CSC 510 SOFTWARE ENGINEERING PROJECT REPORT

CALORIE APP - BURNOUT

SOFTWARE CODE DOCUMENTATION

TEAM MEMBERS

Setu Kumar Basak (sbasak4)

Conor Thomason (cjthom24)

Keertana Vellanki (kvellan)

Muntasir Hoq (mhoq)

Matthew Sohacki (mjsohack)

INTRODUCTION

BurnOut is an easy-to-use application that keeps track of a user's daily calories - gained and burnt. It can help the user in achieving objectives such as weight loss or increase. Users can make changes to their profiles by entering their height, weight, goal, and targeted weight. Users can enroll in a variety of fitness programs, such as yoga classes and workout sessions. They can also connect with their friends by sending a friend request and sharing their progress.

The user can input the food they have eaten and BurnOut displays a record of the calories. The application shows a record of calories consumed and the calories burnt out day-wise in the History tab.

This report offers users a broad overview of the project, allowing individuals to understand it as open-source software and add improvements. The report also helps developers in comprehending the code and serves as a starting point for the project.

POTENTIAL USERS

1. Fitness enthusiasts
2. People with dietary restrictions
3. People who are recently getting into the world of fitness
4. Athletes

ARCHITECTURE

The following technologies were used to complete the development, and it is recommended that the next group of developers who take on this project have these technologies installed and running before proceeding:

1. FrontEnd:
    a. HTML 5
    b. CSS
2. Backend:
    a. Python, Flask
    b. Unittest (Test Framework)
3. Database:
    a. MongoDB

CODE FUNCTIONALITIES

application.py: This is the file that generates the Flask server - upon calling “flask run” within this directory, this is the file that will be called.

1. def home(): HttpGet endpoint for displaying the homepage of the website.
    a. Route: /home or /
    b. Input: The function takes session as the input
    c. Output: Displays the dashboard or login page based on the use session.
2. def login(): HttpPost endpoint for showing the login form.
    a. Route: /login
    b. Input: Email, Password, Login Type
    c. Output: Does authentication and redirects to dashboard
3. def logout(): HttpGet endpoint for log out of the application.
    a. Route: /logout
    b. Output: Returns success after clearing session.
4. def register(): HttpPost endpoint for registering a user.
    a. Route: /register

   b. Input: Username, Email, Password, Confirm Password
   c. Output: Updates in the database and redirects to home login page
5. def calories(): HttpPost endpoint for adding the calorie information.
   a. Route: /calories
   b. Input: Email, date, food, burnout
   c. Output: Updates in database and redirects to home page
6. def user_profile(): HttpPost endpoint for adding the user profile information.
   a. Route: /user_profile
   b. Input: Email, height, weight, goal, Target weight
   c. Output: Updates in database and redirects to home page
7. def user_profile(): HttpPost endpoint for adding the user profile information.
   a. Route: /user_profile
   b. Input: Email, height, weight, goal, target weight
   c. Output: Updates user profile information and redirects to home page
8. def history(): HttpGet endpoint for retrieving the history information.
   a. Route: /history
   b. Input: Email, date
   c. Output: Retrieves the history information and renders the history template
9. def ajaxhistory(): HttpPost endpoint for retrieving the history using ajax call.
   a. Route: /ajaxhistory
   b. Input: Email, date
   c. Output: Returns the history json.
10. def friends(): HttpGet endpoint for showing the friends information.
   a. Route: /friends
   b. Input: Email
   c. Output: Returns the My friends, Pending Approvals, Sent Requests and Add new friends
11. def ajaxsendrequest(): HttpPost endpoint for updating friend request information.
   a. Route: /ajaxsendrequest
   b. Input: Email, receiver
   c. Output: Updates the receiver info into the database and returns TRUE if successful and FALSE otherwise.
12. def ajaxcancelrequest(): HttpPost endpoint for cancelling friend requests.
   a. Route: /ajaxcancelrequest

   b. Input: Email, receiver
   c. Output: Cancels friend request and returns TRUE if successful and FALSE otherwise.

13. def ajaxapproverequest(): HttpPost endpoint for approving friend requests.
    a. Route: /ajaxapproverequest
    b. Input: Email, receiver
    c. Output: Approves friend request and returns TRUE if successful and FALSE otherwise.
14. def dashboard(): HttpGet endpoint for displaying the dashboard.
    a. Route: /dashboard
    b. Output: Redirects to dashboard page and displays the list of activities.
15. def yoga(): HttpPost endpoint for enrolling in yoga.
    a. Route: /yoga
    b. Input: Email
    c. Output: Enrolls in yoga and redirects to new dashboard.
16. def swim(): HttpPost endpoint for enrolling in swim.
    a. Route: /swim
    b. Input: Email
    c. Output: Enrolls in swim and redirects to new dashboard.
17. def gym(): HttpPost endpoint for enrolling in gym.
    a. Route: /gym
    b. Input: Email
    c. Output: Enrolls in gym and redirects to new dashboard.
18. def walk(): HttpPost endpoint for enrolling in walk.
    a. Route: /walk
    b. Input: Email
    c. Output: Enrolls in walk and redirects to new dashboard.
19. def dance(): HttpPost endpoint for enrolling in dance.
    a. Route: /dance
    b. Input: Email
    c. Output: Enrolls in dance and redirects to new dashboard.

TESTING

For functional testing of our Flask Application and Database, we have a separate file named test_app.py in the project directory and included unittest framework.

Test 1 (test_logout_status_check): In this test, we tested the GET method of our Flask client and tested whether the application logs out or not, by checking the status code of the Flask client.

Test 2 (test_logout_return_check_success): In this test, we tested the GET method of our Flask client and tested whether the application logs out or not, by checking the string output of the Flask client.

Test 3 (test_logout_return_check_error): In this test, we tested the GET method of our Flask client and tested whether the application logs out by error, by checking the string output of the Flask client.

Test 4 (test_home): In this test, we tested the GET method of our Flask client and tested whether the application correctly follows the home route, by checking the status code of the Flask client.

Test 5 (test_home_session_email): In this test, we tested the GET method of our Flask client and tested whether the application correctly follows the home route with a valid session email, by checking the status code of the Flask client.

Test 6 (test_login_redirect_to_dashboard): In this test, we tested the POST method of our Flask client and tested whether the application redirects to the dashboard route with a valid login request, by checking the status code of the Flask client.

Test 7 (test_login_redirect_to_home): In this test, we tested the GET method of our Flask client and tested whether the application redirects to the home route with a valid session email, by checking the status code of the Flask client.

Test 8 (test_login_unsuccessful): In this test, we tested the POST method of our Flask client and tested whether the application works properly with an incorrect login request or not, by checking the status code of the Flask client.

Test 9 (test_register_render_template): In this test, we tested the GET method of our Flask client and tested whether the application renders register form correctly or not, by checking the status code of the Flask client.

Test 10 (test_register_success): In this test, we tested the POST method of our Flask client and tested whether the application registers successfully or not, by checking the status code of the Flask client.

Test 11 (test_register_redirect_to_home): In this test, we tested the POST method of our Flask client and tested whether the application redirects to the home route or not, by checking the status code of the Flask client.

Test 12 (test_calories_update): In this test, we tested the POST method of our Flask client and tested whether the application updates calories or not, by checking the status code of the Flask client.

Test 13 (test_calories_insert): In this test, we tested the POST method of our Flask client and tested whether the application inserts calories or not, by checking the status code of the Flask client.

Test 14 (test_calories_non): In this test, we tested the POST method of our Flask client and tested whether the application without valid session email redirecting to home route or not, by checking the status code of the Flask client.

Test 15 (test_user_profile): In this test, we tested the GET method of our Flask client and tested whether the application exits the user profile route without a valid session or not, by checking the status code of the Flask client.

Test 16 (test_user_profile_update): In this test, we tested the POST method of our Flask client and tested whether the application updates the user profile or not, by checking the status code of the Flask client.

Test 17 (test_user_profile_insert): In this test, we tested the POST method of our Flask client and tested whether the application inserts the user profile information or not, by checking the status code of the Flask client.

Test 18 (test_dashboard): In this test, we tested the GET method of our Flask client and tested whether the application exits the dashboard route without a valid session or not, by checking the status code of the Flask client.

Test 19 (test_friends): In this test, we tested the GET method of our Flask client and tested whether the application gets a friend list or not, by checking the status code of the Flask client.

Test 20 (test_alive): In this test, we tested the GET method of our Flask client and tested whether the application is alive or not, by checking the status code of the Flask client.

Test 21 (test_history): In this test, we tested the GET method of our Flask client and tested whether the application exits the history route without a valid session or not, by checking the status code of the Flask client.

Test 22 (test_history_session_email): In this test, we tested the GET method of our Flask client and tested whether the application redirects to the history route with a valid session or not, by checking the status code of the Flask client.

Test 23 (test_ajaxhistory_session_found): In this test, we tested the POST method of our Flask client and tested whether the application fetches information with a valid session or not, by checking the status code of the Flask client.

Test 24 (test_ajaxhistory_session_not_found): In this test, we tested the POST method of our Flask client and tested whether the application fetches information without a valid session or not, by checking the status code of the Flask client.

Test 25 (test_yoga_redirect_to_dashboard): In this test, we tested the POST method of our Flask client and tested whether the application redirects to the dashboard route without a valid session or not, by checking the status code of the Flask client.

Test 26 (test_enroll_success): In this test, we tested the POST method of our Flask client and tested whether the application enrolls yoga successfully with a valid session or not, by checking the status code of the Flask client.

Test 27 (test_swim_redirect_to_dashboard): In this test, we tested the POST method of our Flask client and tested whether the application redirects to the dashboard route without a valid session or not, by checking the status code of the Flask client.

Test 28 (test_swim_enroll_success): In this test, we tested the POST method of our Flask client and tested whether the application enrolls swim successfully with a valid session or not, by checking the status code of the Flask client.

Test 29 (test_gym_redirect_to_dashboard): In this test, we tested the POST method of our Flask client and tested whether the application redirects to the dashboard route without a valid session or not, by checking the status code of the Flask client.

Test 30 (test_gym_enroll_success): In this test, we tested the POST method of our Flask client and tested whether the application enrolls gym successfully with a valid session or not, by checking the status code of the Flask client.

Test 31 (test_walk_redirect_to_dashboard): In this test, we tested the POST method of our Flask client and tested whether the application redirects to the dashboard route without a valid session or not, by checking the status code of the Flask client.

Test 32 (test_walk_enroll_success): In this test, we tested the POST method of our Flask client and tested whether the application enrolls walk successfully with a valid session or not, by checking the status code of the Flask client.

Test 33 (test_dance_redirect_to_dashboard): In this test, we tested the POST method of our Flask client and tested whether the application redirects to the dashboard route without a valid session or not, by checking the status code of the Flask client.

Test 34 (test_dance_enroll_success): In this test, we tested the POST method of our Flask client and tested whether the application enrolls dance successfully with a valid session or not, by checking the status code of the Flask client.

Test 35 (test_ajaxapproverequest_success): In this test, we tested the POST method of our Flask client and tested whether the application updates friend request information with a valid session or not, by checking the status code of the Flask client.

Test 36 (test_ajaxapproverequest_failure): In this test, we tested the POST method of our Flask client and tested whether the application updates friend request information without a valid session or not, by checking the status code of the Flask client.

Test 37 (test_hrx_redirect_to_dashboard): In this test, we tested the POST method of our Flask client and tested whether the application redirects to the dashboard route without a valid session or not, by checking the status code of the Flask client.

Test 38 (test_hrx_enroll_success): In this test, we tested the POST method of our Flask client and tested whether the application enrolls hrx successfully with a valid session or not, by checking the status code of the Flask client.

Test 39 (test_ajaxcancelrequest_success): In this test, we tested the POST method of our Flask client and tested whether the application updates friend request information with a valid session or not, by checking the status code of the Flask client.

Test 40 (test_ajaxcancelrequest_failure): In this test, we tested the POST method of our Flask client and tested whether the application updates friend request information without a valid session or not, by checking the status code of the Flask client.

Test 41 (test_ajaxsendrequest_success): In this test, we tested the POST method of our

Flask client and tested whether the application updates friend request information with a valid session or not, by checking the status code of the Flask client.

Test 42 (test_ajaxsendrequest_failure): In this test, we tested the POST method of our Flask client and tested whether the application updates friend request information without a valid session or not, by checking the status code of the Flask client.

FUTURE SCOPE

1. Predicting workout plans for users based on their history and fitness reports.
2. Create a mobile application for the web version of the application.
3. Make the website view port adaptable - the website should look good on phones, tablets, and computers.
4. Chat functionality for friends
5. Share workout plans with friends
6. Creating an Activities dashboard based on user enrollment
7. Track user progress for each activity he/she enrolled for.
8. Deploy the application to the production environment.