Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет инженерно-экономический

Кафедра экономической информатики

Дисциплина: Программирование сетевых приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

на тему

**РАЗРАБОТКА АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ РАБОТЫ КИНОТЕАТРА**

БГУИР КП 1-40 01 02-02 021 ПЗ

|  |  |  |
| --- | --- | --- |
| Студент |  | Степовой В.К.  гр.672302 |
| Руководитель |  | Петрович Н.О. |

Минск 2018

Оглавление

[Введение 3](#_Toc532384152)

[1 Описание работы кинотеатра 4](#_Toc532384153)

[2 Постановка задачи и обзор методов её решения 7](#_Toc532384154)

[2.1 Постановка задачи 7](#_Toc532384155)

[2.2 Обзор методов решения поставленной задачи 7](#_Toc532384156)

[3 Функциональное моделирование на основе стандарта IDEF0 11](#_Toc532384157)

[4 Информационная модель системы и её описание 13](#_Toc532384158)

[5 Модели представления системы и их описание 15](#_Toc532384159)

[5.1 Диаграмма классов 15](#_Toc532384160)

[5.2 Диаграмма вариантов использования 17](#_Toc532384161)

[5.3 Диаграмма состояний 18](#_Toc532384162)

[5.4 Диаграмма последовательности 19](#_Toc532384163)

[5.5 Диаграмма компонентов 20](#_Toc532384164)

[5.6 Диаграмма развертывания 21](#_Toc532384165)

[6 Описание алгоритмов, реализующих бизнес-логику работы системы работы кинотеатра 22](#_Toc532384166)

[6.2 Описание алгоритма покупки билетов на киносеанс 22](#_Toc532384167)

[6.1 Описание алгоритма авторизации пользователя 22](#_Toc532384168)

[7 Руководствопользователя 23](#_Toc532384169)

[8 Результаты тестирования системы 33](#_Toc532384170)

[Заключение 34](#_Toc532384171)

[Список использованных источников 35](#_Toc532384172)

[ПРИЛОЖЕНИЕ А 36](#_Toc532384173)

[ПРИЛОЖЕНИЕ Б 44](#_Toc532384174)

[ПРИЛОЖЕНИЕ В 46](#_Toc532384175)

[ПРИЛОЖЕНИЕ Г 57](#_Toc532384176)

# ВВЕДЕНИЕ

С каждым годом, месяцем и даже днём информационные технологии становятся всё более неотъемлемой частью нашей жизни. С их развитием наша жизнь становится всё более мобильной, и теперь практически невозможно представить отрасль человеческой деятельности, которая бы ими не пользовалась.

Процесс автоматизации кинотеатра заключается в разработке и внедрении программных продуктов для продажи и автоматического учета билетов с учетом разных типов мест, программ поддержки постоянных клиентов, системы скидок и других рекламных акций. Процесс автоматизации неразрывно связан с обновлением не только программного обеспечения, но и с обновлением, покупкой нового оборудования и затрат на его внедрение и обслуживание.

Благодаря автоматизации становится удобным планирование и бюджетирование деятельности предприятия сферы досуга. Такая система позволяет формировать прогнозные балансы, репертуарные планы, прогнозировать количество посетителей с учетом различных коэффициентов

*Целью данного курсового проекта* является автоматизация процесса покупки билетов на различные киносеансы посредством разработки системы учёта продажи билетов на языке Java, позволяющей контролировать учёт киносеансов, стандартизировать информацию о сотрудниках и клиентах системы и централизовать покупку билетов на различные киносеансы.

Для реализации данной цели были поставлены *следующие задачи*:

* ознакомиться с уже существующими системами продажи билетов;
* разработать функциональную модель основного процесса;
* спроектировать основные алгоритмы и построить по ним блок-схемы;
* разработать систему для автоматизации происходящих в данной предметной области процессов с использованием объектно-ориентированного подхода;
* протестировать программное средство на предмет ошибок и выполнить отладку.

Предмет исследования – проектирование информационных технологий управления автоматизированной системой продажи билетов.

Объекты исследования – кинотеатры и процессы продажи билетов.

# 1 ОПИСАНИЕ РАБОТЫ КИНОТЕАТРА

В современном мире люди уделяют достаточно много внимания развлечениям. Походы в торговые центры, кафе, кинотеатры – все это важная часть жизни каждого, кто хочет хорошо провести свое свободное время. Поэтому, исходя из проанализированной мной ранее информации, я выбрал кинематограф и способы взаимодействия с ним основными объектами своего курсового проекта.

Кинематограф – это неотъемлемая часть культуры и человеческого развития. Первый публичный показ короткометражного фильма состоялся 22 марта 1895 года, после чего «новинка» обошла весь свет, и теперь каждый желающий обладает возможностью самостоятельно произвести на свет свою идею, преобразив ее в кинофильм. Одни ленты сняты на основе реальных событий, другие – по сюжету известных книжных произведений, третьи – плод чьей-то невообразимой фантазии. По данным, предоставленным крупнейшей базой данных по кинематографу IMDb, всего в мире было отснято около 2,6 миллионов кинолент, включающих в себя отдельные эпизоды сериалов, мультфильмы и короткометражки.

Но как же мы узнаем о новинках кино и сможем посмотреть их в рядах самых первых? Конечно же, как и раньше, мы можем воспользоваться услугами специальных заведений под названием кинотеатры. В двадцатом веке они были редкостью для населения, а в современном мире в одном только Минске насчитывается около 30 специальных заведений, не считая частные учреждения.

Личный опыт подсказывает, что в основном хорошие кинотеатры находятся по близости к окраинам столицы, например, на самых дальних станциях метро, путь куда занимает достаточно много времени. Одним из таких достойных кинотеатров, на мой взгляд, является сеть Silver Screen.

Данный вариант от других отличается модернизированными кинозалами; невероятным разнообразием закусок в специализированной зоне; молодым, энергичным и, самое главное, приветливым и позитивным персоналом, а также достаточно высокими ценами. Но вне зависимости от последнего критерия, атмосфера и обслуживание заставляет посетителей возвращаться туда снова и снова. Сеть предоставляет гостям возможность посетить тематические мероприятия по случаю выхода самых культовых фильмов, присутствовать на предпросмотрах, участвовать в конкурсах и приобретать сувениры с логотипами, изображениями или другой символикой любимого кино.

Однако проблемой любого кинотеатра является лишь труднодоступность. В случае, когда человек не может использовать всемирную сеть, ему приходится преодолевать значительное расстояние от своего дома до кинотеатра для приобретения билетов, а потом обратно. Если он обладает собственным средством передвижения, это дает ему преимущества, ведь автомобиль – самый удобный и быстрый вариант. Однако дети, подростки, люди пожилого возраста, не имеющие личного транспорта, часто пользуются городским, что забирает у них слишком много времени.

Ежедневно более 6 тысяч человек совершают покупку билетов в кино в нашей столице. Трудно вместить такое большое количество людей в одно помещение. Это замедляет работу кинотеатров и отнимает много времени как у персонала, так и у посетителей. Именно поэтому считаю нужным автоматизировать данный процесс.

Двадцать первый век считается веком технологий. Сейчас самым популярным средством общения, поиска информации, работы и даже учебы является сеть Интернет. Его используют миллионы людей по всему миру и это в какой-то степени делает их зависимыми. Сложно вообразить современный мир, лишенных уже существующих технологий. Так почему-то не воспользоваться ими во благо обществу?

Покупка билетов онлайн – единственный способ избежать бессмысленных поездок по городу и траты собственного времени в бесконечных очередях. Что для этого нужно? Иметь смартфон, компьютер или другой более удобный для Вас гаджет с наличием интернета, а также банковская карта. И это просто, ведь современный человек всегда имеет доступ к данным средствам.

Следующим очень важным критерием, способствующим покупке билетов в кино онлайн, является сайт. Это может быть сайт как отдельного кинотеатра, так и обобщающий информацию о заведении и новинках кино.

Хороший интернет-сайт всегда удобен в использовании. Интерфейс должен быть прост, чтобы не только специально обученный или осведомленный в этой области смог совершить заказ, но и абсолютно ничего не смыслящий в этом человек. Здесь должна присутствовать информация о каждом кинотеатре: его название, местоположение, вместимость, транспорт, на котором Вы сможете с легкостью добраться до нужного места, а также иллюстрируемые фильмы и время сеансов.

Рассмотрения такого варианта, как автоматизация процесса покупки билетов в кино, считаю необходимым по некоторому ряду причин:

* Данный подход к осуществлению покупок значительно экономит время пользователя.
* С помощью такой функции нет надобности выхода из дома при неблагоприятных погодных условиях или при плохом самочувствии.
* Один клик – и Вы совершаете покупку билета/билетов на нужный Вам сеанс и фильм.
* Вы сможете всегда быть в курсе «новинок» кино и имеете возможность просмотреть их среди первых.
* Если Вы уже купили билеты, но не имеете возможности посетить сеанс, то с легкостью сможете отменить бронь на билеты и вернуть потраченные деньги.
* Здесь Вы сможете иметь личный кабинет. При регистрации Вы получаете некоторую сумму накопительных бонусных баллов.
* При покупке билетов онлайн Вы сможете накапливать бонусные баллы, которые, достигнув необходимого лимита, можно будет обменять на карту VIP-гостя. Она предоставит Вам скидку в размере 10% на сеансы в обычные дни, 15% в дни государственных праздников, 20% - в День Вашего рождения.

# 2 ПОСТАНОВКА ЗАДАЧИ И ОБЗОР МЕТОДОВ ЕЁ РЕШЕНИЯ

## **2.1 Постановка задачи**

Главная задача информационной системы состоит в централизации продаж билетов на киносеансы. Для достижения этой цели разрабатывается настольное приложение, которое можно развернуть на любом компьютере. Данное приложение позволяет клиентам (пользователям в рамках системы) просматривать информацию об актуальных киносеансах в нашей стране и мире, а также предоставляет возможность покупать билеты на конкретные киносеансы, вдобавок к возможностям системы, доступ к приложению предоставляется сотрудникам кинотеатра (модераторы в рамках системы) с возможностью добавлять, удалять, редактировать киносеансы. Программное приложение состоит из клиентской и серверной части.

Серверное приложение является консольным, где производится журналирование (логгирование) всех запросов от клиентской части. Для обеспечения возможности одновременной работы с приложением у нескольких пользователей, используется многопоточная реализация.

В клиентском приложении разработан и реализован графический пользовательский интерфейс (GUI) для более удобного управления программой. Также предусмотрено разделение ролей: пользователь, менеджер и администратор. Функционал ролей будет различаться в связи с различным уровнем допуска для управления системой.

## **2.2 Обзор методов решения поставленной задачи**

Объектно-ориентированное программирование (ООП) — это метод программирования, при использовании которого главными элементами программ являются объекты. В языках программирования понятие объекта реализовано как совокупность свойств (структур данных, характерных для данного объекта), методов их обработки (подпрограмм изменения их свойств) и событий, на которые данный объект может реагировать и, которые приводят, как правило, к изменению свойств объекта. Объединение данных и свойственных им процедур обработки в одном объекте, называется инкапсуляцией и является одним из важнейших принципов ООП.

На сегодняшний момент язык Java является одним из самых распространенных и популярных языков программирования. Первая версия языка появилась еще в 1996 году в недрах компании Sun Microsystems, впоследствии поглощенной компанией Oracle. Java задумывался как универсальный язык программирования, который можно применять для различного рода задач. Текущей версией является Java 11, которая вышла в сентябре 2018 года. А Java превратилась из просто универсального языка в целую платформу и экосистему, которая объединяет различные технологии, используемые в целом ряде задач: от создания десктопных приложений до написания крупных веб-порталов и сервисов. Кроме того, язык Java активно применяется для создания программного обеспечения для целого ряда устройств: обычных ПК, планшетов, смартфонов и мобильных телефонов и даже бытовой техники. Достаточно вспомнить популярность мобильной ОС Android, большинство программ для которой пишутся именно на Java.

Клиентская часть приложения была написана с использованием графической библиотеки классов Swing, библиотека для создания графического интерфейса для программ на языке Java. Swing был разработан компанией Sun Microsystems. Он содержит ряд графических компонентов, таких как кнопки, поля ввода, таблицы и т. д. Для удобной работы с данными была использована база данных MySQL. Связь между сущностями базы данных и моделью приложения обеспечена паттернами проектирования Factory и DAO, а также был использован паттерн Singleton для единоразового создания объекта подключения к базе данных.

 Использующие DAO бизнес-компоненты работают с более простым интерфейсом, предоставляемым объектом DAO своим клиентам. DAO полностью скрывает детали реализации источника данных от клиентов. Поскольку при изменениях реализации источника данных представляемый DAO интерфейс не изменяется, этот паттерн дает возможность DAO принимать различные схемы хранилищ без влияния на клиента или бизнес-компоненты.

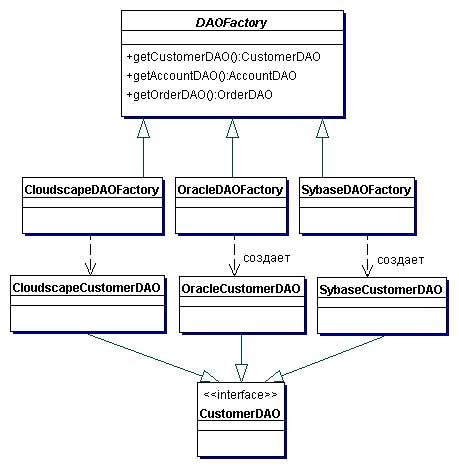
****

Рисунок 2.2 – UML диаграмма паттерна DAO

По существу, DAO выполняет функцию адаптера между компонентом и источником данных.

Для работы с DAO объектами был применён паттерн Factory. Данный шаблон проектирования предоставляет интерфейс для создания экземпляров некоторого класса. В момент создания наследники могут определить, какой класс использовать.

Иными словами, Фабрика делегирует создание объектов наследникам родительского класса. Это позволяет использовать в коде программы не специфические классы, а манипулировать абстрактными объектами на более высоком уровне.

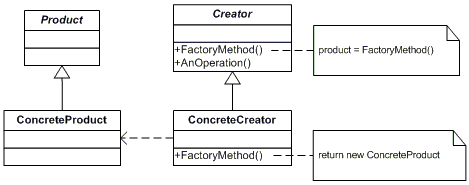


Рисунок 2.2 – UML диаграмма классов паттерна Factory

Одиночка (Singleton) - порождающий паттерн, который гарантирует, что для определенного класса будет создан только один объект, а также предоставит к этому объекту точку доступа.

Этот паттерн необходимо использовать для того, чтобы для класса существовал только один экземпляр

Singleton позволяет создать объект только при его необходимости. Если объект не нужен, то он не будет создан. В этом отличие Singleton от глобальных переменных.

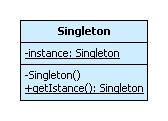


Рисунок 2.3 – UML диаграмма классов паттерна Singleton

# 3 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ НА ОСНОВЕ СТАНДАРТА IDEF0

Для получения более полного представления о том, как можно автоматизировать учёт продажи билетов на киносеансы создана функциональная модель.

На контекстной диаграмме верхнего уровня представлена функциональная модель «Автоматизировать работу кинотеатра» (рисунок А.1, приложение А), в которой определены потоки входных и выходных данных, а также механизмы управления и ограничения.

Входящими параметрами модели являются информация о киносеансах, база данных кинотеатра. Потоки управления модели представлены законодательством, правами потребителя, требованиями к заведениям развлекательного характера и правилами пользования системой. Потоки механизмов модели представлены программным обеспечением, клиентом и персоналом. Выходными параметрами модели являются проданные билеты, прибыль, обслуженные клиенты, отчёт о продажах.

Декомпозиция процесса автоматизации работы кинотеатра (рисунок А.2, приложение А) состоит из следующих подпроцессов:

* проверить актуальность информации о киносеансах;
* управлять продажами билетов;
* осуществить деятельность бухгалтерского отдела;
* проанализировать статистику продаж;
* сформировать отчёт о продажах.

Декомпозиция процесса проверки актуальности информации о киносеансах (рисунок А.3, Приложение А) состоит из следующих подпроцессов:

* запросить список актуальных киносеансов;
* получить актуальную информацию о киносеансах;
* предоставить актуальную информацию пользователям;

Декомпозиция процесса управления продажами (рисунок А.4, приложение А) состоит из следующих подпроцессов:

* найти свободные места;
* выбрать подходящее по расположению место;
* продать билеты на киносеанс;

Декомпозиция осуществление деятельности бухгалтерского отдела (рисунок А.5, приложение А) состоит из следующих подпроцессов:

* рассчитать налог на прибыль;
* проанализировать данные о продажах;
* составить статистику о продажах;

Декомпозиция процесса проанализировать данные о продажах;

(рисунок А.6, приложение А) состоит из следующих подпроцессов:

* составить статистику заполненности зала;
* сравнить количество продаж;

Декомпозиция процесса проанализировать статистику продаж (рисунок А.7, приложение А) состоит из следующих подпроцессов:

* просмотреть информацию о продажах;
* сравнить с планом продаж;
* сформировать предложения об изменении плана продаж;

Декомпозиция процесса формирования отчёта о продажах (рисунок А.7, приложение А) состоит из следующих подпроцессов:

* получить результаты статистического анализа;
* оформить данные в соответствии с уставом предприятия;
* сформировать отчёт в графическом виде;

# 4 ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ И ЕЁ ОПИСАНИЕ

Все данные системы хранятся на сервере базы данных MySQL. Там хранятся все данные о пользователях, кинозалах, киносеансах и фильмах. Модель базы данных приведена на рисунке 4.1.

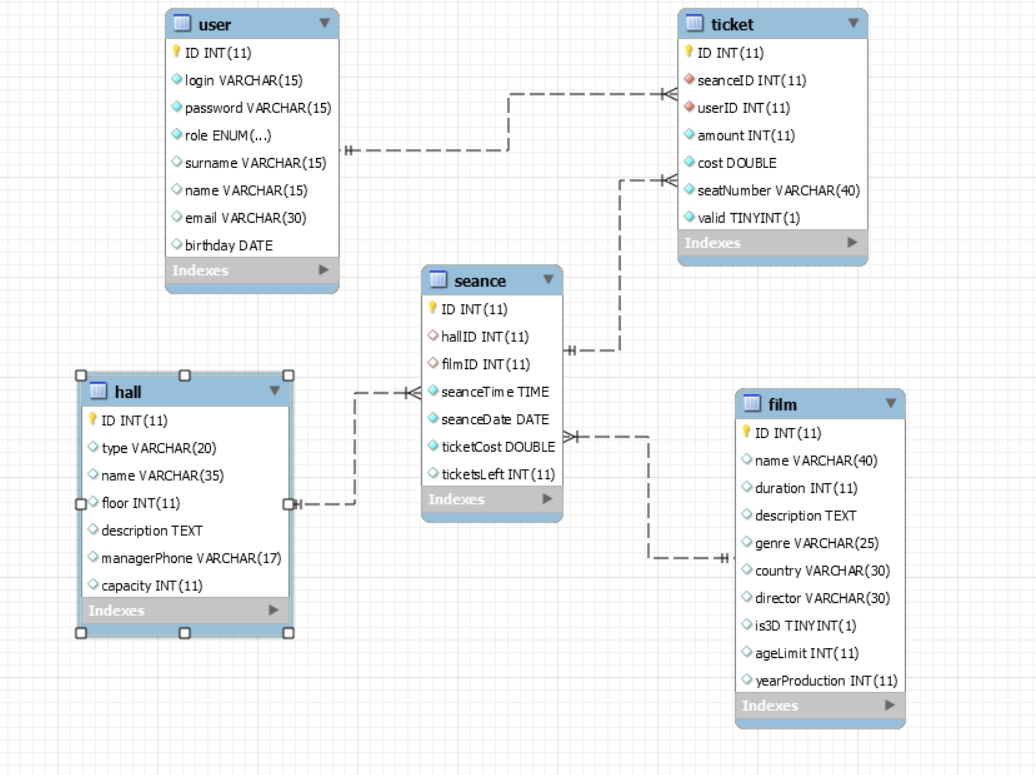


Рисунок 4.1 – ER-диаграмма базы данных “cinema”

База данных «cinema» состоит из 5 таблиц.

Таблица user служит для хранения информации о пользователях:

* ID – уникальный идентификатор пользователя;
* login – логин аккаунта пользователя;
* password – пароль от аккаунта пользователя;
* role – роль пользователя в системе (юзер, модератор или админ);
* surname – фамилия пользователя;
* name – имя пользователя;
* email – адрес электронной почты пользователя;
* birthday – дата рождения пользователя.

Таблицы hall служит для хранения информации о кинозалах кинотеатра:

* ID – уникальный идентификатор кинозала;
* type – тип кинозала;
* name – название кинозала;
* floor – этаж, на котором располагается зал;
* description – дополнительное описание кинозала;
* managerPhone – номер телефона менеджера кинозала;
* capacity – количество посадочных мест в кинозале;

Таблица film служит для хранения информация о кинофильмах:

* ID – уникальный идентификатор кинофильма;
* name – название кинофильма;
* description – описание фильма;
* genre – жанр фильма;
* country – страна, которая выпустила фильм;
* director – режиссёр фильма;
* is3D – показывают ли фильм в 3D;
* ageLimit – возрастное ограничение;
* yearProduction – год выхода фильма;

Таблица séance служит для хранения информации о киносеансах:

* ID – уникальный идентификатор киносеанса;
* hallID – идентификатор кинозала, в котором проходит киносеанс;
* filmID – идентификатор кинофильма, который показывают на данном киносеансе;
* seanceTime – время, в которое начинается кинопоказ;
* seanceDate – дата, в которую проводится кинопоказ;
* ticketCost – стоимость билетов на данных киносеанс;
* ticketsLeft – количество не распроданных билетов на кинопоказ;

Таблица ticket служит для хранения информации о билетах на киносеансы:

* ID – уникальный идентификатор билетов на кинопоказ;
* seanceID – идентификатор киносеанса, на который куплен билет;
* userID – идентификатор пользователя, который купил билет;
* amount – количество купленных билетов;
* cost – общая стоимость билетов;
* seatNumber – номер мест, на которые куплены билеты;
* valid – идентификатор действительности билетов;

# 5 МОДЕЛИ ПРЕДСТАВЛЕНИЯ СИСТЕМЫ И ИХ ОПИСАНИЕ

## **5.1 Диаграмма классов**

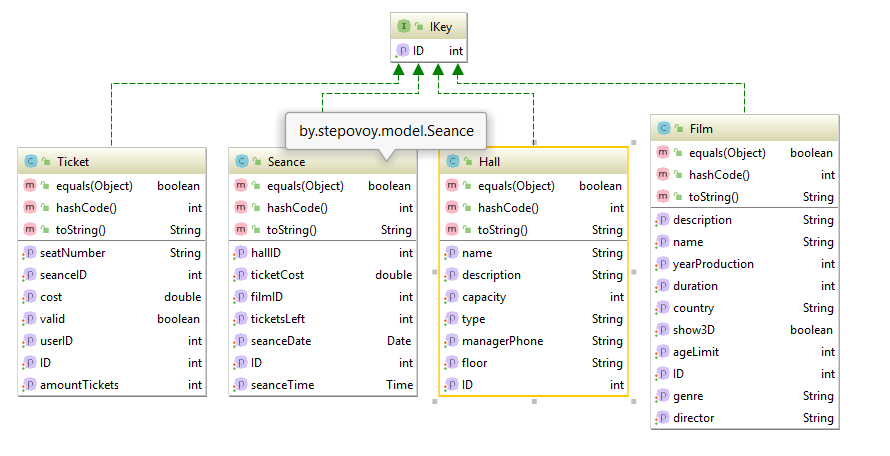


Рисунок 5.1 – Диаграмма классов предметной области

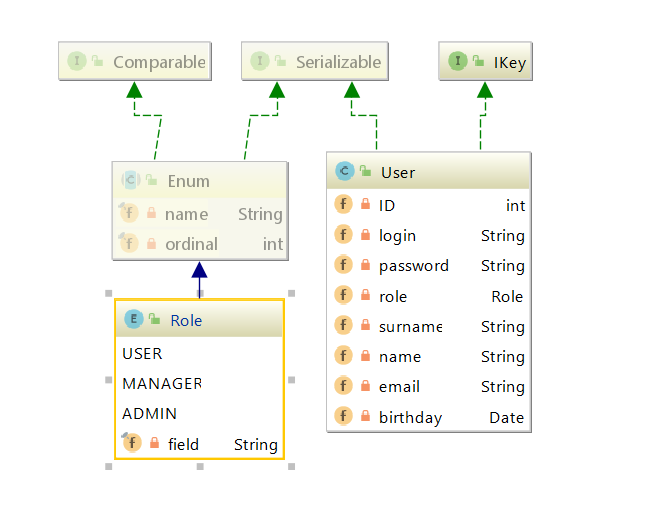


Рисунок 5.1 – Диаграмма классов, для управления пользователями

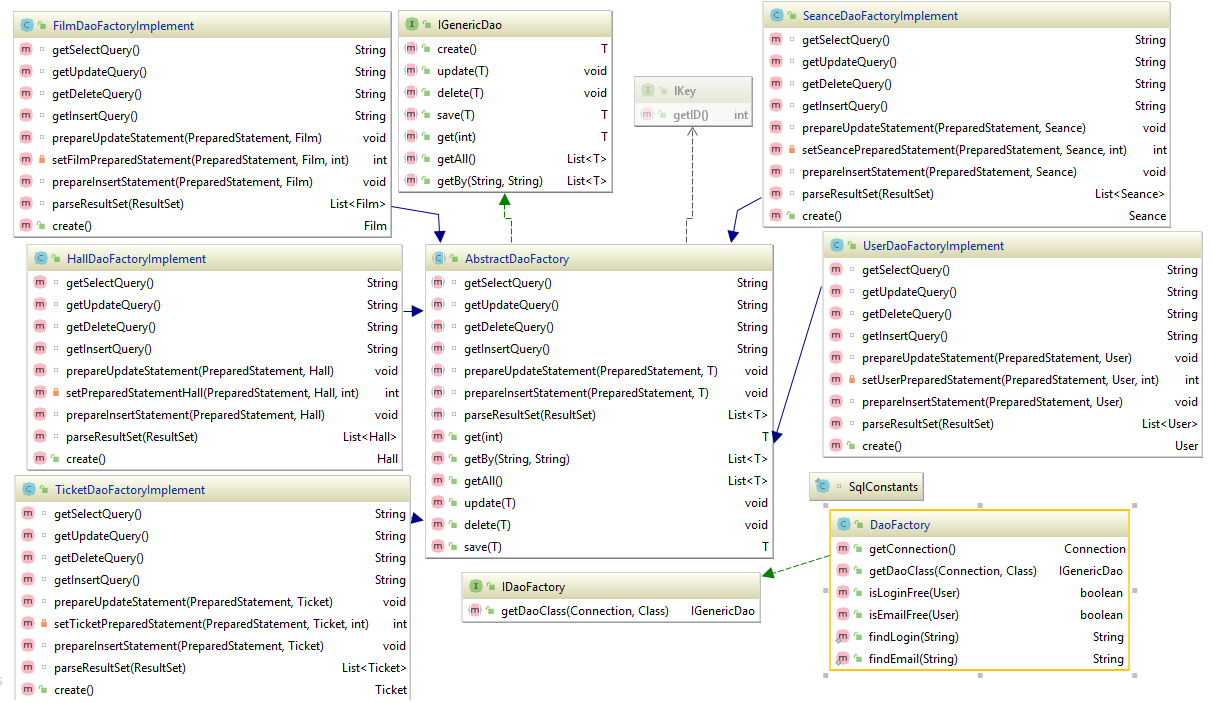
****

Рисунок 5.2 – Диаграмма классов, реализующих паттерн DAO

## **5.2 Диаграмма вариантов использования**

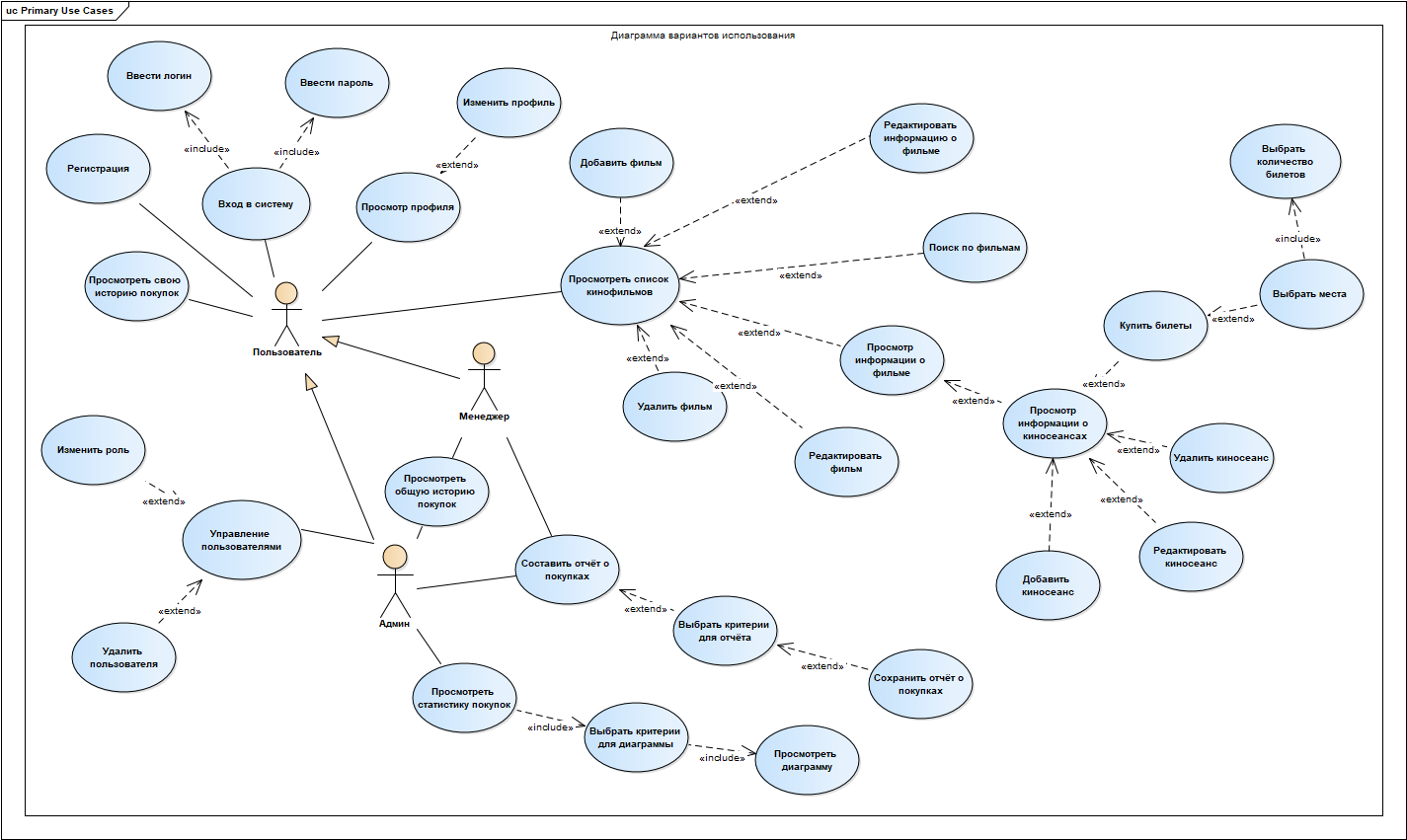


Рисунок 5.3 – Диаграмма вариантов использования приложения

## **5.3 Диаграмма состояний**

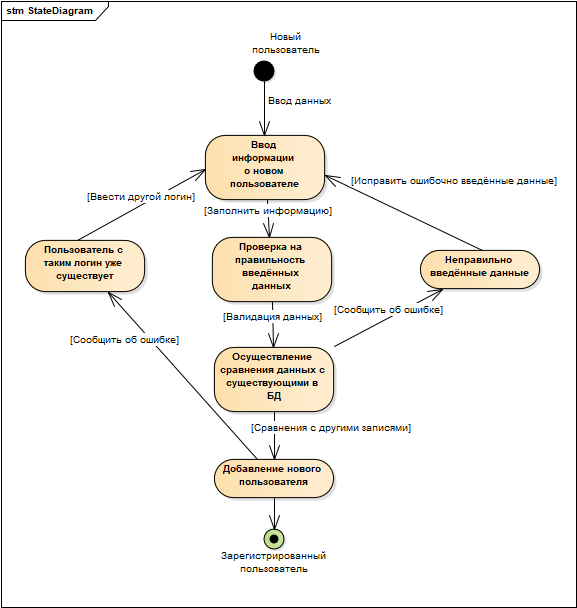


Рисунок 5.4 – Диаграмма состояний процесса регистрации нового пользователя

## **5.4 Диаграмма последовательности**

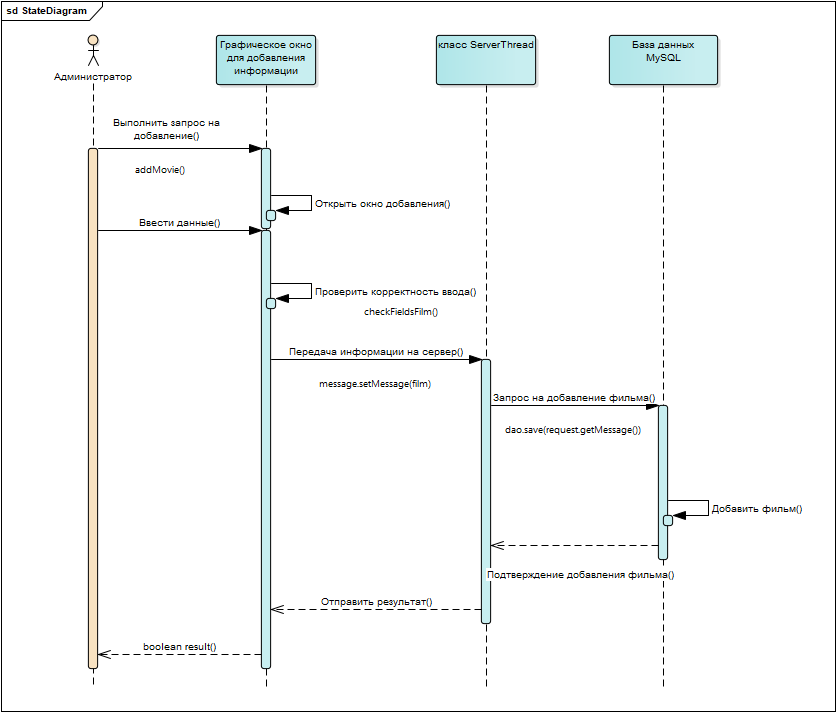


Рисунок 5.5 – Диаграмма последовательности, для процесса добавления фильма

## **5.5 Диаграмма компонентов**

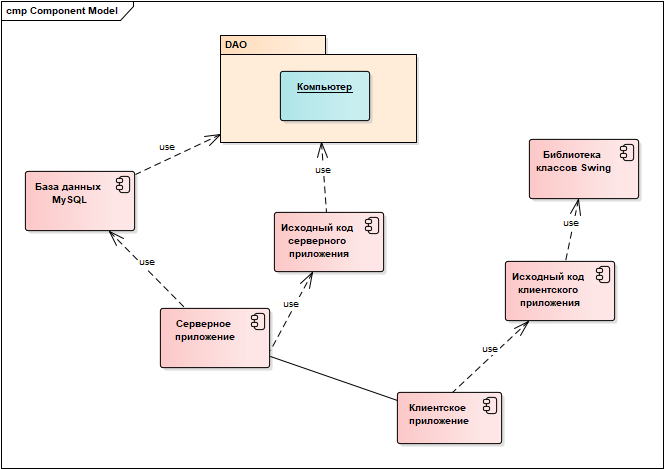


Рисунок 5.6 – Диаграмма компонентов системы управления кинотеатром

## **5.6 Диаграмма развертывания**

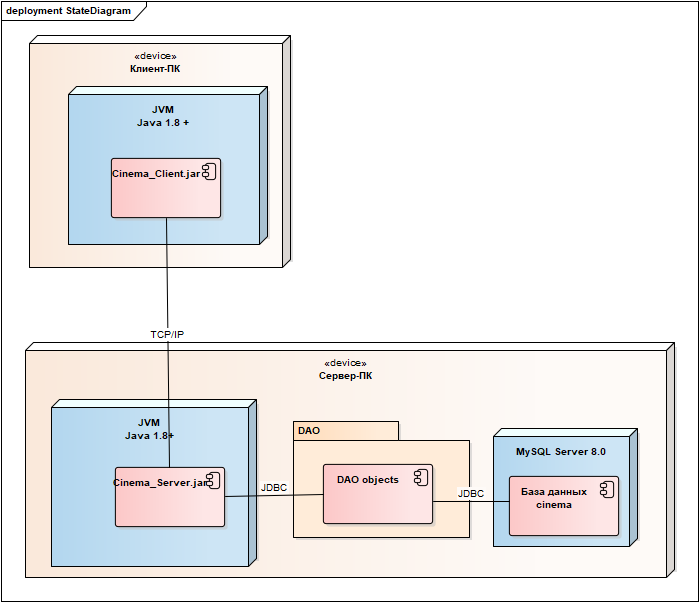


Рисунок 5.7 – Диаграмма развертывания компонентов системы

# 6 ОПИСАНИЕ АЛГОРИТМОВ, РЕАЛИЗУЮЩИХ БИЗНЕС-ЛОГИКУ РАБОТЫ СИСТЕМЫ РАБОТЫ КИНОТЕАТРА

## **6.2 Описание алгоритма покупки билетов на киносеанс**

Рассмотрим алгоритм покупки билетов на киносеанс. Первым делом клиент получает список фильмов посредством обращения к базе данных через сервер. Затем пользователю предоставляется возможность выбрать фильм из предложенных. После выбора фильма клиент получает список киносеансов, доступных для этого фильма. Если киносеансов нет, то клиент получает сообщение о том, что нет киносеансов на данный фильм. Если же киносеансы сущеcтвуют, то клиенту предоставляются возможность купить билеты на киносеанс. Клиент выбирает посадочные места для просмотра и покупает их. Блок-схема алгоритма покупки билетов представлена на рисунке Б.2 в приложении Б.

## **6.1 Описание алгоритма авторизации пользователя**

Рассмотрим алгоритм регистрации нового пользователя. Первым делом, происходит добавление информации о новом пользователе. Затем сервер получает сообщение об запросе на добавление нового пользователя и открывается соединение с базой данных. После этого, сервер проверяет, существует ли пользователь с введёнными логином, email-ом и совпадает ли подтверждение пароля. Если такие данные существуют, то на клиенте посылается сообщение, что такие данные уже существуют и нужно ввести новые, а если нет, то отправляется сообщение об успешной регистрации. Блок-схема алгоритма представлена на рисунке Б.1 в приложении Б.

# 7 РУКОВОДСТВОПОЛЬЗОВАТЕЛЯ

При запуске приложения открывается главное окно, в котором пользователь может авторизироваться или зарегистрироваться (рисунок 7.1).



Рисунок 7.1 – Главное окно приложения

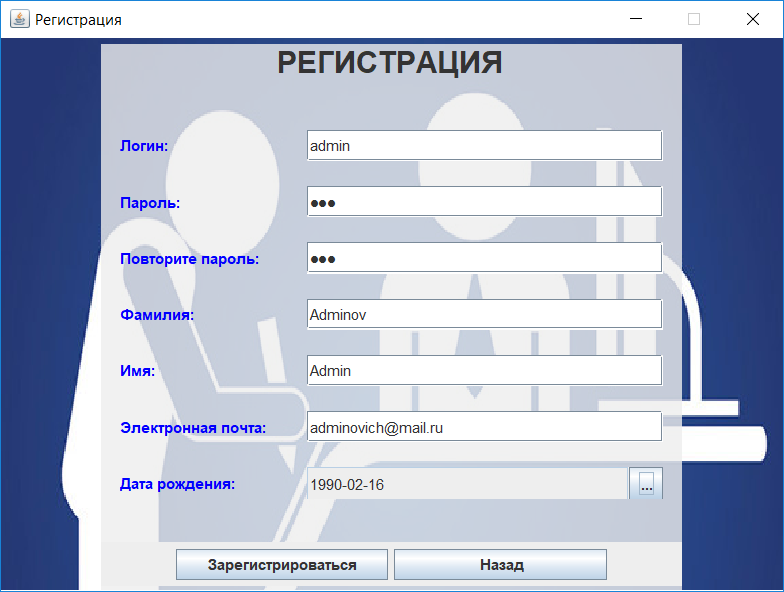


Рисунок 7.2 – Окно регистрации пользователя

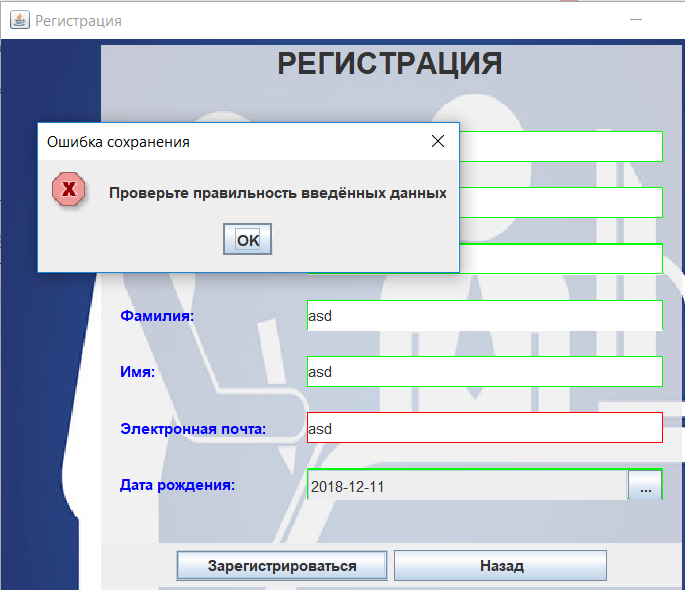


Рисунок 7.3 – Некорректные данные при регистрации

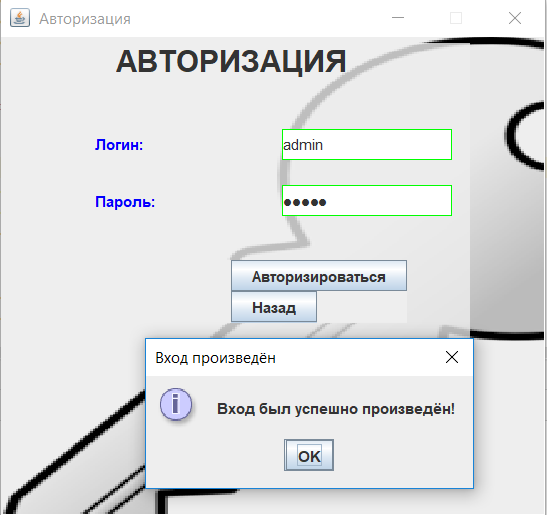


Рисунок 7.4 – Успешная авторизация

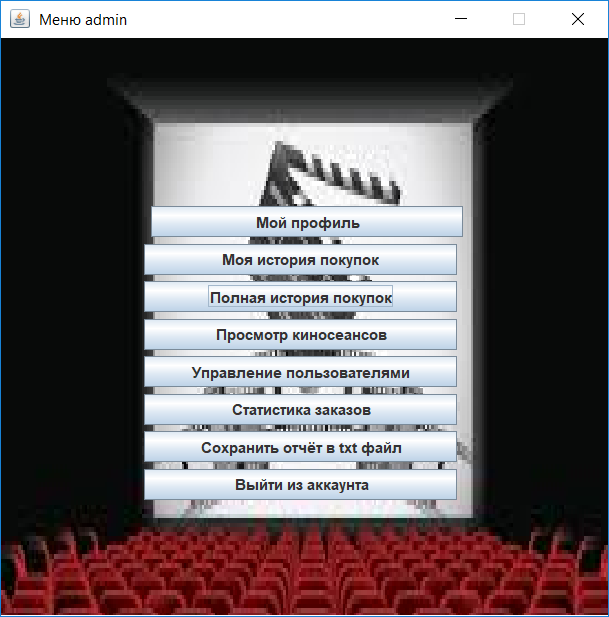


Рисунок 7.5 – Меню администратора

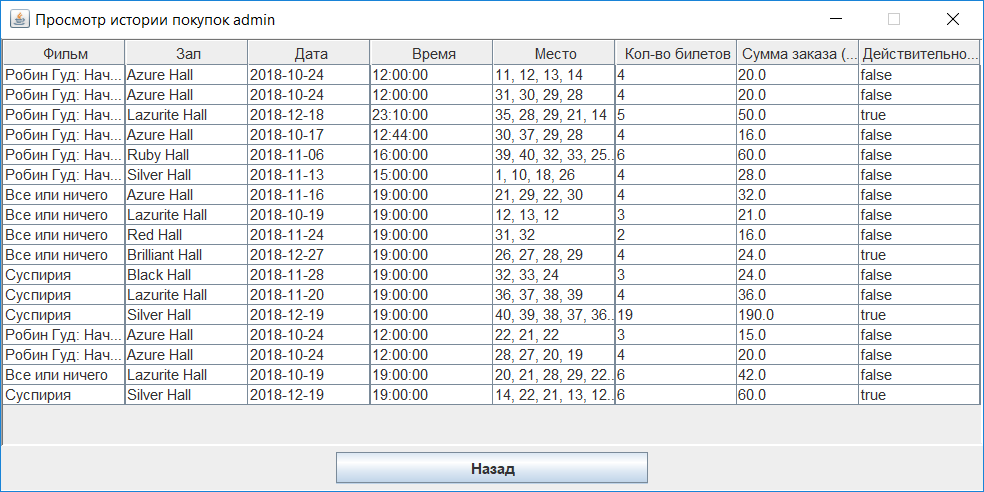


Рисунок 7.6 – История покупок пользователя

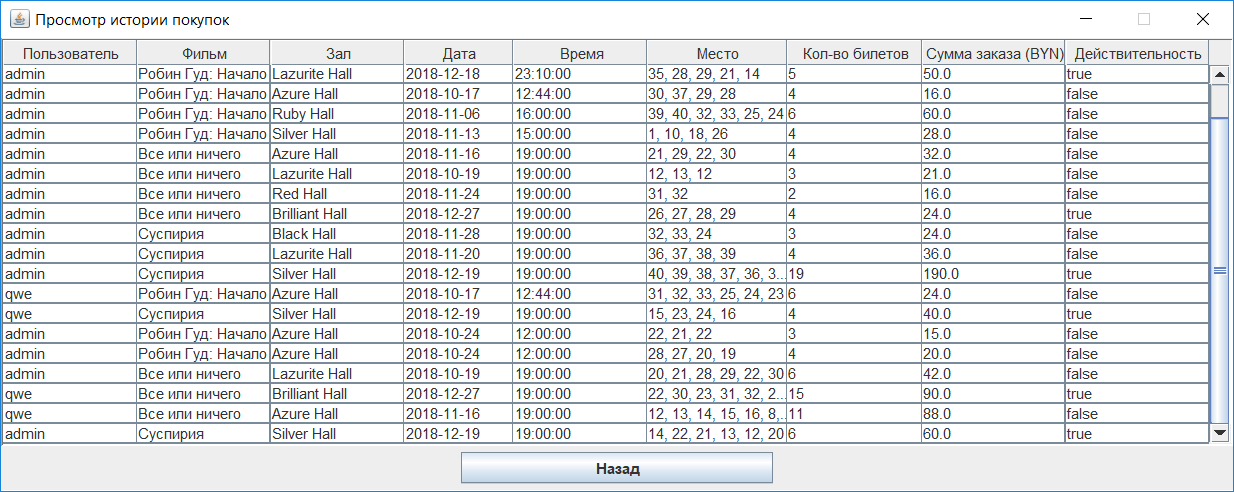


Рисунок 7.7 – Полная история покупок

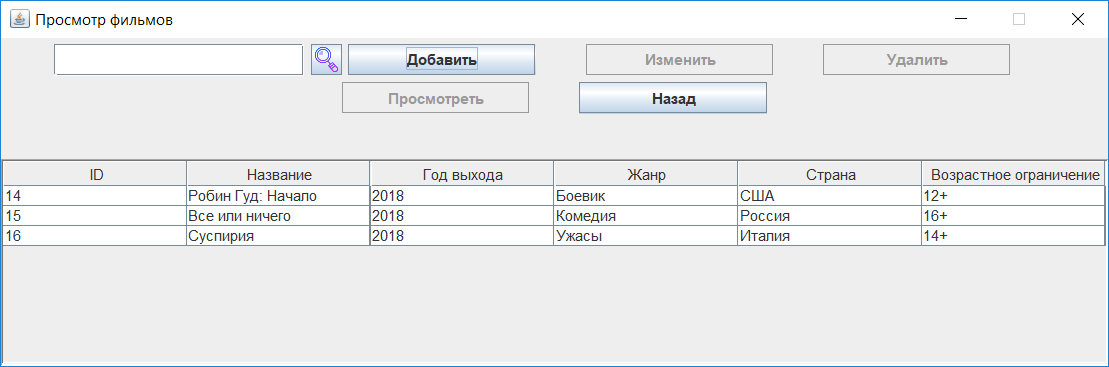


Рисунок 7.8 – Просмотр списка фильмов

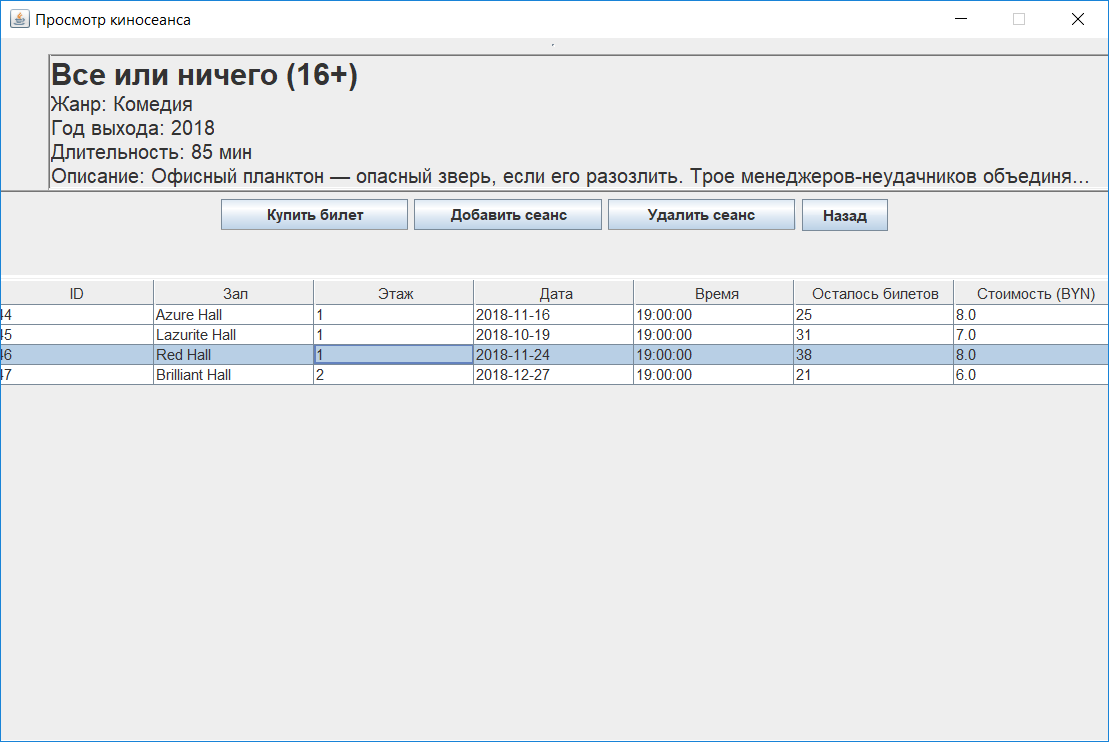


Рисунок 7.9 – Просмотр списка киносеансов на фильм

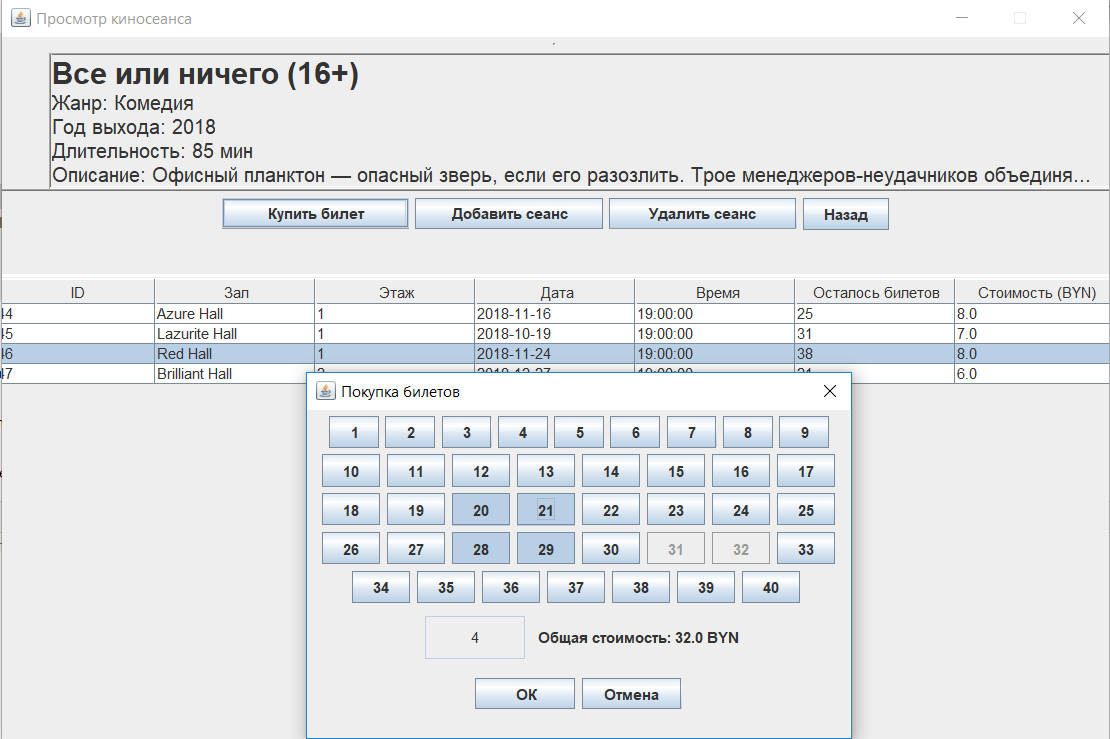


Рисунок 7.10 – Покупка билетов на киносеанс

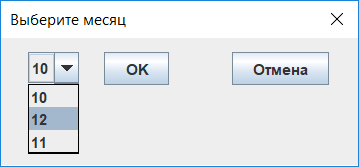


Рисунок 7.11 – Выбор месяца для отчёта

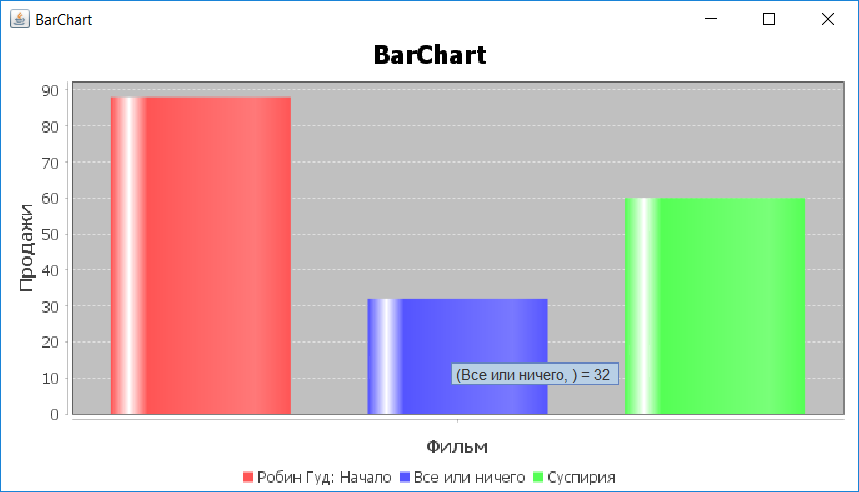


Рисунок 7.12 – Диаграмма продаж по выбранному месяцу

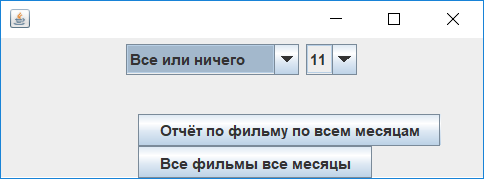


Рисунок 7.13 – Критерии выбора для отчёта в файл

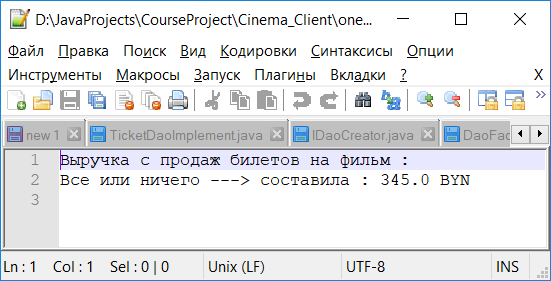


Рисунок 7.14 – Пример отчёта по одному фильму за выбранный месяц

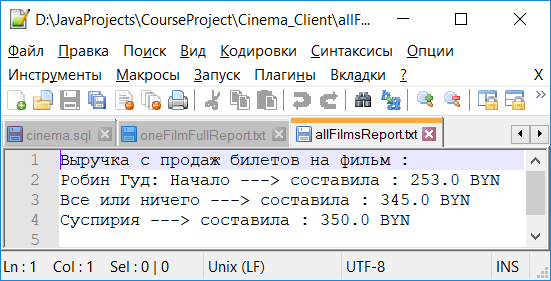


Рисунок 7.15 – Пример отчёта по всем фильмам за выбранный период

# 8 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ СИСТЕМЫ

В ходе разработки приложения применялось тестирование различных частей программы. Для проведения тестов использовалась библиотека JUnit 4.

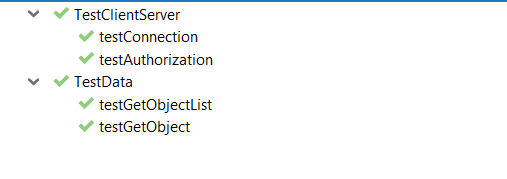


Рисунок 8.1 – Результаты тестирования

# ЗАКЛЮЧЕНИЕ

Целью данного курсового проекта стало изучение процесса продажи билетов на киносеансы. Для того, чтобы оптимизировать данный процесс, работа включила в себя создание автоматизированной системы управления происходящими в данной сфере действиями. К наиболее важным требованиям, поставленным для конкретной специфики предметной области, относят возможность покупки нескольких билетов одновременно.

Помимо данной функции, рассматривается ряд не менее важных. К ним относят:

* Хранение информации без ограничений в сроках;
* Безопасность;
* Устойчивость к отсутствию файлов сериализации (т.е. сохранения);
* Способность повторного использования без нарушения авторских прав и патентов.

Также, в ходе работы были рассмотрены и решены следующие задачи:

* Исследован и изучен процесс продажи билетов;
* Разработан алгоритм работы данной системы;
* Создана организация структуры хранимых данных;
* Разработаны функции приложения;
* Разработан и написан код программы.

Таким образом, в рамках данного курсового проекта реализовано приложение, которое является простым и удобным в использовании способом для приобретения билетов на киносеансы.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. MySQL Workbench [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.mysql.com/products/workbench/
2. Apache Maven [Электронный ресурс]. – Электронные данные. – Режим доступа: https://mvnrepository.com/
3. Кати Сьерра, Элизабет Фримен. Паттерны проектирования. Питер, 2012. – 425 с.
4. Java Swing [Электронный ресурс]. – Электронные данные. – Режим доступа: https://guru99.ru/java-swing-gui/
5. DAO Design Pattern [Электронный ресурс]. – Электронные данные. – Режим доступа: http://javatutor.net/articles/j2ee-pattern-data-access-object
6. Design Patterns [Электронный ресурс]. – Электронные данные. – Режим доступа: https://refactoring.guru/ru/design-patterns

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**IDEF-0 ДИАГРАММА**

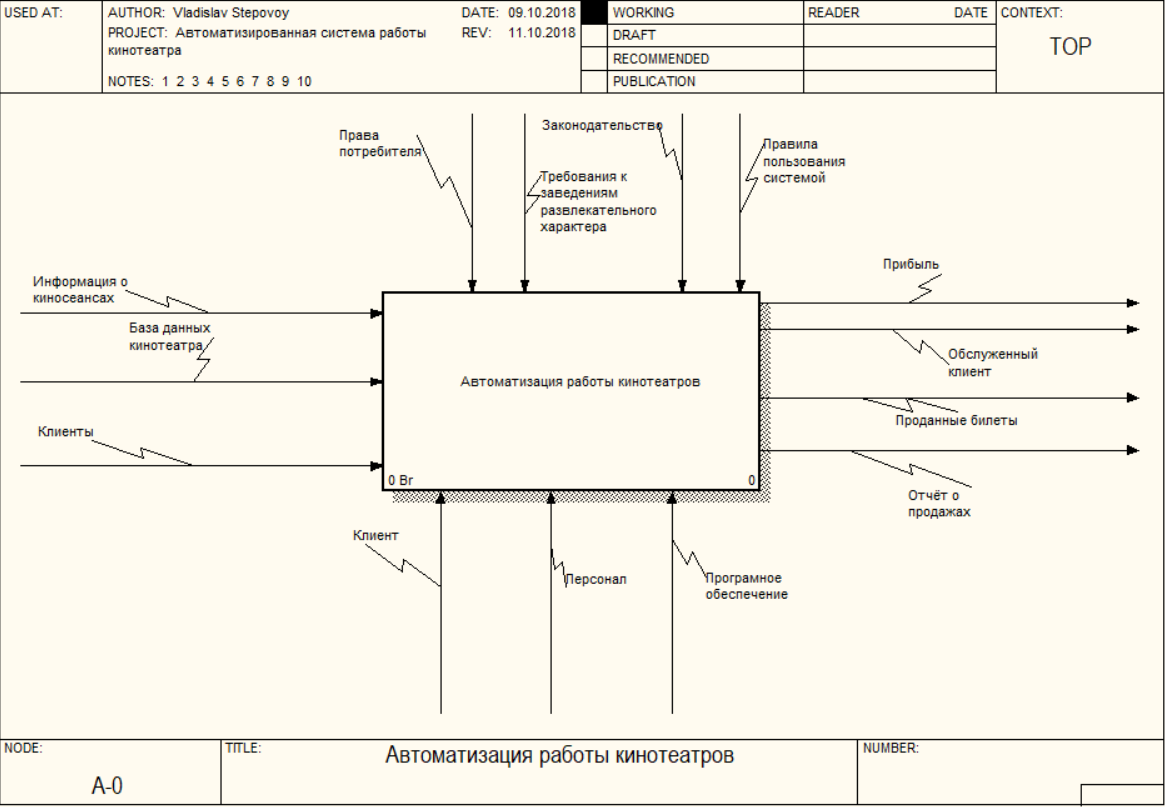


Рисунок А.1 – Контекстная диаграмма системы работы кинотеатра

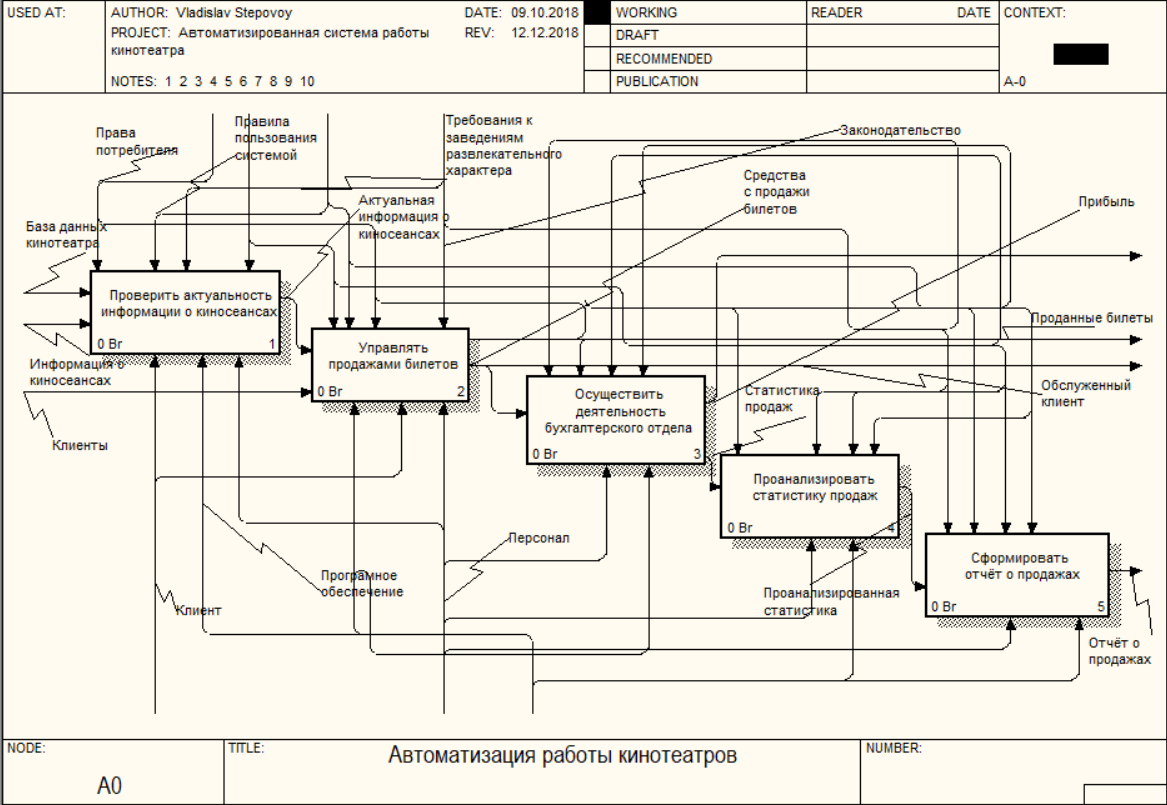


Рисунок А.2 – Декомпозиция контекстной диаграммы

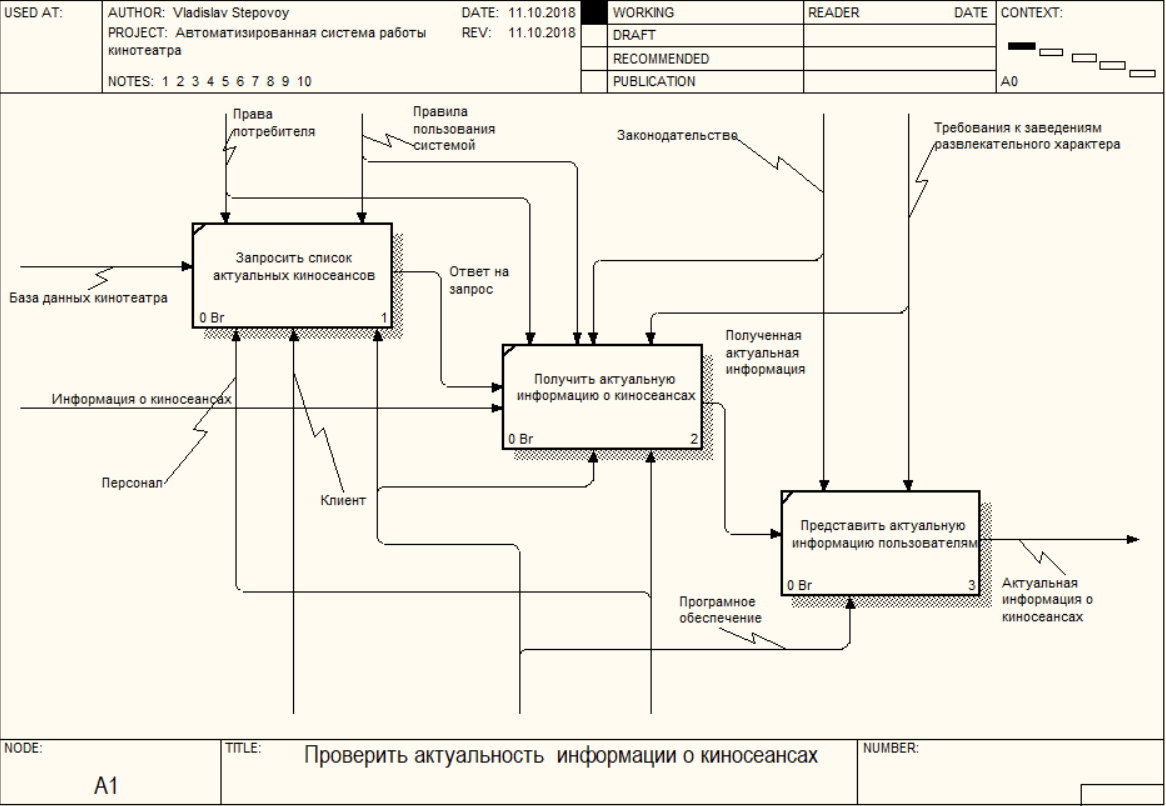


Рисунок А.3 - Декомпозиция процесса проверки актуальности информации о киносеансах

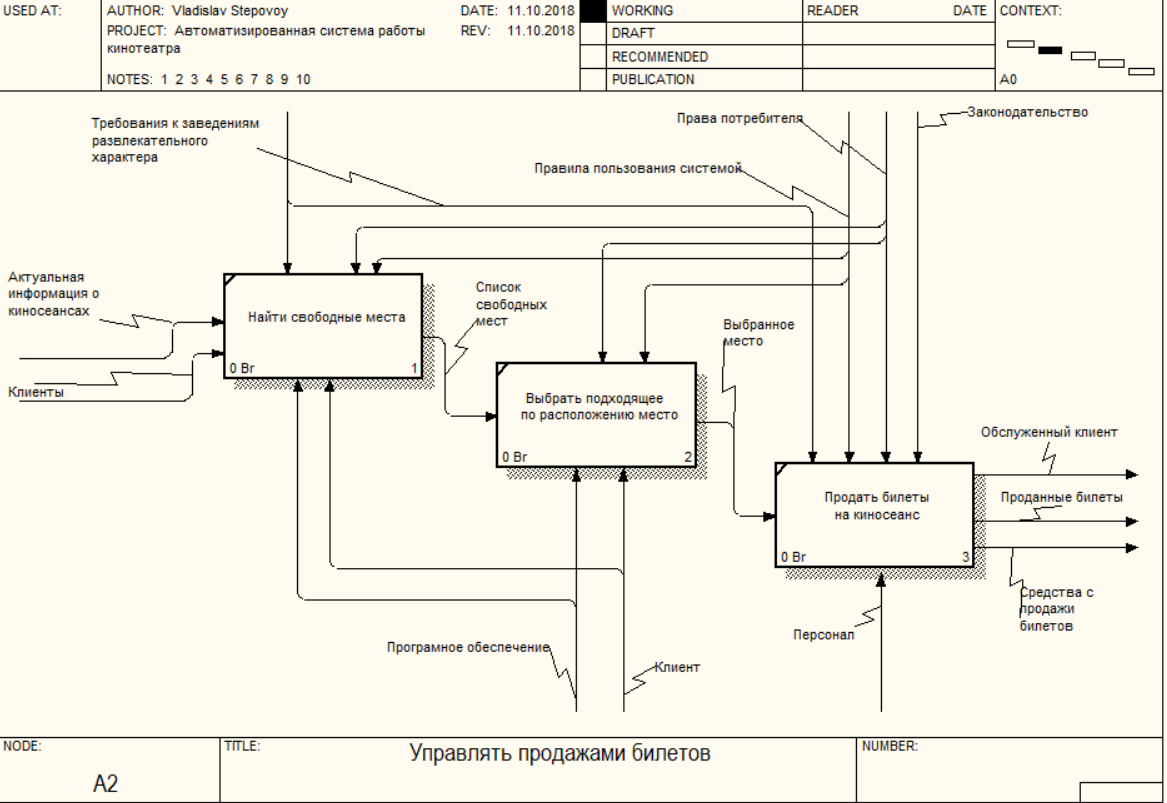


Рисунок А.4 – Декомпозиция процесса управления продажами билетов

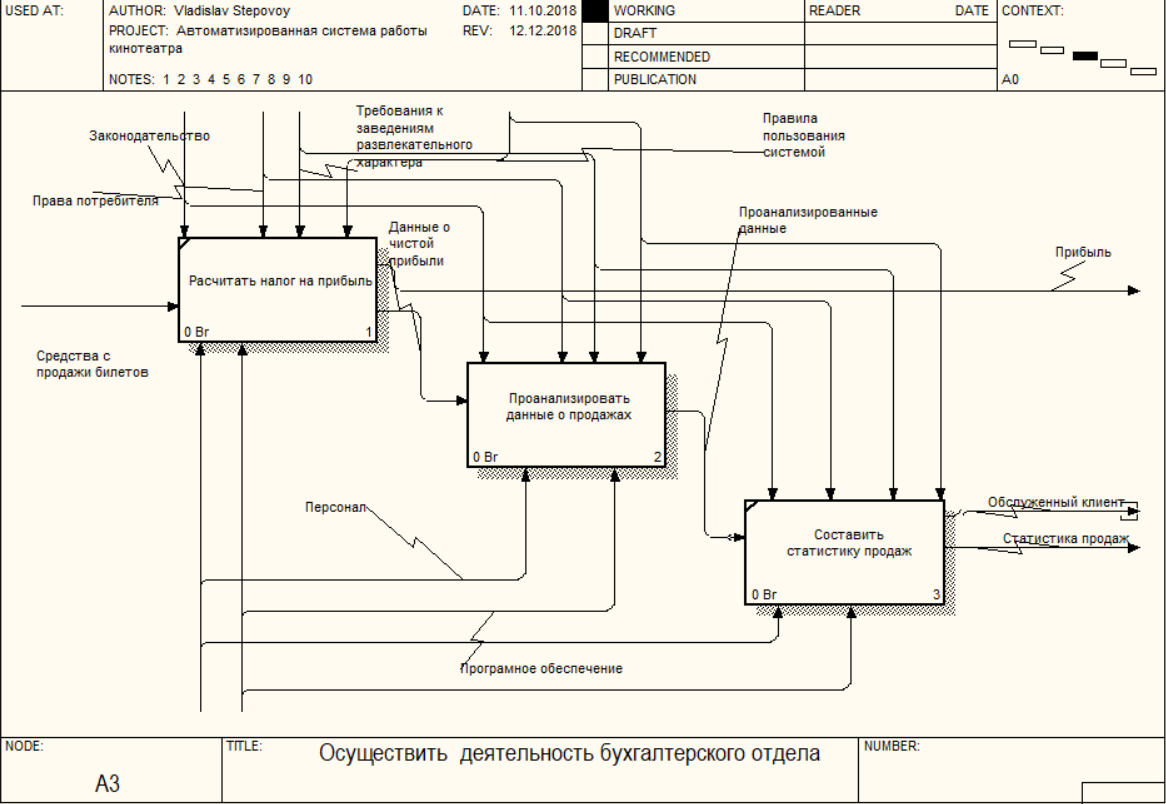


Рисунок А.5 – Декомпозиция осуществления деятельности бухгалтерского отдела

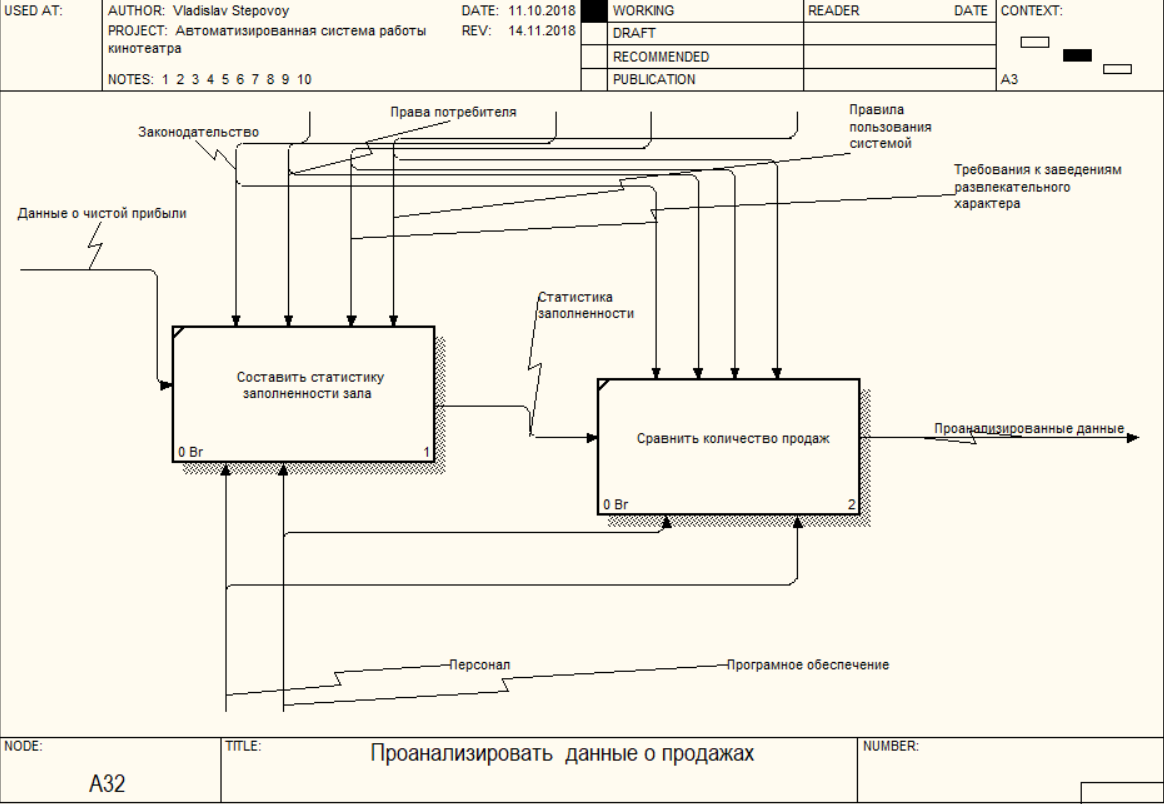


Рисунок А.6 – Декомпозиция процесса анализа данных о продажах

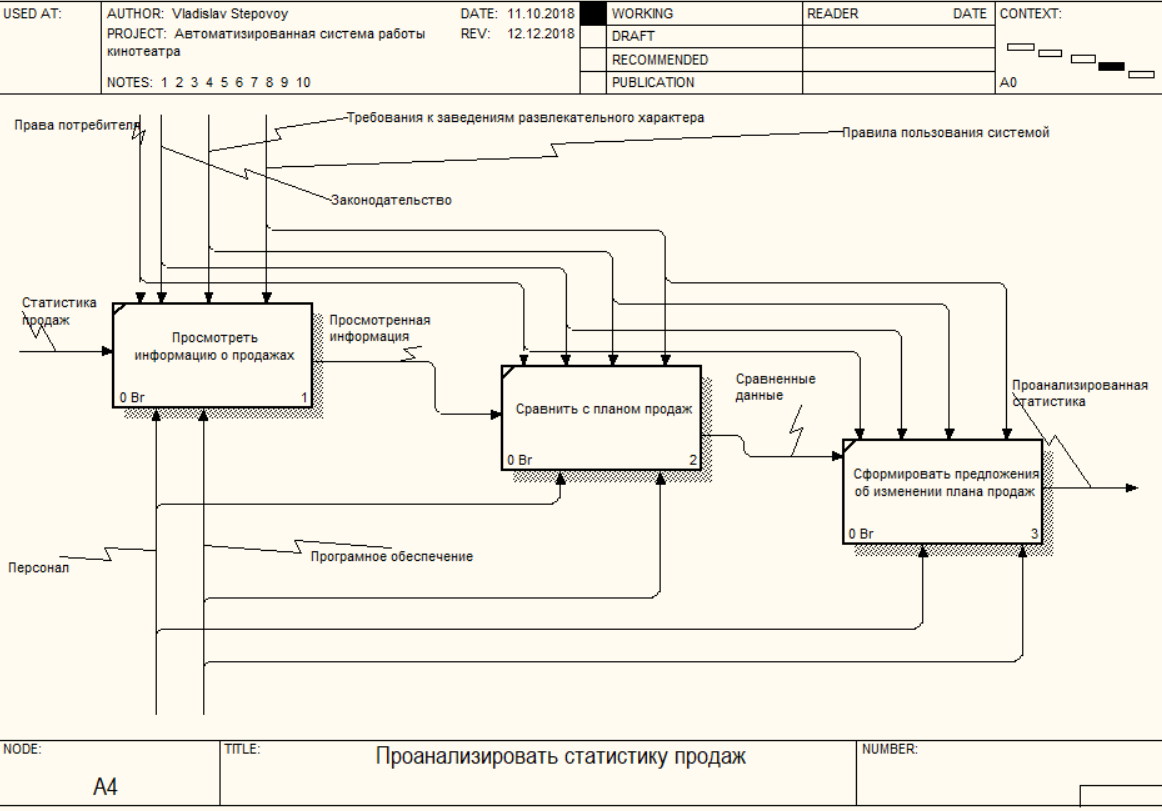


Рисунок А.7 – Декомпозиция процесса анализа статистики о продажах

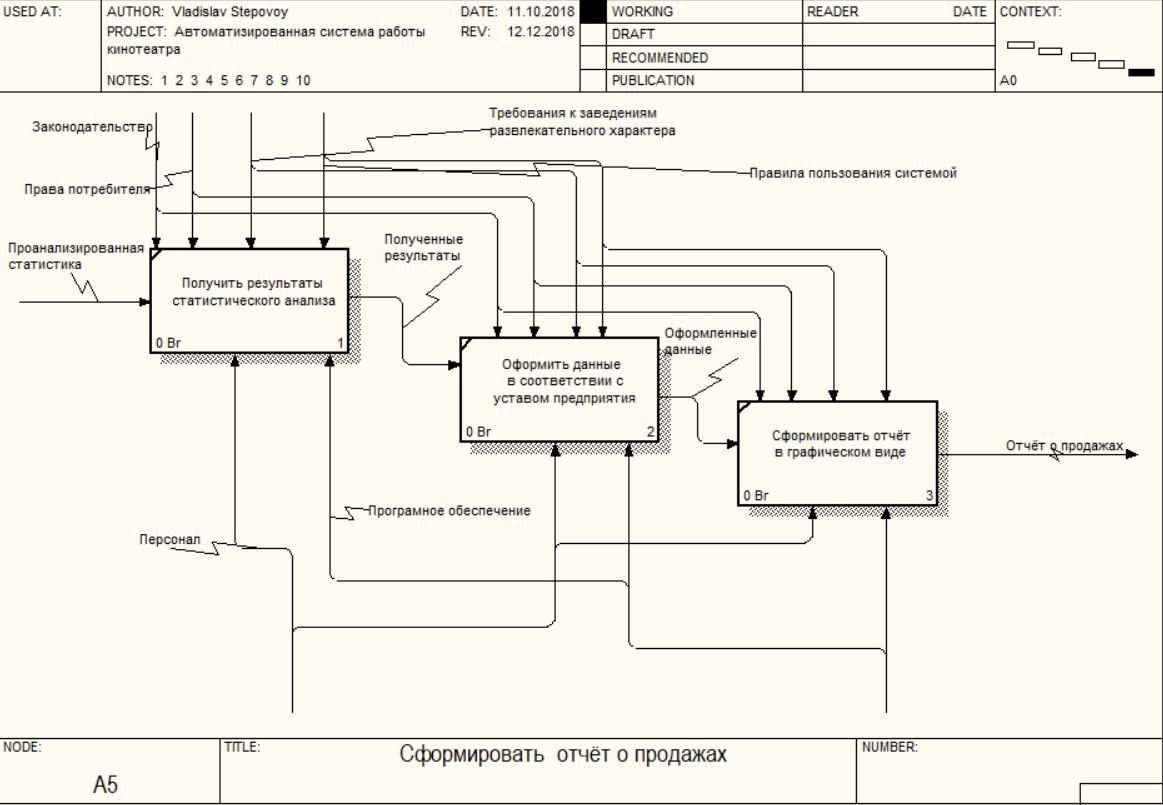


Рисунок А.8 – Декомпозиция процесса формирования отчёта о продажах

# ПРИЛОЖЕНИЕ Б

**(обязательное)**

**Алгоритм процесса авторизации пользователей**



Рисунок Б.1 – Блок-схема алгоритма регистрации пользователей

**Продолжение приложения Б**

**Алгоритм покупки билетов на киносеанс**



Рисунок Б.2 – Блок-схема алгоритма покупки билетов на киносеанс

# ПРИЛОЖЕНИЕ В

**(обязательное)**

**Листинг кода**

**package** by.stepovoy.view;  
  
**import** by.stepovoy.client.ClientThread;  
**import** by.stepovoy.model.Seance;  
**import** by.stepovoy.model.Ticket;  
**import** by.stepovoy.model.User;  
**import** by.stepovoy.utils.Message;  
**import** by.stepovoy.utils.MessageType;  
  
**import** javax.swing.JFrame;  
**import** javax.swing.JTextField;  
**import** javax.swing.Box;  
**import** javax.swing.JDialog;  
**import** javax.swing.JLabel;  
**import** javax.swing.JPanel;  
**import** javax.swing.JButton;  
**import** javax.swing.JToggleButton;  
**import** javax.swing.SwingConstants;  
**import** java.awt.FlowLayout;  
**import** java.awt.Dimension;  
**import** java.awt.Component;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**import** java.io.IOException;  
**import** java.util.List;  
**import** java.util.LinkedList;  
**import** java.util.ArrayList;  
**import** java.util.Date;  
**import** java.util.Arrays;  
  
  
**public class** BuyTicketPanel **extends** JDialog {  
  
 **private** User **user**;  
 **private** Seance **seance**;  
 **private** JTextField **tickets**;  
 **private** JLabel **cost**;  
 **private** JButton **buyButton**;  
 **private double totalCost**;  
 **private int totalTickets**;  
 **private** String **ticketNumber** = **""**;  
  
 **public** BuyTicketPanel(JFrame parentFrame, User user, **final** Seance seance, **boolean** isModal) **throws** IOException, ClassNotFoundException {  
 **super**(parentFrame, isModal);  
 setLocationRelativeTo(**null**);  
 **this**.**user** = user;  
 **this**.**seance** = seance;  
 setBounds(550, 600, 450, 300);  
 JPanel mainPanel = **new** JPanel();  
 mainPanel.setLayout(**new** FlowLayout());  
 mainPanel.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 setContentPane(mainPanel);  
 setTitle(**"Покупка билетов"**);  
 List<String> numbersList = **new** LinkedList<>();  
 String delimiter = **", "**;  
 String[] seats;  
 String str = **""**;  
 List<Ticket> ticketList = ClientThread.*getAllTickets*();  
 **for** (Ticket ticket : ticketList) {  
 **if** (ticket.getSeanceID() == seance.getID()) {  
 str = ticket.getSeatNumber();  
 }  
 }  
 seats = str.split(delimiter);  
 List<String> soldTickets = **new** LinkedList<>(Arrays.*asList*(seats));  
 ArrayList<JToggleButton> buttonList = **new** ArrayList<>();  
 **for** (String string : soldTickets) {  
 **for** (**int** i = 0; i < 40; i++) {  
 buttonList.add(**new** JToggleButton(**""** + (i + 1)));  
 mainPanel.add(buttonList.get(i));  
 **if** (string.equals(buttonList.get(i).getText())) {  
 buttonList.get(i).setEnabled(**false**);  
 }  
 }  
 }  
 **cost** = **new** JLabel(**"Общая стоимость: 0 BYN"**);  
 **cost**.setHorizontalAlignment(SwingConstants.***CENTER***);  
 **tickets** = **new** JTextField();  
 **tickets**.setPreferredSize(**new** Dimension(80, 35));  
 **tickets**.setEditable(**false**);  
 **tickets**.setHorizontalAlignment(JTextField.***CENTER***);  
 **tickets**.setText(**"0"**);  
 **totalTickets** = 0;  
 **totalCost** = 0;  
 **for** (**final** JToggleButton button : buttonList) {  
 button.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **if** (e.getSource() == button && button.isSelected()) {  
 button.setSelected(**true**);  
 **if** (**totalTickets** < seance.getTicketsLeft()) {  
 **totalTickets**++;  
 **totalCost** += seance.getTicketCost();  
 **tickets**.setText(String.*valueOf*(**totalTickets**));  
 **cost**.setText(**"Общая стоимость: "** + **totalCost** + **" BYN"**);  
  
 numbersList.add(button.getText());  
  
 **if** (**totalTickets** == seance.getTicketsLeft() - 1) {  
 **buyButton**.setEnabled(**false**);  
 } **else** {  
 **buyButton**.setEnabled(**true**);  
 }  
 }  
 } **else** {  
 **if** (**totalTickets** > 0) {  
 **totalTickets**--;  
 **totalCost** -= seance.getTicketCost();  
 **tickets**.setText(String.*valueOf*(**totalTickets**));  
 **cost**.setText(**"Общая стоимость: "** + **totalCost** + **" BYN"**);  
 numbersList.remove(numbersList.size() - 1);  
 **if** (**totalTickets** < 1) {  
 **buyButton**.setEnabled(**false**);  
 } **else** {  
 **buyButton**.setEnabled(**true**);  
 }  
 }  
 }  
 String delimiter = **", "**;  
 **ticketNumber** = String.*join*(delimiter, numbersList);  
 System.***out***.println(**ticketNumber**);  
  
 }  
 });  
 }  
 JPanel costPanel = **new** JPanel();  
 costPanel.setLayout(**new** FlowLayout());  
 costPanel.add(**tickets**);  
  
 **buyButton** = **new** JButton(**"ОК"**);  
 **buyButton**.setPreferredSize(**new** Dimension(80, 25));  
 **buyButton**.setEnabled(**false**);  
 **buyButton**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 buyActionPerformed();  
 }  
 });  
 JButton cancelButton = **new** JButton(**"Отмена"**);  
 cancelButton.setPreferredSize(**new** Dimension(80, 25));  
 cancelButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 cancelActionPerformed();  
 }  
 });  
 JPanel buttonPanel = **new** JPanel();  
 buttonPanel.setLayout(**new** FlowLayout());  
 buttonPanel.add(**buyButton**);  
 buttonPanel.add(cancelButton);  
 mainPanel.add(Box.*createRigidArea*(**new** Dimension(0, 10)));  
 mainPanel.add(costPanel);  
 mainPanel.add(**cost**);  
 mainPanel.add(buttonPanel);  
 }  
 **private void** cancelActionPerformed() {  
 dispose();  
 }  
 **private void** buyActionPerformed() {  
 Ticket ticket = **new** Ticket();  
 ticket.setSeatNumber(**ticketNumber**);  
 ticket.setSeanceID(**seance**.getID());  
 **seance**.setTicketsLeft(**seance**.getTicketsLeft() - **totalTickets**);  
 ticket.setUserID(**user**.getID());  
 ticket.setAmountTickets(**totalTickets**);  
 ticket.setCost(**totalCost**);  
 Date currentDate = **new** Date();  
 **if** (currentDate.before(**seance**.getSeanceDate()) ||  
 currentDate.equals(**seance**.getSeanceDate())) {  
 ticket.setValid(**true**);  
 } **else** {  
 ticket.setValid(**false**);  
 }  
 Message message = **new** Message();  
 message.setOperationType(MessageType.***ADD***);  
 message.setMessageType(MessageType.***TICKET***);  
 message.setMessage(ticket);  
 **try** {  
 ClientThread.*sendMessage*(message);  
 ClientThread.*receiveMessage*();  
 } **catch** (  
 Exception e) {  
 e.printStackTrace();  
 }  
 message = **new** Message();  
 message.setOperationType(MessageType.***UPDATE***);  
 message.setMessageType(MessageType.***SEANCE***);  
 message.setMessage(**seance**);  
 **try** {  
 ClientThread.*sendMessage*(message);  
 ClientThread.*receiveMessage*();  
 } **catch** (  
 Exception e) {  
 e.printStackTrace();  
 }  
  
 dispose();  
 }  
}

**package** by.stepovoy.view;  
  
**import** by.stepovoy.client.ClientThread;  
**import** by.stepovoy.model.Film;  
**import** by.stepovoy.model.Seance;  
**import** by.stepovoy.model.Ticket;  
**import** org.joda.time.DateTime;  
  
**import** javax.swing.JFrame;  
**import** javax.swing.JComboBox;  
**import** javax.swing.JPanel;  
**import** javax.swing.JButton;  
**import** javax.swing.BoxLayout;  
**import** java.awt.FlowLayout;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**import** java.io.IOException;  
**import** java.util.List;  
**import** java.util.ArrayList;  
**import** java.io.PrintWriter;  
**import** java.util.Objects;  
  
**public class** ReportGeneratorPanel **extends** JFrame {  
  
 **public** ReportGeneratorPanel() {  
 setSize(400, 150);  
 setLocationRelativeTo(**null**);  
 JPanel panel = **new** JPanel();  
 panel.setLayout(**new** BoxLayout(panel, BoxLayout.***Y\_AXIS***));  
 setContentPane(panel);  
  
 JPanel comboPanel = **new** JPanel();  
 comboPanel.setLayout(**new** FlowLayout());  
 panel.add(comboPanel);  
  
 JComboBox filmComboBox = **new** JComboBox();  
 comboPanel.add(filmComboBox);  
 JComboBox monthComboBox = **new** JComboBox();  
 comboPanel.add(monthComboBox);  
 JPanel buttonsPanel = **new** JPanel();  
 buttonsPanel.setLayout(**new** BoxLayout(buttonsPanel, BoxLayout.***Y\_AXIS***));  
 panel.add(buttonsPanel);  
 JButton showOneFilmAllMonthReport = **new** JButton(**"Отчёт по фильму по всем месяцам"**);  
 buttonsPanel.add(showOneFilmAllMonthReport);  
 JButton showAllFilmsAllMonthReport = **new** JButton(**"Все фильмы все месяцы"**);  
 buttonsPanel.add(showAllFilmsAllMonthReport);  
  
 **try** {  
 List<Seance> seanceList = ClientThread.*getAllSeances*();  
 List<Integer> listOfMonths = **new** ArrayList<>();  
 List<Film> filmList = ClientThread.*getAllFilms*();  
 **for** (Film film : filmList) {  
 filmComboBox.addItem(film.getName());  
 }  
  
 **for** (Seance seance : seanceList) {  
 DateTime datetime = **new** DateTime(seance.getSeanceDate());  
 listOfMonths.add(Integer.*parseInt*(datetime.toString(**"MM"**)));  
 }  
 List<Integer> uniqueElements = ChooseMonthPanel.*makeUniqueElements*(listOfMonths);  
 **for** (Integer month : uniqueElements) {  
 monthComboBox.addItem(month);  
 }  
  
 showAllFilmsAllMonthReport.addActionListener(**new** ActionListener() {  
 List<Ticket> **tickets** = **null**;  
 List<Film> **films** = **null**;  
 List<Seance> **seances** = **null**;  
 **int previousID** = 0;  
 **double profit** = 0;  
  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 PrintWriter printWriter;  
  
  
 **try** {  
 **tickets** = ClientThread.*getAllTickets*();  
 **films** = ClientThread.*getAllFilms*();  
 **seances** = ClientThread.*getAllSeances*();  
 printWriter = **new** PrintWriter(**"allFilmsReport.txt"**);  
 **if** (printWriter != **null**) {  
 printWriter.write(**"Выручка с продаж билетов на фильм : \n"**);  
 }  
 **for** (Film film : **films**) {  
 **for** (Seance seance : **seances**) {  
 **previousID** = seance.getFilmID();  
 **if** (film.getID() == seance.getFilmID()) {  
 **for** (Ticket ticket : **tickets**) {  
 **if** (seance.getID() == ticket.getSeanceID()) {  
 **profit** += ticket.getCost();  
 }  
 }  
 }  
 }  
 **if** (printWriter != **null**) {  
 printWriter.write(film.getName() +  
 **" ---> составила : "** + **profit** + **" BYN\n"**);  
 }  
 **if** (film.getID() != **previousID**) {  
 **profit** = 0;  
 }  
 }  
 **if** (printWriter != **null**) {  
 printWriter.close();  
 }  
 } **catch** (IOException |  
 ClassNotFoundException e1) {  
 e1.printStackTrace();  
 }  
 }  
 });  
  
 showOneFilmAllMonthReport.addActionListener(**new** ActionListener() {  
 List<Film> **films** = **null**;  
 List<Seance> **seances** = **null**;  
 List<Ticket> **tickets** = **null**;  
  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 PrintWriter printWriter = **null**;  
 **try** {  
 **films** = ClientThread.*getAllFilms*();  
 **seances** = ClientThread.*getAllSeances*();  
 **tickets** = ClientThread.*getAllTickets*();  
 **int** previousID = 0;  
 **double** profit = 0.;  
 printWriter = **new** PrintWriter(**"oneFilmFullReport.txt"**);  
 **if** (printWriter != **null**) {  
 printWriter.write(**"Выручка с продаж билетов на фильм : \n"**);  
 }  
 **for** (Film film : **films**) {  
 **for** (Seance seance : **seances**) {  
 previousID = seance.getFilmID();  
 **if** (film.getID() == seance.getFilmID()) {  
 **for** (Ticket ticket : **tickets**) {  
 **if** (seance.getID() == ticket.getSeanceID()) {  
 profit += ticket.getCost();  
 }  
 }  
 }  
 }  
 **if** (Objects.*equals*(filmComboBox.getSelectedItem(), film.getName())) {  
 **if** (printWriter != **null**) {  
 printWriter.write(film.getName() +  
 **" ---> составила : "** + profit + **" BYN\n"**);  
 }  
 }  
 **if** (film.getID() != previousID) {  
 profit = 0;  
 }  
  
 }  
 **if** (printWriter != **null**) {  
 printWriter.close();  
 }  
  
  
 } **catch** (Exception e1) {  
 e1.getCause();  
 }  
 }  
 });  
  
 } **catch** (IOException |  
 ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 }  
}

**package** by.stepovoy.view;  
**import** by.stepovoy.client.ClientThread;  
**import** by.stepovoy.utils.Message;  
**import** by.stepovoy.utils.MessageType;  
**import** by.stepovoy.model.Film;  
**import** by.stepovoy.model.Seance;  
**import** by.stepovoy.model.Ticket;  
**import** org.jfree.chart.ChartFactory;  
**import** org.jfree.chart.ChartFrame;  
**import** org.jfree.chart.JFreeChart;  
**import** org.jfree.data.category.DefaultCategoryDataset;  
**import** org.joda.time.DateTime;  
  
**import** javax.swing.JFrame;  
**import** javax.swing.JComboBox;  
**import** javax.swing.Box;  
**import** javax.swing.JDialog;  
**import** javax.swing.JPanel;  
**import** javax.swing.JButton;  
**import** javax.swing.BoxLayout;  
**import** java.awt.FlowLayout;  
**import** java.awt.Dimension;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**import** java.io.IOException;  
**import** java.util.ArrayList;  
**import** java.util.LinkedHashSet;  
**import** java.util.List;  
**import** java.util.Set;  
  
**import static** org.jfree.chart.plot.PlotOrientation.***VERTICAL***;  
  
**public class** ChooseMonthPanel **extends** JDialog {  
  
 **public** ChooseMonthPanel(**final** JFrame parentFrame) {  
 **super**(parentFrame);  
 setLocationRelativeTo(**null**);  
 **final** JComboBox monthsList = **new** JComboBox();  
 JPanel comboBoxPanel = **new** JPanel();  
 **try** {  
 List<Seance> seanceList = ClientThread.*getAllSeances*();  
 List<Integer> listOfMonths = **new** ArrayList<>();  
 **for** (Seance seance : seanceList) {  
 DateTime datetime = **new** DateTime(seance.getSeanceDate());  
 listOfMonths.add(Integer.*parseInt*(datetime.toString(**"MM"**)));  
 }  
 List<Integer> uniqueElements = *makeUniqueElements*(listOfMonths);  
 **for** (Integer month : uniqueElements) {  
 monthsList.addItem(month);  
 }  
  
  
 } **catch** (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 comboBoxPanel.add(monthsList);  
 JButton okButton = **new** JButton(**"OK"**);  
 okButton.setEnabled(**true**);  
 okButton.addActionListener(**new** ActionListener() {  
 **public void** actionPerformed(ActionEvent e) {  
  
 Seance seance = **new** Seance(0);  
 **int** filmID = 0;  
 **int** previousID = 0;  
 Film film;  
 String filmName = **null**;  
 List<Ticket> ticketList;  
 **double** sumOfOrder = 0.;  
 DefaultCategoryDataset dataset = **new** DefaultCategoryDataset();  
 **try** {  
 ticketList = ClientThread.*getAllTickets*();  
 **for** (Ticket ticket : ticketList) {  
  
 Message message = **new** Message();  
 message.setOperationType(MessageType.***GET***);  
 message.setMessageType(MessageType.***SEANCE***);  
 message.setMessage(ticket.getSeanceID());  
 ClientThread.*sendMessage*(message);  
 previousID = seance.getFilmID();  
 seance = (Seance) ClientThread.*receiveMessage*().getMessage();  
 DateTime datetime = **new** DateTime(seance.getSeanceDate());  
 **if** (monthsList.getSelectedItem() == (Integer) Integer.*parseInt*(datetime.toString(**"MM"**))) {  
 message = **new** Message();  
 message.setOperationType(MessageType.***GET***);  
 message.setMessageType(MessageType.***FILM***);  
 message.setMessage(seance.getFilmID());  
 ClientThread.*sendMessage*(message);  
  
 **if** (message.getMessageType() == MessageType.***FILM***) {  
 film = (Film) ClientThread.*receiveMessage*().getMessage();  
 filmID = film.getID();  
 filmName = film.getName();  
 **if** (filmID != previousID) {  
 sumOfOrder = 0.;  
 }  
 }  
  
 sumOfOrder += ticket.getCost();  
 dataset.setValue(sumOfOrder, filmName, **""**);  
 }  
 }  
 } **catch** (IOException | ClassNotFoundException e1) {  
 e1.printStackTrace();  
 }  
  
 JFreeChart jFreeChart = ChartFactory.*createBarChart*(**"BarChart"**,  
 **"Фильм"**,  
 **"Продажи"**,  
 dataset, ***VERTICAL***,  
 **true**, **true**, **false**);  
 ChartFrame frame = **new** ChartFrame(jFreeChart.getTitle().getText(), jFreeChart);  
 frame.setSize(700, 400);  
 frame.setLocationRelativeTo(**null**);  
 frame.setVisible(**true**);  
 dispose();  
 }  
  
 });  
 JButton cancelButton = **new** JButton(**"Отмена"**);  
 cancelButton.addActionListener(**new** ActionListener() {  
 **public void** actionPerformed(ActionEvent e) {  
 parentFrame.setVisible(**true**);  
 dispose();  
 }  
 });  
 JPanel buttonsPanel = **new** JPanel();  
 buttonsPanel.setLayout(**new** FlowLayout());  
 buttonsPanel.add(okButton);  
 buttonsPanel.add(Box.*createRigidArea*(**new** Dimension(40, 0)));  
 buttonsPanel.add(cancelButton);  
 **this**.setBounds(720, 480, 300, 140);  
 **this**.setLayout(**new** BoxLayout(**this**, BoxLayout.***Y\_AXIS***));  
 JPanel mainPanel = **new** JPanel();  
 mainPanel.add(comboBoxPanel);  
 mainPanel.add(Box.*createRigidArea*(**new** Dimension(0, 40)));  
 mainPanel.add(buttonsPanel);  
 mainPanel.setVisible(**true**);  
 **this**.setContentPane(mainPanel);  
 **this**.setTitle(**"Выберите месяц"**);  
 **this**.setResizable(**false**);  
 **this**.setVisible(**true**);  
 }  
  
  
 **public static** List<Integer> makeUniqueElements(List<Integer> arrayList) {  
 Set<Integer> set = **new** LinkedHashSet<>(arrayList);  
 **return new** ArrayList<>(set);  
 }  
}

**package** by.stepovoy.client;  
  
**import** by.stepovoy.view.MainPanel;  
  
**import** java.util.logging.Logger;  
  
**public class** Client {  
  
 **private static final int *PORT*** = 1888;  
 **private static** Logger *LOGGER* = Logger.*getLogger*(**"MyLogger"**);  
  
 **public static void** main(String[] args) {  
 **int** port = 0;  
 **if** (args.**length** == 0) {  
 port = ***PORT***;  
 } **else if** (Integer.*parseInt*(args[0]) != 0 && args[0].length() == 4) {  
 port = Integer.*parseInt*(args[0]);  
 } **else** {  
 *LOGGER*.info(**"Invalid port!"**);  
 System.*exit*(0);  
 }  
 MainPanel mainPanel = **new** MainPanel(port);  
 mainPanel.setVisible(**true**);  
 }  
}

**package** by.stepovoy.server;  
  
**import** java.net.ServerSocket;  
**import** java.net.Socket;  
**import** java.util.logging.Logger;

**public class** Server {  
  
 **private static final int *PORT*** = 1888;  
 **private static int** *clientAmount* = 1;  
 **private static** Logger *LOGGER* = Logger.*getLogger*(**"MyLogger"**);  
  
 **public static void** main(String[] args) {  
 **try** {  
  
 **int** port = 0;  
 **if** (args.**length** != 0 && Integer.*parseInt*(args[0]) != 0 && args[0].length() == 4) {  
 port = Integer.*parseInt*(args[0]);  
 } **else if** (args.**length** == 0) {  
 port = ***PORT***;  
 } **else** {  
 *LOGGER*.info(**"INVALID PORT!"**);  
 System.*exit*(0);  
 }  
  
 ServerSocket serverSocket = **new** ServerSocket(port);  
 *LOGGER*.info(**"Server waiting connections ...."**);  
 **while** (**true**) {  
 Socket client = serverSocket.accept();  
 *LOGGER*.info(**"Connected "** + *clientAmount* + **" client :\n"** +  
 **"Client information: "** +  
 client.getInetAddress().getHostAddress());  
 *clientAmount*++;  
 ServerThread serverThread = **new** ServerThread(client);  
 *LOGGER*.info(**"Current users online : "** + ServerThread.*connectionNumber*);  
 serverThread.start();  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

# ПРИЛОЖЕНИЕ Г

**(обязательное)**

**Листинг SQL-скрипт**

-- MySQL Workbench Forward Engineering

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

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

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

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

-- Schema cinema

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

CREATE SCHEMA IF NOT EXISTS `cinema` DEFAULT CHARACTER SET utf8;

USE `cinema`;

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

-- Table `cinema`.`film`

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

CREATE TABLE IF NOT EXISTS `cinema`.`film`

(

`ID` INT(11) NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(40) NULL DEFAULT NULL,

`duration` INT(11) NULL DEFAULT NULL,

`description` TEXT NULL DEFAULT NULL,

`genre` VARCHAR(25) NULL DEFAULT NULL,

`country` VARCHAR(30) NULL DEFAULT NULL,

`director` VARCHAR(30) NULL DEFAULT NULL,

`is3D` TINYINT(1) NULL DEFAULT NULL,

`ageLimit` INT(11) NULL DEFAULT NULL,

`yearProduction` INT(11) NULL DEFAULT NULL,

PRIMARY KEY (`ID`)

)

ENGINE = InnoDB

AUTO\_INCREMENT = 18

DEFAULT CHARACTER SET = utf8;

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

-- Table `cinema`.`hall`

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

CREATE TABLE IF NOT EXISTS `cinema`.`hall`

(

`ID` INT(11) NOT NULL AUTO\_INCREMENT,

`type` VARCHAR(20) NULL DEFAULT NULL,

`name` VARCHAR(35) NULL DEFAULT NULL,

`floor` INT(11) NULL DEFAULT NULL,

`description` TEXT NULL DEFAULT NULL,

`managerPhone` VARCHAR(17) NULL DEFAULT NULL,

`capacity` INT(11) NULL DEFAULT NULL,

PRIMARY KEY (`ID`)

)

ENGINE = InnoDB

AUTO\_INCREMENT = 10

DEFAULT CHARACTER SET = utf8;

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

-- Table `cinema`.`seance`

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

CREATE TABLE IF NOT EXISTS `cinema`.`seance`

(

`ID` INT(11) NOT NULL AUTO\_INCREMENT,

`hallID` INT(11) NULL DEFAULT NULL,

`filmID` INT(11) NULL DEFAULT NULL,

`seanceTime` TIME NOT NULL,

`seanceDate` DATE NOT NULL,

`ticketCost` DOUBLE NOT NULL,

`ticketsLeft` INT(11) NULL DEFAULT NULL,

PRIMARY KEY (`ID`),

INDEX `seance\_hall\_ID\_fk` (`hallID` ASC) VISIBLE,

INDEX `seance\_film\_ID\_fk\_idx` (`filmID` ASC) VISIBLE,

CONSTRAINT `seance\_film\_ID\_fk`

FOREIGN KEY (`filmID`)

REFERENCES `cinema`.`film` (`ID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `seance\_hall\_ID\_fk`

FOREIGN KEY (`hallID`)

REFERENCES `cinema`.`hall` (`ID`)

ON DELETE CASCADE

ON UPDATE CASCADE

)

ENGINE = InnoDB

AUTO\_INCREMENT = 51

DEFAULT CHARACTER SET = utf8;

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

-- Table `cinema`.`user`

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

CREATE TABLE IF NOT EXISTS `cinema`.`user`

(

`ID` INT(11) NOT NULL AUTO\_INCREMENT,

`login` VARCHAR(15) NOT NULL,

`password` VARCHAR(15) NOT NULL,

`role` ENUM ('User', 'Moder', 'Admin') NOT NULL,

`surname` VARCHAR(15) NULL DEFAULT NULL,

`name` VARCHAR(15) NULL DEFAULT NULL,

`email` VARCHAR(30) NULL DEFAULT NULL,

`birthday` DATE NULL DEFAULT NULL,

PRIMARY KEY (`ID`)

)

ENGINE = InnoDB

AUTO\_INCREMENT = 7

DEFAULT CHARACTER SET = utf8;

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

-- Table `cinema`.`ticket`

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

CREATE TABLE IF NOT EXISTS `cinema`.`ticket`

(

`ID` INT(11) NOT NULL AUTO\_INCREMENT,

`seanceID` INT(11) NOT NULL,

`userID` INT(11) NOT NULL,

`amount` INT(11) NOT NULL,

`cost` DOUBLE NOT NULL,

`seatNumber` VARCHAR(2000) NOT NULL,

`valid` TINYINT(1) NOT NULL,

PRIMARY KEY (`ID`),

INDEX `ticket\_seance\_ID\_fk` (`seanceID` ASC) VISIBLE,

INDEX `ticket\_user\_ID\_fk` (`userID` ASC) VISIBLE,

CONSTRAINT `ticket\_seance\_ID\_fk`

FOREIGN KEY (`seanceID`)

REFERENCES `cinema`.`seance` (`ID`),

CONSTRAINT `ticket\_user\_ID\_fk`

FOREIGN KEY (`userID`)

REFERENCES `cinema`.`user` (`ID`)

)

ENGINE = InnoDB

AUTO\_INCREMENT = 72

DEFAULT CHARACTER SET = utf8;

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

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

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