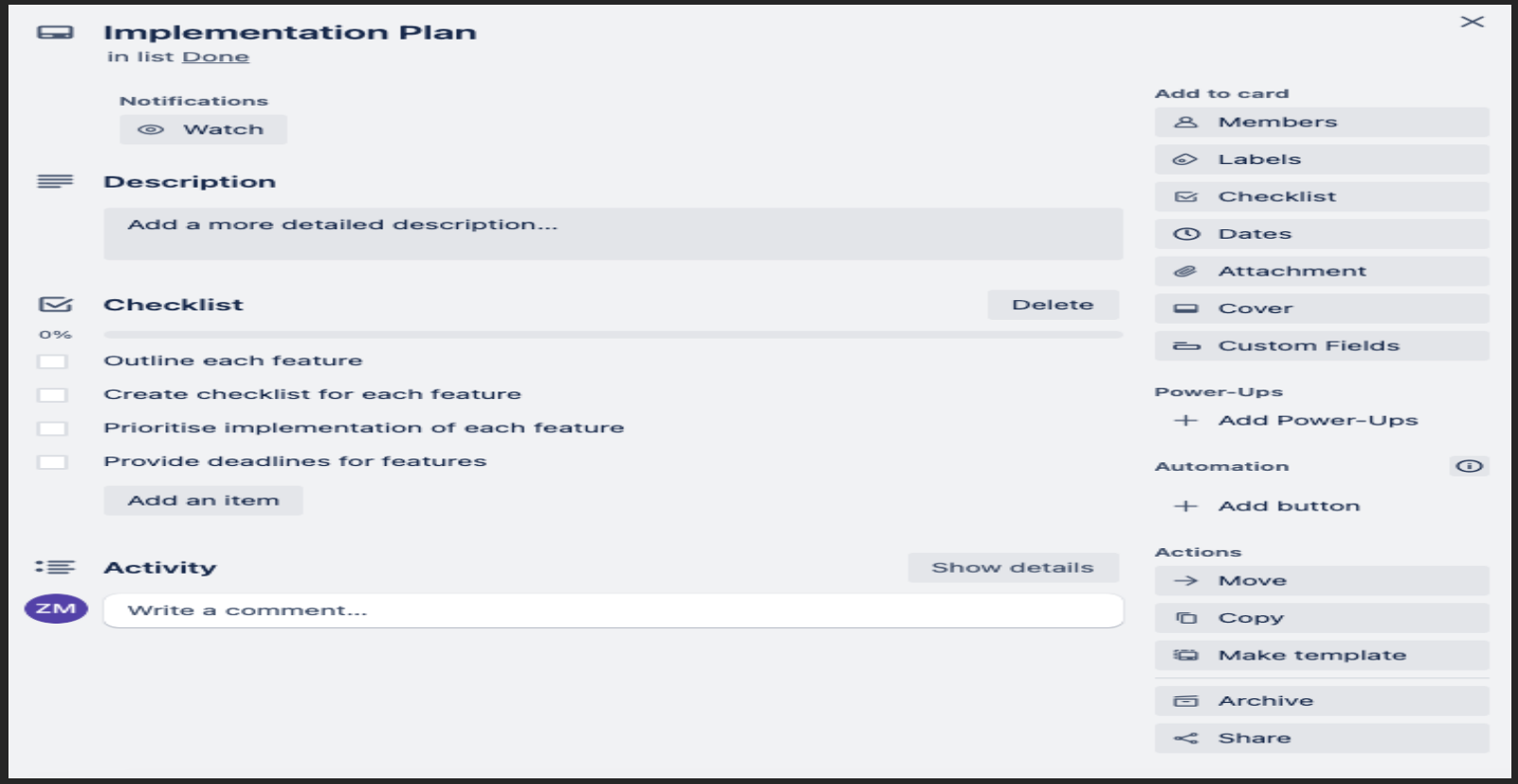
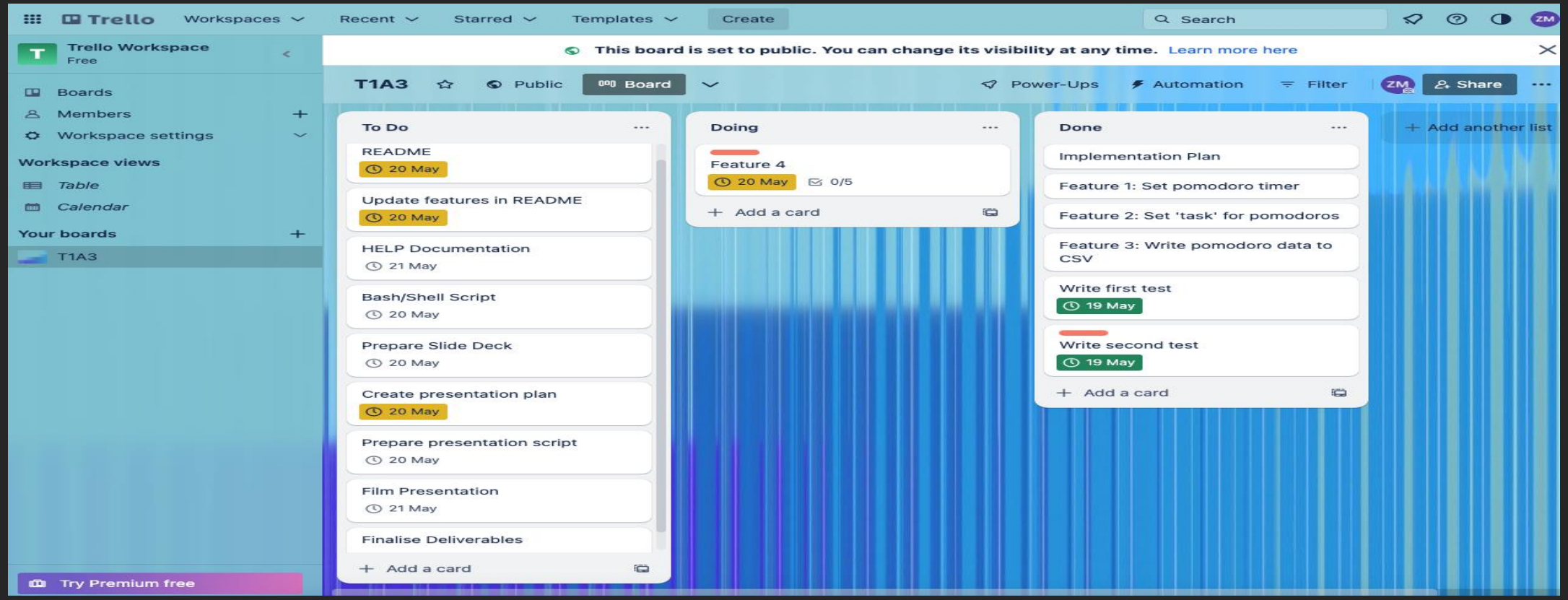
Pomodoro Application

ZAKERIYA MOHAMED T1A3

Implementation Plan



Project Management (Trello)



Overview

- The Pomodoro method involves working in set intervals, usually 25 minutes of focused work followed by a 5 minute break, although the time intervals are up to each individual
- It has been shown to improve focus and productivity
- I built a pomodoro app which functions in the terminal
- User can set work and break times, and is alerted to break/start times with sound notifications
- User can assign Pomodoro to 'tasks'
- Pomodoro data is written to a file, so user can view/manipulate with data visualisation tools

App Features

Features

- Set a cycle length
- Set a task name for pomodoro cycle
- Set work time and short break, long break time
- Alert user with sound alert when break/work time starts
- Write pomodoro data to csv file

App Functionality

User Input: Getting Pomodoro Settings

```python
def pomodoro_length():

    while True:
        print("Pomodoro timers involve short breaks and long breaks.\nA 'cycle' refers to how many pomodoros you complete before your lor
        print("For example a cycle length of '4' means you will have your 'long break' after your 4th pomodoro,\nand the first 3 pomodors
        cycle_length_input = input("Enter how many pomodoros you want in this 'cycle': ")
        if cycle_length_input == '':
            print("You must enter a number for your 'cycle length'.")
        elif not cycle_length_input.isdigit():
            print("Invalid input for cycle length, please enter a number.")
        else:
            cycle_length = int(cycle_length_input)
            break

    while True:
        work_time_in_minutes = input('Enter how many minutes you would like to work?: ')
        if work_time_in_minutes == '':
            print("You must enter how many minutes you would like to work.")
        elif not work_time_in_minutes.isdigit():
            print("Invalid input for work time, please enter time in minutes.")
        else:
            work_time = int(work_time_in_minutes)
            break

    while True:
        short_break_time_in_minutes = input('Enter how many minutes you would like to rest for your short break?: ')
        if short_break_time_in_minutes == '':
            print("You must enter how many minutes you would like to rest for your short break.")
        elif not short_break_time_in_minutes.isdigit():
            print("Invalid input for short break time, please enter time in minutes.")
        else:
            short_break_time = int(short_break_time_in_minutes)
            break

    while True:
        long_break_time_in_minutes = input('Enter how many minutes you would like to rest for your long break?: ')
        if long_break_time_in_minutes == '':
            print("You must enter how many minutes you would like to rest for your long break.")
        elif not long_break_time_in_minutes.isdigit():
            print("Invalid input for long break time, please enter time in minutes.")
        else:
            long_break_time = int(long_break_time_in_minutes)
            break

    # Return the work and break times for future reference
    return cycle_length, work_time, short_break_time, long_break_time
```

Displaying Timer and Playing Sound Alerts

```python
import time
from playsound import playsound
import os

# Need absolute path for my audio files since my bash script is not in this directory
AUDIO_BASE_DIR = os.path.dirname(os.path.abspath(__file__))


def run_pomodoro(cycle_length, work_time, short_break_time, long_break_time):
    # This function takes cycle length, work time, short break time, and long break time as arguments
    # For each pomodoro cycle, it starts a work timer, then a short break timer
    # This is repeated until we reach pomodoro #cycle length, then long break timer is started
    # Each stage is signaled by a unique audio alert

    for _ in range(cycle_length):
        print("Work timer started for", work_time, "minutes")
        countdown(work_time * 60) # Run countdown for work time
        playsound(os.path.join(AUDIO_BASE_DIR, 'break-start.wav')) # Play alert at start of short break

        print("\nShort break timer started for", short_break_time, "minutes")
        countdown(short_break_time * 60) # Run countdown for short break time
        playsound(os.path.join(AUDIO_BASE_DIR, 'pomodoro-start.wav')) # Play alert at start of pomodoro

    print("\nLong break timer started for", long_break_time, "minutes")
    countdown(long_break_time * 60) # Run countdown for long break time
    playsound(os.path.join(AUDIO_BASE_DIR, 'end-of-long-break.wav')) # Play alert at start of long break (last pomodoro)

def countdown(t):
    #This function takes an amount of time in seconds and counts down to zero
    #It prints the remaining time every second
    while t:
        # While loop runs as long as t (time in seconds) > 0
        mins, secs = divmod(t, 60) #Divmod splits total seconds into mins and remaining secs and formats to give our timer
        timer = '{:02d}:{:02d}'.format(mins, secs) # Formats mins and secs as 2-digit numbers
        print(timer, end="\r") # Prints timer to console, replacing prev line with end="\r" every second for live countdown
        # Print statement does this by bringing cursor to start of line, therefore overwriting prev timer line
        time.sleep(1) # Pause for 1 second for each iteration
        t -= 1 #Decrement remaining time by 1 second
```

Writing Pomodoro Data To CSV

```python
import os
import csv

def write_to_csv(task_name, pomodoro):
    # CSV file headers
    headers = ['Task Name', 'Start Time', 'End Time', 'Duration']
    # Check if the CSV file already exists
    file_exists = os.path.isfile('pomodoros.csv')
    # Now, write the pomodoro to a CSV file
    with open('pomodoros.csv', 'a', newline='') as file:
        writer = csv.writer(file)
        # If file doesn't already exist (first pomodoro), write headers first
        if not file_exists:
            writer.writerow(headers)
        # Write Pomodoro data to CSV
        writer.writerow([task_name, pomodoro['start_time'], pomodoro['end_time'], f"{pomodoro['duration']} min"])

```

Creating, Assigning, Executing, and Writing Pomodoro

```python
from pomodoro_length import pomodoro_length
import datetime
from run_pomodoro import run_pomodoro
from write_to_csv import write_to_csv



pomodoros = []
tasks = []


def start():
    try:
        while True:
          task_name = input('Enter the name of the task this pomodoro is for: ')
          if task_name:
              break
          else:
              print("You must enter a task name. Please try again.")
        cycle_length, work_time, short_break_time, long_break_time = pomodoro_length()  # my pomodoro as defined by pom

        pomodoro = {
            'start_time': datetime.datetime.now().replace(microsecond=0),
            'end_time': datetime.datetime.now().replace(microsecond=0) + datetime.timedelta(minutes=work_time),
            'duration': work_time,
            'task': task_name,
        }
        pomodoros.append(pomodoro)

        # Check if the task already exists
        task_exists = False
        for task in tasks:
            if task['name'] == task_name:
                task['pomodoros'].append(pomodoro)
                task_exists = True
                break

        # If the task doesn't exist, create a new one
        if not task_exists:
            task = {
                'name': task_name,
                'pomodoros': [pomodoro],
            }
            tasks.append(task)

        # WRITING TO CSV
        write_to_csv(task_name, pomodoro)
```

```python
        # WRITING TO CSV
        write_to_csv(task_name, pomodoro)

        # Call the run_pomodoro function below starts the pomodoro
        # Number of pomodoros it runs will be equal to cycle length, with the long break after the nth (cycle_length) pomodoro
        # Start of work work time, short break, and long break will be signalled by unique alert sounds
        run_pomodoro(cycle_length, work_time, short_break_time, long_break_time)



    except KeyboardInterrupt:
        # If the user interrupts the program, print message below and record the current pomodoro to CSV file
        print("\nInterrupted. Writing current pomodoro to file...")
        # Writing pomodoro to CSV file
        write_to_csv(task_name, pomodoro)
```

Running Application

```
1    from start_pomodoro import start
2
3    # Run application
4    start()
5
```

Tests

Test Dependencies

```
import unittest
from unittest.mock import patch, mock_open
from pomodoro_length import pomodoro_length
from start_pomodoro import start
```

Timer Functionality Test

```
class TestPomodoroApp(unittest.TestCase):
    # test that my pomodoro_length function correctly handles user input
    def test_pomodoro_length(self):
        with patch('builtins.input', side_effect=['4', '25', '5', '15']):
            cycle_length, work_time, short_break_time, long_break_time = pomodoro_length()
            self.assertEqual(cycle_length, 4) # checking if cycle_length is correct
            self.assertEqual(work_time, 25) # checking if work_time is correct
            self.assertEqual(short_break_time, 5) # checking if short_break_time is correct
            self.assertEqual(long_break_time, 15) # checking if long_break_time is correct
    """
```

File-Writing Test

```
    """
    My second test checks that my start funciton correctly writes something to csv.
    I can't check the exact timestamps are 'corect' because assert_called_once_with doesn't support regex
    But I am happy in checking that;
      > the headers and pomodoro are written correctly
      > the string that the write method is called with starts with the task name, and has the structure we expect
    """
    @patch('start_pomodoro.run_pomodoro')
    @patch('builtins.input')
    @patch('builtins.open', new_callable=mock_open)
    @patch('os.path.isfile')
    def test_write_pomodoro_to_csv(self, isfile_mock, mock_open, input_mock, run_pomodoro_mock):
        # Define the behavior of the mocks
        input_mock.side_effect = ['test_task', '4', '25', '5', '15']
        run_pomodoro_mock.return_value = None
        isfile_mock.return_value = False  # mock isfile should return False, to simulate non-existent csv file (first pomodoro)


        # Call the function to test
        start()

        # Get all calls to write
        write_calls = mock_open().write.call_args_list

        # The first call should be writing the headers
        call_args = write_calls[0][0][0]
        print(call_args)
        # Check that first call writes headers, or statement for checking newline characters in unix-based systems vs windows
        self.assertTrue(call_args == 'Task Name,Start Time,End Time,Duration\n' or call_args == 'Task Name,Start Time,End Time,Duration\r\n')

        # The second call should be writing the pomodoro
        call_args = write_calls[1][0][0]
        self.assertTrue(call_args.startswith('test_task,'))
        self.assertTrue(call_args.endswith('\n'))


if __name__ == '__main__':
    unittest.main()
```

Sources

[Playsound library](https://pypi.org/project/playsound/)

[Python 'datetime' module](https://docs.python.org/3/library/datetime.html)

[Python 'time' module](https://docs.python.org/3/library/time.html)

[Python 'os' module](https://docs.python.org/3/library/os.html)

[Python 'unittest' framework](https://docs.python.org/3/library/unittest.html)

[Python unittest.mock](https://docs.python.org/3/library/unittest.mock.html)

[Understanding patching and mocking](https://medium.com/geekculture/right-way-to-test-mock-and-patch-in-python-b02138fc5040)

[Getting absolute path in Python](https://docs.python.org/3/library/os.path.html)

[Python divmod function](https://www.programiz.com/python-programming/methods/built-in/divmod)

Challenges

- Formatting information in a useful way, I wanted to incorporate data visualisation but left that out of the assignment, will be implementing it in the future
- Had difficulty writing pomodoro data and visualing with pandas, matplotlib (error codes to the moon)

Favourite parts

- Familiarising myself with some of Python's built-in modules like os, datetime etc
- Learning about mocking 'user input' with Python's unittest.mock library
- No CSS 
- My favourite aspect was getting familiar with the philosophy behind test driven development

Application Demonstration