De La Salle University- Manila
Gokongwei College of Engineering

LBYCPA2
Data Structures and Algorithms Laboratory

Project Documentation

FlixPick: Movie Recommendation System

Samuelle P. Barja

Aaron Tristan Jethro G. Dizon

November 23, 2023

I. Project Description

Overview

The "Movie Recommendation System" is a sophisticated program designed to tackle the daunting challenge of selecting movies from an ever-expanding cinematic landscape. At its core, this system relies on user preferences and leverages data from TMDB, also known as The Movie Database. TMDB is a widely utilized database that provides a wealth of information about movies, including details about cast, crew, production, release dates, reviews, and ratings [1].

TMDB serves as a foundation for the Movie Recommendation System, offering a rich and up-to-date repository of movie-related data. This includes not only the essential facts about each film but also nuanced details that contribute to the system's ability to generate tailored movie suggestions. The inclusion of TMDB in the system's architecture is pivotal, as it ensures that users receive personalized recommendations based on accurate and relevant information.

The Movie Recommendation System's multifaceted functionality encompasses a user-friendly login system, efficient playlist and bookmark management, and a powerful recommendation engine. The recommendation engine, fueled by data from TMDB, employs advanced algorithms to analyze user choices, preferences, and historical viewing patterns. By processing this information alongside the extensive movie details available on TMDB, the system can offer precise and personalized movie suggestions.

The significance of this study lies in its response to the pervasive issue of information overload encountered by users when navigating the vast sea of available movies. TMDB not only serves as a data source but also as a solution to this problem. With personalized movie suggestions, the system alleviates the burden on users, making the movie selection process more streamlined and enjoyable. By tapping into the capabilities of TMDB, the Movie Recommendation System aims to enhance user satisfaction, allowing individuals to discover movies that align with their interests and preferences seamlessly.

Problem Statement

This project aims to resolve the common issue of users feeling overwhelmed by the vast selection of movies. [3] According to a report, it is revealed that 46% of the streaming respondents who took the survey are overwhelmed, due to the fact that they are exposed to continuous growth in the number of platforms and titles, which results in difficulty in choosing what they really want. Additionally, decision fatigue may develop in such a situation, making it more difficult to make decisions with assurance and effectiveness. [2] Due to the greater likelihood of making a poor choice, this abundance of alternatives can cause tension and anxiety. These problems may be resolved by narrowing the options and concentrating on a manageable number of possibilities. This will simplify the decision-making process, make it more fun, and lessen the weight of self-blame in the case that the decision has a negative effect. People may make better informed selections that are in line with their genuine preferences by being aware of the disadvantages of having too many options. By offering

personalized recommendations and a platform for organizing movie selections, it aims to simplify the decision-making process.

II. Objectives

The primary goals and objectives of this project include the development of a recommendation system, user authentication, and efficient management of movie playlists and bookmarks:

1. Recommendation System:

Develop a robust recommendation engine that offers movie suggestions based on user preferences, previous selections, and comprehensive movie data. The system should continuously refine recommendations based on user feedback and viewing history.

2. User Authentication:

Implement a secure login system that allows users to create accounts, manage their profiles, and access personalized recommendations.

3. Playlist Management:

Enable users to add and delete playlists generated from the recommendation itself for future viewing. These features should be user-friendly and adaptable to user needs.

The construction of a recommendation system, which seeks to make movie suggestions to users based on their tastes and watching history, is at the heart of the project's scope. The establishment of a user authentication system is also a crucial element since it will make it easier to create and manage user accounts and store user data for customization. The project also places a strong emphasis on the construction and administration of user-specific playlists or bookmarks for the movies users want to view. However, successful constraint management is essential to the project's success. Notably, the correctness and completeness of the movie dataset are necessary for the quality of movie suggestions, underlining the need for accurate data. To guarantee secure user authentication, data protection, and the preservation of user preferences and information, privacy and security are high objectives. To successfully complete the project, resource restrictions such as possible limits on server storage for user data and movie information must also be appropriately handled.

III. Methodology

In this section, we provide an overview of the Movie Recommendation System project, highlighting the need for such a system in the context of the overwhelming abundance of movies available today. The methodology described here outlines the major steps to be taken during the project's development.

1. Materials/Softwares used:

This section outlines the software and resources employed in the development of the program.

Integrated Development Environment (IDE): IntelliJ IDEA is the primary integrated development environment (IDE) used for software development. It provides a robust platform for coding, testing, and debugging.

User Interface Framework: JavaFX is utilized for creating the user interface of the application. It allows for the development of interactive and visually appealing desktop applications.

Bcrypt JAR File: This file is essential for implementing secure password hashing within the user authentication system. By utilizing Bcrypt, the program enhances the security of user account information, protecting sensitive data from unauthorized access.

json JAR File: JSON (JavaScript Object Notation) is a widely used data interchange format. The json JAR file is incorporated to parse and handle JSON data efficiently. This is particularly crucial when interacting with external APIs or managing configuration files in the application.

json-simple JAR File: Similar to the json JAR file, json-simple is included to streamline the processing of JSON data. It provides simple APIs for parsing and manipulating JSON structures, contributing to the overall functionality and flexibility of the Movie Recommendation System.

TMDB API: To ensure the program maintains an up-to-date and relevant movie database, integration with external data sources is recommended. The program is designed to access data from external sources, such as the TMDB (The Movie Database) API. This integration enables the retrieval of the latest movie information, reviews, and ratings, enriching the recommendations provided to users.

By incorporating external data sources, the program ensures that users have access to the most current and comprehensive movie database, enhancing the quality of movie recommendations and user experience.

2. System Design:

Here, we delve into the design of the Movie Recommendation System's architecture. We discuss the major components and create visualization tools such as the Information Processing Diagrams (IPO). This diagram will help illustrate the system's flow, user interactions, and structural elements.

A. Input-Process-Output Table

Feature	Input	Process	Output
User Authentication System	User registration data (Password & Username)	Stores the user's data in a hash table to be read in the system	Confirmation of successful registration
	User Login data (Password & Username)	Reads the data to see if it is in the system	Forwarding to the main menu of the program

Recommendation Algorithm	User preferences	Utilize a decision tree algorithm to generate movie recommendations	List of movie recommendations based on user preferences
Playlist Management	User selection to remove playlist or add playlist from recommendation algorithm	Add or remove playlists stored in a linked list representing playlists	Customizable list of playlists to the user's liking

Table 1. I-P-O Chart

B. Code Implementation and program design:

a. User Authentication System:

The User Authentication System is a fundamental element of the Movie Recommendation System, ensuring secure account creation and login processes. This section outlines the meticulous implementation of the user authentication system, highlighting the emphasis on data security in managing sensitive user account information.

Implementation:

1. Data Security Measures:
   - The implementation prioritizes data security, recognizing the sensitivity of user account information.
   - Hashing techniques are applied to passwords, ensuring secure storage and safeguarding against unauthorized access.

2. Use of Maps and Hash Tables:
   - Maps and Hash Tables play a pivotal role in managing user accounts, providing an efficient and secure means of storage.
   - User account information, comprising usernames and hashed passwords, is stored in a hash table for optimized data retrieval during login.

3. Rapid Retrieval and Privacy Maintenance:
   - Hashing Techniques are employed to facilitate rapid retrieval of user data during login while maintaining user privacy.
   - The use of hash tables enhances security, preventing unauthorized access to sensitive user account information.

The User Authentication System's meticulous implementation, featuring secure password hashing and efficient data management through maps and hash tables, reflects the Movie Recommendation System's commitment to safeguarding user information. By prioritizing data security and utilizing different data structures, the

system ensures a trustworthy and protected environment for users to create accounts and log in securely.

b. Recommendation Algorithm:

The Recommendation Algorithm is a pivotal component of the Movie Recommendation System, responsible for generating personalized movie suggestions based on user preferences and historical data. This sophisticated feature explores various recommendation algorithms, including collaborative filtering, content-based filtering, and hybrid methods, to ensure a comprehensive and accurate movie recommendation process.

Implementation:

1. Decision Tree Utilization:
   - The recommendation algorithm employs a tree data structure, specifically a decision tree, to enhance the precision of movie recommendations.
   - The decision tree construction is a crucial aspect of the system, considering multiple attributes such as genre, subgenre, and user ratings.

2. Informed Movie Recommendations:
   - The decision tree is designed to make informed recommendations by traversing different attributes, providing a nuanced and tailored suggestion process.

3. Consideration of User Preferences:
   - User preferences play a vital role in the decision tree traversal, allowing the system to narrow down recommendations based on individual tastes.

The implementation of a decision tree in the Recommendation Algorithm underscores the Movie Recommendation System's commitment to providing users with personalized movie suggestions. By exploring various recommendation algorithms and incorporating a decision tree that considers multiple attributes, the system enhances its ability to navigate and understand user preferences, ultimately delivering a superior movie recommendation experience.

c. Playlist Management:

The Playlist Management feature of the Movie Recommendation System is designed to empower users in effortlessly organizing and customizing their favorite selections. This functionality encompasses the addition, removal, and reordering of playlists to ensure a personalized and streamlined movie-watching experience.

Implementation:

1. Adding/Removing Playlists:
- Users can seamlessly add or remove playlists generated by the recommendation algorithm.

- The program leverages a linked list data structure to efficiently manage user interactions within playlists.

2. Linked List Representation:
- Each playlist is represented as a linked list of movies.
- This design choice enhances flexibility, allowing users to easily add, remove, or rearrange movies within a playlist without the need for resizing operations.
- The linked list structure simplifies user interactions, providing an intuitive and straightforward method for managing playlists.

The incorporation of linked lists in the Playlist Management feature underscores the Movie Recommendation System's commitment to user-centric design. This approach optimizes efficiency, ensuring that users can effortlessly curate and customize their movie playlists to align with their preferences and viewing moods.

d. Navigating Back and Forth Feature:

Within the Movie Recommendation System, a user-friendly feature facilitates seamless navigation between pages during the preferences-setting phase for the movie finder.

Implementation:

The implementation involves the utilization of a **stack data structure**, specifically designed to enhance the user experience. A stack allows the user to move both backward and forward by efficiently managing the sequence of pages in a last-in, first-out (LIFO) manner.

Functionality:

1. Pushing Pages:
   - As the user progresses through the preference-setting journey, each page is pushed onto the stack.
   - This ensures that the system retains the sequence of pages visited, creating a navigational trail.

2. Popping Pages:
   - In the event that the user wishes to revise or modify a previous preference, the system pops the current page from the stack.
   - This action effectively allows the user to return to the previous page, preserving the input made on each step.

By implementing this stack-based navigation system, the Movie Recommendation System prioritizes user convenience and flexibility, ensuring a smooth and user-friendly experience throughout the preferences-setting phase.

In conclusion, this research paper's methodology serves as a comprehensive guide for the development of the Movie Recommendation System. The structured approach and detailed descriptions of each phase and component ensure that the system effectively addresses the challenge of assisting users in finding movies to watch.

C. Pseudocode

```
MODULE Flixpick

 MODULE UserAuthentication
  InitializeUserCredentials
    Load user accounts from file during class initialization
    Store them in a Map
   END MODULE

  RegisterUser(username, password) RETURNS Boolean
     IF username is not in userCredentials
        hashedPassword ← HashPassword(password)
        Add (username, hashedPassword) to userCredentials
        Save userCredentials to file
        RETURN true (registration successful)
     ELSE
        RETURN false (user already exists)
    END FUNCTION

  AuthenticateUser(username, password) RETURNS Boolean
     IF username is in userCredentials
        storedHashedPassword ← Get hashed password from userCredentials
        RETURN result of BCrypt.checkpw(password, storedHashedPassword)
     ELSE
        RETURN false (user not found)
    END FUNCTION
 END MODULE

 MODULE TMDbApiClient
  FetchMovies(year, originalLanguage, rating, genres) RETURNS HttpResponse<JsonNode>
     Construct API endpoint and query parameters
     Make HTTP request to fetch movies based on parameters
     RETURN the HTTP response
    END FUNCTION

  GetCertification(movieId) RETURNS String
     Make HTTP request to fetch movie details including release dates
     Extract certification from the response for the "US" region
```

```
    RETURN the certification or "Unknown" if not found
  END FUNCTION
END MODULE

MODULE RecommendationAlgorithm
 BuildMovieTree(response) RETURNS movieNode
    Parse API response to create a list of movieNode objects
    Build a balanced binary search tree from the list
    RETURN the root of the binary search tree
  END FUNCTION

 RecommendMovies(allMovies, recommendedMovies, desiredCertification)
    Sort allMovies based on rating in descending order
    Initialize addedMovies as an empty HashSet
    FOR EACH movie IN sorted allMovies
       certification ← GetCertification(movie.getID())
       desired ← MapRatingToAgeScale(desiredCertification)
       IF certification <= desired AND movie.getID() NOT IN addedMovies
          PrintMovieDetails(movie)
          Add movie to recommendedMovies
          Add movie.getID() to addedMovies
          IF Size of addedMovies equals 10
             EXIT FOR loop (stop when 10 movies are printed)
          END IF
       END IF
    END FOR
  END MODULE

 InOrderTraversal(node, rating) RETURNS List<movieNode>
    Initialize movies as an empty List<movieNode>
    InOrderTraversal(node, rating, movies)
    RETURN movies
  END FUNCTION

 InOrderTraversal(node, rating, movies)
    IF node is not null
       InOrderTraversal(node.right, rating, movies)
       IF node.getRating() >= rating
          Add node to movies
       END IF
       InOrderTraversal(node.left, rating, movies)
    END IF
  END PROCEDURE

```

```
  PrintMovieDetails(node)
    Print details of the movieNode (title, overview, date, genres, rating, language, image)
  END PROCEDURE

  ConvertGenreIdsToNames(genreIds) RETURNS List<String>
    Map genre IDs to names based on predefined mapping
    RETURN List of genre names
  END FUNCTION

  MapRatingToAgeScale(rating) RETURNS Integer
    Map different rating systems to a common age-based scale
    RETURN mapped age
  END FUNCTION
END MODULE

MODULE PlaylistManagement
  createPlaylist(playlist)
    Add a new Playlist to the playlists
  END PROCEDURE

  removePlaylist(playlistIndex)
    IF isValidPlaylistIndex(playlistIndex)
      Remove playlist at index playlistIndex from playlists
    ELSE
      Print "Invalid playlist index."
    END IF
  END PROCEDURE

  isValidPlaylistIndex(playlistIndex) RETURNS Boolean
    RETURN (playlistIndex >= 0 AND playlistIndex < playlists.size())
  END FUNCTION

  getPlaylists() RETURNS LinkedList<Playlist>
    RETURN playlists
  END FUNCTION
END MODULE

MODULE Playlist
  Playlist(playlist)
    Initialize the playlist with given movies
  END PROCEDURE

  getMovies() RETURNS LinkedList<movieNode>
    RETURN the playlist
```

```
126     END FUNCTION
127  END MODULE
128
129  MODULE FileManagement
130    loadUserAccounts() RETURNS Map<String, String>
131       Try to
132          Open user accounts file for reading
133          Read and return the user accounts map
134       Catch FileNotFoundException
135          Print "File not found," return an empty map
136       Catch IOException or ClassNotFoundException
137          Print stack trace, return an empty map
138     END FUNCTION
139
140    saveUserAccounts(userAccounts)
141       Try to
142          Open user accounts file for writing
143          Write the user accounts map
144       Catch IOException
145          Print stack trace
146     END PROCEDURE
147
148    loadPlaylists(username) RETURNS playlistManagement
149       playlistsFileName ← GetPlaylistsFileName(username)
150       Try to
151          Open playlists file for reading
152          Read and return the playlistManagement object
153       Catch FileNotFoundException
154          Print "File not found," return a new instance
155       Catch IOException or ClassNotFoundException
156          Print stack trace, return a new instance
157     END FUNCTION
158
159    savePlaylists(username, playlists)
160       playlistsFileName ← GetPlaylistsFileName(username)
161       Try to
162          Open playlists file for writing
163          Write the playlists object
164       Catch IOException
165          Print stack trace
166     END PROCEDURE
167
168    getPlaylistsFileName(username) RETURNS String
169       playlistsDirectory ← GetPlaylistsDirectory()
```

```
170       IF playlistsDirectory does not exist
171           Create playlistsDirectory
172       RETURN playlistsDirectory + File.separator + username + "_playlists.ser"
173     END FUNCTION
174
175   getPlaylistsDirectory() RETURNS File
176       RETURN new File(PLAYLISTS_DIRECTORY)
177     END FUNCTION
178  END MODULE
179
180  CLASS FlixpickApplication
181    main(args)
182       Initialize Flixpick application
183       Start the application
184     END PROCEDURE
185  END CLASS
186 END MODULE
```

D. UML Diagram

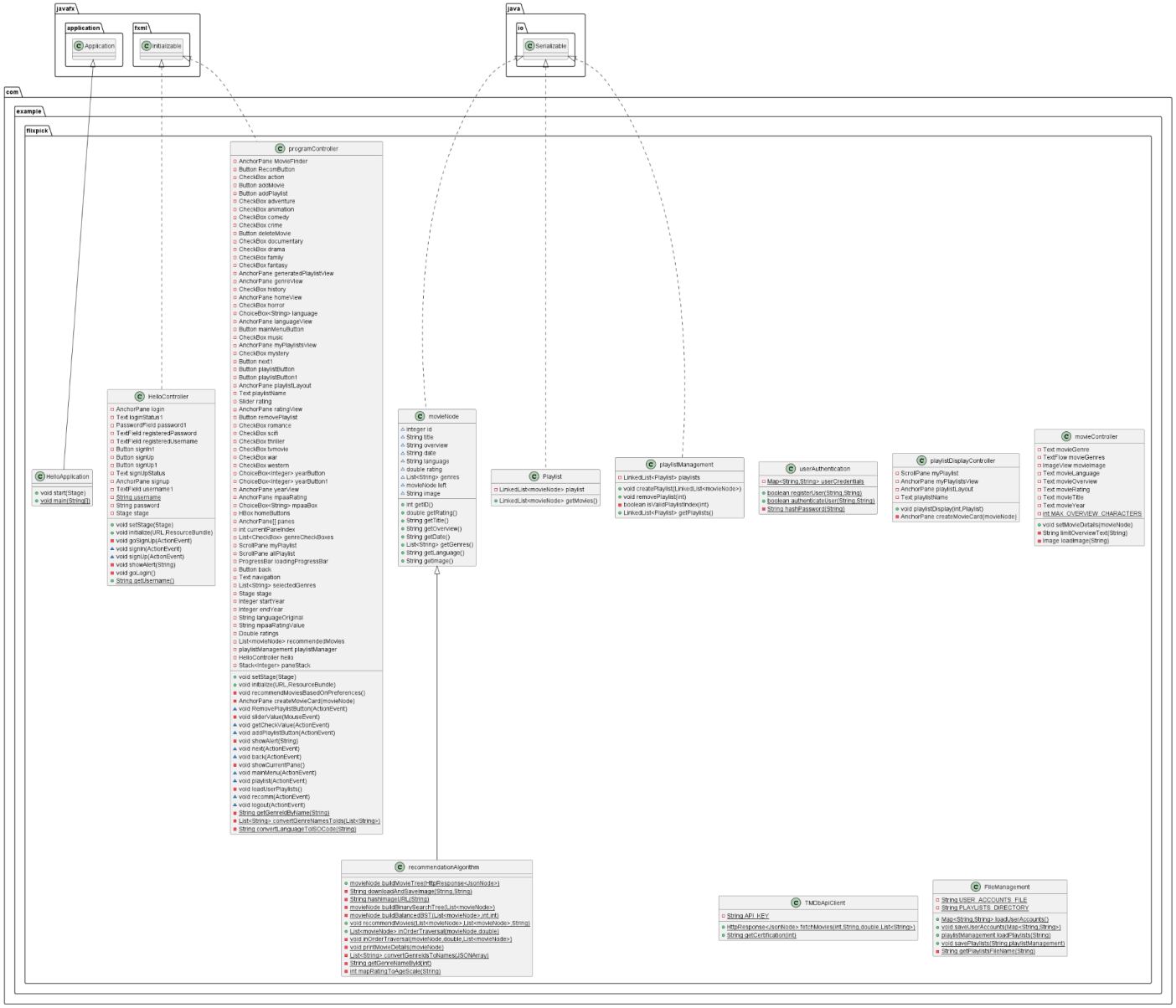

Diagram 1 UML Diagram

In the development of the Movie Recommendation System, a thoughtful selection of materials and software played a pivotal role in ensuring robust functionality and a seamless user experience. The Integrated Development Environment (IDE) of choice was IntelliJ IDEA, providing an efficient platform for coding, testing, and debugging. For crafting an interactive and visually appealing user interface, JavaFX served as the User Interface Framework, facilitating the development of a desktop application that meets both aesthetic and functional requirements.

The security of user account information was prioritized through the incorporation of the bcrypt JAR file, which enabled secure password hashing within the user authentication system. This implementation added an extra layer of protection, safeguarding sensitive data from unauthorized access. JSON (JavaScript Object Notation) played a crucial role in data interchange, with the inclusion of the json JAR file for parsing and handling JSON data efficiently. Additionally, the json-simple JAR

file was introduced to streamline the processing of JSON data, enhancing the overall functionality and flexibility of the Movie Recommendation System.

To ensure the program's access to an up-to-date and relevant movie database, integration with external data sources was achieved through the TMDB (The Movie Database) API. This integration allowed the retrieval of the latest movie information, reviews, and ratings, enriching the recommendations provided to users. Conversions for user details, such as MPAA rating, ISO 639-2, and genre IDs, were strategically implemented to translate and harmonize user preferences with the TMDB API. This thoughtful integration improved the readability and relevance of the recommendations, contributing to an enhanced user experience.

Moving on to the system design, an Information Processing Diagram (I-P-O) was utilized to visualize the major components, system flow, and user interactions. The Input-Process-Output (I-P-O) chart illustrated the key features of the Movie Recommendation System, including the User Authentication System, Recommendation Algorithm, and Playlist Management. Each component's data structure, such as the use of maps and hash tables for user authentication and linked lists for playlist management, was carefully selected to optimize efficiency and ensure seamless user interactions.

In the code implementation and program design section, specific emphasis was placed on user authentication, where maps and hash tables were employed for secure management of user accounts. The Recommendation Algorithm section detailed the use of a tree data structure, specifically a decision tree, to generate informed movie recommendations based on user preferences. The Playlist Management section highlighted the use of linked lists for efficient handling of user-generated playlists, allowing for easy addition, removal, and organization of movies within playlists.

IV. Results and Discussion

A. Results

A smooth and safe user experience is guaranteed by the careful development of the FlixPick Movie Recommendation System, which placed a strong emphasis on fundamental design principles and exact technical implementations. To ensure its efficacy and a satisfying user experience, key elements including playlist management, user identification, the recommendation algorithm, and external API integration went through a lot of testing.

In the realm of user authentication, advanced security measures were implemented, utilizing bcrypt hashing for password security. User account data, comprising usernames and hashed passwords, is stored in a hash table. The hashing process adds an extra layer of security, safeguarding sensitive information. The `FileManagement` class orchestrates the serialization and deserialization of user accounts, ensuring data integrity across program executions.

The recommendation algorithm, powered by a decision tree, is a sophisticated feature of FlixPick. The decision tree was meticulously constructed to consider multiple attributes, including genre, subgenre, and user ratings, ensuring accurate and tailored movie suggestions. Rigorous testing was conducted to validate the algorithm's precision in generating personalized recommendations based on individual user preferences.

Playlist management within FlixPick leverages a linked list structure, providing efficiency in organizing and manipulating user playlists. The `FileManagement` class oversees the serialization of playlists, ensuring seamless storage and retrieval. Testing focused on the seamless addition, removal, and organization of playlists, validating the program's efficiency in managing user-generated content.

The program's navigation system, facilitated by a stack data structure, allows users to move back and forth between pages when specifying preferences for the movie finder. The stack implementation ensures an intuitive and smooth user experience during the preference-setting process.

In terms of testing and verification, serialization and deserialization processes, managed by the `FileManagement` class, were thoroughly scrutinized to ensure correct and seamless transitions of user data across different program executions. External API integration, particularly with the TMDB API, was a pivotal aspect of FlixPick. The `TMDBService` class facilitated communication with the API, retrieving crucial movie details, reviews, and ratings. Testing validated the program's ability to incorporate the latest movie information into its recommendations.

The program also incorporates ISO 639-2 code conversion [4] for language preferences and MPAA rating conversion [5] for different countries, showcasing its commitment to inclusivity. Technical implementations of these features ensure the seamless adaptation of the program to diverse user preferences, contributing to a more personalized recommendation experience.

In conclusion, FlixPick successfully combines innovative features with meticulous technical implementations, creating a dynamic and user-centric Movie Recommendation System. Thorough testing and verification confirm the program's reliability, adaptability to user preferences, and commitment to security. ISO 639-2 code and MPAA rating conversion add a layer of personalization, contributing to a positive user experience. Continuous refinement and updates will further enhance the program's capabilities, ensuring its sustained success in providing tailored movie recommendations to a diverse and global user base.

B. Walkthrough

The FlixPick Movie Recommendation System provides users with a comprehensive and user-friendly walkthrough, ensuring a seamless journey from account creation to personalized movie recommendations. Let's delve into each step of the walkthrough, exploring additional details:

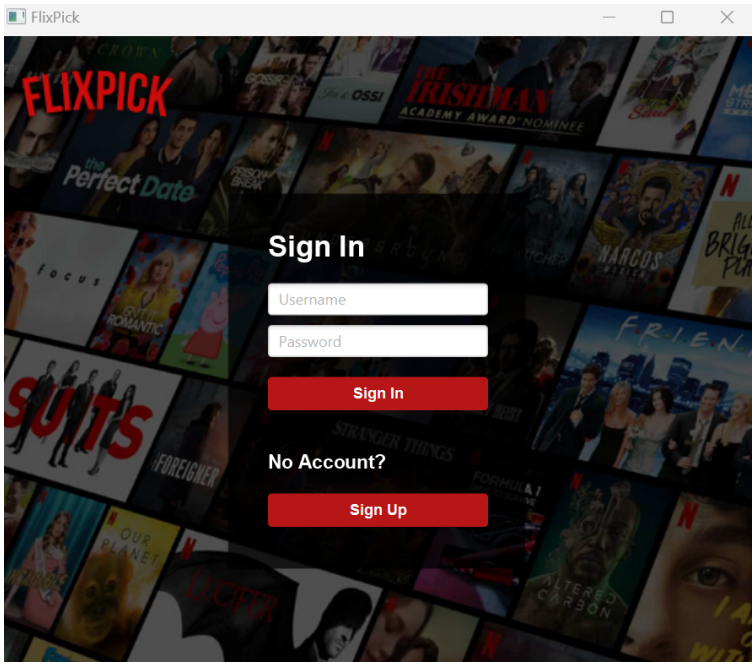

Figure 1.1 Sign In Page

The initiation point, the Sign In page (Figure 1.1), boasts a clean and straightforward design. Users input their account username and password to access the system.

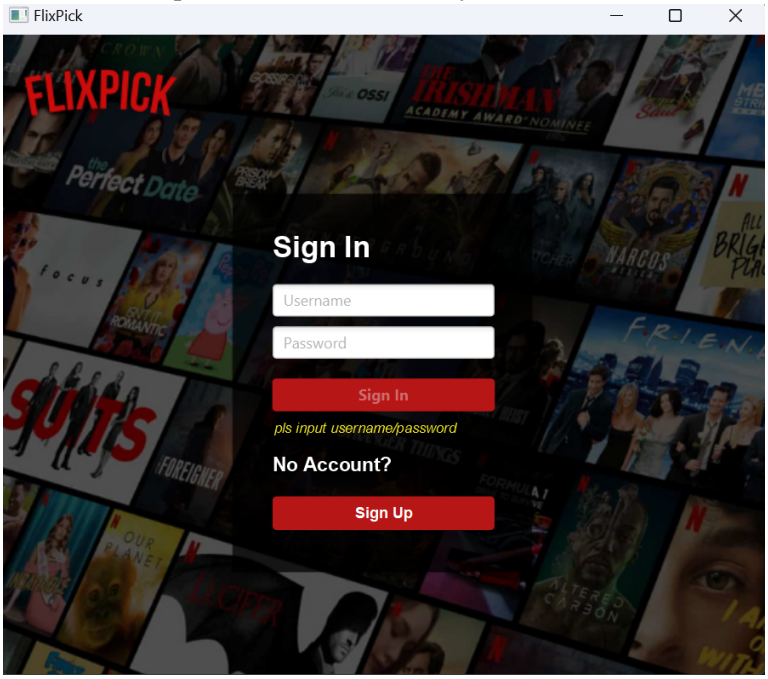

Figure 1.2 Sign In Error

In the event of missing parameters, the system responds with a user-friendly error message (Figure 1.2), guiding users to rectify the input.

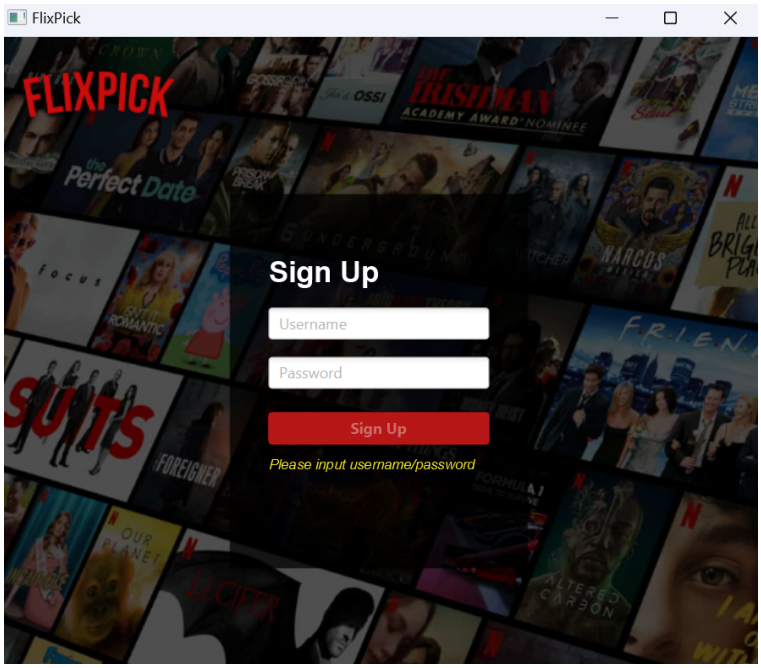

Figure 1.3 Sign Up page

Upon selecting the sign-up button, users are seamlessly transitioned to the Sign Up page (Figure 1.3). Here, a username and password can be securely entered for account registration. The system intelligently prevents duplicate account creation and ensures all necessary parameters are completed.

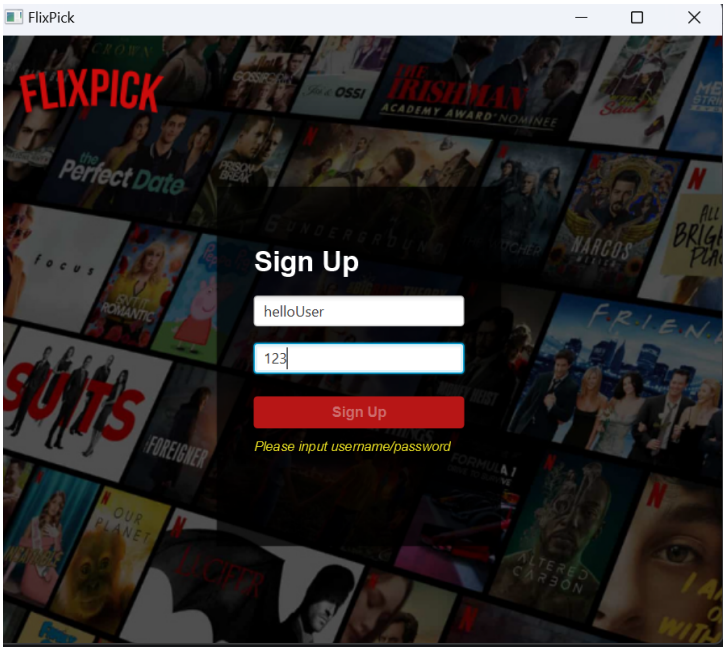

Figure 1.4 Registering account details

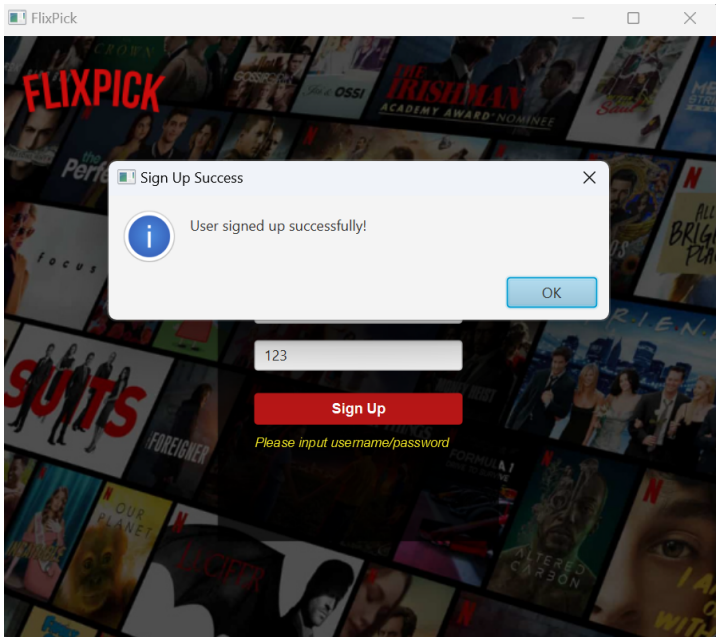

Figure 1.5 Successfully registering account

Once the user inputs their desired credentials (Figure 1.4), a reassuring alert prompt (Figure 1.5) confirms the successful registration, providing a positive user experience.

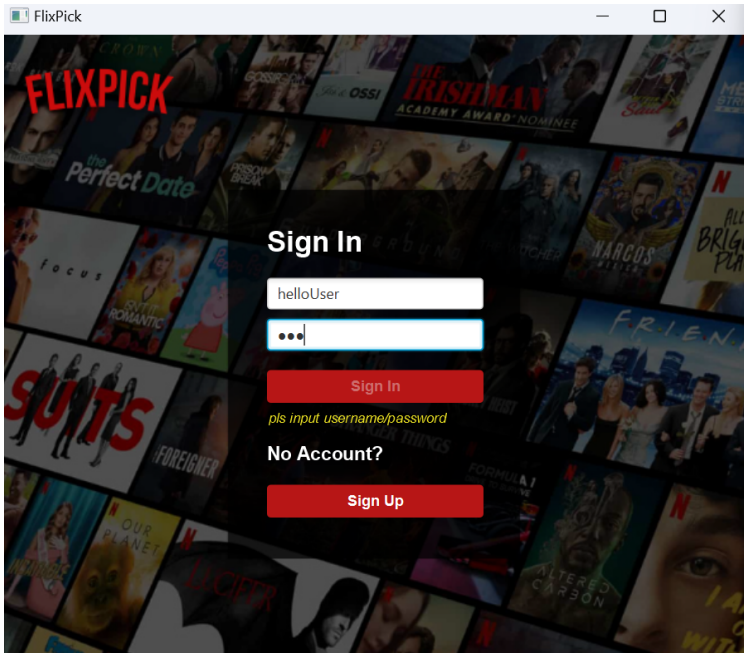

Figure 1.6 Inputting user details

Post-registration, users are smoothly redirected to the login page. Upon entering valid credentials (Figure 1.6), the system seamlessly navigates users to the main menu.

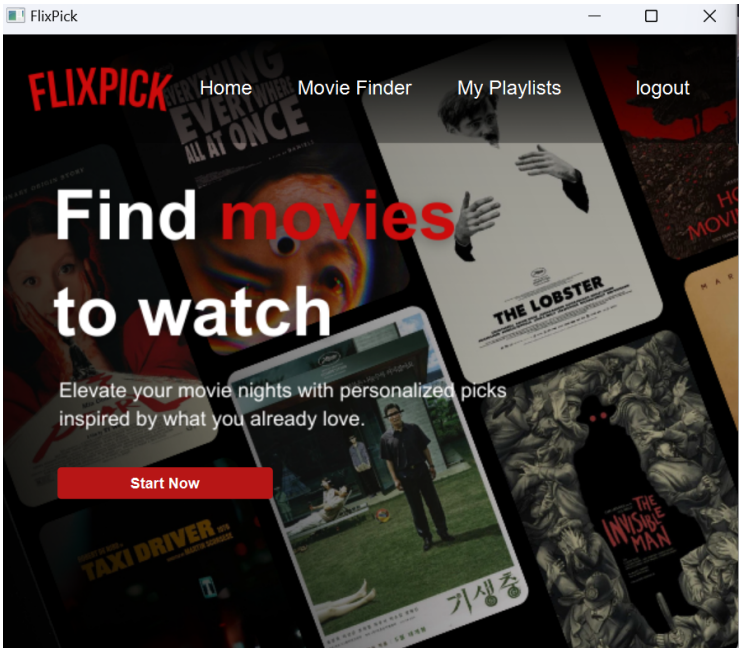

Figure 2. Home page

The main menu (Figure 2) serves as the program's hub, offering a concise program description and user-friendly navigation buttons. Clicking the "Start Now" button propels users into the engaging movie finder.

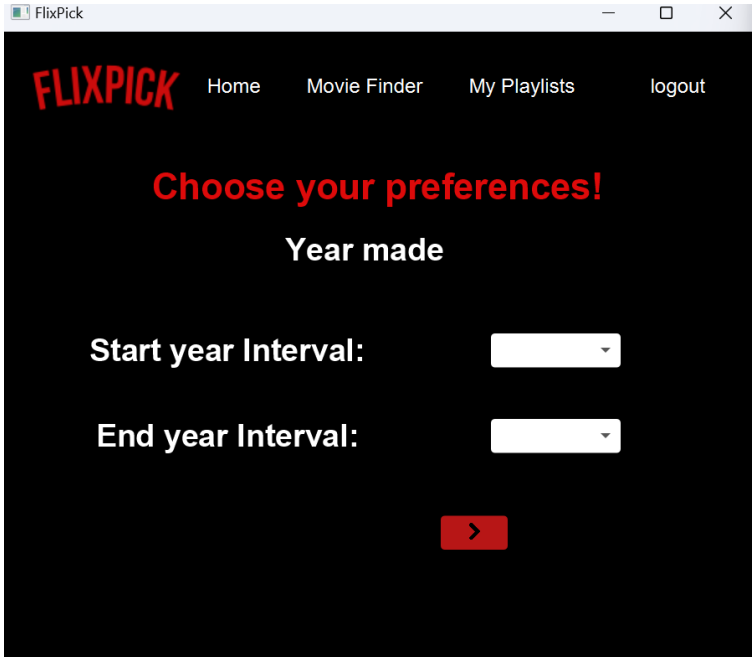

Figure 3.1 Year preferences

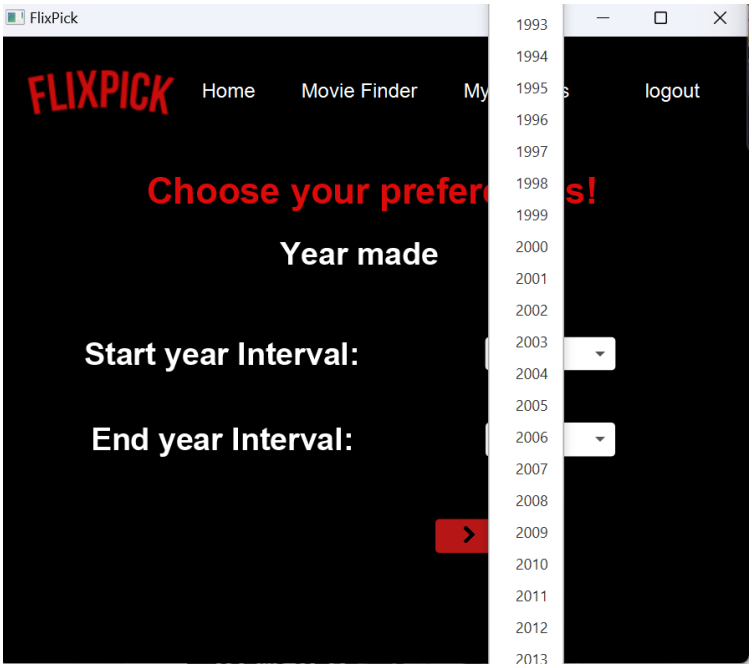

Figure 3.2 Choices of years

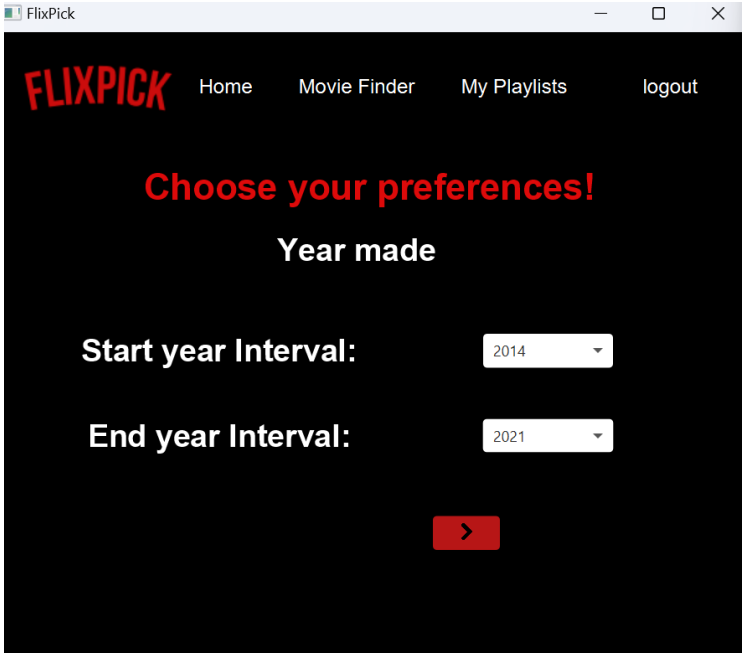

Figure 3.3 Preference Years Inputted

The movie finder commences with selecting a desired time interval (Figure 3.1). Users can choose both start and end years, introducing a flexible and customizable dimension.The system dynamically presents a list of selectable years (Figure 3.2), ensuring user satisfaction. Users input their chosen years (Figure 3.3) before effortlessly progressing to the next stage.

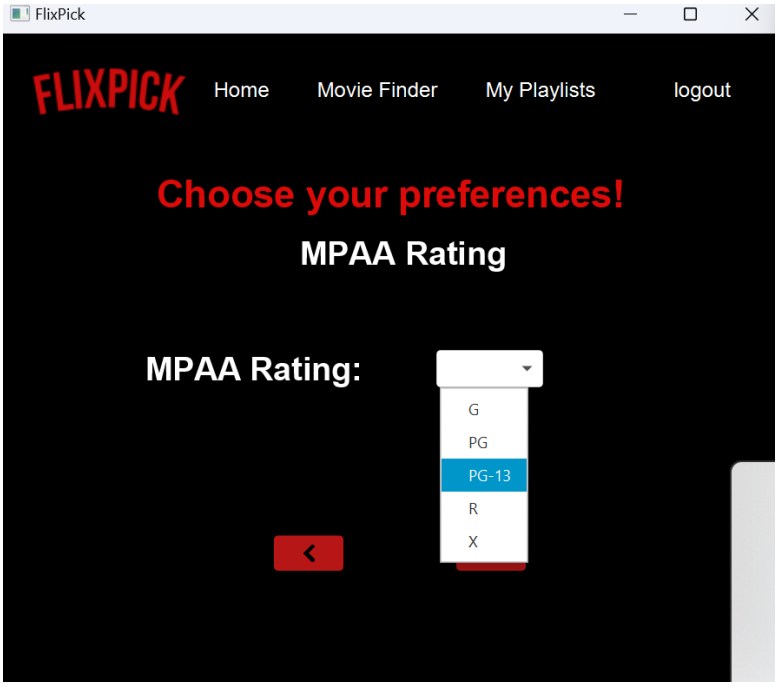

Figure 4.1 MPAA Rating preferences

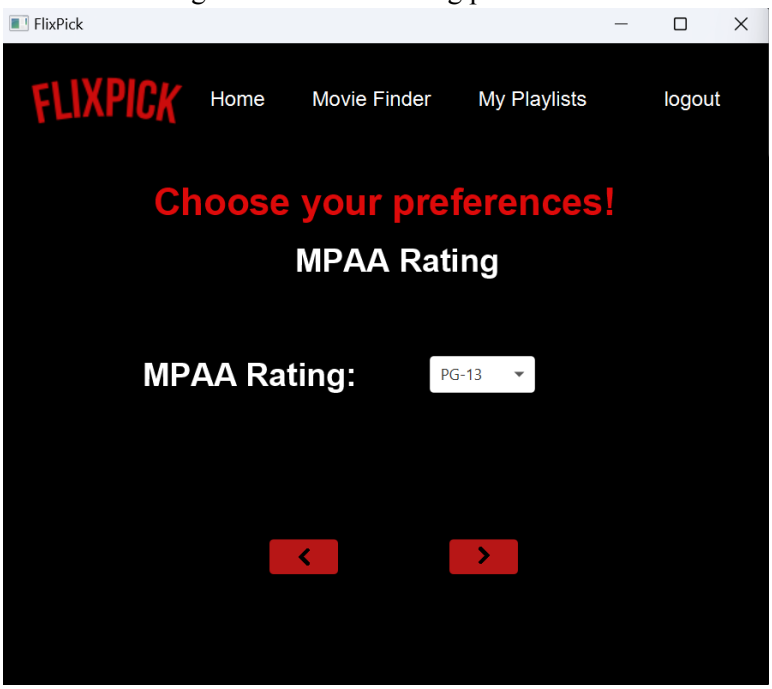

Figure 4.2 Inputted MPAA Rating

Continuing the personalized journey, users specify their preferred MPAA Ratings (Figure 4.1). The chosen ratings are prominently displayed (Figure 4.2), allowing users to review and modify preferences before advancing.

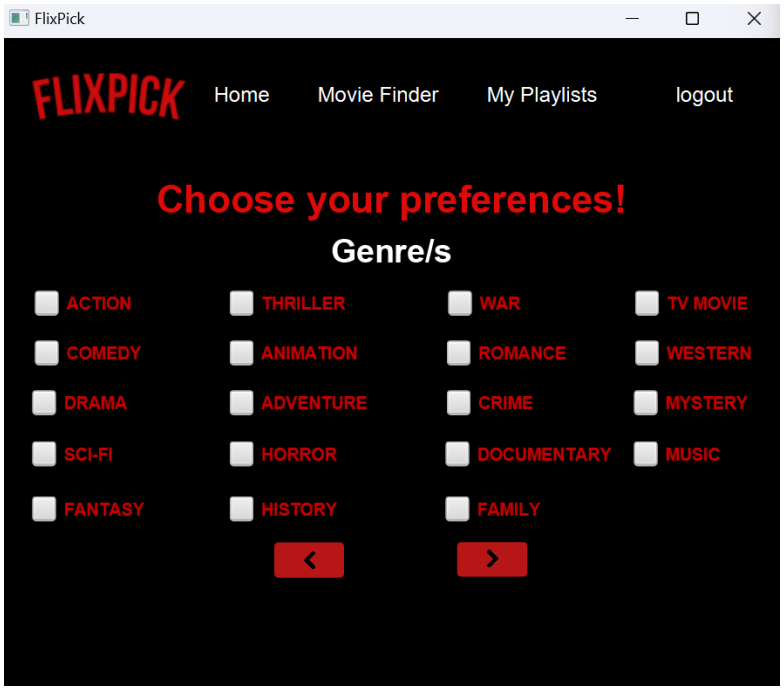

Figure 5.1 Genre Preferences Page

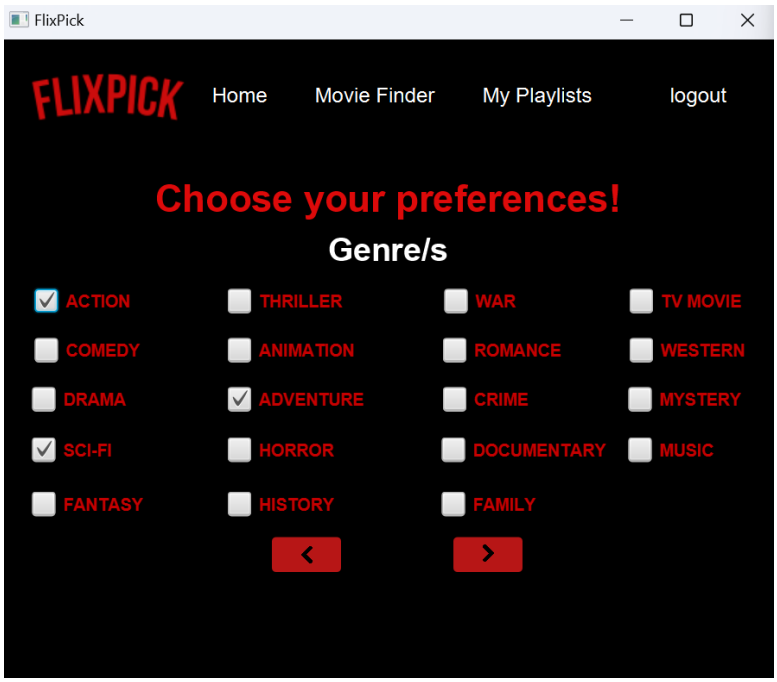

Figure 5.2 Inputted Genre Preferences

Diversity is embraced in the next step, where users select genres (Figure 5.1). Unrestricted by limitations, users can freely choose multiple genres (Figure 5.2), emphasizing inclusivity while ensuring a minimum of one genre is selected.

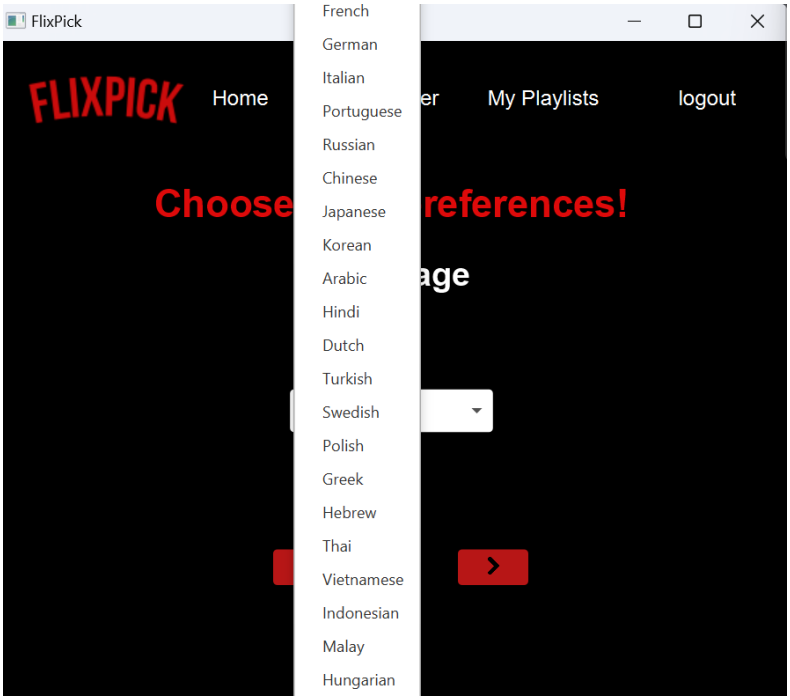

Figure 6.1 Language Page with List of Languages

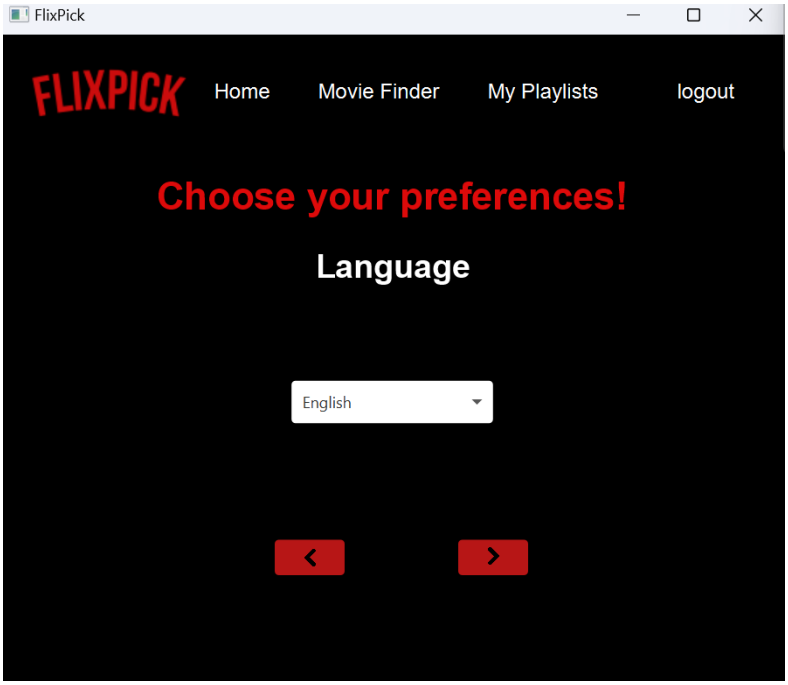

Figure 6.2 Inputted Language Preferences

The subsequent page introduces language preferences (Figure 6.1). A thoughtfully designed choice box accommodates diverse language options. Users confirm their language choice (Figure 6.2) before seamlessly proceeding.

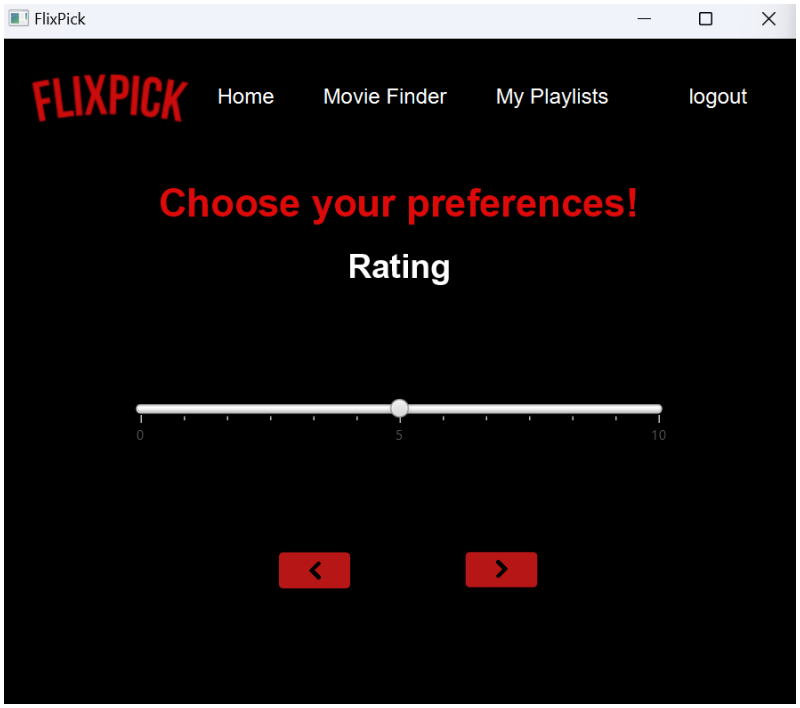

Figure 7.1 Rating Preferences Page

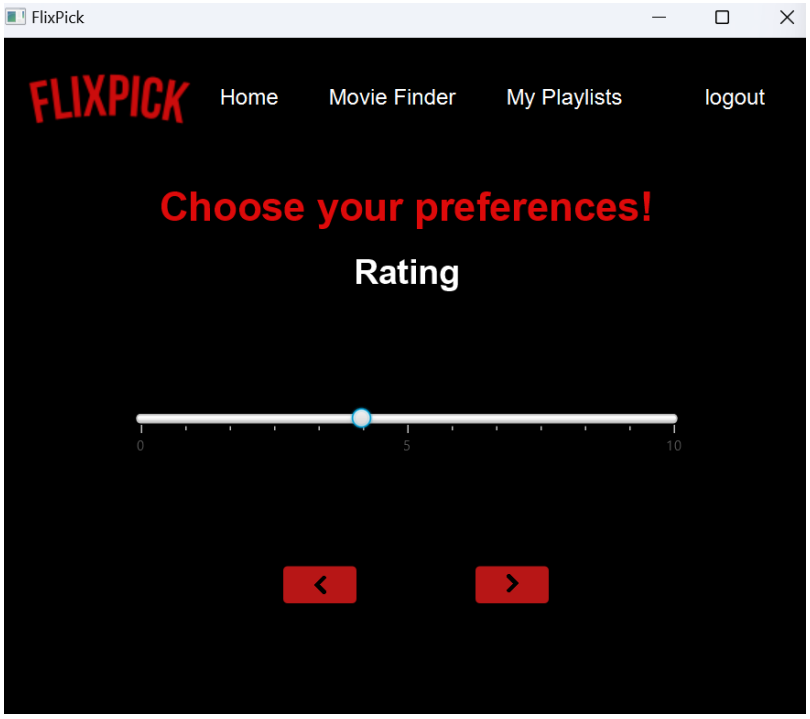

Figure 7.2 Inputted Rating Preferences

The next page showcases the preferred movie ratings of the user ranging from 0 to 10. The page provides a slider wherein the user can drag the pointer to their desired rating as seen in figure 7.2. Once the user is done, they may click the next button to continue.

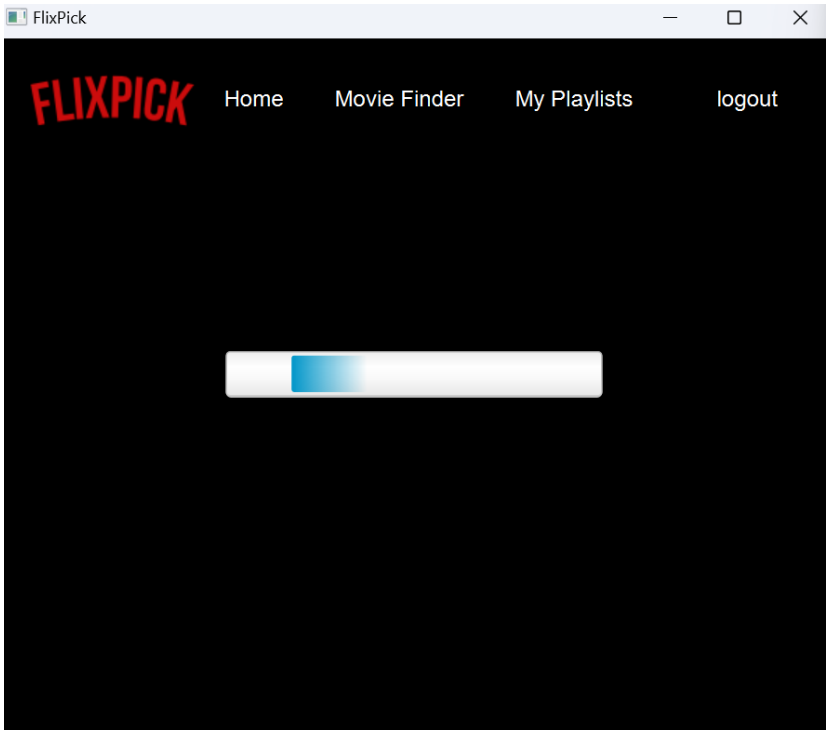

Figure 8. Progress Bar

The concluding rating page triggers a progress bar (Figure 8), providing users with visual feedback as the system processes their preferences and generates personalized movie recommendations.

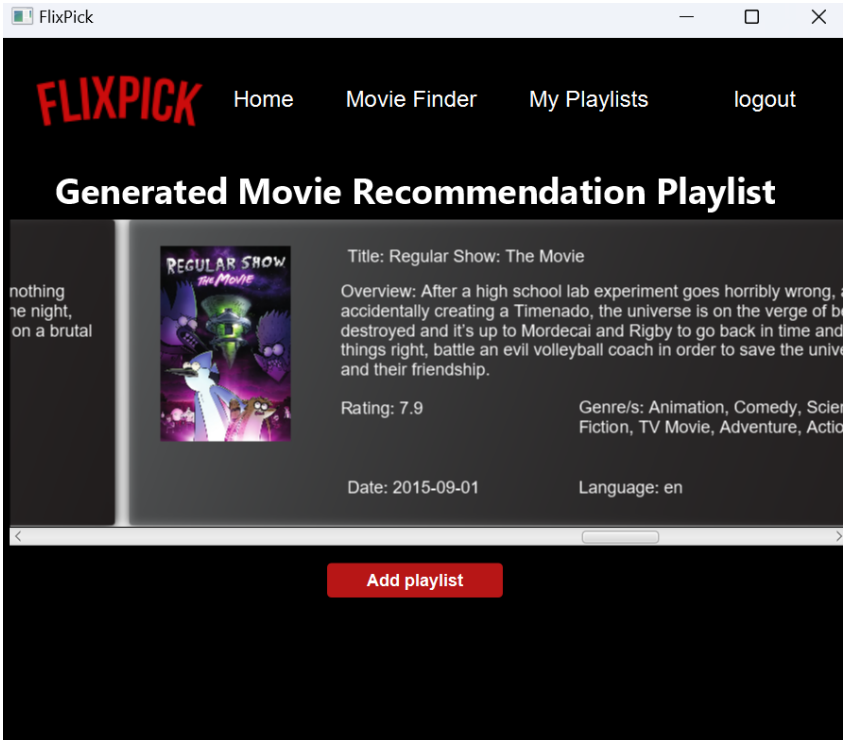

Figure 9.1 Recommended Movies Results

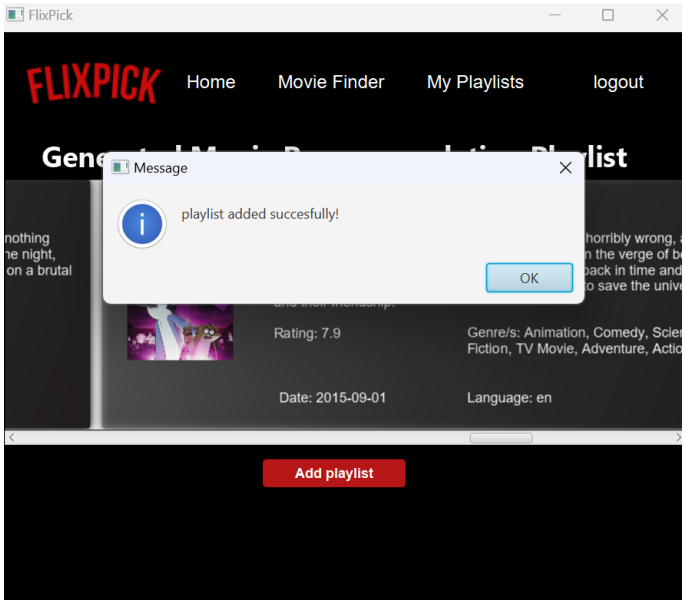

Figure 9.2 Success Prompt

Upon completion, users are greeted with the results page (Figure 9.1). This page showcases horizontally scrollable movie cards, each containing vital information such as poster, title, overview, genre/s, date, language, and movie rating. A prompt (Figure 9.2) confirms the successful addition of the playlist.

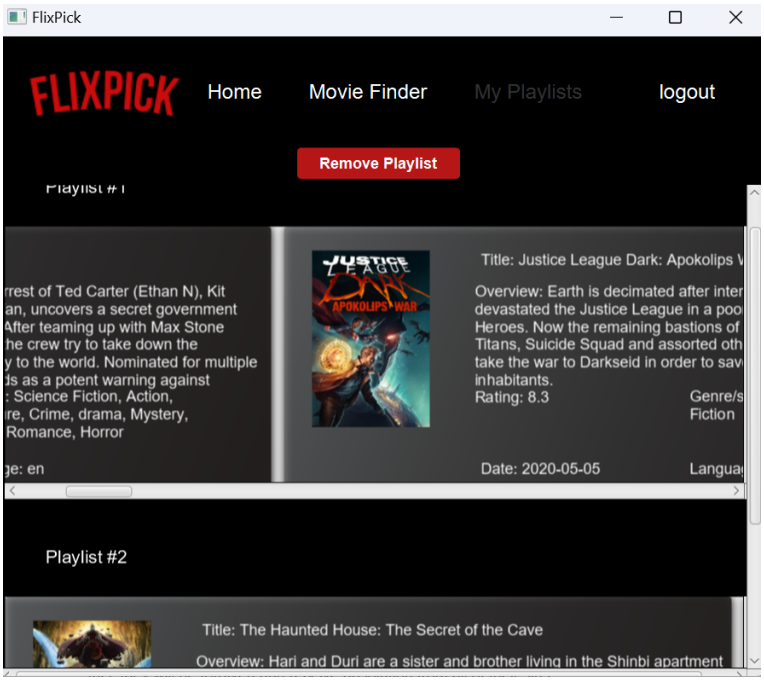

Figure 10.1 My Playlists Page

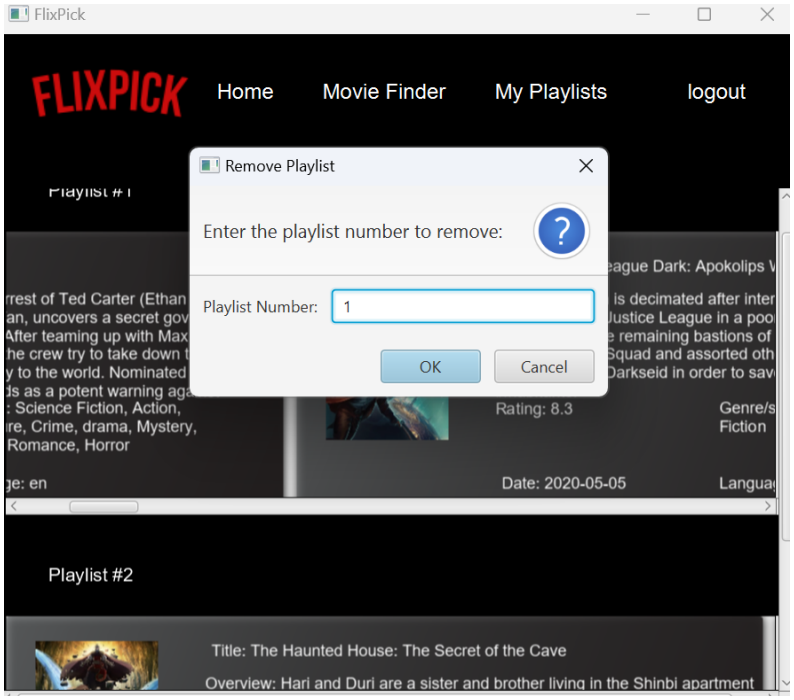

Figure 10.2 Remove with Playlist Number Prompt

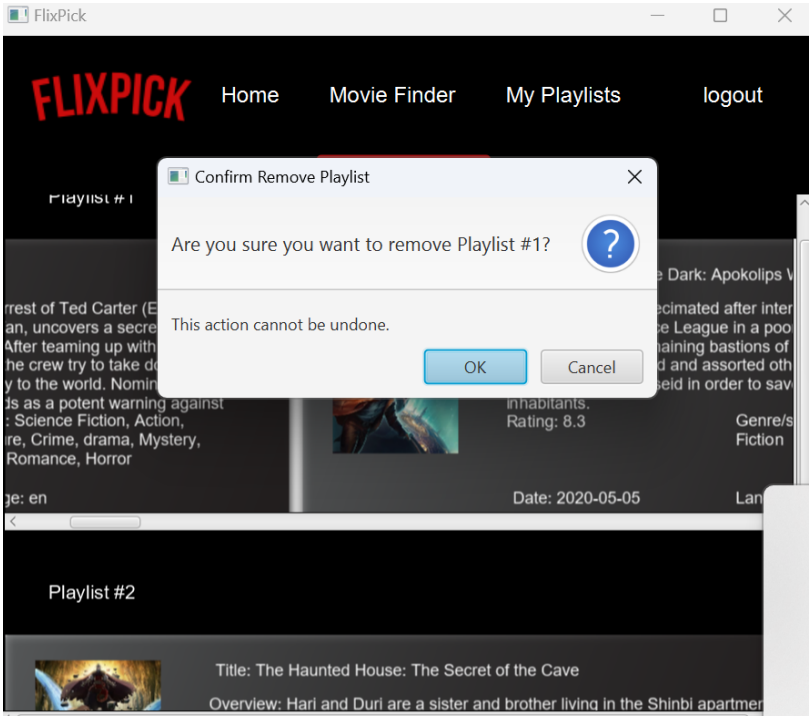

Figure 10.3 Confirmation of Removal Prompt

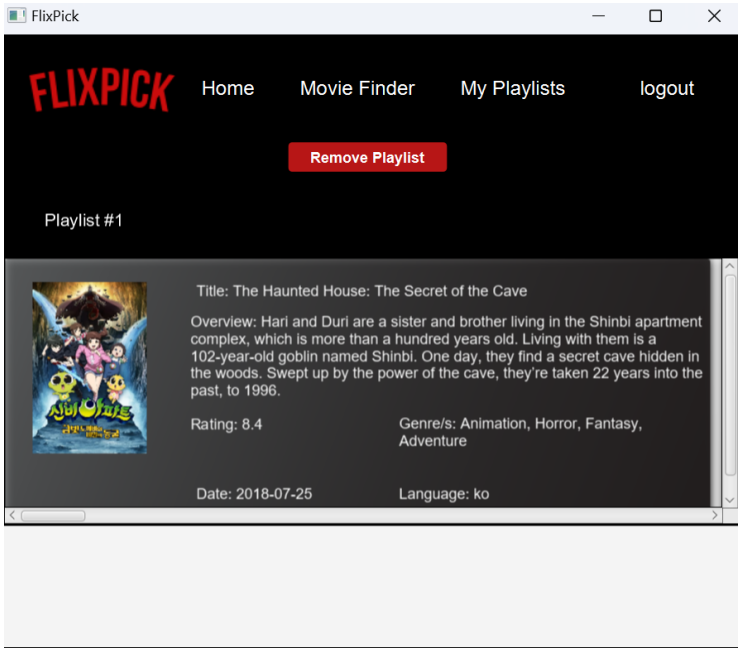

Figure 10.4 Updated Playlist Page

The user's journey extends to managing playlists. The My Playlists page (Figure 10.1) provides a chronological list of playlists. Users can remove playlists by inputting the playlist number (Figure 10.2), confirm removal (Figure 10.3), and witness the page dynamically update with the removed playlist (Figure 10.4).

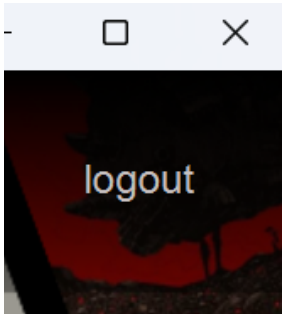

Figure 11. Logout button

Ensuring data persistence, users are prompted to press the logout button (Figure 11) before exiting the program, ensuring the preservation of any new changes made to their account. This thoughtful detail enhances the overall user experience.

This detailed walkthrough not only guides users through the FlixPick Movie Recommendation System but also highlights the system's commitment to user satisfaction, flexibility, and inclusivity. The seamless integration of user-friendly design and sophisticated functionality makes FlixPick a standout in the realm of movie recommendation systems.

V. Conclusion and Future Work

In conclusion, the development of the FlixPick Movie Recommendation System marks a significant advancement in addressing the challenges associated with navigating an extensive array of movie choices. The integration of sophisticated recommendation algorithms, secure user authentication, and efficient playlist management underscores FlixPick's commitment to simplifying the movie selection process and enhancing overall user satisfaction. Noteworthy achievements include the implementation of a robust user authentication system with advanced security measures like Bcrypt hashing to safeguard user data. The recommendation algorithm, centered around a decision tree, ensures personalized movie suggestions based on user preferences and viewing history, continuously adapting to user feedback.

The playlist management system, utilizing linked lists, empowers users to effortlessly organize and customize their movie selections, adding to the user-centric design. FlixPick's commitment to inclusivity is evident through features like ISO 639-2 code conversion for language preferences and MPAA rating conversion for different countries, catering to a diverse user base. Additional design considerations, such as using a stack data structure for navigation, contribute to the system's intuitive and user-friendly interface.

Looking forward, potential areas for improvement and expansion include enhancing the recommendation engine through advanced machine learning techniques, introducing collaborative playlist creation, extending the FlixPick experience to multiple platforms, and exploring partnerships for content enrichment. Future efforts may also focus on user testing, feedback incorporation for usability improvements, and implementing localization features to make FlixPick accessible to a global audience. In summary, FlixPick is poised for continuous refinement and evolution, promising an increasingly personalized and enriched movie-watching experience for users in the future.

VI. Contributions

Task	Member
Introduction	Aaron Dizon
Objectives	Aaron Dizon
Methodology	Samuelle Barja
Pseudocode	Samuelle Barja
UML	Aaron Dizon
IPO Chart	Samuelle Barja
Results & Discussions	Samuelle Barja
Conclusion	Samuelle Barja

Video Editing	Aaron Dizon
Presentation	Aaron Dizon
Demo	Samuelle Barja
Code	Samuelle Barja & Aaron Dizon

VII. References

[1] "Introduction to The Movie Database API - Integrate The Movie Database API in Python," *Educative*. https://www.educative.io/courses/movie-database-api-python/introduction-to-the-movie-database-api

[2] Jake, "The paradox of too much choice," *Everyday Psych*, Mar. 24, 2021. https://everydaypsych.com/paradox-much-choice/?fbclid=IwAR1fvGcu1lK_7K7LQP-b3TC9a1EOiDr18PxwkC2Y9ijgjtZG7_U_QgoUwyc

[3] R. Porter, "The Hollywood Reporter," The Hollywood Reporter, Apr. 06, 2022. [Online]. https://www.hollywoodreporter.com/business/business-news/streaming-users-overload-choices-1235125685/?fbclid=IwAR0wok5kmDWDCDUp-d7Ktz8KfrfOfUCdBZojr3bWmN7EmTeeFCtmfRiIl2s

[4] "ISO 639-2 Language Code List - Codes for the representation of names of languages (Library of Congress)," *www.loc.gov*. https://www.loc.gov/standards/iso639-2/php/code_list.php

[5] "Rating systems (Movies)," *Rating System Wiki*. https://rating-system.fandom.com/wiki/Rating_systems_(Movies)