

1. Project Title and Authors

Team No.: 21

Project: KeepFit

Members:

1. Ayushi Gupta - ayushigu@usc.edu - 8595068906
2. Armaan Pishori - pishori@usc.edu - 4252700260
3. Samantha Tripp - stripp@usc.edu - 7855953482
4. Jason Ahn - ahnjason@usc.edu - 3832800566
5. Vedant Nevatia - vnevatia@usc.edu - 6562275237
6. Joseph Yoon - josephy@usc.edu - 2233295549

2. Preface

KeepFit is a fitness-centric social media platform. It is intended for people to watch and/or create exercise videos or live streams to stay fit during the pandemic.

The aim of this document is to include the tests and the process used to test our implementation of KeepFit. We organize our testing process into black-box and white-box tests. Within these 2 categories, we also have subcategories of tests relating to the functionality being tested (e.g. “Post tests” or “Login tests”). For black-box tests, we used Cypress, because of its compatibility with React Native. For white-box tests, we used unittest, which was compatible with Python/Django in keep-fit’s backend stack. Some of the tests conducted uncovered bugs, and the process by which we fixed such bugs is detailed in the document below.

Table of Contents

[1. Project Title and Authors](#)

[2. Preface](#)

[Table of Contents](#)

[3. Instructions](#)

[4. Black-box Tests](#)

[Posts & Streams \(Front-End\)](#)

[1. Homescreen post load and navigation](#)

[2. Homepage stream load and navigation](#)

[3. Like Post](#)

[4. Upload Post](#)

[5. Upload Stream](#)

[5. White-box Tests](#)

[Users \(Endpoints\)](#)

[Get User Profile:](#)

[Register User:](#)

[Login User:](#)

[Update User:](#)

[User Search List:](#)

3. Instructions

a. The instruction for executing your test cases

To run black-box tests:

- Cd keep-fit/frontend
- \$(npm bin)/cypress open

To run white-box tests:

- Cd keep-fit/backend
- python manage.py test

4. Black-box Tests

a. You should have at least 15 executable black-box test cases.***

b. Each test case must have the following information

- The location of the test case (e.g., names of folder, file, and test case).
- Description of what the test case does and how to execute the test case.
- Description of the rationale behind the test case (e.g., Why do you choose the specific inputs? How do you generate the test case?).
- The result of executing the test case (e.g., screenshot with description).
- If the test case helps you uncover a bug, document the details (e.g., What is the bug? How did you fix it?). Remember to do regression testing after fixing the bug, to make sure you did not break anything that was working previously.

Posts & Streams (Front-End)

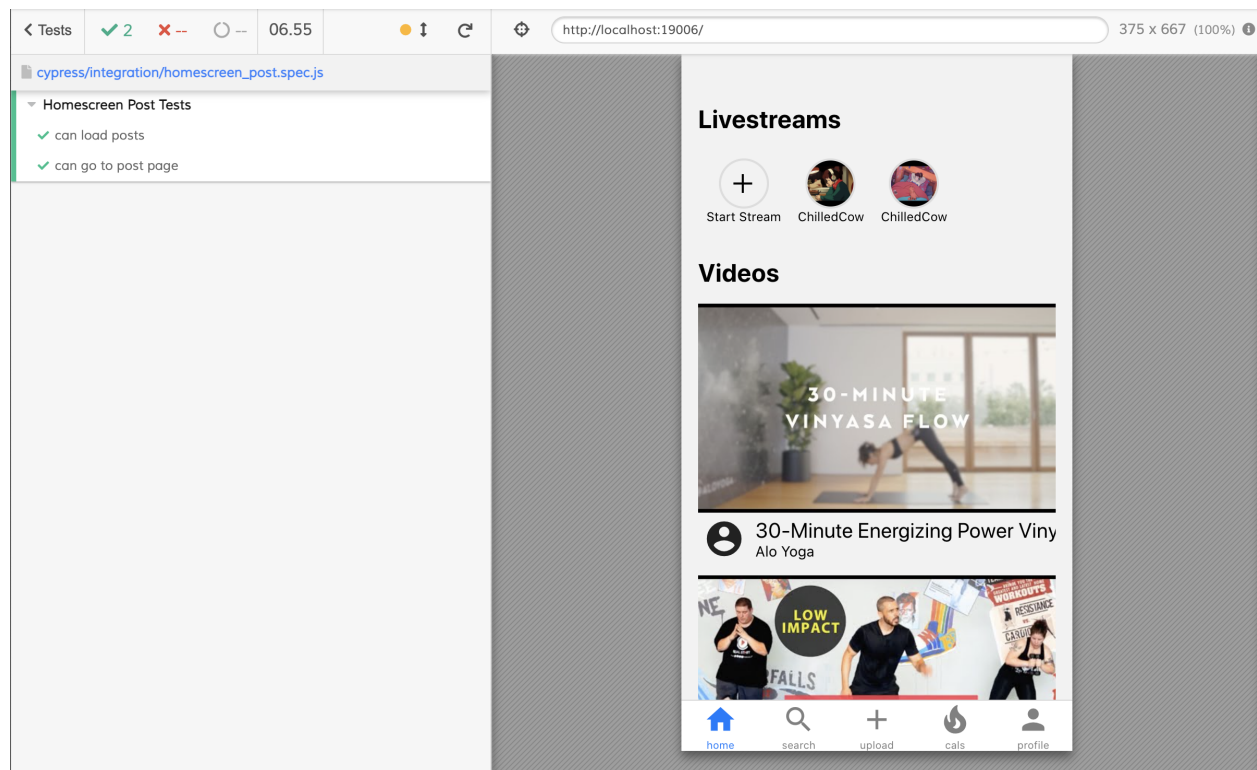
1. Homescreen post load and navigation

Location: /keep-fit/frontend/cypress/integration/homescreen_post.spec.js

Description: This tests if posts load and if navigation to a post page works correctly. Execute this test case by following the black-box test instructions and clicking on “homescreen_post.spec.js” linked in blue in the Cypress window.

Rationale: Test chosen because it is important to validate whether or not posts have appeared and post navigation functionality. Not only is this an important feature of the app, but also other tests such as uploading posts and liking posts require posts to load correctly.

Result:



Screenshot of homepage with posts loaded. All tests have passed.

Findings: This test verified post functionality on the homepage, and did not result in uncovering any bugs

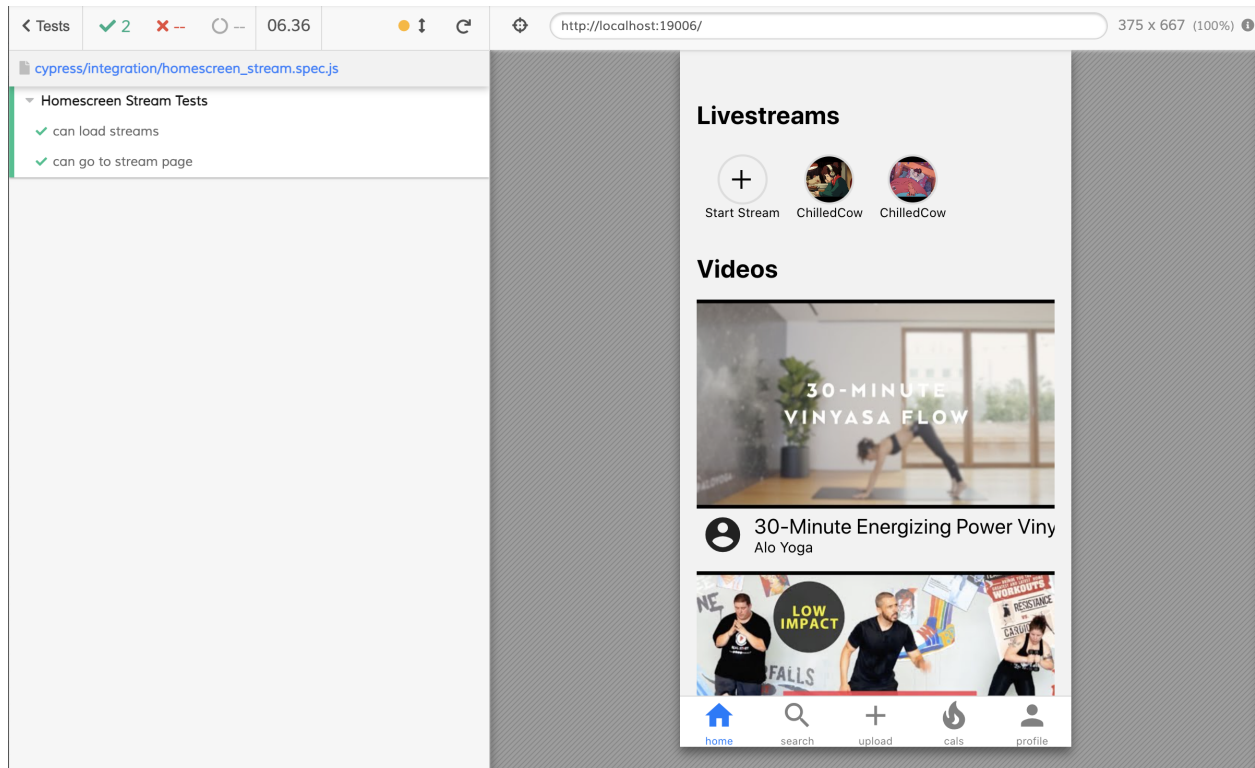
2. Homescreen stream load and navigation

Location: /keep-fit/frontend/cypress/integration/homescreen_stream.spec.js

Description: This tests if streams load on homepage and if navigation to a stream page works correctly. Execute this test case by following the black-box test instructions and clicking on “homescreen_stream.spec.js” linked in blue in the Cypress window.

Rationale: Test chosen because it is important to validate whether or not streams have appeared and stream navigation functionality. This is one of the core requirements of the app, and would lead to a poor user experience if no streams appeared on the home page.

Result:



Screenshot of homepage with streams loaded. All tests have passed.

Findings: This test verified stream functionality on the homepage, and did not result in uncovering any bugs

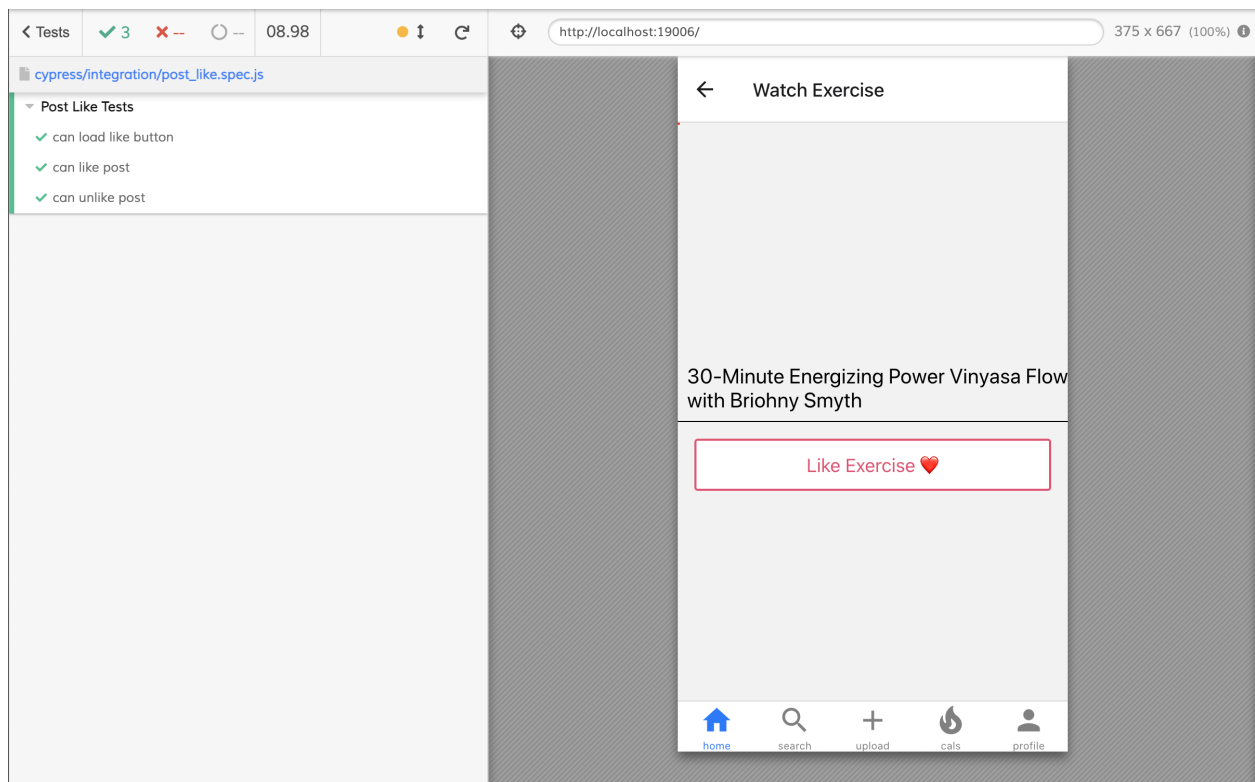
3. Like Post

Location: /keep-fit/frontend/cypress/integration/post_like.spec.js

Description: This tests if posts can be liked and disliked. Execute this test case by following the black-box test instructions and clicking on "post_like.spec.js" linked in blue in the Cypress window.

Rationale: Test chosen because it is necessary to validate whether or not posts have appeared and post navigation functionality. This is a core requirement of the app and provides convenience to users, as it is the only way they can save videos they enjoy to their profile.

Result:



Screenshot of post page with like button. All tests have passed.**

Findings: This test verified post like/dislike functionality, and did not result in uncovering any bugs.

**Note: the exercise video does not appear when the app is accessed by browser due to lack of compatibility. Exercise videos are displayed in <webview>, which is functional in iOS and android simulators.

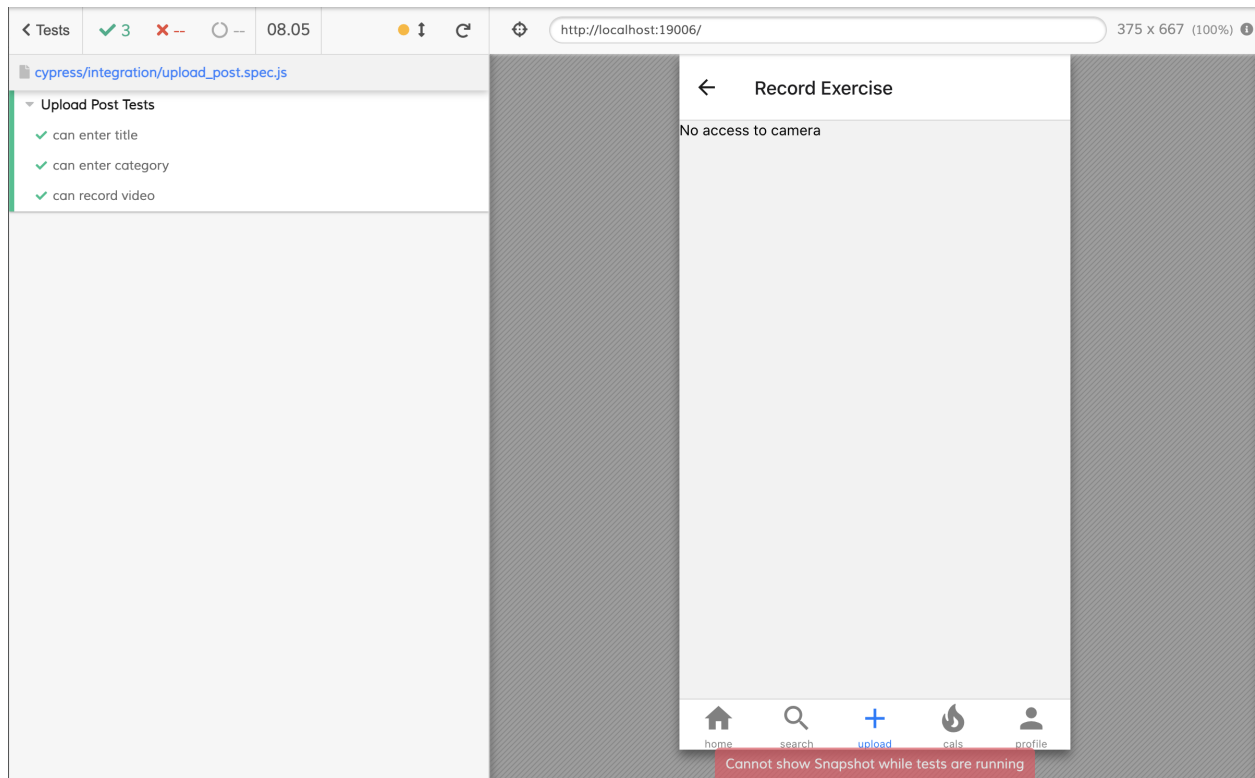
4. Upload Post

Location: /keep-fit/frontend/cypress/integration/upload_post.spec.js

Description: This tests if a user can provide all the necessary information to create a post. Execute this test case by following the black-box test instructions and clicking on "upload_post.spec.js" linked in blue in the Cypress window.

Rationale: Test chosen because it is important to validate that information needed to create a post is being input in the correct form.

Result:



Screenshot of tests run after fix to text fields. Note: camera does not work in simulator, which is why there is an error message stating “no access to camera”.

Findings: This test found a bug relating to the text input for titles and categories. Previously, a user could only enter one letter for categories and titles of a post. Changes to `keep-fit/frontend/src/uploadvideoscreen.js` fixed this issue.

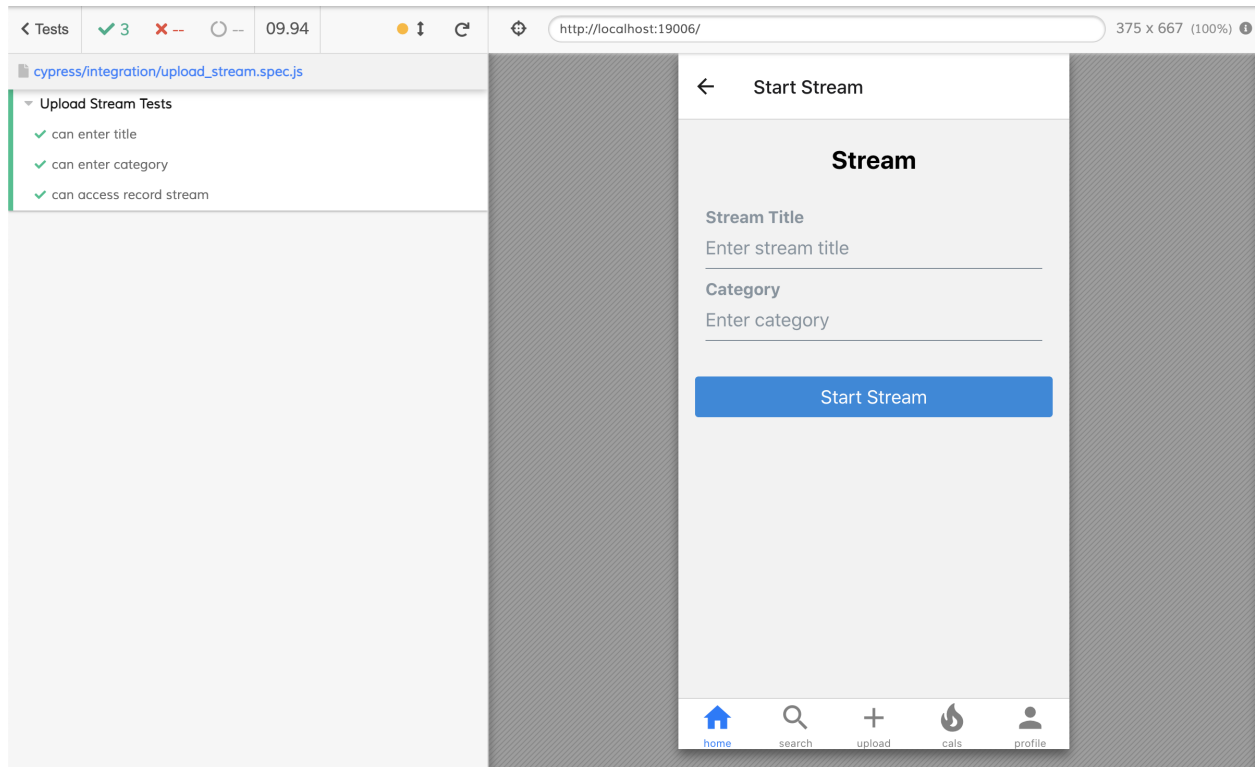
5. Upload Stream

Location: `/keep-fit/frontend/cypress/integration/upload_stream.spec.js`

Description: This tests if streams load and if navigation to a stream page works correctly. Execute this test case by following the black-box test instructions and clicking on “`upload_stream.spec.js`” linked in blue in the Cypress window.

Rationale: Test chosen because it is important to validate that information needed to create a stream is being input in the correct form, so that users may access live streams created by others.

Result:



Screenshot of stream upload form a user must fill out before starting a stream. All tests have passed.

Findings: This test found a bug relating to the text input for titles and categories, similar to the issue in upload post tests. Previously, a user could only enter one letter for categories and titles of a stream. Changes to `keep-fit/frontend/src/uploadstreamscreen.js` fixed this issue.

Login & Search

6. Successful Login

Location: `/keep-fit/frontend/cypress/integration/login_spec.js`

Description: This tests if correct user credentials take the user to the home screen when the submit button is pressed. Execute this test case by following the black-box test instructions and clicking on “`login_spec.js`” linked in blue in the Cypress window.

Rationale: Test chosen because it is important to validate that the back-end checks if user credentials are accurate in order to log a user in.

Result: This test found a bug in the login process which allowed us to check the backend login authentication function.

7. Unsuccessful Login

Location: `/keep-fit/frontend/cypress/integration/login_spec.js`

Description: This tests if incorrect user credentials take the user back to the login screen when the submit button is pressed. Execute this test case by following the black-box test instructions and clicking on “login_spec.js” linked in blue in the Cypress window.

Rationale: Test chosen because it is important to validate that the back-end checks if user credentials are accurate in order to log a user in. Also, it is important to validate that users who do not use correct credentials are not able to access the home screen.

Result: This test found a bug in the login process which allowed us to check the backend login authentication function.

8. Delete Profile

Location: /keep-fit/frontend/cypress/integration/profile_spec.js

Description: This tests if clicking the button to delete a user's profile actually deletes the users profile and takes the user to the login screen. Execute this test case by following the black-box test instructions and clicking on “profile_spec.js” linked in blue in the Cypress window.

Rationale: Test chosen because it is important to validate that the delete profile button properly and accurately deletes the user's profile.

Result:

Findings: This test validated the accuracy of the delete profile button.

9. Update Profile

Location: /keep-fit/frontend/cypress/integration/profile_spec.js

Description: This tests if clicking the button to update a user's profile takes the user to the update profile page. Execute this test case by following the black-box test instructions and clicking on “profile_spec.js” linked in blue in the Cypress window.

Rationale: Test chosen because it is important to verify the user's ability to update their profile.

Result: This test verified the accuracy of the update profile button.

10. Forgot Password

Location: /keep-fit/frontend/cypress/integration/forgot_spec.js

Description: This tests if clicking the forgot password button on the login page takes the user to the forgot password page, and if from there the user is able to reset their password and be taken to the login screen. Execute this test case by following the black-box test instructions and clicking on “forgot_spec.js” linked in blue in the Cypress window.

Rationale: Test chosen because it is important to validate that the forgot password button allows the user to reset their password.

Result: This test validated the accuracy of the forgot profile button and found a bug in the forgot password page.

Search & Calorie Counter

11. Search Page Display

Location: /keep-fit/frontend/cypress/integration/user_search.spec.js (line 13)

Description: This tests if elements of the search page are displayed. Execute this test case by following the black-box test instructions and clicking on “user_search.spec.js” linked in blue in the Cypress window.

Rationale: Test chosen because it is important that all elements display.

Result: This test validated the initial page displays normally.

12. Dropdown Toggles

Location: /keep-fit/frontend/cypress/integration/user_search.spec.js (line 26)

Description: This tests if the dropdown menu for selecting search type toggles correctly. Execute this test case by following the black-box test instructions and clicking on “user_search.spec.js” linked in blue in the Cypress window.

Rationale: Test chosen to validate that the dropdown menu for different search types can be shown/hidden.

Result: This test validated the dropdown menu functionality.

13. User Search

Location: /keep-fit/frontend/cypress/integration/user_search.spec.js (line 36)

Description: This tests if searching for users works. Uses hardcoded test values that were input into a local database. Execute this test case by following the black-box test instructions and clicking on “user_search.spec.js” linked in blue in the Cypress window.

Rationale: Test chosen because it is important to validate that the forgot password button allows the user to reset their password.

Result: This test validated that the search functionality works and can partially render user info.

14. Empty Search

Location: /keep-fit/frontend/cypress/integration/user_search.spec.js (line 52)

Description: This tests if submitting a search with no term results in the right result status. Execute this test case by following the black-box test instructions and clicking on “user_search.spec.js” linked in blue in the Cypress window.

Rationale: Test chosen to validate that empty results are displayed properly.

Result: This test validated the accuracy of displayed search result info when no search term is given.

15. Posts/Streams Search

Location: /keep-fit/frontend/cypress/integration/user_search.spec.js (line 64)

Description: This tests if clicking different options in the dropdown menu and performing the selected type of search works. Execute this test case by following the black-box test instructions and clicking on “user_search.spec.js” linked in blue in the Cypress window.

Rationale: Test chosen because it validates that different types of searches work..

Result: This test validated that the search itself works, but is unable to display results.

16. Exercise List Display

Location: /keep-fit/frontend/cypress/integration/calorie_counter.spec.js (line 9)

Description: This tests if the calorie counting page can be accessed and that all elements are displayed. Execute this test case by following the black-box test instructions and clicking on “calorie_counter.spec.js” linked in blue in the Cypress window.

Rationale: Test chosen to validate that initial list of exercises to pick from is shown.

Result: This test validated that all exercises are properly displayed.

17. Select Exercise for Counter

Location: /keep-fit/frontend/cypress/integration/calorie_counter.spec.js (line 26)

Description: This tests if the counter page can be accessed and if the selected exercise is correctly used for the counter. Execute this test case by following the black-box test instructions and clicking on “calorie_counter.spec.js” linked in blue in the Cypress window.

Rationale: Test chosen to validate that counter can be accessed and that the correct exercise is used.

Result: This test validated the functionality of selecting an exercise and going to the counter to begin tracking calories burned.

18. Calorie Counter Displays

Location: /keep-fit/frontend/cypress/integration/calorie_counter.spec.js (line 34)

Description: This tests if all elements of the counter are displayed (timer, start, pause, stop, chosen exercise/back to list, calories burned/reset calories). Execute this test case by following the black-box test instructions and clicking on “calorie_counter.spec.js” linked in blue in the Cypress window.

Rationale: Test chosen to validate that all necessary elements are properly displaying.

Result: This test validated that all elements display properly.

19. Basic Timer Functionality

Location: /keep-fit/frontend/cypress/integration/calorie_counter.spec.js (line 60)

Description: This tests if basic functions (start, stop, pause, reset calories) work as expected. Execute this test case by following the black-box test instructions and clicking on “calorie_counter.spec.js” linked in blue in the Cypress window.

Rationale: Test chosen to validate that all basic functionality of timer works, which is needed to then calculate the calories burned.

Result: This test validates that all basic functionality of the timer works as expected.

20. Pick New Exercise

Location: /keep-fit/frontend/cypress/integration/calorie_counter.spec.js (line 85)

Description: This tests going from the counter back to the list of exercises and selecting a new exercise. Execute this test case by following the black-box test instructions and clicking on “calorie_counter.spec.js” linked in blue in the Cypress window.

Rationale: Test chosen to validate that selecting different exercises would all work after having already set one.

Result: This test validates that navigating back and forth between exercise list and counter while selecting different exercises all work properly.

5. White-box Tests

--

Identify the coverage criteria used in your white-box testing: What coverage objective(s) did you opt for and why? How did you ensure you are meeting your coverage criterion?

write something here?

--

Users (Endpoints)

The location of all the following test cases are located in the keep-fit/backend/users/test.py file. Within this file, for each endpoint there is an appropriate TestCase class which contains the setup for each test case and functions to test the endpoint. Executing these test cases can be done by running “python manage.py test users” in the keep-fit/backend folder. The result of running all these cases can be seen in the following image.

```
PS C:\Users\armaa\repos\keep-fit\backend> python .\manage.py test users
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
-----
Ran 16 tests in 5.369s

OK
Destroying test database for alias 'default'...
```

1. Get User Profile:

These test cases test whether we are able to retrieve a user's profile based on the username given in the url. To test it we chose two cases:

- Providing a valid username in the url (Status: 200, It should retrieve that users profile data)
- Providing an invalid username in the url (Status: 404, Should not retrieve any info)

2. Register User:

This will test whether we can register a new user into the system based on the given user data. We chose three cases:

- Providing valid data (Status: 201, Should return the new users data)
- Providing invalid data (Status: 400, Should return error message about the data)
- Providing valid data of an existing user (Status: 400, Should return error message that user already exists)

While testing this endpoint, there was a bug in the implementation of registering a user which did not account for the username given. Thus, the appropriate changes were made to make this work. We also needed to validate whether the user already existed in the database or not.

3. Login User:

This will test whether users can log into the system or not according to their username and password. We chose four cases:

- Valid username and valid password (Status: 200)
- Valid username and invalid password (Status: 400)
- Invalid username and valid password (Status: 400)
- Invalid username and invalid password (Status: 400)

4. Update User:

These test cases will test whether a user can update their profile information or not based on the data they send and the currently logged in user.

- Valid update data and valid logged in user (Status: 201, returns the updated user information)
- Valid update user data but invalid logged in user (Status: 400, returns errors mentioning that the user already exists)
- Valid user data but no existing user (Status: 404, returns an error that the user does not exist)

5. User Search List:

These cases will test whether we can retrieve a list of users based on a given query.

- No query provided (Status: 200, Returns an empty list of users)
- Query retrieving 2 users (Status: 200, Returns a list of two users)
- Query retrieving the first user (Status: 200, Returns a list of just the first user)
- Query retrieving the second user (Status: 200, Returns a list of just the second user)

--

Posts (Endpoints)

The location of all the following test cases are located in the **keep-fit/backend/posts/test.py** file. Within this file, for each endpoint there is an appropriate TestCase class which contains the setup for each test case and functions to test the endpoint. Executing these test cases can be done by running “**python manage.py test posts**” in the **keep-fit/backend** folder. The result of running all these cases can be seen in the following image.

```
vnevatia@Vedants-MacBook-Pro backend % python3 manage.py test posts
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
-----
Ran 10 tests in 1.150s

OK
Destroying test database for alias 'default'...
vnevatia@Vedants-MacBook-Pro backend %
```

6. Create Post:

These test cases test the backend functionality of whether we are able to create a post given the input from the front-end. To test it we chose **two test cases**:

- Providing valid inputs to all necessary fields (Status: 201, it will create the post with the relevant input)

- Providing an invalid (blank) response to the 'title' field (Status: 400, it will not create the post due to invalid input). My rationale for this was that people often forget to fill the field in, or choose not to. I would like each post to have a title, and so this test ensures that intended functionality is seen through.

7. Search Posts by Title:

This will test the backend functionality of whether we can search through all posts by title, based on a given string. We chose **four test cases**:

- No query string provided (Status: 200, Returns an empty list of posts)
- Query string retrieving all (2) posts (Status: 200, Returns a list of all posts)
- Query string retrieving post 1 (Status: 200, Returns a list of just the first post)
- Query string retrieving post 2 (Status: 200, Returns a list of just the second post)

8. Toggle Like Post:

This **one test case** will test the backend functionality of whether users can like and unlike posts according to their preferences. Since it was implemented as a toggle functionality, we chose **one test case** which covers both like and unlike:

- Like and unlike get requests sent successively, boolean values were verified after each to be representative of toggling between like and unlike

9. Get all Posts (for Feed):

This **one test case** will test the backend functionality of whether a user can access the main feed on keepfit to view all the video posts that have been created by other users.

- 3 posts were created with different input using a get request (Status: 200, Returns a list of all 3 posts)

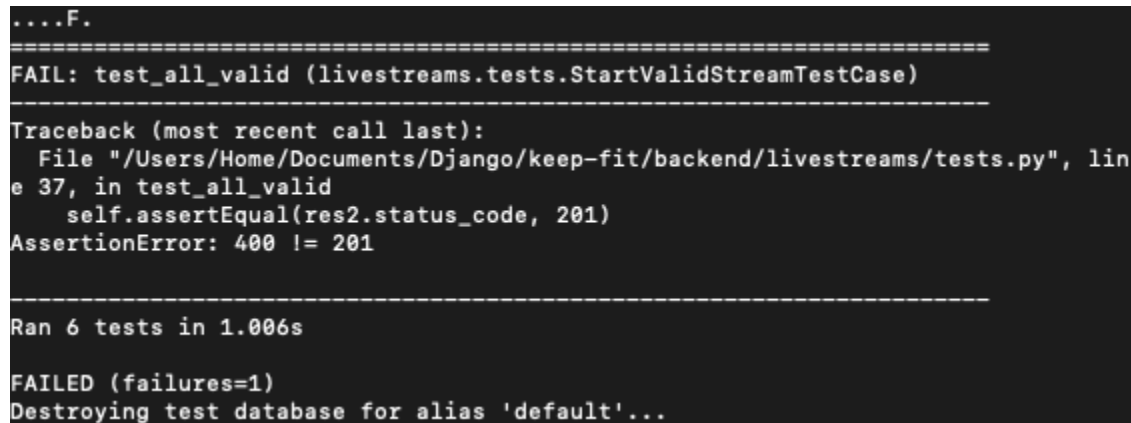
10. Get Posts by Author (for viewing another profile):

These **two test cases** will test the backend functionality of whether users can view other user's posts by going to their profiles. We can retrieve a list of posts based on a given user.

- A user uploaded 2 posts after logging in- then we used a get request (Status: 200, the appropriate user's posts were retrieved)
- Another user uploaded a single post after logging in- then we used another get request (Status: 200, the appropriate user's post was retrieved)

Livestreams (Endpoints)

The location of all the following test cases are located in the **keep-fit/backend/livestreams/test.py** file. Within this file, for each endpoint there is an appropriate TestCase class which contains the setup for each test case and functions to test the endpoint. Executing these test cases can be done by running “**python manage.py test livestreams**” in the **keep-fit/backend** folder. The result of running all these cases can be seen in the following image.

A terminal window showing the output of running tests. It starts with '....F.' indicating one failure. The failure is for 'test_all_valid' in 'livestreams.tests.StartValidStreamTestCase'. A traceback shows the error occurred in 'tests.py' at line 37, in the 'test_all_valid' method, where 'self.assertEqual(res2.status_code, 201)' failed because the actual status code was 400. The terminal also shows 'Ran 6 tests in 1.006s' and 'FAILED (failures=1)'.

11. Start Valid Stream

This test case will test the backend functionality of whether users can create a working livestream. It is important as users being able to create live streams is a major component of KeepFit. This is what should happen in the test case.

- A user creates a stream after logging in- then we used a get request (Status: 201, the user's live stream was created)
- Bugs: The Livestream Serializer File had incorrect variables, having video instead of video_APIKey. Fixing this bug may take some time so it is only fixed to a limited extent.

12. Start Stream without Title

This test case will test the backend functionality of what happens when a user forgets to put a title when starting a livestream. Users could forget to put a title in, which would make it difficult for other users to search for.

- A user creates a stream after logging in but forgets to put a title. (Status: 400, the user's live stream is invalid, having no title.)

13. Display all Streams

This test case will test the backend functionality of being able to show currently running live streams on the app. This is a big part of the app as displaying live streams is a way for users to find exercise streams to join without searching for a specific one.

- The user's live streams page will display all current live streams. (Status: 200, the request was processed.)

14. Search Stream by Title

This test case will test the backend functionality of a user inputting text into a search query to find given live streams. There are two test cases, one for checking if a user put in text or if a user put in no text into the query.

- The user inputs no text into the search query. (Status: 200, the user's search is empty, so no results are returned.)
- The user's search request returns a live stream with a title that matches the user input. (Status: 200, the user's search was processed.)

15. Display Multiple Streams from Search

This test case will test the backend functionality of being able to display multiple live streams by the user's search input. The importance here is to stretch the ability of single searching and display multiple live streams whose titles include the user's input.

- The user's search request returns with multiple live streams whose title includes the user's input. (Status: 200, the user's search was processed.)