Web Server Test Plan

# Overview

This document serves as a test plan for verifying correct behavior of the web server. This test plan will test both the user facing web serving capabilities of the application, as well as the REST API used by a mobile client and XHR JavaScript requests.

This test plan covers higher level integration/systems tests. To test individual functions of the application, unit tests should be written and executed.

# Test Setup

Prior to beginning the test, the tester should generate a JSON file with a week’s worth of step data. This known data will be used to verify that endpoints are returning the expected content, whether that be a dynamic HTML page or a JSON documents.

# Begin Test

1. Verify that the backend database has been emptied. Run the development web server by executing the command `py manage.py runserver`. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
2. Attempt to create a new user by sending the following body in a POST request to /api/user/

{

    "user": {

        "username": "test8",

        "email": "test@aol.com",

        "first\_name": "Mitchell",

        "last\_name": "Larson",

        "password": "test"

    },

    "right\_shoe": {

        "foot": "R",

        "size": 8.5

    },

    "left\_shoe":{

        "foot": "L",

        "size": 8.5

    },

    "height": 67,

    "weight": 165,

    "step\_goal": 10000

}

* Verify that a 201-response code was returned. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
* Verify that the same JSON object was return, with the password removed and an id number present.

**Indicate pass or fail: \_\_\_\_\_\_\_\_\_**

* Verify that a User object was added to the auth\_user table.

**Indicate pass or fail: \_\_\_\_\_\_\_\_\_**

* Verify that two shoe objects were added to the Shoe table.

**Indicate pass or fail: \_\_\_\_\_\_\_\_\_**

* Verify that a LogistepsUser object was added to the LogistepsUser table.

**Indicate pass or fail: \_\_\_\_\_\_\_\_\_**

1. Attempt to create the same user again by sending the same HTTP request. Verify that the server responds with a 400-response, indicating that the user exists. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
2. Attempt to create a user a field omitted. Verify that the server responds with a 400-reponse, indicating that the data is invalid. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
3. Attempt to retrieve the user without providing valid credentials by making a GET request to /api/user/test8/. Verify that the server response with a 403-response, indicating in the response body that an invalid username/password was provided. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
4. Repeat the same request but provide the user’s username and password in the request header for authorization. The server should respond with a 200-response, providing the user object shown below in the JSON body. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**

{

"user": {

"id": 7,

"username": "test8",

"email": "test@aol.com",

"first\_name": "Mitchell",

"last\_name": "Larson"

},

"left\_shoe": {

"size": "8.5",

"foot": "L"

},

"right\_shoe": {

"size": "8.5",

"foot": "R"

},

"height": 67,

"weight": 165,

"step\_goal": 10000

}

1. Attempt to update a user’s profile by making a PUT request to /api/user/test8/, providing the correct authorizations in the header of the request. Use the following JSON object as the body of the request.

{

    "user": {

        "username": "test9",

        "email": "test@aol.com",

        "first\_name": "Mitchell",

        "last\_name": "Larson",

        "password": "test"

    },

    "right\_shoe": {

        "foot": "R",

        "size": 8.5

    },

    "left\_shoe":{

        "foot": "L",

        "size": 8.5

    },

    "height": 67,

    "weight": 165,

    "step\_goal": 10000

}

Verify that the server response with a 400-response, indicating that the user’s username cannot be changed, and verify that the user has not been updated in the LogistepsUser database table. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**

1. Repeat the same request as step 7, but omit the username field, and change the user’s step goal to 50. Verify that the server responses with a 200-response and that the user object has been updated in the LogistepsUser table in the database. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
2. Attempt to post step data for the user created in the previous steps by making a POST request to /api/steps/. The user’s username and password should be provided in the header of the request, and the JSON object generated in the test setup should be provided as the body of the request.

* Verify that the server responds with a 201-reponse containing the step data posted to the server. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
* Verify that the Step database table now holds rows corresponding to the posted step data. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**

1. Attempt to get a list of all step data that the user generated for a given day in the step data generated in the test setup by making a GET request to `/api/steps/steplist/?date=`, providing the given day at the end of the url in mm-dd-yyyy format. Verify that the server responds with a 200-response, proving an array of step data that occurred for the given day. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
2. Repeat the same request but omit the GET parameters (/api/steps/steplist). Verify the server responds with a 200-response, providing an array of step data that occurred on the present day.  
   **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
3. Repeat the same request as step 10 but provide a malformed date string. Verify that the server responds with a 400-responds. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
4. Request a step summary for a given day in the data generated in the test setup by making a GET request to ‘/api/steps/summary/?date=’, providing the given day at the end of the url in mm-dd-yyyy format. Verify that the server responds with a 200-response, with a body containing a JSON object with format shown below. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**

{

"steps": 33,

"goal": 49,

"percent": 67.3469387755102,

"least\_active": {

"hour": 2,

"steps": 1

},

"most\_active": {

"hour": 0,

"steps": 5

},

"inactive\_time": {

"hours": 23,

"minutes": 29

},

"steps\_per\_hour": 1.375

}

* Manually verify that the calculated statistics match with the data generated in the test setup. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**

1. Repeat the same request but omit the GET parameters. Verify that the statistics return match with manual calculations for step data posted for the current day. **Indicate pass or fail: \_\_\_\_\_\_\_**
2. Request a step count for the range of dates covered by the data generated in the test setup by making a GET request to /api/steps/count/?start=<startdate>;end=<enddate>. Provide the dates in mm-dd-yyyy format. Verify that the server responds with a 200-response containing a JSON object with the following format. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**

{

"range": {

"start": "10-22-2018",

"end": "10-26-2018"

},

"counts": {

"10-22-2018": 9435,

"10-23-2018": 2345,

"10-24-2018": 5555,

"10-25-2018": 6483,

"10-26-2018": 5093

}

}

* Verify that the start date and end dates match the GET parameters.

**Indicate pass or fail: \_\_\_\_\_\_\_\_\_**

* Verify that there is a key/value pair for each date in the range.

**Indicate pass or fail: \_\_\_\_\_\_\_\_\_**

* Manually verify that the correct step counts are calculated.

**Indicate pass or fail: \_\_\_\_\_\_\_\_\_**

1. Request a breakdown for the number of steps taken, grouped by week. This can be done by making a GET request to /api/steps/breakdown/. Provide the GET parameters “groupby=weekly”. Verify that the server responds with a 200-response in the format shown below. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**

{

"groupby": "weekly",

"week": [

{

"day": 0,

"steps": 45235,

"active time": 3542,

"inactive time": 3245

},

{

"day": 1,

"steps": 45235,

"active time": 3542,

"inactive time": 3245

},

{

"day": 2,

"steps": 45235,

"active time": 3542,

"inactive time": 3245

},

…

* Perform a manual calculation based on the setup data, and verify that the step count, active time, and inactive time match. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**

1. Request a pressure breakdown from the server by making a GET request to /api/steps/pressure, and append the GET a date parameter for a given day. Verify that the server responds with a 200-response containing a body with the following structure. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**

{

"query\_date": "04-30-2019",

"pressure": {

"past\_day": {

"left\_shoe": [

{

"location": "T",

"avg\_pressure": 0

},

{

"location": "B",

"avg\_pressure": 0

}

],

"right\_shoe": [

{

"location": "B",

"avg\_pressure": 31.1591

},

{

"location": "T",

"avg\_pressure": 33.3683

}

]

},

"past\_week": {

"left\_shoe": [

{

"location": "B",

"avg\_pressure": 96.4787878787879

},

{

"location": "T",

"avg\_pressure": 96.7030303030303

}

],

"right\_shoe": [

{

"location": "B",

"avg\_pressure": 31.1591

},

{

"location": "T",

"avg\_pressure": 33.3683

}

]

},

"past\_month": {

"left\_shoe": [

{

"location": "T",

"avg\_pressure": 8.30329144225015

},

{

"location": "B",

"avg\_pressure": 96.4787878787879

}

],

"right\_shoe": [

{

"location": "T",

"avg\_pressure": 33.3683

},

{

"location": "B",

"avg\_pressure": 31.1591

}

]

}

}

}

* Manually calculate, based on the data generated in the test setup, the average pressure for the past day, past week, and past month. Verify that the manual calculations match the response from the server. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
* Verify the server returns a query\_date the same as the date parameter appended to the URL. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**

1. Perform the same query but omit the date parameter. Verify that the server returns data for the previous day, week, and month from the current day. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
2. Request the user’s location data for a given day by making a GET request to /api/steps/location/, appending a date parameter to the URL. Verify that the server returns a 200-response with an array of JSON objects containing latitude and longitude key pairs for the date requested. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**

This concludes the tests for verifying proper behavior of the REST web API. The next tests will verify proper user facing web behavior. The pages served to a user use the same REST API that was previously documented, so the following tests aim to verify the proper elements are present in the page, and that the user’s path through the application is correct.

1. Navigate to /logisteps/ and verify that the web server redirects the browser to /accounts/login/, displaying a form with a username field, a password field, and a login button.

**Indicate pass or fail: \_\_\_\_\_\_\_\_\_**

1. Enter invalid credentials into the form and submit it to the web server by clicking login. Verify that an error message indicating that the username or password was incorrect, and that the web client remained at the login screen. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
2. Re-enter the user’s correct credentials, and verify that the web client is redirected to /logisteps/  
   **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
3. Upon Login, verify that the user is presented with the recent view by default.   
   **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
4. Verify that there is a navigation bar present spanning the top of the screen, allowing a user to navigate between their dashboard, connections, and global data.   
   **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
5. Verify that a side navigation bar is present which allows a user to navigate between the recent tab, “steps over time” tab, a “steps by weekday” tab, an “activity by week” tab, a “pressure” tab, and a “map” tab. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
6. Verify that the recent view presents the user with the following data:

* A progress graphic for the current date and the previous date, displaying how close they are to meeting their daily step goal. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
* Verify that under each graphic, a table is present with total steps, average steps per hour, least active hour, most active hour, and inactive time. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**

1. Navigate to the steps over time page and verify that the following elements are present:

* Verify that a line graph is present, with days of the week on the x-axis, and step count on the y-axis. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
* Verify that there is a filter at the bottom of the page allowing a user to adjust the viewing/data window. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
* Verify that the graph re-renders when the filter is changed. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**

1. Navigate to the Activity by Week page and verify that the following elements are present:

* Verify that inactive time vs active time is displayed using stacked bar charts with days of the week as the x-axis, and time as the y-axis. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
* Verify that a interactive filter is present, allowing a user to change the week displayed week.  
  **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
* Verify that the graph re-renders when a different week is selected.   
  **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**

1. Navigate to the pressure tab and verify that the following elements are present:

* Verify that there are 3 elements displaying average pressure. One for the past day, past week, and past month. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
* Verify that there is an interactive graphic present for changing the date.  
  **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
* Verify that the graphics re-render when the filter is changed.  
   **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**

1. Navigate to the Location tab and verify that the following elements are present:

* Verify that a map is displayed, with a user’s location data overlaid onto the graph.  
  **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
* Verify that the user can change the displayed day using an interactive filter.  
  **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
* Verify that when the date is changed, the graph re-renders with updated user data.  
  **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**

1. Navigate to /logisteps/profile/ and verify that a form is present allowing a user to update their profile information. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
2. Change a user’s first name in the form and press submit. Verify that the user profile was updated in the database by inspecting the auth\_user table. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
3. Press the logout button in the upper-right corner of the screen and verify that the user is redirected to /logisteps/logout.html. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**
4. Lastly, use an HTTP client to send a DELETE request to /logitstep/user/test8/, providing the correct credentials, and verify that the user was successfully deleted from the auth\_user and LogistepsUser database. **Indicate pass or fail: \_\_\_\_\_\_\_\_\_**