CS 166 Project Report

Group Information

Nishant Tiwari - ntiwa006
Ynah Novilla - ynovi001

Implementation Description

Our implementation of the Game Rental Store covers user creation, profile management, catalog browsing, order placement, tracking, and viewing. We have also incorporated error handling to ensure database resources are properly managed, and that only users with higher level authority have access to higher-level functions such as updating a user's profile, and updating the game catalog.

**** ALL OUTPUT SCREENSHOTS PROVIDED AT THE END OF THE REPORT!****

1. **Option 1: Create User**
   We prompt the user for input, and before adding an instance in the User table, we first check if the login already exists. If the user inputs a login that exists, the program simply tells the user to try again. Otherwise, we do a simple "INSERT INTO USERS" query along with an "esql.executeUpdate()".

```
public static void CreateUser(GameRental esql){
   try {
      // prompting user for input
      System.out.print("\n \tACCOUNT SETUP\n");
      System.out.print("\tEnter login: ");
      String user_login = in.readLine();
      System.out.print("\tEnter password: ");
      String user_password = in.readLine();
      System.out.print("\tEnter phone number: ");
      String user_phonenumber = in.readLine();

      // checking if login already exists
      String login_query = String.format("SELECT * FROM Users WHERE login = '%s';", user_login);
      int userTrue = esql.executeQuery(login_query);

      if (userTrue > 0) {
        System.out.print("\nLogin already taken. Please try again.\n");
        return;
      }

      // initializing query
      String add_user_query = String.format("INSERT INTO Users(login, password, role, favGames, phoneNum, numOverDueGames)
                                             VALUES('%s', '%s', 'customer', NULL, '%s', 0)",
                                             user_login, user_password, user_phonenumber);

      // executing query
      esql.executeUpdate(add_user_query);
      System.out.print("\nUser succesfully created. Welcome, " + user_login + "! \n");
      System.out.print("\n");
   } catch(Exception e){
      System.err.println (e.getMessage ());
   }
}//end CreateUser
```

2. **Option 2: Log In**

   We prompt the user for their login and password and check if a row in the Users table exists with that login and password combination with the query “SELECT * FROM Users WHERE login = inputted_login AND password = inputted_password”. If so, we let the user log in, otherwise, they are forced to re-enter their password.

```
public static String LogIn(GameRental esql){
   try {
      // prompting user for login info
      System.out.print("\n");
      System.out.print("\tEnter login: ");
      String user_login = in.readLine();
      System.out.print("\tEnter password: ");
      String user_password = in.readLine();

      // initializing sql query
      String login_query = String.format("SELECT * FROM Users WHERE login = '%s' AND password = '%s';", user_login, user_password);

      // executing query
      int userTrue = esql.executeQuery(login_query);

      // checking if login exists
      if (userTrue > 0) {
         System.out.print("\nLogin successful. Welcome, " + user_login + "!\n");
         return user_login;
      }
      else {
         System.out.print("\nLogin unsuccesful. Please re-enter your login/password.\n");
         return null;
      }

   } catch(Exception e){
      System.err.println (e.getMessage ());
      return null;
   }
} //end logIn
```

3. **Option 1: View Profile**

   For a user to view their profile there are 3 different queries to output favGames, numOverDueGames, and phone number with a simple “SELECT <attribute> FROM Users U WHERE U.login = user_login”. All 3 queries are executed with the command “esql.executeQueryAndPrintResult(query)” so that the result is outputted to the user.

```
public static void viewProfile(GameRental esql, String user_login) {
   try{
      // skip line
      System.out.print("\n");

      // view favGames
      String view_favGames = String.format("SELECT favGames AS \"Favorite Games\" FROM Users U WHERE U.login = '%s'", user_login);
      esql.executeQueryAndPrintResult(view_favGames);
      System.out.print("\n");

      // view numOverDueGames
      String view_numOverDueGames = String.format("SELECT numOverDueGames AS \"Number of Overdue Games:\" FROM Users U WHERE U.login = '%s'", user_login);
      esql.executeQueryAndPrintResult(view_numOverDueGames);
      System.out.print("\n");

      // view phoneNum
      String view_phoneNum = String.format("SELECT phoneNum AS \"Phone Number\" FROM Users U WHERE U.login = '%s'", user_login);
      esql.executeQueryAndPrintResult(view_phoneNum);

   } catch(Exception e){
      System.err.println (e.getMessage ());
   }
  // end viewProfile
```

4. **Option 2: Update Profile**

   For each attribute that the user would like to update, the syntax for the queries is all the same. The query template is “UPDATE Users SET <attribute> = <updated_attribute> WHERE login = ‘user_login’”. To execute this query, we use “esql.executeUpdate(query) to change each row in the table that matches the login of the logged-in user.

```
public static void updateProfile(GameRental esql, String user_login) {
  try{
     boolean update = true;

     while(update == true) { // iterate while user wants to update
        System.out.print("\n What would you like to update? \n1. Favorite Games \n2. Password \n3. Phone Number \n");
        switch(readChoice()) {
           case 1: System.out.print("\nFavorite Games: ");
           String new_games = in.readLine();
           String query1 = String.format("UPDATE Users SET favGames = '%s' WHERE login = '%s'", new_games, user_login);
           esql.executeUpdate(query1); break;

           case 2: System.out.print("\nNew Password: ");
           String new_password = in.readLine();
           String query2 = String.format("UPDATE Users SET password = '%s' WHERE login = '%s'", new_password, user_login);
           esql.executeUpdate(query2); break;

           case 3: System.out.print("\nNew Phone Number: ");
           String new_phoneNum = in.readLine();
           String query3 = String.format("UPDATE Users SET phoneNum = '%s' WHERE login = '%s'", new_phoneNum, user_login);
           esql.executeUpdate(query3); break;
        }
        System.out.print("\nProfile updated succesfully. Would you like to update again? \n1. Yes \n2. No\n");
        switch(readChoice()) {
           case 1: break;
           case 2: update = false; break;
        }
     }
  } catch(Exception e){
     System.err.println (e.getMessage ());
  }
} // end updateProfile
```

5. **Option 3: View Catalog**

   View catalog uses 4 different types of helpers, filter by genre which filters the catalog.csv as per the genre entered by the user

- sort by price: filters the catalog.csv content as per the maximum price entered by the user, it has two subparts: if the user wants in ascending it will do that but if the user wants in descending it will also do that
- view all games: displays all the columns in catalog.csv file

```
public static void viewCatalog(GameRental esql, String user_login) {
  try {
     boolean keepLooking = true;
     while (keepLooking) {
        System.out.println("\nBROWSE CATALOG");
        System.out.println("------------");
        System.out.println("1. View all Games");
        System.out.println("2. Filter by Genre");
        System.out.println("3. Filter by Price");
        System.out.println("4. Sort by Price (low to high)");
        System.out.println("5. Sort by price (high to low)");
        System.out.println("6. Exit Catalog");

        switch (readChoice()){
           case 1: viewAllGames(esql); break;
           case 2: filterByGenre(esql); break;
           case 3: filterByPrice(esql); break;
           case 4: sortByPrice(esql, ascending:true); break;
           case 5: sortByPrice(esql, ascending:false); break;
           case 6: keepLooking = false; break;
           default: System.out.println("Invalid Input choice!!"); break;
        }
     }

  } catch (Exception e) {
        System.err.println(e.getMessage());
     }
} // end viewCatalog
```

6. **Option 4: Place Order**
   We insert game and rental order information using “INSERT INTO <table> VALUES(<values>)” followed by “esl.executeUpdate(query)”. To access the price of a game, we “SELECT price FROM Catalog WHERE gameID = inputted_gameID” followed by “esql.executeQueryAndReturnResult” so that we can store the price, extract its values, and convert it into a double for addition.

```
public static void placeOrder(GameRental esql, String user_login) {
  try {
    boolean getGame = true;
    double totalPrice = 0.0;
    int totalGames = 0;

    String orderID = generateUniqueRental(esql); // initialize unique orderID
    Timestamp dueDate = generateRandomDueDate(); // initialize dueDate
    Timestamp timeStamp = getCurrentTimestamp(); // initialize timeStamp
    String trackingid = generateUniquetrackingID(esql); // initialize trackingID

    // query
    String add_order_id = String.format("INSERT INTO rentalorder(rentalorderid, login, noOfGames, totalprice, orderTimestamp, dueDate) VALUES('%s', '%s', 0, 0.0, '%s',
    esql.executeUpdate(add_order_id);

    while (getGame == true) { // iterate until user no longer wants to purchase a game
      System.out.print("\n--PLACE RENTAL ORDER--\n");
      System.out.print("Enter the game ID of the game you'd like to order: ");
      String user_game = in.readLine();

      System.out.print("How many units would you like?: ");
      int unitsToOrder = Integer.parseInt(in.readLine());
      totalGames = unitsToOrder + totalGames;

      String query = String.format("SELECT price FROM Catalog WHERE gameID = '%s'", user_game);
      List<List<String>> price = esql.executeQueryAndReturnResult(query);
      String stringPrice = price.get(0).get(0);
      double intPrice = Double.parseDouble(stringPrice);
      totalPrice = (unitsToOrder * intPrice) + totalPrice;
      System.out.print("\nCurrent Price: $" + totalPrice + "\n");

      String add_into_games = String.format("INSERT INTO gamesinorder(rentalorderid, gameID, unitsOrdered) VALUES('%s', '%s', '%s')", orderID, user_game, unitsToOrder)
      esql.executeUpdate(add_into_games);

      System.out.print ("\nWould you like to order more? (Y or N): ");
      String response = in.readLine();
      if (response.equalsIgnoreCase("N")) {
        getGame = false;
      }
    }

    // add into rental order
    String add_order_query = String.format("UPDATE rentalorder SET noOfGames = '%s', totalprice = '%s' WHERE rentalorderid = '%s'", totalGames, totalPrice, orderID);
    esql.executeUpdate(add_order_query);

    // add into tracking info
    String add_tracking_query = String.format("INSERT INTO trackinginfo(trackingID, rentalorderid, status, currentLocation, courierName, lastUpdateDate, additionalComme
    esql.executeUpdate(add_tracking_query);

    System.out.print("\nThe total cost of your order is: $" + totalPrice + "\n\nYour order has been placed.\nOrder ID: " + orderID + "\nTracking ID: " + trackingid + "\

  } catch (Exception e) {
    System.err.println(e.getMessage());
  }
} // end placeOrder
```

7. **Option 5: View All Orders**

   this query uses user_login and displays all the orders that are under their login, it uses execute query and print result and uses three different SQL tables such as catalog, games in order, and rental order

```
public static void viewAllOrders(GameRental esql, String user_login) {
  try {
     String query = "SELECT r1.rentalOrderID, c.gameName, r1.orderTimestamp, r1.dueDate " +
                    "FROM RentalOrder r1 " +
                    "JOIN GamesInOrder gm ON r1.rentalOrderID = gm.rentalOrderID " +
                    "JOIN Catalog c ON gm.gameID = c.gameID " +
                    "WHERE r1.login = '" + user_login + "'" +
                    "ORDER BY r1.orderTimestamp";
     System.out.print("Your order history: ");
     esql.executeQueryAndPrintResult(query);

  } catch (Exception e) {
     System.err.println(e.getMessage());
  }
} // end viewAllOrders
```

8. **Option 6: View Recent Orders**

   it has the same functionality as above but limits the result to the most recent 5 values

```
public static void viewRecentOrders(GameRental esql, String user_login) {
  try{
     String query = "SELECT r1.rentalOrderID, c.gameName, r1.orderTimestamp, r1.dueDate " +
                    "FROM RentalOrder r1 " +
                    "JOIN GamesInOrder gm ON r1.rentalOrderID = gm.rentalOrderID " +
                    "JOIN Catalog c ON gm.gameID = c.gameID " +
                    "WHERE r1.login = '" + user_login + "'" +
                    "ORDER BY r1.orderTimestamp DESC " +
                    "LIMIT 5 ";
     System.out.print("Your recent 5 orders: ");
     esql.executeQueryAndPrintResult(query);
     System.out.print("\n");

  } catch (Exception e) {
     System.err.println(e.getMessage());
  }
} // end viewRecentOrders
```

9. **Option 7: View Order Info**

   Uses the rentalOrderID function firstly it checks if the person is a manager or a customer, if it is a manager then it can display all the rental orders associated with any rental order ID but if it is a customer, then it will only display the rental order that are associated with that login for security purposes.

```
public static void viewOrderInfo(GameRental esql, String user_login) {
    try {
        System.out.print("\nEnter your rental Order ID: ");
        String rentalOrderID = in.readLine();
        System.out.print("\n");

        // Check if user is a manager
        boolean isManager = isManager(esql, user_login);

        // Query to get the login associated with the rental order ID
        String query = String.format("SELECT login FROM RentalOrder WHERE rentalOrderID = '%s'", rentalOrderID);
        List<List<String>> result = esql.executeQueryAndReturnResult(query);

        if (result.isEmpty()) {
            System.out.println("Order ID not found. Returning to main menu.");
            return;
        }

        String orderLogin = result.get(0).get(0);

        // If user is not a manager, check if the order belongs to them
        if (!isManager && !orderLogin.equals(user_login)) {
            System.out.println("You do not have permission to view this order. Please input another order ID or quit to the main menu.");
            System.out.print("Enter '1' to input another order ID or '2' to quit: ");
            int choice = Integer.parseInt(in.readLine());
            if (choice == 1) {
                viewOrderInfo(esql, user_login);
            } else {
                return;
            }
        }

        // Query to get the order information
        query = "SELECT r1.rentalOrderID, r1.login, r1.noOfGames, r1.totalPrice, r1.orderTimestamp, r1.dueDate, " +
                "ti.trackingID, ti.status, ti.currentLocation, ti.courierName, ti.lastUpdateDate, ti.additionalComments " +
                "FROM RentalOrder r1 " +
                "JOIN TrackingInfo ti ON r1.rentalOrderID = ti.rentalOrderID " +
                "WHERE r1.rentalOrderID = '" + rentalOrderID + "'";

        System.out.print("\n");
        esql.executeQueryAndPrintResult(query);
```

10. Option 8: View Tracking Info

User will be prompted to enter the tracking ID and rental order ID and will provide the result based on that. Use of the rental order ID is essential for security because it is used to verify if the user should only be allowed to track their order, this combination of rental order and tracking order.

```
// OPTION 8
public static void viewTrackingInfo(GameRental esql, String user_login) {
   try {
      System.out.print("\nEnter your tracking ID: ");
      String trackingID = in.readLine();
      System.out.print("Enter your rental order ID: ");
      String rentalOrderID = in.readLine();
      System.out.print("\n");

      // Check if the user is a manager or employee
      boolean isEmployeeOrManager = isEmployeeOrManager(esql, user_login);

      // Query to get the login associated with the rental order ID
      String loginQuery = String.format("SELECT login FROM RentalOrder WHERE rentalOrderID = '%s'", rentalOrderID);
      List<List<String>> result = esql.executeQueryAndReturnResult(loginQuery);

      if (result.isEmpty()) {
         System.out.println("Order ID not found. Returning to main menu.");
         return;
      }

      String orderLogin = result.get(0).get(0);

      // If user is not a manager or employee, check if the order belongs to them
      if (!isEmployeeOrManager && !orderLogin.equals(user_login)) {
         System.out.println("You do not have permission to view this tracking information. Please input another order ID or quit to the main menu.");
         System.out.print("Enter '1' to input another order ID or '2' to quit: ");
         int choice = Integer.parseInt(in.readLine());
         if (choice == 1) {
            viewTrackingInfo(esql, user_login);
         } else {
            return;
         }
      }

      // Query to get the tracking information
      String query = "SELECT ti.trackingID, ti.rentalOrderID, ti.courierName, ti.currentLocation, " +
                     "ti.status, ti.lastUpdateDate, ti.additionalComments " +
                     "FROM TrackingInfo ti " +
                     "JOIN RentalOrder ro ON ti.rentalOrderID = ro.rentalOrderID " +
                     "WHERE ti.trackingID = '" + trackingID + "' AND ro.rentalOrderID = '" + rentalOrderID + "'";

      System.out.print("\n");
      esql.executeQueryAndPrintResult(query);

   } catch (Exception e) {
      System.err.println(e.getMessage());
   }
} // end viewTrackingInfo
```

11. **Option 9: Update Tracking Info**

This feature is only available for managers or employees, there is a check in the beginning to see if the logged-in user is either a manager or employee or a customer. If they are customer, then access is denied, but if they are managers or employees they can be given access and can update the tracking order status.

It uses 4 helpers:

- Is manager: which checks if the user logged in is a manager or not
- is employee: which checks if the user logged in is an employee or not
- isManageroremployee: a combination of both of them, that is used in function
- It has a timestamp helper that uses local time data library, which prompts the local time at my current location to provide a timestamp after manager or employee are finishing their process of updating the tracking status

```
public static void updateTrackingInfo(GameRental esql, String user_login) {
   try {

      if (!isEmployeeOrManager(esql, user_login)) {
         System.out.println("Access Denied: Only employees or managers can update the tracking information.");
         return;
      }

      System.out.print("\nEnter your tracking ID: ");
      String trackingID = in.readLine();

      System.out.print("Enter new status: ");
      String newStatus = in.readLine();

      System.out.print("Enter new current Location: ");
      String newLocation = in.readLine();

      System.out.print("Enter new courier name:");
      String newCourierName = in.readLine();

      System.out.print("Enter new additional comments: ");
      String newComments = in.readLine();

      Timestamp currentTimestamp = getCurrentTimestamp();

      String query = "UPDATE TrackingInfo " +
        "SET status = '" + newStatus + "', " +
        "currentLocation = '" + newLocation + "', " +
       "courierName = '" + newCourierName + "', " +
       "additionalComments = '" + newComments + "', " +
        "lastUpdateDate = '" + currentTimestamp + "' " +
        "WHERE trackingID = '" + trackingID + "'";

                  esql.executeUpdate(query);
               System.out.println("Tracking Information updated successfully.");
   }
```

12. Option 10: Update Catalog

If a user is a manager, they can choose to update any game theyd like from the catalog. Depending on what attribute and game they pic to update, we use the query “UPDATE Catalog SET <attribute> = <updated_value> WHERE gameID = <game_to_update>”. We execute this query using “esql.executeUpdate(query)” for each update the manager wants.

```
public static void updateCatalog(GameRental esql, String user_login) {
  try{
     if (isManager(esql, user_login)) {
        boolean update = true;

        while(update == true) {
           System.out.print("\nPlease input the gameID of the game you would like to update: ");
           String game_update = in.readLine();

           String game_name = String.format("SELECT gameName FROM Catalog WHERE gameID = '%s'", game_update);
           String game_genre = String.format("SELECT genre FROM Catalog WHERE gameID = '%s'", game_update);
           String game_price = String.format("SELECT price FROM Catalog WHERE gameID = '%s'", game_update);
           String game_description = String.format("SELECT description FROM Catalog WHERE gameID = '%s'", game_update);
           String game_imageURL = String.format("SELECT imageURL FROM Catalog WHERE gameID = '%s'", game_update);

           System.out.print("\n");
           esql.executeQueryAndPrintResult(game_name);

           System.out.print("\n");
           esql.executeQueryAndPrintResult(game_genre);

           System.out.print("\n");
           esql.executeQueryAndPrintResult(game_price);

           System.out.print("\n");
           esql.executeQueryAndPrintResult(game_description);

           System.out.print("\n");
           esql.executeQueryAndPrintResult(game_imageURL);

           System.out.print("\nWhat would you like to update? \n1. Game Name \n2. Genre \n3. Price \n4. Description \n5. imageURL \n");

           switch(readChoice()) {
              case 1: System.out.print("\nGame Name: ");
              String gamename_update = in.readLine();
              String query1 = String.format("UPDATE Catalog SET gameName = '%s' WHERE gameID = '%s'", gamename_update, game_update);
              esql.executeUpdate(query1); break;
```

13. Option 11: Update User

If a user is a manager they will be able to input any user they’d like to update, and will be able to choose which attribute they’d like to update. After inputting the updated value, the query “UPDATE Users SET <attribute> = <updated value> WHERE login = ‘user_login’”, followed by “esql.executeUpdate(query)” which effectively updates only the instance for the user that is inputted.

```java
public static void updateUser(GameRental esql, String user_login) {
  try{
    if (isManager(esql, user_login)) { // choices for a manager
      boolean update = true;

      while(update == true) {
        System.out.print("\nEnter the user's login to update: ");
        String user_update = in.readLine();

        if (isUser(esql,user_update)) { // if valid user found
          String userlogin = String.format("SELECT login FROM Users WHERE login = '%s'", user_update);
          String user_password = String.format("SELECT password FROM Users WHERE login = '%s'", user_update);
          String user_role = String.format("SELECT role FROM Users WHERE login = '%s'", user_update);
          String user_games = String.format("SELECT favGames FROM Users WHERE login = '%s'", user_update);
          String user_phone = String.format("SELECT phoneNum FROM Users WHERE login = '%s'", user_update);
          String user_overdue = String.format("SELECT numOverDueGames FROM Users WHERE login = '%s'", user_update);

          System.out.print("\n");
          esql.executeQueryAndPrintResult(userlogin);
          System.out.print("\n");
          esql.executeQueryAndPrintResult(user_password);
          System.out.print("\n");
          esql.executeQueryAndPrintResult(user_role);
          System.out.print("\n");
          esql.executeQueryAndPrintResult(user_games);
          System.out.print("\n");
          esql.executeQueryAndPrintResult(user_phone);
          System.out.print("\n");
          esql.executeQueryAndPrintResult(user_overdue);

          System.out.print("\nWhat would you like to update? \n1. User's Role \n2. User's Number of Overdue Games \n");

          switch(readChoice()) {
            case 1: System.out.print("\nUser's New Role: ");
            String role_update = in.readLine();
            String query1 = String.format("UPDATE Users SET role = '%s' WHERE login = '%s'", role_update, user_update);
            esql.executeUpdate(query1); break;

            case 2: System.out.print("\nUser's New Number of Overdue Games: ");
            String overdue_update = in.readLine();
            String query2 = String.format("UPDATE Users SET numOverDueGames = '%s' WHERE login = '%s'", overdue_update, user_update);
            esql.executeUpdate(query2); break;
          }

          System.out.print("\nProfile updated succesfully. Would you like to update again? \n1. Yes \n2. No\n");
          switch(readChoice()) {
            case 1: break;
            case 2: update = false; break;
          }
        }
        else {
          System.out.print("\nUser not found. Please try again.\n");
        }
      }
    }
    else { // not manager!
      System.out.print("\nYou do not have permission to access this.\n");
      return;
    }
  } catch(Exception e){
    System.err.println (e.getMessage ());
  }
} // end updateUser
```

Problems/Findings:

At the very start of the project, we were unable to figure out how to access the information of the user that is already logged in, until we did a deep dive of the java file and realized that authorizedUser is apart of the login function. We then realized that we could pass in authorizedUser as parameter within the functions itself, which would be the user's login. With the user's login being passed into each function called from the main menu, we were able to effectively access their information within menu option functions with a simple query: "SELECT <> FROM <> WHERE login = user_login. We also had issues figuring out how to access the cost of a game when implementing the place order function. The issue lied in the fact that when we executed the query to access the price of a game, it would be in this form: "<<35.0>>" if a game cost $35 for example. We were able to work around this by first extracting the values into a string, and then parsing this new string to effectively convert it into a double.

Lastly, we had issues with the place order function, specifically, we were unable to add a new row into the GamesInOrder table. An error occurred whenever we tried to run the function because we first executed an INSERT into the GamesInOrder table before executing an INSERT into the RentalOrder table. We realized that this was because the GamesInOrder table specifies that "rentalOrderID" is a foreign key that references the RentalOrder table. Therefore, to fix this, we created the unique rental ID first and inserted a row into the RentalOrder table with this ID. That way, when we insert a row into GamesInOrder, the rentalOrderID that we are using in that new instance already exists.

Contributions:

Ynah worked on the initial project setup, covering user creation and login, profile management (updating and viewing), and rental order placements, and allowed managers to view and update both the information of users as well as the catalog information. She also implemented helper functions to ensure that only users with specific permissions can access specific functions. Meanwhile, Nishant implemented all rental history functionalities including displaying the most 5 recent rental orders, displaying all rental orders, and displaying a specific rental order. He also implemented helper functions to ensure that the logged-in customer cannot see anyone else's orders, and they should only have access to their individual rental history, as well as due order time stamp function, along with separate helps for option 5 which is place order.

Extra credit:

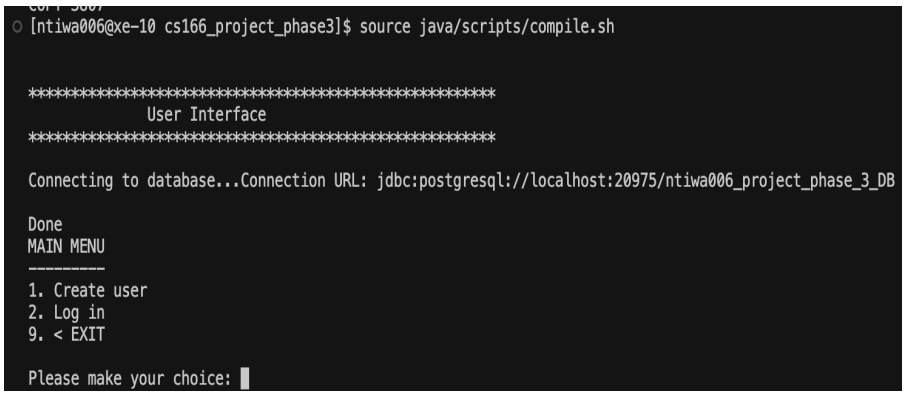
We also implemented indexes for the columns as well, file is attached with submission.

```sql
-- Drop indexes if they exist
DROP INDEX IF EXISTS index_users_login;
DROP INDEX IF EXISTS index_rentalOrder_login;

DROP INDEX IF EXISTS index_catalog_gameID;
DROP INDEX IF EXISTS index_gamesInOrder_gameID;

DROP INDEX IF EXISTS index_rentalOrder_rentalOrderID;
DROP INDEX IF EXISTS index_trackingInfo_rentalOrderID;
DROP INDEX IF EXISTS index_gamesInOrder_rentalOrderID;

DROP INDEX IF EXISTS index_trackingInfo_trackingID;

-- Create indexes
CREATE INDEX IF NOT EXISTS index_users_login ON Users(login);
CREATE INDEX IF NOT EXISTS index_rentalOrder_login ON RentalOrder(login);

CREATE INDEX IF NOT EXISTS index_catalog_gameID ON Catalog(gameID);
CREATE INDEX IF NOT EXISTS index_gamesInOrder_gameID ON GamesInOrder (gameID);

CREATE INDEX IF NOT EXISTS index_rentalOrder_rentalOrderID ON RentalOrder(rentalOrderID);
CREATE INDEX IF NOT EXISTS index_trackingInfo_rentalOrderID ON TrackingInfo(rentalOrderID);
CREATE INDEX IF NOT EXISTS index_gamesInOrder_rentalOrderID ON GamesInOrder(rentalOrderID);

CREATE INDEX IF NOT EXISTS index_trackingInfo_trackingID ON TrackingInfo(trackingID);
```

```
[ntiwa006@xe-10 cs166_project_phase3]$ source sql/scripts/create_db.sh
NOTICE:  drop cascades to constraint rentalorder_login_fkey on table rentalorder
DROP TABLE
NOTICE:  drop cascades to constraint gamesinorder_gameid_fkey on table gamesinorder
DROP TABLE
NOTICE:  drop cascades to 2 other objects
DETAIL:  drop cascades to constraint trackinginfo_rentalorderid_fkey on table trackinginfo
drop cascades to constraint gamesinorder_rentalorderid_fkey on table gamesinorder
DROP TABLE
DROP TABLE
DROP TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
NOTICE:  index "index_users_login" does not exist, skipping
DROP INDEX
NOTICE:  index "index_rentalorder_login" does not exist, skipping
DROP INDEX
NOTICE:  index "index_catalog_gameid" does not exist, skipping
DROP INDEX
NOTICE:  index "index_gamesinorder_gameid" does not exist, skipping
DROP INDEX
NOTICE:  index "index_rentalorder_rentalorderid" does not exist, skipping
DROP INDEX
NOTICE:  index "index_trackinginfo_rentalorderid" does not exist, skipping
DROP INDEX
NOTICE:  index "index_gamesinorder_rentalorderid" does not exist, skipping
DROP INDEX
NOTICE:  index "index_trackinginfo_trackingid" does not exist, skipping
DROP INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
COPY 500
COPY 500
COPY 3007
COPY 3007
COPY 3807
[ntiwa006@xe-10 cs166_project_phase3]$
```

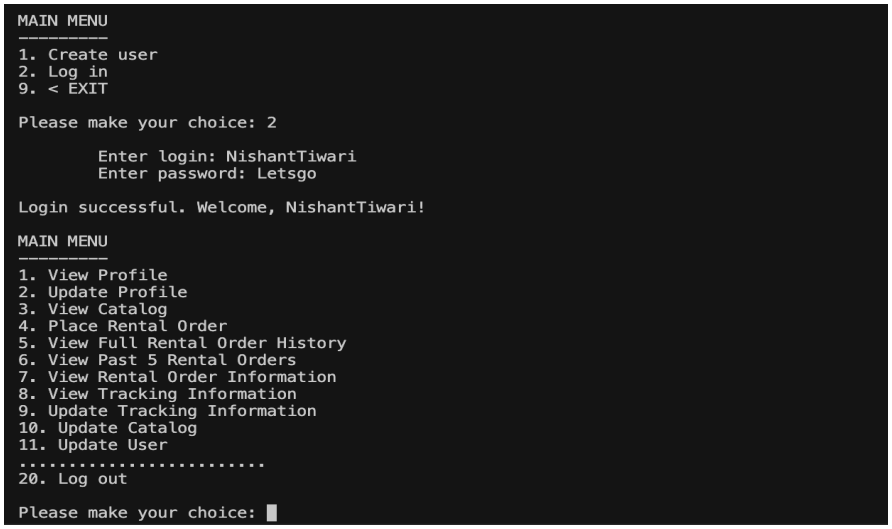
Screenshots

1. Welcome Page after compiling the program.

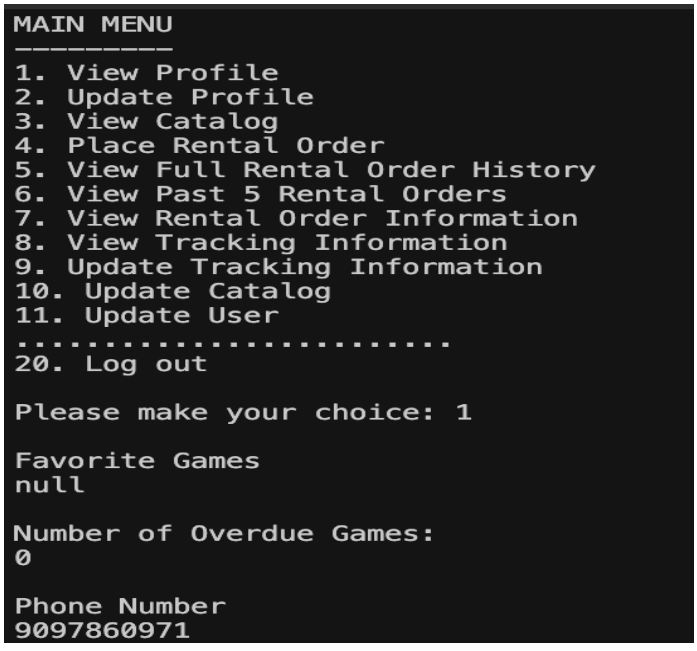


2. Testing the create user program: Login: NishantTiwari, Password: Letsgo, Phone number: 9097860971, and once you hit enter, new user will be successfully created and prompted back to main menu window.

3. Testing the login function, user NishantTiwari successfully logs in using the login and password created previously.

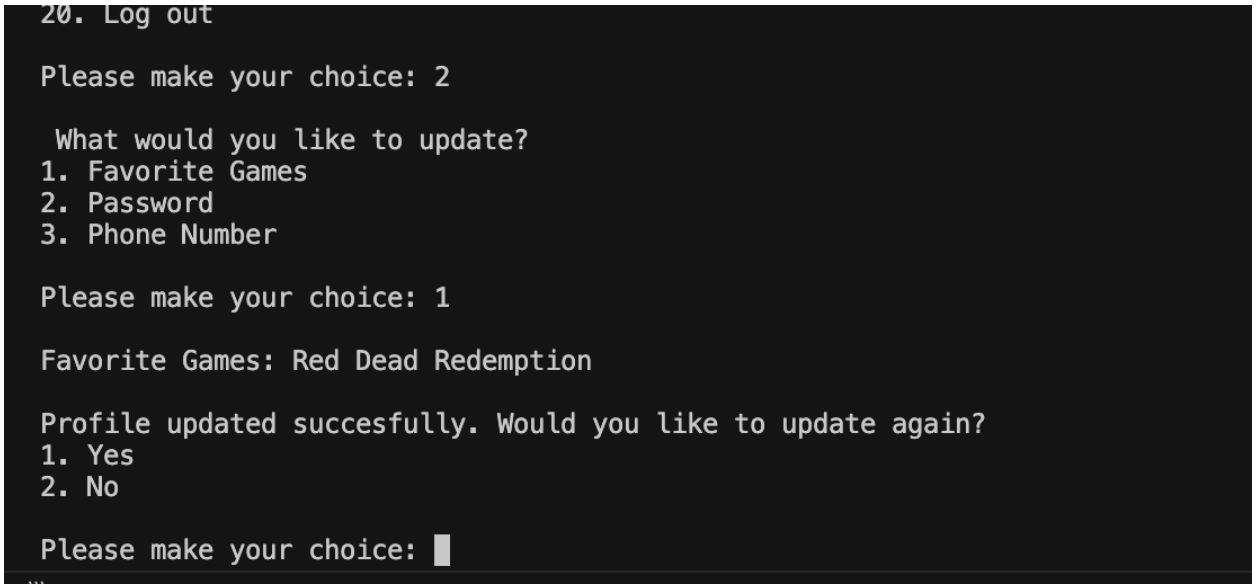


4. Testing the view profile: it displays all the information available in database for user NishantTiwari.



5. Testing the update profile of user NishantTiwari

6. Updating the favorite game to Red Dead Redemption.



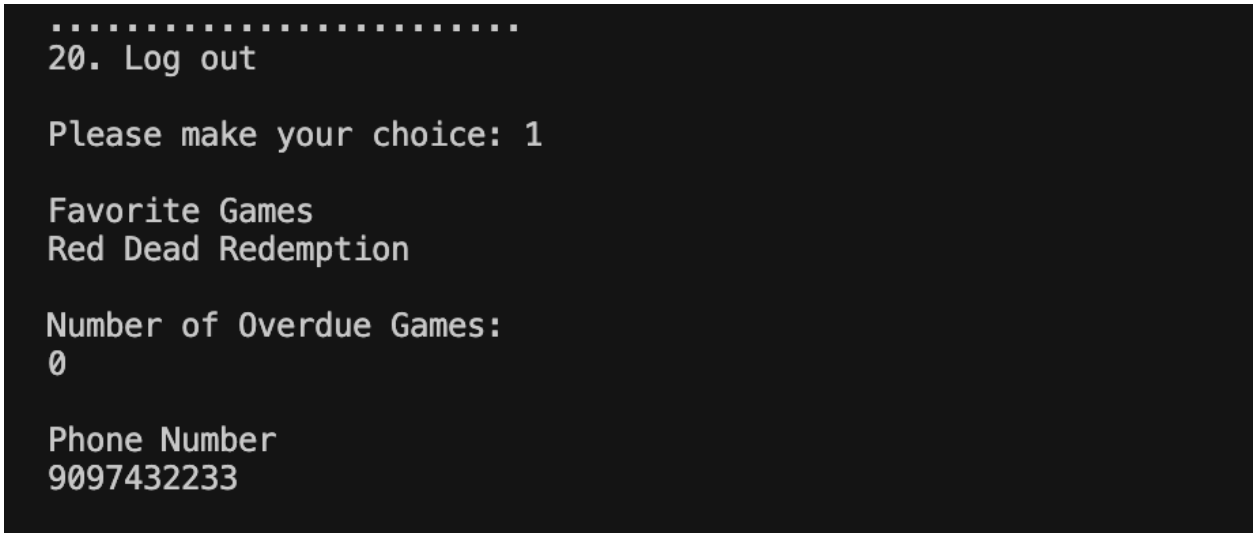
7. Updating password to ‘CS166Goat’

```
Please make your choice: 1

Favorite Games: Red Dead Redemption

Profile updated succesfully. Would you like to update again?
1. Yes
2. No

Please make your choice: 1

 What would you like to update?
1. Favorite Games
2. Password
3. Phone Number

Please make your choice:
```

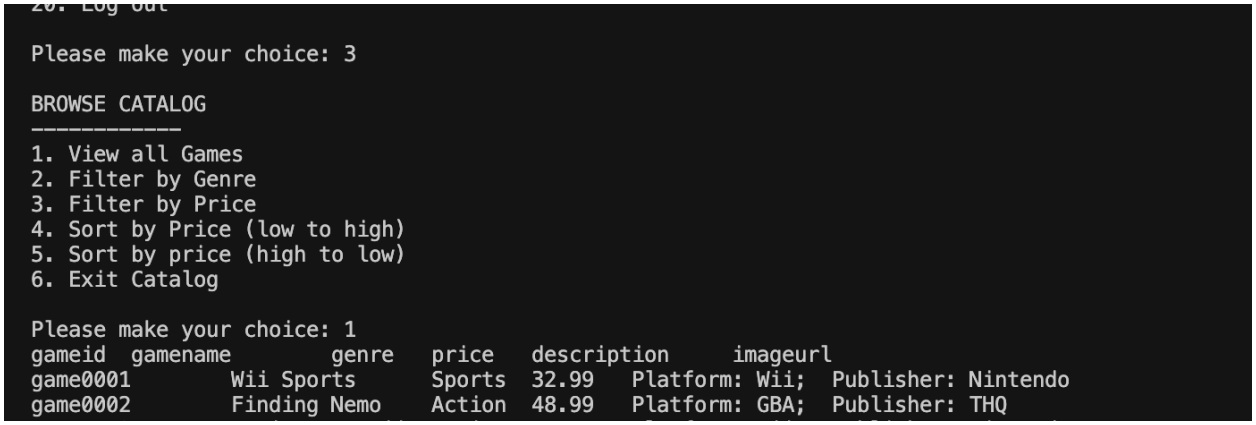
8. Updating the phonenumber to: 9097432233

```
2. No

Please make your choice: 1

 What would you like to update?
1. Favorite Games
2. Password
3. Phone Number

Please make your choice: 3

New Phone Number: 9097432233

Profile updated succesfully. Would you like to update again?
1. Yes
2. No

Please make your choice:
```

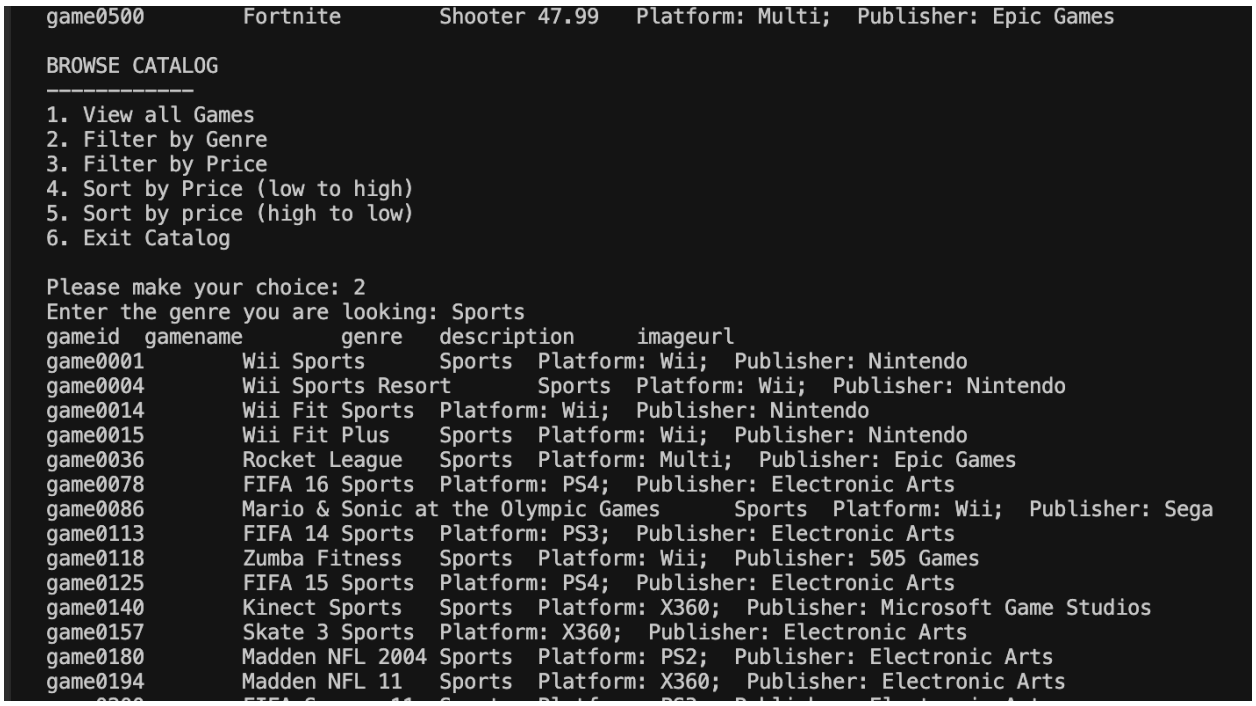
9. In this screenshot you can see the all reflected changes, such as new favorite game as well as phone number, due to security reasons password is not shown here.



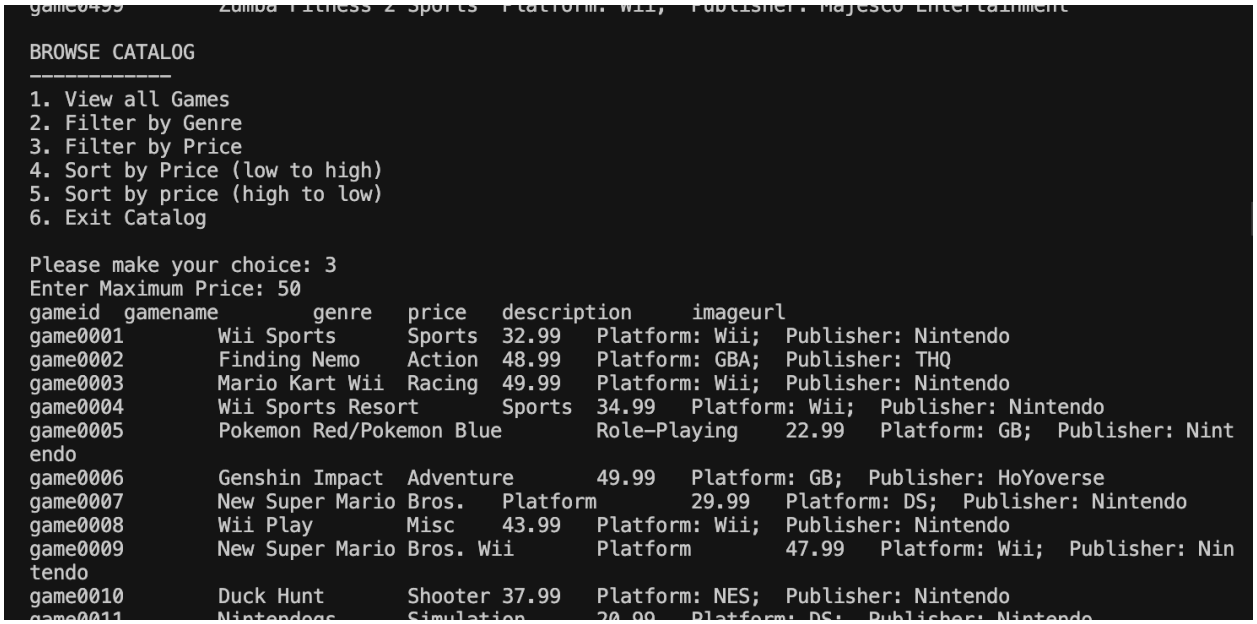
10. Testing the catalog, function: 1. View all games displays everything thats in catalog, Screenshot shows only few because it is very hard to take the screenshot of whole interface.



11. Testing the filter by genre: in this case it is sports and it will display all the game that has sports genre.



12. Filtering by price enables user to select the maximum price range by themselves and it will display all the games.



13. Sort by price (low to high): games will be arranged in low to high with the lowest price at the top going to bottom.

```
BROWSE CATALOG
------------
1. View all Games
2. Filter by Genre
3. Filter by Price
4. Sort by Price (low to high)
5. Sort by price (high to low)
6. Exit Catalog

Please make your choice: 4
gameid  gamename        genre   price   description     imageurl
game0498        World Soccer Winning Eleven 7 International     Sports  15.99   Platform: PS2;  Publis
her: Konami Digital Entertainment
game0029        Gran Turismo 3: A-Spec  Racing  15.99   Platform: PS2;  Publisher: Sony Computer Enter
tainment
game0490        Valorant        Shooter 15.99   Platform: Multi;  Publisher: Riot Games
game0083        FIFA Soccer 13  Action  15.99   Platform: PS3;  Publisher: Electronic Arts
game0348        Professor Layton and the Last Specter   Puzzle  15.99   Platform: DS;  Publisher: Nint
endo
game0233        The Legend of Zelda: The Wind Waker     Action  15.99   Platform: GC;  Publisher: Nint
endo
game0268        Warcraft II: Tides of Darkness  Strategy        15.99   Platform: PC;  Publisher: Acti
vision
game0240        Pitfall!        Platform        15.99   Platform: 2600;  Publisher: Activision
```

14. Sort by price (high to low) will display the highest price games on top and then go down in descending order.

```
BROWSE CATALOG
------------
1. View all Games
2. Filter by Genre
3. Filter by Price
4. Sort by Price (low to high)
5. Sort by price (high to low)
6. Exit Catalog

Please make your choice: 5
gameid  gamename        genre   price   description     imageurl
game0284        FIFA Soccer 07  Sports  50.99   Platform: PS2;  Publisher: Electronic Arts
game0060        Skylanders Giants       Action  50.99   Platform: Wii;  Publisher: Activision
game0387        Imagine: Fashion Designer       Simulation      50.99   Platform: DS;  Publisher: Ubis
oft
game0197        Resident Evil 5 Action  50.99   Platform: PS3;  Publisher: Capcom
game0031        Pokemon Yellow: Special Pikachu Edition Role-Playing    50.99   Platform: GB;  Publish
er: Nintendo
game0298        Ratchet & Clank: Going Commando Platform        50.99   Platform: PS2;  Publisher: Son
```

15. Testing the place order function: placing an order using game id and no of units we would like to order, it will display the total price, and once it is accepted by user, it will generate the orderID and tracking ID to user.

```
5. View Full Rental Order History
6. View Past 5 Rental Orders
7. View Rental Order Information
8. View Tracking Information
9. Update Tracking Information
10. Update Catalog
11. Update User
.......................
20. Log out

Please make your choice: 4

--PLACE RENTAL ORDER--
Enter the game ID of the game you'd like to order: game0002
How many units would you like?: 2

Current Price: $97.98

Would you like to order more? (Y or N): N

The total cost of your order is: $97.98

Your order has been placed.
Order ID: gamerentalorder4446
Tracking ID: trackingid5019
```

16. Placing multiple orders to test the functionality.

```
Please make your choice: 4

--PLACE RENTAL ORDER--
Enter the game ID of the game you'd like to order: game0004
How many units would you like?: 4

Current Price: $139.96

Would you like to order more? (Y or N): Y

--PLACE RENTAL ORDER--
Enter the game ID of the game you'd like to order: game0002
How many units would you like?: 0

Current Price: $139.96

Would you like to order more? (Y or N): N

The total cost of your order is: $139.96

Your order has been placed.
Order ID: gamerentalorder5832
Tracking ID: trackingid7465
```

17. Placing more orders…

```
Would you like to order more? (Y or N): Y

--PLACE RENTAL ORDER--
Enter the game ID of the game you'd like to order: game0010
How many units would you like?: 1

Current Price: $81.98

Would you like to order more? (Y or N): N

The total cost of your order is: $81.98

Your order has been placed.
Order ID: gamerentalorder9206
Tracking ID: trackingid6461
```

18. Placing more orders.

```
Please make your choice: 4

--PLACE RENTAL ORDER--
Enter the game ID of the game you'd like to order: game0020
How many units would you like?: 2

Current Price: $81.98

Would you like to order more? (Y or N): N

The total cost of your order is: $81.98

Your order has been placed.
Order ID: gamerentalorder7768
Tracking ID: trackingid9469
```

19. Placing more orders.

```
Please make your choice: 4

--PLACE RENTAL ORDER--
Enter the game ID of the game you'd like to order: game0036
How many units would you like?: 1

Current Price: $17.99

Would you like to order more? (Y or N): N

The total cost of your order is: $17.99

Your order has been placed.
Order ID: gamerentalorder6500
Tracking ID: trackingid4195
```

20. Placing more orders

```
Please make your choice: 4

--PLACE RENTAL ORDER--
Enter the game ID of the game you'd like to order: game0051
How many units would you like?: 2

Current Price: $83.98

Would you like to order more? (Y or N): N

The total cost of your order is: $83.98

Your order has been placed.
Order ID: gamerentalorder6638
Tracking ID: trackingid7300
```

21. Placing more orders

```
Please make your choice: 4

--PLACE RENTAL ORDER--
Enter the game ID of the game you'd like to order: game0057
How many units would you like?: 2

Current Price: $85.98

Would you like to order more? (Y or N): Y

--PLACE RENTAL ORDER--
Enter the game ID of the game you'd like to order: game0058
How many units would you like?: 1

Current Price: $123.97

Would you like to order more? (Y or N): N

The total cost of your order is: $123.97

Your order has been placed.
Order ID: gamerentalorder6470
Tracking ID: trackingid7048
```

22. Testing the view all rental orders. It will display all the rental orders under Nishant

```
.....................
20. Log out

Please make your choice: 5
Your order history: rentalorderid       gamename        ordertimestamp  duedate
gamerentalorder4446     Finding Nemo    2024-06-11 14:52:56.091 2024-06-16 14:52:56.026
gamerentalorder5832     Finding Nemo    2024-06-11 14:53:37.259 2024-06-25 14:53:37.259
gamerentalorder5832     Wii Sports Resort       2024-06-11 14:53:37.259 2024-06-25 14:53:37.259
gamerentalorder9206     Duck Hunt       2024-06-11 14:54:19.762 2024-06-16 14:54:19.762
gamerentalorder9206     Wii Play        2024-06-11 14:54:19.762 2024-06-16 14:54:19.762
gamerentalorder7768     Brain Age: Train Your Brain in Minutes a Day    2024-06-11 14:54:54.043 2024-0
6-16 14:54:54.043
gamerentalorder6500     Rocket League   2024-06-11 14:55:35.596 2024-06-23 14:55:35.596
gamerentalorder6638     Super Mario Land 2: 6 Golden Coins      2024-06-11 14:55:56.884 2024-07-01 14:
55:56.884
gamerentalorder6470     Super Mario All-Stars   2024-06-11 14:56:18.607 2024-07-02 14:56:18.607
gamerentalorder6470     Grand Theft Auto IV     2024-06-11 14:56:18.607 2024-07-02 14:56:18.607
```

23. Testing the most recent "5" orders from the list.

```
Please make your choice: 6
Your recent 5 orders: rentalorderid     gamename        ordertimestamp  duedate
gamerentalorder6470     Grand Theft Auto IV     2024-06-11 14:56:18.607 2024-07-02 14:56:18.607
gamerentalorder6470     Super Mario All-Stars   2024-06-11 14:56:18.607 2024-07-02 14:56:18.607
gamerentalorder6638     Super Mario Land 2: 6 Golden Coins      2024-06-11 14:55:56.884 2024-07-01 14:
55:56.884
gamerentalorder6500     Rocket League   2024-06-11 14:55:35.596 2024-06-23 14:55:35.596
gamerentalorder7768     Brain Age: Train Your Brain in Minutes a Day    2024-06-11 14:54:54.043 2024-0
6-16 14:54:54.043
```

24. Testing the process finding the most specific rental order using rental order id.

```
Please make your choice: 7

Enter your rental Order ID: gamerentalorder6470


rentalorderid   login   noofgames       totalprice      ordertimestamp  duedate trackingid      status
currentlocation couriername     lastupdatedate  additionalcomments
gamerentalorder6470     NishantTiwari   3       123.97  2024-06-11 14:56:18.607 2024-07-02 14:56:18.60
7       trackingid7048  Processing      Los Angeles, CA USPS    2024-06-11 14:56:18.607
```

25. Checking the tracking status of the rental order using tracking id and rental order id.

```
Please make your choice: 8

Enter your tracking ID: trackingid7048
Enter your rental order ID: gamerentalorder6470


trackingid      rentalorderid   couriername     currentlocation status  lastupdatedate  additionalcomm
ents
trackingid7048  gamerentalorder6470     USPS    Los Angeles, CA Processing      2024-06-11 14:56:18.60
7

MAIN MENU
```

26. If user tries to access the update tracking which is only available for employee and manager it will print a access denied error

```
Please make your choice: 9
Access Denied: Only employees or managers can update the tracking information.
```

27. If user tries to access the update catalog feature then it will also print a error message of access denied because that feature is only for managers.

```
Please make your choice: 10

You do not have permision to access this.
```

28. Same case for updating profile if user tries to access it

```
Please make your choice: 11

You do not have permission to access this.
```

29. Logging out, ending page.

```
MAIN MENU
---------
1. View Profile
2. Update Profile
3. View Catalog
4. Place Rental Order
5. View Full Rental Order History
6. View Past 5 Rental Orders
7. View Rental Order Information
8. View Tracking Information
9. Update Tracking Information
10. Update Catalog
11. Update User
.........................
20. Log out

Please make your choice: 20
MAIN MENU
---------
1. Create user
2. Log in
9. < EXIT

Please make your choice: 9
Disconnecting from database...Done

Bye !
```