



Inspiring Excellence

CSE370 : Database Systems
Project Report
Project Title : Game Store

Group No : 02, CSE370 Lab Section : 20, Summer 2025		
ID	Name	Contribution
23201415	Arik Anjum Alif	Post Thread, Comment in Thread, Buy and Sell in Marketplace, Thread management, Marketplace post approval
23201432	Musfikur Rahman Mahin	Admin games and user management, Transaction, User game management

Table of Contents

Section No	Content	Page No
1	Introduction	03
2	Project Features	04
3	ER/EER Diagram	05
4	Schema Diagram	06
5	Normalization	07
6	Frontend Development	07
7	Backend Development	16
8	Source Code Repository	28
9	Conclusion	28
10	References	29

Introduction

This project is a web-based marketplace platform designed for buying and selling used video games, developed using PHP and MySQL. The primary aim of the system is to create a convenient, secure, and interactive environment where users can trade games they no longer need, while also gaining access to affordable titles from other sellers. With the growth of online marketplaces, there is an increasing demand for platforms that not only allow smooth transactions but also ensure administrative oversight and community engagement. This project emphasizes user-friendly interaction, accessibility, and effective monitoring, ensuring that both buyers and sellers enjoy a seamless experience.

The system allows users to register and log in securely, after which they gain access to the marketplace's core features. Registered users can enlist their own games for sale, browse available listings, and purchase games using a store coin system. These coins function as the internal currency of the platform and can be managed through a dedicated transaction module. Users also have the ability to maintain and monitor their personal listings and purchase history, which helps them keep track of their activity and financial balance within the marketplace. To guarantee quality and prevent misuse, every new game listing must first receive administrative approval before being displayed to other users. This process ensures that inappropriate, duplicate, or misleading entries are filtered out, thereby preserving marketplace integrity.

The platform integrates community-driven features through a thread and comment system. This enables users to interact with one another, share experiences, and engage in discussions related to games, strategies, or trading. Such a feature strengthens the sense of community and extends the platform beyond a simple buy-and-sell model into a more interactive gaming hub.

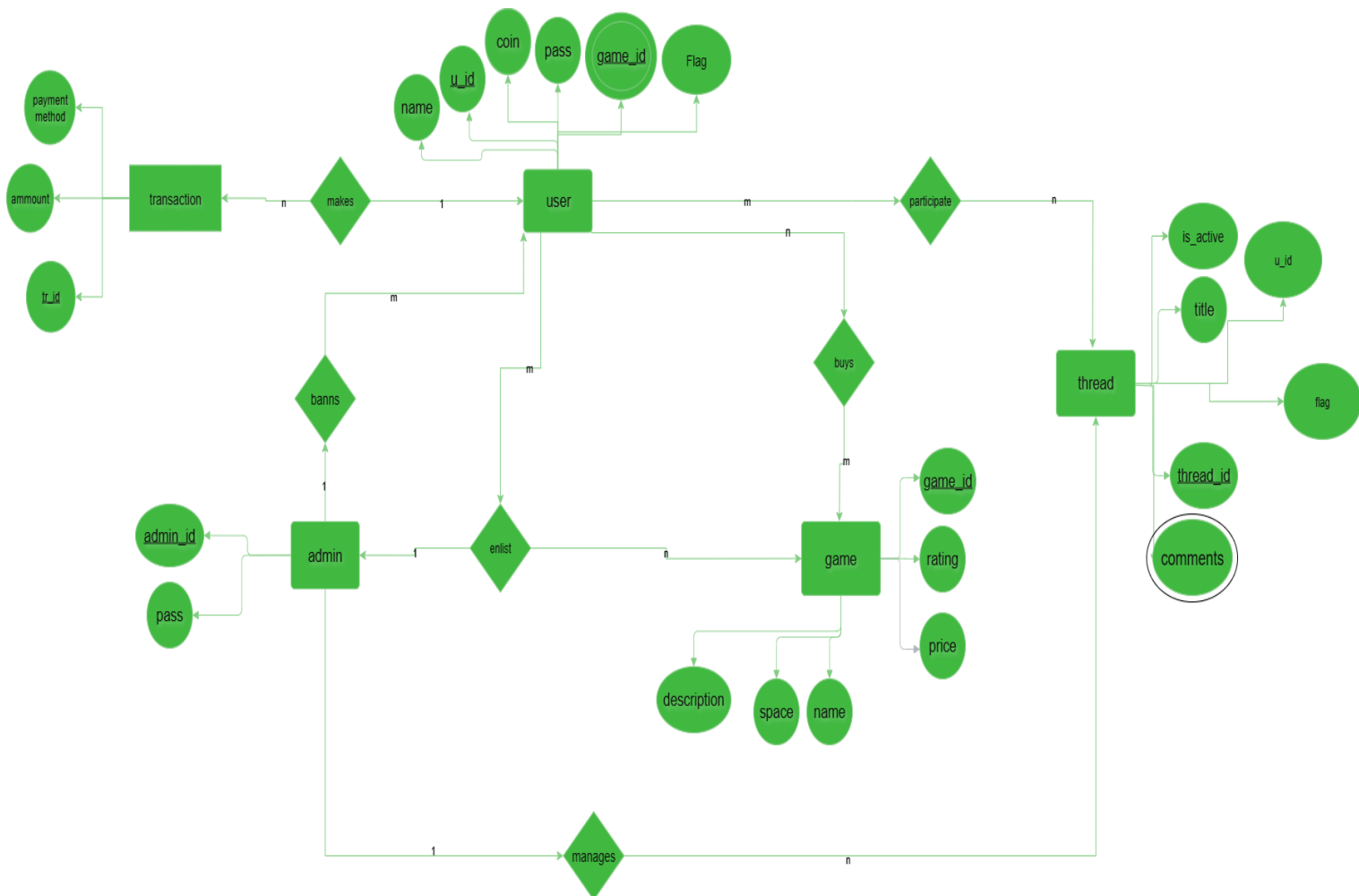
Administrators are provided with a dedicated dashboard through which they can efficiently manage the overall operation of the platform. Their responsibilities include approving or rejecting game listings, monitoring transactions, managing user accounts, moderating community threads, and ensuring fairness in all marketplace activities. They also hold the authority to restrict or ban accounts if necessary, reinforcing the importance of responsible user behavior. This role-based access structure distinguishes between normal users and administrators, protecting sensitive functionalities from unauthorized access.

The system also prioritizes security and reliability in its design. Passwords are stored using hashing functions, SQL injection vulnerabilities are mitigated through the use of prepared statements, and input validation is enforced to prevent malicious activity. In addition, the responsive design ensures that the platform is accessible across a wide variety of devices, including desktops, laptops, and mobile phones, without compromising usability.

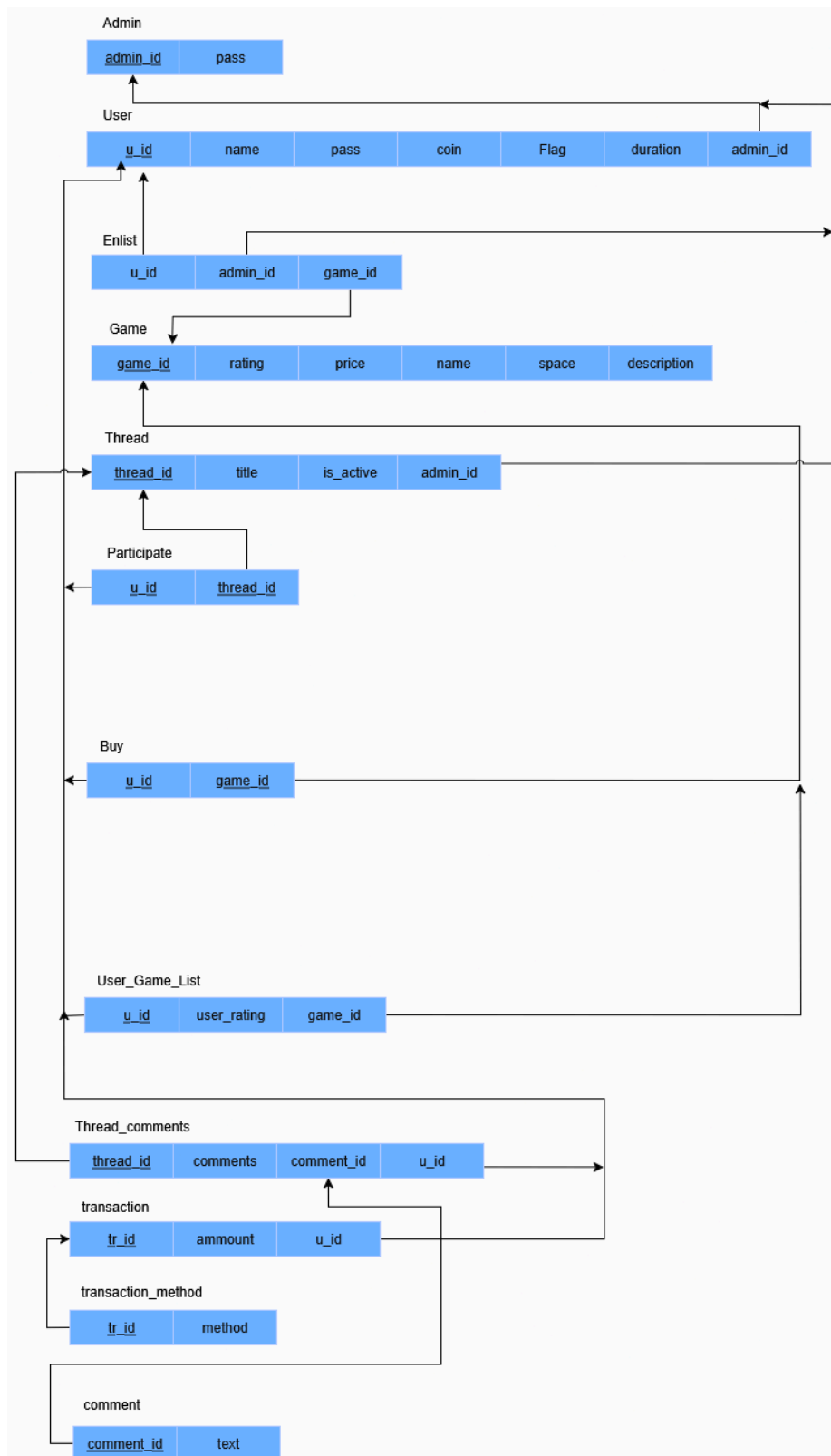
Project Features

ID, Name	Features [3 per member]	
23201432, Musfikur Rahman Mahin	Ft 1	Admin games and user management
	Ft 2	User game management
	Ft 3	Add coins
23201415, Arik Anjum Alif	Ft 1	Admin and user thread management
	Ft 2	User marketplace buy and sell
	Ft 3	Admin marketplace management

ER/EER Diagram



Schema



Normalization

- a) Our converted Schema is already in 1NF as there are no multivalued/composite attributes or nested relations.
- b) Our converted Schema is already in 2NF as there are no partial functional dependencies in our relational schema.
- c) Our converted Schema is already in 3NF as there are no transitive functional dependencies in our relational schema.

Frontend Development

Contribution of ID : 23201432, Name : Musfikur Rahman Mahin

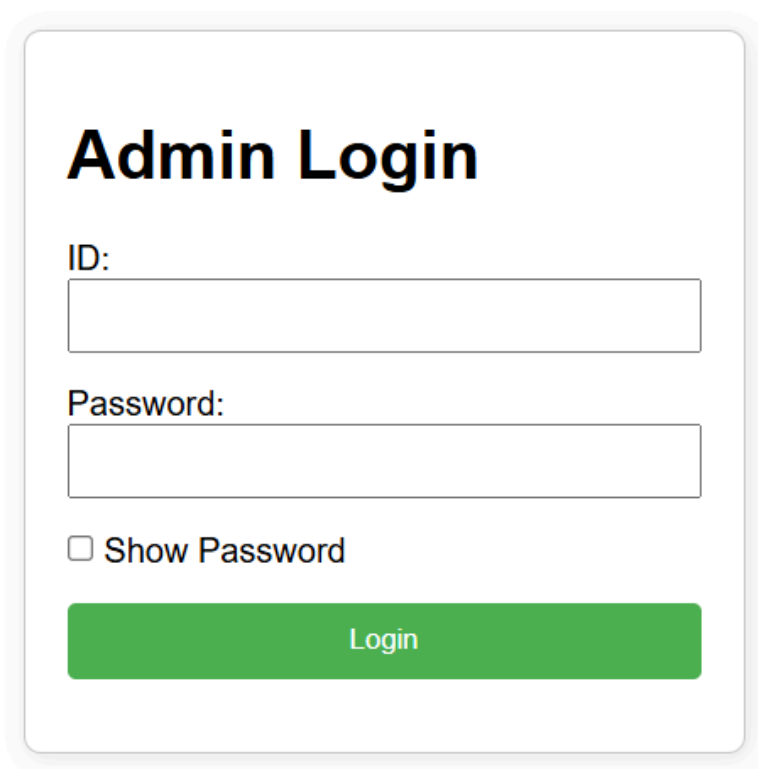
The frontend of this project is designed to provide usability, and efficient navigation for both administrators and users. Feedback messages are incorporated throughout the system to confirm user actions and provide error handling, ensuring smooth interaction.

Admin Dashboard: Acts as the central hub for administrators, providing quick access to user management, game management, thread moderation, and pending approvals. The layout is structured to prioritize efficiency and ease of navigation.

Admin Dashboard

[Manage Users](#) | [Manage Games](#) | [Manage Threads](#) | [Manage Pending](#) | [Logout](#)

Admin Login Page: Provides a secure login interface for administrators, incorporating validation and error feedback to ensure safe authentication before granting access to administrative features.



The image shows a mockup of an 'Admin Login' form. It is contained within a light gray rounded rectangle with a subtle drop shadow. The form itself has a white background and rounded corners. At the top, the title 'Admin Login' is displayed in a large, bold, black sans-serif font. Below the title, there are two input fields. The first is labeled 'ID:' in a bold black font, followed by a white rectangular input box with a thin gray border. The second is labeled 'Password:' in a bold black font, followed by a similar white rectangular input box. Below the password field, there is a checkbox with an unchecked square icon, followed by the text 'Show Password' in a standard black font. At the bottom of the form, there is a solid green rectangular button with the word 'Login' centered in white text.

Game Management Page: Enables administrators to view, add, and delete games through an organized table layout and input forms.

Manage Games

Name	Description	Rating	Price	Space	Action
Balor Rant #1	5v5	5.00	2.00	5.00	<button>Delete</button>

Add Game

Name:

Description:

Price:

Space:

Add

[Back](#) ?>

Coin Management Page: Allows users to add coins to their accounts. A simple form and transaction feedback system ensure clarity in the process of updating balances.

Add Coins

Amount:

Method:

Add

[Back](#)

User Dashboard: Displays key user information, including profile details, coin balance, and quick navigation links to core modules. The design focuses on accessibility, allowing users to efficiently manage their activities.

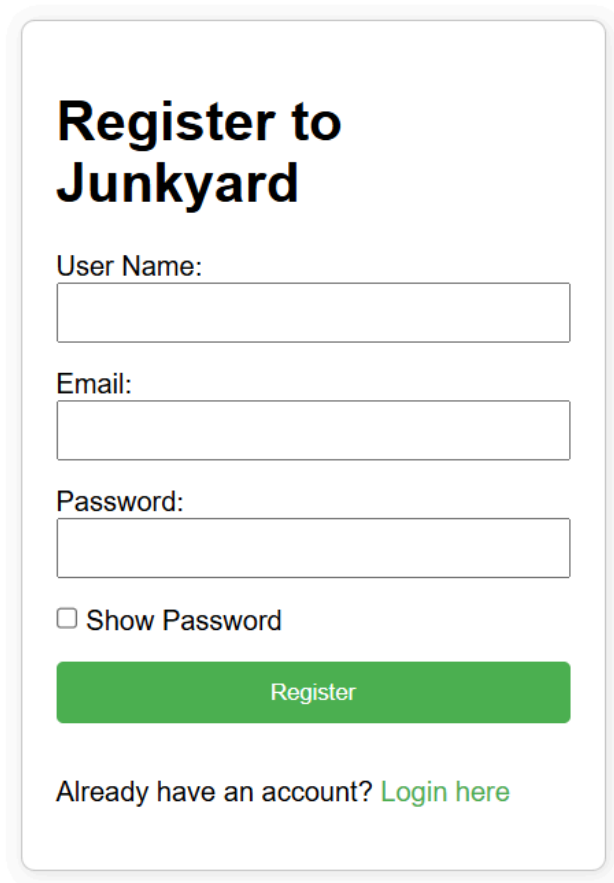
Dashboard

User name: musfi

coins: 52

[View Games](#) | [My Games](#) | [View Threads](#) | [Create Thread](#) | [Add Coins](#) | [Marketplace](#) | [Logout](#)

Registration Page: Provides a structured registration form with validation rules, password visibility options, and feedback messages. This ensures a smooth onboarding process for new users.



The registration form is titled "Register to Junkyard". It contains three input fields for "User Name:", "Email:", and "Password:". Below the password field is a checkbox labeled "Show Password". A green "Register" button is positioned below the checkbox. At the bottom, there is a link "Already have an account? Login here" in green text.

User Game List Page: Shows a personalized list of games owned by the user. It includes options to rate or remove games, with confirmation prompts to prevent accidental actions.

My Games

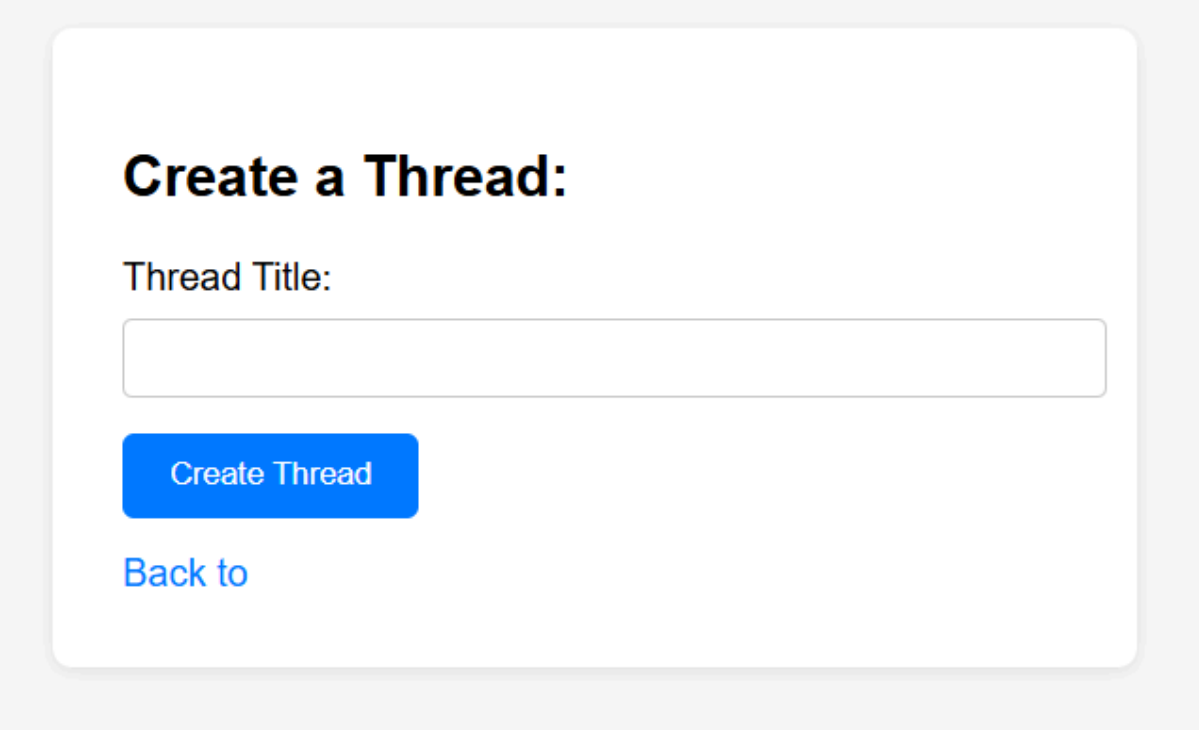
Name	Description	Rating	Action
Balor Rant	#1 5v5	<button>Rate</button>	<button>Delete</button>

[Back](#)

Contribution of ID : 23201415, Name : Arik Anjum Alif

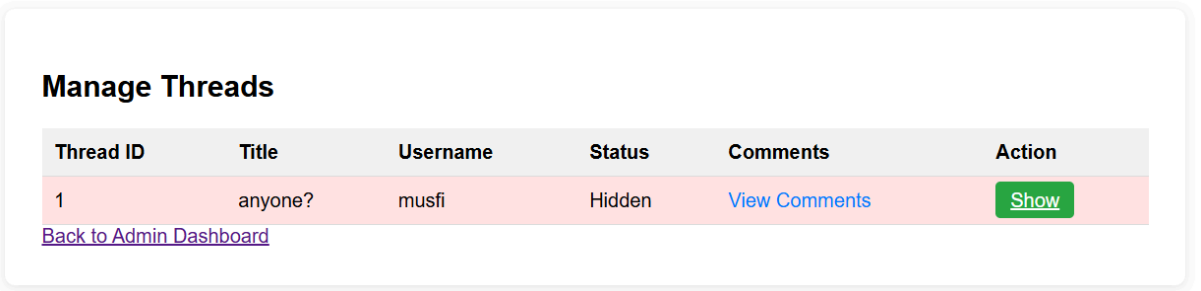
The frontend of this project is designed to provide a clean, responsive, and user-friendly interface that ensures smooth navigation and clarity for both users and administrators. The design prioritizes accessibility.

Create Thread Page: Provides users with a simple and accessible form to create new discussion threads. The layout emphasizes clarity and displays feedback messages to confirm actions, making it easy for users to contribute to the community.



The image shows a web form titled "Create a Thread:". Below the title is a label "Thread Title:" followed by a text input field. Under the input field is a blue button labeled "Create Thread". At the bottom of the form is a blue link labeled "Back to".

Manage Thread Page: Equips administrators with a straightforward interface to view all threads. It includes action buttons to hide or display threads, along with status indicators, enabling efficient moderation and control.



The image shows a table titled "Manage Threads". The table has six columns: Thread ID, Title, Username, Status, Comments, and Action. There is one data row with the following values: 1, anyone?, musfi, Hidden, View Comments, and a green button labeled Show. Below the table is a blue link labeled "Back to Admin Dashboard".

Thread ID	Title	Username	Status	Comments	Action
1	anyone?	musfi	Hidden	View Comments	Show

[Back to Admin Dashboard](#)

Comment Page: Displays user comments under discussion threads, showing contributor details and timestamps for context. It also provides a comment submission form, encouraging active participation and interaction among users.

Comments for: anyone?

Add a comment:

Post Comment

musfi | 2025-09-09 00:09:29

hi

[Back to Threads](#)

Marketplace and Purchase Pages: These pages form the central interface for browsing, buying, and managing games. Users can view available listings in a structured format with clear action buttons and navigation links, while purchase history is presented in an organized and accessible manner.

Marketplace - Used Games for Sale

Available Coins: 52

[Enlist Your Game](#)[My Purchases](#)[Dashboard](#)

No games available for sale.

Enlist Your Game for Sale

Game Name:

Price:

Game User ID:

Game Password:

Description (Unlocked items, levels, etc.):

Enlist Game

[Back to Marketplace](#)
[Dashboard](#)

Pending Listing Page: Provides administrators with a structured table layout to review, approve, or reject pending game enlistments. Action buttons and status updates ensure smooth and efficient moderation of the marketplace.

Pending Game Enlistments

Game Name	Seller	Price	Description	Enlisted At	Action
sadf	musfi	12	fd s	2025-09-09 23:03:43	<div>Approve</div> <div>Disapprove</div>

Back to Dashboard

Backend Development

Briefly discuss about Backend Development and add relevant Screenshots (if required) by mentioning Individual Contributions

Contribution of ID : 23201432, Name : Musfikur Rahman Mahin

1. Login & Registration System

- Implemented separate login and registration for users and admins.
- Passwords are hashed before storing for better security.
- User details stored in **user** table, and admin details stored in **admin** table.


```

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $admin_id = $conn->real_escape_string($_POST['admin_id']);
    $password = password_hash($_POST['password'], PASSWORD_DEFAULT);

    // check if admin id exists
    $sql_query = "SELECT * FROM admin WHERE admin_id='$admin_id'";
    $result = $conn->query($sql_query);

    if ($check_result->num_rows > 0) {
        $message = "Admin ID already exists!";
    } else {
        $sql_query = "INSERT INTO admin (admin_id, pass) VALUES ('$admin_id', '$password')";
        if ($conn->query($sql_query) === TRUE) {
            $message = "New admin account created successfully!";
        } else {
            $message = "Error: " . $conn->error;
        }
    }
}
}

```

```

if ($_SERVER["REQUEST_METHOD"] == "POST"){

    $admin_id = $conn->real_escape_string($_POST['admin_id']);
    $pass = $_POST['password'];

    // fetching the password with matching email
    $sql_check = "SELECT pass FROM admin WHERE admin_id = '$admin_id'";
    // sql query
    $result = $conn->query($sql_check);

    if ($result->num_rows > 0) {
        // turning the result into a row
        $row = $result->fetch_assoc();

        // verify entered password
        if (password_verify($pass, $row['pass'])) {

            $_SESSION['admin_id'] = $admin_id; // Save admin login
            $_SESSION['type'] = 'admin';
            // redirect to a dashboard
            header("Location: admin_dashboard.php");
            exit();
        } else {
            echo "Invalid password.";
        }
    } else {
        echo "No admin found with that id.";
    }
}

```

```

if ($_SERVER["REQUEST_METHOD"] == "POST")
{

    $email = $conn->real_escape_string($_POST['email']);
    $password = $_POST['password'];

    // fetching the password with matching email
    $sql_check = "SELECT pass, u_id, flag FROM user WHERE email = '$email'";
    // sql query
    $result = $conn->query($sql_check);

    if ($result->num_rows > 0) {
        // turning the result into a row
        $row = $result->fetch_assoc();

        // verify
        if ($row['flag'] == 0){
            if (password_verify($password, $row['pass'])) {

                $_SESSION['u_id'] = $row['u_id']; // Save user login
                $_SESSION['type'] = 'user';
                // redirect to a dashboard
                header("Location: dashboard.php");
                exit();
            } else {
                echo "Invalid password.";
            }
        } else {
            echo "You are banned currently! Please try later.";
        }
    } else {
        echo "No user found with that email.";
    }
}
?>

```

```

if ($_SERVER["REQUEST_METHOD"] == "POST")
{
    $user_name = $conn->real_escape_string($_POST['user_name']);
    $email = $conn->real_escape_string($_POST['email']);
    $password = password_hash($_POST['password'], PASSWORD_DEFAULT); // hasing the password with default algorithm

    //checking if the email exists
    $sql_check = "SELECT * FROM user WHERE email='$email'";
    // sql query
    $result = $conn->query($sql_check);

    if ($result->num_rows > 0)
    {
        header("Location: index.php");
    }
    else
    {
        // insert new user
        $sql_insert = "INSERT INTO user (name, email, pass) VALUES ('$user_name', '$email', '$password')";
        // sql query
        if ($conn->query($sql_insert) == TRUE) {
            // a message to show in login page
            $_SESSION['success_message'] = "Registration successful! Please login.";
            header("Location: index.php");
            exit();
        } else {
            echo "Error: " . $sql_insert . "<br>" . $conn->error;
        }
    }

    // Close the connection
    $conn->close();
}

```

2. User Game Management

- **View Games:** Games are fetched from **game** table and displayed to users.

```
// List all games
$sql = "SELECT * FROM Game";
// sql query
$result = $conn->query($sql);
?>

if ($result->num_rows > 0) {
    while($row = $result->fetch_assoc()) {
        echo "<tr>";
        echo "<td>" . $row['name'] . "</td>";
        echo "<td>" . $row['description'] . "</td>";
        echo "<td>" . $row['rating'] . "</td>";
        echo "<td>" . $row['price'] . "</td>";
        // Check if already owned (Read)
        $game_id = $row['game_id'];
        $buy_sql = "SELECT * FROM user_game_list WHERE u_id=$u_id AND game_id=$game_id";
        // sql query
        $buy_result = $conn->query($buy_sql);
    }
}
```

- **Buy Games:**
 - Users spend coins to buy games.
 - Purchase details stored in **buy** table and **user_game_list** table.

```

$sql_query = "SELECT coin FROM user WHERE u_id=$u_id";
// sql query
$result = $conn->query($sql_query);
$row = $result->fetch_assoc();
$coin = $row['coin'];

if ($coin >= $price) {
    // Deduct coins
    $new_coin = $coin - $price;
    $sql_query = "UPDATE user SET coin=$new_coin WHERE u_id=$u_id";
    // sql query
    $conn->query($sql_query);

    // Add to Buy
    $sql_query = "INSERT INTO buy (u_id, game_id) VALUES ($u_id, $game_id)";
    // sql query
    $conn->query($sql_query);

    // Add to user_Game_list
    $sql_query = "INSERT INTO user_Game_list (u_id, game_id) VALUES ($u_id, $game_id)";
    // sql query
    $conn->query($sql_query);
}

```

- **User Games:** Displays all games owned by a user from **user_game_list**.

```

// Get owned games from user_game_list and game table
$sql_query = "SELECT g.name, g.game_id, g.description, g.rating, l.user_rating FROM user_game_list l Join game g on l.game_id = g.game_id WHERE l.u_id = $u_id";
$result = $conn->query($sql_query);
?>

```

```

<?php
if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        echo "<tr>";
        echo "<td>" . $row['name'] . "</td>";
        echo "<td>" . $row['description'] . "</td>";
        if ($row['user_rating'] == 0){
            echo "<td><form action='rate_user_game.php' method='post'>
                <input type='hidden' name='game_id' value= $row[game_id]>
                <input type='submit' name='action' value='Rate'></form></td>";
        } else {
            echo "<td>" . $row['rating'] . "</td>";
        }
        echo "<td><form action='delete_user_game.php' method='post'>
            <input type='hidden' name='game_id' value= $row[game_id]>
            <input type='submit' name='action' value='Delete'></form></td>";
        echo "</tr>";
    }
} else {
    echo "<tr><td colspan='4'>No games owned</td></tr>";
}

```

- **Rating System:**

- Users can rate the games they purchased.
- Ratings stored in **user_game_list** (individual user rating).
- Overall game rating updated in **game** table.

```
$sql_query = "UPDATE user_game_list SET user_rating=$star where game_id=$game_id AND u_id=$u_id";
$conn->query($sql_query);

$sql_query = "UPDATE game SET rating =
              (SELECT AVG(user_rating)
               FROM user_game_list
               WHERE game_id=$game_id AND user_rating > 0)
              WHERE game_id=$game_id";
$conn->query($sql_query);
```

- **Delete Games:** Users can delete owned games, removing the row from **user_game_list**.

```
// delete from user_game_list
$sql_query = "DELETE FROM user_game_list WHERE u_id=$u_id and game_id=$game_id";
$conn->query($sql_query);

if ($conn->query($sql_query) == TRUE) {
    header("Location: user_game_list.php");
} else {
    echo "<p class='error'>Error deleting game: " . $conn->error . "</p>";
}
```

3. Admin Management

- **User Control:**
 - Admin can ban or unban users.
 - If banned (via **flag** in **user** table), user cannot log in until unbanned.

```

$u_id = $_POST['u_id'];
$action = $_POST['action'];

if ($action == 'Ban') {
    $sql_query = "UPDATE user SET flag=TRUE WHERE u_id=$u_id";
} else {
    $sql_query = "UPDATE user SET flag=FALSE WHERE u_id=$u_id";
}

if ($conn->query($sql_query) == TRUE) {
    header("Location: manage_user.php");
} else {
    echo "<p class='error'>Error: " . $conn->error . "</p>";
}
?>

```

- **Game Control:**

- Admin can add new games to **game** table.
- Admin can delete games, which also removes them from **buy**, **user_game_list**, and **game** tables.

```

// add new game
$sql_query = "INSERT INTO Game (name, description, price, space) VALUES ('$name', '$description', $price, $space)";

if ($conn->query($sql_query) == TRUE) {
    header("Location: manage_game.php");
} else {
    echo "<p class='error'>Error: " . $conn->error . "</p>";
}

?>

```

4. Transaction System

- **Add Coins:**

- Users can add coins by providing a payment method (e.g., phone number, card details).

- Updated balance stored in **user** table.

```
// transaction table
$sql_query = "INSERT INTO Transaction (amount, u_id) VALUES ($amount, $u_id)";
if ($conn->query($sql_query) == TRUE) {
    $tr_id = $conn->insert_id; // Get new tr_id (auto incremented)

    // method table
    $sql_query = "INSERT INTO Transaction_Method (tr_id, method) VALUES ($tr_id, '$method')";
    $conn->query($sql_query);

    // Add to coins
    $sql_query = "UPDATE user SET coin = coin + $amount WHERE u_id = $u_id";
    $conn->query($sql_query);

    echo "<p class='success'>Coins added! <a href='../user/dashboard.php'>Back</a></p>";
} else {
    echo "<p class='error'>Error: " . $conn->error . "</p>";
}
```

- **Transaction History:**

- All coin transactions recorded in **transaction** table.
- Payment details stored separately in **transaction_method** table for traceability.

Contribution of ID : 23201415, Name : Arik Anjum Alif

The backend of this project is built using PHP and MySQL, handling all logic, data management, and workflow enforcement for both users and administrators. It ensures secure processing of requests, proper validation of inputs, management of transactions, and efficient communication between the frontend and the database. Security measures include password hashing, prepared statements to prevent SQL injection, and role-based access control to protect sensitive operations.

create_thread.php: Handles the creation of new discussion threads by storing thread details in the Thread table and recording user participation in the Participate table.

```

if ($title != '') {
    // Insert into Thread table
    $stmt = $conn->prepare("INSERT INTO Thread (title, u_id) VALUES (?, ?)");
    $stmt->bind_param("si", $title, $u_id);
    if ($stmt->execute()) {
        $thread_id = $stmt->insert_id;
        // Insert into Participate table
        $stmt2 = $conn->prepare("INSERT INTO Participate (u_id, thread_id) VALUES (?, ?)");
        $stmt2->bind_param("ii", $u_id, $thread_id);
        $stmt2->execute();
        $stmt2->close();
        $message = "Thread created successfully!";
    } else {
        $message = "Error creating thread.";
    }
    $stmt->close();
} else {
    $message = "Thread title cannot be empty.";
}

```

manage_threads.php: Provides administrators with moderation capabilities by updating the flag field in the Thread table to hide or show threads as needed.

```

// Handle hide/show actions
if (isset($_GET['action']) && isset($_GET['thread_id'])) {
    $thread_id = intval($_GET['thread_id']);
    if ($_GET['action'] === 'hide') {
        $stmt = $conn->prepare("UPDATE Thread SET flag = TRUE WHERE thread_id = ?");
        $stmt->bind_param("i", $thread_id);
        $stmt->execute();
        $stmt->close();
    } elseif ($_GET['action'] === 'show') {
        $stmt = $conn->prepare("UPDATE Thread SET flag = FALSE WHERE thread_id = ?");
        $stmt->bind_param("i", $thread_id);
        $stmt->execute();
        $stmt->close();
    }
}

// Fetch all threads (including hidden)
$sql = "SELECT t.thread_id, t.title, u.name AS username, t.flag
        FROM Thread t
        JOIN user u ON t.u_id = u.u_id
        ORDER BY t.thread_id DESC";
$result = $conn->query($sql);

```

comments.php: Manages user comments by inserting data into the Thread_Comments table and fetching user information from the user table for display.


```
// Handle new comment submission
$message = '';
if ($_SERVER['REQUEST_METHOD'] === 'POST' && !empty($_POST['comment'])) {
    $comment = trim($_POST['comment']);
    $u_id = isset($_SESSION['u_id']) ? $_SESSION['u_id'] : null;
    if ($comment !== '' && $u_id) {
        $stmt = $conn->prepare("INSERT INTO Thread_Comments (thread_id, comments, u_id) VALUES (?, ?, ?)");
        $stmt->bind_param("isi", $thread_id, $comment, $u_id);
        if ($stmt->execute()) {
            $message = "Comment posted!";
        } else {
            $message = "Failed to post comment.";
        }
        $stmt->close();
    }
}

// Fetch comments for this thread with username and time
$comments = [];
$sql = "SELECT tc.comments, tc.created_at, u.name AS username
FROM Thread_Comments tc
LEFT JOIN user u ON tc.u_id = u.u_id
WHERE tc.thread_id = ?
ORDER BY tc.created_at DESC";
$stmt = $conn->prepare($sql);
$stmt->bind_param("i", $thread_id);
$stmt->execute();
$result = $stmt->get_result();
while ($row = $result->fetch_assoc()) {
    $comments[] = $row;
}
```

enlisted.php: Processes game enlistment requests by inserting new game records into the Used_Game_Marketplace table, marking them for administrative approval.

```
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $game_name = trim($_POST['game_name']);
    $price = floatval($_POST['price']);
    $description = trim($_POST['description']);
    $game_user_id = trim($_POST['game_user_id']);
    $game_password = trim($_POST['game_password']);

    if ($game_name && $price > 0 && $description !== '' && $game_user_id !== '' && $game_password !== '') {
        // Insert into Used_Game_Marketplace with admin_status 'PENDING'
        $stmt = $conn->prepare("INSERT INTO Used_Game_Marketplace (u_id, game_name, price, description, game_user_id, game_password, admin_status) VALUES (?, ?, ?, ?, ?, ?, 'PENDING')");
        $stmt->bind_param("isdsss", $u_id, $game_name, $price, $description, $game_user_id, $game_password);
        $stmt->execute();
        $stmt->close();
        $message = "Game enlistment request sent to admin for approval.";
    } else {
        $message = "Please fill all fields correctly.";
    }
}
```

marketplace.php: Implements core marketplace functionality by reading and updating records in the Used_Game_Marketplace table and updating user coin balances in the user table during transactions.

```
// Handle buy request
if (isset($_POST['buy_market_id']) && isset($_SESSION['u_id'])) {
    $market_id = intval($_POST['buy_market_id']);
    $buyer_id = $_SESSION['u_id'];

    // Check if already bought
    $check_sql = "SELECT status, buyer_id FROM Used_Game_Marketplace WHERE market_id = ?";
    $check_stmt = $conn->prepare($check_sql);
    $check_stmt->bind_param("i", $market_id);
    $check_stmt->execute();
    $check_stmt->bind_result($current_status, $current_buyer);
    $check_stmt->fetch();
    $check_stmt->close();
    if ($current_status === 'SOLD' || $current_buyer) {
        $buy_message = "This game has already been sold.";
    } else {
        // Fetch user's coin balance
        $coin_sql = "SELECT coin FROM user WHERE u_id = ?";
        $coin_stmt = $conn->prepare($coin_sql);
        $coin_stmt->bind_param("i", $buyer_id);
        $coin_stmt->execute();
        $coin_stmt->bind_result($user_coin);
        $coin_stmt->fetch();
        $coin_stmt->close();

        // Fetch game price and seller id
        $price_sql = "SELECT price, u_id FROM Used_Game_Marketplace WHERE market_id = ?";
        $price_stmt = $conn->prepare($price_sql);
        $price_stmt->bind_param("i", $market_id);
        $price_stmt->execute();
        $price_stmt->bind_result($game_price, $seller_id);
        $price_stmt->fetch();
        $price_stmt->close();

        if ($user_coin >= $game_price) {
            // Deduct coins from buyer
            $update_coin_sql = "UPDATE user SET coin = coin - ? WHERE u_id = ?";
            $update_coin_stmt = $conn->prepare($update_coin_sql);
            $update_coin_stmt->bind_param("di", $game_price, $buyer_id);
            $update_coin_stmt->execute();
            $update_coin_stmt->close();

            // Add coins to seller
            $add_coin_sql = "UPDATE user SET coin = coin + ? WHERE u_id = ?";
            $add_coin_stmt = $conn->prepare($add_coin_sql);
            $add_coin_stmt->bind_param("di", $game_price, $seller_id);
            $add_coin_stmt->execute();
            $add_coin_stmt->close();
        }
    }
}
```

purchases.php: Retrieves and displays purchased games for users, fetching relevant records from the Used_Game_Marketplace table while ensuring only approved and completed transactions are shown.

```
// Fetch threads with user name, title, and time using Participate, Thread, and user tables
$sql = "SELECT t.thread_id, t.title, u.name AS username, t.created_at
FROM Participate p
JOIN Thread t ON p.thread_id = t.thread_id
JOIN user u ON p.u_id = u.u_id
WHERE t.flag = FALSE
ORDER BY t.created_at DESC"; // Show most recent threads first
$result = $conn->query($sql);
```

manage_pending.php: Allows administrators to review pending game listings by updating or deleting records in the Used_Game_Marketplace table.

```

// Approve pending game
if (isset($_POST['approve_market_id'])) {
    $market_id = intval($_POST['approve_market_id']);
    $stmt = $conn->prepare("UPDATE Used_Game_Marketplace SET admin_status = 'APPROVED' WHERE market_id = ?");
    $stmt->bind_param("i", $market_id);
    $stmt->execute();
    $stmt->close();
}

// Disapprove pending game
if (isset($_POST['disapprove_market_id'])) {
    $market_id = intval($_POST['disapprove_market_id']);
    $stmt = $conn->prepare("DELETE FROM Used_Game_Marketplace WHERE market_id = ?");
    $stmt->bind_param("i", $market_id);
    $stmt->execute();
    $stmt->close();
}

// Fetch all pending games
$sql = "SELECT m.market_id, m.game_name, m.price, m.description, m.listed_at, u.name AS seller
FROM Used_Game_Marketplace m
JOIN user u ON m.u_id = u.u_id
WHERE m.admin_status = 'PENDING' AND m.buyer_id IS NULL AND m.u_id IS NOT NULL ORDER BY m.listed_at DESC";
$result = $conn->query($sql);

```

threads.php: Displays all active discussion threads by reading thread and user data from the Thread, Participate, and user tables, providing navigation for users to view comments or create new threads.

```

// Fetch threads with user name, title, and time using Participate, Thread, and user tables
$sql = "SELECT t.thread_id, t.title, u.name AS username, t.created_at
FROM Participate p
JOIN Thread t ON p.thread_id = t.thread_id
JOIN user u ON p.u_id = u.u_id
WHERE t.flag = FALSE
ORDER BY t.created_at DESC"; // Show most recent threads first
$result = $conn->query($sql);

```

Source Code Repository

Github repository:

<https://github.com/musfikur-rahman-mahin/Project370>

Conclusion

This project provides a complete solution for the buying and selling of used video games, combining commerce, security, and community into one platform. It simplifies the trading process for users, offers oversight and management tools for administrators, and fosters interaction through community features. By integrating a store coin system, admin-controlled approvals, and robust security practices, the project ensures both fairness and sustainability. Overall, it delivers a secure, interactive, and efficient environment that encourages active participation, supports smooth transactions, and strengthens the gaming community.

References

Php documentation

<https://www.php.net/docs.php>

W3school Introduction to HTML

https://www.w3schools.com/html/html_intro.asp