Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет инженерно-экономический

Кафедра экономической информатики

Дисциплина «Программирование сетевых приложений»

|  |  |
| --- | --- |
|  | «К ЗАЩИТЕ ДОПУСТИТЬ» |
|  | Руководитель курсового проекта  Ассистент кафедры ЭИ  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.И. Булыга |
|  | \_\_\_.\_\_\_\_.2023 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

на тему:

**«**Сервис оценки кинофильмов**»**

БГУИР КР 1-40 01 02-08 006 ПЗ

|  |  |
| --- | --- |
|  | Выполнил студент группы 073601  Гайсенок Данила Павлович  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  | Курсовая работа представлена на проверку \_\_\_.\_\_\_\_.2023  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |

Минск 2023

Содержание

[Перечень условных обозначений, символов и терминов 6](#_Toc134009668)

[ВВЕДЕНИЕ 8](#_Toc134009669)

[1 АНАЛИЗ И МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 9](#_Toc134009670)

[1.1 Описание предметной области 9](#_Toc134009671)

[1.2 Разработка функциональной модели предметной области 10](#_Toc134009672)

[1.3 Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований 12](#_Toc134009673)

[1.4 Разработка информационной модели предметной области 13](#_Toc134009674)

[1.5 Модели представления программного средства и их описание 15](#_Toc134009675)

[2 КОНСТРУИРОВАНИЕ И ПРОЕКТИРОВАНИЕ 18](#_Toc134009676)

[2.1 Постановка задачи 18](#_Toc134009677)

[2.2 Архитектурные решения 18](#_Toc134009678)

[2.3 Описание алгоритмов, реализующих бизнес логику разрабатываемого программного средства 20](#_Toc134009679)

[2.4 Проектирование пользовательского интерфейса 24](#_Toc134009680)

[2.5 Обоснование выбора компонентов и технологий для реализации программного средства 26](#_Toc134009681)

[3 ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ 28](#_Toc134009682)

[4 ИНСТРУКЦИЯ ПО РАЗВЕРТЫВАНИЮ И ИСПОЛЬЗОВАНИЮ 30](#_Toc134009683)

[4.1 Руководство по развертыванию 30](#_Toc134009684)

[4.2 Руководство пользователя 30](#_Toc134009685)

[ЗАКЛЮЧЕНИЕ 40](#_Toc134009686)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 41](#_Toc134009687)

[ПРИЛОЖЕНИЕ А 42](#_Toc134009688)

[ПРИЛОЖЕНИЕ Б 43](#_Toc134009689)

[ПРИЛОЖЕНИЕ В 58](#_Toc134009690)

# **Перечень условных обозначений, символов и терминов**

|  |  |
| --- | --- |
| БД (база данных) | – представленная в объективной форме совокупность самостоятельных материалов, систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины |
| Информационно-коммуникационные технологии | – процессы, методы поиска, сбора, хранения, обработки, предоставления, распространения информации и способы осуществления таких процессов и методов |
| Информационная система | – система, предназначенная для хранения, поиска и обработки информации, и соответствующие организационные ресурсы (человеческие, технические, финансовые), которые обеспечивают и распространяют информацию |
| Нормальная форма | – свойство отношения в реляционной модели данных, характеризующее его с точки зрения избыточности, потенциально приводящей к логически ошибочным результатам выборки или изменения данных |
| Среда выполнения | – вычислительное окружение, необходимое для выполнения компьютерной программы и доступное во время выполнения компьютерной программы |
| СУБД (система управления базами данных) | – совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных |
| ТЗ (техническое задание) | – документ, содержащий требования заказчика к объекту разработки, определяющий порядок и условия её проведения |
| *API* (*application programming interface*) | – описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой |
| *АО* | – аппаратное обеспечение |
| *ПО* | – программное обеспечение |
| *IDE* (*Integrated development environment*) | – комплекс программных средств, используемый программистами для разработки программного обеспечения |
| *IDEF* | – методология функционального моделирования (англ. *function modeling*) и графическая нотация, предназначенная для формализации и описания бизнес-процессов |
| *Java* | – строго типизированный объектно-ориентированный язык программирования общего назначения, разработанный компанией *Sun* *Microsystems* |
| *HTML* (*HyperText Markup Language*) | – стандартизированный язык разметки документов во Всемирной паутине |
| *HTTP* (*HyperText Transfer Protocol*) | – протокол прикладного уровня передачи данных изначально – в виде гипертекстовых документов в формате *html*, в настоящий момент используется для передачи произвольных данных. |
| *SQL* (*structured query language*) | – язык структурированных запросов, декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных |
| *MySQL Server* | – свободная реляционная система управления базами данных |
| *UML* (*Unified Modeling Language*) | – язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур |
| *URL* (*Uniform Resource Locator*) | – система унифицированных адресов электронных ресурсов, или единообразный определитель местонахождения ресурса |
| *Usability* | – способность продукта быть понимаемым, изучаемым, используемым и привлекательным для пользователя в заданных условиях |

# **ВВЕДЕНИЕ**

На сегодняшний день информационные технологии играют очень важную роль для каждого человека. Любая сфера жизни людей зависит от компьютеров, мобильных телефонов, современного ПО и т.д. Эти устройства и программы позволяют общаться на больших расстояниях, обмениваться фото- и видео сообщениями, текстовыми и другими файлами.

Очень легко отслеживать предпочтения пользователей через разнообразные сервисы оценки разных аспектов жизни, начиная от оценки магазинов рядом с домом на картах, также существуют сервисы оценки специалистов в поликлиниках. Все описанные сервисы помогают сделать выбор в пользу лучшего магазина, специалиста или же кинофильма.

В современном киномире очень много фильмов. Перед тем, как начать смотреть фильм, зритель тратит много времени на выбор кинокартины. Поэтому как никогда будет актуально создать сервис для оценки кинофильмов. Именно на его основе зритель может сделать долгожданный выбор.

Данное приложение будет предназначено для помощи при выборе фильма, а также дальнейшей его оценки

Целевая аудитория ̶ это люди, которые не могут самостоятельно определиться с выбором, которым необходим сервис для быстрого поиска кинофильмов.

Целью данного курсового проекта является создание сервиса, позволяющего упростить процесс поиска и оценивания кинокартин.

Для достижения поставленной цели необходимо решить ряд следующих задач:

− изучить сферу деятельности сервисов оценивания кинофильмов;

− произвести исследование работы данных сервисов;

− разработать объектную модель и спроектировать диаграмму классов;

− разработать пользовательские функции;

− предусмотреть разделение функционала системы на этапе

авторизации пользователя;

− создать программное средство;

− обеспечить хранение данных в базе данных;

− протестировать программу, исправить возникшие ошибки и предложить способы по оптимизации и дальнейшему развитию данной программы;

# **АНАЛИЗ И МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ**

**ПРОГРАММНОГО СРЕДСТВА**

* 1. **Описание предметной области**

Из всех видов искусства кино занимает уникальное место в современном мире, а соответственно и в жизни человека. Кинематограф влияет на жизнь общества, формируя сознание зрителя. Кино предлагает человеку посмотреть на жизнь немного под другим углом, чем он ее воспринимал до того.

Кинематограф действенно влияет на человека и занимает в его жизни определенное, важное место. Поэтому система оценки фильмов также играет очень важную роль в нашей жизни и в искусстве в целом.

В современном мире получили большое распространение сервисы оценки разных аспектов жизни. Такие сервисы позволяют оценивать на основе бальной системы: 1 – самая низкая оценка, означающая полную неудовлетворенность оказанной услугой, качеством сервиса или же каким-либо продуктом, 5 – (для некоторых случаев 10 баллов) – самая высокая оценка, которая означает, что пользователь был полностью удовлетворен данной услугой, сервисом или продуктом.

Классическая система оценки, ориентированная на массовую аудиторию и применяемая сегодня на всех ресурсах, посвящённых кинематографу, представляет собой простой алгоритм, позволяющий пользователю оценить фильм, поставив ему от 1 до 10 звёздочек.

Подобными системами оценки пользуются популярные сервисы проката кинофильмов онлайн, называемые стриминговыми. Данные сервисы предоставляют возможность просмотра фильмов по окончании просмотра есть возможность оставить отзыв о фильме, также предложено оставить отметку в баллах данному фильму.

Стриминговые платформы окончательно закрепили логику потребления кино, которую сформировали у нас IMDB и RottenTomatoes. Судьба фильмов и их культурный статус во многом зависят от усредненного балла, который выводится из оценок зрителей и критиков. Использование таких рейтинговых систем неизбежно — рядовой зритель не найдет достаточно времени в сутках, чтобы каждый раз вдумчиво изучать рецензии перед просмотром.

Сам процесс оценивания фильма проходит следующим образом: пользователю необходимо выбрать фильм, который он хотел бы оценить. Затем происходит ознакомление с дополнительной информацией по этому фильму. После полного ознакомления с кинофильмом пользователь может оценить его, а также оставить свой отзыв по данному произведению.

* 1. **Разработка функциональной модели предметной области**

Процесс работы сервиса по оценке кинофильмов можно разделить на следующие этапы:

* Получение данных от пользователя и вход в аккаунт
* Выбор и оценка фильма
* Добавление комментария

Рассматриваемые в данном курсовом проекте бизнес-процессы поликлиники, связанные с обслуживанием пациентов, построением отчетов и принятием управленческих решений, можно представить в виде системы, спроектированной в *CASE*-средстве *AllFusion* *Process* *Modeler* с точки зрения стандарта *IDEF*0.

В качестве входных параметров выступают данные о клиенте и о фильмах. Управляющее воздействие оказывают критерии оценки, рейтинг фильмов и закон о защите персональных данных. Механизм осуществления составляют пользователь и система оценки кинофильмов. Выходным процессом является оценка фильма в приложении. Функциональная модель представлена на рисунках 1.1 – 1.4

|  |
| --- |
|  |
| Рисунок 1.1 – Верхний уровень функциональной модели |
|  |
| Рисунок 1.2 – Декомпозиция контекстной диаграммы |
|  |
| Рисунок 1.3 – Декомпозиция блока «Войти в аккаунт в системе» |
|  |
| Рисунок 1.4 – Декомпозиция блока «Выбрать и оценить фильм в системе» |

Перед проектированием программного средства следует определить, какие требования должны быть предъявлены к нему, так как невыполнение некоторых требований, которые были сформированы еще на ранней стадии, говорит о том, что разработанный сервис не сможет эффективно и с максимальной пользой использоваться, как было задумано изначально. [3]

* 1. **Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований**

Для разработки программного средства необходимо понять принцип работы по оценке кинофильмов в онлайн сервисе.

Так как сервисы по оценке кинофильмов для оценки фильмов используют оценки, оставленные пользователями, по которым происходит ранжирование фильма, то необходимо, чтобы был удобный сервис для обработки данной информации.

Для хранения информации необходимо создать базу данных, в которой будет содержаться вся информация о пользователях, фильмах и критиках. Для этого будет использован MySQL Server. Подключение к нему будет осуществляться через сервер, на котором будут обрабатываться запросы от пользователей.

Суть диаграммы вариантов использования состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью вариантов использования. Диаграмма вариантов использования программного средства, реализованного в данном проекте, представлена на рисунке 1.5

|  |
| --- |
|  |
| Рисунок 1.5 – Диаграмма вариантов использования |

Для взаимодействия с программным средством существует 2 роли: Администратор и пользователь. Набор предоставленных им возможностей отличается.

Администратор имеет полный доступ к информации и может добавлять, удалять и редактировать данные пользователей.

Пользователь может добавлять данные о своей рецензии и изменять ее, также пользователь имеет доступ к списку фильмов и другим рецензиям, оставленным ранее.

* 1. **Разработка информационной модели предметной области**

При проектировании системы было принято решение использовать следующие сущности

* Users
* User\_role
* Role
* Mark
* Movie
* Critics

Графическое отображение информационной модели представлено на рисунке 1.6

|  |
| --- |
|  |
| Рисунок 1.6 – Информационная модель |

Ниже приведено подробное описание атрибутов каждой сущности, входящей в модель.

Сущность Users содержит атрибуты:

id\_users – хранит уникального идентификатора;

Active – хранит отметку, может ли пользователь входить в систему;

Bio – хранит краткую информацию о пользователе;

First\_name – хранит имя пользователя;

Last\_name – хранит фамилию пользователя;

Password – хранит пароль для входа в систему (в зашифрованном виде);

Phone – хранит номер телефона пользователя;

Username – хранит логин для входа в систему;

Age – хранит возраст пользователя;

Сущность User\_role наследуется от сущностей Users и Role, содержит атрибуты:

id\_user – унаследованный от сущности Users идентификатор;

id\_role – унаследованный от сущности Role идентификатор

Сущность Role содержит атрибуты:

id\_role - уникальный идентификатор;

Role – хранит роль пользователя;

Сущность Movie содержит атрибуты:

id – уникальный идентификатор;

Title – хранит название фильма;

Genre – хранит жанр фильма;

Country – хранит название страны производителя фильма;

Director – хранит имя режиссера фильма;

Description – хранит описание фильма;

Сущность Critics содержит атрибуты:

id – уникальный идентификатор;

Name – хранит имя критика;

Surname – хранит фамилию критика;

Specialty – хранит специальность критика;

University – хранит название университета, оконченного критиком;

Experience – хранит опыт критика;

Сущность Mark наследуется от сущностей Critics, Movie и содержит атрибуты:

id – уникальный идентификатор;

Comment – хранит комментарий к рецензии;

Title – хранит заголовок рецензии;

Duration – хранит продолжительность фильма;

Rating – хранит рейтинг, выставленный в данной рецензии;

Critics\_id – хранит уникальный идентификатор критика, который написал данную рецензию;

Movie\_id - хранит уникальный идентификатор фильма.

*SQL*-скрипт для генерации базы данных приведен в приложении В.

* 1. **Модели представления программного средства и их описание**
     1. Диаграмма состояний. Для моделирования поведения на логическом уровне в языке UML могут использоваться сразу несколько канонических диаграмм: состояний, деятельности, последовательности и кооперации, каждая из которых фиксирует внимание на отдельном аспекте функционирования системы. [10]

Диаграмма состояний. Диаграмма состояний по существу является графом специального вида, который представляет некоторый автомат. Вершинами этого графа являются состояния и некоторые другие типы элементов автомата (псевдосостояния), которые изображаются соответствующими графическими символами. Дуги графа служат для обозначения переходов из состояния в состояние. Диаграммы состояний могут быть вложены друг в друга, образуя вложенные диаграммы более детального представления отдельных элементов модели. Для понимания семантики конкретной диаграммы состояний необходимо представлять не только особенности поведения моделируемой сущности, но и знать общие сведения по теории автоматов. [2, 4]

|  |
| --- |
| D:\Универ\ПСП\Курсовой\Диаграмма.jpg |
| Рисунок 1.7 – Диаграмма состояний |

* + 1. Диаграмма последовательности. На диаграмме последовательности показан процесс составления заказа. Когда пользователь посылает данные на сервер, сервер принимает данные, отправляет запрос к БД на добавление данных о заказах. Получив ответ, что процедура добавления успешна сервер отправляет ответ об успешности операции. После этого пользователю выводится сообщение об успешности операции.

|  |
| --- |
|  |
| Рисунок 1.8 Диаграмма последовательности |

* + 1. Диаграмма развёртывания. На диаграмме отражено, что для ПК пользователя обязательно наличие среды выполнения *jdk* 1.8+, также, как и для ПК, который будет выступать в качестве сервера. Взаимодействие между приложением клиента и сервера будет осуществляться посредством *TCP*/*IP* соединения. Для соединения с базой данных необходимо подключение *JDBC* *Driver*. База данных хранится на *MySQL* *SERVER*.

|  |
| --- |
|  |
| Рисунок 1.9 Диаграмма развёртывания |

1. **КОНСТРУИРОВАНИЕ И ПРОЕКТИРОВАНИЕ**

**ПРОГРАММНОГО СРЕДСТВА**

* 1. **Постановка задачи**

Для разработки программного средства необходимо понять принцип работы сервиса по оценке кинофильмов.

Так как каждый пользователь может иметь собственное мнение об том или ином фильме, то необходимо чтобы каждый пользователь мог оставить оценку или отзыв о фильме в удобном сервисе.

Для решения этих задач необходимо создать базу данных, в которой будет содержаться информация об пользователях, фильмах, рецензиях, в электронном виде. Это позволит облегчить поиск и работу с данными для пользователя и администратора.

Для корректной работы необходимо предусмотреть возможность добавления, удаления и редактирования информации в базе данных. Для хранения информации будет использован *MYSQL* *Server*. Подключение к нему будет осуществляться при авторизации и в зависимости от полученных прав, пользователь будет получать определённый набор возможностей для выполнения своей работы.

Администратор будет отвечать за регистрацию и работу с данными пользователей, также будет работать с критиками и фильмами, на которое будут писаться рецензии:

* Добавление
* Удаление
* Редактирование

Пользователю необходимо работать с фильмами, которые присутствуют в каталоге для их оценки, видеть другие оценки, и иметь возможность более подробно с ними ознакомиться.

* 1. **Архитектурные решения**

Диаграммы классов используются при моделировании программных средств наиболее часто. На диаграммах классов показываются классы, интерфейсы и отношения между ними.

В данном курсовом проекте используются 7 пакетов с классами:

Config – в этом пакте содержатся 3 класса для конфигурации проекта. Реализованы классы: MvcConfig, SecurityConfig, WebSecurityConfig. Класс MvcConfig реализует добавление контроллеров, класс SecurityConfig отвечает за создание Bean’ов для реализации шифрования и создания сессий. Класс WebSecurityConfig используется для использования пакета org.springframework.security и настройки внутренних классов.

Controller – в этом пакете содержатся классы-контроллеры для организации взаимодействия с web-страницами.

Entity – этот пакет содержит в себе POJO-классы для работы с таблицами базы данных.

Mail – содержит классы для отправки логина и пароля на электронную почту.

Repository – классы-репозитории для работы с объектами сущностей.

Service – содержит пакет impl для реализации интерфейсов сервисов и сами интерфейсы сервисов.

|  |
| --- |
|  |
| Рисунок 2.1 Диаграмма классов |

Диаграмма компонентов описывает особенности физического представления системы. Она позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный и исполняемый код. [6,7] Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и зависимости между ними. Диаграмму можно увидеть на рисунке 2.2.

|  |
| --- |
|  |
| Рисунок 2.2 Диаграмма компонентов |

* 1. **Описание алгоритмов, реализующих бизнес логику разрабатываемого программного средства**

Для входа в систему необходимо пройти авторизацию на первом окне, после прохождения авторизации, в зависимости от прав доступа откроется окно администратора или пользователя. Администратор в данном алгоритме участвует в качестве механизма одобрения или отклонения заказа, после его создания. Клиент входит в программу через механизм авторизации, далее для заказа оборудования необходимо выбрать компанию(фирму), связанную с данным пользователем от имени которой будет производится заказ. Далее происходит выбор типа аппаратного обеспечения и его количество. После чего происходит отправка заказа на одобрение.

|  |
| --- |
|  |
| Рисунок 2.3 Первая часть схемы алгоритма |

Рассмотрим ветку работа администратора

После авторизации под ролью администратор открывается окно для выбора функции с которой будет производиться работа.

Для администратора доступны варианты функций для работы с рецензиями (добавление, удаление, редактирование информации), критиками и фильмами (добавление, удаление, обновление информации), просмотр пользователей и работа с ними (будет рассмотрено далее). Работа администратора представлена на рисунке 2.4.

Также рассмотрим работу пользователя. Для пользователей в данной системе доступна работа с рецензиями (добавление, редактирование информации о них), работа с Фильмами (просмотр и оценка фильмов),

Рассмотрим реализацию алгоритма добавления рецензии пользователем и работы с пользователями администратором.

Далее рассмотрим схему работы алгоритма с позиции администратора (см. рисунок 2.4)

Процесс входа в систему под ролью администратора был показан на рисунке 2.3. Работа администратора на данном участке заключается в том, что он переходит на экран с пользователями, для работы с ними. Далее происходит выбор конкретного пользователя, с которым будет происходить работа: администратор выбирает заблокировать пользователя или отредактировать информацию о нем. При нажатии кнопки бан статус пользователя становится «неактивен», при нажатии кнопки редактировать открывается окно редактирования информации.

|  |
| --- |
|  |
| Рисунок 2.4 Реализация алгоритма со стороны администратора |

Рассмотрим схему реализации алгоритма со стороны пользователя (см. рисунок 2.5)

Для создания рецензии пользователь, прошедший авторизацию (см. рисунок 2.3), выбирает фильм, на который будет создана рецензия, далее, открывается окно создания рецензий в котором выбирается фильм, вводится его оценка. После вводится комментарий и выбирается критик из списка и добавляется рецензия.

|  |
| --- |
|  |
| Рисунок 2.5 Реализация алгоритма со стороны пользователя |
| Рисунок 2.6 Конец блок-схемы |

* 1. **Проектирование пользовательского интерфейса**

Пользовательский интерфейс – это система средств для взаимодействия пользователя с компьютером, основанная на представлении всех доступных пользователю системных объектов и функций в виде графических компонентов экрана (окон, значков, меню, кнопок, списков и т.п.). При этом, в отличие от интерфейса командной строки, пользователь имеет произвольный доступ (с помощью клавиатуры или указательного устройства ввода) ко всем видимым экранным объектам, а на экране реализуется модель мира в соответствии с некоторой метафорой и осуществляется прямое манипулирование. [9]

Приложение имеет графический интерфейс, реализованный при помощи html страниц.

Все компоненты взаимодействуют друг с другом и связаны. Так же могут работать по отдельности. Для каждой части разработано графическое представление данных.

|  |
| --- |
|  |
| Рисунок 2.7 Интерфейс администратора |

Для администратора доступно управление пользователями путем модификации записей в графическом интерфейсе, изменения заносятся в базу данных. Также администратор может управлять записями о фильмах, критиках и рецензиях.

Для пользователя доступно управление рецензиями, которые были созданы, с помощью внесения и изменения данных в интерфейсе. Интерфейс пользователя представлен на рисунке 2.8.

Для входа в систему необходимо пройти авторизацию. Окно авторизации открывается при первом запуске программы имеет следующую функциональность:

Для входа используются два текстовых поля для ввода логина и пароля, для отправки данных на проверку используется кнопка «Log in».

По нажатию на кнопку «Регистрация» открывается окно, в котором можно зарегистрировать нового пользователя. Интерфейс окна авторизации представлен на рисунке 2.9, окна регистрации – на рисунке 2.10.

|  |
| --- |
|  |
| Рисунок 2.8 Интерфейс пользователя |
|  |
| Рисунок 2.9 Интерфейс окна авторизации |
|  |
| Рисунок 2.10 Интерфейс окна регистрации |

* 1. **Обоснование выбора компонентов и технологий для реализации программного средства**

Программное приложение должно быть разработано на объектно-ориентированном языке *Java* и использовать архитектуру «web-приложение с базой данных». Web-приложения состоят из двух частей – клиентской и серверной. Клиентская часть представляет собой web-страницы в браузере для взаимодействия с пользователем. Серверная часть скрыта от глаз пользователей, представляет обработку запросов HTTP и отвечает за связь с базой данных.

Веб-серверы – программы, которые реализуют определённый функционал для обработки HTTP запросов и связи с базами данных.

Клиенты – веб-браузеры, которые предоставляют пользователям веб-страницы по запросу. Общение с серверами происходит путем HTTP запросов и ответов.

Таким образом, архитектура «web-приложение» позволяет эффективно распределить работу между клиентскими и серверными частями системы: приложения, которые работают на стороне клиента не читают записи базы данных «напрямую», а посылают запросы на сервер, где они обрабатываются, результаты обработки отсылаются назад клиенту, что сокращает потоки информации.

|  |
| --- |
|  |
| Рисунок 2.11 Архитектура web-приложения с базой данных |

Бизнес-логика курсового проекта выполняется на серверной части. Клиент отправляет запросы на сервер, где с помощью методов, предоставленных Spring-data-JPA будет добавляться, обновляться, удаляться и выбираться вся необходимая информация из БД.

Серверная часть приложения реализована на Spring Framework как удобный набор библиотек и классов для web-разработке на языке программирования на java. Также Spring предоставляет набор библиотек для организации авторизации в web-приложении. Основной средой разработки был выбран *IDE IntelliJ IDEA*.

В качестве СУБД используется *MySQL Server как наиболее удобная в плане совместимости с Spring framework*. Обладает высокой скоростью обработки запросов, является одной из самых популярных и востребованных СУБД в мире.

*Enterprise Architect* используется для разработки и построения *UML*-диаграмм (*Use Case, Statechart, Sequence diagram*, диаграмма классов, *Component diagram, Deployment diagram*). *CASE-*средство *CA AllFusion Process Modeler r*7 (*BPwin*) используется для проектирование *IDEF*0 модели. *CA AllFusion ERwin Data Modeler r*7 (*ERwin*) используется для проектирования информационной модели.

1. **ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ**

**ПРОГРАММНОГО СРЕДСТВА**

Тестирование – это процесс выполнения программы, целью которого является выявление ошибок.

Для тестирования разрабатываемого приложения был использован J-Unit. J-Unit – библиотека для модульного тестирования ПО на java.

Преимущество модульного тестирования заключается в тестировании отдельного модуля независимо от других.

Было протестировано взаимодействие сервера с таблицей Critics. Для этого был создан класс CriticsServiceImplTest.

@RunWith(SpringRunner.class)  
@SpringBootTest  
@ContextConfiguration(classes = {CriticsServiceImpl.class})  
public class CriticsServiceImplTest {  
 @Autowired  
 private CriticsService criticsService;  
 @MockBean  
 private CriticsRepository criticsRepository;  
 private final int id = 11;  
 private final String name = "danila";  
 private final List<Critics> list;  
 private Critics inputCritics;  
 private Critics outputCritics;  
  
  
 public CriticsServiceImplTest() {  
 inputCritics = Critics.builder()  
 .id(id)  
 .name(name)  
 .build();  
 outputCritics = Critics.builder()  
 .id(id)  
 .name(name)  
 .build();  
 list = Arrays.asList(outputCritics);  
 }  
  
 @Test  
 public void findAllCritics\_success() {  
 when(criticsRepository.findAll()).thenReturn(list);  
 assertEquals(list, criticsService.findAllCustomers());  
 }  
  
 @Test  
 public void findCriticById\_success() {  
 when(criticsRepository.findById(id)).thenReturn(Optional.of(outputCritics));  
 Assertions.assertEquals(outputCritics, criticsService.findById(id));  
 }  
  
 @Test  
 public void removeCritic\_success() {  
 when(criticsRepository.findById(id)).thenReturn(Optional.of(inputCritics));  
 doNothing().when(criticsRepository).delete(inputCritics);  
 criticsService.removeCustomer(id);  
 verify(criticsRepository).delete(inputCritics);  
 }  
  
 @Test  
 public void findCriticById\_invalidIndex() {  
 when(criticsRepository.findById(id)).thenReturn(null);  
 assertThrows(NullPointerException.class, ()-> criticsService.findById(id));  
 }  
  
}

|  |
| --- |
|  |
| Рисунок 3.1 Результаты тестирования взаимодействия сервера с таблицей |

# **ИНСТРУКЦИЯ ПО РАЗВЕРТЫВАНИЮ И ИСПОЛЬЗОВАНИЮ**

**ПРИЛОЖЕНИЯ**

## **4.1 Руководство по развертыванию**

Для развертывания приложения установите Docker и Docker-compose, и выполните в консоли/в терминале команду docker-compose up –b. После развертывания приложение будет полностью готово к использованию.

## **4.2 Руководство пользователя**

В первую очередь необходимо запустить MovieMarkingApplication открыв проект в IntellijIdea или запустить MovieMarkingApplication.java из консоли.

После запуска нужно перейти в браузер и ввести в поисковую строку localhost:8088. Далее откроется окно авторизации (см. рисунок 4.1). Пройдем авторизацию как Администратор, после чего откроется окно работы администратора (см. рисунок 4.2).

|  |
| --- |
|  |
| Рисунок 4.1 Окно авторизации |

Для прохождения авторизации необходимо ввести логин и пароль, которые будут отправлены на сервер для прохождения валидации, после прохождения которой, в зависимости от прав доступа, откроется окно администратора или клиента.

|  |
| --- |
|  |
| Рисунок 4.2 Окно администратора |

У администратора есть инструменты управления пользователями: редактирование, удаление и изменение статуса аккаунта. Начнем с редактирования информации о пользователях. Для этого необходимо нажать справа вверху на кнопку «Пользователи», выбрать пользователя для редактирования информации о нем и нажать «Просмотр». После нажатия на кнопку откроется окно (см. рисунок 4.3) с подробной информацией о пользователе.

|  |
| --- |
|  |
| Рисунок 4.3 Окно подробной информации о пользователе |

Изменим имя пользователя. Для этого необходимо нажать кнопку «Редактировать» и, в открывшемся окне, ввести новое имя (см. рисунок 4.4).

|  |
| --- |
|  |
| Рисунок 4.4 Окно редактирования информации о пользователях |

Результат редактирования информации о пользователе представлен на рисунке 4.5. Работая с пользователем можно изменить статус его аккаунта, Нажав на кнопку «Бан» (см. рисунок 4.6-4.7), или удалить запись о нем (см. рисунок 4.8).

|  |
| --- |
|  |
| Рисунок 4.5 Информация о пользователе после редактирования |
|  |
| Рисунок 4.6 Расположение кнопки «Бан» |
|  |
| Рисунок 4.7 Информация о пользователе после нажатия кнопки «Бан» |
|  |
| Рисунок 4.8 Пользователи после процедуры удаления |

Для добавления пользователя необходимо нажать на синюю кнопку «Добавить пользователя» вверху слева, после чего откроется окно добавления пользователя (см. рисунок 4.9). При регистрации пользователя Логин и пароль высылаются на почту, которая будет введена в поле «Почта» (см. рисунок 4.10).

|  |
| --- |
|  |
| Рисунок 4.9 Окно добавления пользователя |
|  |
| Рисунок 4.10 Логин и пароль на почте |

При нажатии на вкладку «Фильмы» откроется окно с информацией о фильмах (см. рисунок 4.11). Далее добавим фильм нажав на кнопку «Добавить фильм» для которого в качестве пользователя будем создавать рецензию (см. рисунок 4.12). Создав фильм прейдем к созданию критика, которого можно будет указать в рецензии (см. рисунок 4.13).

|  |
| --- |
|  |
| Рисунок 4.11 Информация о фильмах |
|  |
| Рисунок 4.12 Окно пользователя |
|  |
| Рисунок 4.13 Добавление критика |

После добавления всех необходимых компонентов перейдем в окно пользователя по логину и паролю, которые были высланы на почту (см. рисунок 4.14). Перейдем к созданию рецензии на фильм. Для этого необходимо перейти во вкладку «Добавить рецензию» (см. рисунок 4.15). Далее введём информацию о фильме, добавим комментарий к этому фильму, введем рейтинг и др. информацию (см. рисунок 4.16).

|  |
| --- |
|  |
| Рисунок 4.14 Окно пользователя |
|  |
| Рисунок 4.15 Добавление рецензии |
|  |
| Рисунок 4.16 Введение информации |

Также есть возможность провести экспертную оценку фильма со стороны администратора (см. рисунок 4.17). После расчета выдается оценка весомости каждого параметра и выдается самый весомый параметр (см. рисунок 4.18).

|  |
| --- |
|  |
| 4.17 Окно ввода оценок перед расчетом |
|  |
| 4.18 Оценка фильма |

**ЗАКЛЮЧЕНИЕ**

В результате выполнения данного курсового проекта было разработано программное приложение, позволяющие проводить работу по оценке кинофильмов. В нем предусмотрена система администрирования, а также проверка учетных данных при авторизации пользователей.

Приложение удовлетворяет основным характеристикам, которые были заявлены. Оно удобно в эксплуатации, целостно, конкретизировано в рамках заданной предметной области.

При разработке данного программного продукта была учтена логика пользователя, и интерфейс данной программы сделан удобным и понятным.

Выполнено проектирование и разработка программного средства планирования и оценки нагрузки персонала медицинского учреждения: выполнена постановка задачи и определены основные методы ее решения; в ходе объектного моделирования системы построен ряд *UML*-диаграмм; разработана информационная модель системы, представленная в виде схемы базы данных; описаны основные алгоритмы работы программного средства; разработано руководство пользователя; выполнено тестирование системы показавшее ее соответствие функциональным требованиям, поставленным в задании на разработку.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Издательство «Открытые системы» [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.osp.ru/os/2011/03/13008203

[2] Технология разработки ПО [Электронный ресурс]. – Электронные данные. – Режим доступа: https://technologiarpo.blogspot.com/p/blog-page\_16.html

[3] Дубейковский, В. И. Эффективное моделирование с AllFusion Process Modeler 4.1.4 и AllFusion PM / В. И. Дубейковский. – Москва: Нолидж, 2011. – 284 с.

[4] Леоненков, А. В. Самоучитель UML / А. В. Леоненков. – СПб.: BHV, 2014. – 304 с.

[5] Роб П., Коронел К. Системы баз данных: проектирование, реализация и управление – М: БХВ-Петербург, 2004. – 1040 с.

[6] [Бабич, А. В. Введение в UML. [Электронный ресурс]. – Режим доступа : http://www.intuit.ru/studies/courses/1007/229/info.](file:///C:\Users\Svetlana\Downloads\Бабич%20А. В. Введение%20в%20UML.%20%5bЭлектронный%20ресурс%5d.%20–%20Электронные%20данные.%20Лекции.%20–%20Режим%20доступа:%20http:\www.intuit.ru\studies\courses\1007\229\info)

[7] Басс, Л., Архитектура программного обеспечения на практике. 2-е издание / Л. Басс, П. Клементс, Р. Кацман. – СПб.: Питер, 2006. – 575 с.

[8] Харрингтон Джен Л. Проектирование реляционных баз данных – М: Лори, 2016. – 514 с.

[9] Фаулер, М. Архитектура корпоративных программных приложений / М. Фаулер. – М.: Вильямс, 2016 – 544 с.

[10] Маклаков, С. В. Книга моделирования бизнес процессов при помощи BpWin 4.0: справ. пособие / С. В. Маклов. – Москва: Диалог-Мифи, 2002. – 74 с.

**ПРИЛОЖЕНИЕ А**

(обязательное)

Отчет о проверке на заимствования в системе «Атиплагиат»

|  |
| --- |
|  |
| Рисунок А.1 Отчет о проверке на заимствования в системе «Антиплагиат» |

# **ПРИЛОЖЕНИЕ Б**

(обязательное)

Листинг кода алгоритмов, реализующих бизнес логику

package com.company.moviemarking.repository;  
  
import com.company.moviemarking.entity.Mark;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import java.util.Optional;  
  
public interface MarkRepository extends JpaRepository<Mark, String> {  
 Optional<Mark> findById(int id);  
}

package com.company.moviemarking.repository;  
  
import com.company.moviemarking.entity.Movie;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
  
import java.util.List;  
import java.util.Optional;  
  
public interface MovieRepository extends JpaRepository<Movie, String> {  
 Optional<Movie> findById(int id);  
 @Query("select m from Movie m where title like %?1%")  
 List<Movie> findByTitle(String title);  
}

package com.company.moviemarking.repository;  
  
import com.company.moviemarking.entity.Critics;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import java.util.Optional;  
  
public interface CriticsRepository extends JpaRepository<Critics, String> {  
 Optional<Critics> findById(int id);  
}

package com.company.moviemarking.repository;  
  
import com.company.moviemarking.entity.User;

Продолжение приложения Б

import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
import java.util.List;  
import java.util.Optional;  
  
@Repository  
public interface UserRepository extends JpaRepository<User, String> {  
 List<User> findAll();  
  
 List<User> findByActive(boolean active);  
  
 Optional<User> findUserById(int id);  
  
 Optional<User> findUserByUsername(String username);  
}

package com.company.moviemarking.repository;  
  
import com.company.moviemarking.entity.Role;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface RoleRepository extends JpaRepository<Role, Integer> {  
 Role findByRole(String role);  
}

package com.company.moviemarking.service.impl;  
  
import com.company.moviemarking.dto.MovieDTO;  
import com.company.moviemarking.entity.Mark;  
import com.company.moviemarking.repository.MarkRepository;  
import com.company.moviemarking.service.MarkService;  
import com.company.moviemarking.service.mapper.MarkMapper;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
import java.util.NoSuchElementException;  
import java.util.Optional;

Продолжение приложения Б

@Service  
public class MarkServiceImpl implements MarkService {  
 private MarkRepository markRepository;  
 private MarkMapper markMapper;  
  
 @Autowired  
 public MarkServiceImpl(MarkRepository markRepository, MarkMapper markMapper) {  
 this.markRepository = markRepository;  
 this.markMapper = markMapper;  
 }  
  
 @Override  
 public Mark save(MovieDTO movieDTO) {  
 Mark mark = markMapper.toEntitySup(movieDTO);  
 return markRepository.save(mark);  
 }  
 @Override  
 public Mark findById(int id) {  
 return markRepository.findById(id)  
 .orElseThrow(() -> new NoSuchElementException());  
 }  
  
 @Override  
 public List<Mark> findAllMarks() {  
 return markRepository.findAll();  
 }  
  
 @Override  
 public void removeMark(int markId) {  
 Mark mark = findById(markId);  
 markRepository.delete(mark);  
 }  
  
 public void update(Integer id, String script, Integer rating, Integer duration, String comment){  
 Optional<Mark> mark = markRepository.findById(id);  
 if (!(mark.get()==null)){  
 mark.get().setScript(script);  
 mark.get().setRating(rating);  
 mark.get().setDuration(duration);

Продолжение приложения Б

mark.get().setComment(comment);}  
 markRepository.save(mark.get());  
 }  
}

package com.company.moviemarking.service.impl;  
  
import com.company.moviemarking.entity.Critics;  
import com.company.moviemarking.repository.CriticsRepository;  
import com.company.moviemarking.service.CriticsService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
import java.util.NoSuchElementException;  
import java.util.Optional;  
  
@Service

public class CriticsServiceImpl implements CriticsService {  
 private CriticsRepository criticsRepository;  
  
 @Autowired  
 public CriticsServiceImpl(CriticsRepository criticsRepository) {  
 this.criticsRepository = criticsRepository;  
 }  
  
 @Override  
 public Critics save(Critics critics) {  
 return criticsRepository.save(critics);  
 }  
  
 @Override  
 public Critics findById(int id) {  
 return criticsRepository.findById(id)  
 .orElseThrow(()-> new NoSuchElementException());  
 }  
  
 @Override  
 public List<Critics> findAllCustomers() {  
 return criticsRepository.findAll(); }

Продолжение приложения Б

@Override  
 public void removeCustomer(int id) {  
 Critics critics = findById(id);  
 criticsRepository.delete(critics);  
 }  
  
 public void update(Integer id, String name, String surname, String specialty, String experience, String university){  
 Optional<Critics> critic = criticsRepository.findById(id);  
 if (!(critic.get()==null)){  
 critic.get().setName(name);  
 critic.get().setSurname(surname);  
 critic.get().setSpecialty(specialty);  
 critic.get().setExperience(experience);  
 critic.get().setUniversity(university);  
 }  
 criticsRepository.save(critic.get());  
 }  
}

package com.company.moviemarking.service.impl;  
  
import com.company.moviemarking.entity.Movie;  
import com.company.moviemarking.repository.MovieRepository;  
import com.company.moviemarking.service.MovieService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
import java.util.NoSuchElementException;  
import java.util.Optional;  
  
@Service  
public class MovieServiceImpl implements MovieService {  
 private MovieRepository movieRepository;  
  
 @Autowired  
 public MovieServiceImpl(MovieRepository movieRepository) {  
 this.movieRepository = movieRepository;  
 }

Продолжение приложения Б

@Override

public Movie save(Movie movie) {  
 return movieRepository.save(movie);  
 }  
  
 @Override  
 public Movie findById(int id) {  
 return movieRepository.findById(id)  
 .orElseThrow(()->new NoSuchElementException());  
 }  
  
 @Override  
 public List<Movie> findAllMovies() {  
 return movieRepository.findAll();  
 }  
 @Override  
 public void removeMovies(int id) {  
 Movie movie = findById(id);  
 movieRepository.delete(movie);  
 }  
 @Override  
 public List<Movie> listMovies(String title){  
 if (title != null) return movieRepository.findByTitle(title);  
 return movieRepository.findAll();  
 }  
  
 public void updateS(Integer id, String title, String genre, String country, String director, String description){  
 Optional<Movie> movie = movieRepository.findById(id);  
 if (!(movie.get() ==null)){  
 movie.get().setTitle(title);  
 movie.get().setGenre(genre);  
 movie.get().setCountry(country);  
 movie.get().setDirector(director);  
 movie.get().setDescription(description);  
 }  
 movieRepository.save(movie.get());  
 }  
}

Продолжение приложения Б

package com.company.moviemarking.service.impl;  
  
import com.company.moviemarking.entity.Role;  
import com.company.moviemarking.entity.User;  
import com.company.moviemarking.service.UserService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.core.authority.SimpleGrantedAuthority;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.security.core.userdetails.UsernameNotFoundException;  
import org.springframework.stereotype.Service;  
  
import javax.transaction.Transactional;  
import java.util.ArrayList;

import java.util.HashSet;  
import java.util.List;  
import java.util.Set;  
  
@Service  
public class UserDetailsServiceImpl implements UserDetailsService {  
  
 @Autowired  
 private UserService userService;  
  
 @Override  
 @Transactional  
 public UserDetails loadUserByUsername(String userName) throws UsernameNotFoundException {  
 User user = userService.findUserByUsername(userName);  
 if (user == null) {  
 throw new UsernameNotFoundException(String.format("No user found with username: %s", userName));  
 }

List<GrantedAuthority> authorities = getUserAuthority(user.getRoles());  
 return buildUserForAuthentication(user, authorities);  
 }

Продолжение приложения Б

private List<GrantedAuthority> getUserAuthority(Set<Role> userRoles) {  
 Set<GrantedAuthority> roles = new HashSet<>();  
 for (Role role : userRoles) {  
 roles.add(new SimpleGrantedAuthority(role.getRole()));  
 }  
 return new ArrayList<>(roles);  
 }

private UserDetails buildUserForAuthentication(User user, List<GrantedAuthority> authorities) {  
 return new org.springframework.security.core.userdetails.User(user.getUsername(), user.getPassword(),  
 user.isActive(), true, true, true, authorities);  
 }  
}

package com.company.moviemarking.service.impl;  
  
import com.company.moviemarking.entity.Role;  
import com.company.moviemarking.entity.User;  
import com.company.moviemarking.mail.EmailDetails;  
import com.company.moviemarking.mail.EmailService;  
import com.company.moviemarking.repository.RoleRepository;  
import com.company.moviemarking.repository.UserRepository;  
import com.company.moviemarking.service.UserService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.stereotype.Service;  
  
import java.util.HashSet;  
import java.util.List;  
import java.util.NoSuchElementException;  
  
@Service  
public class UserServiceImpl implements UserService {  
 private UserRepository userRepository;  
 private RoleRepository roleRepository;  
 private List<User> list ;  
 @Autowired private BCryptPasswordEncoder bCryptPasswordEncoder; @Autowired private EmailService emailService;

Продолжение приложения Б

@Autowired  
 public UserServiceImpl(UserRepository userRepository, RoleRepository roleRepository) {  
 this.userRepository = userRepository;  
 this.roleRepository = roleRepository;  
 this.list = userRepository.findAll();  
 }  
  
 public List<User> findAllUsers() {  
 return userRepository.findAll();  
 }  
  
 public User save(User user) {

EmailDetails details = new EmailDetails(user.getUsername(),"Добро пожаловать на сервис оценки кинофильмов. Ваш логин:" + user.getUsername() +". Ваш пароль:"+user.getPassword(), "Регистрация на сервисе", null);  
 user.setPassword(this.bCryptPasswordEncoder.encode(user.getPassword()));  
 user.setActive(true);  
 Role role = list.isEmpty()? roleRepository.findByRole("ROLE\_ADMIN") : roleRepository.findByRole("ROLE\_USER");  
 //Role role = roleRepository.findByRole("ROLE\_ADMIN");  
 user.setRoles(new HashSet<>(List.of(role)));  
 if (user.getUsername()!=null) {  
 emailService.sendSimpleMail(details);  
 }  
 return userRepository.save(user);  
 }  
  
 public void banUser(Integer id) {  
 User user = userRepository.findUserById(id).orElse(null);  
 if (user!=null){  
 if (user.isActive()){  
 user.setActive(false);  
 } else {  
 user.setActive(true);  
 }  
 }  
 userRepository.save(user);  
 }  
  
 public void update(Integer id, String firstName, String lastName, String phone,

Продолжение приложения Б

String bio, String age){  
 User user = userRepository.findUserById(id).orElse(null);  
 if (user!=null){  
 user.setFirstName(firstName);  
 user.setLastName(lastName);  
 user.setPhone(phone);  
 user.setBio(bio);  
 user.setAge(age);  
 }  
 userRepository.save(user);  
 }  
  
 @Override  
 public void removeUser(int id) {  
 User user = findUserById(id);  
 userRepository.delete(user);  
 }  
  
 @Override  
 public User findUserByUsername(String username) {  
 return userRepository.findUserByUsername(username)  
 .orElseThrow(() -> new NoSuchElementException());  
 }  
  
 @Override  
 public User findUserById(int id) {  
 return userRepository.findUserById(id)  
 .orElseThrow(()->new NoSuchElementException());  
 }  
}

package com.company.moviemarking.controller;  
  
import com.company.moviemarking.entity.Critics;  
import com.company.moviemarking.service.CriticsService;  
import com.company.moviemarking.service.impl.CriticsServiceImpl;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.ArrayList;

Продолжение приложения Б

import java.util.List;  
@Controller  
public class CriticsController {  
 private final CriticsService criticsService;  
 private final CriticsServiceImpl criticService;  
  
 @Autowired  
 public CriticsController(CriticsService criticsService, CriticsServiceImpl criticService) {  
 this.criticsService = criticsService;  
 this.criticService = criticService;  
 }  
  
 @GetMapping("/add-customer")  
 public String addCustomer(Model model) {  
 model.addAttribute(new Critics());  
 return "add-customer";  
 }  
  
 @PostMapping("/add-customer")  
 public String addCustomer(@ModelAttribute Critics critics) {  
 criticsService.save(critics);  
 return "redirect:customers";  
 }  
  
 @GetMapping("/customers")  
 public String findAllCustomers(Model model) {  
 List<Critics> critics = criticsService.findAllCustomers();  
 model.addAttribute("customers", critics);  
 return "customers";  
 }  
  
 @GetMapping("/customer/{id}")  
 public String findCustomer(@PathVariable(value = "id") int id, Model model) {  
 Critics critic = criticsService.findById(id);  
 List<Critics> critics = new ArrayList<>();  
 critics.add(critic);  
 model.addAttribute("customer", critics);  
 return "customer-details";  
 }

Продолжение приложения Б

@PostMapping("/customer/{id}/remove")  
 public String removeCustomer(@PathVariable(value = "id") int id, Model model) {  
 criticsService.removeCustomer(id);  
 return "customers";  
 }  
  
 @GetMapping("/customer/{id}/update")  
 public String updateCustomer(@PathVariable(value="id")int id, Model model){  
 Critics critic = criticsService.findById(id);  
 List<Critics> critics = new ArrayList<>();  
 critics.add(critic);  
 model.addAttribute("customer", critics);  
 return "updateCustomer";  
 }  
 @PostMapping("/customer/{id}/updateC")  
 public String updateC(@PathVariable(value = "id")int id, @RequestParam(value = "name")String name,  
 @RequestParam(value = "surname")String surname, @RequestParam(value = "specialty")String specialty,  
 @RequestParam(value = "experience")String experience, @RequestParam(value="university")String university,  
 Model model){  
 criticService.update(id,name,surname,specialty,experience,university);  
 return "redirect:/customers";  
 }  
  
}

package com.company.moviemarking.controller;  
  
import com.company.moviemarking.dto.MovieDTO;  
import com.company.moviemarking.entity.Critics;  
import com.company.moviemarking.entity.Mark;  
import com.company.moviemarking.entity.Movie;  
import com.company.moviemarking.service.MarkService;  
import com.company.moviemarking.service.MovieService;  
import com.company.moviemarking.service.impl.MarkServiceImpl;  
import com.company.moviemarking.service.mapper.MarkMapper;  
import com.company.moviemarking.service.CriticsService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;

Продолжение приложения Б

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.\*;  
  
import java.util.ArrayList;  
import java.util.List;  
  
@Controller  
public class MarkController {  
 private final MarkService markService;  
 private final MovieService movieService;  
 private final CriticsService criticsService;  
 private final MarkMapper markMapper;  
 private final MarkServiceImpl markServiceI;  
  
 @Autowired  
 public MarkController(MarkService productService, MovieService movieService,  
 CriticsService criticsService, MarkMapper markMapper,  
 MarkServiceImpl markService) {  
 this.markService = productService;  
 this.movieService = movieService;  
 this.criticsService = criticsService;  
 this.markMapper = markMapper;  
 this.markServiceI = markService;  
 }  
  
 @GetMapping("/add-product")  
 public String addProduct(Model model) {  
 List<Movie> movies = movieService.findAllMovies();  
 model.addAttribute("suppliers", movies);  
 List<Critics> critics = criticsService.findAllCustomers();  
 model.addAttribute("critics", critics);  
 model.addAttribute(new MovieDTO());  
 return "add-product";  
 }  
  
 @PostMapping("/add-product")  
 public String addProduct(@ModelAttribute MovieDTO movieDTO) {  
 markService.save(movieDTO);  
 return "add-product";  
 }

Продолжение приложения Б

@GetMapping("/sell-product")  
 public String sellProduct(Model model) {  
 List<Critics> critics = criticsService.findAllCustomers();  
 model.addAttribute("customers", critics);  
 List<Mark> marks = markService.findAllMarks();  
 model.addAttribute("products", marks);  
 model.addAttribute(new MovieDTO());  
 return "sell-product";  
 }  
  
 @PostMapping("/sell-product/{id}")  
 public String sellProduct(@ModelAttribute MovieDTO movieDTO) {  
 //markService.sellProduct(id, quantity);  
 markService.save(movieDTO);  
 return "sell-product";  
 }  
  
 @GetMapping("/products")  
 public String findAllProducts(Model model) {  
 List<Mark> marks = markService.findAllMarks();  
 model.addAttribute("products", marks);  
 return "products";  
 }  
  
 @GetMapping()  
 public String home() {  
 return "redirect:products";  
 }  
  
 @GetMapping("/product/{id}")  
 public String findProduct(@PathVariable(value = "id") int id, Model model) {  
 Mark mark = markService.findById(id);  
 List<Mark> marks = new ArrayList<>();  
 marks.add(mark);  
 model.addAttribute("product", marks);  
 return "product-details";  
 }  
  
 @PostMapping("/product/{id}/remove")  
 public String removeProduct(@PathVariable(value = "id") int id, Model model) {

Продолжение приложения Б

markService.removeMark(id);

return "products";  
 }  
 @GetMapping("/product/{id}/update")  
 public String updateProduct(@PathVariable(value="id")int id, Model model){  
 Mark mark = markService.findById(id);  
 List<Mark> marks = new ArrayList<>();  
 marks.add(mark);  
 model.addAttribute("product", marks);  
 List<Movie> movies = movieService.findAllMovies();  
 model.addAttribute("suppliers", movies);  
 List<Critics> critics = criticsService.findAllCustomers();  
 model.addAttribute("critics", critics);  
 model.addAttribute(new MovieDTO());  
 return "updateProductU";  
 }  
 @PostMapping("/product/{id}/updateP")  
 public String updateP(@PathVariable(value="id")int id, @RequestParam(value="script")String script,  
 @RequestParam(value = "rating")int rating, @RequestParam(value = "duration")int duration,  
 @RequestParam(value = "comment")String comment, Model model){  
 markServiceI.update(id,script,rating,duration,comment);  
 return "redirect:/products";  
 }  
}

package com.company.moviemarking.controller;  
  
import com.company.moviemarking.entity.User;  
import com.company.moviemarking.service.impl.UserServiceImpl;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestParam;  
  
import java.util.ArrayList;  
import java.util.List;

Продолжения приложения Б

@Controller  
public class UserController {  
 private final UserServiceImpl userService;  
  
 @Autowired  
 public UserController(UserServiceImpl userService) {  
 this.userService = userService;  
 }  
  
 @GetMapping("/users")  
 public String findAllUsers(Model model) {  
 List<User> users = userService.findAllUsers();  
 users.forEach(System.out::println);  
 model.addAttribute("users", users);  
 return "users";  
 }  
  
 @GetMapping("/user/{id}")  
 public String findUser(@PathVariable(value = "id") int id, Model model) {  
 User user = userService.findUserById(id);  
 List<User> users = new ArrayList<>();  
 users.add(user);  
 model.addAttribute("user", users);  
 return "user-details";  
 }  
  
 @PostMapping("/user/{id}/remove")  
 public String removeUser(@PathVariable(value = "id") int id, Model model) {  
 userService.removeUser(id);  
 return "users";  
 }  
  
 @PostMapping("user/{id}/ban")  
 public String banUser(@PathVariable(value = "id")int id, Model model){  
 userService.banUser(id);  
 return"redirect:/users";  
 }  
 @GetMapping("/user/{id}/update")  
 public String updateUser(@PathVariable(value = "id") int id, Model model) {  
 User user = userService.findUserById(id);  
 List<User> users = new ArrayList<>();//? почему тут список? Так было в

Продолжение приложения Б

других методах

users.add(user);  
 model.addAttribute("user",users);  
 return "updateUser";  
 }  
 @PostMapping("/user/{id}/updateU")  
 public String updateU(@PathVariable(value = "id") int id, @RequestParam(value="firstName")String firstName,  
 @RequestParam(value="lastName")String lastName, @RequestParam(value="phone")String phone,  
 @RequestParam(value="bio")String bio,@RequestParam(value="age")String age, Model model) {  
 userService.update(id,firstName,lastName,phone,bio,age);  
 //List<User> users = new ArrayList<>();  
 //users.add(user);  
 //model.addAttribute("user",users);  
 return "redirect:/users";  
 }  
}

# **ПРИЛОЖЕНИЕ В**

(обязательное)

Листинг скрипта генерации базы данных

-- MySQL Script generated by MySQL Workbench

-- Sun Apr 30 14:01:56 2023

-- Model: New Model Version: 1.0

-- MySQL Workbench Forward Engineering

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

-- -----------------------------------------------------

-- Schema mydb

-- -----------------------------------------------------

-- -----------------------------------------------------

-- Schema kpsitairis

-- -----------------------------------------------------

-- -----------------------------------------------------

-- Schema kpsitairis

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `kpsitairis` DEFAULT CHARACTER SET utf8mb3 ;

USE `kpsitairis` ;

-- -----------------------------------------------------

-- Table `kpsitairis`.`critics`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `kpsitairis`.`critics` (

`id` INT NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(255) NULL DEFAULT NULL,

`surname` VARCHAR(255) NULL DEFAULT NULL,

Продолжение приложения В

`specialty` VARCHAR(255) NULL DEFAULT NULL,

`university` VARCHAR(255) NULL DEFAULT NULL,

`experience` VARCHAR(255) NULL DEFAULT NULL,

PRIMARY KEY (`id`))

ENGINE = InnoDB

AUTO\_INCREMENT = 2

DEFAULT CHARACTER SET = utf8mb3;

-- -----------------------------------------------------

-- Table `kpsitairis`.`hibernate\_sequence`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `kpsitairis`.`hibernate\_sequence` (

`next\_val` BIGINT NULL DEFAULT NULL)

ENGINE = MyISAM

DEFAULT CHARACTER SET = utf8mb3;

-- -----------------------------------------------------

-- Table `kpsitairis`.`movie`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `kpsitairis`.`movie` (

`id` INT NOT NULL AUTO\_INCREMENT,

`title` VARCHAR(255) NULL DEFAULT NULL,

`genre` VARCHAR(255) NULL DEFAULT NULL,

`country` VARCHAR(255) NULL DEFAULT NULL,

`director` VARCHAR(255) NULL DEFAULT NULL,

`description` VARCHAR(255) NULL DEFAULT NULL,

PRIMARY KEY (`id`))

ENGINE = InnoDB

AUTO\_INCREMENT = 2

DEFAULT CHARACTER SET = utf8mb3;

-- -----------------------------------------------------

-- Table `kpsitairis`.`mark`

Продолжение приложения В

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `kpsitairis`.`mark` (

`id` INT NOT NULL AUTO\_INCREMENT,

`comment` VARCHAR(255) NULL DEFAULT NULL,

`script` VARCHAR(255) NULL DEFAULT NULL,

`duration` INT NULL DEFAULT NULL,

`rating` INT NULL DEFAULT NULL,

`critics\_id` INT NULL DEFAULT NULL,

`movie\_id` INT NULL DEFAULT NULL,

PRIMARY KEY (`id`),

INDEX `critic\_mark\_idx` (`critics\_id` ASC) VISIBLE,

INDEX `movie\_mark\_idx` (`movie\_id` ASC) VISIBLE,

CONSTRAINT `critic\_mark`

FOREIGN KEY (`critics\_id`)

REFERENCES `kpsitairis`.`critics` (`id`),

CONSTRAINT `movie\_mark`

FOREIGN KEY (`movie\_id`)

REFERENCES `kpsitairis`.`movie` (`id`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb3;

-- -----------------------------------------------------

-- Table `kpsitairis`.`role`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `kpsitairis`.`role` (

`id\_role` INT NOT NULL AUTO\_INCREMENT,

`role` VARCHAR(255) NULL DEFAULT NULL,

PRIMARY KEY (`id\_role`))

ENGINE = InnoDB

AUTO\_INCREMENT = 3

DEFAULT CHARACTER SET = utf8mb3;

-- -----------------------------------------------------

-- Table `kpsitairis`.`users`

Продолжение приложения В

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `kpsitairis`.`users` (

`id\_users` INT NOT NULL AUTO\_INCREMENT,

`active` BIT(1) NULL DEFAULT NULL,

`bio` VARCHAR(255) NULL DEFAULT NULL,

`first\_name` VARCHAR(255) NULL DEFAULT NULL,

`last\_name` VARCHAR(255) NULL DEFAULT NULL,

`password` VARCHAR(255) NULL DEFAULT NULL,

`phone` VARCHAR(255) NULL DEFAULT NULL,

`username` VARCHAR(45) NULL DEFAULT NULL,

`age` VARCHAR(255) NULL DEFAULT NULL,

`zip` VARCHAR(255) NULL DEFAULT NULL,

PRIMARY KEY (`id\_users`))

ENGINE = InnoDB

AUTO\_INCREMENT = 2

DEFAULT CHARACTER SET = utf8mb3;

-- -----------------------------------------------------

-- Table `kpsitairis`.`user\_role`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `kpsitairis`.`user\_role` (

`id\_user` INT NOT NULL,

`id\_role` INT NOT NULL,

PRIMARY KEY (`id\_user`, `id\_role`),

INDEX `id\_user` (`id\_user` ASC) INVISIBLE,

INDEX `id\_role` (`id\_role` ASC) VISIBLE,

CONSTRAINT `user\_role\_role`

FOREIGN KEY (`id\_role`)

REFERENCES `kpsitairis`.`role` (`id\_role`),

CONSTRAINT `user\_role\_user`

FOREIGN KEY (`id\_user`)

REFERENCES `kpsitairis`.`users` (`id\_users`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb3;

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;