**CS673 Software Engineering** 

**Team 3 - Movie Data Visualization Platform**

**Software Design Document**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Xuansheng Xia | Team Leader & Design and Implementation Leader | *Xuansheng Xia* | 09/26/2024 |
| Xiaojuan Li | Requirement Leader | *Xiaojuan Li* | 09/26/2024 |
| Hengyi Song | Security Leader | *Hengyi Song* | 09/26/2024 |
| Ziliang Ren | QA Leader | *Ziliang Ren* | 09/26/2024 |
| Mingyuan Xu | Configuration Leader | *Mingyuan Xu* | 09/26/2024 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| **Iteration 1** | **Xuansheng Xia, Xiaojuan Li, Hengyi Song, Ziliang Ren, Mingyuan Xu** | **10/16/2024** |  |
|  |  |  |  |

[Introduction](#_heading=h.gjdgxs)

[Software Architecture](#_heading=h.30j0zll)

[Class Diagram](#_heading=h.1fob9te)

[UI Design (if applicable)](#_heading=h.3znysh7)

[Database Design (if applicable)](#_heading=h.2et92p0)

[Security Design](#_heading=h.tyjcwt)

[Business Logic and/or Key Algorithms](#_heading=h.3dy6vkm)

[Design Patterns](#_heading=h.1t3h5sf)

[Any Additional Topics you would like to include.](#_heading=h.2s8eyo1)

[References](#_heading=h.17dp8vu)

[Glossary](#_heading=h.3rdcrjn)

# Introduction - Xiaojuan Li

This Software Design Document (SDD) outlines the design and implementation of a web-based platform for analyzing and visualizing movie data sourced from IMDb. The document serves to provide an overview of the system's structure and design principles, with a focus on creating a scalable, efficient, and user-friendly platform.

The project's objective is to build a platform that enables users to process and visualize large movie datasets, facilitating the identification of trends and insights within the film industry. This platform is intended for users such as movie enthusiasts, data analysts, film critics, and researchers who seek to explore movie releases, ratings, genres, and user reviews. Key features include interactive visualizations of movie release trends by year, genre, and production country, sentiment analysis on user reviews, and word clouds generated from frequently used terms in reviews.

To guarantee simple deployment and scalability, the system's architecture is constructed on a Docker-based environment. The database, frontend, and backend components will all be encapsulated in Docker containers, facilitating smooth integration and uniform runtime environments. Python and the Django framework will power the backend development, handling database interactions and server-side logic management. HTML, CSS, and JavaScript will be used to construct the frontend, and ECharts will be used to produce interactive and visually compelling data visualizations. IMDb movie data and related information will be stored in a MySQL database, managed within Docker containers. Pandas will be employed for data processing, and Git will be used for version control, with GitHub hosting the project and supporting collaboration.

This document further details the system's design goals, key architectural components, and implementation strategies, ensuring the platform delivers a robust and flexible solution that meets the needs of its users.

# Software Architecture - Xuansheng Xia

1. **Overview**

The software architecture for our project is based on the Django web framework, employing a Model-View-Controller (MVC) architectural pattern. The system is decomposed into several components, each residing in its own package or folder to maintain modularity and facilitate ease of maintenance and scalability.

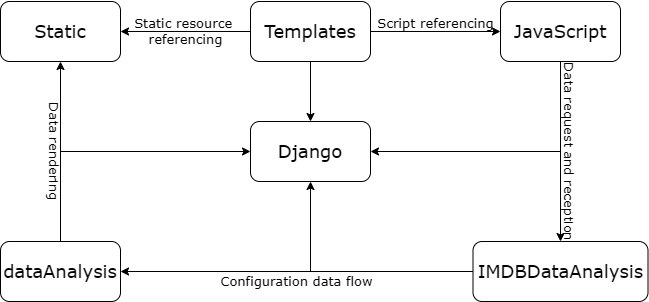
1. **Component Decomposition**
   1. Django Applications:

* **dataAnalysis**: This package is central to the backend logic, handling data processing and analysis. It interacts with the database to perform data fetching, manipulation, and complex analytical operations. It is considered a standalone component due to its specific responsibility for data operations.
* **IMDBDataAnalysis**: Manages configuration settings for the entire Django project, including settings, URL routing, and WSGI/ASGI applications which are critical for web server communication.
  1. Static Files and Templates:
* **static**: This directory houses CSS files for styling, JavaScript files for frontend functionalities, and images. It acts as a repository for all client-side assets.
* **templates**: Contains HTML files that define the structural layout of web pages, which are crucial for the view component of the MVC pattern.
  1. JavaScript Modules:
* The JavaScript files within the **static** directory are crucial for the dynamic visualization of data using ECharts and for enhancing user interaction with the frontend. These scripts are tightly coupled with the HTML templates to render interactive data visualizations.

1. **Relationships and Dependencies**
   1. **dataAnalysis** and **IMDBDataAnalysis**:

* **dataAnalysis** depends on **IMDBDataAnalysis** for central configurations such as database settings, installed applications, and middleware configurations.
  1. **Backend (Django)** and **Frontend (Static/Templates)** Interaction:
* **dataAnalysis** views use the HTML templates from the **templates** directory to render data to the user. The templates link to JavaScript and CSS files located in the **static** directory, which add styling and interactivity to the web pages.
* The JavaScript files make asynchronous calls (AJAX) to the backend to fetch or submit data without needing to reload the web page, creating a dynamic user experience.
  1. JavaScript and External Libraries:
* The JavaScript files utilize jQuery for DOM manipulations and AJAX calls, and ECharts for generating complex graphical data representations. These scripts depend on the respective libraries, which are included via CDN or local files linked in the HTML templates.

1. **Frameworks Used**
   1. **Django**: A high-level Python web framework that encourages rapid development and clean, pragmatic design. It is used to handle request routing, database operations, and server-side logic.
   2. **ECharts**: A powerful, interactive charting and visualization library for browser environments. ECharts is used for rendering dynamic data visualizations.
2. **Architecture Diagram**



# Class Diagram - Xuansheng Xia

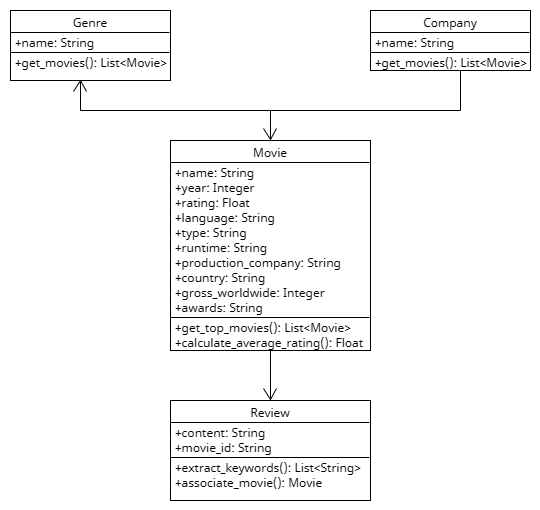
1. **Overview**

A class diagram will represent the structure of the ***dataAnalysis*** application, highlighting the main classes, their attributes, and operations, along with the relationships between these classes such as inheritance and associations.

1. **Components**
   1. **movie:**

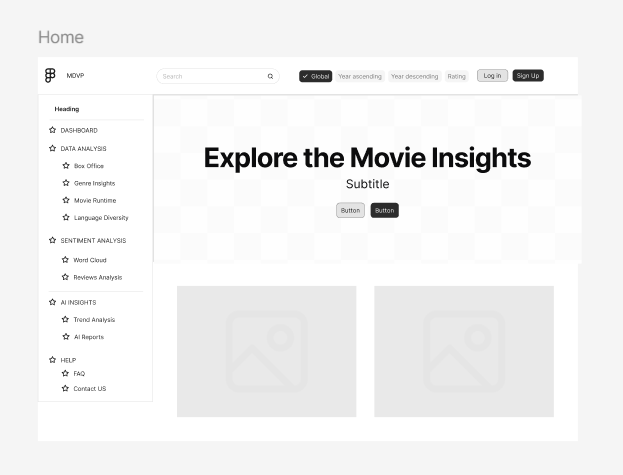
* **Attributes:**
  + name: String
  + year: Integer
  + rating: Float
  + language: String
  + type: String
  + runtime: String
  + production\_company: String
  + country: String
  + gross\_worldwide: Integer
  + awards: String
* **Methods:**
  + get\_top\_movies(): Returns a list of top-rated movies.
  + calculate\_average\_rating(): Calculates the average rating for movies.
  1. **Genre:**
* **Attributes**:
  + name: String
* **templates**:
  + get\_movies(): Returns all movies under a genre.
  1. **Company**:
* **Attributes**:
  + name: String
* **templates**:
  + get\_movies(): Returns all movies produced by the company.
  1. **Reviews:**
* **Attributes:**
  + content: Text (String) - The text content of the review.
  + movie\_id: varchar(255) - A reference to the Movie class, indicating which movie the review is about.
* **Methods:**
  + extract\_keywords(): Returns the most frequent words from the review content for word cloud generation.
  + associate\_movie(): Links the review to its corresponding movie, facilitating data aggregation and analysis.

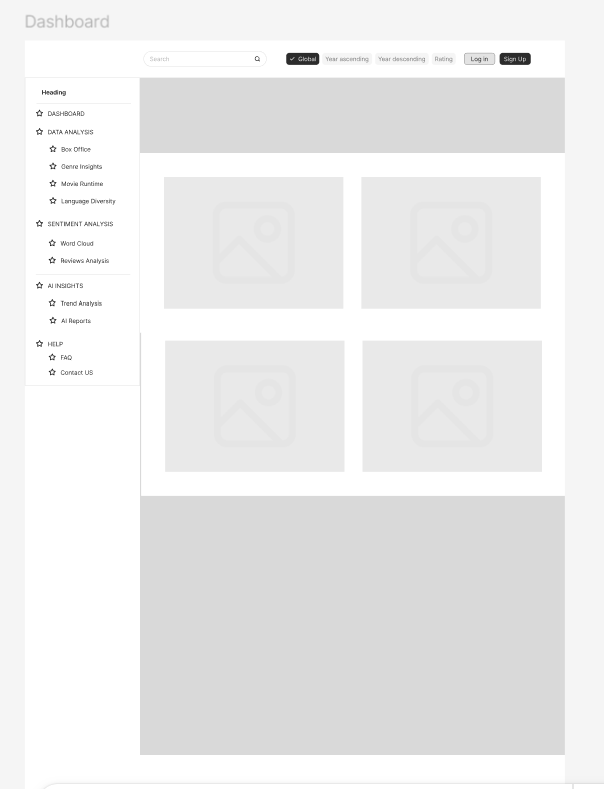
1. **Relationships**
   1. **Movie to Genre**: Many-to-Many (A movie can have multiple genres and a genre can include multiple movies.)
   2. **Movie to Company**: Many-to-One (Multiple movies can be produced by a single company.)
   3. **Review to Movie**: Many-to-One (Multiple reviews can be associated with a single movie). This is critical for aggregating data to produce meaningful insights such as sentiment analysis or frequency analysis of terms across reviews for a particular film.
2. **Diagram**



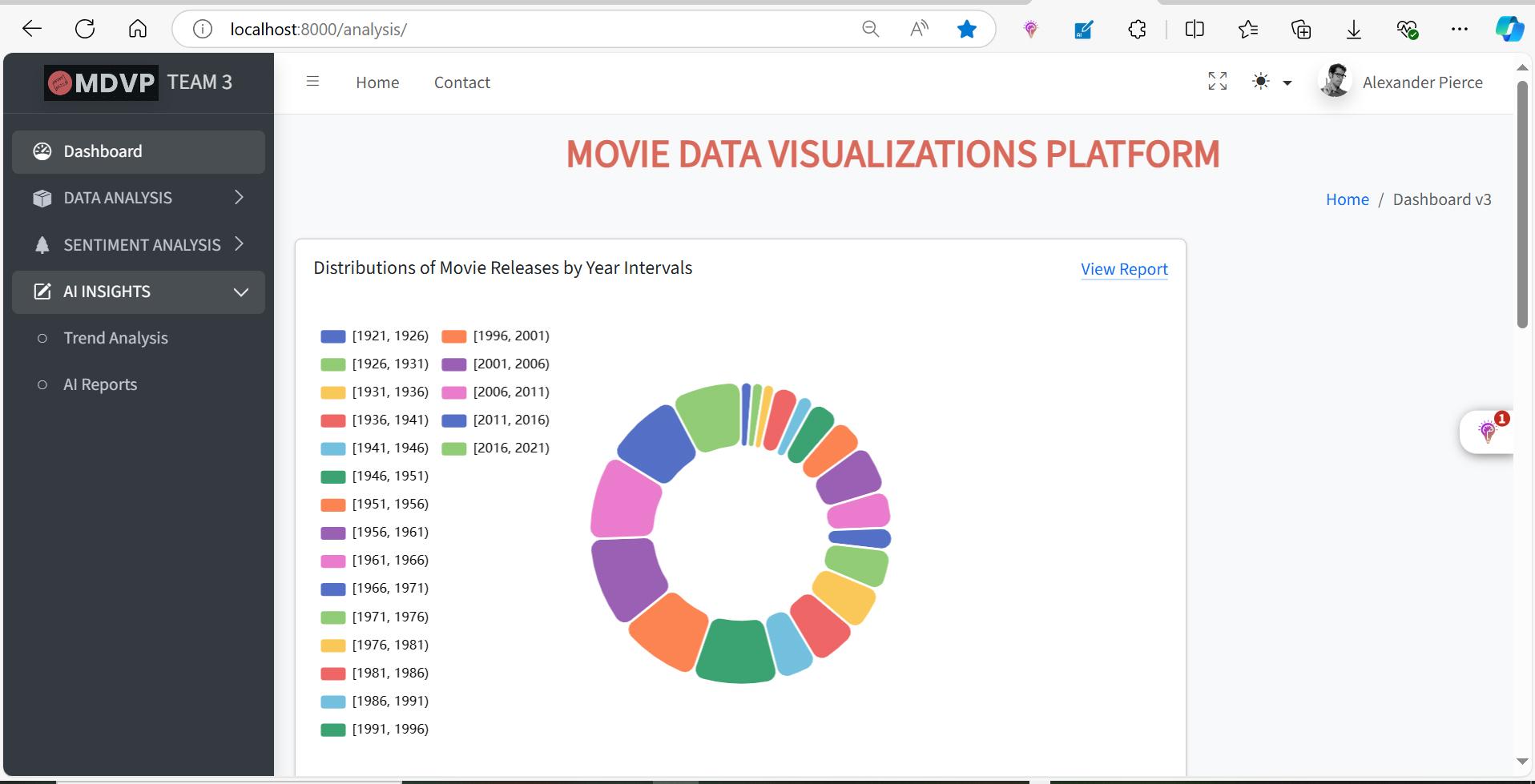
# UI Design (if applicable) - Xiaojuan Li

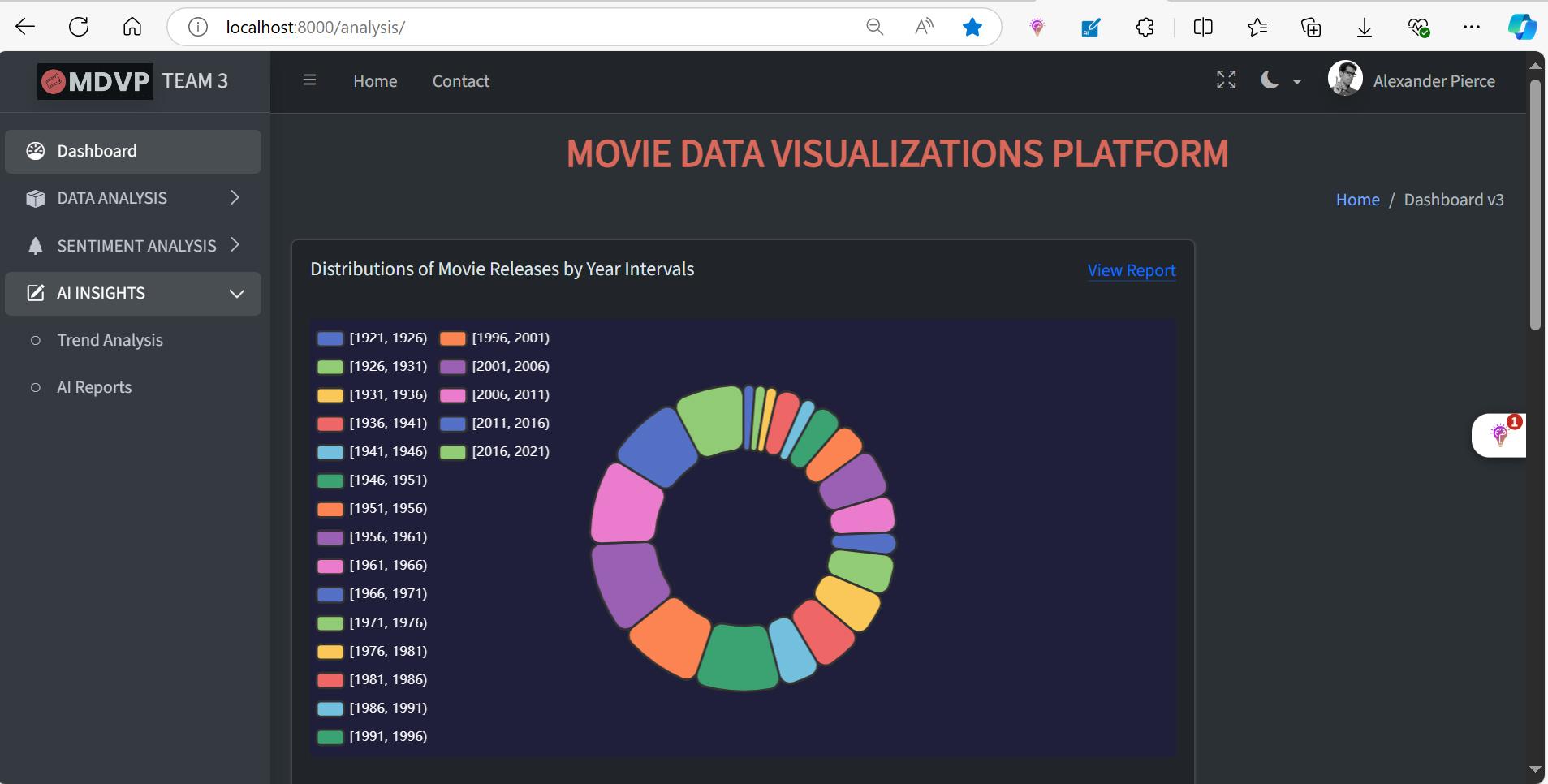
In the initial phase of the project, we created a wireframe to establish the basic ideas for the UI design of the Movie Data Visualizations Platform (MDVP). The wireframe helped define the structure and layout, guiding the design direction of the platform. Please see the details of layout as following:





After refining the initial concepts, we proceeded to design and implement a more polished version, which includes both dark and light modes for a better user experience.





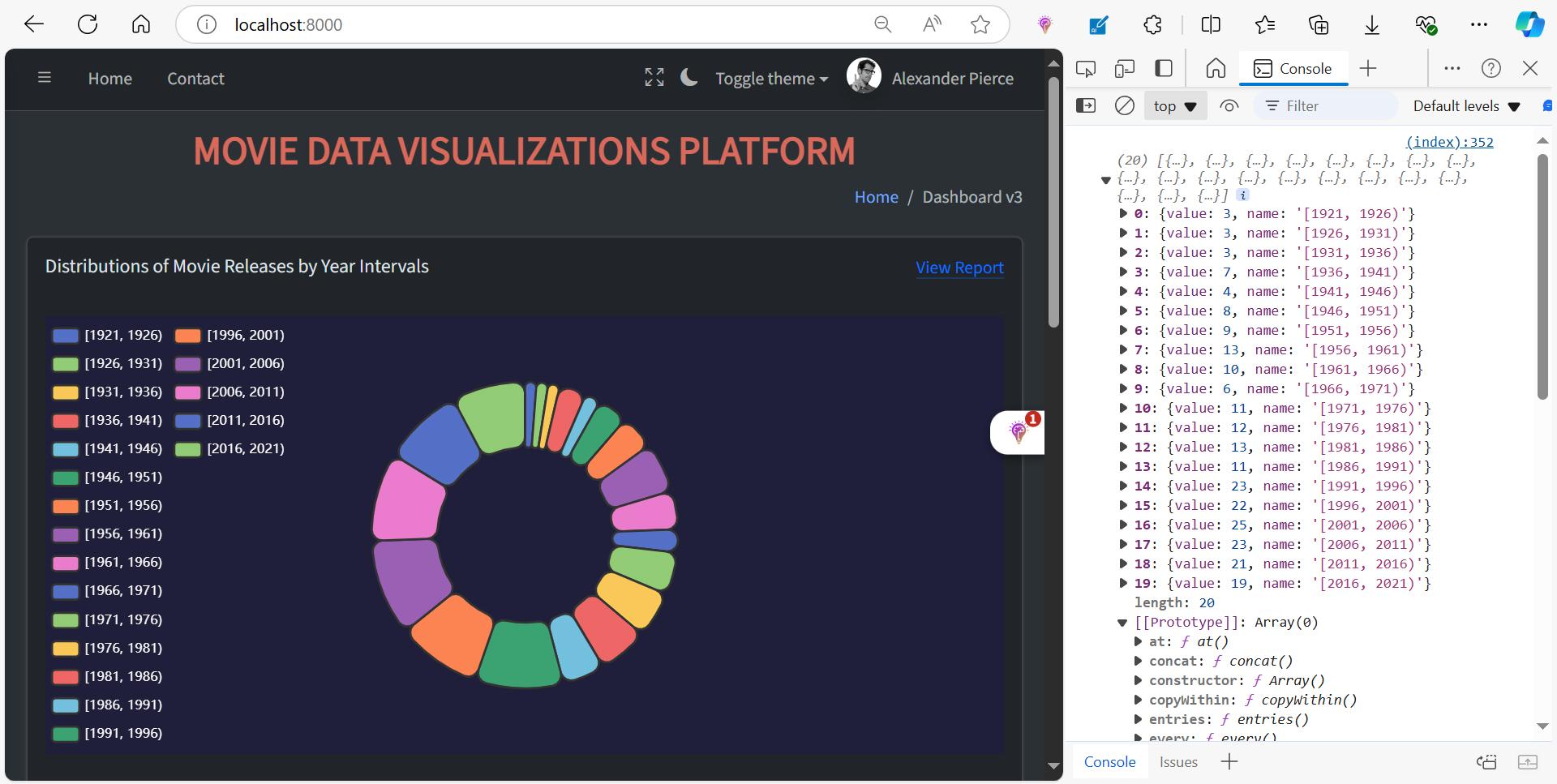
The UI design of our platform focuses on providing a user-friendly and intuitive interface to explore movie-related insights. The design is structured to ensure easy navigation and interaction with various data visualizations and tools for exploring IMDb data.

**Key Features of the UI Design:**

* **Navigation Panel**: Located on the left, the navigation panel allows users to easily access different sections of the platform, such as the Dashboard, Data Analysis (Box Office, Genre Insights, Movie Runtime, and Language Diversity), Sentiment Analysis (Word Cloud, Reviews Analysis), AI Insights, and Help sections.
* **Search and Filters**: At the top of the page, users can search for specific movie data and apply filters such as 'Global', 'Year Ascending', 'Year Descending', and 'Rating' to refine the data they want to visualize.
* **Dashboard and Data Visualizations**: The dashboard serves as the home screen where users are greeted with the primary message, "Explore the Movie Insights". Users will see interactive visualizations, which can be further customized based on the available analysis options.
* **Buttons and Actions**: The UI includes clear action buttons for logging in, signing up, and interacting with the platform's functionalities.
* **Interactive Elements**: Data visualizations are dynamic and customizable, offering an interactive experience for users to explore insights, filter data, and view details such as trends and sentiment analysis.

This design emphasizes simplicity and efficiency, ensuring that users can seamlessly navigate through various movie insights while engaging with data visualizations tailored to their specific needs. The interface is clean, modern, and responsive, providing a cohesive user experience across devices.

The integration of the frontend and backend for the Movie Data Visualizations Platform (MDVP) has been successfully completed. This ensures that the visualizations, user interactions, and data processing work seamlessly together, enabling users to explore movie insights with dynamic data fetched from the backend. This integration is a key milestone, ensuring that the system is fully functional with both the interface and backend components communicating effectively.



# Database Design (if applicable) - Ziliang Ren & Xuansheng Xia

* **Table data:**

**id (BIGINT, PRIMARY KEY, AUTO\_INCREMENT)**

* **Description**: A unique identifier for each movie entry.
* **Purpose**: Ensures each movie has a distinct identity. Using AUTO\_INCREMENT helps automatically assign a unique ID to each record.

**title (TEXT, NOT NULL)**

* **Description**: The title of the movie.
* **Purpose**: Stores the name of the movie, which is essential for display and searching.

**year (BIGINT, NOT NULL)**

* **Description**: The release year of the movie.
* **Purpose**: Useful for filtering movies by release date or sorting chronologically.

**score (DOUBLE, NOT NULL)**

* **Description**: The rating or score of the movie, typically on a scale from 1.0 to 10.0.
* **Purpose**: Represents the movie's rating, allowing for rating-based sorting and analysis.

**url (TEXT, NOT NULL)**

* **Description**: A URL link, likely pointing to a page with more information about the movie (e.g., IMDb).
* **Purpose**: Provides a reference for more detailed information about the movie.

**language (TEXT, NOT NULL)**

* **Description**: The language(s) in which the movie is available.
* **Purpose**: Helps identify t he primary language(s) spoken in the movie, useful for filtering movies by language.

**type (TEXT, NOT NULL)**

* **Description**: The genre(s) or type(s) of the movie (e.g., Drama, Action).
* **Purpose**: Allows categorization of movies by genre, which is useful for filtering and visualizing by genre type.

**ontime (TEXT, NOT NULL)**

* **Description**: The runtime of the movie, typically formatted as "2h 30m" or similar.
* **Purpose**: Indicates the length of the movie, which can be used for filtering by runtime.

**production\_company (TEXT, NULLABLE)**

* **Description**: The name of the production company or companies responsible for the movie.
* **Purpose**: Helps associate movies with their production companies for analysis.

**production\_country (TEXT, NOT NULL)**

* **Description**: The country or countries where the movie was produced.
* **Purpose**: Indicates the production location, which is useful for geographical analysis.

**gross (TEXT, NULLABLE)**

* **Description**: The gross revenue of the movie, expressed as a string for storing large amounts.
* **Purpose**: Used to analyze the financial success of the movie.

**awards (TEXT, NULLABLE)**

* **Description**: Awards won or nominated for, with a descriptive format (e.g., "Won 3 Oscars, Nominated for 5 more").
* **Purpose**: Lists the achievements of the movie, useful for identifying critically acclaimed films.

**geted (BIGINT, NULLABLE)**

* **Description**: The field's meaning is not specified; it might represent a count related to the movie, such as the number of times it has been reviewed or accessed.
* **Purpose**: This field would need further clarification or renaming for better understanding.
* **Table review:**

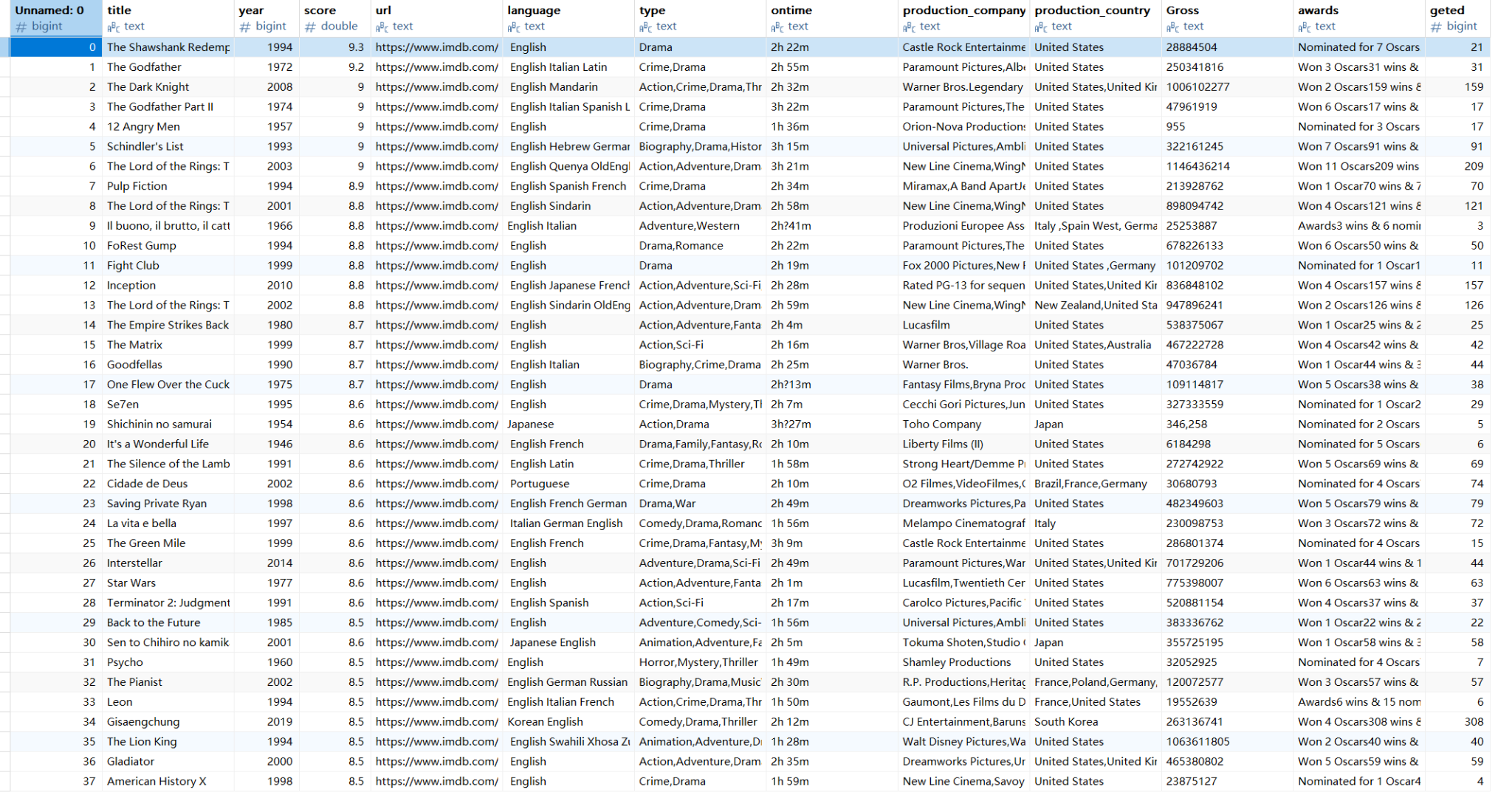
**id (BIGINT, PRIMARY KEY, AUTO\_INCREMENT)**

* **Description**: A unique identifier for each review entry.
* **Purpose**: Ensures each review has a unique identity. Using AUTO\_INCREMENT helps automatically assign a new ID to each review.

**movie\_id (VARCHAR(255), NOT NULL, FOREIGN KEY)**

* **Description**: Links the review to a specific movie in the data table.
* **Purpose**: Establishes a relationship between the review table and the data table, allowing for easy reference to the movie being reviewed.
* **Foreign Key Constraint**: Should reference the id column of the data table (if the primary key is available) to maintain referential integrity.

**content(TEXT, NOT NULL)**

* **Description**: The actual content of the review.
* **Purpose**: Stores the textual feedback from the reviewer, providing details about their opinion of the movie

# Security Design - Hengyi Song

### Iteration 1 - Security Design

#### **1. Secure Database Management**

**Goal**: Protect the database from unauthorized access and attacks.

**Plan**:

**Environment Variables**: Store database credentials in environment variables using Django’s .env files for secure storage.

**SQL Injection Protection**: Use Django’s ORM to prevent SQL injections by avoiding raw SQL queries.

**Data Encryption**:

Encrypt sensitive data (e.g., passwords) using Django’s default password hashing mechanism.

Use SSL/TLS for secure communication between the application and the database.

**Access Control**: Limit direct database access to the application and designated administrators.

#### **2. API Security**

**Goal**: Ensure secure communication between external services and your platform.

**Plan**:

**API Key Management**:

Store API keys for IMDB and other external services in environment variables.

Restrict API usage by rate limiting requests to prevent abuse.

**HTTPS**: Ensure all communication with external APIs uses HTTPS to prevent eavesdropping and man-in-the-middle attacks.

**Authentication**: Implement OAuth 2.0 or token-based authentication for APIs that require user-specific access.

#### **3. Session Management**

**Goal**: Secure user sessions to prevent unauthorized access.

**Plan**:

**Session Timeout**: Implement automatic session timeouts for inactive users to reduce the risk of unauthorized access.

**CSRF Protection**: Use Django’s CSRF protection middleware to prevent cross-site request forgery attacks.

**Secure Cookies**: Ensure that session cookies are marked as HttpOnly and Secure to prevent them from being accessed via JavaScript or transmitted over non-secure channels.

#### **4. Input Validation**

**Goal**: Prevent security vulnerabilities caused by improper input handling.

**Plan**:

**Form Validation**: Use Django’s form validation to sanitize and validate user input.

**Whitelist Approach**: For specific fields (e.g., text fields, dates), ensure only valid characters and formats are accepted to prevent script injection attacks (XSS).

**Error Handling**: Provide informative yet secure error messages that do not reveal system details to potential attackers.

#### **5. Audit Logging**

**Goal**: Track critical actions within the system for security monitoring.

**Plan**:

Log all critical actions, including logins, data modifications, and permission changes.

Store logs in a secure location, ensuring they are accessible only to authorized personnel (e.g., administrators).

Set up alerts for suspicious activities (e.g., multiple failed login attempts).

#### **6. Regular Security Audits**

**Goal**: Continuously monitor and improve system security.

**Plan**:

Conduct regular vulnerability assessments using tools such as SonarQube to detect code vulnerabilities and bad practices.

Perform penetration testing before each release to ensure no new security flaws are introduced.

# Business Logic and/or Key Algorithms - Mingyuan Xu

Input: Movie dataset from MySQL

Step 1: Load dataset using Pandas

Step 2: Define year intervals (e.g., [1921, 1926), [1926, 1931), ..., [2016, 2021))

Step 3: Group movies by year intervals using pd.cut()

Step 4: Count the number of movies in each group

Step 5: Prepare data for visualization (e.g., convert to JSON format for ECharts)

Output: Frequency of movies in each year interval

Input: 'language' column from movie dataset

Step 1: Extract all language entries from dataset

Step 2: Split each entry by space to get individual languages (as one movie may have multiple languages)

Step 3: Create an empty dictionary to store language counts

Step 4: For each language:

If language exists in dictionary:

Increment its count

Else:

Add language to dictionary with count 1

Step 5: Remove empty entries (if any)

Step 6: Sort dictionary by frequency in descending order

Step 7: Select top 20 languages for visualization

Output: Top 20 languages and their frequencies

Input: 'Runtime' column from movie dataset (e.g., '2h 30m')

Step 1: Define a function to convert runtime to minutes

Replace any '?' with empty string

Extract hours and convert to minutes (if available)

Extract remaining minutes (if available)

Return total runtime in minutes

Step 2: Apply conversion function to each movie runtime

Step 3: Define runtime intervals (e.g., [67, 77), [77, 87), ..., [237, 247))

Step 4: Group movies by runtime intervals using pd.cut()

Step 5: Count the number of movies in each group

Output: Frequency of movies in each runtime interval

Input: Movie reviews dataset from MySQL

Step 1: Load all reviews from dataset

Step 2: Concatenate all review texts into a single string

Step 3: Clean the text by removing unnecessary characters (e.g., punctuation, special symbols)

Step 4: Split text into individual words

Step 5: Create a function to count word frequencies

For each word in text:

If word exists in dictionary:

Increment its count

Else:

Add word to dictionary with count 1

Step 6: Remove common stopwords from the dictionary

Step 7: Sort dictionary by frequency in descending order

Step 8: Select top words to generate word cloud using Pyecharts

Output: Word cloud visualization of most common words in movie reviews

# Design Patterns - Ziliang Ren

**1. Web Scraping**

For the web scraping part, I’d go with **Strategy Pattern** and **Template Method Pattern**:

* **Strategy Pattern**: Since I might be scraping data from different websites, I’ll use this pattern to define different scraping strategies for each site. This way, I can easily switch strategies without messing with the core logic—pretty handy when dealing with different website structures.
* **Template Method Pattern**: I’d set up a general template for scraping, with standard steps like fetching, parsing, and cleaning data. But the specifics, like how I parse different types of pages, would be handled in subclasses, while keeping the main workflow intact.

**2. Data Storage and Processing**

I’d utilize the **Data Access Object (DAO) Pattern** here:

* **DAO Pattern**: In Django, using model classes feels like a natural fit for this pattern. The models would take care of all the database access, letting me keep the data handling separate from the business logic. This keeps things neat and makes it easier to manage changes.

**3. Django Views and Templates**

For this part, I’d stick with **MVC Architecture** and **Observer Pattern**:

* **MVC Architecture**: Django's MVT structure (Model-View-Template) aligns nicely with the MVC concept. Models handle data, views manage the logic, and templates take care of rendering the visuals. It’s a good way to keep things organized.
* **Observer Pattern**: If I want the front-end to update automatically when data changes—like in real-time chart updates—I’d set up WebSockets to push updates from the server to the front-end. It’s like setting up a watcher for data changes.

**4. Visualization**

I’d apply **Factory Pattern** and **Decorator Pattern** for the charts:

* **Factory Pattern**: I can use this pattern to dynamically create different chart types (bar charts, line charts, scatter plots, etc.) depending on the parameters passed. It simplifies the process of managing multiple chart types.
* **Decorator Pattern**: If I need to add extra features to the charts, like filtering data or tweaking styles, I’d use this pattern to wrap the chart objects with additional functionality, without altering the core chart code.

**5. Word Cloud and Data Analysis**

For analyzing data and creating word clouds, I’d use the **Command Pattern**:

* **Command Pattern**: I’d encapsulate analysis tasks (like generating word clouds or sentiment analysis) into command objects. This way, I can easily manage and execute different analysis commands when needed, making the code more flexible.

# Rest APIs

# Any Additional Topics you would like to include.

# References

# Glossary