APPLIED DATA SCIENCE WITH PYTHON

PROJECT ASSIGNMENT: TASK 3
HOLLYWOOD ACTORS AND ACTRESSES

January 10, 2025

Yevhenii Chernenko
Faculty of Computer Science and Mathematics
OTH Regensburg

Contents

1 Brief description

Project description: In this project, you need to create a user-friendly software that stores and extracts information about the top 50 popular Hollywood actor and actresses from this **list**.

1.1 Software functionality

The web application provides users with detailed information about actors and actresses listed within the platform. This includes key biographical details, awards, preferred genres, overall ratings, and a selection of their top movies. Additionally, the application features a list of the top 100 movies.

1.2 Tools

- **IMDb website** and **list** of actors to look through. Possibly, **Rapid API** to obtain additional information about films and actors. As well **Cinemagoer** package to get some additional information.
- **Python** as a main programming language for the project. **Jinja** - as templating engine. **Pandas** - to work with data.
- **SQLAlchemy** and **PostgreSQL** to store and access data.
- **PyCharm** - IDE (Integrated Development Editor) that allows to work with Python projects, create virtual environments for them.

1.3 Modules

The application will be structured into the following main components, each with distinct responsibilities to ensure modularity and maintainability:

- Interface: represents the user's interaction point with the application. Sends requests to the application and receives templates (HTML pages);
- Back-end application:handles incoming requests from the user browser. Processes these requests, communicates databases, and sends back rendered templates or error responses;

- PostgreSQL: the primary database store. Stores persistent data (Actor, Movie, and Award records);
- Redis: a cache database. Stores data frequently accessed to reduce load on PostgreSQL and improve application speed.

1.4 Data Structures

- Python Basic Structures: List, Tuple, Dictionary, Set, String.
- JSON and CSV for data processing.
- Pandas structures: throughout whole project it is planned to use main data structures in Pandas that are Series and DataFrame (2-dimensional array or table)
- Custom models for project presented in database: Actor, Movie, Award (description presented in Appendix B)

1.5 High-level design

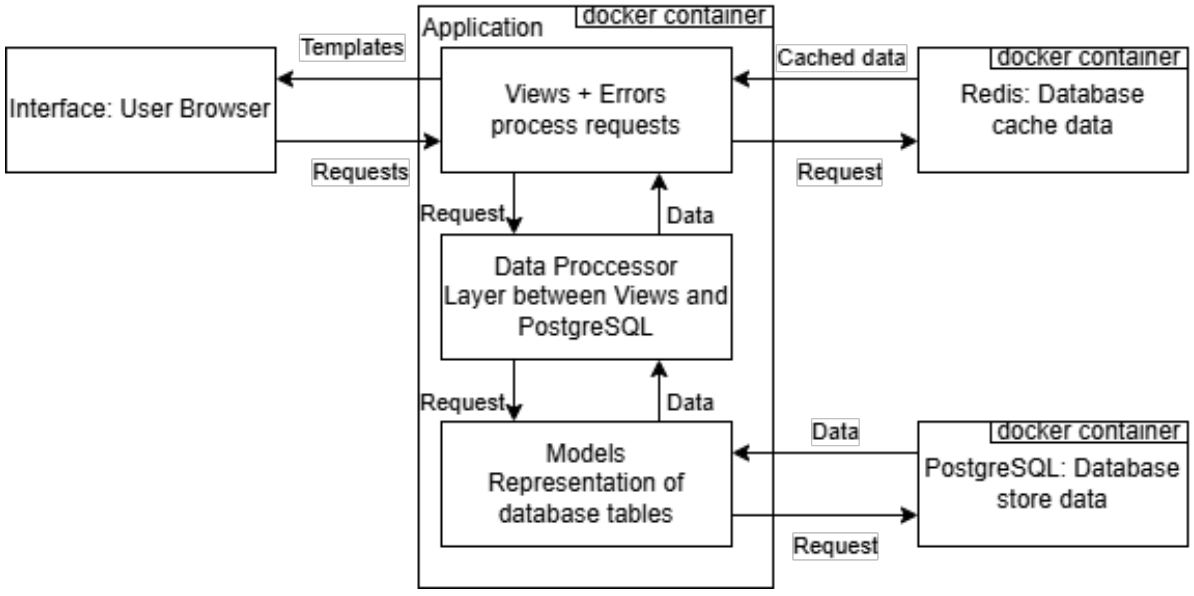

Figure 1: Design of application

2 Partial results

Pseudo Code

As a partial result of project I would like to describe algorithms for further implementation of application features. Here I use db as a hypothetic library for the connection to the database.

```
1. function actor_list():
    actors_query = db.Actor.get()
    actors = Array()
    FOR actor IN actors_query:
        actors.ADD(actor)
    return actors
```

```
2. function actor_info(const):
    actor = db.Actor.get(const)
    return actor
```

```
3. function all_time_movie_list():
    movies = db.Movie.get()
    data = Array()
    FOR movie IN movies:
        data.ADD(movie)
    return data.TOP(100)
```

```
4. function awards_info(const):
    awards = db.Award.get(const=const)
    awards_filt = {}
    FOR award IN awards:
        awards_filt[award.Year] +=
                        award.Name
    return SORT(awards_filtered)
```

```
5. function movie_genres(const):
    movies = db.Movie.get(const=const)
    genres = Array()
    FOR movie IN movies:
        genres.ADD(movie.Genre)
    return genres.COUNT().TOP(5)
```

```
6. function movie_rating(const):
    movies = db.Movie.get(const=const)
    data = Array()
    FOR movie IN movies:
        data.ADD({Year: movie.Year,
                Rating: movie.Rating})
    overall_rating = data.MEAN("Rating")
    yearly_ratings = data.GROUPBY("Year")
                .MEAN("Rating")
    return {"Overall": overall_rating,
                "Yearly": yearly_ratings}
```

```
7. function movie_top(const):
    movies = db.Movie.get(const=const)
    data = Array()
    FOR movie IN movies:
        data.ADD(movie)
    return data.TOP(5)
```

3 Project implementation

3.1 Functions description

Here JSON presented as Python dictionaries.

#	Name	Input Data	Output Data
1	actor_list	None	List of JSON with main information about actors.
2	actor_info	const (str): IMDb actor identifier	JSON with more detailed actor information.
3	all_time_ movie_list	None	List of top 100 movies.
4	awards_info	const (str): IMDb actor identifier	JSON with total awards and a list of awards by year.
5	movie_genres	const (str): IMDb actor identifier	JSON of top 5 genres for an actor.
6	movie_rating	const (str): IMDb actor identifier	JSON with overall average rating and yearly average ratings.
7	movie_top	const (str): IMDb actor identifier	List of JSON with top 5 movies.

Table 1: Function descriptions for the project.

3.2 Examples of output

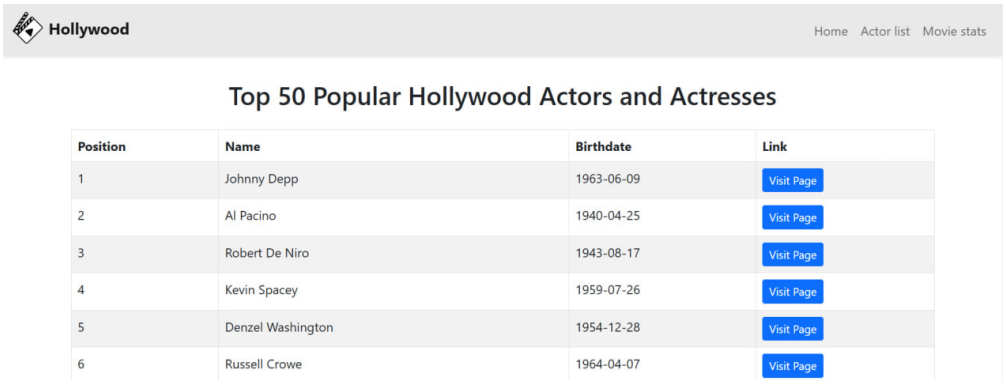

Figure 2: List of actors

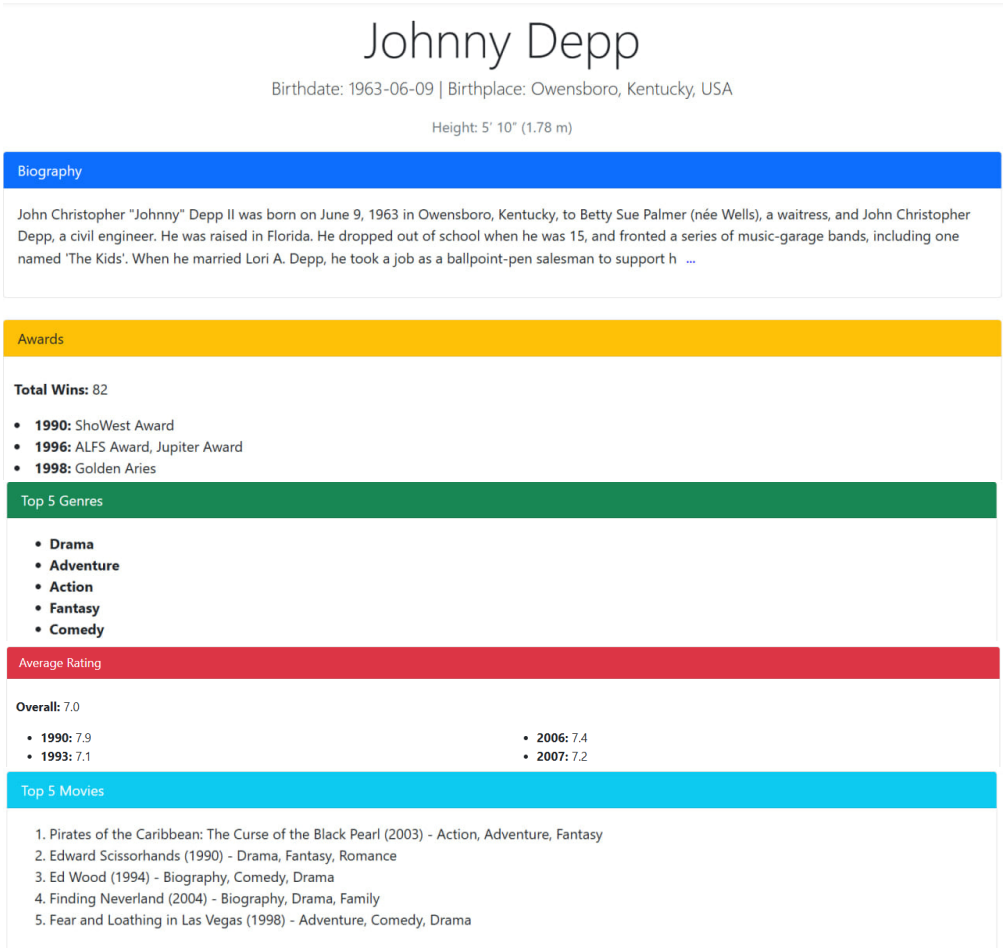

Figure 3: Actor page example

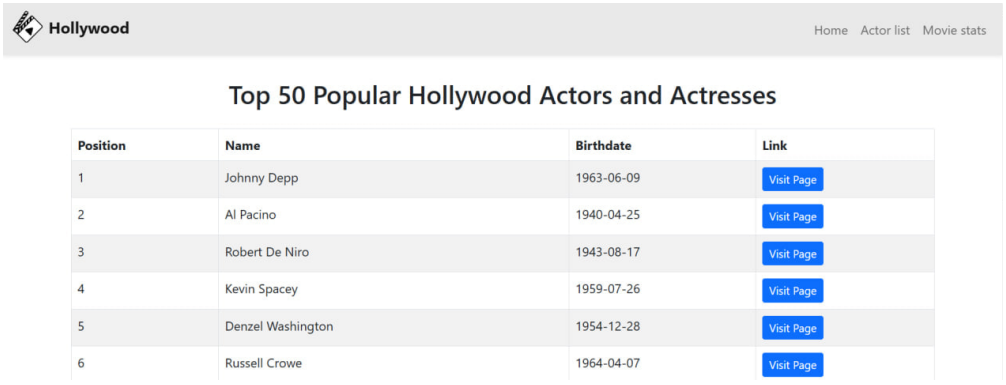

Figure 4: Movie statistics

A User guide to setup application

Manual Setup With Docker Compose

1. **Clone the Repository**:

   ```
   git clone https://github.com/malighters/AppliedDataScience
   cd AppliedDataScience
   ```

2. **Start Docker Compose**:

   ```
   docker-compose up --build
   ```

3. **Access the Application**: Visit `http://localhost:5000`.

4. **Stop the Application**:

   ```
   docker-compose down
   ```

Manual Setup Without Docker Compose

1. **Install Python Dependencies**: Ensure you have Python 3.x and `pip` installed, then run:

   ```
   pip install -r requirements.txt
   ```

2. **Set Up PostgreSQL**:

   - Install PostgreSQL and create a new database:

     ```
     CREATE DATABASE postgres;
     ```

   - Execute the SQL file to set up the database schema and seed data:

     ```
     psql -U yourusername -d postgres -f actor_app/data/data.sql
     ```

3. **Set Up Redis**: Install Redis, start it, and ensure it's running on port 6379.

4. **Run the Application**:

   ```
   python app.py
   ```

5. **Access the Application**: Open `http://localhost:5000` in your browser.

B Model description

The database comprises three primary models: `Actor`, `Movie`, and `Award`.

Attribute	Description
Const	The unique identifier for the actor.
Name	The name of the actor.
BirthDate	The birth date of the actor.
BirthLocation	The birth location of the actor.
Height	The height of the actor.
Bio	The biography of the actor.
OfficialLinks	The official links related to the actor.

Table 2: Attributes and relationships of the Actor model.

Attribute	Description
Tconst	The unique identifier for the movie.
Const	The unique identifier for the related actor (foreign key).
Name	The name of the movie.
Year	The release year of the movie.
Rating	The rating of the movie.
Genre	The genre of the movie.

Table 3: Attributes of the Movie model.

Attribute	Description
Id	The unique identifier for the award.
Const	The unique identifier for the related actor (foreign key).
Name	The name of the award.
Year	The year the award was received.

Table 4: Attributes of the Award model.

C Issues occurred during the project

During the developing of the project, several challenges were encountered that affected the project structure and the integrity of the data. Below, we discuss these issues in more detail:

Problems with IMDb API Subscription

Firstly, it was planned to use IMDb API for getting data for the project. Then, the request for API was declined, so some parsers and open APIs were used for collecting a dataset that was used further to create the database.

CSV Read Data

Another significant issue involved reading data from CSV files. It was planned to do full interaction through CSV files but then it turned out that to get data from CSV file the application needs to upload the table fully (it is not a problem for 1000+ records, but if we get a million records dataset, it will become a huge problem). So it was decided to change the main source of data to database.

Social Links Absent for Some Actors

One of the problems faced during the project was the absence of social media links for some actors. These links were expected to be fetched from external APIs, but not all actors had their social media profiles listed or available through the sources we were using. To resolve this issue, if actor/actrees does not have social links, it will be shown 'No social links available'.

Error Handling

It is important to ensure the stable operation of the application, so during the development, we worked out the following cases. For example, when users try to get an actor who is unavailable in database, they will get a message that they want to access non-existent page. Also, there were some situations when some fields were unavailable for some actors, in this case users get information that such information is absent.

D Performance in comparison

Tests were conducted to verify the effectiveness of using a database instead of reading CSV files. Subsequently, additional tests were performed to evaluate the effectiveness of caching. Now we cannot see much difference in time between the CSV and the DB times because the datasets contain not more than 4,000 records. If we get a million records or more like in the original IMDb DB, the time difference will become much bigger. Also, we can see that caching is paramount when some data will be used often as it works much quicker than straight access + calculation.

#	Name	CSV Time (s)	DB Time (s)	Cache Time (s)
1	Katherine Heigl	0.0385	0.4047	0.0021
2	Gerard Butler	0.0495	0.0313	0.0021
3	Al Pacino	0.0481	0.0337	0.0010
4	Leonardo DiCaprio	0.0583	0.0359	0.0009
5	Russell Crowe	0.0562	0.0313	0.0010
6	Mel Gibson	0.0493	0.0358	0.0024
7	Robert Downey Jr.	0.0676	0.0329	0.0010
8	Megan Fox	0.0509	0.0243	0.0010
9	Vin Diesel	0.0483	0.0236	0.0010
10	Sean Connery	0.0488	0.0277	0.0012

Table 5: Performance metrics for actors (CSV, DB, and Cache times).

Average (CSV/DB) = 1.8 times and Average (DB/Cache) = 25 times