Яндекс.Лицей

Реферат на тему:

«Разработка интернет сайта для моделирования работы системы дистанционного бронирования билетов в кинотеатр»

Над проектом работали:

Коршунов Анатолий,

Олег Ролдугин,

Ксения Бондаренко

Преподаватель:

Косов А.В

Волгоград 2021

**СОДЕРЖАНИЕ**

Введение

1. Обоснование выбора среды разработки
   1. Обзор аналогов
   2. Анализ и декомпозиция предметной области
2. Проектирование
   1. Описание некоторых алгоритмов
   2. Описание реализации
3. Описание программы для пользователя
4. Заключение

Список используемых источников

Приложение

**ВВЕДЕНИЕ**

Цель проекта — разработать сайт в сети интернет, который предоставит пользователю возможность ознакомиться с фильмами, которые находятся в прокате кинотеатрах вашего города, а также оформить бронь одного или нескольких мест на сеанс выбранного кинофильма.

Процесс работы проходил в 4 основных этапа:

1. Исследование предметной области.
2. Постановка и анализ задач.
3. Проектирование.
4. Разработка.

В результате было разработан сайт «FilmCenter», соответствующий поставленным целям.

Пояснительная записка к курсовому проекту была выполнена в текстовом редакторе Microsoft Office Word 2010 и представлена в печатном виде.

|  |
| --- |
| ЗАДАНИЕ |

1. Тема проекта: Разработка интернет сайта для моделирования работы системы дистанционного бронирования билетов в кинотеатр .  
2. Срок сдачи проекта: 24.04.2021  
  
Программа состоит из:

* Пользовательского интерфейса
* Администраторского интерфейса

Программа имеет функции:

* Показ доступных фильмов.
* Поиск интересующего фильма через возможности встроенного фильтра и поисковую строку.
* Просмотр описания, трейлеров, а также изображений фильмов
* Заказ билетов на фильм
* Отправка билетов на почту.
* Добавление новых фильмов и редактирование старых.
* Ручное и автоматическое создание сессий для фильмов.

Проект представляет собой сайт, в котором пользователь может ознакомиться с доступными фильмами и выбрать интересующий его фильм или подобрать фильм по нескольким критериям (год, жанр, продолжительность, режиссер). Выбрав фильм, пользователь может узнать краткое описание и жанр, длительность, год премьеры, а также актеров, принимавших участие в фильме, и режиссера. Кроме этого пользователь может ознакомиться с несколькими кадрами из фильма и трейлером к фильму. Ознакомившись, пользователь может забронировать билет. Для этого необходимо выбрать удобную дату, доступное время на сеанс и место в зале. Перед подтверждением брони пользователю необходимо ввести адрес электронной почты, на которую будут высланы билеты. Далее пользователь может, как заказать билеты на другое время или дату, так и вернуться к списку всех доступных фильмов в прокате.

Также проект подразумевает и работу администратора с базой данных. У администратора имеется доступ ко всем использующимся таблицам в проекте. Администратор может изменять, добавлять новые и удалять старые элементы. Кроме этого администратор имеет возможность сформировать новое расписание на неделю вперед или же вручную добавить сеанс в базу.

**1. ОБОСНОВАНИЕ ВЫБОРА СРЕДЫ РАЗРАБОТКИ**

Так как программа курсового проекта должна быть реализована на языке программирования Python, то было принято решение использовать среду разработки Pycharm. Данная среда является мощным и удобным инструментом в разработке программ на языке программирования Python.

Сайт разрабатывался с использованием библиотеки Flask, позволяющей создать сервер для обработки url запросов и настроить корректную работу с апи.

Разработанный сайт был размещен на платформе Heroku, предоставляющая удаленно запускать различные приложения, и получать доступ через url ссылки. Платформа работает с PostgreSQL бд и предоставляет доступ к удаленной базе. Создана база данных была с использованием библиотеки Sqlalchemy. Добавление новых элементов осуществлялось как с использованием разработанного интерфейса администратора, так и с использованием приложения pgAdmin4, осуществляющего подключение к удаленной базе и предоставляющее инструменты для внесения изменений в бд.

**1.1 ОБЗОР АНАЛОГОВ**

Разработанный сайт имеет аналоги. Рассмотрим один из них. Сайт <https://hd.kinopoisk.ru/> реализует аналогичные функции отображения и ознакомления с фильмами. Главное окно сайта-аналога представлено на рисунке 2.1.

|  |
| --- |
|  |
| Рисунок 2.1 — Главное окно сайта-аналога |

Главными недостатками данного сайта является:

отсутствие системы заказа и получения билетов на почту;

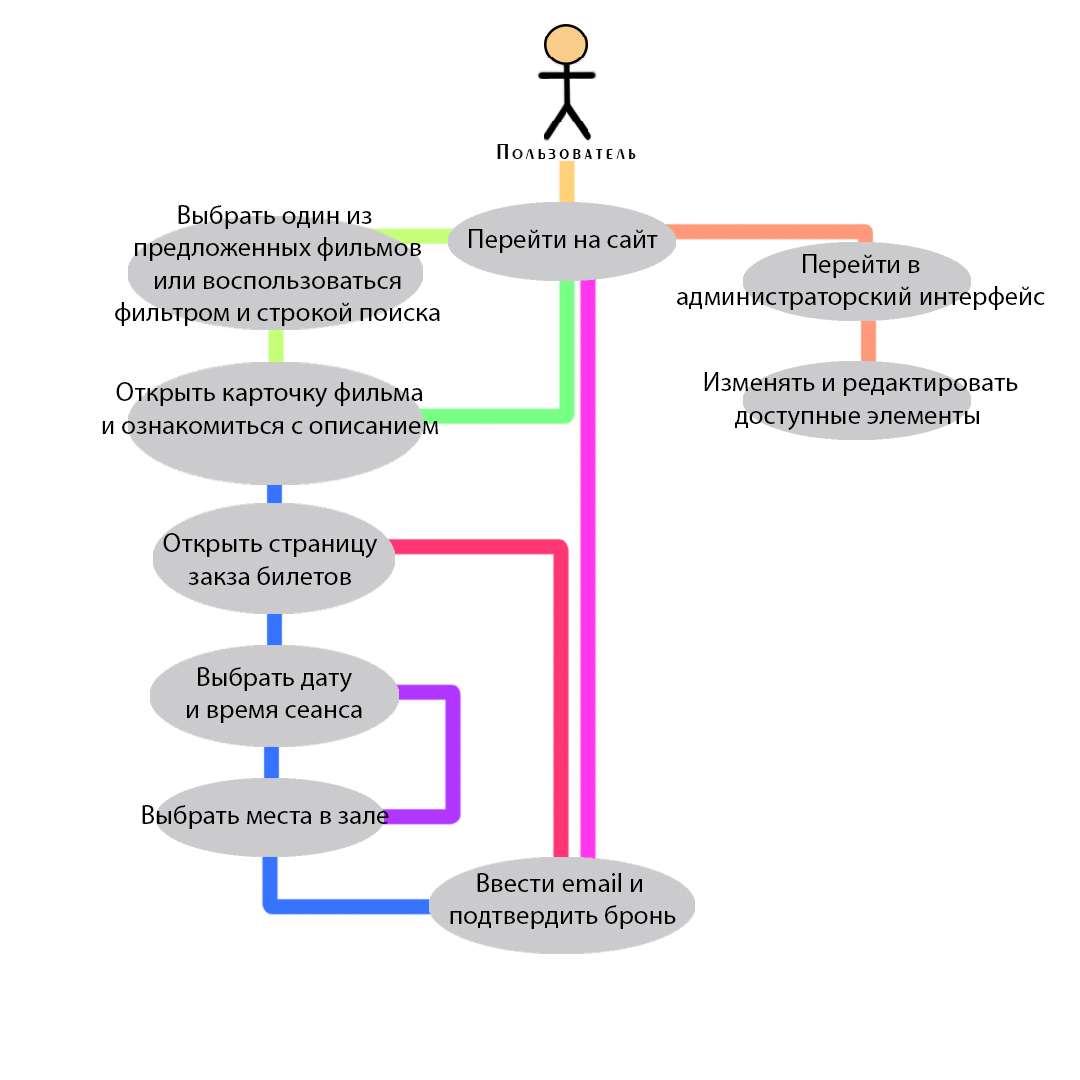
**1.2 АНАЛИЗ И ДЕКОМПОЗИЦИЯ ПРЕДМЕТНОЙ ОБЛАСТИ**

Исходя из требований к программе, были выделены основные классы:

* data.films.Film – класс, представляющий структуру таблицы films (фильмы) в бд.
* data.images.Image – класс, представляющий структуру таблицы images (картинок к фильмам) в бд.
* data.film\_sessions.FilmSession - класс, представляющий структуру таблицы film\_session (сеансов на фильмы) в бд.
* data.places.Place – класс, представляющий структуру таблицы place (заказанных мест) в бд.
* data.associations.Genre – класс, представляющий собой структуру таблицы genres (жанров) в бд.
* data.admins.AdminRole – класс, представляющий структуру таблицы админов в бд.
* api.films\_resource.FilmResource – класс апи, предоставляющий инструменты для получения определенных фильмов из бд, а также удаления и изменения существующих фильмов.
* api.films\_resource.FilmListResource – класс апи, предоставляющий инструменты для получения списка существующих фильмов и добавления новых фильмов.
* api.film\_session\_resource.FilmSessionResource - класс апи, предоставляющий инструменты для получения определенных сессий на фильм из бд, а также удаления и изменения существующих сессий.
* api.film\_session\_resource.FilmSessionListResource - класс апи, предоставляющий инструменты для получения списка существующих сессий на фильмы и добавления новых сессий.
* forms.admin.LoginForm – класс, представляющий форму для логина администратора.
* modules.tiket\_PIL.Ticket – класс, создающий новый билет и возвращающий картинку в байтах.

Для того чтобы более точно определить последовательности действий, которые может осуществлять система в ответ на внешние воздействия пользователей была составлена диаграмма вариантов использования.  
Диаграмма вариантов использования представлена на рисунке 2.2.

Рисунок 2.2 — Диаграмма вариантов использования



**2. ПРОЕКТИРОВАНИЕ**

## 2.1. Описание некоторых алгоритмов

Алгоритм подключения к базе данных.

В проекте реализовано подключение к бд с помощью библиотеки Sqlalchemy. Данная библиотека позволяет подключиться к удаленной бд и создавать сессии для каких-либо изменений в бд.

Т.к. в нашем проекте использована система автоматического очищения каждые 24 часа, то появилась проблема с незакрытыми сессиями. Закрывать сессии можно как явно, вызывая метод close() у сессии, так и неявно, используя оператор with при создании сессии, который автоматически закроет сессию после завершения работы с бд.

Итак, порядок работы с бд можно описать по следующим пунктам:  
1) При создании приложения

1.1) Создается движок sqlalchemy.create\_engine для определенного типа бд. В нашем случае для PostgerSQL. Ссылка на PostgerSQL бд выглядит примерно так: postgresql://username:password@host:port/database\_name .

1.2) Создается sessionmaker, объект, который позволяет создавать сессии для работы с бд.

2) При работе приложения

2.1)Или с помощью оператора with создается сессия или сессия создается автоматически перед тем, как начнет выполняться код, на которую вела ссылка.

2.1.1) С помощью декоратора Flask before\_request перехватывается вызов функции ссылки. Далее создается сессия с бд и объект помещается с параметр Flask.g – позволяющий переносить какие-либо объекты между областями видимости функций.

2.1.2) С помощью декоратора Flask after\_request перехватывается событие завершения выполнения кода ссылкы. Тут и закрывается сессия с бд с помощью вызова метода close() у активной сессии.

2.2) Выполняется любая задача, требующая соединения с бд. После чего коннект с бд закрывается автоматически или необходимо вызвать метод close() у активной сессии, если не был использованы варианты соединения с бд, описанные в пунктах 2.1.1 и 2.1.2.

Алгоритм создания обработки запросов от пользователя:

1) Обычные ссылки перехватываются Flask декоратором route, который позволяет соединить ссылку с функцией, обрабатывающей эту ссылку.

2) Ссылки, ведущий на апи к нашему сайту реализованы через архитектурный стиль программирования RESTful. Перехваткой запросов от пользователя занимается внутренняя реализация библиотеки flask\_restful. От нас требуется лишь создать какой либо класс ресурса и зарегистрировать его с помощью вызова Flask. add\_resource.

Кроме библиотеки flask\_restful, также был использован класс Flask. Blueprint, который также позволяется перехватывать события от пользователя через декоратор route.

Алгоритм загрузки данных в шаблоны.

В проекте использован шаблонизатор Jinja. При загрузке шаблона на html через python с использованием render\_template(args), в функцию добавляются аргументы, к которым впоследствии можно будет обращаться через синтаксис Jinja. Также Jinja предоставляет огромный набор инструментов для работы с любыми объектами python.

Алгоритм создания билетов и отправка их на почту:

1) Из шаблона через POST запрос мы получаем данные о заказанных билетах.

2) Функция make\_order() занимается обработкой данных, вызывает функцию create\_places(), а также вызывает функцию send\_mail().

3) Функция create\_places() перебирает список с выбранными местами и создает объект Place, в котором хранится информация о месте, сеансе, а также уникальный код, который позволяет идентифицировать заказанный билет.

4) Функция send\_mail() запускается в отдельном потоке с использованием библиотеки threading. Функция формирует объект письма, создает картинки билетов с использованием класса Ticket, который возвращает картинку в байтах. Далее созданные билеты добавляются во вложения к письму.

5) После подготовки объекта письма оно отправляется на указанный адрес с помощью библиотеки smtplib.

Зачастую, такие сервисы почт, как Yandex.Mail, распознают наше письмо как спам. В результате оно оказывается в папке спама. И чтобы предупредить пользователя, после заказа билета, на странице всплывает подсказка о том, что билеты отправлены, и могут оказаться в спаме.

Также хочется отметить и работу класса Ticket, который формирует картинку билета с использованием библиотеки PIL. Данный класс открывает шаблон билета, и заполняет его переданными данными. Также при заполнении используется библиотека qrcode, которая позволяет сформировать картинку и отобразить ее через инструменты PIL.

**2.3. Описание реализации**

Реализация программы проходила на основе библиотек Flask, flask\_resful, flask\_admin, Sqlalchemy.

В ходе разработки программы было принято решение реализовать ее в виде проекта с открытым исходным кодом. Так же было принято решение использовать систему контроля версий. В связи с этим было решено размещать проект на сайте <https://github.com>.

Разрабатываемый программный продукт представляет собой многостраничный сайт.

Для обработки пользовательских запросов были применены декораторы flask.route, который перехватывают запрос и вызывают ту или иную функцию. Также для добавления апи к нашему проекту, были использованы библиотеки flask\_restful и Flask.Blueprint. Перечисленные библиотеки позволяют реализовать апи взаимодействия с данными в нашем проекте. Реализованы апи как для получения, удаления и изменения какого то конкретного фильма или сессии на фильм, получения списка фильмов и сессий, так и для получения списка популярных фильмов и новых фильмов (тех, у кого дата премьеры была в текущем месяце). Также доступно апи для поиска фильмов по определенным параметрам (году, жанрам, продолжительности, названию, режиссеру).

Результат апи возвращается в формате json, с помощью вызова функции jsonify. Данная функция формирует json объект, используя сериализацию классов данных бд (которая наследуется от SerializerMixin).

Работа апи основана на библиотеке Sqlalchemy. Библиотека позволяет подключаться к базам данных и получать какие-либо данные или добавлять и изменять существующие. В проекте соединения с базой происходит при вызове create\_engine (). Последующие подключения происходят при вызове объекта sessionmaker. Объект создает подключение к бд, и с помощью query(), и filter() позволяет получать различные типы, фильтруемые по различным критериям.

Создание таблиц в бд происходил посредством формирования класса, который передается в create\_engine. Подобный класс должен наследоваться от SqlAlchemyBase. Чтобы класс был полноценной таблицей, ему нужно задать параметры, которые станут колонками в бд. Создание подобных параметров происходит с помощью sqlalchemy.Column(), куда передается какой тип данных будет хранить колонка и различные другие парамтеры.

Отрисовка шаблонов происходит с помощью вызова render\_template(). В функцию передается шаблон, который необходимо отобразить, и данные, которые будут переданы в шаблон для заполнения.

Шаблоны заполняются с использованием инструментов Jinja. Jinja позволяет использовать, как циклы и проверки в html шаблоне, так и методы у объектов python.

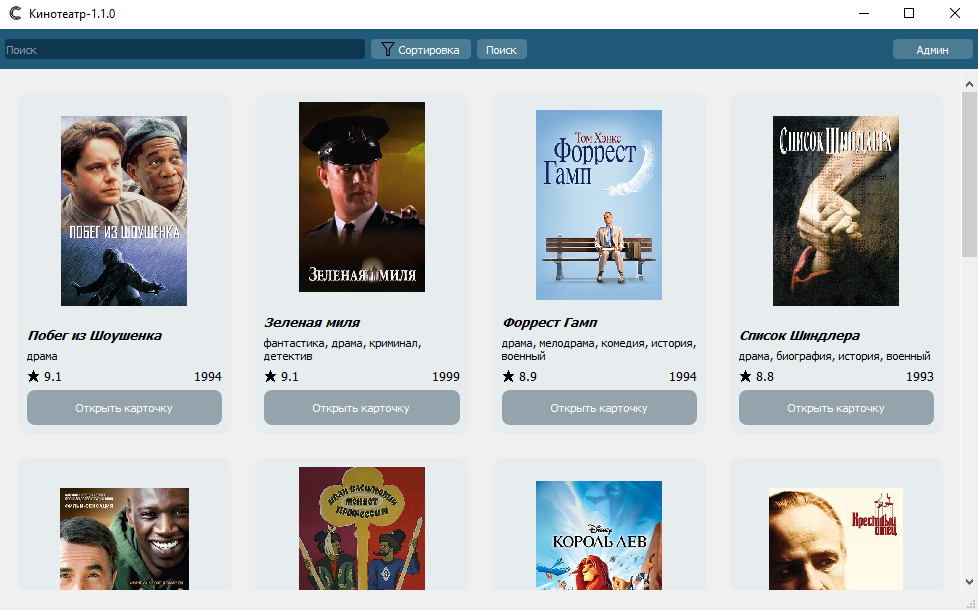
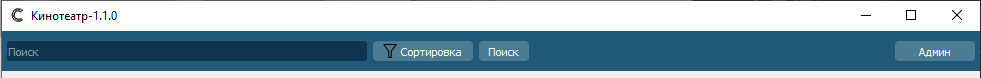
Администраторский интерфейс создан с использование flask\_admin. Библиотека позволяет использовать созданные шаблоны по работе с бд. Также имеется возможность полностью вручную настроить отображение данные в админке. Достаточно создать класс, унаследованный от ModelView из flask\_admin.contrib.sqla. flask\_admin имеет множество различных параметров, изменяя которые можно получить свой собственный, ни на что не похожий, интерфейс.

Таблицы описания классов приведены в приложении А.

**3 Описание программы для пользователя**

Проект представляет из себя компьютерное приложение, в котором пользователь может выбрать интересующий его фильм, или подобрать фильм по нескольким критериям (год, жанр, рейтинг, режиссер). Найдя подходящий фильм, пользователь может узнать краткое описание фильма, актеров, принимавших участие в фильме, режиссера, длительность. Кроме этого пользователь может просмотреть несколько постеров из фильма, а также ознакомиться с трейлером к фильму.

Ознакомившись с фильмом, пользователь может заказать билет. При заказе билета он может выбрать необходимый кинотеатр, подобрать удобное время, а также выбрать предпочтительное место в зале. Выбрав необходимые места и нажав «Подтвердить» пользователю откроются заказанные билеты, которые необходимо будет сохранить (выбрать путь для сохранения).

При запуске программы отрывается главное окно. Изображения главного окна представлено на рисунке 5.1.  
  
  
  
Рисунок 5.1 — Изображение главного окна  
  
В верхней части окна находится панель поиска с фильтром, а также с кнопкой для входа администратора. Изображение панели представлено на рисунке 5.2.  
  
  
Рисунок 5.2 — Панель.

Чтобы настроить фильтр нужно нажать кнопку «Сортировка».

Чтобы выбрать по какому параметру искать фильм необходимо выбрать нужный чекбокс, поставить в нем галочку, а также выбрать нужный элемент из выпадающего списка. Изображение окна фильтра представлено на рисунке 5.2.1

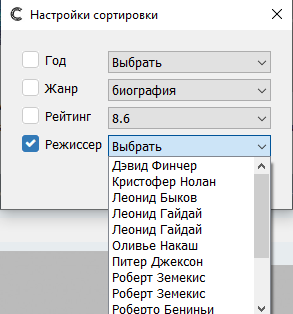


Рисунок 5.1 — Изображение фильтра

Для открытия карточки фильма необходимо нажать на кнопку «Открыть карточку».

Изображение карточки фильма представлено на рисунке 5.3.  
  
Рисунок 5.3 — Карточка фильма.

Чтобы посмотреть трейлер нужно нажать на кнопку «Смотреть трейлер».

В открытом окне представлены кнопки паузы и воспроизведения.

Изображение окна просмотра трейлера представлено на рисунке 5.3.1

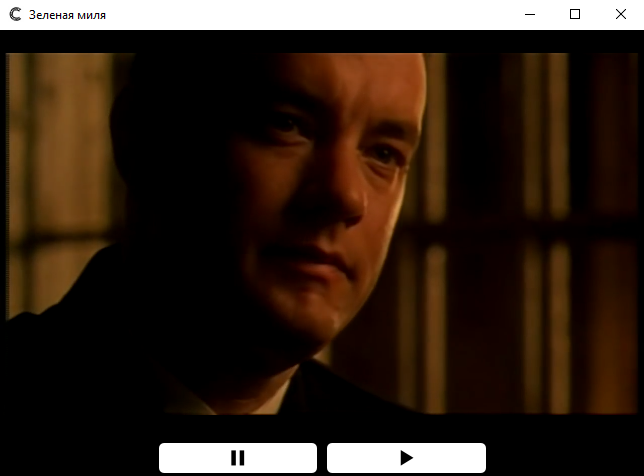


Рисунок 5.3.1 — Окно просмотра трейлера.

Чтобы увидеть увеличенное изображение необходимо нажать и удерживать на дополнительное изображение в карточке фильма.   
Изображение увеличенной картинки представлено на рисунке 5.3.2



Рисунок 5.3.2 — Увеличенная картинка.

Для покупки билета необходимо нажать «Купить билет», и заполнить все поля.

Изображение окна покупки билета представлено на рисунке 5.3.3

Изображение окна выбора места представлено на рисунке 5.3.4

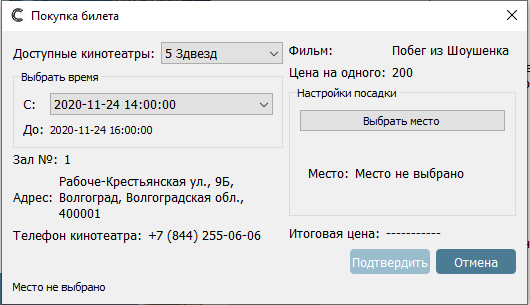


Рисунок 5.3.3 — Окно покупки билета.

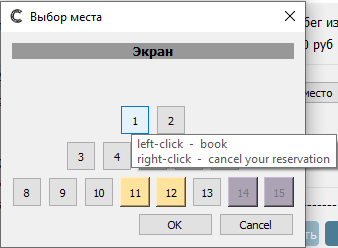


Рисунок 5.3.4 — Окно выбора места. Желтые места – выбранные. Фиолетовые – забронированные. Также изображена подсказка.

При подтверждении покупки появляется окно с купленным билетом. При нажатии кнопки сохранить появляется диалог для выбора места сохранения. После подтверждения выбора окно закроется.

Изображение билет представлено на рисунке 5.3.5

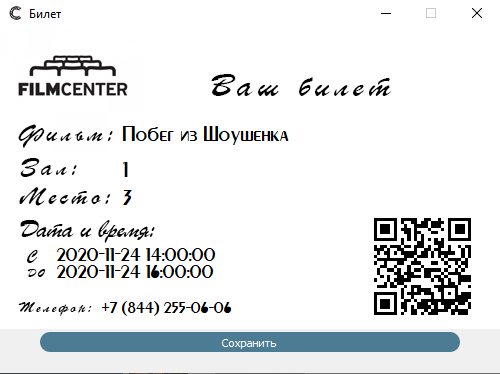


Рисунок 5.3.5 – Пример билета.

Для выхода из программы следует последовательно закрыть отрытые окна. При закрытии главного она программа всегда покажет открытые окна.

**4 Заключение**

Цель и задачи, поставленные в работе, выполнены. Создано приложение «Кинотеатр», имеющее интуитивно-понятный графический интерфейс пользователя. Для реализации была выбрана среда разработки Pycharm, библиотека PyQt5. Программа предоставляет возможность ознакомиться со множеством фильмов, а также купить билет на доступный сеанс.

**Список использованных источников**

<https://doc.qt.io/qtforpython/>

**Приложение А**(справочное)  
Таблицы описания классов

Описание класса «Database» представлено в таблице А.1.  
Таблица А.1 — описание класса «DataBase».

|  |  |
| --- | --- |
| Метод | Описание |
| \_\_init\_\_ | Создает объект и подключается к бд |
| request | Принимает строку запроса и доп.параметры. Делает запрос в бд и позвращает результат. |
| commit | Делает commit для базы |

Описание класса «WindowArr» представлено в таблице А.2.  
Таблица А.2 — описание класса «WindowArr».

|  |  |
| --- | --- |
| \_\_init\_\_ | Создает объект в котором имеется список, словарь и список для окон, которые нельзя открывать. |
| setActive | Активирует окна в списке основных окон (т.е карточки фильмов) или активирует какой-то определенный объект (объект определяется по hash-коду, который содержит родитель, т.е карточка фильма) |
| check\_for\_main\_w | Проверяет есть ли активные диалоги в гл.окне или открытые карточки фильмов. |
| сheck\_window | Проверка, является ли объект последним в списке открытых окон (потомков у карточки и у гл.окна). |
| check\_wind\_in\_list | Проверяем, если ли объект в списке окон, для удаления при открытии. Т.е если у окна есть дублер, то добавляем его в этот список у удаляем при вызове show() у объекта |
| append | Добавляет объекты в списки и в словарь. Делает проверку на дублера. |
| \_\_getitem\_\_ | Возвращает объект из списка по индексу |
| del\_item | Удаляет объект из списка или из списка и из словаря, в зависимости от класса. |

Описание класса «MyQWidget» представлено в таблице А.3.

Класс MyQDialog идентичен классу MyQWidget

Таблица А.3 — описание класса «MyQWidget».

|  |  |
| --- | --- |
| \_\_init\_\_ | Создание объекта |
| closeEvent | Делает проверку на то, является ли окно последним в списке (если потомок у основного окна(карточка фильма или гл.окно)  В зависимости от результата или закрывает окно, или игнорирует событие и активирует окна, для последовательного закрытия. |
| show | Узнает есть ли дублер, и в результате или показывает окно, или закрывает. |
| \_\_hash\_\_ | Возвращает hash родителя |

Описание класса «MainWindow» представлено в таблице А.4.  
Таблица А.4 — описание класса «MainWindow».

|  |  |
| --- | --- |
| \_\_init\_\_ | Происходит создание объекта. Подключение стилей, загрузка gui, установка размеров окна, добавление событий к кнопкам.  Вызов функций загрузки фильтра и добавлений фильмов в главное окно. |
| load\_films | Происходит запрос в базу и добавление мини-карточек в QScrollAree в главном окне. |
| make\_card\_film | Создает виждет мини-картчоку фильма |
| open\_card | Создает окно карточки фильма |
| Filter\_wind\_open | Активирует окно фильтра. После закрытия окна сортировки вызывается load\_film |
| filter\_load | Загружает информацию (жанры, года, рейтинги, режиссеров в окно сортировки |
| create\_request\_for\_filter | На основе данных из окна фильтра создает запрос в базу |
| admin\_sign\_in | Открывает окно входа для админа |
| closeEvent | Реализует тот же функционал что и у MyQWidget |
| \_\_hash\_\_ | Возвращает свой hash |

Описание класса «MyPopup» представлено в таблице А.5.  
Таблица А.5 — описание класса «MyPopup».

|  |  |
| --- | --- |
| \_\_init\_\_ | Создает объект. Настраивает размеры окна и картинки. |

Описание класса «Film» представлено в таблице А.6.  
Таблица А.6 — описание класса «Film».

|  |  |
| --- | --- |
| \_\_init\_\_ | Создание переменных для хранения данных о фильме |

Описание класса «CardOfFilm» представлено в таблице А.7.  
Таблица А.7 — описание класса «CardOfFilm».

|  |  |
| --- | --- |
| \_\_init\_\_ | Происходит создание объекта. Подключение стилей, загрузка gui, добавление событий к кнопкам и подключение собственных событий. |
| eventFilter | Событие, которое реагирует на нажатие на QLabel и открывает новое всплывающее окно. |
| resizeEvent | Событие, реагирующее на изменение размеров окна. |
| resize\_image | Метод, который увеличивал бы картику в карточке. На данный момент не реализовано. |
| buy\_ticket | Создает окно покупки билета. |
| Load\_info | Происходит запрос в базу. Загрузка всей возможной информации о фильме из базы. |
| play\_trailer | Создает окно для проигрывания трейлера. Проверят, существует ли данный файл на диске и открывает его, если он есть. |
| \_\_hash\_\_ | Возвращает свой hash код |

Описание класса «BuyTct» представлено в таблице А.8.  
Таблица А.8 — описание класса «BuyTct».

|  |  |
| --- | --- |
| \_\_init\_\_ | Происходит создание объекта. Подключение стилей, загрузка gui, добавление событий к кнопкам. |
| set\_path | Создавалась с целью автоматически сохранить билеты по определенному пути.  В данный момент не используется |
| choose\_seat | Создается окно с выбором мест в зале. Сохраняет выбранные места в переменную self.numb |
| creat\_new\_seat | Вызывается, чтобы добавить в QScrollArea выбранные места в виде строк.  Создает новую QScrollArea. Добавляет в нее элементы и помещает в gui. |
| load\_other | Закрузка времени, цены, мест в зале, номер зала из бд. |
| load\_time | Загрузка времени в выбранном кинотеатре. Телефона и адреса кинотеатра. |
| load\_cinemas | Загрузка доступных кинотеатров из бд. |
| closeEvent | Создавалось для техже целей, что и set\_path.  В данный момент не используется. |
| accept\_action | Создает окно повторного подтверждения у пользователя. Создает билеты. Сохраняет забронированные места в бд..  Блокирует интерфейс в окне, для предотвращения повторной покупки. |

Описание класса «Ticket» представлено в таблице А.9.  
Таблица А.9 — описание класса «Ticket».

|  |  |
| --- | --- |
| \_\_init\_\_ | Создание объекта. Добавление текста и Qr кода на QPixmap.  Установление событий для кнопки. |
| make\_qrcode | Создает изображение Qr кода в объекте QPixmap |
| save\_tct | Сохраняет картинку билета по выбранному пользователем пути. |
| choose\_way | Открывает диалог для выбора места для сохранения файла. Если путь корректен, то вызывается save\_tct() |
| closeEvent | Узнает выбрал ли пользователь место для сохранения картинки. |

Описание класса «MyPushButton» представлено в таблице А.10.  
Таблица А.10 — описание класса «MyPushButton».

|  |  |
| --- | --- |
| \_\_init\_\_ | Создает объект. Устанавливает флаг isSelected |
| mousePressEvent | Реагирует на нажатие правой и левой кнопками мыши по кнопке. В результате перекрашивает себя в тот или иной цвет. |

Описание класса «ChooseSeat» представлено в таблице А.11.  
Таблица А.11 — описание класса «ChooseSeat».

|  |  |
| --- | --- |
| \_\_init\_\_ | Создает объект. Устанавливает сигналы для кнопок. |
| setupUi | Загрузка gui. |
| set\_selected\_btn | При повторном открытии окна загружает ранее выбранные места. |
| c\_action | При закрытии окна сохраняет выбранные кнопки (ок) |
| set\_default\_places | При закрытии окна устанавливает ранее выбранные места (отмена) |
| get\_btn\_numb | Возвращает выбранные кнопки |

Описание класса «TrailerWidget» представлено в таблице А.12.  
Таблица А.12 — описание класса «TrailerWidget».

|  |  |
| --- | --- |
| \_\_init\_\_ | Происходит создание объекта. Подключение стилей, загрузка gui, добавление событий к кнопкам. |
| closeEvent | Перед закрытием окна воспроизведение ставится на паузу. |

Описание класса «FilterDialog» представлено в таблице А.13.  
Таблица А.13 — описание класса «FilterDialog».

|  |  |
| --- | --- |
| \_\_init\_\_ | Происходит создание объекта. Подключение стилей, загрузка gui, добавление событий к кнопкам. |
| reload\_ui | Добавление информации в QComboBox |
| accept\_data | При подтверждении окно закрывается и сохраняет выбранную информацию в словарь. |
| get\_items | Возвращает выбарнную информацию |

Описание класса «AdminSignIn» представлено в таблице А.14.  
Таблица А.14 — описание класса «AdminSignIn».

|  |  |
| --- | --- |
| \_\_init\_\_ | Происходит создание объекта. Подключение стилей, загрузка gui, добавление событий к кнопкам. |
| accept\_data | Проверяет правильность данных и в зависимости от результата или блокирует закрытие и выводит сообщение о неудаче или же открывает окно для администратора. |
| reject\_data | Закрывает окно. |

Описание класса «MyWidget» представлено в таблице А.15

Таблица А.15 — описание класса «MyWidget»

|  |  |
| --- | --- |
| \_\_init\_\_ | Загрузка gui, добавление событий к кнопкам, заполнение comboBox’ов, вызов функций обновления информации |
| update\_films | Заполнение таблицы фильмов с учётом сортировки и поиска |
| add\_film | Добавление фильма в базу данных |
| change\_film | Изменение информации о фильме |
| del\_film | Удаление фильма из базы данных |
| update\_sessions | Заполнение таблицы расписания с учётом сортировки и поиска |
| add\_session | Добавление сеанса в базу данных |
| change\_session | Изменение информации о сеансе |
| del\_session | Удаление сеанса |
| update\_cinemas | Заполнение таблицы кинотеатров с учётом сортировки и поиска |
| add\_cinema | Добавление кинотеатра в базу данных |
| change\_cinema | Изменение информации о кинотеатре |
| del\_cinema | Удаление кинотеатра |
| update\_cinema\_halls | Заполнение таблицы кинозалов с учётом сортировки и поиска |
| add\_cinema\_hall | Добавление кинозала в базу данных |
| change\_cinema\_hall | Изменение информации о кинозале |
| del\_cinema\_hall | Удаление информации о кинозале |
| \_\_hash\_\_ | Возвращает hash код, который идентифицирует окно для работы системы закрытия окон. |
| closeEvent | Перехватывает событие закрытия окна. В зависимости от имеющихся открытых окон, либо закрывает администраторское окно, либо активирует незакрытые окна. |

Описание класса «AdminSignIn» представлено в таблице А.16

Таблица А.16 — описание класса «AdminSignIn»

|  |  |
| --- | --- |
| \_\_init\_\_ | Загрузка gui, добавление событий к кнопкам |
| check\_data | Проверка введенной информации |
| reject\_data | Закрытие окна |

Описание класса «AddFilmDialog» представлено в таблице А.17

Таблица А.17 — описание класса «AddFilmDialog»

|  |  |
| --- | --- |
| \_\_init\_\_ | Загрузка gui, добавление событий к кнопкам, заполнение формы |
| check\_data | Проверка введенной информации |
| get\_items | Возвращение введенной информации |

Описание класса «AddSessionDialog» представлено в таблице А.18

Таблица А.18 — описание класса «AddSessionDialog»

|  |  |
| --- | --- |
| \_\_init\_\_ | Загрузка gui, добавление событий к кнопкам, заполнение формы |
| check\_data | Проверка введенной информации |
| get\_items | Возвращение введенной информации |

Описание класса «AddCinemaDialog» представлено в таблице А.19

Таблица А.19 — описание класса «AddCinemaDialog»

|  |  |
| --- | --- |
| \_\_init\_\_ | Загрузка gui, добавление событий к кнопкам, заполнение формы |
| check\_data | Проверка введенной информации |
| get\_items | Возвращение введенной информации |

Описание класса «AddCinemaHallDialog» представлено в таблице А.20

Таблица А.20 — описание класса «AddCinemaHallDialog»

|  |  |
| --- | --- |
| \_\_init\_\_ | Загрузка gui, добавление событий к кнопкам, заполнение формы |
| check\_data | Проверка введенной информации |
| get\_items | Возвращение введенной информации |