# Steam Tag-Based Game Recommendation System

# Start Date

* 09/04/2023

# Pre-requisites

## Project based recommendations

* ASP.net MVC
* JS interactivity front end -> partial renders
  + Ability to enter a page submit a form which posts to the API and updates the database which is reflected on the page
* Reusability
* ASP.net for the API
* MSSQL database
  + 3-4 tables
* Bootstrap framework front end
* Use SCSS to potentially replace CSS
* Proper authentication all levels
* Restrict your scope or have hard lines drawn on scope creep
* Design some wireframes
  + Low quality black, white, grey mockups of web pages

# Technologies

* C#, ASP.NET Core API/MVC, Javascript, CSS, HTML, Bootstrap, Git/Github

# High Priority (Core Features)

|  |  |  |
| --- | --- | --- |
| **Feature** | **Feature description** | **Priority** |
| Steam Authentication | Integration of Steam’s OpenID authentication system to allow users to sign in with their Steam accounts securely | Necessary for the sake of enabling the recommendation page |
| Steam Library Analysis | Analyzing the user's Steam library data and identifying the most popular tags is crucial for providing personalized game recommendations based on user preferences. |  |
| Game Recommendations | Generating and displaying a list of games based on the user’s popular tags is the primary function of the website |  |
| Back-end and API | Developing the back-end using ASP.NET MVC and creating an API for handling user authentication, fetching Steam data, and generating game recommendations is necessary to support the core features of the website. |  |
| Database | Storing user profile data, user preferences, and cached game data from the Steam API in an MSSQL database is vital for efficient recommendation generation and user management. |  |
| Authentication and Security | Ensuring proper authentication and security measures are in place is crucial to protect user data and maintain a secure integration with the Steam API |  |

# Medium Priority (Enhanced Features)

|  |  |  |
| --- | --- | --- |
| User Profiles and Preferences | Enabling users to create profiles and manually adjust their tag preferences adds an extra layer of personalization and control over the recommendations. |  |
| Personalized Game Filters | Allowing users to further filter their recommendations by factors like price, review scores, or release dates provides a more refined and tailored recommendation experience. |  |
| Responsive Design | Implementing a responsive front-end design using Bootstrap, HTML, CSS, and SCSS ensures the website looks great on various devices and screen sizes, enhancing the overall user experience. |  |

# Pages

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Necessity** |
| Home/Landing Page | * Entry point for users * Provides an overview of the website's purpose and features. * Include a call-to-action (CTA) button for users to log in or sign up with their Steam account * Normal game search bar * Advanced search by categories * When advanced options are not ticked displays most popular games | Core |
| Login/Authentication Page/Button | * Allows users to securely sign in with their Steam account using Steam's OpenID authentication system. | Core |
| Personal Recommendation Page (Explore) | * Once logged in, users are directed to their dashboard where they can view their personalized game recommendations based on their popular tags. * Displays the list of recommended games along with relevant information (e.g., game images, descriptions, tags, and links to the Steam store) | Core |
| User Preferences Page (Profile) | * Allows users to adjust their tag preferences manually and apply additional filters (e.g., price, review score, release date) to further refine their game recommendations. | Optional |
| Game Details Page (Optional) | This page displays detailed information about a specific game, including its description, tags, reviews, and other relevant information. You can choose to implement this page to provide more in-depth information about each recommended game, or simply link to the game's Steam store page. | Core |
| About/FAQ Page (Optional) | This page provides background information about the website and answers frequently asked questions. It can help users understand the purpose of the project and how the recommendation system works. | Optional |

Project Timeline

**1. Project setup and planning:**

a. Set up project folder structure.

b. Initialize version control with Git.

c. Create a project timeline with deadlines for each task.

**2. Design:**

a. Sketch wireframes for each page (Home, Personal Recommendation, Game Page, About/FAQ/Disclaimer).

b. Choose a colour scheme and typography.

**3. Front-end development:**

a. Set up React with ASP.NET Core MVC

b. Install and configure Bootstrap and SCSS.

c. Develop the Home/Landing page:

- Implement Steam tag selection.

- Add a search functionality.

- Include a login button.

d. Develop the Personal Recommendation page:

- Display personalized game recommendations.

- Implement user preferences and filters.

e. Develop the Game Page:

- Display game details, price, description, associated tags, release date, development company, and Steam link.

f. Develop the About/FAQ/Disclaimer page:

- Add general site information, data usage details, and common questions.

**4. Back-end development and API integration:**

a. Implement Steam OpenID authentication for user login.

b. Fetch data from the Steam API.

c. Develop game recommendation logic based on popular user tags.

d. Set up API endpoints for front-end communication.

**5. Database setup:**

a. Create an MSSQL database.

b. Design tables for user data, preferences, and cached game data.

c. Establish relationships between tables.

**6. Integration and testing:**

a. Connect the front-end to the back-end and API.

b. Implement authentication and security measures.

c. Test the website on various devices and browsers.

d. Fix any issues and optimize performance.

**7. Deployment:**

a. Deploy the website to a web hosting service or cloud platform.

b. Set up a custom domain (optional)

**8. Post-launch:**

a. Monitor website performance and user feedback.

b. Address any issues and make updates as needed.

# User flow

* Home/landing page
  + Also allows users to manually search for games with the search bar
  + Allows users to search games with advanced filtering options
* Game page
  + Displays further information related to a game click from a user
  + Provides a link to the steam store page for the game
* Explore page
  + Only accessible once logged in
  + Provides game recommendations based on your top steam tags
  + Next and previous buttons to return or progress through recommendation list
* Navbar
  + Provides links to Home, Game, and Explore page

# Database Schema

1. Games:
   1. id (int, primary key): Unique identifier for the game.
   2. steam\_appid (int): Steam application ID
   3. type (varchar): Product type
   4. name (varchar): Name of the game.
   5. is\_free (bit): Whether the game is free (1) or not (0).
   6. detailed\_description (text): Detailed description of the game.
   7. about\_the\_game (text): Information about the game.
   8. short\_description (varchar): Short description of the game.
   9. header\_image (varchar): URL of the header image for the game.
   10. release\_date (date): Release date of the game.
   11. total\_recommendations (int): Total number of recommendations for the game.
2. Developers:
   1. id (int, primary key): Unique identifier for the developer.
   2. name (varchar): Name of the developer.
3. Publishers:
   1. id (int, primary key): Unique identifier for the publisher.
   2. name (varchar): Name of the publisher.
4. Game\_Developers:
   1. game\_id (int, foreign key): References the Games table (id).
   2. developer\_id (int, foreign key): References the Developers table (id).
5. Game\_Publishers:
   1. game\_id (int, foreign key): References the Games table (id).
   2. publisher\_id (int, foreign key): References the Publishers table (id).
6. Platforms:
   1. id (int, primary key): Unique identifier for the platform.
   2. name (varchar): Name of the platform (e.g., "Windows", "Mac", "Linux").
7. Game\_Platforms:
   1. game\_id (int, foreign key): References the Games table (id).
   2. platform\_id (int, foreign key): References the Platforms table (id).
8. Price\_Overview:
   1. id (int, primary key): Unique identifier for the price overview.
   2. game\_id (int, foreign key): References the Games table (id).
   3. currency (varchar): Currency code (e.g., "USD", "EUR", "AUD").
   4. discount\_percent (int): Discount percentage for the game.
   5. final\_formatted (varchar): Final formatted price for the game.
9. Screenshots:
   1. id (int, primary key): Unique identifier for the screenshot.
   2. game\_id (int, foreign key): References the Games table (id).
   3. path\_thumbnail (varchar): URL to the thumbnail version of the screenshot.
   4. path\_full (varchar): URL to the full-sized version of the screenshot.
10. Movies:
    1. id (int, primary key): Unique identifier for the movie.
    2. game\_id (int, foreign key): References the Games table (id).
    3. name (varchar): The name of the movie.
    4. thumbnail (varchar): The URL of the movie thumbnail.
    5. webm\_480 (varchar): The URL of the WebM format movie at 480p resolution.
    6. webm\_max (varchar): The URL of the WebM format movie at the highest resolution.
    7. mp4\_480 (varchar): The URL of the MP4 format movie at 480p resolution.
    8. mp4\_max (varchar): The URL of the MP4 format movie at the highest resolution.

header\_image (varchar): The URL of the header image

highlight (bool): true or false whether the video is highlighted

type (varchar): Product type

recommendations": { "total": 133542 }

# Issues

* API request service down
  + Due to incomplete documentation
  + Retrieve new endpoint
* Total recommendations stored in different api call to game details
  + Second request made to AppReviews to retrieve information
* Max requests of reviews limited to 100
  + Scrapping of as information wouldn’t be accurate
* Issue using Object Relational Mapping due to a connection string stored in source code
  + Moving connection string to external JSON
* Connection string could not be retrieved for scaffolding purposes
  + Introduction of connection string to root
* Server certificate not trusted -> disabled by adding command to the connection string
* Automapping too complicated and intended for complex to simple objects not vice versa
* [DatabaseGenerated(DatabaseGeneratedOption.None)] required to prefill the ID key for the game object
* Not all objects are the same and not all have the same data even when the fields are shared

## Mapping Plan

1. Open a database connection: Before you can perform any database operations, you need to open a connection to the MSSQL database. You can use a data access technology such as Entity Framework, ADO.NET, or Dapper to open a connection.
2. Map the C# object to the database table: You can use a mapping library like AutoMapper to map the properties of the C# object to the columns of the corresponding table in the MSSQL database.
3. Prepare the SQL statement: Once you have mapped the C# object to the database table, you can prepare an SQL statement that will insert the data into the table. The exact SQL statement will depend on the specific database schema, but you can use a parameterized query to avoid SQL injection attacks.
4. Execute the SQL statement: After you have prepared the SQL statement, you can execute it using your data access technology. You can use a method like ExecuteNonQuery() to insert the data into the database.
5. Close the database connection: After you have finished adding the data to the database, you should close the database connection to free up resources.

## Mapping Test Cases

|  |  |  |  |
| --- | --- | --- | --- |
| **Case** | **Description** | **Pass/Fail** | **Reason** |
| Duplicate prevention | Values in relevant columns are unique. E.g. games with shared devs/publ. are not adding duplicate records |  |  |
| Updating objects | If a values within an object changes the changes will be reflected within the database |  |  |
| Simple add | New objects are correctly added to the database |  |  |
| Type: game | Only objects of the game type are added into the database |  |  |
| Completeness | Data added to the database is complete |  |  |
| Record count check | Ensures that the expected number of records in the destination database matches that of the source database |  |  |
| Identity check | Allocated identities are correctly assigned |  |  |
| Key checking | Foreign/primary keys are maintained |  |  |
| Data consistency | Data is the same in the destination database as its source |  |  |
| Mapping | Objects are correctly mapped to one another |  |  |