**E-commerce – Term Project**

**Deliverable 2: User Stories**

**Employee**

|  |
| --- |
| **Card**: As an admin, I can register a seller.  **Conversation**: For security purposes, only an admin can register a new seller. If the user is logged in as an admin, they have the option of creating a new seller account. All other users do not have this option.  **Conformity**: Create a new account and then successfully log in as this new user.  **SQL/Implementation**: We use an INSERT statement.  INSERT INTO employee(employee\_id, username, password\_hash, isAdmin)  VALUES(1, ‘johndoe’, ‘password’, false); |

|  |
| --- |
| **Card**: As an admin, I can edit a seller’s account information.  **Conversation**: For security purposes, only an admin can edit a seller’s account information. If the user is logged in as an admin, they have the option of editing a seller’s account information for any seller currently in the database. All other users do not have this option.  **Conformity**: Login using the changed credentials. If the isAdmin property was changed, check if the seller has gained or lost admin privileges.  **SQL/Implementation**: We use an UPDATE statement.  UPDATE employee SET username = ‘jondoe’, isAdmin = true  WHERE username = ‘johndoe’; |

|  |
| --- |
| **Card**: As an admin, I can unregister a seller.  **Conversation**: For security purposes, only an admin can unregister a seller. If the user is logged in as an admin, they have the option of deleting a seller’s account. All other users do not have this option. Before being unregistered, all of the tickets associated with that seller will be unassigned to be reassigned at a later time.  **Conformity**: Login using the credentials of the recently deleted seller account. The account should no longer exist so the login should be unsuccessful.  **SQL/Implementation**: We use an UPDATE and a DELETE statement.  UPDATE ticket SET assigned\_to = NULL  WHERE assigned\_to = ‘johndoe’;  DELETE FROM employee  WHERE username = ‘johndoe’; |

|  |
| --- |
| **Card**: As a seller, I can login.  **Conversation**: A seller can login to the system with their given credentials at the login section. No actions can be taken on the site without being logged in.  **Conformity**: Create a new seller in the database then attempt to login using the newly created seller’s credentials. If the correct information is used, the login will be successful. If the wrong information is used, the login will fail and redirect to the login page.  **SQL/Implementation**: We use a SELECT statement to display all skins once the seller is logged in.  SELECT \* FROM skin  WHERE available = 1;  We also use $\_SESSION['user\_id'] = $user->user\_id to login the seller. |

|  |
| --- |
| **Card**: As a seller, I can logout.  **Conversation**: A seller can logout of the system with the logout button.  **Conformity**: Logout with the logout button. Check if we can access the sales page. If the logout was successful, we should not be able to view the page. It should redirect us to the login page.  **SQL/Implementation**: No SQL is required. We use session\_destroy() to logout the seller. |

|  |
| --- |
| **Card**: As a seller, I can change my password.  **Conversation**: A seller can change their password at any time with the change password button. They will need to provide a new password and confirm it to save the changes.  **Conformity**: Change the password with the same value to see that it doesn’t change. Change password but enter a different password in the confirmation box to see that it doesn’t change. Change password correctly in both boxes to see that it does save the changes. Then logout and try to log back in to see if the changes were successful.  **SQL/Implementation**: We use an UPDATE statement  UPDATE employee SET password\_hash = ‘newPassword’  WHERE username = ‘johndoe’; |

|  |
| --- |
| **Card**: As a seller, I can add RP to a user’s account.  **Conversation**: A seller can add RP to a user’s account if an issue is brought up in a ticket and a skin needs to be refunded.  **Conformity**: Login as a seller and give a user RP then login as said user and see if they gained the RP.  **SQL/Implementation**: We use an UPDATE statement  UPDATE user SET riot\_points = riot\_points + 975  WHERE username = ‘jondoe’; |

|  |
| --- |
| **Card**: As an employee, I can sort skins by name.  **Conversation**: An employee can choose to order skins in the shop alphabetically.  **Conformity**: When reviewing the list of skins, the default order should be the order that the skins were added. Click the alphabetical checkbox on the left and verify that the skin names are now sorted in alphabetical order.  **SQL/Implementation**: We use a SELECT statement.  SELECT \* FROM skin  WHERE available = 1  ORDER BY name; |

|  |
| --- |
| **Card**: As an employee, I can sort skins by price.  **Conversation**: An employee can order skins in the list of skins by price, from lowest to highest.  **Conformity**: When reviewing the list of skins, click the price checkbox on the left side and pick and verify that the skins are now sorted from lowest to highest price.  **SQL/Implementation**: We use a SELECT statement.  SELECT \* FROM skin  WHERE available = 1  ORDER BY price – discount \* price; |

**Skin**

|  |
| --- |
| **Card**: As a seller, I can discount skins.  **Conversation**: A seller can change the price of selected skins either by changing the price of the skin of by giving it a temporary discount.  **Conformity**: Change the price of a skin and see if the price is affected on the home page that displays all the skins for sale. Change the discount for a skin to see if the price is affected on the home page.  **SQL/Implementation**: We use an UPDATE statement.  UPDATE skin SET discount = .1  WHERE skin\_id = 1; |

|  |
| --- |
| **Card**: As a seller, I can track the sales of each skin.  **Conversation**: A seller can see how much money each skin has made as a list or by viewing an individual skin.  **Conformity**: Buy a skin and see if that skin’s sale’s total has gone up by the amount the skin costs on the home page that displays all the skins for sale.  **SQL/Implementation**: We use an UPDATE statement.  UPDATE skin SET sold = sold + 15  WHERE skin\_id = 1; |

|  |
| --- |
| **Card**: As a seller, I can add a new skin to be sold.  **Conversation**: A seller can add new skins to be sold to the website.  **Conformity**: Add a new skin to the database and see that the list of skins on the home page includes the new skin.  **SQL/Implementation**: We use an INSERT statement.  INSERT INTO skin(name, champion, description, price, discount, sold, image, available)  VALUES(‘Arcade Ahri’, Ahri, ‘Bright Colors’, 1350, 0, 0, ‘imageFile.jpg’, 1); |

|  |
| --- |
| **Card**: As a seller, I can edit a skin that is currently being sold.  **Conversation**: A seller can edit a skin that is currently being sold on the website.  **Conformity**: Change the name of a skin that is in the database and see if the changes apply by viewing the list of skins. Change the image of a skin that is in the database and see if the changes apply by viewing the list of skins.  **SQL/Implementation**: We use an UPDATE statement.  UPDATE skin SET name = ‘Super Arcade Ahri’  WHERE skin\_id = 1; |

|  |
| --- |
| **Card**: As a seller, I can remove a skin that is currently being sold.  **Conversation**: A seller can remove a skin that is currently being sold on the website.  **Conformity**: Create a skin and set the availability to true. View the list of skins on the home page and verify that the skin is in the list. Change the availability of that skin from true to false and see if it is still in the list of skins to be sold on the home page.  **SQL/Implementation**: We use an UPDATE statement.  UPDATE skin SET available = 0  WHERE skin\_id = 1; |

|  |
| --- |
| **Card**: As a seller, I can remove skins from the list of discounted skins.  **Conversation**: A seller can remove a skin from the discounted skins table in the database.  **Conformity**: Remove a skin from the list of discounted skins and see if the price is reverted to its normal price.  **SQL/Implementation**: We use an UPDATE statement.  UPDATE skin SET discount = 0  WHERE skin\_id = 1; |

|  |
| --- |
| **Card**: As a seller, I can add skins to the list of discounted skins.  **Conversation**: A seller can add a skin to the discounted skins table in the database.  **Conformity**: Add a skin to the list of discounted skins and see if the price is changed to match the given discount.  **SQL/Implementation**: We use an UPDATE statement.  UPDATE skin SET discount = 10  WHERE skin\_id = 1; |

|  |
| --- |
| **Card**: As a user, I can see the list of skins that are on sale.  **Conversation**: A user can choose to display only the skins that are on sale.  **Conformity**: When browsing the shop, check the “on sale” checkbox. Verify that the shop only displays all the skins that are on sale.  **SQL/Implementation**: We use a SELECT statement.  SELECT \* FROM skin  WHERE discount > 0; |

|  |
| --- |
| **Card**: As a user, I can sort skins by name.  **Conversation**: A user can choose to order skins in the shop in alphabetically, from A-Z or vice versa.  **Conformity**: When browsing the shop, the default order should be alphabetical, ascending. Click the “order by” dropdown on the left side and pick “Alphabetical (desc)”. Verify that the skin names are now sorted in descending order alphabetically.  **SQL/Implementation**: We use a SELECT statement.  SELECT \* FROM skin  ORDER BY name DESC, champion, DESC; |

|  |
| --- |
| **Card**: As a user, I can sort skins by price.  **Conversation**: A user can order skins in the shop by price, from lowest to highest or vice versa.  **Conformity**: When browsing the shop, click the “order by” dropdown on the left side and pick “Price (asc)”. Verify that the skins are now sorted from lowest to highest price.  **SQL/Implementation**: We use a SELECT statement.  SELECT \* FROM skin  ORDER BY price – discount \* price; |

|  |
| --- |
| **Card**: As a user, I can search for skins by name or by champion name.  **Conversation**: A user can search for skins by entering the name or champion name in the search field.  **Conformity**: Add 2 skins for the champion “Ahri”. When browsing the shop, enter “ahri” in the search field. Verify that the 2 skins are displayed.  **SQL/Implementation**: We use a SELECT statement.  SELECT \* FROM skin  WHERE name LIKE ‘%ahri%’ OR champion LIKE ‘%ahri%’; |

**User**

|  |
| --- |
| **Card**: As a user, I can register an account.  **Conversation**: A user can register an account to play the game and make purchases.  **Conformity**: Login using the credentials of the newly created user account.  **SQL/Implementation**: We use an INSERT statement.  INSERT INTO user(user\_id, username, password\_hash, riot\_points)  VALUES(1, ‘johndoe’, ‘password’, 0); |

|  |
| --- |
| **Card**: As a user, I can login.  **Conversation**: A user can login to the system with their given credentials at the login section. No actions can be taken on the site without being logged in.  **Conformity**: Login using the user’s credentials. If the correct information is used, the login will be successful. If the wrong information is used, the login will fail and redirect to the login page.  **SQL/Implementation**: We use a SELECT statement to display all skins once the user is logged in and browsing the shop.  SELECT \* FROM skin  WHERE available = 1;  We also use $\_SESSION['user\_id'] = $user->user\_id to login the user. |

|  |
| --- |
| **Card**: As a user, I can change my username.  **Conversation**: A user can change their username if it is not already taken.  **Conformity**: Click on your username in the top right and select the “change username” option when the dropdown appears. Enter the desired username in the confirmation box. A null username will not be accepted. If the username is taken, a message should be displayed that the username is already taken. If the username is available, the user is logged out. If the user attempts to log in with the new username, the login should be successful.  **SQL/Implementation**: We use an UPDATE statement.  UPDATE user SET username = ‘newUsername’  WHERE user\_id = 1; |

|  |
| --- |
| **Card**: As a user, I can change my password.  **Conversation**: A user can change their password.  **Conformity**: Click on your username in the top right and select the “change password” option when the dropdown appears. Enter the new password in both the new password box and the confirmation box. A null password will not be accepted. If both password boxes match, the user should be logged out. If the user attempts to log in with the new password, the login should be successful.  **SQL/Implementation**: We use an UPDATE statement.  UPDATE user SET password\_hash = ‘newPassword’  WHERE user\_id = 1; |

|  |
| --- |
| **Card**: As a user, I can convert real money to in-game currency called RP (Riot Points).  **Conversation**: A user can spend money to purchase RP, which is required to buy skins. RP is sold at 3 different prices, with higher prices giving more RP per dollar. 10$ gives 1020 RP, 25$ gives 2600 RP, and 50$ gives 5350 RP.  **Conformity**: Click the “Buy RP” button in the shop. Select the 10$, 25$ or 50$ option and confirm  your purchase. Verify that the correct amount of RP was added to your account.  **SQL/Implementation**: We use an UPDATE statement.  UPDATE user SET riot\_points = riot\_points + 1020  WHERE user\_id = 1; |

|  |
| --- |
| **Card**: As a user, I can proceed to checkout.  **Conversation**: A user can proceed to checkout once they have added at least 1 skin to their cart. The user can not add a skin to their cart that they already own.  **Conformity**: Add a skin to the cart and click on the proceed to checkout button. Click the buy button and see if the proper amount of RP is removed from the account and if the skin is added to the users account. Add a skin to the cart that costs more RP than the user has to make sure the purchase doesn’t go through and instead redirects to the buy RP page.  **SQL/Implementation**: We use an UPDATE statement to remove RP from the user. We also use an UPDATE statement to remove the item from the cart.  UPDATE user SET riot\_points = riot\_points - 1500  WHERE user\_id = 1;  UPDATE skin\_owned SET in\_cart = false  WHERE user\_id = 1 AND skin\_id = 1; |

**Skin\_owned**

|  |
| --- |
| **Card**: As a user, I can filter the list of skins to remove owned skins.  **Conversation**: A user can choose to filter out the skins they already own in the list of skins to buy.  **Conformity**: Uncheck the “Show owned skins” checkbox and see if the skins the user owns are removed from the list. Then buy a new skin and check to see if it still works.  **SQL/Implementation**: We use a SELECT statement.  SELECT \*  FROM skin s, skin\_owned so USING (skin\_id)  WHERE so.user\_id = 1; |

|  |
| --- |
| **Card**: As a user, I can add a skin to my cart.  **Conversation**: A user can add skins to their cart before checkout.  **Conformity**: When browsing the shop, click the “add to cart” button below several skins. View cart and verify that the selected skins are present. Proceed to checkout and verify that the selected skins are present.  **SQL/Implementation**: We use an INSERT statement.  INSERT INTO skin\_owned(user\_id, skin\_id, in\_cart)  VALUES(1, 1, true); |

|  |
| --- |
| **Card**: As a user, I can remove a skin from my cart.  **Conversation**: A user can remove skins from their cart before checkout.  **Conformity**: When browsing the shop, add several skins to your cart. The “add to cart” button should now say “added to cart” and have a X next to it. Click on the X to remove it from your cart and verify that it has been removed at checkout. Click on “View cart” and attempt to remove a skin by clicking the X next to it. Proceed to checkout and verify that all skins that were removed are not present.  **SQL/Implementation**: We use a DELETE statement.  DELETE FROM skin\_owned  WHERE user\_id = 1 AND skin\_id = 1; |

|  |
| --- |
| **Card**: As a user, I can view the skins I have purchased.  **Conversation**: A user can look at their collection of skins.  **Conformity**: Click the “My Collection” tab at the top of the screen to view the skins that you have purchased. Verify that all the skins you own are shown.  **SQL/Implementation**: We use a SELECT statement.  SELECT \*  FROM skin, skin\_owned USING (skin\_id)  WHERE user\_id = 1 AND in\_cart = false; |