

Testing Documentation for Campus Care

Introduction

This testing documentation provides an overview of the unit tests written for Campus Care. These unit tests are designed to verify the functionality and behaviour of various components of the software.

Test Environment Setup

The tests are written using the Python unittest framework. Additionally, the unittest.mock module is utilised for mocking external dependencies and controlling their behaviour during testing.

Test Cases

The following test cases ensure various functionalities within the software application are properly validated. Each test case focuses on specific behaviours or scenarios:

- `test_increment_current_question_index`: This test verifies that the current question index is incremented correctly within the application. It initializes the application and ensures that the index moves to the next question as expected.
- `test_question_index_resets`: This test checks that the question index resets correctly under specified conditions. It ensures the application handles the reset scenario by setting the index to a specific point and verifying its behaviour.
- `test_screen_transitions`: This test validates the screen transitions within the application, ensuring that the user can navigate between different screens (e.g., from login to home) seamlessly.
- `test_daily_quiz_completion_check`: This test verifies the daily quiz completion function, checking if the function correctly determines whether the quiz has been completed for the day by a specific user.
- `test_handle_empty_user_preferences`: This test handles the scenario where user preferences are empty, ensuring the application can manage and return appropriate results when no preferences are set for a user.
- `test_no_data_returned_from_db_queries`: This test verifies the application's behaviour when no data is returned from database queries, ensuring the application can handle empty datasets without errors.
- `test_handle_non_existent_user_ids`: This test checks the handling of non-existent user IDs. It ensures the application can manage scenarios where user IDs do not exist in the database.
- `test_no_selected_activities_for_quiz`: This test verifies the behavior when no activities are selected for the quiz, ensuring the application can handle and respond appropriately to this scenario.
- `test_popup_display_functionality`: This test verifies the functionality of popup displays within the application, ensuring that popups are displayed correctly when triggered.

Test Classes for Specific Screens

TestDailyQuizScreen:

Git Repository: https://github.com/newcastleuniversity-computing/CSC2033_Team37_23_24.git

- `test_on_enter_quiz_completion_check`: This test ensures that the quiz completion check is performed correctly when the user enters the daily quiz screen.
- `test_update_content_populates_correctly`: This test verifies that the quiz content updates correctly, populating the screen with the appropriate questions and answers.
- `test_handle_corrupted_quiz_data`: This test checks the application's behaviour when it encounters corrupted quiz data, ensuring it handles such scenarios gracefully.

TestInitialOptionsScreen:

- `Test_fetch_selected_activities_valid`: This test case verifies that the `fetch_selected_activities` method retrieves activities for a valid user from the database.
- `Test_fetch_selected_activities_no_activities`: This test case verifies that the `fetch_selected_activities` method handles the scenario where no activities are found.

TestInitialOptionsScreen:

- `test_toggle_adds_activity`: This test case verifies that when a button representing an activity is toggled (pressed), the activity is added to the list of selected activities in the OptionsScreen class.
- `test_toggle_activity_remove_activity`: This test case verifies that when a button representing an activity is toggled off (released), the activity is removed from the list of selected activities in the OptionsScreen class.

TestingAdmin:

- `test_popup_displays_with_correct_title`: This test case verifies that the SuccessPopup displays with the correct title and message when instantiated with a given message.
- `test_successfully_deletes_user`: This test case verifies that a user can be successfully deleted from the system using the DeleteAccountPopup class.

To run the tests, execute the test file using a Python interpreter. The `unittest.main()` function at the end of the file runs all test cases defined within the file.

User Management Tests:

We found it would be easier to conduct user tests for elements that the user would be interacting with. These tests are described below.

Test Case	Description	Expected Outcome	Pass/Fail
Registration			
Valid Registration	Provide all required fields correctly and ensure successful registration.	User is registered successfully.	Pass

Duplicate Username	Attempt to register with an existing username and verify the appropriate error message.	Error message: Username already in use.	Pass
Duplicate Email	Attempt to register with an existing email and verify the appropriate error message.	Error message: Email already in use.	Pass
Missing Fields	Submit registration with missing required fields and verify the appropriate error message.	Error message: Missing required fields.	Pass
Invalid Date of Birth Format	Register with an invalid date format for the date of birth field and verify error handling.	Error message: Invalid date format.	Pass
Login			
Successful Login	Provide correct username and password and ensure a valid access token is returned.	Valid access token is returned.	Pass
Incorrect Password	Attempt to log in with an incorrect password and verify the appropriate error message.	Error message: Password incorrect.	Pass
Nonexistent Username	Try to log in with a username that doesn't exist and verify the appropriate error message.	Error message: Username not found.	Pass
Change Email			
Successful Email Change	Provide a valid new email address and ensure it's updated successfully.	Email is updated successfully.	Pass
Invalid New Email	Attempt to change the email to an invalid format and verify the error message.	Error message: Invalid email format.	Pass
Unauthenticated Access	Try to change email without a valid access token and ensure it's rejected.	Error message: Unauthorized access.	Pass

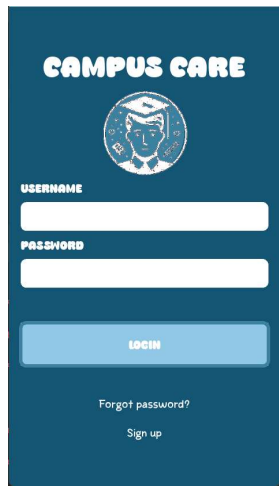

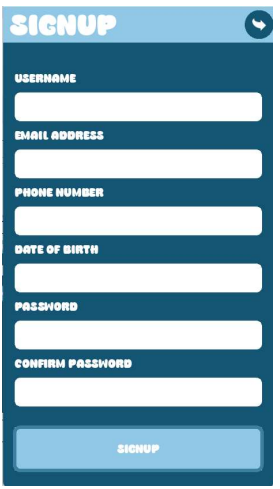
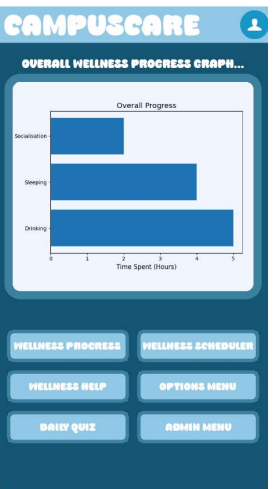
Change Password			
Successful Password Change	Provide correct current and new passwords and ensure the change is successful.	Password is updated successfully.	Pass
Mismatched New Passwords	Enter two different new passwords and verify the error message.	Error message: New passwords do not match.	Pass
Incorrect Current Password	Provide an incorrect current password and ensure it's rejected.	Error message: Current password is incorrect.	Pass
Account Deletion			
Successful Deletion	Log in, then delete the account and verify removal from the database.	Account is deleted successfully.	Pass
Account Not Found	Try to delete a non-existent account and verify the error message.	Error message: User not found.	Pass
Admin Role Required	Attempt to delete an account as a non-admin user and ensure rejection.	Error message: Role required is admin.	Pass
Logout			
Successful Logout	Log in, receive a valid access token, and then successfully log out.	Success message: Successfully logged out.	Pass
Invalid JWT Token	Attempt to log out with an invalid/expired JWT token and verify rejection.	Error message: Unauthorized access.	Pass
Admin Account Deletion			
Successful Admin Deletion	Log in as admin, delete another user's account successfully.	Account is deleted successfully.	Pass

User Not Found	Try to delete a non-existent user account and verify the error message.	Error message: Account not found.	Pass
Non-admin Access	Attempt to delete an account as a non-admin user and ensure rejection.	Error message: Role required is admin.	Pass
View Users			
Successful User Retrieval	Ensure the endpoint returns a list of usernames when accessed by an authenticated user.	List of usernames is returned successfully.	Pass
Unauthenticated Access	Try to access the endpoint without providing a valid access token and verify rejection.	Error message: Unauthorized access.	Pass

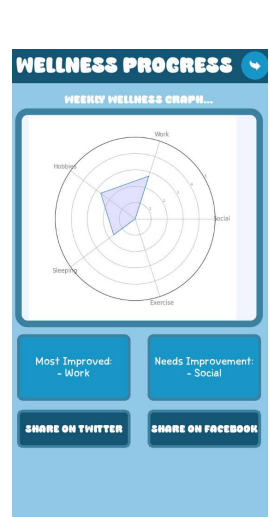
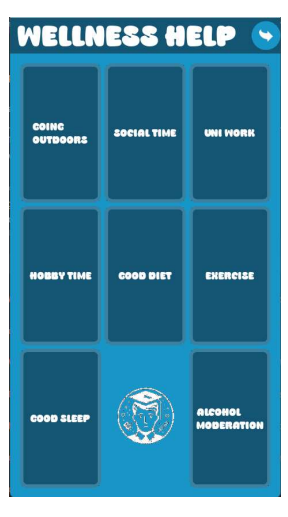


Conclusion

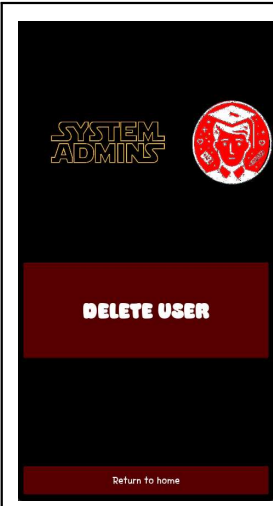
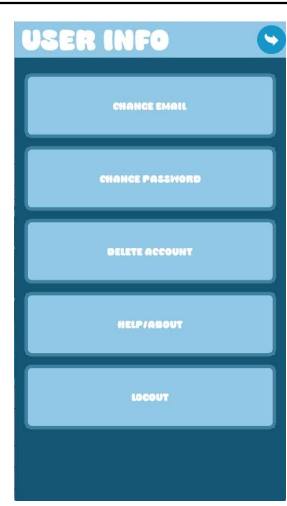
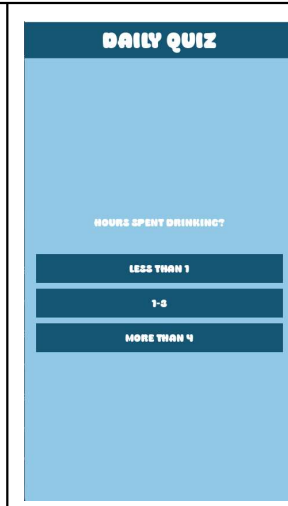
This testing documentation outlines the various test cases written to ensure the correctness and reliability of the Campus Care app. By executing these tests, developers can identify and address potential issues in the software.

GUI:

 <p>The login screen features the 'CAMPUS CARE' title, a university crest logo, and input fields for 'USERNAME' and 'PASSWORD'. A 'LOGIN' button is at the bottom, with links for 'Forgot password?' and 'Sign up' below it.</p>	 <p>The 'RESET PASSWORD' screen has an 'EMAIL ADDRESS' input field and a 'SEND LINK' button.</p>	 <p>The 'SIGNUP' screen includes input fields for 'USERNAME', 'EMAIL ADDRESS', 'PHONE NUMBER', 'DATE OF BIRTH', 'PASSWORD', and 'CONFIRM PASSWORD', followed by a 'SIGNUP' button.</p>	 <p>The home screen displays the 'CAMPUSCARE' title, a user profile icon, and an 'OVERALL WELLNESS PROGRESS CHART'. The chart shows a bar graph for 'Overall Progress' with categories: Socialization (2 hours), Sleeping (4 hours), and Drinking (5 hours). Below the chart are buttons for 'WELLNESS PROGRESS', 'WELLNESS SCHEDULER', 'WELLNESS HELP', 'OPTIONS MENU', 'DAILY QUIZ', and 'ADMIN MENU'.</p>
This is the login screen that the Campus Care app opens to. In this screen, the user has the option to login, sign up or reset their password.	This is the reset password screen. Here the user has the option to enter the email connected to their account. Where a time-based pin is sent which they can enter to the GUI in order to	This is the signup/register screen where the user must enter their username, email, phone number, date of birth and password. Any validation errors appear as a popup	This is the app's home screen. It shows the users overall progress through displaying their average daily time spent on each activity in a graph. From here the user can access the

	automatically login.	when the user tries to sign in.	screens; user info (top right button), wellness progress, wellness help, daily quiz (if not taken that day), wellness schedule, options, and admin menu (if user is admin).
--	----------------------	---------------------------------	---

			
<p>This is the wellness progress screen, it displays a graph showing last week's average daily time spent on each activity. It also displays the activity that the user has improved on the most and needs to improve on the most, based on the last two weeks of data. There is also the option to post their most and least improved activities on social media (Facebook or Twitter).</p>	<p>This is the wellness help screen, where the user can view the wellness/health benefits of each monitored activity (through a popup that appears when a button is selected).</p>	<p>This is the wellness schedule screen, where the user has the option to log activities to days in a month. The coloured lines in each button indicate the activity entered, however when a date button is selected the user can view/remove any specific activities logged. The clear all button simply removes all logged activities.</p>	<p>This is the options menu, where the user can select/change the specific activities they want to be asked about on the daily quiz.</p>

		
<p>This is the page exclusively visible by admins. It allows admins to view all users and delete users of their liking. The design is designed to stand out.</p>	<p>This is the user info page, it is where the user has the options to change their email, change their password, delete their account, view the apps About info and log out of their account (returns them to login).</p>	<p>This is the daily quiz., where the user is asked how many hours, they have spent doing certain activities that day. This quiz can only be taken once a day.</p>


Accessibility:


To ensure the GUI is accessible, we ran each screen through the online colour blindness simulator (COBLIS Color Blindness Simulator, n.d.) to see if there are any clashing colours, and each one was accessible with every colour blindness filter on the website. We also ensured all text was readable by making sure any text in the app is no smaller than 16 pixels - which is the standard minimum font size.

References:

COBLIS Color Blindness Simulator. (n.d.). Retrieved May 30, 2024, from <https://www.color-blindness.com/coblis-color-blindness-simulator>

Results of Twitter and Facebook links:





(Facebook has removed the ability to post a quotation through a link but if the app were deployed this feature would be fully functional)

This enables a user to post the most improved and least improved activity on social media

Contribution Matrix

Task	Harrison c2008863	Clo c2040208	Jessie c2059743	Filip c2029187	James c1017174	Matthew c0091403
User Database	M,R,T	C, R, T		R,M,T,	R,T	
User Management	C,M,R,T	R, T	M, R, T	C,M,R,T	R,T	
Sign-up /Log-in Functions	C,M,R,T	R, T	M, R, T	C,M,R,T	R,T	
Database security	C,M,R,T	R, T	M, R, T	M,R,T	R,T	
Admin Management				C,M,R,T	M,R,T	
Screen functionality	M,R,T	M, R, T	C, M, R, T	M,R,T	R	
Screen manager	R,T	M, R, T	C, M, R, T	R,T		
Kivy (kv) files		M, R, T	C, M, R, T	M,R,T	M,R	
Activity Database		C, M, R, T	M, R, T	R,T		
User preferences Database	M,R,T	C, M, R, T	M, R, T	R,T	R	
Daily Quiz Management		C, M, R, T	C, M, R, T			
Generating graphs based on User stats		C, M, R, T	M, R, T			
Activity Logging		C, M, R, T	C, M, R, T		R	
Validation checks	C,M,R,T	T	M, R, T	R,T	R,T	
Encryption	C,M,R,T				R	
Test		C, M, R	M, R			

Documentati on and Unit Tests						
--	--	--	--	--	--	--

Key:

C = Create R = Review

M = Modify T = Test