Cover Page

Daily Use Reminder

https://github.com/NKNKINU/DailyUseReminder

Group#
Name of Student Liu, James
Name of Student Zhang, Jalen
Name of Student Zhu, Eric

Date of Submission: 08/14/2022

Table of Work

(Please write x in the boxes to mention what each student achieved in this project)

	Student-1 James	Student-2 Jalen	Student-3 Eric
Project Description			X
Uses Cases Diagram(s)	X	X	X
Sequence Diagrams	X	X	X
Class diagram(s)	X	X	X
Implementation	X	X	
Conclusion	X	X	X

Table of Contents

- Terminology Glossary (If needed)
- System Analysis
    - Project Description (One Page)
        - General Description, Goals and Benefits
        - System input(s) and output(s)
        - Special requirements (Performance, Interfaces, Constraints, Reliability, if any)
    - Uses Cases Diagram(s) and use cases description.
- System Design
    - Sequence Diagrams
    - Class diagram(s)
- Conclusion
- Appendix (any related reports, questionnaires, docs.. If any).

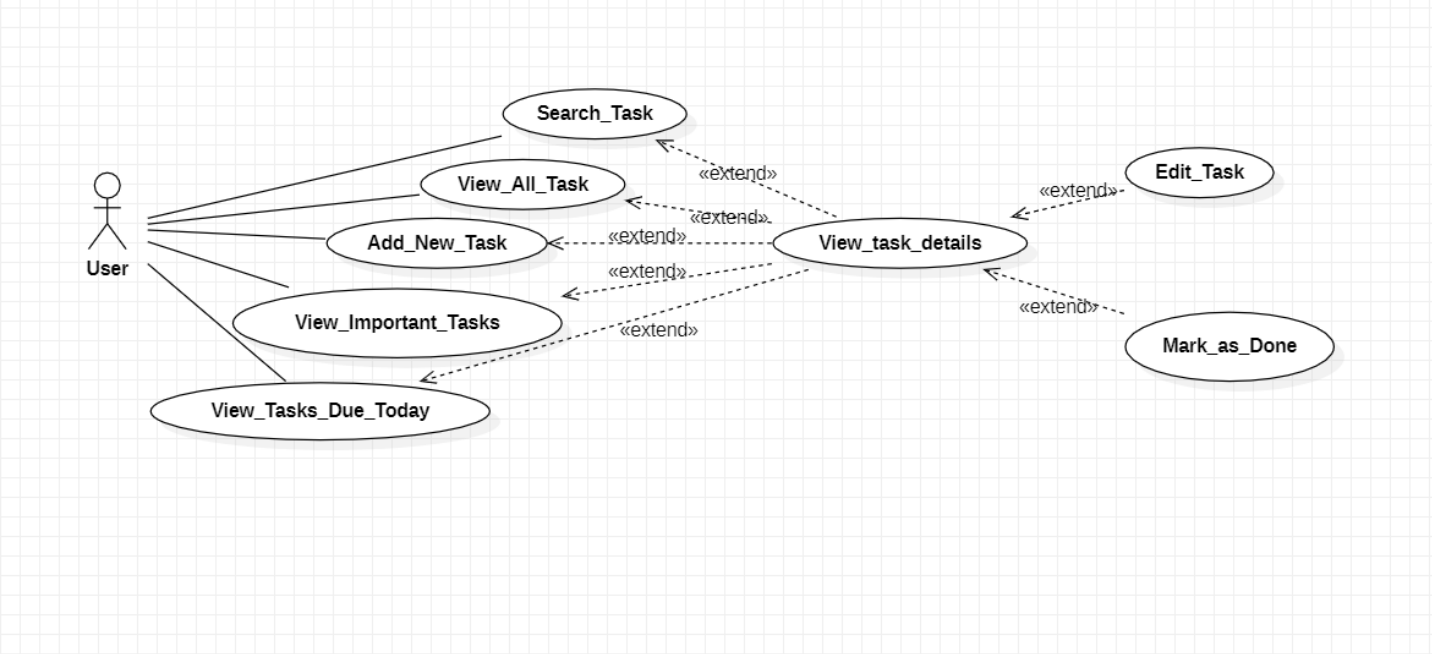
Description:

Nowadays, people encounter so many different events in their daily lives that it is difficult to recall all their tasks that must be done by a specific time. We were successful in creating an extendable, effective, and user-friendly reminder for our project. It is crucial to categorize and arrange our everyday tasks utilizing the reminder in order to assist us in managing our future plan. For instance, students are constantly concerned by the numerous homework due dates that follow one another, in addition to the need that they arrive at class promptly on weekdays. At this point, have a great schedule to determine what to do first becomes extremely significant.

Rewinding to the software itself, our objective is to provide a useful and simple reminder. There are a number of options on our home page, including Add New Task, View All Task, Tasks Due Today, Important Tasks, Search Task, and Exit. Users will be prompted to submit a few inputs when they attempt to create or add a new task by clicking the Add New Task button. Users merely need to provide the task's name, notes and its due date (time, day, month, year). Additionally, the users will notice an importance checkbox on the add task page, allowing them whether to click the checkbox depending on its priority. Clicking the checkbox, users will add this specific task into an important list, where users can find in the Important Tasks.

After creating several tasks, users may check all of them by clicking the **View All Task** button on the home page. The **Tasks Due Today** button will take users to a list of tasks that must be completed by today. **Important tasks** can be found by clicking the Important Tasks button. Lastly, users can **search a specific task** by utilizing the Search Task button. Users may click on a **specific task** in each task list (Tasks Due Today, Important Tasks, View All Task, and Search Task) to access detailed information. There is also an **Edit button** that allows user to edit the task. User can remove a task by clicking the **Is Done button** after they finished a task. There is refresh button on each task list, after clicking the refresh button, all the edit users do will be implemented, if users pressed the refresh button after clicking the Is Done button for a task, this task will be deleted and will not be display in any task list.

Uses Cases Diagram:



Use Cases Description:

UC Reference Name/Number:	Add_New_Task
Overview	add a new task
Related use cases:	None
Actors	User
Pre Conditions (Optional)	None
Post Conditions (Optional)	A new task will be created

UC Reference Name/Number:	View_All_Tasks
Overview	show all-task list
Related use cases:	View_task_details
Actors	User
Pre Conditions (Optional)	None
Post Conditions (Optional)	A list with all tasks will be displayed.

UC Reference Name/Number:	View_Tasks_Due_Today
Overview	show a list of tasks due today
Related use cases:	View_task_details
Actors	User
Pre Conditions (Optional)	None
Post Conditions (Optional)	A list with all today's tasks will be displayed.

UC Reference Name/Number:	Search_Task
Overview	search for a specific task
Related use cases:	View_task_details
Actors	User
Pre Conditions (Optional)	None
Post Conditions (Optional)	The specific task user looks for will be displayed.

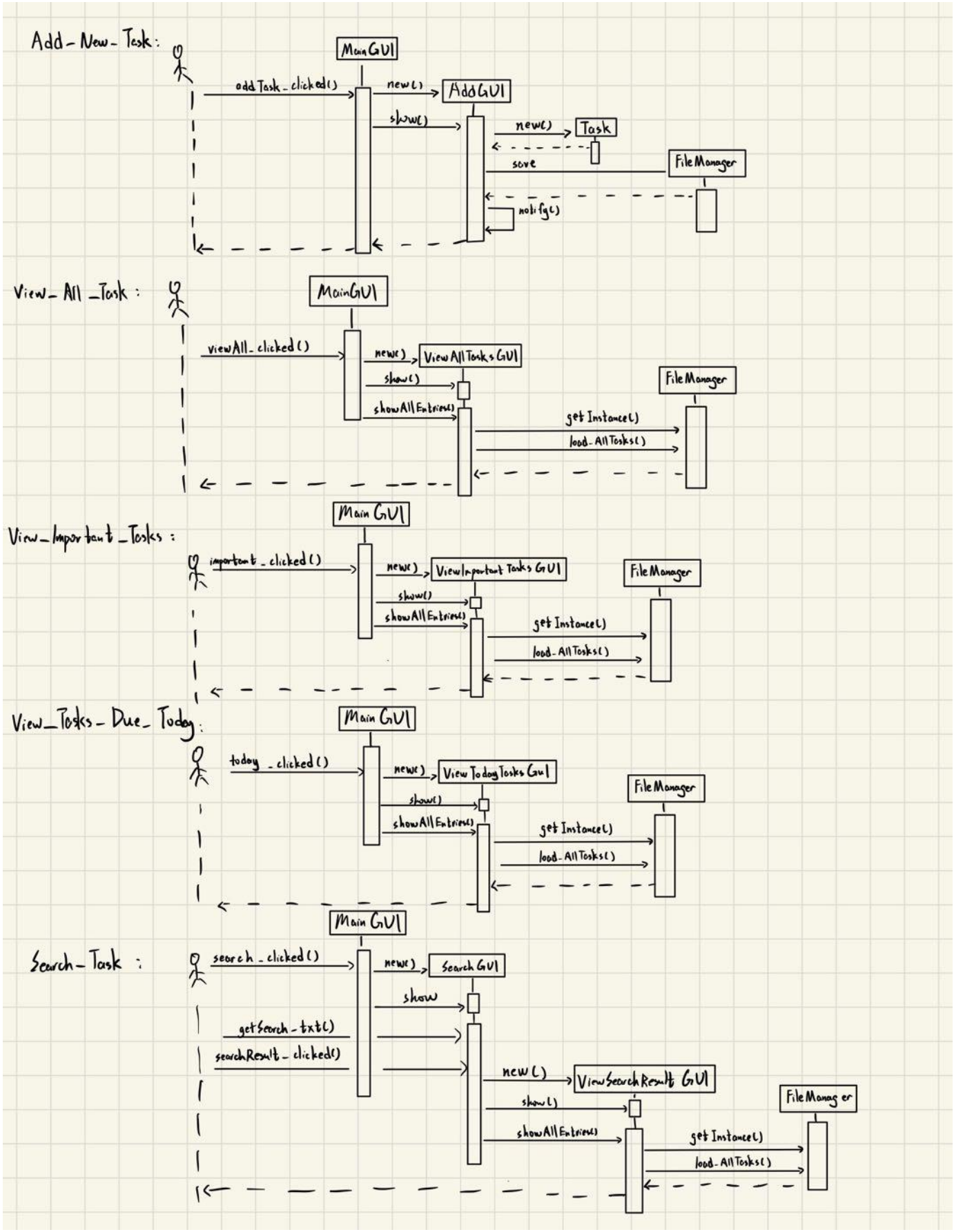
UC Reference Name/Number:	View_task_details
Overview	show a specific task, user can make more operations on this page.
Related use cases:	Edit_Task
Actors	User
Pre Conditions (Optional)	Tasks need to be already created.
Post Conditions (Optional)	A task will be displayed, more operations can be performed.

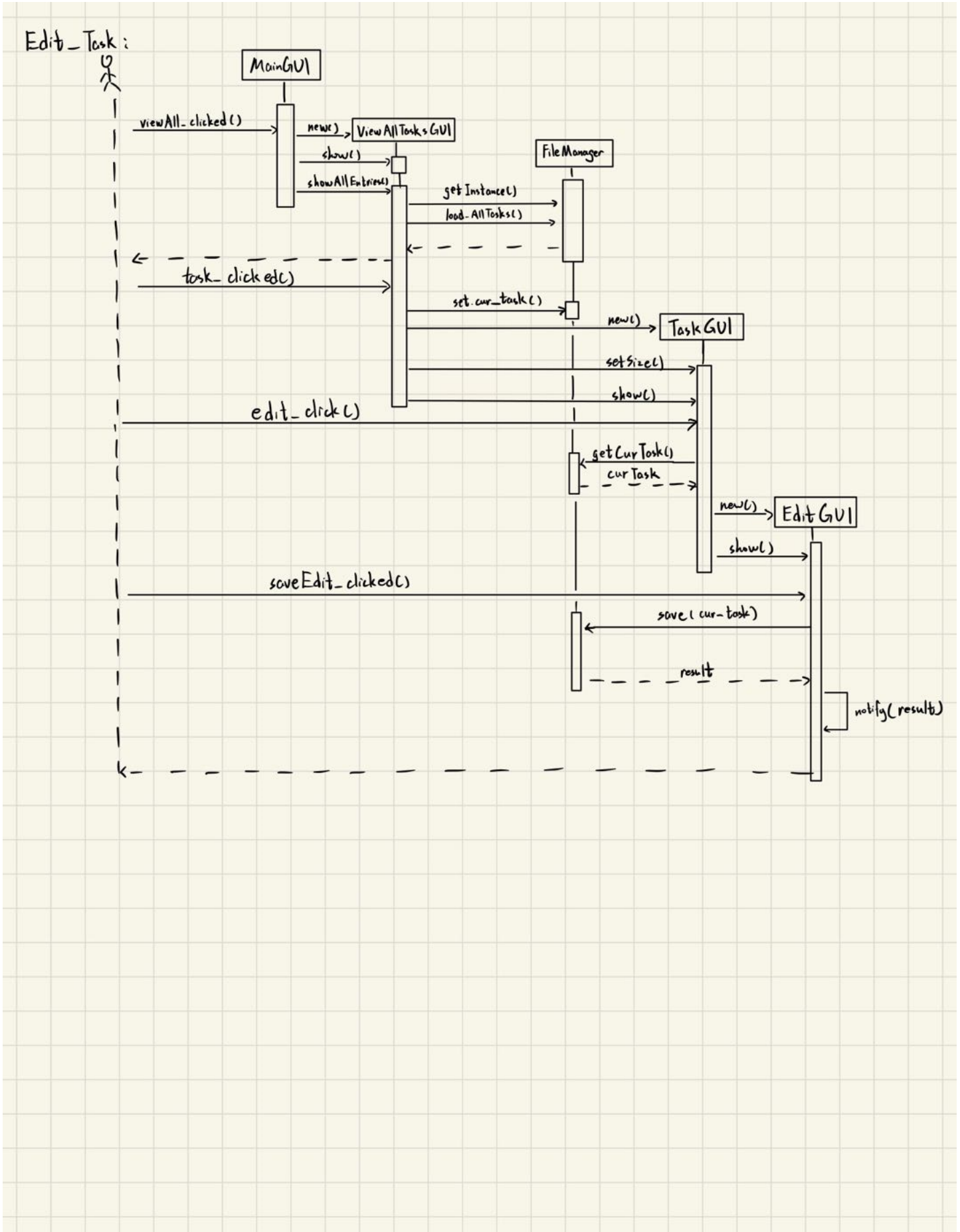
UC Reference Name/Number:	View_Important_Tasks
Overview	show a list of tasks that are marked as important
Related use cases:	View_task_details
Actors	User
Pre Conditions (Optional)	None
Post Conditions (Optional)	A list of tasks that are marked as important will be displayed.

UC Reference Name/Number:	Edit_Task
Overview	edit a specific task.
Related use cases:	View_task_details
Actors	User
Pre Conditions (Optional)	tasks need to be already created
Post Conditions (Optional)	the specific task can be editted, such as name or date can be changed.

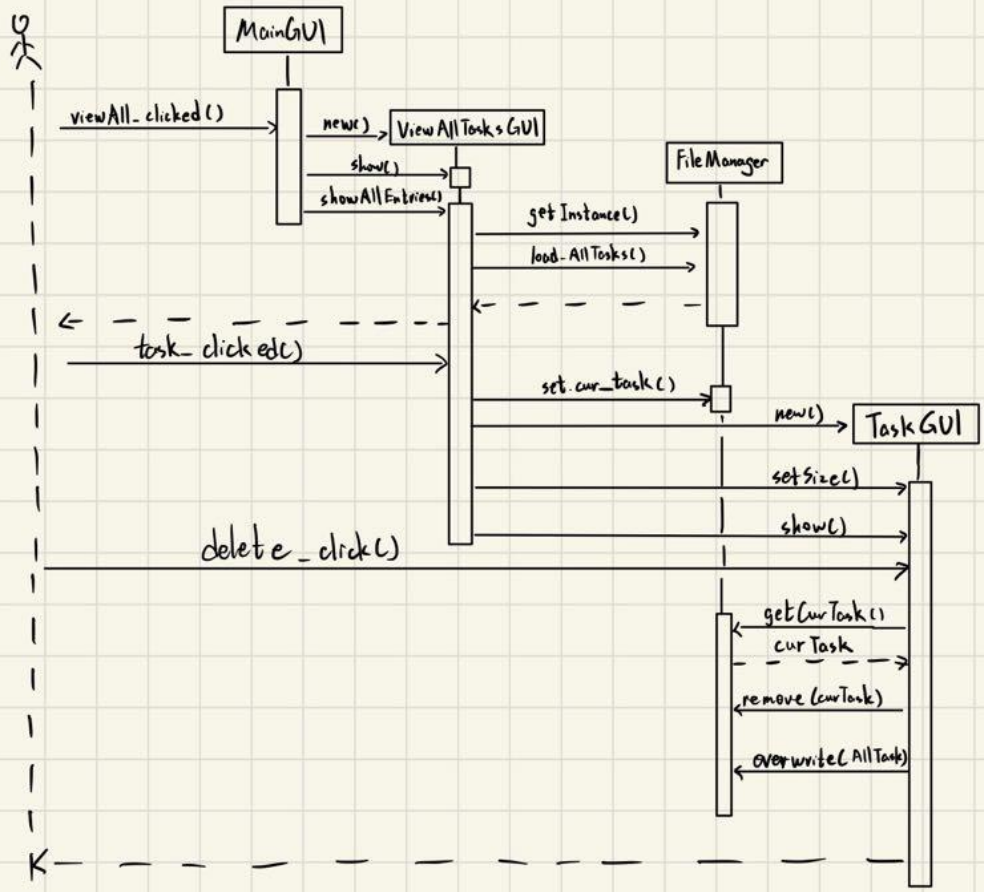
UC Reference Name/Number:	Mark_as_Done
Overview	Mark a task as done and delete it.
Related use cases:	View_task_details
Actors	User.
Pre Conditions (Optional)	The task must be existed.
Post Conditions (Optional)	The task will be deleted from all task lists

Sequence Diagrams:

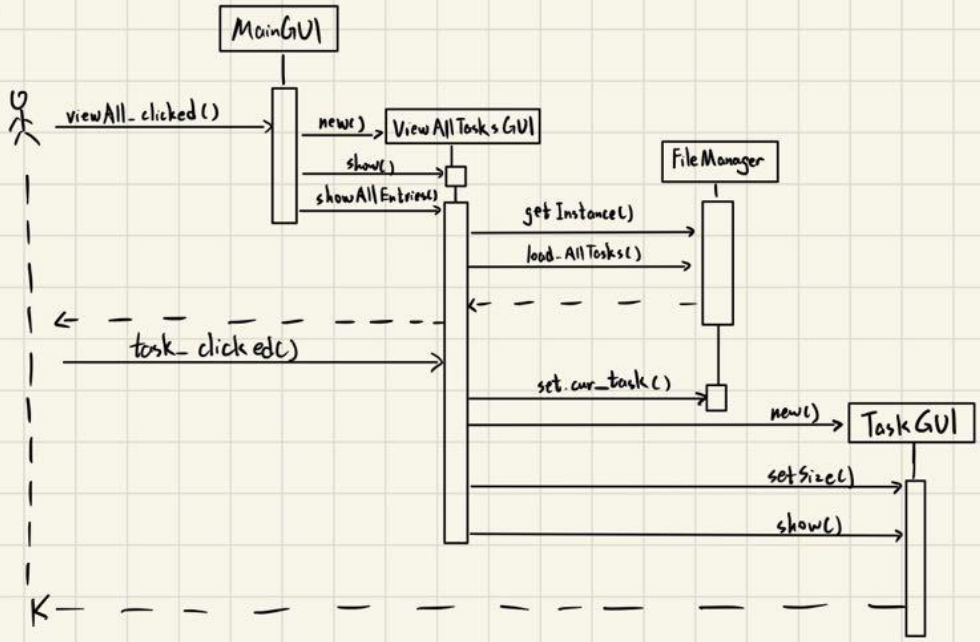



Edit_Task :
MainGUI
viewAll_clicked()
new()
ViewAllTasksGUI
show()
showAllEntries()
FileManager
getInstance()
load_AllTasks()
task_clicked()
set_cur_task()
new()
TaskGUI
setSize()
show()
edit_click()
getCurTask()
curTask
new()
EditGUI
show()
saveEdit_clicked()
save(cur_task)
result
notify(result)

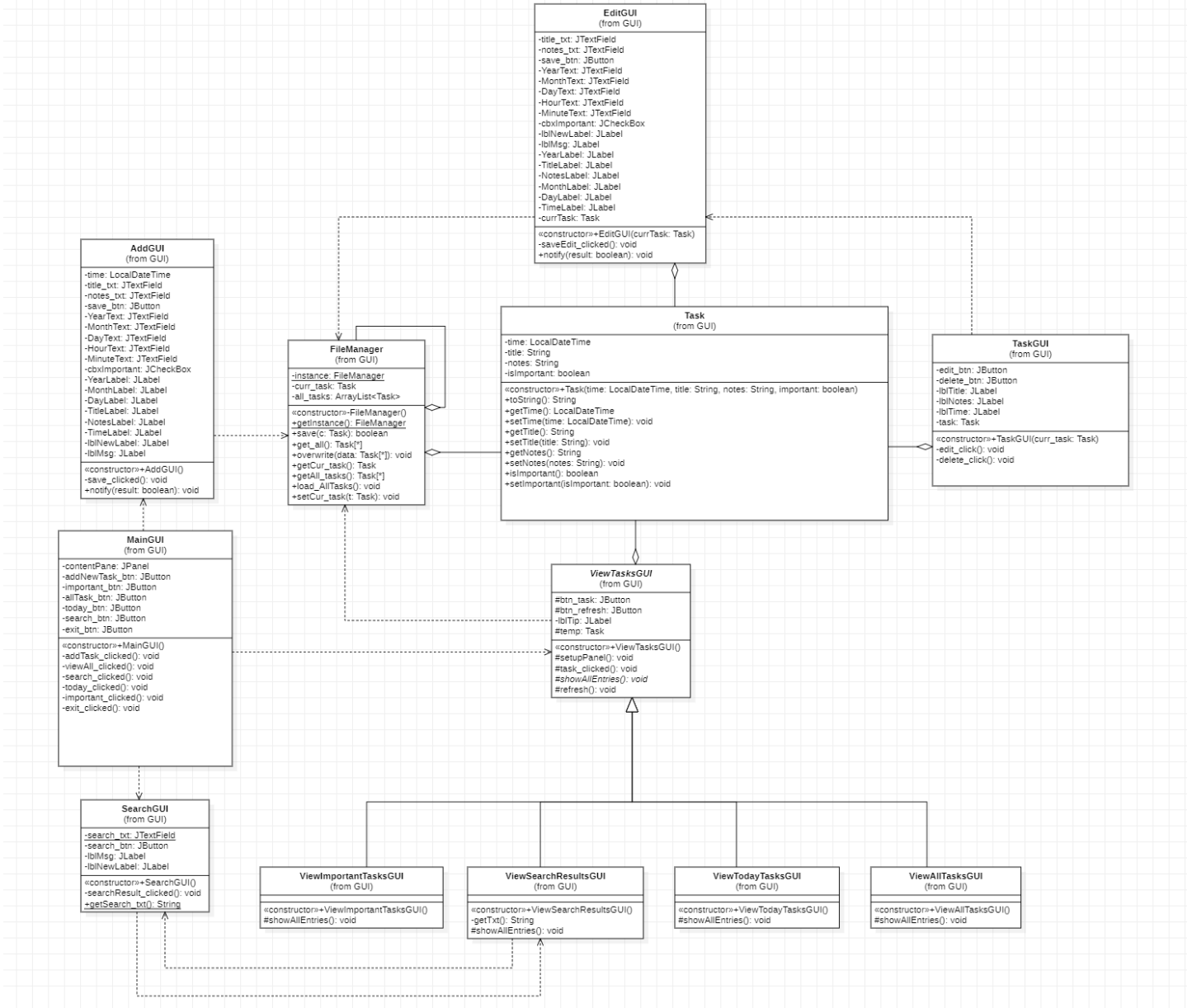
Mark_as_Done:



View_task_details:



Class diagram:



Conclusion:

We implement **singleton design pattern** for our FileManager class. By doing so, the filemanager can be globally accessed by each class and makes sure that only one class instance is created. We are able to reuse the instance throughout the project rather than initializing one everytime.

During the final phase of this project, we noticed that making AddGUI and EditGUI extend an abstract class, for instance, ModifyGUI, can make our class diagram easier to understand and also simplify the code. However, there was no enough time for us to do the implementation.

One thing that this project taught us is to have an insight of how your program is structured before you start working on the implementation. A scalable, robust program comes from deep understanding of the usage of each class, how they are connected, and the sequence of methods. This hand-on experience helps us reinforce concepts in object-oriented-programming. We found ourselves more confident in differencing relationships such as association, aggregation, composition, and dependency. We tried to use the concepts such as inheritance and design pattern to improve redundant code.

During the process, we learn plenty of knowledges and have a great experience; the software that our team developed has the fundamental functionality of a reminder and may actually be quite helpful. Despite our efforts, there is still much space for improvement. For instance, we were thinking that strategy mode can be a good implementation in our “view task”. Strategy pattern can give our program flexibility in switching different algorithm, which in our case are the different ways of showing tasks in real time. We throughout that it can be a very good fit as an improvement of structure and opens to future extension.