

Mellivora

Sprint Report 5

Capstone Computing Project 2 - Group 02

Semester 2, 2017

Table of Contents

Progress Report..... 2

 Tasks Completed..... 2

 Tasks Planned..... 2

 Difficulties..... 2

Task Breakdown..... 3

 Task 37..... 3

 Task 37.1..... 3

 Task 37.2..... 4

 Task 52..... 6

 Task 52.1..... 6

 Task 52.5..... 9

Development Methodology..... 12

 Minutes..... 12

 Burndown Chart (Task 47.2)..... 12

 Sprint Retrospective (Task 47.3)..... 13

 Task Summary (Task 48.1)..... 14

 Time Management..... 14

 Task Selection..... 14

Progress Report

Sprint 5

25th September → 8th September

Tasks Completed

Task 37 – Home page additions.

Task 52 – Timers.

Tasks Planned

Task 13 – Question ordering.

Difficulties

There were a few problems encountered during the implementation of Task 52. View the Task Breakdown for Task 52 for more information.

View the **Sprint Retrospective** and individual for a more detailed reflection on the sprint.

Task Breakdown

Task 37

Task 37.1

([19165ad](#))

Estimated Time:	3 Hours
Actual Time:	2.5 Hours
Actual Time (this sprint):	2.5 Hours

Description

For the admin account only, display a button that allows the admin to edit the text on the front page.

Implementation

On the home page ([htdocs/home.php](#)), when a user is logged in as staff an edit button will appear in the top right of the home page. Additionally, when logged in as staff an edit form will also be included on the page that is hidden by default ([htdocs/css/home.css](#)). Pressing the edit button will toggle the form's visibility ([htdocs/js/home.js](#)) on the page. The home page body text, similar to the already existing news functionalities' body text, supports BBcode formatting. When the save button is pressed, the new body text is saved into the database.

Mellivora

Manage
Home
Test
Scoreboard
Profile
Log out

Logged in as [Test Admin](#)

Edit

Body

Save

BBcode

- **Text Styles:**
 - [b]...[/b]
 - [i]...[/i]
 - [u]...[/u]
 - [s]...[/s]
 - [sup]...[/sup]
 - [sub]...[/sub]
 - [spoiler]...[/spoiler]
 - [acronym]...[/acronym]
 - [size=6]...[/size]
 - [color=red]...[/color]
 - [font=verdana]...[/font]
- **Links:**
 - [url]...[/url]
 - [url=...]text[/url]
 - [email]...[/email]
 - [wiki]

- **Replaced Items:**
 - [img]...[/img]
 - [rule]
 - [br]
- **Alignment:**
 - [center]...[/center]
 - [left]...[/left]
 - [right]...[/right]
 - [indent]...[/indent]
- **Columns:**
 - [columns]...[/columns]
 - [nextcol]
- **Containers:**
 - [code]...[/code]
 - [quote]...[/quote]

- **Lists:**
 - [list]...[/list]
 - [*]...

Testing

A Codeception acceptance test class named [HomePageCest.php](#) is used to automatically test the home page. This class contains the following two tests for testing the home page edit functionality:

testHomePageEditVisible: This test checks to ensure that the edit button (and the form on the page) is not visible when logged in as a normal user. Should only be visible when logged in as staff (are an admin).

testAdminHomePageEdit: This test ensures that when the edit button is pressed the form becomes visible and can be edited. Once a new home page body is saved the test checks it is correctly displayed on the page.

Codeception acceptance test output from HomePageCest.

```
Acceptance Tests (2) -----
✓ HomePageCest: Test home page edit visible (4.67s)
✓ HomePageCest: Test admin home page edit (2.25s)
-----
```

Note: To run this test, use the command `./run_tests acceptance/HomePageCest` from the tests folder. For more information on how to run the tests, follow tests/[README](#).

Task 37.2

([58c1d39](#), [d1da1d6](#))

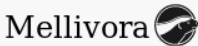
Estimated Time:	1 Hours
Actual Time:	0.5 Hours
Actual Time (this sprint):	0.5 Hours

Description

Add a button to the home page that directs the user to the 'student assessment page'.

Implementation

On the home page ([htdocs/home.php](#)), when a user is logged in and is a competing user a message will appear in the top center of the home page. This includes a button that links to the questions page.



[Home](#) [Test](#) [Scoreboard](#) [Profile](#) [Log out](#)

Score: 0 / 10,825 (0%)

01:00:00

Logged in as [Test Competitor 4](#)

You are currently competing in this test! Click below to go to the questions page.

[View Test](#)

Testing

A Codeception acceptance test class named [HomePageCest.php](#) is used to automatically test the home page. This class contains the following test for testing the competing message:

testHomePageCompetingMessageVisible: This test checks that the competing message is only visible for a logged in user that is currently competing. The competing field in the database is modified to ensure that when the page refreshes the message is not displayed.

Codeception acceptance test output from HomePageCest.

```
Acceptance Tests (1) -----
✓ HomePageCest: Test home page competing message visible (1.64s)
-----
```

Note: To run this test, use the command './run_tests acceptance/HomePageCest' from the tests folder. For more information on how to run the tests, follow tests/[README](#).

Task 52

Task 52.1

([23ebb56](#), [40b131a](#), [b076cea](#), [10acbce](#), [40c97cb](#), [d10a368](#), [b89742a](#), [65f45cb](#), [603237f](#), [1aa7d10](#))

Estimated Time: 8 Hours
Actual Time: 12.25 Hours
Actual Time (this sprint): 12.25 Hours

Description

Add a global countdown timer to the interface. The timer is to be controllable by admin.

Implementation

Originally this task was to be a timer that is global to all users. It was mentioned by the client that they would like to be able to control timers specific to each user. For example, if a student starts late or needs to be paused and started later on for whatever reason. When `CONFIG_DEFAULT_TIMERS_SET` is true (it is by default), new users' timers will be set to `CONFIG_DEFAULT_TIMERS_DURATION` (1 hours by default). To implement this task for each user, the following fields were added to the users table in the database:

`timer_status`: This value is either 0 or 1 indicating whether users timer is set, paused or stopped or has been started. When the `timer_status` is 1 the timer will visually countdown on the page (JavaScript).

`timer_start`: This value is -1 by default. When a timer is started, this will be set to current time.

`timer_duration`: This value is -1 by default. When a timer is set the timer is updated with the specified duration. If a timer was started previously and paused the timer duration will be set to the time remaining.

When the users' timer status is 1, the `user-timer-active` class is added to the timer element on the page. This is targeted by the JavaScript (`htdocs/js/timers.js`) and at 1 second intervals the timer value will be decreased. The timers work similar to how the older challenge availability timers in Mellivora worked. This has the advantage of not having to track the timer remaining time for each second. With the information in this format the timer remaining time can be found by:

$$\text{timerRemaining} = \text{timer_start} + \text{timer_duration} - \text{currentTime}$$

OR

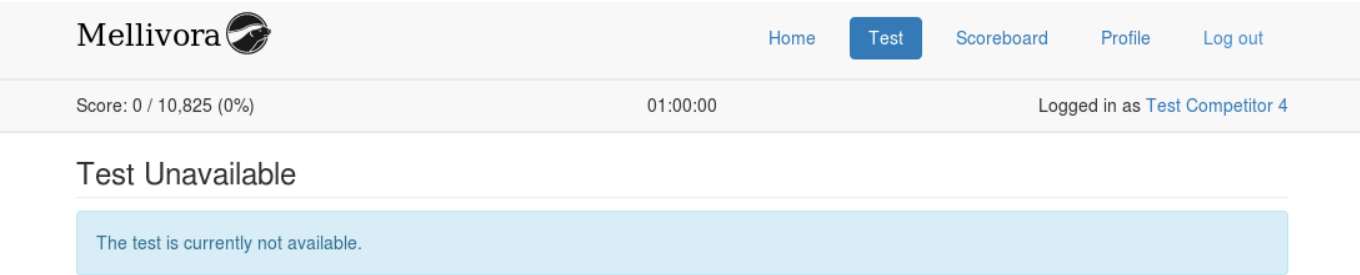
$$\text{timerRemaining} = \text{timer_duration}$$

OR

$$\text{timerRemaining} = -1$$

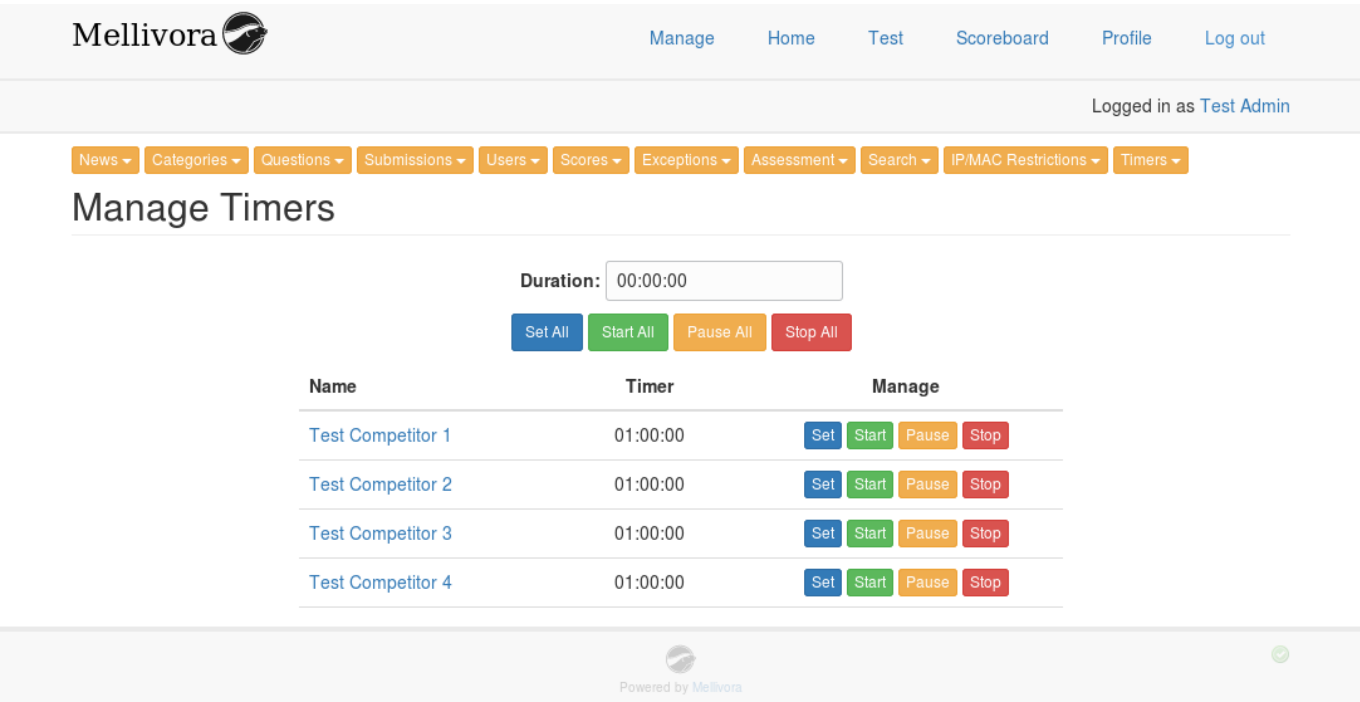
This operation is performed in the `user_timer_remaining` function in `include/user.inc.php`. The second function is used when the timer has not been started, only set or paused (`timer_start = -1`). The third function is used when the timer has been stopped (`timer_start = -1` and `timer_duration = -1`), in this state the timer will be displayed as '--:--:--'. Originally I was using NULL instead of -1 for when these values are disabled but the database functions in `include/db.inc.php` replace NULL values with 0.

When a users' timer is stopped, they will have access to the test and timers will take not effect. Once a users' timer has been set (or is paused) the test will be unavailable with a message displayed to the user. Once the users' timer is started the test will become available. When a users' timer runs out of the time the question page will be made unavailable and they will be redirected to a different page. This functionality was originally part of Task 52.5 but a WebSocket isn't really necessary for it.



In the above image, the users' timer has been set to 1 hour but has not been started. When the test page is visited the above message is displayed. Once the users' timer is started the test will become available.

In order to control user timers a manage timers page was added to the admin interface (htdocs/admin/[manage_timers.php](#)). This page allows timers to be controlled for all users as well as to the specific user. The duration field specifies what value to set the users timer. This can be in the format <hours>:<minutes>:<seconds> or simply the value of seconds.



I did run into a problem whilst implementing this task. When creating the form on the manage_timers page I was originally using <button> tags for the buttons. Like so (for setting timer for user with id 902):

```
<button name="set" value="902">Set</button>
```

This *should* result in the values like \$_POST['set'] == '902' or \$_POST['set'] == 'all'. This for an unknown reason wasn't working so I tried <input> tags. Like so (for setting timer of user with id 902):

```
<input type="submit" name="set_902" value="Set">
```


These do work but have one major downside of not being able to have the button value attribute different to the text displayed inside the button. This means value attribute must be "Set" for the button text and results in the value `$_POST['set_902'] == 'Set'`. For the action page (`htdocs/admin/action/manage_timers.php`) I therefore check if the value `$_POST['<action>_(<user_id>|"all")']` is set (is not null). The string `<action>_(<user_id>|"all")` is then split around the `'_'` to check what action is performed. This works but not as nice as the first solution.

For this task, similar to the scoreboard task, does work without the WebSocket, However, without the WebSocket the users' page will not be automatically updated, so it is recommended. For example, if the admin pauses a users' timer it will continue to visually tick on their end until they refresh the page. When the WebSocket is active users' timers will update automatically (Task 52.5).

Testing

A Codeception unit test class named [UserTimerCest.php](#) is used to automatically test the user timer functions. This class contains the following tests:

`testPrettyPrintTimer`: This tests the `pretty_print_timer($seconds)` function from `include/general.inc.php`. This function is used to format the timers to the format `<hours>:<minutes>:<seconds>`. The test simply checks the returned values are expected for the given inputs.

`testUnprettyPrintTimer`: This tests the `unpretty_print_timer($timer)` function from `include/general.inc.php`. This function is used to convert a timer string in the format `<hours>:<minutes>:<seconds>` to an integer value in seconds. The test simply checks the returned values are expected for the given inputs.

`testUserTimerStatus`: This tests the `user_timer_status($user_id)` function from `include/user.inc.php`. This function returns the current status of a specific users timers. Expected return values are 0 for inactive or 1 for an active timer. This test modifies a users' `timer_status` field in the database table 'users'.

`testUserTimerRemaining`: This tests the `user_timer_remaining($user_id)` function from `include/user.inc.php`. This function returns the amount of time remaining on the users timer. The return value will be an integer in the range `-1 <= x <= inf`. The time remaining will be -1 when the timer is stopped (not set). The test modifies the users' timer fields in the database and checks the time remaining is returned as expected.

Codeception unit test output from UserTimerCest.

```
Unit Tests (4) -----
✓ UserTimerCest: Test pretty print timer (0.00s)
✓ UserTimerCest: Test unpretty print timer (0.00s)
✓ UserTimerCest: Test user timer status (0.02s)
✓ UserTimerCest: Test user timer remaining (10.03s)
-----
```

Note: To run this test, use the command `./run_tests unit/UserTimerCest` from the tests folder. For more information on how to run the tests, follow tests/[README](#).

Task 52.5

([23ebb56](#), [b076cea](#), [0d596f7](#), [c36594c](#), [5447ede](#), [e60a56d](#), [cce3293](#), [d10a368](#), [65f45cb](#), [603237f](#), [1aa7d10](#))

Estimated Time:	4 Hours
Actual Time:	7.75 Hours
Actual Time (this sprint):	7.75 Hours

Description

Alter the timer so that when the timer ends, the websocket will cause the assessment page to refresh and display the "assessment has stopped" message

Implementation

Originally this task was planned to go along with Task 52.2 to Task 52.4, since those were removed due to a change in requirements this task's implementation is slightly different. For this task a WebSocket is used to automatically update a user's timer when it is modified by an admin. The assessment page refresh functionality (and the subsequent "assessment has stopped" message) was implemented in Task 52.1 as a WebSocket isn't really necessary for this functionality. If a user's timer is modified in a certain way (for example paused) by a `TIMER_UPDATE` from the WebSocket and the user is on the assessment page, it will reload to be updated (for example to show 'Test Unavailable').

The implementation for this began by generalising the WebSocket code created for the scoreboard updating WebSocket. This makes it easier to add more WebSocket servers. For this task I created `UserServer` ([websocket/UserServer.php](#)). This WebSocket server has basic user authentication (so the server knows which connected user is which). `UserServer` is planned to be used by xxxxx for some of his tasks relating to notifications. I opted for creating a general `UserServer` as opposed to something specific to just the timers for this reason. The WebSocket connection status is displayed faded-out inside the footer of each of the pages.



WebSocket status indicator.

When a user is logged in, they will be connected to `UserServer` (on every page). In order to send timer updates to users. On the page `htdocs/admin/manage_timers.php`, when an action is performed, the page will redirect back with the action variable set in the URL. For example `'action=set_all'`. When on the `manage_timers` page and an action is present in the URL the JavaScript (`htdocs/js/timers.js`) will send a `TIMER_UPDATE` message along with the action to the connected `UserServer` WebSocket. Originally I had this trigger when the button was pressed, the problem with this was the timer's were trying to update before the database values were updated.

When `UserServer` receives the `TIMER_UPDATE` message it will send the relevant authenticated user(s) new timer information. When the user's JavaScript receives the new timer data, the timer on the page is automatically updated. Additionally if your timer is updated and you are on the test page the appropriate actions are taken. For example, if your timer is paused the test page will be reloaded and you will no longer have access to the questions. Similarly if the test is unavailable (timer is set or paused) and your timer is started, the test page will automatically reload to reveal the questions.

Testing

A Codeception acceptance test class named [TimersCest.php](#) is used to automatically test the timers. This class contains the following tests:

checkUsersWebSocketConnection: This test checks that a successful connection is made to the UserServer WebSocket. The next test depends on this test passing successfully.

testTimerUpdate: This test is a bit different from the other tests and did take a fair bit of time to complete due to the problems encountered. The main issue with testing this task is the requirement of needing to be able to test two separate browser windows at a time, one logged in as a competing user and one logged in as an admin controlling the timers. The way Codeception works only allows for a single WebDriver browser session at a time. It is possible to open and control new windows in Codeception, however this test requires a different browsing session as it needs to be logged into two different accounts. Luckily, Codeception does contain a function [executeInSelenium](#) which allows access to the underlying php-webdriver [RemoteWebDriver](#) object. From this it is possible to use the getKeyboard function followed by sendKeys to send custom key input to the WebDriver controller the browser.

My idea for this was to send CTRL+SHIFT+P (or CTRL+SHIFT+N in Chrome) to open a new private browsing window in the same browsing session. This would allow me to have the main window logged into a competing user account and a secondary window logged in as an admin account. Unfortunately Chrome doesn't seem to allow for sending key inputs in this way as nothing happens. We switched to using Chrome for the WebDriver ([ChromeDriver](#)) testing last semester as the Firefox WebDriver ([GeckoDriver](#)) was causing a lot of problems. Those problems with the Firefox WebDriver seem to be fixed however similar to Chrome, it doesn't allow for sending key inputs. For Firefox we know this is the case since it throws an UnknownCommandException when trying to send key inputs.

After doing some searching I found out that Firefox's new automation interface named Firefox Marionette is now used by default and doesn't currently allow sending keys. When Marionette is disabled Firefox will fall back to its old method. This still, as before, seems to be a bit unstable as it sometimes failing for no apparent reason, that being said it seems to work most of the time. This now allows send keys to work and the secondary private window is successfully opened. Unfortunately as a result this test (as far as I know) currently only works on Firefox with Marionette disabled.

It is possible that in the future the send key function will work with Firefox Marionette or maybe even with Chrome. There are a few open issues on Firefox's WebDriver's GitHub page that I think may be related to this problem. I think a test like this that needs more than one browser session would probably best be done interacting with Selenium directly through their provided language bindings as that would allow more control and as well as more than one WebDriver session at a time. Or maybe there's a better way...

This test, after opening the secondary private window, will log in as a competing user in the main window and navigate to the test page. Then an admin is logged in the second private browsing window. The admin will then make modifications to the competing users timer. When a modification is made the test will switch windows and ensure that the timer or test page was updated correctly. When the test completes the secondary private browsing window is closed.

Codeception acceptance test output from TimersCest.

Acceptance Tests (2) -----

✓ TimersCest: Check users web socket connection (3.98s)

✓ TimersCest: Test timer update (24.77s)

Note: To run this test, the following is required:

- Switch Codeception WebDriver testing browser to 'firefox'. This can be done by editing the browser setting under WebDriver in tests/codeception/[acceptance.suite.yml](#).
- The next requirement for this test is to disable Firefox Marionette. To do this edit tests/[run_tests](#) and on the java Selenium execution line change webdriver.firefox.marionette to false. I would recommend keeping this on for other tests if still using Firefox as Marionette seems more stable.
- Run the command './run_tests acceptance/TimersCest' from the tests folder.

For more information on how to run the tests, follow tests/[README](#).

Development Methodology

Minutes

A document containing the team's chat (through Messenger) for this sprint is available in documentation/[Team Chat - Sprint 5.txt](#).

There was one meeting with the client this sprint:

Meeting on 4th of October to discuss the results of the hackathon event and discuss if any changes need to be made or if any new requirements are to be added. The minutes for this meeting can be found in documentation/minutes/[meeting_04_10_17.docx](#).

Burndown Chart (Task 47.2)

Note: This is a continuing sub-task for each sprint.

Estimated Time: 9.5 Hours

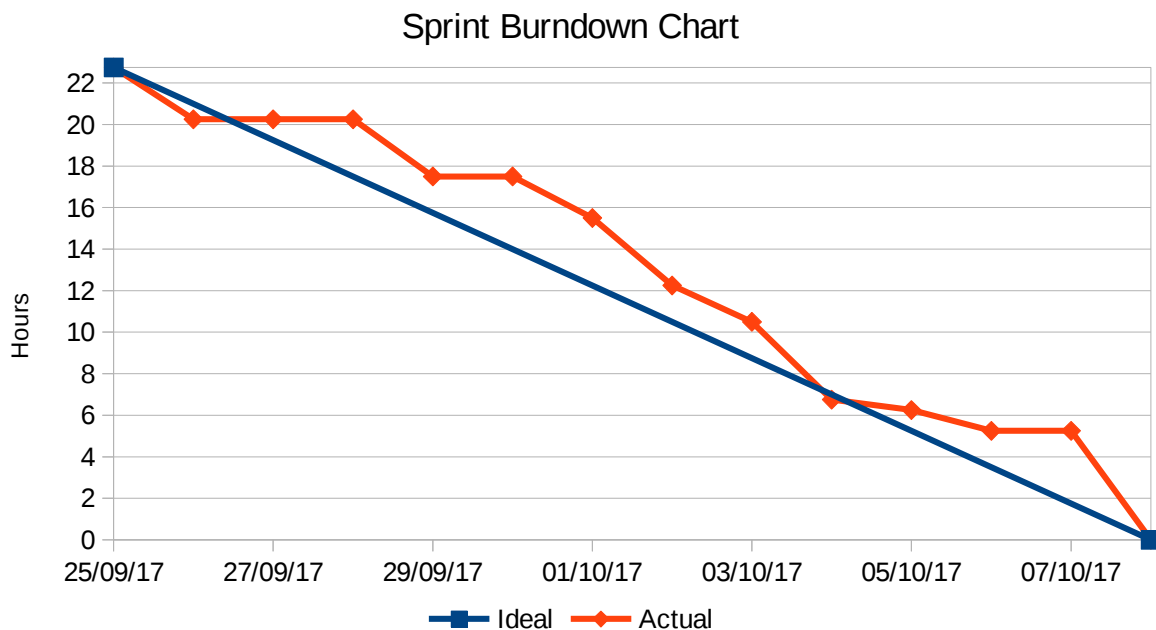
Actual Time: 8.5 Hours

Actual Time (this sprint): ~0.5 Hours

Description

Update sprint burn down charts.

Burndown Chart



Sprint Retrospective (Task 47.3)

Note: This is a continuing sub-task for each sprint.

Estimated Time:	3.5 Hours
Actual Time:	2.5 Hours
Actual Time (this sprint):	~0.5 Hours

Description

Reflect and record the sprint retrospective.

Sprint Retrospective

What went well during the sprint?

On balance the sprint went well. I was able to complete Task 37 fairly quickly and move on to the larger Task 52 early on in the sprint. I am happy overall with the results of Task 52. Going into the task I was concerned about the timers being specific to each user as opposed to a single global countdown for all users but it turned out to not as difficult as I originally thought.

Another thing that happened this sprint was the hackathon event. This was the first real world test of Mellivora and I think it went well.

What went wrong during the sprint?

There were a few problems encountered with Task 52, mainly the form on the manage_timers page in Task 52.1 and the TimersCest acceptance test created for testing the updating timers in Task 52.5. This resulted in a little more time being needed than originally planned. That being said I was able to find solutions to these problems to complete the tasks. It would have been nice to have had a better solution for testing the updating timer functionality.

What could we do differently to improve?

Once again, the estimates for Task 52 were a little low, more specifically for Task 52.5. I did spend more time on this than the estimate since the original version of the task didn't plan for actually updating the timer information on the page, only disabling the test when the timer ends. This is described in more detail in the implementation section of Task 52.5. These task's descriptions probably should have been updated by me before version 8 of the task allocation document (documentation/[Mellivora Task Allocation v8.xlsx](#)) was sent out.

Task Summary (Task 48.1)

Note: This is a continuing sub-task for each sprint.

Estimated Time: 28 Hours
 Actual Time: 20.5 Hours
 Actual Time (this sprint): ~3.5 Hours

Description

Write a summary of the tasks completed in the sprint.

Time Management

Sub-Task	Estimated (This Sprint)	Actual (This Sprint)
47.2	0.5	0.5
47.3	0.5	0.5
48.1	3	3.5
37.1	3	2.5
37.2	1	0.5
52.1	8	12.25
52.5	4	7.5
Total	16	22.75
Total (With Extra Tasks)	20	27.25

Note: The latest task allocation document can be viewed in documentation/[Mellivora Task Allocation v8.xlsx](#).

This sprint I continued making use of the Python tool named 'ti' (<https://github.com/sharat87/ti>) to keep track of time when starting, stopping and continuing work on any sub-tasks.

Task Selection

Task 13 has been selected for next sprint as it is one of the last remaining features left to be implemented. Currently this is the only task planned for next sprint.