

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА 44

ОТЧЕТ
ЗАЩИЩЕН С
ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

Старший преподаватель

должность, уч. степень, звание

подпись, дата

Аксенов А.В.

инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ

по курсу: Базы данных

РАБОТУ ВЫПОЛНИЛ(А)

СТУДЕНТ(КА) ГР. №

В1441

группа

подпись, дата

Е.А. Зотов

инициалы, фамилия

Санкт-Петербург 2023

1 Тема курсовой работы

Справочник онлайн-сериалов

2 Словесное описание предметной области и актуальность

В наше время выходит множество сериалов. Человек, решая что-то посмотреть, испытывает проблемы с выбором, так как название того или иного сериала редко может отразить его содержание. Также у людей могут быть какие-то предпочтения по жанрам, актерам и режиссерам, или же им перед просмотром сериала важно узнать его рейтинг. Справочник онлайн-сериалов решает эту проблему.

3 Описание данных, хранящихся в базе данных

База данных должна содержать данные о:

1. Людях, которые зарегистрировались в системе в качестве пользователей или администраторов и о их действиях, совершенных на сайте.
2. Сериалах, которые были добавлены в систему и информации о них.
3. Актерах, режиссерах, сценаристах, которые принимали участие в съемках сериалов.

4 Роли пользователей приложения

1. Пользователь
2. Администратор

5 Развернутое описание функционала приложения для каждой из ролей

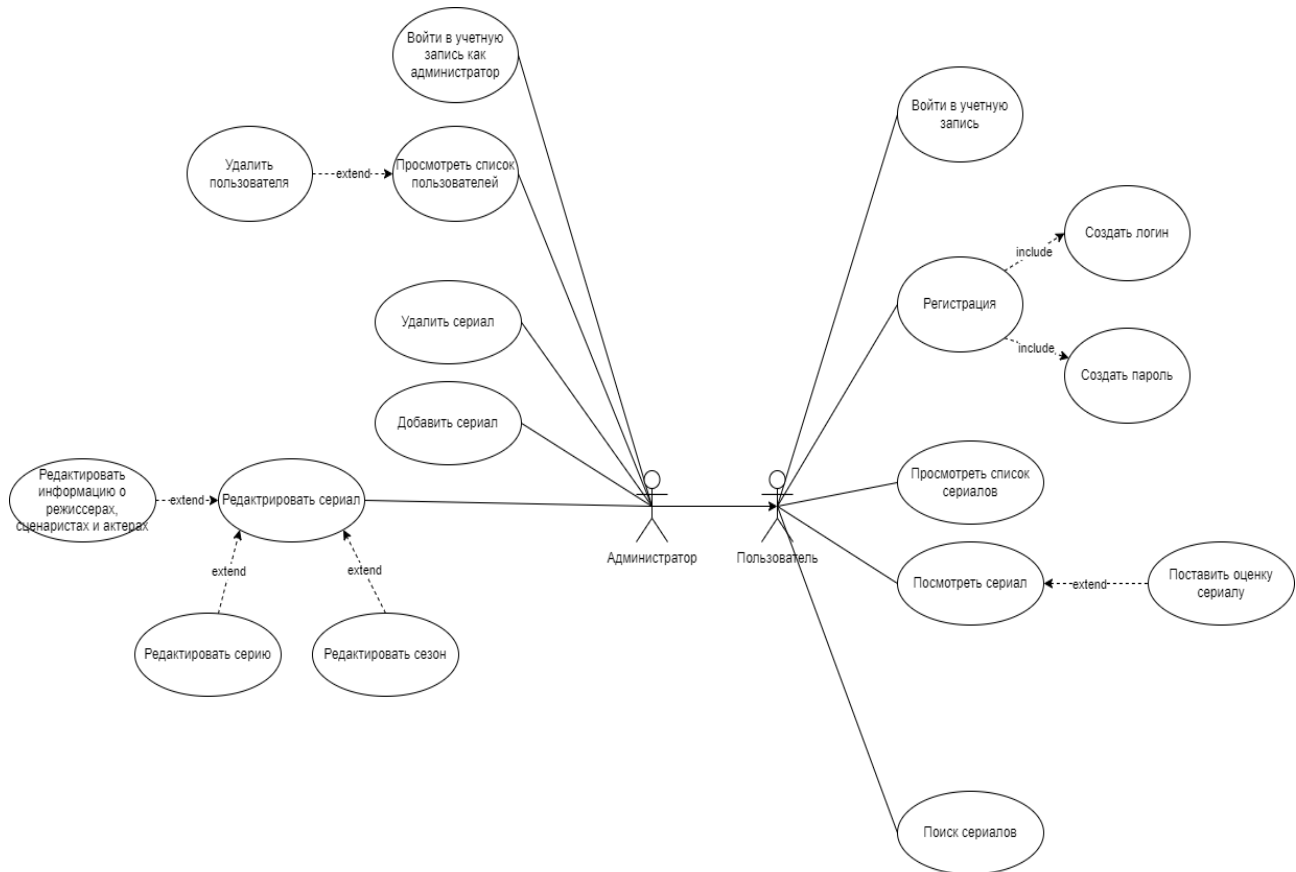
Пользователь: Пользователь может войти в систему под своей учетной записью или произвести регистрацию в случае отсутствия учетной записи. Пользователю доступен список сериалов. При переходе к конкретному сериалу, ему открывается информация о нем: название, жанр, годы выпуска, актеры, режиссеры, сценаристы, рейтинг, количество сезонов и список серий. Пользователь может добавить сериал в избранное, оставить оценку или открыть

список серий. Информация о серии содержит в себе название и продолжительность. Пользователь может отметить серию как просмотренную. Общий рейтинг сериала складывается из среднего рейтингов, оставленных пользователями.

Пользователю доступен поиск. Он может найти сериалы по названию, жанру, а также по актеру, режиссеру и сценаристу. Также он может посмотреть информацию про актеров, режиссеров и сценаристов, включающую в себя их имена, годы жизни и сериалы, в которых они принимали участие.

Администратор: Администратор имеет тот же функционал, что и пользователь, но имеет возможности редактирования данных. Администратор способен изменять список сериалов, серий сериалов, информацию о них, информацию о людях, которые участвовали в съемке сериала. Также ему доступен список зарегистрированных пользователей. Администратор может удалять учетные записи обычных пользователей.

6 Диаграмма вариантов использования



7 Предполагаемые технологии и платформа реализации

- СУБД: SQLite;
- ОС: Windows / GNU/Linux / MacOS;
- язык программирования: Python;
- фреймворк: Flask;
- тип приложения: веб-приложение.

8 Срок представления курсовой работы 01.01.2024

На рисунках 1 и 2 изображены диаграмма «Сущность-связь» и реляционная диаграмма.

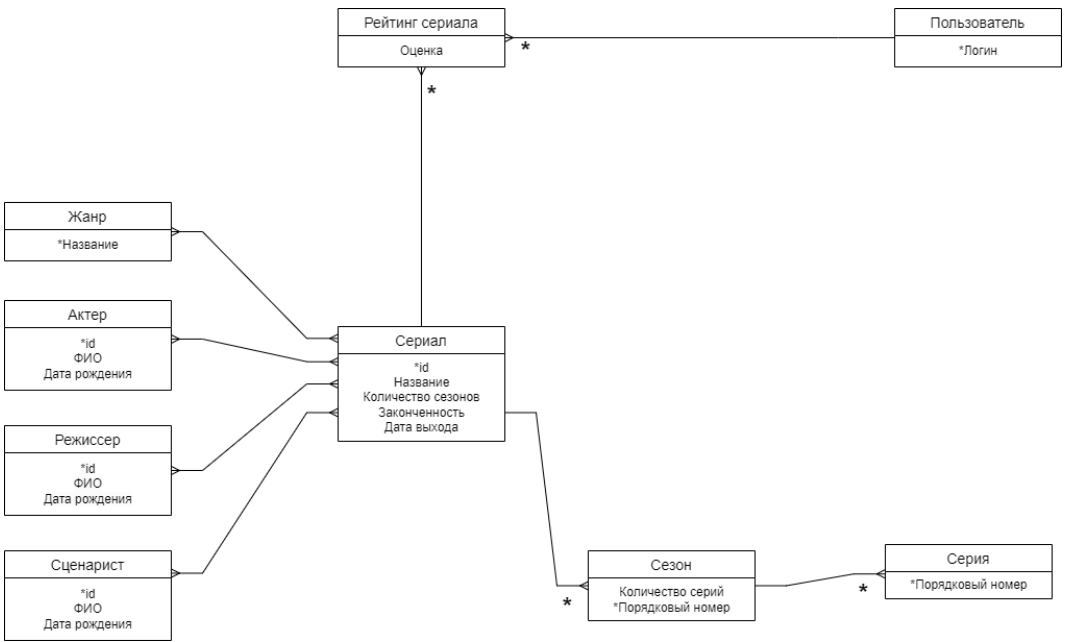


Рисунок 1 — Сущность-Связь

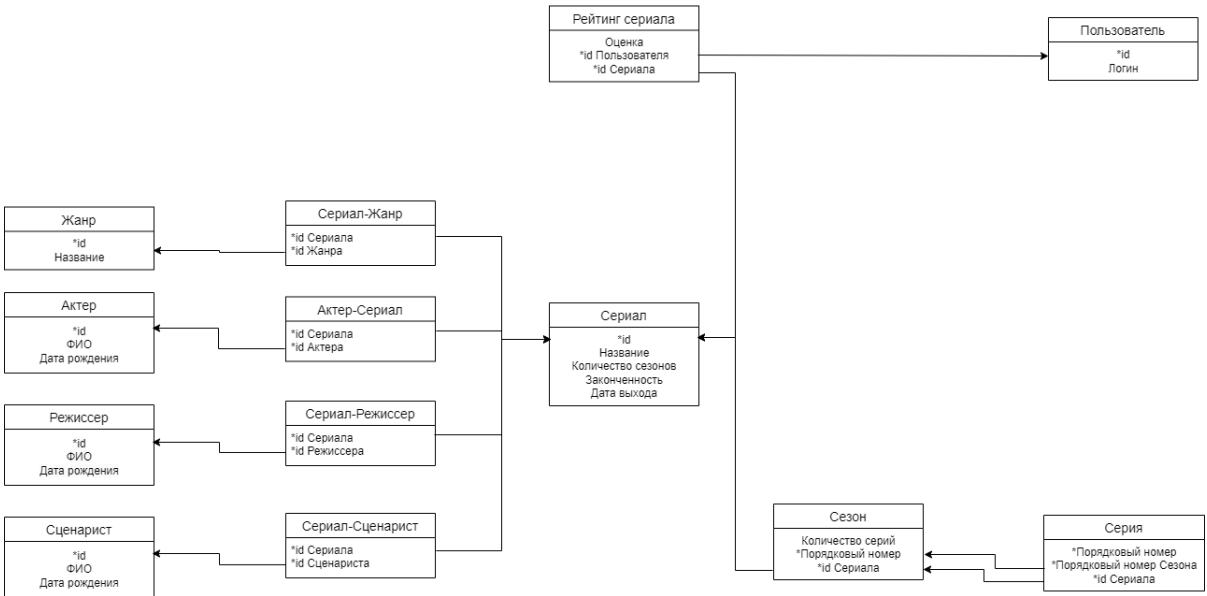


Рисунок 2 — Реляционная диаграмма

Для реализации темы курсовой работы были использованы следующие инструменты: Flask и SQLite.

Flask — фреймворк для создания веб-приложений на языке программирования Python. Относится к категории так называемых микрофреймворков. Используется для реализации логики на сайте и взаимодействия с Базой

Данных.

SQLite — компактная встраиваемая СУБД. Взаимодействие происходит при помощи Python.

База данных создается при помощи выполнения sql-кода. При запуске веб-приложения, необходимо подключиться к созданной базе данных для дальнейшего с ней взаимодействия. Далее в программе БД объявляется при помощи класса, который хранит в себе текущую базу данных и ее курсор. В python, чтобы избежать возможных ошибок, взаимодействие с БД происходит в конструкции try-except. Используемые методы: cursor.execute() для выполнения sql-кода из строки; db.commit() для сохранения измененных данных в БД; cursor.fetchone/fetchall() возвращает упорядоченный список из курсора из одного/всех взятых элементов.

Приложение запускается выполнением файла main.py на локальном сервере.

Файл main.py: Главный файл проекта, используется для запуска приложения. В нем инициализируется база данных, а также реализован функционал взаимодействия внутри приложения.

Текст программы:

```
from flask import Flask, render_template, request, g, flash, abort, redirect
from flask_login import LoginManager, login_user, login_required, logout_user, current_user
from werkzeug.security import generate_password_hash, check_password_hash
import os
import sqlite3

from data import Data
from UserLogin import UserLogin

DATABASE = "database.db"
DEBUG = True
SECRET_KEY = "abc"

app = Flask(__name__)
app.config.from_object(__name__)
```

```
app.config.update(dict(DATABASE=os.path.join(app.root_path,
"database.db")))
```

```
login_manager = LoginManager(app)
login_manager.login_view = 'login'
```

```
@login_manager.user_loader
def load_user(user_id):
    print("load user")
    return UserLogin().fromDB(user_id, dbase)
```

```
def connect_db():
    connect = sqlite3.connect(app.config['DATABASE'])
    connect.row_factory = sqlite3.Row
    return connect
```

```
def create_db():
    db = connect_db()
    with app.open_resource('database.sql', mode= 'r') as f:
        db.cursor().executescript(f.read())
    db.commit()
    db.close()
```

```
def get_db():
    if not hasattr(g, "link_db"):
        g.link_db = connect_db()
        print("БАЗА ДАННЫХ ПОЛУЧЕНА")
    return g.link_db
```

```
dbase = None
@app.before_request
def before_request():
    global dbase
    db = get_db()
    dbase = Data(db)
```

```
@app.teardown_appcontext
def close_db(error):
    if hasattr(g, 'link_db'):
        g.link_db.close()
```

```
#-----
```

```
@app.route("/")
def index():
    db = get_db()
    return render_template("index.html")
```

```
@app.route("/about")
```

```

def about():
    return render_template("about.html")

@app.route("/all_series", methods=["POST", "GET"])
def all_series():
    db = get_db()
    dbase = Data(db)
    return render_template("all_series.html", posts = dbase.get_posts())

@app.route("/series/<int:id_series>", methods=["POST", "GET"])
def series(id_series):
    title, seasons, finished, release = dbase.get_series(id_series)
    series_genres = dbase.get_genre_from_series(id_series)
    series_actors = dbase.get_actors_from_series(id_series)
    series_directors = dbase.get_directors_from_series(id_series)
    series_writers = dbase.get_writers_from_series(id_series)
    series_seasons = dbase.get_season(id_series)

    print(series_genres)

    if request.method == "POST":

        if "rating" in request.form:
            user_id = current_user.get_id()
            dbase.add_rating(user_id, id_series, request.form["rating"])

        if "genre" in request.form:
            dbase.add_genre_to_series(id_series, request.form["genre"])

        if "actor" in request.form:
            dbase.add_actor_to_series(id_series, request.form["actor"])

        if "director" in request.form:
            dbase.add_director_to_series(id_series,
request.form["director"])

        if "writer" in request.form:
            dbase.add_writer_to_series(id_series, request.form["writer"])

        if "create_season" in request.form:
            print("add_season")
            dbase.add_season(request.form["season_title"], id_series,
request.form["order_number"], request.form["episodes_number"])

    return render_template("series.html", id_series = id_series, posts =
dbase.get_posts(), title = title, seasons = seasons,
finished = finished, release = release,
rating = dbase.get_rating(id_series), series_genres = series_genres,
series_actors = series_actors,
series_directors = series_directors, series_writers = series_writers,

```



```

series_seasons = series_seasons,
                genres = dbase.get_genres(),
                actors = dbase.get_actors(), directors =
dbase.get_directors(), writers = dbase.get_writers())

@app.route("/add_series", methods=["POST", "GET"])
@login_required
def add_series():

    if request.method == "POST":
        res = dbase.add(request.form['title'], request.form['seasons'],
request.form['finished'], request.form['release'])
        flash("Сериал успешно добавлен")
        return render_template("add_series.html", genres =
dbase.get_genres(), actors = dbase.get_actors(), directors =
dbase.get_directors(), writers = dbase.get_writers())

@app.route("/login", methods=["POST", "GET"])
def login():
    if request.method == "POST":
        user = dbase.get_user_by_nickname(request.form['nickname'])
        if user and check_password_hash(user['psw'],
request.form['psw']):
            user_login = UserLogin().create(user)
            login_user(user_login)
            return redirect("/")
        return render_template("login.html")

@app.route("/register", methods=["POST", "GET"])
def register():
    if request.method == "POST":
        if len(request.form["nickname"]) > 4 and len(request.form["psw"])
> 4:
            hash = generate_password_hash(request.form["psw"])
            res = dbase.add_user(request.form['nickname'], hash)
            print("HASH", hash)
            if res: return redirect("/login")
            else:
                print("ОШИБКА РЕГИСТРАЦИИ")

        return render_template("register.html")

@app.route("/profile")
@login_required
def profile():
    return render_template("profile.html", nickname =
current_user.get_nickname(), id = current_user.get_id())

@app.route("/logout")

```

```

@login_required
def logout():
    logout_user()
    return redirect("/login")

@app.route("/config")
@login_required
def config():
    return render_template("config.html")

@app.route("/config/add_genre", methods = ["POST", "GET"])
def add_genre():
    if request.method == "POST":
        res = dbase.add_genre(request.form['title'])

    return render_template("add_genre.html")

@app.route("/config/add_human", methods = ["POST", "GET"])
def add_human():
    if request.method == "POST":
        dbase.add_human(request.form["fullname"],
request.form["birthday"], request.form["select"])
    return render_template("add_human.html")

create_db()
if __name__ == "__main__":
    app.run(debug = True)

```

Файл data.py. Состоит из одного класса, в котором методами реализовано взаимодействие с базой данных: внесение и получение данных.

Текст программы:

```

import sqlite3
from flask import request, flash

class Data:
    def __init__(self, db):
        self.__db = db
        self.__cur = db.cursor()

    def add(self, title, seasons, finished, release):
        try:
            self.__cur.execute("INSERT INTO series
VALUES(NULL, ?, ?, ?, ?)", (title, seasons, finished, release))
            self.__db.commit()

        except sqlite3.Error as e:
            print("Ошибка добавления сериала" + str(e))
            return False
        return True

```

```

def get_series(self, id):
    try:
        self.__cur.execute(f"SELECT title, seasons, finished, release
FROM series WHERE id = {id} LIMIT 1")
        res = self.__cur.fetchone()
        print(res)
        if res:
            return res
    except sqlite3.Error as e:
        print("Ошибка получения сериала " + str(e))

    return(False, False)

def get_series_id(self, id):
    try:
        self.__cur.execute(f"SELECT id FROM series WHERE id = {id}
LIMIT 1")
        res = self.__cur.fetchone()
        print(res)
        if res:
            return res
    except sqlite3.Error as e:
        print("Ошибка получения id сериала " + str(e))

def get_posts(self):
    try:
        self.__cur.execute("SELECT id, title FROM series ORDER BY
title DESC")
        res = self.__cur.fetchall()
        if res: return res
    except sqlite3.Error as e:
        print("Ошибка получения поста" + str(e))

    return []

def add_user(self, nickname, hash):
    try:
        self.__cur.execute(f"SELECT COUNT() as 'count' FROM user
WHERE nickname LIKE '{nickname}' ")
        res = self.__cur.fetchone()
        if res['count'] > 0:
            print("ПОЛЬЗОВАТЕЛЬ С ТАКИМ ИМЕНЕМ УЖЕ СУЩЕСТВУЕТ")

        self.__cur.execute("INSERT INTO user VALUES(NULL, ?, ?)",
(nickname, hash))
        self.__db.commit()

    except sqlite3.Error as e:
        print("Ошибка добавления юзера" + str(e))
        return False
    return True

```

```

def get_user(self, user_id):
    try:
        self.__cur.execute(f"SELECT * FROM user WHERE id = {user_id}
LIMIT 1")
        res = self.__cur.fetchone()
        if not res:
            print("ПОЛЬЗОВАТЕЛЬ НЕ НАЙДЕН")
            return False
        return res

    except sqlite3.Error as e:
        print("Ошибка получения юзера" + str(e))

    return False

def get_user_by_nickname(self, nickname):
    try:
        self.__cur.execute(f"SELECT * FROM user WHERE nickname =
'{nickname}' LIMIT 1")
        res = self.__cur.fetchone()
        if not res:
            print("Пользователь не найден")
            return False
        return res
    except sqlite3.Error as e:
        print("Ошибка получения юзера" + str(e))

    return False

def add_genre(self, name):
    try:
        self.__cur.execute(f"SELECT COUNT() as 'count' FROM genre
WHERE title LIKE '{name}' ")
        res = self.__cur.fetchone()
        if res['count'] > 0:
            flash("ЖАНР С ТАКИМ ИМЕНЕМ УЖЕ СУЩЕСТВУЕТ")

        else:
            self.__cur.execute("INSERT INTO genre VALUES(NULL, ?)",
(name,))
            self.__db.commit()
            flash("Жанр успешно добавлен")

    except sqlite3.Error as e:
        print("Ошибка добавления жанра " + str(e))

def add_genre_to_series(self, id_series, genre):
    try:
        genre_title = genre

```

```

        self.__cur.execute(f"SELECT id FROM genre WHERE title =
'{genre}' ")
        genre_id = self.__cur.fetchone()
        self.__cur.execute("INSERT INTO series_genre
VALUES(?, ?, ?)", (id_series, genre_id[0], genre_title))
        self.__db.commit()
        flash("Жанр успешно добавлен")

    except sqlite3.Error as e:
        print("Ошибка добавления жанра " + str(e))

    def get_genre_from_series(self, series_id):
        try:
            self.__cur.execute(f"SELECT genre_title FROM series_genre
WHERE id_series = {series_id}")
            s_genres = self.__cur.fetchall()

            return s_genres
        except sqlite3.Error as e:
            print("Ошибка получения жанра " + str(e))

    def add_human(self, fullname, birthday, job):
        if job == "Актер":
            try:
                self.__cur.execute("INSERT INTO actor
VALUES(NULL, ?, ?)", (fullname, birthday))
                self.__db.commit()
                flash("Актер успешно добавлен")

            except sqlite3.Error as e:
                print("Ошибка добавления актера " + str(e))

        if job == "Режиссер":
            try:
                self.__cur.execute("INSERT INTO director
VALUES(NULL, ?, ?)", (fullname, birthday))
                self.__db.commit()
                flash("Режиссер успешно добавлен")

            except sqlite3.Error as e:
                print("Ошибка добавления режиссера " + str(e))

        if job == "Сценарист":
            try:
                self.__cur.execute("INSERT INTO writer
VALUES(NULL, ?, ?)", (fullname, birthday))
                self.__db.commit()
                flash("Сценарист успешно добавлен")

            except sqlite3.Error as e:
                print("Ошибка добавления сценариста " + str(e))

```

```

def get_genres(self):
    try:
        self.__cur.execute("SELECT title FROM genre ORDER BY title
DESC")
        genres = self.__cur.fetchall()
        if genres:
            return genres

    except sqlite3.Error as e:
        print("Ошибка получения Жанра " + str(e))

def get_actors(self):
    try:
        self.__cur.execute("SELECT fullname FROM actor ORDER BY
fullname DESC")
        actors = self.__cur.fetchall()
        if actors:
            return actors

    except sqlite3.Error as e:
        print("Ошибка получения актера " + str(e))

def get_directors(self):
    try:
        self.__cur.execute("SELECT fullname FROM director ORDER
BY fullname DESC")
        directors = self.__cur.fetchall()

        return directors

    except sqlite3.Error as e:
        print("Ошибка получения режиссера " + str(e))

def get_writers(self):
    try:
        self.__cur.execute("SELECT fullname FROM writer ORDER BY
fullname DESC")
        writers = self.__cur.fetchall()
        if writers:
            return writers

    except sqlite3.Error as e:
        print("Ошибка получения сценариста " + str(e))

def add_actor_to_series(self, id_series, actor):
    try:
        actor_name = actor
        self.__cur.execute(f"SELECT id FROM actor WHERE fullname =
'{actor}' ")
        actor_id = self.__cur.fetchone()

```

```

        self.__cur.execute("INSERT INTO series_actor
VALUES(?, ?, ?)", (id_series, actor_id[0], actor_name))
        self.__db.commit()
        flash("Актер успешно добавлен")

    except sqlite3.Error as e:
        print("Ошибка добавления Актер " + str(e))

    def get_actors_from_series(self, series_id):
        try:
            self.__cur.execute(f"SELECT actor_name FROM series_actor
WHERE id_series = {series_id}")
            actors = self.__cur.fetchall()
            if actors:
                return actors

        except sqlite3.Error as e:
            print("Ошибка получения актера " + str(e))

    def add_director_to_series(self, id_series, director):
        try:
            director_name = director
            self.__cur.execute(f"SELECT id FROM director WHERE fullname =
'{director}'")
            director_id = self.__cur.fetchone()
            self.__cur.execute("INSERT INTO series_director
VALUES(?, ?, ?)", (id_series, director_id[0], director_name))
            self.__db.commit()
            flash("Режиссер успешно добавлен")

        except sqlite3.Error as e:
            print("Ошибка добавления Режиссера " + str(e))

    def add_writer_to_series(self, id_series, writer):
        try:
            writer_name = writer
            self.__cur.execute(f"SELECT id FROM writer WHERE fullname =
'{writer}'")
            writer_id = self.__cur.fetchone()
            self.__cur.execute("INSERT INTO series_writer
VALUES(?, ?, ?)", (id_series, writer_id[0], writer_name))
            self.__db.commit()
            flash("Сценарист успешно добавлен")

        except sqlite3.Error as e:
            print("Ошибка добавления Сценариста " + str(e))

    def get_directors_from_series(self, series_id):
        try:
            self.__cur.execute(f"SELECT director_name FROM
series_director WHERE id_series = {series_id}")

```

```

        directors = self.__cur.fetchall()
        if directors:
            return directors

    except sqlite3.Error as e:
        print("Ошибка получения режиссера " + str(e))

    def get_writers_from_series(self, series_id):
        try:
            self.__cur.execute(f"SELECT writer_name FROM series_writer
WHERE id_series = {series_id}")
            writers = self.__cur.fetchall()
            if writers:
                return writers

        except sqlite3.Error as e:
            print("Ошибка получения сценариста " + str(e))

    def add_rating(self, user, sereies_id, rating):
        try:
            self.__cur.execute("INSERT INTO series_rating
VALUES(?, ?, ?)", (sereies_id, user, rating))
            self.__db.commit()
            flash("Вы оставили оценку")

        except sqlite3.Error as e:
            print("Ошибка добавления рейтинга " + str(e))
            if str(e) == "UNIQUE constraint failed:
series_rating.id_user, series_rating.id_series":
                flash("Вы уже оставили оценку.")

    def get_rating(self, series_id):
        try:
            self.__cur.execute(f"SELECT AVG(rating) FROM series_rating
WHERE id_series = {series_id}")
            rating = self.__cur.fetchone()
            if rating:
                return rating[0]

        except sqlite3.Error as e:
            print("Ошибка получения сценариста " + str(e))

    def add_season(self, title, series_id, order_number,
episodes_number):
        try:
            self.__cur.execute(f"INSERT INTO season VALUES (?, ?, ?, ?)",
(title, series_id, order_number, episodes_number))
            self.__db.commit()

        except sqlite3.Error as e:

```



```

        print("Ошибка добавления сезона " + str(e))

    def get_season(self, series_id):
        try:
            self.__cur.execute(f"SELECT title, order_number,
episodes_number FROM season WHERE id_series = {series_id} ORDER BY
order_number ")
            series_seasons = self.__cur.fetchall()
            if series_seasons:
                return series_seasons

        except sqlite3.Error as e:
            print("Ошибка получения сезона " + str(e))

```

Файл UserLogin.py. Нужен для использования модуля flask-login. Состоит из одного класса, позволяет взаимодействовать с текущим пользователем, авторизованным на сайте.

```

class UserLogin():
    def fromDB(self, user_id, db):
        self.__user = db.get_user(user_id)
        return self

    def create(self, user):
        self.__user = user
        return self

    def is_authenticated(self):
        return True

    def is_active(self):
        return True

    def is_anonymous(self):
        return False

    def get_id(self):
        return str(self.__user['id'])

    def get_nickname(self):
        return str(self.__user["nickname"])

```

database.sql, sql-код, создающий таблицы в базе данных.

```
CREATE TABLE IF NOT EXISTS user
(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nickname TEXT NOT NULL,
    psw TEXT NOT NULL
);
```

```
CREATE TABLE IF NOT EXISTS series
(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    title TEXT NOT NULL,
    seasons INTEGER NOT NULL,
    finished BOOLEAN NOT NULL,
    release DATE NOT NULL
);
```

```
CREATE TABLE IF NOT EXISTS genre
(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    title TEXT NOT NULL
);
```

```
CREATE TABLE IF NOT EXISTS actor
(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    fullname TEXT NOT NULL,
    birthday DATE NOT NULL
);
```

```
CREATE TABLE IF NOT EXISTS director
(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    fullname TEXT NOT NULL,
    birthday DATE NOT NULL
);
```

```
CREATE TABLE IF NOT EXISTS writer
(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    fullname TEXT NOT NULL,
    birthday DATE NOT NULL
);
```

```
CREATE TABLE IF NOT EXISTS series_rating
(
    id_series INTEGER,
    id_user INTEGER,
    rating INTEGER,
```

```
    PRIMARY KEY(id_user, id_series),
    FOREIGN KEY(id_series) REFERENCES series(id) ON UPDATE CASCADE,
    FOREIGN KEY(id_user) REFERENCES user(id) ON UPDATE CASCADE
);
```

```
CREATE TABLE IF NOT EXISTS season
(
    title TEXT,
    id_series INT,
    order_number INT,
    episodes_number INT,
    PRIMARY KEY (order_number, id_series),
    FOREIGN KEY (id_series) REFERENCES series(id) ON UPDATE CASCADE
);
```

```
CREATE TABLE IF NOT EXISTS episode
(
    id_series VARCHAR(20),
    season_order_number INT,
    order_number INT,
    PRIMARY KEY(order_number, season_order_number, id_series),
    FOREIGN KEY (season_order_number, id_series) REFERENCES
season(order_number, id_series)
);
```

```
CREATE TABLE IF NOT EXISTS series_genre
(
    id_series INTEGER,
    id_genre INTEGER,
    genre_title TEXT,

    PRIMARY KEY(id_genre, id_series),
    FOREIGN KEY(id_series) REFERENCES series(id) ON UPDATE CASCADE,
    FOREIGN KEY(id_genre) REFERENCES genre(id) ON UPDATE CASCADE
);
```

```
CREATE TABLE IF NOT EXISTS series_actor
(
    id_series INTEGER,
    id_actor INTEGER,
    actor_name TEXT,

    PRIMARY KEY(id_actor, id_series),
    FOREIGN KEY(id_series) REFERENCES series(id) ON UPDATE CASCADE,
    FOREIGN KEY(id_actor) REFERENCES actor(id) ON UPDATE CASCADE
);
```

```
CREATE TABLE IF NOT EXISTS series_director
(
    id_series INTEGER,
    id_director INTEGER,
```

```

director_name TEXT,

PRIMARY KEY(id_director, id_series),
FOREIGN KEY(id_series) REFERENCES series(id) ON UPDATE CASCADE,
FOREIGN KEY(id_director) REFERENCES director(id) ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS series_writer
(
    id_series INTEGER,
    id_writer INTEGER,
    writer_name TEXT,

    PRIMARY KEY(id_writer, id_series),
    FOREIGN KEY(id_series) REFERENCES series(id) ON UPDATE CASCADE,
    FOREIGN KEY(id_writer) REFERENCES writer(id) ON UPDATE CASCADE
);

```

Описание сценариев использования приложения

На главной странице представлено несколько функций. Без регистрации функционал ограничен.

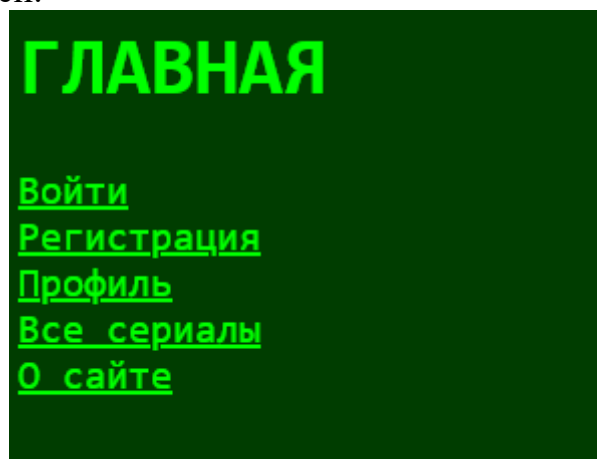


Рисунок 3 — Главная страница сайта

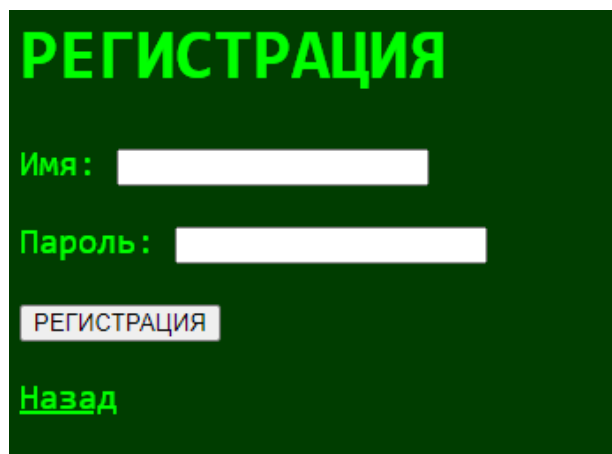


Рисунок 4 — Окно регистрации

После входа главная страница меняется: исчезают поля с логином и регистрацией. Если пользователь авторизовался как администратор, ему становятся доступны функции добавления сериала и других настроек.

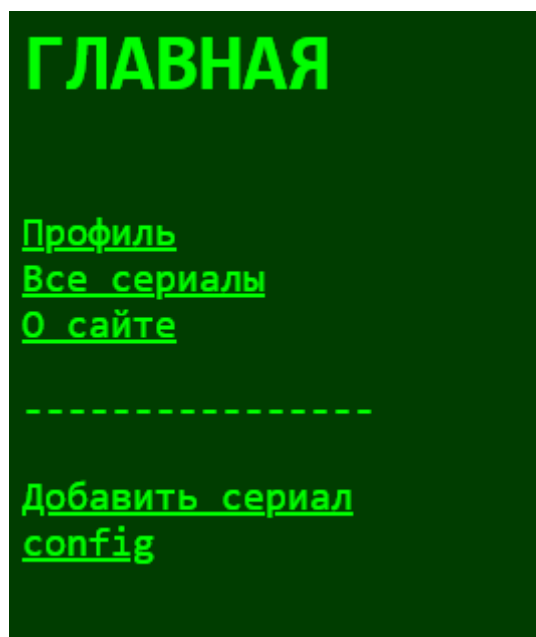


Рисунок 5 — Главная страница

На странице «Добавить сериал» можно ввести название сериала, количество сезонов, дату выхода и завершился ли сериал. После чего полученная информация заносится в базу данных, а сам сериал будет доступен на странице «Все сериалы».

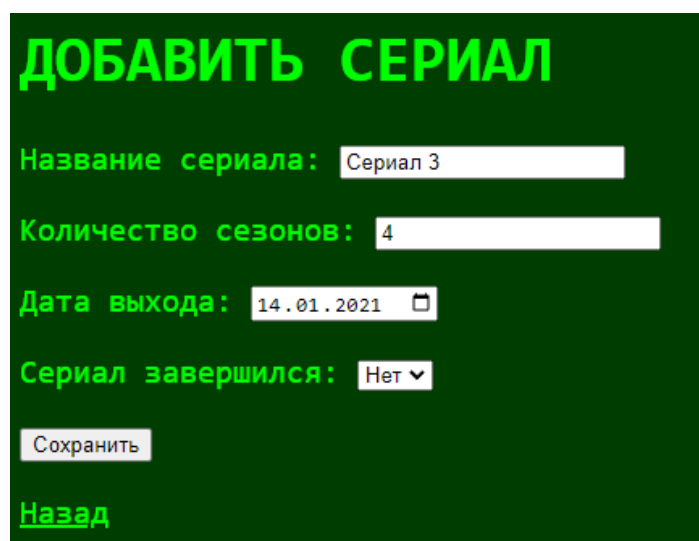
The image shows a dark-themed form titled "ДОБАВИТЬ СЕРИАЛ" in large, bold, light blue letters. Below the title are four rows of input fields, each with a label in light blue and a corresponding input area. The first row is "Название сериала:" followed by a text box containing "Сериал 3". The second row is "Количество сезонов:" followed by a text box containing "4". The third row is "Дата выхода:" followed by a date picker showing "14.01.2021" and a calendar icon. The fourth row is "Сериал завершился:" followed by a dropdown menu showing "Нет" and a downward arrow. Below these fields is a light blue button labeled "Сохранить". At the bottom left, there is a light blue link labeled "Назад".

Рисунок 6 — Добавить сериал

На странице config можно добавить жанр, либо человека, который принимал участие в сериале: актер, режиссер или сценарист. После добавления на странице конкретного сериала администратор может внести эти данные в информацию о сериале.

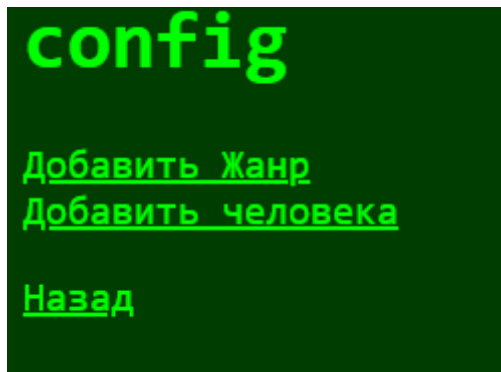


Рисунок 7 — Config

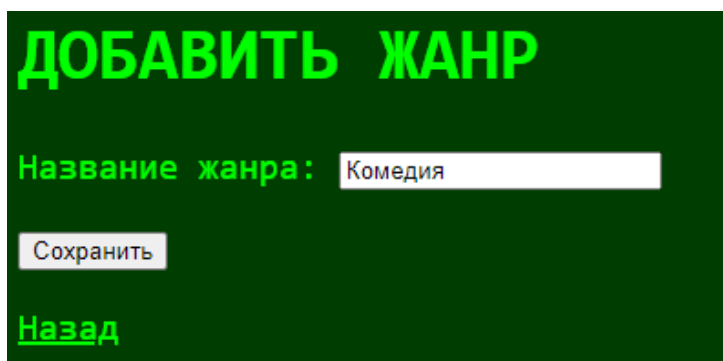


Рисунок 8 — Добавить жанр

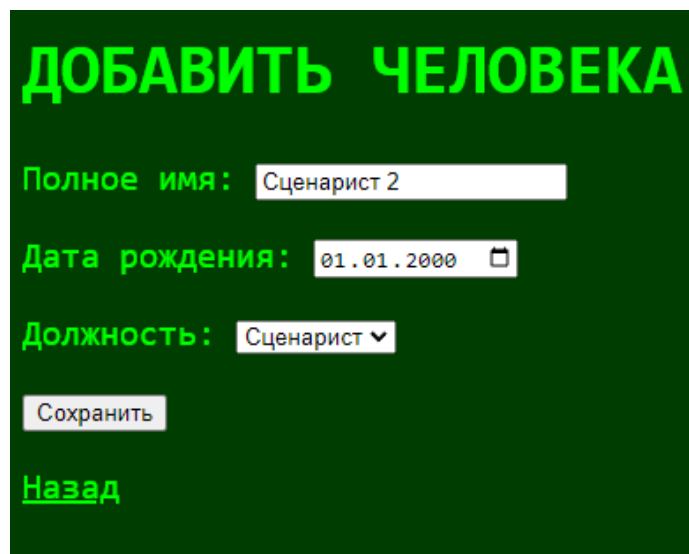


Рисунок 9 — Добавить человека

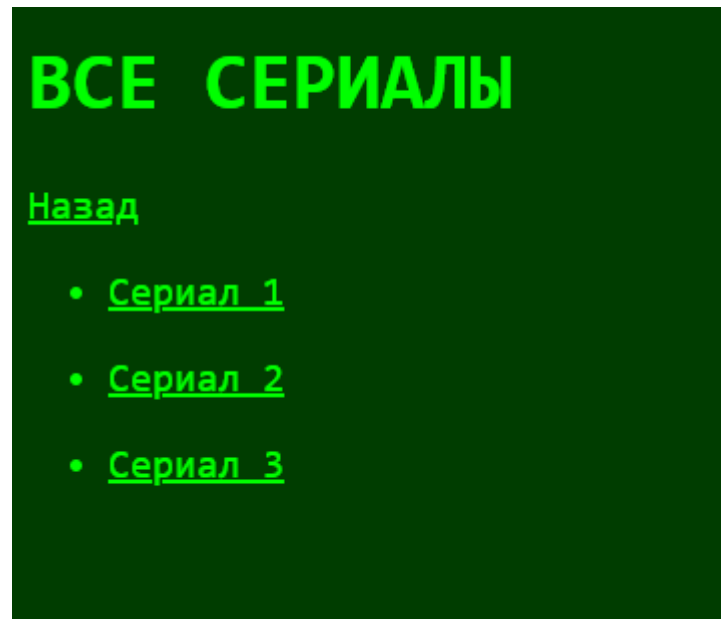


Рисунок 10 — Все сериалы

На странице конкретного сериала отображается информация о нем. Зарегистрированные пользователи могут поставить оценку сериалу, после чего отобразится средняя всех оценок. Администратор может добавить сериалу жанры, актеров, режиссеров и сценаристов, а также добавить сезоны и информацию о них.



Рисунок 11 — Страница сериала

Название: Сериал 3
Количество сезонов: 4
Закончен: Нет
Дата выхода: 2023-12-01
Рейтинг: 4.0

Жанры: Комедия, Фантастика,

Актеры: Актер 2, Актер 3,

Режиссеры: Режиссер 1,

Сценаристы: Сценарист 2,

Оценить: 1 ▾

Отправить

Добавить Жанр

Фантастика ▾

Сохранить

Добавить актера

Актер 3 ▾

Сохранить

Добавить Режиссера

Режиссер 3 ▾

Сохранить

Добавить Сценариста

Сценарист 2 ▾

Сохранить

[Назад](#)

Сезоны:
1. Первый сезон, 5 серий
2. Второй сезон, 6 серий
3. Третий сезон , 7 серий

Создание сезонов:
Название:

text

Порядковый номер:
Количество серий:

Создать сезон

Рисунок 12 — Страница сериала после заполнения

Вывод: В результате выполнения лабораторной работы было создано веб-приложение на языке Python с использованием фреймворка Flask и СУБД SQLite по теме «Справочник онлайн-сериалов».