IFCS2 Formative 1

Initial Idea

The plan was to make a to do list with a front end using streamlit, where the user can manage their tasks by adding, modifying and deleting. Besides this, it's important to be able to see all the tasks, but also save the created task lists. This resulted in a drafted idea of the following functions: add_task, complete_task, delete_task, change_status_task, view_tasks, export_tasks.

I opted for streamlit came over the fact that tkinter can be quite monotonous to utilise and doesn't demonstrate a high level of reusability, since all buttons must be created individually as an example. Streamlit is supported by a much neater looking GUI interface, and using the feedback from the last IFCS module, I remembered the comments surrounding hosting of programs to avoid user self-hosting.

Design Choices

With all of the initial sketches complete, and understanding of how all the different sections interlink, I began to create a skeleton structure in my code with all the different functions as seen below:

```
def add_task(task_list: list, new_task: Task) -> list:
    # insert code here
    return task_list
```

This skeleton code allowed me to understand all the inputs and outputs for each function, before inserting the appropriate user messages throughout the functions, resulting in a user-oriented design even before implementing any functionality. With the original idea using a logs file to communicate with the user, this was quickly replaced with streamlit's inbuilt functionality to show colour coded error messages on the screen, another advantage of using streamlit over tkinter or similar GUI applications.

Originally, I had a series of buttons that would call certain functions, but this looked rather messy considering the buttons sprawling across the page, and to make this as user friendly as possible, I used the streamlit sidebar to contain all the different actions that a user could take.

With the user loading up the app for the first time, there are two options where the user could be a new or returning user. If they're a returning user, then it's likely that they'll want to open up an existing list of tasks that they created before, and to implement this, I created a button that displays the available list of task list files in a drop-down menu for the user to select from. Since this functionality is found in streamlit, then it means that it comes with perks such as being able to collapse the side bar.

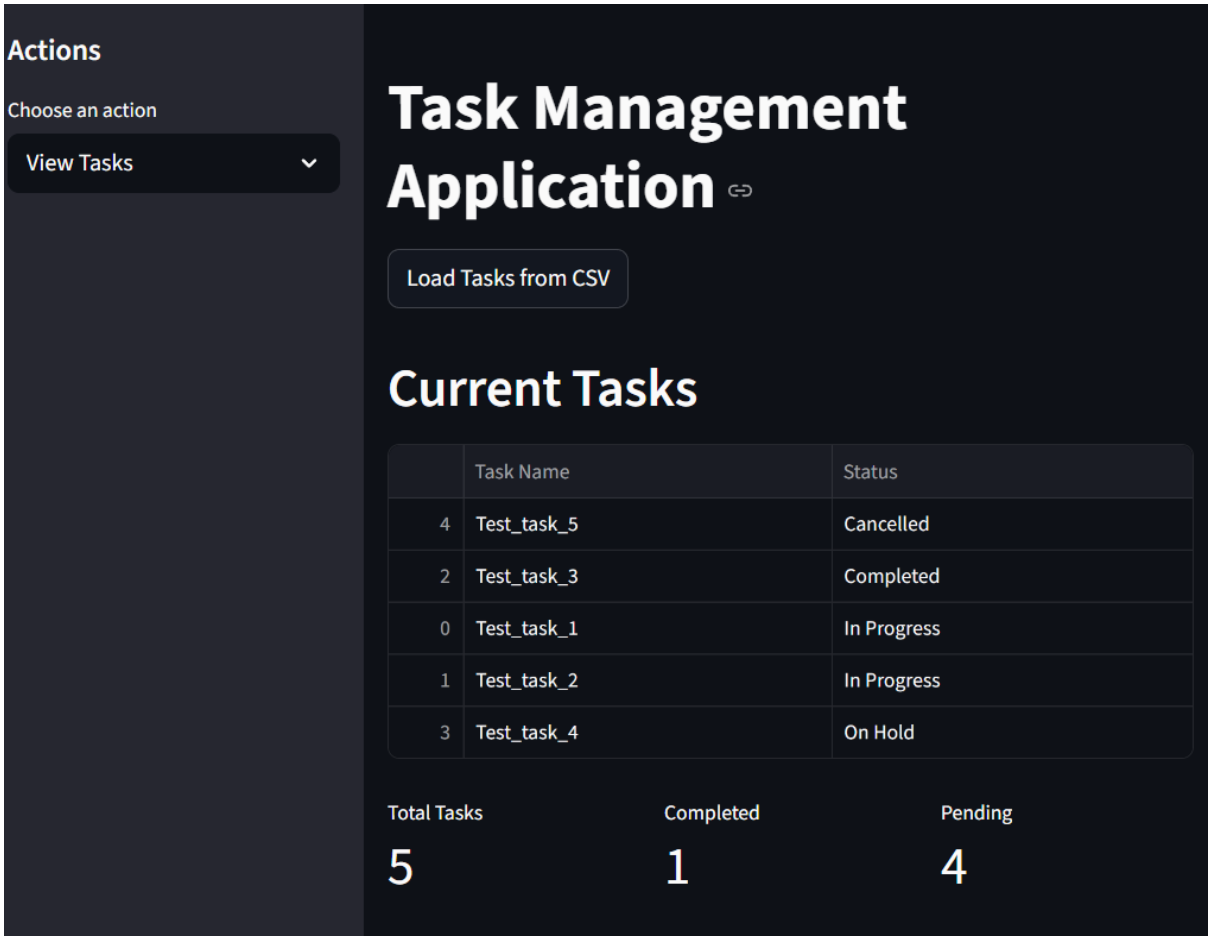
App development process

This was a fairly simple app development process which began with filling out the functions, and ensuring that they suited their purpose. Then I followed up by creating a main function that brought all of these functions together as a wrapper function, as streamlit would come on to need this. Nearing the final stages, I was left with a program that was able to perform the

desired functionality given a user's input, and all that was left was inserting the streamlit code, to combine all of this together.

There was one hurdle that I came across, upon realisation that I could only save my task lists, and once it was saved, I could never update them with my new set of tasks. As a result, I created the 'import' tab in the action menu to be able to save progress and continually work on the same task list or even have multiple task lists.

Demo use case



Where the tasks are automatically sorted by status, alphabetically. But using streamlit, it's possible to sort by any of the columns, in ascending or descending order, combined with the ability to pin/hide columns.

After exporting the table above, I get the csv below (opened in Dataspell)