## Purpose of the website

The purpose of the website was to allow users to have a personalised record of the games that they have played. A user would search for the game that they want to add to their list on the website. The website then remembers the games that they have stored, allowing the user to have a record of their game list.

### How to use the website

#### Users:

1. The user would need to login into their account.
2. The user can search for the game that they want to add to their list in the search bar.
3. The user then clicks on the "Add to list" button.
4. The game can be found on the user's feed page.
5. If the user wants to remove a game from the list, they can click on the "Remove from list" button on their feed page.

#### Admins:

1. The admin can login into their account on the login page
2. The admin then goes to the page https://sc19hhl.pythonanywhere.com/admin/ to access the admin interface
3. The admin can check all users, create / remove games, developers, publishers, genres, models, platforms and images in the database

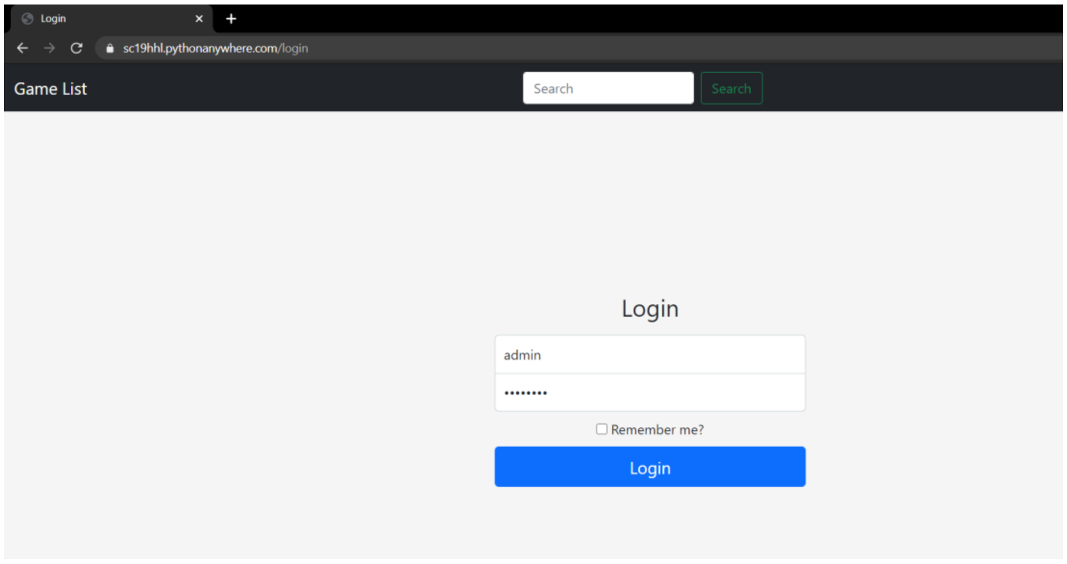
## Link to the website

The website can be found on https://sc19hhl.pythonanywhere.com/. The website does not require any authentication to access.

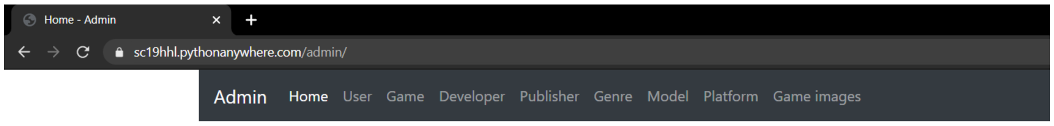
### Steps to access the admin page

The admin page can only be accessed with the credentials:

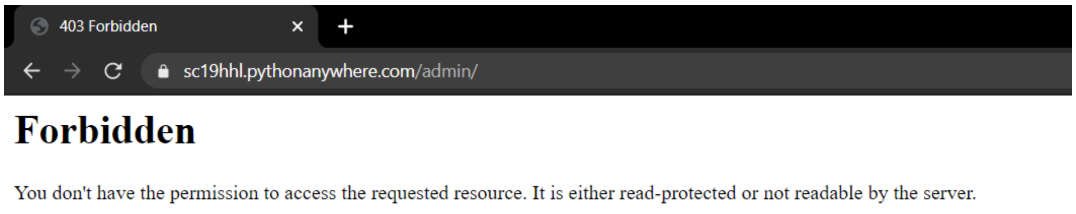
Username: admin
Password: password



Step 1: Enter the credentials of the admin user on https://sc19hhl.pythonanywhere.com/login



Step 2: Go to https://sc19hhl.pythonanywhere.com/admin/ to access the admin interface.



Forbidden

You don't have the permission to access the requested resource. It is either read-protected or not readable by the server.

Any unauthorized access would be returned an Error:403 forbidden page.

## Analysis of web application

### Web forms

The web application uses Flask-WTF and WTForms to handle form rendering and form submissions. The web application uses both client-side and server-side validation to ensure that the data submitted by a user is correct and can be processed without any errors.

### Validations

Client-side validation includes:

- Specifying the field type to an appropriate data type
- Specifying if the form field is a required filled
- Specifying the minimum length of the data

Server-side validation includes:

- Validating fields to be the correct data type
- Validating length of input
- Validating if data is inputted
- Validating if the fields password and confirm password matches

Suppose if a user has submitted an invalid form, the web application will display error messages using the Flask message flashing. This will provide the relevant feedback to the user.

## Database

The database of the web application mainly consists of holding the information of a user and the information of the games. Flask-SQLAlchemy was used to create and manage the database.

### Validations

Validations are used to ensure that the data in the database is kept consistent. Many constraints are created to minimize the errors.

Constraints include:

- Data types are specified for each field
- Setting important fields to be not nullable
- Restricting specific fields to be unique only
- Restricting data to specific lengths

### Tables

| User | |
|---|---|
| Column name | Data type |
| User ID (PK, NN) | Integer |
| Email (U, NN) | String (35) |
| Username (U, NN) | String (25) |
| Password (NN) | String |
| Admin (NN) | Boolean |

| Game | |
|---|---|
| Column | Data type |
| Game ID (PK, NN) | Integer |
| Title (NN) | String |
| Description | String |
| Release date (NN) | Date |
| Image URL | String |

| Developer | |
|---|---|
| Column | Data type |
| Developer ID (PK, NN) | Integer |
| Name (U, NN) | String |

| Publisher | |
|---|---|
| Column | Data type |
| Publisher ID (PK, NN) | Integer |
| Name (U, NN) | String |

| Genre | |
|---|---|
| Column | Data type |
| Genre ID (PK, NN) | Integer |
| Genre type (U, NN) | String |

| Model | |
|---|---|
| Column | Data type |
| Model ID (PK, NN) | Integer |
| Model type (U, NN) | String |

| Platform | |
|---|---|
| Column | Data type |
| Platform ID (PK) | Integer |
| Platform name (U, NN) | String |

*PK – Primary key* *FK – Foreign key* *U – Unique* *NN – Not nullable*

### Relationships

There are multiple many-to-many relationships implemented in the database. The primary many-to-many relationship would be the user's game list. A user would have a list of many games. A many-to-many relationship is then created.

Many-to-many relationships in the web application include:

- The game and the developers

- The game and the publishers
- The game and the genre
- The game and the model
- The game and the platform
- The user and the game

| Game Developer | |
|---|---|
| Column | Data type |
| Game ID (FK) | Integer |
| Developer ID (FK) | Integer |

| Game Publisher | |
|---|---|
| Column | Data type |
| Game ID (FK) | Integer |
| Publisher ID (FK) | Integer |

| Game Genre | |
|---|---|
| Column | Data type |
| Game ID (FK) | Integer |
| Genre ID (FK) | Integer |

| Game Model | |
|---|---|
| Column | Data type |
| Game ID (FK) | Integer |
| Model ID (FK) | Integer |

| Game Platform | |
|---|---|
| Column | Data type |
| Game ID (FK) | Integer |
| Platform ID (FK) | Integer |

| User Game | |
|---|---|
| Column | Data type |
| User ID (FK) | Integer |
| Game ID (FK) | Integer |

*PK – Primary key* *FK – Foreign key* *U – Unique* *NN – Not nullable*

## Incorrect input feedback
Similarly, Flask message flashing is used to provide feedback to the user if the data was added to the database successfully or if an error has occurred.

## Authentication and sessions
The web application uses Flask-Login to handle authentication and sessions. A login manager was used from Flask-Login to ensure that users have a session when they use the web application. This allows users to login and stay logged in until the session is destroyed or if they choose to logout. The authentication of the user is done by querying the user with the same username. The application then checks the password hash of the password entered by the user with the hashed password stored in the database.

Certain routes are only able to be accessed once the user is logged in. A user is unable to view their feed if they are not logged in. Any attempt to access a restricted page would be redirected to the login page. Once the user has successfully logged in, the user would be able to see their personalised feed of their game list. The user is also able to update their password in the settings route if they are logged in. Since sessions uses cookies, a privacy warning is displayed on the index page of the website.

## Appropriate styling
The website uses Bootstrap 5 for styling. Bootstrap makes the website to be responsive and look natural both on a desktop and a tablet. The web application uses templating to ensure that certain elements of the webpages are kept consistent. Bootstrap colour schemes are used to convey messages to the user. For example, the warning colour (yellow) from Bootstrap was used as the colour when user is flashed an error message. Red and green was the button colours for remove and add to the user's game list.

## Unit testing
The web application uses a set of unit test to test the functionality and features of the application. Testing for database queries are also included within the tests.

The unit tests implemented includes:

- Testing sign up functionality
- Testing login functionality
- Testing feed page
- Testing search functionality and page
- Testing add to game list feature
- Testing remove from game list feature
- Testing change password functionality
- Testing logout functionality

### Additional features

Several features are implemented in the web application. Bootstrap and jQuery were used to implement a responsive user interface for the users. For example, Bootstrap was used to render a remove button that removes a game from the user's game list. In the event where a user clicks on the remove button, jQuery will animate the game fading out of the user's game list.

Additionally, AJAX was used to update the user's game list and updating the website that the user is currently viewing. For example, in an event where a user clicks on the add button to add to their game list, the client would send a request to the server. The server can then update the database and sends a response back to the client. This response will update the display of the user, informing the user that the game is added to their game list.

## Evaluation

When a user interacts with our web application, they would want to achieve their objective safely, effectively and efficiently. One of the features for the web application is to add and remove games to and from their list. From a user's point of view, we would want a simple function to achieve this task. The design choice to facilitate this feature was to have a button that allows users to add a game to their list or remove a game from their list.

There are many games in the games database. Suppose that a user queries for a game which returns 10 results. The game is then displayed on screen without any structure or formatting. This would confuse the users, leaving them with a bad impression. It was then decided that the application would provide a box for each game. The box will contain the game's information and a thumbnail to identify the game easily. This ensures that the games are ordered and organized which will result in a better user experience.

Some elements of the design were kept consistent throughout all the webpages to allow users to easily complete their task. For example, suppose that a user wants to return to their feed after searching for a game. If the navigation bar was not there or was moved to the bottom of the screen, the user would be confused and would require more time to find the navigation bar to return to their feed. In contrast, if we kept the navigation bar on the top of the screen for every page, the user could navigate through the webpage easily as they know that the functionality of the navigation bar would be the same at every webpage.

We also would want administrators to have access and manage the database. Hence only specific users with the attribute of admin would be apple to access an admin interface. This admin interface allows the administrators to manage the user's database and the game's database.

## Security

The forms submitted across the internet may be vulnerable to Cross Site Request Forgery (CSRF). A solution to this would be to use a CSRF token to ensure the security of the data when submitting forms.

Another potential security risk would be SQL injection attack. A SQL injection attack may potentially leak private information of users or destroy the database of the web application. In our case, the web application uses SQLAlchemy to query from the database. SQLAlchemy uses prepared statements when querying a database. This prevents any potential SQL injection attacks from happening in our web application.

If passwords were stored directly onto the database and there is a security breach in our database, the hackers would have access to the exact passwords of the users. Therefore, this web application does not store passwords directly onto the database but instead it stores hashed passwords in the database. Hence, if passwords are leaked from the web application, the hackers are only able to view the hashed passwords of the users which is a lot less valuable compared to the real passwords.

The web application only uses single factor authentication (username and password). Potential issues such as if passwords are leaked, a hacker would have direct access to the victim's account. Future developments of the web application may include adding two-factor authentication to increase the security of the application.

Since administrator have access to the database, if unauthorised personnel have access to the administrator page, they can view all the sensitive information of the users. Therefore, the application uses Flask-Admin model view to limit the display of certain tables or columns to prevent sensitive information such as the hashed passwords from being viewed.