
Steam Gaming Platform

Name - Puneet

Project Title: Steam Gaming Platform

GitHub Repository:

<https://github.com/puneet73/DBMS-Steam-Gaming-Platform>

I. Abstract

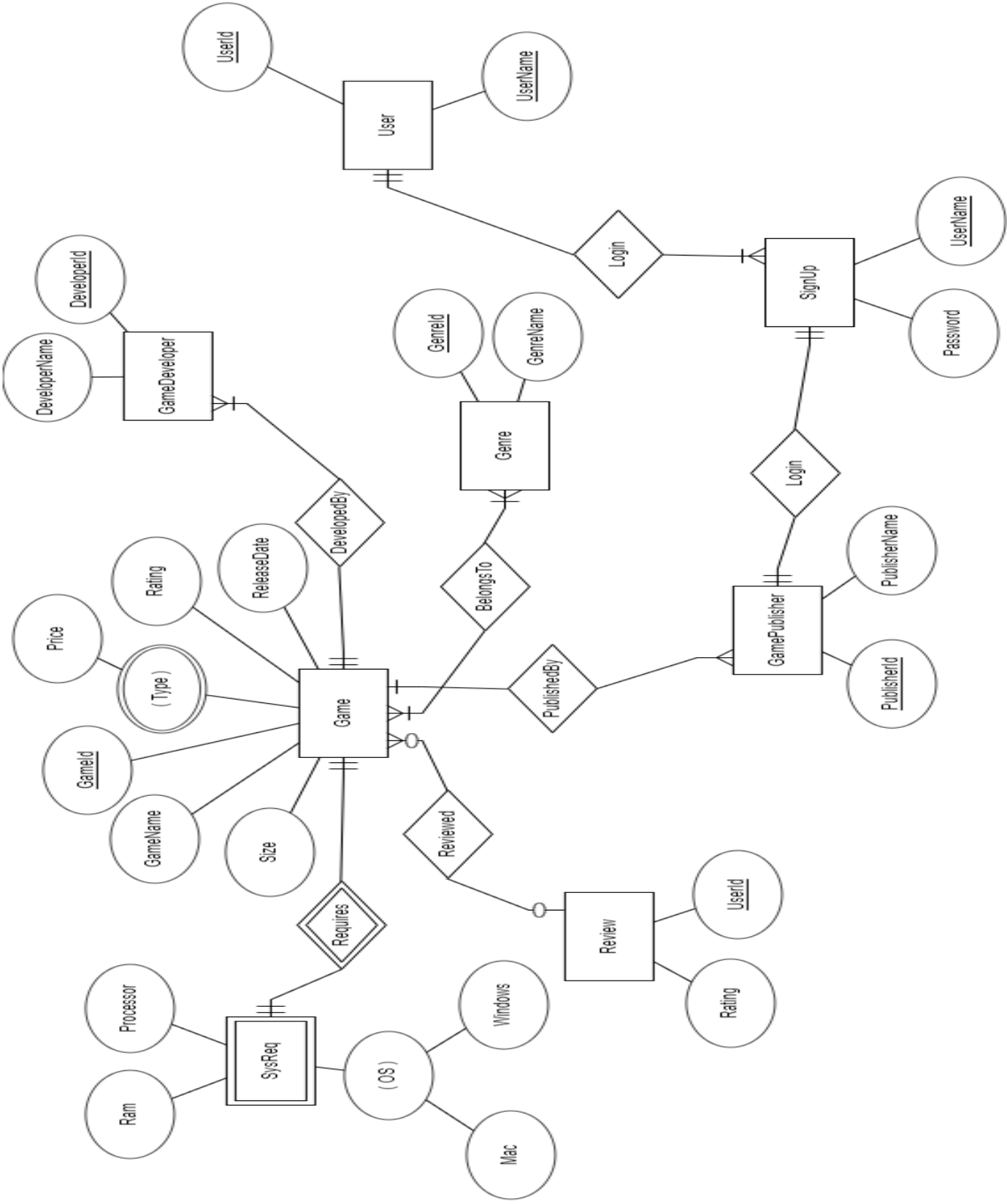
The Steam Gaming Platform Database Management System (DBMS) is designed to serve as the backend infrastructure for an online gaming platform. This project aims to provide a robust and scalable solution for managing various aspects of an online gaming ecosystem, including game information, user authentication, developer interactions, publisher management, and user-game interactions.

The system comprises several key components:

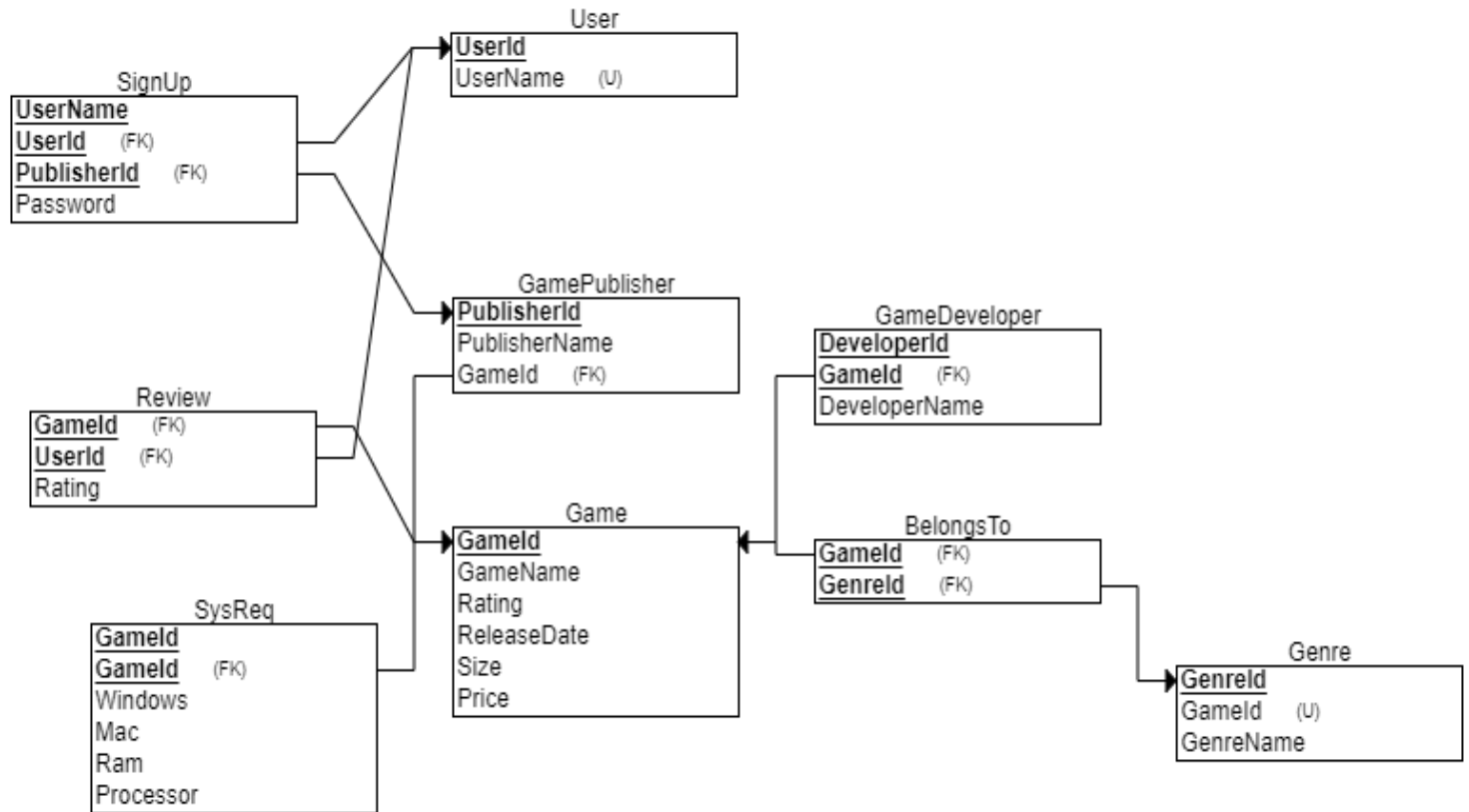
1. **Game Information Management:** Steam Gaming Platform facilitates the storage and retrieval of comprehensive game details, including names, release dates, sizes, genres, and ratings. Developers can add new games, and users can view and interact with this information.
2. **User Authentication:** User authentication is a critical component, allowing developers, publishers, and general users to securely access the platform. Different authentication mechanisms are implemented based on user roles.
3. **Developer and Publisher Interaction:** The system enables developers to associate their games with the platform and publishers to manage the games they publish. These interactions are tracked, ensuring a seamless collaboration process.
4. **User-Game Interactions:** Steam Gaming Platform tracks user interactions with games, providing a personalized experience. Users can view their owned games, and developers can access information on games they've developed.
5. **Payment Processing:** The system supports in-app purchases, allowing users to buy games securely. It includes a payment processing feature with basic validation, enhancing the user experience.
6. **System Requirements:** Steam Gaming Platform integrates system requirements for games, ensuring users have the necessary information before making a purchase.

Overall, the Steam Gaming Platform DBMS offers a comprehensive and extensible solution for managing an online gaming platform. Its modular design allows for easy scalability and adaptation to evolving requirements in the dynamic landscape of the gaming industry.

ER Diagram



Relational Schema



Relational Schema is in **3NF**

Website Front Page :

Front Page of the Website has following functionalities ::

1. **Game Info Retrieval:** The details of games, including prices, discounts, and categories, are fetched from the MySQL server's database, where all this information is stored.
2. **Sign-In Authentication:** When you sign in, the authentication process checks your credentials with the user data stored in the MySQL database, ensuring a secure and accurate sign-in.
3. **User Profiles Management:** Once signed in, your user profile information, including game history and achievements, is retrieved from the MySQL database to personalize your gaming experience.
4. **Community Chat Integration:** The chat feature pulls and displays messages from the MySQL server, allowing users to communicate and share experiences in real-time.
5. **Developer Corner Data:** Information in the developer's corner, showcasing games and facilitating discussions, is fetched from the MySQL database, where developers update and manage their projects.
6. **Publisher Dashboard Insights:** The publisher's dashboard retrieves data from the MySQL server, offering insights into game performance, user engagement, and market trends for informed decision-making.
7. **Discount Alerts:** Information about discounts and special offers is regularly updated in the MySQL server, and users receive real-time notifications based on this data.
8. **Cross-Device Compatibility:** The front page's responsive design is achieved by fetching and adapting data from the MySQL server, ensuring a consistent user experience across various devices.
9. **Game Suggestions Algorithm:** The game suggestion feature uses algorithms that analyze user preferences stored in the MySQL database, providing personalized recommendations based on individual gaming history and interests.

In essence, the MySQL server acts as a central repository for all the data needed to power these features, ensuring seamless and efficient retrieval of information to enhance the user experience on the front end.



Racing Gun Games Action Story Classics

Play Games With Your Friends



Gaming Room



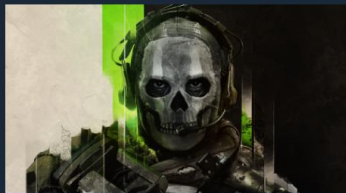
Play Now

Discount

₹999



Games on Discount



COD Modern Warfare 3

-50% Off

₹1855 ₹928



COD Modern Warfare 2

-50% Off

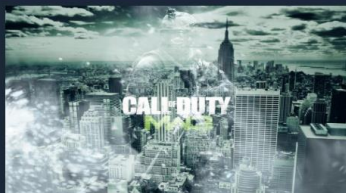
₹1907 ₹954



COD 1

-50% Off

₹1852 ₹926



COD 2

-50% Off

₹1489 ₹595



COD 3

-50% Off

₹1949 ₹975



COD 4

-50% Off

₹2689 ₹1345



Forza Horizon 5

-50% Off

₹3225 ₹1613



Forza Horizon 6

-50% Off

₹1542 ₹756



GTA 5

-50% Off

₹2999 ₹1500



GTA 6

-50% Off

₹1676 ₹838

User Page

Authentication :

User authentication involves verifying a user's identity during login. In our Game system, when a user enters their login credentials, such as a user ID and password, the system queries the Game login table in the SQL database. The entered credentials are matched with the stored values in the table, allowing access only if there's a secure match, ensuring a reliable and protected sign-in process for our users.



Viewing Game Details:

Once a user logs in, their credentials are authenticated against the user table in the SQL database. After successful authentication, the user can execute an SQL SELECT statement to view details of games available on the platform. Triggers are set up to ensure that the user has the necessary permissions to view specific game details, maintaining data security and privacy.

Purchasing Games:

When a user decides to purchase a game, the system authenticates their credentials and executes an SQL INSERT statement to create a record in the user's purchase history table. Triggers are implemented to verify that the user ID matches the purchaser ID in the game entry, ensuring that only authorized users can make purchases.

Triggers are triggered while Viewing a game::

```
CREATE TRIGGER before_game_select_user
BEFORE SELECT ON game
FOR EACH ROW
BEGIN
    IF NEW.user_id != (SELECT user_id FROM user WHERE user_id = NEW.user_id) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: Unauthorized user';
    END IF;
END;
```

Triggers are triggered while Purchasing a game::

```
CREATE TRIGGER before_game_purchase
BEFORE INSERT ON user_purchase_history
FOR EACH ROW
BEGIN
    IF NEW.user_id != (SELECT user_id FROM user WHERE user_id = NEW.user_id) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: Unauthorized purchase';
    END IF;
END;
```


User can View and Buy the Games

Welcome User!

chota bheem

[View Details](#)

car

[View Details](#)

Batman

[View Details](#)

Batman

[View Details](#)

GTA 6

[View Details](#)

COD MW3

[View Details](#)

Batman

[View Details](#)

COD MW2

[View Details](#)

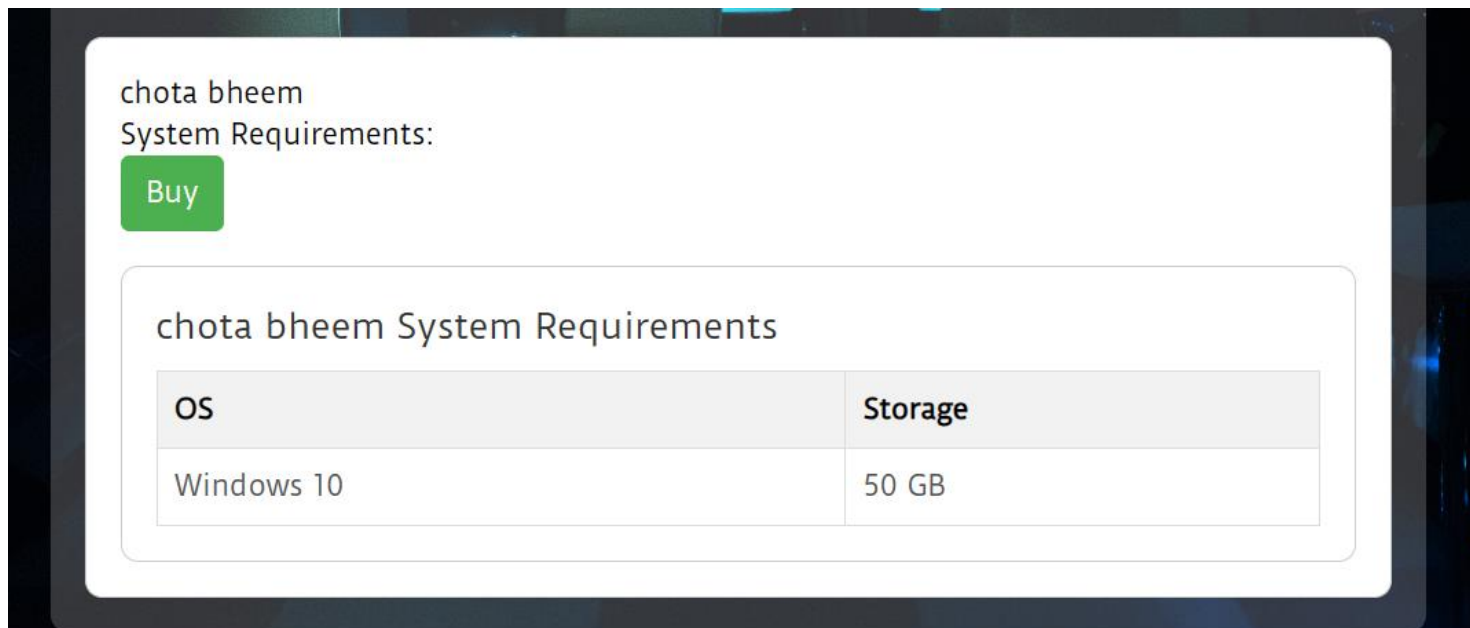
Batman2

[View Details](#)

COD 1

[View Details](#)

Details of the games user clicked will be displayed



chota bheem
System Requirements:

[Buy](#)

chota bheem System Requirements

OS	Storage
Windows 10	50 GB

Form will open for Buying the Game



Buy chota bheem

Card Number:

Expiry Date:

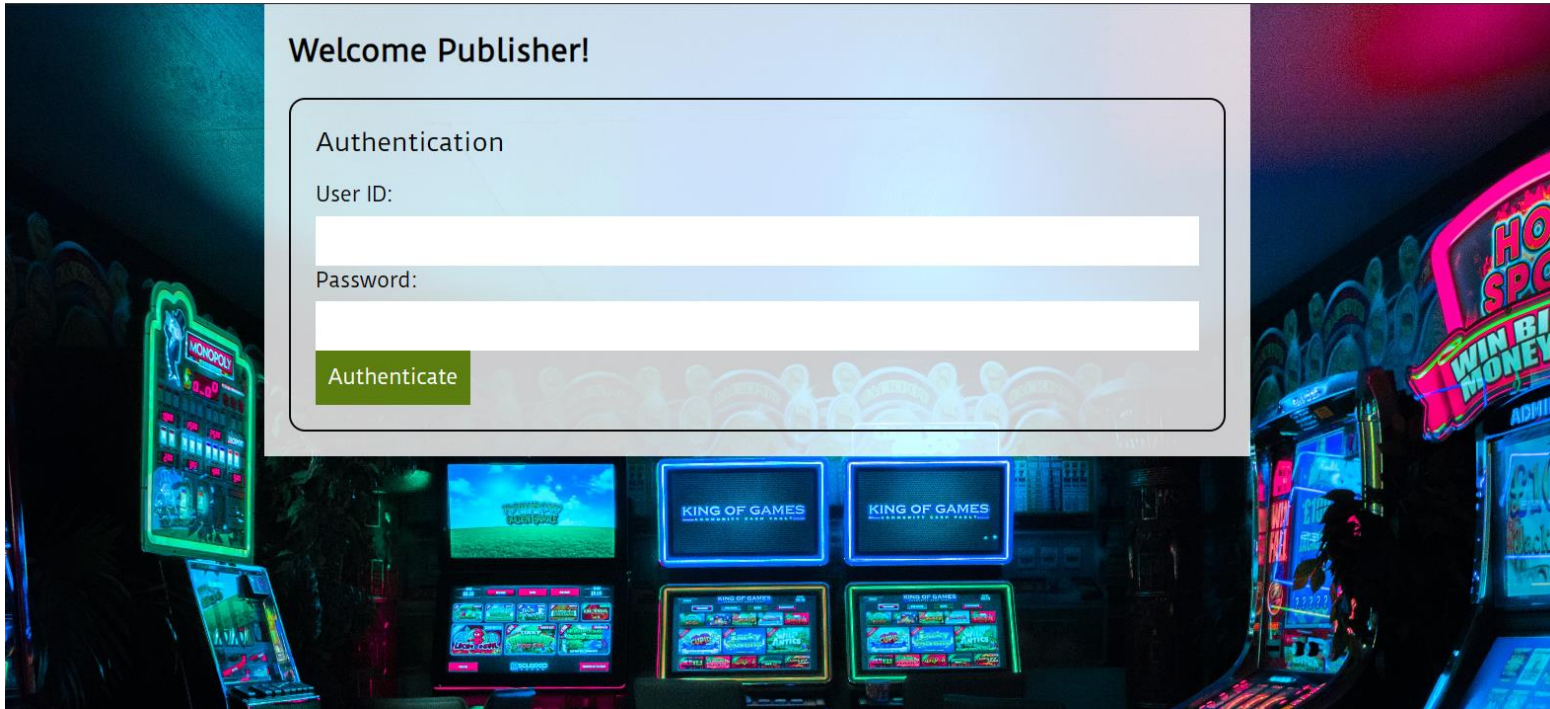
CVV:

[Submit](#)

Publisher Page

Authentication :

Publisher authentication involves validating a publisher's identity at login. When a publisher enters their credentials, like a specific publisher ID and password, the system queries the Game publisher table in the SQL database. The entered details are then cross-referenced with the stored information, allowing access only if there is a secure match. This process ensures that only authorized publishers can access their dedicated dashboard, providing a secure environment for managing and analyzing game-related data.



Adding Games:

When a publisher wishes to add a new game, the ATeamGame system checks their credentials against the publisher table in the SQL database. Once authenticated, the publisher can execute an SQL INSERT statement, adding the relevant game details to the game table. Triggers are set up to validate that the publisher ID matches the creator ID in the game entry, ensuring that only games created by the authenticated publisher are added to the database. This trigger-based approach enhances data integrity and security.

Deleting Games:

For game deletion, the publisher logs in, and after successful authentication, issues a request to delete a specific game. The system triggers an SQL DELETE statement, removing the corresponding game entry from the game table. Again, a trigger is in place to verify that the publisher ID matches the creator ID of the game, preventing unauthorized deletion of games. This two-step authentication process, along with triggers, establishes a robust mechanism for publishers to manage their games securely.

Triggers are triggered while adding game::

```
CREATE TRIGGER before_game_insert
BEFORE INSERT ON game
FOR EACH ROW
BEGIN
  IF NEW.publisher_id != (SELECT publisher_id FROM publisher WHERE publisher_id = NEW.publisher_id) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Error: Unauthorized publisher';
  END IF;
END;
```

Triggers are triggered while deleting game::

```
CREATE TRIGGER before_game_delete
BEFORE DELETE ON game
FOR EACH ROW
BEGIN
  IF OLD.publisher_id != (SELECT publisher_id FROM publisher WHERE publisher_id = OLD.publisher_id) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Error: Unauthorized deletion';
  END IF;
END;
```

Triggers are triggered while adding game to develop table based on publisher commands::

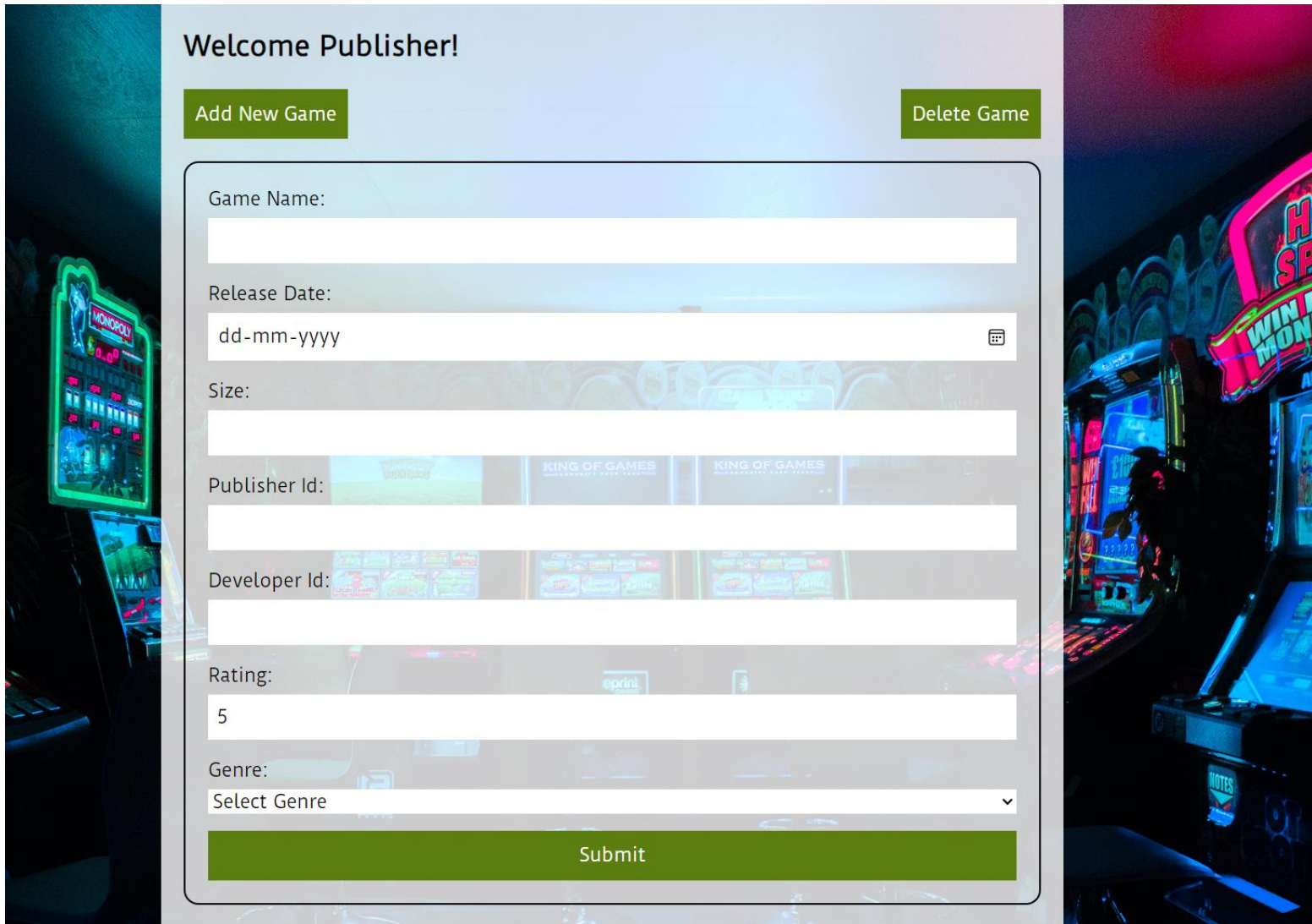
```
// Associate a game with a publisher and simulate a trigger to associate with a developer
app.post('/api/publishergame', (req, res) => {
  console.log('Received request to /api/publishergame');
  const { PublisherId, GameId } = req.body;

  db.beginTransaction((err) => {
    if (err) {
      console.error('Error starting transaction:', err);
      res.status(500).json({ error: 'Internal Server Error' });
      return;
    }

    // Insert into publishergame table
    db.query('INSERT INTO publishergame (PublisherId, GameId) VALUES (?, ?)', [PublisherId, GameId],
    (err, results) => {
      if (err) {
        console.error('Error updating publishergame table:', err);
        db.rollback(() => {
          res.status(500).json({ error: 'Internal Server Error' });
        });
      } else {
```

```
db.query('SELECT DeveloperId FROM Game WHERE gameId = ?', [GameId], (err, developerResults) => {
    if (err) {
        console.error('Error fetching DeveloperId:', err);
        db.rollback(() => {
            res.status(500).json({ error: 'Internal Server Error' });
        });
    } else {
        const DeveloperId = developerResults[0].DeveloperId;
        db.query('INSERT INTO developergame (DeveloperId, gameId) VALUES (?, ?)', [DeveloperId, GameId], (err, results) => {
            if (err) {
                console.error('Error updating developergame table:', err);
                db.rollback(() => {
                    res.status(500).json({ error: 'Internal Server Error' });
                });
            } else {
                // Commit the transaction
                db.commit((err) => {
                    if (err) {
                        console.error('Error committing transaction:', err);
                        db.rollback(() => {
                            res.status(500).json({ error: 'Internal Server Error' });
                        });
                    } else {
                        res.json({ success: true });
                    }
                });
            }
        });
    }
});
```



Publisher can Add new games after successfully filling the Game Details



Welcome Publisher!

[Add New Game](#) [Delete Game](#)

Game Name:


Release Date:
 

Size:

Publisher Id:

Developer Id:

Rating:

Genre:
 

[Submit](#)

Publisher can Delete games if the Gameld matches with the Gameld of a game under his ownership

Welcome Publisher!

Add New Game

Delete Game

Delete Game

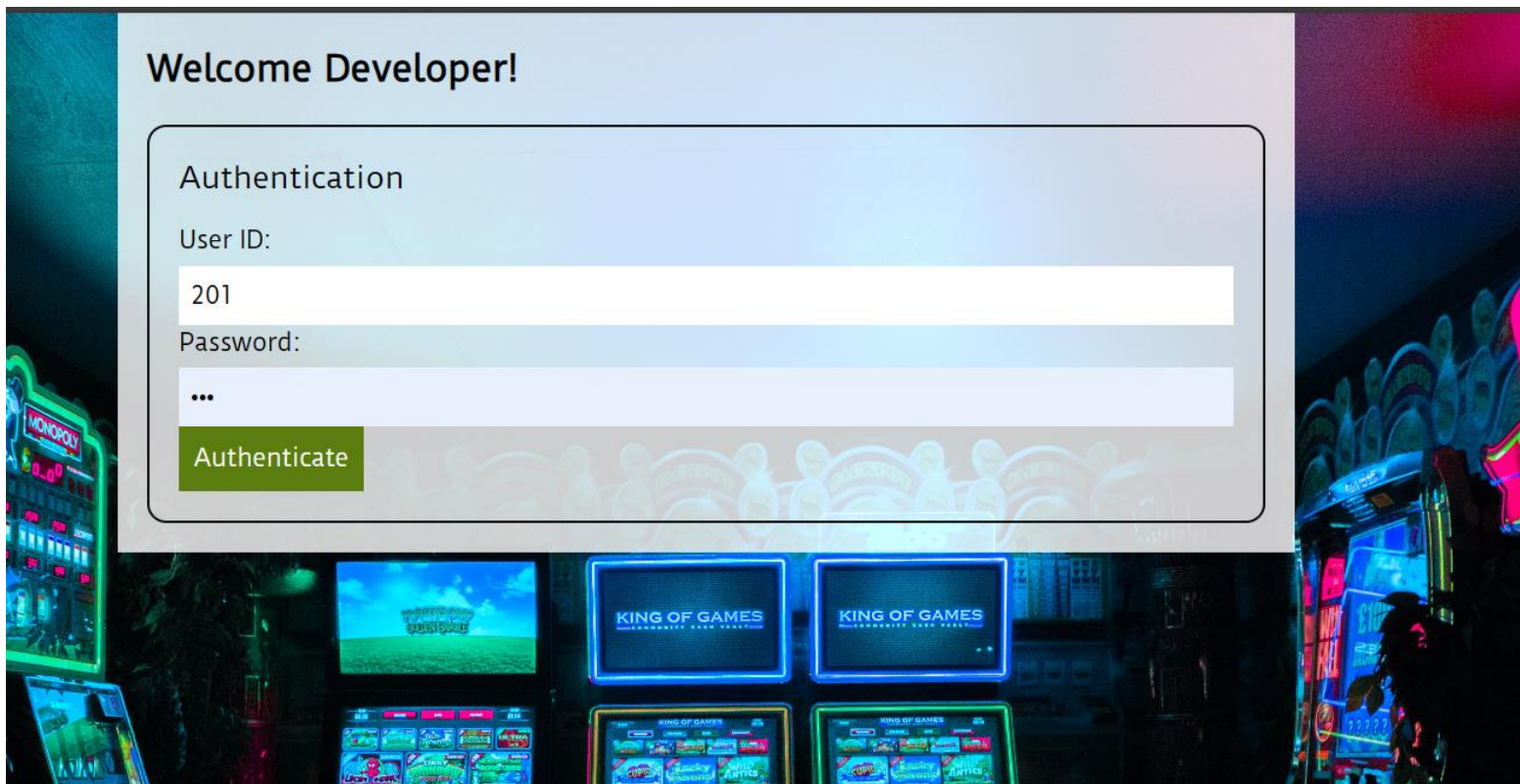
Game ID:

Delete

Developer Page

Authentication :

Similarly, developer authentication follows a stringent process. When a developer attempts to log in, their provided credentials, including a unique developer ID and password, are compared against the records stored in the Game developer table in the SQL database. Successful authentication is granted only when the entered details align securely with the stored information. This approach safeguards the developer's exclusive space on the platform, ensuring that only authenticated individuals have access to project management tools and the showcase area for their games.



Viewing Game Details:

After a developer successfully logs in, the ATeamGame system authenticates their credentials against the developer table in the SQL database. Once authenticated, the developer can execute an SQL SELECT statement to view details of games associated with their developer ID. Triggers are set up to ensure that only games created by the authenticated developer can be accessed, maintaining data security.

Updating Game Details:

When a developer wishes to update game details, they log in, and upon successful authentication, execute an SQL UPDATE statement to modify specific game information. Triggers are implemented to verify that the developer ID in the game entry matches the authenticated developer, preventing unauthorized updates. This trigger-based approach enhances data integrity and security, ensuring that only the developer who created the game can make changes to its details.

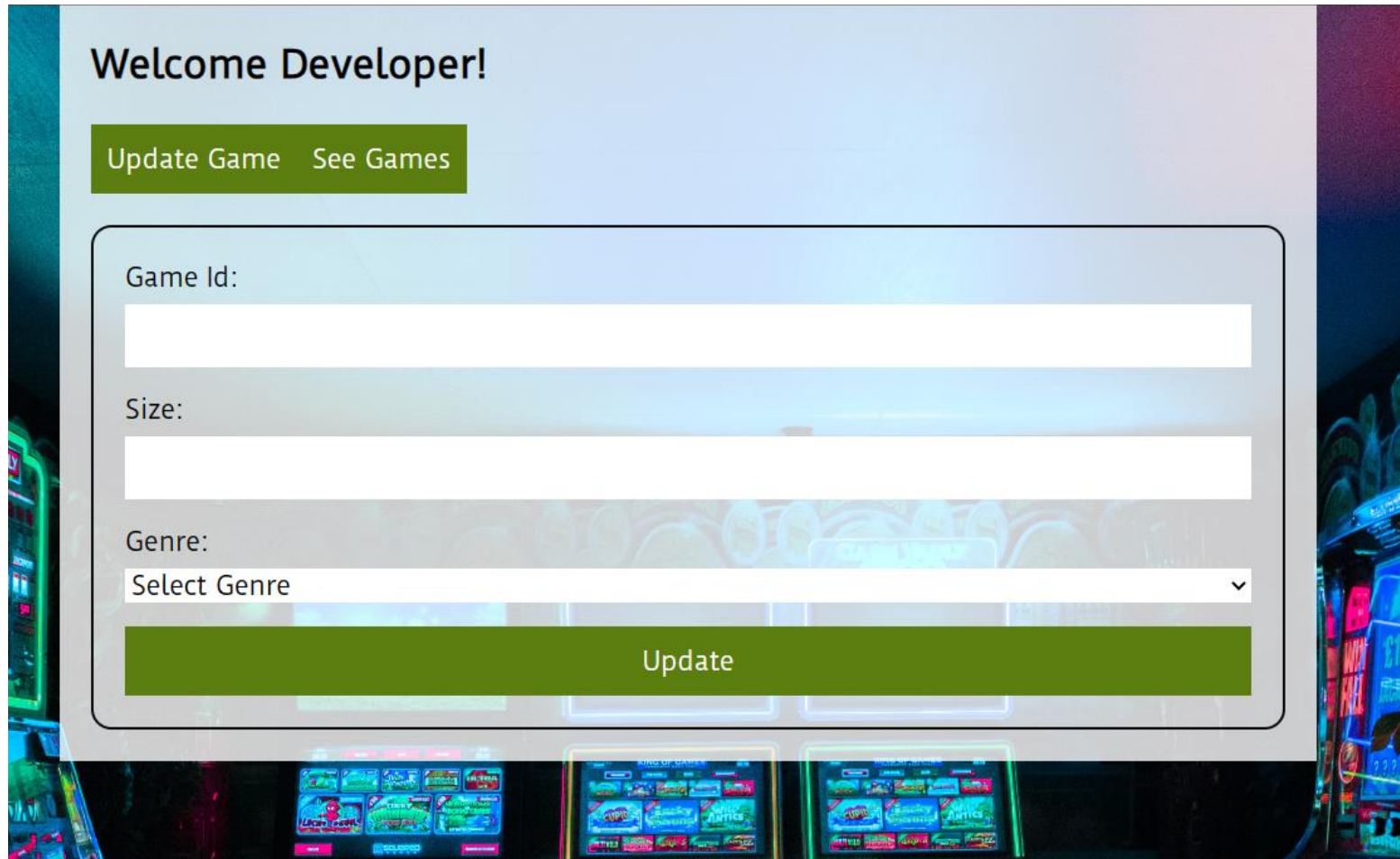
Triggers are triggered while Updating a game::

```
CREATE TRIGGER before_game_update
BEFORE UPDATE ON game
FOR EACH ROW
BEGIN
    IF OLD.developer_id != (SELECT developer_id FROM developer WHERE developer_id = OLD.developer_id) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: Unauthorized update';
    END IF;
END;
```

Triggers are triggered while Viewing the games :owned by the develop:

```
CREATE TRIGGER before_game_select
BEFORE SELECT ON game
FOR EACH ROW
BEGIN
    IF NEW.developer_id != (SELECT developer_id FROM developer WHERE developer_id = NEW.developer_id) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: Unauthorized developer';
    END IF;
END;
```

Developers can Update the games details if the Gameld matches with the Gameld of a game under his ownership



The image shows a user interface for a game developer. At the top, it says "Welcome Developer!". Below this, there are two buttons: "Update Game" and "See Games". The "Update Game" button is highlighted in green. Below the buttons, there is a form with three input fields: "Game Id:", "Size:", and "Genre:". The "Game Id:" and "Size:" fields are empty text boxes. The "Genre:" field is a dropdown menu with "Select Genre" as the current selection. Below the form, there is a green "Update" button. The background of the interface is a blurred image of a game arcade.

Welcome Developer!

Update Game See Games

Game Id:

Size:

Genre:

Select Genre

Update