4156 First Iteration Report

Team name: Falcon

Team members:
Weihan Chen wc2681
Zeyu Zhu zz2676
Xiaobin Lin xl2894
Yuyao Zhong yz3618

Part 1

Github link: https://github.com/YuyaoZhong/4156-team-project/tree/master/backend

Part 2 User Stories and Acceptance Testing

1. Pomodoro Timer:

User story

As a user, I want to have a pomodoro timer with a Zoom link attached so that I can pace myself and boost my work efficiency.

Conditions of satisfaction:

- Should be able to create a Pomodoro timer with options to select a start time, name, duration and description
- Should be required to select the start time again if he or she have chosen a past time point
- Should be able to generate a related Zoom link
- Should fail if a user create a timer which has overlaps with other existing timers
- Should see a message when a timer is created
- Successfully created timers can be seen as a schedule on the dashboard page
- A timer can start counting down at the configured time

Acceptance testing plan

(1) Let a user try to create a Pomodoro timer, with a title, start time, duration and description. The start time will be set to be a time starting within a few minutes.

Expected results to accept:

- The user can see a message that indicates the timer has been successfully created and he can review and update this timer at any time

(2) (Exception) Let a user create a Pomodoro timer which overlaps with an existing Pomodoro

Expect results to accept:
- The user will see a message that indicates there is an overlap and must select the start time and duration again.

Discussion

Considering that our front-end is not fully completed, so in this test we simulated the input from the front-end to the back-end by using postman.

For the test with valid input, the server will return a correct status code and all information about this newly created timer. Users can also update the settings of this timer at any time.

And for the test with invalid input, the server will return an error status code and a message that indicates there is an overlap and must select the start time and duration again. Initially we did not handle the problem of determining whether two timers overlapped very well(due to boundary issues), in this case, if the user wants to start a new timer immediately when a timer ends, it will be judged as an overlap situation. This bug has already been fixed after code updating

2.Tasks and task lists

User story

As a user, I want to create a task list so that I can have a holistic overview of what is left to do.
Conditions of satisfaction:
- Should be able to create a daily task list, weekly task list, or monthly task list.
- Should be able to add/delete tasks at any time.
- Can mark tasks as completed or incomplete.
- Can review all previous task lists.
- A task list should not contain the same tasks

Acceptance testing plan

(1) Let a user try to create a task list with task list id, task list name. We assume that this user has many tasks that he wants to add to this task list.

Expected results to accept:

- Users can add tasks successfully.
- Users can delete tasks successfully.
- Users can change the name of the existing task or task list.

(2) （Exception）Let the user try to add an existing task to the task list.

Expected results to accept:

- The user will see a message on the UI that indicates this operation is invalid because this task already exists.

Discussion

As we have not finished the front-end part yet, we need to use Postman to simulate the behavior of the front-end.

For the acceptance test, we simulate the behavior of adding/deleting tasks from task lists by putting an updated task to an existing task using API for tasks, and we simulate the behavior of changing existing task lists by using the APIs for task lists.

For the exception case, this part should be tested in the front-end, by checking if the existing task list has the same ID as the target task list, so we did not test anything other than get API for tasks for it.

3. Attach tasks to pomodoro timers

User story

As a user, I want to attach tasks to a pomodoro timer so that I can see the work planned to be done during the pomodoro sessions.

Conditions of satisfaction:

- The user can add a created task to a pomodoro timer
- The user can add a task to multiple pomodoro timers
- The user can add multiple tasks to a pomodoro timer
- The user can not add a task again if the task has been added to the timer

Acceptance testing plan

(1) Let a user try to attach an existing task to a pomodoro timer by clicking the “add” icon on a page of a pomodoro timer.

Expected results to accept:

- The user can see all the tasks has been created
- The user can add one of the tasks that have not been attached to the timer by clicking it
- The user can see the task being successfully added to the timer

(2) (Exception) Let a user try to add a task that has been added to a pomodoro timer by clicking it again

Expected results to accept:
- The user will see a message indicate that the task can not be added successfully and the lists of task attached to the timer remain to be the same

Discussion

For the first iteration, since the front-end has not been completely developed, we consider the acceptance tests from an api perspective. The user will call the apis for the data operation needed to be done, and the API will respond to the user requests.

The process of our designed tasks is:

1. Let the user create a relation between an existing timer and an existing task by calling the route '/task_timers' with POST method. The relation has not been recorded in the database. That is to say, a valid input should be a pair of task id and timer id that has not been recorded.

    - The user will see a code of 201 and a message indicates the relation has been successfully created.
    - The user will also see the complete data of the newly created relation.

2. Let the user try to retrieve all the attached tasks by giving a timer id as the request parameter by calling the route '/task_timers' with request parameters 'userId' and 'timerId'.

    - The user will see a code of 200 and a message indicates that the data has been successfully retrieved
    - The user will see a list of related tasks if there are tasks attached.

3. Let the user try to retrieve all the related timers by giving a task id as the request parameter by calling the route '/task_timers' with request parameters 'userId' and 'taskId'.

- The user will see a code of 200 and a message indicates that the data has been successfully retrieved
- The user will see a list of related timers if the task has been attached to any timers.

4. (Exception, a bug fixed) Let the user try to create the relation with the same task id and timer id again.

    - At first, the scenario has not been considered, and the relation can be created multiple times with different relation id. This will potentially cause problems.
    - The bug is fixed by adding a `UNIQUE` constraint to the databases.
    - The user will see a code of 500 and an error message that indicates the request can not been processed successfully

5. (Exception) Let the user try to create the relation between a non-existing task and an existing timer, or the relation between a non-existing timer and an existing task.

    - The exception is considered from the backend development perspective, since the frontend interface may not espouse non-existing task ids to the users. But the api can be mistakenly called by non-existing ids which will cause problems if these relations have been added to the database.
    - The user will see a code of 404 and an error message that indicates the requested resource have not been found and the request can not been processed successfully

6. (Exception, security) Let the user try to create the relation or retrieve related tasks and related timers by giving an unauthorized user id.
    - The exception is considered from the backend development perspective, in case of changing data unexpectedly with an unrelated user id.
    - The user will see a code of 401 and an error message that indicates the request can not be processed since authentication failed.

Part 3

Package manager

- We use **`pip`** for Python as a package manager. All the installed packages are recorded in the `requirements.txt`, and can be executed with command:

    `pip install -r requirements.txt`

Automated unit tests

- We use the `**unittest**` library for Python to automatically run the unit test. All the unit tests can be automatically run after giving the password in `app/config.py`
- There are two ways to run all the unit tests:

1. **With command**
   In the project directory (‘backend’), run the following command in the terminal:

   `python -m unittest discover tests`

   This will only run all the tests and output the report in the terminal without generating a file.

   Example output

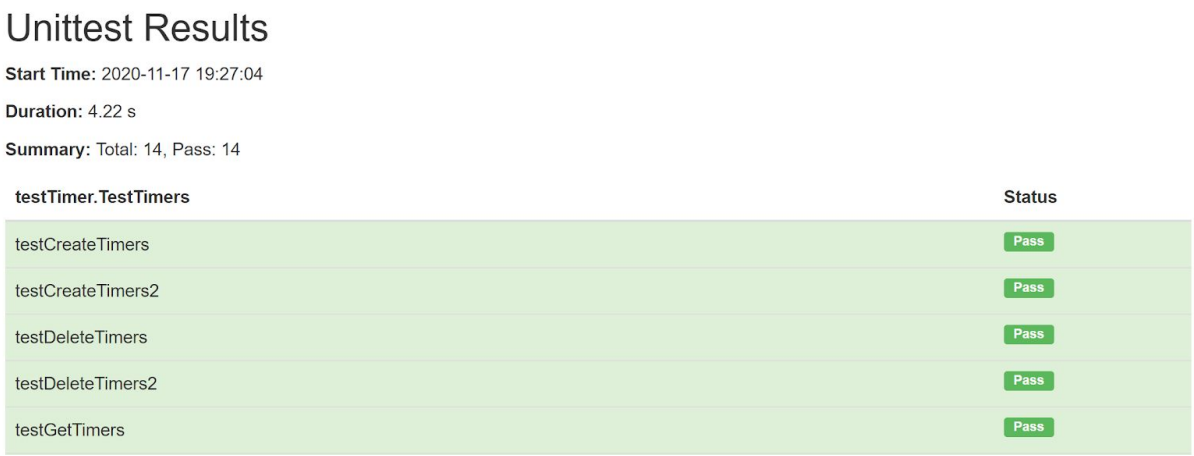
   ```
   ..............................
   ----------------------------------------------------------------------
   Ran 30 tests in 6.157s
   ```

2. **Use `.py` file written to automate tests and generate reports**
   In the project directory (‘backend’), run the following command in the terminal:

   `python runTests.py`

   This will only run all the tests and generate test reports with timestamps in directory ‘/backend/test_reports’.

   Example output



Unittest Results

Start Time: 2020-11-17 19:27:04

Duration: 4.22 s

Summary: Total: 14, Pass: 14

testTimer.TestTimers	Status
testCreateTimers	Pass
testCreateTimers2	Pass
testDeleteTimers	Pass
testDeleteTimers2	Pass
testGetTimers	Pass

Links

1. **Folder of all the test cases**
   https://github.com/YuyaoZhong/4156-team-project/tree/master/backend/tests

2. **Package manager**
   https://github.com/YuyaoZhong/4156-team-project/blob/master/backend/requirements.txt

3. **Automated unit tests**
   https://github.com/YuyaoZhong/4156-team-project/blob/master/backend/runTests.py

Part 4

We use PyLint as both a style checker and a bug finder. PyLint will report errors if finds a bug.

An old report with errors and warnings not completely fixed after implementing the `app` module:
https://github.com/YuyaoZhong/4156-team-project/blob/master/backend/pylint_old_report.html

An old report with errors and warnings not completely fixed after implementing unit tests:
https://github.com/YuyaoZhong/4156-team-project/blob/master/backend/pylint_old_report_tests.html

The latest report:
https://github.com/YuyaoZhong/4156-team-project/blob/master/backend/pylint_report.html

As shown below, all the warnings and errors are resolved.

Pylint report from report.jinja2

Messages

Metrics

Count per types

Name	Count

Count per messages

Name	Count

Count per modules

Name	Count

Count per path

Name	Count