

**«Санкт-Петербургский государственный электротехнический университет
«ЛЭТИ» им. В.И.Ульянова (Ленина)»
(СПбГЭТУ «ЛЭТИ»)**

Направление	09.03.04 – Программная инженерия
Профиль	Разработка распределенных программных систем
Факультет	ФКТИ
Кафедра	МО ЭВМ

К защите допустить

Зав. кафедрой

Кринкин К.В.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
МАГИСТРА**

Тема: «Исследование модели обращения к Web-службе ASP.NET через прокси сборку на примере разработки сервиса по аренде видеофильмов»

Студент		<hr/>	Маслов Н.Д.
		<i>подпись</i>	
Руководитель	К.Т.Н.	<hr/>	Попова Е.В.
	<i>(Уч. степень, уч. звание)</i>	<i>подпись</i>	
Консультант кафедры ПЭ	к.н. доцент кафедры ПЭ	<hr/>	Садырин И.А.
	<i>(Уч. степень, уч. звание)</i>	<i>подпись</i>	

Санкт-Петербург

2021

ЗАДАНИЕ

Утверждаю

Зав. кафедрой МО ЭВМ

_____ Кринкин К.В.

« » 20__ г.

Студент Маслов Н.Д.

Группа 5304

Тема работы: «Исследование модели обращения к Web-службе ASP.NET через прокси сборку на примере разработки сервиса по аренде видеофильмов»

Место выполнения ВКР: Санкт-Петербург

Исходные данные (технические требования):

В ВКР должно быть создано Web-приложение, Web-сервис и должна быть настроена модель общения приложения с сервисом напрямую и через прокси.

Содержание ВКР: Теоретические основы разработки на ASP.NET;

Разработка Web-сервиса, Web-приложения, создания их модели общения напрямую и через прокси; Исследование модели обращения; Составление бизнес-плана по коммерциализации результатов НИР магистранта;

Перечень отчетных материалов: пояснительная записка, иллюстративный материал.

Дополнительные разделы: Составление бизнес-плана по коммерциализации результатов НИР магистранта

Дата выдачи задания

Дата представления ВКР к защите

« » 20__ г.

« » 20__ г.

Студент

Маслов Н.Д.

Руководитель

Попова Е.В.

(Уч. степень, уч. звание)

КАЛЕНДАРНЫЙ ПЛАН ВЫПОЛНЕНИЯ ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Утверждаю

Зав. кафедрой МО ЭВМ

_____ Кринкин К.В.

« » 2021г.

Студент Маслов Н.Д.

Група 5304

Тема работы: «Исследование модели обращения к Web-службе ASP.NET через прокси сборку на примере разработки сервиса по аренде видеофильмов»

№ п/п	Наименование работ	Срок выполнения
1	Обзор литературы по теме работы	01.01 – 15.01
2	Теоретические основы разработки на ASP.NET;	16.01 – 01.02
3	Разработка Web-сервиса, Web-приложения, создания их модели общения напрямую и через прокси	02.02 – 15.03
4	Исследование модели обращения	16.03 – 01.04
5	Составление бизнес-плана по коммерциализации результатов НИР магистранта	01.04 – 15.04
6	Оформление пояснительной записки	16.04 – 01.05
7	Оформление иллюстративного материала	02.05 – 13.05

Студент

Маслов Н.Д.

Руководитель

Попова Е.В.

(Уч. степень, уч. звание)

РЕФЕРАТ

Пояснительная записка 97 с., 49 рис., 13 табл., 25 источников., 1 прил.

ASP.NET, WEB-ПРИЛОЖЕНИЕ, WEB-СЛУЖБА, ПРОТОКОЛЫ, REST, SOAP, API, ПРОКСИ СБОРКА, MVC, C#, СЕРВИС ПО АРЕНДЕ ВИДЕОФИЛЬМОВ.

Целью выпускной квалификационной работы является исследование модели обращения Web-приложения к Web-службе. Для этого были созданы Web-приложение и Web-служба при помощи языка программирования C# и технологии ASP.net

Объектом исследования являются протоколы подключения, которые используются для передачи данных. Предметом исследования является Web-службы. В выпускной квалификационной работе служба отвечает за работу с базой данных, к которой обращается Web-приложение, демонстрирующее основной функционал сервиса по аренде видеофильмов.

ABSTRACT

Explanatory note 97 p., 49 fig., 13 tab., 25 sources., 1 adj.

ASP.NET, WEB-APPLICATION, WEB-SERVICE, PROTOCOLS, REST, SOAP, API, PROXY ASSEMBLY, MVC, C #, VIDEO RENTAL SERVICE.

The purpose of the final qualifying work is to study the model of accessing a Web application to a Web service. For this, a Web application and a Web service were created using the C # programming language and ASP.net technology.

The object of research is the connection protocols that are used to transfer data. The subject of research is Web services. In the final qualifying work, the service is responsible for working with the database, which is accessed by a Web application that demonstrates the main functionality of the video rental service.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ.....	7
ВВЕДЕНИЕ	8
1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ РАЗРАБОТКИ НА ASP.NET	10
1.1 ASP.NET.....	10
1.2 ПАТТЕРН ПРОЕКТИРОВАНИЯ MVC.....	15
2 РАЗРАБОТКА МОДЕЛИ ВЗАИМОДЕЙСТВИЯ WEB-СЛУЖБЫ И WEB-ПРИЛОЖЕНИЯ	24
2.1 WEB-СЛУЖБА.....	24
2.2 УРОВНИ ПРОТОКОЛОВ ДЛЯ ПЕРЕДАЧИ ДАННЫХ.....	27
2.3 SOAP.....	31
2.4 REST.....	33
2.5 СРАВНЕНИЕ ИСПОЛЬЗОВАНИЯ SOAP И REST	35
3 РАЗРАБОТКА WEB-ПРИЛОЖЕНИЯ И WEB-СЛУЖБЫ И ИССЛЕДОВАНИЕ МОДЕЛИ ИХ ВЗАИМОДЕЙСТВИЯ.	42
3.1 РАЗРАБОТКА СЛУЖБЫ	43
3.2 РАЗРАБОТКА ПРИЛОЖЕНИЯ.....	51
3.3 РЕАЛИЗАЦИЯ ПРОКСИ.....	57
3.4 ИССЛЕДОВАНИЕ МОДЕЛИ ВЗАИМОДЕЙСТВИЯ WEB-ПРИЛОЖЕНИЯ И WEB- СЛУЖБЫ.....	62
4 СОСТАВЛЕНИЕ БИЗНЕС-ПЛАНА ПО КОММЕРЦИАЛИЗАЦИИ РЕЗУЛЬТАТОВ ВКР	72
4.1 ОПИСАНИЕ ПРОЕКТА.	72
4.2 ПЛАН МАРКЕТИНГА.	79
4.3 ПЛАН ПРОИЗВОДСТВА.....	83
4.4 ФИНАНСОВЫЙ ПЛАН	88
ЗАКЛЮЧЕНИЕ	93
СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ	95
ПРИЛОЖЕНИЕ А.....	98

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

MVC – Model View Controller

API – Application Program Interface

SOAP – Simple Object Access Protocol

HTML – Hyper Text Markup Language

HTTP(S) – Hyper Text Transfer Protocol (Security)

URL – Uniform Resource Location

XML - eXtensible Markup Language

DRY - Don't Repeat Yourself

REST - Representational State Transfer

FTP - File Transfer Protocol

DNS - Domain Name System

SSH - Secure Shell

MAC - Media Access Control

LLC - Logical Link Control

TCP - Transmission Control Protocol

UDP - User Datagram Protocol

CORBA - Common Object Request Broker Architecture

RPC - Remote Procedure Call

IDE - Integrated Development Environment

MVP - Minimum Viable Product

RTT- Road trip time

ВВЕДЕНИЕ

В мире для разработки веб-приложений и веб сервисов существует огромное множество различных фреймворков и языков. У каждой платформы есть свои особенности и преимущества. ASP.NET – это платформа для разработки веб-приложений с открытым исходным кодом, разработанная Microsoft для Windows и запущенная в начале 2000-х годов.

ASP.NET в настоящее время является одной из самых популярных и перспективных платформ. Это кроссплатформенная высокопроизводительная среда с открытым исходным кодом. При разработке был использован паттерн MVC(Model-View-Controller), который соответствует архитектурному шаблону: модель – представление – контроллер. Веб-сайты и приложения, созданные с помощью ASP.NET, могут быть быстрее и эффективнее, чем, например, создание web-служб с помощью PHP. Приложения ASP.NET компилируются, что означает, что код переводится в объектный код, который затем выполняется. Этот процесс компиляции занимает мало времени, но происходит только один раз. После компиляции код может очень быстро выполняться платформой .NET, это обеспечивает высокую скорость работы приложений, разработанных при помощи данной платформы.

Объектом исследования выпускной квалификационной работы является архитектура REST, которая используется для создания web-служб. Для взаимодействия клиента с сервером используется протокол HTTP – протокол передачи гипертекста. Этот протокол передачи данных прост в понимании, удобен в поддержке, отладке, а также удобен для межплатформенного взаимодействия.

При разработке больших приложений требуется быстрое и надёжное взаимодействие клиентской и серверной частей. Для того что бы приложение быстро и без ошибок выполняло возложенные на него функции необходимо правильно разработать модель взаимодействия двух его частей. Поэтому исследование модели взаимодействия Web-службы и Web-приложения будет являться всегда актуальной.

Целью данной работы является исследование модели взаимодействия между клиентом и службой напрямую и через прокси. Для проведения исследований необходимо:

- Изучить теоретические аспекты разработки web-приложений с использованием технологии ASP.NET.
- Разработать модель взаимодействия Web-службы и Web-приложения.
- Разработать Web-приложение и Web-службу и провести исследование модели их взаимодействия.

1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ РАЗРАБОТКИ НА ASP.NET

1.1 ASP.NET

ASP.NET — это бесплатная веб-платформа для создания веб-сайтов и веб-приложений с использованием HTML, CSS и JavaScript. Вы также можете создавать веб-API и использовать технологии реального времени, такие как веб-сокеты.

В данной технологии активно используются такие понятия как web-приложение, web-сайт и web-сервер. «Web-приложение — это набор взаимосвязанных файлов (*.htm, *.asp, *.aspx, файлов изображения и т.п.), а так же связанных с ними компонентов (двоичных файлов .NET или классического COM), которые размещены на web-сервере [1].» Web-сайт в свою очередь является информационной единицей в интернете, состоящий из web-страниц, которые несут исключительно информацию и не предоставляют сложного функционала. Web-сервер — с точки зрения программного обеспечения, это функционал, который контролирует доступ пользователей к размещённым на сервере файлам.

Технология ASP.NET была создана на основе популярной архитектуры ASP, которая использовалась для создания web-приложений, но данная архитектура обладала недостатками. Главный из них — это использование скриптов, что приводило к снижению производительности всего приложения, помимо этого использование скриптов ограничивало возможности применения различных технологий объектно-ориентированного программирования.

Помимо этого, в классических ASP не было хорошей реализации разделения логики представления от бизнес-логики, т.е. HTML (Hypertext Markup Language) код страниц, смешивался с кодом скриптов

Ещё одним недостатком, который стоит упомянуть, является необходимость переносить один и тот же функционал из проекта в проект. Например действия, которые требуется выполнять всегда, независимо от проекта, такие как форматирование HTML, проверка данных, вводимых пользователем и тому

подобные. Вместо использования готовых решений приходилось копировать код и вставлять в новый проект.

В ASP.NET устранены многие недоработки ASP. ASP.NET обладает следующими преимуществами в сравнении с классическим ASP:

- Ускоряет процесс разработки в сравнении с написанием без фреймворка
- Высокое быстродействие как следствие компилируемого кода
- Большой набор компонентов и продуманную схему их взаимодействия, скрывающие от программиста массу проблем, например хранение состояния
- Отличную документацию, что позволяет быстрее вникнуть во все процессы разработки
- Развитое комьюнити, большинство проблем с которыми можно столкнуться при разработке, уже решены другими разработчиками
- Большое количество, готовых решений из-за активного использования большими компаниями

Но помимо преимуществ ASP.NET так же имеет некоторые недостатки:

- Клиентская логика приложения тесно связана с логикой сервера, т.е. нет четкого разделения между уровнями приложения
- Большой размер страниц, так как они вмещают в себя часть серверной логики приложения

В ASP.NET для каждого сеанса подключения к приложению хранит свою уникальную информацию при помощи типа `HttpSessionState`. Иначе говоря каждому подключенному пользователю выделяется область оперативной памяти приложения, в которой хранится информация о его взаимодействии с приложением. Каждый `HttpSessionState` является вложенным типом глобального состояния приложения `HttpApplicationState`. На рисунке 1 представлена схема подключения клиента к приложению.

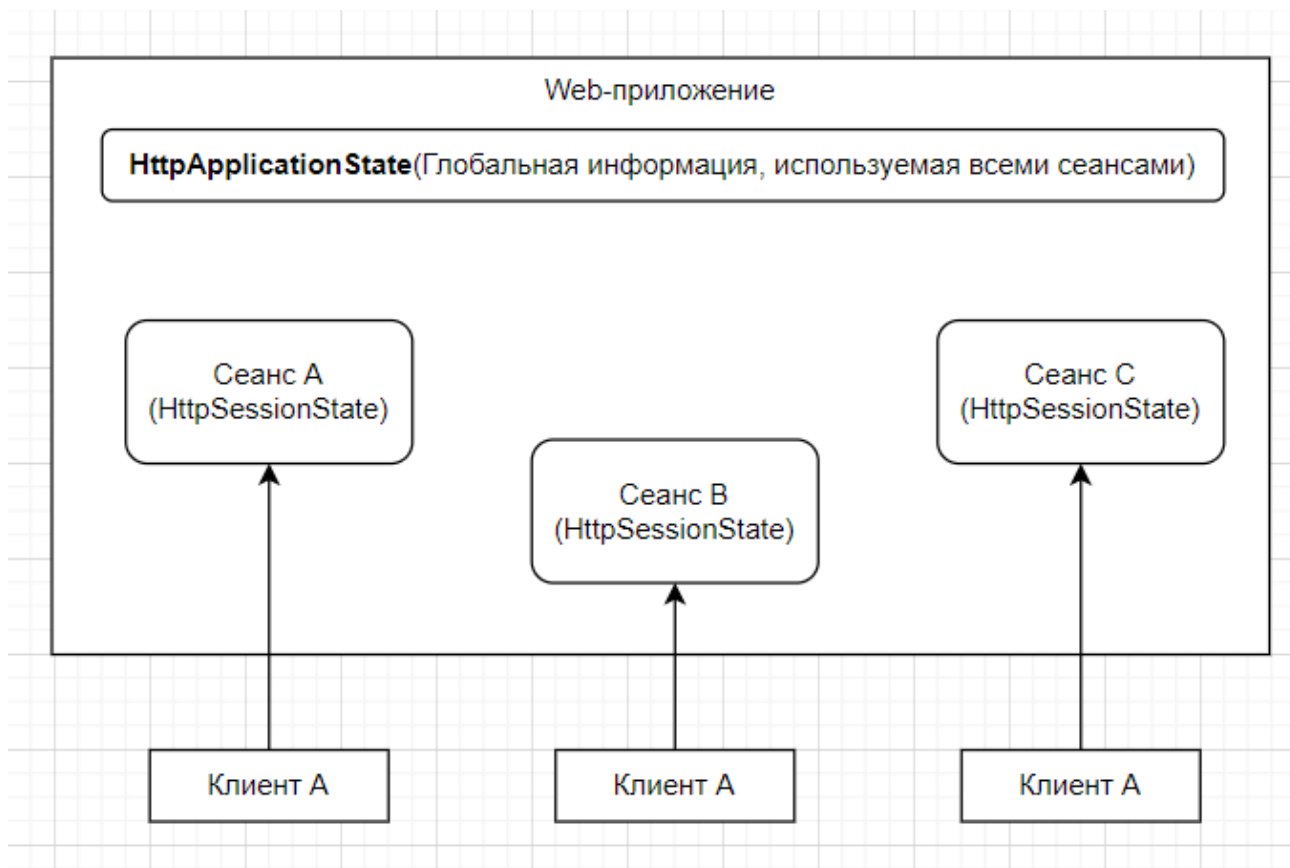


Рисунок 1 – Приложение и сеансы подключения в ASP.net

ASP.NET предлагает три платформы для создания веб-приложений: веб-формы, ASP.NET MVC и ASP.NET Web Pages. Все три фреймворка стабильны и зрелы, и с любой из них можно создавать отличные веб-приложения. Независимо от того, какой фреймворк выбрать, везде можно получить все преимущества и возможности ASP.NET.

Каждый фреймворк нацелен на свой стиль разработки. Выбор должен зависеть от комбинации программных активов (знаний, навыков и опыта разработки), типа создаваемого вами приложения и удобного для разработчиков подхода к разработке. Сравнительные характеристики для выбора фреймворков приведены в таблице 1.

Таблица 1 — Сравнительные характеристики для выбора фреймворков

Технология ASP.NET	Требуемый опыт	Стиль разработки
Web Forms	Win Forms, WPF, .NET	Быстрая разработка с помощью широких возможностей библиотеки элементов управления, которые инкапсулируют разметку HTML
MVC	Ruby on Rails, .NET	Полный контроль над разметкой HTML, код и разметка разделены, упрощенное написание тестов. Лучший выбор для мобильных устройств и одностраничных приложений (SPA).
Web Pages	Classic ASP, PHP	Разметка HTML и код вместе в одном файле

С помощью веб-форм ASP.NET можно создавать динамические веб-сайты. Область разработки и сотни элементов управления и компонентов позволяют быстро создавать сложные и мощные сайты на основе пользовательского интерфейса с доступом к данным.

ASP.NET MVC предоставляет мощный, основанный на шаблонах способ создания динамических веб-сайтов, который обеспечивает четкое разделение задач и дает вам полный контроль над разметкой для приятной и быстрой разработки. ASP.NET MVC включает в себя множество функций, которые обеспечивают быструю, удобную для TDD разработку для создания сложных приложений, использующих новейшие веб-стандарты.

Веб-страницы ASP.NET и синтаксис Razor обеспечивают быстрый, доступный и легкий способ объединения серверного кода с HTML для создания динамического веб-содержимого. Можно подключаться к базам данных, добавлять видео, ссылаться на сайты социальных сетей и включать множество других функций, которые помогут создавать красивые сайты, соответствующие последним веб-стандартам.

Все три платформы ASP.NET основаны на .NET Framework и разделяют основные функции .NET и ASP.NET. Например, все три платформы предлагают модель безопасности входа, основанную на членстве, и все три используют одни и те же средства для управления запросами, обработки сеансов и т. д., Которые являются частью основных функций ASP.NET.

Кроме того, эти три фреймворка не являются полностью независимыми, и выбор одного не исключает использования другого. Поскольку фреймворки могут сосуществовать в одном и том же веб-приложении, нередко можно увидеть отдельные компоненты приложений, написанных с использованием разных фреймворков. Например, части приложения, ориентированные на клиента, могут быть разработаны в MVC для оптимизации разметки, в то время как части доступа к данным и административные части разрабатываются в веб-формах, чтобы использовать преимущества элементов управления данными и простого доступа к данным.

ASP.NET может поддерживать собственные мобильные приложения с серверной частью веб-API, а также мобильные веб-сайты с помощью гибких платформ дизайна, таких как Twitter Bootstrap. Если вы создаете собственное мобильное приложение, легко создать веб-API на основе JSON для обработки доступа к данным, аутентификации и push-уведомлений для приложения. Если создается адаптивный мобильный сайт, можно использовать любую структуру CSS или открытую сеточную систему, которую предпочитают разработчики, или выбрать мощную мобильную систему, такую как jQuery Mobile или Sencha.

Одностраничное приложение ASP.NET (SPA) помогает создавать приложения, которые включают значительные взаимодействия на стороне клиента с использованием HTML 5, CSS 3 и JavaScript.

WebHooks — это упрощенный шаблон HTTP, обеспечивающий простую модель публикации/подгруппы для соединения веб-API и служб SaaS. Когда в службе происходит событие, зарегистрированным подписчикам отправляется уведомление в виде HTTP-запроса POST. Запрос POST содержит информацию о событии, которая позволяет получателю действовать соответствующим образом.

WebHooks доступны в большом количестве сервисов, включая Dropbox, GitHub, Instagram, MailChimp, PayPal, Slack, Trello и многих других. Например, WebHook может указывать на то, что файл был изменен в Dropbox, или изменение кода было зафиксировано в GitHub, или платеж был инициирован в PayPal.

1.2 Паттерн проектирования MVC

ASP.NET MVC — это среда веб-разработки от Microsoft, которая сочетает в себе эффективность и аккуратность архитектуры модель-представление-контроллер (MVC), самые современные идеи и методы гибкой разработки и лучшие части существующего ASP. Платформа .NET. Это полная альтернатива традиционным веб-формам ASP.NET, обеспечивающая преимущества для всех, кроме самых тривиальных проектов веб-разработки.

Когда в 2002 году впервые был выпущен ASP.NET 1.0, ASP.NET и веб-формам представлялись как одно и то же. Однако ASP.NET всегда поддерживал два уровня абстракции:

- `System.Web.ui`: слой веб-форм, включающий серверные элементы управления, `ViewState` и т. д.
- `System.Web`: инструменты, которые обеспечивают базовый веб-стек, включая модули, обработчики, HTTP-стек и т. д.

Основной метод разработки с помощью ASP.NET включал весь стек веб-форм, используя преимущества серверных элементов управления сохранности состояния, одновременно устраняя скрытые сложности (жизненный цикл страницы, менее оптимальный HTML, который сложно настроить, и т. д.).

Однако всегда была возможность отказаться от всего этого — отвечать напрямую на HTTP-запросы, создавать веб-фреймворки именно так, как нужно разработчикам, чтобы они работали, создавая масштабируемы HTML-код — используя обработчики, модули и другие методы. При использовании нативного HTML сделать это было затруднительно из-за отсутствия встроенного шаблона, который поддерживал бы что-либо из этих вещей. К моменту анонса ASP.NET MVC в 2007 году шаблон MVC становился одним из самых популярных способов создания веб-фреймворков. На рисунке 2 визуализирован паттерн MVC.

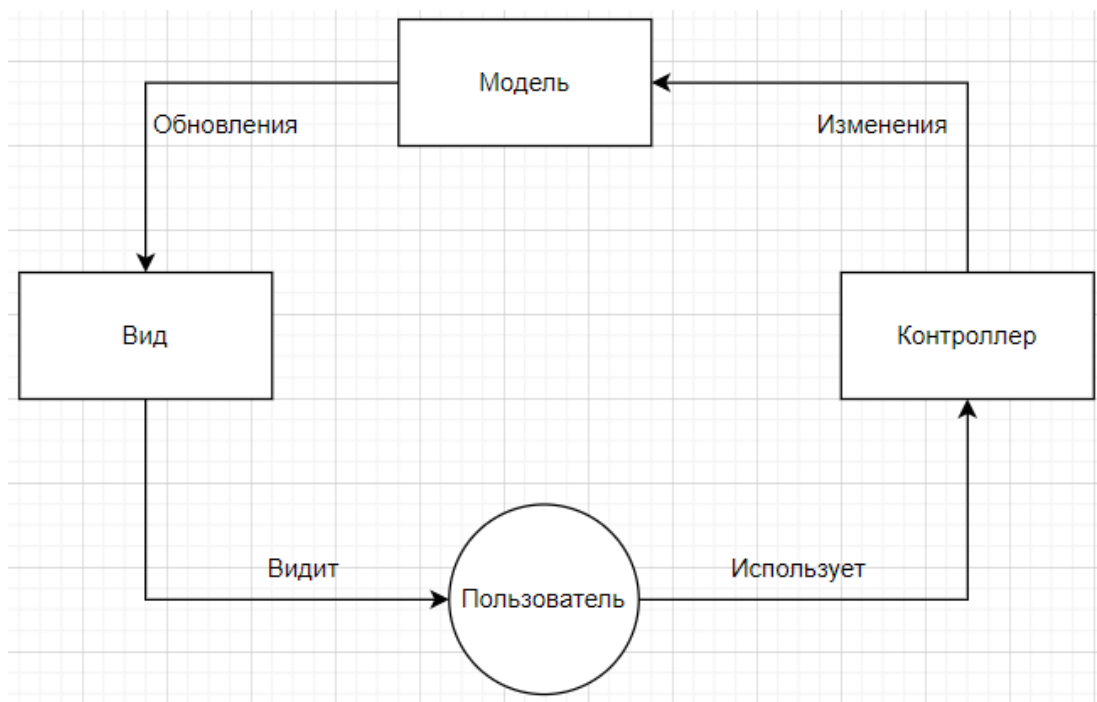


Рисунок 2 – Визуализация паттерна MVC

Модель-представление-контроллер (MVC) уже много лет является важным архитектурным шаблоном в информатике. Первоначально названный Thing-Model-View-Editor в 1979 году, позже он был упрощен до Model-View-Controller. Это мощное средство разделения проблем в приложении (например, отделение логики доступа к данным от логики отображения), которое очень хорошо применимо к веб-приложениям. Явное разделение задач добавляет небольшую дополнительную сложность к дизайну приложения, но исключительные преимущества перевешивают дополнительные усилия. С момента своего появления он использовался в десятках фреймворков. MVC разделяет пользовательский интерфейс (UI) приложения на три основных аспекта:

- **Модель:** набор классов, описывающих данные, с которыми работает разработчик, а также бизнес-правила для изменения и управления данными.
- **Вид:** определяет, как будет отображаться пользовательский интерфейс приложения.
- **Контроллер:** набор классов, которые обрабатывают запросы от пользователя, общий поток приложения и логику, специфичную для приложения.

Шаблон MVC часто используется в веб-программировании. В ASP.NET MVC интерпретируется как:

- Модели: это классы, которые представляют интересующий домен. Эти объекты домена часто инкапсулируют данные, хранящиеся в базе данных, а также код, который манипулирует данными и обеспечивает выполнение бизнес-логики, зависящей от предметной области. С ASP.NET MVC это, скорее всего, какой-то уровень доступа к данным, использующий такой инструмент, как Entity Framework или NHibernate, в сочетании с настраиваемым кодом, содержащим логику, зависящую от предметной области.

- Вид: это шаблон для динамической генерации HTML.

- Контроллер: это специальный класс, который управляет отношениями между представлением и моделью. Он реагирует на ввод пользователя, взаимодействует с моделью и решает, какой вид визуализировать (если есть). В ASP.NET MVC этот класс обычно обозначается суффиксом Controller.

Важно помнить, что MVC — это архитектурный шаблон высокого уровня, и его применение зависит от использования. ASP.NET MVC контекстно привязан как к проблемной области (веб-среда без сохранения состояния), так и к хост-системе (ASP.NET).

ASP.NET MVC опирается на многие из тех же основных стратегий, что и другие платформы MVC, плюс он предлагает преимущества скомпилированного и управляемого кода и использует новые возможности языка .NET, такие как лямбда-выражения, динамические и анонимные типы. Однако в своей основе ASP.NET применяет фундаментальные принципы, обнаруженные в большинстве веб-фреймворков на основе MVC:

- Соглашение важнее конфигурации
- Не повторяйся (также известный как принцип «DRY (Don't repeat yourself)»)
- Возможность подключения везде, где это возможно

- Постарайтесь быть полезным, но при необходимости не мешайте разработчику

MVC 5 был выпущен вместе с Visual Studio 2013 в октябре 2013 года. Основное внимание в этом выпуске уделялось инициативе «One ASP.NET» и основным улучшениям во фреймворках ASP.NET. Некоторые из основных функций включают в себя:

- One ASP.NET
- ASP.NET Identity
- Шаблоны начальной загрузки
- Маршрутизация атрибутов
- Шаблоны ASP.NET
- Фильтры аутентификации
- Переопределения фильтра

One ASP.NET. Может показаться, что возможность комплектации фреймворка является преимуществом. Веб-приложения довольно сильно различаются, а веб-инструменты и платформы не подходят для всех. С другой стороны, некоторые варианты выбора могут парализовать. Многим разработчикам не нравится выбирать что-то одно, если это означает отказ от чего-то другого. Это вдвойне применимо к выбору в начале проекта: не всегда известно, что потребуется для этого проекта через год.

В предыдущих версиях MVC разработчики часто сталкивались с выбором при создании проекта. Нужно было выбирать между приложением MVC, приложением веб-форм или каким-либо другим типом проекта. После того, как было принято решение, можно столкнуться со сложностями в реализации, например можно добавить веб-формы в приложение MVC, но добавить MVC в приложение веб-форм было сложно. Приложения MVC имели специальный GUID типа проекта, скрытый в их файле `csproj`, и это было лишь одно из таинственных изменений, которые приходилось внести при попытке добавить MVC в приложения веб-форм.

В MVC 5 все это уходит, потому что существует только один тип проекта ASP.NET. Когда создается новое веб-приложение в Visual Studio 2013 и выше, нет сложного выбора, просто веб-приложение. Это поддерживается не только при первом создании проекта ASP.NET; можно добавить поддержку других фреймворков по мере разработки, поскольку инструменты и функции поставляются в виде пакетов NuGet. Например, если разработчик позже передумает, он может использовать шаблоны ASP.NET для добавления MVC в любое существующее приложение ASP.NET.

ASP.NET Identity. Системы членства и аутентификации в MVC 5 были полностью переписаны как часть новой системы идентификации ASP.NET. Эта новая система выходит за рамки некоторых устаревших ограничений предыдущей системы членства ASP.NET, добавляя при этом некоторую сложность и настраиваемость в систему простого членства, которая поставлялась с MVC 4.

Вот некоторые из основных новых функций в ASP.NET Identity:

- Единая система идентификации ASP.NET: в поддержку фокусировки One ASP.NET, новая идентификация ASP.NET была разработана для работы во всем семействе ASP.NET (MVC, веб-формы, веб-страницы, веб-API, SignalR, и гибридные приложения, использующие любую комбинацию).
- Контроль над данными профиля пользователя. Хотя это приложение часто используется для хранения дополнительной настраиваемой информации о пользователях, система членства ASP.NET очень затрудняла это. ASP.NET Identity позволяет хранить дополнительную информацию о пользователях (например, номера учетных записей, информацию в социальных сетях и контактный адрес) так же легко, как добавление свойств в класс модели, представляющий пользователя. По умолчанию вся пользовательская информация хранится с использованием Entity Framework Code First. Это дает простоту и контроль. Однако можно подключать любой другой механизм хранения, включая другие ORM, базы данных, собственные пользовательские веб-службы и так далее.

- **Возможность тестирования:** ASP.NET Identity API был разработан с использованием интерфейсов. Это позволяет писать модульные тесты для кода пользовательского приложения.

- **Проверка подлинности на основе утверждений:** хотя ASP.NET Identity по-прежнему предлагает поддержку ролей пользователей, она также поддерживает проверку подлинности на основе утверждений. Утверждения намного более выразительны, чем роли, поэтому это дает гораздо больше возможностей и гибкости. Принимая во внимание, что членство в роли — это простое логическое значение (пользователь либо является, либо не входит в роль администратора), утверждение пользователя может нести обширную информацию, такую как уровень членства пользователя или особенности идентификации.

- **Поставщики входа:** вместо того, чтобы сосредоточиться только на аутентификации имени пользователя и пароля, ASP.NET Identity понимает, что пользователи часто проходят аутентификацию через социальных провайдеров (например, учетную запись Microsoft, Facebook или Twitter) и Windows Azure Active Directory.

- **Распространение NuGet:** ASP.NET Identity устанавливается в ваших приложениях как пакет NuGet. Это означает, что можно установить его отдельно, а также обновить до более новых выпусков, просто обновив один пакет NuGet.

Шаблоны начальной загрузки. В MVC 4 и HTML, и CSS шаблоны по умолчанию были переработаны, чтобы выглядеть более презентабельно из коробки. Также они хорошо работают с разными разрешениями экрана. Однако все HTML и CSS в шаблонах по умолчанию MVC 4 были настраиваемыми, что было не идеально. Обновления визуального дизайна были привязаны к циклу выпуска продукта MVC, и нельзя было легко поделиться шаблонами дизайна с более широким сообществом веб-разработчиков.

В MVC 5 шаблоны проектов были переведены на популярную платформу Bootstrap. Первоначально Bootstrap был создан разработчиком и дизайнером из Twitter, они позже отделились, чтобы полностью сосредоточиться на Bootstrap.

Дизайн по умолчанию для MVC 5 выглядит, что его можно развернуть его в производственной среде, как показано на рисунке 3.

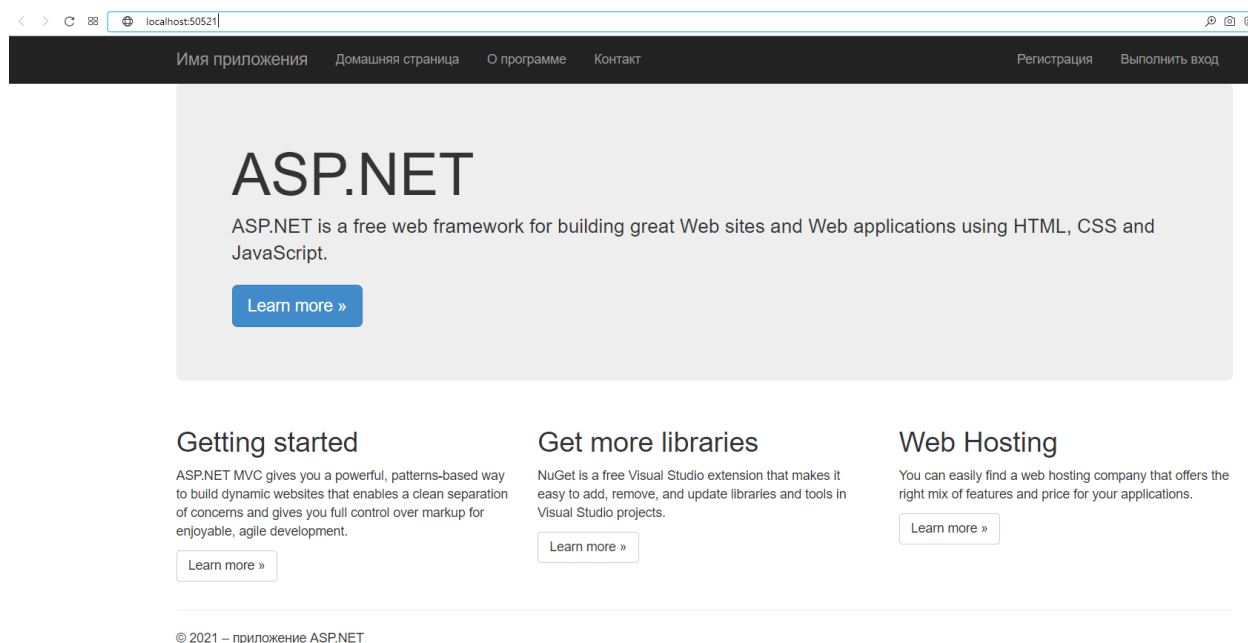


Рисунок 3 — Дизайн по умолчанию для MVC 5

Поскольку фреймворк Bootstrap получил широкое признание в сообществе веб-разработчиков, большое количество тем Bootstrap (как бесплатных, так и платных) доступно на таких сайтах, как <http://wrapbootstrap.com> и <http://bootswatch.com>.

Маршрутизация атрибутов — это новая опция для указания маршрутов путем размещения аннотаций в классах контроллера или методах действий. Это стало возможным благодаря вкладу с открытым исходным кодом популярного проекта *AttributeRouting* (<http://attributerouting.net>).

ASP.NET Scaffolding. Scaffolding — это процесс создания шаблонного кода на основе классов модели. В MVC есть scaffolding, начиная с версии 1, но он был ограничен проектами MVC. Новая система шаблонов ASP.NET работает в любом приложении ASP.NET. Кроме того, имеется поддержка создания мощных пользовательских шаблонов, в комплекте с настраиваемыми диалоговыми окнами и комплексным API-интерфейсом шаблонов.

Фильтры аутентификации. MVC уже давно поддерживает функцию, называемую фильтрами авторизации, которые позволяют ограничивать доступ к контроллеру или действию на основе членства в ролях или другой настраиваемой логики. Однако, существует важное различие между аутентификацией (подтверждение пользователя) и авторизацией (что разрешено делать аутентифицированному пользователю). Недавно добавленные фильтры проверки подлинности выполняются перед фильтром авторизации, что позволяет получить доступ к утверждениям пользователей, которые предоставляет ASP.NET Identity, и запустить собственную логику проверки подлинности.

Переопределения фильтров. Фильтры — это расширенная функция MVC, которая позволяет разработчику участвовать в конвейере выполнения действий и результатов. Переопределения фильтра означают, что можно исключить контроллер или действия из выполнения глобального фильтра.

Структура приложения MVC. Когда создается новое приложение ASP.NET MVC с помощью Visual Studio, автоматически добавляется в проект несколько файлов и каталогов, как показано на рисунке 4.

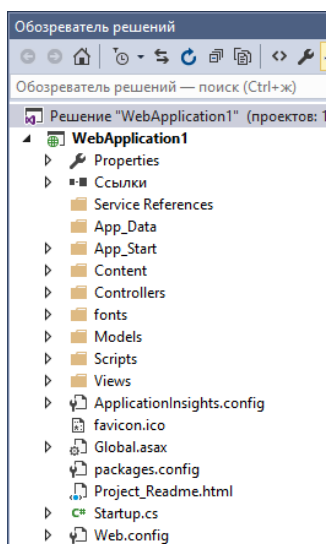


Рисунок 4 — Структура приложения MVC

Проекты ASP.NET MVC, созданные с помощью шаблона MVC, имеют восемь каталогов верхнего уровня, показанных в таблице 2.

Таблица 2 — Описание директорий приложения MVC

Директория	Назначение
/Controllers	Классы контроллеров, которые обрабатывают URL-запросы.
/Models	Классы, которые представляют и управляют данными и бизнес-объектами.
/Views	Файлы шаблонов пользовательского интерфейса, которые отвечают за рендеринг вывода, например HTML.
/Scripts	Файлы библиотеки JavaScript и скрипты (.js).
/fonts	Система шаблонов Bootstrap включает некоторые пользовательские веб-шрифты, которые помещаются в этот каталог.
/Content	CSS, изображения и другое содержимое сайта, кроме скриптов.
/App Data	Файлы данных, которые нужно читать/писать.
/App_Start	Код конфигурации для функций веб-API, таких как маршрутизация, объединение и Web API.

ASP.NET MVC не требует этой структуры. Фактически, разработчики, работающие над большими приложениями, обычно разделяют приложение на несколько проектов, чтобы сделать его более управляемым (например, классы модели данных часто входят в отдельный проект библиотеки классов из веб-приложения). Однако структура проекта по умолчанию обеспечивает удобное соглашение о каталогах по умолчанию.

2 РАЗРАБОТКА МОДЕЛИ ВЗАИМОДЕЙСТВИЯ WEB-СЛУЖБЫ И WEB-ПРИЛОЖЕНИЯ

2.1 Web-служба

Web-служба — это изолированный модуль кода выполняемый на Web-сервере, что обеспечивает лёгкую поддержку функционала и масштабируемость. Эта технология, которая используется в таких отраслях, как банковское дело, покупки и коммуникации. Web-служба в простом понимании можно описать, как некую услугу, которая предлагается через интернет. Например, в веб-приложении погоды пользователь открывает веб-сайт(клиент) и вводит какие-либо данные, например свой город, который необходим для определения его местоположения, Web-сервер сайта обрабатывает данные и отправляет ответ, в данном случае прогноз погоды. Сервер предоставляет определенные услуги для клиентов через Интернет. Клиенты бывают самых разных видов и типов. В зависимости от в процессе разработки сервиса у клиентов будет минимальный исполняемый код или вообще его не будет. Иногда термин «Web-служба» ошибочно принимают за термин «Web-сайт». Одно важное различие между этими двумя понятиями заключается в том, что Web-служба не предоставляют никакого графического интерфейса для пользователя, как это делают сайты. Web-сайт и web-служба совершенно разные сущности. Web-сайт — это набор страниц, и web-страница может обращаться сразу к нескольким web-службам. Например, сайт, предоставляющий конечному пользователю информацию о курсе валют, может запрашивать информацию о разных валютах у разных источников.

Еще одна функция web-служб является связывание программ друг с другом через удаленные точки доступа и обеспечение более эффективной передачи больших объемов данных. Раннее программное обеспечение работало на одной вычислительной машине с использованием локальных данные. Позже, с появлением распределенных вычислительных системы, стало возможным передавать данные, опубликованные на разных компьютерах, работающих на разных платформах. Аналогично, Web служба

— это тип распределенных вычислений, которые были разработаны для предоставления услуг через Интернет. Web-службы работают на уровне абстракции и способны соединить любую операционную систему, аппаратную платформу или язык программирования точно так же, как Интернет. Web-службы могут обмениваться ресурсами, логикой и данными через сеть. Консорциум World Wide Web характеризует Web-службы как программное обеспечение, которое предназначено для поддержки взаимодействия между машинами через сеть. Большинство web-сайтов взаимодействуют с несколькими web-службами, действующими индивидуально или совместно, в зависимости от требований владельца бизнеса. В качестве примера рассмотрим web-сайт предоставляющий услуги интернет магазина. На таком ресурсе можно увидеть такие функции, как добавление товаров в корзину, их обработка, оформление заказа, и оплата может быть разработана с использованием различных Web-служб, такими как PayPal, GooglePay, ApplePay и другими технологиями. На таком сайте эти интегрированные Web-службы взаимодействуют друг с другом, обеспечивая приятное опыт покупок для пользователя.

Во время работы Web-службы необходимо постоянно обмениваться данными, XML (eXtensible Markup Language) - наиболее распространенный формат для обмена данными между веб-службами. Приложения работающий с web-службами могут обмениваться сообщениями, используя стандартизированные XML-документы или HTML-документами. Таким образом, web-служба также может быть характеризована как набор стандартов на основе XML, которые позволяют обмениваться электронными сообщениями и взаимодействовать независимо от компьютерных платформ или конкретных технологии, используемые взаимодействующими сторонам.

Web-службы позволяют программам эффективно взаимодействовать между собой. Web-служба является единым, независимым компонентов серверной части приложения, что обеспечивает пере используемость данной

службы. А также можно заключить, что Web-службы могут применяться для решения таких задач как:

- Соединение различных систем. Например, в Web-службах можно обращаться к API сторонних сервисов для получения каких-либо данных, сводок погоды, курса валюты и т.д.
- В силу того, что службы являются атомарной единицей серверной части приложения, они легко переиспользуемы в качестве готовых модулей. Возможно, задачи для которых нужна служба уже были решены другими разработчиками и код находится в открытом доступе
- Для реализации видеосвязи между пользователями, все необходимые действия для установки видео связи можно выполнять внутри службы, а наружу модуля возвращать уже готовые данные.

Для того что бы обмениваться данными между Web-службами или между Web-службой и Web-приложением, World Wide Web разработал SOAP (Simple Object Access Protocol), который использует стандарты, такие как XML и HTTP, чтобы облегчить обмен между Web-приложениями, независимо от платформы, на которой они были разработаны. SOAP — это технологическая спецификация, разработанная для помощи при обмене данными в Интернете между Web-службами.

Другой конкурирующий подход, поддерживающий разработку интерфейсов web-служб, является REST (Representational State Transfer). REST — это набор архитектурных принципов для разработки web-приложений. Принципы REST фокусируются на том, как решаются состояния приложения и то как происходит передача данных через HTTP.

Протокол HTTP является протоколом прикладного уровня и имеет 4 основных метода: GET, POST, PUT, DELETE.

Браузер клиента обращается к серверу при помощи протокола передачи данных HTTP, в соответствии со схемой, представленной на рисунке 5.

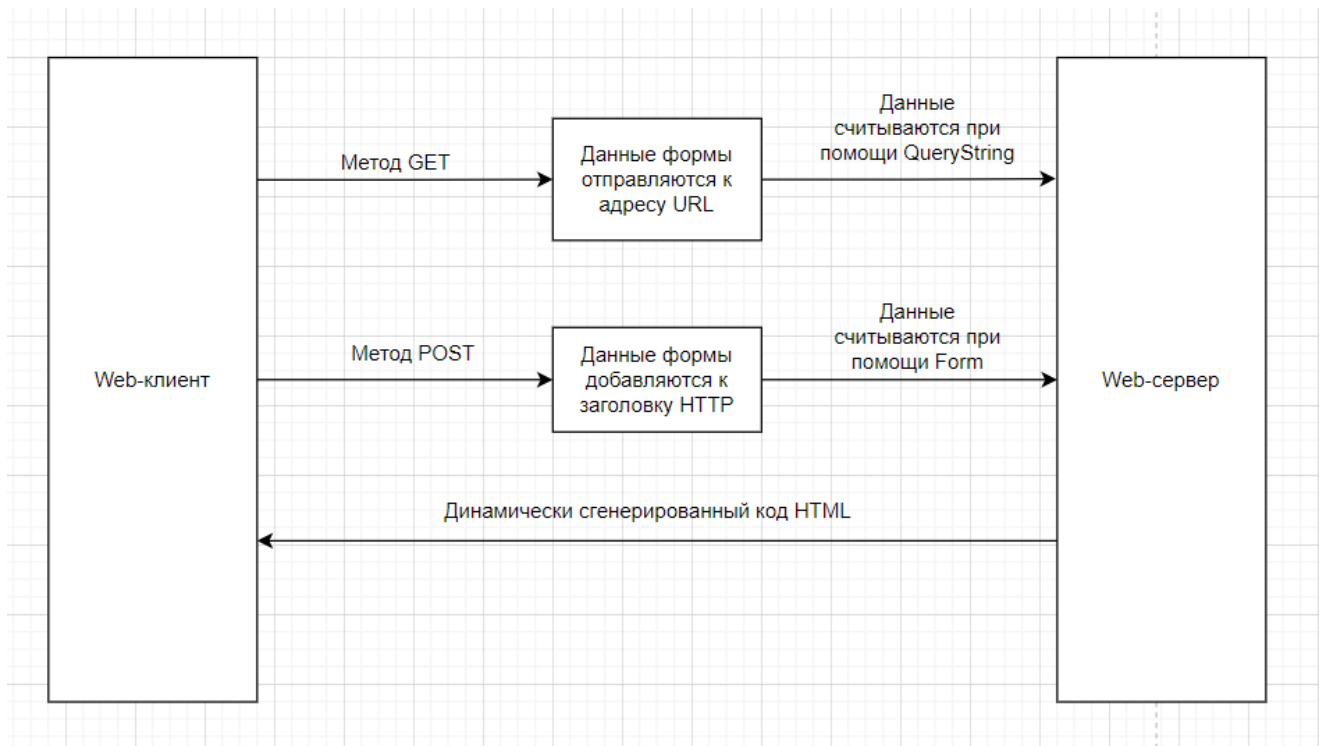


Рисунок 5 – Представление методов HTTP протокола

2.2 Уровни протоколов для передачи данных

Прикладной, уровень которому принадлежит HTTP, является верхним из семи, так же называется уровнем приложения. Помимо прикладного уровня так же существуют такие протоколы передачи данных:

- уровень представления;
- сеансовый уровень;
- транспортный уровень;
- сетевой уровень;
- уровень звена данных (или канальный)
- физический уровень.

Каждый из них выполняет определённую функцию, которая необходимо для быстрой и надёжной передачи пакетов данных. Уровни протоколов представлены на рисунке 6.

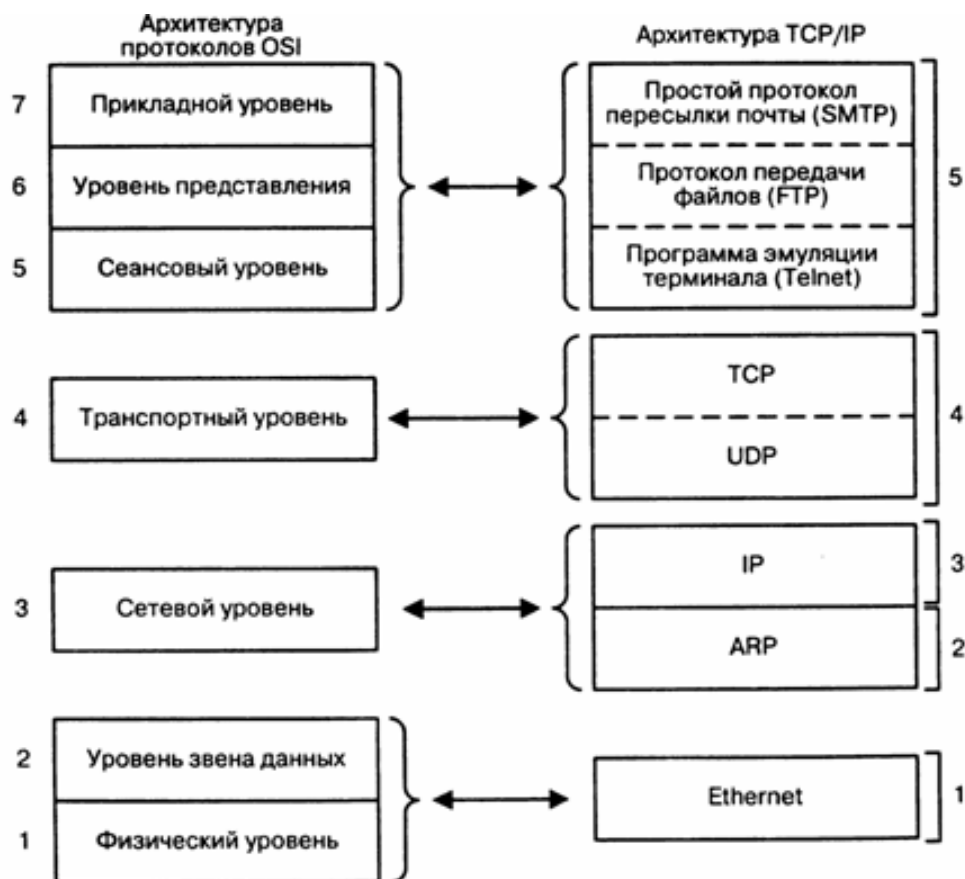


Рисунок 6 – Соответствие протоколов архитектуры OSI к архитектуре TCP/IP

Условно принято разделять 7 уровней на уровни среды и уровни хоста. Уровни среды обеспечивает физическую передачу данных при помощи различных сетевых устройств (коммутатор, маршрутизаторы). Уровни хоста применяются на устройстве, независимо от его типа.

Первым уровнем является физический уровень, на нём происходит физическая передача между устройствами. Средства передачи на этом уровне оперируют битами. Примеров передачи сигнала является оптоволокно, но помимо провода передача через Wi-Fi, Bluetooth или 4G тоже относится к физическому уровню.

После физического уровня идёт канальный уровень, на этом уровне решается проблема адресации во время передачи информации. На канальном уровне происходит компоновка битов. После чего сформированные пакеты с данными отправляются по сети. Проверка и исправление данных на этом уровне

осуществляется при помощи двух подуровней: MAC (Media Access Control) – адрес устройства и LLC (Logical Link Control);

На третьем уровне происходит маршрутизация запроса. При помощи полученного от предыдущего уровня MAC-адреса от коммутатора, происходит построение оптимального маршрута от одного устройства сети к другому. На данном уровне используется протокол ARP (Address Resolution Protocol), который преобразует 64-битные MAC-адреса в 32-битные IP-адреса, благодаря этому происходит инкапсуляция и декапсуляция данных.

Транспортный уровень является четвертым, он является прослойкой между уровнями среды и уровнями хоста. Основной задачей, которая выполняется на данном уровне – это транспортировка пакетов с данными. На транспортном уровне обычно применяется два протокола: TCP (Transmission Control Protocol) и UDP (User Datagram Protocol). Каждый из этих протоколов имеет свои минусы и недостатки. В TCP данные разделяются на сегменты (пакеты) в зависимости от пропускной способности системы. Разделение на пакеты является требованием не надёжных сетей, когда предполагается, что пакет может уйти не тому адресату. В случае с UDP делятся на датаграммы. В отличие от пакетов датаграммы полностью автономны и содержат в себе все необходимые заголовки, что бы дойти до конечного адресата. Датаграммы не зависят от сети и могут отправляться в разном порядке. TCP используется там где требуется более надёжная передача данных, например отправка писем. Отправка каждого пакета данных осуществляется в несколько запросов, что позволяет отследить дошёл ли каждый пакет до получателя и в нужном ли порядке. При UDP проверки не осуществляется, поэтому этот протокол хоть и менее надёжен, но зато обладает большей скоростью, его применяют обычно при передаче видео, где потеря пакета будет не заметной.

Пятый или сеансовый уровень отвечает за синхронизацию данных, например при проведении видеоконференции необходимо, что видео и аудио потоки шли синхронно.

Уровень представления занимается преобразованием протоколов. На данном уровне также происходит шифрование данных.

Последний, прикладной, уровень занимается тем, что бы пользователь видел данные в понятном для него виде.

В таблице 3 представлен процесс передачи запроса между уровнями и изменения передаваемых данных.

Таблица 3 – Процесс передачи запроса между уровнями протоколов

Семиуровневая модель ISO		PDU			
1	Прикладной уровень	Данные	Уровень Хоста	↑ Декапсуляция	↓ Инкапсуляция
2	Уровень представления				
3	Сеансовый уровень				
4	Транспортный уровень	Сегмент, Датаграмма			
5	Сетевой уровень	Пакет	Уровень Среды		
6	Канальный уровень	Кадр			
7	Физический уровень	Бит			

На прикладном уровне помимо HTTP можно использовать множество протоколов, такие FTP (File Transfer Protocol), DNS (Domain Name System), SSH (Secure Shell) и т.д. Все они имеют свои особенности и применяются для разных целях, Но самым распространённым является HTTP, именно благодаря ему заходя на сайт мы получаем страницу HTML. Для эффективного построения Web-службы на основе протокола HTTP можно использовать несколько подходов, самыми популярными является, применяя архитектуру REST или с использованием SOAP.

Хотя SOAP и REST оба используются для создания web-службы, они различаются способами обработки данных, но главное их различие заключается в том, что SOAP — это протокол, а REST – это архитектура построения API (Application Program Interface). Сравнивать SOAP и REST не имеет смысла, так как у них фундаментальные различия, но можно сравнить производительность и эффективность этих двух технологий.

2.3 SOAP

SOAP, представляет собой протокол для обмена структурированной при использовании такой технологии как Web-службы. WSDL (Web Service Description Language) используется вместе с SOAP, что бы web-службы могли обмениваться сообщения. Без SOAP веб-службы не поддерживают работу с разными операционными системами и платформами и часто не могут взаимодействовать друг с другом. SOAP был разработан, чтобы восполнить этот пробел, облегчая разработчикам создание веб-программ не зависящих от этих ограничений.

WSDL – это формат схемы XML для описания сетевых сервисов как набора конечных точек, оперирующий сообщениями, которые содержат информацию в виде документа. Сообщения описываются абстрактно, а затем привязываются к конкретной сети, протоколу и формату сообщения, чтобы определить конечную точку.

SOAP сообщение это WSDL документ, состоящий из трёх основных элементов: конверт (Envelope), заголовки (Header) и тело (Body). Пример Soap-сообщения представлен в листинге 1.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:t="www.example.com">
  <SOAP-ENV:Header>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <t:someData>
      <someField>Some field data</someField>
    </t:someData>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Листинг 1 – Пример SOAP сообщения

В конвертере устанавливаются основные атрибуты сообщения, версия, определения пространства имён и т.д. В заголовке содержится информация о сообщении необходимая для доставки сообщения и его различные параметры. В теле запроса устанавливаются данные, которые необходимо передать.

В настоящее время SOAP использует HTTP как базовый протокол, но может использовать и другие протоколы для передачи данных. Microsoft была первой, кто начал разработку этого протокола. SOAP использует XML для передачи сообщений через транспортный уровень. Использование WSDL показано на рисунке 7. Клиент SOAP может быть разработан, зная URL-адрес конечной точки сервера и адрес порта.

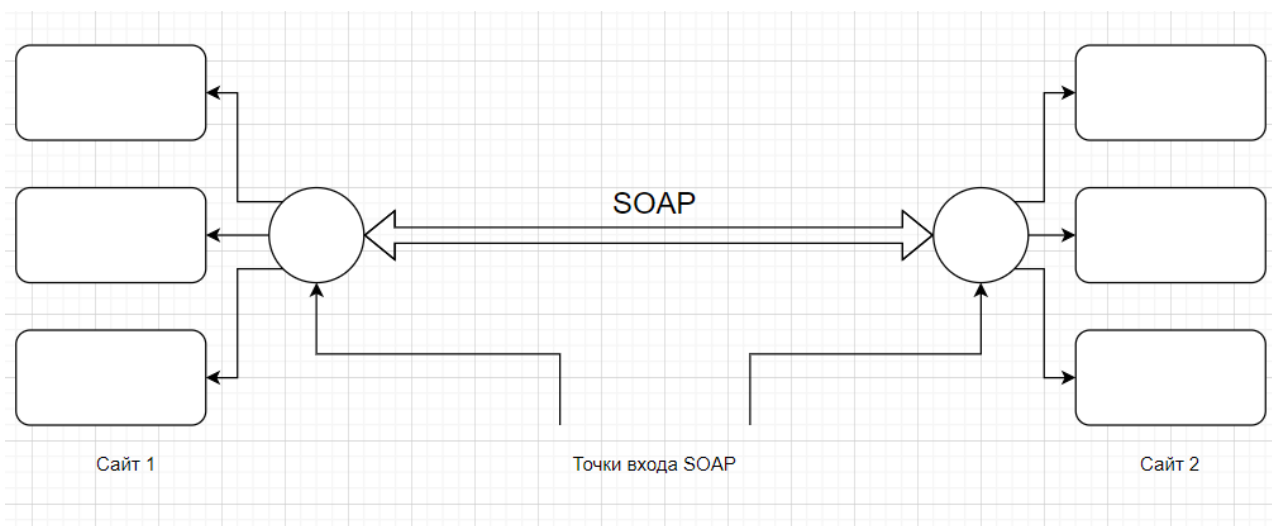


Рисунок 7 – Схематичной представления работы протокола SOAP

Как показано на рисунке 7, программы и приложения на сайте 1 привязаны к WSDL и доступны в Интернет как Web-служба и к ней можно обратиться с использованием SOAP. Сайт 1 подключен к сайту 2 по протоколу SOAP. Рисунок 7 также можно рассматривать как Web-приложение и Web-клиент обмениваются данными друг с другом по протоколу SOAP. Клиент на втором сайте отправляет запросы на сервер/сайт 2 в виде сообщения SOAP. Сообщение обрабатывается на сайте 1 и отправляется ответное сообщение на сайт 2 с помощью SOAP. SOAP сообщение, полученное клиентом на сайте 2, обрабатывается процессором SOAP и делается доступны для локальных программ. Таким образом, SOAP использует уровень приложения как транспортный слой.

2.4 REST

REST (Representational State Transfer) — это архитектура для разработки Web-служб. Основная идея REST заключалась в использовании HTTP для передачи данных между машинами, вместо использования протокола, который работает поверх уровня HTTP для передачи сообщений. Приложение, разработанное в соответствии с принципами REST, будет использовать HTTP для совершения запросов между машинами, а не полагаться на сложные технологии, такие как CORBA (Common Object Request Broker Architecture), RPC (Remote Procedure Call) или SOAP. Поэтому приложения REST используют функции HTTP-запроса для публикации данных, чтения данных и удаления данных, тем самым используя полную функциональность HTTP CRUD (Create Read Update Delete). REST также может работать по HTTPS, обеспечивая безопасную передачу данных.

REST архитектура подразумевает выполнение шести основных правил при разработке приложения.

Клиент-сервер

Первый принцип REST говорит о необходимости разделения пользовательского интерфейса от хранилища данных, таким образом клиентскую часть будет проще переносить на другие платформы, а также улучшается масштабируемость серверной.

Stateless

Каждый запрос к серверу индивидуален, не основывается на каком-либо состоянии со стороны сервера и хранит в себе информацию.

Кэшируемость

Ответ клиенту должен кэшироваться, если это возможно и в случае необходимости ответ может быть переиспользован без повторного запроса к серверу.

Единообразие интерфейса

Этот принцип предполагает, что архитектура приложения станет проще при её унифицированности. Данное ограничение предполагает, что сервер и клиент «разговаривают на одном языке», под этим подразумевается единые пути

обращения для клиент и для сервера. Помимо этого правило единого интерфейса включает в себя ограничение об управлении ресурсами через представление, т.е. клиент никогда не отправляет инструкции по изменению, а всегда оперирует представлением того как он хочет, что бы ресурс выглядел.

Многоуровневая система

Так же этот принцип можно назвать модульностью. Компоненты изолированы своим уровнем приложения и не имеют доступа к остальным, кроме тех с которыми необходимо взаимодействовать.

Код по мере необходимости

REST позволяет увеличивать функционал приложения путём скачивания и использования дополнительных модулей

При архитектуре REST использует четыре основных HTTP метода:

- POST – Для добавления или создания сущности
- PUT – Для обновления или исправления уже существующих данных.
- GET – Для получения существующих данных.
- DELETE – Для удаления существующих данных.

На рисунке 8 представлена модули web-служб архитектуры REST.

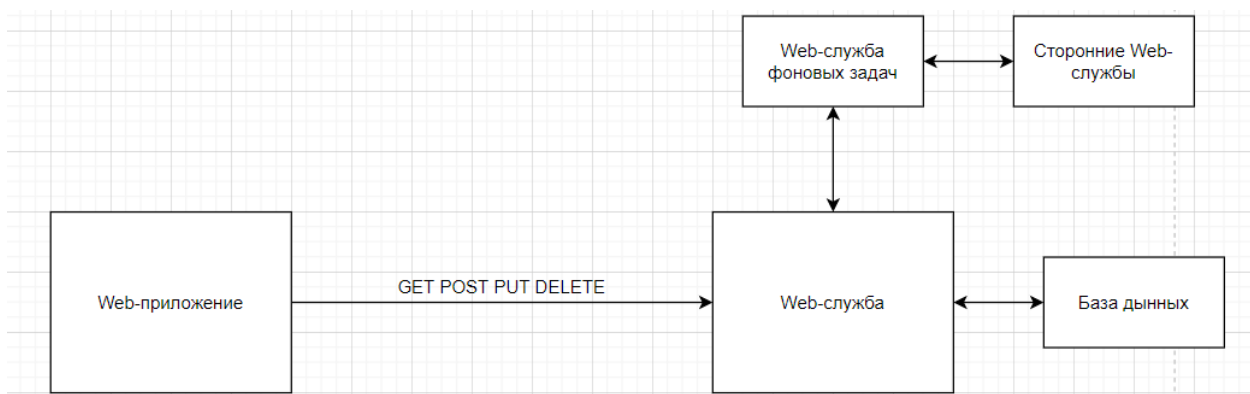


Рисунок 8 – модули web-служб архитектуры REST

Google - один из первых кто начал использовать REST. Карты Google – это служба полностью построен по принципу REST, как и поиск Google. Если начать что-то искать с помощью поисковой системы Google. Страница результатов будет иметь адрес. Адрес второй страницы результатов будет отличается от первой. Этот адрес можно скопировать, сохранить и получить доступ позже. Не будет

необходимости в повторном переходе на web-страницу Google для повторного поиска необходимой информации и нажимать «Далее», чтобы увидеть результаты второй страницы. Это один из примеров web-службы использующей принцип отсутствия состояния. У Web-службы без сохранения состояния есть и другие преимущества, такие как использование серверов с балансировкой нагрузки. Поскольку ни один запрос к Web-службе не зависит от другого, и поскольку каждый запрос клиента несет всю необходимую информацию, нет необходимости согласовывать различные инстансы приложения между собой.

Так как в качестве транспортного протокола в REST используется HTTP и в отличие от SOAP, который предполагает использование только формата передачи данных XML, можно использовать любой удобный формат, но на практике чаще всего используется JSON.

2.5 Сравнение использования SOAP и REST

Понимание REST легче, если есть базовые знания о HTTP, из-за их схожей архитектуры. REST сложен в написании серверной части, но проще в плане клиентской. В SOAP наоборот. Помимо легковесности запросов, неоспоримое преимущество REST – это возможность получать данные от Web-службы в любом формате, а конкретно JSON. Так как большинство Web-приложений написаны с использованием JavaScript, то JSON формат парсить проще, чем XML. Упрощённая схема взаимодействия клиента и сервера REST и SOAP, представлена на рисунке 9.

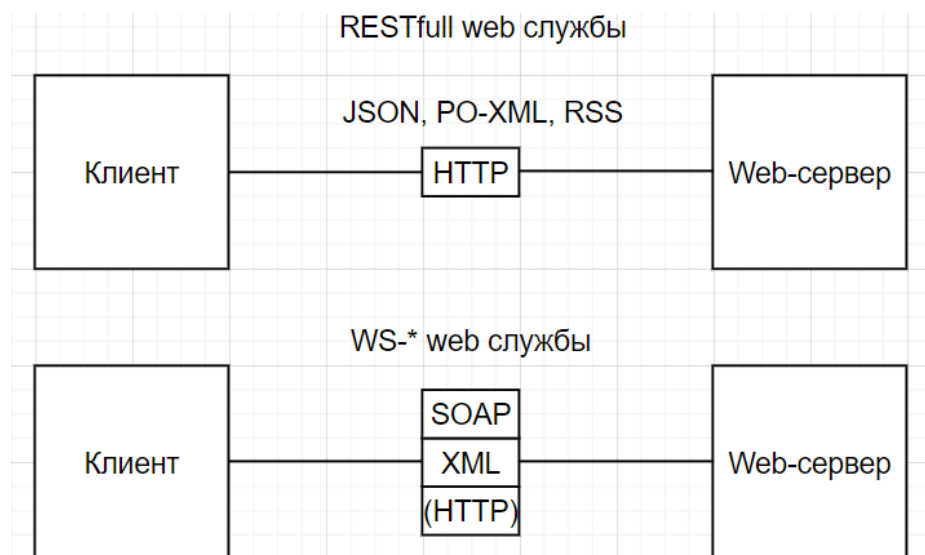


Рисунок 9 – Схема соединения при использовании REST и SOAP

В силу того, что данные при использовании SOAP передаются в XML формате, запросы получаются объёмнее того же JSON при использовании REST. Поэтому время обработки запроса и ответа у REST выше.

SOAP использует HTTP протокол в качестве транспортного уровня, а REST базируется на нём, т.е. при использовании REST становятся доступны все ранние наработки, сделанные на основе этого протокола. Например, серверное кэширование запросов. REST не предполагает использование других протоколов для передачи запросов, в то время как SOAP может применять любой протокол, но по сути эта возможность используется редко и зачастую SOAP так же использует HTTP протокол.

С точки зрения обработки ошибок, REST имеет преимущество перед SOAP, так как в нём используются стандартные коды ошибок HTTP протокола. В таблице 4 представлено сравнение SOAP и REST.

Таблица 4 – Сравнение REST и SOAP

Критерий	SOAP с использованием WSDL	Rest
Протоколы	Любой	HTTP
Транзакции	WS-AtomicTransaction	Нет
Формат данных	XML	JSON, XML, HTML, CSV
Размер передаваемых данных	Относительно большой размер пересылаемых данных	Относительно малый размер пересылаемых данных
Методы	Любые	GET, POST, PUT, DELETE

Как итог можно сделать вывод, что и SOAP и REST имеют свои преимущества и недостатки и их применимость зависит от задачи. В случае использования REST вы получите скорость, масштабируемость, и поддержание большого количества форматов, в случае с SOAP приложение будет больше возможностей для безопасности с использованием WS-security и транзакционная безопасность (ACID).

Для реализации Web-приложения необходимо получать список фильмов, редактировать их, добавлять и удалять, создавать модель аренды фильма отслеживать её и редактировать, в рамках выпускной квалификационной работы нет необходимости в реализации сложной логики транзакций. Предполагается, что в будущем оплата будет происходить через сторонние службы. На основе этого при разработке актуальнее использовать REST архитектуры.

2.6 Создание модели обращения Web-приложения к Web-службе

Для осуществления связи между Web-приложением и Web-службы можно использовать два способа: напрямую и с использованием прокси. Перед началом

создания модели важно определить, что такое модель. Модель – это упрощённое представление, реального объекта, сохраняющее только его важнейшие свойства.

Прокси – это так называемая “прокладка” между web-приложением и Web-службой, через которую осуществляются запросы к службе. На рисунке 10 представлена схема соединения Web-приложения с Web службой через прокси.

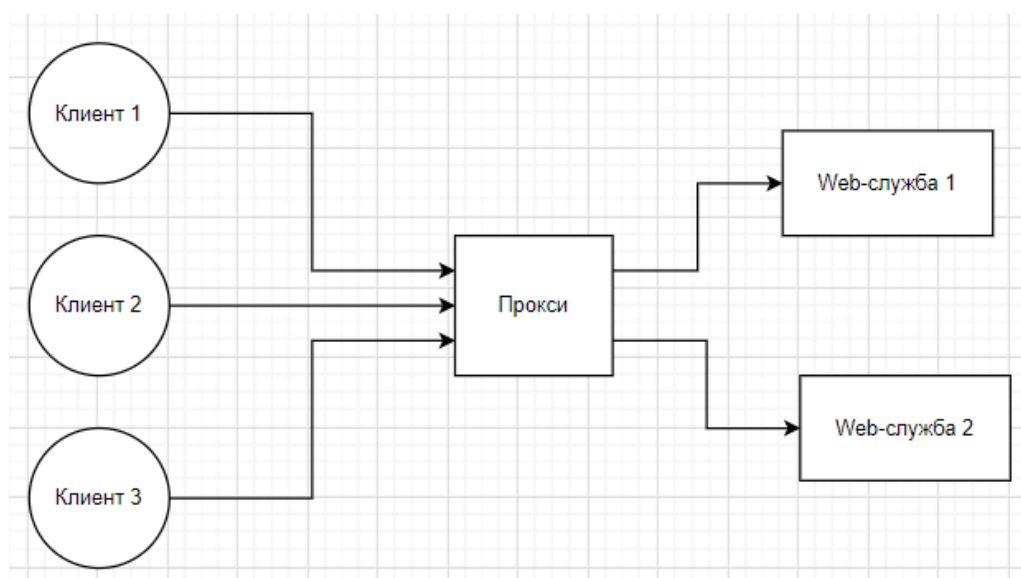


Рисунок 10 – Схема связи клиентов и служб через прокси

Соединение через прокси имеет следующие преимущества:

Скрытие IP

Использование прокси позволяет скрывать IP-адреса, т.е. клиент будет иметь доступ только к прокси и не будет знать о службах используемых на сервер – это повышает безопасность приложения.

Фильтрация запросов к серверу

Прокси позволяет осуществлять блокировку нежелательных запросов от каких-либо ресурсов или лиц.

Кэширование

При повторении одинаковых запросов прокси позволяет отвечать клиентам, кэшированными запросами без повторного обращения к серверу, это увеличивает скорость ответа клиенту и снижает нагрузку на сервер.

Использование нескольких серверов

Прокси может осуществлять маршрутизацию запросов. Это удобно при использовании нескольких служб. С точки зрения клиента запросы будут идти на один и тот же адрес. Помимо этого, маршрутизация позволяет упростить масштабирование и равномерно распределяет нагрузку между несколькими инстансами сервера.

К недостаткам можно отнести возможность кражи личных данных пользователя, если злоумышленник получит доступ к прокси. Во-вторых, это скорость обработки запроса. RTT (Round Trip Time) становится больше и за счет этого время ответа клиенту увеличивается.

Целью моделирования является определить целесообразность использования прокси при создании связи между web-приложением и web-службой. Для проведения исследования необходимо разработать Web-приложение и web-службу, а также провести исследование модели их взаимодействия напрямую и через прокси.

Web-приложение будет демонстрировать основной функционал сервиса по аренде видеофильмов. Отображать список фильмов, для пользователя с правами администратора будет возможность их добавлять, редактировать и удалять, обычный пользователь будет иметь возможность видеть список всех покупателей, иметь возможность редактировать их, удалять, а также создавать нового покупателя, который представляет собой тарифный план подходящий пользователю. В качестве представления действия аренды, пользователя будет заполнять форму аренды, где он сможет выбрать покупателя и фильм доступный для аренды.

Модель взаимодействия Web-приложения с Web-службой представлена на рисунке 11.

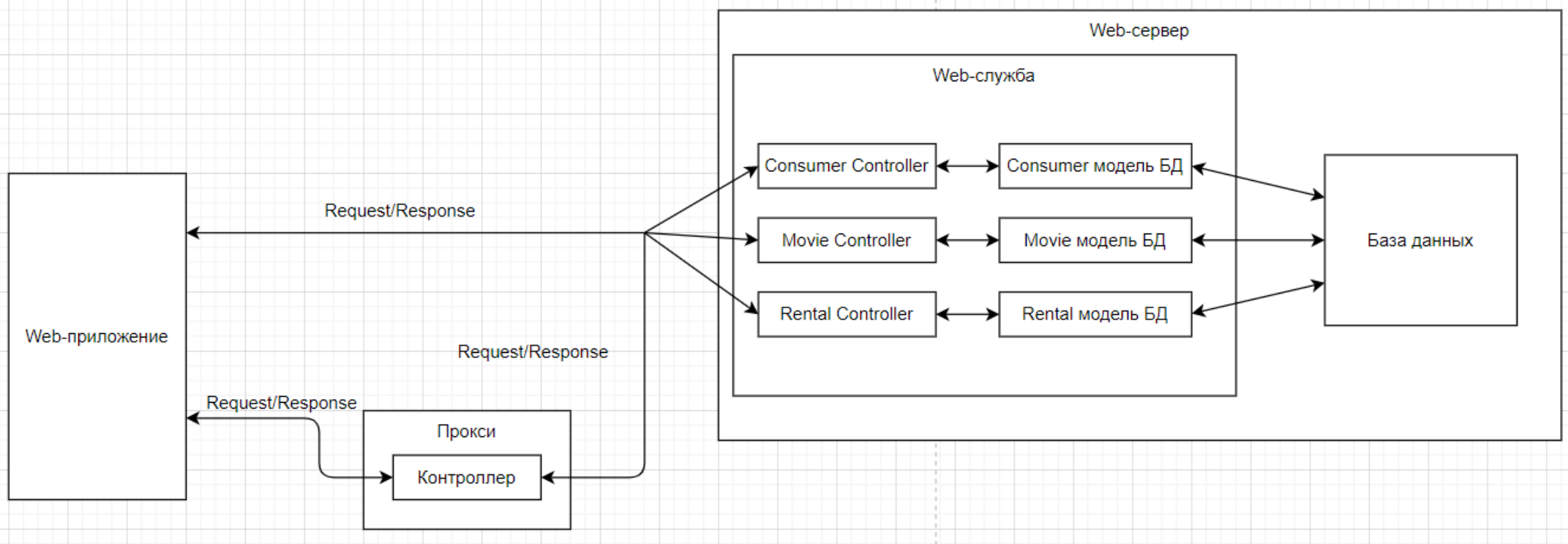


Рисунок 11 – Модель взаимодействия Web-приложения с Web-службой

Функциональностью web-службы, разработанной в выпускной квалификационной работе, является работа с базой данных. Она принимает различные представления через методы GET, POST, PUT и DELETE. Логика Web-службы разделена на три контроллера:

- Контроллер покупателя для действий с моделью из базы данных покупателя, получения списка покупателей, создание нового покупателя, исправления существующего покупателя и удаление.
- Контроллер фильмов для действия с фильмами, получение списка фильмов, создание нового фильма исправление и удаление существующего.
- Контроллер аренды фильмов, предназначен для того чтобы пользователь мог арендовать фильм.

Запрос от web-приложения к web-службе передаётся либо напрямую по адресу расположения web-сервера с указанием конкретных маршрутов, либо через прокси. В случае, когда запрос отправляется через прокси, клиент не обращается напрямую к web-серверу, запрос сначала идёт в контроллер прокси, где определяется к какому контроллеру web-сервера идёт обращение и после этого происходит переадресация на web-сервер.

Для исследования модели взаимодействия будет использована программа Postman, которая позволяет производить запросы к службам, отслеживать данные запроса и его параметры, а также инструменты разработчика браузера, которые так же позволяют отслеживать запросы Web-приложения.

3 РАЗРАБОТКА WEB-ПРИЛОЖНИИ И WEB-СЛУЖБЫ И ИССЛЕДОВАНИЕ МОДЕЛИ ИХ ВЗАИМОДЕЙСТВИЯ.

Во время разработки приложения был выбран MVC паттерн проектирования, в силу того что он наиболее удобен в случае прямого обращения клиента к серверу при использовании ASP.net технологии. Суть этого архитектурного паттерна заключается в передаче модели информации из базы данных через контроллер в виде HTML-страницы определённому пользователю, запросившему эту информацию.

В качестве среды разработки было использовано IDE (Integrated Development Environment) Visual Studio Community, так как она рекомендуется Microsoft, создателями ASP.net. «В Visual Studio можно создавать приложения, которые подключаются к данным практически в любом продукте или службе базы данных, в любом формате, где угодно — на локальном компьютере, в локальной сети или в общедоступном, частном или гибридном облаке [5].»

Web-приложение в выпускной квалификационной работе выполняет роль MVP (Minimum Viable Product) сервиса по аренде видеофильмов. В приложении пользователь может зарегистрироваться или если у него уже есть аккаунт авторизоваться в нём. После успешной авторизации пользователю доступны две основных страницы, список всех доступных фильмов, а также страницу списка всех его покупателей, под покупателями подразумевается, тарифный план по которому пользователь оплачивает фильмы, которые ему понравились, они создаются отдельно от аренды фильма, это позволяет отказаться от общей подписки на всей платформе, а использовать разные тарифы для разных фильмов. На платформе пользователь может произвести действие, которое считается арендой фильма, для это необходимо нажать на «New Rental» в меню, после чего в открывшейся форме начать вводить имя покупателя с нужным тарифным планом, после чего из выпадающего списка выбрать нужного. Аналогичные действия необходимо выполнить с фильмом и нажать кнопку «Submit». Данное действие в рамках MVP считается арендой фильма. Так же было сделано разделение на роли. Существует две роли обычный пользователь и админ. Администратор может

добавлять новые фильмы, заполнив соответствующую форму, а также редактировать существующие. Обычный пользователь выполнить эти действия не может.

3.1 Разработка службы

Для того что бы данными было удобно манипулировать, добавлять, удалять, исправлять и получать, каждый документ должен иметь свою модель в базе данных, в соответствии с которыми над ними будут производиться различные действия. В рамках выпускной квалификационной работы в web-службе были реализованы следующие модели для базы данных:

Покупатель:

Имя	Тип данных
Id	int
Name	nvarchar(255)
IsSubscribedToNewsletter	bit
MembershipTypeId	tinyint
Birthdate	datetime

Рисунок 12 – Модель покупателя из базы данных

Id – идентификационный номер покупателя

Name – имя под которым зарегистрирован пользователь

IsSubscribeToNewsLetter – булево выражение показывающее готов ли пользователь получать рассылку с новостями платформы, которая будет реализована в поздних версиях приложения

MemberShipTypeId – ссылка на модель типа подписки пользователя

Birthdate – дата рождения пользователя

Фильм:

Имя	Тип данных
Id	int
Name	nvarchar(255)
GenreId	tinyint
DateAdded	datetime
ReleaseDate	datetime
NumberInStock	tinyint
NumberAvailable	tinyint

Рисунок 13 – Модель фильма из базы данных

Id – идентификационный номер пользователя

Name – название фильма, которое показывается пользователю

GenreId – ссылка на модель жанров из базы данных

DateAdded – дата добавления фильма на платформу

ReleaseDate – дата релиза фильма

NumberInStock – идентификатор фильма в хранилище

NumberAvailable – количество фильмов доступных для аренды

Жанр:

Имя	Тип данных
Id	tinyint
Name	nvarchar(255)

Рисунок 14 – Модель жанра из базы данных

Id – идентификационный номер жанра.

Name – название жанра, для категоризации и удобной сортировки фильмов

Тип подписки пользователя:

Имя	Тип данных
Id	tinyint
SignUpFee	smallint
DurationInMonths	tinyint
DiscountRate	tinyint
Name	nvarchar(MAX)

Рисунок 15 – Модель типа подписки пользователя из базы данных

Id – идентификационный номер типа подписки
 SignUpFee – месячная стоимость
 DurationInMonth – продолжительность подписки в месяцах
 DiscountRate – коэффициент скидки
 Name – название типа подписки пользователя

Аренда фильма:

Имя	Тип данных
Id	int
DateRented	datetime
DateReturned	datetime
Customer_Id	int
Movie_Id	int

Рисунок 16 – Модель аренды фильма из базы данных

Id – идентификационный номер аренды фильма
 DateRented – дата, когда фильм был арендован
 DateReturned – дата, когда фильм должен был заблокирован для просмотра
 Customer_Id – ссылка на модель пользователя из базы данных, арендовавшего фильм
 Movie_Id – ссылка на модель фильма из базы данных, который был арендован

Пользователь

Имя	Тип данных
Id	nvarchar(128)
Email	nvarchar(256)
EmailConfirmed	bit
PasswordHash	nvarchar(MAX)
SecurityStamp	nvarchar(MAX)
PhoneNumber	nvarchar(MAX)
AccessFailedCount	int
UserName	nvarchar(256)

Рисунок 17 – Модель пользователя из базы данных

Id – идентификатор пользователя
 Email – email пользователя
 EmailConfirmed – булево выражение для определения верифицирован
 пользователь или нет

PasswordHash – пароль пользователя хранимый в базе данных в хэшированном виде

SecurityStamp – штамп безопасности, показывающий изменения авторизационных данных пользователя

PhoneNumber – телефонный номер пользователя

AccessFailedCount – количество неудачных попыток авторизации

UserName – уникальное имя пользователя

Привязка роли к пользователю

Имя	Тип данных
UserId	nvarchar(128)
RoleId	nvarchar(128)

Рисунок 18 – Модель привязки пользователя к роли из базы данных

UserId – ссылка на модель пользователя из базы данных

RoleId – ссылка на роль

Роль

Имя	Тип данных
Id	nvarchar(128)
Name	nvarchar(256)

Рисунок 19 – Модель роли из базы данных

Id – идентификатор пользователя

Name – название роли

Это базовый набор моделей для базы данных и их полей, которые необходимы для реализации первичного функционала службы по аренде видеофильмов. В дальнейшем для полноценного функционирования платформы будут необходимы больше моделей базы данных и дополнение новыми полями существующих.

Для того что бы доставать данные из базы данных, эффективно манипулировать ими без избыточной выборки, было реализовано API следующими путями и контроллерами.

Контроллер покупателя, который включает в себя все действия выполняемые в серверной части приложения, необходимые для манипулирования данными покупателей. Далее представлен список с описание путей по которому web-служба принимает запросы, соответствующие им методы и краткое описание.

Метод GET по адресу «[URL web-службы]/api/customers»

Для получения списка всех пользователей платформы, в качестве query-параметров получает имя пользователя, по которому и осуществляется поиск в базе данных.

Контроллер возвращает поля: id, имя, согласие на получение уведомлений платформы, модель подписки пользователя из базы данных, дату рождения

Метод GET по адресу «[URL web-службы]/api/customers/[:id]»

Для получения конкретного покупателя, в качестве query-параметров получает уникальный идентификатор покупателя.

В контроллере сделана проверка на пустого покупателя.

Контроллер возвращает поля: id, имя, согласие на получение уведомлений платформы, модель подписки покупателя из базы данных, дату рождения.

Метод POST по адресу «[URL web-службы]/api/customers»

Для добавления нового покупателя, в теле запроса передаются поля необходимые для создания покупателя и в контроллере осуществляется проверка на валидность полей и в случае их правильности, вызывается сервис создания документа.

Метод PUT по адресу «[URL web-службы]/api/customers/[:id]»

Для внесения изменения в существующую модель покупателя из базы данных, в теле запроса обязательным полем является id по которому осуществляется поиск в БД. После проверки на валидность передаваемых полей, происходит вызов сервиса изменения модели покупателя.

Метод DELETE по адресу «[URL web-службы]/api/customers/[:id]»

Для удаления существующего покупателя в теле запроса обязательным полем является id по которому осуществляется поиск в БД и в случае, если такой покупатель существует происходит его удаление.

Контроллер фильма, который включает в себя все действия выполняемые в серверной части приложения, необходимые для эффективной манипуляции с фильмами.

Метод GET по адресу «[URL web-службы]/api/movies»

Для получения списка всех фильмов платформы, в качестве query-параметров получает название фильма, по которому и осуществляется поиск в базе данных. В качестве дефолтного значения для query является null, для того что бы в случае пустого запроса возвращался список всех доступных на платформе фильмов.

Контроллер возвращает поля: id, название, ссылку на жанр фильма, дату добавления, дату релиза, идентификатор в хранилище.

Метод GET по адресу «[URL web-службы]/api/movies/[:id]»

Для получения конкретного фильма, в качестве query-параметров получает уникальный идентификатор фильма.

В контроллере сделана проверка на несуществующий фильм и в этом случае возвращается ошибка.

Контроллер возвращает поля: id, название, ссылку на жанр фильма, дату добавления, дату релиза, идентификатор в хранилище

Метод POST по адресу «[URL web-службы]/api/movies»

Для добавления нового фильма, в теле запроса передаются поля необходимые для создания модели фильма и в контроллере осуществляется проверка на валидность полей и в случае их правильности, вызывается сервис создания документа.

Метод PUT по адресу «[URL web-службы]/api/movies/[:id]»

Для внесения изменения в существующую модель фильма из базы данных, в теле запроса обязательным полем является id по которому осуществляется поиск в БД. После проверки на валидность передаваемых полей, происходит вызов сервис изменения модели фильма.

Метод DELETE по адресу «[URL web-службы]/api/movies/[:id]»

Для удаления существующего фильма, в теле запроса обязательным полем является id по которому осуществляется поиск в БД и в случае, если такой фильм существует, происходит его удаление.

Контроллер для аренды фильма

Метод POST по адресу «[URL web-службы]/api/NewRental»

Контроллер для осуществления аренды фильма, который принимает в теле запроса модель аренды фильма из базы данных, после чего осуществляется проверка на существование передаваемого пользователя и фильма. В случае валидности всех полей вызывается сервис создания документа аренды фильма, в противном случае возвращается соответствующая ошибка.

Помимо контроллеров Api были созданы контроллеры отвечающие за управление представления, возвращаемые приложением. Некоторые действия в приложения должны быть доступны только пользователям с определённой ролью, для этого был реализован функционал авторизации при помощи Microsoft.Asp.Net.Identity. В cookie в браузере пользователя хранится программный токен, который помещается туда фреймворком. «Токен — средство идентификации пользователя или отдельного сеанса работы в компьютерных сетях и приложениях [4].»

Контроллер аккаунта, необходим авторизации в приложении, для прохождения регистрации и возвращения соответствующих HTML-страниц:

Метод GET по адресу «[URL web-службы]/Account/Login

Для получения вида формы логина

Метод POST по адресу «[URL web-службы]/Account/Login»

Для отправки и валидации формы логина. В контроллере вызывается SignInManager, который осуществляет проверку полей, отправленных пользователем во время логина и в случае их валидности, устанавливает в cookie браузера.

Метод GET по адресу «[URL web-службы]/Account/Register»

Для получения вида формы регистрации

Метод POST по адресу «[URL web-службы]/Account/Register»

Для отправки и валидации формы регистрации. В этом контроллере создаётся новый пользователь и в случае успеха пользователь автоматически авторизируется.

Метод POST по адресу «[URL web-службы]/Account/LogOff»

Для выхода из учетной записи пользователя

Контроллер покупателя, нужен для возвращения страницы списка пользователей, страницы определённого пользователя и для его редактирования и создания:

Метод GET по адресу «[URL web-службы]/Consumers»

Для получения списка всех покупателей

Метод GET по адресу «[URL web-службы]/Customers/new»

Для получения страницы с формой создания нового покупателей

Метод POST по адресу «[URL web-службы]/Customers/new»

Для отправки формы

Метод GET по адресу «[URL web-службы]/Customers/details/[:id]»

Для получения страницы покупателей с общей информацией

Метод GET по адресу «[URL web-службы]/Customers/edit/[:id]»

Для получения страницы с формой редактирования покупателей

Метод POST по адресу «[URL web-службы]/Customers/edit/[:id]»

Для отправки формы

Контроллер главной страницы, отвечающий за отправку HTML-документов с общей информацией о платформе.

Метод GET по адресу «[URL web-службы]/»

Для получения главной страницы

Метод GET по адресу «[URL web-службы]/About»

Для получения страницы с описанием платформы

Метод GET по адресу «[URL web-службы]/Contact»

Для получения страницы с контактной информацией

Контроллер фильмов для отправки форм и возвращения HTML-страниц

Метод GET по адресу «[URL web-службы]/Movies»

Получения страницы со списком всех фильмов

Метод GET по адресу «[URL web-службы]/Movies/new»

Для получения страницы с формой создания нового фильма

Метод POST по адресу «[URL web-службы]/Movies/new»

Для отправки формы

Метод GET по адресу «[URL web-службы]/Movies/details/[:id]»

Для получения страницы фильма с общей информацией

Метод GET по адресу «[URL web-службы]/Movies/edit/[:id]»

Для получения страницы с формой редактирования фильма

Метод POST по адресу «[URL web-службы]/Movies/edit/[:id]»

Для отправки формы

В данной версии платформы функционал аренды фильма представлен в виде формы из двух полей, покупателя и фильма, для это используется контроллер аренды. В этих полях необходимо выбрать существующего покупателя и фильм, после чего создастся документ.

Метод GET по адресу «[URL web-службы]/Rentals/new»

Для получения формы создания аренды

3.2 Разработка приложения

Для отображения страниц приложения на клиенте были созданы следующие представления:

На главной странице (рис. 20) слева сверху было сделано навигационное меню состоящее из четырех кнопок: для перехода на главную страницу «MovieRent», кнопка создания новой аренды «New Rental», кнопка перехода на страницу покупателей «Customers», кнопка перехода на страницу фильмов «Movies». Помимо этого в правом верхнем углу страницы приложения были сделаны кнопки «Register» и «Log in» для авторизации и регистрации на платформе соответственно. На основной странице имя платформы с кратким описанием. Так же были сделаны страницы описания и контактной информации. Для стилей использовались готовые

шаблоны Bootstrap.

MovieRent New Rental Customers Movies Register Log in

MovieRent

Renting a movie has never been easier. Start with us now

© 2021 - MovieRent

Рисунок 20 – Главная страница

Страницы регистрации и логина (рис. 21)

MovieRent New Rental Customers Movies

Register.

Create a new account.

Email

someEmail@gmail.com

Password

.....

Confirm password

.....

Driving License

21321

Phone

213432

REGISTER

Рисунок 21 – Страница регистрации

На странице логина существует возможность перейти на страницу регистрации.

MovieRent New Rental Customers Movies

Log in.

Use a local account to log in.

Email

nikitamas1998@mail.ru

Password

.....

☐ Remember me?

LOG IN

[Register as a new user](#)

Рисунок 22 – Страница логина

На странице списка всех покупателей, был реализован поиск по именам, кнопки создания, нового пользователя, удаления пользователя с возможность сортировки и по страничному отображению (рис. 23).

MovieRent New Rental Customers Movies Hello weqw@qewe.qwe! Log off

Customers

NEW CUSTOMER

Show 10 entries Search:

Customer	Membership Type	Delete
weqeq	Monthly	Delete

Showing 1 to 1 of 1 entries

PREVIOUS 1 NEXT

Рисунок 23 – Страница списка всех покупателей

Для страницы фильмов была сделана проверка на роль авторизованного пользователя, таким образом вносить изменения в список фильмов может только пользователь с ролью Admin.

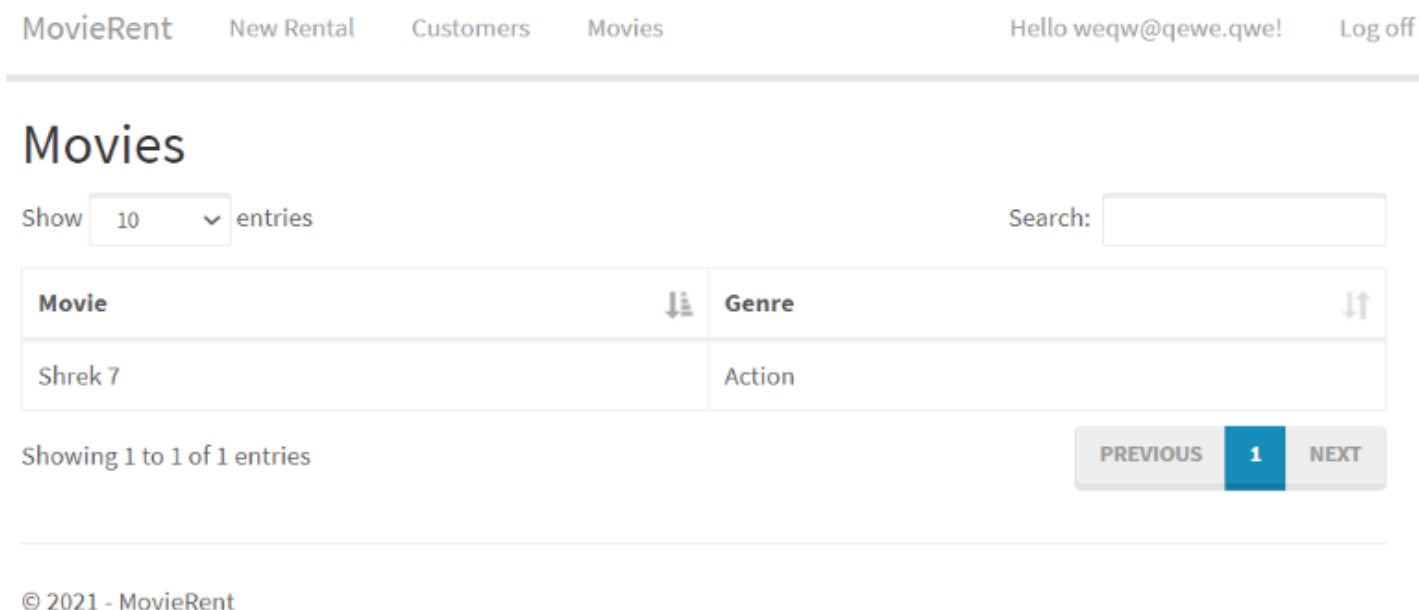


Рисунок 24 – Вид страницы списка всех фильмов для обычного пользователя

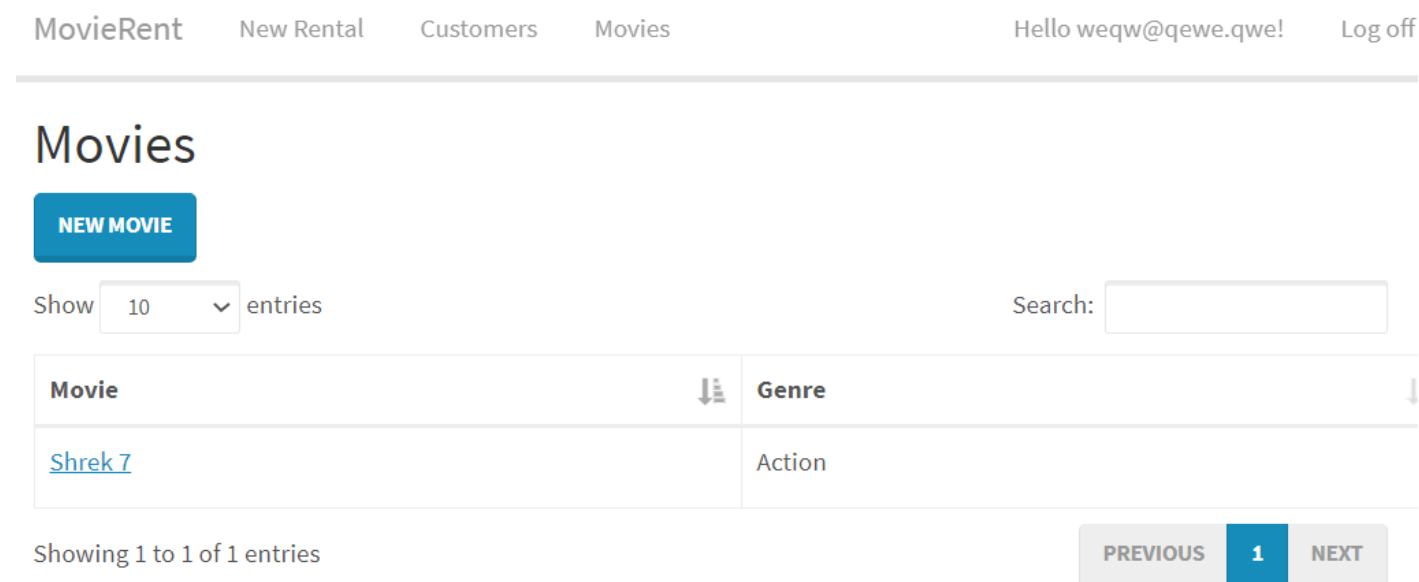


Рисунок 25 – Страница всех фильмов для админа

Была реализована возможность добавление фильмов на клиенте для этого была создана форма. Добавить фильм может только пользователь с правами админа. Ниже представлен процесс добавления фильма.

New Movie

Name

Film

Release Date

01.01.2020

Genre

Action

Number in Stock

10

SAVE

Рисунок 26 – Форма создания нового фильма на платформе

После чего как видно из рисунка 27 в базе данных появилась новая запись

	Id	Name	GenreId	DateAdded	ReleaseDate	NumberInStock	NumberAvaila...
	1	Shrek 7	1	24.12.2020 20:22:59	10.10.2021 0:00:...	2	1
	2	Shrek 7	1	24.12.2020 20:23:23	10.10.2021 0:00:...	2	12
	11	Film	2	24.12.2020 21:47:23	02.11.2020 0:00:...	2	2
	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Рисунок 27 – созданный фильм в базе данных

Movies

NEW MOVIE

Show 10 entries

Search:

Movie	Genre	Delete
Film	Action	Delete
Shrek 7	Action	Delete
Shrek 7	Action	Delete

Showing 1 to 3 of 3 entries

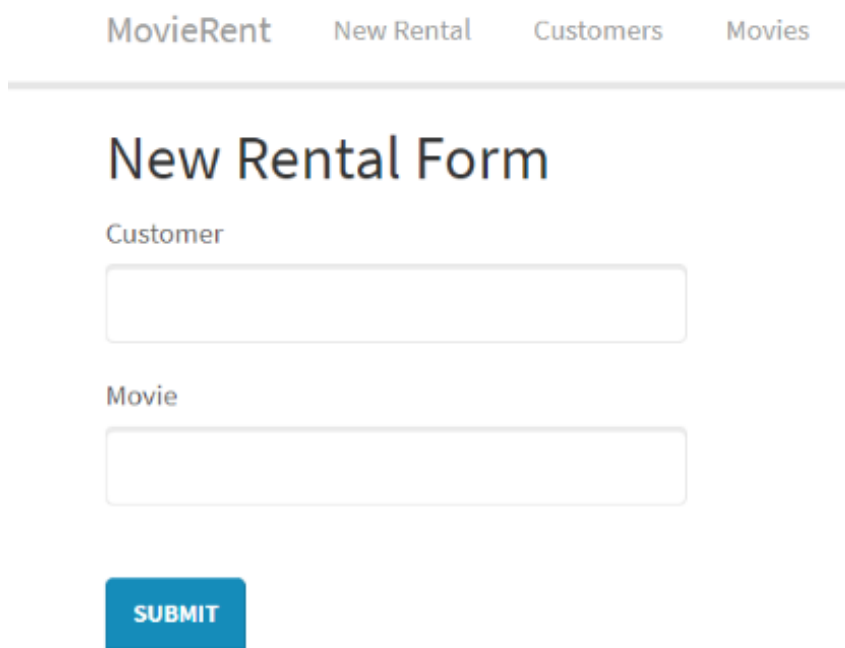
PREVIOUS

1

NEXT

Рисунки 28 – созданный фильм в списке фильмов

Для создания того, что бы пользователь мог арендовать существующий на платформе фильм, была сделана отдельная страница.



MovieRent New Rental Customers Movies

New Rental Form

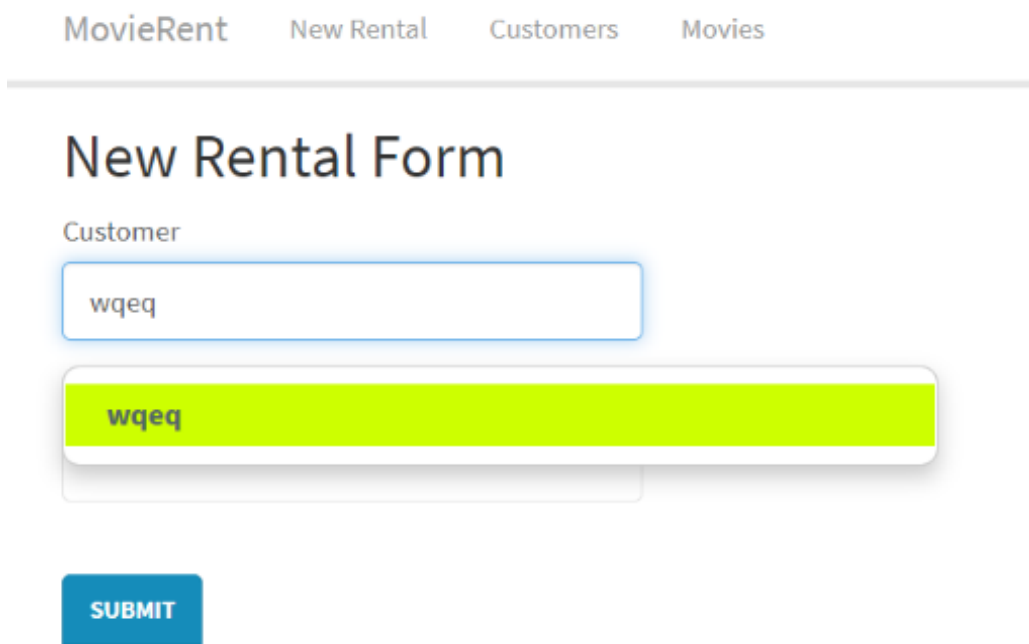
Customer

Movie

SUBMIT

Рисунок 29 – форма аренды фильма

При начале ввода названия фильма или имени пользователя, появляется выпадающий список в соответствии с введённой строкой. Пользователю необходимо выбрать подходящее ему значение.



MovieRent New Rental Customers Movies

New Rental Form

Customer

wqeq

Movie

SUBMIT

Рисунок 30 – выпадающий список

3.3 Реализация прокси

Для реализации прокси сборки было создан отдельный проект в Visual Studio Community и к нему была подключена специальная библиотека для перенаправления запросов к сторонним сервиса Microsoft.Extensions.DependencyInjection.

В настройках проекта были прописаны протокол передачи данных, хост апи, порт и роутинги перенаправления запросов. В качестве API использовался сервис разработанный ранее. Созданное ранее приложение запускается на “<https://localhost:44300/>”, а вновь созданное приложение для реализации прокси сборки приложение на “<https://localhost:44355/>”.

```
app.MapWhen(ctx => ctx.Request.Path.StartsWithSegments("/api/Movies"),
builder => builder.RunProxy(new ProxyOptions
{
    Scheme = "https",
    Host = "localhost",
    Port = "44300"
}));

app.MapWhen(ctx => ctx.Request.Path.StartsWithSegments("/api/Customers"),
builder => builder.RunProxy(new ProxyOptions
{
    Scheme = "https",
    Host = "localhost",
    Port = "44300"
}));

app.MapWhen(ctx => ctx.Request.Path.StartsWithSegments("/api/NewRentals"),
builder => builder.RunProxy(new ProxyOptions
{
    Scheme = "https",
    Host = "localhost",
    Port = "44300"
}));
```

Листинг 2 – Настройки прокси

В результате конфигурации все запросы перенаправляются по прописанным роутингам, перенаправляются к роутингам стороннего API. Для проверки работы прокси сборки использовался менеджер запросов Postman. Работой прокси сборки можно считать успешной, так как при запросе на <https://localhost:44355/api/Movies> запросы был перенаправлен на <https://localhost:44300/api/Movies> и запрашиваемые данные, в данном случае список всех фильмов, были получены. (рис. 31 и рис. 32)

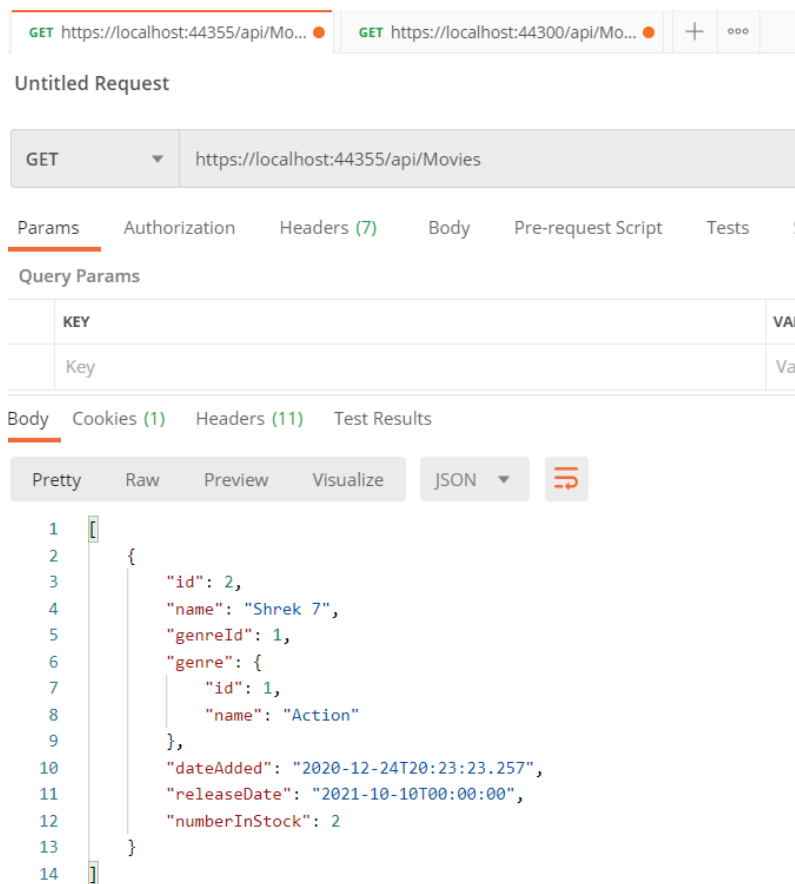


Рисунок 31 – Запрос к API через сборку с использованием прокси

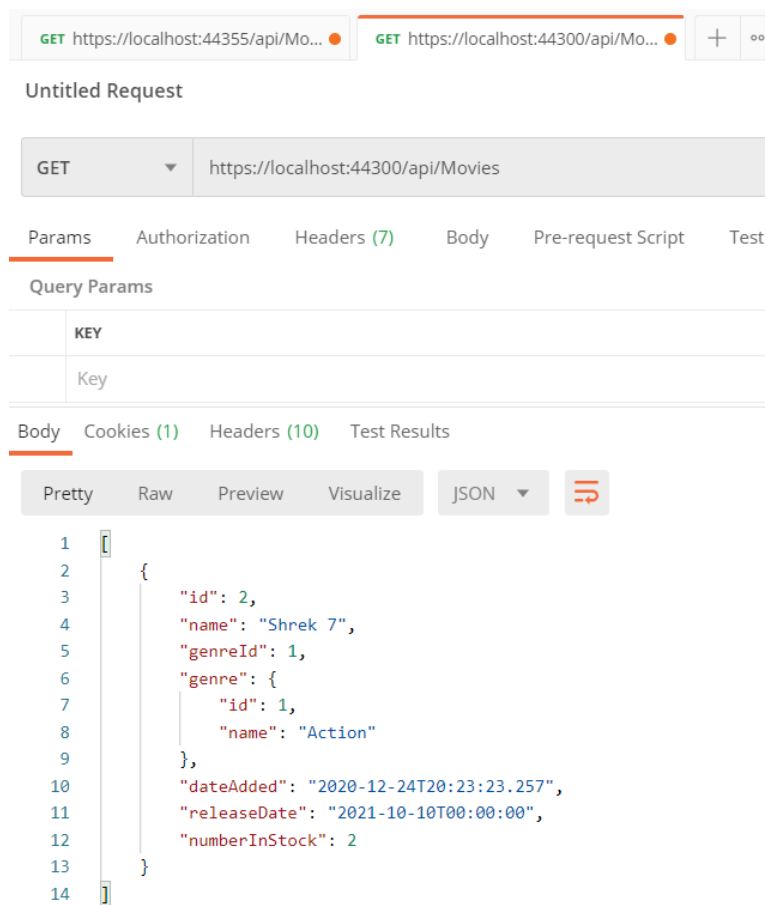


Рисунок 32 – Запрос к API напрямую

В качестве приложения, которое подключается к сервису через прокси сборку, было использовано приложение, разработанное ранее, для этого все URL адреса форм и запросов на получение данные были перенаправлены на прокси.

```
$( "#customers" ).on( "click", ".js-delete", function () {  
    var button = $(this);  
  
    bootbox.confirm("Are you sure you want to delete this customer?", function (result) {  
        if (result) {  
            $.ajax({  
                url: "https://localhost:44355/api/customers/" + button.attr("data-customer-id"),  
                method: "DELETE",  
                success: function () {  
                    table.row(button.parents("tr")).remove().draw();  
                }  
            });  
        }  
    });  
});
```

Рисунок 33 – Замена на адрес прокси в url

В результате такой замены запросы будут перенаправляться на прокси, а не напрямую.

Для того, чтобы API могло принимать запросы со сторонних сервисов в список допустимых адресов, должен быть добавлен адрес прокси. Это было реализовано при помощи пакета System.Web.Http.Cors. При помощи этой утилиты можно разрешить запросы с некоторых или со всех сторонних источников.

В первую очередь необходимо сделать CORS(Cross-origin resource sharing) политику активной в API. «CORS это система, состоящая из отправки HTTP заголовков, которые определяют: заблокировать или выполнить запрос к ограниченному ресурсу на веб-странице из другого домена, отличного от домена происхождения запрашиваемого ресурса[6].»

```

namespace MovieRent
{
    ссылка: 1
    public static class WebApiConfig
    {
        ссылка: 1
        public static void Register(HttpConfiguration config)
        {
            var settings = config.Formatters.JsonFormatter.SerializerSettings;
            settings.ContractResolver = new CamelCasePropertyNamesContractResolver();
            settings.Formatting = Formatting.Indented;
            config.EnableCors();
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}

```

Рисунок 34 – Настройки CORS

После чего в классе каждого контроллера, а данном случае во всех, необходимо прописать разрешённые источники, от которых можно принимать запросы, необходимые заголовки и возможные методы.

```

namespace MovieRent.Controllers.Api
{
    [EnableCors(origins: "*", headers: "*", methods: "*")]
    ссылка: 1
    public class NewRentalsController : ApiController
    {
        private ApplicationDbContext context;
    }
}

```

Рисунок 35 – Определение возможных запросов

В результате этих действий приложение будет получать данные от API не напрямую. Как видно на рисунках 36 и 37, приложение отправило запрос к прокси, та в свою очередь к API и в результате на клиент вернулись необходимые данные.

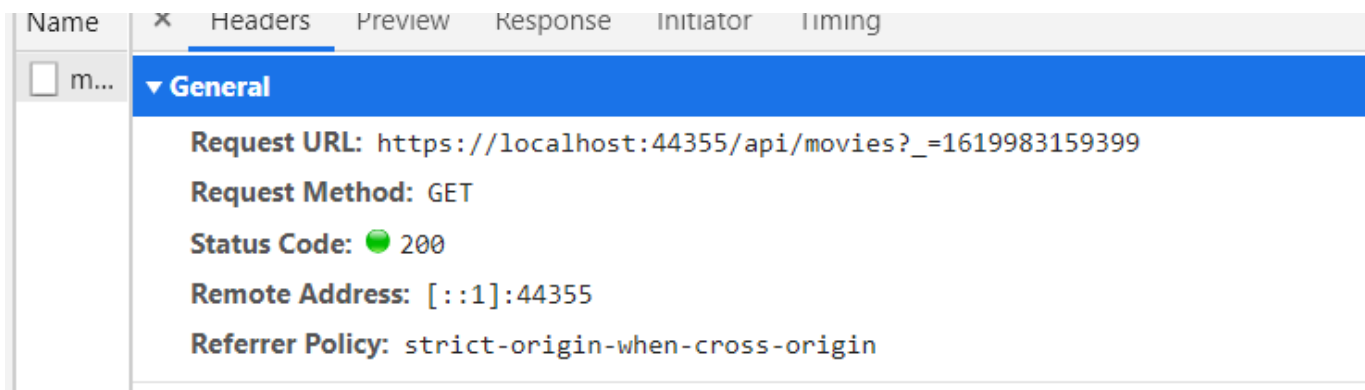


Рисунок 36 – Основная информация о запросе

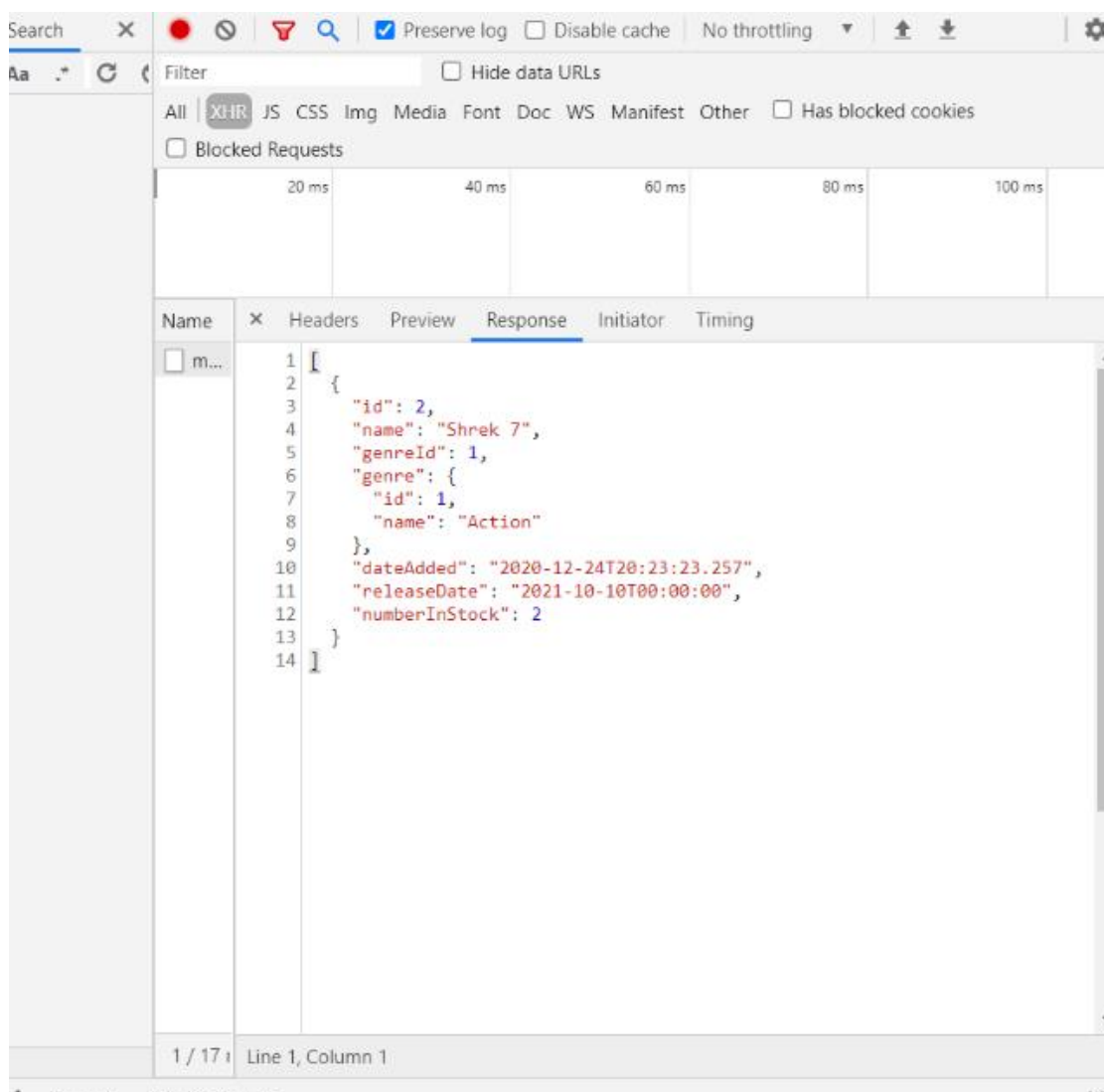


Рисунок 37 – Данные полученные от API

3.4 Исследование модели взаимодействия web-приложения и web-службы

В качестве инструмента исследования в выпускной квалификационной работе будет выступать программное обеспечение «Postman», а также браузерные инструменты разработчика. Данные технологии помогут быстро и эффективно исследовать модель. Postman даёт возможность написания API тестов, помимо совершения обычных запросов. Во вкладке «Tests», можно специфицировать, какие конкретно данные о запросах нужно исследовать.

Проведём тестирование скорости ответа службы при обращении к ней напрямую и через прокси. Для этого необходимо написать тестовые функции. Листинг программы представлена ниже.

```
iterations = 1000;
delay = 100;
responseTimes = [];

i=0;

function sendRequest() {
  pm.sendRequest({
    url: "https://localhost:44300/api/movies",
    method: 'GET'
  }, function (err, res) {
    pm.test("Response time is " + res.responseTime, function () {
      pm.expect(err).to.equal(null);
      pm.expect(res).to.have.property('code', 200);
      responseTimes.push(res.responseTime);
    });
    if (i < iterations - 1) {
      i++;
      setTimeout(sendRequest, delay);
    }
    else {
      averageResponseTime = average(responseTimes);
      pm.test("Average response time is " + averageResponseTime + " ms; the number of iterations is " + iterations);
    }
  });
}

sendRequest();
function average(nums) {
  return nums.reduce((a, b) => (a + b)) / nums.length;
};
```

Листинг 4 – Тестирование ответов службы

Предоставленная функция позволяет отправить тысячу запросов к службе с промежутком в сто миллисекунд и вывести усреднённое время ответа. Для тестирования запросов через прокси была написана такая же функция, но с другим адресом запроса. В связи с тем что запросы происходят локально время ответа очень маленькое, но даже так можно сделать выводы о разнице между запросами напрямую и через прокси. Ниже представлен результат работы

The screenshot displays a REST client interface with the 'Tests' tab selected. The test script is as follows:

```
1 iterations = 1000;  
2 delay = 100;  
3 responseTimes = [];  
4  
5 i=0;  
6  
7 function sendRequest() {  
8   pm.sendRequest({  
9     url: "https://localhost:44300/api/movies",  
10    method: 'GET'  
11  }, function (err, res) {
```

Below the script, the 'Test Results (1001/1001)' tab is active, showing a list of passed tests:

Test Result
PASS Response time is 9
PASS Response time is 13
PASS Response time is 6
PASS Response time is 12
PASS Average response time is 14.845 ms; the number of iterations is 1000

Рисунок 38 – GET запросов напрямую к службе

Params Authorization Headers (7) Body Pre-request Script Tests ●

```
1 iterations = 1000;
2 delay = 100;
3 responseTimes = [];
4
5 i=0;
6
7 function sendRequest() {
8   pm.sendRequest({
9     url: "https://localhost:44355/api/movies",
10    method: 'GET'
11  }, function (err, res) {
12    pm.test("Response time is " + res.responseTime, function () {
13      pm.expect(err).to.equal(null);
14      pm.expect(res).to.have.property('code', 200);
15    });
16  });
17  i++;
18  if (i < iterations) {
19    sendRequest();
20  }
21}
```

Body Cookies (1) Headers (11) Test Results (1001/1001)

All Passed Skipped Failed

PASS

Response time is 21

PASS

Response time is 103

PASS

Response time is 10

PASS

Average response time is 23.048 ms; the number of iterations is 1000

Рисунок 39 - Тестирование GET запросов через прокси к службе

Далее исследуем POST запрос для этого в код теста, добавим заголово типа передаваемых данных, поменяем метод и добавим тело запроса. Изменения функции тестирования приведены в листинге 5.

```
pm.sendRequest({
  url: "https://localhost:44300/api/movies",
  method: 'POST',
  headers: {
    "Content-Type": "application/json"
  },
  body: {
    "name": "Shrek",
    "genreId": 1,
    "genre": {
      "id": 1,
      "name": "Action"
    }
  }
});
```



```
        "dateAdded": "2020-12-24T20:23:23.257",  
        "releaseDate": "2021-10-10T00:00:00",  
        "numberInStock": 2  
    }  
}
```

Листинг 5 – Изменения функции тестирования

На рисунках 40 и 41 редставлены результаты тестирования POST запросов

The screenshot displays a REST client interface with the following components:

- Params**, **Authorization**, **Headers (10)**, **Body** (selected), **Pre-request Script**, and **Tests** (indicated by a green dot).
- Code Editor:** Contains a JavaScript function `sendRequest()` that uses `pm.sendRequest()` to perform a POST request to `https://localhost:44300/api/Customers` with headers `{ "content-Type": "application/json" }` and a body `{ name: "wqeq" }`.
- Test Results (1001/1001)** (indicated by a green dot):
 - Buttons: **All**, **Passed**, **Skipped**, **Failed**.
 - Results list:
 - PASS** Response time is 28
 - PASS** Response time is 93
 - PASS** Response time is 28
 - PASS** Response time is 27
 - PASS** Response time is 27
 - PASS** Average response time is 28.405 ms; the number of iterations is 1000

Рисунок 40 – Тестирование POST запросов напрямую к службе

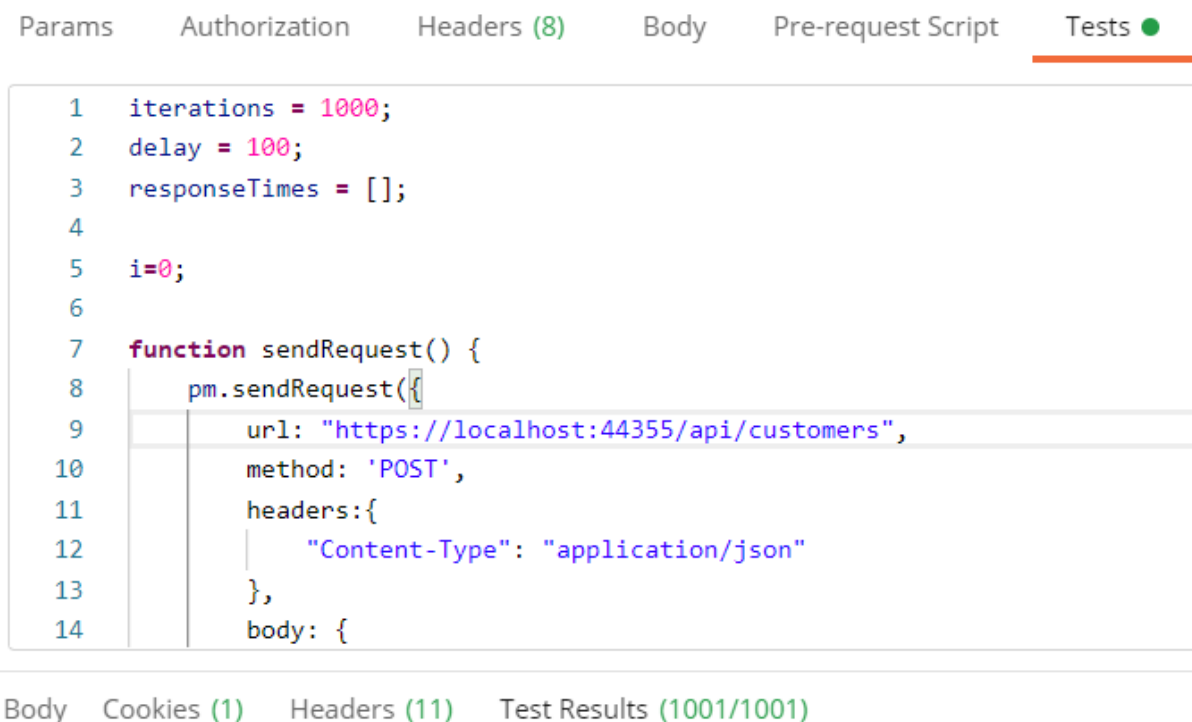


Рисунок 41 – Тестирование POST запросов через прокси к службе

Так же произведём проверку API на безопасность. Только авторизованный пользователь должен иметь возможно получать, изменять и удалять данные. Для того что использовать программный токен для авторизации в запросах, возьмём его из браузера. Для этого необходимо авторизироваться и зайти в инструменты разработчика, нажав F12 на клавиатуре. После чего в верхней части открывшейся панели, перейти во вкладку «Application», далее слева из списка типов хранилища выбрать «Cookie». На рисунке 42 показан программный токен пользователя и где он хранится в браузере.

Manifest	Name	Value	Do...	P...	Exp...	Size	Http...	Sec...	Sa...	Sa...	Pri...
Service Workers	1P_JAR	2021-05-08-14	.gst...	/	20...	19		✓	No...		Me...
Storage	.AspNet.Applicatio...	RcBUYtnu89p-vJ8gjbGSpR26eO...	loc...	/	Ses...	516	✓	✓			Me...
Storage	next-i18next	en	loc...	/	20...	14			Strict		Me...
Local Storage	__RequestVerificati...	XRku1eU1MBrEFhxeMdj_ckrfa8y...	loc...	/	Ses...	134	✓				Me...
Session Storage	_ga	GA1.1.1871714445.1586808233	loc...	/	20...	30					Me...
IndexedDB	vc	6	loc...	/	20...	3					Me...
Web SQL											
Cookies											
https://localhost:44300											

Рисунок 42 – Куки браузера

Неавторизированный пользователь при попытке получить информацию, требующей авторизацию, должен перенаправляться на страницу логина, для подтверждения этого проведём серию запросов при помощи утилиты Postman.

Уберём заголовок передающий куки в запросе и сделаем запрос к службе на получение списка пользователей. Результат представлен на рисунке 43.

GET
https://localhost:44300/api/customers

Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings

Headers
/ hidden

	KEY	VALUE	DESCRIPTION
<input type="checkbox"/>	Cookie	vc=6; _ga=GA1.1.1871714445.1586808233; next-i18next=en; __RequestVerificationToken=XRku1eU1MBrEFhxeMdj_ckrfa8y4	
	Key	mBf2yO7K3wK4ogfp6P6conzohTPU2cWH3QONipgfvHWrLGyE	Description

Body
Cookies (2)
Headers (11)
Test Results

Status: 200 OK
Time: 16 ms

Pretty
Raw
Preview
Visualize
HTML

```

48
49
50      <h2>Log in.</h2>
51      <div class="row">
52          <div class="col-md-8">
53              <section id="loginForm">
54                  <form action="/Account/Login?ReturnUrl=%2Fapi%2Fcustomers" class="form-horizontal" method="post"
55                      role="form"><input name="__RequestVerificationToken" type="hidden"
56                          value="t8uVacSu-8q8_726zI613PFRmJ13Lgyuk9PxjtzTNMaIYk0FknayrnUzBCxiIJ303NhpKSDvEFWnz2Tg11-qrbgP6JvLFzk0V4i
57                      <h4>Use a local account to log in.</h4>
58                      <hr />
59                      <div class="form-group">
60                          <label class="col-md-2 control-label" for="Email">Email</label>
61                          <div class="col-md-10">
                              <input class="form-control" data-val="true" data-val-email="Поле Email не содержит допустимый адрес эл
                                  data-val-required="Требуется поле Email " id="Email" name="Email" type="text" value="" />

```

Рисунок 43 – Неавторизированный запрос

Как видно из тела ответа службы, неавторизованный запрос в ответ получил страницу логина. Если добавить заголовок куки с токеном аторизации в ответ придёт списов всех покупателей. Результат представлен на рисунке 44.

The screenshot displays the Chrome DevTools network panel for a GET request to `https://localhost:44300/api/customers`. The 'Headers' tab is active, showing a 'Cookie' header with a value starting with `vc=6; _ga=GA1.1.1871714445.1586808233; next-i18next=en; _...`. Below the headers, the 'Body' tab shows a JSON response in 'Pretty' format:

```
[
  {
    "id": 1,
    "name": "wqeq",
    "isSubscribedToNewsletter": false,
    "membershipTypeId": 2,
    "membershipType": {
      "id": 2,
      "name": "Monthly"
    },
    "birthdate": "1998-03-01T00:00:00"
  },
  ...
]
```

The status bar at the bottom indicates a successful response with status `200 OK`.

Рисунок 44 – Авторизированный запрос к службе

Запросы к службе через прокси должны вести себя аналогичным образом. Результаты проведения серии аналогичных запросов чреез прокси представлены на рисунках 45 и 46.

GET ▼ https://localhost:44355/api/Customers

Params Authorization **Headers (8)** Body Pre-request Script Tests Settings

Headers 👁 7 hidden

	KEY	VALUE
<input type="checkbox"/>	Cookie	vc=6; _ga=GA1.1.1871714445.1586808233; next-i18next=en; .
	Key	Value

Body **Cookies (1)** Headers (12) Test Results

Pretty Raw Preview Visualize HTML ▼ ↺

```

1 <!DOCTYPE html>
2 <html>
3
4 <head>
5   <meta charset="utf-8" />
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Log in - My ASP.NET Application</title>
8   <link href="/Content/bootstrap-lumen.css" rel="stylesheet" />

```

Рисунок 45 – Не авторизованный запрос через прокси

GET ▼ https://localhost:44355/api/customers

Params Authorization **Headers (8)** Body Pre-request Script Tests Settings

Headers 👁 7 hidden

	KEY	VALUE
<input checked="" type="checkbox"/>	Cookie	vc=6; _ga=GA1.1.1871714445.1586808233; next-i18next=en; .
	Key	Value

Body **Cookies (1)** Headers (12) Test Results 🌐

Pretty Raw Preview Visualize JSON ▼ ↺

```

1 [
2   {
3     "id": 1,
4     "name": "wqeq",
5     "isSubscribedToNewsletter": false,
6     "membershipTypeId": 2,
7     "membershipType": {
8       "id": 2,
9       "name": "Monthly"
10    },
11    "birthdate": "1998-03-01T00:00:00"

```

Рисунок 46 – Авторизованный запрос через прокси

На рисунке 47, представлена информация о запросе к службе, которой обладает пользователь. Как видно из полей главной информации и заголовков запроса, пользователь не видит адреса web-службы, а только адрес прокси, а значит сокрытие адреса службы работает.

▼ General

Request URL: https://localhost:44355/api/movies?_=1621282724949

Request Method: GET

Status Code:  200

Remote Address: [::1]:44355

Referrer Policy: strict-origin-when-cross-origin

▼ Response Headers

access-control-allow-origin: *

cache-control: no-cache

content-length: 247

content-type: application/json; charset=utf-8

date: Mon, 17 May 2021 20:18:44 GMT

expires: -1

pragma: no-cache

server: Microsoft-IIS/10.0

x-aspnet-version: 4.0.30319

x-powered-by: ASP.NET

x-powered-by: ASP.NET

x-sourcefiles: =?UTF-8?B?QzpcVXNlcnNcbmlraXRcT25lRHJpdmVc0KDQsNCx0L7Rh9C40Lkg0YHRgtC+0Ltc0J3QmNCgXE1vdm1lUmVudFxB3ZpZVJlbnRcYXBpXG1vdm1lcw==?=

Рисунок 47 – Информация о запросе

Таблица 5 – Результаты исследования

	Время среднее время 1000 GET запросов	Время среднее время 1000 POST запросов	Авторизация	Соккрытие адреса Web- службы
Напрямую	14.845 ms	23.048 ms	Работает	Нет
Через прокси	28.405 ms	37.927 ms	Работает	Да

Исследование модели показало, что в среднем ответ службы через прокси происходит медленнее, это связано с увеличением RTT(Road trip time) из-за промежуточного узла при передаче запроса. Так как исследование проходило локально это не критично, но если запросы делать на реальных серверах, которые

могу находиться в разных регионах на достаточно больших расстояниях, то разница по времени может быть ощутима. Так же была исследована авторизация, результаты исследования показали, что запросы себя ведут аналогичным образом и при запросе напрямую и через прокси, авторизированный пользователь может выполнять действия с моделями базы данных, а не авторизированный не может, и пересылается на страницу логина. Кроме того изучив информацию доступную пользователю о запросе, можно сделать вывод, что соккрытие адресов работает.

Из всего выше сказанного можно заключить, что внедрение прокси увеличивает время запросов, что может сильно влиять на скорость работы всего приложения целиком. Внедрение прокси должно быть обусловлено преследования определённых целей, например упрощение работы с несколькими службами расположенных на разных серверах, увеличение безопасности, путём добавления дополнительных токенов при персылке запроса через прокси или для снижения нагрузки на сервера путём кэширования пользовательских запросов.

4 СОСТАВЛЕНИЕ БИЗНЕС-ПЛАНА ПО КОММЕРЦИАЛИЗАЦИИ РЕЗУЛЬТАТОВ ВКР

4.1 Описание проекта.

4.1.1 Резюме.

Данный проект по созданию web-приложения по аренде видео фильмов и ведению бизнеса на его основе. Назначение проекта состоит в оказании услуг по демонстрации видео фильмов.

Разработчик проекта: Маслов Н.Д. студент группы 5304.

Цель проекта: обеспечить зрителей наиболее интересными новинками видео фильмов.

Новизна продукции: видео фильмы являются последними и самыми ожидаемыми.

Сведения об объеме продаж: 83 тыс.

Сведения о выручке: 12,45 млн. р.

Сведения о затратах: 6864 млн. р.

Сведения о прибыли: 8103 тыс. р.

Оценка риска: высокий уровень риска данного бизнеса, потому что после карантина ожидается уменьшение доходов пользователей и недостаточный выпуск новых интересных видео фильмов.

Сроки и этапы реализации проекта: в данном бизнес-плане длительность расчета составляет 1 год. Можно выделить следующие этапы: предварительный, подготовительный и коммерческий.

Сумма инвестиций: 3 млн. р.

Структура финансирования: закупка фильмов 95,9 %, покупка сервера 3,3 %, реклама 0,4 %, регистрация 0,3 %.

Для реализации проекта необходима регулярная закупка видео фильмов, потребуется сервер для хранения информации и других нужд, необходима реклама для привлечения клиентов и регистрация бизнеса.

Основные результаты реализации проекта и показатели его эффективности: дисконтированный срок окупаемости 3 квартала, $NPV = 3780$ тыс. р., $IRR = 223 \%$.

4.1.2 Описание продукции.

Продукцией являются видео фильмы, которые арендуют клиенты. Фильм – это совокупность изображений, последовательно размещенных на киноплёнке, которые связаны одним сюжетом и предназначены для воспроизведения на экране. Видеофильм – фильм, записанный на магнитную ленту или оптический диск для последующего показа.

«Разрешение видео – это основная его техническая характеристика. Формат высокой четкости видео фильма начинается с разрешения 1280 на 720, затем идет разрешение 1920 на 1080, 3840 на 2160, 7680 на 4320 [20].»

Уникальность продукта состоит в том, что его уже ждут на рынке. Каждый новый видео фильм уникальный, другого аналога нет.

Инновационность продукции (проекта): у клиентов компании нет нужды искать, выбирать видео фильмы, потому что в web-приложение попадают только лучшие, у пользователя нет мук выбора.

Степень защищенности патентами: при нарушении авторского права на фильм, нарушитель может быть оштрафован. Лицензионное требование затрагивает коммерческие компании. Лицензия на показ выдается за определенный период.

Прогноз цены и затрат на производство: в данном случае отсутствуют данные о продажах, поэтому используется метод экспертных оценок. Этот метод опирается на мнения и оценки экспертов. В основе определения цены заложены цены на рынке, которые представлены в таблице 1. Исходя из сложившихся на рынке цен, принимается нижний уровень цены аренды видео фильмов около 80 руб.

Анализ товаров конкурентов: товары конкурентов являются качественными и предназначены для определенной целевой аудитории. Крупные компании предлагают широкий ассортимент видео фильмов.

4.1.3 Анализ рынка сбыта.

Бизнес, основанный на аренде видео фильмов, является достаточно востребованным. Поскольку такой формат пользования видео фильмами позволяет клиенту сэкономить. В крупных городах подобный арендный бизнес давно получил распространение. Эта услуга пользуется высоким спросом, так как в условиях кризиса, люди еще больше экономят и активнее обращаются к услугам аренды.

Целевой аудиторией аренды видео фильмов является около 30 % пользователей Интернета. Возраст пользователей от 18 до 50 лет. Большую часть аудитории составляют платежеспособные пользователи со средним и выше среднего доходами в возрасте от 25 до 45 лет. Мужчины больше интересуются видео фильмами, чем женщины.

«Рынок аренды видео фильмов в России находится на стадии развития. За период с февраля по сентябрь 2020 г. доля пользователей возросла почти в 2 раза. На 1 пользователя аренды видео фильмов в сентябре приходилось 3 подписки. Совокупные затраты пользователя составили 285 руб. в месяц (рис. 48). По данным опроса, 46 % пользователей на просмотр затрачивают несколько часов в неделю, 19 % – несколько часов в день, 32 % – несколько часов в месяц [21].»

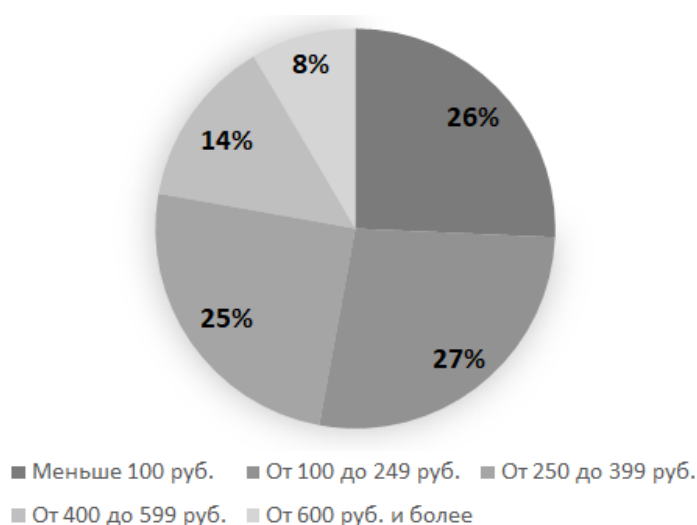


Рисунок 48 – Затраты россиян в месяц на услуги онлайн-видеосервисов

Компания будет предлагать самые ожидаемые видео фильмы. Эти фильмы являются одними из лучших на рынке, потому что их выхода ожидают миллионы зрителей. В этих видео фильмах задействованы наиболее известные звезды кинематографа и у них крупные бюджеты. Когда выходят трейлеры к этим видео фильмам их сразу смотрят миллионы зрителей.

В 2020 г. показатели онлайн кинотеатров увеличились на 66 %, в 2019 г. этот показатель составил 46 %. Это наибольший показатель за все время. Такой значительный рост в этой отрасли связан с пандемией во время которой многие люди были вынуждены дома на карантине. Аудитория онлайн кинотеатров увеличилась на 17 % до 63 млн. человек.

В 2019 году объем рынка онлайн кинотеатров составил 17,1 млрд рублей, более полные данные по объему рынка представлены на рисунке 49.

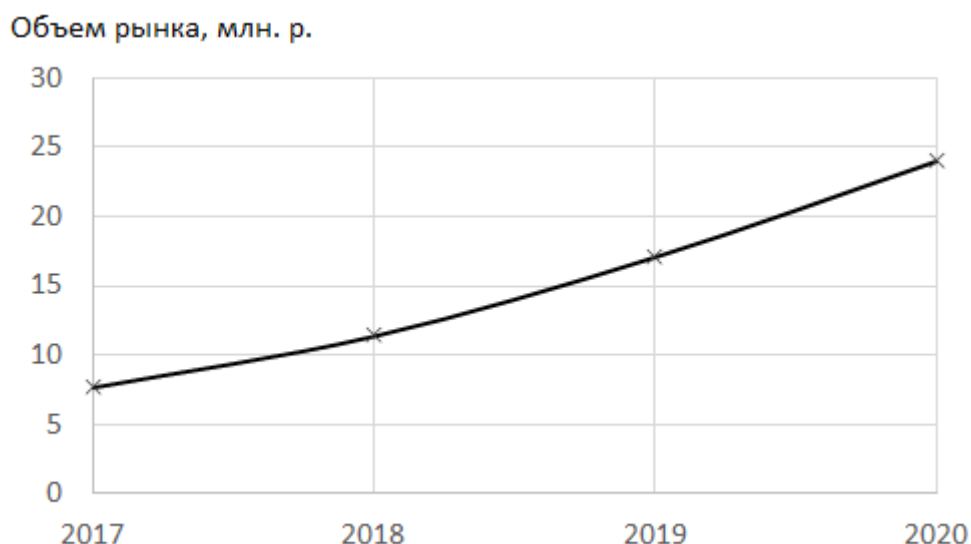


Рисунок 49 – Объем рынка онлайн кинотеатров за период 2017-2020 гг.

Темпы роста рынка онлайн кинотеатров за период 2017-2020 гг. были высокими и устойчивыми. В этих условиях многие участники рынка могли увеличить объемы сбыта. Следовательно, за рассматриваемый период ситуация на рынке была благоприятной.

4.1.4 Анализ конкурентов

Основными конкурентами компании будут онлайн кинотеатры и магазины контента. Самая дешевая подписка у Megogo, которая стоит 197 руб. в месяц, но эта подписка не дает доступа ко всем фильмам. Сервис Megogo предлагает доступ к видео фильмам и телесериалам и программам телеканалов России. Этот онлайн кинотеатр можно смотреть с компьютера и Smart TV.

Второй по доступности подписки кинотеатр – «Okko, в котором подписка стоит 249 руб. [23].» Okko позиционирует себя как премиальный. Этот сервис функционирует на планшетах, смартфонах, Смарт ТВ, игровых приставках PlayStation. В этом онлайн кинотеатре в коллекции имеются фильмы в разрешении 4K и со звуком Dolby 5.1.

Среди существующих на рынке онлайн кинотеатров можно выделить Netflix, который предлагает уникальные видео собственного производства, которые больше нигде не встречаются. Это привлекает клиентов, потому что такой контент можно увидеть только по подписке.

Амедиатека специализируется на предоставлении доступа к сериалам от НВО. В 2017 г. этот онлайн кинотеатр получил статус Home of НВО, что означает выход телепремьер в Российской Федерации одновременно с США. В его каталоге представлены фильмы и сериалы различных студий, например, Sony Pictures, BBC и т.д.

TVzavr располагает большой коллекцией фестивального и авторского кино. Этот контент можно смотреть на смартфонах, компьютерах, Smart TV и планшетах. При просмотре с рекламой видео фильмы можно смотреть бесплатно. Новинки отмечены красным знаком, этот контент можно взять в аренду либо приобрести от «229 руб [24].»

Кинопоиск HD предлагает подписку, оформив которую подписчик получает доступ к Яндекс.Музыке, скидки, дополнительные 10 Гб для хранения файлов на Яндекс.Диске. Кинопоиск HD входит в экосистему Яндекса. Не весь контент онлайн кинотеатра доступен по подписке. Этот сервис предлагает несколько вариантов подписок, самая дешевая подписка –

Плюс (199 руб.), которая позволяет получить доступ к более чем 7,5 тыс. фильмам и сериалам.

Google Play не является полностью онлайн кинотеатром, а в большей мере – сервис аренды. В этом сервисе отсутствует ежемесячная подписка, пользователи могут оплатить фильм, который им нужен. Компьютерная версия Google Play интегрирована с Ютуб. Клиенты сервиса установлены на телевизоры и боксы с Андроид.

Результаты конкурентного анализа представлены в табл. 6.

Таблица 6 – Анализ конкурентов

Конкуренты	Основные преимущества	Основные недостатки	Основные ассортиментные группы	Стоимость
Netflix	Фирменный стиль сайта, отзывчивая техподдержка	Высокая цена подписки, ограниченный выбор фильмов	Огромный выбор фильмов, но сильно урезан для россиян	660 руб.
Google Play	Большой выбор фильмов, доступные цены	Доступен только на планшетах и смартфонах Андроид	Большой размер медиатеки, кино, сериалы и шоу	Нет подписки
TVzavr	Фильмы в высоком качестве, низкая цена	Новые фильмы продаются отдельно	Существует несколько разновидностей кино	От 250 руб.
Амедиатека	Быстро появляются новые фильмы, пробная версия	Напоминания часто глючат, не приходят уведомления о выходе новых серий	Акцент сделан на предоставление доступа к новинкам кинематографа	600 руб.
Собственная продукция	Зрители могут посмотреть новинки кино	Небольшой ассортимент фильмов	Новинки видео фильмов	От 80 руб.

В условиях конкуренции каждая из компаний находит собственные способы для удержания позиций. TVzavr предложил 3 мес. Бесплатной подписки при наличии больничного листа. Об отмене платы за подписку объявили Premier, More.tv и Кинопоиск HD. Бесплатную библиотеку расширили в онлайн кинотеатре Megogo. Okko использует чат-бота Telegram, который предоставляет промокод на 2 недельный пакет подписок после ответа

на несколько вопросов. Окко ввел бесплатную подписку «Когда мы дома», которая предоставляет доступ к спектаклям, кулинарным урокам, виртуальным экскурсиям, трансляциям концертов и тренировкам онлайн.

Компании по аренде фильмов и услугам онлайн кинотеатров находятся в условиях жесткой конкуренции. На рынке можно выделить около пятнадцати крупных компаний, из которых на трое компаний приходится около 50 % продаж. Процесс формирования этого рынка почти закончен, наблюдается начало укрупнений и вытеснений. Крупные онлайн кинотеатры с широким ассортиментом контента и лучшим программным обеспечением почти не оставляют шанса небольшим онлайн кинотеатрам. В этих условиях одним из решающих параметров для эффективной работы остается качество видео и звука, большой выбор контента.

На Западе тенденция захвата рынка основными компаниями была реализована. В США основными компаниями являются Netflix, Disney+, Amazon и Hulu. В России рынок более разнородный, но в ближайшее время должны выделиться лидеры рынка. Эти компании скорее всего будут входить в состав экосистем крупнейших компаний страны.

Целью конкурентов может стать захват лидирующих позиций на рынке аренды видео фильмов.

Анализ сильных и слабых сторон, разрабатываемой продукции: сильными сторонами являются: предлагаемые новинки видео фильмов, предложение лучших на рынке фильмов и собственное приложение. Слабыми сторонами являются: недостаточно широкий ассортимент предлагаемых для аренды фильмов, новые фильмы могут быстро устареть.

4.2 План маркетинга.

4.2.1 План продаж.

План продаж представлен в табл. 7, табл. 8 и табл. 9. Зачастую реклама не сразу начинает работать, что отображено в плане продаж.

Таблица 7 – План продаж в базовом сценарии

Показатели	Квартал				Всего
	I	II	III	IV	
Фильмы					
Ожидаемый объем продаж, ед.	6000	15000	30000	32000	83000
Цена с НДС, р.	150	150	150	150	150
Выручка с НДС, тыс. р.	900000	2250000	4500000	4800000	12450000
Нетто-выручка (без НДС), тыс. р.	720000	1800000	3600000	3840000	9960000
Сумма НДС, тыс. р.	180000	450000	900000	960000	2490000

Таблица 8 – План продаж в пессимистичном сценарии

Показатели	Квартал				Всего
	I	II	III	IV	
Фильмы					
Ожидаемый объем продаж, ед.	1000	8000	18000	22000	49000
Цена с НДС, р.	100	100	100	100	100
Выручка с НДС, тыс. р.	100000	800000	1800000	2200000	4900000
Нетто-выручка (без НДС), тыс. р.	80000	640000	1440000	1760000	3920000
Сумма НДС, тыс. р.	20000	160000	360000	440000	980000

Таблица 9 – План продаж в оптимистичном сценарии

Показатели	Квартал				Всего
	I	II	III	IV	
Фильмы					
Ожидаемый объем продаж, ед.	9000	18000	37000	40000	104000
Цена с НДС, р.	190	190	190	190	100
Выручка с НДС, тыс. р.	1710000	3420000	7030000	7600000	19760000
Нетто-выручка (без НДС), тыс. р.	1368000	2736000	5624000	6080000	15808000
Сумма НДС, тыс. р.	342000	684000	1406000	1520000	3952000

НДС платят все компании, которые становятся налоговыми агентами. Размер ставки НДС зависит от вида операции. При экспорте ставка НДС составляет 0 %, при продажах еды, товаров для детей, журналов и книг, медицинской продукции – 10 %, в остальных случаях ставка НДС составляет 20 %, поэтому для компании НДС составляет 20 %.

4.2.2 Товарная политика.

Комплекс продуктов: аренда видео фильма на заданный пользователем временной промежуток в web-приложении.

Качество: продукт полностью соответствует современным стандартам программного обеспечения и видео фильмов, имеет достаточный уровень безопасности осуществления платежей и интуитивно понятный интерфейс. Соответственно услуги компании «MovieRent» отвечают требованиям потребителей (по сегментам).

Дизайн и товарный знак продукта: будут уточнены в процессе разработки.

Техническое обслуживание: обновление и настройка web-приложения.

Гарантийное обслуживание: компания несет ответственность за недостатки видео фильма в период аренды видео фильма, если они возникли не по вине зрителя.

Полезность услуг: зрители получают новые видео фильмы, удовлетворяется потребность в развлечениях и досуге.

Преимущества перед продукцией конкурентов: преимущества состоят в том, что зрители получают в аренду новинки кино по доступным ценам.

4.2.3 Ценовая политика.

Коммерческая организация использует рыночные методы ценообразования. Используется один из наиболее надежных методов ценообразования – затратный. Такой метод подходит для определения цены на новые фильмы. Этот метод самый простой, цена определяется как затраты плюс дополнительная сумма. Цена определяется на основе затрат, подтвержденных бухгалтерскими документами.

Стоимость аренды видео фильмов низкая, цена от 80 руб. Что соответствует стратегии низких цен, которую можно применить на любой фазе жизненного цикла компании. Эта стратегия соответствует цели проникновения на рынок. Такие низкие цены не соответствуют интересам конкурентов, потому что они дают низкую прибыль. Преимуществом этой стратегии является возможность получения прибыли в долгосрочный период.

Скидки: акция для тех, кто приводит друга, если он купит фильм по их коду, то получит скидку на покупку фильма – 60 %. Также могут быть промокоды на первый видео фильм для рекламных постов в социальных сетях. Что позволит сделать цены ниже, чем у конкурентов, при том, что услуги идентичные.

Условия платежа: онлайн-платеж через PayPal.

Срок и условия кредитования: оплата в кредит не предусмотрена.

4.2.4 Сбытовая политика и мероприятия.

Основной задачей сбытовой политики является определение наиболее выгодных каналов сбыта. Самым подходящим каналом сбыта для компании «MovieRent» является сеть Интернет. Этот канал для нас самый подходящий, потому что мы планируем работать через Интернет. Соответственно сбытовой персонал и доставка не понадобятся.

Планируется запускать рекламу на следующих платформах:

- 1) где бывает молодежь – «ВКонтакте», Instagram;
- 2) люди старшего возраста – «Одноклассники».

Реклама имеет большое значение, потому что от нее зависят результаты работы. Основная задача нашей рекламы, чтобы люди о нас узнали и запомнили. Но рекламу необходимо продолжать, чтобы потенциальные клиенты узнавали о новых видео фильмах и где их можно посмотреть. Нужна постоянная и обновляемая реклама в Интернете, которая будет размещаться в социальных сетях и на сайте.

Продвижением постоянно будет заниматься человек на аутсорсинге, персональные продажи не требуются. На рекламу в социальных сетях предполагается выделять 5000 руб. в месяц. Дополнительно реклама будет осуществляться на сайте. Расходы на создание сайта – 5000 руб. и на продвижение – 3000 руб. в месяц.

4.3 План производства.

Для ведения подобного бизнеса подходит форма ведения бизнеса – общество с ограниченной ответственностью. Основные операции технологического процесса:

1. Настройка web-приложения;
2. Поиск и приобретение фильмов;
3. Загрузка фильмов в приложение;
4. Раскрутка web-приложения;
5. Прием заказов;
6. Модерация сайта, обновление приложения.

Самостоятельно будут выполняться следующие операции: настройка web-приложения, поиск и приобретение фильмов, загрузка фильмов, прием заказов, модерация сайта и обновление приложения. На субконтракт (аутсорсинг) будут переданы: осуществление рекламы и ведение бухгалтерского учета.

Потребуется закупить следующее производственное оборудование:

1. Сервер;
2. Диски хранения данных.

Необходимый объем первоначальных инвестиций составляет 3 млн. рублей. Закупки сырья не потребуются.

Для организации бизнеса потребуется один человек с постоянным наймом и два человека на аутсорсинге. Сотрудник будет заниматься поиском видео фильмов. Человек на аутсорсинге будет заниматься рекламой проекта. Для ведения бухгалтерского учета потребуется бухгалтер на аутсорсинге.

Оплата труда сотрудника составит 40 тыс. руб., зарплата сотрудника с неполной занятостью – 10 тыс. руб., оплата труда бухгалтера – 5 тыс. руб. раз в квартал или около 2 тыс. руб. в месяц. Общая сумма затрат на оплату труда в месяц – 52 тыс. руб. Оплата страховых взносов за постоянного сотрудника – 12 тыс. руб.

Расчет планируемого объема производства представим в таблице 10.

Таблица 10 – План производства

Показатели	Квартал				Всего
	I	II	III	IV	
Количество услуг	6000	15000	30000	32000	83000
Планируемый объем производства, $V_{пр}$	900000	2250000	4500000	4800000	12450000

Планируется оказывать за год работы 83000 услуг. Объем продаж за год 12,45 млн. руб.

«К производственным затратам относятся затраты, которые напрямую связаны с оказанием услуг аренды видео фильмов [16].» В данном случае производственными затратами являются: расходы на заработную плату – 40 тыс. руб. и социальные отчисления – 12 тыс. р., затраты на закупку фильмов – 500 тыс. рублей. Общехозяйственные расходы – расходы на аутсорсинг – 12 тыс. руб. Коммерческие расходы – на рекламу и сайт – 8 тыс. р.

Наибольшие расходы из числа производственных расходов понадобятся на закупку фильмов. Необходимо постоянно пополнять или обновлять подборку видео фильмов. Предполагается закупать 50 видео фильмов в месяц на сумму около 500 тыс. рублей.

Общая сумма производственных затрат может составить 552 тыс. рублей.

Для дальнейших расчетов необходимо знать амортизационные отчисления. Остаточная стоимость представляет собой разность между начальной стоимостью основных фондов и суммой износа (амортизации).

В инвестиционном проекте используется линейная форма начисления амортизации. Величина амортизационных отчислений начисляется только после окончания периода освоения инвестиций. Результаты вычисления амортизационных отчислений по проекту представлены в табл. 11.

Таблица 11 – Амортизационные отчисления по проекту

Группа ОПФ	Номер шага				
Сервер и диски, тыс. д. е.	0	1 квартал	2 кв.	3 кв.	4 кв.
Начальная стоимость, тыс. д. е.	100				
Амортизационные отчисления, тыс. д. е.	0	8,3	8,3	8,3	8,3
Остаточная стоимость, тыс. д. е.		91,7	83,4	75,1	66,8
Итого амортизационных отчислений по проекту, тыс. д. е.	0	8,3	16,6	24,9	33,2

Себестоимость представляет собой совокупность текущих затрат компании на производство и реализацию услуг аренды видео фильмов. Основные статьи калькуляции при расчете себестоимости калькуляционной единицы представлены в табл. 12.

Таблица 12 – Основные статьи калькуляции

Статьи калькуляции	Формулы для расчета	Условные обозначения
1. Сырье и материалы (за вычетом возвратных отходов)	$З_{Mi} = \sum_{j=1}^n G_{ji} \Pi_j \left(1 + \frac{H_{т.з}}{100} \right) - O_v$ <p>Сырье и материалы не используются, поэтому $З_{Mi}=0$</p>	j – индекс вида сырья или материала; G_{ji} – норма расхода j -го материала на единицу i -й продукции; Π_j – цена приобретения единицы j -го материала, р./ед.; $H_{т.з}$ – норма транспортно-заготовительных расходов; O_v – возвратные отходы, которые определяются как $S_{omxi} \cdot \Pi_{omxi}$; S_{omxi} – норма возвратных (реализуемых) отходов, нат. ед.; Π_{omxi} – цена отходов, р./ед.
2. Покупные комплектующие изделия и полуфабрикаты	$З_{ni} = \sum_{j=1}^n N_{ji} \Pi_j \left(1 + \frac{H_{т.з}}{100} \right)$ <p>Покупные комплектующие и полуфабрикаты не потребуются, для аренды используются фильмы $З_{ni}=6000/12450=0,482$</p>	N_{ji} – норма расходов j -го комплектующего изделия или полуфабриката; Π_j – цена единицы комплектующего изделия или полуфабриката, р./шт.; n – количество видов комплектующих изделий, входящих в единицу i -й продукции

Продолжение табл. 12

Статьи калькуляции	Формулы для расчета	Условные обозначения
3. Основная зарплата производственных рабочих	$З_{оснi} = t_i P_{cp\ i} \left(1 + \frac{H_{пр}}{100} \right)$ <p>В данном случае система и уровень основной зарплаты устанавливаются предпринимателем = 480 тыс. р. $Z_{оснi}=480/12450=0,038$</p>	t_i – трудоемкость изготовления i -го изделия, н.-ч; $P_{срi}$ – средняя расценка по i -й операции, которая определяется с учетом сложности и характера операции (часовая тарифная ставка), р./н.-ч; $H_{пр}$ – процент премии, выплачиваемый по действующей премиальной системе
4. Дополнительная зарплата	$З_{допi} = З_{оснi} \frac{H_{доп}}{100}$ <p>Дополнительная зарплата не предусмотрена, $З_{допi}=0$</p> $H_{доп} = \frac{\Phi_{допi}}{\Phi_{осн}} 100 \%$	$H_{доп}$ – процент дополнительной заработной платы, определяемый в целом по организации (предприятию); $\Phi_{допi}$ – годовой фонд дополнительной заработной платы, р.; $\Phi_{оснi}$ – годовой фонд основной заработной платы, р.
5. Отчисления на социальные нужды	$З_{соцi} = (З_{оснi} + З_{допi}) \frac{H_{соц}}{100}$ $З_{соцi} = (0,038+0)30/100=0,011$	$H_{соц}$ – норма отчислений на социальные нужды (тариф страховых взносов), %
6. Расходы на содержание и эксплуатацию оборудования	$З_{э.о} = t_m S_{м.ч}$ <p>Оборудование находится на гарантии, расходы на содержание и эксплуатацию не предусмотрены. $З_{э.о}=0$</p>	t_m – средние затраты машинного времени по оборудованию на единицу i -й продукции, маш.-ч.; $S_{м.ч}$ – средняя стоимость машиночаса работы оборудования, р./маш.-ч
Итого сумма прямых затрат	$З_{прi} = З_{ми} + З_{пи} + З_{оснi} + З_{допi} +$ $+ З_{с.нi} + З_{э.и}$ $З_{прi}=0+0,482+0,038+0+0,011+0=0,531$	$З_{прi}$ – сумма прямых (переменных) затрат на i -е изделие, р.

Продолжение табл. 12

Статьи калькуляции	Формулы для расчета	Условные обозначения
7. Общепроизводственные расходы (цеховые)	$З'_{o.прі} = (З_{осні} + З_{допі}) \frac{H'_{o.пр}}{100};$ $H'_{o.пр} = \frac{S_{o.пр}}{\Phi_{осн} + \Phi_{доп}} \cdot 100\%$	$H_{o.пр}$, $H'_{o.пр}$ – общепроизводственные расходы, %; $S_{o.пр}$ – годовая смета общепроизводственных расходов организации, тыс. р.;
<i>Итого цеховая себестоимость</i>	$C_{цi} = З_{прі} + З_{o.прі}$ $C_{цi}=0,531$	–
8. Общехозяйственные расходы (общезаводские)	$З'_{o.xi} = (З_{оснi} + З_{допi}) \frac{H'_{o.x}}{100};$ $H'_{o.x} = \frac{S_{o.x}}{\Phi_{осн} + \Phi_{доп}} \cdot 100\%$ $З'_{o.xi}=144/12450=0,011$	$H_{o.x}$, $H'_{o.x}$ – общехозяйственные расходы, %; $S_{o.пр}$ – годовая смета общехозяйственных расходов организации, тыс. р.;
<i>Итого производственная себестоимость</i>	$C_{прі} = C_{цi} + З_{o.xi}$ $C_{прі}=0,531+0,011=0,542$	–
9. Коммерческие расходы	$З_{комі} = C_{прі} \frac{H_{ком}}{100};$ $H_{ком} = \frac{S_{ком}}{V_{т.пр.с}} 100 \%$ $З_{комі}=96/12450=0,008$	$H_{ком}$ – коммерческие расходы, %; $S_{ком}$ – годовая смета коммерческих расходов организации, тыс. р.; $V_{т.пр.с}$ – годовой объем товарной продукции организации, рассчитанный по производственной себестоимости, тыс. р.
Всего полная себестоимость	$C_{пi} = C_{прі} + З_{комі}$ $C_{пi}=0,542+0,008=0,55$	–

Полная себестоимость составляет 0,55 руб. за 1 услугу аренды видео фильмов.

4.4 Финансовый план

Величина финансирования составит 3 млн. рублей. Эти вложения распределяются на следующие направления:

- 1) Уставный фонд – 10000 руб.;
- 2) Реклама и создание сайта – 13000 руб.;
- 3) Сервер и диски для хранения данных – 100000 руб.
- 4) Расходы на фильмы и прочее – остальные средства.

Наибольшая сумма расходов на видео фильмы.

Для открытия бизнеса потребуется кредит на сумму 3 млн. рублей. В качестве кредитора выбран банк Тинькофф, процент по кредиту – 12 %.

Расчет дисконтированных денежных потоков представлен в табл. 13. Шаг расчета обычно соответствует какому-то календарному периоду, в данном случае кварталу. Определения денежной единицы проекта (как правило используется российский рубль).

Значения выручки от реализации заносятся из табл. 13. Значения производственных затрат заносятся из раздела 3.

Налог на прибыль является обязательной статьей расходов для фирм, которые работают в Российской Федерации на общей системе налогообложения. Этот налог платят только юридические лица. Налог на прибыль исчисляется из налогооблагаемой прибыли. Если компания только начинает работать, то прибыли не получает и налог не начисляется.

В налоговой базе учитываются доходы, которые относятся к следующим видам:

1. Доходы от выручки реализации услуг и товаров;
2. Доходы от реализации имущественных прав;
3. Внереализационные доходы.

Расходами компаний считаются только документально подтвержденные и обоснованные затраты. Не все доходы и расходы учитываются при определении налоговой базы, кредитные деньги не являются доходами.

Деньги, которые пойдут на погашение долга по кредиту тоже, не считаются расходами.

Стандартная ставка налога на прибыль составляет 20 % от чистой прибыли компании. Налоговая база вычисляется как разница между доходами и расходами фирмы, которые учитываются при налогообложении. Например, налоговая база в первом квартале:

$$900 - 572 - 8,3 = 319,7 \text{ тыс. руб.}$$

Определяем налог на прибыль:

$$319,7 \times 20 \% = 63,94 \text{ тыс. руб.}$$

При выборе ставки дисконтирования в качестве ориентира могут использоваться различные показатели. В экономической литературе предлагаются макроэкономические индикаторы, которые дают возможность оценить размер ставки процента. Это могут быть ставка рефинансирования, которую устанавливает Банк России либо ставка доходности по быстроликвидным и надежным ГКО.

На уровне определенной компании могут использоваться различные ориентиры при выборе ставки, которая в общем случае может находиться в зависимости от следующих факторов:

1. Цели инвестирования;
2. Уровня инфляции;
3. Величины рисков для бизнеса;
4. Альтернативных возможностей вложений финансовых средств.

Показатель рентабельности инвестиций определяется как отношение среднегодовой прибыли к суммарным инвестиционным затратам в проект:

$$ROI = 8103/3000 = 2,7$$

Получено высокое значение ROI, при таком значении инвестиции окупаются.

Таблица 13 – расчет дисконтированных денежных потоков, тыс. р

Показатели	Номер шага				
	0	1	2	3	4
Операционная деятельность (ОД)					
1. Выручка от реализации	0	900	2250	4500	4800
2. Производственные затраты	0	572	572	572	572
3. Амортизация	0	8,3	8,3	8,3	8,3
4. Налогооблагаемая прибыль	0	319,7	1669,7	3919,7	4219,7
5. Налог на прибыль	0	63,94	333,94	783,94	843,94
6. Чистая прибыль	0	255,76	1335,76	3135,76	3375,76
7. Денежный поток от ОД	0	264,06	1344,06	3144,06	3384,06
Инвестиционная деятельность (ИД)					
8. Инвестиции	3000	0	0	0	0
9. Ликвидационная стоимость		91,7	83,4	75,1	66,8
10. Денежный поток (ДП) от ИД	-3000	91,7	83,4	75,1	66,8
11. ДП проекта	-3000	355,76	1427,46	3219,16	3450,86
12. ДП накопленным итогом (ЧД)	-3000	-2644,24	-1216,78	2002,38	5453,24
13. Коэффициент дисконтирования	1	0,9524	0,9070	0,8638	0,8227
14. Дисконтированный ДП (ДДП)	-3000	338,82	1294,75	2780,83	2839,03
15. ДДП накопленным итогом (ЧДД)	-3000	-2661,18	-1366,43	1414,40	4253,43

Чистая текущая стоимость проекта (NPV – Net Present Value) рассчитывается как разность дисконтированных денежных потоков поступлений и платежей, производимых в процессе реализации проекта за весь инвестиционный период:

$$NPV = \sum_{t=0}^T \frac{CIF_t}{(1+R)^t} - \sum_{t=0}^T \frac{COF_t}{(1+R)^t},$$

где CIF_t – поступление денежных средств, связанных с реализацией проекта, в интервале t (притоки денежных средств);

COF_t – платежи денежных средств, связанных с реализацией проекта, в интервале t (оттоки денежных средств);

R – ставка дисконтирования принятая для оценки анализируемого проекта;

T – время реализации проекта, вычисляемое как количество интервалов инвестиционного периода, т. е. количество процентных периодов, по окончании которых производится начисление процентов. Если в качестве процентного периода принимается год, то T определяет количество лет, в течение которых реализуется проект (жизненный цикл проекта).

В частности, если инвестиции в проект производятся единовременно, то чистая текущая стоимость проекта может быть рассчитана следующим образом:

$$NPV = \sum_{t=1}^T \frac{CIF_t}{(1+R)^t} - I_0,$$

где I_0 – единовременные затраты, совершаемые в инвестиционном интервале; $NCF_t = (CIF_t - COF_t)$ – чистый денежный поток.

Определим чистую текущую стоимость при $R=0,15$:

$$NPV_{0,15} = -3000 + \frac{8136,2}{1,15} = 4074,9 \text{ тыс. р.}$$

Определим чистую текущую стоимость при $R=0,2$:

$$NPV_{0,2} = -3000 + \frac{8136,2}{0,2} = 3780,2 \text{ тыс. р.}$$

Определим чистую текущую стоимость при $R=0,25$:

$$NPV_{0,25} = -3000 + \frac{8136,2}{0,25} = 3508,9 \text{ тыс. р.}$$

Полученные положительные значения NPV свидетельствуют о целесообразности принятия решения о финансировании и реализации проекта.

Показатель внутренней нормы рентабельности проекта (IRR – Internal Rate of Return) определяет такую ставку дисконта, при которой дисконтированная стоимость поступлений денежных средств по проекту равна дисконтированной стоимости платежей:

$$\sum_{t=0}^T \frac{CIF_t}{(1 + IRR)^t} = \sum_{t=0}^T \frac{COF_t}{(1 + IRR)^t},$$

где IRR – искомая ставка внутренней рентабельности проекта.

Внутреннюю норму доходности можно рассчитать в электронных таблицах с помощью функции ЧИСТВНДОХ.

$IRR = \text{ЧИСТВНДОХ}(B1:B5; A1:A5) = 223\%$, где B1:B5 – ряд поступлений денежного потока, A1:A5 – даты платежей.

При внутренней норме доходности проекта равной 223 % использование заемного капитала привлеченного по более высокой ставке является нерентабельным.

Дисконтированный период окупаемости инвестиций ($T_{ок}$) (срок возврата) определяет промежуток времени от момента первоначального вложения капитала в проект до момента, когда нарастающий итог суммарного чистого дисконтированного дохода становится равным нулю. Для определения периода возврата можно воспользоваться данными прогноза движения денежных средств и установить инвестиционный интервал, после которого показатель, определяемый как нарастающий итог чистого дисконтированного денежного потока, становится положительной величиной. Этот интервал и определяет срок окупаемости инвестиций. Очевидно, что чем меньше период возврата инвестиций, тем более экономически привлекательным является проект.

Дисконтированный срок окупаемости определяем на основе данных таблицы 8, проект предположительно окупится за 3 квартала.

ЗАКЛЮЧЕНИЕ

В выпускной квалификационной работе было проведено исследование модели взаимодействия web-службы и web-приложения, в рамках этого исследования были выполнены следующие задачи:

- Были изучены теоретические аспекты разработки web-приложений с использованием технологии ASP.NET.
- Была разработана модель взаимодействия Web-службы и Web-приложения.
- Были разработаны Web-приложение и Web-службу и проведено исследование модели их взаимодействия.

В результате проведённых теоретического исследования и сравнительного анализа, была разработана модель взаимодействия web-службы и web-приложения. Данная модель схематично показывает, как происходит запрос/ответ между приложением и службой напрямую и через прокси. Для проверки работоспособности созданной теоретической модели и исследования проведения запросов напрямую и через прокси, в рамках выпускной квалификационной работы были разработаны web-приложение и web-служба сервиса, который предоставляет услуги по аренде видеофильмов. После чего было проведено исследования модели, реализованной на практике, при помощи программного обеспечения «Postman».

Исследование заключалось в проведении серии POST и GET запросов напрямую и через прокси. В утилите Postman, была написана функция для тестирования, которая отправляет тысячу запросов с промежутком в сто миллисекунд. Сначала были отправлены запросы GET напрямую к службе и было вычислено среднее время ответа, после чего была выполнена серия POST запросов на добавление новых покупателей и вычислено среднее время ответа. Аналогичные действия были сделаны через прокси. Так же была проверена работа авторизации и было выявлено, что она работает как при запросе напрямую к службе, так и через прокси.

Исходя из проведённого исследования модели можно сделать вывод, что применение прокси увеличивает время ответа службы, авторизация работает аналогичным также как и при запросе напрямую, сокрытие адреса обращения от пользователя работает. Прокси необходимо использовать для реализации какого-либо функционала, например для увеличения безопасности приложения, упрощением взаимодействия между несколькими службами или снижения нагрузки на сервер, путём кэширования запросов пользователя в прокси, в противном случае внедрение прокси может замедлить время ответов сервера и как следствие замедлить работу всего приложения в целом.

Помимо этого, в четвертой главе выпускной квалификационной работы был составлен бизнес-план сервиса, предоставляющего услуги по аренде видеофильмов.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Троелсон Э. С# и платформа .Net / пер. с английского ЗАО Издательский дом «Питер» 2004г. 782 с.: ил. (дата обращения – 12.01.2021)
2. Троелсон Э., Джепкинск Ф. Язык программирования С# и платформы .Net и .Net Core / пер. с английского ООО «Диалектика» 2018г. 1316 с.: ил. (дата обращения – 15.01.2021)
3. ASP.NET MVC [Электронный ресурс] <https://habr.com/ru/post/175999/> (дата обращения – 15.01.2021)
4. Определение web-сервера [Электронный ресурс] <https://encyclopedia.kaspersky.ru/> (дата обращения – 20.02.2021)
5. Visual Studio [Электронный ресурс] <https://docs.microsoft.com/ru-ru/visualstudio/data-tools/accessing-data-in-visual-studio?view=vs-2019> (дата обращения – 21.02.2021)
6. CORS [Электронный ресурс] <https://developer.mozilla.org/ru/docs/Glossary/CORS> (дата обращения - 25.03.2021)
7. Дакетт Джон, HTML и CSS. Разработка и дизайн веб-сайтов, Изд-во «Эксмо», 2013, 480с. (дата обращения – 21.02.2021)
8. Алан Бьюли, Изучаем SQL, Изд-во «Символ-Плюс», 2007, 312 с. (22.02.2021)
9. Санджей Патни, Pro RESTfull APIs, США: Изд-во «Apress», 2017. 130с. (дата обращения – 22.02.2021)
10. Марк Дж. Прайс, С# 9 и .NET 5, США: Изд-во «Apress», 2020. 822с. (дата обращения – 23.02.2021)
11. Марк Дж. Прайс, С# 8.0 и .NET 3.0, США: Изд-во Apress, 2019. 820с. (дата обращения – 23.02.2021)
12. Скит Джон, С# для профессионалов. Тонкости программирования, Изд-во «Вильямс», 2019, 608с. (дата обращения – 24.03.2021)
13. Адам Фримен, ASP.Net Core MVC с примерами на С# для профессионалов, Изд-во «Вильямс», 2017, 992с. (дата обращения -24.03.2021)

14. Уймин Антон Григорьевич, Сетевое и системное администрирование. Демонстрационный экзамен КОД 1.1. Учебно-методическое пособие для СПО, Изд-во «Лань», 2021. 480 с. (дата обращения – 25.02.2021)
15. Джон Галоувей, Брэд Уилсон, Professional ASP.NET MVC 5, Изд-во «Джон Вилли & Сунс», 2014 565с. (дата обращения – 25.02.2021)
16. Стрекалова Н. Д. Бизнес-планирование: учеб. пособие. СПб.: Питер, 2013. 352 с.: ил. (дата обращения – 22.03.2021)
17. Букатов А.А. Гуда С.А., Компьютерные сети. Расширенный начальный курс, Изд-во «Питер», 2019, 496с. (дата обращения -23.03.2021)
18. Сравнение онлайн кинотеатров [Электронный ресурс] <https://ichip.ru/obzory/programmy-i-prilozheniya/sravniyaem-i-vybiraem-onlajn-kinoteatr-742042> (дата обращения - 27.04.2021)
19. Форматы видео файлов [Электронный ресурс] <https://info.sibnet.ru/article/566085> (дата обращения - 28.04.2021)
20. Статистика использования онлайн кинотеатров [Электронный ресурс] <https://www.kommersant.ru/doc/4520809> (дата обращения - 28.04.2021)
21. Российские онлайн кинотеатры [Электронный ресурс] <https://dtf.ru/cinema/666338-rossiyskie-legalnye-onlayn-kinoteatry-za-2020-god-pokazali-rekordnyy-rost-ih-pokazateli-uvelichilis-na-66> (дата обращения - 26.04.2021)
22. Рост рынка российских кинотеатров [Электронный ресурс] <https://sber.pro/publication/gonka-onlain-kinoteatrov-kak-karantin-izmenit-rynok> (дата обращения - 29.04.2021)
23. Заработок онлайн кинотеатров [Электронный ресурс] <https://vc.ru/media/48238-onlayn-kinoteatry-v-rossii-ceny-dostupnost-na-razlichnyh-platformah-i-perspektivy-rynka> (дата обращения - 29.04.2021)
24. Цены на подписки онлайн кинотеатров [Электронный ресурс] <https://www.sravni.ru/text/2017/7/20/gde-desheвле-bitva-onlajn-kinoteatrov> (дата обращения - 29.04.2021)

25.Васильев А. В. Бизнес-планирование инвестиционных проектов: учеб. пособие. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2000. 63 с.: ил. (дата обращения – 28.04.2021)

ПРИЛОЖЕНИЕ А

Листинг основных файлов программы

CustomersController.cs

```
using AutoMapper;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.Owin;
using Microsoft.Owin.Security;
using System;
using System.Data.Entity;
using System.Web.Services;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;
using MovieRent.Dtos;
using MovieRent.Models;
using System.Web.Http.Cors;

namespace MovieRent.Controllers.Api
{
    [EnableCors(origins: "*", headers: "*", methods: "*")]
    [Authorize]
    public class CustomersController : ApiController
    {
        private ApplicationDbContext _context;

        public CustomersController()
        {
            _context = new ApplicationDbContext();
        }

        // GET /api/customers
        public IHttpActionResult GetCustomers(string query = null)
        {
            var customersQuery = _context.Customers
                .Include(c => c.MembershipType);

            if (!String.IsNullOrEmpty(query))
                customersQuery = customersQuery.Where(c => c.Name.Contains(query));

            var customerDtos = customersQuery
                .ToList()
                .Select(Mapper.Map<Customer, CustomerDto>);

            return Ok(customerDtos);
        }

        // GET /api/customers/1
        public IHttpActionResult GetCustomer(int id)
        {
            var customer = _context.Customers.SingleOrDefault(c => c.Id == id);

            if (customer == null)
                return NotFound();

            return Ok(Mapper.Map<Customer, CustomerDto>(customer));
        }

        // POST /api/customers
        [HttpPost]
        public IHttpActionResult CreateCustomer(CustomerDto customerDto)
        {

```

```

        if (!ModelState.IsValid)
            return BadRequest();

        var customer = Mapper.Map<CustomerDto, Customer>(customerDto);
        _context.Customers.Add(customer);
        _context.SaveChanges();

        customerDto.Id = customer.Id;
        return Created(new Uri(Request.RequestUri + "/" + customer.Id), customerDto);
    }

    // PUT /api/customers/1
    [HttpPut]
    public IActionResult UpdateCustomer(int id, CustomerDto customerDto)
    {
        if (!ModelState.IsValid)
            return BadRequest();

        var customerInDb = _context.Customers.SingleOrDefault(c => c.Id == id);

        if (customerInDb == null)
            return NotFound();

        Mapper.Map(customerDto, customerInDb);

        _context.SaveChanges();

        return Ok();
    }

    // DELETE /api/customers/1
    [HttpDelete]
    public IActionResult DeleteCustomer(int id)
    {
        var customerInDb = _context.Customers.SingleOrDefault(c => c.Id == id);

        if (customerInDb == null)
            return NotFound();

        _context.Customers.Remove(customerInDb);
        _context.SaveChanges();

        return Ok();
    }
}

```

MovieController.cs

```

using AutoMapper;
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Web.Http;
using MovieRent.Dtos;
using MovieRent.Models;
using System.Web.Http.Cors;

namespace MovieRent.Controllers.Api
{
    [EnableCors(origins: "*", headers: "*", methods: "*")]
    [Authorize]
    public class MoviesController : ApiController
    {
        private ApplicationDbContext _context;
    }
}

```

```

public MoviesController()
{
    _context = new ApplicationDbContext();
}

public IEnumerable<MovieDto> GetMovies(string query = null)
{
    var moviesQuery = _context.Movies
        .Include(m => m.Genre)
        .Where(m => m.NumberAvailable > 0);

    if (!String.IsNullOrEmpty(query))
        moviesQuery = moviesQuery.Where(m => m.Name.Contains(query));

    return moviesQuery
        .ToList()
        .Select(Mapper.Map<Movie, MovieDto>);
}

public IHttpActionResult GetMovie(int id)
{
    var movie = _context.Movies.SingleOrDefault(c => c.Id == id);

    if (movie == null)
        return NotFound();

    return Ok(Mapper.Map<Movie, MovieDto>(movie));
}

[HttpPost]
[Authorize(Roles = RoleName.CanManageMovies)]
public IHttpActionResult CreateMovie(MovieDto movieDto)
{
    if (!ModelState.IsValid)
        return BadRequest();

    var movie = Mapper.Map<MovieDto, Movie>(movieDto);
    _context.Movies.Add(movie);
    _context.SaveChanges();

    movieDto.Id = movie.Id;
    return Created(new Uri(Request.RequestUri + "/" + movie.Id), movieDto);
}

[HttpPut]
[Authorize(Roles = RoleName.CanManageMovies)]
public IHttpActionResult UpdateMovie(int id, MovieDto movieDto)
{
    if (!ModelState.IsValid)
        return BadRequest();

    var movieInDb = _context.Movies.SingleOrDefault(c => c.Id == id);

    if (movieInDb == null)
        return NotFound();

    Mapper.Map(movieDto, movieInDb);

    _context.SaveChanges();

    return Ok();
}

```

```

[HttpDelete]
[Authorize(Roles = RoleName.CanManageMovies)]
public IActionResult DeleteMovie(int id)
{
    var movieInDb = _context.Movies.SingleOrDefault(c => c.Id == id);

    if (movieInDb == null)
        return NotFound();

    _context.Movies.Remove(movieInDb);
    _context.SaveChanges();

    return Ok();
}
}
}

```

NewRentalController.cs

```

using System;
using System.Linq;
using System.Web.Http;
using MovieRent.Dtos;
using MovieRent.Models;
using System.Web.Http.Cors;

namespace MovieRent.Controllers.Api
{
    [EnableCors(origins: "*", headers: "*", methods: "*")]
    [Authorize]
    public class NewRentalsController : ApiController
    {
        private ApplicationDbContext _context;

        public NewRentalsController()
        {
            _context = new ApplicationDbContext();
        }

        [HttpPost]
        public IActionResult CreateNewRentals(NewRentalDto newRental)
        {
            var customer = _context.Customers.Single(
                c => c.Id == newRental.CustomerId);

            var movies = _context.Movies.Where(
                m => newRental.MovieIds.Contains(m.Id)).ToList();

            foreach (var movie in movies)
            {
                if (movie.NumberAvailable == 0)
                    return BadRequest("Movie is not available.");

                movie.NumberAvailable--;

                var rental = new Rental
                {
                    Customer = customer,
                    Movie = movie,
                    DateRented = DateTime.Now
                };

                _context.Rentals.Add(rental);
            }

            _context.SaveChanges();
        }
    }
}

```

```

        return Ok();
    }
}

```

Movie.cs – модель для базы данных

```

using System;
using System.ComponentModel.DataAnnotations;

namespace MovieRent.Models
{
    public class Movie
    {
        public int Id { get; set; }

        [Required]
        [StringLength(255)]
        public string Name { get; set; }

        public Genre Genre { get; set; }

        [Display(Name = "Genre")]
        [Required]
        public byte GenreId { get; set; }

        public DateTime DateAdded { get; set; }

        [Display(Name = "Release Date")]
        public DateTime ReleaseDate { get; set; }

        [Display(Name = "Number in Stock")]
        [Range(1, 20)]
        public byte NumberInStock { get; set; }

        public byte NumberAvailable { get; set; }
    }
}

```

Rental.cs – модель для базы данных

```

using System;
using System.ComponentModel.DataAnnotations;

namespace MovieRent.Models
{
    public class Rental
    {
        public int Id { get; set; }

        [Required]
        public Customer Customer { get; set; }

        [Required]
        public Movie Movie { get; set; }

        public DateTime DateRented { get; set; }

        public DateTime? DateReturned { get; set; }
    }
}

```

Customer.cs – модель для базы данных

```

using System;

```

```

using System.ComponentModel.DataAnnotations;

namespace MovieRent.Models
{
    public class Customer
    {
        public int Id { get; set; }

        [Required]
        [StringLength(255)]
        public string Name { get; set; }

        public bool IsSubscribedToNewsletter { get; set; }

        public MembershipType MembershipType { get; set; }

        [Display(Name = "Membership Type")]
        public byte MembershipTypeId { get; set; }

        [Display(Name = "Date of Birth")]
        [Min18YearsIfAMember]
        public DateTime? Birthdate { get; set; }
    }
}

```

MembershipType.cs

```

using System.ComponentModel.DataAnnotations;

namespace MovieRent.Models
{
    public class MembershipType
    {
        public byte Id { get; set; }
        [Required]
        public string Name { get; set; }
        public short SignUpFee { get; set; }
        public byte DurationInMonths { get; set; }
        public byte DiscountRate { get; set; }

        public static readonly byte Unknown = 0;
        public static readonly byte PayAsYouGo = 1;
    }
}

```

CustomerFormViewModel.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using MovieRent.Models;

namespace MovieRent.ViewModels
{
    public class CustomerFormViewModel
    {
        public IEnumerable<MembershipType> MembershipTypes { get; set; }
        public Customer Customer { get; set; }
    }
}

```

MovieFormViewModel.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

```

```

using MovieRent.Models;

namespace MovieRent.ViewModels
{
    public class MovieFormViewModel
    {
        public IEnumerable<Genre> Genres { get; set; }

        public int? Id { get; set; }

        [Required]
        [StringLength(255)]
        public string Name { get; set; }

        [Display(Name = "Genre")]
        [Required]
        public byte? GenreId { get; set; }

        [Display(Name = "Release Date")]
        [Required]
        public DateTime? ReleaseDate { get; set; }

        [Display(Name = "Number in Stock")]
        [Range(1, 20)]
        [Required]
        public byte? NumberInStock { get; set; }

        public string Title
        {
            get
            {
                return Id != 0 ? "Edit Movie" : "New Movie";
            }
        }

        public MovieFormViewModel()
        {
            Id = 0;
        }

        public MovieFormViewModel(Movie movie)
        {
            Id = movie.Id;
            Name = movie.Name;
            ReleaseDate = movie.ReleaseDate;
            NumberInStock = movie.NumberInStock;
            GenreId = movie.GenreId;
        }
    }
}

```

Views

Customers

CustomersForm.cshtml

```

@model MovieRent.ViewModels.CustomerFormViewModel
@{
    ViewBag.Title = "New";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>New Customer</h2>

```



```

@using (Html.BeginForm("Save", "Customers"))
{
    @Html.ValidationSummary(true, "Please fix the following errors.")
    <div class="form-group">
        @Html.LabelFor(m => m.Customer.Name)
        @Html.TextBoxFor(m => m.Customer.Name, new { @class = "form-control" })
        @Html.ValidationMessageFor(m => m.Customer.Name)
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.Customer.MembershipTypeId)
        @Html.DropDownListFor(m => m.Customer.MembershipTypeId, new
SelectList(Model.MembershipTypes, "Id", "Name"), "", new { @class = "form-control" })
        @Html.ValidationMessageFor(m => m.Customer.MembershipTypeId)
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.Customer.Birthdate)
        @Html.TextBoxFor(m => m.Customer.Birthdate, "{0:d MMM yyyy}", new { @class =
"form-control" })
        @Html.ValidationMessageFor(m => m.Customer.Birthdate)
    </div>
    <div class="checkbox">
        <label>
            @Html.CheckBoxFor(m => m.Customer.IsSubscribedToNewsletter) Subscribed to
Newsletter?
        </label>
    </div>
    @Html.HiddenFor(m => m.Customer.Id)
    @Html.AntiForgeryToken()
    <button type="submit" class="btn btn-primary">Save</button>
}

@section scripts
{
    @Scripts.Render("~/bundles/jqueryval")
}

Index.cshtml
@model IEnumerable<MovieRent.Models.Customer>
@{
    ViewBag.Title = "Customers";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Customers</h2>
<p>
    @Html.ActionLink("New Customer", "New", "Customers", null, new { @class = "btn btn-
primary" })
</p>

<table id="customers" class="table table-bordered table-hover">
    <thead>
        <tr>
            <th>Customer</th>
            <th>Membership Type</th>
            <th>Delete</th>
        </tr>
    </thead>
    <tbody></tbody>
</table>

@section scripts
{
    <script>
        $(document).ready(function () {
            var table = $("#customers").DataTable({

```

```

        ajax: {
            url: "https://localhost:44300/api/customers",
            dataSrc: ""
        },
        columns: [
            {
                data: "name",
                render: function(data, type, customer) {
                    return "<a href='/customers/edit/' + customer.id + '>' +
customer.name + "</a>";
                }
            },
            {
                data: "membershipType.name"
            },
            {
                data: "id",
                render: function(data) {
                    return "<button class='btn-link js-delete' data-customer-id="
+ data + ">Delete</button>";
                }
            }
        ]
    });

    $("#customers").on("click", ".js-delete", function () {
        var button = $(this);

        bootbox.confirm("Are you sure you want to delete this customer?",
function (result) {
            if (result) {
                $.ajax({
                    url: "https://localhost:44300/api/customers/" +
button.attr("data-customer-id"),
                    method: "DELETE",
                    success: function () {
                        table.row(button.parents("tr")).remove().draw();
                    }
                });
            }
        });
    });
});
</script>
}

```

Home pages

Home.cshtml

```

@{
    ViewBag.Title = "Home Page";
}

<div class="jumbotron">
    <h1>MovieRent</h1>
    <p class="lead">Renting a movie has never been easier. Start with us now</p>
</div>

```

About.cshtml

```

@{
    ViewBag.Title = "About";
}
<h2>@ViewBag.Title.</h2>
<h3>@ViewBag.Message</h3>

```

<p>Use this area to provide additional information.</p>

Movies

MovieForm.cshtml

@model MovieRent.ViewModels.MovieFormViewModel

@{

ViewBag.Title = Model.Title;

}

<h2>@Model.Title</h2>

@using (Html.BeginForm("Save", "Movies"))

{

@Html.ValidationSummary(true, "Please fix the following errors.")

<div class="form-group">

@Html.LabelFor(m => m.Name)

@Html.TextBoxFor(m => m.Name, new { @class = "form-control" })

@Html.ValidationMessageFor(m => m.Name)

</div>

<div class="form-group">

@Html.LabelFor(m => m.ReleaseDate)

@Html.TextBoxFor(m => m.ReleaseDate, "{0:d MMM yyyy}", new { @class = "form-control" })

@Html.ValidationMessageFor(m => m.ReleaseDate)

</div>

<div class="form-group">

@Html.LabelFor(m => m.GenreId)

@Html.DropDownListFor(m => m.GenreId, new SelectList(Model.Genres, "Id", "Name"), "", new { @class = "form-control" })

@Html.ValidationMessageFor(m => m.GenreId)

</div>

<div class="form-group">

@Html.LabelFor(m => m.NumberInStock)

@Html.TextBoxFor(m => m.NumberInStock, new { @class = "form-control" })

@Html.ValidationMessageFor(m => m.NumberInStock)

</div>

@Html.HiddenFor(m => m.Id)

@Html.AntiForgeryToken()

<button type="submit" class="btn btn-primary">Save</button>

}

@section scripts

{

@Scripts.Render("~/bundles/jqueryval")

}

List.cshtml

@model IEnumerable<MovieRent.Models.Movie>

@{

ViewBag.Title = "Movies";

Layout = "~/Views/Shared/_Layout.cshtml";

}

<h2>Movies</h2>

<p>

@Html.ActionLink("New Movie", "New", "Movies", null, new { @class = "btn btn-primary" })

</p>

<table id="movies" class="table table-bordered table-hover">

<thead>

<tr>

<th>Movie</th>

```

        <th>Genre</th>
        <th>Delete</th>
    </tr>
</thead>
<tbody>
</tbody>
</table>
@section scripts
{
    <script>
        $(document).ready(function () {
            var table = $("#movies").DataTable({
                ajax: {
                    url: "https://localhost:44355/api/movies",
                    dataSrc: ""
                },
                columns: [
                    {
                        data: "name",
                        render: function(data, type, movie) {
                            return "<a href='/movies/edit/' + movie.id + '>' +
movie.name + "</a>";
                        }
                    },
                    {
                        data: "genre.name"
                    },
                    {
                        data: "id",
                        render: function(data) {
                            return "<button class='btn-link js-delete' data-movie-id=" +
data + ">Delete</button>";
                        }
                    }
                ]
            });

            $("#movies").on("click", ".js-delete", function () {
                var button = $(this);

                bootbox.confirm("Are you sure you want to delete this movie?", function
(result) {
                    if (result) {
                        $.ajax({
                            url: "https://localhost:44355/api/movies/" +
button.attr("data-movie-id"),
                            method: "DELETE",
                            success: function () {
                                table.row(button.parents("tr")).remove().draw();
                            }
                        });
                    }
                });
            });
        });
    </script>
}
}
Rentals
New.cshtml
@model dynamic
@{
    ViewBag.Title = "New Rental Form";

```

```

}

<h2>New Rental Form</h2>

<form id="newRental">
  <div class="form-group">
    <label>Customer</label>
    <div class="tt-container">
      <input id="customer" name="customer" data-rule-validCustomer="true" required
type="text" value="" class="form-control" />
    </div>
  </div>

  <div class="form-group">
    <label>Movie</label>
    <div class="tt-container">
      <input id="movie" name="movie" data-rule-atLeastOneMovie="true" type="text"
value="" class="form-control" />
    </div>
  </div>

  <div class="row">
    <div class="col-md-4 col-sm-4">
      <ul id="movies" class="list-group"></ul>
    </div>
  </div>

  <button class="btn btn-primary">Submit</button>
</form>
@section scripts
{
  @Scripts.Render("~/bundles/jqueryval")
  <script>
    $(document).ready(function () {

      var vm = {
        movieIds: []
      };

      var customers = new Bloodhound({
        datumTokenizer: Bloodhound.tokenizers.obj.whitespace('name'),
        queryTokenizer: Bloodhound.tokenizers.whitespace,
        remote: {
          url: 'https://localhost:44355/api/customers?query=%QUERY',
          wildcard: '%QUERY'
        }
      });

      $('#customer').typeahead({
        minLength: 3,
        highlight: true
      }, {
        name: 'customers',
        display: 'name',
        source: customers
      }).on("typeahead:select", function(e, customer) {
        vm.customerId = customer.id;
      });

      var movies = new Bloodhound({
        datumTokenizer: Bloodhound.tokenizers.obj.whitespace('name'),
        queryTokenizer: Bloodhound.tokenizers.whitespace,
        remote: {
          url: 'https://localhost:44355/api/movies?query=%QUERY',

```

```

        wildcard: '%QUERY'
    });
    $('#movie').typeahead({
        minLength: 3,
        highlight: true
    }, {
        name: 'movies',
        display: 'name',
        source: movies
    }).on("typeahead:select", function (e, movie) {
        $("#movies").append("<li class='list-group-item'>" + movie.name +
"</li>");

        $("#movie").typeahead("val", "");

        vm.movieIds.push(movie.id);
    });

$.validator.addMethod("validCustomer", function () {
    return vm.customerId && vm.customerId !== 0;
}, "Please select a valid customer.");

$.validator.addMethod("atLeastOneMovie", function () {
    return vm.movieIds.length > 0;
}, "Please select at least one movie.");

var validator = $("#newRental").validate({
    submitHandler: function () {
        $.ajax({
            url: "https://localhost:44355/api/newRentals",
            method: "post",
            data: vm
        })
        .done(function () {
            toastr.success("Rentals successfully recorded.");

            $("#customer").typeahead("val", "");
            $("#movie").typeahead("val", "");
            $("#movies").empty();

            vm = { movieIds: [] };

            validator.resetForm();
        })
        .fail(function () {
            toastr.error("Something unexpected happened.");
        });

        return false;
    }
});
</script>
}

```