

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Компьютерных систем и сетей
Кафедра Программного обеспечения информационных технологий
Дисциплина Операционные системы и системное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему:

ОНЛАЙН-СЕРВИС "МАГАЗИН ИГР"

БГУИР КП 1-40 01 01 208 ПЗ

Студент
Руководитель

гр. 851002 М. Г. Кудрявцев
В. Н. Видничук

Минск 2021

СОДЕРЖАНИЕ

Введение	5
1 Постановка задачи	6
2 Анализ литературных источников	7
2.1 ASP.NET Core	7
2.2 Angular	7
2.3 Принципы создания ASP.NET Core WebApi + Angular приложения	8
3 Обзор аналогов и формирование функциональных требований	10
3.1 Steam	10
3.2 Epic Games Store	11
3.3 Origin	12
3.4 Формирование функциональных требований	13
4 Проектирование и разработка программного средства	14
5 Тестирование	17
6 Руководство пользователя	22
6.1 Управление аккаунтом	22
6.2 Основной функционал	23
6.3 Функционал администратора	24
Заключение	27
Список использованных источников	28
Приложение А	29

ВВЕДЕНИЕ

В современном мире всё большую популярность набирают онлайн-сервисы, предназначенные для продажи различных товаров или услуг. Причинами такого быстрого роста популярности являются, в первую очередь, отсутствие необходимости в посещении магазина, а также экономия финансов на аренду и содержание помещения. Действительно, такая система распространения товаров является выгодной и для клиентов, и для создателей платформ, а повсеместное распространение и доступность мобильного интернета ещё больше увеличивают актуальность онлайн-сервисов.

В качестве темы курсового проекта был выбран "Онлайн-сервис Магазин игр", задачей которого является продажа цифровых копий видеоигр и организация площадки, где пользователи могут найти информацию и отзывы других пользователей о интересующих их продуктах. До распространения онлайн-сервисов, когда человек следил за некоторой интересующей его видеоигрой, чтобы найти какую-нибудь актуальную информацию о процессе её разработки, ему было необходимо приобрести свежий выпуск игрового журнала и надеяться, что там он найдёт что-то интересное. Для приобретения видеоигры также надо было идти в специализированные магазины и покупать копию на физическом носителе.

Очевидно, что такая система обладала множеством недостатков и приводила к большим тратам времени. Также стоит отметить, что физические носители в последнее время очень сильно утратили свою актуальность, потому что высокоскоростной интернет стал доступным широкому кругу пользователей. Конечно, нельзя не отметить, что несмотря на всё неудобство и неэффективность такого способа приобретения видеоигр, ещё остаются люди, для которых он является приоритетным. Это связано с тем, что для них обладание физической копией служит гарантом владения игрой, и такую логику можно понять. К тому же, некоторые люди занимаются коллекционированием видеоигр, а цифровые копии не способны удовлетворить такую потребность.

Таким образом, можно отметить, что распространение видеоигр на физических носителях ещё не утратило свою популярность до конца, однако всё больше и больше людей предпочитают не тратить время на походы в магазины и пользуются сервисами цифровой дистрибуции. Количество зарегистрированных пользователей в самых популярных сервисах исчисляется сотнями миллионов, поэтому тема курсового проекта обладает высокой актуальностью и востребованностью.

1 ПОСТАНОВКА ЗАДАЧИ

В рамках курсовой работы необходимо изучить принципы построения веб-приложений с использованием популярных фреймворков для создания серверной и клиентской части приложения, таких как ASP.NET Core WebApi, Angular и Entity Framework Core.

Осуществить анализ предметной области разработки веб-сервисов цифровой дистрибуции, в частности – продажи цифровых копий видеоигр. Выявить основные задачи и функционал сервиса.

На основании анализа предметной области разработать веб-приложение для цифровой дистрибуции видеоигр.

Осуществить развертывание веб-приложения и выполнить его тестирование.

2 АНАЛИЗ ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

2.1 ASP.NET Core

ASP.NET Core – свободно-распространяемый кросс-платформенный фреймворк для создания веб-приложений с открытым исходным кодом [1]. Данная платформа разрабатывается компанией Майкрософт совместно с сообществом и имеет большую производительность по сравнению с ASP.NET. Имеет модульную структуру и совместима с такими операционными системами как Windows, Linux и macOS.

Несмотря на то, что это новый фреймворк, построенный на новом веб-стеке, он обладает высокой степенью совместимости концепций с ASP-.NET. Приложения ASP.NET Core поддерживают параллельное управление версиями, при котором разные приложения, работающие на одном компьютере, могут ориентироваться на разные версии ASP.NET Core. Это было невозможно в предыдущих версиях ASP.NET.

В данном курсовом проекте, в соответствии с поставленной задачей, для разработки серверной части веб-приложения будет использоваться ASP.NET Core WebAPI, что позволит создать удобный интерфейс для взаимодействия с клиентской частью приложения используя платформу .NET.

2.2 Angular

Angular (версия 2 и выше) – открытая и свободная платформа для разработки веб-приложений, написанная на языке TypeScript, разрабатываемая командой из компании Google, а также сообществом разработчиков из различных компаний [2]. Angular – полностью переписанный фреймворк от той же команды, которая написала AngularJS.

Предназначен для разработки одностраничных приложений. Его цель – расширение браузерных приложений на основе MVC-шаблона, а также упрощение тестирования и разработки.

Angular спроектирован с убеждением, что декларативное программирование лучше всего подходит для построения пользовательских интерфейсов и описания программных компонентов, в то время как императивное программирование отлично подходит для описания бизнес-логики. Фреймворк адаптирует и расширяет традиционный HTML, чтобы обеспечить двухстороннюю привязку данных для динамического контента, что позволяет автоматически синхронизировать модель и представление. В результате Angular уменьшает роль DOM-манипуляций и улучшает тестируемость.

Angular придерживается MVC-шаблона проектирования и поощряет слабую связь между представлением, данными и логикой компонентов. Используя внедрение зависимости, Angular переносит на клиентскую сторону такие классические серверные службы, как видозависимые контроллеры. Следовательно, уменьшается нагрузка на сервер и веб-приложение становится легче.

В данном курсовом проекте будет использоваться Angular, так как он легко интегрируется с ASP.NET Core и удобен в использовании. Также, Angular является фреймворком для языка TypeScript, обладающего строгой типизацией, что может упростить процесс тестирования приложения.

2.3 Принципы создания ASP.NET Core WebApi + Angular приложения

Серверная часть веб-приложения, написанная с использованием ASP.NET Core WebApi имеет следующую структуру [3]:

- Класс Startup, в котором настраиваются службы, необходимые приложению и определяется конвейер обработки запросов.
- В методе ConfigureServices регистрируются службы для контейнера внедрения зависимостей.
- В методе Configure в конвейер обработки запросов встраиваются компоненты промежуточного слоя, в том числе компонент, отвечающий за маршрутизацию.
- Определяются маршруты, ведущие к конечным точкам приложения.
- Логика обработки запросов описывается в конечных точках – контроллерах WebApi.

Деление веб-приложения на два отдельных проекта – WebApi для серверной части и Angular для клиентской, может показаться неудобным, ведь это увеличивает объём кода и требует использования разных языков программирования в каждом проекте. Можно воспользоваться и старым подходом, когда всё приложение состоит лишь из сервера, который генерирует и отправляет HTML-страницы в ответ на запросы клиента, однако с развитием веб-приложений стало ясно, что такой подход обладает определённым количеством недостатков, среди которых, например, неизбежная перезагрузка страницы при любом действии пользователя. Всё более предпочтительным вариантом становится создание Single-Page приложений, в которых всё взаимодействие с пользователем выносится в отдельное кли-

ентское приложение, которое посыпает запросы на сервер и отображает изменения без перезагрузки веб-страницы, что положительно сказывается на пользовательском опыте.

Основными принципами разработки клиентской части веб-приложения с использованием Angular являются [4]:

- Angular-приложения состоят из независимых элементов. Эти элементы называются компонентами, и у каждого компонента своё поведение.
- Все компоненты подключаются к главному или дополнительным модулям. Модули управляют компонентами.
- Главный модуль контролирует всё приложение, а дополнительные модули следят за работой отдельных элементов.
- Для сложных операций вместо компонентов используют сервисы.
- Сервисы отвечают только за тот набор логических операций, для которых он предназначен.

3 ОБЗОР АНАЛОГОВ И ФОРМИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

3.1 Steam

Крупнейший сервис цифровой дистрибуции видеоигр, прошедший путь от небольшого приложения для сопровождения многопользовательских шутеров от Valve (Team Fortress Classic, Counter-Strike и Half-Life 2) в 2003 году до более чем 30 тысяч игр, одного миллиарда аккаунтов и пиком в более 20 миллионов одновременно находящихся в сети пользователей в 2020-м. Интерфейс веб-сайта представлен на рисунке 3.1.

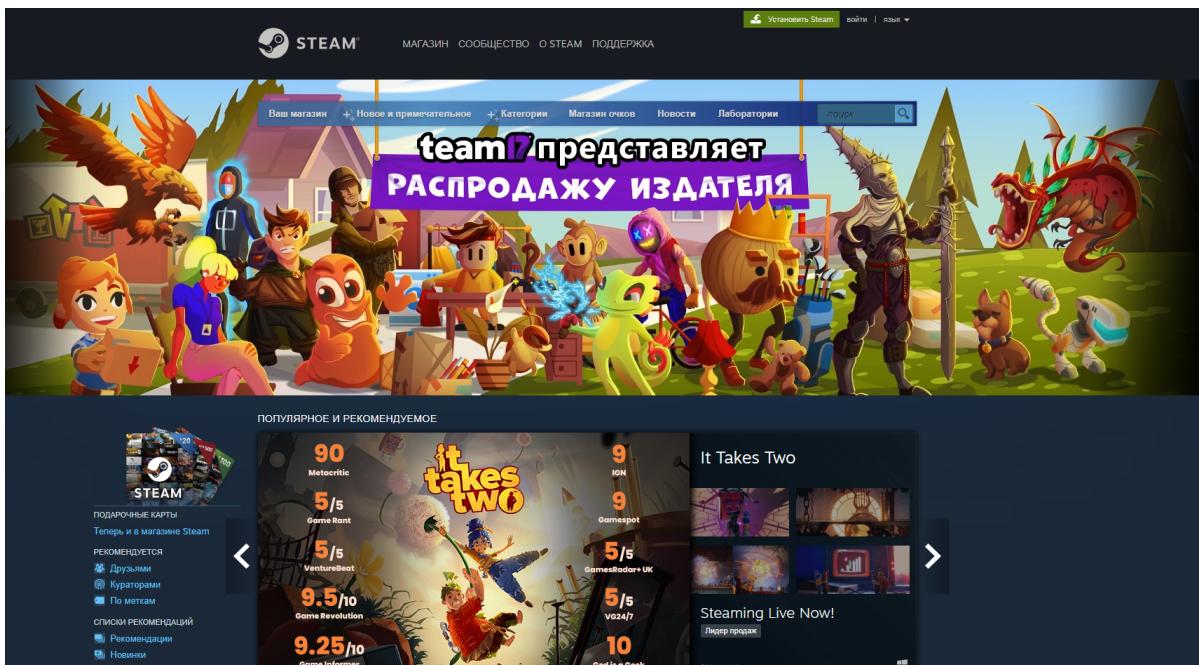


Рисунок 3.1 – Интерфейс веб-сайта store.steampowered.com

Такая аудитория досталась магазину благодаря упорному труду на протяжении многих лет его развития. Valve вкладывает огромные ресурсы в разработку Steam, за счет чего его можно назвать не просто сервисом, а целой социальной сетью для игроков: с магазином и библиотекой игр, списком друзей и чатом, пользовательскими отзывами, обсуждениями и трансляциями, руководствами и скриншотами, пользовательскими дополнениями и торговой площадкой, достижениями, карточками и кастомизацией профилей.

Сервис предлагает, пожалуй, самые комфортные условия для клиентов. В каталоге игр присутствует удобный поиск по жанрам, тегам, популярности и другим критериям, разработчики могут устанавливать региональ-

ные цены, а сами цены представлены в самых разных валютах, начиная от американского доллара и российского рубля, заканчивая мексиканским песо и вьетнамским донгом. Поддерживаются и различные способы оплаты, включая банковские карты и внутренний кошелек Steam, который можно пополнить посредством многих платежных систем.

Впрочем, есть у сервиса и недостатки. Так, крайне свободная политика Valve относительно публикации игр в Steam и почти полное отсутствие цензуры привели к появлению на платформе большого количества низкокачественных продуктов. Кроме того, некоторым пользователям не нравится неповоротливость Valve в плане развития магазина: компания слишком редко добавляет интерфейсные изменения в клиент и почти не реагирует на действия конкурентов.

3.2 Epic Games Store

Относительно молодой (запущен в декабре 2018 года), но уже обративший на себя внимание сервис от компании Epic Games. Все из-за неоднозначной политики создателей: с самого момента появления сервиса, он постоянно фигурирует в различных скандалах. Это и покупка эксклюзивности игр с изъятием их из Steam, и информация о сканировании приложением Epic Games Launcher (являющимся, по совместительству и магазином, и библиотекой игр) папок Steam, и распродажа, в ходе которой издатели снимали с продажи свои игры. Интерфейс веб-сайта представлен на рисунке 3.2.

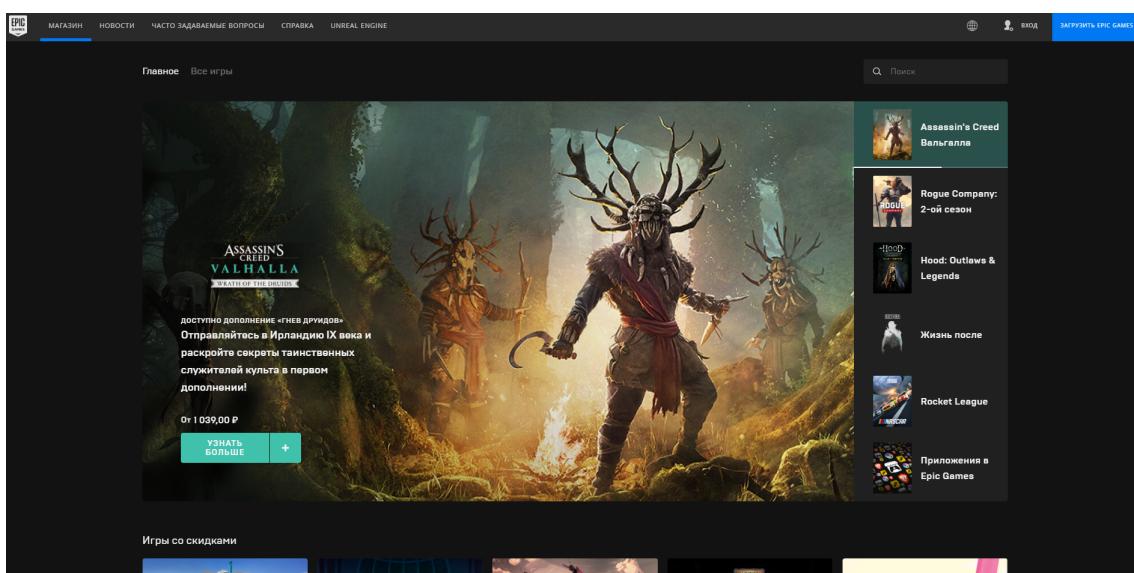


Рисунок 3.2 – Интерфейс веб-сайта [epicgames.com](https://www.epicgames.com)

Так как сервис ещё очень молод, он не обладает каким-то внушительным функционалом, однако есть несколько преимуществ, выгодно выделяющих его на уровне конкурентов: во-первых, из-за низкой комиссии, многие разработчики выставляют свои видеоигры эксклюзивно в Epic Games; во-вторых, на сервисе бывают гораздо более выгодные по сравнению с конкурентами предложения.

Однако сервис имеет и несколько недостатков, самыми существенными из которых являются его медленная работа, отсутствие корзины и невозможность оставлять и читать отзывы других покупателей.

3.3 Origin

Сервис Origin был запущен компанией Electronic Arts в 2011 году: издатель хотел продавать издаваемые им игры единолично, не делясь прибылью с владельцами других сервисов (до старта Origin игры от EA издавались в Steam). Интерфейс веб-сайта представлен на рисунке 3.3.

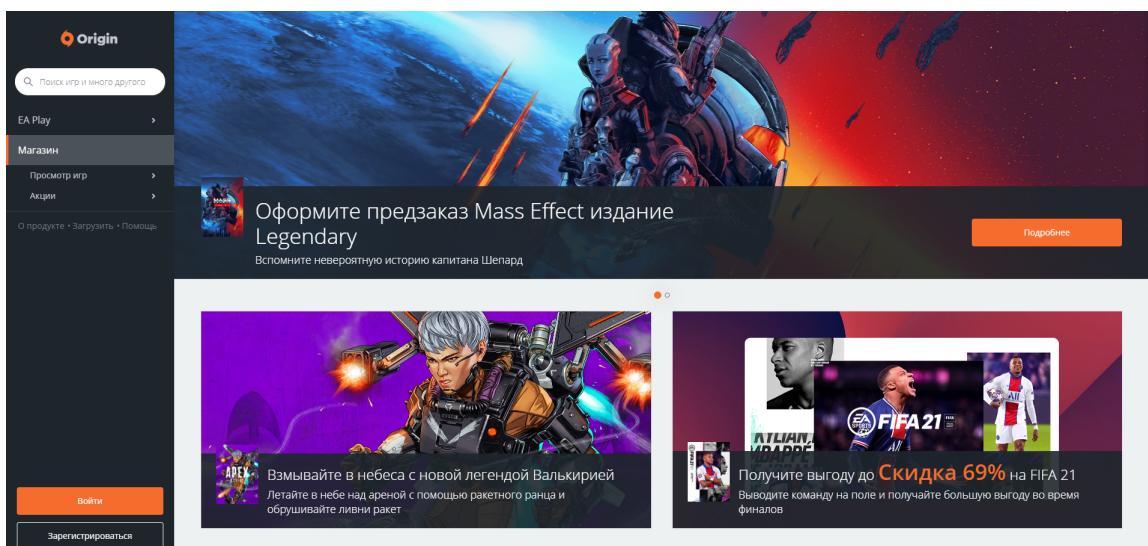


Рисунок 3.3 – Интерфейс веб-сайта origin.com

Сервис предлагает большую часть функционала, присутствующего у конкурентов: оплатить покупки можно с помощью распространенных платежных систем, а для пользователей из разных частей мира действуют региональные цены. Кроме того, EA регулярно проводит распродажи и бесплатные раздачи видеоигр. Присутствуют и социальные функции: список друзей и чат, автообновления игр, но по их количеству сервис все же не дотягивает до Steam.

Основными недостатками сервиса являются отсутствие возможности

оставлять и просматривать отзывы покупателей, небольшой каталог видеоигр и медленная работа.

3.4 Формирование функциональных требований

Проанализировав существующие на рынке сервисы цифровой дистрибуции видеоигр, можно сделать вывод, что разрабатываемый сервис должен обладать следующими функциями:

- Наличие корзины, позволяющей приобрести несколько видеоигр за раз.
- Фильтрация каталога видеоигр по названию, жанру, цене.
- Возможность смены локализации сервиса: русский либо английский язык на выбор.
- Возможность восстановить доступ к аккаунту в случае утери пароля или электронной почты.
- Возможность оставлять и просматривать отзывы пользователей о видеоиграх.

4 ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

В данном курсовом проекте программное средство будет разделено на клиентскую и серверную часть. Рассмотрим архитектуру каждой из частей подробнее.

Архитектура серверной части будет представлять из себя WebApi, реализованное с помощью ASP.NET Core. Наборы операций, такие как аутентификация, работа с магазином, администрирование, будут представлять из себя отдельные контроллеры, где каждый контроллер может обрабатывать строго определенный набор запросов. Такой подход очень схож с модульным программированием и упрощает разработку и отладку приложения. Например, контроллер аутентификации должен обрабатывать запросы на авторизацию, регистрацию, выход из аккаунта и восстановление аккаунта. Контроллер администрирование должен уметь обрабатывать запросы на добавление/изменение/удаление информации пред назначенной для пользователей. Для того чтобы отличать контроллеры, будет использоваться уникальный и подходящий по смыслу префикс для api. Например для авторизации это '/auth', для магазина это '/store' и т.д.

Для разграничения доступа к контроллерам будет использоваться механизм Identity, который является встроенным в ASP.NET Core. Его суть заключается в том, что при авторизации пользователю назначаются роли, которые обычно берутся из базы данных. Затем, при каждом запросе к серверу, проверяются роли пользователя, и если требуемых ролей нет, то запрос не обрабатывается. В ASP.NET Core для разграничения доступа контроллерам добавляются атрибуты:

- AllowAnonymous – доступ к контроллеру/методу имеют все, проверок ролей не происходит. Типичным применением такого атрибута являются методы авторизации/регистрации/просмотра публичной информации.
- Authorized – доступ к контроллеру/методу имеют только авторизованные пользователи с любыми ролями. Применяется при просмотре/редактировании информации, специфической для конкретного пользователя.
- Authorized, Roles("Admin") - доступ к контроллеру/методу имеет только авторизованный пользователь с ролью "Admin". Применяется для скрытия администрирующих методов от рядовых пользователей.

Так как зачастую запросам будут необходимы параметры, то сервер должен проводить их валидацию перед выполнением запроса. Для этого в ASP.NET Core предусмотрены модели запросов. При запросе данные от

клиента автоматически сопоставляются с моделью запроса по принципу совпадения имен. В свою очередь, обработчики запросов перед выполнением проверяют валидность модели, и в случае ошибки возвращают нужный код. Модели запросов представляют из себя обычные классы, их отличие заключается в том, что для каждого свойства класса используются атрибуты валидации, например:

- Required – свойство является обязательным.
- EmailAddress – свойство должно соответствовать формату электронной почты.
- MaxLength(320) – свойство не должно превышать заданную длину (320).
- Range(1, 10) – свойство должно находиться в диапазоне от 1 до 10.

Для просмотра контента в требуемых объемах следует реализовать страничное представление. Это лучше делать на стороне сервера, так как в таком случае по сети будет передаваться меньше трафика.

Клиентская часть будет представлять из себя совокупность компонентов, модулей и сервисов, реализованных на Angular.

Компонент - базовый блок для создания приложений в angular. Один компонент является своеобразным строительным блоком, который инкапсулирует в себе всю необходимую функциональность. Пример компонента отвечающего за добавление новой игры приведён в приложении A. Каждый компонент состоит как минимум из файла разметки (.html), в котором можно использовать привязки (bindings) из angular, и файла кода (.ts) в котором содержится логика обработки действий пользователя. При желании компонент может содержать специфические стили (.css). Все страницы приложения в angular – это компоненты, при этом компонент может содержать внутри себя другие компоненты. Таким образом все приложение будет состоять из иерархически упорядоченных компонентов. На вершине иерархии будет стоять AppComponent, который и является приложением.

При проектировании приложения подразумевается, что компоненты являются лишь удобным посредником для общения пользователя и серверного приложения. Они не содержат бизнес-логики и их назначение сводится к отображению информации и использованию специальных сервисов. Сервисы как раз являются связующим звеном между клиентским и серверным приложением. Их задача состоит в отправке требуемых запросов на сервер и получении соответствующих результатов в удобной для компонентов форме. Сервисы содержат всю клиентскую бизнес-логику, необходимую для работы приложения. Учитывая тот факт, что Angular использует TypeScript, то

сервисы должны возвращать типизированные объекты. Такие объекты называются моделями и обычно они являются прямым соответствием ответа сервера клиенту. Используя такие модели компоненты могут привязаться к полям для последующего отображения пользователю.

Как и в серверном приложении, клиенту так же необходима валидация входных данных. В Angular для этого предусмотрены классы Validators. Принцип использования валидации прост:

- Создать произвольный компонент, который представляет из себя веб-страницу.
- В файле кода определить форму и ее поля с соответствующими валидаторами.
- В html разметке, используя Angular-специфические теги, изменять свойства html-элементов в соответствии с результатами валидации.

Типичный пример использования – запретить отправку формы путем установки тега [disabled], если форма не прошла валидацию.

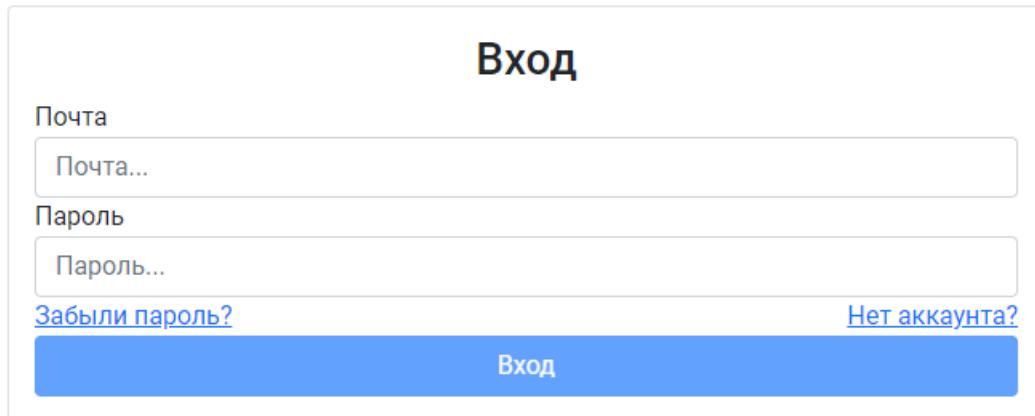
Несмотря на то, что приложение является одностраничным (Single paged), по факту оно состоит из множества компонентов, которые сменяют друг друга, а отсутствие перезагрузки создает иллюзию одностраничности. Для перехода между компонентами в Angular предусмотрены классы Router и ActivatedRoute. Класс Router служит для простого перехода между компонентами, а класс ActivatedRoute необходим для передачи параметров из компонента в компонент через адресную строку. Учитывая тот факт, что помимо простых пользователей существуют и продвинутые, то возникает необходимость защиты от подбора ссылок к функционалу администратора. В то время как доступ к методам администрирования ограничен на уровне сервера, все равно следует ограничить доступ на уровне клиента. Для этого используется класс guard. Этот класс предназначен для фильтрации запросов клиента к определенным маршрутам основываясь на некоторых внутренних состояниях приложения. Типичный пример использования – запретить переход к панели администрирования если пользователь не администратор.

Следует отметить что оптимальным форматом данных для общения клиента и сервера был выбран формат JSON, так как он прост для понимания, Angular и ASP.NET Core умеют работать с ним 'из коробки' и он автоматически сопоставляется с моделями на основе совпадения имен в обоих фреймворках.

5 ТЕСТИРОВАНИЕ

Таблица 5.1 – Тест-кейс регистрации аккаунта

Тест	Ожидаемый результат	Результат
Регистрация аккаунта 1. Перейти во вкладку "Вход" на навигационной панели сайта 2. Нажать на кнопку "Нет аккаунта?" 3. Заполнить поля формы и нажать на кнопку регистрация	1. Появление формы для входа 2. Появление формы для регистрации 3. Перенаправление на форму входа	Успех (Рисунок 5.1)



Форма входа в аккаунт после регистрации. Видимые элементы: заголовок "Вход", поле для ввода почты с подсказкой "Почта...", поле для ввода пароля с подсказкой "Пароль...", ссылка "Забыли пароль?", ссылка "Нет аккаунта?", большая синяя кнопка "Вход".

Рисунок 5.1 – Форма входа в аккаунт после регистрации

Таблица 5.2 – Тест-кейс аутентификации

Тест	Ожидаемый результат	Результат
Вход в аккаунт 1. Перейти во вкладку "Вход" на навигационной панели сайта 2. Заполнить почту и пароль используя значения из предыдущего тест-кейса и нажать на кнопку "Вход"	1. Появление формы для входа 2. Перенаправление на страницу с каталогом игр	Успех (Рисунок 5.2)

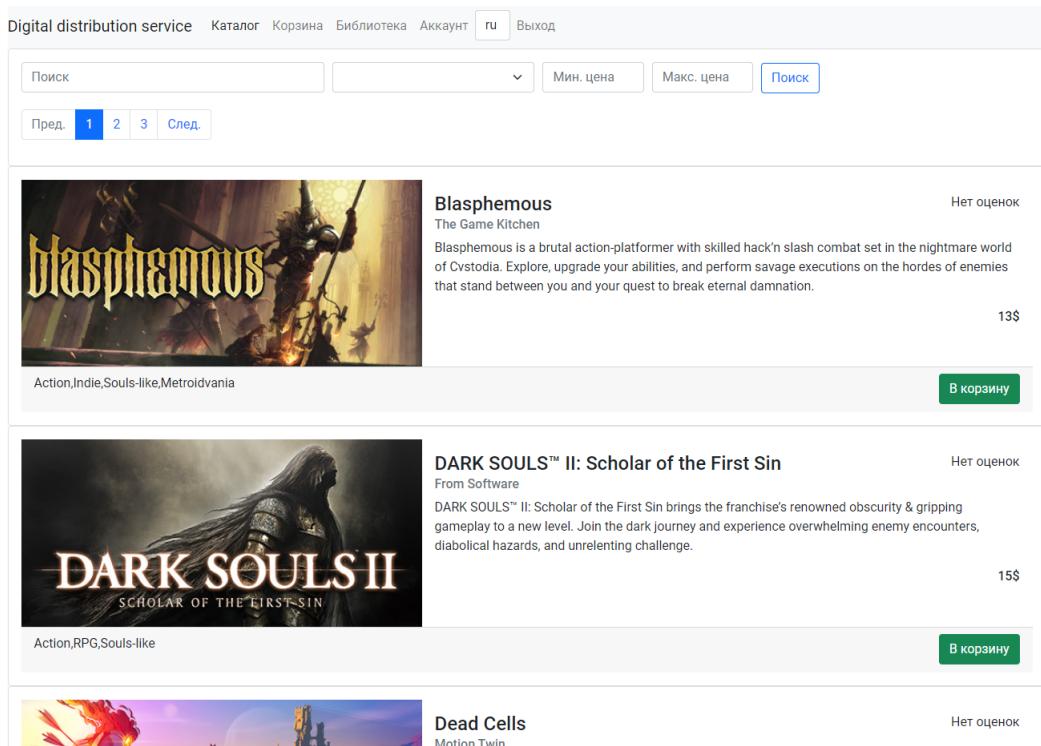


Рисунок 5.2 – Страница с каталогом после успешного входа

Таблица 5.3 – Тест-кейс восстановления доступа к аккаунту

Тест	Ожидаемый результат	Результат
Восстановление доступа к аккаунту 1. Перейти во вкладку "Вход" на навигационной панели сайта 2. Нажать на кнопку "Забыли пароль?" 3. Ввести почту и нажать на кнопку "Отправить" справа от поля 4. Ввести полученный по почте код и нажать на кнопку "Отправить" внизу формы	1. Появление формы для входа 2. Появление формы для восстановления доступа 3. Получения письма с кодом на указанный почтовый ящик 4. Перенаправление на страницу смены пароля	Успех (Рисунки 5.3, 5.4)

 Support maxd4567@gmail.com
 Вам:  max.kudryavtzeff@yandex.by
сегодня в 20:25

Your account restore code: d4a569ae-c0e7-4c24-8ab4-d7f6c16177c3

Рисунок 5.3 – Письмо с кодом, полученное на указанный почтовый ящик

Digital distribution service Каталог Корзина Библиотека Аккаунт ru Выход

Настройки аккаунта

Новый пароль Пароль Сменить пароль

Рисунок 5.4 – Форма для смены пароля

Таблица 5.4 – Тест-кейс добавления игр в корзину

Тест	Ожидаемый результат	Результат
Добавление игр в корзину 1. Зайти в аккаунт 2. Добавить несколько игр в корзину 3. Перейти на страницу "Корзина" в навигационной панели	1. Перенаправление на страницу "Каталог" 2. Кнопки "В корзину" становятся неактивными 3. В корзине отображаются добавленные игры	Успех (Рисунок 5.5)

Digital distribution service Каталог Корзина Библиотека Аккаунт ru Выход

Игра	Цена	Дата добавления	Действие
Elite Dangerous	30\$	15/05/2021	<button>Удалить</button>
Forza Horizon 4	28\$	15/05/2021	<button>Удалить</button>
Sekiro™: Shadows Die Twice	32\$	15/05/2021	<button>Удалить</button>
Всего:	90 \$		

Оплатить

Рисунок 5.5 – Корзина пользователя

Таблица 5.5 – Тест-кейс приобретения игр

Тест	Ожидаемый результат	Результат
Приобретение игр 1. Зайти в аккаунт и перейти на страницу "Корзина" в навигационной панели 2. Нажать на кнопку "Оплатить"	1. Перенаправление на страницу "Корзина" со списком добавленных игр 2. Перенаправление на страницу "Библиотека" с приобретёнными играми	Успех (Рисунок 5.6)

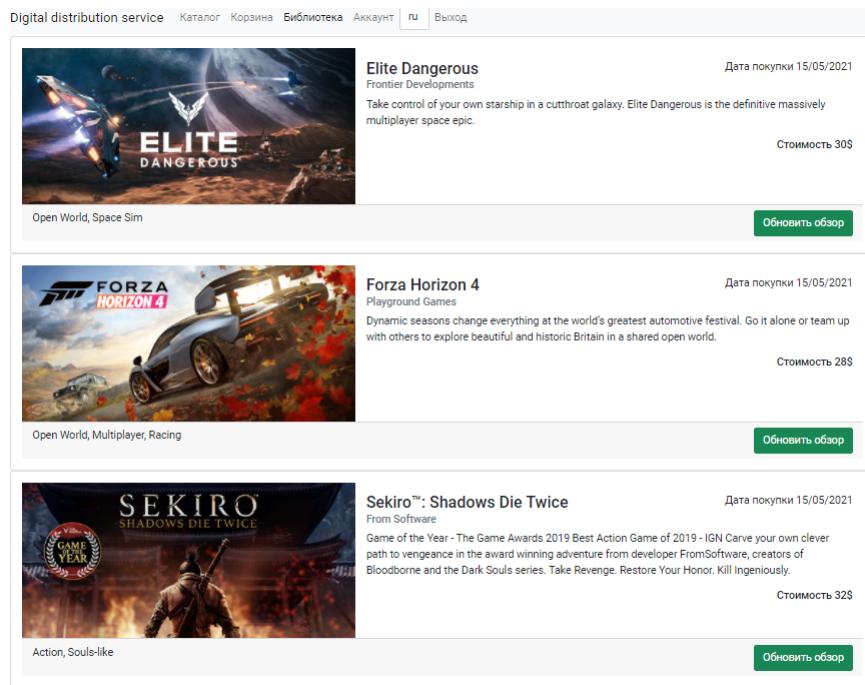


Рисунок 5.6 – Библиотека пользователя

Таблица 5.6 – Тест-кейс добавления отзыва

Тест	Ожидаемый результат	Результат
Добавление отзыва 1. Зайти в аккаунт и перейти на страницу "Библиотека" в навигационной панели 2. Нажать на кнопку "Обновить обзор" напротив выбранной игры 3. Заполнить поля и нажать кнопку "Обновить" 4. Перейти на страницу "Каталог"	1. Перенаправление на страницу "Библиотека" с приобретёнными играми 2. Появление формы для отзыва 3. Появление надписи "Success" 4. Напротив игры отображается её оценка	Успех (Рисунок 5.7)

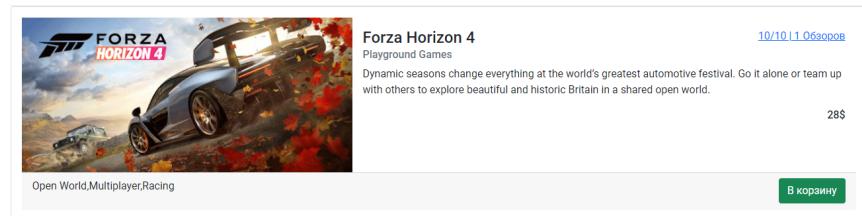


Рисунок 5.7 – Игра с отзывами

Таблица 5.7 – Тест-кейс смены локализации

Тест	Ожидаемый результат	Результат
Смена локализации 1. Зайти в аккаунт и перейти на страницу "Каталог" 2. В навигационной панели выбрать английский язык	1. Перенаправление на страницу "Каталог" 2. Локализация сайта сменилась на английскую	Успех (Рисунок 5.8)

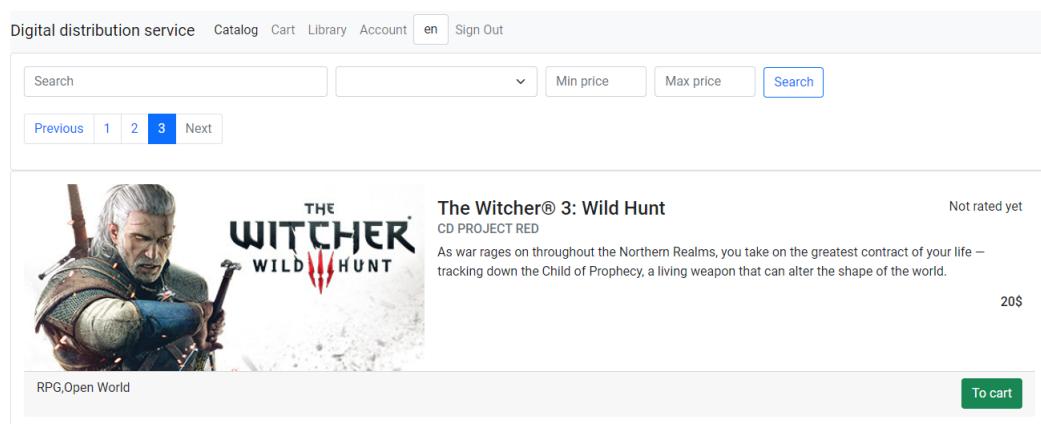


Рисунок 5.8 – Английская локализация сайта

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

6.1 Управление аккаунтом

При входе на сайт первое, что должен сделать пользователь, это зарегистрироваться в системе. Только зарегистрированные пользователи могут приобретать игры и оставлять отзывы. Для создания нового аккаунта необходимо выбрать пункт "Вход" на панели навигации, и на появившейся странице нажать на кнопку "Нет аккаунта?". После этого пользователь будет перенаправлен на форму регистрации (Рисунок 6.1), где необходимо ввести адрес электронной почты и придумать пароль. После успешной регистрации пользователь будет перенаправлен к форме входа и сможет войти в созданный аккаунт.

Авторизованному пользователю становится доступен основной функционал сайта и отображаются новые пункты в навигационной панели. Среди них есть пункт "Аккаунт" при переходе к которому пользователь может сменить свой пароль, заполнив соответствующую форму (Рисунок 6.2).

Также, в случае если пользователь забыл свой пароль и не может войти в аккаунт, он имеет возможность восстановить доступ. Для этого на странице входа необходимо нажать на кнопку "Забыли пароль?", после чего пользователь будет перенаправлен на соответствующую страницу (Рисунок 6.3). Для восстановления доступа необходимо ввести почту, к которой привязан аккаунт, и нажать на кнопку "Отправить" справа от поля. После этого на указанный почтовый ящик пользователю придёт письмо с кодом для восстановления, который необходимо ввести в поле "Пароль" и нажать на кнопку "Отправить" внизу формы. Пользователь будет авторизован в системе и перенаправлен на страницу для смены пароля.

Форма для регистрации (Рисунок 6.1) показывает страницу входа. Виджеты включают: поле для ввода почты (заголовок 'Почта', поле 'Почта...'), поле для ввода пароля (заголовок 'Пароль', поле 'Пароль...'), ссылки 'Забыли пароль?' и 'Нет аккаунта?' (обе ссылки синего цвета), и большая синяя кнопка 'Вход' в нижней части.

Рисунок 6.1 – Форма для регистрации

Digital distribution service Каталог Корзина Библиотека Аккаунт ru Выход

Настройки аккаунта

Новый пароль Пароль Сменить пароль

Рисунок 6.2 – Форма для смены пароля

Восстановление аккаунта

Почта... Отправить

Пароль... К входу в аккаунт

Отправить

Рисунок 6.3 – Форма для восстановления доступа к аккаунту

6.2 Основной функционал

После того, как пользователь вошёл в свой аккаунт, ему становится доступен основной функционал сервиса, который включает в себя добавление игр в корзину, приобретение видеоигр и написание обзоров. На панели навигации появляются пункты "Корзина" и "Библиотека". Для добавления игр в корзину необходимо нажать на соответствующую кнопку в каталоге напротив выбранной игры. После этого кнопка становится неактивной, а игра помещается в корзину пользователя. Количество игр в корзине не ограничено. Когда пользователь определился с выбором и хочет приобрести игры, он должен перейти в корзину через пункт на панели навигации (Рисунок 6.4). Там он может ознакомиться со списком игр готовых к приобретению, увидеть суммарную стоимость и по желанию удалить продукты из корзины. После нажатия на кнопку "Оплатить", игры будут помещены в библиотеку пользователя, попасть в которую можно выбрав соответствующий пункт на панели навигации (Рисунок 6.5). Для того, чтобы оставить отзыв, необходимо нажать на кнопку "Обновить обзор" напротив выбранной игры, и заполнить появившуюся форму. Там же можно отредактировать уже существующий обзор либо удалить его. Все пользовательские обзоры видны в каталоге игр – напротив игры появляется её средняя оценка и ссылка на

все обзоры. По нажатию на эту ссылку пользователь попадает на страницу обзоров, где может подробнее ознакомиться с ними (Рисунок 6.6).

Игра	Цена	Дата добавления	Действие
Elite Dangerous	30\$	15/05/2021	<button>Удалить</button>
Forza Horizon 4	28\$	15/05/2021	<button>Удалить</button>
Sekiro™: Shadows Die Twice	32\$	15/05/2021	<button>Удалить</button>
Всего:	90 \$		

Рисунок 6.4 – Корзина пользователя

Игра	Цена	Дата покупки
 Elite Dangerous Frontier Developments Take control of your own starship in a cutthroat galaxy. Elite Dangerous is the definitive massively multiplayer space epic.	30\$	15/05/2021
 Forza Horizon 4 Playground Games Dynamic seasons change everything at the world's greatest automotive festival. Go it alone or team up with others to explore beautiful and historic Britain in a shared open world.	28\$	15/05/2021
 Sekiro™: Shadows Die Twice From Software Game of the Year - The Game Awards 2019 Best Action Game of 2019 - IGN Carve your own clever path to vengeance in the award winning adventure from developer FromSoftware, creators of Bloodborne and the Dark Souls series. Take Revenge. Restore Your Honor. Kill Ingeniously.	32\$	15/05/2021

Рисунок 6.5 – Библиотека пользователя

test@test.com	15/05/2021
8/10 Great graphics and optimization. Galaxy size is impressive, but the game has no plot so some people can find it boring.	
user@test.com	15/05/2021
9/10 Great experience. Best space sim atm	

Рисунок 6.6 – Подробный просмотр обзоров

6.3 Функционал администратора

Пользователь с ролью "Администратор" обладает дополнительными возможностями в сервисе. Для того, чтобы добавить аккаунт администратора, необходимо создать запись о новом пользователе с соответствующей

ролью в базе данных. При входе в аккаунт администратора, на панели навигации появляется новый пункт "Админ" , по нажатию на который пользователь попадает на панель администратора. Отсюда он может управлять жанрами и добавлять в каталог новые игры. По нажатию на кнопку "Управление жанрами" , пользователь будет перенаправлен на страницу, где можно просмотреть список всех жанров в системе, удалить выбранные, либо добавить новые (Рисунок 6.7). По нажатию на кнопку "Добавить игру" , администратор попадает на страницу для добавления новой игры в каталог (Рисунок 6.8). Для этого необходимо выбрать изображение и заполнить все поля формы. Справа от формы можно добавить к игре несколько жанров из присутствующих в системе.

Дополнительный функционал также появляется на странице с каталогом игр. Здесь напротив каждой игры возникают две дополнительные кнопки: "Удалить" и "Изменить" , нажав на которые администратор может удалить игру из каталога либо отредактировать её описание и жанры (Рисунок 6.9).

The screenshot shows a web-based application interface for managing game genres. On the left, there's a table titled 'Список жанров' (List of genres) with columns 'Жанр' (Genre) and 'Действие' (Action). It lists four genres: Action, First Person, Indie, and Metroidvania, each with a red 'Удалить' (Delete) button. On the right, there's a separate box titled 'Добавить жанр' (Add genre) containing a single input field labeled 'Жанр' (Genre) and a blue 'Добавить' (Add) button.

Рисунок 6.7 – Управление жанрами

The screenshot shows a web-based application interface for updating a game. The main area features a large image of a blue sports car from the game 'Forza Horizon 4'. Below the image is a file selection input field showing 'forza.jpg'. There are several input fields for game details: 'Название игры' (Game title) with 'Forza Horizon 4', 'Разработчик' (Developer) with 'Playground Games', and 'Описание' (Description) with a placeholder text about dynamic seasons. At the bottom, there's a price input field with '28' and a blue 'Обновить игру' (Update game) button. To the right, there's a sidebar with a 'Список жанров' (List of genres) table and a 'Добавить жанр' (Add genre) form.

Рисунок 6.8 – Добавление игры

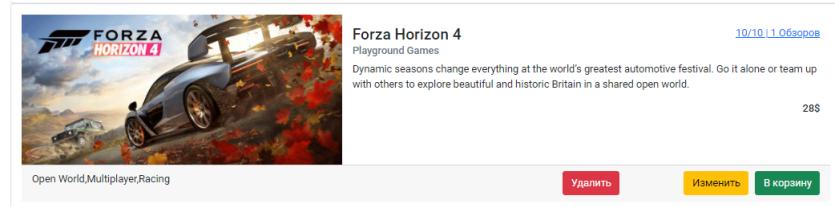


Рисунок 6.9 – Дополнительный функционал в каталоге

ЗАКЛЮЧЕНИЕ

В ходе выполнения данного курсового проекта был спроектирован и разработан онлайн-сервис для продажи цифровых копий видеоигр. Сервис представляет из себя два отдельных проекта – один для серверной части и второй для клиентской. Такое решение позволило вести независимую разработку бизнес-логики сервиса и его пользовательского интерфейса [5]. Это означает, что в будущем сервис может быть дополнен мобильным и настольным приложением без необходимости заново реализовывать логику его работы. Серверная часть сервиса была реализована с помощью технологии ASP.Net WebApi, и взаимодействует с базой данных MS-SQL Server. Для реализации клиентской части был использован язык программирования TypeScript и фреймворк Angular, с помощью которого было создано одностраничное веб-приложение.

Сервис обладает удобным пользовательским интерфейсом и высоким быстродействием по сравнению с рассмотренными аналогами. Также сервис имеет социальный функционал в виде пользовательских отзывов. Для повышения пользовательского опыта была добавлена функция восстановления доступа к аккаунту, реализованы версии сервиса на английском и русском языках.

Полученный в результате выполнения курсового проекта онлайн-сервис удовлетворяет всем функциональным требованиям, выдвинутым на стадии анализа, и успешно прошёл все запланированные тесты.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] ASP.NET Core – Википедия [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/ASP.NET_Core. – Дата доступа: 16.05.21.
- [2] Angular (Фреймворк) – Википедия [Электронный ресурс]. – Электронные данные. – Режим доступа: [https://ru.wikipedia.org/wiki/Angular_\(фреймворк\)](https://ru.wikipedia.org/wiki/Angular_(фреймворк)). – Дата доступа: 16.05.21.
- [3] Фриман, А. Pro ASP.NET Core 3 / А. Фриман. – apress, 2020. – 992 Р.
- [4] Фриман, А. Angular для профессионалов / А. Фриман. – apress, 2018. – 800 Р.
- [5] Пьюоривал, С. Основы разработки веб-приложений / С. Пьюоривал. – Питер, 2015. – 272 Р.

ПРИЛОЖЕНИЕ А
(рекомендуемое)
Исходный код компонента добавления игры

```
import { Component, Input, OnInit } from '@angular/core';
import { FormControl, FormGroup, Validators } from '@angular/
forms';
import { Router, ActivatedRoute } from '@angular/router';
import { Game } from '../.../.../.../models/game';
import { GameGenre } from '../.../.../.../models/gameGenre';
import { Genre } from '../.../.../.../models/genre';
import { DataService } from '../.../.../.../services/data.service';

@Component({
  selector: 'app-update-game',
  templateUrl: './update-game.component.html',
  styleUrls: ['./update-game.component.css']
})
export class UpdateGameComponent implements OnInit {
  game: Game = null;
  leftoverGenres: Genre[];
  imageFile: File;

  form = new FormGroup({
    name: new FormControl('', [Validators.required]),
    developer: new FormControl('', [Validators.required]),
    description: new FormControl('', [Validators.required]),
    price: new FormControl('', [Validators.required]),
  });
  formGenre = new FormGroup({
    name: new FormControl('', [Validators.required]),
    selectedGenre: new FormControl('', [Validators.required])
  });

  error: string;
  message: string;

  constructor(private dataService: DataService, private route: ActivatedRoute) { }
  ngOnInit(): void {
    this.dataService.storeService.getGenres().subscribe(data =>
      this.leftoverGenres = data);
    this.route.params.subscribe(params => {
      let id = params['id'];
    })
  }
}
```

```

    if (id === 0) {
        this.game = new Game();
        this.game.gameId = 0;
    }
    else {
        this.dataService.storeService.getGame(id).subscribe(data
            =>
        {
            this.game = data;
            for (var i = 0; i < this.game.gameGenres.length; i++) {
                this.leftoverGenres.splice(this.leftoverGenres.
                    findIndex(g => g.genreId === this.game.gameGenres[i].
                        genreId), 1);
            }
        });
    }
}

onSubmit() {
    if (this.game === null || this.game.gameId === 0) {
        this.dataService.adminService.addGame(
            this.form.get('name').value,
            this.form.get('developer').value,
            this.form.get('description').value,
            this.form.get('price').value,
            this.imageFile
        ).subscribe((data: Game) => {
            this.game = data;
            this.message = "Game_was_added_to_catalog"
        }, (error: string) => this.error = error)
    }
    else {
        this.dataService.adminService.updateGame(
            this.game.gameId,
            this.form.get('name').value,
            this.form.get('developer').value,
            this.form.get('description').value,
            this.form.get('price').value,
            this.imageFile
        ).subscribe((data: Game) => {
            this.game = data;
            this.message = "Game_was_updated"
        }, (error: string) => this.error = error)
    }
}

```

```

    }

onDeleteGameGenre(gameId: number, genreId: number) {
  this.dataService.adminService.deleteGameGenre(gameId, genreId)
    .subscribe(() => {
      var genre = this.game.gameGenres.find(g => g.genreId ===
        genreId).genre;
      this.game.gameGenres.splice(this.game.gameGenres.findIndex(
        g => g.genreId === genre.genreId), 1);
      this.leftoverGenres.push(genre);
    });
}

onAddGameGenre(gameId: number, genreId: number) {
  this.dataService.adminService.addGameGenre(gameId, genreId).
    subscribe(() =>
  {
    var genre = this.leftoverGenres.find(g => g.genreId ===
      genreId);
    this.leftoverGenres.splice(this.leftoverGenres.findIndex(g
      => g.genreId === genre.genreId), 1);
    this.game.gameGenres.push(new GameGenre(genre.genreId,
      gameId, genre));
  })
}

handleFileInput(files: FileList) {
  var reader = new FileReader();
  reader.readAsDataURL(files[0]);
  reader.onload = (_event) => {
    this.game.image = reader.result.toString().split(',') [1];
  }
  this.imageFile = files.item(0);
}

}

```