WhiskedIn Progress Report

Sprint 1:

For the first sprint the following user stories were selected:

- As an unregistered user I can visit the app main page and learn about what the app can do for me and create a new account.
- As a registered user I can visit the app main page and login with my username and password.
- As a registered user I can sequentially scan through my list of collectible items forward and backward by clicking on right and left buttons respectively.

To complete the first story, the REST api had to receive credentials and with those credentials create a user account. So first a database was created and using an SQLAlchemy (ORM), the following User class was created

Class User:

username
hashed_password

get_user(username)
create_user(username, password)

For this sprint no whiskies were involved but for database purposes a Whisky class was also created, but not tested since there was no functionality involved with it other than to create the table.

With this class created the endpoints for the REST api could be implemented. Two endpoints were made:

- /login
- /register

Both of these endpoints only accept POST requests, and both return the same results which is JSON containing the access token for the user to continue using the application. If a username exists when registering then a 400 response is returned to the user. If the login credentials are incorrect, a 400 response is returned to the user.

For the third story there was no backend implementation, since the list was going to be hardcoded for this sprint.

To test the REST api, first unit tests were made to test the methods inside the user class. The first method get_user() received a username (string) as an argument and would return a User instance. The create_user() would receive a username and a password (both strings) and would return a new User instance. So to test get_user() there were only two possibilities, either the username exists or it doesn't. Two unit tests were made to test this. For create_user() the only test made was for a successful creation of a user. For all of these tests, a new test database is created, they are all independent from each other.

Tests were also made for the endpoints themselves. Each test would instantiate a new test client for the REST api and a test database. Requests were made in the tests themselves and sent to the test client. These were the tests made:

- Test_register - test a successful register
- Test_register_username_exists - tests that a user cannot create an account with an existing username. Makes sure that the correct response is returned
- Test_login_no_user - tests that a user cannot log in with an account that does not exist
- Test_login_user_exists - tests a successful login
- Test_login_user_wrong_password - tests that a user with a correct username cannot login with an incorrect password

Front end is implemented using react. When the site is first visited a user lands on the main page where they can register or log in.

Home page displays a “whiskey card” which contains its properties. To the right and left of the “whiskey card” there is a next and back button that when pushed change between the next and the prev card respectively.

The whiskey card displays the whiskey’s name, the company that makes it, the type, the age, its origin, the flavor, a description and a rating. Below the whiskey card there is also a share button. This button is not functional yet, same as the logout button.

All functions that handle the HomePage logic are in the HomePage class. The only two functions at the moment are handleNext and handleBack which change the current index state variable to the next or previous whiskey card, which itself changes the whiskey displayed on screen to the appropriate one.

Two unit tests were created, one for handleNext and another for handleBack. The first one verifies that the current index is increased by one and the other that it is decreased by one.

System tests

After all functionality and tests were made for the frontend and backend, then system tests were made using selenium to test a deployment of the end product. Two tests were made:

- Test_register_login - tests that a user can register an account and login, also tests that adequate error messages appear when unsuccessful login or register occur. Verify that all elements are visible and there.
- Test_whisk_list - tests that after a successful login/register, the user can see the whisky cards and that the user can click the buttons and that they behave the way they are supposed to behave.

Sprint 2:

For the second sprint the following user stories were selected:

- As a registered user I can always look at and edit the information associated with the collectible item that I am currently viewing.
- As a registered user I can search for collectible items by specifying terms. The app will move to the first item beyond the current item that contains these terms in any of its information fields.

The REST api now needs an endpoint for a user to interact with their whiskies. '/whiskies' is now born. This endpoint accepts GET, POST, PUT requests. If a user sends a GET request then their whiskies and only theirs will be returned. A user can create a new whiskey by sending required fields in the POST request. Any whisky that a user creates can only be seen by them. Users can edit only their whiskies by sending a PUT request with the fields from the whisky including the wid. A user can only edit their own whiskies. A user can also search for specific terms in their own whiskies by sending a GET request with the query parameter 's={search}'.

To implement these features in the REST api now the class that was setup in the past sprint had to be elaborated on. The following methods were added:

- create_whisky() - receives the required arguments for creating a whisky, returns a whisky instance

- update_whisky() - receives the fields that are to be updated, returns the updated whisky instance

- get_whisky() - receives the wid, returns the whisky instance

- get_whiskies() - receives the search term, returns a list of found whiskies

- build_dict() - returns a dictionary of the whisky

For the user class a build_dict() method was also created.

So now that we have all this, lets test. The following unit/integration tests were made:

- Test_create_whisky - tests successful creation of a whisky using create_whisky()
- Test_create_whisky_empty_field - tests that if a required field is not sent to create_whisky() method, that it fails
- Test_get_whisky - tests a successful search of a whisky
- Test_get_whisky_not_existing - tests that if get_whisky tries to find a whisky that does not exist, None is returned
- Test_update_whisky - tests a successful update of an existing whisky using update_whisky()
- Test_whisky_update_only_owner - Only the creator of a whisky can update that whisky
- Test_get_whiskies - tests a successful search for whiskies using get_whiskies()
- Test_get_whiskies_endpoint_no_token - tests that access is not allowed without an access token to the '/whiskies' endpoint
- Test_get_whiskies_endpoint_successful - tests that the correct whiskies are returned for a successful request to the '/whiskies' endpoint
- Test_get_whiskies_user_only - tests that a user can only see his whiskies
- Test_post_whisky_endpoint - tests a successful creation of a whisky using the endpoint and that no other user sees that whisky
- Test_sending_str_age_rating - tests that if a user sends a string a 400 response is returned with the appropriate error message
- Test_put_whisky_endpoint - tests a successful update of a whisky using the endpoint
- Test_search_endpoint - tests a successful search for a whisky using the endpoint

Front end

In the front end, Material-ui was used to implement react elements and components in the HomePage. When first entering the HomePage log in, the application will get all whiskeys the user has created before and show that list and display the first one as a whiskey card. If no whiskeys have been created the list will be empty and in place of the whiskey card a string indicating that the user has no inputs will appear. All properties were added to whiskey card for display. A create floating action button was created to allow creation of new whiskeys. This button when clicked calls the post method to post the new whiskey onto the user whiskey list. A list of user whiskeys is displayed below. When clicked any of the items in the list will change the currently displayed whiskey card to the clicked one. Every list item also has an edit button. Pressing either the create or any of the edit buttons will open the appropriate dialog to either create a new whiskey or edit the fields of an existing one.

Above the whiskey list and to the side of the create button, there is a text field for search input. As you write letters a get request is sent to the api to search for any whiskeys that contain the search input text in any of their fields.

Unit tests for this implementations are pending.

System tests

After all functionality is done, a build is deployed. This sprint's result is deployed at http://whiskeding2.herokuapp.com. The following tests were done to simulate user interaction:

- Test_empty_whisk_list - tests that a user with no whiskies is shown a message and that there are in fact no whiskies
- test_visible _whiskies - tests that a user with whiskies is shown those whiskies when they log in
- Test_creating_whiskies - tests that a user can create a whisky and that when they log in the whisky is there
- Test_creating_whisky_fail - tests that if an error occurs while creating a whisky, a message is shown to the user
- Test_editing_whisky - Tests that a user can edit a whisky and that when they log in the edited whisky is there with the correct value
- Test_search_whisky - Tests that a user can search for a whisky and that the correct whisky is shown

The next system tests test the front end independently from the back end using a mock api that returns a generated list of whiskeys.

- Check all login page elements are visible
- Login button and signup button redirect when success
- All homepage elements are visible
- Add button clicked, check create form elements - clicks create button and checks that all elements of the dialog are displayed
- List item clicked and card is changed - checks that when an item on the list of whiskeys is clicked it is displayed on the whiskey card.
- Edit button clicked, cheks edit form appears, and clicks create button on form - checks that every edit button in the list by clicking them and ensuring that the edit dialog is displayed.
- Can click next and back - cycles through the list of whiskeys by clicking the next and back buttons. It also ensures that the whiskey is changed when appropriate.

These tests are for the common case. No edge cases are tested in this sprint.