Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра проектирования информационных компьютерных систем

Дисциплина "Объектно-ориентированное программирование"

К защите допустить:
Руководитель курсовой работы
старший преподаватель
кафедры
_____ А.В.Михалькевич
22.12.2024

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

Разработка книжного интернет-магазина

БГУИР КР 1-40 05 01-10 № 157 ПЗ

Студент (подпись студента) И.В. Антоненко

Курсовая работа
представлена на проверку
22.12.2024
_____
(подпись студента)

Ельск 2024

Реферат

БГУИР КР 1-40 05 01-10 № 157 ПЗ, гр. 814302

И.В. Антоненко, Разработка книжного интернет-магазина, Ельск: БГУИР - 2024.

Пояснительная записка 75304 с., 20 рис., 0 табл.

Ключевые слова: C#, книжный интернет-магазин, MVC, трагедия, ужасы, драма, комедия

Предмет Объектно-ориентированное программирование, А.В.Михалькевич

Предмет: создание веб-приложения Объект: шаблоны проектирования, разработка пользовательского интерфейса. Цель: Верстка и программирование книжного интернет-магазина на языке C# с помощью технологии MVC. Методология проведения работы: в процессе решения поставленных задач спроектирован и разработан простой и удобный интерфейс веб-приложения, разработана логика приложения, изучена и применена работа с базой данных. Результаты работы: разработан простой сайт без лишних функций, чтобы не напрягать пользователей излишними формами. Интерфейс сайта был максимально упрощен и в меру информативен, чтобы вся важная информация, была на главной странице. Область применения результатов: Поскольку львиной доле населения планеты не чуждо самообразование и чтение книг, сайт будет пользоваться спросом безоговорочно, как дети так и взрослые могут приобрести необходимые книжки для развития своих интеллектуальных возможностей, а также для расширения кругозора. Ссылка на онлайн-репозиторий: GitHub:https://github.com/CattyJay/bookshop

Subject: creating a web application Object: design patterns, user interface development. Purpose: Layout and programming of an online bookstore in C # using MVC technology. Methodology of the work: in the process of solving the tasks, a simple and convenient web application interface was designed and developed, the application logic was developed, the work with the database was studied and applied. Results: designed a simple site without unnecessary features to. The interface of the site was maximally simplified and reasonably informative so that all the important information was on the main page. Scope of the results: Since the lion's share of the world's population is not alien to self-education and reading books, the site will be in demand unconditionally, both children and adults can purchase the necessary books to develop their intellectual capabilities, as well as to broaden their horizons. Link to online repository: GitHub:https://github.com/CattyJay/bookshop

Содержание

Введение
1 ОПИСАНИЕ ПРОЕКТА
2 ОБОСНОВАНИЕ ВЫБОРА ТЕХНОЛОГИИ
3 ИНСТРУМЕНТАРИЙ
4 АРХИТЕКТУРНЫЙ ШАБЛОН ПРОЕКТИРОВАНИЯ MVC
5 ШАБЛОН ПРОЕКТИРОВАНИЯ ГРАФИЧЕСКИХ ЗАДАЧ
6 ОПИСАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА
Заключение
Список использованных источников
Приложения

Введение

Как всем известно, книги читает 100% населения планеты, и разумеется должны существовать сервисы для удовлетворения желания читателей. Такие сервисы как раз есть, это сервисы для онлайн-чтения, для скачивания книг в электронном формате, а также для прослушивания аудиокниг. Но как же бумажный и более традиционный вариант книг. Конечно в наше время должны существовать книжные магазины, но сейчас же век цифровизации и высоких технологий, и вдруг ты заказал себе доставку пары пицц, после съедения которых тебе вряд ли захочется куда-либо идти, а именно в этот момент у тебя появилось невероятное желание купить какую-нибудь книжечку, например, про 7 смертных грехов, одним из которых является обжорство, или кулинарную книгу для воплощения своих аппетитных мечт в реальность. В такой ситуации непременно может помочь онлайн-сервис по продаже книг и журналов. Это идеальный вариант для любителей посидеть дома и не выбираться на улицу. Вы просто заходите на этот сервис и в 2 клика приобретаете необходимую книжку без лишних усилий. Как раз мой проект полностью подходит под это описание, потому что на нём даже не требуется авторизация/регистрация, которая, как известно, вызывает негодование у многих людей, которые заходят на этот сайт, например, раз в неделю, а то и реже. Для чего это вечное требование авторизоваться, если за раз можно купить все необходимые товары, что помогает избежать повторного посещения сайта. А то, что на сайт заходят нечасто, это не значит, что он плохой, в этом просто нет необходимости. Но сайт же изначально создавался не для коммерции, а чтоб предоставить услуги по продаже книг. Поэтому всё сделано практично и удобно. Целью моей курсовой работы является создание как раз такого сервиса по продаже книг онлайн с помощью языка C# и его фреймворка ASP.NET Core. Используемая технология – MVC(модель-представление-контроллер).

1 ОПИСАНИЕ ПРОЕКТА

Проект представляет собой книжный интернет-магазин, и вся суть магазина заключается в том, чтобы приобрести книгу. Со стороны бэка требуется следующий функционал: добавление книг на сайт, возможность просмотра товаров разных категорий, просмотр дополнительной информации по товару, возможность добавления товара в корзину для последующего оформления заказа и непосредственного приобретения этого самого товара.

Всё начинается с создания моделей, так как проект реализован с помощью технологии MVC(Model-View-Controller), а именно, разрабатывая книжный магазин, нам разумеется нужна модель книги (рис.1.1), а также модель категории (рис.1.2), к которой каждая книга будет относится:

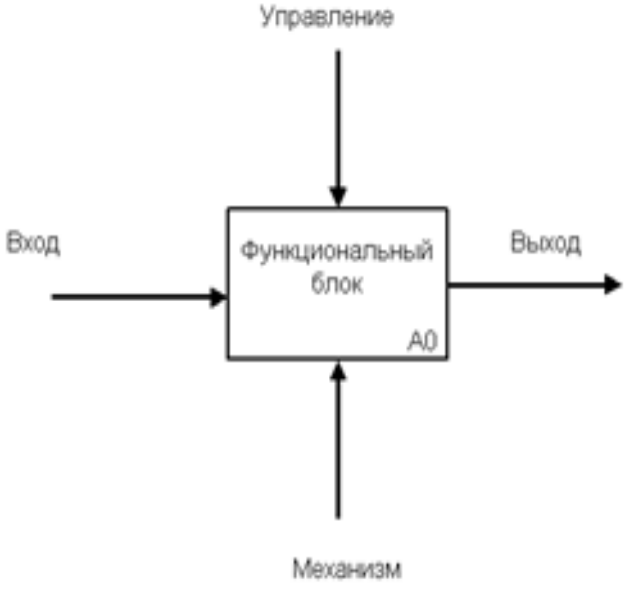

Рисунок 1.1 – Модель Book

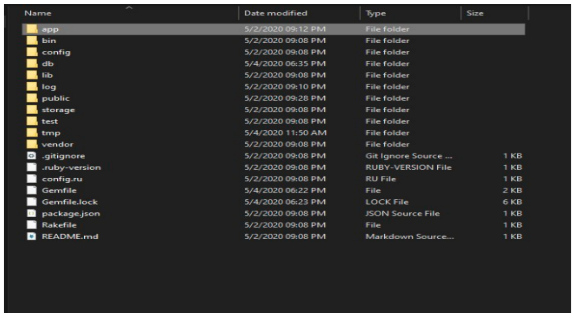

Рисунок 1.2 – Модель Category

Итак, фундамент для создания книг построен, теперь нужно добавить сами экземпляры книг. Во-первых хранятся данные на локальном сервере базы данных SQL, который подключается к проекту с помощью разных пакетов, во-вторых добавление объектов книг и создание соответствующих категорий происходит именно в коде непосредственно в саму базу данных. Это происходит в классе DBObjects в статической функции Initial() (рис.1.3)

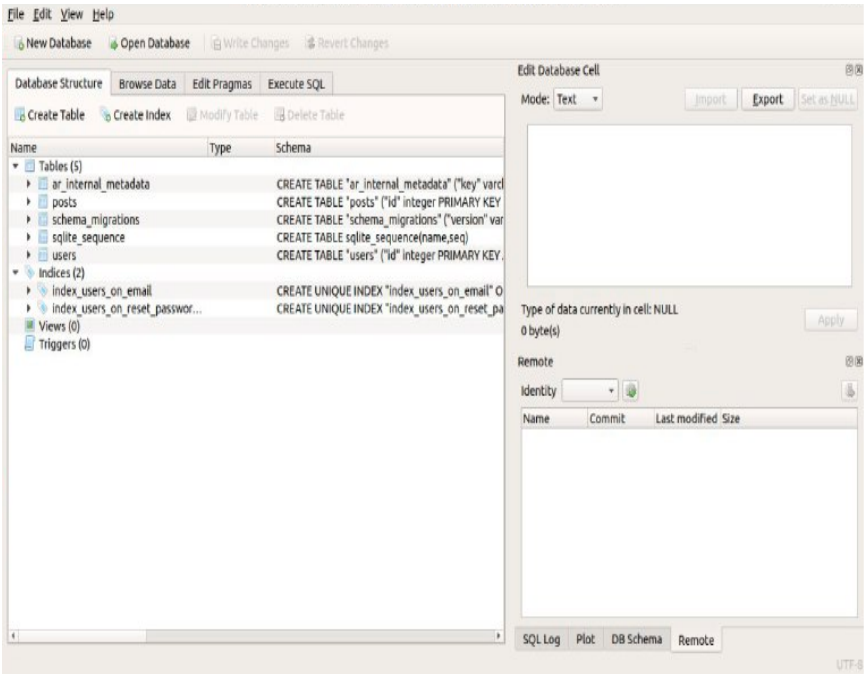

Рисунок 1.3 – Функция Initial()

Далее по технологии MVC следует создание контроллера, который связывает модель с отображением её на странице у пользователя. Так как отображение категорий нам ни к чему, то контроллер (рис.1.4) создаётся только для книг:

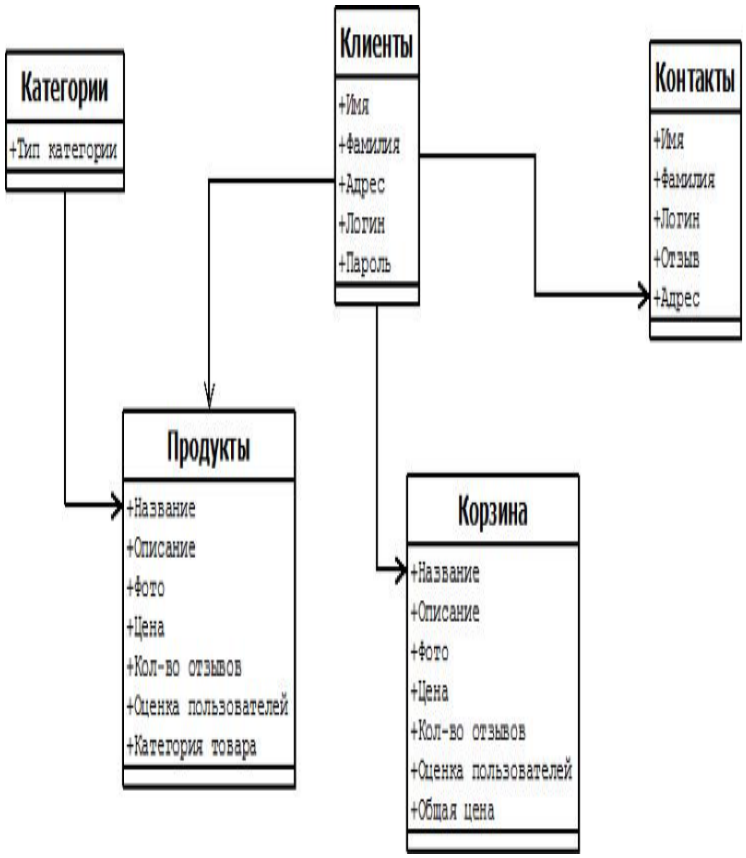

Рисунок 1.4 – Контроллер BookController

В нём есть функция List типа ViewResult, которая возвращает представление, отображающее все книги на странице. В файле AllBooks.cshtml (рис.1.5), производится вывод всех книг той категории, которой мы захотим, а также к каждому объекту привязана кнопка «Добавить в

корзину», которая обращается к контроллеру ShopCartController.

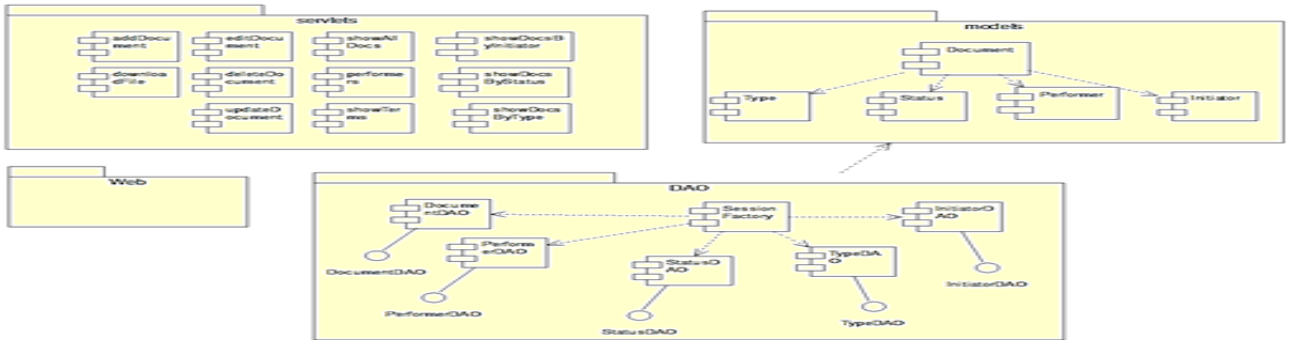

Рис 1.5 – Представление AllBooks.cshtml

Этот контроллер ShopCartController (рис.1.6) связывает модель корзины, в которой содержатся все книги, добавленные ранее, и представление этой корзины с объектами на странице.

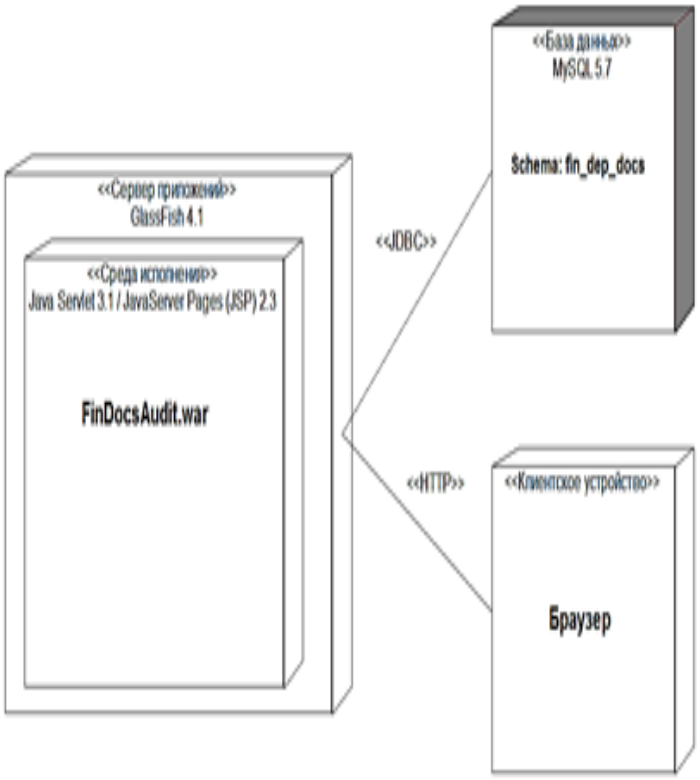

Рисунок 1.6 – Контроллер ShopCartController

Представление данных в корзине описано в файле Index.cshtml (рис.1.7), где помимо вывода списка всех книг, добавленных в неё, с соответствующими ценами, присутствует кнопка «Оплатить», переводящая нас на форму заполнения данных для оформления заказа и непосредственной покупки товаров.

Рисунок 1.7 - Представление Index.cshtml

Таким образом, дойдя до оформления заказа, появляется необходимость создания следующей модели, а именно модели заказа (рис.1.8).

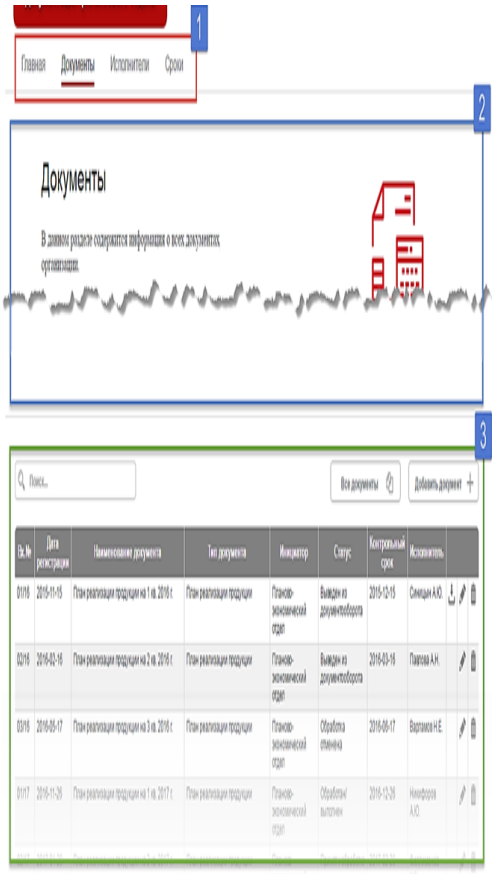

Рисунок 1.8 – Модель Order

Здесь наблюдаются элементы пакета TagHelpers, которые используются для работы на странице, так как эти данные модели Order, вводимые пользователем, отправляются в базу данных и не могут быть абы какими, то необходимо проверять их корректность. Следовательно, для каждого поля устанавливается формат, тип вводимых данных, а также максимальный размер, вводимой строки. И после ввода корректных данных происходит переадресация на завершающую страницу, на которой отображается сообщение об успешно завершённом заказе. Все данные добавляются в базу данных в таблицу Order. После чего продавец связывается с заказчиком и происходит товарно-денежный обмен, в результате которого все удовлетворят

свои желания. Это то, для чего магазин и был создан.

2 ОБОСНОВАНИЕ ВЫБОРА ТЕХНОЛОГИИ

2.1 Технологии программирования

Существует огромное количество технологий программирования, из которых сложно выбрать ту самую, которая больше всего подойдет. Этот проект был написан с помощью технологии ASP.NET Core на языке C# совместно html и css файлов для работы с front-end.

ASP.NET Core – свободно-распространяемый кросс-платформенный фреймворк для создания веб-приложений с открытым исходным кодом.

Платформа ASP.NET Core представляет технологию от компании Microsoft, предназначенную для создания различного рода веб-приложений: от небольших веб-сайтов до крупных веб-порталов и веб-сервисов.

С одной стороны, ASP.NET Core является продолжением развития платформы ASP.NET. Но с другой стороны, это не просто очередной релиз. Выход ASP.NET Core фактически означает революцию всей платформы, ее качественное изменение.

Разработка над платформой началась еще в 2014 году. Тогда платформа условно называлась ASP.NET vNext. В июне 2016 года вышел первый релиз платформы. А в декабре 2019 года вышла версия ASP.NET Core 3.1, которая собственно и будет охвачена в текущем руководстве.

ASP.NET Core теперь полностью является opensource-фреймворком. Все исходные файлы фреймворка доступны на GitHub.

ASP.NET Core может работать поверх кросс-платформенной среды .NET Core, которая может быть развернута на основных популярных операционных системах: Windows, Mac OS, Linux. И таким образом, с помощью ASP.NET Core мы можем создавать кросс-платформенные приложения. И хотя Windows в качестве среды для разработки и развертывания приложения до сих пор превалирует, но теперь уже мы не ограничены только этой операционной системой. То есть мы можем запускать веб-приложения не только на ОС Windows, но и на Linux и Mac OS. А для развертывания веб-приложения можно использовать традиционный IIS, либо кросс-платформенный веб-сервер Kestrel.

Благодаря модульности фреймворка все необходимые компоненты веб-приложения могут загружаться как отдельные модули через пакетный менеджер Nuget. Кроме того, в отличие от предыдущих версий платформы нет необходимости использовать библиотеку System.Web.dll.

ASP.NET Core включает в себя фреймворк MVC, который объединяет функциональность MVC, Web API и Web Pages. В предыдущих версии платформы данные технологии реализовались отдельно и поэтому содержали много дублирующей функциональности. Сейчас же они объединены в одну программную модель ASP.NET Core MVC. А Web Forms полностью ушли в прошлое.

Кроме объединения вышеупомянутых технологий в одну модель в MVC был добавлен ряд дополнительных функций.

Одной из таких функций являются тэг-хелперы (tag helper), которые позволяют более органично соединять синтаксис html с кодом C#.

ASP.NET Core характеризуется расширяемостью. Фреймворк построен из набора относительно независимых компонентов. И мы можем либо использовать встроенную реализацию этих компонентов, либо расширить их с помощью механизма наследования, либо вовсе создать и применять свои компоненты со своим функционалом.

Также было упрощено управление зависимостями и конфигурирование проекта. Фреймворк теперь имеет свой легковесный контейнер для внедрения зависимостей, и больше нет необходимости применять сторонние контейнеры, такие как Autofac, Ninject. Хотя при желании их также можно продолжать использовать.

В качестве инструментария разработки мы можем использовать последние выпуски Visual Studio, начиная с версии Visual Studio 2015. Кроме того, мы можем создавать приложения в среде Visual Studio Code, которая является кросс-платформенной и может работать как на Windows, так и на Mac OS X и Linux.

Для обработки запросов теперь используется новый конвейер HTTP, который основан на компонентах Katana и спецификации OWIN. А его модульность позволяет легко добавить свои собственные компоненты.

Если суммировать, то можно выделить следующие ключевые отличия ASP.NET Core от предыдущих версий ASP.NET:

- Новый легковесный и модульный конвейер HTTP-запросов
- Возможность развертывать приложение как на IIS, так и в рамках своего собственного процесса
- Использование платформы .NET Core и ее функциональности
- Распространение пакетов платформы через NuGet
- Интегрированная поддержка для создания и использования пакетов NuGet
- Единый стек веб-разработки, сочетающий Web UI и Web API
- Конфигурация для упрощенного использования в облаке
- Встроенная поддержка для внедрения зависимостей
- Расширяемость
- Кроссплатформенность: возможность разработки и развертывания приложений ASP.NET на Windows, Mac и Linux
- Развитие как open source, открытость к изменениям

Эти и другие особенности и возможности стали основой для новой модели программирования.

Сразу после создания проекта, предлагается выбрать шаблон проектирования (рис.2.2), который сильно облегчит задачу программисту и сэкономит уйму времени, в моём же случае был выбран пустой шаблон, чтобы понять как происходит создание веб-приложения с помощью технологии MVC с самостоятельным добавление всех необходимых пакетов и библиотек.

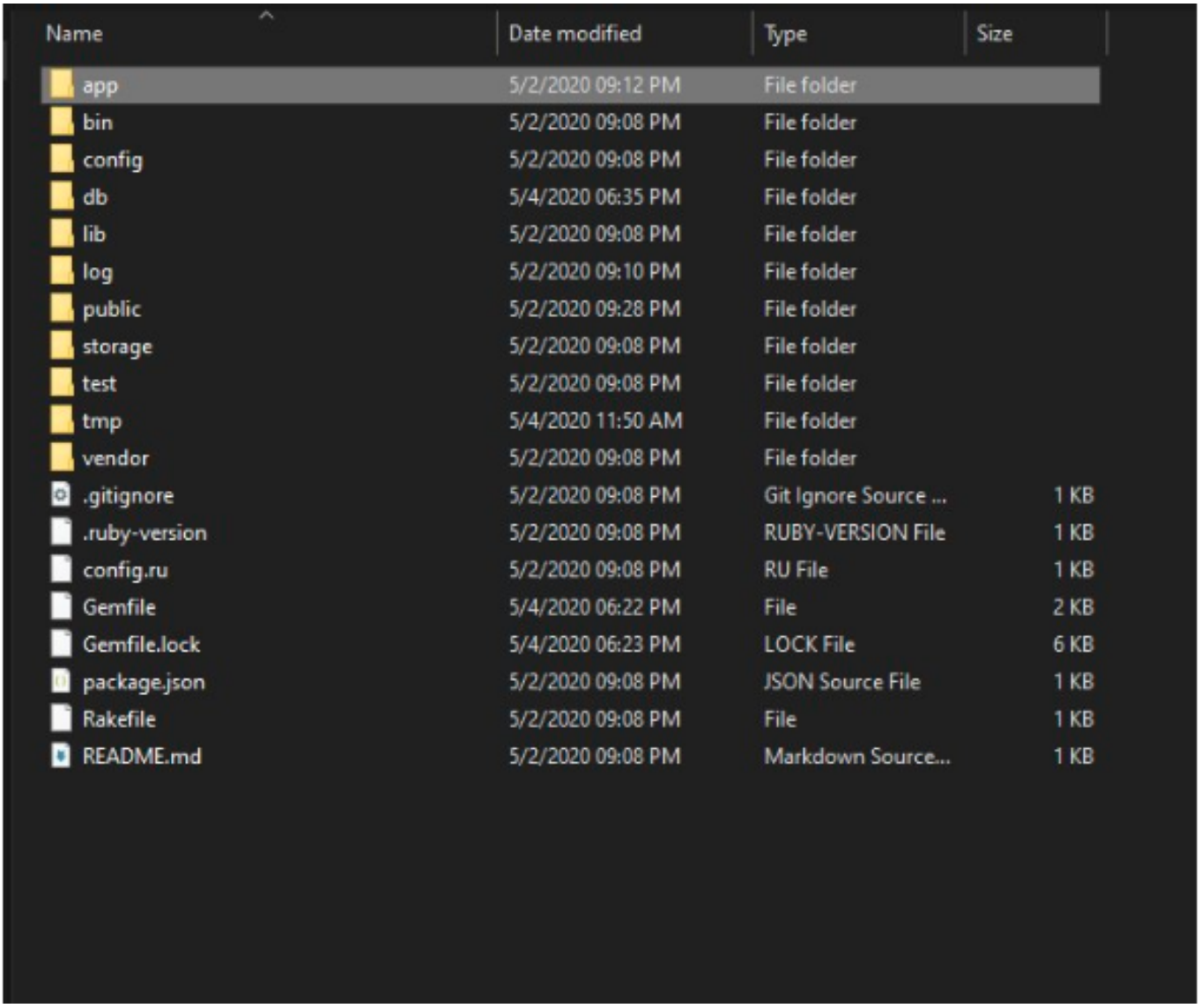

Рисунок 2.2 – Выбор шаблона проектирования

После создания проекта выходит не совсем пустой проект, прописаны некоторые универсальные и используемые везде файлы и классы. И в начальном классе, с которого стартует весь проект, Startup.cs, где мы как раз и подключаем технологию MVC в функции Configure() (рис.2.3)

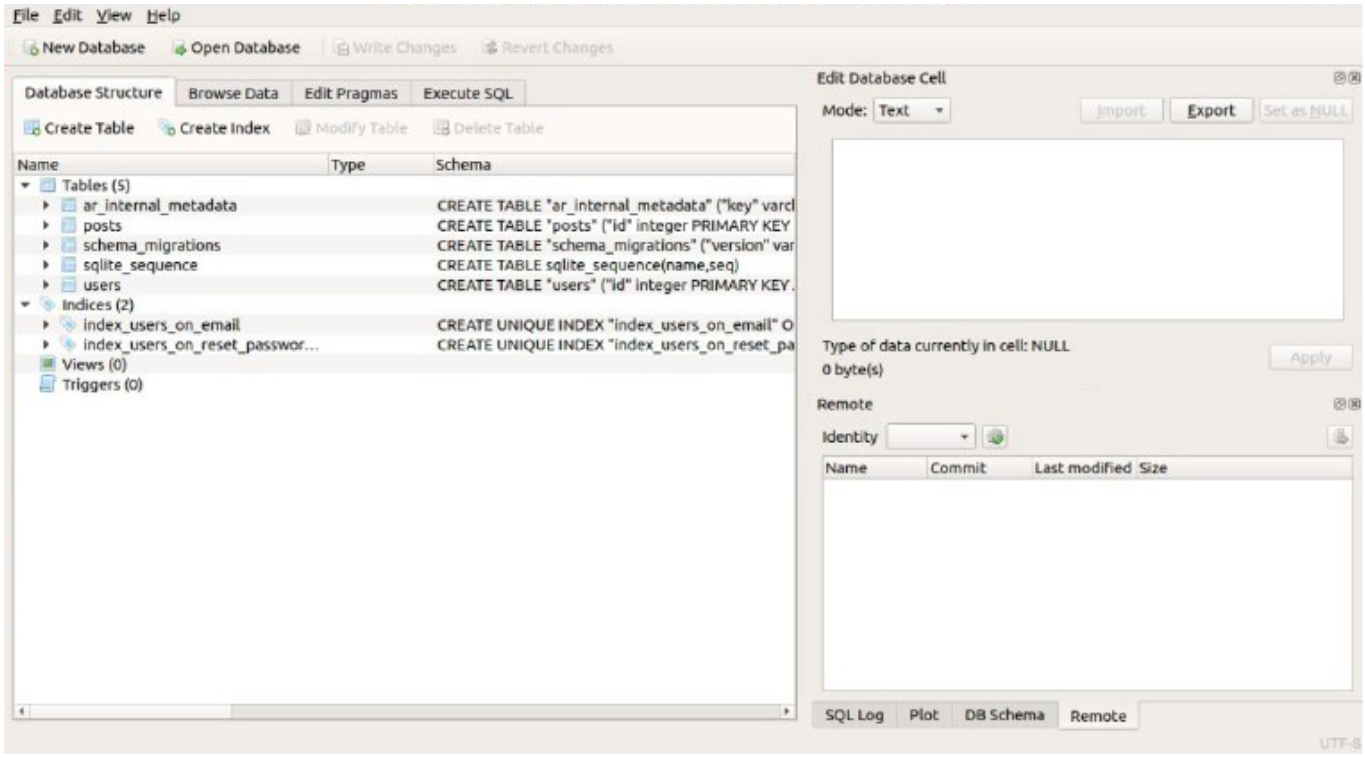

Рисунок 2.3 — Функция Configure()

После подключения основных зависимостей связанных с самой технологией MVC потребуется подключение различных пакетов (рис.2.4) для работы со статическими файлами, с базой данных и прочими мелочами.

Рисунок 2.4 – Все подключаемые пакеты в проект

2.2 Реализация ООП технологий

Для парадигмы ООП характерно, что программист рассматривает программу в виде набора взаимодействующих объектов, в то время как, например, в функциональном программировании программа представляется в виде последовательности вычисления функций. Процедурное программирование или, как его еще называют, классическое операциональное, подразумевает написание алгоритма для решения задачи; при этом ожидаемые свойства конечного результата не описываются и не указываются. Структурное программирование в основном придерживается тех же принципов, что и процедурное, лишь немного дополняя их полезными приемами.

Парадигмы непроцедурного программирования, к которым можно отнести объектно-ориентированную парадигму, имеют совершенно другие идеи. Определение Гради Буча гласит: “Объектно-ориентированное программирование — это методология программирования, которая основана на представлении программы в виде совокупности объектов, каждый из которых является реализацией определенного класса (типа особого вида), а классы образуют иерархию на принципах наследуемости”. Важно подметить, что, по определению, программа считается объектно-ориентированной, только если выполнены следующие требования:

1. объектно-ориентированное программирование использует в качестве основных логических конструктивных элементов объекты, а не алгоритмы;
2. каждый объект является экземпляром определенного класса;
3. классы образуют иерархии.

Идеологически ООП — подход к программированию как к моделированию информационных объектов, решающий на новом уровне основную задачу структурного программирования: структурирование информации с точки зрения управляемости, что существенно улучшает управляемость самим процессом моделирования, что, в свою очередь, особенно важно при реализации крупных проектов.

Другим фундаментальным понятием является класс. Класс – это шаблон, на основе которого может быть создан конкретный программный объект, он описывает свойства и методы, определяющие поведение объектов этого класса. Каждый конкретный объект, имеющий структуру этого класса, называется экземпляром класса.

Следующими важнейшими принципами (технологиями) ООП являются:

1. Инкапсуляция – это механизм, объединяющий атрибуты и методы и охраняющий их от внешнего вмешательства (рис.2.5);

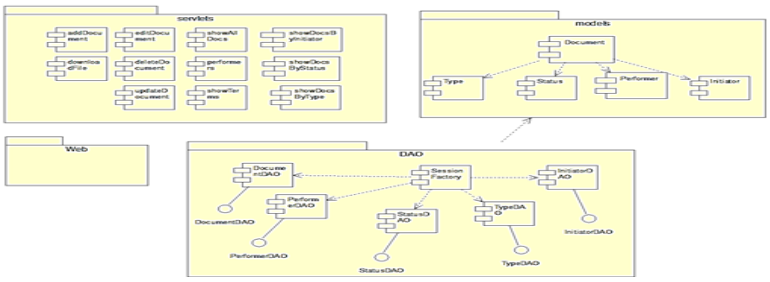

Рисунок 2.5 — Пример инкапсуляции в курсовой работе

1. Наследование – это механизм расширения классов, который позволяет использовать уже существующие классы для описания новых (рис.2.6);

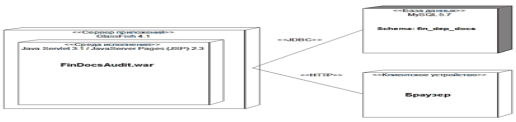

Рисунок 2.6 — Пример наследования в курсовой работе

1. Полиморфизм – концепция ООП, которая означает, что рожденные объекты обладают информацией о том, какие методы они должны использовать в зависимости от того, в каком месте цепочки (дерева классов) они находятся, иными словами – это концепция, реализующая "множество методов в одном интерфейсе".
2. Абстракция – выделение наиболее важных характеристик и информации об объекте.
3. Модульность – это такая организация объектов, когда они заключают в себе полное определение их характеристик, никакие определения методов и свойств не должны располагаться вне его, это делает возможным свободное копирование и внедрение одного объекта в другие (рис.2.7).

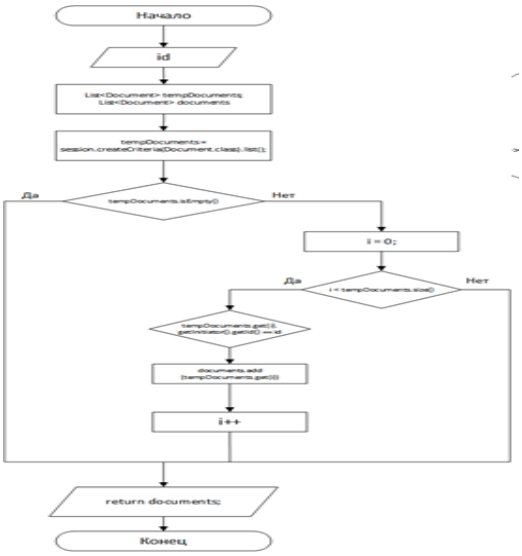

Рисунок 2.7 — Пример модульности в курсовой работе

Общими ключевыми понятиями, с которыми работает любой объектно-ориентированный язык являются:

Объект – сущность в адресном пространстве вычислительной системы, появляющаяся при создании экземпляра класса;
Метод – функция класса;
Свойство – переменная, определяемые в классе;
Спецификаторы доступа: public, private, protected – определение доступа к методам или свойствам класса;
Абстрактный класс – класс, от которого нельзя создать объект, предназначен только для дальнейшего наследования;

Финальный класс – класс, от которого нельзя наследоваться, предназначен только для создания объектов, но не для дальнейшего наследования;
Конструктор – метод, вызываемый в момент создания объекта;
Деструктор – метод, вызываемый в момент уничтожения объекта;
Константа – неизменное свойства класса;
Интерфейс – это класс, в котором все методы являются абстрактными и открытыми и задаются только имена методов и их параметры, а реализованы они могут быть позже;

Имплемент – это класс, реализующий методы интерфейса.

3 ИНСТРУМЕНТАРИЙ

3.1 Используемые инструменты

В работе любого современного разработчика существуют инструменты, ставшие практически обязательными для каждодневного применения. Такими инструментами, в первую очередь, будут, конечно, интегрированная среда разработки (IDE) и система управления базами данных (СУБД). Конечно, опытный программист может обойтись и без использования IDE и СУБД, но это существенно скажется на таких моментах, как скорость разработки, качество кода и, конечно же, на удобстве самого процесса написания этого кода.

В своем проекте я использовал Microsoft Visual Studio и СУБД MSSQL.

Интегрированная среда разработки (англ. Integrated development environment — IDE) — комплекс программных средств, используемый программистами для разработки программного обеспечения (ПО).

Среда разработки включает в себя:

Текстовый редактор;
Транслятор (компилятор и/или интерпретатор);
Средства автоматизации сборки;
Отладчик.

Microsoft Visual Studio — линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние

дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как, например, Subversion и Visual SourceSafe), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения (например, клиент Team Explorer для работы с Team Foundation Server).

Система управления базами данных (СУБД) — комплекс программ, позволяющих создать базу данных и манипулировать данными (вставлять, обновлять, удалять и выбирать). Система обеспечивает безопасность, надёжность хранения и целостность данных, а также предоставляет средства для администрирования БД.

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.

Возможности и преимущества Microsoft SQL Server 2017

Межплатформенная совместимость
Снижение общей стоимости владения
Улучшенные сервисы статистического анализа и анализа данных
Лучшая в своем классе безопасность
Межплатформенный Visual Studio
Поддержка Linux

Для запуска веб-приложения и откладки ее в браузере использовался веб-браузер Chrome. Его инструментария полностью хватает для успешной разработки приложения. Он оснащён мощными инструментами для веб-разработчика. Эти инструменты позволяют производить различные операции, от изучения загруженных в настоящий момент HTML, CSS и JavaScript до отображения в каких ресурсах нуждается страница и как долго она будет загружаться.

3.2 GIT

Система управления версиями (Version Control System) — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы. Однако они могут с успехом

применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов.

Git — распределённая система управления версиями. Система спроектирована как набор программ, специально разработанных с учётом их использования в сценариях. Это позволяет удобно создавать специализированные системы контроля версий на базе Git или пользовательские интерфейсы. Удалённый доступ к репозиториям Git обеспечивается git-демоном, SSH- или HTTP-сервером.

GitHub — крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки. Веб-сервис основан на системе контроля версий Git. Сервис бесплатен для проектов с открытым исходным кодом и небольших частных проектов, предоставляя им все возможности (включая SSL), а для крупных корпоративных проектов предлагаются различные платные тарифные планы.

Создатели сайта называют GitHub «социальной сетью для разработчиков». Кроме размещения кода, участники могут общаться, комментировать правки друг друга, а также следить за новостями знакомых.

С помощью широких возможностей Git программисты могут объединять свои репозитории — GitHub предлагает удобный интерфейс для этого и может отображать вклад каждого участника в виде дерева. Для проектов есть личные страницы, небольшие Вики и система отслеживания ошибок. Прямо на сайте можно просмотреть файлы проектов с подсветкой синтаксиса для большинства языков программирования. Можно создавать приватные репозитории, которые будут видны только вам и выбранным вами людям.

4 АРХИТЕКТУРНЫЙ ШАБЛОН ПРОЕКТИРОВАНИЯ MVC

Термин модель-представление-контроллер (model-view-controller) был в употреблении с конца 1970-х годов и происходит из проекта Smalltalk в Xerox PARC, где он был задуман как способ организации ряда ранних приложений с графическим пользовательским интерфейсом. Некоторые нюансы первоначального паттерна MVC были связаны с концепциями, специфичными для Smalltalk, такими как экраны и инструменты, но более широкие понятия по-прежнему применимы к приложениям — и особенно хорошо они подходят для веб-приложений.

Если оперировать высокоуровневыми понятиями, то паттерн MVC означает, что приложение MVC будет разделено, по крайней мере, на три указанные далее части.

• Модели, содержащие или представляющие данные, с которыми работают пользователи.

• Представления, используемые для визуализации некоторой части модели в виде пользовательского интерфейса.

• Контроллеры, которые обрабатывают входящие запросы, выполняют операции с моделью и выбирают представления для визуализации пользователю. Каждая порция архитектуры MVC четко определена и самодостаточна; такое положение вещей называют разделением обязанностей. Логика, которая манипулирует данными в модели, содержится только в модели. Логика, отображающая данные, присутствует только в представлении. Код. который обрабатывает пользовательские запросы и ввод, находится только в контроллере. Благодаря

ясному разделению между порциями приложение будет легче сопровождать и расширять на протяжении его времени существования вне зависимости от того, насколько большим оно станет.

Модели (M в MVQ содержат данные, с которыми работают пользователи. Существуют два обширных типа моделей: модели представлений, которые выражают сами данные, передаваемые из контроллера в представление, и модели предметной области, которые содержат данные в предметной области наряду с операциями, трансформациями и правилами для создания, хранения и манипулирования данными, все вместе называемыми логикой моделей. Модели — это определение 'вселенной", в которой функционирует приложение. Например, в банковском приложении модель представляет все аспекты банковской деятельности, поддерживаемые приложением, такие как расчетные счета, главная бухгалтерская книга и кредитные лимиты для клиентов, равно как и операции, которые могут применяться для манипулирования данными в модели, подобные внесению денежных средств и списанию их со счетов. Модель отвечает также за сохранение общего состояния и целостности данных — например, удостоверяясь, что все транзакции внесены в главную книгу, а клиент не снимает со счета больше денежных средств, чем имеет на то право, или больше, чем находится в распоряжении самого банка. Для каждого компонента в паттерне MVC будет описано, что он должен включать, а что не должен. Модель в приложении, построенном с использованием паттерна MVC, должна:

• содержать данные предметной области;

• содержать логику для создания, управления и модификации данных предметной области:

• предоставлять чистый API-интерфейс, который открывает доступ к данным модели и операциям с ними. Модель не должна:

• показывать детали того, как осуществляется получение или управление данными модели (другими словами, подробности механизма хранения данных не должны быть видны контроллерам и представлениям);

• содержать логику, которая трансформирует модель на основе взаимодействия с пользователем (поскольку это работа контроллера);

• содержать логику для отображения данных пользователю (т.к. это работа представления). Преимущества обеспечения изоляции модели от контроллера и представлений заключаются в том, что вы можете гораздо легче тестировать логику и проще расширять и сопровождать приложение в целом.

Контроллеры являются “соединительной тканью" паттерна MVC, исполняя роль каналов между моделью данных и представлениями. Контроллеры определяют действия, предоставляющие бизнес-логику, которая оперирует на модели данных и обеспечивает представления данными, подлежащими отображению для пользователя. Контроллер, построенный с применением паттерна MVC, должен:

• содержать действия, требующиеся для обновления модели на основе взаимодействия с пользователем. Контроллер не должен:

• содержать логику, которая управляет внешним видом данных (это работа представления);

• содержать логику, которая управляет постоянством данных (это работа модели).

Представления содержат логику, которая требуется для отображения данных пользователю или для сбора данных от пользователя, так что они могут быть обработаны каким-то действием контроллера. Представления должны:

• содержать логику и разметку, необходимые для показа данных пользователю.

Представления не должны:

• содержать сложную логику (ее лучше поместить в контроллер):

• содержать логику, которая создает, сохраняет или манипулирует моделью предметной области. Представления могут содержать логику, но она должна быть простой и использоваться умеренно. Помещение в представление чего угодно кроме вызовов простейших методов или несложных выражений затрудняет тестирование и сопровождение приложения в целом.

Основные принципы проектирования приложений MVC были уже описаны, особенно те из них. которые применимы к реализации ASP.NET Core MVC. Другие реализации интерпретируют аспекты паттерна MVC иначе, дополняя, подстраивая или как-то еще адаптируя его для соответствия области охвата и целям своих проектов. В последующих разделах мы кратко рассмотрим две наиболее распространенных вариации на тему MVC. Понимание этих разновидностей не имеет особого значения в случае работы с ASP.NET Core MVC. Данная информация включена ради полноты картины, потому что такие термины будут употребляться в большинстве обсуждений паттернов проектирования ПО.

Паттерн “модель-представление-презентатор* (model-view-presenter — MVP) является разновидностью MVC и разработан для того, чтобы облегчить согласование с поддерживающими состояние платформами графического пользовательского интерфейса. такими как Windows Forms или ASP.NET Web Forms. Это достойная попытка извлечь лучшее из паттерна интеллектуального пользовательского интерфейса, избежав проблем, которые он обычно привносит. В этом паттерне презентатор имеет те же обязанности, что и контроллер MVC, но он также более непосредственно связан с представлением, сохраняющим информацию о состоянии, напрямую управляя значениями, которые отображаются в компонентах пользовательского интерфейса в соответствии с вводом и действиями пользователя. Существуют две реализации этого паттерна:

• Реализация пассивного представления, в которой представление не содержит никакой логики. Такое представление служит контейнером для элементов управления пользовательского интерфейса, которыми напрямую управляет презентатор.

• Реализация координирующего контроллера, в которой представление может отвечать за определенные элементы логики презентации, такие как привязка данных, и получать ссылку на источник данных от моделей предметной области. Отличие между этими двумя подходами касается уровня интеллектуальности представления. В любом случае презентатор отделен от инфраструктуры графического пользовательского интерфейса, что делает логику презентатора более простой и подходящей для модульного тестирования.

Паттерн "модель-представление-модель представления” (model-view-view model — MWM) — это последняя разновидность MVC. Он появился в Microsoft и используется в инфраструктуре Windows Presentation Foundation (WPF). В паттерне MWM модели и представления играют те

же самые роли, что и в MVC. Разница связана с присутствующей в MWM концепцией модели представления, которая является абстрактным представлением пользовательского интерфейса. Как правило, модель представления — это класс C#, который открывает доступ к свойствам для данных, подлежащих отображению в пользовательском интерфейсе, и операциям с данными, инициируемым из пользовательского интерфейса. В отличие от контроллера MVC модель представления MVVM не имеет ни малейшего понятия о существовании представления (или любой конкретной технологии пользовательских интерфейсов). Представление MVVM применяет средство привязки WPF. чтобы установить двунаправленное соединение между свойствами, доступ к которым открывают элементы управления в представлении (вроде пунктов раскрывающегося меню или эффекта от щелчка на кнопке), и свойствами, доступ к которым открывает модель представления.

5 ШАБЛОН ПРОЕКТИРОВАНИЯ ГРАФИЧЕСКИХ ЗАДАЧ

Шаблон проектирования или паттерн в разработке программного обеспечения — повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код; это лишь пример решения задачи, который можно использовать в различных ситуациях. Объектно-ориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

«Низкоуровневые» шаблоны, учитывающие специфику конкретного языка программирования, называются идиомами. Это хорошие решения проектирования, характерные для конкретного языка или программной платформы, и потому не универсальные.

На наивысшем уровне существуют архитектурные шаблоны, они охватывают архитектуру всей программной системы.

В сравнении с полностью самостоятельным проектированием, шаблоны обладают рядом преимуществ. Основная польза от использования шаблонов состоит в снижении сложности разработки за счёт готовых абстракций для решения целого класса проблем. Шаблон даёт решению своё имя, что облегчает коммуникацию между разработчиками, позволяя ссылаться на известные шаблоны. Таким образом, за счёт шаблонов производится унификация деталей решений: модулей, элементов проекта, — снижается количество ошибок. Применение шаблонов концептуально сродни использованию готовых библиотек кода. Правильно сформулированный шаблон проектирования позволяет, отыскав удачное решение, пользоваться им снова и снова. Набор шаблонов помогает разработчику выбрать возможный, наиболее подходящий вариант проектирования.

Хотя легкое изменение кода под известный шаблон может упростить понимание кода, есть мнение, что с применением шаблонов могут быть связаны две сложности. Во-первых, слепое следование некоторому выбранному шаблону может привести к усложнению программы. Во-вторых, у разработчика может возникнуть желание попробовать некоторый шаблон в деле без особых оснований.

Поэтому шаблоны проектирования стоит использовать осмысленно и только там, где они требуются.

Шаблоны бывают следующих видов:

- Порождающие;
- Структурные;
- Поведенческие;

Порождающие шаблоны — шаблоны проектирования, которые абстрагируют процесс инстанцирования. Они позволяют сделать систему независимой от способа создания, композиции и представления объектов. Шаблон, порождающий классы, использует наследование, чтобы изменять наследуемый класс, а шаблон, порождающий объекты, делегирует инстанцирование другому объекту.

Существуют следующие порождающие шаблоны:

- Простая фабрика (Simple Factory);
- Фабричный метод (Factory Method);
- Абстрактная фабрика (Abstract Factory);
- Строитель (Builder);
- Прототип (Prototype);
- Одиночка (Singleton).

Структурные шаблоны — шаблоны проектирования, в которых рассматривается вопрос о том, как из классов и объектов образуются более крупные структуры.

Список структурных шаблонов проектирования:

- Адаптер (Adapter);
- Мост (Bridge);
- Компоновщик (Composite);
- Декоратор (Decorator);
- Фасад (Facade);
- Приспособленец (Flyweight);
- Заместитель (Proxy).

Поведенческие шаблоны — шаблоны проектирования, определяющие алгоритмы и способы реализации взаимодействия различных объектов и классов.

Поведенческие шаблоны:

- Цепочка обязанностей (Chain of Responsibility);
- Команда (Command);
- Итератор (Iterator);
- Посредник (Mediator);
- Хранитель (Memento);
- Наблюдатель (Observer);
- Посетитель (Visitor);
- Стратегия (Strategy);
- Состояние (State);

Шаблонный метод (Template Method).

6 ОПИСАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

При запуске приложения пользователь попадает на страницу с каталогом книг (рис.6.1).

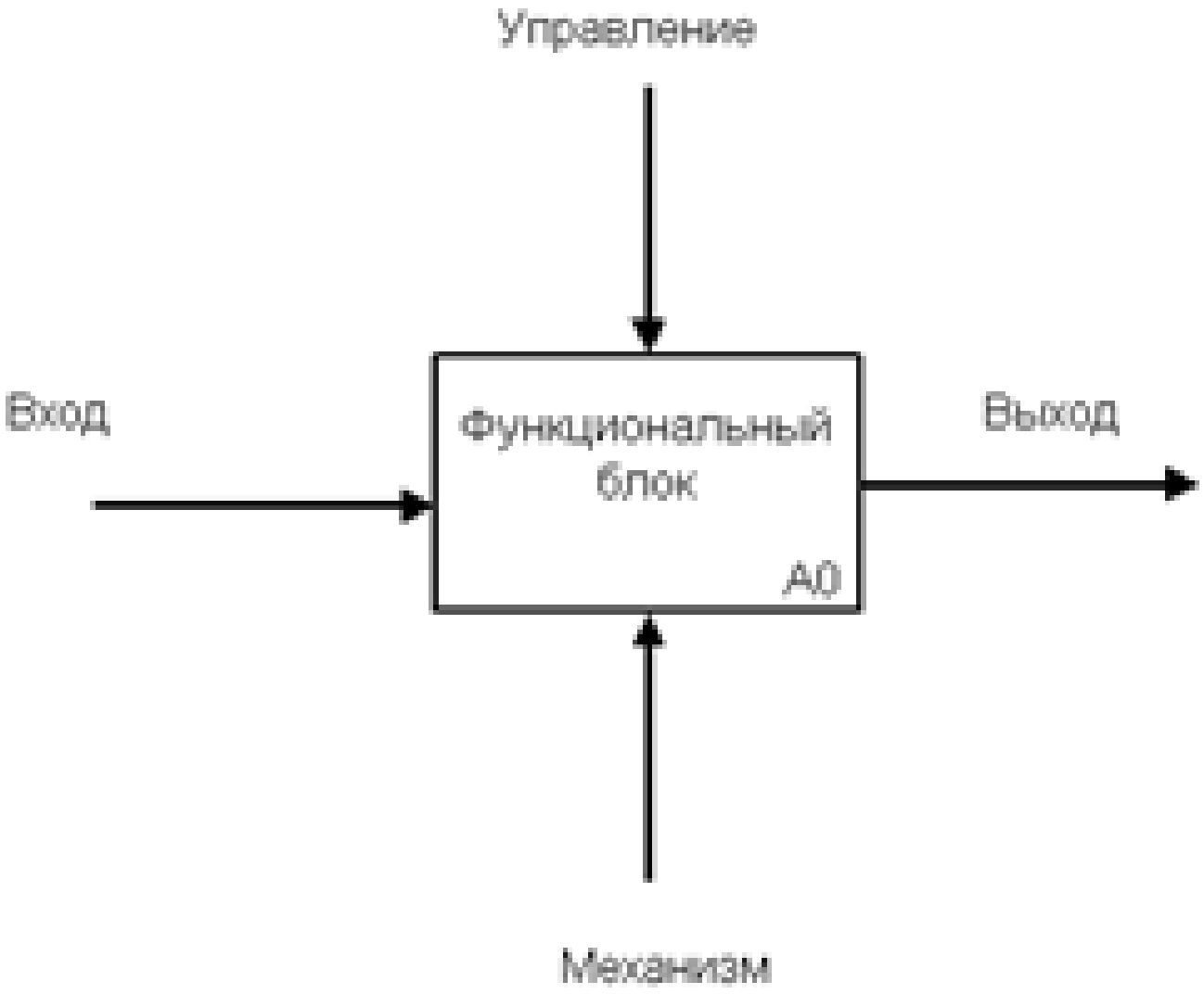

Рисунок 6.1 – Каталог книг

На этой странице есть возможность выбрать товары, которые по нраву посетителю интернет-магазина, и добавить их в корзину (рис.6.2).

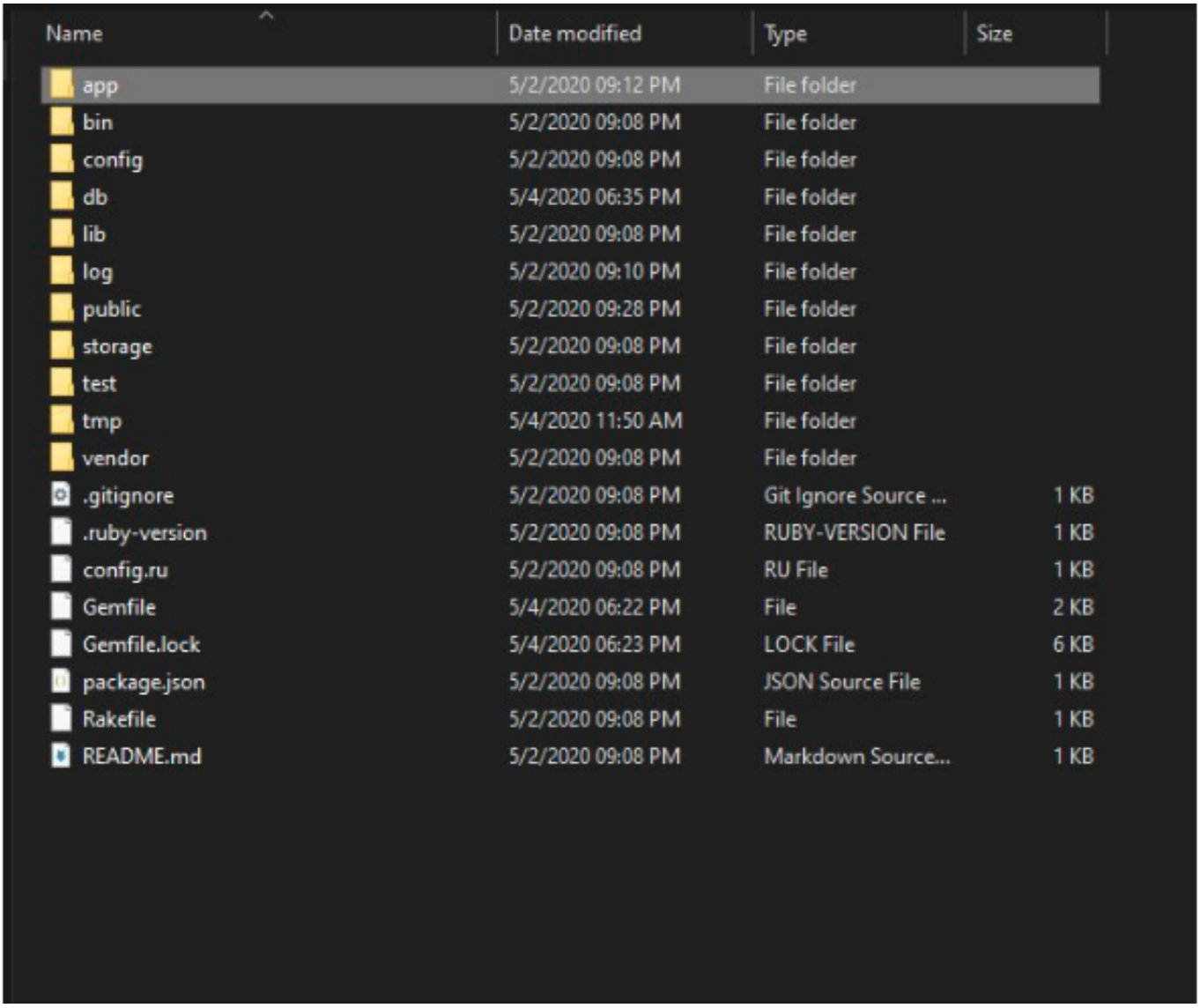

Рисунок 6.2 – Корзина

После попадания в корзину товара, можно нажать на кнопку оплатить и ввести соответствующие данные для завершения заказа (рис.6.3).

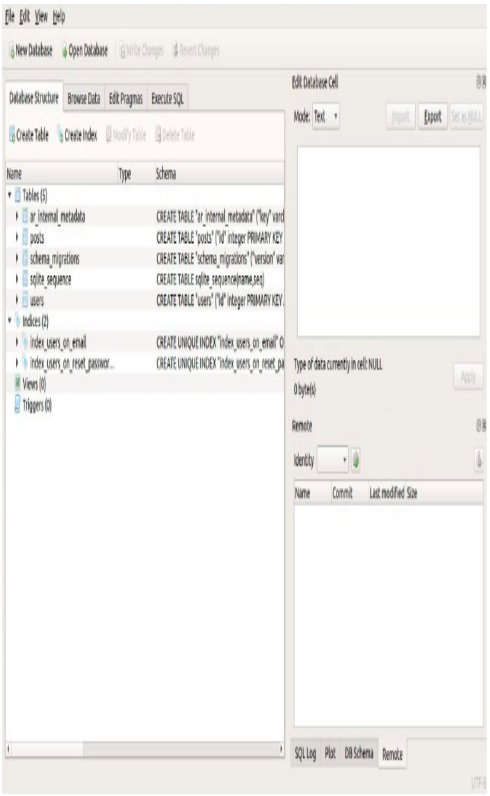

Рисунок 6.3 – Оформление заказа

И если данные введены корректно, они заносятся в базу данных, а пользователь видит новую страницу, которая сообщает ему об успешном завершении заказа (рис.6.4).

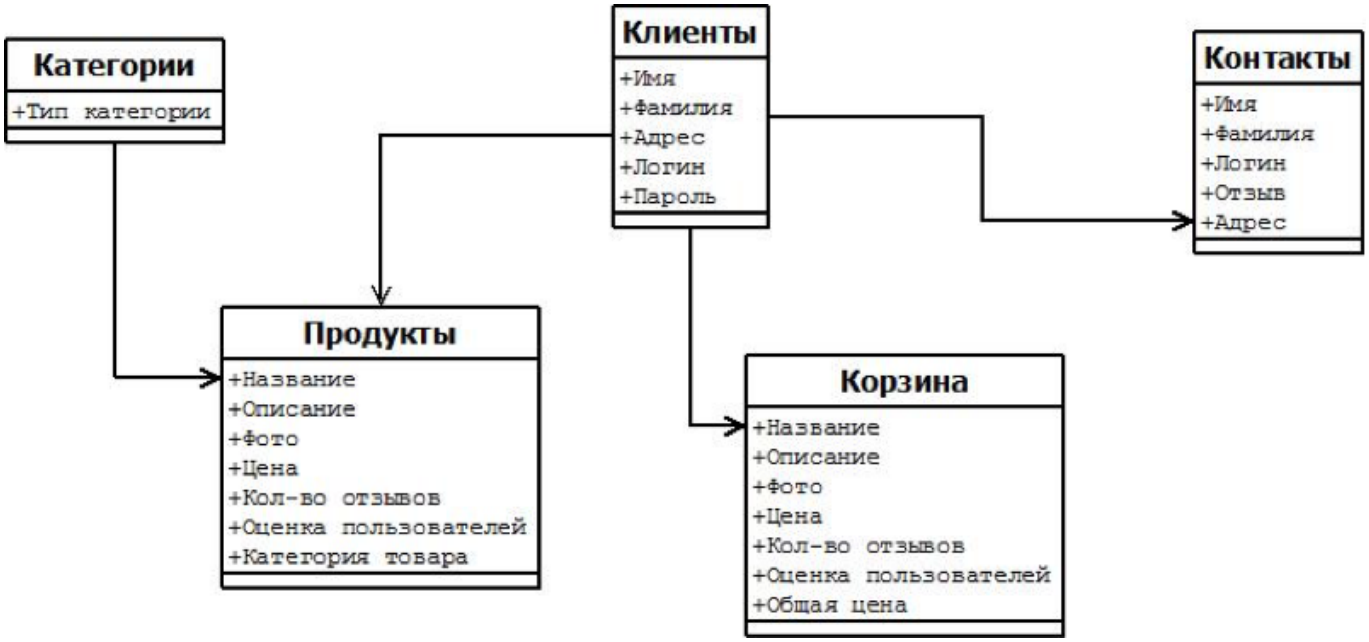

Рисунок 6.4 – Завершение заказа

Также есть возможность попрыгать по разным категориям, например «Трагедии» (рис.6.5) или «Ужасы» (рис.6.6)

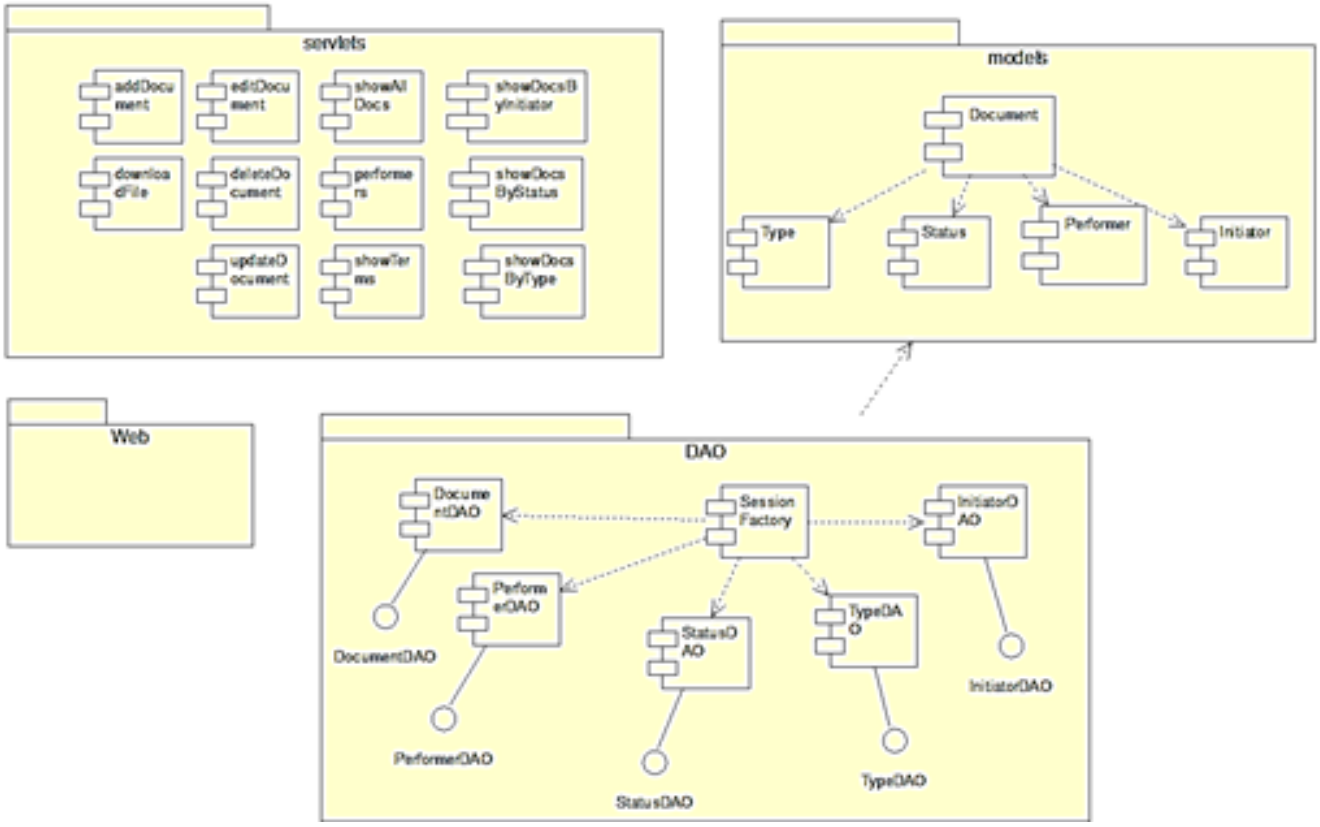

Рисунок 6.5 – Категория «Трагедии»

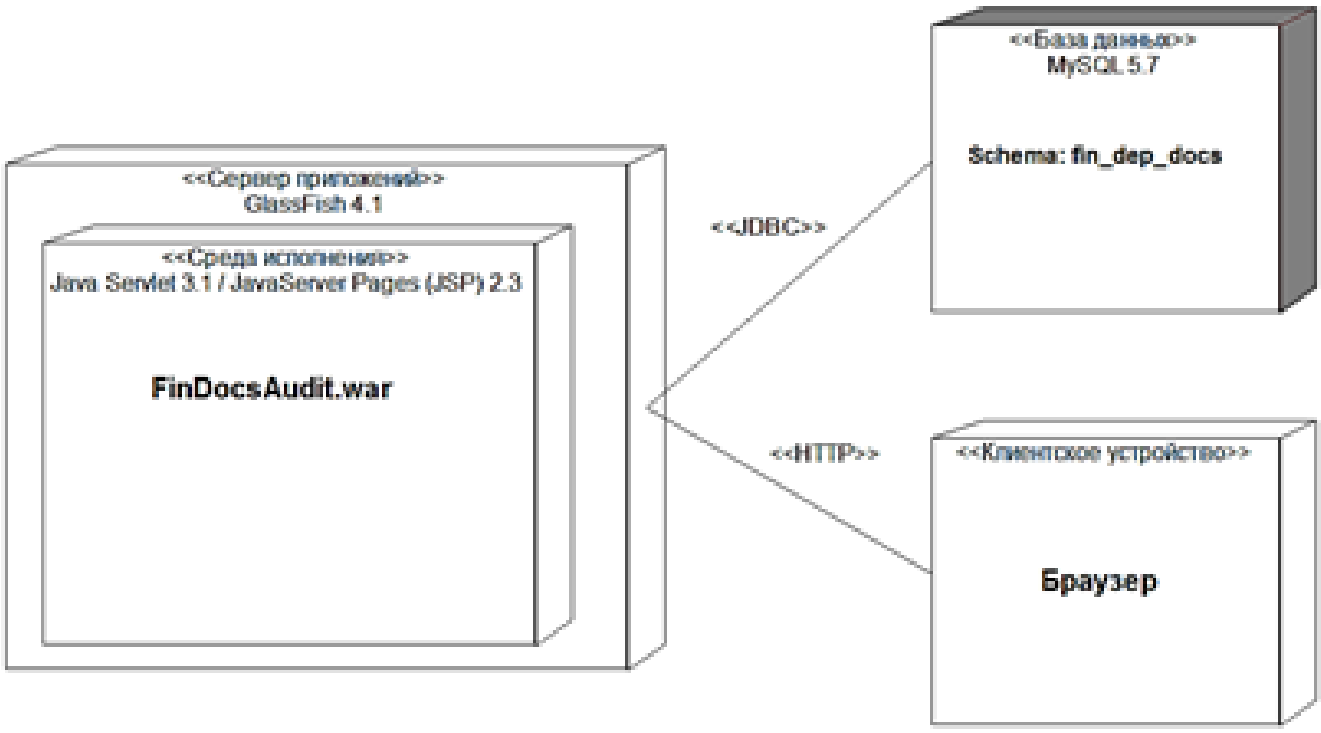

Рисунок 6.6 – Категория «Ужасы»

Заключение

В данной курсовой работе на тему «Книжный интернет магазин» с помощью специальных интернет-ресурсов были выполнены следующие задачи: создание интуитивно понятного пользовательского интерфейса, рациональное оформление back-end, а также правильное

проектирование и создание базы данных. При написании курсового проекта были использованы ресурсы для изучения концепций и технологий объектно-ориентированного программирования. Работа с ними происходила в среде разработки Visual Studio. В ходе выполнения курсового проекта была составлена схема структуры графического пользовательского интерфейса, который использовался в данной работы. Убедившись в правильности и верном выполнении работы программы после пройденных тестов, можно сказать, что результат курсовой работы полностью соответствует всем требованиям, поставленным вначале разработки программы.

Список использованных источников

1. [url] **Фримен Адам. ASP.NET Core MVC с примерами на C# для профессионалов**
2. [url] **Паттерны проектирования, используемые в ASP.NET Core**
https://metanit.com/sharp/patterns/

Приложения

1. [электронный документ] 5ec1b28039204_ООП.docx
2. [электронный документ] 5ecd388750d51_Бланк задания.docx