T5: Second Iteration

**The Snake Squad:**

Yifei Wang (yw3471),

Yuanxin Yang (yy3036),

Harsh Yadav (hy2616),

Daria Manea(dem2184),

[Ki Cooley-Winters](https://courseworks2.columbia.edu/groups/91530/users/389874) (nhc2116)

Part 1:

User stories we implemented:

1. As a college student who is preparing for finals, I want to manage my time so that I can study more efficiently and more thoroughly.

My condition of satisfactions:

a. keep tracks of how long I have been study and schedules break time wisely over the final week

b. I should be able to review how much I have done

## Acceptance Test for user story 1:

* 1. When the user begins a timed session, the timer should start and display the amount of time left in real time. Otherwise, the test should fail. When a study session ends the user should be notified and the study session should be saved. Otherwise, the test should fail.
  2. If a user wants to reset the timer, they can click on the reset button and the timer should start again from the top. Test passes if the timer starts at 25 minutes again. It fails if it goes back to the previous time.
  3. A user should be able to skip a pomodoro period and go to the break session. The test passes if the break button is clicked, and it fails if the timer doesn’t redirect to the break view.

a)

|  |  |  |  |
| --- | --- | --- | --- |
| input | GIVEN | WHEN | THEN |
| good | Start button is clicked | User wants to start timer | Timer displays countdown |
| bad | Start button is NOT clicked, and the background is clicked instead | User wants to start timer | Timer doesn’t displays countdown |

b)

|  |  |  |
| --- | --- | --- |
| GIVEN | WHEN | THEN |
| Start button is clicked | User wants to restart timer |  |
| Start button is not clicked, and the background is clicked instead | User wants to restart timer |  |

c)

|  |  |  |  |
| --- | --- | --- | --- |
| input | GIVEN | WHEN | THEN |
| good | Break button is clicked | User wants to skip to the break timer | Timer displays countdown: the 5 minute break timer starts |
| bad | Break button is NOT clicked, and the background is clicked instead | User wants to skip to the break timer | Timer doesn’t displays countdown: session continues |

1. As a college student I want to be able to review my history and see how I am spending my time online.

My condition of satisfactions:

If I am able to retrieve information about the websites I visit, the test should pass. Otherwise, it should fail.

|  |  |  |  |
| --- | --- | --- | --- |
| input | GIVEN | WHEN | THEN |
| good | User runs the local script when the browser is closed, and the script asks for their user id. The post request is sent and the body of the request is printed: | User wants to send their chrome browsing history to the web app | The server shows that the data was received successfully, and the browsing history was saved to the database: |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| bad | User attempts to run the local script to get browser history while the browser is still open (invalid): | User wants to send their chrome browsing history to the web app | Script displays an error message and tells user to close their browser first |

1. As a father of a young child, I want to minimize the chances of my child being exposed to some harmful/inappropriate websites which may cause him to have bad habits before he can be responsible for himself, so that I want to try my best to block such websites when he is using the computer.

My condition of satisfactions:

I want to block the harmful websites during my child’s daily usage of the computer (i.e. use the productive interval of the timer to cover my child’s daily usage can satisfy this condition)

|  |  |  |  |
| --- | --- | --- | --- |
| Input with sudo permission | Website blocker activation | During blocked time | After blocked time |
| Sudo with correct system indicated and valid block time | Print: website blocker is activated ... | when trying to access a blocked website | Blocked websites can now be accessed |
| Either one is incorrect | Prompt user to input valid info | blocked and unblocked websites can be accessed if the blocker is not started properly | All websites can be accessed |

1. As a software developer who sits all day working, I want to spend some time doing some workouts everyday so that I can live a healthy lifestyle.

My condition of satisfactions:

1. As a beginner, I want to make a not-so-intense workout cycles
2. I want to be able to see how many hours of exercise I have done over some period of time so that I can keep a reasonable amount of workout while not over exercising

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Login Page | Dashboard Home Page | Session Detail Page |
| User with login credentials |  | Displays all the sessions when the person switched the timer on | Displays the details of browser activity during this time. |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Login Page | Input | Register page |
| User with login credentials |  | User with valid user info inserted |  |
| User without login credentials |  | User with invalid email address |  |
|  |  | User with invalid password |  |
| Input | Logout Page | Input | OAuth Login Page |
| Successfully Logged out the valid account |  | Valid OAuth github user authentication |  |
|  |  | Valid access token requested, and successfully redirected to the homepage |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Part 2:

The application consists of 4 major services namely Timer, Dashboard, Activity Tracker, and Website Blocker. Unit tests were conducted on each of the services with the following equivalence partitions in mind:

1. Dashboard: The dashboard is a one-stop-shop for a user to view their previous activity and visualize the results of pomodoro sessions.   
   In a Django application, major functionality is handled by views. Following are the views:
   1. Home: This is the view that displays all the previous pomodoro sessions.
      1. Input: User
      2. Valid Equivalence Partitions: Authenticated/Registered user.
         1. Tests:
            1. test\_dashboard\_home\_authenticated(): Dashboard loads with a status 200 code.
      3. Invalid Equivalence Partitions: Random unauthorized user.
         1. Tests:
            1. test\_dashboard\_home\_unauthenticated(): Dashboard return with error code 302.
   2. Timer Sessions: This subroutine grabs all the previous pomodoro sessions by the user.
      1. Input: User
      2. Valid Equivalence Partitions: Authenticated/Registered user.
         1. Tests:
            1. test\_dashboard\_session\_authenticated(): Returns all the data for the user using the user id.
      3. Invalid Equivalence Partitions: Random unauthorized user.
         1. Tests:
            1. test\_dashboard\_session\_unauthenticated(): Dashboard return with error code 302.
   3. Timer Session Detail: This is the subroutine that displays in detail the information about a pomodoro session like the sites visited, sites blocked, and start and end time of the session.
      1. Input: User, Session
      2. Valid Equivalence Partitions: Authenticated User, Valid Session ID.
         1. test\_session\_detail\_authenticated(): Returns all the details of a valid session id if the user is authenticated.
      3. Invalid Equivalence Partitions: Unauthorized user, session id not belonging to the user, session id does not exist.
         1. test\_session\_detail\_unauthenticated(): Returns with error code 302 when the user isn’t authorized.
         2. test\_session\_detail\_not\_exist(): Returns with error code 401.
         3. test\_session\_detail\_belongstootheruser(): Returns with error code 302.
   4. Get Timer Sessions:
      1. Input: User, Start Time to get timer sessions from, End Time to get timer sessions to.
      2. Valid Equivalence Partitions: Authenticated user, all timer sessions, timer sessions within the last week, timer sessions within the last month.
         1. test\_timer\_session\_all\_api(): Returns all the timer sessions.
         2. test\_timer\_session\_weekly\_api(): Returns the timer sessions in the past week.
         3. test\_timer\_session\_monthly\_api(): Returns the timer sessions in the past month.
      3. Invalid Equivalence Partitions: Unauthorized user, time in the future, time format invalid
   5. Get Session Activity:
      1. Input: User, Start Time to get session activity from, End Time to get session activity to.
      2. Valid Equivalence Partitions: Authenticated user, all session activity, timer sessions within the last week, timer sessions within the last month.
         1. test\_timer\_session\_activity\_all\_api(): Returns all the timer session activity.
         2. test\_timer\_session\_activity\_weekly\_api(): Returns the timer session activity in the past week.
         3. test\_timer\_session\_activity\_monthly\_api(): Returns the timer sessions in the past month.
      3. Invalid Equivalence Partitions: Unauthorized user, time in the future, time format invalid, start and end time don’t correspond to a valid session.
   6. Get Blocked Sites:
      1. Input: User
      2. Valid Equivalence Partitions: Authenticated user.
         1. test\_blocked\_sites\_all\_api(): Returns all the blocked sites.
      3. Invalid Equivalence Partitions: Unauthorized user.
         1. test\_blocked\_sites\_unauthorized(): Returns 404 error.

**Timer**

* The views include the following methods:

**HOME**

1. Input: User. Partitions: authenticated and unauthenticated users .
2. Valid Equivalence partition: authenticated User.
   1. test\_home\_authenticated(self): checks for valid authenticated user and returns 200 (good) as response code from webserver
3. Invalid Equivalence partition: unauthenticated user.
   1. test\_home\_nonauthenticated(self): checks for valid non-authenticated user and returns 302 (bad) as response code from webserver

**START**

1. Input: User. Partitions: authenticated and unauthenticated users .
2. Valid Equivalence Partitions: authenticated User.
   1. test\_start\_authenticated(self): checks for valid authenticated user and returns 200 (good) as response code from webserver
3. Invalid Equivalence Partitions: Unauthenticated user.
   1. test\_start\_nonauthenticated(self): checks for valid non-authenticated user and returns 302 (bad) as response code from webserver

**START\_BREAK**

1. Input: User. Partitions: authenticated and unauthenticated users .
2. Valid Equivalence Partitions: authenticated User.
   1. test\_break\_authenticated(self): checks for valid authenticated user and returns 200 (good) as response code from webserver
3. Invalid Equivalence Partitions:

Unauthenticated user.

* 1. test\_break\_nonauthenticated(self): checks for valid non-authenticated user and returns 302 (bad) as response code from webserver

FRONT END:

1. **Start**.html is tested in \_\_tests\_\_/test.start.js

Partitions: the declared value in the html file for each button as valid partition, anything else as invalid.

Valid input: the right value for the button ({pomodoro-reset: reset} and break button ({pomodoro-break: break}

Invalid Input: ({pomodoro-reset: something-else} and ({pomodoro-break: something-else}

1. **Home**.html is tested in \_\_tests\_\_/test.home.js

Partitions: the declared value in the html file for each button as valid partition, anything else as invalid.

Valid input: the right value for the reset button ({pomodoro-reset: reset}

Invalid Input: ({pomodoro-reset: something else}

1. **Break**.html is tested in \_\_tests\_\_/test.break.js

Partitions: the declared value in the html file for each button as valid partition, anything else as invalid.

Valid input: the right value for the break button ({pomodoro-reset: reset}

Invalid Input: ({pomodoro-reset: something else}

**Register**

* The views include the following methods:

**Registration**

1. Input: User
2. Valid Equivalence Partitions: authenticated User.
   1. test\_register(cls): checks for valid authenticated user and returns 200 (good) as response code from web server
3. Invalid Equivalence Partitions: Invalid user.
   1. I have used django built in functionality, django.contrib.auth.urls, so it’s not necessary to implement the functionality within the views.py. Since the api is well defined and encapsulated and all the error cases are well handled, we indeed do not need to test it. That’s why I only put the valid equivalence partition. You can see the invalid case handling of the register in page 6.

**Login**

1. Input: User
2. Valid Equivalence Partitions: authenticated User.
   1. Same reason as 3.a in Registration. You can see the valid case handling of the login started at page 6.
3. Invalid Equivalence Partitions: Unauthenticated user.
   1. Same reason as 3.a in Registration. You can see the invalid case handling of the login started at page 6.

**Logout**

* 1. Input: User
  2. Valid Equivalence Partitions: authenticated User.
     1. test\_logout\_page(cls): checks for valid authenticated user and returns 200 (good) as response code from webserver
  3. Invalid Equivalence Partitions:

Unauthenticated user.

* + 1. Same reason as 3.a in Registration. I think the invalid cases for logout is handled implicitly by the django.contrib.auth.urls, so I cannot really find the failing condition for logout (that means I have to login with some invalid account in the first place, which is impossible b/c it will be handled by the login functionality).

**OAuth with Github**

* 1. Input: User
  2. Valid Equivalence Partitions: authenticated User.
     1. checks for authenticated access token that used to request the data from github and returns 200 (good) as response code. This check is surprisingly hard because each token has an expiration time, and the period is pretty short. Once you hard coded the current working tokens to request the data in the github, it will work at the time you test it, but when the time TA checks your test case has exceeded the expiration time of that token, then your test case will not work. That’s why I cannot put the acceptance case in unit test of OAuth.
     2. checks for unauthenticated user request the authorization layer to return the token for requesting the access token and returns 200 (good) as response code from webserver. This has the same reason listed in part a.
  3. Invalid Equivalence Partitions:

Unauthenticated user.

* + 1. test\_github\_user(cls): checks for unauthenticated access token that used to request the data from github and returns 400 (bad) as response code from webserver
    2. test\_github\_token(cls): checks for unauthenticated user request the authorization layer to return the token for requesting the access token and returns 400 (bad) as response code from webserver

-Tracker:

-There are tests for the various operating systems, because the correct path to the chrome history directory varies across systems. However, because Travis CI as is a linux machine, only part of these tests were run there. Additionally, the Travis CI machine did not have chrome browser installed, which meant both tests failed as the chrome history file could not be found. These tests did pass on the local machine: <https://github.com/ki-cooley/4156TeamProject/blob/main/pomodoro/htmlcov/timer_browserhistory_py.html>

-There were also tests written for the endpoints and their integration with the local components, but unsolved errors in Django’s testing framework (which failed to create a clean database for tests) meant these tests did not pass cleanly. For the local login feature, the tests partition checks a valid username and password to ensure a token is returned, as well as an incorrect password and incorrect username as two invalid inputs.

**Blocker**

The blocker function mainly works locally. We start with asking the user for his/her id and password to login to the database. If it is a valid login, the user can proceed to use the blocker and the tracker. If not, a window will pop up, tell the user it’s not a valid login, and ask them to do it again.

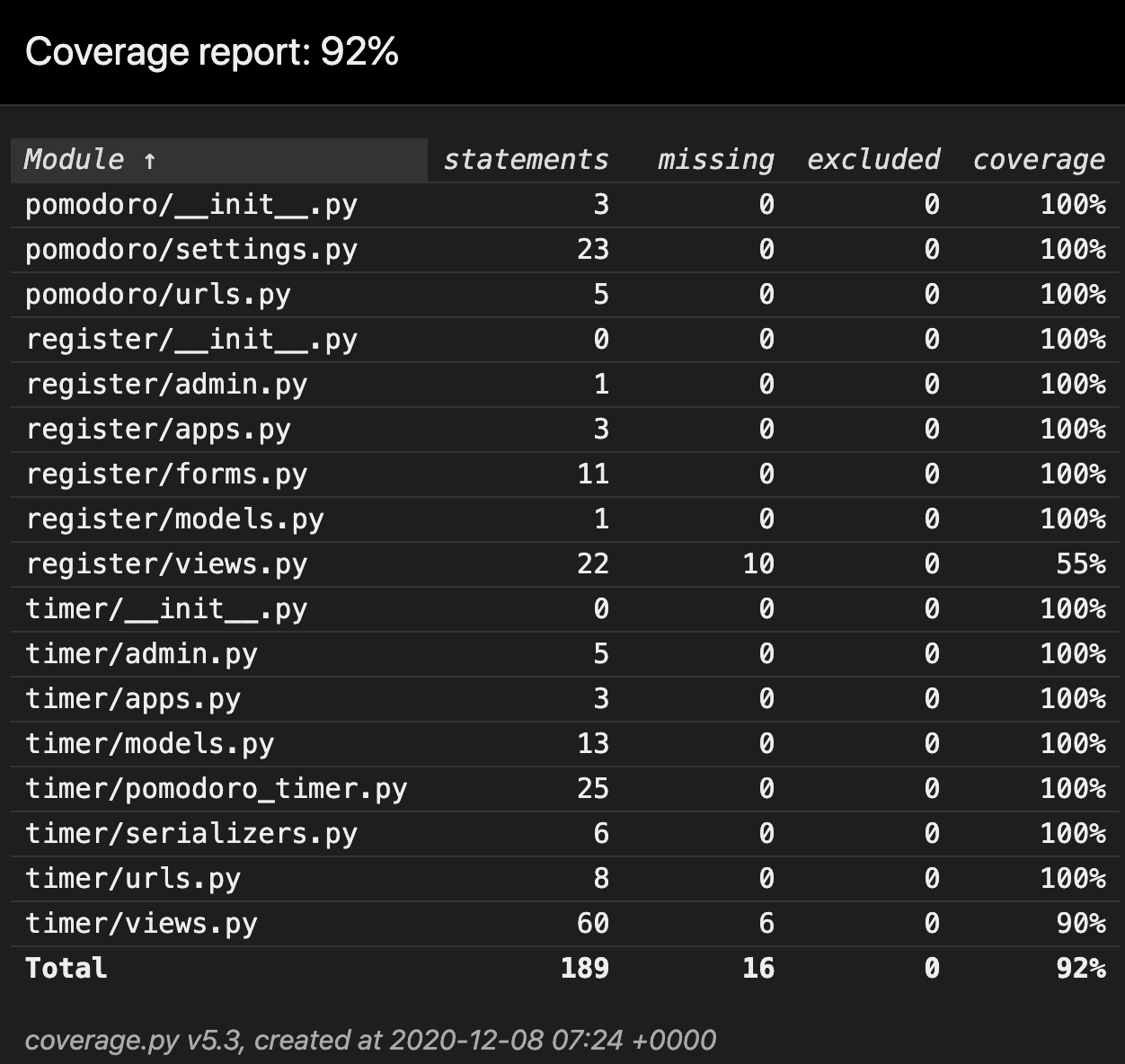
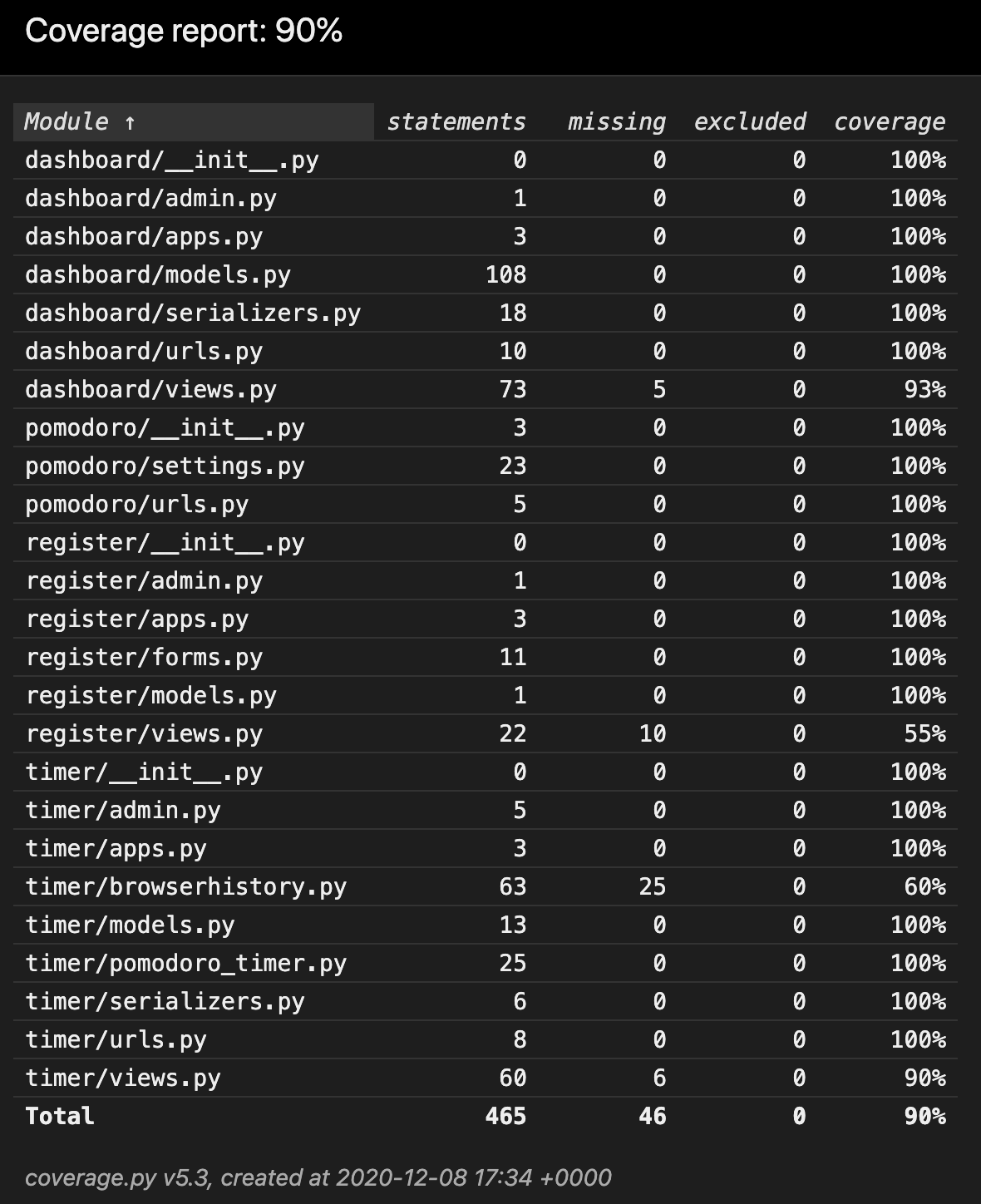
Part 3:

Coverage Percentage: 80%

Coverage Report: <https://github.com/ki-cooley/4156TeamProject/blob/main/pomodoro/htmlcov/index.html>

Unit testing and the subsequent coverage was performed by each team member for their individual modules. The results of unit tests are as under:

1. Dashboard (Harsh)
   1. Coverage: 93%
   2. Unable to Test: Branches within methods that dealt with unauthenticated users. Django uses the @login-required decorator to ensure a view is accessible only for authenticated users, and hence the conditions handling unauthenticated users weren’t met.
   3. Link:
      1. Unit Tests: <https://github.com/ki-cooley/4156TeamProject/tree/main/pomodoro/dashboard/tests>
      2. Coverage Report: <https://github.com/ki-cooley/4156TeamProject/blob/main/pomodoro/htmlcov/dashboard_views_py.html>, <https://github.com/ki-cooley/4156TeamProject/blob/main/pomodoro/htmlcov/dashboard_urls_py.html>
2. Timer (Daria)
   1. Coverage: 92%
   2. Unable to test \_\_init\_\_ method that acts as constructor.
   3. Unit tests:
      1. Front end: <https://github.com/ki-cooley/4156TeamProject/tree/main/pomodoro/timer/__tests__>
      2. Back end: <https://github.com/ki-cooley/4156TeamProject/blob/main/pomodoro/timer/tests/test_urls.py> and <https://github.com/ki-cooley/4156TeamProject/blob/main/pomodoro/timer/tests/test_views.py>
3. Tracker
   1. The coverage for browserhistory.py is only 68%, because there is a fair amount of os-specific code that does not get executed when the tests are run through Travis CI (the code is able to handle windows, mac, and linux, but the tests are run only a linux machine)
   2. https://github.com/ki-cooley/4156TeamProject/blob/main/pomodoro/htmlcov/timer\_browserhistory\_py.html



Part 4:

1. We used Travis Ci: <https://github.com/ki-cooley/4156TeamProject/blob/main/.travis.yml>
2. Report: <https://travis-ci.com/github/ki-cooley/4156TeamProject>
   1. The README file also has a building badge that shows the current state of the build. The link with the report can be accessed by clicking on it. 