**Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

# К КУРСОВОЙ РАБОТЕ

***НА ТЕМУ:***

***Система бронирования билетов в кинотеатрах.***

Студент ИУ7-21М

**Осипов О.К.**

(Подпись, дата)

Руководитель курсовой работы **Романов А.С.**

(Подпись, дата)

Консультант

(Подпись, дата)

*2020 г.*

**Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

**З А Д А Н И Е**

УТВЕРЖДАЮ

Заведующий кафедрой ИУ-7

Рудаков И.В.

« » 20 г.

**на выполнение курсовой работы**

по дисциплине *распределённые системы обработки информации*

Студент группы *ИУ7-21М Осипов Олег Константинович*

Тема курсовой работы *Система бронирования билетов в кинотеатрах.*

Направленность КР *учебная*

Источник тематики *ИУ7*

График выполнения работы: 25% к 3 нед., 50% к 9 нед., 75% к 12 нед., 100% к 15 нед.

**Задание** провести анализ предметной области: изучить существующие сервисы бронирования билетов на фильмы в кинотеатрах. Разработать распределённую систему, предназначенную для бронирования билетов на определённый фильм в определённом кинотеатре. Система должна включать в себя следующие сервисы: gateway, сервис авторизации, сервис-проксировщик, предназначенный для раздачи статических файлов, сервис, предоставляющий внешний API, сервис сбора, хранения и обработки статистики, а также сервисы, реализующие бизнес логику – сервис данных о фильмах, сервис кинотеатров , и сервис сеансов, и сервис бронирования билетов, т.е. сервис заказов. Реализовать ролевую модель пользователей, выделить роли Администратора, Пользователя и Администратора Кинотеатра, именуемого далее Кинодиректором. Дополнительно выделить сервис учёта, хранящий информацию о действиях пользователя.

Подготовить в качестве примера описание процедуры бронирования билетов. Подготовить модульные, интеграционные и функциональные тесты. Оформить расчётно-пояснительную записку, содержащую результаты проделанной работы.

***Оформление курсовой работы:***

Расчетно-пояснительная записка на 81 листах формата А4. Дата выдачи задания « » 20 г.

**Руководитель курсовой работы** Романов А.С.

(Подпись, дата)

**Студент** Осипов О. К.

(Подпись, дата)

**Содержание.**

**1. Аналитический раздел…………………………….…………………………5**

**1.1 Анализ существующих аналогов……….………………………….5**

**1.2 Описание предметной области………..……………………….......5**

**1.3 Бизнес-логика системы…………………………………….………..6**

**1.4 Требования к системе……………………………………….……….8**

**1.4.1 Требования к структуре и функционированию системы…………………………………………………………………………...8**

**1.4.2 Требования к системе с точки зрения пользователей……………………………………………………………………9**

**1.4.3 Входные и выходные данные системы………...………..10**

**1.4.4 Требования к безопасности………………………………11**

**1.4.5 Функциональные характеристики………………………11**

**1.5 Топология разрабатываемой системы……………………………11**

**1.5.1 Описание топологии системы…………………………….11**

**1.5.2 Требования по реализации топологии…………………..14**

**1.6 Функциональные требования к подсистемам…………………..15**

**2. Конструкторский раздел……………………………………………………18**

**2.1 Описание сущностей бизнес-логики системы..………………….18**

**2.2 Варианты использования системы и возможные прецеденты.**

**……………………………………………………………………………..……...28**

**2.3 Взаимодействие между сервисами, системой и пользователями…………………………………………………………………46**

**3. Технологический раздел.**

**3.1 Структура проекта………………………………………………...50**

**3.2. Описание схемы Model-View-Controller проекта……………..52**

**3.2.1 API сервиса кинотеатров………………………………...55**

**3.2.2 API для сервиса фильмов………………………………...59**

**3.2.3 API для сервиса сеансов…………………………………..62**

**3.2.4 API сервиса заказов……………………………………….64**

**3.2.5 API сеансового сервиса SessionService…………………...66**

**3.2.6 API сервиса статистики……………………………………68**

**3.3 Основная процедура авторизации в системе…………………..70**

**3.4 Описание фронтэнд приложения………………………………….72**

**3.5 Запуск и развёртывание приложения…………………………….79**

**Список использованных источников………………………………………81**

**1 Аналитический раздел.**

**1.1 Существующие аналоги.**

Среди аналогов можно отметить порталы *afisha.yandex.ru*, *m.kassa.rambler.ru*, а также *kinohod.ru* у которых есть сервисы по выбору покупке билетов в кинотеатрах на определённый сеанс. Интерфейс afisha.yandex.ru позволяет отобрать билеты по фильму, жанру и кинотеатру, однако, он не позволяет напрямую отобрать билеты по цене. Интерфейс m.kassa.rambler.ru позволяет посмотреть и отобрать доступные билеты по дате, времени, названию кинотеатра или фильма. Однако он требует указать наименование фильма или кинотеатра для доступа к списку текущих сеансов. Отметим, что данный проект должен обладать следующими преимуществами перед аналогами:

* Поиск сеанса по названию фильма.
* Возможность выбора билета по цене.
* Возможность резервирования билета на определённое время.
* Высокая скорость загрузки страниц портала.

**1.2 Описание предметной области.**

В Москве, и Московской области есть множество кинотеатров.

из себя портал, в котором регистрируются пользователи и директора кинотеатра. Зарегистрированный пользователь может забронировать место в кинотеатре на указанный сеанс. Любой незарегистрированный может увидеть список сеансов, проводимых в кинотеатрах. Также он может отсортировать их по названию фильма, кинотеатра, местоположения, жанра фильма, цене билета.

Директор заносит информацию о сеансах в его кинотеатре.

Регистрируясь как директор, он обязывается предоставить следующую информацию о кинотеатре: его название, местоположение, название компании. Она создаёт запись о новом кинотеатре.

Соответственно директор заносит информацию о сеансах, проводимых в кинотеатре. Кроме того, как и клиент так и директор обязаны предоставить для регистрации адрес электронной почты и пароль. Также директор добавляет информацию о фильмах (название, жанр, рейтинг).

Администратор следит за работой сайта Он также может собрать статистику о работе приложения, а также заблокировать пользователей.

Таким образом, в системе должна быть реализована ролевая модель, содержащая четыре основные роли: *пользователь*, *директор кинотеатра*, *администратор системы, неавторизованный пользователь*, он же *гость*.

**1.3 Бизнес-логика системы.**

Портал должен обеспечивать реализацию следующих функций:

1). Система должна обеспечивать регистрацию пользователей

валидацией вводимых данных.

2). Система должна обеспечивать аутентификацию пользователей.

3). Система должна обеспечивать разделение пользователей на три роли:

* Администратор кинотеатра
* Клиент
* Администратор системы.

4) Любой незарегистрированный пользователь может

увидеть информацию о текущих сеансах проводимых в кинотеатрах. Желающий может отфильтровать список по дате, кинотеатру, месту, фильму, жанру фильма и так далее. Для бронирования требуется авторизоваться как клиент.

5) Для клиента, приложение должно обеспечить следующие функции:

* Возможность бронирования билетов в выбранном кинотеатре на

определённый фильм в указанное время, просмотр выбранных билетов (список заказов), просмотр информации о выбранном билете (заказе).

6) Для директора, приложение должно обеспечить следующие функции:

* Возможность добавления, изменения, удаления информации о кинотеатре.
* Возможность внесения информации о сеансах (время, название фильма, номер зала.
* Возможность внесения информации о ценах, акциях на билеты.
* Возможность редактирования комнат в кинотеатре.

7) Для администратора система должна провести:

* Возможность блокировки пользователей исходя из данных статистики.
* Возможность сбора статистики от сервиса Статистики.

8) Пользователю не требуется знать идентификаторы сущностей (фильмов, кинотеатров, жанров) хранимых в базах данных.

9) Всё, что может делать клиент, может делать и администратор кинотеатра (т.е. он также имеет возможность выбрать и оформить бронирование билетов).

10) Сеансы привязаны к комнате определённого кинотеатра.

11) Каждый заказ (бронь) имеет описание, и может состоять из нескольких выбранных билетов.

12) Система проверяет вводимые данные пользователем. Проверка осуществляется на стороне клиента (с помощью скриптов), а также на стороне сервера.

**1.4 Требования к системе.**

**1.4.1 Требования к структуре и функционированию системы.**

1. Разрабатываемое программное обеспечение должно обеспечивать функционирование системы в режиме 24/7/365 со среднегодовым временем доступности не менее 99.9%. Допустимое время, в течение которого система не доступна, за год должна составлять 24\*365\*0.001=8.76 часа.

2. Время восстановления системы после сбоя не должно превышать 15 минут.

3. Архитектура приложения – *микросервисная*. Система должна состоять из 3-4 независимых бизнес-сервисов, каждый отвечающий за свою функциональность, и взаимодействующих друг с другом посредством HTTP вызовов или через очередь.

4. Каждый сервис имеет свое собственное хранилище, если оно ему нужно. Предпочтительно использовать реляционные БД и NoSQL.

5. В случае ошибки/недоступности некритичного функционала выполнять деградацию функциональности.

6. Между сервисами допускаются горизонтальные связи. Выделенного gateway *для работы через UI* не нужно.

7. Для защиты информации от несанкционированного доступа необходимо реализовать механизм аутентификации и авторизации. Для авторизации выделить отдельный сервис, куда будут отправляться все запросы на проверку авторизации как от пользователя, так и от взаимодействующих внутренних систем.

8. Пользовательский интерфейс на базе SPA (react, angular, vue, etc.), использование CSS *обязательно*. Статику для SPA хранить на отдельном сервере nginx. Через UI также должна быть регистрация и авторизация пользователей. Все запросы на авторизацию отправляются к сервису Авторизации.

9. Все действия совершаемые на каждом сервисе сохраняются в логах. Для сбора статистики необходимо выделить отдельный сервис. Он получает информацию обо всех операциях пользователя. Данные в этот сервис передаются через очередь.

**1.4.2 Требования к системе с точки зрения пользователей.**

Портал должен обеспечивать реализацию следующих функций:

1). Система должна обеспечивать регистрацию пользователей

валидацией вводимых данных.

2). Система должна обеспечивать аутентификацию пользователей.

3). Система должна обеспечивать разделение пользователей на три роли:

* Администратор кинотеатра
* Клиент
* Администратор системы.

4) Любой незарегистрированный пользователь может

увидеть информацию о текущих сеансах проводимых в кинотеатрах. Желающий может отфильтровать список по дате, кинотеатру, месту, фильму, жанру фильма и так далее. Для бронирования требуется авторизоваться как клиент.

5) Для клиента, приложение должно обеспечить следующие функции:

* Возможность бронирования билетов в выбранном кинотеатре на

определённый фильм в указанное время

6) Для директора, приложение должно обеспечить следующие функции:

* Возможность добавления, изменения, удаления информации о кинотеатре.
* Возможность внесения информации о сеансах (время, название фильма, номер зала.
* Возможность внесения информации о ценах, акциях на билеты.

7) Для администратора система должна провести:

* Возможность блокировки пользователей исходя из данных статистики.
* Возможность сбора статистики от сервиса Статистики.

8) Пользователю не требуется знать идентификаторы сущностей (фильмов, кинотеатров, жанров) хранимых в базах данных.

**1.4.3 Входные и выходные данные системы.**

**Регистрация клиента.**

*Вход*: ФИО пользователя в виде отдельных строк переменной длины с максимально возможным количеством символов равным 80, возраст в виде целого числа в диапазоне от 1 до 150, адрес электронной почты в виде строки переменной длины с максимально возможным количеством символов равным 60, пароль в виде уникальной строки с максимально возможным количеством символов равным 120.

*Вывод*: При успешной регистрации вернуть веб-страницу. В заголовке страницы отобразить ФИО пользователя и его адрес электронной почты. Иначе, при неправильных входных данных отобразить сообщение об ошибке и прикрепить его внизу соответствующего поля.

**Регистрация директора.**

*Вход*: ФИО, email, пароль, как они были определены ранее, а также название кинотеатра, его адрес в виде отдельных полей (город, улица)

в виде переменной последовательности символов не более 120.

*Выход*: Либо веб-страница, либо сообщение об ошибке.

**Авторизация (для клиента и директора и администратора).**

*Вход*: адрес электронной почты в виде строки переменной длины с

максимально возможным количеством символов равным 60, пароль в виде уникальной строки с максимально возможным количеством символов равным 120.

*Выход*: Веб-страница пользователя, или сообщение об ошибке (о неверном пароле или адресе электронной почты).

**Извлечение фильмов по указанному фильтру.**

*Вход:* Название жанра, фильма, кинотеатра, в виде строки переменной длины.

*Выход*: Список фильмов указанного жанра, проводимых в указанном кинотеатре (если это указано, иначе для любого кинотеатра).

Аналогично, возвращается список кинотеатров. Для сеансов можно указать дополнительно время.

**1.4.4 Требования к безопасности.**

Все учётные данные пользователя, используемые для аутентификации и авторизации хранить в базе данных в зашифрованном виде. Для сторонних приложений (3-rd party applications) использовать авторизацию по токену через протокол OAuth2.

**1.4.5 Функциональные характеристики.**

Время отклика при запросах на авторизацию, аутентификацию и регистрацию не превышает 3 секунды.

Время отклика при запросах на получение списков кинотеатров, фильмов и сеансов не превышает 5 секунд.

**1.5 Топология разрабатываемой системы.**

**1.5.1 Описание топологии системы.**

Одна из возможных топологии будущей системы показана на рис. 1.



Рис. 1. Топология системы.

Система будет состоять из одного веб-сервера с приложением фронтенда и 6 серверов бекендов.

* **Сервер авторизации –** на нём расположен сервис авторизации, который реализует следующие функции:
  + Регистрация пользователей
  + Аутентификация (проверка сессии) пользователя
  + Авторизация пользователя (вход по логину/паролю, или по токену).
  + Выход из сессии («логаут»)
* **Файловый сервер –** отвечает за хранение статистических ресурсов, такие, как фотографии пользователей, кинозалов, файлы таблицы стилей и пр.
* **Сервер кинотеатров –** на нём расположен сервис кинотеатров, выполняющий следующие задачи:
  + Получение информации о указанном кинотеатре.
  + Получение списка кинотеатров по определённому фильтру.
  + Добавление, изменение, удаление информации о кинотеатре для их администраторов.
  + Добавление, изменение, удаление информации о комнатах (залах) в указанном кинотеатре (администратором кинотеатра).
* **Сервер фильмов** – на нём расположен сервис фильмов, реализующий:
  + Получение списка фильмов
  + Получение фильма по фильтру. (названию, жанру, дате, названию кинотеатра).
  + Добавление, удаление информации о фильмах для администраторов кинотеатров.
* **Сервер киносеансов –** на нём расположен сервис киносеансов, отвечающий за следующие функции:
  + Получение списка киносеансов по указанным фильтрам (дате, кинотеатру, названию фильма, жанру фильма, цене билетов, по номеру зала в указанном кинотеатре).
  + Создание, изменение, удаление билетов на указанный сеанс и указанное место. При указании сеанса, необходимо также указать идентификатор комнаты. Поскольку комната является частью кинотеатра, и она содержится ровно в одном кинотеатре, то по ней можно определить кинотеатр.
  + Получение списка билетов по указанной цене, форме оплаты, идентификатору места, номеру места, номеру комнаты.
* **Сервер заказов –** на нём расположен сервис заказов, реализующий:
  + Бронирование (или покупка) выбранного сеанса клиентом.
  + Отмена заказа (брони) если пользователь не подтвердит покупкой билета в течении 30 минут (т.е. если пользователь выбрал сеанс, то в течении 30 минут ему необходимо совершить покупку, иначе заказ отменяется).
  + Получение списка выбранных (забронированных) киносеансов для авторизованного клиента.
* **Сервер статистики –** отвечает за хранение статистики. На нём расположены два сервиса – сервис статистики и сервис учёта. Сервис статистики считает и визуализирует информацию о затраченном времени для загрузки страницы. Сервис учёта хранит историю действий для любого клиента и администратора кинотеатра. Клиенты и администраторы кинотеатра не имеют доступа к сервису статистики. Что касается сервиса учета, то оба имеют доступ лишь к одной операции:
  + Получение и сортировка списка проведённых операций для авторизованного пользователя (клиента или администратора кинотеатра).

Полный доступ к вышеописанным сервисам имеет только администратор системы.

Фронтенд принимает запросы от пользователей по протоколу HTTP, анализирует их, выполняет запросы к бекенду, агрегирует ответ от них и отдаёт результат обратно пользователю в виде веб-страницы. Для запросов, требующих авторизацию, фронтенд возвращает веб-страницу авторизации с формой для входа в систему. Получив информацию с формы, фронтенд отправляет запрос на сервер авторизации, и, в случае успеха, возвращает веб-страницу пользователя, иначе возвращает веб-страницу авторизации с сообщением об ошибке.

**1.5.2 Требования по реализации топологии.**

1. Все серверы должны быть запущены изолированно друг от друга.
2. Используется сервис-ориентированная архитектура (СОА).
3. Для фронтенда необходимо реализовать один веб-интерфейс, доступный через тонкий клиент – веб-браузер.
4. Валидация входных данных должна быть как и на стороне клиента (с помощью JavaScript), так и на стороне сервисов (поскольку сторонние приложения, авторизованные по токену через OAuth2, имеют доступ к прикладному программному интерфейсу, позволяющий напрямую обращаться к веб-сервисам.)
5. Базы данных указанные в схеме топологии принадлежат к указанным серверам (указано соединительной линией).

**1.6 Функциональные требования к подсистемам.**

На рис. 2. представлены подсистемы будущего приложения.



Рис. 2. Подсистемы приложения.

1. **Фронтенд –** это серверное приложение. Оно принимает запросы от пользователей системы по протоколу HTTP и формирует ответы для них в формате HTML.
2. Запросы к бекендам выполняет фронтенд в зависимости от типа запроса пользователя. Если запрос требует авторизацию пользователя, то фронтенд возвращает HTML страницу с формой, для осуществления последующего запроса к сервису авторизации.
3. Запросы между сервисами выполнять по протоколу HTTP. Данные необходимо передавать в формате JSON [1][2]. Данный текстовый формат удобен для чтения.
4. Целесообразно использовать библиотеку Bootstrap третей версии для создания HTML-вёрстки и упрощения разработки веб-интерфейса.
5. Учётные данные хранятся в шифрованном в виде в базе данных сервиса авторизации.
6. Сервис авторизации имеет в базе данных пять сущностей – *Пользователя (UserEntity)*, его *Роль (Role)*, *токен OAuth* (*OAuthToken*), *код авторизации приложения (AuthorizationCode),* и клиенты (приложения), проходящие авторизацию по протоколу OAuth 2.0 (OAuthClient).
7. Сервис кинотеатров имеет в базе следующие сущности: *Кинотеатр (Cinema)*, *Кинозал (Room), Места (Seats).*
8. Сервис фильмов имеет в базе следующие сущности: *Фильм (Film)*, *Жанр (Genre)*.
9. Сервис бронирования киносеансов имеет в базе следующие сущности: *Сеанс (Séance), Билет (Ticket), Комнаты Кинотеатра (RoomsCinema).* Доступ к СУБД осуществляется по протоколу TCP.
10. Для каждой из баз данных необходимо создавать её резервную копию спустя определённое время (раз в неделю). Для этого предусмотреть создания соответствующего скрипта.
11. Все сервисы на рис. 2. являются самостоятельными серверными приложениями.
12. **Файловый бекенд –** тоже является серверным приложением которое должно:
    * Принимать и отвечать на запросы в формате JSON по протоколу HTTP.
    * приложение должно уметь обрабатывать запросы на загрузку файлов в бекенд и получение файлов из бекенда.

**2 Конструкторский раздел.**

**2.1 Описание сущностей бизнес-логики системы.**

Прежде чем перейти к описанию сущностей системы, необходимо сделать несколько обобщений, касающихся их структуры. Во-первых, везде, где не указано, что атрибут сущности может принимать значения *null,* считается, что атрибуту запрещено иметь такое значение. Во-вторых, типы атрибутов представлены типами данных языка SQL. Названия таких типов написаны в верхнем регистре. В-третьих, для атрибутов, идентифицирующих сущность и составляющих её первичный ключ используется тип длинных целых чисел *BIGINT*. Также, если для атрибута не указан тип, то считается, что атрибут имеет тип *BIGINT*. Сущности AuthorizationCode, OAuthToken, OAuthClient имеют идентификатор другого типа.

Использовать в качестве идентификатора универсальный уникальный идентификатор (Universal Unique Identificator – UUID) для разрешения проблемы управления идентичностью сущности и ускорения работы ORM-фреймворка, требует в 4 раза больше памяти, чем обычный числовой 8-байтовый идентификатор. Кроме-того, данная проблема усиливается, при наличии внешнего ключа и создании индексов на соответствующие столбцы данного типа. Поэтому для идентификатора сущности был выбран тип BIGINT.

Начнём с описания сущностей для фильма и жанра. Сущность *Жанр* (*Genre*) имеет следующие атрибуты:

* *идентификатор жанра (gid)*
* *имя жанра (gname),* имеющий тип VARCHAR(100)
* *примечания (remarks),* имеющий тип TEXT

Первичным ключом данной сущности является *идентификатор жанра. Имя жанра* представлено строкой переменной длины и состоящей из не более чем 100 символов и не менее 3 символов. Для строк переменной длины в базе данных используется тип *VARCHAR(n),* где *n –* максимально возможная длина строки. Атрибут *примечания (remarks)* представлен типом *TEXT.* Данный тип используется для хранения произвольной последовательности символов. Атрибут может принимать значения null.

Сущность *фильм* (*film*) имеет следующие атрибуты:

* *идентификатор фильма (fid)*
* *имя фильма (fname)*, имеющий тип VARCHAR(100)
* *возрастной рейтинг (rating),* имеющий тип SMALLINT

*Имя фильма* представлено строкой переменной длины имеющей не менее 2 символов и не более 100 символов (тип *VARCHAR(100)*). *Возрастной* *рейтинг* фильма выражен типом *SMALLINT.* Первичным ключом сущности является атрибут – *идентификатор фильма (fid)*.Фильм может иметь несколько жанров. Поэтому для связи фильма и жанра используется промежуточная сущность: *фильм-жанры* (*film-genres)*. Эта сущность состоит из двух атрибутов: *идентификатор фильма* (fid) и *идентификатор жанра (gid).* Оба являются внешним ключом данной сущности, ссылающимися на первичные ключи сущностей фильма и жанра. Первичный ключ данной сущности состоит ровно из двух вышеуказанных атрибутов.

Сущность *кинотеатр (cinema)* состоит из следующих атрибутов:

* *идентификатор кинотеатра (cid)*
* *имени кинотеатра (cname),* представленного строкой переменной длины, состоящей из не менее чем 5 символов и не более чем 127 символов (тип VARCHAR(127))
* *Адреса кинотеатра (address)* представленный отдельными атрибутами:
  + *улица (street)* имеющий тип VARCHAR(50).
  + *город (city)* имеющий тип VARCHAR(50).
  + *регион (region)* имеющий тип VARCHAR(50) и допускающий значения *null.*
  + *страна (country)* имеющий тип VARCHAR(100).
  + *номер здания (building)* имеющий тип *SMALLINT.*

Первичным ключом сущности является атрибут – *идентификатор кинотеатра (cid).*

Cущность *комната (room)* состоит из следующих атрибутов:

* *идентификатор комнаты (rid)*
* *идентификатор кинотеатра (cid)*
* *категория комнаты (category),* имеющий тип VARCHAR(50),
* *количество мест в комнате (seats)* имеющий тип INT,значение атрибута больше или равно 0.
* *Номер комнаты (room\_number)* имеющий тип INТ, значение атрибута больше или равно 0.

Первичным ключом данной сущности является атрибут – *идентификатор комнаты.* Между сущностями *комната* и *кинотеатр* существует отношение *многие-ко-одному –* множество комнат может быть частью одного кинотеатра. Аналогично, между кинотеатром и комнатой существует отношение *один-ко-многим* – кинотеатр имеет множество комнат. Для связи между комнатой и кинотеатром у комнаты определён внешний ключ, ссылающийся на идентификатор кинотеатра. Таким образом, *идентификатор кинотеатра (cid)* является внешним ключом у сущности комнаты.

Сущность *место (seat)* имеет следующие атрибуты:

* *идентификатор места (seat\_id)*
* *номер места (num),* имеющий тип BIGINT, значение атрибута больше или равно 0.
* *идентификатор комнаты (room\_id)*
* *cостояние места (state),* имеющий тип VARCHAR(5), значение атрибута представлено следующим перечислением строк:
  + NA – место в комнате, в которой нет сеанса.
  + A – место в комнате, в которой есть сеанс, но на данное место нет билета.
  + VA – свободное место в комнате, на которое есть билет.
  + B – забронированное место, выбранное клиентом.

Первичным ключом сущности является атрибут – *идентификатор места (seat\_id).* Между *комнатой* и *местом* существует такое же отношение *один-ко-многим*, какое между кинотеатром и комнатой. Т.е. место является частью комнаты, и комната содержит множество мест. Внешним ключ сущности места, выраженный атрибутом *идентификатор комнаты,* ссылается на одноимённый атрибут комнаты.

Cущность *сеанс* имеет следующие атрибуты:

* *Идентификатор сеанса (sid)*
* *Идентификатор комнаты (room\_id)*
* *Идентификатор фильма (fid),* имеющий тип BIGINT.
* *Дата начало фильма,* имеющей тип DATETIME.
* *Время начало фильма,* имеющий тип TIME(2). Время представлено строкой, состоящий из трёх групп цифр, разделённых двоеточием. Эти три группы представляют час, минуту, секунду. Секунды имеют дробную часть, состоящую из двух цифр после запятой.

Первичным ключом сущности являются два атрибута: *идентификатор сеанса* и *идентификатор комнаты*. Так как комната является частью кинотеатра, то для связи комнаты с кинотеатром создана дополнительная сущность *комнаты-кинотеатр (rooms\_cinema).* Эта сущность состоит из двух атрибутов: идентификатора комнаты, который и является первичным ключом сущности, и идентификатора кинотеатра. Для сущности *сеанс* определён внешний ключ, состоящий из одного атрибута *идентификатора комнаты (room\_id),* ссылающегося на одноимённый атрибут сущности *комнаты-кинотеатр (rooms\_cinema).* Идентификатор комнаты определяет кинотеатр, поскольку комната является частью кинотеатра. Каждый сеанс имеет уникальное поле *идентификатор сеанса (sid)* по следующей причине: в одной и той же комнате одного и того же кинотеатра, может показываться один и тот же фильм, но в разное время.

Сущность *билет (ticket)* имеет следующие атрибуты:

* *Идентификатор сеанса (sid)*
* *Идентификатор комнаты (room\_id)*
* *Идентификатор места (seat\_id)*
* *Цена билета (price),* имеющий тип DECIMAL(20,10), т.е. действительное число с 20 цифрами 10 из которых принадлежат дробной части числа.
* *Тип оплаты (payment\_type),* имеющий тип VARCHAR(20). По умолчанию, данный атрибут принимает значение в виде строки “Cash”.

Первичным ключом данной сущности являются два атрибута: идентификатор сеанса и идентификатор места. Внешним ключом являются два атрибута: идентификатор сеанса, и идентификатор комнаты. Внешний ключ представляет из себя ссылку на первичный ключ к сущности сеанса.

Сущность *пользователи (users)* имеет следующие атрибуты:

* *Идентификатор пользователя (id)*
* *Отображаемое имя (name),* имеющий тип VARCHAR(60)
* *Имя пользователя (username),* имеющий тип VARCHAR(20), и уникальные значения. (так как именно это имя и используется для идентификации страницы пользователя).
* *Электронная почта пользователя (email),* имеющий тип VARCHAR(60) и уникальные значения. Она используется в качестве логина пользователя и имеет следующий формат: *имя\_почты@доменное имя почтового сервиса*. Данный формат описан в стандарте RFC5322.
* *Пароль пользователя (password),* имеющий тип VARCHAR(100). Пароль и электронная почта хранятся в базе данных в хешированном виде.
* *Дата создания (createdAt),* имеющий тип DATE.
* *Дата обновления (updatedAt),* имеющий тип DATE.

Первичным ключом сущности является *идентификатор пользователя.*

Сущность *ролей (roles)* имеет следующие атрибуты:

* *Идентификатор роли (rid)*
* *Имя роли (rname),* имеющий тип VARCHAR(60), и значения, представленные следующим перечислением строк:
  + *ROLE\_USER – роль клиента*
  + *ROLE\_CINEMA – роль администратора кинотеатра.*
  + *ROLE\_ADMIN – роль администратора системы.*

Неавторизованный пользователь не имеет роли. Пользователь может иметь несколько ролей. Т.е. между ролями и пользователями определено отношение *многие-ко-многим.* Для реализации такой связи была определена ещё одна сущность – *пользователь-роли (user\_roles).* Эта сущность состоит из двух атрибутов – *идентификатора пользователя (user\_id)* и *идентификатора роли (role\_id).* Первичный ключ полностью состоит из двух вышеуказанных атрибутов. Каждый атрибут является также внешним ключом, ссылающимся на одноимённый атрибут в сущности *пользователей* (*users*).

Сторонние приложение проходят процедуру авторизации по протоколу OpenAuth версии 2.0. В рамках данной процедуры определены три сущности: *код авторизации* (*authorization\_code*), *токен OAuth* (*oauth\_token*), и клиент OAuth (*oauth\_client*), представляющий стороннее приложение.

Сущность кода авторизации (authorization\_code) имеет следующие атрибуты:

* *Код авторизации (code),* имеющий тип VARCHAR(36).
* *Идентификатор пользователя (user\_id)*
* *Идентификатор клиента (client\_id),* имеющий тип VARCHAR(36).
* *Адрес редиректа (redirect\_uri),* имеющий тип VARCHAR(255)
* *Дата создания (createdAt),* имеющий тип DATE.
* *Дата обновления (updatedAt),* имеющий тип DATE.
* *Флаг валидности (validity),* имеющий тип TINYINT(1).

Первичным ключом сущности является сам код авторизации.

Сущность *токена OAuth (oauth\_token)* имеет следующие атрибуты:

* *Токен доступа (access\_token),* имеющий тип VARCHAR(1000)
* *Идентификатор клиента (client\_id),* имеющий тип VARCHAR(36)
* *Идентификатор пользователя (user\_id).*
* *Токен для обновления (refresh\_token),* имеющий тип VARCHAR(1000).
* *Флаг валидности токена доступа (access\_token\_validity),* имеющий тип TINYINT(1)
* *Флаг валидности токена обновления (refresh\_token\_validity),* имеющий тип TINYINT(1).

Первичным ключом сущности является атрибут – *токен доступа (access\_token). Токен обновления* используется, для получения нового *токена доступа*, в случае, если у токена доступа истёк срок годности использования.

Сущность *клиента OAuth (oauth\_client)* имеет следующие атрибуты:

* *Идентификатор клиента (client\_id),* имеющий тип VARCHAR(36).
* *Секрет клиента (client\_secret),* имеющий тип VARCHAR(36)
* *Адрес редиректа (redirect\_uri),* имеющий тип VARCHAR(255)
* *Дата создания (createdAt),* имеющий тип DATE.
* *Дата обновления (updatedAt),* имеющий тип DATE.
* *Имя приложения (application\_name),* имеющий тип VARCHAR(255)
* *Контекст приложения (scope),* имеющий тип VARCHAR(255)
* *Флаг предоставления пароля (grant\_password),* имеющий тип TINYINT(1).

Первичным ключом сущности является *идентификатор клиента.*

Сущность *заказа клиентов “customer\_orders” (Order)* имеет следующие атрибуты:

* *Идентификатор заказа (oid)*
* *Идентификатор клиента (пользователя) (uid)*
* *Суммарная стоимость заказа (totalSum),* имеющий тип DECIMAL(20,10)
* *Дата создания заказа (createdAt),* имеющий тип DATE
* *Время создания заказа (creationTime),* имеющий тип TIME(2)
* *Дата обновления заказа (updatedAt),* имеющий тип DATE
* *Статус заказа (ostatus),* имеющий тип VARCHAR(10) и множество значений, представленное следующим перечислением строк:
  + *SELECTED –* пользователь выбрал места, оформил заказ, но не оплатил его. На оплату даётся время с момента оформления заказа, равное 30 минутам.
  + *CONFIRMED –* пользователь оплатил данный заказ.

Первичным ключом сущности является *идентификатор заказа*.

Сущность *детали заказа “order\_details” (OrderItem)* имеет следующие атрибуты:

* *Идентификатор сущности (id)*
* *Идентификатор заказа (oid)*
* *Скидка (discount)* по билету, имеющий тип DECIMAL (20,10)
* *Цена (price)* по билету, имеющий тип DECIMAL (20,10)
* *Идентификатор сеанса (sid)*
* *Идентификатор места (seat\_id)*

Первичным ключом сущности является её идентификатор (id). Внешний ключ сущности, составленный из атрибута *идентификатора заказа (oid)*, представляет собой ссылку на одноимённый атрибут в сущности заказов.

Ниже дана диаграмма классов, которая обобщает всё вышеизложенное (Рис. 3). Дополнительно на диаграмме указаны, какие сущности относятся к определённым сервисам. Для любого поля класса, если не указаны методы, определены стандартные *get/set*-методы.

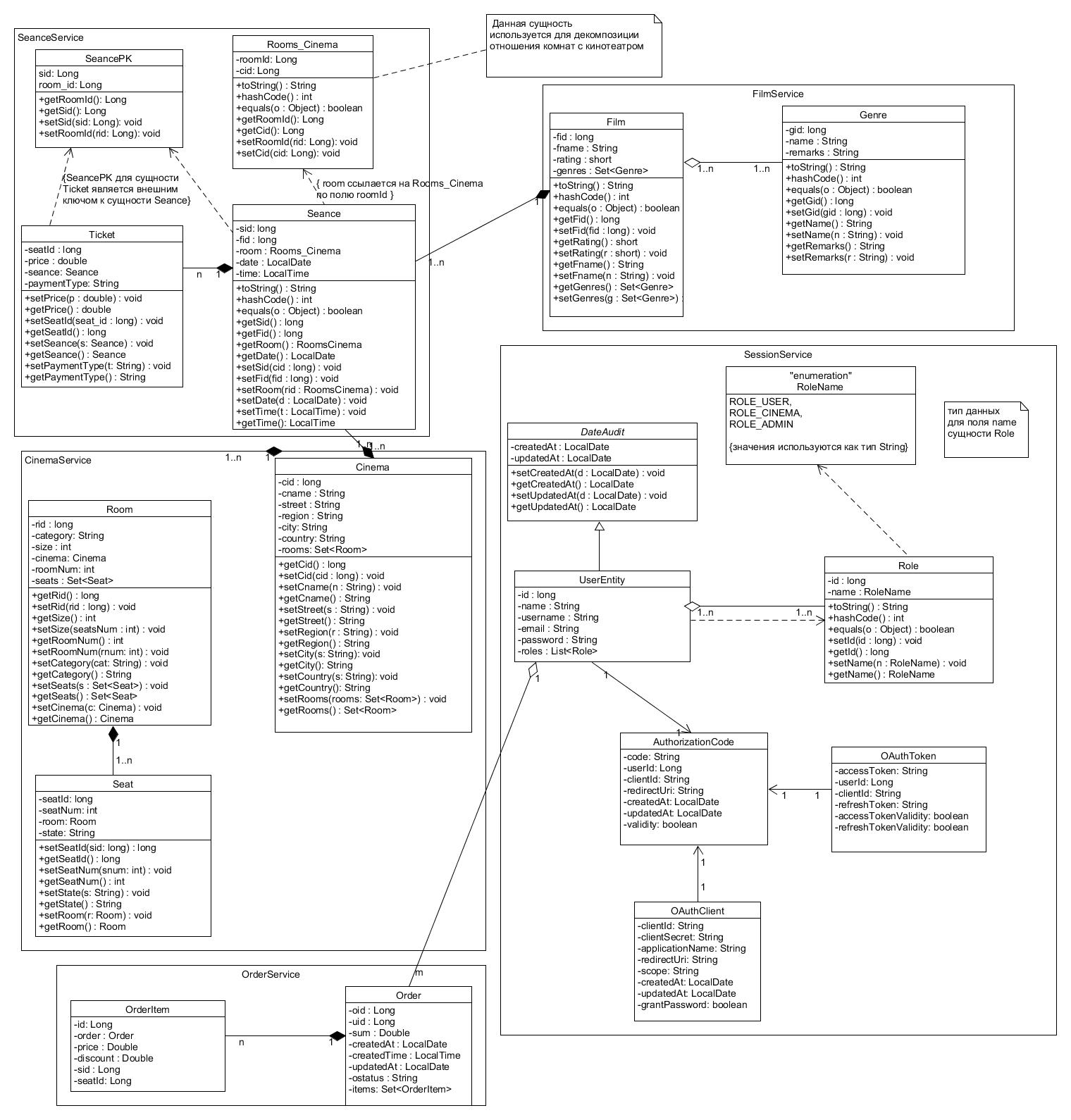


Рис. 3. Классы бизнес-сущностей.

Все данные типы лежат в пакете *entities*. Они все переопределяют методы *equals(Object o)* и *hashCode()*. Переопределение методов связано с тем, что фреймворк объектного реляционного отображения *Hibernate* умеет различать объекты в рамках одной сессии. В случае двух различных сессий, при извлечении одной и той же записи из базы данных, будут созданы два объекта с одинаковыми значениями полей, но различными возвращаемыми значениями методов *equals* и *hashCode*. Т.е. с точки зрения Hibernate они будут отличаться и иметь разные хэш-коды. Чтобы решить данную проблему необходимо переопределить данные методы [3]. Самым простым способом является сравнение по полю первичного ключа, так как он является уникальным и неизменяемым. Однако *Hibernate* создаёт первичный ключ, только в процессе размещения объекта в базе данных, а не при вызове конструктора класса данной бизнес-сущности. Т.е. перед добавлением в коллекцию объектов типа *List, Set, HashSet* и т.д. придётся выполнить сохранение в базе данных [4]. Данный изъян возникает только для вновь создаваемых объектов.

**2.2 Варианты использования системы и возможные прецеденты.**

Все возможные прецеденты в данной системе указаны в нижележащей диаграмме.



Рис. 4. Диаграмма прецедентов.

Так как в рамках ТЗ были определены три возможные роли пользователя системы (см. раздел 1.3 настоящего документа), то далее будут приниматься следующие соглашения.

1) Термином “пользователь” обозначается актёр, который может иметь любую роль, определённую в рамках ТЗ.

2) Везде, где возникает неоднозначность, для актёра будут определены следующие термины:

Термин “Клиент” обозначает актёра, являющимся авторизованным пользователем системы, обладающего ролью клиента.

Термин “Кинодиректор” обозначает актёра, являющимся авторизованным пользователем системы, обладающего ролью администратора кинотеатра.

Термин “Администратор” обозначает актёра, являющимся авторизованным пользователем системы, обладающего ролью администратора всей системы.

Дополнительно обозначим в качестве термина “Логин” адрес электронной почты, определённый в ТЗ.

Если пользователь не авторизован, т.е. не вошёл в систему, то Система будет предлагать ему те действия, как если бы он вошёл с ролью “Клиент” (см. подпоток S-1 элемента UseCase U-1). Однако, при попытке совершить действия, требующие авторизацию (просмотр заказов, забронировать билеты) система будет возвращать его в начало элемента UseCase U-1. (Т.е. пользователю будет возвращена форма, для ввода логина и пароля.)

Подпотоки S считаются выполненными, если они не прерываются никакими альтернативными сценариями E. Начало и конец элементов UseCase “U” определены в их описании. Если Пользователь является *неавторизованным*, то любой элемент UseCase для такого Пользователя должен предоставлять возможность выбрать действие ВОЙТИ, либо РЕГИСТРАЦИЯ ПОЛЬЗОВАТЕЛЯ в качестве “Клиента” или “Кинодиректора”.

**U-1: Элемент “Вход в систему”.**

**Предусловие:** Пользователь должен быть не авторизованным.

**Главный поток:** Этот элемент начинается, когда пользователю необходимо войти в систему для совершения действий, требующих авторизации (совершение заказа, изменение, добавление, удаление информации о кинотеатрах, сеансов). Система запрашивает от пользователя логин и пароль. Система также предлагает Пользователю зарегистрироваться в качестве Клиента или Кинодиректора, предоставляя ссылки на страницы регистрации. Пользователь может ввести свой логин и пароль и Система проверит их (E-1,E-2,E-3). После успешной проверки, система совершает вход, вносит в статистику информацию о входе, и, в зависимости от роли пользователя предлагает ему выбрать одно из действий:

1) Если пользователь вошёл с ролью “Клиента”, то выполнить подпоток (S-1).

2) Если пользователь вошёл как “Кинодиректор”, то выполнить подпоток (S-2).

3) Если пользователь вошёл как Администратор, то выполнить подпоток (S-3).

Если Пользователь не вводил данные, а выбрал регистрацию в системе то:

4) Если пользователь выбрал регистрацию в качестве “Клиента”, то выполнить элемент UseCase U-6 (E-15), а затем осуществить вход в систему (E-3), и начать подпоток S-1.

5) Если пользователь выбрал регистрацию в качестве “Кинодиректора”, то выполнить элемент UseCase U-7 (E-16), а затем осуществить вход в систему (E-3), и начать подпоток S-2.

**Подпотоки:**

**S-1: вход в качестве клиента.**

Система предлагает клиенту выбрать одно из действий: **Просмотр заказов, просмотр списка кинотеатров, просмотр списка фильмов, выход из системы.**

1. Если выбрано действие **Просмотр заказов,** то осуществить вход в элемент U-2. (Начинается элемент UseCase U-2: Просмотр заказов).

2. Если выбрано действие **просмотр списка кинотеатров,** то начинается элемент UseCase U-3: Просмотр списка кинотеатров.

3. Если выбрано действие **просмотр списка фильмов,** то начинается элемент UseCase U-4: Просмотр списка фильмов.

4. Если выбрано действие **выход из системы,** то элемент UseCase заканчивается (Е-4).

**S-2: вход в качестве кинодиректора.**

Система предлагает кинодиректору выбрать одно из действий: **Просмотр списка сеансов, просмотр списка комнат в кинотеатре, изменение данных о кинотеатре, выход из системы.**

Кроме того, система также предлагает те же действия что и для клиента. Если кинодиректор выбирает клиентское действие (такое, как просмотр заказов), то выполняется последовательность действий, описанных в подпотоке S-1.

1. Если выбрано действие **просмотр списка сеансов,** то начинается элемент UseCase U-9.

2. Если выбрано действие **просмотр списка комнат,** то начинается элемент UseCase U-8.

4. Если выбрано действие **изменение данных о кинотеатре,** то начинается элемент UseCase U-10.

5. Если выбрано действие **выход из системы,** то элемент UseCase заканчивается (Е-4).

**S-3: вход в качестве администратора системы.**

Система возвращает страницу для Администратора системы.

Администратор системы может **получить список пользователей, просмотреть статистику, выйти из системы.**

1. Если выбрано действие **Получить список пользователей,** то начинается элемент U-11.

2. Если выбрано действие **получить статистику,** то начинается элемент U-12.

3. Если выбрано действие **выход из системы,** то элемент UseCase заканчивается (Е-4).

**Альтернативные потоки:**

**E-1: Введён отсутствующий логин, пароль.** Система сообщает об отсутствии аккаунта с введённым логином или паролем. Пользователь может попробовать ввести данные ещё раз, либо начать элемент UseCase U-5, либо завершить элемент UseCase.

**E-2: Логин и пароль либо не введены, либо имеют неправильный формат.** Система сообщает о необходимости заполнении полей формы, представляющих логин и пароль пользователю, если он их не заполнил. Также система может сообщить о том, что поле имеет неправильный формат, если пользователь использует недопустимые символы в данных. Пользователь может попробовать ввести данные ещё раз, либо начать элемент UseCase U-5, либо завершить элемент UseCase.

**E-3: Сбой системы во время входа.** Система не отвечает на запросы пользователя. В данном случае прекращается действие элемента UseCase.

**E-4: Сбой системы во время выхода.** Во время процедуры выхода система дала сбой. В данном случае прекращается действие элемента UseCase.

**U-2: Элемент “Просмотр заказов”.**

**Предусловие:** Пользователь должен быть авторизованным (т.е. он должен осуществить вход в систему, см. эл. U-1).

**Главный поток:** Этот элемент начинается, когда пользователь выбрал действие **ПРОСМОТР ЗАКАЗОВ.** Система отображает пользователю его текущие заказы (Е-5,E-7). Каждый элемент списка (т.е. заказ) представлен его суммой (суммарная стоимость выбранных мест, т.е. заказанных билетов), информацией о наличии скидки или акции, а также ссылкой на список зарезервированных мест, связанных с данным заказом. При нажатии на такую ссылку выполняется подпоток S-4. Пользователь может также нажать кнопку BACK или вернуться на одну ссылку назад, тем самым возвращаясь к подпотоку S-1, S-2 или S-3 в зависимости от роли пользователя.

**Подпотоки:**

**S-4: Получение списка зарезервированных мест, связанных с заказом.**

Система выдаёт информацию о заказе (E-6,E-7), в виде списка зарезервированных мест. Такой список представлен элементами, состоящих из названия кинотеатра, номера зала, категории зала, номера места, стоимости места. Пользователь может нажать кнопку BACK или вернуться на одну ссылку назад, тем самым возвращаясь к началу UseCase элемента U-2.

**Альтернативные потоки:**

**E-5: Произошёл сбой во время получения списка заказов.**

Система выдаёт сообщение пользователю об ошибке и происходит возврат в один из подпотоках S-1,S-2,S-3 элемента UseCase U-1 в зависимости от роля пользователя.

**E-6: Произошёл сбой во время получения списка зарезервированных мест.**

Система выдаёт сообщение пользователю об ошибке и возвращается в начало элемента UseCase U-2.

**E-7: Истекло время сессии (сеанса).**

Система выдаёт сообщение пользователю об истечении времени активной сессии и требует повторного входа в систему. Происходит возврат в начало элемента UseCase U-1. Система возвращает модальное окно с текстовыми полями ввода (для логина и пароля). Пользователь заполняет их, совершает вход в систему, и при успешном выполнении действия, возвращается в исходный подпоток.

**U-3: Элемент “Просмотр списка кинотеатров”.**

**Предусловие:** Пользователь должен выбрать действие «**Просмотр списка кинотеатров».** Авторизация при этом не требуется.

**Главный поток:** Система отображает список кинотеатров (E-8). Пользователь может отфильтровать такой список по названию кинотеатра, его адресу (представленным наименованием страны, города, региона, улицы) (E-8). Также пользователь может выбрать ровно один элемент из списка. Выбор элемента вызывает следующий под-поток S-5.

**Подпотоки:**

**S-5: Выбран кинотеатр.**

При выборе кинотеатра, пользователю возвращается список его текущих сеансов (по текущей дате пользователя) (E-9). Элементы списка содержат название фильма, кинотеатра, номер зала, дату и время сеанса. Пользователь может отобрать нужные сеансы по фильму, номеру зала, его категории, дате и времени сеанса (E-9,E-10). Пользователь может выбрать ровно один сеанс. Выбрав сеанс пользователь может заказать билеты на данный сеанс, т.е. начать элемент UseCase U-5.

**Альтернативные потоки:**

**E-8: Произошёл сбой во время получения списка кинотеатров.**

Система пытается получить данные с баз данных. В случае сбоя, пользователю приходит сообщение об ошибке (о временной недоступности просмотра кинотеатров.) В таком случае, пользователь не сможет оформить заказ, т.к. требуется данные о кинотеатре (в сервисе кинотеатров необходимо получать информацию о наличии мест в залах указанного кинотеатра, а также заносить информацию о резервировании мест в указанном кинотеатре). Также он не сможет получить список зарезервированных мест, т.к. опять требуется информацию о кинотеатре. Данный элемент завершается, и происходит возврат в один из подпотоках S-1,S-2,S-3 элемента UseCase U-1 в зависимости от роля пользователя.

**E-9: Произошёл сбой во время получения списка сеансов для указанного кинотеатра.**

Система пытается получить данные с баз данных. В случае сбоя, пользователю приходит сообщение об ошибке (о временной недоступности просмотра сеансов.) В таком случае, пользователь не сможет оформить заказ, т.к. требуется данные о сеансе. Однако, данные о заказах ему будут доступны (в случае авторизованного Клиента, Кинодиректора и исправности функции получения оформленных заказов сервиса сеансов). Данный элемент завершается, и происходит возврат в один из подпотоках S-1,S-2,S-3 элемента UseCase U-1 в зависимости от роля пользователя.

**E-10: Отказ в работе целого сервиса киносеансов.**

В случае сбоя, пользователю приходит сообщение об ошибке (о временной недоступности целого сервиса.) В таком случае, пользователь не сможет оформить заказ, т.к. требуется данные о сеансе. Также пользователь не сможет посмотреть свои заказы, т.к. это функция сервиса киносеансов, который не работает в данной ситуации. Данный элемент завершается, и происходит возврат в один из подпотоках S-1,S-2,S-3 элемента UseCase U-1 в зависимости от роля пользователя.

**U-4: Элемент “Просмотр списка фильмов”.**

**Предусловие:** Пользователь должен выбрать действие «**Просмотр списка фильмов».** Авторизация при этом не требуется.

**Главный поток:** Система отображает список фильмов (E-11). Пользователь может отфильтровать такой список, введя строку с названием фильма или его жанром. (E-11). Жанр пользователь выбирает посредством динамического выпадающего списка с полем ввода. Также пользователь может выбрать ровно один элемент из списка. Выбор элемента вызывает следующий под-поток S-6.

**S-6: Выбран фильм.**

При выборе фильма, пользователю возвращается список текущих сеансов (по текущей дате пользователя), содержащих данный фильм (E-9). Пользователь может отобрать нужные сеансы по кинотеатру, номеру зала, его категории, дате и времени сеанса (E-10,E-12). Пользователь может выбрать ровно один сеанс. Выбрав сеанс пользователь может заказать билеты на данный сеанс, т.е. начать элемент UseCase U-5.

**Альтернативные потоки:**

**E-11: Произошёл сбой во время получения списка фильмов.**

Система пытается получить данные с баз данных. В случае сбоя, пользователю приходит сообщение об ошибке (о временной недоступности просмотра информации о фильмах.) Пользователь может совершать заказы, но только если сервис кинотеатров и сервис киносеансов полностью исправно работают (т.к. сервис киносеансов хранит информацию о сеансах, где есть уже информация о фильме, а сервис кинотеатров хранит данные о кинотеатре его залах и наличии в нём свободных мест). Происходит возврат в один из подпотоках S-1,S-2,S-3 элемента UseCase U-1 в зависимости от роля пользователя.

**E-12: Произошёл сбой во время получения списка сеансов для указанного фильма.**

Система пытается получить данные с баз данных. В случае сбоя, пользователю приходит сообщение об ошибке (о временной недоступности просмотра сеансов.) В таком случае, пользователь не сможет оформить заказ, т.к. требуется данные о сеансе. Однако, данные о заказах ему будут доступны (в случае авторизованного Клиента, Кинодиректора и исправности функции получения оформленных заказов сервиса сеансов). Данный элемент завершается, и происходит возврат в один из подпотоках S-1,S-2,S-3 элемента UseCase U-1 в зависимости от роля пользователя.

**U-5: Элемент “Заказать билеты”.**

**Предусловие:** пользователь должен быть авторизован. Пользователь также должен выбрать сеанс либо для указанного кинотеатра (должен выполниться подпоток S-5 элемента U-3), либо для указанного фильма (должен выполниться подпоток S-6 элемента U-4).

**Главный поток:** Система получает информацию о местах в указанном зале указанного кинотеатра, в котором будет показан указанный фильм в определённое время. Данные о фильме, кинотеатре, зале, времени сеанса уже содержатся в выбранном сеансе.Получив данные, система отображает пользователю массив из элементов, являющихся местами в зале кинотеатра. Номер зала, и сам кинотеатр указан в сеансе (E-13). Пользователь может выбрать несколько мест в зале. В верхнем углу по центру будет отображаться суммарная стоимость выбранных мест. При нажатии на место, фронтэнд приложение выдает информацию о месте (стоимость билета, скидки). Пользователь может либо выбрать места и зарезервировать их, либо вернуться назад, либо перейти к просмотру списка кинотеатров, фильмов.

1. Если пользователь выбрал билеты и нажал, save booking, то происходит выполнение подпотока S-7.

2. Если пользователь выбрал действие **ПОКАЗАТЬ СПИСОК ФИЛЬМОВ**, то UseCase элемент завершается, и происходит возврат в начало элемента UseCase U-4.

3. Если пользователь выбрал действие показать список кинотеатров, то UseCase элемент завершается, и происходит возврат в начало элемента UseCase U-3.

**Подпотоки:**

**S-7 Оформить заказ (бронь) (save booking).**

Выбрав места и действие **СОХРАНИТЬ БРОНИРОВАНИЕ,** (E-7)система изменяет состояния выбранных пользователем мест для указанной комнаты указанного кинотеатра. Она также создаёт запись в таблице заказов в базе данных, указывая идентификатор текущего пользователя и суммарную стоимость заказа. Затем для такого заказа она создаёт записи для каждого выбранного места, сохраняя номер зала, категория зала, номер места, его стоимость, идентификатор заказа в виде отдельной сущности OrderItem (E-14,E-7).

**Альтернативные потоки:**

**E-13: Не удаётся получить информацию о наличии мест в указанном зале для указанного кинотеатра.**

Система не может получить список мест для указанного сеанса в указанном кинотеатре, в указанном зале. В этом случае невозможно совершить бронь. Система оповещает об этом пользователя и возвращает в предыдущий подпоток, в котором начинался данные элемент UseCase. (S-5,S-6)

**E-14: Сбои системы во время совершения бронирования.**

В момент сохранения (занесения информации о заказе в базу данных) произошёл сбой в системе. В этом случае система должна по журналу логов от сервиса учёта откатить не до конца выполненное действие. Система также должна оповестить об этом пользователя, завершить элемент UseCase U-5 и, если это возможно вернуть его в начало элемента UseCase U-5. Если же у пользователя истекла сессия (и, возможно, это послужило причиной прерывания операции бронирования), то система должна также сообщить об этом пользователю и вернуть его в начало элемента UseCase U-1.

**E-7: Истекло время сессии.**

Данный поток ничем не отличается от альтернативного потока E-7 элемента UseCase U-2.

**U-6: Элемент “Регистрация клиента”**

**Предусловие:** Пользователь должен выбрать действие **РЕГИСТРАЦИЯ КЛИЕНТА** в главном потоке элемента UseCase U-1.

**Главный поток:** Система возвращает форму для регистрации, содержащую текстовые поля для фамилии, имени, отчества, возвраста (системе передаётся строковое представление числа), электронной почты и пароля. Пользователь обязан заполнить все поля, для регистрации, соблюдая формат данных для каждого поля формы. Форма имеет загаловок, указывающий тип регистрации (Клиента, Кинодиректора). Заполнив все поля, пользователь сможет выбрать действие **ЗАРЕГИСТРИРОВАТЬСЯ.** Для этого надо нажать на кнопку SIGN UP (E-15). Пользователь также может отменить регистрацию, вернувшись:

1) Либо в тот подпоток того элемента UseCase, который инициировал начало элемента UseCase U-1 (Тот подпоток, который заставил систему вернуть форму для авторизации, так как возможно из этой формы, пользователь перешёл по ссылке на форму регистрации).

2) Либо в предыдущий подпоток, в котором Пользователь был неавторизован.

**Альтернативные потоки:**

**E-15: Не удалось сохранить персональные данные Пользователя.**

Система дала сбой при попытке зарегистрировать пользователя. Возможен сбой сервиса Авторизации. Система должна сообщить пользователю об ошибке. В этом случае происходит возврат в начало элемента UseCase U-6.

**U-7: Элемент “Регистрация кинодиректора (администратора кинотеатра)”**

**Предусловие:** Пользователь должен выбрать действие **РЕГИСТРАЦИЯ АДМИНИСТРАТОРА КИНОТЕАТРА** в главном потоке элемента UseCase U-1.

**Главный поток:** Система возвращает форму для регистрации, которая разделена на две части. Форма имеет загаловок, указывающий тип регистрации (Клиента, Кинодиректора), а также этап регистрации (персональные данные, информация о кинотеатре). Первая часть представляет из себя исходную форму для регистрации Клиента. Заполнив все поля, пользователь сможет нажать на кнопку NEXT, перейдя во вторую часть формы. С помощью кнопки BACK, пользователь может вернуться к первой части формы. Во второй части формы содержатся текстовые поля для ввода данных о названии кинотеатре и его адресе (в виде отдельных полей для названия страны, города, региона, улицы и номера здании). Заполнив все поля всех частей формы, пользователь сможет выбрать действие **ЗАРЕГИСТРИРОВАТЬСЯ.** Для этого надо нажать на кнопку SIGN UP (E-15, E-16). Пользователь также может отменить регистрацию, вернувшись:

1) Либо в тот подпоток того элемента UseCase, который инициировал начало элемента UseCase U-1 (Тот подпоток, который заставил систему вернуть форму для авторизации, так как возможно из этой формы, пользователь перешёл по ссылке на форму регистрации).

2) Либо в предыдущий подпоток, в котором Пользователь был неавторизован.

**Альтернативные потоки:**

**E-16: Не удалось выполнить регистрацию Кинодиректора.**

Система дала сбой при попытке зарегистрировать пользователя. Возможен сбой сервиса Авторизации. Система должна сообщить пользователю об ошибке. В этом случае происходит возврат в начало элемента UseCase U-7. Если же отказал сервис Кинотеатров, и невозможно вставить данные о кинотеатре, то Система выполняет откат операции вставки персональных данных пользователя.

**U-8: Элемент “Просмотр списка комнат”.**

**Предусловие:** Пользователь должен быть авторизован. Авторизованный пользотель должен иметь роль “Администратора кинотеатра” и выбрать действие **ПРОСМОТР СПИСКА КОМНАТ КИНОТЕАТРА.**

**Главный поток:** Система возвращает пользователю список комнат для кинотеатра, указанного при регистрации Администратора кинотеатра (Е-17).

Система также предлагает следующие действия, описанные в подпотоках элемента, для каждой сущности в списке.

**Подпотоки:**

**S-8: Просмотр списка сеансов.**

Выбрав одну комнату, пользователь может посмотреть связанные с ней сеансы, нажав на кнопку GET SEANCES. С данного действия начинается элемент UseCase U-9.

**S-9: Редактирование комнаты.**

Выбрав одну комнату, пользоатель может изменить её данные, нажав на кнопку EDIT ENTRY. Система возвращает форму с заполненным текстовым полем, представляющее текущую категорию зала, а также панель, на которой расположены места текущей комнаты. Пользователь может добавить на панель новые места, или удалить текущие, а также перемещать их посредством мыши, изменяя представление панели. Каждое место представлено графически маленькой панелью. При нажатии на панель всплывает модальное окно с единственным числовым полем для присваивания месту его номеру. Пользователь может назначить номер для каждого места (E-18).

**S-10: Удаление комнаты.**

Пользователь может выбрать несколько комнат и удалить их, нажав на кнопку DELETE ENTRIES. (Выбор нескольких элементов осуществляется посредством клика на флажок (Checkbox) элемента списка (E-19).

**Альтернативные потоки:**

**E-17: Невозможно получить список комнат.**

Система возвращает пользователю сообщение об ошибке. Данный элемент UseCase завершается. Происходит возврат в подпоток S-2 элемента U-1.

**E-18: Невозможно сохранить изменения в комнате.**

Система возвращает пользователю сообщение об ошибке. Данный элемент UseCase завершается. Происходит возврат в подпоток S-2 элемента U-1.

**E-19: Невозможно удалить комнаты.**

В случае если невозможно удалить комнату, система совершает откат операций и сообщает пользователю об этом. Данный элемент UseCase завершается. Происходит возврат в подпоток S-2 элемента U-1.

**U-9: Элемент “Просмотр списка сеансов”.**

**Предусловие:** Пользователь должен быть авторизован. Авторизованный пользотель должен иметь роль “Администратора кинотеатра” и выбрать действие **ПРОСМОТР СПИСКА СЕАНСОВ.**

**Главный поток:** Система возвращает пользователю список сеансов для кинотеатра, указанного при регистрации Администратора кинотеатра (Е-9). Система предлагает также создать новый сеанс, или группу сеансов размещая кнопки CREATE NEW SÉANCE и CREATE SEANCES над списком. Для каждого элемента из списка сеансов система предлагает выбрать два действия, описанные в подпотоках S-11,S-12. Нажав кнопку CREATE NEW SÉANCE пользователь попадает в подпоток S-13. Чтобы создать группу сеансов, пользователь нажимает на кнопку CREATE SEANCES и попадает в подпоток S-14.

**Подпотоки:**

**S-11: Удалить выбранный сеанс.**

Выбрав сеанс, Кинодиректор может его удалить. Для удаления, Кинодиректор может выбрать несколько сеансов. При наличии заказов на указанные сеансы Система должна оповестить всех Клиентов, забронировавших билеты на данный сеанс, а также удалить соответствующие заказы (E-20). Система удаляет информацию о сеансе (E-21), и при успешном выполнении операции оповещает об этом Пользователя. Система также проверяет, не остались ли заказы, ссылающиеся на не существующий сеанс и, если остались, то удаляет данные заказы.

**S-12: Изменить сеанс.**

Выбрав один сеанс, пользоваетль может изменить его данные, нажав на кнопку EDIT ENTRY. Система возвращает форму с заполненными полями текстовых данных, содержащих текущие значения. Форма состоит из двух частей. Первая часть состоит из трёх полей: наименования фильма, даты и времени. С помощью кнопок NEXT и BACK Пользователь может переходить между частями формы. Вторая часть формы содержит список билетов, привязанных к местам в указанном зале кинотеатра. Каждый элемент списка билетов содержит номер места, цену, наличие акции или скидки на место.

**S-13 Создать новый сеанс.**

Нажав на кнопку CREATE NEW SÉANCE, пользователь выбрал действие: **СОЗДАТЬ НОВЫЙ СЕАНС.** Система возвращает форму, состоящую из трёх частей. Первая часть состоит из трёх полей: наименования фильма, даты и времени. С помощью кнопок NEXT и BACK Пользователь может переходить по частям в рамках одной формы. Вторая часть формы содержит список мест в текущей комнате. Каждое место представлено на интерфейсе в виде маленькой панели, по нажатию на которой всплывает модальное окно, содержащую форму, состоящую из двух текстовых полей: цена билета на место, скидка или акция на билет. Заполнив поля окна, пользователь может сохранить изменения для отдельного места.

**S-14 Создать несколько сеансов.**

Нажав на кнопку CREATE SÉANCES, пользователь выбрал действие: **СОЗДАТЬ ГРУППУ СЕАНСОВ.** Система возвращает форму с панелью, на которую можно перетаскивать файлы в формате json. Система сообщает, что должен содержать файл, содержащий информацию о сеансах. При отправке файла, Система проверяет формат данных, и содержимое файла. Если файл не удовлетворяет ограничениям по формату и структуре, то Система сообщает об ошибке и возвращает Пользователя в начало потока S-11. Если произошла ошибка при сохранении данных в базе данных, то Система совершает откат операций, оповещает об этом Пользователя и возвращает его в начало потока S-11.

**Альтернативные потоки:**

**E-9: Произошёл сбой во время получения списка сеансов для указанного кинотеатра или указанной комнаты.**

См. описание альтернативного потока E-9 элемента UseCase U-3.

**E-20: Произошёл сбой во время удаления заказов для удаляемого сеанса.**

В случае сбоя, система совершает откат операций. Пользователю возвращается сообщение об ошибке, и происходит возврат в начало элемента UseCase U-8.

**E-21: Произошёл сбой во время удаления сеанса.**

В случае сбоя, пользователю возвращается сообщение об ошибке, и происходит возврат в начало элемента UseCase U-8. Если при удалении множества сеансов один из них не удалился по причине сбоя, то система также выполняет откат операций, оповещая об этом Пользователя.

**U-10: Элемент “Изменить данные о кинотеатре”.**

**Предусловие:** Пользователь должен быть авторизован. Авторизованный пользотель должен иметь роль “Администратора кинотеатра” и выбрать действие **ИЗМЕНЕНИЕ ДАННЫХ О КИНОТЕАТРЕ.**

**Главный поток:** Система возвращает пользователю форму с текстовыми полями, которые заполнены текущими значениями сущности кинотеатра. Поля содержат наименование кинотеатра и его адрес, представленный в виде отдельных полей (страна, город, регион, улица, номер дома). Изменив данные поля, Пользователь может сохранить изменения нажав на кнопку SAVE CHANGES (E-22,E-23). Редактирование комнат происходит в элементе UseCase U-8.

**Альтернативные потоки:**

**E-22: Значения полей не соответствует формату и ограничениям системы.**

Пользователь некорректно ввёл данные, либо оставил все поля пустыми. Система сообщает об этом пользователю, размещая сообщение под каждым неправильно заполненным полем. Сообщение содержит информацию о том, как правильно надо заполнить то или иное поле. В случае, когда пользователь вводит данные, удовлетворяющие формату данных, но противоречащие бизнес-логике, то Система также возвращает сообщение об ошибке и не сохраняет изменения. Происходит возврат в начало элемента UseCase U-10.

**E-23: Не удалось сохранить изменения.**

В результате сбоя системы не удалось сохранить данные в базе данных. Система сообщает об этом пользователю, и происходит возврат в начало элемента UseCase U-10.

**U-11: Элемент “Получить список пользователей”.**

**Предусловие:** Пользователь должен быть авторизован. Авторизованный пользотель должен иметь роль “Администратора системы” и выбрать действие **ПОЛУЧИТЬ СПИСОК ПОЛЬЗОВАТЕЛЕЙ.**

**Главный поток:** Система возвращает список пользователей (E-24). Элемент списка содержит ФИО пользователя, его логин. Для каждого элемента списка Администратор может удалять пользователей, блокировать их на определённый промежуток времени. В случае неуспешного блокирования или удаления как одного так и нескольких пользователей (а также в случае, когда для выбранных пользователей действие завершилось успешно лишь для части из них) система совершает откат операций с сообщением об ошибке и данный элемент UseCase завершается.

**Альтернативные потоки:**

**E-24: Невозможно получить список.**

Система сообщает Пользователю об ошибке. Данный элемент UseCase завершается.

**U-12: Элемент “Получить статистику”**

**Предусловие:** Пользователь должен быть авторизован. Авторизованный пользотель должен иметь роль “Администратора системы” и выбрать действие **ПОЛУЧИТЬ СТАТИСТИКУ.**

**Главный поток:** Система возвращает Пользователю список (E-24), состоящий из элементов, содержащих имя и логин пользователя, совершившего действие, наименование действия, дату и время операции, а также состояние операции (успешно завершилась, произошла ошибка).

**Альтернативные потоки:**

**E-24: Невозможно получить список.**

См. альтернативный поток E-24 элемента UseCase U-11.

**2.3 Взаимодействие между сервисами, системой и пользователями.**

В предыдущем подразделе было указано, что может делать пользователь системы и как система реагирует на его действия. В настоящем подразделе указана основная последовательность действий.

1. Для действий, требующих авторизацию, пользователь перенаправляется на страницу авторизации.
2. Действия, не требующие авторизацию, это: просмотр списка сеансов, фильмов, жанров, кинотеатров, доступных мест на выбранный сеанс.
3. При обращении по корневому адресу приложения (сайта), клиент попадает на страницу со списком кинотеатров.
4. Выбрав из списка кинотеатров один из них, клиент получает список фильмов, для которых существует хотя бы один сеанс в указанном кинотеатре.
5. Выбрав из списка фильмов желаемый фильм (по имени, жанру, рейтингу), клиент получает список сеансов. Если до этого действия клиент выбирал кинотеатр (п.4), то сеансы будут выбраны не только по идентификатору фильма, но и по идентификатору кинотеатра.
6. Выбрав из списка сеансов желаемый сеанс, клиент получает список доступных мест, которые ещё не забронированы другими. Номер зала, в котором находятся места, определяется по идентификатору зала, хранимый в сеансе. Если пользователь имеет роль *Администратора Кинотеатра*, то он также может получить список всех мест, связанных с комнатой, идентификатор которой указан в сеансе. (Т.е. он может выбрать для данного элемента списка сеансов список всех мест (не связанных с билетами), нажав соответствующую кнопку на элементе списка). Администратор кинотеатра может редактировать данные о места (в частности, удалять билеты, связанные с ними, а также сами места). При удалении места, удаляются все связанные с ним билеты. Администратор кинотеатра также может удалять сеансы либо из списка, либо в самой комнате. В самой комнате администратор кинотеатра удаляет лишь выбранный сеанс (так как он получил доступ к комнате по сеансу). При удалении сеанса также удаляются все связанные с ним билеты.
7. Администратор кинотеатра при успешной авторизации заходит на свою домашнюю страницу пользователя со следующими возможностями:
   1. Получить список сеансов, фильмов в кинотеатре пользователя.
   2. Получить список комнат в кинотеатре пользователя.
   3. Редактировать данные о кинотеатре.
   4. Создать новую комнату.
8. Выбрав действие *получить список комнат*, администратор кинотеатра получает список залов. Он может удалять либо изменять элементы списка. Выбрав действие изменить комнату, он получает список всех мест, связанных с данной комнатой. Для этих мест он может назначать либо удалять билеты. Для назначения билетов, он перенаправляется на список доступных сеансов, связанных с данным местом, данной комнатой и данным кинотеатром. Он может выбрать один из сеансов и назначать билеты на указанный сеанс, но фактически, он перенаправляется на тот же список из пункта 6. При удалении комнаты удаляется вся связанная с ней информация: места, сеансы, билеты, заказы.
9. Выбрав действие *создать комнату*, администратор кинотеатра заполняет соответствующую форму, и при корректном заполнении формы, он попадает на страницу комнаты, где он может назначить сеанс для данной комнаты, либо создавать, удалять, редактировать места. Для назначения билетов к указанным местам, он должен сначала создать сеанс для новой созданной комнаты.
10. Выбрав действие *редактирование данных о кинотеатре*, администратор кинотеатра перенаправляется на страницу с формой, содержащей текущие данные о кинотеатре.
11. Администратор кинотеатра может заявить об удалении учётной записи (т.е. аккаунта из системы). В этом случае удаляется кинотеатр и вся связанная с ним информация (билеты, заказы и т.д.)
12. Администратор системы при успешной авторизации получает свою домашнюю страницу со следующими функциями:
    1. Получить список пользователей системы.
    2. Получить статистику от сервиса системы по URL.
    3. Получить списки кинотеатров и фильмов (п.3-6)

Выбрав пользователя из списка, Администратор системы может его заблокировать, удалить, либо направить уведомление.

В любой момент времени пользователь имеет возможность выйти из системы независимо от последовательности совершённых действий. Единственное условие: пользователь должен быть авторизован.

Ниже дана диаграмма переходов, узлы которой содержат наименования операций. Ребро, исходящее из узла, обозначает совершение операции. Входящие рёбра указывают на набор действий, которые может осуществить пользователь системы. Начальное состояние означает вход в систему. В целях упрощения диаграммы, здесь не показан переход в финальное состояние. C каждой вершины в диаграмме можно осуществить переход в финальное состояние, обозначающее выход из системы. На каждом ребре указано условие, по которому возможно осуществить переход.

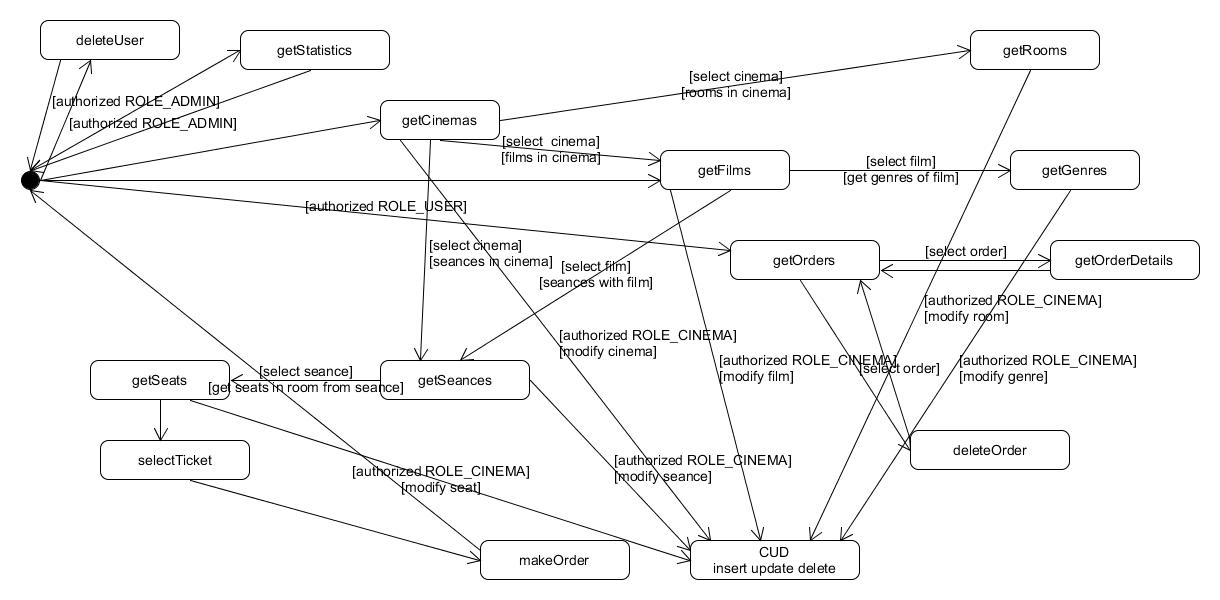


Рис. 5. Диаграмма переходов действий пользователей.

**3. Технологический раздел.**

**3.1 Структура проекта.**

Программное окружение проекта было составлено на основе критериев кроссплатформенности, и количеству доступных библиотек и функций, которые необходимы для разработки. Оно состоит из следующих элементов:

* Язык программирования: Java версии.12.
* Используемые СУБД: MS SQL Server 2016, My SQL.
* Фреймворки: Spring Framework текущей версии, модуль Spring Boot для Spring Framework версии 2.0 и выше, React JS (фреймворк для разработки frontend приложения), Hibernate версии 5 или выше.
* Библиотеки: Lombok ver. 1.18.2 (для автоматической генерации методов доступа и записи к полям класса), jacoco ver. 0.8.2, (для формирования отчётов по тестам), GSON текущей версии (для сериализации/десериализации JSON-данных).

Корневым каталогом проекта является папка с именем *CinemaStreamMicroservices.* В ней содержится папка проекта *Клиентского приложения WebApp* и папка с проектами отдельных сервисов *Microservices.* Клиентское приложение состоит из двух частей: модуля *frontend-а (интерфейс системы)* представляющий одностраничное приложение (лежит в папке *CinemaStreamMicroservices/WebApp/frontend*), реализованное фреймворком ReactJS и модуля *backend,* представляющий gateway-сервис, используемый для обращения ко всем сервисам системы.

Данный модуль лежит в папке *CinemaStreamMicroservices* /*WebApp*/*ApiGatewayApp*. В каталоге *Microservices* лежат корневые каталоги приложений, реализующих определённый сервис. В каждом таком каталоге, содержатся следующие файлы: файлы сборки *build.gradle, gradle.properties, settings.gradle, gradlew, gradlew.bat,* файл - глобальных свойств проекта – system.properties.

Каждый проект реализован на языке Java 12 версии [5][6]. Для сборки используется система сборки *gradle* [7]. Данная система сборки определяет свой собственный язык, определяющий настройки запуска приложения, его сборки и проведения тестов. В отличие от других систем сборок, таких как maven или ant, *gradle* имеет более компактный файл описания конфигурации.

Все файлы исходного кода хранятся в папке *src*. В каталоге *src* исходники для тестов хранятся в папке *test*, а остальные файлы в папке *main*. Оба каталога имеют папку *java*, содержащую одноимённый пакет*. ru.osipov.deploy*. В пакете, находятся три файла *WebConfig, SecurityConfig*, *DeployApp* с расширением файла .java. Точка входа в приложение определена в классе *DeployApp*, описанный в одноимённом файле. Остальные вышеперечисленные файлы служат для настройки контроллеров и установки политики безопасности в приложении. В пакете *java.ru.osipov.deploy* лежат следующие подпакеты:

* *web* – содержит контроллеры сервиса, которые принимают запросы от пользователя по указанному пути, и возвращают результат. Формат входных и выходных файлов является JSON. Содержимое JSON документов представлено объектами передачи данных (Data Transfer Object – DTO) [8].
* *models* – пакет содержит классы, определяющие объекты передачи данных. Именно они и передаются между контроллерами и пользователями. Интерфейс (View) обрабатывает получаемые объекты.
* entities – пакет содержит классы бизнес-сущностей.
* repositories – пакет содержит интерфейсы репозиториев, реализацию которых выполняет Spring framework. Они используются для обращения к базе данных.
* *сonfiguration* – пакет содержит класс конфигурации фреймворка объектно-реляционного отображения (Object Relational Mapping) *Hibernate*. Также пакет содержит подпакет *jwt*, содержащий настройки для создания и управления JWT-токеном. Данный токен используется для авторизацией между сервисами системы.
* *services* – пакет содержит классы и интерфейсы сервисов, используемых контроллерами. Сервисы делают обращение к репозиториям (хранилищам данным). Репозитории выполняют операции с базами данных (сохранение, изменение, удаление сущности (строки в таблице базы данных)). Сервисы возвращают объекты DTO (Data Transfer Object), представляющие неизменяемый экземпляр данных, передаваемый клиенту из базы данных.
* *exception* – пакет, содержащий классы исключительных ситуации.

Объектно-реляционное представление было выбрано из-за сложных отношений бизнес сущностей. В качестве фреймворка, реализующее ORM был выбран Hibernate версии выше 5.0.. Тестирование осуществляется с помощью библотеки JUnit. Для контроллеров и репозиториев были написаны тесты. Перед сборкой проекта, в первую очередь, выполняются все тесты. Если хотя бы один из тестов не выполняется, сборка не осуществляется. Для сериализации и десериализации данных контроллерами используется библиотека Jackson. А для сериализации и десериализации данных во время выполнения тестов используется библотека *Gson*.

**3.2 Описание схемы Model-View-Controller проекта.**

Каждое приложение (включая gateway service) создано на основе шаблона *модель-контроллер-представление. (Model View Controller).*

В каждом проекте микросервиса определён набор моделей (*Models*), лежащих в пакете *ru.osipov.deploy.models*. Ниже дана диаграмма всех классов моделей разрабатываемой системы. На диаграмме указано, к какому микросервису относится определённый набор моделей. Сервис gateway (проект *ApiGatewayApp*) содержит все модели приложения.

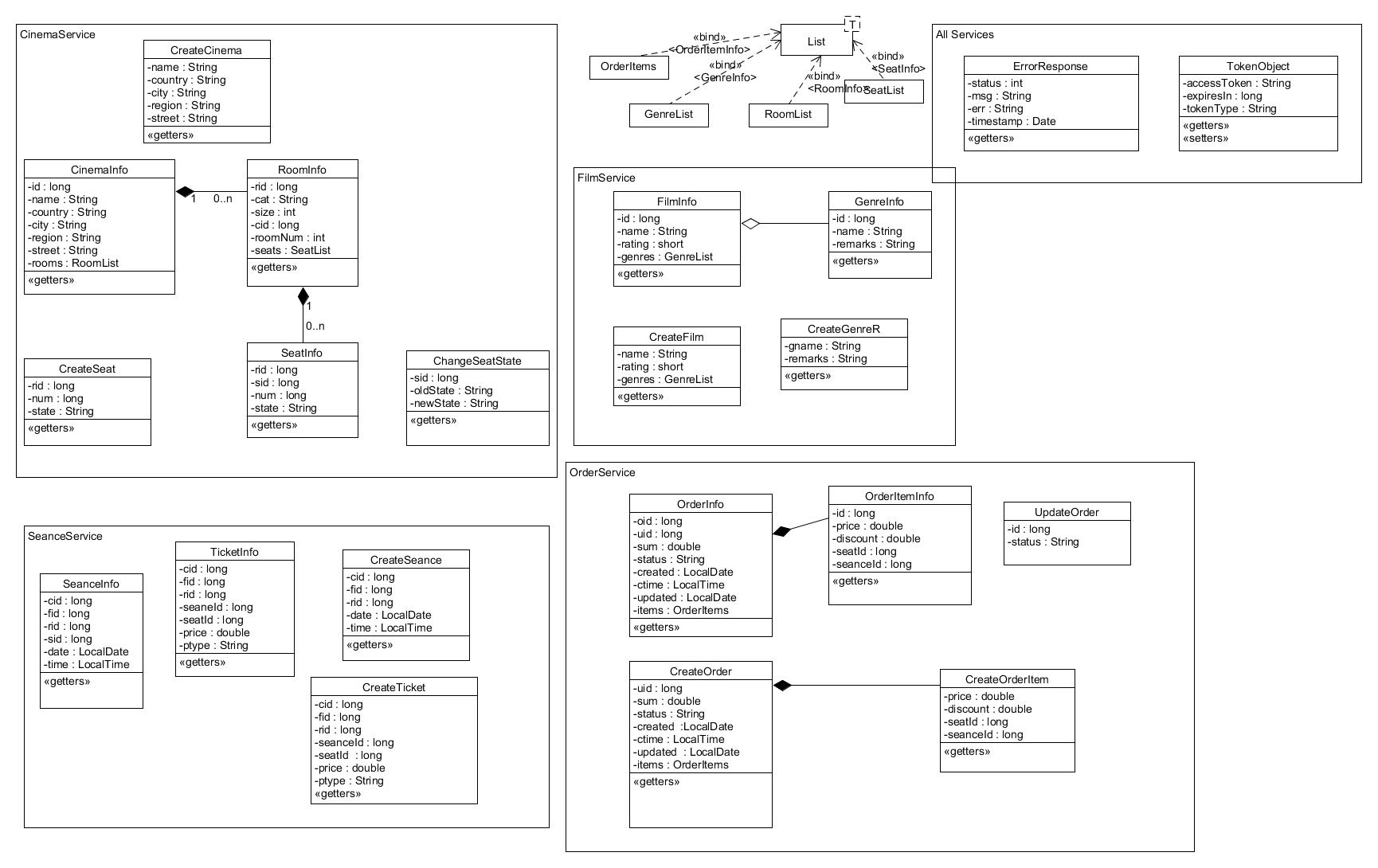


Рис. 6. Диаграмма моделей системы.

Основную работу с базой данных выполняет Spring Data JPA Repository, который позволяет описать интерфейс взаимодействия с базой данных, а также запросы к ней. Запросы пишутся с помощью аннотации *@Query*, и языка HQL (Hibernate Query Language – язык запросов Hibernate).

Repository возвращает данные в виде экземпляра класса бизнес сущности. Чтобы не передавать все данные из базы данных и не компрометировать схему, используются отдельные классы моделей. Работа с базой данных и с запросами пользователей разделена с помощью отдельных классов, реализующих интерфейсы, имена которых имеют суффикс “*Service”*.

Контроллеры определены в пакете *web*. Они реализуют логику по принятию запроса к обработке. URL запросов отображаются на вызовы методов экземпляра класса контроллера. Ниже дано описание прикладного программного интерфейса (API) всей системы, в следующем виде: *url-запроса, тип запроса, http-метод, возвращаемое значение (объект модели DTO), и описание*. Тип запроса, здесь, определяет доступ к операциям пользователям. Тип запроса может быть следующим:

* **PUBLIC** - любой пользователь может выполнить данный запрос.
* **PROTECTED –** любой авторизованный пользователь может выполнить данный запрос. Авторизация происходит либо через стороннее приложение по протоколу openAuth 2.0, либо через фронтэнд приложение с помощью логина и пароля.
* **PROTECTED\_DIR –** любой авторизованный пользователь с ролью Кинодиректора или Администратора системы может выполнить данный запрос.
* **ADMIN\_ONLY –** данный запрос может быть выполнен только Администратором системы.
* **PRIVATE –** данный запрос выполняется сервисом, и является закрытым для пользователя. Никто не может выполнить данный запрос, кроме самого приложения.

Ниже дана основная схема работы микросервисного приложения. Для простоты, здесь не отображён процесс авторизации, поскольку данный процесс описан в следующем разделе (см. параграф 3.3).

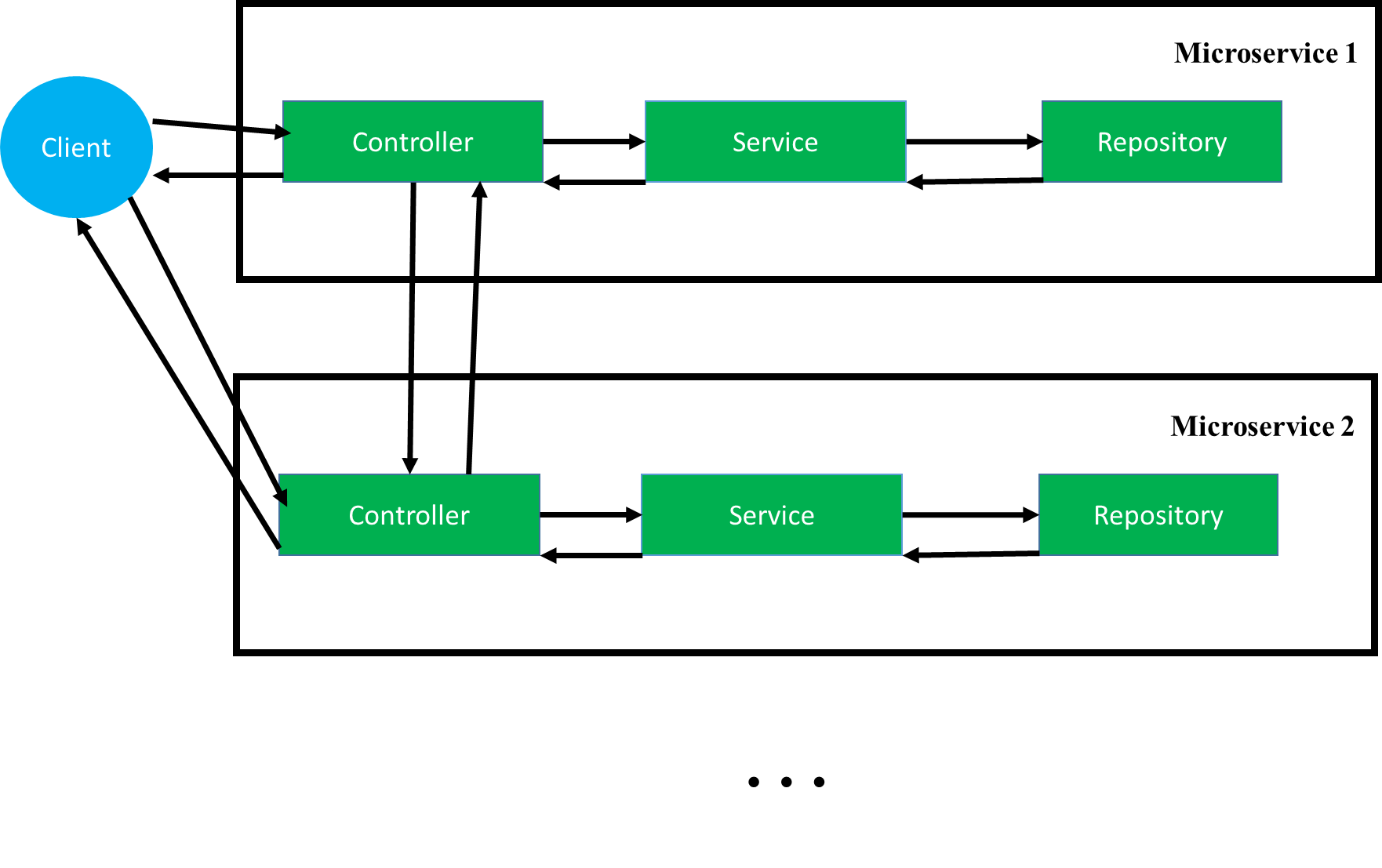


Рис. 7. Основная схема структуры микросервиса.

**3.2.1 API сервиса кинотеатров.**

Данный сервис имеет имя *CinemaService,* и все *url-*пути начинаются с префикса “/CinemaService”. Запросы к кинотеатрам обрабатывает *CinemaController*. Он имеет следующий API:

* */CinemaService/v1/cinemas*, **PUBLIC GET** *List<CinemaInfo>* : получить список кинотеатров. Возвращает список JSON-объектов с информацией о кинотеатре (имя, адрес, идентификатор), а также информацию о комнатах кинотеатра, вместе с местами.
* */CinemaService/v1/cinemas/name/{имя\_кинотеатра},* **PUBLIC GET** *List<CinemaInfo> :* получить кинотеатр по его наименованию. Если такого кинотеатра нет, то возвращается список всех кинотеатров.
* */CinemaService/v1/cinemas/country/{страна},* **PUBLIC GET** *List<CinemaInfo> :* получить кинотеатр по названию страны, в которой он расположен. Если такого кинотеатра нет, то возвращается список всех кинотеатров.
* */CinemaService/v1/cinemas/city/{город },* **PUBLIC GET** *List<CinemaInfo> :* получить кинотеатр по наименованию города, в котором он расположен. Если такого кинотеатра нет, то возвращается список всех кинотеатров.
* */CinemaService/v1/cinemas/street/{название улицы },* **PUBLIC GET** *List<CinemaInfo> :* получить кинотеатр по наименованию улицы, в которой он расположен. Если такого кинотеатра нет, то возвращается список всех кинотеатров.
* */CinemaService/v1/cinemas/{идентификатор кинотеатра },* **PUBLIC GET** *List<CinemaInfo> :* получить кинотеатр по его идентификатору. Если такого кинотеатра нет, то возвращается ошибка со статусом 404.
* */CinemaService/v1/cinemas/create* **PROTECTED\_DIR POST** *ResponseEntity* : создаёт новый кинотеатр в базе кинотеатров и возвращает объект типа *ResponseEntity* в виде JSON-объекта, со свойством *location*, значение которого является url к созданному кинотеатру. В качестве тела запроса, принимает объект типа *CreateCinema*, являющейся моделью. Данный объект преобразуется в JSON-объект, содержащий данные с формы с которой взаимодействует пользователь и frontend-приложение.
* */CinemaService/v1/cinemas/delete/{идентификатор\_кинотеатра}* **PROTECTED\_DIR POST** *ResponseEntity* : удаляет кинотеатр из базы данных по его идентификатору и возвращает JSON-объект, содержащий данные кинотеатра. Если кинотеатр не был найден, вернётся код ошибки 404.
* */CinemaService/v1/cinemas/{идентификатор кинотеатра}* **PROTECTED\_DIR PATCH** *ResponseEntity* : обновляет данные кинотеатра с указанным идентификатором, и возвращает их в виде JSON-объекта.
* */CinemaService/v1/cinemas/{идентификатор\_кинотеатра}/rooms* – корневой url контроллера *RoomsController*. Обрабатывает запросы, связанные с комнатами. Следующие url начинаются с указанного выше префикса.
  + */* **PUBLIC GET** List<RoomInfo> : возвращает список комнат для указанного кинотеатра.
  + */{room\_id}* **PUBLIC GET** ResponseEntity : Возвращает комнату по идентификатору кинотеатра и идентификатора комнаты. Если такой комнаты нет вернёт ошибку с кодом 404.
  + */category/{category\_name}* **PUBLIC GET** List<RoomInfo> : возвращает список комнат по категории. Если категория не существует, то вернётся список всех комнат указанного кинотеатра.
  + */create* **PROTECTED\_DIR POST** ResponseEntity : создаёт комнату в указанном кинотеатре. Данные с формы собираются в объект типа CreateRoom. В случае успеха вернётся JSON-объект, с URL по которому можно найти новую комнату.
  + */delete{room\_id}, /delete* **PROTECTED\_DIR POST** ResponseEntity : удаляет комнату из кинотеатра, а также всё, что с ней связано. В случае успеха вернётся JSON-объект с данными об удалённой комнате. Иначе вернётся ошибка с кодом 404.
  + */update/{room\_id}* **PROTECTED\_DIR PATCH** ResponseEntity : обновляет данные о комнате (размер, категория и т. д.) и возвращае JSON-объект с новыми данными указанной комнаты. Если комната не найдена, то вернётся ошибка 404.
  + */{room\_id}/add,* **PROTECTED\_DIR POST** ResponseEntity : добавляет новое место в указанную комнату. Если указанной комнаты не существует, то вернётся ошибка с кодом 400. В случае успеха вернётся JSON объект с данными о созданном месте. Данные о месте содержит объект класса *CreateSeat*.
  + */{room\_id}/update/{seat\_id}* **PROTECTED\_DIR PATCH** ResponseEntity : обновляет данные о месте в указанной комнате указанного кинотеатра. Новые данные извлекаются из формы и помещаются в объект класса *CreateSeat*. В случае успеха вернётся JSON-объект с новыми данными места, иначе вернётся ошибка 404.
  + */{room\_id}/delete{seat\_id}* **PROTECTED\_DIR POST** ResponseEntity : удаляет место из указанной комнаты указанного кинотеатра. В случае успеха, вернётся JSON-объект с информацией об удалённой комнате. Иначе вернётся ошибка 404.
  + */{room\_id}/edit/state* **PRIVATE PATCH** ResponseEntity : меняет состояние комнаты. Выполняется, когда либо кинодиректор создаёт билеты с указанными местами, либо пользователь покупает или бронирует данные места. Данные запроса представлены списком объектов типа *ChangeSeatState*, который содержит идентификатор места, старое состояние и новое состояние. В случае успеха, вернётся список мест с новым состоянием, иначе вернётся ошибка с кодом 400.

**3.2.2 API для сервиса фильмов.**

Данный сервис имеет имя *FilmService,* и все *url-*пути начинаются с префикса *“/FilmService”*. Запросы к фильмам обрабатывает *FilmController*. Он имеет следующие методы:

* */FilmService/v1/films/* **PUBLIC GET** *List<FilmInfo>* : возвращает список JSON-объектов всех фильмов.
* */FilmService/v1/films/{film\_id}* **PUBLIC GET** ResponseEntity : возвращает JSON-объект, содержащий данные фильма с указанным идентификатором. Если такого фильма нет, то вернётся ошибка 404.
* */FilmService/v1/films/genre/{genre\_name}* **PUBLIC GET** List<FilmInfo> : возвращает список JSON-объектов всех фильмов с указанным жанром (названием жанра *genre\_name,* поле *name* у класса *Genre*). В случае, если такого жанра нет, вернётся пустой список. В случае, если имя жанра не указано, то вернётся ошибка 400.
* */FilmService/v1/films/{film\_id}/genre/add/{genre\_name}* **PROTECTED\_DIR POST** *ResponseEntity* : добавляет жанр с указанным названием к указанному фильму (с указанным идентификатором) к фильму. Если имя жанра не указано вернётся ошибка с кодом 400. Если фильм не найден, или такого жанра не существует, то вернётся ошибка с кодом 404. Иначе вернётся JSON-объект, с новыми данными фильма.
* */FilmService/v1/films/{film\_id}/genre/delete/{genre\_name}* **PROTECTED\_DIR POST** *ResponseEntity* : удаляет жанр с указанным названием из указанного фильма. Если имя жанра не указано, вернётся ошибка с кодом 400. Если фильм не найден, или такого жанра не существует, вернётся ошибка 404. В случае успеха вернутся данные о фильме.
* */FilmService/v1/films/name{film\_name}, /FilmService/v1/films/name/,* **PUBLIC GET** ResponseEntity : возвращает фильм с указанным названием. Если название фильма не указано, вернётся список всех фильмов. Если фильма с указанным названием нет, то вернётся ошибка 404. В случае успеха вернётся JSON-объект, представляющий данные о фильме.
* */FilmService/v1/films?rating=number*, **PUBLIC GET** List<FilmInfo> : возвращает список фильмов с указанным рейтингом (значение *number*). Если параметр не указан, то вернутся все фильмы.
* */FilmService/v1/films/delete/{film\_name}* **PROTECTED POST** ResponseEntity : удаляет фильм с указанным именем. Если фильма нет, вернётся ошибка 404. В случае успеха вернётся JSON-объект с данными об удалённом фильме.

Запросы с жанрами осуществляет *GenreController*. Он имеет следующий API:

* */FilmService/v1/genres/{genre\_id}* **PUBLIC GET** ResponseEntity : возвращает JSON-объект с данными о жанре, с указанным идентификатором. Если такого жанра нет, вернётся ошибка 404.
* */FilmService/v1/genres?name=genreName* **PUBLIC GET** ResponseEntity : возвращает жанр с указанным именем (genreName). Если имя жанра не указано, то вернётся список со всеми жанрами. Если жанр с указанным именем не существует, то вернётся ошибка 404.
* */FilmService/v1/genres/remarks?r=remark&&all=bvalue* **PUBLIC GET** List<GenreInfo> : Возвращает список жанров по указанной ремарке (поле remarks объекта типа *Genre*). Если параметр r не указан (т.е. не указана ремарка), то в зависимости от значения или наличия второго параметра all, вернётся либо список всех жанров *(all = true)*, либо список жанров, у которых нет ремарки *(all = false).* Если параметр *all* не указан, то вернётся список жанров без ремарки (поле *remarks* равно *null*).
* */FilmService/v1/genres/update/{genre\_name}?newName=nName* **PROTECTED\_DIR POST** ResponseEntity : задаёт новое имя указаному жанру. Жанр выбирается по его имени (параметр *genre\_name*). Если не указать старое имя жанра, то его невозможно будет выбрать, следовательно, вернётся ошибка с кодом 400. Если жанр с указанным старым именем *genre\_name* не существует, то вернётся ошибка 404. Если невозможно задать новое имя жанру то вернётся ошибка с кодом 400. В случае успешной операции, вернётся JSON-объект с новыми данными жанра.
* */FilmService/v1/genres/create* **PROTECTED\_DIR POST** ResponseEntity : добавляет новый жанр в базу данных. В случае успешной операции возвращает JSON-объект с url для нового жанра. Данные для нового жанра содержит объект типа *CreateGenreR*. В случае провала вернётся ошибка с кодом 400.
* */FilmService/v1/genres/{genre\_id}* **PROTECTED\_DIR PATCH** ResponseEntity : обновляет указанный жанр с указанным идентификатором. Если такого жанра нет, то вернётся ошибка 404. В случае успешной операции, вернётся JSON-объект с новыми данными жанра. Новые данные для жанра содержатся объекте типа *CreateGenreR*.
* */FilmService/v1/genres/delete/{genre\_id}* **PROTECTED\_DIR POST** ResponseEntity : удаляет жанр с указанным идентификатором. Если такого жанра нет или не указан идентификатор, то вернётся ошибка 404. В случае успеха, вернётся JSON-объект с данными удалённого жанра.
* */FilmService/v1/genres/restore* **PRIVATE POST** ResponseEntity : заново создаёт жанр, в случае провала операции удаления жанра из базы. Возвращает JSON-объект с url созданного жанра.

**3.2.3 API для сервиса сеансов.**

Данный сервис имеет имя *SeanceService* и все url имеют префикс, начинающийся с *“/SeanceService”*.

Запросы к сеансам обрабатывает *SeanceController*. Он имеет следующий API:

* */SeanceService/v1/séances?date=”yyyy-mm-dd”* **PUBLIC GET** List<SeanceInfo> : возвращает список сеансов по дате. Если дата не указана, то вернутся все сеансы.
* */SeanceService/v1/séances/{cinema\_id}, /SeanceService/v1/seances/* **PUBLIC GET** List<SeanceInfo> : возвращает список сеансов, проводимых в указанном кинотеатре. Кинотеатр указывается по его идентификатору. Если такого кинотеатра не существует, то вернётся список всех сеансов.
* */SeanceService/v1/séances/date?d1=’yyyy-mm-dd’&&d2=’yyyy-mm-dd’* **PUBLIC GET** List<SeanceInfo> : возвращает список сеансов, которые имеют дату, принадлежащую указанному временному отрезку дат [d1, d2]. Оба параметра d1 и d2 являются необязательными. В случае их отсутствия, вернётся список всех сеансов. Если указан только параметр d1, то будет получен список сеансов по дате (аналогично запросу по url: /SeanceService/v1/séances?date=d1). Если указан только параметр d2, то вернутся те сеансы, у которых дата не превосходит указанную дату d2 по времени (т.е. не позже даты d2).
* */SeanceService/v1/séances/delete?fid=number* **PROTECTED\_DIR POST** *ResponseEntity* : удаляет сеансы, чей идентификатор фильма *fid* равен указанному числу. Параметр *fid* является обязательным. Если фильма с указанным идентификатором нет, вернётся ошибка 404. В случае успеха вернётся сообщение об успешной операции.
* */SeanceService/v1/séances/{room\_id}/{séance\_id}* **PUBLIC GET** ResponseEntity : получить сеанс по его идентификатору и идентификатору комнаты. Если такого сеанса нет, то вернётся ошибка 404. В случае успеха, вернётся JSON-объект с данными сеанса.
* */SeanceService/v1/séances/{room\_id}/{séance\_id}* **PROTECTED\_DIR PATCH** ResponseEntity : обновляет сеанс, с указанной комнатой и идентификатором сеанса. Если такого сеанса нет, то вернётся ошибка 404. Новые данные сеанса содержит объект класса *CreateSeance*. В случае успеха вернётся JSON-объект с новыми данными сеанса.
* */SeanceService/v1/séances/create* **PROTECTED\_DIR POST** ResponseEntity : создаёт новый сеанс с указанными данными, содержащиеся в типе *CreateSeance*. В случае успешной операции, вернётся JSON-объект с url нового сеанса.
* */SeanceService/v1/séances/delete/{room\_id}/{séance\_id}* **PROTECTED\_DIR POST** ResponseEntity : удаляет сеанс с указанными идентификаторами комнаты и сеанса. Если такого сеанса нет, то вернётся ошибка 404. В случае, когда в ходе удаления информации произошла ошибка, то происходит откат операции.

Для обработки запросов с билетами используется контроллер *TicketController*. Он имеет следующие urls:

* */SeanceService/v1/tickets/{seat\_id}* **PUBLIC GET** ResponseEntity : получить билет по его идентификатору места. Если такого билета нет, вернётся ошибка 404. В случае успеха вернётся JSON-объект с данными билета.
* */SeanceService/v1/tickets/price{price}* **PUBLIC GET** List<TicketInfo> : возвращает список билетов с указанной ценой. Если цена не указана, или она указана неправильно (*price* с отрицательным значением), то вернётся список всех билетов.
* */SeanceService/v1/tickets/create* **PROTECTED\_DIR POST** ResponseEntity : создаёт новый билет. Данные для нового билета содержит объект типа *CreateTicket*. В случае успеха вернётся JSON-объект с url нового билета.
* */SeanceService/v1/tickets/delete/{seat\_id}* **PROTECTED POST** ResponseEntity : удаляет билет с указанным идентификатором места. Если такого билета нет, вернётся ошибка 404. В случае успеха вернётся JSON-объект с данными удалённого билета.

**3.2.4 API сервиса заказов.**

Сервис заказов имеет имя *OrderService*. Все его url имеют префикс, начинающийся с “/OrderService”.

Запросы для обработки заказов осуществляет контроллер *OrderController*. Он имеет следующий API:

* */OrderService/v1/orders/{order\_id}*  **PUBLIC GET** ResponseEntity : получить заказ по его идентификатору. В случае успеха вернётся JSON-объект с данными заказа. Если такого заказа нет, вернётся ошибка 404.
* */OrderService/v1/orders/{user\_id}* **PROTECTED GET** ResponseEntity : возвращает список заказов, представленный списком JSON-объектов, по идентификатору пользователя. Если такого пользователя нет, вернётся ошибка 404. Если пользователь не авторизован вернётся ошибка 401.
* */OrderService/v1/orders/status{order\_status}*  **PUBLIC GET** ResponseEntity : получить заказ по его текущему статусу. В случае успеха вернётся JSON-объект с данными заказа. Если такого заказа нет, вернётся ошибка 404.
* */OrderService/v1/orders/sum/{order\_sum}*  **PUBLIC GET** ResponseEntity : получить заказ по его итоговой сумме. В случае успеха вернётся JSON-объект с данными заказа. Если такого заказа нет, вернётся ошибка 404.
* */OrderService/v1/orders/?cdate=’yyyy-mm-dd’*  **PUBLIC GET** ResponseEntity : получить список заказов по указанной дате создания заказа. В случае успеха вернётся JSON-объект с данными заказов. Если таких заказов нет, вернётся ошибка 404.
* */OrderService/v1/orders/?ctime=’hh:mm:ss’* **PUBLIC GET** ResponseEntity : получить заказы по времени создания заказа. В случае успеха вернётся JSON-объект с данными заказов. Если таких заказов нет, вернётся ошибка 404.
* */OrderService/v1/orders/create* **PROTECTED POST** ResponseEntity : создаёт новый заказ. Данные для нового заказа содержит объект типа *CreateOrder*. Объект *CreateOrder* содержит также позиции заказа, т.е. выбранные места в кинотеатре, представленные типом *CreateOrderItem*. В случае успеха вернётся JSON-объект с url нового заказа.
* */OrderService/v1/orders/delete/{order\_id}* **PROTECTED POST** ResponseEntity : удаляет заказ с указанным идентификатором места. Если такого заказа нет, вернётся ошибка 404. В случае успеха вернётся JSON-объект с данными заказа.
* */OrderService/v1/orders/update* **PRIVATE PATCH** ResponseEntity : обновляет статус заказа. Новый статус и идентификатор заказа содержатся в объекте типа *UpdateOrder*. В случае успешного обновления, вернётся JSON-объект с данными обновлённого заказа.

**3.2.5 API сеансового сервиса SessionService.**

Данный сервис отвечает за аутентификацию и авторизацию. Имеет имя “*SessionService*”. Все url, относящиеся к данному сервису имеют префикс *“/SessionService*”. Он также создаёт новых пользователей и удаляет их из системы. Его API содержится в контроллере *SessionController* и приведён ниже:

* */SessionService/auth/v1/api/users/reg* **PUBLIC POST** TokenObject : создаёт нового пользователя с указанной ролью и возвращает токен, для доступа к приложению. Пользователь регистрируется с помощью формы UI приложения. Данные с формы хранятся в объекте типа *SignUpRequest*. Содержит имя пользователя (*name*), его логин (*username*), адрес электронной почты (*email*), пароль (*password*) и роль пользователя (*role*). Все поля объекта типа *SignUpRequest* имеют тип *String*. Проверка значения поля *role* на имя константы перечисления *RoleName* осуществляется на сервере. Фронтэнд-приложение генерирует правильное имя константы.
* */SessionService/auth/v1/api/users/reg* **PUBLIC POST** OAuthClient : создаёт запись регистрации клиентского приложения. Присваивает ему *client\_id* и *client\_secret*, по которым возможна авторизация по коду. Принимает в качестве параметра объект типа *RegistrationOAuthClientDto*, содержащий имя приложения (*applicationName*), область доступа (*scope*) и адрес редиректа (*redirectUri*). Генерация CLIENT ID и CLIENT SECRET осуществляется с помощью типа UUID.
* */SessionService/auth/v1/api/users/{user\_id}* **ADMIN\_ONLY DELETE** void : удаляет пользователя из системы.
* */SessionService/auth/v1/api/clients/{client\_id}* **ADMIN\_ONLY DELETE** void : удаляет запись о регистрации приложения пользователя. Принимает идентификатор приложения в виде UUID.
* */SessionService/auth/v1/api/oauth?respone\_type=’code’&client\_id=’xxx’&redirect\_uri=’yyy’* **PUBLIC GET** TokenObject : осуществляет авторизацию по коду авторизации. По заданному клиентскому идентификатору находит запись в таблице и получает сущность типа OAuthClient. Если запись не найдена, то вернётся ошибка с кодом 403. С помощью записи, он получает *client\_secret* и *user\_id*. Далее, с помощью *user\_id*, *client\_id* и *redirectUri* создаётся AuthorizationCode. Далее он создаёт словарь с четырьмя свойствами*: client\_id, client\_secret, code,* и *redirect\_uri*. Затем он осуществляет проверку того, что код был успешно создан, и что пользователь с *user\_id* существует. В случае успешной проверки, происходит генерация токена, иначе возвращается ошибка 400.
* */SessionService/auth/v1/api/oauth2/token* **PUBLIC GET** TokenObject : получить токен по логину и паролю пользователя. Пользователь вводит логин и пароль с формы фронтенд-приложения (UI). Фронтэнд приложение добавляет свой полученный CLIENT ID к данным пользователя и отправяет данные к контроллеру. Полученная информация от клиента преобразуется в объект класса *SignInRequest*. Контроллер преобразует данный объект в словарь типа *HashMap<String,String>,* где хранит логин, пароль и client\_id в виде пары: ключ-значение. Ключи и их значения перечислены ниже.
  + username – логин пользователя с формы.
  + password – пароль пользователя с формы.
  + grant\_type – имеет значение в виде строки “password”.
  + client\_id – идентификатор клиентского приложения.
* Далее словарь подаётся на вход процедуре *getTokenByUsernameAndPassword(),* которая осуществляет проверки на null ключей словаря, а также наличие записи о клиентском приложении и пользователя в базе (по ключам *clien\_id* и *username*). Если записи в базе нет, то вернётся ошибка 403. Иначе вернётся токен.

**3.2.6 API сервиса статистики.**

Сервис статистики имеет имя *“StatisticsService”.* К данному сервису имеет доступ только администратор, а также остальные сервисы приложения, которые могут лишь осуществлять запросы на сохранение информации о проводимых операциях пользователями. Действие пользователя представлено сущностью *UserAction* со следующими полями:

* *UUID* – идентификатор сущности
* *userId* – идентификатор пользователя. Если действие совершено неавторизованным пользователем, то поле будет хранить значение -1.
* *username* – имя пользователя.
* *action –* наименование операции. Имена возможных операций перечислены на диаграмме переходов (см. рис. 5).
* *сode –* http-код ответа запроса.
* *path –* url-запроса.

Данные для создания данной сущности представлены типом UserActionCreate, который отличается от UserAction отсутствием поля UUID.

Отметим, что не всякий запрос, сделанный пользователем может быть обработан, в случае отказа сервисов системы. Для хранения истории запросов используется отдельная сущность, представленная классом *UserRequests,* со следующими полями:

* *UUID –* идентификатор записи типа UUID
* *url –* полный url-запроса в виде строки символов.
* *userId –* идентификатор пользователя. Если это неавторизованный пользователь, то в поле запишется значение -1
* *controller -* имя контроллера, обрабатывающий запрос
* *type –* тип HTTP-метода
* *method* – имя метода контроллера, обрабатывающий запрос
* *service* – имя микросервиса, где осуществляется обработка.

Для создания сущности *UserRequests*, был также определён аналогичный класс *UserRequestsCreate*, который отличается лишь отсутствием поля *UUID*.

Весь API сервиса представлен контроллером *StatController*, который имеет следующие *url*:

* ***/****StatisticsService/v1/getActions/{username}*  **ADMIN\_ONLY GET** List<UserAction> : возвращает операции, совершённые пользователем с указанным именем *username* в виде списка JSON-объектов сущностей типа UserAction. Если параметр *username* не указан, или такого пользователя не существует, будет возвращён список действий всех пользователей системы.
* ***/****StatisticsService/v1/getRequests/{service}/{controller}/{method}/{type}/{userId}*  **ADMIN\_ONLY GET** List<UserRequests> : возвращает запросы, совершённые пользователем с указанным идентификатором *userId* в виде списка JSON-объектов сущностей типа UserRequests, которые относятся к указанному сервису с именем *service*, контроллеру с именем *controller*, методу контроллера с именем *method* и типом HTTP-метода *type*. Все параметры являются не обязательными. Если хотя бы один из них пропущен, то считается что выбрано менее строгое условие отбора записей.
* ***/****StatisticsService/v1/createRequest/* **PRIVATE POST** ResponseEntity : принимает JSON-объект, представляющий данные объекта типа *UserRequestsCreate.* Создаёт в базе данных статистики запросов запись о запросе. В случае успеха, возвращает JSON-объект с сообщением об успешной операции.
* ***/****StatisticsService/v1/createAction/* **PRIVATE POST** ResponseEntity : создаёт запись о совершённом пользователем действии. Данные о действии извлекаются из модели типа UserActionCreate. Возвращает сообщение об успешно проведённой операции или сообщение об ошибке.

**3.3 Основная процедура авторизации в системе.**

В системе можно выделить три основных сценария авторизации:

1. Авторизация между сервисами, когда сервис А совершает запрос к сервису Б.
2. Аутентификация с последующей авторизации пользователей системы, через фронтэнд приложение. В данном случае, происходит идентификация роли пользователя.
3. Авторизация сторонних приложений.

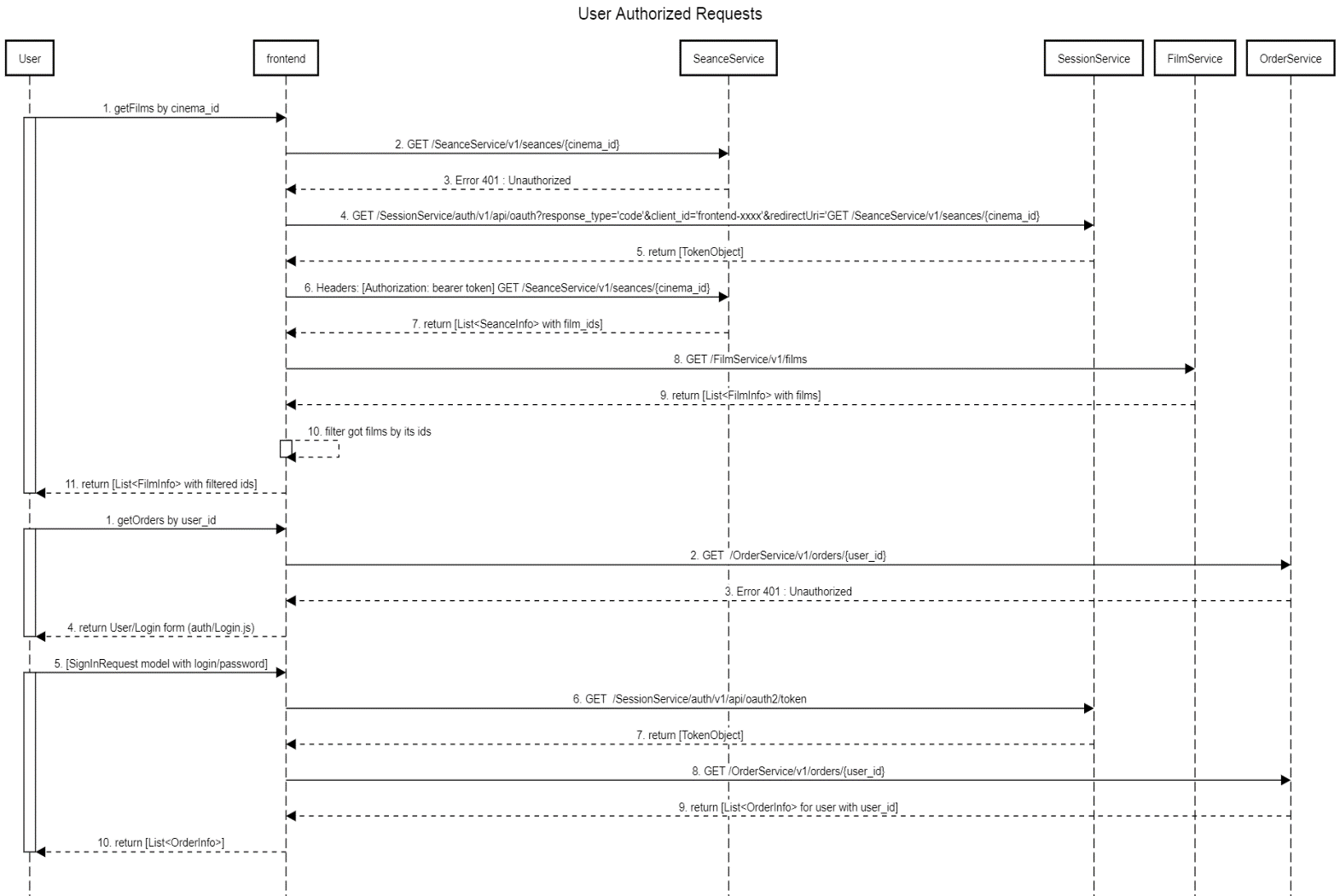
Каждое приложение должно иметь свои учётные данные, представленные в виде строк UUID и называемых соответственно *clientId* и *clientSecret*. Для получения учётных данных приложение должно быть зарегистрировано в системе сервисом авторизации. Регистрация происходит по следующему пути: */SessionService/auth/v1/api/users/reg*. После успешной регистрации каждое приложение может пройти процедуру авторизации по коду. Получая токен по коду, приложение получает доступ к API определённого сервиса. Пользователи системы проходят авторизацию с помощью логина и пароля. Запросы, которые исходят от фронтэнд приложения, и требующие наличие определённой роли пользователя, перенаправляются на сервис авторизации, который возвращает страницу с формой логин/пароль, для авторизации клиентов. Ниже приведены диаграммы последовательности, отображающие авторизацию для зарегистрированных пользователей и клиентских приложений. Авторизация между сервисами системы являются частью процесса обработки от клиента или приложения запроса с авторизацией. Для упрощения, на диаграмме не показана регистрация клиентов и сторонних приложений, считается, что все они уже прошли регистрацию в системе. Также для упрощения, на второй диаграмме не показано обращение сервиса *GatewayApi* к другим сервисам приложения. Можно положить, что *GatewayApi* на второй диаграмме эквивалентен *frontend* на первой.

Рис.8. Диаграмма последовательности запросов пользователя к системе.

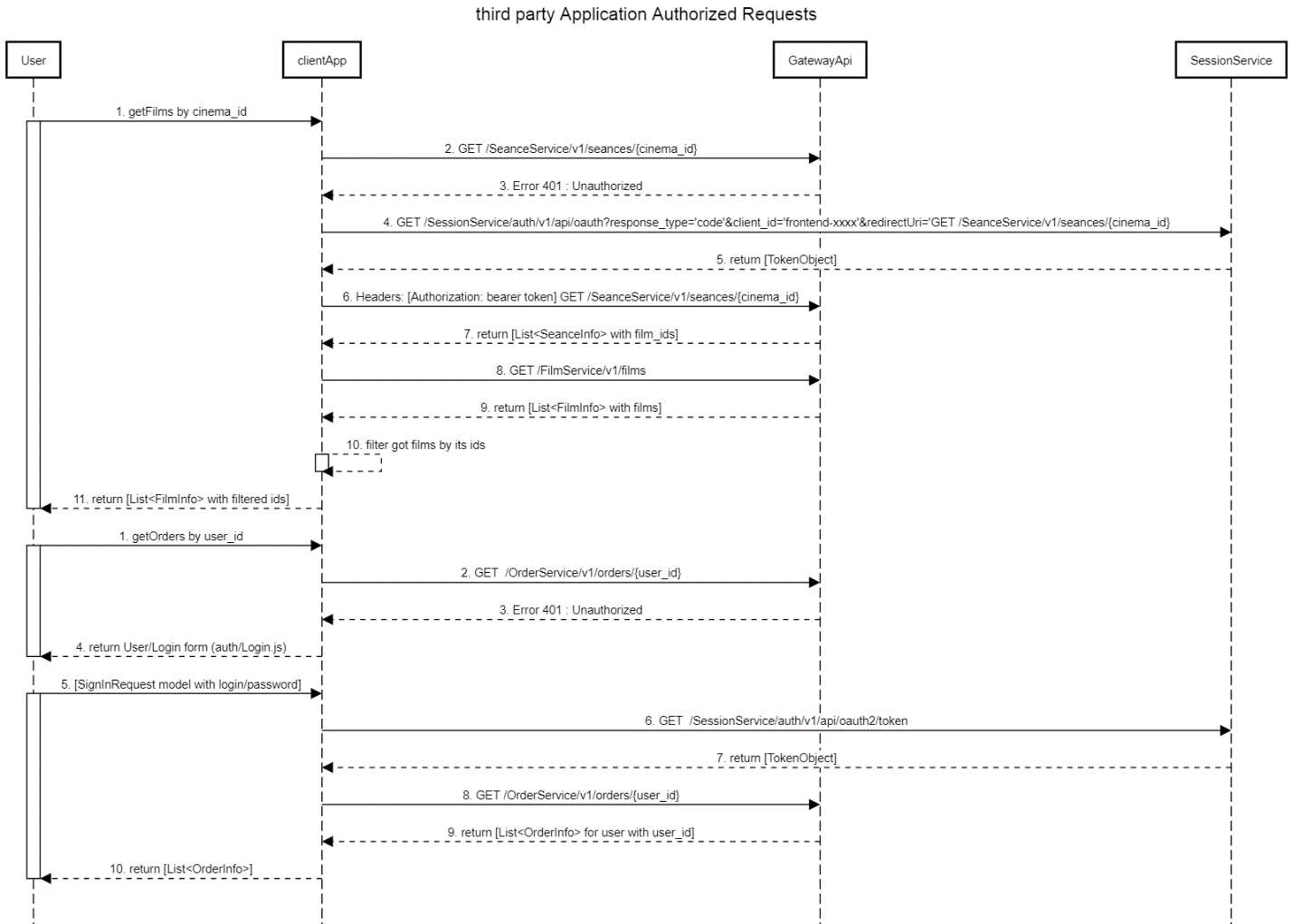
****

Рис. 9. Диаграмма последовательности запросов стороннего приложения, требующего авторизацию.

**3.4 Описание фронтэнд приложения.**

Фронтенд приложение реализовано с помощью фреймворка *ReactJS* последней версии. Оно представляет собой пользовательский интерфейс в браузере клиентов. Построено по шаблону одностраничного приложения. Главным файлом приложения является файл *index.js*. лежащий в каталоге *CinemaStreamMicroservices /WebApp/frontend/src*. В нём подключаются используемые библиотеки, которые определены в файле *package.json.* Он лежит в корневом каталоге фронтэнд приложения (папка *frontend*). Файл *package.json* представляет JSON-объект, имена свойств которого представляют отдельную конфигурацию. Например, сторонние библиотеки определяются как значение свойства с именем *“dependencies”* в виде JSON-объекта, имена свойств которого представляют имена библиотек, а значения – строки, содержащие минимальную версию библиотек. Кроме зависимостей, файл *package.json* определяет также имя приложения (name), его версию (version), список браузеров, на котором следует запускать (browserlist). Содержимое файла package.json дано ниже:

{

"name": "frontend",

"version": "0.1.0",

"private": true,

"dependencies": {

"antd": "^3.26.7",

"bootstrap": "^4.3.1",

"http-proxy-middleware": "^1.0.5",

"jquery": "^3.5.1",

"react": "^16.12.0",

"react-cookie": "^3.0.4",

"react-dom": "^16.12.0",

"react-router-dom": "^4.3.1",

"react-scripts": "^3.4.3",

"reactstrap": "^6.5.0"

},

"scripts": {

"start": "react-scripts start",

"build": "react-scripts build",

"test": "react-scripts test",

"eject": "react-scripts eject"

},

"eslintConfig": {

"extends": "react-app",

"parserOptions": {

"parser": "babel-eslint"

}

},

"browserslist": {

"production": [

">0.2%",

"not dead",

"not op\_mini all"

],

"development": [

"last 1 chrome version",

"last 1 firefox version",

"last 1 safari version"

]

}

}

Листинг 10. Конфигурация фронтэнд приложения. Файл package.json

В файле *index.js* вызывается функция *render* модуля *ReactDOM*, которая берёт определённый html-элемент из файла *index.html*, и добавляет кусок html-разметки вместе с JavaScript кодом к данному элементу. Файл *index.html*, представляет из себя саму веб-страницу приложения, на которой лежат все графические элементы UI, а также ссылки на другие компоненты. Файл *index.html* лежит в директории *CinemaStreamMicroservices /WebApp/frontend/public .* Содержимое файла *index.js* дано ниже:

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import 'bootstrap/dist/css/bootstrap.min.css';

import App from './App';

import \* as serviceWorker from './serviceWorker';

import { BrowserRouter, Route } from 'react-router-dom';

ReactDOM.render(<BrowserRouter>

<div>

<Route path="/" component={App} />

</div>

</BrowserRouter>, document.getElementById('root'));

// If you want your app to work offline and load faster, you can change

// unregister() to register() below. Note this comes with some pitfalls.

// Learn more about service workers: https://bit.ly/CRA-PWA

serviceWorker.unregister();

Листинг 11. Файл index.js.

В вышеприведённом файле, к html-элементу с атрибутом *id* равным   
“root” прикрепляется html-элемент div, содержащий корневой маршрут приложения. По данному маршруту будет возвращён компонент с именем App. С помощью компонента *BrowserRouter* подключается маршрутизация фронтэнд приложения. Фреймворк ReactJS определяет базовый набор компонентов, для отображения графических элементов (таких, как кнопки, текстовые поля, линии, и т. д.) а также для связки компонентов друг с другом, построение маршрутизации и т.п. Компоненты можно использовать в разметке как обычные html-элементы. Фронтэнд приложение задаёт свою собственную маршрутизацию со своими маршрутами, отличными от url определённых в параграфе 3.2 настоящей работы. В данном случае, маршрутизация на клиенте определяет, как пользователь с помощью url получает доступ к компонентам и секциям UI. Все маршруты приложения определены в компоненте App. Каждый компонент определён в файле, имеющем следующее имя: *имя\_компонента.js* . Т.е. компонент App определён в файле App.js. Все компоненты лежат в папке *src*. Ниже приведен код компонента App.

import React, { Component } from 'react';

import logo from './logo.svg';

import './App.css';

import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';

import Home from './Home';

import GenList from './GenList';

import FilmList from './FilmList';

import FilmEdit from './FilmEdit';

import GenreList from './GenreList';

import GenreEdit from './GenreEdit';

import CinemaList from './CinemaList';

import CinemaEdit from './CinemaEdit';

import SeanceCinemaList from './SeanceCinemaList';

import SeanceInformation from './SeanceInformation';

import SeanceList from './SeanceList';

import SeanceEdit from './SeanceEdit';

import Login from './auth/Login';

import Signup from './auth/Signup';

const ReactDOM = require('react-dom');

class App extends Component {

render(){

return (

<Router>

<Switch>

<Route path='/' exact={true} component={Home}/>

<Route path='/views/films' exact={true} render={(props) => (<GenList {...props} path={"/FilmService/v1/films/"} entity={"film"}/>)} />

<Route path='/views/genres' exact={true} component={GenreList} />

<Route path='/views/seances' exact={true} component={SeanceList}/>

<Route path='/views/cinemas' exact={true} render={(props) => (<GenList {...props} path={"/CinemaService/v1/cinemas/"} entity={"cinema"}/>)}/>

<Route path='/views/cinemas/:id' exact={true} component={CinemaEdit}/>

<Route path='/views/genres/new' exact={true} component={GenreEdit}/>

<Route path='/views/films/:id' component={FilmEdit} />

<Route path='/views/cinemas/:id/seances' exact={true} component={SeanceCinemaList} />

<Route path='/views/seances/:cid/:fid' exact={true} component={SeanceInformation} />

<Route path='/views/cinemas/:id/seances/add' exact={true} component={SeanceEdit} />

<Route path='/views/login' exact={true} component={Login} />

<Route path='/views/signup' exact={true} component={Signup} />

</Switch>

</Router>

);

}

}

export default App;

Листинг 12. Компонент App.

Для отображения html-содержимого определённого компонента используется функция *render().* В данном компоненте используется специальный компонент Router, который обрабатывает текущий *url,* извлекает из него *path*, и проверяет его на соответствие значения атрибута *path* компонента Route. В случае соответствия, будет вызвана функция *render()* того компонента, имя которого указано в атрибуте *component*. Если же данный атрибут не определён, то вызывается функция, имя которой определено в атрибуте *render*. Атрибут *render* компонента Route принимает объект функции, которая содержит в качестве параметра поля (свойства) *props* текущего родительского компонента, а возвращает react-компонент, у которого автоматически вызывается функция *render()*. В компоненте App определены анонимные функции, которые передаются в качестве значения атрибута *render*. Они возвращают определённые компоненты с указанными свойствами, которые были написаны вручную. Компоненты фронтэнд-приложения, определяющие UI для пользователя системы были написаны в соответствии с диаграммой переходов, которая была проиллюстрирована на рис.5. Ниже дана диаграмма переходов компонентов, определяющая структуру UI фронтэнда системы.

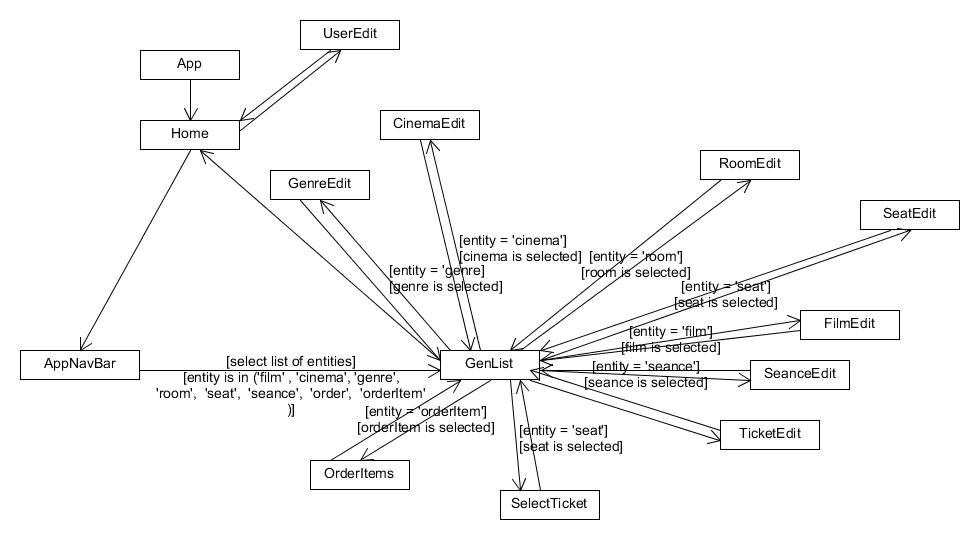


Рис. 13. Основные компоненты пользовательского интерфейса фронтенд приложения.

Стрелки на вышеприведённой диаграмме, отображают переход от одного компонента к другому. Условия перехода указаны на стрелках диаграммы. Для перехода к компонентам с суффиксом Edit, необходимо нажать на одноимённую кнопку, которая расположена в правом углу элемента списка компонента GenList. Компоненты с суффиксом Edit используются для создания и редактирования сущностей системы.

Для списка элементов указанной сущности (поле *entity*) используется компонент GenList. Он имеет два специальных поля:

* *path* – это поле содержит в качестве значения строку, определяющую путь к получаемым данным.
* *entity –*это поле определяет имя сущности, которая содержится в списке. Список содержит сущности одного типа.

Согласно диаграмме переходов из рис.5, для бронирования места в кинотеатре клиенту, необходимо выбрать сначала кинотеатр, затем выбрать нужный сеанс из списка сеансов выбранного кинотеатра, затем выбрать билет, из списка мест указанного сеанса. При выборе места, произойдет переход в компонент SelectTicket, с итоговой информацией о заказе. Компонент имеет две кнопки *Submit* и *Back*, которые осуществляют выбор места или возврат в список мест. Выбрав место, пользователь резервирует данное место, но ещё не оформляет заказ. Для оформления заказа, необходимо нажать на кнопку MakeOrder, которая находится сверху, над списком мест. Данную кнопку можно нажать, только выбрав одно или более место. Если никакого места не было выбрано, то данная кнопка будет недоступна.

Каждый запрос от пользователя, имеющий path соответствующий маршруту из компонента App проксируется на имя хоста, порта и фактического пути контроллера микросервиса. Настройки для проксировки хранятся в файле *setupProxy.js* каталога *src*. Содержимое файла дано ниже:

const { createProxyMiddleware } = require('http-proxy-middleware');

module.exports = function(app) {

app.use(

'/FilmService',

createProxyMiddleware({

target: 'http://localhost:3333',

changeOrigin: true,

})

);

app.use(

'/CinemaService',

createProxyMiddleware({

target: 'http://localhost:2222',

changeOrigin: true,

})

);

app.use(

'/SeanceService',

createProxyMiddleware({

target: 'http://localhost:1111',

changeOrigin: true,

})

);

app.use(

'/OrderService',

createProxyMiddleware({

target: 'http://localhost:4444',

changeOrigin: true,

})

);

};

Листинг 14. Файл setupProxy.js.

**3.5 Запуск и развёртывание приложения**

Для запуска системы необходимо сначала запустить каждое приложение микросервиса, корневые папки которых хранятся в директории *Microservices*. Первым запускается микросервис, хранящий статистику работы системы. Для запуска микросервисов, надо зайти в каждый корневой каталог данных приложений открыть консоль, задать текущую директорию равной папке проекта, и выполнить команду: “*gradlew bootRun*”. Эта команда запустит сборку и запуск приложения микросервиса. Порт и настройки текущего профиля, а также подключения к базам данных хранятся в файлах */{Project\_DIR}/src/main/resources/application.properties* и */{Project\_DIR}/src/main/resources/application-local.properties*.

Запустив все микросервисы из папки Microservices, необходимо перейти в каталог WebApp и выполнить ту же команду: “gradlew bootRun”. В данном каталоге, команда осуществит сборку и запуск как gateway-сервиса, так и фронтэнд приложения.

Весь исходный код проекта выложен на *Github*, и доступен по адресу: <https://github.com/Osipo/CStreamMicroServicesRSOI2/tree/Cproject>

**Список использованных источников.**

1. *Standard RFC 8259: The JavaScript Object Notation (JSON) Data Interchange Format. URL:* [*https://tools.ietf.org/html/rfc8259*](https://tools.ietf.org/html/rfc8259)
2. *Standard ECMA 404: The JSON Data Interchange Standard.* URL: <https://www.json.org/json-en.html>,<https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>
3. *Equals and HashCode.* URL: <https://developer.jboss.org/docs/DOC-13933>
4. Ultimate guide to implementing equals() and hashCode() with Hibernate. URL: <https://thorben-janssen.com/ultimate-guide-to-implementing-equals-and-hashcode-with-hibernate/#:~:text=That%20means%20that%20no%20two,the%20context%20of%20one%20Session>

5. Java Platform, Enterprise Edition (Java EE) 7 Documentation. URL: <https://docs.oracle.com/javaee/7/index.html>

6. Java JDK 12 Documentation. URL: [https://docs.oracle.com/ en/java/ javase/12/](https://docs.oracle.com/%20en/java/%20javase/12/)

7. Gradle documentation. URL: <https://docs.gradle.org/5.2.1/userguide/userguide.html>

8. Паттерн: Data Transfer Object (Объект передачи данных). URL: <http://design-pattern.ru/patterns/data-transfer-object.html>