Comprehensive System Architecture and Implementation for a Gaming Platform

Introduction:

In today's digital landscape, where digital entertainment plays a pivotal role, there is a growing demand for advanced gaming platforms that deliver seamless user experiences and robust management features. This report details the development of a state-of-the-art gaming platform, modeled after Steam, that caters to the modern requirements of both gamers and developers. It provides an in-depth exploration of the system's architecture, implementation, and functionalities, designed to support extensive user interactions such as game searching and account management, while also equipping administrators with efficient tools to manage game inventories and user data.

Built on a tech stack comprising React, Flask, and MySQL, this platform embodies the successful integration of web technologies to foster a responsive and secure user environment. The platform is meticulously engineered to include front-end components that assist in user registration and game searches, alongside back-end services that ensure data integrity and manage API requests. Detailed in this report are the various features that make up the platform, illustrating how the integration of these technologies delivers an enhanced user experience and simplifies administrative processes.

Planned Implementation

Through the UI, the end user can retrieve game data by searching for a game studio, game name, or genre. The user can further filter the result by selecting a price range, game release time window, and game rating range. As a bonus, if time permits, we will also implement a sorting algorithm for users to sort the games by release time, name, price, and rating. Another bonus we plan to implement if time permits is user authentication. Enable users to edit their own list.

1.Data Collection and Cleaning

Data Collection: Utilize a Python script with the *requests* and *lxml* libraries to systematically retrieve and interpret data from Steam's search results and individual game pages. The script employs the *requests* library to make HTTP requests, mimicking a browser through specified cookies and headers. This method helps in accessing the dynamic content of Steam's website reliably.

Data Parsing and Storage:

The script extracts crucial game details using a combination of regular expressions for initial parsing and XPath for more structured extraction from the HTML content. The parsed data includes game titles, tags, platforms, reviews, and pricing information. This information is then structured for storage in a MySQL database, which includes tables designed to store data about users, games, and user favorites. This structured approach facilitates efficient data retrieval and manipulation for various applications like recommendations or analytics.

```
GameName: Iron Harvest: - Operation Eagle DLC
GameName: Slime 3K: Rise Against Despot
GameName: Warhammer: Vermintide 2 - Sister of the Thorn Cosmetic Upgrade
GameName: Tom Clancy's Rainbow Six® Siege - Ruby Weapon Skin
GameName: Exo One
GameName: FSX Steam Edition: Airbus Series Vol. 1 Add-On
GameName: STAR WARS™ Battlefront (Classic, 2004)
GameName: Bear and Breakfast
GameName: Armello - Rivals Hero Pack
```

steam game

GameId	game name	Game tag	game platform	positive rating	game price	
G001	Dead by Daylight	Horror, Survival Horror, Multiplayer, Online Co-Op, Co-op, Survival	win	80%	10.5	72.0
G002	Supermarket Simulator	Simulation, Economy, Management, Immersive Sim, Capitalism, Trading	win	94%	6.6	72.0
G003	The Outlast Trials	Horror, Multiplayer, Co-op, Survival Horror, Psychological Horror, First-Person	win	93%	17.3	84.0
G004	Oxygen Not Included	Colony Sim, Base Building, Survival, Resource Management, Building, Singleplayer	linux, win, mac	96%	7.4	85.0
G005	RiichiCity - ACG mahjong games	Casual, Sexual Content, Psychological Horror, Thriller, Anime, Card Game	win	54%	0.0	77.0
G006	Resident Evil 7 Biohazard	Horror, Survival Horror, First-Person, Singleplayer, Atmospheric, Zombies	win	94%	12.6	78.0
G007	Human Fall Flat	Co-op, Funny, Puzzle, Adventure, Physics, Sandbox	win, mac	94%	7.4	54.0

Functionalities:

get_data(num): fetching the HTML content of Steam's game listings from a specific starting point determined by the pagination parameter *num*.

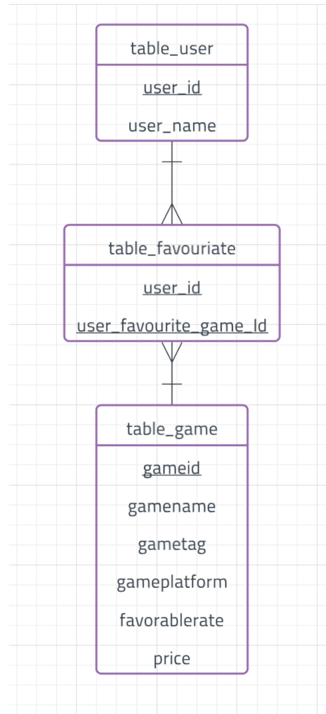
parse_data(data): extracting detailed game information from the HTML content retrieved by *get_data.*

save_data(title, label_list, plat_list, comment, prices): saving the parsed and processed game data into a structured format.

Database Design

In an effort to enhance performance and scalability, the dataset containing information on 5,000 games was thoughtfully segmented into two separate databases. This strategic division is aimed at optimizing data management and accelerating response times, which are crucial for a scalable system infrastructure.

Database Schema Overview:



- Database 1:

table_user1: Stores user data.

table_game1: Contains game data such as tags, platforms, positive ratings, and price.

table_favorite1: Manages user favorites, establishing a relationship between users and games.

- Database 2:

table_user2: Stores user data.

table_game2: Contains game data, similar to table_game1, with tags, platforms, positive ratings, and price.

table_favorite2: Manages user favorites, linking users to their preferred games.

This dual-database structure not only facilitates more efficient data retrieval and management but also supports future growth and the addition of more games and user data without degrading performance.

```
| G996   | Arma Reforger                                                                  | Action,  Simulation,  Shooter,  Strategy,  Mil
itary,  FPS                                               | win                              |          0.63 |  12.8 |
| G997   | Call of Duty?                                                                  | Action,  Gore,  Violent
                                                          | win                              |          0.56 |  16.6 |
| G998   | Warhammer                                                                      | Action,  Gore,  Violent,  Games Workshop,  Fan
tasy,  Co-op                                              | win                              |          0.77 |   5.4 |
| G999   | Anno 1800? - Season 1 Pass                                                     | Simulation,  Strategy
                                                          | win                              |          0.63 |  15.9 |
+--------+--------------------------------------------------------------------------------+-----------------------------------------------
----------------------------------------------------------+----------------------------------+---------------+-------+
2500 rows in set (0.01 sec)

mysql> SELECT * FROM database1.table_favourite1;
Empty set (0.00 sec)
```

```
mysql> SELECT * FROM database1.table_user1;
+---------+-----------+
| user_id | user_name |
+---------+-----------+
| 001     | Jack      |
| 002     | Lucy      |
| 003     | Emma      |
| 004     | Noah      |
| 005     | Olivia    |
| 006     | Liam      |
| 007     | Sophia    |
| 008     | William   |
| 009     | Ava       |
| 010     | James     |
+---------+-----------+
10 rows in set (0.00 sec)
```

2. Backend Functionality

Functionalities:

The steam platform function should be able to: search games by name, tags, and platform and sort by match and sort by the match score by default; manage the database by adding user, adding/dropping user favorite games; retrieve and show the user's favorite games.

Searching:

For the Search functionality, functions: *SearchName()*, *SearchTag()*, and *SearchPlatform()* are implemented in python environment. The functions take string inputs: name, tag, platform, and return a JSON object for showing to users in platform later, sorted basing on match score, or return 'no match'.

Python package mysql.connector, and function *connect_fetch(i)* is used to connect to the SQL server and access to the databases.where *i* is the index assigned to each database.

The difficulty of matching and filtering games basing on user inputs is that SQL query is unable to do delicate string similarity comparison. However, python package *fuzzywuzzy* can assign a match score from 0(low similarity) to 100(high similarity) between two strings ignoring the letter case. The algorithm employs Levenshtein Distance for match score by calculating the minimum number of edits required to transform one string into another, ensuring more relevant results.

Therefore, the functions implement double filtering for searching functionality. First filtering is done by SQL query. Input string is split by space or comma to a list ignoring stop words. Iterating through the list, functions first search the games in the corresponding tables in databases by SQL query which the name or tag or platform that includes the string user input. (Example SQL query: 'SELECT * FROM table_game1 WHERE LOWER(gamename) LIKE LOWER("%atomic%");')

The selected games are imported to python shell as pandas data frame for a more delicate filtering. The game information in the data frame includes *gameid, gamename, gametag, gameplatform, favorablerate,* price.

Second filtering is done by python function *fuzz* from *fuzzywuzzy* and match score is assign to game data frame as *match*. It's to notice that *match* doesn't exist in database initially; it's only

assigned when the function loads the selected games to the data frame, compares the game's value similarity with the input sting, then assign a match score to each game during the second filtering. (Example fuzz call: bool(fuzz.token_set_ratio(x,y) > 60))

Finally, the data frame of game information is sorted basing on *match* by default. Take *SearchName()* as an example, after SQL query selected games are loaded to python shell, games whose names having match score below 60 are dropped. In the end, top 50 matches of the game data frame are kept and returned as JSON object.

Same method applies to *SearchTag()* and *SearchPlatform()*, but the logistic of matching is slightly different: *SearchTag()* takes comma as separator, search by platform by match over 85% or inclusion. *SearchPlatform()* takes comma as separator, search by platform by match over 75% or inclusion.

Example Screen shoot: the out put here is data frame instead of JSON for better illustration

```
SearchName('cdAHAD')
✓ 0.0s
'no match'
```

```
SearchName('atomic')
✓ 0.0s
```

	gameid	gamename	gametag	gameplatform	favorablerate	price	match
0	G447	Atomic Heart	Sexual Content, Horror, FPS, Puzzle, Myste...	win	84%	30.5	100
1	G3189	Atomic Heart - Annihilation Instinct	Action, Adventure, RPG, Gore, Violent, Ex...	win	70%	5.1	100
2	G1219	Atomicrops	Action, Indie, Farming Sim, Roguelike, Bul...	win	93%	7.4	75
3	G3462	Atomicrops	Action, Indie	win	85%	2.8	75
4	G3984	Atomicrops	Action, Indie	win	87%	2.8	75

```
def SearchName(name):
    name_clean = name.replace(',', ' ')
    name_list = name_clean.split()
    stopwords = ['a', 'an', 'and', 'are', 'as', 'at', 'be', 'by', 'for',
                 'if', 'in', 'is', 'it', 'no', 'of', 'on', 'or', 'such', 'that', 'the', 'to']
    name_l = [i for i in name_list if i not in stopwords]
    ss = ''
    for n in name_l:
        s = f' LOWER(gamename) LIKE LOWER("%{n}%")'
        ss = ss + s + ' OR'
    statement1 = ('SELECT * FROM table_game1 WHERE' + ss)[:-3]
    statement2 = ('SELECT * FROM table_game2 WHERE' + ss)[:-3]
    conn1, cursor1 = connect_fetch(1)
    conn2, cursor2 = connect_fetch(2)
    df1 = pd.read_sql(statement1, conn1)
    df2 = pd.read_sql(statement2, conn2)
    cursor1.close()
    conn1.commit()
    conn1.close()
    cursor2.close()
    conn2.commit()
    conn2.close()
    df = pd.concat([df1, df2], axis=0)
    def condition(x):
        if fuzz.token_set_ratio(x, name) > 60:
            return True
        elif name in x:
            return True
        else:
            return False
    dfMatched = df[df['gamename'].apply(condition)]
    if len(dfMatched) == 0:
        return ('no match')
    dfMatched['match'] = dfMatched['gamename'].apply(lambda x: fuzz.token_set_ratio(x, name))
    dfMatched = dfMatched.sort_values(by='match', ascending=False)
    dfMatched = dfMatched.reset_index(drop=True)
    dfMatched = dfMatched.drop(dfMatched.index[maxrow:])
    js = dfMatched.to_json(orient="split")
    return js
```

Manage Database:

For managing databases, functions need to add or deleted user, enable a user to favorite or unfavorite a game, retrieve a user's all favorite games, add, delete, or edit a game. Before any editing is done in the databases, the functions need to check if the user request is legitimate, so function SQL query to check ahead.

The difficulty is to expect any exception or ill-formed input from the users and make sure the function run properly and satisfying all constrains in databases. For example, functions need to check if the editing satisfies the database key constrains, or if the according primary key exits in the table. Functions will return string notifying success or failure.

Functions for users:

Favor(gameid, user_id) and *unFavor(gameid, user_id)*

gameid must be in format "G001" to "G999", or "G1000" to "G9999"

check if game in table_game, check if user in user table_user,

check if game already favored in table_favourite pair with this user

return for each situation: ‘game id non-existent’, ‘game already favorited’, ‘user non-existent’

add or drop gameid to the table_favourite in the database(1 or 2) based on where the game exists

return ‘game added’ or ‘game deleted’

retrieveAllFavGames(user_id)

if user does not exist in table_user1 and table_user1, return ‘user non-existent’,

Join table_favourite and table_game in all databases on foreign key gameid

and select gamename based on user_id

else return a JSON array,

JSON array is empty if the user doesn’t have favorite game

Functions for managers:

insertGame(glist)

input a list glist: ["gameid", "gamename", "gametag", "gameplatform", "favorablerate", price]

use empty string “” in the list if the value is null

gameid must be in format "G001" to "G999", "G1000" to "G9999"

here game is hashed basing on gameid

check if game exist in the corresponding data base basing on the hash table

return ‘success’ or ‘game id invalid’, or ‘game id taken’

deleteGame(gameid)

input string gameid

gameid must be in format "G001" to "G999", "G1000" to "G9999"

here game is hashed basing on gameid

check if game exist in the corresponding data base basing on the hash table

delete game if exists

return messages ‘success’ or ‘game non-existent’

updateGame(gameid, gdict)

gameid cannot be changed by this function, use insertGame or deleteGame instead

update game in the game table basing on dictionary gdict: {key name: new value, key name: new value, …}

use empty string “” if the value is null

return messages ‘success’ or ‘gameid non-existent’

addUser(user_id, user_name)

interact with user table in all databases

check if user in table by SQL query first based on primary key user_id

if user is already in database, return ‘already exists’

add user to user table by SQL query return ‘user added’

deleteUser(user_id)

delete user in all databases with user_id

if success, return ‘user deleted’

or if user doesn’t exist, return ‘user non-existent’

Example screen shot:

```
def updateGame(gameid, gdict):

    # check if gameid valid
    gamei = int(gameid[1:])
    if gamei <= 2500:
        inx = 1
        conn, cursor = connect_fetch(1)
    elif 2500 < gamei <= 9999:
        inx = 2
        conn, cursor = connect_fetch(2)
    else:
        return ('game non-existent')

    # check if gameid exists
    sql_query1 = f"SELECT 1 FROM table_game{inx} WHERE gameid = '{gameid}';"
    cursor.execute(sql_query1)
    result1 = cursor.fetchall()

    if result1:
        for key, value in gdict.items():
            if key == 'gameid':
                break
            if value == '':
                sql_query2 = f"UPDATE table_game{inx} SET {key} = NULL WHERE gameid = '{gameid}';"
            elif key == 'price':
                sql_query2 = f"UPDATE table_game{inx} SET {key} = {value} WHERE gameid = '{gameid}';"
            else:
                sql_query2 = f"UPDATE table_game{inx} SET {key} = '{value}' WHERE gameid = '{gameid}';"
            cursor.execute(sql_query2)
        cursor.close()
        conn.commit()
        conn.close()
        return ('success')

    else:
        cursor.close()
        conn.commit()
        conn.close()
        return ('gameid non-existent')
```

3. Tech Stack

Frontend Implementation (React)

The first frontend instance runs on port 3000 and includes two main components:

- Home.js: This component serves as the entry point for user registration and login. It utilizes axios for API calls to the Flask backend, handling user creation and authentication. The user interface allows users to switch between 'register' and 'login' tabs and perform the corresponding actions.
- SearchPage.js: This component is designed for searching and managing user game collections. It supports various search filters like game name, platform, and tags, and interacts with the Flask backend to fetch and display results. The page also includes functionality to sort search results and manage user favorites, leveraging React's state management to update the UI dynamically based on user interactions and search results.

The second frontend, running on port 3001, is centered around a single main component:

- Manager.js: This management interface is crucial for administrative tasks such as inserting, updating, and deleting game records as well as user management. It includes forms for inputting game details, user details, and performing actions like adding or removing games from favorites. Similar to the first frontend, it uses axios to make API calls to the Flask backend, handling CRUD operations on games and users effectively. The component uses state hooks extensively to manage form inputs and responses, ensuring a responsive user experience.

Backend Application (Flask)

Setup

The Flask app is configured with Cross-Origin Resource Sharing (CORS) enabled to allow secure cross-origin requests, which is crucial for the frontend and backend interaction across different ports or domains. The application defines a function, get_db_connection(DB), which is responsible for establishing a connection to one of multiple MySQL databases based on the input parameter.

API Endpoints

- Game Management:

- Insert Game (/insertgame/<parameters>): Allows insertion of a game into the database with parameters for game details. It processes special cases where "null" is transformed to None in Python for proper database insertion.
- Delete Game (/deletegame/<gameid>): Facilitates the deletion of a game record identified by its unique game ID.
- Update Game (/updategame/<gameid>): Updates game information based on parameters passed through the query string.

- User Management:

- Add User (/adduser/<user_id>/<user_name>): Adds a new user with an ID and name.

- Delete User (/deleteuser/<userid>): Deletes a user from the database using the user ID.
- Update User (/updateuser/<userid>/<username>): Updates user details in the database.

- Search Functionality:

- Search by Name (/games/searchname/<name>): Searches the database for games matching the provided name.
- Search by Tag (/games/searchtag/<tag>): Retrieves games associated with a specific tag.
- Search by Platform (/games/searchplatform/<platform>): Finds games available on a specified gaming platform.

- Favorites Management:

- Add to Favorites (/favourite/<user_id>/<game_id>): Allows users to mark a game as favorite.
- Remove from Favorites (/unfavourite/<user_id>/<game_id>): Removes a game from the user's list of favorites.
- Retrieve All Favorites (/allfavgames/<user_id>): Fetches all favorite games of a user.

Security and Error Handling

The application handles errors such as database connection failures and invalid API requests gracefully, providing error messages and appropriate HTTP status codes. Security measures include validation of incoming data to prevent SQL injection and other common security threats.

Deployment and Maintenance

The Flask application is set to run in debug mode for development purposes with the ability to handle real-time errors and provide detailed logs. For production, it is recommended to disable debug mode to ensure security and performance.

Individual Contribution:

Zixian He: (Data Collection and Cleaning)

developing a system that efficiently gathers, stores, and manages data from Steam's gaming platform. Using a Python script, the system will automatically scrape game data directly from Steam's website and store it in a MySQL database. This database will also allow manager modify users' preferences for games, enabling robust data analysis and management.

Zhaoyi He: (Backend Functionality)

establishing the backend functionality needed for the stream platform, including searching functionality(search by game name, game tag, game platform, retrieve user's all favorite games) and database managing(add/drop user, add/drop/edit game, favorite/unfavorite game).

Gefei Yan: (Frontend Implementation and Backend Application)

working on integrating a React frontend and Flask backend with MySQL to enhance data management and interactions efficiently. The system boasts extensive user and game management features such as secure registration, login, advanced searching, and favorites management which makes the architecture suitable for scalable web applications that demand precise administrative control and tailored user functionalities.

Learning outcomes:

- Data Collection and Cleaning

Handling dynamic web content involves addressing the challenges posed by the constantly changing elements on Steam pages, which can complicate traditional static scraping methods. Maintaining data consistency is crucial to ensure that the scraped data adheres to the structured requirements of the MySQL database, preserving data integrity and accuracy. Additionally, scalability is essential, as the system must be robust enough to manage increasing volumes of data and user requests without compromising on performance.

- Backend Functionality

For learning outcomes of this part, Python coding skill is applied of string matching, comparing, data frame managing, data filtering and sorting, and data managing is applied including hashing, executing SQL statement for implementing the functions.

- Frontend Implementation and Backend Application

Ensuring robust security measures are in place is crucial; all data transmissions should be encrypted using HTTPS and comprehensive input validation implemented to prevent SQL injection and XSS attacks. Additionally, developing a robust error-handling framework is essential for effectively categorizing and managing errors—whether they are client-side, server-side, or database errors—ensuring critical issues are logged and users are informed as necessary. Maintaining a smooth and responsive user experience is also vital; this can be achieved by optimizing frontend performance through efficient design and implementation practices.

Conclusion:

The development of this gaming platform, inspired by Steam, has successfully demonstrated the effective integration of dynamic front-end interactivity with strong back-end management, resulting in a highly functional and user-centric platform. By leveraging a sophisticated tech stack that combines React for the front end, Flask for the back end, and MySQL for database management, the platform has laid a solid foundation for both user engagement and efficient administrative operations.

Throughout the project, notable accomplishments include the implementation of secure user authentication, dynamic search capabilities, and effective game and user management systems. These features not only enhance the overall user experience but also provide administrators with robust tools for precise data control and manipulation. Additionally, the deployment of advanced security protocols and error handling mechanisms has ensured the platform's reliability and security across different operational scenarios.

Looking ahead, there is potential for further enhancements such as introducing more detailed search filters, including options for sorting by user ratings or price, and improvements to the user interface to enhance usability. Furthermore, scalability will be a primary focus to support a growing user base and increasing data loads, ensuring the platform can scale effectively without sacrificing performance.

In summary, this gaming platform serves as a strong example of how modern web technologies can be utilized to create a comprehensive and secure digital environment that meets the contemporary needs of the gaming community and adheres to industry standards. As the platform continues to develop, it is poised to set new standards in the digital entertainment landscape.

Future scope:

1. Enhanced Search Features:

Personalized Recommendations: Employ machine learning algorithms to tailor game suggestions to individual user behaviors and preferences.

Semantic Search Enhancements: Utilize natural language processing (NLP) to more accurately interpret user queries, enhancing the search experience with more natural and intuitive interactions.

2. User Interface and Experience Improvements:

Responsive Design: Enhance the platform's design responsiveness to ensure a smooth user experience across a variety of devices, including smartphones and tablets.

Accessibility Enhancements: Integrate features that make the platform accessible to individuals with disabilities, such as compatibility with screen readers and keyboard-only navigation.

3. Community and Social Interaction Features:

Social Connectivity: Create functionalities that enable users to connect with friends, participate in gaming groups, and share their gaming experiences.

Gameplay Streaming: Incorporate features for users to live-stream their gameplay, integrating with platforms such as Twitch or YouTube.

4. Research and Future Technologies:

Exploration of New Technologies: Research and potentially integrate cutting-edge technologies like Virtual Reality (VR) and Augmented Reality (AR) to provide immersive gaming experiences.

Sustainability Initiatives: Investigate and implement sustainable practices in platform operations, focusing on energy efficiency and responsible data center management.