

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление 09.03.01 «Информатика и вычислительная техника»

подготовки:

Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

**МИКРОСЕРВИС УЧЁТА
СВЕДЕНИЙ О ФИЛЬМАХ**

Студент

Зорин В.Н.
(подпись, дата)

Руководитель

Дубравин А.В.
(подпись, дата)

Нормоконтролёр

Бычков А.С.
(подпись, дата)

Работа допущена к защите (протокол зас. кафедры от **02.06.2022 г. № 13**)

Заведующий кафедрой

(подпись)

М.А. Митрохин

Работа защищена с отметкой отлично (протокол заседания ГЭК от
15.06.22 г. № 11)

Секретарь ГЭК

(подпись)

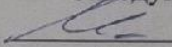
Е.И. Гурин

Пенза 2022

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
«26» апреля 2022 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Зорин Всеволод Николаевич, гр.18ВВ2 факультета вычислительной техники.

2. Тема работы: «Микросервис учёта сведений о фильмах»

Тема утверждена приказом ПГУ №0357/о от "26" апреля 2022 г.

3. Руководитель бакалаврской работы к.т.н., доцент каф. ВТ ПГУ Дубрагин А.В.

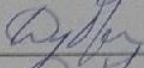
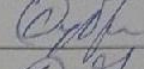
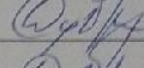
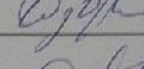
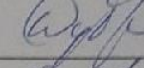
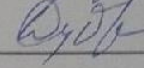
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать приложение «Микросервис учёта сведений о фильмах». Система должна обеспечивать сохранение и обновление данных о фильмах, создаваемые пользователем, просмотр сведений об актуальных премьерах, а также иметь возможность беспрепятственно расширяться и масштабироваться.

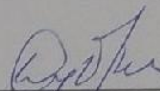
5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметных областей: изучить аналоги порталов о фильмах, изучить целевую архитектуру приложения, изучить типы СУБД для хранения данных, рассмотреть языки программирования для разработки микросервиса, а также изучить функционал программных библиотек *ASP.NET Core* и *WCF*.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы , %	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	5	26.04.2022 – 30.04.2022	
2.	Анализ требований	10	30.04.2022 – 02.05.2022	
3.	Выбор технологий для создания приложения	15	03.05.2022 – 07.05.2022	
4.	Описание алгоритмов	25	08.05.2022 – 11.05.2022	
5.	Разработка приложения «Микросервис учёта сведений о фильмах»	35	12.05.2022 – 29.05.2022	
6.	Оформление пояснительной записки	10	30.05.2022 - 31.05.2022	

Дата выдачи задания "26" апреля 2022 г.

Руководитель бакалаврской работы  26.04.22 Дубравин А.В.
(подпись, дата)

Задание к исполнению принял студент  26.04.22 Зорин В.Н.
(подпись, дата)

Работу к защите допустить
Декан факультета  14.04.22 Фионова Л.Р.
(подпись, дата)

Реферат

Работа содержит 115 листов, 28 рисунков, 2 таблицы, 14 источников, 4 приложения.

БД, СУБД, МИКРОСЕРВИСНАЯ АРХИТЕКТУРА, С#

Цель ВКР – разработать микросервис, который будет обеспечивать учёт сведений о фильмах.

Объектом разработки является микросервис, обеспечивающее выполнение следующих функций: обработка запросов на создание, удаление или обновление данных о фильмах, а также возможность просмотра сведений о предстоящих премьерах. Также необходимо создать *API* для отображения информации в веб-клиенте.

В результате проведенной работы было разработан программный продукт, удовлетворяющий поставленной задаче: разработана структура БД, создан *API*, создан целевой микросервис, а также веб-приложение клиента. Приложение было протестировано и успешно отлажено.

						ПГУ 1.090301.14.001 ПЗ			
Изм.	Лист	№ докум	№	Подпись	Дата	Микросервис учёта сведений о фильмах Пояснительная записка	Лит.	Лист	Листов
Разраб.		Зорин В.Н.							
Провер.		Дубравин А.В.						2	115
Н. Контр.		Бычков А.С.					ФВТ, каф. ВТ гр. 18ВВ2		
Утверд.		Митрохин М.А.							

Содержание

Перечень сокращений и обозначений.....	7
Введение.....	8
1 Микросервис учёта сведений о фильмах.....	10
1.1 Анализ предметной области	10
1.1.1 Анализ аналогов.....	10
1.1.2 Варианты реализации приложения	13
1.1.3 Описание целевой системы.....	20
1.1.4 Постановка задачи	22
1.1.5 Выбор языка программирования.....	23
1.1.6 Выбор СУБД.....	26
1.1.7 Выбор библиотеки для микросервиса	29
1.1.8 Выбор библиотеки для веб-клиента.....	30
1.1.9 Описание архитектуры <i>REST API</i>	33
1.1.10 Выводы.....	34
1.2 Реализация	35
1.2.1 Проектирование базы данных	35
1.2.2 Разработка основного микросервиса	37
1.2.3 Разработка <i>REST API</i>	38
1.2.4 Разработка <i>web</i> -приложения.....	40
1.2.5 Взаимодействие классов	44
1.3 Результат работы приложения.....	45
1.3.1 Результат работы приложения с использованием веб-клиента	45
2 Безопасность жизнедеятельности оператора ЭВМ	54

2.1 Требования к помещениям для работы с ЭВМ.....	54
2.2 Требования к микроклимату во время работы с ПЭВМ	55
Заключение	56
Список использованных источников	57
Приложение А – <i>UML</i> -диаграммы	59
Приложение Б – Листинг программы <i>FilmPortal.ServiceWCF</i>	64
Приложение В – Листинг программы <i>FilmPortal</i>	82
Приложение Г – Презентация	104

Перечень сокращений и обозначений

API (Application Programming Interface) – программный интерфейс приложения

HTTP (HyperText Transfer Protocol) – протокол передачи гипертекста

JSON (JavaScript Object Notation) – текстовый формат обмена данными, основанный на языке JavaScript

NPGSQL – библиотека для работы с базами данных *PostgreSQL* на *C#*

REST (Representational State Transfer) – передача состояния представления

SQL (Structured Query Language) – язык структурированных запросов

UML (Unified Modeling Language) – унифицированный язык моделирования

WCF (англ. Windows Communication Foundation) – среда выполнения и набор интерфейсов *API* для создания систем, отправляющих сообщения между службами и клиентами, разработанная корпорацией Microsoft

XML (eXtensible Markup Language) – расширяемый язык разметки

БД – база данных

ОС – операционная система

СУБД – система управления базами данных

ЯП – язык программирования

Введение

Сегодня невозможно обойтись без помощи компьютера, и соответственно Интернета. Его первое появление датируется 1969-ым годом. В этом году в США была создана специальная компьютерная сеть, которая объединяла компьютерные центры министерства обороны и ряда академических организаций. В данный периода сеть была узконаправленной и решала только проблемы внутри группы, где была она организована.

В 1992-ом году Интернет появляется уже доступным для пользователей всего мира. Появляются первые веб-сайты. Они привлекали большое внимание первых пользователь Всемирной Паутины. Со временем они стали неотъемлемой частью повседневной жизни. Если инструмент удобен в использовании, то к нему пользователи прибегают чаще всего. Если же нет, то использовать его никто не станет. По мере роста количества *web*-сайтов пользователи становятся все менее терпимы к тем ресурсам, которыми сложно пользоваться [1]. Поэтому каждая ошибка при разработке может привести к крушению бизнеса и банкротству компании-разработчика. Значение простоты и удобства использования веб-ресурсов в наше время сильно возросло, особенно в киноиндустрии.

Киноиндустрия всегда была популярной областью во всём мире. Вместе с появлением Интернета появились тематические порталы, которые могли удовлетворить потребность в обсуждении той или иной киноленты. Со временем помимо обсуждений в веб-ресурсах стали появляться всё новые и новые функции (например, онлайн-кинотеатр), которые могли удовлетворить потребность пользователя. В наши дни киноиндустрия переживает сложные времена. Человеку предоставляются огромные возможности для выбора фильма, но при этом он не может сделать определённый выбор. Также некоторые сервисы кино требуют платную подписку, чтобы размещать информацию о том или ином фильме, что усложняет пользователю работу с

ресурсом.

Главная проблема киноресурсов – перегруженность лишними сервисами и деталями, которые не относятся к тематике кино. Из-за этого пользователь не может найти то, что ему нужно. Кроме того, из-за этого снижается скорость работы ресурса, что негативно влияет на восприятие пользователем. Поэтому решением проблемы является создание нового веб-ресурса, в основе которого будет лежать микросервисная архитектура. Он будет содержать в себе функции, которые действительно будут нужны пользователю.

					ПГУ1.090301.14.001 ПЗ	Лист
						9
Изм.	Лист	№ докум.	Подпись	Дата		

1 Микросервис учёта сведений о фильмах

1.1 Анализ предметной области

1.1.1 Анализ аналогов

Для того, чтобы создать веб-ресурс, необходимо проанализировать аналоги, чтобы определить потребности конечного пользователя. По данным ресурса *infoselection.ru* в России насчитывается 30 веб-ресурсов, посвящённых кино. Наиболее популярными среди них являются: Кинопоиск, Искусство кино, Афиша.ру.

«Кинопоиск» является крупнейшим в России сайтом о кино. Он был основан в 1990-ом году и продолжает функционировать в наши дни. Сайт предоставляет функционал, позволяющий пользователю не только читать новые статьи про фильмы, но и смотреть их через внутренний сервис. На рисунке 1 представлена домашняя страница сайта.

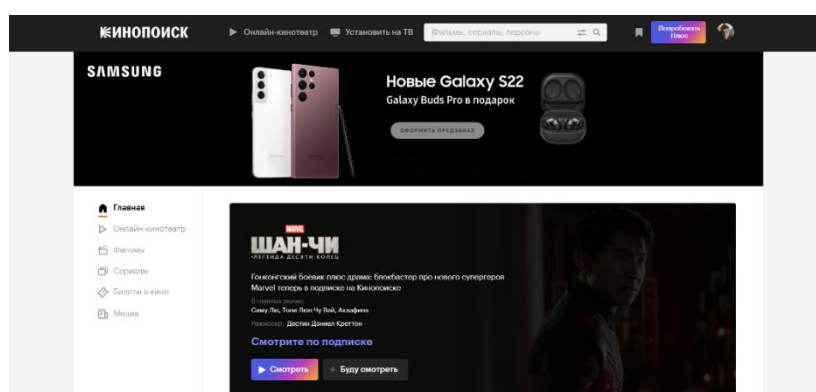


Рисунок 1 – Домашняя страница сайта «Кинопоиска»

Имеется возможность авторизации на сайте с помощью сервисов компании «Яндекс», что существенно упрощает процесс. Помимо «Яндекса» присутствует интеграция с известным онлайн-кинотеатром «Amediateka», предоставляющая зарубежные сериалы на русском языке, а также с *IMDB*, мировым сайтом о кино, откуда берётся оценка разных фильмов.

Главный недостаток «Кинопоиска» заключается в огромном количестве рекламы на портале, усложняющая работу с ним. Поэтому в создаваемом веб-ресурсе будет отсутствовать реклама, чтобы работа с ним была простой и понятной для конечного пользователя.

«Искусство кино» - веб-портал одноимённого издания, который специализируется в узкой сфере авторского кино. В отличие от вышеупомянутого «Кинопоиска», он не требует авторизации и комментариев к записям. Нажав на кнопку «Содержание», сайт предлагает свои функции. Домашняя страница сайта представлен на рисунке 2.

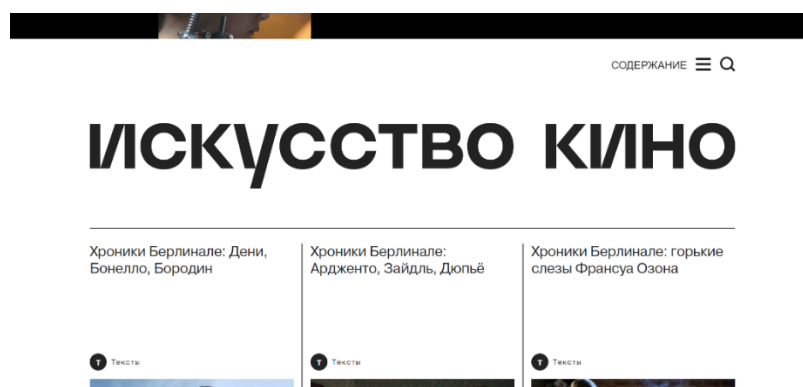


Рисунок 2 - Домашняя страница сайта «Искусство кино»

Благодаря компактному интерфейсу сайтом удобно пользоваться. В нём нет лишних элементов, присутствует по минимуму реклама (в основном электронной версии журнала) и используется чёрно-белая гамма с приятными для глаза шрифтами. Помимо этого, вверху сайта есть индикатор, обозначающий, сколько осталось до конца статьи. Это позволяет пользователю быстро проанализировать объём статьи, которую он будет просматривать. Несмотря на удобство пользования сайтом, «Искусство кино» проигрывает своим конкурентам по причине узкоспециализированной тематике авторского кино, а также малого функционала.

«Афиша.ру» охватывает куда больше областей, чем два вышеперечисленных веб-ресурса. Сайт сотрудничает с известным онлайн-

кинотеатром *Okko*, что позволяет пользователю смотреть фильмы за определённую плату.

На домашней странице имеется календарь, позволяющий пользователю просматривать будущие культурные события (не только в сфере кино) и тем самым спланировать время для их посещения, что является плюсом в плане дизайна сайта и привлекает более широкую аудиторию сайта. На рисунке 3 приведён скриншот данной страницы.

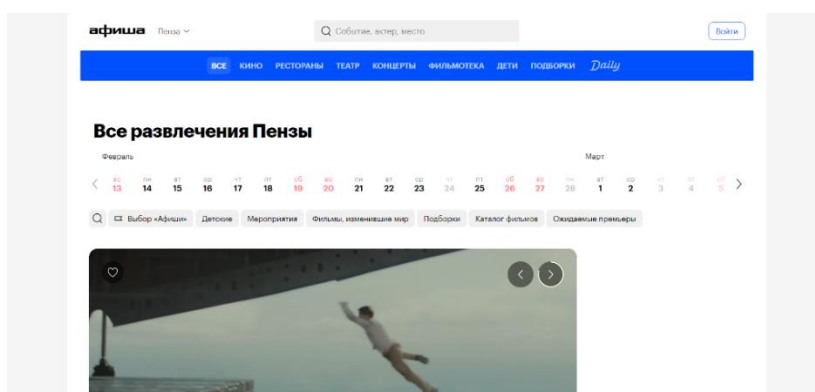


Рисунок 3 - Домашняя страница сайта «Афиши.ру»

Но, к сожалению, веб-ресурс перегружен, как и «Кинопоиск». Но, в отличие от конкурента, он перегружен сервисами, которые не относятся к тематике кино, а также полноценного инструмента для написания статей к фильмам. Из-за этого сайтом довольно трудно пользоваться.

Рассмотрев вышеперечисленные аналоги, были сделаны следующие выводы. «Кинопоиск» предоставляет широкие возможности пользователю для просмотра кино и написания пользовательских рецензий, но при этом страдает громоздким дизайном, перегруженным рекламой и рекомендациями от сайта. «Искусство кино» является ярким примером качественного дизайна без лишних элементов, но при этом небольшая функциональность не привлекает широкую аудиторию. «Афиша.ру» справляется с привлечением широкой аудитории благодаря предоставляемому функционалу, но при этом он страдает от

несбалансированного дизайна. Поэтому необходимо разработать веб-ресурс без лишнего функционала, которым пользователю будет удобно пользоваться.

1.1.2 Варианты реализации приложения

Поскольку необходимо создать веб-приложение с только необходимыми для пользователя функциями, необходимо проанализировать существующие архитектуры ПО для его реализации.

Архитектура ПО – описание структуры программы, включающее в себя её компоненты, свойства и связи между ними. При правильном выборе она помогает создать IT-продукт, который легко модернизировать в соответствии с требованиями заказчика. Благодаря правильно спроектированной архитектуре, программный продукт удовлетворяет необходимым функциональным требованиям [2].

В основе качественно спроектированной программной архитектуры лежат 5 основных принципов.

Первый принцип - принцип разделения функций. ПО должно быть разделено на отдельные составные части. Важно предельное уменьшение количества точек соприкосновения, что обеспечит высокую связность и слабую связанность в программном продукте. Неверное разграничение функциональности может привести к высокой связанности компонентов программы и сложностям в добавлении новых функций.

Второй принцип – принцип основной ответственности. Каждый компонент должен отвечать за реализацию определённого свойства или функциональной возможности.

Третий принцип - принцип минимального знания. Сложные элементы программы не должны знать о реализации других элементов в программе.

Четвёртый принцип - отсутствие повторений. Функциональность отдельных элементов программы должна быть реализована в одном модуле или компоненте по схеме «один функционал – один компонент». Дублирование в таком случае не допускается.

Пятый принцип - принцип предварительного проектирования. Осуществляется только в случае необходимости. Если требования к ПО могут быть изменены или не определены, то следует создавать только те функции, которые могут потребоваться в ближайшей перспективе. Масштабное создание новых функций делается тогда, когда неверная архитектура несет значительные издержки.

Архитектура ПО делится на многослойную, многоуровневую, сервис-ориентированную (*Service Oriented Architecture — SOA*) и микросервисную архитектура.

1.1.2.1 Многослойная архитектура

Многослойная архитектура (англ. *Layered Architecture*) – архитектура ПО, работающая по принципу разделения операций в программе. Она разбита на слои, взаимосвязанные между собой. Каждый из них реализует определённый функционал. Яркий пример реализации данной архитектуры – монолитное приложение.

Оно полностью замкнуто в контексте поведения. Во время работы оно может взаимодействовать с другими службами или базами данных. Основное его отличие заключается в том, что данный тип приложения запускается в отдельном процессе и разворачивается, как отдельный элемент. Для горизонтального масштабирования приложение данного типа целиком дублируется на нескольких серверах. На рисунке 4 представлена схема многослойной архитектуры.

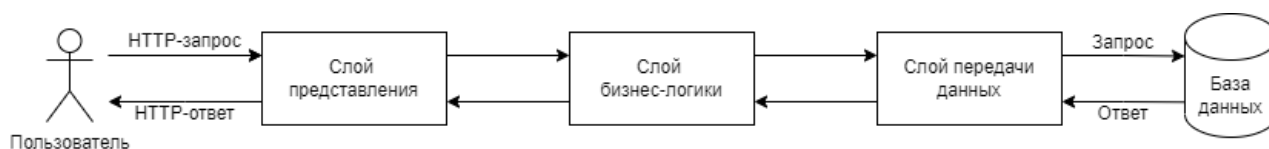


Рисунок 4 - Многослойная архитектура

Данный вид архитектуры делит ПО на следующие слои:

- слой представления – в нём находится пользовательский интерфейс, отвечающий за обеспечение хорошего пользовательского опыта;

- слой бизнес-логики – в нём содержится основной механизм работы приложения. Он отделяет интерфейс от основной программы, что даёт разработчику без затруднений менять логику в соответствии с требованиями, никак не влияя на другие слои ПО;

- слой передачи данных – отвечает за взаимодействие с постоянными хранилищами данных и другой информацией, которая никак не связана с основной логикой приложения.

Данные и элементы управления проходят через каждый слой и передаются от одного к другому. Данная архитектура повышает стабильную работу ПО.

Преимущества:

- простая реализация по сравнению с другими подходами, что позволяет разработать ПО в сжатые сроки;

- предлагает абстракцию благодаря разделению функционала между слоями приложения;

- изолированность слоёв приложения друг от друга, что позволяет исправлять ошибки в определённой области;

Недостатки:

- не предлагает большой масштабируемости, что в будущем приводит к проблемам в перестроении архитектуры приложения;

- программа с применением данной архитектуры будет иметь монолитную структуру, усложняющая добавление нового функционала в приложение и уменьшающая надёжность приложения;

- данные должны проходить по всем слоям приложения, даже если в этом нет необходимости.

Поскольку данная архитектура усложняет добавление новых функций, а также является ненадёжной ввиду того, что при сбое одного компонента будет сбой у всей системы, то она не будет использоваться при создании веб-ресурса.

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		15

1.1.2.2 Многоуровневая архитектура

Этот архитектурный подход разделяет комплекс ПО на уровни по принципу “клиент-сервер”. Архитектура может несколько уровней, разделяющих ответственности между поставщиком данных и потребителем.

Этот подход использует шаблон *Request Response* для связи между уровнями. В отличие от многослойной архитектуры, он предлагает масштабируемость, которая может быть как горизонтальной (масштабирование сети с помощью высокопроизводительных узлов), так и вертикальной (масштабирование каждого узла путем повышения его производительности).

Многоуровневая архитектура включает одноуровневую, двухуровневую и более системы.

Одноуровневая система работает как на стороне сервера, так и на стороне клиента. Это обеспечивает простоту развертывания и отличную скорость связи, а также устраняет необходимость межсистемного взаимодействия. Такая система подходит только для небольших однопользовательских приложений.

Двухуровневая система состоит из двух физических машин: сервера и клиента. Сервер включает хранилища и базы данных для работы программы, а клиент содержит слои презентации, бизнес-логики и передачи данных. Данный вид системы обеспечивает изоляцию операций управления данными, обработки данных и операций представления. Пример данной архитектуры приведен на рисунке 5.

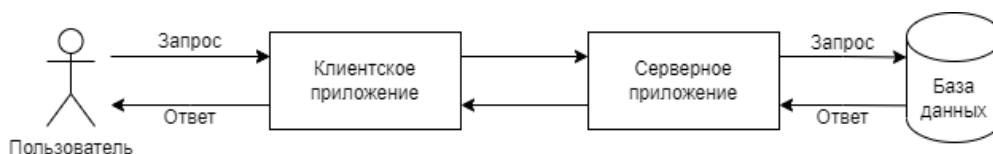


Рисунок 5 - Двухуровневая архитектура

Трехуровневая и n-уровневая системы обладают высокой масштабируемостью как по горизонтали, так и по вертикали. Реализация n-

уровневой системы, как правило, обходится дороже, но обеспечивает высокую производительность. Поэтому она обычно применяется в крупных и комплексных программных решениях.

Данный подход можно сочетать с современной сервис-ориентированной архитектурой, чтобы создавать сложнейшие модели. Поскольку реализация может оказаться дорогостоящей с точки зрения времени и ресурсов, рекомендуется использовать его для сложных ПО, требующих высокой производительности.

Преимущества:

- основная бизнес-логика приложения находится на сервере;
- простая настройка безопасности приложения;
- низкие требования к скорости канала между клиентами и сервером;

Недостатки:

- низкая скорость работы;
- сложность обслуживания и тестирования приложения;
- высокая стоимость обслуживания.

1.1.2.3 Сервис-ориентированная архитектура (SOA)

SOA – архитектура ПО, чей подход связан с определением, связыванием и интеграцией повторно используемых бизнес-сервисов, имеющих четкие границы и самодостаточных по своей функциональности. В рамках такой архитектуры можно организовывать бизнес-сервисы в бизнес-процессах. Внедряя концепцию сервисов, SOA переносит информационные технологии на следующий уровень, более подходящий для обеспечения функциональной совместимости и реализации в гетерогенных средах. Эта архитектурная модель состоит из компонентов и приложений, которые связываются друг с другом с помощью четко определенных сервисов.

Работает данная архитектура следующим образом. Клиент отправляет запрос с использованием стандартного протокола и формата данных по сети. Этот запрос обрабатывается *ESB (enterprise service bus — сервисная шина*

предприятия), которая считается сердцем сервис-ориентированной архитектуры и отвечает за оркестровку и маршрутизацию. С помощью сервисного репозитория ESB направляет запрос в специальный сервис, который может взаимодействовать с другими сервисами и базами данных, чтобы составить полезную нагрузку (данные) ответа. На рисунке 6 представлена схема данной архитектуры.

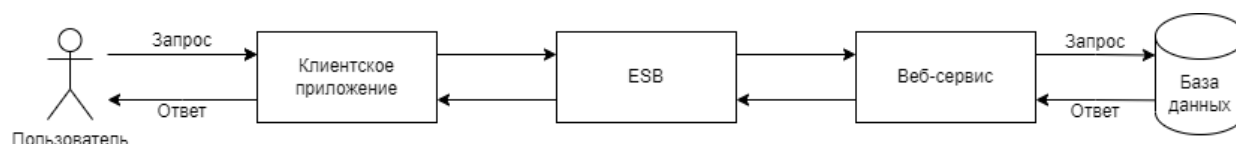


Рисунок 6 - SOA-архитектура

Полный вызов ответа на запрос согласуется с правилами управления и безопасности *SOA* для выполнения безопасной и корректной транзакции.

Сервисы *SOA* делятся на атомарные и композиционные. Атомарные сервисы предоставляют функции, не подлежащие дальнейшей декомпозиции. Композиционные сервисы сочетают в себе несколько атомарных сервисов, чтобы предоставлять сложную программу с различными функциями.

Преимущества:

- открытость, стандартизация протоколов доступа;
- поддержка параллелизма, масштабируемость;
- отказоустойчивость.

Недостатки:

- зависимость от состояния сетевых соединений;
- дополнительные вычислительные ресурсы и ПО для поддержки;

1.1.2.4 Микросервисная архитектура

В данном подходе приложение разрабатывается как набор небольших сервисов, каждый из которых работает в собственном процессе и связывается с легковесными механизмами, обычно *API* для *HTTP*-ресурса.

Особенности микросервисной архитектуры:

- основываются на бизнес-возможностях и могут развертываться независимо друг от друга с помощью полностью автоматизированного механизма;

- централизованное управление между сервисами минимально. Они могут быть написаны на разных языках, использовать разные технологии хранения данных.

Архитектура работает по принципу компонентизации сервисов. Она разделяет программное обеспечение на различные изолированные компоненты (сервисы), каждый из которых несет единую ответственность. Изменения в одном сервисе не должны затрагивать другие.

Рекомендуется развивать каждый микросервис отдельно под управлением разных команд. Поскольку передача данных происходит по стандартному протоколу и формату данных, структура одного сервиса не затронет функциональность сопутствующих. На рисунке 7 приведён пример реализации данной архитектуры.

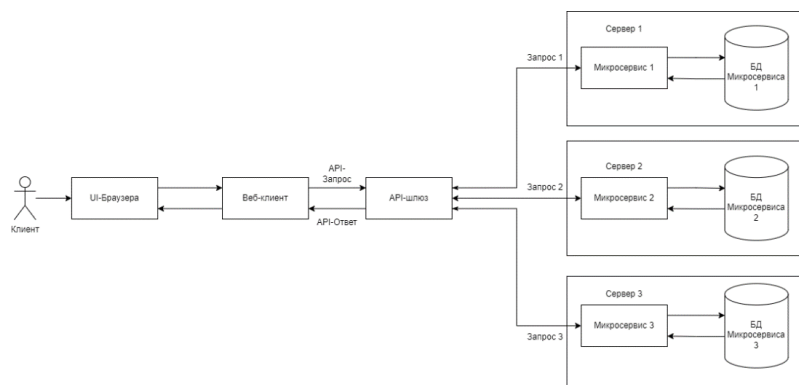


Рисунок 7 - Пример реализации микросервисной архитектуры

Преимущества:

- предлагает слабую связанность благодаря высокой степени изоляции;
- повышает модульность;
- сбой в одном сервисе не затронет всю систему, поскольку они изолированы;

- предлагает высокую гибкость и масштабируемость;
- простота модификации может ускорить итерации;
- позволяет реализовать улучшенную систему обработки ошибок;
- решает проблемы с потоками данных, которые бывают у многослойной архитектуры.

Недостатки:

- повышенный риск сбоя при обмене данными между сервисами;
- большим количеством сервисов трудно управлять;
- требует решения таких проблем, как задержки в сети, балансировка нагрузки и прочих трудностей, свойственных распределенной архитектуре;
- нуждается в комплексном тестировании в распределенной среде;
- на реализацию потребуется гораздо больше времени.

1.1.2.5 Выводы

В результате сравнения для реализации веб-ресурса была выбрана микросервисная архитектура. Многослойная архитектура не позволяет создать эффективное многофункциональное приложение из-за сложности добавления новых возможностей. Многоуровневая архитектура не подходит для реализации веб-ресурса по причине низкой скорости работы и сложностями в тестировании приложения с её использованием, а сервис-ориентированная архитектура не подходит из-за ограниченности сетевых соединений. Микросервисная архитектура лишена данных недостатков и позволяет создавать изолированные сервисы без сбоев в основной системе и обладает возможностью простой модификации системы.

1.1.3 Описание целевой системы

Целевой микросервис будет состоять из *API*-модуля, который будет обрабатывать запросы, поступающие от клиента, и модуля обработки данных, который будет связан с БД. Он будет работать по принципу компонентизации. В отличие от монолитной архитектуры, где все слои приложения располагаются в одном модуле и запускаются в одном процессе, микросервисы разделяют ПО

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		20

на различные изолированные компоненты, и каждый из которых несет единую ответственность.

Это значит, что целевой микросервис должен работать в отдельном процессе, то есть разворачиваться на отдельном сервере для бесперебойной работы, если хотя бы один из компонентов будет содержать в себе программный сбой. Изменения в одном сервисе не должны затрагивать другие за счёт слабой связности. Наглядный пример представляет собой веб-приложение. Пример его работы с использованием микросервисов представлен на рисунке 8.

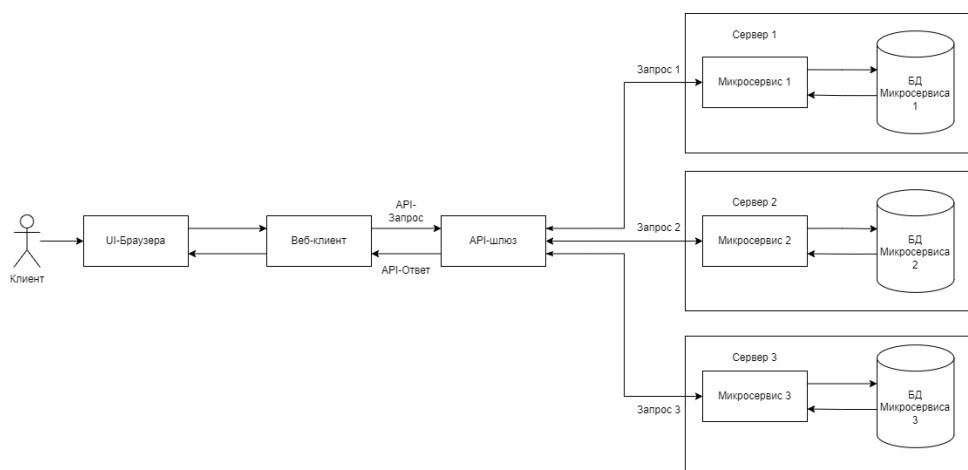


Рисунок 8 - Графическое представление работы веб-приложения с использованием микросервисов

Каждый компонент веб-приложения содержит свою БД, что позволяет разделять данные и операции между собой. Также микросервис может быть написан на любом доступном ЯП и работать с другими приложениями, что обеспечивает его кроссплатформенность. Отдельно стоит упомянуть работу веб-клиента, взаимодействующего с микросервисами.

Клиент должен быть развёрнут на собственном сервере и состоять из *API*-контроллера и *frontend*-части, которая выводит данные на веб-страницу. Оно обрабатывает данные, которые поступают с *frontend*-части, и отправляет их в контроллер. Контроллер отправляет их микросервису и ожидает ответа от него.

После получения информации контроллер микросервиса обрабатывает данные и в зависимости от запроса, полученного от клиента, выполняет операцию и возвращает ответ пользователю либо в виде сообщения, либо в виде потока данных.

Стоит отметить, что использование микросервисов помогает не только ускорить разработку продукта, но и сделать её гибкой. Проект из неделимого монолита превращается в систему связанных блоков — сервисов. Её использует такая крупная компания, как *Amazon*. В 2020 году по данным годового отчёта корпорации *Cloud Microservices Market Research* стоимость разработки облачных микросервисов оценили в 831,45 млн долларов. Это означает, что их использование будет востребовано в будущем.

1.1.4 Постановка задачи

Требуется создать микросервис, который будет вести учёт записей о фильмах и разворачиваться в отдельном процессе. Он должен соответствовать следующим требованиям:

- гетерогенность. Он может разрабатываться с применением разных ЯП и программных библиотек, что повысит его интегрируемость в другие приложения, а также добавит возможность работы на других ОС;

- децентрализованное управление данными. У микросервиса должен быть свой набор данных, а с другими приложениями он должен работать через специальный интерфейс. У него должен быть свой протокол данных, который позволит получать данные независимо от платформы;

- масштабируемость. Для расширения производительности системы необходимо предусмотреть возможность расширения функционала;

- независимость инфраструктуры. Микросервис должен быть самостоятельной единицей и разворачиваться в системе независимо от других приложений;

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		22

- отказоустойчивость. Это означает, что при сбое одного из микросервисов система должна продолжать работу и не создавать новые ошибки, которые могут усложнить разработку.

Исходя из требований требуется выбрать стек технологий, на которых будет реализован данный проект.

1.1.5 Выбор языка программирования

Прежде, чем приступить к разработке, необходимо выбрать ЯП, на котором будет написан микросервис. В данном разделе будут приведены основные языки программирования, используемые для создания микросервисов, а именно: *Python*, *Java*, *C#*.

1.1.5.1 *Python*

Python - популярный высокоуровневый язык программирования, предназначенный для создания приложений в различных сферах деятельности человека. Используется в веб-разработке, создании игр и работе с базами данных. Наиболее популярен в сфере машинного обучения и исследований искусственного интеллекта.

Впервые язык *Python* был упомянут в 1991-ом году Гвидо Ван Россумом. С этого года данный язык проделал большой путь развития. В 2000 году была издана версия 2.0, в 2008-ом - версия 3.0.

Преимущество данного языка заключается в том, что с его помощью можно создать читабельный код, позволяющий разработчику легко выявить проблемы во время разработки. Также *Python* является кроссплатформенным, что позволяет создавать приложения как для персонального компьютера, так и для мобильных устройств.

Стоит отметить, что *Python* является интерпретируемым ЯП, а это является существенным недостатком, поскольку приложения, написанные на данном языке, обладают низким быстродействием. Также у него имеется динамическая типизация, что подразумевает под собой написание краткого кода без объявления типа переменной. Данная особенность экономит время на

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		23

разработку и оптимизирует программный код, но вместе с тем приводит к ошибке времени выполнения (runtime error) и требует дополнительных тестов.

1.1.5.2 Java

Java является одним из самых распространенных и популярных ЯП. Первая версия появилась в 1996 году в компании *Sun Microsystems*, которая впоследствии была куплена корпорацией *Oracle*. Изначально он задумывался, как универсальный ЯП для решения различных задач.

За всё время существования *Java* превратилась из универсального языка в большую экосистему, объединяющую различные технологии. Также *Java* активно применяется для создания программного обеспечения для множества устройств, начиная от персональных компьютеров и заканчивая бытовой техникой. Ключевой особенностью языка *Java* является то, что его код сначала транслируется в специальный байт-код, независимый от платформы. Затем он выполняется виртуальной машиной *JVM* (*Java Virtual Machine*). В этом плане *Java* отличается от стандартных интерпретируемых языков, как *Python* или *Perl*, код которых сразу же выполняется интерпретатором.

Из преимуществ программирования на данном языке стоит отметить поддержку концепции ООП, позволяющий не только определить тип данных, но и функции, которые будут к нему применены. Кроме того, приложение, написанное на данном языке, можно использовать практически на любых платформах.

К недостаткам языка относят низкую скорость выполнения кода из-за компиляции и абстракции с помощью *JVM*, а также сложный для восприятия код ввиду перегруженности различными атрибутами.

1.1.5.3 C#

C# — современный объектно-ориентированный ЯП. Он позволяет разработчикам создавать разные типы безопасных и надежных приложений, выполняющихся в среде *.NET*. Благодаря этому данный язык подходит для создания и использования программных компонентов.

Главным архитектором *C#* был Андерс Хейлсберг (*Anders Hejlsberg*) — один из ведущих специалистов в области ЯП, получивший признание во всем мире. Изначально данный ЯП создавался, как прямой конкурент *Java*. Разница заключалась в том, чтобы в *C#* устранить проблемы, которые присутствовали в таких ЯП, как *C*, *C++*, *Delphi* и в особенности в *Java*.

Также *C#* связан со средой *.NET Framework* и имеет в наличии очень важные средства. Одно из них, *CLR*, представляет собой систему, которая управляет выполнением пользовательских программ [3]. Она является составной частью *.NET Framework*, которая делает программы переносимыми, поддерживает многоязыковое программирование и обеспечивает безопасность создаваемого приложения.

Данный язык используется для веб-разработке (в частности библиотека *ASP.NET Core*), машинного обучения (пример – сервис *ML.NET*) и разработки веб-сервисов (например, с применением инструмента *WCF* или *WPF*) [4]. Его преимущество заключается в использовании концепции ООП, как и в *Java*, что позволяет повысить скорость разработки, а также упростить восприятие кода. Кроме того, *C#* очень близок к синтаксису таких языков, как *C* и *C++*, что повышает порог вхождения разработчику ПО.

Вместе с тем у данного языка имеется ряд недостатков. Например, для того чтобы написать код на *C#*, требуется актуальная версия платформы *.NET* и ОС *Windows*. Связано это с тем, что данный язык является частью инфраструктуры корпорации *Microsoft*.

1.1.5.4 Итог

В качестве используемого языка программирования был выбран *C#*. Интерпретируемый язык *Python* не обеспечивает необходимую для решения задачи производительность, а *Java*, несмотря на широкий функционал, обладает низкой скоростью работы и сложным для понимания кодом, что негативно влияет на скорость разработки приложения.

1.1.6 Выбор СУБД

После выбора языка программирования необходимо определить, где будут храниться данные для веб-приложения. Для решения такой задачи разработчики создали мощный инструмент – базы данных.

На рынке присутствует большое количество программных продуктов, предоставляющих возможности для хранения и управления данными, такие как *Oracle*, *MySQL*, *Microsoft SQL*, *PostgreSQL* и др. В данной работе будут рассмотрены данные СУБД.

1.1.6.1 Oracle 12c

Oracle является лидером в использовании декларативного, непроцедурного подхода к проектированию баз данных и приложений. Технология *Oracle Server* считается одной из самых прогрессивных, мощных и надежных реляционных баз данных в мире [5].

Одной из важных особенностей данной среды является его тесная интеграция с *SQL*. Для выполнения *SQL*-запросов в программах с применением *Oracle* не требуется никакой промежуточной программной «прослойки» *ODBC* (*Open Database Connectivity*) или *JDBC* (*Java Database Connectivity*).

Также благодаря системе изолированных транзакций *Oracle* предоставляет надёжное хранение данных для разработчика. Но вместе с тем для работы с данным продуктом требуется серьёзный бюджет. Это означает, что данная база данных не подходит для решения задачи.

Преимущества:

- высокая скорость работы;
- постоянные обновления, добавляющие новый функционал в данный продукт;
- является лучшей БД по надёжности.

Недостатки:

- высокая стоимость использования для небольших компаний;
- высокая ресурсоёмкость БД, требующая мощный компьютер.

К сожалению, стоимость коммерческой версии и высокие аппаратные требования не позволяют использовать данную СУБД в данной разработке.

1.1.6.2 MySQL

MySQL - одна из самых популярных СУБД, используемых в разработке ПО. Является стандартом для веб-серверов под управлением ОС *Linux*. Включает в себя бесплатный пакет программ, однако новые версии выходят постоянно, расширяя функционал и улучшая безопасность.

MySQL позволяет применять встраиваемые приложения, хранилище данных и индексирование содержимого, программное обеспечение для доставки, высоконадежные системы с резервированием, обработку транзакций в реальном времени (*OLTP*) [6].

Преимущества:

- простая установка среды и большое количество инструментов для работы;
- высокая производительность БД;
- подробная документация, позволяющая разработчику тщательно разобраться в данном программном продукте.

Недостатки:

- платная для коммерческого использования;
- высокое потребление ресурсов компьютера даже при правильной настройке среды;
- низкая стабильность;
- большие промежутки времени между выходами новых стабильных версий.

Из-за своей низкой стабильности, которая может привести к критическому сбою в приложении, а также высокой стоимости использование данной СУБД в проекте нежелательно.

1.1.6.3 Microsoft SQL (MS SQL)

В коммерческой разработке чаще используется такая СУБД, как *Microsoft SQL*-сервер. Её движок работает на облачном сервере, а также на локальных серверах. Стоит отметить, что использование серверов в данной СУБД можно комбинировать.

Важной особенностью данной СУБД является *temporal data support* (временная поддержка данных), позволяющая отслеживать изменение данных. Кроме того, *Microsoft SQL*-сервер поддерживает *dynamic data masking* (динамическую маскировку данных), гарантирующая безопасность данных пользователей.

Преимущества:

- простота использования;
- высокая скорость работы СУБД;

Недостатки:

- высокое потребление ресурсов вычислительной машины даже при грамотной настройке среды;
- высокая цена для коммерческого пользования;
- проблемы с совместимостью на других платформах.

Как и в случае с СУБД *Oracle*, главными недостатками *MS SQL* является высокая стоимость и требовательность к ресурсам вычислительной машины. Следовательно, данная СУБД не подходит для решения задачи.

1.1.6.4 PostgreSQL

PostgreSQL —объектно-реляционная система управления базами данных, основанная на программе *Postgres*, Version 4.2 [7].

Postgres применяется для решения разных задач, начиная от научных и заканчивая производственными. Благодаря данной СУБД была создана система анализа финансовых данных, база данных наблюдений за астероидами, база данных медицинской информации. Отличие данной СУБД от остальных

заключается в том, что она поддерживает концепцию *ACID* – стандарт требования к транзакционным системам.

Преимущества:

- масштабное решение, позволяющее обрабатывать большие объёмы данных;
- большое количество доступных функций для бесплатной версии;
- возможность работы на других платформах;
- подробная документация, доступная на русском языке.

Недостатки:

- невысокая скорость работы;
- отсутствие некоторых функций в бесплатной версии.

1.1.6.5 Выводы

По результатам анализа СУБД наиболее подходящая для решения задачи является БД *PostgreSQL*. Она обеспечивает надёжное хранение больших данных, поддерживает концепт *ACID* и содержит необходимые функции для реализации микросервиса.

1.1.7 Выбор библиотеки для микросервиса

1.1.7.1 Библиотека WPF

WPF (англ. *Windows Presentation Foundation*) – библиотека, являющаяся частью экосистемы корпорации *Microsoft*. Она была создана в 2006-ом году и выпущена вместе с новой версией программной библиотеки *.NET 3.0* [8].

Изначально она могла работать только под управлением ОС *Windows Vista*, однако со временем она получила поддержку других ОС *Windows*. Ключевой особенностью данной библиотеки является работа с графикой. В отличие от *WinForms*, где за графику отвечали такие компоненты ОС *Windows*, как *User32* и *GDI+*, *WPF* использует *DirectX*.

Основное преимущество использования данной библиотеки заключается в том, что она предлагает широкие возможности для реализации множества функций в приложении, от мультимедийных приложений до полноценного веб-

приложения. Кроме того, *WPF* прекрасно взаимодействует со другими продуктами компании *Microsoft*, что обеспечивает вариативность создаваемых приложений.

Главный недостаток *WPF* заключается в его неэффективном потреблении ресурсов вычислительной машины. Приложение, написанное с его применением, может потреблять до 50% памяти, что может привести к замедлению работы системы. Кроме того, программу на *WPF* можно запускать только под управлением ОС *Windows*, а это противоречит требованиям к задаче. Поэтому данный вариант не будет использоваться.

1.1.7.2 Библиотека *WCF*

WCF (англ. *Windows Communication Foundation*) – библиотека корпорации *Microsoft*, предназначенная для разработки веб-приложений [9]. Была выпущена вместе с *WPF* и ОС *Windows Vista* в 2006-ом году и представляет собой шаблон для создания веб-приложений.

На основе данной библиотеки можно разворачивать самостоятельные веб-приложения, которые могут быть доступны на любой платформе. Кроме того, приложению, куда будет интегрирован данный сервис, требуется лишь адрес в сети. Например, *WCF*-приложение может быть легко интегрировано в *java*-клиент. Это означает, что данная технология подходит под требования задачи, поскольку на ней возможно развернуть независимый от остальных микросервис.

1.1.8 Выбор библиотеки для веб-клиента

1.1.8.1 Библиотека *Spring*

Spring framework – набор библиотек, специально созданная для написания веб-приложений на ЯП *Java*. В отличие от других платформ, *Spring* фокусируется на нескольких областях функционирования программы и предоставляет для них широкий выбор функций. Одно из главных преимуществ данной библиотеки – разделение компонентов и программного кода. Настройки компонентов вынесены в отдельный файл, что позволяет легко управлять

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		30

зависимостями и в случае изменения конфигурации облегчить изменения в проекте [10].

Данная библиотека позволяет создать код, который будет прост для дальнейшего тестирования, а также выбирать контейнер, которым разработчик будет пользоваться. Это способствует повышению надёжности приложения, а также его легковесности, так как в нём будут только необходимые модули.

Однако у библиотеки есть недостаток. Он заключается в том, что у него низкая скорость работы, которая может негативно сказаться на функционировании веб-приложения. Кроме того, Spring сложна для первичной настройки. Исходя из данных аргументов, данная библиотека не будет использоваться для разработки-веб-клиента.

1.1.8.2 Библиотека *Django*

Django – веб-библиотека, предназначенная для написания сайтов на *Python*. Его создали разработчики издания *Lawrence-Journal World*. Им понадобился сайт для публикаций новостей в интернете. Программисты Эдриан Головатый и Саймон Виллисон решили данную задачу и написали веб-приложение [11]. Позже они выложили его в открытый доступ. Значительную роль в успехе библиотеки сыграли известные сайты, которые использовали его в своих целях. Его используют *Pinterest*, *Dropbox* и газета *The Washington Post*. В настоящее время сообщество *Django* включает более 11 тыс. разработчиков из 166 стран мира.

Преимущество данной библиотеки состоит в присутствии типовых блоков. Например, блока авторизации или блок рассылки новостей. Это позволяет сэкономить время на разработку веб-клиента. Немаловажным преимуществом также является поддержка *ORM*, предоставляющая возможность работы с БД с минимальными трудозатратами.

Недостаток *Django* состоит в том, что приложение будет иметь монолитную структуру. Это означает, что разработчику в случае сбоя одного из модулей придётся искать альтернативные инструменты для решения задачи.

Кроме того, данная библиотека имеет сложность в плане поведения компонентов, не входящих в экосистему *Django*. В случае, если потребуется добавить новый компонент в приложение, то для этого потребуется много времени и усилий. Поэтому от использования библиотеки придётся отказаться.

1.1.8.3 Библиотека *ASP.NET Core MVC*

Библиотека *ASP.NET Core MVC* является частью платформы *ASP.NET Core*, его отличительная особенность - применение паттерна *MVC*, появившегося еще в конце 1970-х годов в компании *Xerox* [12].

Преимуществом использования данной библиотеки является то, что он упрощает в ряде ситуаций и сценариев организацию и создание приложений, особенно в больших.

Также преимуществом является использование такого инструмента, как *Razor Pages*. В отличие от своих конкурентов в лице *JavaScript* и *React* он использует возможности ЯП *C#*, а это существенно упрощает разработку веб-приложения под различные нужды заказчика [13]. Также в технологии *ASP.NET Core* не требуются зависимости от ядра систем на базе *Windows*, что делает данную библиотеку кроссплатформенной. Кроме того, в *ASP.NET Core* присутствует внедрение зависимостей, которое упрощает работу с дополнительными модулями приложения, а также высокая производительность.

Из недостатков стоит отметить документацию библиотеки. Она пока не такая обширная, как у *ASP.NET MVC*, но *Microsoft* активно увеличивает количество описаний и примеров использования с привлечением сторонних специалистов. Проблема нивелируется из-за открытого исходного кода, ведь теперь каждый может найти любой интересующий его компонент и посмотреть логику работы.

1.1.8.4 Выводы

В ходе выбора библиотеки было принято решение выбрать библиотеку *ASP.NET Core MVC*. Она обладает большей скоростью, чем *Spring* и *Django*, и

более удобной настройкой по сравнению с библиотекой *Spring*. Также у *ASP.NET Core MVC* присутствует возможность настройки графической части приложения с минимальными трудозатратами в отличие от вышеприведённых библиотек.

1.1.9 Описание архитектуры *REST API*

REST — архитектурная концепция создания распределенных систем на основе гиперсред. Модель *REST* не зависит от каких-либо протоколов и не требует привязки к *HTTP*. Наиболее распространенные реализации данной концепции используют протокол *HTTP* в качестве основного протокола приложения. Она была предложена в 2000 году Роем Филдингом, который продвигал архитектурный подход к разработке веб-служб — передачу репрезентативного состояния [14].

Основное преимущество при использовании *REST* с протоколом *HTTP* заключается в применении открытых стандартов и отсутствии необходимости в определенной реализации *API*. Например, веб-службу *REST* можно написать в *ASP.NET*, а клиентские приложения могут использовать любой язык или набор инструментов, позволяющие создавать *HTTP*-запросы и анализировать *HTTP*-ответы. Обмен информации происходит с использованием таких форматов *JSON* и *XML*.

Наиболее часто выполняемыми операциями в *REST API* являются:

- *GET* – используется для запроса содержимого из страницы веб-ресурса;
- *POST* – используется для передачи данных веб-ресурсу. Например, для передачи логина и пароля в систему для авторизации;
- *PUT* – используется для передачи данных веб-ресурсу, указанному в запросе. Используется для смены значения веб-ресурса;
- *PATCH* – используется только для создания или указания ресурса документа;
- *DELETE* – удаляет указанный в запросе веб-ресурс.

В микросервисе под управлением *WCF* возможно использование *REST API* с помощью использования атрибута *WebInvoke*. Он содержит в себе два параметра: тип запроса и текст *url*, который будет указан при выполнении запроса.

Пример реализации веб-запроса представлен ниже.

Листинг 1. Пример веб-запроса.

```
[OperationContract]
[WebInvoke(Method = "GET", UriTemplate = "FilmFindInfo?id={id}")]
string FilmFindInfo (int id);
```

Первый атрибут, *OperationContract*, указывает на то, что служба *WCF* может использовать данный метод в приложении в веб-запросах. Следующий атрибут, *WebInvoke*, отвечает за реализацию веб-запроса *REST* в приложении. Первый аргумент, *Method*, указывает тип запроса, в нашем случае это *GET*. Второй аргумент указывает часть ссылки, которая будет задействована во время реализации запроса. В нашем случае это *FilmFindInfo?id={id}* – операция поиска фильма по уникальному идентификатору.

Стоит также отметить, что веб-запросы, отправляемые с применением архитектуры *REST API*, увеличивают нагрузку на веб-сервер, которая растет вместе с количеством запросов. Поэтому важно избегать создания *API* с множественными операциями ввода-вывода, предоставляющие большое количество данных.

1.1.10 Выводы

В ходе выбора стека технологий был выбран ЯП *C#*, позволяющий написать быстрые приложения с понятным для разработчика кодом, реляционная СУБД *PostgreSQL*, обеспечивающая надёжное хранение данных и концепцию *ACID*, библиотека *WCF*, позволяющая легко интегрировать микросервисы, разработанные на ней, в другие системы, а для веб-клиента –

библиотека *ASP.NET Core MVC*, предоставляющая разработчику необходимые инструменты для быстрой разработки веб-ресурса.

1.2 Реализация

В данной части приведены результаты разработки микросервиса с применением библиотеки *WCF*, *REST API* и *NPGSQL*, а также реализация клиентского приложения с применением библиотеки *ASP.NET Core MVC* и *Razor Pages*.

Разработку микросервиса можно поделить на следующие этапы:

- разработка структуры БД;
- разработка основного микросервиса;
- разработка *REST API*;
- разработка клиентского приложения для взаимодействия с микросервисом.

1.2.1 Проектирование базы данных

Прежде, чем приступить к разработке программы, необходимо описать структуру БД, которая будет использоваться в микросервисе. В качестве основной СУБД была выбрана *PostgreSQL*. Ниже представлена описание разработанной БД. Таблица содержит описание структуры таблицы «*FilmTable*».

Таблица 1 - Структура таблицы «*FilmTable*»

Название поля	Описание поля	Тип данных
id	Идентификатор фильма	(PK) Integer
FilmName	Название фильма	Text
FilmDescription	Описание фильма	Text
FilmYear	Год выхода фильма	Integer
FilmRate	Рейтинг фильма	Text

Продолжение таблицы 2

Название поля	Описание поля	Тип данных
ImageLink	Ссылка на постер	Text

Для настройки подключения к БД мы будем использовать библиотеки *EntityFrameworkCore* и *NPGSQL*. Для соединения с БД нужно использовать следующие данные: название необходимого хоста, его порт, имя целевой БД, логин и пароль локальной учётной записи *PostgreSQL*. Механизм подключения к БД вынесен в файл *ApplicationContext.cs*, где происходит как соединение к БД, так и её инициализация.

Листинг 2. Класс подключения к БД *ApplicationContext.cs*.

```
public class ApplicationContext : DbContext
{
    public DbSet<FilmEntity> FilmTable { get; set; }
    public ApplicationContext()
    {
        Database.EnsureCreated();
    }

    protected override void
    OnConfiguring(DbContextOptionsBuilder optionsBuilder) {

        optionsBuilder.UseNpgsql("Host=localhost;Port=5432;Database=FilmDB;Username=test;Password=test");

    }
}
```

Класс *PremierEntity* имеет 6 полей: *id*, *FilmName*, *FilmDescription*, *FilmYear*, *FilmRate*, *ImageLink*. Для связи с БД используется класс *FilmRepository*, который имеет возможность делать все операции с записями.

Листинг 3. Интерфейс класса взаимодействия с БД *FilmRepository*

```
public interface IFilmRepository
{
    void FilmAddInfo(FilmEntity film);

    void FilmRemoveInfo(int id);
}
```

```

void FilmUpdateInfo(FilmDTO film, int id);

List<FilmEntity> FilmReadAllInfo();

FilmEntity FilmFindInfo(int userId);
}

```

1.2.2 Разработка основного микросервиса

1.2.2.1 Разработка основной части

Основная часть программы будет состоять из следующих компонентов:

- контроллера;
- сервиса;
- репозитория.

Контроллер передаёт основные данные, полученные в результате запроса пользователя. Она отправляет их в DTO-модель (англ. *Data Transfer Object*) и передаёт их в сервис. После сервис преобразовывает DTO-объект в объект, предназначенный для записи в БД. После преобразования объект передаётся в репозиторий, где производятся основные операции над объектами, а именно:

- создание фильма;
- удаление фильма;
- обновление фильма;
- поиск фильма в БД по идентификатору;
- вывод списка всех фильмов.

1.2.2.2 Разработка инструмента для парсинга сайтов о кино

Для того, чтобы программно получить данные с сайта, необходимы дополнительные библиотеки для микросервиса. Таковыми являются *AngleSharp* и *HtmlAgilityPack*. Первая позволяет подгружать страницу сайта и преобразовывать его в текстовый файл, вторая – производить поиск элементов в данном файле и передавать их основной программе.

Механизм работает следующим образом: сначала настраивается конфигурация для браузера. Затем программа открывает этот браузер и

подгружает туда адрес сайта. После загрузки библиотека в асинхронном режиме подгружает данные. После загрузки сайт становится доступным для изучения разработчиком.

Данный инструмент полезен тем, что он позволяет получить данные из браузера без использования графического интерфейса, как в библиотеке *Selenium*, основываясь только на языке запросов. Тем не менее, схожесть с библиотекой имеется в механизме поиска данных на сайте. И в *Selenium*, и в вышеперечисленных библиотеках используется похожий механизм поиска элементов по локаторам.

В данной работе производится парсинг таких крупных кинопорталов, как Кинопоиск и IMDb. В данных сайтах производится поиск свежих фильмов, которые скоро выйдут в прокат. Реализация поиска данных на сайте IMDb отличается от Кинопоиска использованием различных локаторов для поиска и преобразования информации о фильмах в отдельный объект.

Пример парсинга данных с сайта IMDb приведён ниже в листинге программы в приложении Б.

1.2.3 Разработка REST API

В данной работе будут использоваться такие операции REST API, как POST и GET.

Все операции будут указаны в основном интерфейсе микросервиса *IFilmMicroService* с применением декоратора *WebInvoke*, позволяющий указывать тип операции и url, по которому она будет осуществлена.

Листинг 4. Интерфейс *IFilmMicroService.cs*

```
public interface IFilmMicroService
{
    [OperationContract]
    [WebInvoke(Method = "POST", UriTemplate =
        "FilmAddInfo?filmName={filmName}&filmDescription={filmDescription}
        &filmYear={filmYear}&filmRate={filmRate}")]
    string FilmAddInfo(string filmName, string
        filmDescription, int filmYear, string filmRate, string
        imageLink);
}
```

```

        [OperationContract]
        [WebInvoke(Method = "POST", UriTemplate =
"FilmRemoveInfo?id={id}")]
        string FilmRemoveInfo(int id);

        [OperationContract]
        [WebInvoke(Method = "POST", UriTemplate =
"FilmUpdateInfo?filmName={filmName}&filmDescription={filmDescript
ion}&filmYear={filmYear}&filmRate={filmRate}&id={id}")]
        string FilmUpdateInfo(string filmName, string
filmDescription, int filmYear, string filmRate, int id, string
imageLink);

        [OperationContract]
        [WebInvoke(Method = "GET", UriTemplate =
"FilmReadAllInfo?all")]
        List<FilmEntity> FilmReadAllInfo();

        [OperationContract]

[WebInvoke(Method = "GET", UriTemplate = "FilmFindInfo?id={id}")]
        FilmEntity FilmFindInfo(int id);

        [OperationContract]

[WebInvoke(Method = "GET", UriTemplate = "PremiersList?All")]
        List<PremierDTO> PremiersList();

        [OperationContract]
        [WebInvoke(Method = "GET", UriTemplate =
"PremiersInRussiaList?All")]
        List<PremierDTO> PremiersInRussiaList();

    }

```

Однако добавление специального декоратора недостаточно для того, чтобы с приложением могли взаимодействовать пользователи. Для этого необходимо добавить в конфигурацию приложения *Web.config* разрешение на передачу данных по протоколу *HTTP*, а также на передачу данных в формате *XML* и *JSON*. После добавления микросервис сможет обмениваться данными по протоколу *HTTP* в формате *XML* и *JSON*.

Листинг 5. Разрешение на передачу данных по протоколу *HTTP*

```
<serviceMetadata httpGetEnabled="true" httpsGetEnabled="true" />
```

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		39

Листинг 6. Разрешение на передачу данных в формате *JSON* и *XML*.

```
<endpoint address="json" binding="webHttpBinding"
behaviorConfiguration="WebJson" bindingConfiguration="WebJson"
name="WebJson"
bindingNamespace="http://localhost/FilmPortal.ServiceWCF"
contract="FilmPortal.ServiceWCF.IFilmMicroService" />

<endpoint address="xml" binding="webHttpBinding"
behaviorConfiguration="WebXML" bindingConfiguration="WebXML"
name="WebXML"
bindingNamespace="http://localhost/FilmPortal.ServiceWCF"
contract="FilmPortal.ServiceWCF.IFilmMicroService" />
```

1.2.4 Разработка *web*-приложения

После создания микросервиса необходимо создать веб-клиент, который смог бы взаимодействовать с ним. В качестве инструмента была выбрана библиотека на *C# ASP.NET Core MVC*. Она позволяет создать веб-приложение без тяжёлых затрат и возможность взаимодействовать с другими сервисами.

1.2.4.1 Интеграция микросервиса в клиентское приложение

Важнейшей частью работы клиентского приложения является его взаимодействие с микросервисом. Поскольку микросервис написан с применением *WCF*, то процесс интеграции займёт большого количества времени.

Для того, чтобы интегрировать *WCF*-микросервис в клиентское приложение *ASP.NET Core*, потребовались следующие действия. Для этого необходимо нажать правой кнопкой мыши по проекту, далее «Добавить» и выбрать «Ссылка на службу». Будет отображено окно, представленное на рисунке 9.

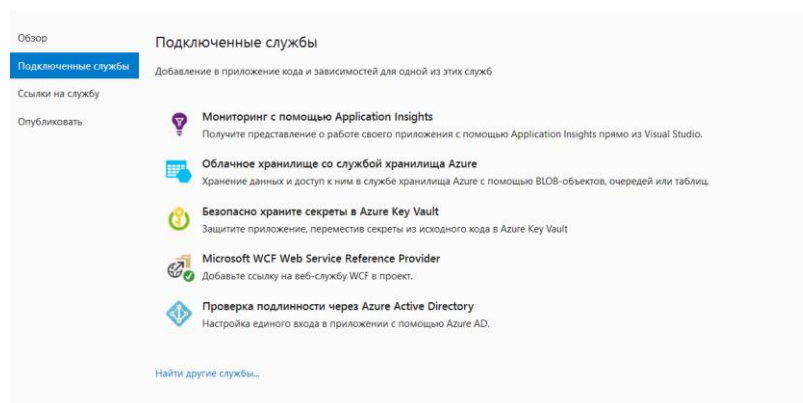


Рисунок 9 - Подключение службы

Далее необходимо выбрать пункт «*Microsoft WCF Service Reference Provider*» и скопировать ссылку адреса размещения микросервиса. После того, как ссылка была скопирована, нажать кнопку «Перейти», в результате которого отображается требуемый нам микросервис. Результат поиска представлен на рисунке 10.

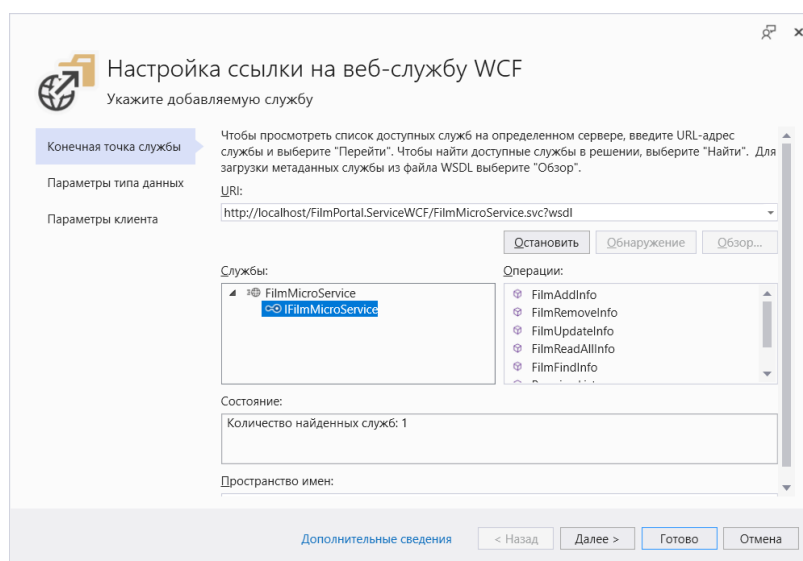


Рисунок 10 - Обнаруженный микросервис

Далее мы нажали кнопку «Готово». В результате данной операции в проект была добавлена ссылка на микросервис, позволяющая взаимодействовать непосредственно с ним в клиентском приложении.

1.2.4.2 Разработка контроллера

ASP.NET Core MVC предоставляет инструменты для работы с контроллером. Он взаимодействует непосредственно с веб-страницей, перехватывая оттуда данные. Основным контроллером в веб-приложении является *HomeController*. Он отвечает за бизнес-логику, которая находится в данном приложении. В него включены все базовые операции для работы с микросервисом: создание фильма, удаление фильма, обновление фильма, просмотр всех записей фильма, просмотр премьер в мире, просмотр премьер в России, контактная страница, страница отдельной премьеры в мире и страница отдельной премьеры в России. Пример метода из контроллера приведён в Листинге 6 ниже.

Листинг 6. Пример метода из *HomeController.cs*:

```
public IActionResult AllFilmsPage(string name, int page = 1)
{
    int pageSize = 3;
    var items = _filmList;
    var count = items.Length;
    if (!string.IsNullOrEmpty(name))
    { items = items.Where(n =>
n.FilmName.Contains(name)).ToArray(); }
    PageViewModel pageViewModel = new PageViewModel(count, page,
pageSize);
    IndexViewModel viewModel = new IndexViewModel
    {
        PageViewModel = pageViewModel,
        Films = items
    };
    return View(viewModel); }
```

Данный метод отвечает за отображение списка всех фильмов в табличном виде. В нём происходит обработка данных, полученных в результате взаимодействия с микросервисом, они преобразовываются в классе *IndexViewModel*. Затем экземпляр данного класса передаётся в конструктор для frontend-части, где происходит их отображение.

1.2.4.3 Разработка Web-страниц

Для того, чтобы отобразить данные на клиенте, используется библиотека *Razor Pages*. Она позволяет использовать язык вёрстки HTML совместно с ЯП *C#*, что облегчает разработку.

Всего представлено 10 страниц в веб-приложении в формате *.cshtml*: домашняя страница, создание фильма, удаление фильма, обновление фильма, просмотр всех записей фильма, просмотр премьер в мире, просмотр премьер в России, контактная страница, страница отдельной премьеры в мире и страница отдельной премьеры в России.

Также на всех страницах присутствует верхняя панель навигации по функциям, что облегчает работу с сайтом. Пример страницы написанной с применением данной библиотеки, представлена ниже в листинге Полный листинг страниц приведён в Приложении Б.

Листинг 7. Пример страницы на *Razor Pages*.

```
@{ViewData["Title"] = "Страница премьеры";

}

<title>0 премьер</title>
<div>
    <form method="post" action="">
        <input type="hidden" name="Id" value="@ViewBag.Id">
        <h1>@ViewBag.Name</h1>
        @if (@ViewBag.ImageLink.Trim().StartsWith("http"))
        {
            <a href="@ViewBag.ImageLink" target="_blank"> </a>
        }
        else
        {
            <h4><em>Пока нет постера</em></h4>
        }
        <h4>Режиссёр: @ViewBag.Director</h4>
        <h4>@ViewBag.Description</h4>
        <h4>Год выхода: @ViewBag.Year</h4>
        <a href="/Home/Premiers">Назад</a>
    </form>
</div>
```

1.2.5 Взаимодействие классов

В ходе разработки микросервиса и веб-приложение были созданы:

- класс-контроллер – обрабатывает поступающие от клиента запросы;
- классы-сущности – отвечают за удобное взаимодействия с реляционными моделями путем представления их в виде объекта;
- транспортные классы – обеспечивают конвертацию данных из в объект или обратно для передачи клиенту;
- классы-сервисы – классы, которые выполняют основную часть функционала обработки пакета. Каждый класс-сервис обрабатывает свою секцию.

UML-диаграмма классов микросервиса предоставлена на рисунке 11.

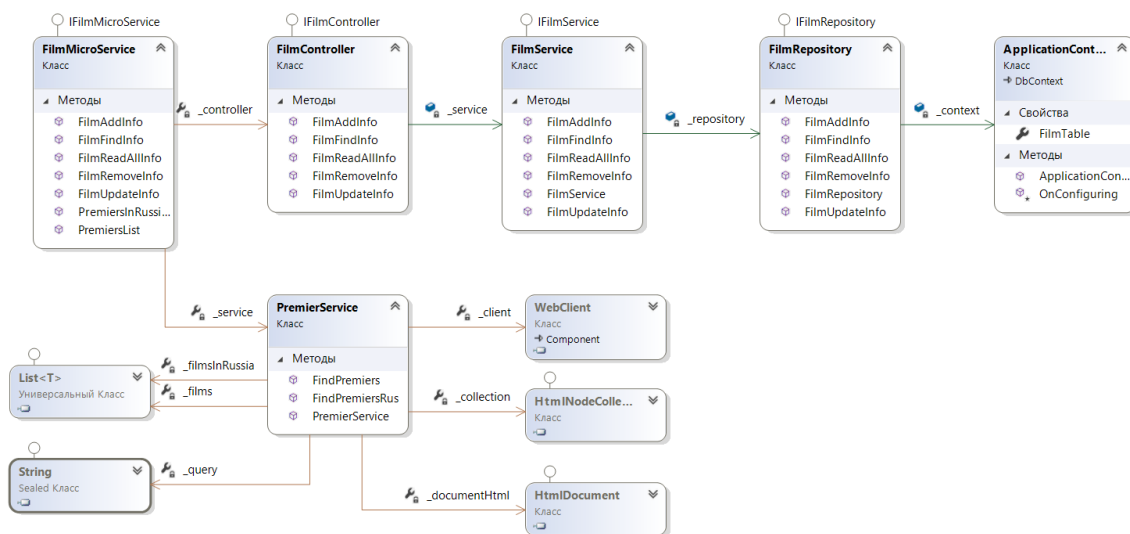


Рисунок 11 - UML-диаграмма микросервиса

UML-диаграмма, представляющая структуру клиентского приложения, представлена на рисунке 12.

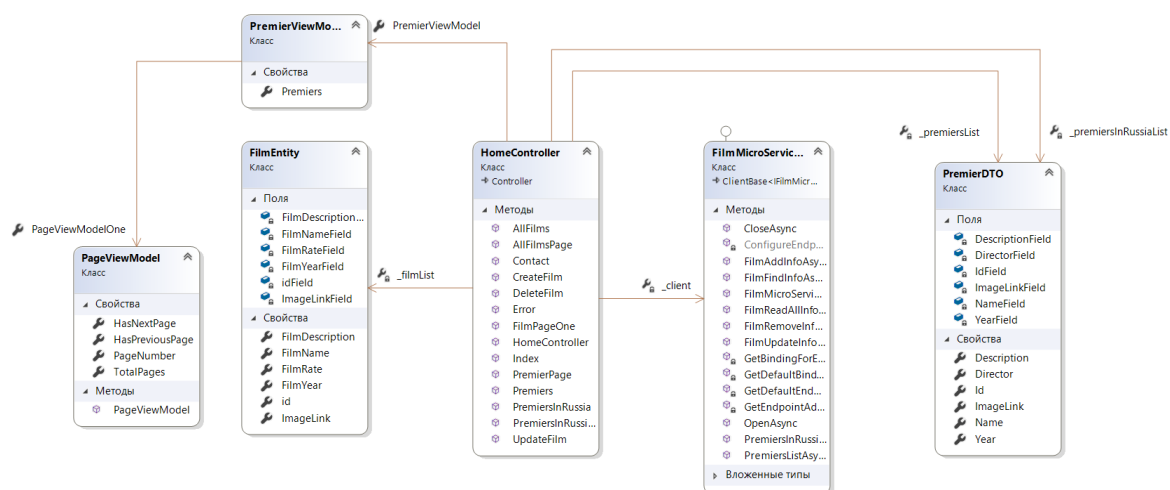


Рисунок 12 - UML-диаграмма клиентского приложения

1.3 Результат работы приложения

1.3.1 Результат работы приложения с использованием веб-клиента

Пользоваться приложением достаточно просто. Для этого необходимо перейти на главную страницу клиентского веб-приложения. На рисунке 13 представлена главная страница сайта.

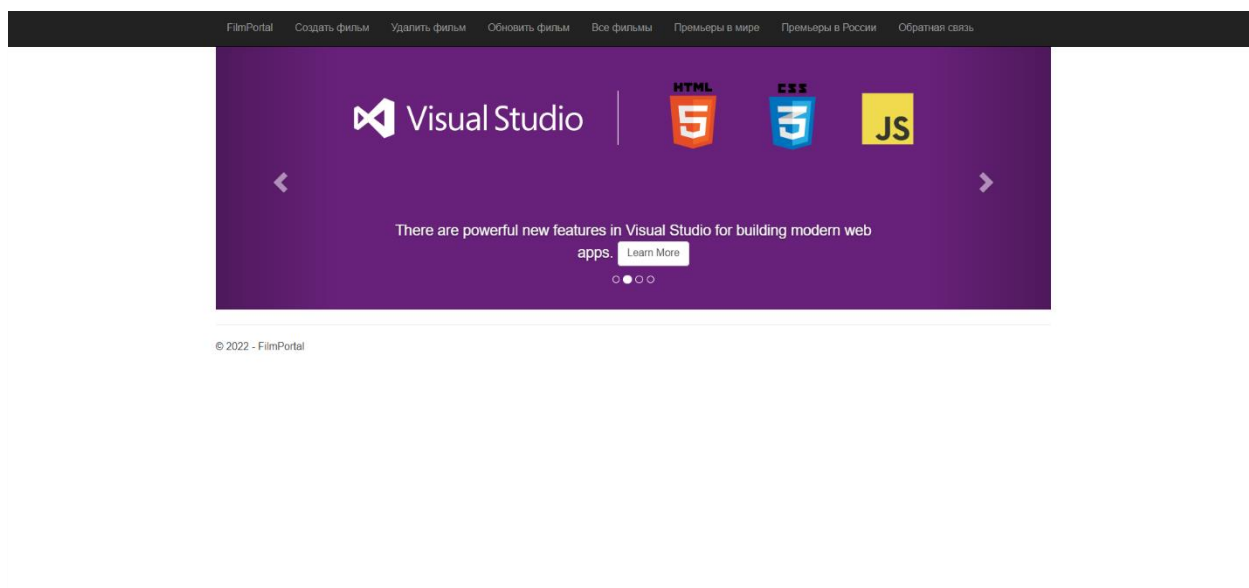
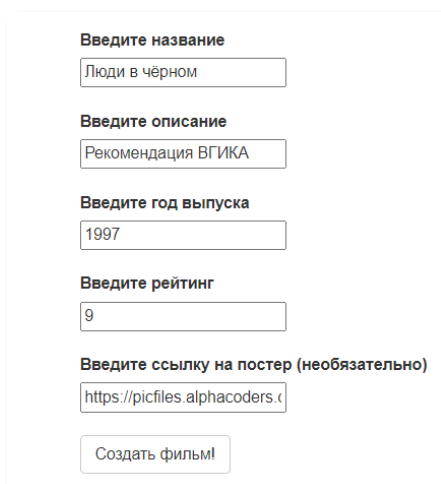


Рисунок 13 - Главная страница

В верхней панели будут представлены функции, предоставляемые микросервисом: создание, удаление, обновление и чтение записей о фильмах, а также просмотр предстоящих премьер.

Для того, чтобы создать запись, необходимо перейти во вкладку «Создать фильм». Пользователю будет предложено ввести название фильма, его описание, год выхода, рейтинг и постер (если имеется соответствующая ссылка). На рисунке 14 представлена форма ввода данных.



Введите название
Люди в чёрном

Введите описание
Рекомендация ВГИКА

Введите год выпуска
1997

Введите рейтинг
9

Введите ссылку на постер (необязательно)
<https://picfiles.alphacoders.c>

Создать фильм!

Рисунок 14 - Поле ввода для новой записи

После ввода данных клиент отправит их на микросервис. После выполнения операции он вернёт клиенту сообщение о её статусе. На рисунке 15 указано, что фильм добавлен и выведено соответствующее сообщение.

Фильм добавлен

Введите название

Введите описание

Введите год выпуска

Введите рейтинг

Введите ссылку на постер (необязательно)

Создать фильм!

Рисунок 15 - Фильм добавлен

В БД будет создана соответствующая запись. На рисунке 16 представлена БД с добавленной записью.

	id [PK] integer	FilmYear integer	FilmName text	FilmDescription text	FilmRate text
1	2	2017	F7	We are strong like a family	7
2	3	1999	The Matrix	No one can be told, what the Matrix is. You have to see it for yourself.	8.5
3	6	1976	Taxi Driver	De Niro performance	9
4	9	2013	The Wolf of Wall Street	Best film of 2013	8
5	10	1987	Evil Dead II	Bruce Campbell performance	7
6	11	1997	Men in Black	Best comedy of 1997	8
7	12	1997	Люди в чёрном	Рекомендация ВГИКА	9

Рисунок 16 – БД после добавления записи

Также предусмотрена возможность просмотреть все добавленные фильмы. Для этого необходимо перейти во вкладку «Все фильмы». На рисунке 17 представлен список всех фильмов, полученных в результате обращения к микросервису.

Все фильмы

Поиск по имени:

ID	Название	Рейтинг	Год	Информация
2	F7	7	2017	О фильме
3	The Matrix	8.5	1999	О фильме
9	The Wolf of Wall Street	8	2013	О фильме
10	Evil Dead II	7	1987	О фильме
11	Men in Black	8	1997	О фильме
6	Taxi Driver	9	1976	О фильме
12	Люди в чёрном	9	1997	О фильме

Рисунок 17 – Список всех фильмов

На странице также присутствует возможность фильтрации записей по имени фильма. Данная функция особенно полезна в тех случаях, когда необходимо найти нужный фильм в огромном списке. На рисунке 18 представлена данная возможность.

Все фильмы				
Поиск по имени: <input type="text"/> <input type="button" value="Filter"/>				
ID	Название	Рейтинг	Год	Информация
6	Taxi Driver	9	1976	О фильме

Рисунок 18 – Список всех найденных фильмов

Кроме того, имеется возможность просмотреть запись о том или ином фильме подробнее. Для этого пользователь должен нажать рядом с выбранной записью ссылку «О фильме». После нажатия он будет перенаправлен на страницу записи с её описанием. На рисунке 19 представлен пример подробного отображения записи.

Помимо просмотра записей о фильмах пользователь может удалить их или обновить содержимое записи. При удалении записи ему необходимо знать уникальный идентификатор записи. На рисунке 20 представлена функция удаления записи.



Рисунок 19 – Пример подробного отображения записи

Рисунок 20 – Удаление записи по идентификатору

Во время операции идентификатор передаётся в микросервис, который использует полученные данные для удаления записи из БД. После завершения операции страница обновляется и появляется сообщение об успешном удалении. На рисунке 21 представлено данное сообщение.

Помимо этого, предусмотрена возможность обновления записи. Для этого пользователю необходимо знать её уникальный идентификатор и перейти во вкладку «Обновить фильм». После ввода данных пользователь должен нажать кнопку «Обновить фильм!».

На рисунке 22 представлена страница обновления фильма.

Рисунок 21 – Результат удаления записи

Рисунок 22 – Форма для обновления фильма

После нажатия данные будут переданы в контроллер клиентского приложения, а затем – непосредственно в микросервис. После выполнения операции на странице будет отображено сообщение о том, что запись обновлена. Оно представлено на рисунке 23.

Фильм обновлён

Enter film id

Enter film name

Enter film description

Enter film year

Enter film rate

Enter film poster link

Обновить фильм!

Рисунок 23 – Результат обновления данных

В БД также будет обновлена запись. На рисунке 24 представлена обновлённая БД.

	id [PK] integer	FilmYear integer	FilmName text	FilmDescription text	FilmRate text
1	2	2017	F7	We are strong like a family	7
2	3	1999	The Matrix	No one can be told, what the Matrix is. You have to see it for yourself.	8.5
3	6	1976	Таксист	Классика кинематографа	10
4	9	2013	The Wolf of Wall Street	Best film of 2013	8
5	10	1987	Evil Dead II	Bruce Campbell performance	7
6	12	1997	Люди в чёрном	Рекомендация ВГИКА	9

Рисунок 24 – БД после обновления записи

Кроме работы с записями пользователь может просматривать список будущих фильмов. Для этого он должен перейти во вкладку «Премьеры в мире». Клиент отправит запрос микросервису на получение списка премьер. На рисунке 25 представлен список предстоящих премьер.

Премьеры в мире

Найти по имени:

Имя	Режиссёр	Год	Информация
Мир Юрского периода: Господство	Colin Trevorrow	2022	О фильме
Базз Лайтер	Angus MacLane	2022	О фильме
Спайдерхед	Joseph Kosinski	2022	О фильме
Brian and Charles	Jim Archer	2022	О фильме
Элвис	Baz Luhrmann	2022	О фильме
Чёрный телефон	Scott Derrickson	2021	О фильме

© 2022 - FilmPortal

Рисунок 25 – Список предстоящих премьер

Как и в случае с обычными записями, пользователь может просматривать подробное описание предстоящего фильма по нажатию на ссылку «О фильме». На рисунке 26 представлена страница одной из предстоящих премьер.

Мир Юрского периода: Господство



Режиссёр: Colin Trevorrow

Four years after the destruction of Isla Nublar, dinosaurs now live--and hunt--alongside humans all over the world. This fragile balance will reshape the future and determine, once and for all, whether human beings are to remain the apex predators on a planet they now share with history's most fearsome creatures in a new Era.

Год выхода: 2022

Рисунок 26 – Пример подробного отображения записи

Также имеется возможность посмотреть премьеры в России. Для этого необходимо перейти во вкладку «Премьеры в России». На рисунке 27 представлен список предстоящих премьер в России.

Премьеры в России

Поиск по имени:

Filter

Название	Режиссёр	Год	Информация
Артек. Большое путешествие	Армен Ананикян	2022	О фильме
Своя война. Шторм в пустыне	Алексей Чадов	2022	О фильме
Мой папа – вождь	Егор Кончаловский	2022	О фильме
Аманат	Антон Сиверс	2022	О фильме
Три кота и море приключений	Дмитрий Высоцкий	2022	О фильме
Коати. Легенда джунглей	Родриго Перес-Кастро	2022	О фильме
Остров призраков	Расселл Оуэн	2022	О фильме
Вендетта. Банды Атланты	Джаред Кон	2022	О фильме
Манюня в кино	Арман Марутян	2022	О фильме
Приключения Буратино	Иван Иванов-Вано	2022	О фильме

Рисунок 27 – Список предстоящих премьер в России

Как и в случае с остальным функционалом, пользователь также может просматривать подробное описание фильма. На рисунке 28 представлена одна из таких страниц.

Три кота и море приключений



Режиссёр: Дмитрий Высоцкий

Releases in Russia

Год выхода: 2022

[Назад](#)

Рисунок 28 – Пример подробного отображения записи

2 Безопасность жизнедеятельности оператора ЭВМ

Безопасность жизнедеятельности (БЖД) – система знаний, обеспечивающая безопасность человека в производственной и непроизводственной среде.

Как и всякая наука, БЖД имеет свои цели, задачи и принципы, используемые для решения задач.

Её цель исходит из определения этой науки и представляет собой достижение безопасности в среде обитания. Безопасность человека определяется отсутствием аварий, стихийных бедствий и других природных явлений и опасных факторов, вызывающих травмы или резкое ухудшение здоровья, вредных факторов, вызывающих заболевание человека и снижение его работоспособности. Ниже приведены требования, которые должны соблюдаться при использовании программы.

2.1 Требования к помещениям для работы с ЭВМ

Согласно СанПин 2.2.2/2.4.1340-03 предъявляются следующие требования к рабочим помещениям:

- естественное и искусственное освещение должно соответствовать требованиям действующей нормативной документации. Это означает, что сочетание двух видов освещения должно регламентироваться на предприятии, использующие ЭВМ;

- эксплуатация ПЭВМ в помещениях без естественного освещения допускается только при наличии расчетов, обосновывающих соответствие нормам естественного освещения и безопасность их деятельности для здоровья работающих. В случае неправильных расчётов у работника может ухудшиться зрение вследствие длительной работы при неправильном освещении;

- также рабочее место согласно регламенту должно быть оборудовано заземлением во избежание производственных травм.

2.2 Требования к микроклимату во время работы с ПЭВМ

Для того, чтобы оператору было комфортно работать в помещении, необходимо позаботиться и о температуре, которая будет сопровождать работника во время работы с ПЭВМ. СанПин указывает следующие требования к микроклимату: температура должна сохраняться в пределах 19-21 градуса Цельсия.

Таблица 3 – Требования к температуре

Температура, град. С	Относительная влажность, %	Абсолютная влажность, г/м3	Скорость движения воздуха, м/с
19	62	10	< 0,1
20	58	10	< 0,1
21	55	10	< 0,1

Кроме того, содержание вредных химических веществ в воздухе производственных помещений, в которых работа с использованием ПЭВМ является вспомогательной, не должно превышать предельно допустимых концентраций вредных веществ в воздухе рабочей зоны в соответствии с действующими гигиеническими нормативами.

Заключение

В ходе выполнения данной работы был разработан микросервис учёта сведений о фильмах на языке *C#* с использованием библиотеки *WCF*. Был получен опыт разработки масштабируемых микросервисов и разработки веб-приложений на *ASP.NET Core MVC*.

При реализации поставленной задачи были получены следующие результаты:

- реализован механизм обмена информацией с БД;
- реализован парсинг данных с сайтов Кинопоиск и *IMDB*;
- разработано веб-приложение для тестирования микросервиса.

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		56

Список использованных источников

1. Якоб Нильсен, Хоа Лоранжер. *Web-дизайн: удобство использования Web-сайтов.* : Пер. с англ. — М. : ООО «И.Д. Вильямс», 2009. — 368 с. + 8 с. цв. ил. — Парал. тит. англ. Режим доступа: https://itexts.net/files/get_book.php?f=pdf&file_name=nilsen-yakob-web-dizayn-udobstvo-ispolzovaniya-web-saytov-224774&e=pdf.
2. Дарья Сидорова. 4 типа архитектуры программного обеспечения. IT-портал «Medium» [Электронный ресурс]: <https://medium.com/nuances-of-programming/4-типа-архитектуры-программного-обеспечения-917133174724> (дата обращения: 10.05.2022).
3. Шилдт, Герберт. Полный справочник по C#. : Пер. с англ. — М. : Издательский дом “Вильямс”, 2006: Режим доступа: https://sharpened.ucoz.ru/lib/polnyj_spravochnik_po_c-gerbert_shildt-2006.pdf.
4. Ведение в C#. Язык C# и платформа .NET. [Электронный ресурс]: Режим доступа: <https://metanit.com/sharp/tutorial/1.1.php> (дата обращения: 12.05.2022).
5. Владимир Драч. Сравнение современных СУБД. 30.05.2017 [Электронный ресурс]: Режим доступа: <https://drach.pro/blog/hi-tech/item/145-db-comparison> (дата обращения: 14.05.2022).
6. Шварц Б., Зайцев П., Ткаченко В. *MySQL по максимуму.* 3-е изд. - СПб.: Питер, 2018: Режим доступа: https://vk.com/doc288422991_525288529?hash=xCmIjzmniCPZhN4mM8PSPG5tX9PkfJdJYOJmLR7YGJH&dl=cgfnUgrCRGDdILzrqvEYnHU3N8WbaMl6I6BEUiZyzKD (дата обращения: 14.05.2022).
7. Документация к PostgreSQL 31.7 Перевод на русский язык —The PostgreSQL Global Development Group, 2015-2022 гг.: Режим доступа: <https://postgrespro.ru/media/docs/postgresql/13/ru/postgres-A4.pdf>.
8. Руководство по классическим приложениям (WPF .NET). Официальная документация компании Microsoft [Электронный ресурс]:

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		57

<https://docs.microsoft.com/ru-ru/dotnet/desktop/wpf/overview/?view=netdesktop-6.0> (дата обращения: 14.05.2022).

9. Основные понятия *Windows Communication Foundation*. Официальная документация компании *Microsoft* [Электронный ресурс]: <https://docs.microsoft.com/ru-ru/dotnet/framework/wcf/fundamental-concepts> (дата обращения: 15.05.2022).

10. Что такое *Spring*? Обзор фреймворка *Java*. IT-портал «*Patches*» [Электронный ресурс]: <https://oracle-patches.com/coding/chto-takoe-spring-obzor-frejmworka-java#h2-evolyutsiya-spring-framework> (дата обращения: 15.05.2022).

11. Почему *Django* – лучший фреймворк для разработки веб сайтов. [Электронный ресурс]: Режим доступа: <https://ru.hexlet.io/blog/posts/pochemu-django-luchshiy-freymvork-dlya-razrabotki-saytov> (дата обращения: 15.05.2022).

12. Общие сведения об *ASP.NET Core*. Официальная документация компании *Microsoft* [Электронный ресурс]: <https://docs.microsoft.com/ru-ru/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-5.0> (дата обращения: 15.05.2022).

13. Макаров О.С., Щенникова Е.В. Сравнительный анализ технологии *ASP.NET Core*. ФГБОУ ВО «НИ МГУ им. Н.П. Огарёва» — <https://cyberleninka.ru/article/n/sravnitelnyy-analiz-tehnologii-asp-net-core/viewer> (дата обращения: 17.05.2022).

14. Проектирование Веб-*Api Restful*. Официальная документация компании *Microsoft* [Электронный ресурс]: <https://docs.microsoft.com/ru-ru/azure/architecture/best-practices/api-design> (дата обращения: 19.05.2022).

Приложение А

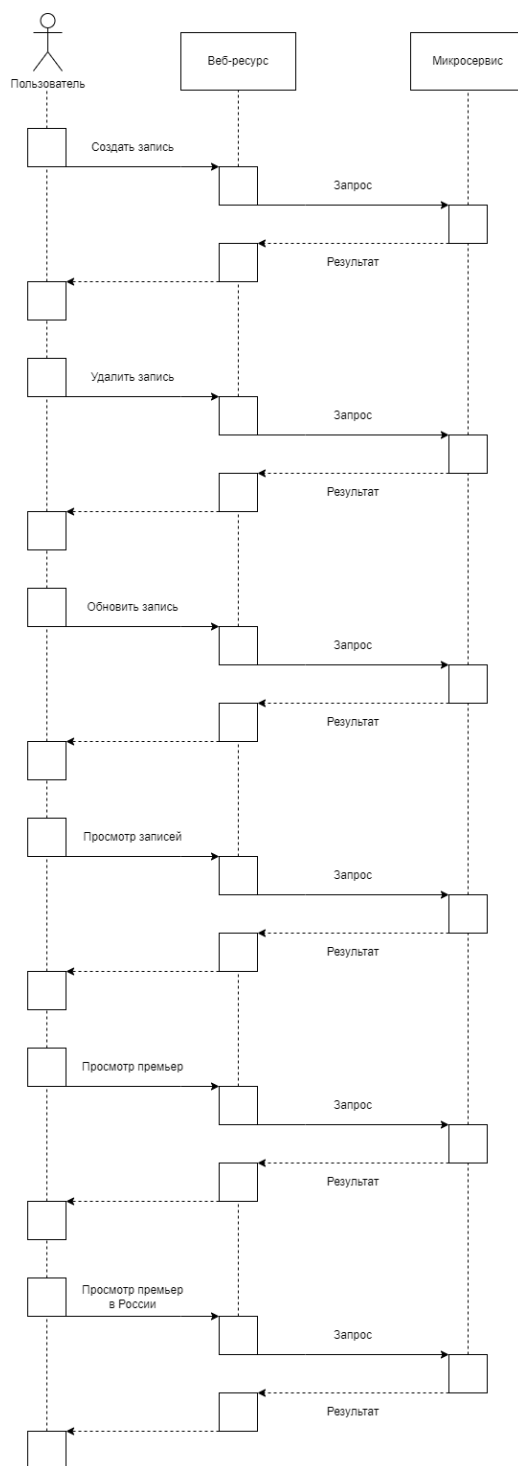
(Обязательное)

Микросервис учёта сведений о фильмах

UML-диаграммы

					ПГУ1.090301.14.001 ПЗ	Лист
						59
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1 - Диаграмма последовательностей



ПГУ1.090301.14.002 Д14				
Изм./Лист	№ докум.	Пооп.	Дата	Микросервис учета сведений о фильмах
Разраб.	Зорин В.И.			Лит.
Пров.	Дубровин А.В.			Масса
				Масштаб
Исполн.	Бычков А.С.			Лист 1
Утв.	Митрагин М.А.			Листов 1

Гр. 18BB2

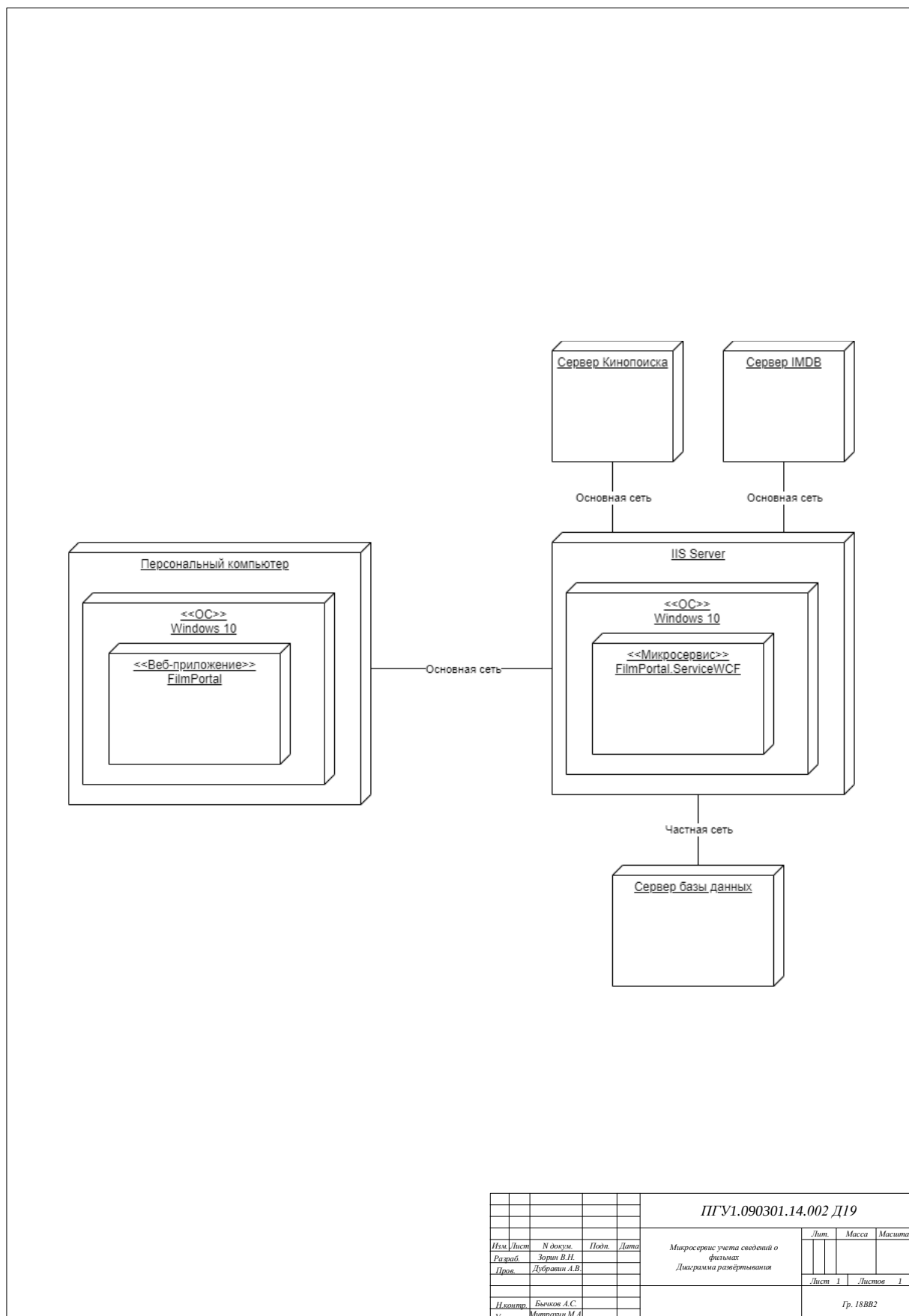
Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

ПГУ1.090301.14.001 ПЗ

Лист

60

Приложение А.2 - Диаграмма развертывания



					ПГУ1.090301.14.002 Д19				
					Микросервис учета сведений о фильмах Диаграмма развертывания	Лист	Масса	Масштаб	
Изм.	Лист	№ докум.	Подп.	Дата					
Разраб.		Зорин В.Н.							
Пров.		Дубравин А.В.							
						Лист 1	Листов	1	
Исполн.		Бычков А.С.				Гр. 18ВВ2			
		Митрошин М.А.							

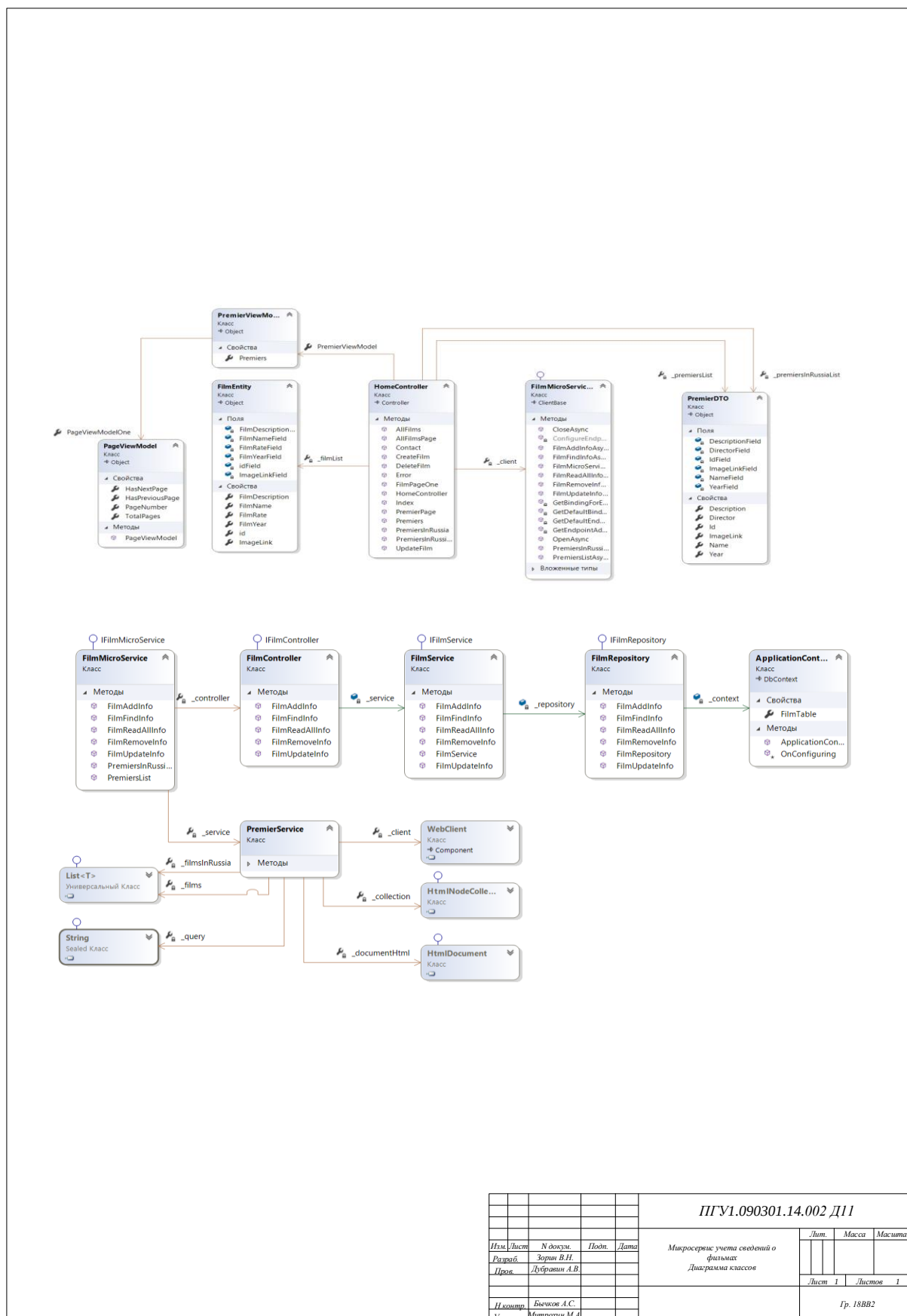
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.14.001 ПЗ

Лист

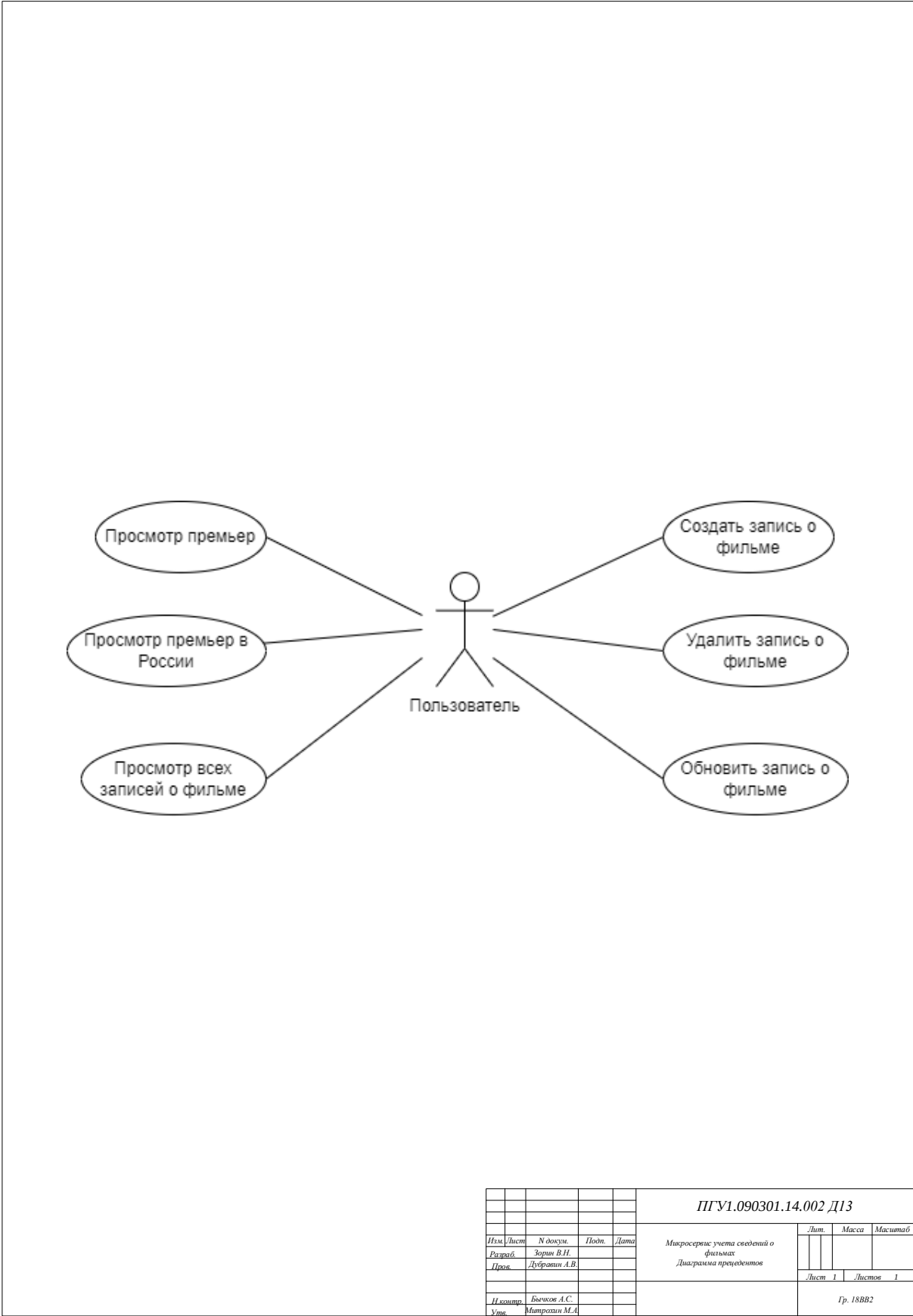
61

Приложение А.3 - Диаграмма классов



					ПГУ1.090301.14.002 Д11				
					Микросервис учета сведений о фильмах Диаграмма классов	Лит.	Масса	Масштаб	
Изм.	Лист	№ докум.	Подп.	Дата					
Разраб.	Зорин В.И.								
Пров.	Дубровин А.В.								
						Лист 1	Листов 1		
Исполн.	Белов А.С.					Гр. 18BB2			
Утв.	Митрошин М.А.								

Приложение А.4 –Диаграмма прецедентов



Приложение Б
(обязательное)

Микросервис учёта сведений о фильмах
Листинг программы *FilmPortal.ServiceWCF*

					<i>ПГУ1.090301.14.001 ПЗ</i>	Лист
						64
Изм.	Лист	№ докум.	Подпись	Дата		

Файл FilmMicroService.svc.cs

```
using FilmPortal.Controllers;
using FilmPortal.ServiceWCF.DBContext;
using FilmPortal.ServiceWCF.DTO;
using FilmPortal.ServiceWCF.Service;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

namespace FilmPortal.ServiceWCF
{
    [ServiceBehavior(IncludeExceptionDetailInFaults = true)]
    public class FilmMicroService : IFilmMicroService
    {
        #region Main
        private FilmController _controller { get; set; } = new
FilmController();

        private PremierService _service { get; set; } = new
PremierService();

        public string FilmAddInfo(string filmName, string
filmDescription, int filmYear, string filmRate, string imageLink)
        {
            try
            {
                _controller.FilmAddInfo(filmName,
filmDescription, filmYear, filmRate, imageLink);
                return "Фильм добавлен";
            } catch
            {
                return "Что-то пошло не так. Повторите попытку
позже.";
            }
        }

        public FilmEntity FilmFindInfo(int id)
        {
            return _controller.FilmFindInfo(id);
        }

        public List<FilmEntity> FilmReadAllInfo()
        {
            return _controller.FilmReadAllInfo();
        }
    }
}
```

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		65

```

        public string FilmRemoveInfo(int id)
        {
            try
            {
                _controller.FilmRemoveInfo(id);
                return "Фильм удалён";
            } catch
            {
                return "Что-то пошло не так. Повторите попытку
позже.";
            }
        }

        public string FilmUpdateInfo(string filmName, string
filmDescription, int filmYear, string filmRate, int id, string
imageLink)
        {
            if (filmName != null || filmDescription != null
|| filmRate != null || filmYear != 0 || id != 0) {
                _controller.FilmUpdateInfo(filmName,
filmDescription, filmYear, filmRate, id, imageLink);
                return "Фильм обновлён";
            } else
            {
                return "Что-то пошло не так. Повторите попытку
позже.";
            }
        }

        public List<PremierDTO> PremiersList()
        {
            return _service.FindPremiers().Result;
        }

        public List<PremierDTO> PremiersInRussiaList()
        {
            return _service.FindPremiersRus().Result;
        }
    }
}
#endregion
}

```

Файл IFilmMicroService.cs

```

using FilmPortal.ServiceWCF.DBContext;
using FilmPortal.ServiceWCF.DTO;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;

```

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		66

```

using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

namespace FilmPortal.ServiceWCF
{
    [ServiceContract(Namespace =
"http://localhost/FilmPortal.ServiceWCF")]
    public interface IFilmMicroService
    {
        [OperationContract]
        [WebInvoke(Method = "POST", UriTemplate =
"FilmAddInfo?filmName={filmName}&filmDescription={filmDescription
}&filmYear={filmYear}&filmRate={filmRate}")]
        string FilmAddInfo(string filmName, string
filmDescription, int filmYear, string filmRate, string
imageLink);

        [OperationContract]
        [WebInvoke(Method = "POST", UriTemplate =
"FilmRemoveInfo?id={id}")]
        string FilmRemoveInfo(int id);

        [OperationContract]
        [WebInvoke(Method = "POST", UriTemplate =
"FilmUpdateInfo?filmName={filmName}&filmDescription={filmDescript
ion}&filmYear={filmYear}&filmRate={filmRate}&id={id}")]
        string FilmUpdateInfo(string filmName, string
filmDescription, int filmYear, string filmRate, int id, string
imageLink);

        [OperationContract]
        [WebInvoke(Method = "GET", UriTemplate =
"FilmReadAllInfo?all")]
        List<FilmEntity> FilmReadAllInfo();

        [OperationContract]
        [WebInvoke(Method = "GET", UriTemplate =
"FilmFindInfo?id={id}")]
        FilmEntity FilmFindInfo(int id);

        [OperationContract]
        [WebInvoke(Method = "GET", UriTemplate =
"PremiersList?All")]
        List<PremierDTO> PremiersList();

        [OperationContract]
        [WebInvoke(Method = "GET", UriTemplate =
"PremiersInRussiaList?All")]
        List<PremierDTO> PremiersInRussiaList();
    }
}

```

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		67

```

    }
}

```

Файл FilmController.cs

```

using FilmPortal.ServiceWCF.DBContext;
using FilmPortal.ServiceWCF.DTO;
using FilmPortal.ServiceWCF.Interfaces;
using FilmPortal.Service;
using System.Collections.Generic;

namespace FilmPortal.Controllers
{
    public class FilmController : IFilmController
    {
        private FilmService _service = new FilmService();

        public void FilmAddInfo(string filmName, string
filmDescription, int filmYear, string filmRate, string imageLink)
        {
            var film = new FilmDTO() {
                FilmName = filmName,
                FilmDescription = filmDescription,
                FilmYear = filmYear,
                FilmRate = filmRate,
                ImageLink = imageLink ?? "No image"
            };
            _service.FilmAddInfo(film);
        }

        public FilmEntity FilmFindInfo(int id)
        {
            return _service.FilmFindInfo(id);
        }

        public List<FilmEntity> FilmReadAllInfo()
        {
            return _service.FilmReadAllInfo();
        }

        public void FilmRemoveInfo(int id)
        {
            _service.FilmRemoveInfo(id);
        }

        public void FilmUpdateInfo(string filmName, string
filmDescription, int filmYear, string filmRate, int id, string
imageLink)
        {
            var film = new FilmDTO()
            {
                FilmName = filmName,
                FilmDescription = filmDescription,

```

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		68

```

        FilmYear = filmYear,
        FilmRate = filmRate,
        ImageLink = imageLink ?? "No image"
    };
    _service.FilmUpdateInfo(film, id);
}
}
}

```

Файл FilmService.cs

```

using FilmPortal.ServiceWCF.DBContext;
using FilmPortal.ServiceWCF.DTO;
using FilmPortal.ServiceWCF.Interfaces;
using FilmPortal.Repository;
using System;
using System.Collections.Generic;

namespace FilmPortal.Service
{
    public class FilmService : IFilmService
    {
        private FilmRepository _repository = new
FilmRepository();
        public FilmService()
        {
        }
        public void FilmAddInfo(FilmDTO film)
        {
            var newEntity = new FilmEntity()
            {
                FilmName = film.FilmName,
                FilmDescription = film.FilmDescription,
                FilmYear = film.FilmYear,
                FilmRate = film.FilmRate,
                ImageLink = film.ImageLink
            };
            _repository.FilmAddInfo(newEntity);
        }

        public FilmEntity FilmFindInfo(int id)
        {
            return _repository.FilmFindInfo(id);
        }
        public List<FilmEntity> FilmReadAllInfo()
        {
            return _repository.FilmReadAllInfo();
        }
    }
}

```

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		69

```

        public void FilmRemoveInfo(int id)
        {
            _repository.FilmRemoveInfo(id);
        }

        public void FilmUpdateInfo(FilmDTO film, int id)
        {
            _repository.FilmUpdateInfo(film, id);
        }
    }
}

```

Файл FilmRepository.cs

```

using FilmPortal.ServiceWCF.DBContext;
using FilmPortal.ServiceWCF.DTO;
using FilmPortal.ServiceWCF.Interfaces;
using System;
using System.Collections.Generic;
using System.Linq;

namespace FilmPortal.Repository
{
    public class FilmRepository : IFilmRepository
    {
        private ApplicationContext _context = new
        ApplicationContext();
        public FilmRepository()
        {
        }

        public void FilmAddInfo(FilmEntity film)
        {
            var entities = _context.FilmTable.ToList();
            film.id = entities.Max(item => item.id) + 1;
            _context.FilmTable.Add(film);
            _context.SaveChanges();
        }

        public FilmEntity FilmFindInfo(int Id)
        {
            var id = Id;
            var entity = _context.FilmTable.Find(id);
            return entity;
        }

        public List<FilmEntity> FilmReadAllInfo()
        {
            return _context.FilmTable.ToList();
        }
    }
}

```

					ПГУ1.090301.14.001 ПЗ	Лист
						70
Изм.	Лист	№ докум.	Подпись	Дата		

```

    }

    public void FilmRemoveInfo(int id)
    {
        var entity = _context.FilmTable.Find(id);
        _context.FilmTable.Remove(entity);
        _context.SaveChanges();
    }

    public void FilmUpdateInfo(FilmDTO film, int id)
    {
        var entity = _context.FilmTable.Find(id);
        entity.FilmName = film.FilmName;
        entity.FilmDescription = film.FilmDescription;
        entity.FilmRate = film.FilmRate;
        entity.FilmYear = film.FilmYear;
        entity.ImageLink = film.ImageLink;
        _context.FilmTable.Update(entity);
        _context.SaveChanges();
    }
}

```

Файл IFilmController.cs

```

using FilmPortal.ServiceWCF;
using FilmPortal.ServiceWCF.DBContext;
using System.Collections.Generic;

namespace FilmPortal.ServiceWCF.Interfaces
{
    public interface IFilmController
    {
        void FilmAddInfo(string filmName, string filmDescription,
            int filmYear, string filmRate, string imageLink);

        void FilmRemoveInfo(int id);

        void FilmUpdateInfo(string filmName, string
            filmDescription, int filmYear, string filmRate, int id, string
            imageLink);

        List<FilmEntity> FilmReadAllInfo();

        FilmEntity FilmFindInfo(int userId);
    }
}

```

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		71

Файл IFilmService.cs

```
using FilmPortal.ServiceWCF.DBContext;
using FilmPortal.ServiceWCF.DTO;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace FilmPortal.ServiceWCF.Interfaces
{
    public interface IFilmService
    {
        void FilmAddInfo(FilmDTO film);

        void FilmRemoveInfo(int id);

        void FilmUpdateInfo(FilmDTO film, int id);

        List<FilmEntity> FilmReadAllInfo();

        FilmEntity FilmFindInfo(int userId);
    }
}
```

Файл IFilmRepository.cs

```
using FilmPortal.ServiceWCF.DBContext;
using FilmPortal.ServiceWCF.DTO;
using System.Collections.Generic;

namespace FilmPortal.ServiceWCF.Interfaces
{
    public interface IFilmRepository
    {
        void FilmAddInfo(FilmEntity film);

        void FilmRemoveInfo(int id);

        void FilmUpdateInfo(FilmDTO film, int id);

        List<FilmEntity> FilmReadAllInfo();

        FilmEntity FilmFindInfo(int userId);
    }
}
```

Файл PremierService.cs

					ПГУ1.090301.14.001 ПЗ	Лист
						72
Изм.	Лист	№ докум.	Подпись	Дата		

```

using AngleSharp;
using FilmPortal.ServiceWCF.DBContext;
using FilmPortal.ServiceWCF.DTO;
using HtmlAgilityPack;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Threading.Tasks;
using System.Web;

namespace FilmPortal.ServiceWCF.Service
{
    public class PremierService
    {
        private WebClient _client { get; set; }
        private HtmlDocument _documentHtml { get; set; }
        private HtmlNodeCollection _collection { get; set; }
        private List<PremierDTO> _films { get; set; } = new
List<PremierDTO>();

        private List<PremierDTO> _filmsInRussia { get; set; } =
new List<PremierDTO>();
        private string _query { get; set; }
        public PremierService()
        {
            _client = new WebClient();
            _documentHtml = new HtmlDocument();
        }

        public async Task<List<PremierDTO>> FindPremiersRus()
        {
            string url = "https://www.kinopoisk.ru/afisha/new/";
            var browserConfig =
Configuration.Default.WithDefaultLoader();
            using (var context =
BrowsingContext.New(browserConfig))
            using (var doc = await context.OpenAsync(url))
            {
                var parsedFilms =
doc.GetElementsByClassName("filmsListNew")[0].GetElementsByClassN
ame("item");
                for (int i = 0; i < parsedFilms.Length; i++)
                {
                    var name =
parsedFilms[i].GetElementsByClassName("info")[0].GetElementsByCla
ssName("name")[0].TextContent.Trim().Split('\n');
                    var director =
parsedFilms[i].GetElementsByClassName("info")[0].GetElementsByCla
ssName("gray")[0].QuerySelector("a").TextContent.Trim();

```

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		73

```

        var imageLink =
parsedFilms[i].GetElementsByClassName("poster")[0].QuerySelector(
"a").GetElementsByTagName("img")[0].GetAttribute("src");
        _filmsInRussia.Add(new PremierDTO()
{
    Id = i,
    Name = name[0],
    Description = "Releases in Russia",
    Director = director,
    ImageLink = imageLink,
    Year = "2022",

    });
    }
}
if (_filmsInRussia == null) return null;
return _filmsInRussia;
}
public async Task<List<PremierDTO>> FindPremiers()
{
    string url = "https://www.imdb.com/movies-coming-
soon/";
    var browserConfig =
Configuration.Default.WithDefaultLoader();
    using (var context =
BrowsingContext.New(browserConfig))
    using (var doc = await context.OpenAsync(url))
    {
        var parsedFilms =
doc.GetElementsByClassName("list
detail")[0].GetElementsByClassName("list_item");
        for (int i = 0; i < parsedFilms.Length; i++)
        {
            string premierName =
parsedFilms[i].QuerySelector("h4").TextContent.Trim();

            var premierDescription =
parsedFilms[i].GetElementsByClassName("outline")[0].TextContent.T
rim();

            var imageLink =
parsedFilms[i].GetElementsByClassName("poster")[0].GetAttribute("
src").Trim();

            _films.Add(new PremierDTO()
{
                Id = i,
                Name =
premierName.Remove(premierName.Length - 7, 7),

```

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		74

```

        Director =
parsedFilms[i].GetElementsByClassName("txt-
block")[0].QuerySelector("a").TextContent.Trim(),
        Description = premierDescription,
        Year =
premierName.Substring(premierName.Length - 5, 4),
        ImageLink = imageLink
    });
    }
}
if (_films == null) return null;
return _films;
}
}
}

```

Файл ApplicationContext.cs

```

using Microsoft.EntityFrameworkCore;

namespace FilmPortal.ServiceWCF.DBContext
{
    public class ApplicationContext : DbContext
    {
        public DbSet<FilmEntity> FilmTable { get; set; }
        public ApplicationContext()
        {
            Database.EnsureCreated();
        }
        protected override void
OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseNpgsql("Host=localhost;Port=5432;Database=FilmD
B;Username=test;Password=test");
        }
    }
}

```

Файл FilmEntity.cs

```

namespace FilmPortal.ServiceWCF.DBContext
{
    public class FilmEntity
    {
        public int id { get; set; }

        public int FilmYear { get; set; }

        public string FilmName { get; set; }
    }
}

```

```

        public string FilmDescription { get; set; }

        public string FilmRate { get; set; }

        public string ImageLink { get; set; }

        public override string ToString()
        {
            return $"{id},{FilmYear},{FilmName}";
        }
    }
}

```

Файл FilmDTO.cs

```

namespace FilmPortal.ServiceWCF.DTO
{
    public class FilmDTO
    {
        public string FilmName { get; set; }

        public string FilmDescription { get; set; }

        public int FilmYear { get; set; }

        public string FilmRate { get; set; }

        public string ImageLink { get; set; }
    }
}

```

Файл PremierDTO.cs

```

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

namespace FilmPortal.ServiceWCF.DTO
{
    public class PremierDTO
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Director { get; set; }
        public string Description { get; set; }
        public string Year { get; set; }
        public string ImageLink { get; set; }
    }
}

```

Файл Web.config

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		76

```

<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <appSettings>
    <add key="aspnet:UseTaskFriendlySynchronizationContext"
value="true" />
  </appSettings>
  <system.web>
    <compilation targetFramework="4.7.2" debug="true" />
    <httpRuntime targetFramework="4.7.2" />
  </system.web>
  <system.serviceModel>
    <behaviors>
      <endpointBehaviors>
        <behavior name="Basic" />
        <behavior name="WebJson">
          <webHttp defaultOutgoingResponseFormat="Json"
faultExceptionEnabled="true" />
        </behavior>
        <behavior name="WebXML">
          <webHttp defaultOutgoingResponseFormat="Xml"
faultExceptionEnabled="true" />
        </behavior>
      </endpointBehaviors>
      <serviceBehaviors>
        <behavior>
          <!--Чтобы избежать раскрытия метаданных, до
развертывания задайте следующим параметрам значение "false". -->
          <serviceMetadata httpGetEnabled="true"
httpsGetEnabled="true" />
          <!--Чтобы при сбое получать подробные сведения об
исключении для отладки, задайте следующему параметру значение
"true". Чтобы информация об исключении не выдавалась, задайте до
развертывания значение "false". -->
          <serviceDebug includeExceptionDetailInFaults="false" />
        </behavior>
        <behavior name="debug">
          <serviceDebug includeExceptionDetailInFaults="true" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <bindings>
      <basicHttpBinding>
        <binding name="Basic" closeTimeout="00:01:00"
openTimeout="00:01:00" receiveTimeout="00:20:00"
sendTimeout="00:20:00" maxBufferPoolSize="524288000"
maxBufferSize="65536000" maxReceivedMessageSize="65536000">
          <readerQuotas maxDepth="65536000"
maxStringContentLength="65536000" maxArrayLength="65536000"
maxBytesPerRead="65536000" maxNameTableCharCount="65536000" />
        </binding>

```

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		77

```

        </basicHttpBinding>
        <webHttpBinding>
            <binding name="WebJson" closeTimeout="00:01:00"
openTimeout="00:01:00" receiveTimeout="00:20:00"
sendTimeout="00:20:00" maxBufferSize="65536000"
maxBufferPoolSize="524288000" maxReceivedMessageSize="65536000">
                <readerQuotas maxDepth="65536000"
maxStringContentLength="65536000" maxArrayLength="65536000"
maxBytesPerRead="65536000" maxNameTableCharCount="65536000" />
            </binding>
            <binding name="WebXML" closeTimeout="00:01:00"
openTimeout="00:01:00" receiveTimeout="00:20:00"
sendTimeout="00:20:00" maxBufferSize="65536000"
maxBufferPoolSize="524288000" maxReceivedMessageSize="65536000">
                <readerQuotas maxDepth="65536000"
maxStringContentLength="65536000" maxArrayLength="65536000"
maxBytesPerRead="65536000" maxNameTableCharCount="65536000" />
            </binding>
        </webHttpBinding>
    </bindings>
    <services>
        <service name="FilmPortal.ServiceWCF.FilmMicroService">
            <!-- конечная точка SOAP -->
            <endpoint address="basic" binding="basicHttpBinding"
behaviorConfiguration="Basic" bindingConfiguration="Basic"
name="Basic"
bindingNamespace="http://localhost/FilmPortal.ServiceWCF"
contract="FilmPortal.ServiceWCF.IFilmMicroService" />
            <!-- конечная точка REST (формат Json) -->
            <endpoint address="json" binding="webHttpBinding"
behaviorConfiguration="WebJson" bindingConfiguration="WebJson"
name="WebJson"
bindingNamespace="http://localhost/FilmPortal.ServiceWCF"
contract="FilmPortal.ServiceWCF.IFilmMicroService" />
            <!-- конечная точка REST (формат XML) -->
            <endpoint address="xml" binding="webHttpBinding"
behaviorConfiguration="WebXML" bindingConfiguration="WebXML"
name="WebXML"
bindingNamespace="http://localhost/FilmPortal.ServiceWCF"
contract="FilmPortal.ServiceWCF.IFilmMicroService" />
        </service>
    </services>
    <protocolMapping>
        <add binding="basicHttpsBinding" scheme="https" />
    </protocolMapping>
    <serviceHostingEnvironment aspNetCompatibilityEnabled="true"
multipleSiteBindingsEnabled="true" />
</system.serviceModel>
<system.webServer>
    <modules runAllManagedModulesForAllRequests="true" />

```

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		78

<!--

Для просмотра корневого каталога веб-приложения во время отладки установите значение true.

Перед развертыванием установите значение false, чтобы избежать раскрытия сведений в папке веб-приложения.

-->

```
<directoryBrowse enabled="true" />
</system.webServer>
<runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
    <dependentAssembly>
      <assemblyIdentity name="System.Numerics.Vectors"
publicKeyToken="b03f5f7f11d50a3a" culture="neutral" />
      <bindingRedirect oldVersion="0.0.0.0-4.1.4.0"
newVersion="4.1.4.0" />
    </dependentAssembly>
    <dependentAssembly>
      <assemblyIdentity
name="System.Runtime.CompilerServices.Unsafe"
publicKeyToken="b03f5f7f11d50a3a" culture="neutral" />
      <bindingRedirect oldVersion="0.0.0.0-5.0.0.0"
newVersion="5.0.0.0" />
    </dependentAssembly>
    <dependentAssembly>
      <assemblyIdentity name="System Buffers"
publicKeyToken="cc7b13ffcd2ddd51" culture="neutral" />
      <bindingRedirect oldVersion="0.0.0.0-4.0.3.0"
newVersion="4.0.3.0" />
    </dependentAssembly>
    <dependentAssembly>
      <assemblyIdentity name="System.Memory"
publicKeyToken="cc7b13ffcd2ddd51" culture="neutral" />
      <bindingRedirect oldVersion="0.0.0.0-4.0.1.1"
newVersion="4.0.1.1" />
    </dependentAssembly>
    <dependentAssembly>
      <assemblyIdentity name="Microsoft.Extensions.Primitives"
publicKeyToken="adb9793829ddae60" culture="neutral" />
      <bindingRedirect oldVersion="0.0.0.0-3.1.1.0"
newVersion="3.1.1.0" />
    </dependentAssembly>
    <dependentAssembly>
      <assemblyIdentity
name="Microsoft.Extensions.Configuration.Abstractions"
publicKeyToken="adb9793829ddae60" culture="neutral" />
      <bindingRedirect oldVersion="0.0.0.0-3.1.1.0"
newVersion="3.1.1.0" />
    </dependentAssembly>
    <dependentAssembly>
```

```

    <assemblyIdentity
name="Microsoft.Extensions.DependencyInjection.Abstractions"
publicKeyToken="adb9793829ddae60" culture="neutral" />
    <bindingRedirect oldVersion="0.0.0.0-3.1.1.0"
newVersion="3.1.1.0" />
    </dependentAssembly>
    <dependentAssembly>
    <assemblyIdentity
name="Microsoft.Extensions.Caching.Abstractions"
publicKeyToken="adb9793829ddae60" culture="neutral" />
    <bindingRedirect oldVersion="0.0.0.0-3.1.1.0"
newVersion="3.1.1.0" />
    </dependentAssembly>
    <dependentAssembly>
    <assemblyIdentity name="Microsoft.Extensions.Options"
publicKeyToken="adb9793829ddae60" culture="neutral" />
    <bindingRedirect oldVersion="0.0.0.0-3.1.1.0"
newVersion="3.1.1.0" />
    </dependentAssembly>
    <dependentAssembly>
    <assemblyIdentity
name="Microsoft.Extensions.Logging.Abstractions"
publicKeyToken="adb9793829ddae60" culture="neutral" />
    <bindingRedirect oldVersion="0.0.0.0-3.1.1.0"
newVersion="3.1.1.0" />
    </dependentAssembly>
    <dependentAssembly>
    <assemblyIdentity
name="System.Threading.Tasks.Extensions"
publicKeyToken="cc7b13ffcd2ddd51" culture="neutral" />
    <bindingRedirect oldVersion="0.0.0.0-4.2.0.1"
newVersion="4.2.0.1" />
    </dependentAssembly>
    <dependentAssembly>
    <assemblyIdentity
name="Microsoft.Extensions.DependencyInjection"
publicKeyToken="adb9793829ddae60" culture="neutral" />
    <bindingRedirect oldVersion="0.0.0.0-3.1.1.0"
newVersion="3.1.1.0" />
    </dependentAssembly>
    <dependentAssembly>
    <assemblyIdentity
name="System.ComponentModel.Annotations"
publicKeyToken="b03f5f7f11d50a3a" culture="neutral" />
    <bindingRedirect oldVersion="0.0.0.0-4.2.1.0"
newVersion="4.2.1.0" />
    </dependentAssembly>
    <dependentAssembly>
    <assemblyIdentity name="System.ValueTuple"
publicKeyToken="cc7b13ffcd2ddd51" culture="neutral" />

```

```

        <bindingRedirect oldVersion="0.0.0.0-4.0.3.0"
newVersion="4.0.3.0" />
    </dependentAssembly>
</assemblyBinding>
</runtime>
</configuration>
<!--ProjectGuid: 885DF432-ABE1-4144-A42B-80BC6865E82A-->
<!--ProjectGuid: BECAEF00-E9E6-4301-81EF-65C6DBF52F60-->

```

					ПГУ1.090301.14.001 ПЗ	Лист
						81
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение В
(обязательное)

Микросервис учёта сведений о фильмах

Листинг программы веб-приложения *FilmPortal*

					ПГУ1.090301.14.001 ПЗ	Лист
						82
Изм.	Лист	№ докум.	Подпись	Дата		

Файл HomeController.cs

```
using System.Collections.Generic;
using System.Diagnostics;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using FilmPortal.Models;
using System.Linq;
using FilmPortal.ParseWeb;
using ServiceReference3;

namespace FilmPortal.Controllers
{
    public class HomeController : Controller
    {
        private FilmMicroServiceClient _client { get; set; }
        private FilmEntity[] _filmList { get; set; }
        private PremierDTO[] _premiersList { get; set; }
        private PremierDTO[] _premiersInRussiaList { get; set; }
        public PremierViewModel PremierViewModel { get; set; }

        public HomeController()
        {
            try
            {
                _client = new FilmMicroServiceClient();
                _filmList =
                _client.FilmReadAllInfoAsync().Result;
                _premiersList =
                _client.PremiersListAsync().Result;
                _premiersInRussiaList =
                _client.PremiersInRussiaListAsync().Result;
            } catch
            {
            }
        }

        public IActionResult Index()
        {
            return View();
        }

        public IActionResult Contact()
        {
            ViewData["Message"] = "Any questions? Connect with
creator of web-site!";
            return View();
        }

        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId =
Activity.Current?.Id ?? HttpContext.TraceIdentifier });
        }
    }
}
```

					ПГУ1.090301.14.001 ПЗ	Лист
						83
Изм.	Лист	№ докум.	Подпись	Дата		

```

    }

    public async Task<IActionResult> CreateFilm(string
filmName, string filmDescription, int filmYear, string filmRate,
string imageLink)
    {
        if (filmName == null || filmDescription == null ||
filmRate == null || filmYear == 0) return View();
        try
        {
            ViewData["Message"] = await
_client.FilmAddInfoAsync(filmName, filmDescription, filmYear,
filmRate, imageLink);

            return View();
        } catch
        {
            ViewData["Message"] = "Сбой в работе. Повторите
попытку позже.";
            return View();
        }
    }

    public async Task<IActionResult> DeleteFilm(int id)
    {
        if (id == 0) return View();
        try
        {
            ViewData["Message"] = await
_client.FilmRemoveInfoAsync(id);
            return View();
        } catch
        {
            ViewData["Message"] = "Сбой в работе. Повторите
попытку позже.";
            return View();
        }
    }

    public async Task<IActionResult> UpdateFilm(int id,
string filmName, string filmDescription, int filmYear, string
filmRate, string imageLink)
    {
        if (filmName == null || filmDescription == null ||
filmRate == null || filmYear == 0 || id == 0) return View();
        try
        {
            ViewData["Message"] = await
_client.FilmUpdateInfoAsync(filmName, filmDescription, filmYear,
filmRate, id, imageLink);

```

```

        return View();
    } catch
    {
        ViewData["Message"] = "Сбой в работе. Повторите
попытку позже.";
        return View();
    }
}

public IActionResult AllFilms(string name)
{
    List<ServiceReference3.FilmEntity> filmEntities =
    _filmList.ToList();
    if (!string.IsNullOrEmpty(name))
    {
        filmEntities = filmEntities.Where(n =>
n.FilmName.Contains(name)).ToList();
    }
    return View(filmEntities);
}

public IActionResult AllFilmsPage(string name, int page =
1)
{
    int pageSize = 3;
    var items = _filmList;
    var count = items.Length;
    if (!string.IsNullOrEmpty(name))
    {
        items = items.Where(n =>
n.FilmName.Contains(name)).ToArray();
    }

    PageViewModel pageViewModel = new
PageViewModel(count, page, pageSize);
    IndexViewModel viewModel = new IndexViewModel
    {
        PageViewModel = pageViewModel,
        Films = items
    };
    return View(viewModel);
}

public IActionResult Premiers(string name, int page = 1)
{
    try {
        var items = _premiersList.ToList();
        if (!string.IsNullOrEmpty(name))

```

```

        {
            items = items.Where(n =>
n.Name.Contains(name)).ToList();
        }

        int pageSize = 3;
        int count = items.Count;
        PageViewModel pageViewModel = new
PageViewModel(count, page, pageSize);
        PremierViewModel = new PremierViewModel
        {
            PageViewModelOne = pageViewModel,
            Premiers = items
        };
        return View(PremierViewModel);
    }
    catch {
        return Redirect("~/Home/Index");
    }
}

[HttpGet]
public async Task<IActionResult> FilmPageOne(int id)
{
    try {
        var entityToFind = await
_client.FilmFindInfoAsync(id);
        ViewBag.Id = entityToFind.id;
        ViewBag.FilmName = entityToFind.FilmName;
        ViewBag.FilmDescription =
entityToFind.FilmDescription;
        ViewBag.FilmYear = entityToFind.FilmYear;
        ViewBag.FilmRate = entityToFind.FilmRate;
        ViewBag.ImageLink = entityToFind.ImageLink;
        return View();
    }
    catch { return Redirect("~/Home/AllFilmsPage"); }
}

[HttpGet]
public IActionResult PremierPage(int Id)
{
    try {
        var premierToFind = _premiersList.ToList().Find(x =>
x.Id == Id);
        ViewBag.ImageLink = premierToFind.ImageLink;
        ViewBag.Name = premierToFind.Name;
        ViewBag.Director = premierToFind.Director;
        ViewBag.Description = premierToFind.Description;
        ViewBag.Year = premierToFind.Year;
    }
    catch { return Redirect("~/Home/AllPremiersPage"); }
}

```

```

        return View();
    }
    catch {
        return Redirect("~/Home/Premiers");
    }
}

public IActionResult PremiersInRussia(string name, int
page = 1)
{
    try {
        var items = _premiersInRussiaList.ToList();
        if (!string.IsNullOrEmpty(name))
        {
            items = items.Where(n =>
n.Name.Contains(name)).ToList();
        }

        int pageSize = 3;
        int count = items.Count;
        PageViewModel pageViewModel = new
PageViewModel(count, page, pageSize);
        PremierViewModel = new PremierViewModel
        {
            PageViewModelOne = pageViewModel,
            Premiers = items
        };
        return View(PremierViewModel);
    }
    catch
    {
        return Redirect("~/Home/Index");
    }
}

[HttpGet]
public IActionResult PremiersInRussiaPage(int Id)
{
    try {
        var premierToFind =
        _premiersInRussiaList.ToList().Find(x => x.Id == Id);
        ViewBag.ImageLink = premierToFind.ImageLink;
        ViewBag.Name = premierToFind.Name;
        ViewBag.Director = premierToFind.Director;
        ViewBag.Description = premierToFind.Description;
        ViewBag.Year = premierToFind.Year;
        return View();
    }
    catch
    {
        return Redirect("~/Home/PremiersInRussia");
    }
}

```

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		87

```

    }
}
}

```

Файл ErrorViewModel.cs

```

using System;

namespace FilmPortal.Models
{
    public class ErrorViewModel
    {
        public string RequestId { get; set; }

        public bool ShowRequestId =>
!string.IsNullOrEmpty(RequestId);
    }
}

```

Файл IndexViewModel.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace FilmPortal.Models
{
    public class IndexViewModel
    {
        public IEnumerable<ServiceReference3.FilmEntity> Films {
get; set; }
        public PageViewModel PageViewModel { get; set; }
    }
}

```

Файл PageViewModel.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace FilmPortal.Models
{
    public class PageViewModel
    {
        public int PageNumber { get; private set; }
        public int TotalPages { get; private set; }
    }
}

```

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		88

```

        public PageViewModel(int count, int pageNumber, int
pageSize)
        {
            PageNumber = pageNumber;
            TotalPages = (int)Math.Ceiling(count /
(double)pageSize);
        }

        public bool HasPreviousPage
        {
            get
            {
                return (PageNumber > 1);
            }
        }

        public bool HasNextPage
        {
            get
            {
                return (PageNumber < TotalPages);
            }
        }
    }
}

```

Файл PremierViewModel.cs

```

using System.Collections.Generic;
using ServiceReference3;
namespace FilmPortal.Models
{
    public class PremierViewModel
    {
        public IEnumerable<PremierDTO> Premiers { get; set; }
        public PageViewModel PageViewModelOne { get; set; }
    }
}

```

Файл _Layout.cshtml

```

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
    <title>@ViewData["Title"]</title>

    <environment include="Development">

```

					ПГУ1.090301.14.001 ПЗ	Лист
						89
Изм.	Лист	№ докум.	Подпись	Дата		

```

        <link rel="stylesheet"
href="~/lib/bootstrap/dist/css/bootstrap.css" />
        <link rel="stylesheet" href="~/css/site.css" />
    </environment>
    <environment exclude="Development">
        <link rel="stylesheet"
href="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/css/bootstr
ap.min.css"
        asp-fallback-
href="~/lib/bootstrap/dist/css/bootstrap.min.css"
        asp-fallback-test-class="sr-only" asp-fallback-
test-property="position" asp-fallback-test-value="absolute" />
        <link rel="stylesheet" href="~/css/site.min.css" asp-
append-version="true" />
    </environment>
</head>
<body>
    <nav class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-
toggle="collapse" data-target=".navbar-collapse">
                    <span class="sr-only">Toggle
navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li><a asp-area="" asp-controller="Home" asp-
action="Index">FilmPortal</a></li>
                    <li><a asp-area="" asp-controller="Home" asp-
action="CreateFilm">Создать фильм</a></li>
                    <li><a asp-area="" asp-controller="Home" asp-
action="DeleteFilm">Удалить фильм</a></li>
                    <li><a asp-area="" asp-controller="Home" asp-
action="UpdateFilm">Обновить фильм</a></li>
                    <li><a asp-area="" asp-controller="Home" asp-
action="AllFilmsPage">Все фильмы</a></li>
                    <li><a asp-area="" asp-controller="Home" asp-
action="Premiers">Премьеры в мире</a></li>
                    <li><a asp-area="" asp-controller="Home" asp-
action="PremiersInRussia">Премьеры в России</a></li>
                    <li><a asp-area="" asp-controller="Home" asp-
action="Contact">Обратная связь</a></li>
                </ul>
            </div>
        </div>
    </nav>

```

					ПГУ1.090301.14.001 ПЗ	Лист
						90
Изм.	Лист	№ докум.	Подпись	Дата		

```

</nav>
<div class="container body-content">
    @RenderBody()
    <hr />
    <footer>
        <p>&copy; 2022 - FilmPortal</p>
    </footer>
</div>

<environment include="Development">
    <script src="~/lib/jquery/dist/jquery.js"></script>
    <script
src="~/lib/bootstrap/dist/js/bootstrap.js"></script>
    <script src="~/js/site.js" asp-append-
version="true"></script>
</environment>
<environment exclude="Development">
    <script
src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-2.2.0.min.js"
    asp-fallback-
src="~/lib/jquery/dist/jquery.min.js"
    asp-fallback-test="window.jQuery"
    crossorigin="anonymous"
    integrity="sha384-
K+ctZQ+LL8q6tP7I94W+qzQsfRV2a+AfHII9k8z8l9ggpc8X+Ytst4yBo/hH+8Fk"
>
    </script>
    <script
src="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/bootstrap.mi
n.js"
    asp-fallback-
src="~/lib/bootstrap/dist/js/bootstrap.min.js"
    asp-fallback-test="window.jQuery &&
window.jQuery.fn && window.jQuery.fn.modal"
    crossorigin="anonymous"
    integrity="sha384-
Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIPG9mGCD8wGNICPD7Txa"
>
    </script>
    <script src="~/js/site.min.js" asp-append-
version="true"></script>
</environment>

    @RenderSection("Scripts", required: false)
</body>
</html>

```

Файл _ValidationScriptPartial.cshtml

```
<environment include="Development">
```

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		91

```

        <script src="~/lib/jquery-
validation/dist/jquery.validate.js"></script>
        <script src="~/lib/jquery-validation-
unobtrusive/jquery.validate.unobtrusive.js"></script>
</environment>
<environment exclude="Development">
    <script
src="https://ajax.aspnetcdn.com/ajax/jquery.validate/1.14.0/jquer
y.validate.min.js"
        asp-fallback-src="~/lib/jquery-
validation/dist/jquery.validate.min.js"
        asp-fallback-test="window.jQuery &&
window.jQuery.validator"
        crossorigin="anonymous"
        integrity="sha384-
Fnqn3nXP3506LP/7Y3j/25BlWeA3PXTyT1l78LjECcPaKCV12TsZP7yyMxOe/G/k"
    >
        </script>
    <script
src="https://ajax.aspnetcdn.com/ajax/jquery.validation.unobtrusiv
e/3.2.6/jquery.validate.unobtrusive.min.js"
        asp-fallback-src="~/lib/jquery-validation-
unobtrusive/jquery.validate.unobtrusive.min.js"
        asp-fallback-test="window.jQuery &&
window.jQuery.validator && window.jQuery.validator.unobtrusive"
        crossorigin="anonymous"
        integrity="sha384-
JrXK+k53HACyavUKOsL+NkmSesD2P+73eDMrbTtTk0h4RmOF8hF8apPlkp26JlyH"
    >
        </script>
</environment>

```

Файл Error.cshtml

```

@model ErrorViewModel
@{
    ViewData["Title"] = "Error";
}

<h1 class="text-danger">Error.</h1>
<h2 class="text-danger">An error occurred while processing your
request.</h2>

@if (Model.ShowRequestId)
{
    <p>
        <strong>Request ID:</strong>
        <code>@Model.RequestId</code>
    </p>
}

```

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		92

<h3>Development Mode</h3>

<p>

Swapping to Development environment will display more detailed information about the error that occurred.

</p>

<p>

Development environment should not be enabled in deployed applications, as it can result in sensitive information from exceptions being displayed to end users. For local debugging, development environment can be enabled by setting the ASPNETCORE_ENVIRONMENT environment variable to Development, and restarting the application.

</p>

Файл AllFilmsPage.cshtml

```
@using FilmPortal.Models
```

```
@model IndexViewModel
```

```
@{
```

```
    ViewData["Title"] = "Все фильмы";
```

```
}
```

```
<style>
```

```
    @@font-face {
```

```
        font-family: 'FontAwesome';
```

```
        src: url('https://maxcdn.bootstrapcdn.com/font-awesome/4.7.0/fonts/fontawesome-webfont.woff2') format('woff2'),
        url('https://maxcdn.bootstrapcdn.com/font-awesome/4.7.0/fonts/fontawesome-webfont.woff') format('woff'),
        url('https://maxcdn.bootstrapcdn.com/font-awesome/4.4.0/fonts/fontawesome-webfont.ttf') format('truetype');
```

```
        font-weight: normal;
```

```
        font-style: normal
```

```
    }
```

```
.glyphicon {
```

```
    display: inline-block;
```

```
    font: normal normal normal 14px/1 FontAwesome;
```

```
    font-size: inherit;
```

```
    text-rendering: auto;
```

```
    -webkit-font-smoothing: antialiased;
```

```
    -moz-osx-font-smoothing: grayscale
```

```
}
```

```
.glyphicon-chevron-right:before {
```

```
    content: "\f054";
```

```
}
```

```
.glyphicon-chevron-left:before {
```

```
    content: "\f053";
```

```
}
```

					ПГУ1.090301.14.001 ПЗ	Лист
						93
Изм.	Лист	№ докум.	Подпись	Дата		

```

</style>
<h1>Все фильмы</h1>
<h3>@ViewData["TestMessage"]</h3>
<form method="get">
    <div class="form-inline form-group">
        <label class="control-label">Поиск по имени: </label>
        <input name="name" class="form-control" />
        <input type="submit" value="Filter" class="btn btn-
default" />
    </div>
</form>
<table class="table">

<tr><th>ID</th><th>Название</th><th>Рейтинг</th><th>Год</th><th>И
нформация</th></tr>
    @foreach (ServiceReference3.FilmEntity u in Model.Films)
    {

<tr><td>@u.id</td><td>@u.FilmName</td><td>@u.FilmRate</td><td>@u.
FilmYear</td><td><a href="/Home/FilmPageOne/@u.id">O
фильме</a></td></tr>
    }
</table>

```

Файл Contact.cshtml

```

@{
    ViewData["Title"] = "Contact";
}
<h2>@ViewData["Title"]</h2>
<h3>@ViewData["Message"]</h3>

<address>
    <abbr title="Phone to describe new ideas">Phone:</abbr>
    +7 (964) 875-09-50
</address>

<address>
    <abbr title="Email to contact with author">Support:</abbr>
    <a href="mailto:seva89423@gmail.com">seva89423@gmail.com</a>
</address>

```

Файл CreateFilm.cshtml

```
@{
    ViewData["Title"] = "Создать фильм";
}
<h3>@ViewData["Message"]</h3>
<form method="post">
    <label>Введите название</label><br />
    <input type="text" name="filmName" /><br /><br />
    <label>Введите описание</label><br />
    <input type="text" name="filmDescription" /><br /><br />
    <label>Введите год выпуска</label><br />
    <input type="number" name="filmYear" /><br /><br />
    <label>Введите рейтинг</label><br />
    <input type="text" name="filmRate" /><br /><br />
    <label>Введите ссылку на постер (необязательно)</label><br />
    <input type="text" name="imageLink" /><br /><br />
    <input type="submit" value="Создать фильм!" class="btn btn-
default" />
</form>
```

Файл DeleteFilm.cshtml

```
@{
    ViewData["Title"] = "Delete Film";
}
<h3>@ViewData["Message"]</h3>
<form method="post">
    <label>Введите id фильма</label><br />
    <input type="text" name="id" /><br /><br />
    <input type="submit" value="Удалить фильм!" class="btn btn-
default"/>
</form>
```

Файл FilmPageOne.cshtml

```
@{
    ViewData["Title"] = "О фильме";
}

<title>О фильме</title>
<div>
    <form method="post" action="">
        <input type="hidden" name="id" value="@ViewBag.id">
        <h1>@ViewBag.FilmName</h1>
        @if (@ViewBag.ImageLink.Trim().StartsWith("http"))
        {
            <a href="@ViewBag.ImageLink" target="_blank"> </a>
        }
        else
        {

```

```

        <h4><em>Пока нет постера</em></h4>
    }
    <h4>@ViewBag.FilmDescription</h4>
    <h4>Год: @ViewBag.FilmYear</h4>
    <h4>Рейтинг: @ViewBag.FilmRate</h4>
    <a href="/Home/AllFilmsPage">Назад</a>
</form>
</div>
Файл Index.cshtml

@{
    ViewData["Title"] = "Добро пожаловать на FilmPortal";
}

<div id="myCarousel" class="carousel slide" data-ride="carousel"
data-interval="6000">
    <ol class="carousel-indicators">
        <li data-target="#myCarousel" data-slide-to="0"
class="active"></li>
        <li data-target="#myCarousel" data-slide-to="1"></li>
        <li data-target="#myCarousel" data-slide-to="2"></li>
        <li data-target="#myCarousel" data-slide-to="3"></li>
    </ol>
    <div class="carousel-inner" role="listbox">
        <div class="item active">
            
            <div class="carousel-caption" role="option">
                <p>
                    Learn how to build ASP.NET apps that can run
anywhere.
                    <a class="btn btn-default"
href="https://go.microsoft.com/fwlink/?LinkID=525028&clcid=0x409"
>
                        Learn More
                    </a>
                </p>
            </div>
        </div>
        <div class="item">
            
            <div class="carousel-caption" role="option">
                <p>
                    There are powerful new features in Visual
Studio for building modern web apps.
                    <a class="btn btn-default"
href="https://go.microsoft.com/fwlink/?LinkID=525030&clcid=0x409"
>
                        Learn More
                    </a>
                </p>
            </div>
        </div>
    </div>
</div>

```

```

        </p>
    </div>
</div>
<div class="item">
    
    <div class="carousel-caption" role="option">
        <p>
            Bring in libraries from NuGet and npm, and
            automate tasks using Grunt or Gulp.
            <a class="btn btn-default"
href="https://go.microsoft.com/fwlink/?LinkID=525029&clcid=0x409"
>
                Learn More
            </a>
        </p>
    </div>
</div>
<div class="item">
    
    <div class="carousel-caption" role="option">
        <p>
            Learn how Microsoft's Azure cloud platform
            allows you to build, deploy, and scale web apps.
            <a class="btn btn-default"
href="https://go.microsoft.com/fwlink/?LinkID=525027&clcid=0x409"
>
                Learn More
            </a>
        </p>
    </div>
</div>
</div>
<a class="left carousel-control" href="#myCarousel"
role="button" data-slide="prev">
    <span class="glyphicon glyphicon-chevron-left" aria-
hidden="true"></span>
    <span class="sr-only">Previous</span>
</a>
<a class="right carousel-control" href="#myCarousel"
role="button" data-slide="next">
    <span class="glyphicon glyphicon-chevron-right" aria-
hidden="true"></span>
    <span class="sr-only">Next</span>
</a>
</div>

```

Файл PremierPage.cshtml

@{

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		97

```

        ViewData["Title"] = "Страница премьеры";
    }
<title>0 премьер</title>
<div>
    <form method="post" action="">
        <input type="hidden" name="Id" value="@ViewBag.Id">
        <h1>@ViewBag.Name</h1>
        @if (@ViewBag.ImageLink.Trim().StartsWith("http"))
        {
            <a href="@ViewBag.ImageLink" target="_blank"> </a>
        }
        else
        {
            <h4><em>Пока нет постера</em></h4>
        }
        <h4>Режиссёр: @ViewBag.Director</h4>
        <h4>@ViewBag.Description</h4>
        <h4>Год выхода: @ViewBag.Year</h4>
        <a href="/Home/Premiers">Назад</a>
    </form>
</div>

```

Файл Premiers.cshtml

```

@using FilmPortal.Models
@using ServiceReference3
@model PremierViewModel
@{
    ViewData["Title"] = "Премьеры в мире";
}
<style>
    @@font-face {
        font-family: 'FontAwesome';
        src: url('https://maxcdn.bootstrapcdn.com/font-
awesome/4.7.0/fonts/fontawesome-webfont.woff2') format('woff2'),
url('https://maxcdn.bootstrapcdn.com/font-
awesome/4.7.0/fonts/fontawesome-webfont.woff') format('woff'),
url('https://maxcdn.bootstrapcdn.com/font-
awesome/4.4.0/fonts/fontawesome-webfont.ttf') format('truetype');
        font-weight: normal;
        font-style: normal
    }

    .glyphicon {
        display: inline-block;
        font: normal normal normal 14px/1 FontAwesome;
        font-size: inherit;
        text-rendering: auto;
        -webkit-font-smoothing: antialiased;
        -moz-osx-font-smoothing: grayscale
    }

```

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		98

```

    }

    .glyphicon-chevron-right:before {
        content: "\f054";
    }

    .glyphicon-chevron-left:before {
        content: "\f053";
    }
</style>
<h2>Премьеры в мире</h2>
<form method="get">
    <div class="form-inline form-group">
        <label class="control-label">Найти по имени: </label>
        <input name="name" class="form-control" />
        <input type="submit" value="Filter" class="btn btn-
default" />
    </div>
</form>
<table class="table">

<tr><th>Имя</th><th>Режиссёр</th><th>Год</th><th>Информация</th><
/tr>
    @foreach (PremierDTO p in Model.Premiers)
    {

<tr><td>@p.Name</td><td>@p.Director</td><td>@p.Year</td><td><a
href="/Home/PremierPage/@p.Id">О фильме</a></td></tr>
    }
</table>

```

Файл PremiersInRussia.cs

```

@using FilmPortal.Models
@using ServiceReference3
@model PremierViewModel
@{
    ViewData["Title"] = "Премьеры в России";
}
<style>
    @@font-face {
        font-family: 'FontAwesome';
        src: url('https://maxcdn.bootstrapcdn.com/font-
awesome/4.7.0/fonts/fontawesome-webfont.woff2') format('woff2'),
url('https://maxcdn.bootstrapcdn.com/font-
awesome/4.7.0/fonts/fontawesome-webfont.woff') format('woff'),
url('https://maxcdn.bootstrapcdn.com/font-
awesome/4.4.0/fonts/fontawesome-webfont.ttf') format('truetype');
        font-weight: normal;
        font-style: normal
    }

```

```

.glyphicon {
    display: inline-block;
    font: normal normal normal 14px/1 FontAwesome;
    font-size: inherit;
    text-rendering: auto;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale
}

.glyphicon-chevron-right:before {
    content: "\f054";
}

.glyphicon-chevron-left:before {
    content: "\f053";
}
</style>
<h2>Премьеры в России</h2>
<form method="get">
    <div class="form-inline form-group">
        <label class="control-label">Поиск по имени: </label>
        <input name="name" class="form-control" />
        <input type="submit" value="Filter" class="btn btn-
default" />
    </div>
</form>
<table class="table">

<tr><th>Название</th><th>Режиссёр</th><th>Год</th><th>Информация<
/th></tr>
    @foreach (PremierDTO p in Model.Premiers)
    {

<tr><td>@p.Name</td><td>@p.Director</td><td>@p.Year</td><td><a
href="/Home/PremiersInRussiaPage/@p.Id">О фильме</a></td></tr>
    }
</table>

```

Файл PremiersInRussiaPage.cs

```

@{
    ViewData["Title"] = "Страница премьеры в России";
}
<title>О премьерере</title>
<div>
    <form method="post" action="">
        <input type="hidden" name="Id" value="@ViewBag.Id">
        <h1>@ViewBag.Name</h1>
        @if (@ViewBag.ImageLink.Trim().StartsWith("http"))
        {

```

```

        <a href="@ViewBag.ImageLink" target="_blank"> </a>
    }
    else
    {
        <h4><em>Пока нет постера</em></h4>
    }
    <h4>Режиссёр: @ViewBag.Director</h4>
    <h4>@ViewBag.Description</h4>
    <h4>Год выхода: @ViewBag.Year</h4>
    <a href="/Home/PremiersInRussia">Назад</a>
</form>
</div>

```

Файл UpdateFilm.cs

```

@{
    ViewData["Title"] = "Update Film";
}
<h3>@ViewData["Message"]</h3>
<form method="post">
    <label>Enter film id</label><br />
    <input type="text" name="id" /><br /><br />
    <label>Enter film name</label><br />
    <input type="text" name="filmName" /><br /><br />
    <label>Enter film description</label><br />
    <input type="text" name="filmDescription" /><br /><br />
    <label>Enter film year</label><br />
    <input type="number" name="filmYear" /><br /><br />
    <label>Enter film rate</label><br />
    <input type="text" name="filmRate" /><br /><br />
    <label>Enter film poster link</label><br />
    <input type="text" name="imageLink" /><br /><br />
    <input type="submit" value="Обновить фильм!" class="btn btn-
default" />
</form>

```

Файл Program.cs

```

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Logging;

namespace FilmPortal
{
    public class Program

```

```

    {
        public static void Main(string[] args)
        {
            BuildWebHost(args).Run();
        }

        public static IWebHost BuildWebHost(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>()
                .Build();
    }
}

```

Файл Startup.cs

```

using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;

namespace FilmPortal
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this
        method to add services to the container.
        public void ConfigureServices(IServiceCollection
        services)
        {
            services.AddMvc();
        }

        // This method gets called by the runtime. Use this
        method to configure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app,
        IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseBrowserLink();
                app.UseDeveloperExceptionPage();
            }
            else
            {

```

					ПГУ1.090301.14.001 ПЗ	Лист
						102
Изм.	Лист	№ докум.	Подпись	Дата		

```

        app.UseExceptionHandler("/Home/Error");
    }

    app.UseStaticFiles();

    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template:
"{controller=Home}/{action=Index}/{id?}");
        });
    }
}

```

					ПГУ1.090301.14.001 ПЗ	Лист
						103
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Г
(обязательное)

Микросервис учёта сведений о фильмах

Презентация

					<i>ПГУ1.090301.14.001 ПЗ</i>	<i>Лист</i>
						104
<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>		

Микросервис учёта сведений о фильмах

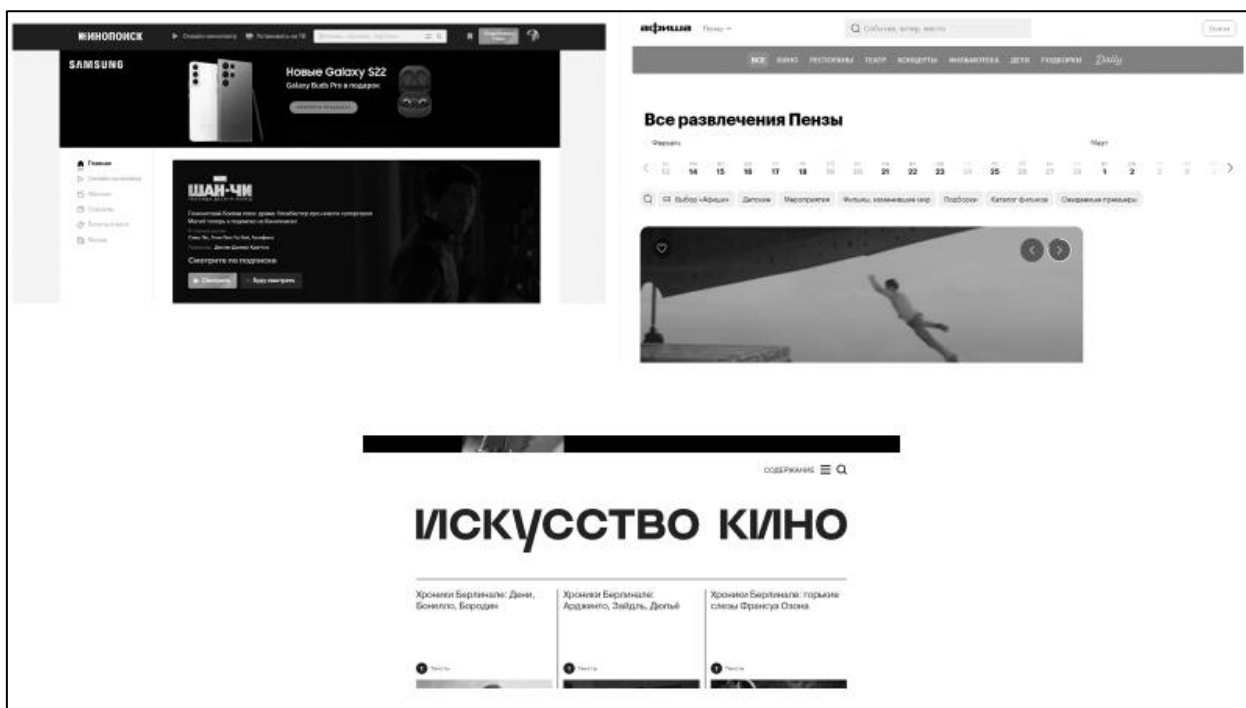
Автор работы: студент 18BB2 Зорин Всеволод Николаевич
Научный руководитель: к.т.н., доцент Дубравин Алексей Викторович

Цель и объект работы

Цель работы – разработка микросервиса для учета и поиска информации о фильмах

Объект работы – приложение, реализующее следующие операции:

- Создание записи о фильме;
- Удаление записи о фильме;
- Обновление записи о фильме;
- Просмотр всех записей о фильмах;
- Просмотр записей о премьерах в России и в мире.



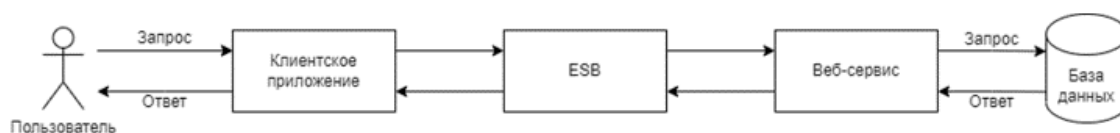
Варианты реализации приложения

- Многослойная архитектура;
- Сервис-ориентированная архитектура (Service Oriented Architecture – SOA);
- Микросервисная архитектура.

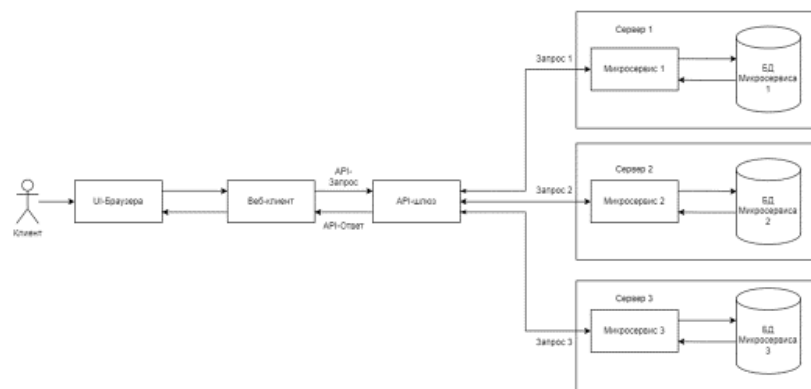
Многослойная архитектура



Сервис-ориентированная архитектура



Микросервисная архитектура



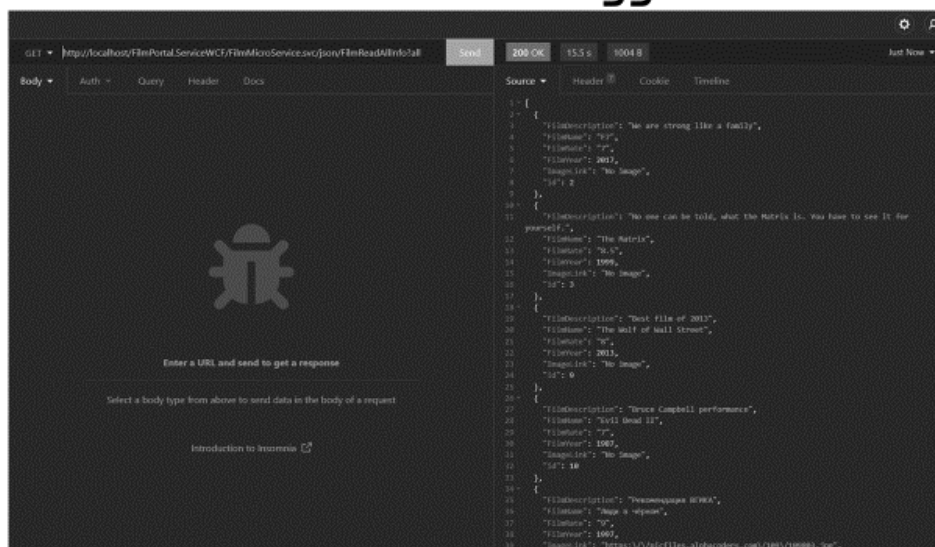
Выбор стека технологий



Стек технологий



Описание Swagger



Разработка web-приложения

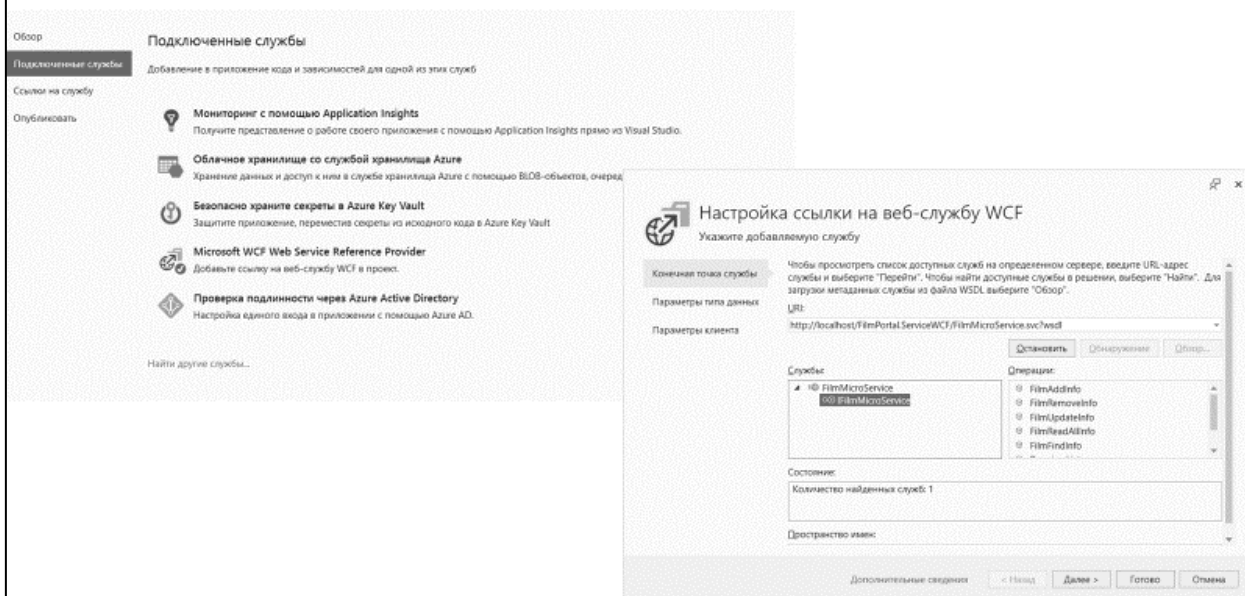


Диаграмма классов

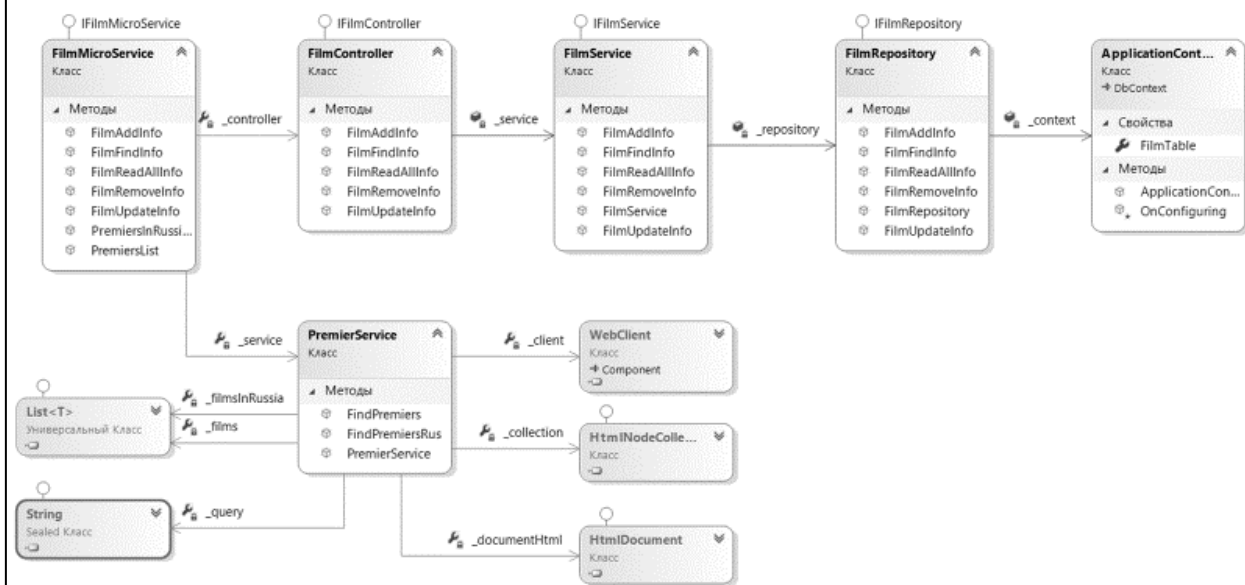
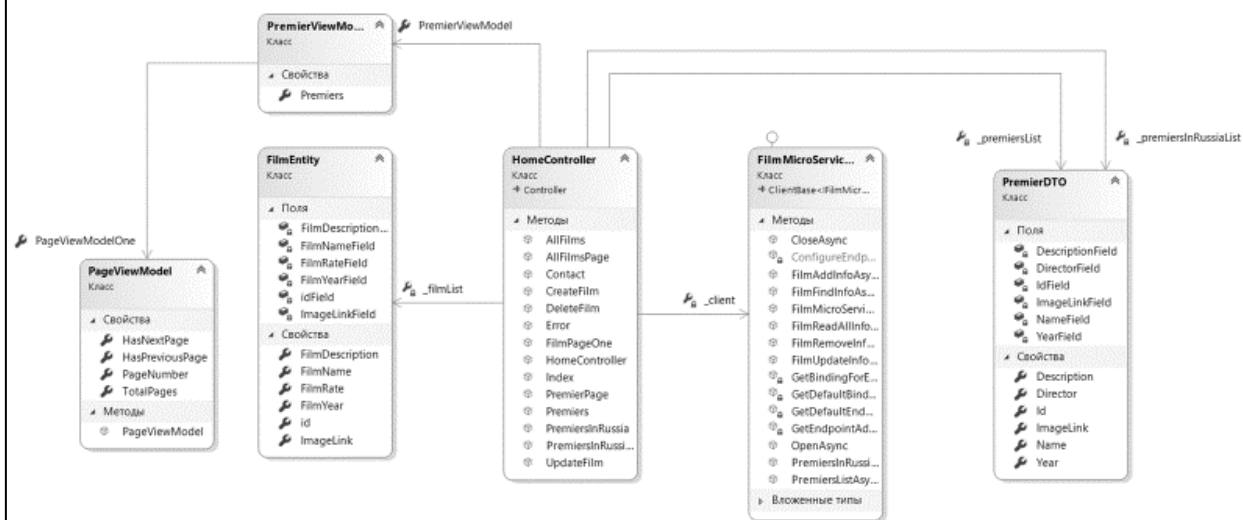
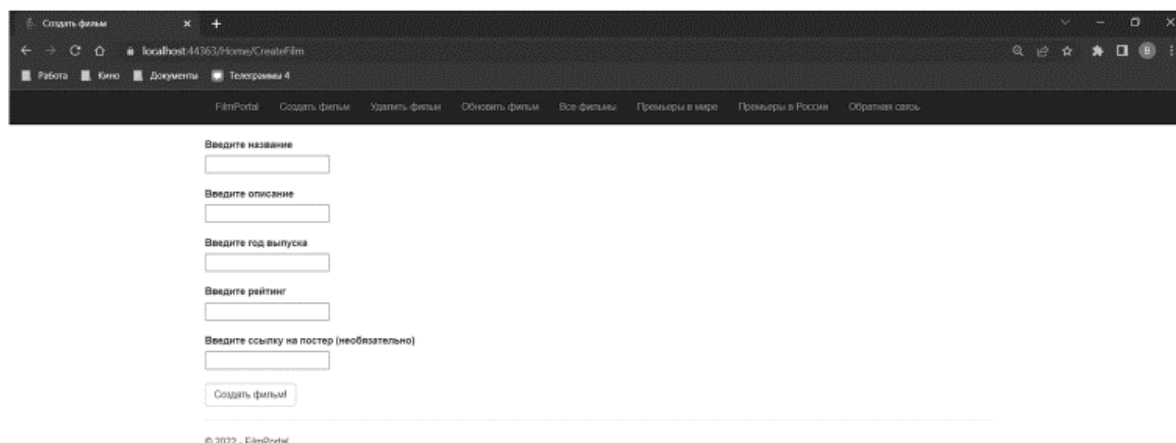


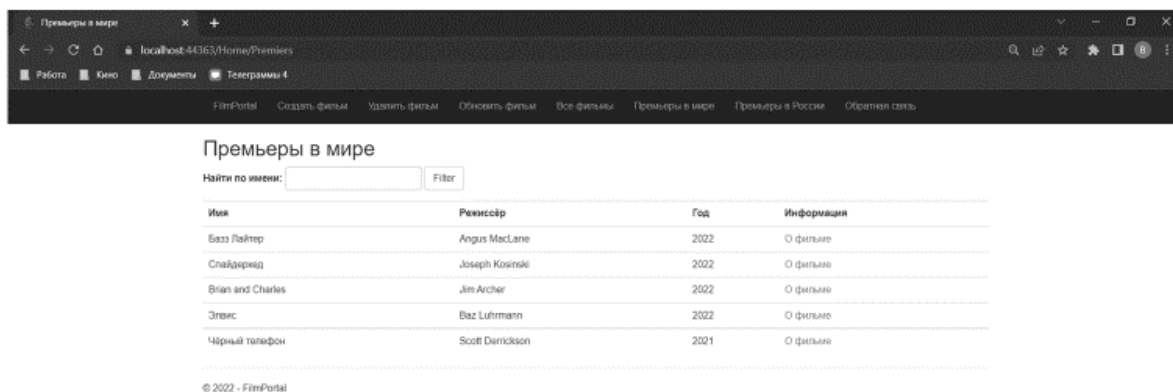
Диаграмма классов



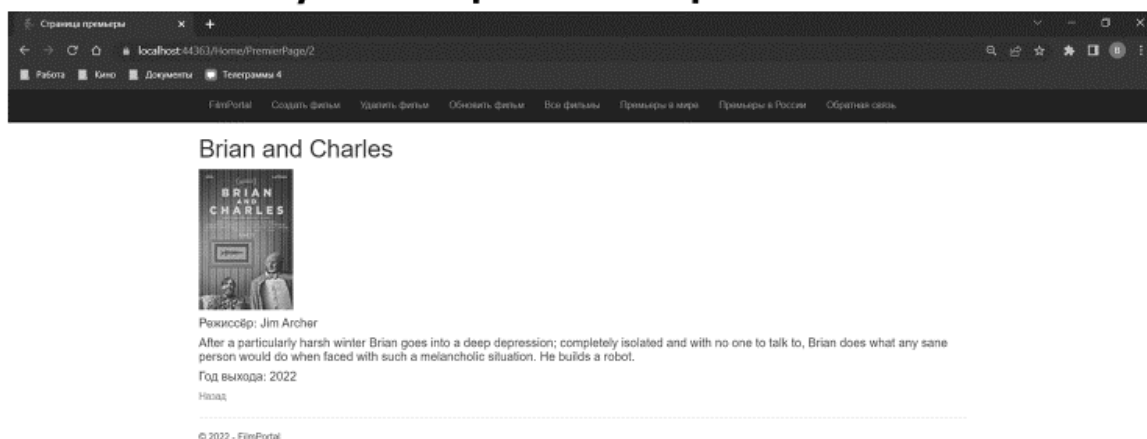
Результат работы приложения



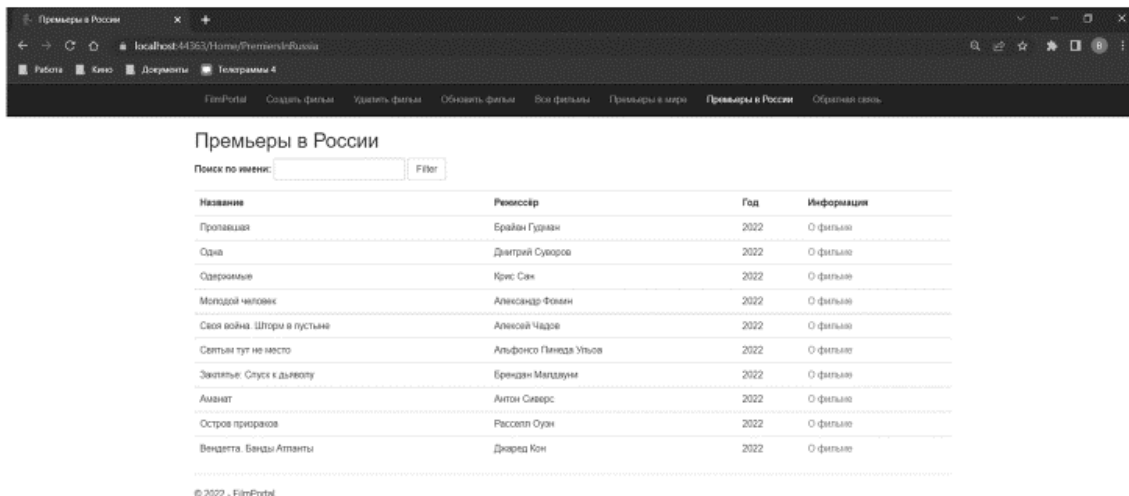
Результат работы приложения



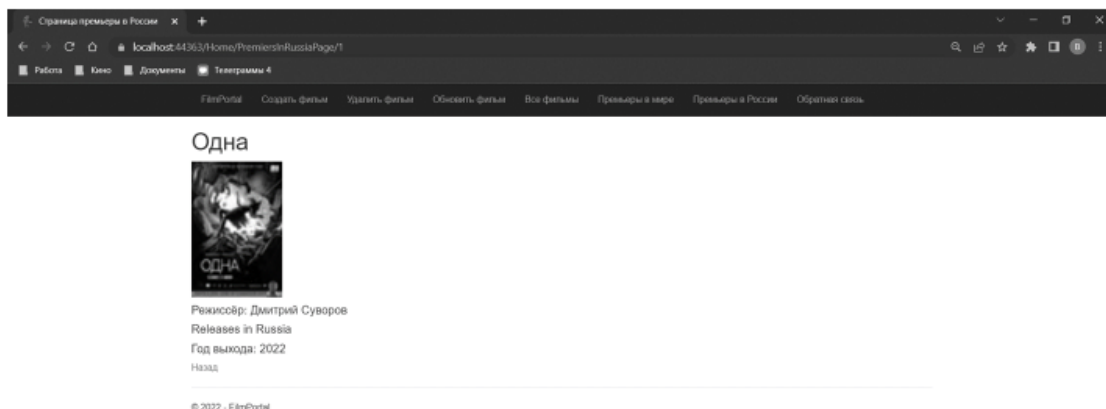
Результат работы приложения



Результат работы приложения



Результат работы приложения



Заключение

В ходе выполнения данной работы был разработан микросервис учёта сведений о фильмах на языке C# с использованием библиотеки WCF. Был получен опыт разработки масштабируемых микросервисов и разработки веб-приложений на ASP.NET Core MVC.

При реализации поставленной задачи были получены следующие результаты:

- реализован механизм обмена информацией с БД;
- реализован парсинг данных с сайтов Кинопоиск и IMDb;
- разработано веб-приложение для тестирования микросервиса.

Спасибо за внимание!

					<i>ПГУ1.090301.14.001 ПЗ</i>	Лист
Изм.	Лист	№ докум.	Подпись	Дата		114