ScrumBuddies – [Sustainability Game]

Overview

This technical document is set to lay out all relevant information about the source code for [Sustainability Game]. All source code has been included in the ZIP file. The instructions on deploying the deploying the software will also be provided in this document.

Source Code Structure

Below is the structure of [Sustainability Game], with all of the most significant files and folders displayed:

/Sustainability-Game

/members

/\_\_pycache\_\_

/migrations

/templates

delete.html

login.html

privacy\_policy.html

profile.html

register.html

update\_password.html

update\_user.html

\_\_init\_\_.py

admin.py

apps.py

forms.py

models.py

tests.py

urls.py

views.py

/ScrumBuddies

/\_\_pycache\_\_

\_\_init\_\_.py

asgi.py

settings.py

urls.py

wsgi.py

/static

/css

/img

/Sustain

/\_\_pycache\_\_

/migrations

/templates

base.html

index.html

learning.html

navbar.html

\_\_init\_\_.py

admin.py

apps.py

models.py

tests.py

urls.py

views.py

/tasks

/\_\_pycache\_\_

/migrations

/templates

tasks.html

\_\_init\_\_.py

admin.py

apps.py

forms.py

models.py

tests.py

urls.py

views.py

.gitignore

db.sqlite3

manage.py

README.md

The Django backend system is compartmentalised into various aspects of the web app. The Sustain Folder is the default folder, in which all the basic templates for the rest of the web app to use are stored. This includes the HTML files that hold the basic layout of the web page and the navigation bar. The members folder holds all the information about the users. This includes the account creation system, the account login system, the account deletion system, etc. The folder also includes an extension to the Users class – which is provided by Django and cannot be directly changed. The tasks folder includes all aspects of the challenges that the users will have to complete. This includes a class identifying the type of tasks that must interact with, as well as a class which holds all the information about one specific task.

Deployment Instructions

Prerequisites:

* Django **must** be installed (<https://docs.djangoproject.com/en/4.2/topics/install/>)
* Pillow **must** be installed (<https://pypi.org/project/Pillow/>)
* Responses **must** be installed (<https://pypi.org/project/responses/>)

The web app runs using the Django backend system. This means that it can be run on a local machine by running the command ‘python manage.py runserver’ on the command line in the ‘Sustainability-Game’ directory. The user can then type in the URL 127:0:0:1:8000, which will take them to the main website.

Authorship Identification

Authors:

* Charlie Silver
* Luke Hales
* Achille Fossiez-Revenu
* Gregory Goursaud
* William Smith
* Liam Buchanan

Coding Conventions

Python:

* 4 spaces per indentation.
* No whitespace in expressions and statements.
* Different types of imports are on new lines.
* Functions use underscores and classes follow the CamelCase convention.
* Comments explaining functionality of code.

HTML:

* Tags are indented when one element is inside another and indented by 4 spaces.
* Double quotes for attribute values.
* Attributes are in a consistent order.
* Comments explaining functionality of code.

Testing Strategy

**Members**:

* **testUserCreation**:
  + **Purpose:** This test is designed to test the user creation functionality.
  + **Steps:** 
    - The test creates an instance of the website that can be interacted with in a testing environment.
    - The initial number of users is stored in a variable so that it can be compared with the number of accounts after the test account has been attempted to be created.
    - Test data is created to be used as the test account’s details. The details are checked using the RegisterUserForm – a form for adding users. The form is validated by using ‘assertTrue’.
    - The test attempts to create the test account by using the test data and the view ‘register\_user’.
    - The number of accounts is tchecked against the original number of accounts by using ‘assertEquals’ on the current number of accounts and the initial number of accounts plus one. Since the test account is the last account in the table, it can be easily accessed, stored in a variable and used for further testing. The name of the user is checked against the data that was initially inputted to again verify that the account is in the database using ‘assertEquals’. The status code is also checked against what it should be by using ‘assertEquals’. The final part of the test is to ensure that the user has been redirected to the correct page by using ‘assertRedirects’.
* **testUserLogin**: This test is designed to test the user login functionality. The test begins by creating an instance of the website that can be interacted with in a testing environment. A test user is then created by using test data. This data is replicated in another variable and is used in an attempt to log the user into their account using the view ‘login\_user’. ‘assertEquals’ is then used to compare the status code of the response and what the status code should be.
* **testUserLogout**: This test is designed to test the user creation functionality. The test begins by creating an instance of the website that can be interacted with in a testing environment. A test user is created and logged into the system. After this, the test attempts to log the test user out of their account by using the view ‘logout\_user’. The status of the response is then checked using ‘assertEquals’ to ensure that the code is what it should be.
* **testUserDelete**: This test is designed to test the user deletion functionality. The test begins by creating an instance of the website that can be interacted with in a testing environment. A test user is created and the number of users in the system is counted and stored in a variable. This variable will be compared with the number of users once the test has attempted to delete the test user from the system. The test account is then logged into in order to test the ‘delete\_user’ view, as it requires the user that is currently logged in. Once the view is used, ‘assertEquals’ is used to compare the current number of users in the system with the number of users before the view was called.
* **testUserUpdate**: This test is designed to test the user/profile updating functionality. The test begins by creating an instance of the website that can be interacted with in a testing environment. A test user is created and stored in a variable so that the changes in its attributes can be assessed and analysed. Since the user class has a one-to-one relationship with the profile class, the profile of the test user is also stored in a variable. The test user is then logged in so that the forms can be processed correctly. Test data is created that will be used to update the attributes of the test user and their associated profile. All of the attributes that will be updated are checked using ‘assertEquals’ to make sure that they are empty or of type ‘None’. The data is then passed through the relevant forms with the correct instances. The forms are all validated using ‘assertTrue’ after the data has been passed through. The view ‘update\_user’ is called to attempt to update the attributes of the test user and their profile. Each time the view is called, the status code is checked using ‘assertEquals’. The attributes that have been updated are also checked using ‘assertEquals’ to ensure that the attributes are what was expected.