "Очки-авиаторы") },
        { "OxfordShoes",
            new("OxfordShoes", GenderTypes.Male, Type.Shoes, Styles.Formal | Styles.Casual,
                Weather.Sunny | Weather.Cloudy | Weather.Indoor, "Оксфордские подкрадули") },
        { "Sneakers",
            new("Sneakers", GenderTypes.Unisex, Type.Shoes, Styles.Casual | Styles.Summer,
                Weather.Sunny | Weather.Cloudy | Weather.Indoor, "Сникерсы, сникерсы") },
        { "HeeledShoes",
            new("HeeledShoes", GenderTypes.Female, Type.Shoes, Styles.Formal | Styles.Summer | Styles.Casual,
                Weather.Sunny | Weather.Cloudy | Weather.Indoor, "Туфли с кирилом") },
        { "Shorts",
            new("Shorts", GenderTypes.Male, Type.Bottom, Styles.Casual | Styles.Summer,
                Weather.Sunny | Weather.Cloudy | Weather.Indoor, "Шорты") },
        { "Skirt",
            new("Skirt", GenderTypes.Female, Type.Bottom, Styles.Summer | Styles.Formal | Styles.Casual,
                Weather.Sunny | Weather.Cloudy | Weather.Indoor, "Юбка") },
        { "Trousers",
            new("Trousers", GenderTypes.Unisex, Type.Bottom, Styles.Casual | Styles.Formal,
                Weather.Sunny | Weather.Cloudy | Weather.Indoor, "Брюки") },
    }
}
```

Рисунок 13 – Словарь предметов

Для упрощения доступа к словарю были добавлены различные методы, выполняющие возврат предмета, согласно заданным параметрам (Рисунок 14).

```

Ссылка 1
public static List<Item> SelectItems(GenderTypes G, Weather W, Styles S)
{
    var res = new List<Item>();
    foreach (var Itemon in items)
    {
        if (((int)Itemon.Value.Gender & (int)G) > 0 || (((int)Itemon.Value.Weather & (int)W) > 0 || (((int)Itemon.Value.Style & (int)S) > 0))
            res.Add(Itemon.Value);
    }
    return res;
}

Ссылка 8
public static List<string> SelectItems(GenderTypes G, Type T)
{
    var res = new List<string>() { "Her" };
    foreach (var Itemon in items)
    {
        if (((int)Itemon.Value.Gender & (int)G) > 0 && (Itemon.Value.Type == T))
            res.Add(Itemon.Value.ViewName);
    }
    return res;
}

Ссылка 1
public static Item GetItem(string name)
{
    if (items.ContainsKey(name))
    {
        return items[name];
    }
    else
    {
        throw new ArgumentException($"Item with name {name} not found. Before creating the item, add it to the ClosetItemList");
    }
}

Ссылка 8
public static Item GetItem2(string name)
{
    if (name == null || name == "Her") return null;
    foreach (var item in items)
    {
        if (item.Value.ViewName == name) return item.Value;
    }
    throw new ArgumentException($"Item with name {name} not found. Before creating the item, add it to the ClosetItemList");
}

```

Рисунок 14 – Методы для получения предметов базовой библиотеки

Для хранения текстур, моделей, кубических карт и иконок была выделена следующая иерархия файлов:

1. Environment – директория, хранящая кубические карты окружения и текстуры поверхности;
2. Icons – хранит, используемые иконки кнопок;
3. Items – директория, которая хранит модели и материалы стартовой библиотеки материалов. Она содержит папки, которые соотносятся с каждым параметром из перечислений параметров;
4. Mannequins – используется для хранения obj и mtl файлов манекенов, а также их текстур (Рисунок 15).

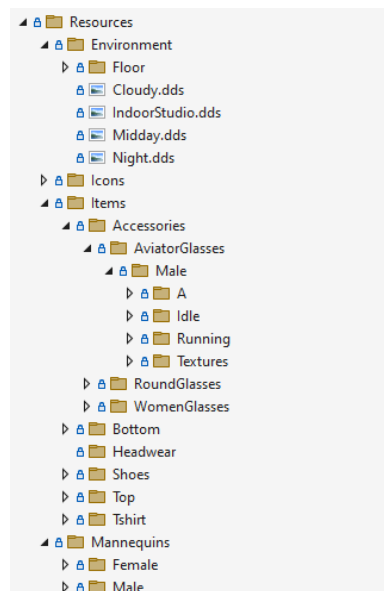


Рисунок 15 – Иерархия ресурсов приложения

Также был реализован класс, в котором представлен метод алгоритма выборки предметов из списка доступной одежды. Классы алгоритмов должны реализовывать разработанный общий интерфейс алгоритма, это позволит расширять приложение разными алгоритмами подбора. На рисунке 16 представлен пример такого алгоритма.

```

class SelectClozses : AlgorytmForWear
{
    Random random = new();
    private MannequinSettings _mannequinSettings = MannequinSettings.Instance;

    COMBIOIC 2
    public void WearMe(GenderTypes _g, Weather _w, Styles _s)
    {
        List<Item> Wordrobe = ClosetItemList.SelectItems(_g, _w, _s);
        List<Item> Top = new();
        List<Item> Bottom = new();
        List<Item> Shoes = new();
        List<Item> Accessory = new();

        _mannequinSettings.ResetItems();

        foreach (var itm in Wordrobe)
        {
            if (itm.Type == Type.Top)
            {
                Top.Add(itm);
            }
            if (itm.Type == Type.Accessories)
            {
                Accessory.Add(itm);
            }
            if (itm.Type == Type.Shoes)
            {
                Shoes.Add(itm);
            }
            if (itm.Type == Type.Bottom)
            {
                Bottom.Add(itm);
            }
        }

        if (Top.Count > 0)
            _mannequinSettings.AddClosetItem(Top[random.Next(Top.Count)]);
        if (Accessory.Count > 0)
            _mannequinSettings.AddClosetItem(Accessory[random.Next(Accessory.Count)]);
        if (Bottom.Count > 0)
            _mannequinSettings.AddClosetItem(Bottom[random.Next(Bottom.Count)]);
        if (Shoes.Count > 0)
            _mannequinSettings.AddClosetItem(Shoes[random.Next(Shoes.Count)]);
    }
}

```

Рисунок 16 – Методы среднего и левого прямоугольников

Представления в WPF создаются с помощью языка разметки XAML. Благодаря технологии привязки данных (binding) можно связать представления с компонентами ViewModel. При разработке приложения были созданы 5 представлений и 5 элементов компонента ViewModel связанных соответственно. Вследствие чего удалось разделить логику приложения и отображение представлений (Рисунки 17 и 18).



## 6.2 Описание тестовых планов и наборов тестов

Для тестирования были выбраны классы BottomCreator – для проверки корректного создания модели предмета гардероба и RealisticModel – для проверки загрузки геометрии и материалов у манекена.

Для первого тестируются методы загрузки и построения геометрии из файла формата obj, задания параметров и установки флага инициализации (Листинг 1).

```
[TestClass]
public class BottomCreatorTests
{
    CloseItemCreator creator;
    IClosetItemModel closetItem;
    [TestInitialize]
    public void Init()
    {
        creator = new BottomCreator();
        creator.SetParams(Poses.Idle, GenderTypes.Male);
        closetItem = creator.CreateClosetItem("Shorts");
    }
    [TestMethod]
    public void GeometryLoadTest()
    {
        var reader = new ObjReader();
        var objCol =
reader.Read($"{AppDomain.CurrentDomain.BaseDirectory}TestResourc
es\\BottomModel\\Shorts.obj");
        var meshArray = new MeshGeometry3D[objCol.Count];
        for (int i = 0; i < objCol.Count; i++)
        {
            meshArray[i] = new MeshGeometry3D();
            objCol[i].Geometry.AssignTo(meshArray[i]);
        }
        var expected =
MeshGeometry3D.Merge(meshArray).Indices.Average();
        var actual =
creator.CreateClosetItem("Shorts").Geometry.Indices.Average();
        Assert.AreEqual(expected, actual);
    }
    [TestMethod]
    public void SetParamsTest()
    {
        var expected = Poses.Running;
        creator.SetParams(Poses.Running);
    }
}
```

```

        var actual = creator.Pose;
        Assert.AreEqual(expected, actual);
    }
    [TestMethod]
    public void InitializationTest()
    {
        var expected = false;
        creator.SetParams(Poses.A, GenderTypes.Unisex);
        closetItem = creator.CreateClosetItem("Shorts");
        var actual = closetItem.IInitialized;
        Assert.AreEqual(expected, actual);
    }
}

```

### Листинг 1 – Тестирование Bottom Creator

Для **RealisticModel** происходит тестирование корректного создания материалов, геометрии и корректности установки параметров при создании экземпляра класса (Листинг 2).

```

[TestClass]
public class RealisticModelTests
{
    IMannequinModel mannequinModel;
    ObservableElement3DCollection expectedMannequinModel;
    [TestInitialize]
    public void Init()
    {
        mannequinModel = new RealisticModel("Joe", true,
        GenderTypes.Male, Poses.Idle);
        expectedMannequinModel =
        LoadModelAndMaterials($"{AppDomain.CurrentDomain.BaseDirectory}TestResources\\MaleMannequin\\IdlePose.obj");
    }
    private ObservableElement3DCollection
    LoadModelAndMaterials(string modelPath)
    {
        ObservableElement3DCollection mannequin = new();
        var reader = new ObjReader();
        var objCol = reader.Read(modelPath);
        for (int i = 0; i < objCol.Count; ++i)
        {
            var ob = objCol[i];
            var vertColor = new Color4((float)i / objCol.Count,
            0, 1 - (float)i / objCol.Count, 1);

```



```

        ob.Geometry.Colors = new
Color4Collection(Enumerable.Repeat(vertColor,
ob.Geometry.Positions.Count));
        ob.Geometry.UpdateOctree();
        ob.Geometry.UpdateBounds();

var scaleTransform = new Media3D.ScaleTransform3D(1,
1, 1);

var s = new MeshGeometryModel3D
{
    Geometry = ob.Geometry,
    CullMode = SharpDX.Direct3D11.CullMode.Back,
    IsThrowingShadow = true,
    Transform = scaleTransform,
};

var diffuseMaterial = new DiffuseMaterial();
PBRMaterial pbrMaterial = null;
PhongMaterial phong = null;
if (ob.Material is PhongMaterialCore p)
{
    phong = p.ConvertToPhongMaterial();
    phong.RenderEnvironmentMap = true;
    phong.RenderShadowMap = true;
    phong.RenderSpecularColorMap = false;
    phong.RenderDiffuseAlphaMap = false;
    phong.EnableAutoTangent = true;
    s.Material = phong;
    diffuseMaterial.DiffuseColor = p.DiffuseColor;
    diffuseMaterial.DiffuseMap = p.DiffuseMap;
    pbrMaterial = new PBRMaterial()
    {
        AlbedoColor = p.DiffuseColor,
        AlbedoMap = p.DiffuseMap,
        NormalMap = p.NormalMap,
        RoughnessMetallicMap = p.SpecularColorMap,
        AmbientOcclusionMap = p.SpecularColorMap,
        RenderShadowMap = true,
        RenderEnvironmentMap = true,
        RenderAlbedoMap = true,
        RenderDisplacementMap = true,
        RenderNormalMap = true,
        RenderAmbientOcclusionMap = true,
        EnableTessellation = false
    };
}

var mesh = new MeshGeometryModel3D
{
    Geometry = ob.Geometry,

```

```

        CullMode = SharpDX.Direct3D11.CullMode.Back,
        IsThrowingShadow = true,
        Transform = scaleTransform,
        Material = pbrMaterial,
        DepthBias = i << 2
    };
    mannequin.Add(mesh);
}
return mannequin;
}

[TestMethod]
public void MaterialLoadTest()
{
    var expected =
        ((PBRMaterial)((MeshGeometryModel3D)expectedMannequinModel[0]).Material).AlbedoColor;
    var actual =
        ((PBRMaterial)((MeshGeometryModel3D)mannequinModel.Mannequin[0]).Material).AlbedoColor;
    Assert.AreEqual(expected, actual);
}

[TestMethod]
public void GeometryLoadTest()
{
    var expected =
        ((MeshGeometryModel3D)expectedMannequinModel[0]).Geometry.Indices.Average();
    var actual =
        ((MeshGeometryModel3D)mannequinModel.Mannequin[0]).Geometry.Indices.Average();
    Assert.AreEqual(expected, actual);
}

[TestMethod]
public void NameSetTest()
{
    var expected = "Joe";
    var actual = mannequinModel.Name;
    Assert.AreEqual(expected, actual);
}

[TestMethod]
public void PoseSetTest()
{
    var expected = Poses.Idle;
    var actual = mannequinModel.Pose;
    Assert.AreEqual(expected, actual);
}

[TestMethod]

```

```
public void GenderSetTest()  
{  
    var expected = GenderTypes.Male;  
    var actual = mannequinModel.Gender;  
    Assert.AreEqual(expected, actual);  
}  
}
```

## Листинг 2 – Тестирование класса RealisticModel

### 6.3 Результаты тестирования программы

Все 8 тестов были корректно выполнены, следовательно, создание предметов гардероба и манекенов работает правильно (Рисунок 19).

Тестирование	Длительн...	Признаки	Сообщение об ошибке
✓ YOSTests (8)	14,3 с		
✓ YOSTests (8)	14,3 с		
✓ BottomCreatorTests (3)	4,2 с		
✓ GeometryLoadTest	3,4 с		
✓ InitializationTest	429 мс		
✓ SetParamsTest	389 мс		
✓ RealisticModelTests (5)	10,1 с		
✓ GenderSetTest	2,5 с		
✓ GeometryLoadTest	1,9 с		
✓ MaterialLoadTest	1,8 с		
✓ NameSetTest	1,9 с		
✓ PoseSetTest	2 с		

Рисунок 19 – Результаты тестирования

Так же было проведено ручное тестирование, для проверки корректного взаимодействия пользователя с программой. Выполнение программы показано на рисунках 20 и 21.

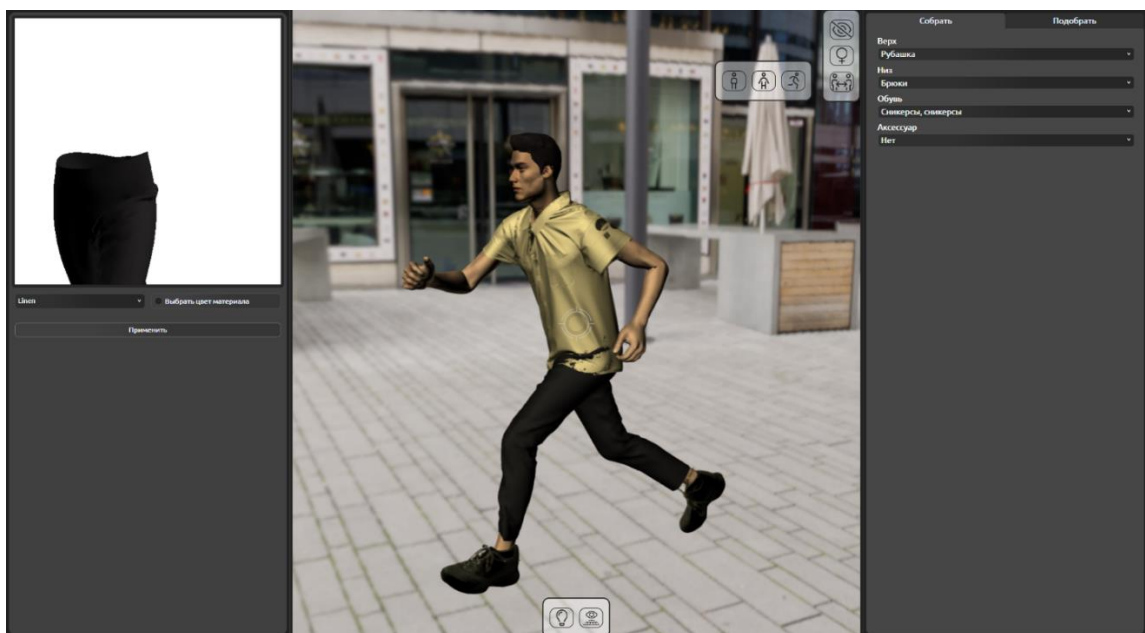


Рисунок 20 – Ручное тестирование 1

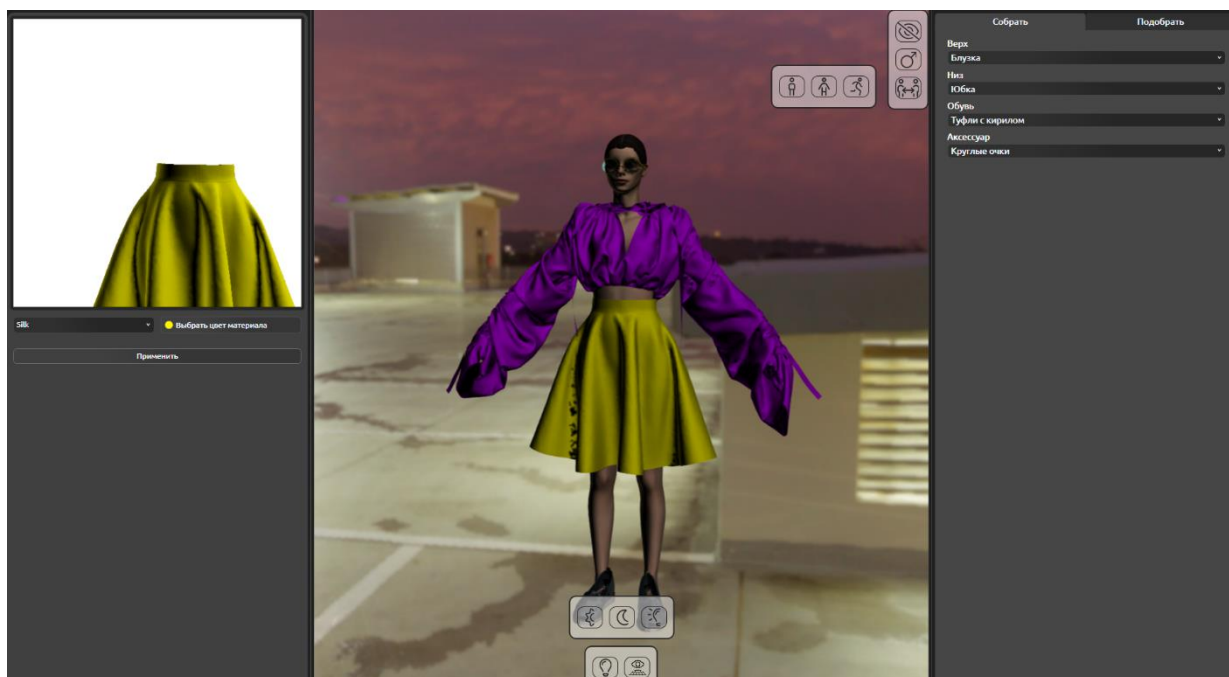


Рисунок 21 – Ручное тестирование 2

Пользователь может собрать образ сам, либо воспользоваться панелью для работы с алгоритмом, протестированным на корректность работы (Рисунок 22).

Собрать

Подобрать

Пол

☒ Мужчина
 ☐ Женщина
 ☐ Унисекс

Погода

☒ В помещении
 ☐ Пасмурно
 ☐ Ясно

Стиль

☒ Casual
 ☐ Летний
 ☐ Формальный

Применить

Рисунок 22 – Панель для работы с алгоритмом автоматического подбора образа

Пользователь также изменять предметы одежды на свое усмотрение через специальную панель. Если пользователь не введет материал у редактируемой вещи, то приложение укажет о необходимости ввода данных (Рисунок 23).

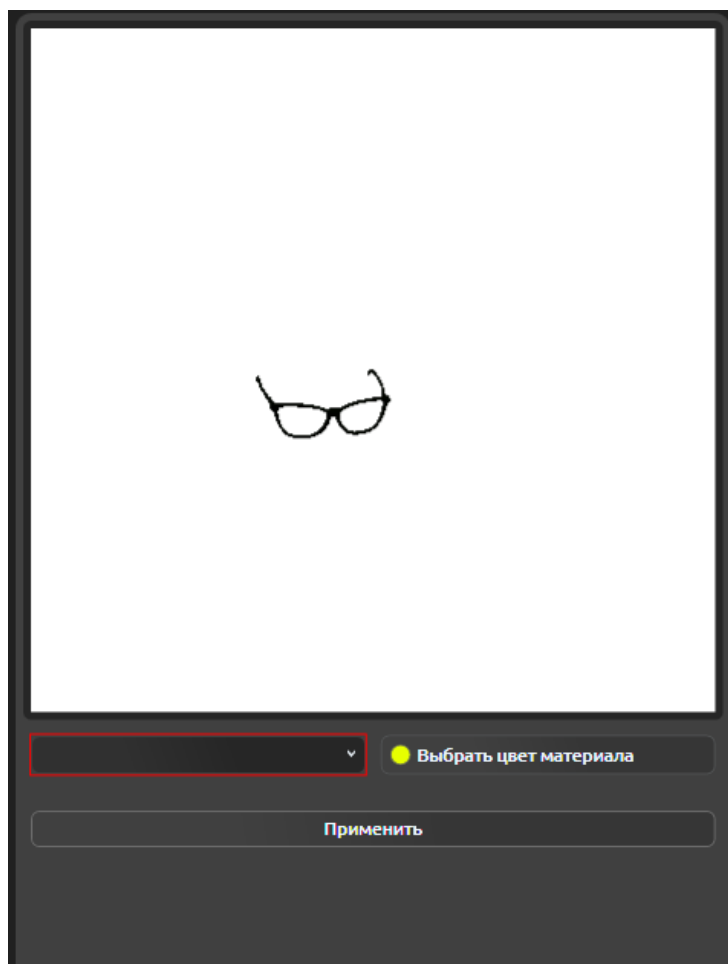


Рисунок 23 – Указание о необходимости выбора материала

## **Заключение**

Цель курсового проекта заключалась в разработке приложения для создания и подбора образа одежды, а также изучение объектно-ориентированного дизайна и работа с 3D-графикой.

Выполнение поставленных в курсовой работе задач:

1. Изучены основы построения и применения архитектурных паттернов и паттернов проектирования.
2. Разработано приложение, позволяющее создавать образ одежды.
3. Построены диаграммы, описывающие программный продукт.
4. Написаны тесты, проверяющие правильность работы приложения.
5. Разработан интерфейс для удобного пользования приложением.

Данная программа помогает наглядно смоделировать образ одежды, используя 3D-графику, стандартную библиотеку моделей и текстур, а также цветовую палитру формата RGBA. Также возможно дальнейшее развитие программы путём совершенствования алгоритма подбора, реализации возможности сохранения и открытия проектов и 3D моделей, и их перемещение внутри сцены.

Программа протестирована. Корректность работы не вызывает сомнений. Поставленные в курсовом проекте задачи решены в полном объеме.

### **Список использованных источников**

1. Технология разработки программного обеспечения: учебное пособие / Л.Г.Гагарина, Е.В.Кокорева, Б.Д.Виснадул; Под ред. проф. Л.Г.Гагариной
2. Архитектура и проектирование программных систем: монография / С.В. Назаров
3. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. П75 Паттерны объектно-ориентированного проектирования. — СПб.: Питер, 2020. — 448 с.
4. Мартин Р. М29 Чистая архитектура. Искусство разработки программного обеспечения. — СПб.: Питер, 2021. — 352 с.
5. WPF - Windows Presentation Foundation documentation.: сайт. URL. – <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/?view=netdesktop-7.0> (дата обращения: 02.06.2023).
6. WPF и C# | Полное руководство.: сайт. URL. – <https://metanit.com/sharp/wpf/> (дата обращения 29.05.2023).
7. Obj и mlt формат.: сайт. URL. – <https://ru.wikipedia.org/wiki/Obj> (дата обращения 15.05.2023)



## Приложение А

### Листинг программы

#### **AlgorytmForWear.cs**

```
interface AlgorytmForWear
{
    void WearMe(GenderTypes _g, Weather _w, Styles _s);
}
```

#### **SelectClozzes.cs**

```
class SelectClozzes : AlgorytmForWear
{
    Random random = new();

    private MannequinSettings _mannequinSettings =
        MannequinSettings.Instance;

    public void WearMe(GenderTypes _g, Weather _w, Styles
        _s)
    {
        List<Item> Wordrobe = ClosetItemList.SelectItems(_g,
            _w, _s);

        List<Item> Top = new();
        List<Item> Bottom = new();
        List<Item> Shoes = new();
        List<Item> Accessory = new();

        _mannequinSettings.ResetItems();

        foreach (var itm in Wordrobe)
        {
            if (itm.Type == Type.Top)
            {
                Top.Add(itm);
            }
        }
    }
}
```

```

        if (itm.Type == Type.Accessories)
        {
            Accessory.Add(itm);
        }
        if (itm.Type == Type.Shoes)
        {
            Shoes.Add(itm);
        }
        if (itm.Type == Type.Bottom)
        {
            Bottom.Add(itm);
        }
    }

    if (Top.Count > 0)

_mannequinSettings.AddClosetItem(Top[random.Next(Top.Count)]);

        if (Accessory.Count > 0)

_mannequinSettings.AddClosetItem(Accessory[random.Next(Accessory
.Count)]);

        if (Bottom.Count > 0)

_mannequinSettings.AddClosetItem(Bottom[random.Next(Bottom.Count
)]);

        if (Shoes.Count > 0)

_mannequinSettings.AddClosetItem(Shoes[random.Next(Shoes.Count)]
);

    }
}

```

### **CloseItemCreator.cs**

```

public abstract class CloseItemCreator
{

```

```

        public Poses Pose { get; protected set; }
        public GenderTypes Gender { get; protected set; }
        public abstract IClosetItemModel CreateClosetItem(string
name);
        public void SetParams(Poses pose = Poses.Idle,
            GenderTypes gender=GenderTypes.Male)
        {
            Pose = pose;
            Gender = gender;
        }
    }
}

```

#### **AccessoriesCreator.cs**

```

public class AccessoriesCreator : CloseItemCreator
{
    public override IClosetItemModel CreateClosetItem(string
name) => new AccessoriesItem(name, Gender, Pose);
}

```

#### **BottomCreator.cs**

```

public class BottomCreator : CloseItemCreator
{
    public override IClosetItemModel CreateClosetItem(string
name) => new BottomItem(name, Gender, Pose);
}

```

#### **HeadwearCreator.cs**

```

public class HeadwearCreator : CloseItemCreator
{
    public override IClosetItemModel CreateClosetItem(string
name) => new HeadwearItem(name, Gender, Pose);
}

```

#### **ShoesCreator.cs**

```

public class ShoesCreator : CloseItemCreator
{
    public override IClosetItemModel CreateClosetItem(string
name) => new ShoesItem(name, Gender, Pose);
}

```

```
}
```

### **TopCreator.cs**

```
public class TopCreator : CloseItemCreator
{
    public override IClosetItemModel CreateClosetItem(string
name) => new TopItem(name, Gender, Pose);
}
```

### **GenderTypes.cs**

```
public enum GenderTypes
{
    Male = 1,
    Female = 2,
    Unisex = 3
}
```

### **Materials.cs**

```
[Flags]
public enum Materials
{
    Cotton = 1,
    Linen = 2,
    Wool = 4,
    Denim = 8,
    Polyester = 16,
    Silk = 32,
    Leather = 64,
    Plastic = 128,
    Textile = 256,
    Metal = 512,
}
```

### **Poses.cs**

```
public enum Poses
{
    Idle, Running, A
```

```
}
```

### **Styles.cs**

```
[Flags]
```

```
public enum Styles
{
    Casual = 1,
    Summer = 2,
    Formal = 4
}
```

### **Type.cs**

```
public enum Type
{
    Top,
    Bottom,
    Accessories,
    Shoes,
    Headwear
}
```

### **Weather.cs**

```
[Flags]
```

```
public enum Weather
{
    Sunny = 1,
    Cloudy = 2,
    Indoor = 4
}
```

**AccessoriesItem.cs** - аналогично и для **BottomItem.cs**,  
**HeadwearItem.cs**, **ShoesItem.cs** и **TopItem.cs**

```
public class AccessoriesItem : ModelItemBase
{
    public AccessoriesItem(string name,
        GenderTypes gender = GenderTypes.Male,
        Poses pose = Poses.Idle,
```

```

        Styles style = Styles.Casual,
        Weather weather = Weather.Indoor) : base(name,
gender, pose, style, weather)
    {
        _type = Type.Accessories;
        _mainPath =
$"{AppDomain.CurrentDomain.BaseDirectory}Resources\\Items\\{_type}
\\{_name}\\{_gender}";
        if (Directory.Exists(_mainPath))
        {
            _texPath = $"{_mainPath}\\Textures";
            _modelPath =
$"{_mainPath}\\{_pose}\\{_name}.obj";
            InitAvaliableMaterials();
            InitGeometryAndMaterials();
            _material.AlbedoColor = Color4.White;
        }
    }

    public override object Clone()
    {
        var clone = new AccessoriesItem(Name, Gender, Pose,
Style, Weather);
        clone._material = (PBRMaterial)_material.Clone();
        clone._geometry = _geometry;
        clone._type = _type;
        return clone;
    }
}

```

#### **ClosettemList.cs**

```

public static class ClosetItemList
{
    private static Dictionary<string, Item> items = new()
    {

```

```

    { "Tshirt",
        new("Tshirt", GenderTypes.Unisex, Type.Top,
        Styles.Casual | Styles.Summer,
            Weather.Sunny | Weather.Cloudy |
        Weather.Indoor, "Футболка") },
    { "Blouse",
        new("Blouse", GenderTypes.Female, Type.Top,
        Styles.Casual | Styles.Summer | Styles.Formal,
            Weather.Sunny | Weather.Cloudy |
        Weather.Indoor, "Блузка") },
    { "Shirt",
        new("Shirt", GenderTypes.Male, Type.Top,
        Styles.Casual,
            Weather.Sunny | Weather.Cloudy |
        Weather.Indoor, "Рубашка") },
    { "RoundGlasses",
        new("RoundGlasses", GenderTypes.Unisex,
        Type.Accessories, Styles.Casual | Styles.Formal,
            Weather.Sunny | Weather.Cloudy |
        Weather.Indoor, "Круглые очки") },
    { "WomenGlasses",
        new("WomenGlasses", GenderTypes.Female,
        Type.Accessories, Styles.Casual | Styles.Formal,
            Weather.Sunny | Weather.Cloudy |
        Weather.Indoor, "Женские очки") },
    { "AviatorGlasses",
        new("AviatorGlasses", GenderTypes.Male,
        Type.Accessories, Styles.Casual | Styles.Summer,
            Weather.Sunny | Weather.Cloudy |
        Weather.Indoor, "Очки-авиаторы") },
    { "OxfordShoes",
        new("OxfordShoes", GenderTypes.Male, Type.Shoes,
        Styles.Formal | Styles.Casual,
            Weather.Sunny | Weather.Cloudy |
        Weather.Indoor, "Оксфордские подкрадули") },
    { "Sneakers",
        new("Sneakers", GenderTypes.Unisex, Type.Shoes,
        Styles.Casual | Styles.Summer,

```

```

        Weather.Sunny | Weather.Cloudy |
Weather.Indoor, "Сникерсы, сникерсы") },

    { "HeeledShoes",
        new("HeeledShoes", GenderTypes.Female,
Type.Shoes, Styles.Formal | Styles.Summer | Styles.Casual,
        Weather.Sunny | Weather.Cloudy
|Weather.Indoor, "Туфли с кирилом") },

    { "Shorts",
        new("Shorts", GenderTypes.Male, Type.Bottom,
Styles.Casual | Styles.Summer,
        Weather.Sunny | Weather.Cloudy |
Weather.Indoor, "Шорты") },

    { "Skirt",
        new("Skirt", GenderTypes.Female, Type.Bottom,
Styles.Summer | Styles.Formal | Styles.Casual,
        Weather.Sunny | Weather.Cloudy |
Weather.Indoor, "Юбка") },

    { "Trousers",
        new("Trousers", GenderTypes.Unisex, Type.Bottom,
Styles.Casual | Styles.Formal,
        Weather.Sunny | Weather.Cloudy |
Weather.Indoor, "Брюки") },

    { "Beret",
        new("Beret", GenderTypes.Female, Type.Headwear,
Styles.Casual | Styles.Formal,
        Weather.Sunny | Weather.Cloudy |
Weather.Indoor, "Кандибобер") },

    { "Kippah",
        new("Kippah", GenderTypes.Male, Type.Headwear,
Styles.Formal,
        Weather.Sunny | Weather.Cloudy |
Weather.Indoor, "Арабская шапка") },

    { "KnitHat",
        new("KnitHat", GenderTypes.Unisex,
Type.Headwear, Styles.Casual,
        Weather.Cloudy, "Шапка") },

    { "BeanieHat",

```



```

        new("BeanieHat", GenderTypes.Unisex,
Type.Headwear, Styles.Casual,
        Weather.Cloudy, "Шапка зайко-девочки") }
};

    public static List<Item> SelectItems(GenderTypes G,
Weather W, Styles S)
    {
        var res = new List<Item>();
        foreach (var Itemon in items)
        {
            if (((int)Itemon.Value.Gender & (int)G) > 0) ||
(((int)Itemon.Value.Weather & (int)W) > 0) ||
(((int)Itemon.Value.Style & (int)S) > 0))
                res.Add(Itemon.Value);
        }
        return res;
    }

    public static List<string> SelectItems(GenderTypes G,
Type T)
    {
        var res = new List<string>() { "Нет" };
        foreach (var Itemon in items)
        {
            if (((int)Itemon.Value.Gender & (int)G) > 0) &&
(Itemon.Value.Type == T))
                res.Add(Itemon.Value.ViewName);
        }
        return res;
    }

    public static Item GetItem(string name)
    {
        if (items.ContainsKey(name))

```

```

        {
            return items[name];
        }
        else
        {
            throw new ArgumentException($"Item with name
{name} not found. Before creating the item, add it to the
ClosetItemList");
        }
    }

    public static Item GetItem2(string name)
    {
        if (name == null || name == "Her") return null;
        foreach (var item in items)
        {
            if (item.Value.ViewName == name) return
item.Value;
        }
        throw new ArgumentException($"Item with name {name}
not found. Before creating the item, add it to the
ClosetItemList");
    }
}

public class Item
{
    public string ViewName { get; init; }
    public string Name { get; init; }
    public GenderTypes Gender { get; init; }
    public Type Type { get; init; }
    public Styles Style { get; init; }
    public Weather Weather { get; init; }

    public Item(string name, GenderTypes gender, Type type,
Styles style, Weather weather, string translation)

```

```

    {
        Name = name;
        Gender = gender;
        Type = type;
        Style = style;
        Weather = weather;
        ViewName = translation;
    }
}

```

### **IClosetItemModel.cs**

```

public interface IClosetItemModel : ICloneable
{
    string Name { get; }
    Poses Pose { get; }
    PBRMaterial Material { get; set; }
    Geometry3D Geometry { get; }
    GenderTypes Gender { get; }
    Type Type { get; }
    List<Materials> AvailableMaterials { get; }
    public void SetColor(Color4 color);
    public void SetMaterial(Materials material);
    public Materials TextureMaterial { get; }
    public Color4 Color { get; }
    public bool IInitialized { get; }
}

```

### **ModelItemBase.cs**

```

public abstract class ModelItemBase :
OnPropertyChangedClass, IClosetItemModel
{
    protected string _mainPath;
    protected string _texPath;
    protected string _modelPath;
    protected List<Materials> _materials;
}

```

```

protected string _name;
protected Styles _style;
protected Weather _weather;
protected Poses _pose;
protected GenderTypes _gender;
protected Type _type;
protected PBRMaterial _material;
protected Geometry3D _geometry;

public string Name { get => _name; protected set =>
_name = value; }

public Styles Style { get => _style; protected set =>
_style = value; }

public Weather Weather { get => _weather; protected set
=> _weather = value; }

public Poses Pose { get => _pose; protected set => _pose
= value; }

public PBRMaterial Material { get => _material; set {
_material = value; OnPropertyChanged(); } }

public Geometry3D Geometry { get => _geometry; protected
set => _geometry = value; }

public GenderTypes Gender { get => _gender; protected
set => _gender = value; }

public Type Type { get => _type; protected set => _type
= value; }

public List<Materials> AvaliableMaterials => _materials;

public Materials TextureMaterial { get; protected set; }

public Color4 Color => _material.AlbedoColor;

public bool IInitialized { get; private set; }

public ModelItemBase(string name,
    GenderTypes gender = GenderTypes.Male,
    Poses pose = Poses.Idle,

```

```

        Styles style = Styles.Casual,
        Weather weather = Weather.Indoor)
    {
        IInitialized = false;
        _name = name;
        _gender = gender;
        _style = style;
        _weather = weather;
        _pose = pose;
        _material = new PBRMaterial();
    }

    public abstract object Clone();

    public void SetColor(Color4 color) =>
        _material.AlbedoColor = color;

    public void SetMaterial(Materials material)
    {
        _texPath = $"{_mainPath}\\Textures\\{material}";
        TextureMaterial = material;
        SetMaterialMaps();
    }

    protected void InitAvaliableMaterials()
    {
        _materials = new List<Materials>();
        foreach (var mat in
            (Materials[])Enum.GetValues(typeof(Materials)))
            if (Directory.Exists($"{_texPath}\\{mat}"))
                _materials.Add(mat);
    }

    protected void InitGeometryAndMaterials()
    {
        var reader = new ObjReader();
    }

```

```

var objCol = reader.Read(_modelPath);
var meshArray = new MeshGeometry3D[objCol.Count];
for (int i = 0; i < objCol.Count; i++)
{
    meshArray[i] = new MeshGeometry3D();
    objCol[i].Geometry.AssignTo(meshArray[i]);
}
var m = MeshGeometry3D.Merge(meshArray);
_geometry = m;
_material = new PBRMaterial
{
    AlbedoColor = new Color(255, 128, 0),
    RenderShadowMap = true,
    RenderEnvironmentMap = true,
    RenderAlbedoMap = true,
    RenderDisplacementMap = true,
    RenderNormalMap = true,
    RenderAmbientOcclusionMap = true,
    EnableTessellation = false,

};
_texPath += "\\\" + _materials[0].ToString();
TextureMaterial = _materials[0];
SetMaterialMaps();
IInitialized = true;
GC.Collect();
}
protected void SetMaterialMaps()
{
    string diffuseFileName = "diffuse";
    string normalFileName = "normal";
    string metallicRoughnessFileName =
"metallicroughness";

```

```

        string ambientOcclusionFileName =
"ambientocclusion";

        string displacementFileName = "displacement";

        string[] diffuseFiles = Directory.GetFiles(_texPath,
"*" + diffuseFileName + ".*");

        string[] normalFiles = Directory.GetFiles(_texPath,
"*" + normalFileName + ".*");

        string[] metallicRoughnessFiles =
Directory.GetFiles(_texPath, "*" + metallicRoughnessFileName +
"*.");

        string[] ambientOcclusionFiles =
Directory.GetFiles(_texPath, "*" + ambientOcclusionFileName +
"*.");

        string[] displacementFiles =
Directory.GetFiles(_texPath, "*" + displacementFileName +
"*.");

        if (diffuseFiles.Length > 0)
        {
            _material.AlbedoMap =
TextureModel.Create(diffuseFiles[0]);
        }

        if (normalFiles.Length > 0)
        {
            _material.NormalMap =
TextureModel.Create(normalFiles[0]);
        }

        if (metallicRoughnessFiles.Length > 0)
        {
            _material.RoughnessMetallicMap =
TextureModel.Create(metallicRoughnessFiles[0]);
        }

        if (ambientOcclusionFiles.Length > 0)
        {

```

```

        _material.AmbientOcclusionMap =
TextureModel.Create(ambientOcclusionFiles[0]);
    }

    if (displacementFiles.Length > 0)
    {
        _material.DisplacementMap =
TextureModel.Create(displacementFiles[0]);
    }
}
}

```

### **IMannequinModel.cs**

```

public interface IMannequinModel
{
    string Name { get; }
    bool IsVisible { get; set; }
    Poses Pose { get; set; }
    GenderTypes Gender { get; set; }
    ObservableElement3DCollection Mannequin { get; }
}

```

### **MannequinSettings.cs**

```

public class MannequinSettings : OnPropertyChangedClass
{
    IMannequinModel _mannequin;
    GenderTypes _gender;
    Poses _pose;
    IClosetItemModel? _accessory;
    IClosetItemModel? _headwear;
    IClosetItemModel? _shoes;
    IClosetItemModel? _top;
    IClosetItemModel? _bottom;
    IClosetItemModel _selectedItem;
}

```



```

    public IClosetItemModel SelectedItem => _selectedItem;
    public IClosetItemModel Bottom => _bottom;
    public IClosetItemModel Top => _top;
    public IClosetItemModel Headwear => _headwear;
    public IClosetItemModel Shoes => _shoes;
    public IClosetItemModel Accessory => _accessory;
    public IMannequinModel Mannequin => _mannequin;
    readonly ObservableCollection<IClosetItemModel>
_closetItemList;

    public ObservableCollection<IClosetItemModel>
ClosetItems => _closetItemList;

    public ObservableElement3DCollection MannequinModel =>
_mannequin.Mannequin;

    public bool MannequinIsVisible
    {
        get => _mannequin.IsVisible;
        set
        {
            _mannequin.IsVisible = value;
            OnPropertyChanged();
        }
    }

    private bool _iLoadModels;
    public bool ILoadModels
    {
        get => _iLoadModels;
        private set
        {
            SetProperty<bool>(ref _iLoadModels, value);
            Debug.WriteLine($"ILoadModels {value}");
        }
    }

    private static readonly Lazy<MannequinSettings> lazy =
new(() => new MannequinSettings());

```

```

        public static MannequinSettings Instance { get { return
lazy.Value; } }

        private MannequinSettings()
        {
            _mannequin = new RealisticModel("Joe", true,
GenderTypes.Male, Poses.A);
            _pose = _mannequin.Pose;
            _gender = _mannequin.Gender;
            _closetItemList = new
ObservableCollection<IClosetItemModel>()
            {
                _top,
                _bottom,
                _accessory,
                _shoes,
                _headwear,
            };
        }

        public void SelectItemByType(Type type)
        {
            switch (type)
            {
                case Type.Top:
                    _selectedItem = _top;
                    break;
                case Type.Bottom:
                    _selectedItem = _bottom;
                    break;
                case Type.Accessories:
                    _selectedItem = _accessory;
                    break;
                case Type.Shoes:
                    _selectedItem = _shoes;

```

```

        break;
    case Type.Headwear:
        _selectedItem = _headwear;
        break;
    default:
        break;
    }
    OnPropertyChanged(nameof(SelectedItem));
}

public GenderTypes Gender
{
    get => _gender;
    set
    {
        SetProperty<GenderTypes>(ref _gender, value);
        ILoadModels = true;

        SetProperty<IMannequinModel>(ref _mannequin, new
RealisticModel("noob", true, _gender, _pose),
nameof(MannequinModel));

        ILoadModels = false;
        UpdateItems();
    }
}

public Poses Pose
{
    get => _pose;
    set
    {
        SetProperty<Poses>(ref _pose, value);
        ILoadModels = true;

        SetProperty<IMannequinModel>(ref _mannequin, new
RealisticModel("noob", true, _gender, _pose),
nameof(MannequinModel));

        ILoadModels = false;
    }
}

```

```

        UpdateItems();
    }
}

private void UpdateItems()
{
    for (int i = 0; i < _closetItemList.Count; i++)
    {
        if (_closetItemList[i] != null)
        {
            var prevColor = _closetItemList[i].Color;
            var prevMat =
            _closetItemList[i].TextureMaterial;

            AddClosetItem(ClosetItemList.GetItem(_closetItemList[i].Name));
            if (_closetItemList[i] != null)
            {
                _closetItemList[i].SetColor(prevColor);
                _closetItemList[i].SetMaterial(prevMat);
            }
        }
    }
}

public void AddClosetItem(string name, Type type)
{
    try
    {
        ILoadModels = true;
        switch (type)
        {
            case Type.Top:
                var topCreator = new TopCreator();
                topCreator.SetParams(_pose, _gender);

```

```

        _top =
topCreator.CreateClosetItem(name);
        if (_top == null || !_top.IInitialized)
_top = null;

        _closetItemList[0] = _top;
        OnPropertyChanged(nameof(Top));
        break;
    case Type.Bottom:
        var bottomCreator = new BottomCreator();
        bottomCreator.SetParams(_pose, _gender);
        _bottom =
bottomCreator.CreateClosetItem(name);
        if (_bottom == null ||
!_bottom.IInitialized) _bottom = null;
        _closetItemList[1] = _bottom;
        OnPropertyChanged(nameof(Bottom));
        break;
    case Type.Accessories:
        var accCreator = new
AccessoriesCreator();
        accCreator.SetParams(_pose, _gender);
        _accessory =
accCreator.CreateClosetItem(name);
        if (_accessory == null ||
!_accessory.IInitialized) _accessory = null;
        _closetItemList[2] = _accessory;
        OnPropertyChanged(nameof(Accessory));
        break;
    case Type.Shoes:
        var shoeCreator = new ShoesCreator();
        shoeCreator.SetParams(_pose, _gender);
        _shoes =
shoeCreator.CreateClosetItem(name);
        if (_shoes == null ||
!_shoes.IInitialized) _shoes = null;
        _closetItemList[3] = _shoes;

```

```

        OnPropertyChanged(nameof(Shoes));
        break;
    case Type.Headwear:
        var headCreator = new HeadwearCreator();
        headCreator.SetParams(_pose, _gender);
        _headwear =
headCreator.CreateClosetItem(name);
        if (_headwear == null ||
!_headwear.IInitialized) _headwear = null;
        _closetItemList[4] = _headwear;
        OnPropertyChanged(nameof(Headwear));
        break;
    }
    ILoadModels = false;
}
catch (Exception e)
{
    Debug.WriteLine(e.Message);
    throw;
}
}
public void AddClosetItem(Item item)
{
    try
    {
        ILoadModels = true;
        switch (item.Type)
        {
            case Type.Top:
                var topCreator = new TopCreator();
                topCreator.SetParams(_pose, _gender);
                _top =
topCreator.CreateClosetItem(item.Name);

```

```

_top = null;

        if (_top == null || !_top.IInitialized)

            _closetItemList[0] = _top;
            OnPropertyChanged(nameof(Top));
            break;

        case Type.Bottom:
            var bottomCreator = new BottomCreator();
            bottomCreator.SetParams(_pose, _gender);
            _bottom =
bottomCreator.CreateClosetItem(item.Name);
            if (_bottom == null ||
!_bottom.IInitialized) _bottom = null;
            _closetItemList[1] = _bottom;
            OnPropertyChanged(nameof(Bottom));
            break;

        case Type.Accessories:
            var accCreator = new
AccessoriesCreator();
            accCreator.SetParams(_pose, _gender);
            _accessory =
accCreator.CreateClosetItem(item.Name);
            if (_accessory == null ||
!_accessory.IInitialized) _accessory = null;
            _closetItemList[2] = _accessory;
            OnPropertyChanged(nameof(Accessory));
            break;

        case Type.Shoes:
            var shoeCreator = new ShoesCreator();
            shoeCreator.SetParams(_pose, _gender);
            _shoes =
shoeCreator.CreateClosetItem(item.Name);
            if (_shoes == null ||
!_shoes.IInitialized) _shoes = null;
            _closetItemList[3] = _shoes;
            OnPropertyChanged(nameof(Shoes));

```

```

        break;
    case Type.Headwear:
        var headCreator = new HeadwearCreator();
        headCreator.SetParams(_pose, _gender);
        _headwear =
headCreator.CreateClosetItem(item.Name);
        if (_headwear == null ||
!_headwear.IIInitialized) _headwear = null;
        _closetItemList[4] = _headwear;
        OnPropertyChanged(nameof(Headwear));
        break;
    }
    ILoadModels = false;
}
catch (Exception e)
{
    Debug.WriteLine(e.Message);
    throw;
}
}

public void SetMaterialToSelectedItem(Materials
material, Color4 color)
{
    SelectedItem.SetColor(color);
    SelectedItem.SetMaterial(material);
    switch (SelectedItem.Type)
    {
        case Type.Top:
            if (Top == null) return;
            Top.SetColor(color);
            Top.SetMaterial(material);
            OnPropertyChanged(nameof(Top));
            break;

```



```

        case Type.Bottom:
            if (Bottom == null) return;
            Bottom.SetColor(color);
            Bottom.SetMaterial(material);
            OnPropertyChanged(nameof(Bottom));
            break;
        case Type.Accessories:
            if (Accessory == null) return;
            Accessory.SetColor(color);
            Accessory.SetMaterial(material);
            OnPropertyChanged(nameof(Accessory));
            break;
        case Type.Shoes:
            if (Shoes == null) return;
            Shoes.SetColor(color);
            Shoes.SetMaterial(material);
            OnPropertyChanged(nameof(Shoes));
            break;
        case Type.Headwear:
            if (Headwear == null) return;
            Headwear.SetColor(color);
            Headwear.SetMaterial(material);
            OnPropertyChanged(nameof(Headwear));
            break;
        default:
            break;
    }
}

public void ResetItems()
{
    _accessory = null;
    _shoes = null;
}

```

```

        _headwear = null;
        _top = null;
        _bottom = null;
        OnPropertyChanged();
    }
    public void ResetTop()
    {
        _top = null;
        OnPropertyChanged(nameof(Top));
    }
    public void ResetBottom()
    {
        _bottom = null;
        OnPropertyChanged(nameof(Bottom));
    }
    public void ResetShoes()
    {
        _shoes = null;
        OnPropertyChanged(nameof(Shoes));
    }
    public void ResetAccessory()
    {
        _accessory = null;
        OnPropertyChanged(nameof(Accessory));
    }
}

```

### **RealisticModel.cs**

```

public class RealisticModel : IMannequinModel
{
    public string Name { get; init; }
    bool _isVisible;
    Poses _pose;
}

```

```

GenderTypes _gender;
private string _mainPath;
private string _modelPath;
private ObservableElement3DCollection _mannequin;
public bool IsVisible
{
    get => _isVisible;
    set
    {
        if (_isVisible)
            foreach (var item in _mannequin)
            {
                item.Visibility =
System.Windows.Visibility.Hidden;
            }
        else
            foreach (var item in _mannequin)
            {
                item.Visibility =
System.Windows.Visibility.Visible;
            }
        _isVisible = value;
    }
}
public Poses Pose
{
    get => _pose;
    set
    {
        _pose = value;
        UpdatePaths();
        LoadModelAndMaterials();
    }
}

```

```

    }

    public GenderTypes Gender
    {
        get => _gender;
        set
        {
            _gender = value;
            UpdatePaths();
            LoadModelAndMaterials();
        }
    }

    public ObservableElement3DCollection Mannequin =>
    _mannequin;

    public RealisticModel(string name, bool visible,
    GenderTypes gender = GenderTypes.Male, Poses pose = Poses.A)
    {
        Name = name;
        _gender = gender;
        _pose = pose;
        _isVisible = visible;
        _mannequin = new ObservableElement3DCollection();
        _mainPath =
        $"{AppDomain.CurrentDomain.BaseDirectory}Resources\\Mannequins\\
        {_gender}";
        _modelPath = Directory.GetFiles(_mainPath, "*" +
        Pose + "*.obj")[0];

        LoadModelAndMaterials();
    }

    private void UpdatePaths()
    {

```

```

        _mainPath =
$"{AppDomain.CurrentDomain.BaseDirectory}Resources\\Mannequins\\
{_gender}";

        _modelPath = Directory.GetFiles(_mainPath, "*" +
Pose + "*.obj")[0];
    }

    private void LoadModelAndMaterials()
    {
        var reader = new ObjReader();
        var objCol = reader.Read(_modelPath);
        for (int i = 0; i < objCol.Count; ++i)
        {
            var ob = objCol[i];

            var vertColor = new Color4((float)i /
objCol.Count, 0, 1 - (float)i / objCol.Count, 1);

            ob.Geometry.Colors = new
Color4Collection(Enumerable.Repeat(vertColor,
ob.Geometry.Positions.Count));

            ob.Geometry.UpdateOctree();
            ob.Geometry.UpdateBounds();

            var scaleTransform = new
Media3D.ScaleTransform3D(1, 1, 1);

            var s = new MeshGeometryModel3D
            {
                Geometry = ob.Geometry,
                CullMode = SharpDX.Direct3D11.CullMode.Back,
                IsThrowingShadow = true,
                Transform = scaleTransform,
            };

            var diffuseMaterial = new DiffuseMaterial();
            PBRMaterial pbrMaterial = null;
            PhongMaterial phong = null;
            if (ob.Material is PhongMaterialCore p)

```

```

    {
        phong = p.ConvertToPhongMaterial();
        phong.RenderEnvironmentMap = true;
        phong.RenderShadowMap = true;
        phong.RenderSpecularColorMap = false;
        phong.RenderDiffuseAlphaMap = false;
        phong.EnableAutoTangent = true;
        s.Material = phong;
        diffuseMaterial.DiffuseColor =
p.DiffuseColor;

        diffuseMaterial.DiffuseMap = p.DiffuseMap;
        pbrMaterial = new PBRMaterial()
        {
            AlbedoColor = p.DiffuseColor,
            AlbedoMap = p.DiffuseMap,
            NormalMap = p.NormalMap,
            RoughnessMetallicMap =
p.SpecularColorMap,

            AmbientOcclusionMap =
p.SpecularColorMap,

            RenderShadowMap = true,
            RenderEnvironmentMap = true,
            RenderAlbedoMap = true,
            RenderDisplacementMap = true,
            RenderNormalMap = true,
            RenderAmbientOcclusionMap = true,
            EnableTessellation = false

        };
    }
    var mesh = new MeshGeometryModel3D
    {
        Geometry = ob.Geometry,

```

```

        CullMode = SharpDX.Direct3D11.CullMode.Back,
        IsThrowingShadow = true,
        Transform = scaleTransform,
        Material = pbrMaterial,
        DepthBias = i << 2
    };
    _mannequin.Add(mesh);
}
}
}

```

### **ViewPortSettings.cs**

```

public class ViewPortSettings : OnPropertyChangedClass
{
    private byte _indexOfCurrentLightPreset;
    private byte _indexOfCurrentEnvironmentMap;
    private byte _indexOfCurrentFloorTexture;
    private MannequinSettings _mannequinSettings;
    private bool _groundIsVisible;
    private List<Light3DCollection> _lightPresets;
    private List<TextureModel> _environmentMaps;
    private List<PBRMaterial> _floorTextures;
    public byte IndexOfCurrentLightPreset
    {
        get => _indexOfCurrentLightPreset;
        set
        {
            if (_indexOfCurrentLightPreset >=
                _lightPresets.Count)
                return;
            _indexOfCurrentLightPreset = value;
            OnPropertyChanged(nameof(CurrentLightPreset));
        }
    }
}

```

```

public byte IndexOfCurrentEnvironmentMap
{
    get => _indexOfCurrentEnvironmentMap;
    set
    {
        if (_indexOfCurrentEnvironmentMap >=
_environmentMaps.Count)
            return;
        _indexOfCurrentEnvironmentMap = value;
OnPropertyChanged(nameof(CurrentEnvironmentMap));
    }
}

public byte IndexOfCurrentFloorTexture
{
    get => _indexOfCurrentFloorTexture;
    set
    {
        if (_indexOfCurrentFloorTexture >=
_floorTextures.Count)
            return;
        _indexOfCurrentFloorTexture = value;
OnPropertyChanged(nameof(CurrentFloorTexture));
    }
}

public Light3DCollection CurrentLightPreset
{
    get => _lightPresets[_indexOfCurrentLightPreset];
}

public TextureModel CurrentEnvironmentMap
{
    get =>
_environmentMaps[_indexOfCurrentEnvironmentMap];
}

```



```

public PBRMaterial CurrentFloorTexture
{
    get => _floorTextures[_indexOfCurrentFloorTexture];
}

public Camera Camera { get; private set; }
public bool GroundIsVisible
{
    get => _groundIsVisible;
    set => SetProperty(ref _groundIsVisible, value);
}

private static readonly Lazy<ViewPortSettings> lazy =
new(() => new ViewPortSettings());

public static ViewPortSettings Instance { get { return
lazy.Value; } }

private ViewPortSettings()
{
    _mannequinSettings = MannequinSettings.Instance;
    Camera = new PerspectiveCamera
    {
        LookDirection = new Media3D.Vector3D(34, -9, -
303),
        UpDirection = new Media3D.Vector3D(0, 1, 0),
        Position = new Media3D.Point3D(-35, 109, 304)
    };
    InitLightPresets();
    InitEnvironmentMapsAndFloorTextures();
}

private void InitLightPresets()
{
    // Дневной пресет

```

```

        var dirLight1 = new DirectionalLight3D { Direction =
new Media3D.Vector3D(-1, -1, -1), Color = Color.FromRgb(255,
244, 214), Name = "Directional1" };

        var dirLight2 = new DirectionalLight3D { Direction =
new Media3D.Vector3D(1, 1, 1), Color = Color.FromRgb(255, 244,
214), Name = "Directional2" };

        var pointLight1 = new PointLight3D { Position = new
Media3D.Point3D(0, 200, 0), Color = Color.FromRgb(255, 244,
214), Name = "Point1", Attenuation = new Media3D.Vector3D(1,
0.02, 0.002) };

        var DayLightPreset = new Light3DCollection();
        DayLightPreset.Children.Add(dirLight1);
        DayLightPreset.Children.Add(dirLight2);
        DayLightPreset.Children.Add(pointLight1);
        DayLightPreset.Tag = "preset_daylight";

        // Ночной пресет

        var dirLight3 = new DirectionalLight3D { Direction =
new Media3D.Vector3D(-1, -1, -1), Color = Color.FromRgb(180,
180, 180), Name = "Directional1" };

        var dirLight4 = new DirectionalLight3D { Direction =
new Media3D.Vector3D(1, 1, 1), Color = Color.FromRgb(150, 150,
150), Name = "Directional2" };

        var pointLight2 = new PointLight3D { Position = new
Media3D.Point3D(0, 200, 0), Color = Color.FromRgb(200, 200,
200), Name = "Point1",
            Attenuation = new Media3D.Vector3D(1, 0.02,
0.002), Range = 500 };

        var NightLightPreset = new Light3DCollection();
        NightLightPreset.Children.Add(dirLight3);
        NightLightPreset.Children.Add(dirLight4);
        NightLightPreset.Children.Add(pointLight2);
        NightLightPreset.Tag = "preset_nightlight";

        var light7 = new DirectionalLight3D { Direction =
new Media3D.Vector3D(-1, -1, -1), Color = Colors.White, Name =
"Directional1" };

```

```

        var light8 = new DirectionalLight3D { Direction =
new Media3D.Vector3D(1, 1, 1), Color = Colors.White, Name =
"Directional2" };

        var light9 = new PointLight3D { Position = new
Media3D.Point3D(0, 0, 500), Color = Colors.White, Name = "Point"
};

        var collection = new Light3DCollection();
        collection.Children.Add(light7);
        collection.Children.Add(light8);
        collection.Children.Add(light9);
        collection.Tag = "preset_classic3pointlight";

        _lightPresets = new List<Light3DCollection>
        {
            DayLightPreset,
            NightLightPreset,
            collection
        };
    }

    private void InitEnvironmentMapsAndFloorTextures()
    {
        var pathOfMaps =
        $"{AppDomain.CurrentDomain.BaseDirectory}Resources\\Environment\\
        \";

        var indoorStudio =
        TextureModel.Create($"{pathOfMaps}IndoorStudio.dds");

        var Night =
        TextureModel.Create($"{pathOfMaps}Night.dds");

        var Midday =
        TextureModel.Create($"{pathOfMaps}Midday.dds");

        _environmentMaps = new List<TextureModel>
        {
            Midday,
            Night,
            indoorStudio,
        };
    }

```

```

var flIndoor = new PBRMaterial()
{
    AlbedoMap =
TextureModel.Create($"{pathOfMaps}\\Floor\\indoor_ny.png"),
    RenderAlbedoMap = true,
    RenderShadowMap = true,
    AlbedoColor = new SharpDX.Color4(1, 1, 1, 1),
    RenderEnvironmentMap = true,
    EnableAutoTangent = true,
    EmissiveColor = new SharpDX.Color4(0.5f, 0.5f,
0.5f, 1),
};

var flDay = new PBRMaterial()
{
    AlbedoMap =
TextureModel.Create($"{pathOfMaps}\\Floor\\day_ny.png"),
    RenderAlbedoMap = true,
    RenderShadowMap = true,
    AlbedoColor= new SharpDX.Color4(1,1,1,1),
    RenderEnvironmentMap = true,
    EnableAutoTangent = true,
    EmissiveColor = new SharpDX.Color4(0.5f, 0.5f,
0.5f, 1),

};

var flNight = new PBRMaterial()
{
    AlbedoMap =
TextureModel.Create($"{pathOfMaps}\\Floor\\night_ny.png"),
    RenderAlbedoMap = true,
    RenderShadowMap = true,
    AlbedoColor = new SharpDX.Color4(1, 1, 1, 1),
    RenderEnvironmentMap = true,

```

```

        EnableAutoTangent = true,
        EnableFlatShading = true,
        EmissiveColor = new SharpDX.Color4(0.1f, 0.1f,
0.1f, 0.9f),
    };
    _floorTextures = new List<PBRMaterial>
    {
        flDay,
        flNight,
        flIndoor,
    };
}
}

```

#### **OnPropertyChangedClass.cs**

```

    /// <summary>Базовый класс с реализацией INPC </summary>
    public abstract class OnPropertyChangedClass :
INotifyPropertyChanged
    {
        #region Событие PropertyChanged
        /// <summary>Событие для извещения об изменении
свойства</summary>
        public event PropertyChangedEventHandler?
PropertyChanged;
        #endregion

        #region Методы вызова события PropertyChanged
        /// <summary>Метод для вызова события извещения об
изменении свойства</summary>
        /// <param name="propertyName">Изменившееся
свойство</param>
        protected void OnPropertyChanged([CallerMemberName]
string propertyName = "")
        {
            => PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
        }
    }

```

```

        /// <summary>Метод для вызова события извещения об
изменении списка свойств</summary>

        /// <param name="propList">Список имён свойств</param>
        protected void OnPropertyChanged(IEnumerable<string>
propList)
        {
            foreach (string propertyName in propList.Where(name
=> !string.IsNullOrEmpty(name)))
            {
                PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
            }
        }

        /// <summary>Метод для вызова события извещения об
изменении перечня свойств</summary>

        /// <param name="propList">Список имён свойств</param>
        protected void OnPropertyChanged(params string[]
propList)
        {
            foreach (string propertyName in propList.Where(name
=> !string.IsNullOrEmpty(name)))
            {
                PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
            }
        }

        /// <summary>Метод для вызова события извещения об
изменении всех свойств</summary>

        /// <param name="propList">Список свойств</param>
        protected void OnAllPropertyChanged()
        {
            => PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(null));
        }

#endregion

#region Виртуальные защищённые методы для изменения
значений свойств

        /// <summary>Виртуальный метод определяющий изменения в
значении поля значения свойства</summary>

```

```

        /// <param name="fieldProperty">Ссылка на поле значения
свойства</param>

        /// <param name="newValue">Новое значение</param>

        /// <param name="propertyName">Название свойства</param>

        protected virtual void SetProperty<T>(ref T
fieldProperty, T newValue, [CallerMemberName] string
propertyName = "")
        {
            if ((fieldProperty != null &&
!fieldProperty.Equals(newValue)) || (fieldProperty == null &&
newValue != null))
                PropertyNewValue(ref fieldProperty, newValue,
propertyName);
        }

        /// <summary>Виртуальный метод изменяющий значение поля
значения свойства</summary>

        /// <param name="fieldProperty">Ссылка на поле значения
свойства</param>


        /// <param name="newValue">Новое значение</param>

        /// <param name="propertyName">Название свойства</param>


        protected virtual void PropertyNewValue<T>(ref T
fieldProperty, T newValue, string propertyName)
        {
            fieldProperty = newValue;
            OnPropertyChanged(propertyName);
        }
    #endregion
}

```


Проверка на антиплагиат



АНТИПЛАГИАТ  
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ


 Участник

ТАРИФ

Free 


ИЗМЕНИТЬ

ПРОВЕРКИ

Недоступны 


ПРОВЕРИТЬ ДОКУМЕНТ

ПОЛЬЗОВАТЕЛЬ

 timtukanal@gmail.com

ВОЙТИ В КАБИНЕТ

МЕНЮ

ru 

ГЛАВНАЯ / КАБИНЕТ / РЕЗУЛЬТАТЫ ПРОВЕРКИ

Оригинальность

92.74%

Совпадения

7.26%

Цитирования

0%


Самоцитирования


0%


ПОЛНЫЙ ОТЧЕТ

КРАТКИЙ ОТЧЕТ

ИСТОРИЯ ОТЧЕТОВ

РАСПЕЧАТАТЬ 

ВЫГРУЗИТЬ 

СОЗДАТЬ ССЫЛКУ 

Свойства документа


Структура документа

Текстовые метрики 

NEW

Параметры проверки

Имя исходного файла

Авторы документа 

Название документа

Тип документа

Курсовая.pdf

Кочнев

Соловьев

ООО/ИД\_Пояснительная\_Записка\_Приложение\_для\_подбора\_одежды

Курсовая работа

Кирилл Вадимович

Даниил Витальевич