Getting things done…fast!

Project Manual

TEAM FINI

		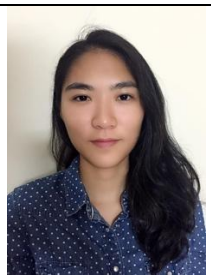	
Venkatesan Harish Team lead, Deliverables-in-charge	Tan Ngee Joel Jonas Integration-in-charge, Scheduling & Tracking Expert	Yu Qiyun Documentation & Code Quality Expert	Wang Jie Testing & Tools Expert

Table of Contents

USER GUIDE

DEVELOPER GUIDE

APPENDICES

FINI USER GUIDE

1 | About FINI

FINI is a free and simple user-friendly software that aims to solve one of the most common dilemmas we face daily: Task Management. In the context of our modern world, we are often swarmed with many tasks and deadlines from work, activities with friends and events that we cannot afford to miss. In order to successfully complete every work assignment you have and be the life of the party at every event you attend, you need to manage your priorities effectively and efficiently. How is that possible you may ask? FINI is the answer.

With FINI, you will be able to organize your tasks and accomplish them efficiently. That, is the very meaning of FINI, which translates to "finished" or "accomplished" in French.

2 | Getting Started

From assignments, to events, changing your child's diapers to groceries. These tasks may be filling up your mind but all that can wait until you learn how to utilize FINI.

FINI has been designed to be a personal productivity assistant who can help to keep track of your various tasks. If you are new to FINI, the following guide will help you to get started.

2.1 Launching FINI

To launch FINI, simply double-click on the FINI icon in the folder where FINI is saved. When FINI is launched, you will be greeted by the welcome screen as shown in *Appendix A, Figure 1.*

2.2 Adding Tasks

Adding tasks in FINI is extremely simple. The following command shows how you can specify a task with its attributes:

```
>> add <TASK_TITLE> // <12/10/2015> from <7pm> to <9pm> with priority
<high>
```

Please note that the parameters given between "< >" can be substituted with your own. In example, the priority parameter can take values of "high", "medium", "normal" or "low".
*The attributes you include when adding the task will also determine the task type.
**The next section of this manual elaborates more on what these "task types" are.

Floating Tasks

FINI supports different task types. The first type of task is known as the floating task. Floating tasks do not have a start or end date and time. For example, if you would like to watch Ironman 3 someday. You give yourself no specific deadline associated with this task and so you enter:

```
>> add Watch Ironman 3
```

This command will add a new task to FINI. The corresponding screenshot is given in *Appendix A, Figure 2*.

Apart from looking at your updated list of tasks, the space above the command box will also confirm that the new task you specified has been successfully added.

Tasks with Deadlines

The second type of task are those which you have to complete by a specific date or time. For example, you have an upcoming project meeting with your team manager tomorrow evening at 7pm. To add this meeting to FINI, simply enter:

```
>> add Attend project meeting // 16/10/2015 at 7pm with priority high
```

The corresponding screenshot is available at *Appendix A, Figure 3*.

Tasks with Start and End Date/Time

FINI reminds you about events or tasks with a start and end time. In example, if you would like to attend Bob's birthday party tomorrow from 7pm to 9pm, you enter:

```
>> add Bob's Birthday Party // 11/10/2015 from 7pm to 9pm with priority
medium
```

A sample screenshot what you should see is given in *Appendix A, Figure 4.*

Tasks with Recurring Deadlines

Sometimes you may encounter tasks that you need to do on a recurring basis. For example, you may need to submit a weekly report every Friday at 9pm. To add such a task, enter:

```
>> add Submit weekly report // every Friday at 9pm
```

This example is also illustrated as a screenshot in *Appendix A, Figure 5*. You will be able to observe that the column titled "R" now contains "Fri" and the time column contains "9pm". This signifies that the event/task recurs every Friday at 9pm.

2.4 Undo a Previous Command

We sometimes make mistakes and the FINI Team recognizes that the "undo" function is therefore crucial to almost every user. This function will be especially useful in scenarios where a task has accidentally been deleted, a task has been incorrectly updated or if the user wants to quickly cancel a task that has just been added. To undo, simply enter:

```
>> undo
```

This command will then remove the latest command you entered, reversing all changes since then.

Quick Tip! *FINI supports undo for up to three past commands. To undo up to 3 commands use:*

```
>> undo <num>
```

Note that `<num>` can take values from 1 to 3.

2.5 Deleting a Task

Sometimes you may want to delete tasks which have been cancelled or tasks which details have been keyed in incorrectly. In this case, you may delete the task by keying in:

```
>> delete <task_id>
```

Note that `<task_id>` takes the value of the task ID which is displayed on your screen.

In example, to delete the task "Watch Ironman 3" which we added earlier, we key in:

```
>> delete 1
```

The following screen will then be seen, with the feedback reflecting that the task has been successfully deleted.

2.6 Editing a Task

Editing a task can come in really handy at times. For example, let us imagine you have already added a report that was due on Friday:

```
>> add Submit Project Report // 16/10/2015
```

On Wednesday, you then receive an email saying that the report's deadline has been changed to Saturday. To update the task, all you have to do is:

```
>> update <task_id> // 17/10/2015
```

If there is a need to edit the title of the task as well, you can input:

```
>> update <task_id> New Task Title // 17/10/2015
```

2.7 Completing a Task

FINI wants you to mark your accomplishments by indicating when tasks are complete. To mark a task that

```
>> complete <task_id>
```

Well done! Don't forget to give yourself a pat on the back!

2.8 Saving Your Progress

FINI automatically saves your session after every command. However, if you prefer to save your session manually, you can choose to enter:

```
>> save
```

To save your session under a new <file_name> such as "MyNewTaskFile" you may enter:

```
>> save as MyNewTaskFile
```

2.9 Getting Help

With FINI, help is always given to those who ask for it. Entering the command below will give you an instant help screen with an overview of all commands:

```
>> help
```

FINI DEVELOPER GUIDE

1 | Introduction

FINI is a desktop application written in Java. It utilizes JavaFX as the primary GUI library and utilizes the keyboard as the primary mode of user input. Hence, the use of mouse clicks is unnecessary and might only be useful to launch/close FINI through the system icons.

This developer guide provides an overview of FINI's project architecture as well as the necessary components and interactions involved. With greater clarity on how FINI is coded, you will be able to contribute further towards the development of FINI.

2 | Project Architecture

As shown in the project's architecture *(Appendix B, Figure 1)*, FINI comprises of 4 main components.

The **Graphical User Interface (GUI)** component consists of JavaFX's XML files. These files define the layout of the GUI and provide important components for users to interact. A Java controller file is also linked to the FXML file to allow the GUI components to respond to the user input.

The **Logic** component is responsible for interpreting the user's actions.

The **Parser** component is responsible for parsing the user's commands and creating or manipulating the relevant models

The **Storage** component consists of the storage controller and a text file. This file is the primary storage area for FINI to store the user's tasks. This file is also loaded when the user launches FINI.

3 | Command Flow

The sequence diagram *(refer to Appendix B, Figure 2)* illustrates the general process of how the various components of FINI interact with one another. The interactions are simple and logically guided, starting from the point of user input.

To facilitate a better understanding of the command flow, with reference to the sequence diagram (*Appendix B, Figure 2*), a sample scenario is illustrated below for greater understanding and clarity.

James is a user of FINI. Firstly, he launches FINI and is greeted by the Welcome Screen. He subsequently presses ENTER and he is brought to the main screen of FINI.

From here, he decides to add a new task, "Call mother at 7pm". This command is now received by the Controller which validates the user input and interprets the intended action of the user. Once the Controller interprets that the user wants to add a task, it calls FiniParser.

FiniParser parses the user's input to identify and segregate the various parts of the user's command and calls the necessary methods to create the task. FiniParser now calls the MainApp component to add the created task to the master-list of tasks.

Subsequently, FiniParser calls StorageController to ensure that the new task is updated in the text file of the user, which is the main form of storage.

4 | Components

The overview of FINI's components and class diagram can be found at *Appendix B, Figure 3.*

Graphical User Interface (GUI)

The GUI of FINI mainly consists of 2 "scenes". Scenes are JavaFX terms which describe what the user sees on the screen. The first scene is the "Welcome Screen" which greets the user and prompts the user to press "ENTER" to continue. The second scene is FINI's main layout.

The GUI component also consists of numerous FXML files. FXML files serve as the backbone of FINI's GUI as they define the layout of the various components such as the command box (for users to enter their input), task boxes etc. Specifically for the main command box, the FXML file is linked to a Controller Java file, rootLayoutController. This

Controller class is responsible in providing a response to the user's interactions. As for all other displaying FXMLs, they are linked to DisplayController.

The GUI component also includes the stylesheet, <style.css>. Cascading Style Sheets (CSS) is the main markup language that is used to customize the appearance and style of the JavaFX components.

The GUI component is the main component for users to control and manipulate FINI, since any input methods except for command line text is prohibited. It is handled by RootLayoutController class, DisplayController class and interacts with MainController which collaborates with Parser component to generate suitable objects and display the appropriate output.

The RootLayoutController class extends the JavaFX's BorderPane class. It serves two main functions. First, it is the main display background of FINI. It contains a JavaFX AnchorPane component which provides a graphical output. Second, it receives user input from a text field element and passes them to the MainController class in the Logic component to handle. The class decides when and how the application is going to process and react to the command. After processing, the DisplayController class within the BorderPane will display all the appropriate statistics.

Please refer to *Appendix B, Figure 4* for an outline of the RootLayoutController.
The code snippet shows you how the root layout, which receives user's command, handles the input and passes it to handleUserInput. The process repeats continuously until it gets an exit command.

Notable APIs - RootLayoutController Class

Return Type	Method and Description
void	handleKeyEvent(KeyEvent) : decide the follow-up action when KeyEvent is entered
void	handleUserInput() : capture user input, pass to Logic component

Notable APIs - DisplayController

Return Type	Method and Description
void	initFolderView() : initialize the view for displaying task categories
void	setDisplay(ObservableList<Task>) : initialize the view for displaying tasks
void	sendFeedback(String) : display the feedback text for user

Logic

The Logic component is pivotal in the operations of FINI. It is crucial for interpreting and executing the user's commands correctly. The Logic component is reliant only on the Storage component and works independently from the GUI component.

When the user inputs the commands, <executeCommand(String)> method is called which then parses the user's input. After interpreting the user's commands FINI identifies whether the command might be to add/delete/update/complete tasks. Once the command is identified, it then calls the necessary secondary methods to execute the action. Finally, the Logic component updates the variables present as well as the internal data storage by interacting with the Storage component. The Storage component then writes the updated data to the local file saved on the user's disk.

The MainController handles majority of the functionality of the Logic component. It can be regarded as the brain of FINI. The User's commands are passed to the Logic command and are processed using executeCommand(String) method. Following that, the FiniParser class extracts the date and time for the CreateTask class, which manipulates the Task object.

The MainController also passes the tasks and groups to the DisplayController class in GUI component which will be addressed in the later part of the guide.

Notable APIs - MainController

Return Types	Method and Description
`String`	executeCommand(String): Handle the execution of user inputs and return feedback to be shown on the feedback label
`ObservableList<Task>`	getDisplayedTasks() : Get the tasks that are going to be displayed

Parser

The Parser is another important component of FINI. It processes the information input by the user and translates the information into data that FINI can understand. The data is handed over to the Logic component that executes the command given.

The Parser makes use of keywords such as "//", "Next", "with priority" to make sense of the information input by the user. In example, if the user were to input "add Birthday party // next Saturday with priority high". FINI's parser will take note that the date for the task will be "next" week and also on "Saturday". FINI will also pass information to the logic component that this task is of high priority since "high" comes immediately after "with priority" in the string of input.

Notable APIs - FiniParser Class

Return Type	Method and Description
void	parseUserInput(String) : analyze the given String to detect the command type, and call secondary methods to execute them

Storage

FINI automatically saves the user's session after every command. The storage component is crucial as the storage file is required for FINI to load all the tasks on the next user session. FINI stores the user's tasks in a text file (.txt). In this file, each task is stored in Google JSON (GSON) format. GSON is a Java serialization library that can convert Java Objects into JSON and back to Java Objects when loaded by FINI.

Notable APIs – Storage Controller

Return Types	Method and Description
Boolean	updateFiles(ArrayList<Group>) : Handles the saving of storage file. It returns a Boolean value to indicate if the process is successful.

5 | Known Issues

As of version 0.2, there are no issues in FINI. This release is stable and as described above and the features have been implemented successfully without any bugs. If you are aware of any issues, please do not hesitate to inform us at FINIDevOfficial@Gmail.com

6 | Future Development

The alpha release (V0.2) of FINI was received with an overwhelming response. Many users gave positive feedback and there has been requests for FINI to be available on both Android and iOS mobile platforms. The developers are currently looking to expand FINI to those devices in the future. Hence, we consider the involvement of developers such as yourself crucial for FINI to progress. The following items are some areas for future development which you may want to consider implementing.

- Thorough testing
- Implement a history of user actions
- Enhance the GUI of FINI
- Implement auto-correction libraries
- Allow synchronization with Google Calendar and iCal (for OS X)

7 | Acknowledgements

The following third party libraries have been crucial for the successful development of FINI

Key Dependencies:

1. Gson (https://github.com/google/gson)
2. GraphAdaptBuilder (http://google-gson.googlecode.com/svn-history/r1170/trunk/extras/src/main/java/com/google/gson/graph/GraphAdapterBuilder.java)

We would like to specially thank Professor Damith, Ms. Lim and Lynette for their guidance. We hope you enjoy FINI as much as we enjoyed creating it. Thank you.

Appendix

Cheatsheet

Command	Description

`add <task_title>`
`add <task_title> // <start_date>`
`add <task_title> // <start_date> <end_date>`
`add <task_title> // every <day> at <time>`
`add <task_title> // <start_date> <end_date> with priority high`
	The given commands add the specified tasks into FINI. As described earlier, there are 4 main types of tasks that can be added:
	1. Floating Tasks – no date or time
	2. Tasks with deadlines
	3. Tasks with start and end time
	4. Recurring tasks

`undo` `undo <num>`	The undo command comes helpful to reverse a previously input command.
`delete <task_id>`	This command deletes the specific task associated with the id. The id will be reflected on the screen.
`update <task_id> // <your new task details>`	This command updates the task with new details. The new task details follow the same format as adding tasks.
`complete <task_id>`	This command allows the user to complete a task associated with the task id.
`save` `save as <new_file_name>`	Save helps you to manually save the current session. The "save as" command saves your current progress into a new file storage as specified by you. All storage files will be stored in FINI's root folder.
`help`	At any time, if help is required, key in "help" to get a condensed overview of all the commands you can input in FINI.

Appendix

Appendix A

The following appendix contains the relevant screenshots for the aforementioned features in the user guide.

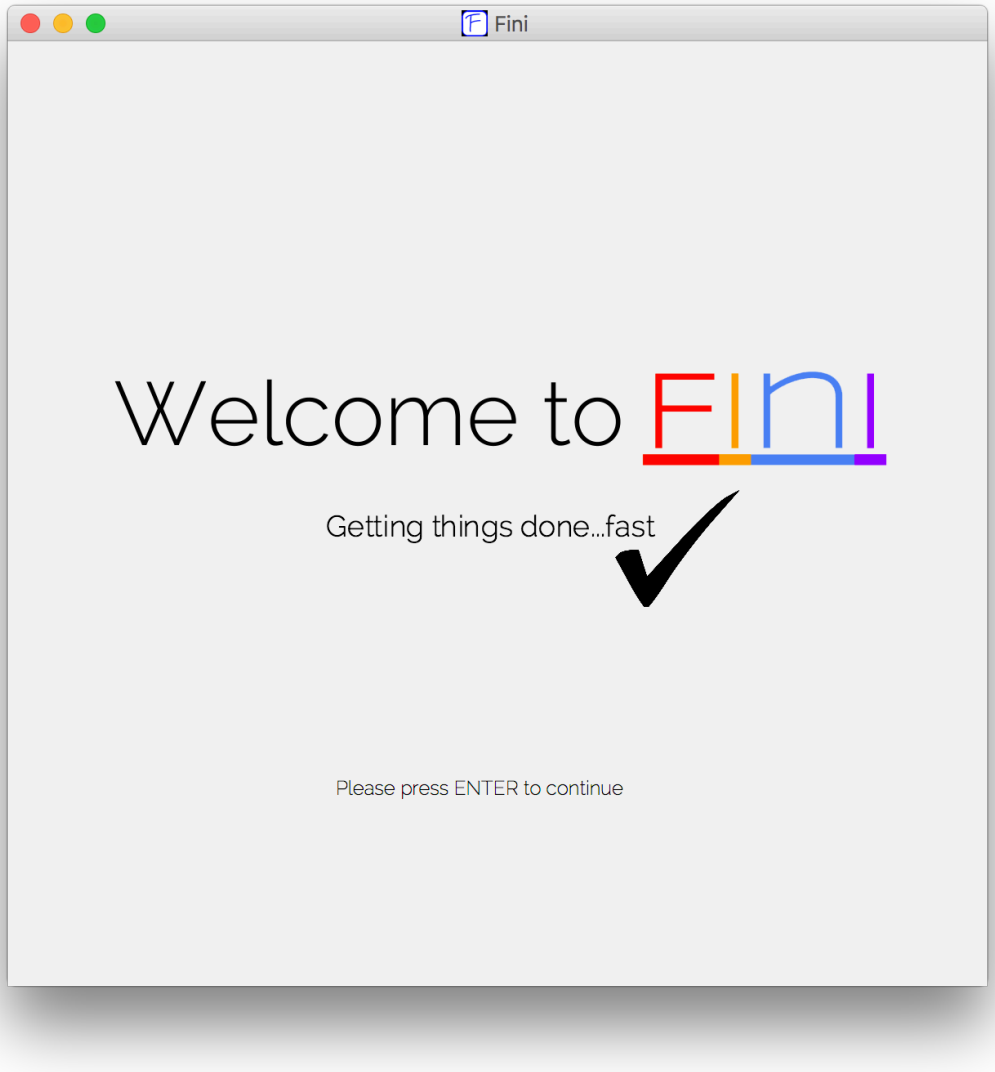

Figure 1. Fini's Welcome Screen

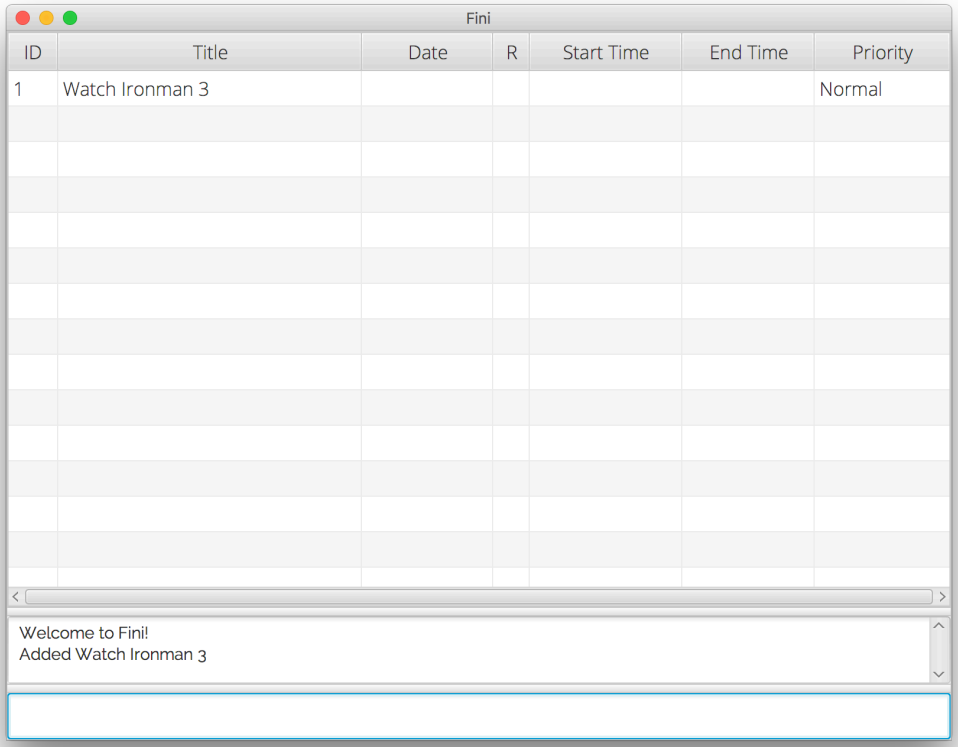

Figure 2. Adding a floating task

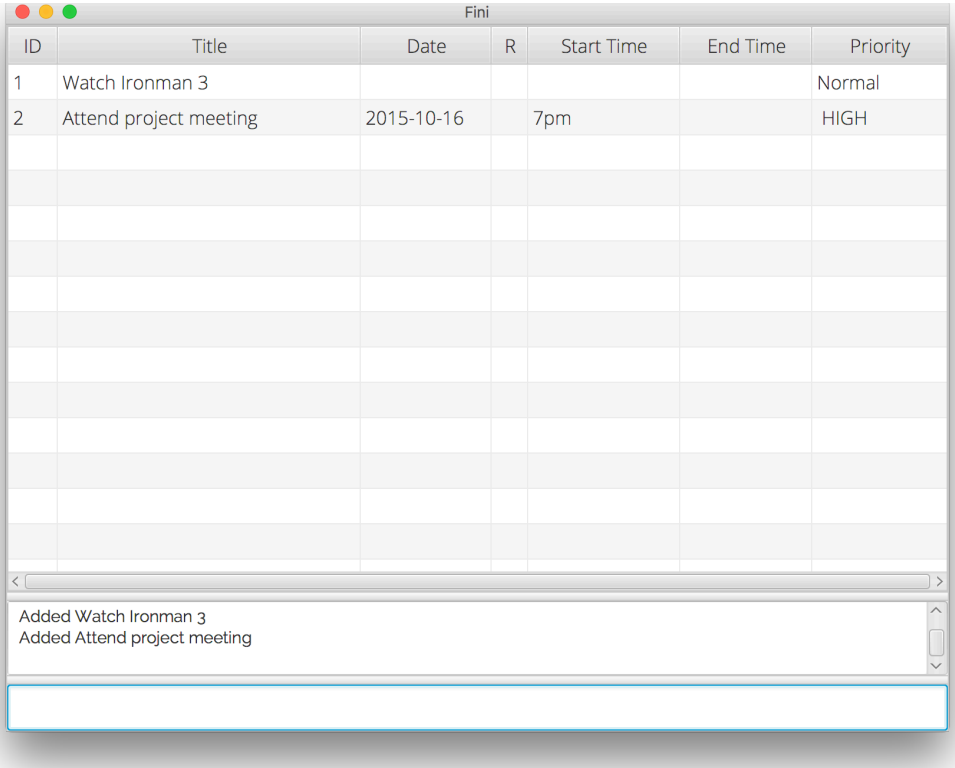

Figure 3. Adding a task with a deadline

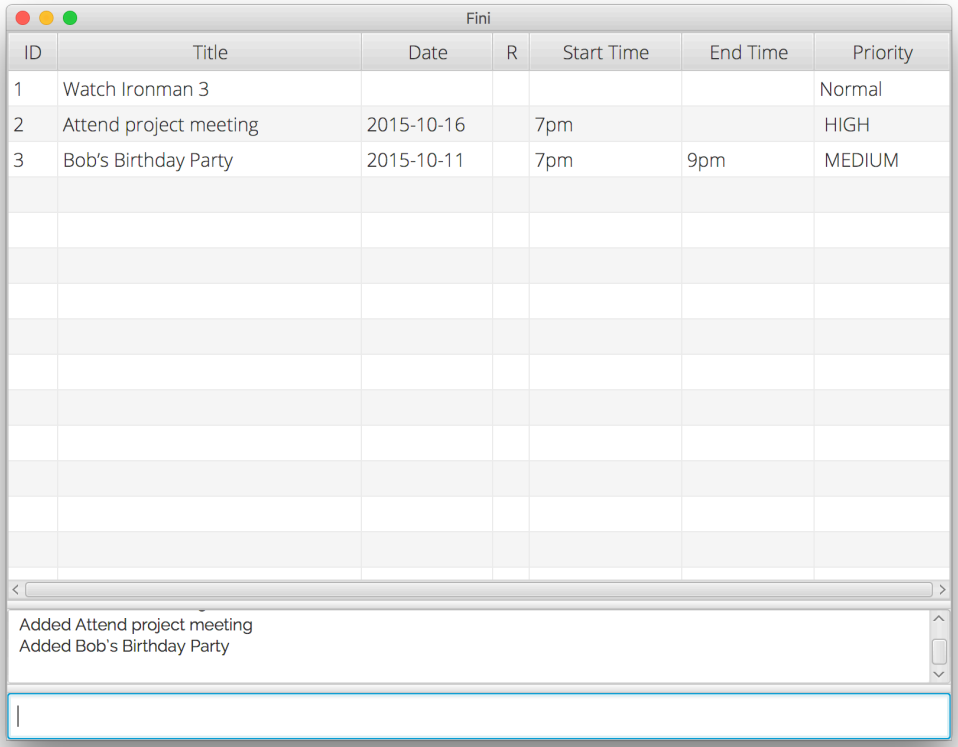

Figure 4. Adding a task with a start time and end time

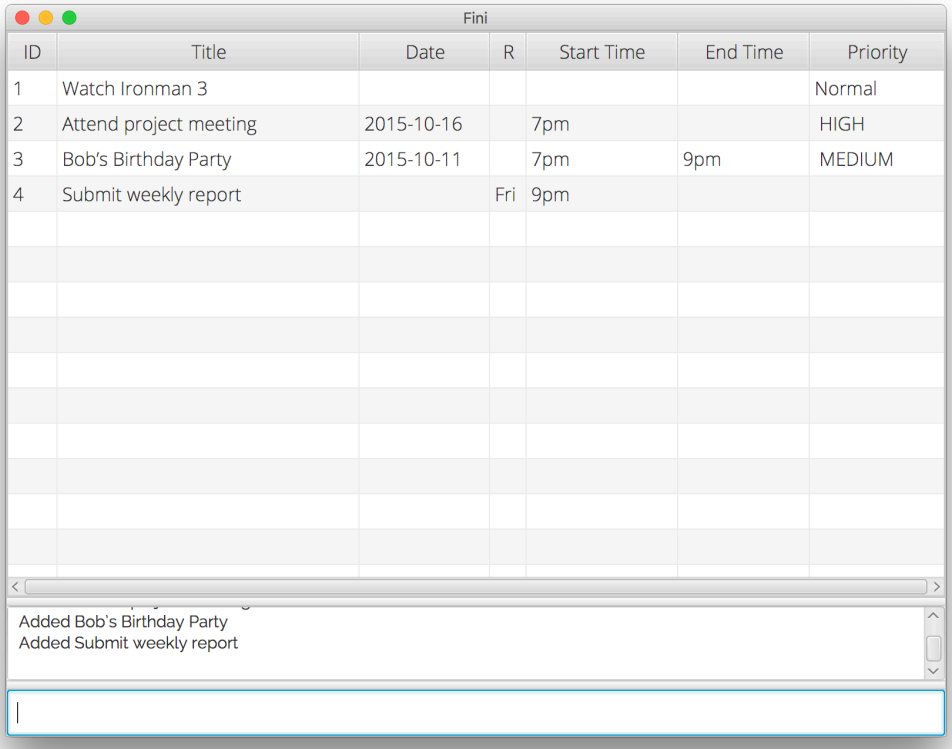

Figure 5. Adding a recurring task

Appendix

Appendix B

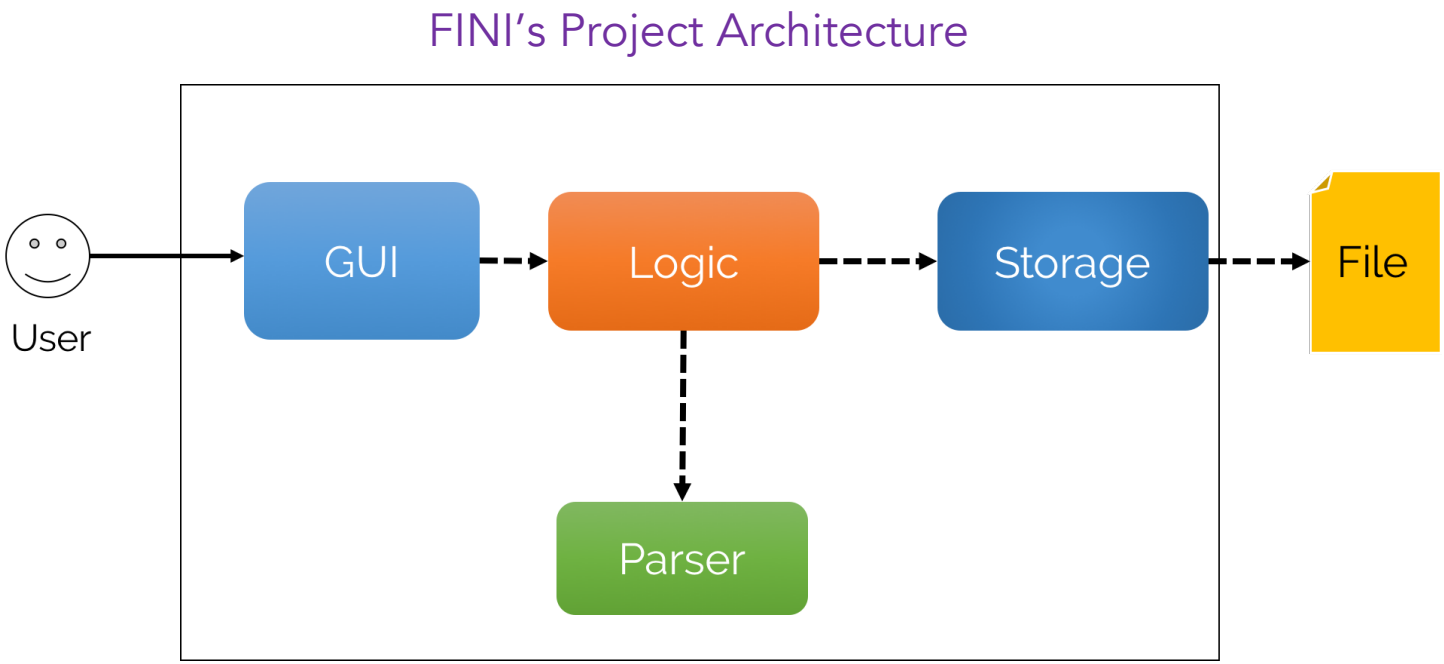

Figure 1. FINI's Project Architecture

FINI's Sequence Diagram

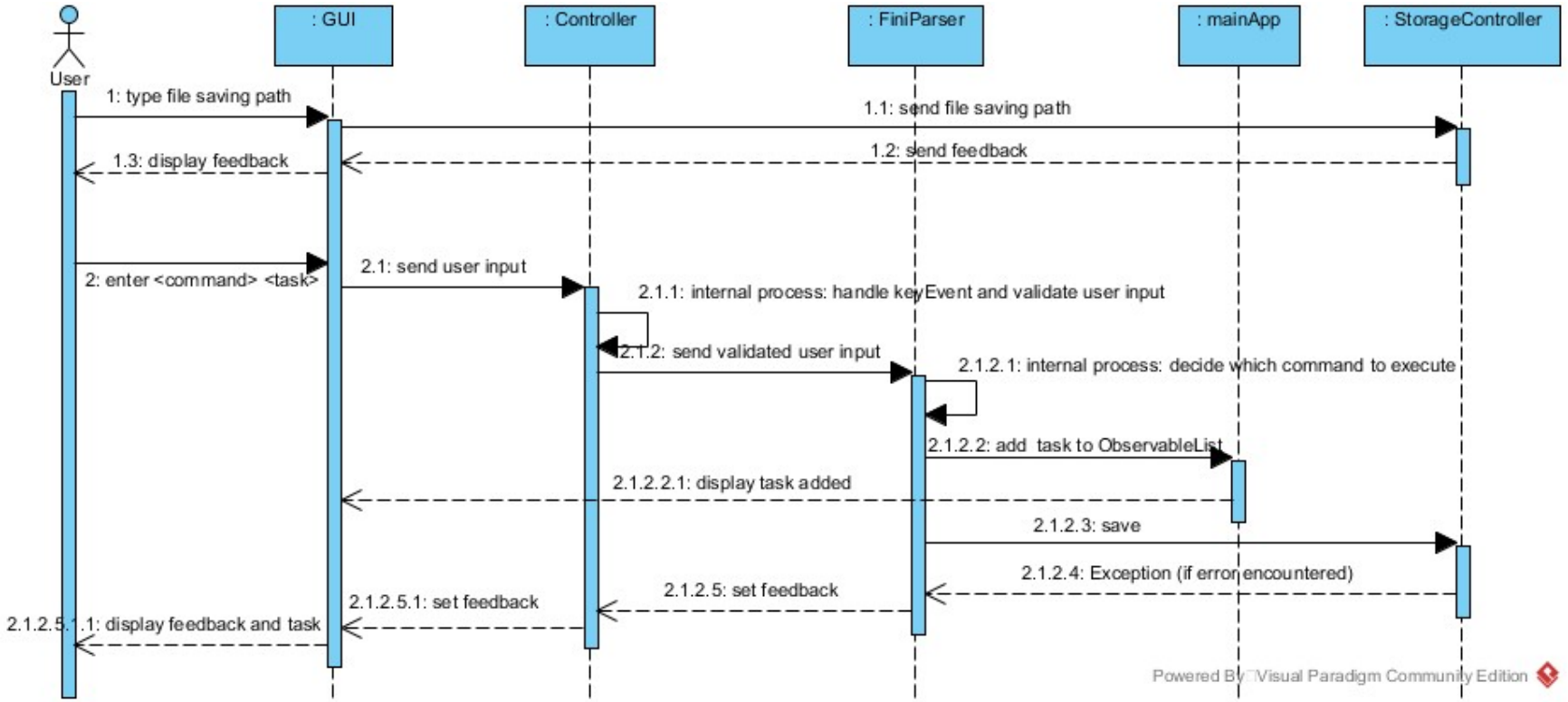

Figure 2. FINI's Sequence Diagram

FINI's Class Diagram

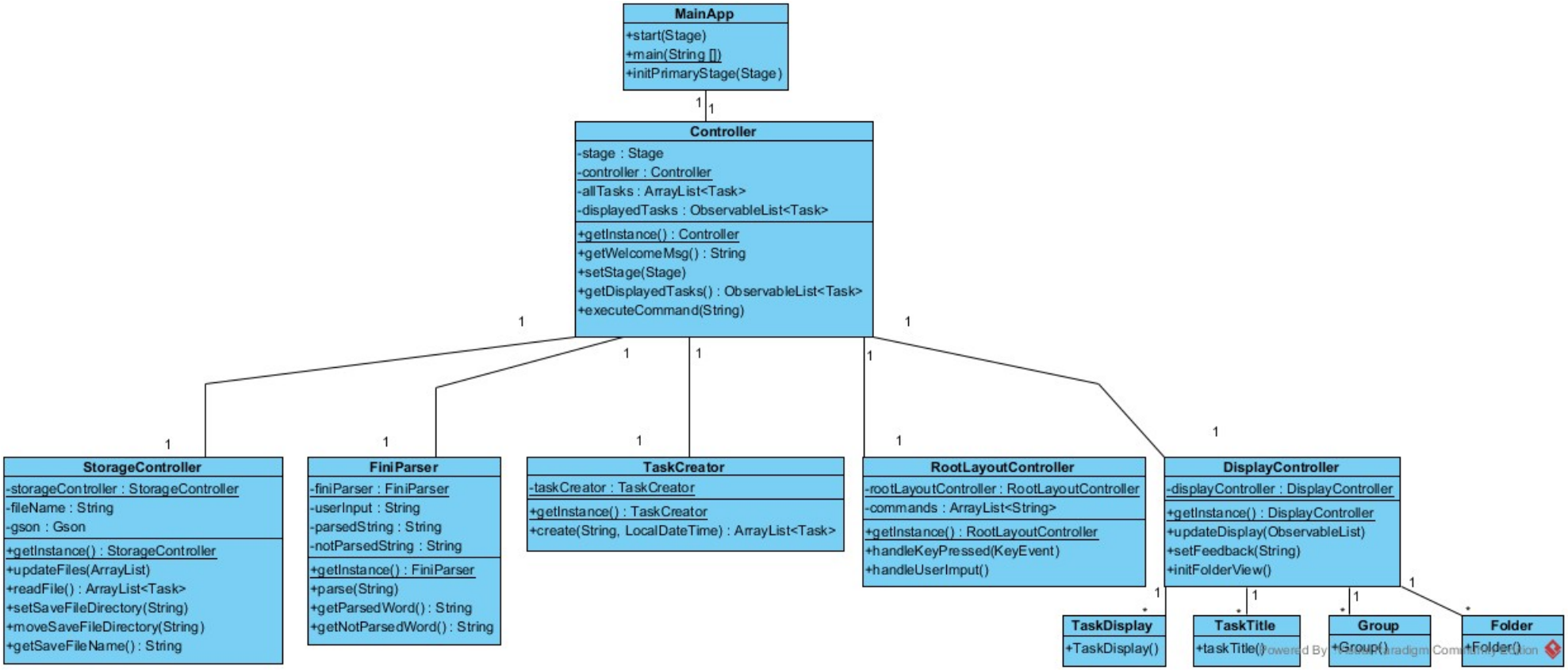

Figure 3. Class Diagram

GUI Component Code Snippet

```
public class RootLayoutController extends BorderPane {

    @FXML
    private TextField userInput;

    private Controller controller;
    private DisplayController displayController;

    . . .

    @FXML
    public void handleKeyPress(KeyEvent event) {
        . . .
        if (event.getCode() == KeyCode.ENTER) {
            handleUserInput();
        }
        . . .
    }

    . . .

}
```

Figure 4. Code Snippet 1 - RootLayoutController

Appendix

Appendix C

User stories. As a user, …

[Likely]

ID	I can … (i.e. Functionality)	so that I … (i.e. Value)
addFloating	add a task by specifying a task description only	can record tasks that I want to do some day
	change info for a task	the change of deadline can be done
	record repeated task under one go, e.g. daily/monthly task	can manage tasks more easily
	view the past events I have created quickly (time-efficiency)	can retrieve hem easily
	be able to add events with a start datetime and an end datetime	can know when I am required to complete the task by
	update the details of the previous task I have created	can add on important details even after creating the task
	Manually save my current state	Won't take the risk of data loss
	delete the previous tasks I have created	in the case of there is no need for the task anymore
	view/read the previous tasks I have created	can view a comprehensive list of tasks at hand
	specify specific folders to use as a datastorage location	convenient for me to locate them
	undo my previous action in a single session	Won't regret for accidental deletion or modificaton of tasks
	be able to add my tasks into the program	collate my tasks into the program
	be able to type my commands in a natural way	can add my tasks with ease

	view which activities have been already completed and which are pending on my watchlis	can have an overview of the tasks at hand
	add deadlines to the events	Can keep track of which items are due at which time
	search for my tasks efficiently which can suggest the tasks I am trying to read/update/create	can find it with ease
	delete tasks	can remove unimportant task

[Unlikely]

ID	I can … (i.e. Functionality)	so that I … (i.e. Value)
	auto-save/auto-sync every certain time interval	there will not be any info-loss accidentally
	specify the level of priority to the various tasks I added	can prioritise my tasks
	be able to have a good user interface in the form of a GUI	can get visual feedback from the program
	be able to sort my tasks in various order, e.g. due date, priority, lexicographically	can view them with ease
	want to search for the item I am looking for with reasonable level of ease, through various key words	can find the tasks/items quickly
	want to have a daily summary of the tasks due for the day	Am clear with what I am going to do that day
	be able to sync across my other sources of schedules, such as Google Calendar	all my tasks are synchronised.

Non Functional Requirements

e.g. The software should run on Windows 7 or later

- The software should work on a desktop without network/Internet connection.
- The software's primary mode of input is through a Command Line Interface (CLI).
- The software should work stand-alone without any dependencies on other software.
- The software should not use any relational databases to store data. Data must be stored with text files created by the software.
- The software stores data locally on the system and in a text file that is editable by users.
- The software should follow Object-oriented programming paradigms.
- The software should run on Windows 7 or Windows Vista OS.
- The software should run on both 32 bit and 64 bits PCs.
- The software should work without requiring any installers.
- The software should only utilize third-party frameworks and libraries which are free, does not require any installation by the users and does not violate any of the aforementioned constraints.

Product Survey

Product: Todoist, **Documented by**: Venkatesan Harish

Strengths:

- Able to categorize tasks into "Projects" which acts as categories.
- Pleasant User Interface (UI)
- Color tagging for "Projects"
- Overview of the schedule and tasks to be done for 7 consecutive days
- Fast view for the tasks due today
- Able to filter tasks according to the "priority level" and by "who assigned it"
- Able to add tasks with keyboard shortcuts
- Able to add recurring tasks
- Available on all web/desktop/mobile platforms
- Provides incentives for task completion in the form of karma points
- Location-based reminders (PRO version)
- Add tasks via email (PRO version)
- Task reminders via email/mobile messages (PRO version)
- Add task notes and files related to the item (PRO version)

Weaknesses:

- Maximum of 4 indent levels for a task
- Maximum of 3 indent levels for a project
- Maximum view of the next 7 days' agenda only. No monthly/yearly view available

Product: Remember the Milk, **Documented by**: Wang Jie

Strengths:

- The Smart Add allows ^ to specify the due date and = to specify time estimate, this syntax is quite intuitive
- Uses ! to set priority, serves as the ranking index, we can do sorting according to priority
- Uses # to add tag/group by, an effective way of grouping tasks together, we shall go that way as well.
- Uses @ to set places, intuitive and quite useful if the places varies widely.
- Date is within description but not as a task property, easy for development
- Repeating tasks, daily/weekly/biweekly/monthly are provided
- Sort by lexicographical order, maybe easy for search.
- Duplicate task, make an identical copy of task, easy for user.
- Undo function, really safe because every modification will have an undo option.
- Allow URL, can be used as memo to save important web pages.

Weaknesses:

- The Smart Add allows ^ to specify the due date and = to specify time estimate, but the time estimate plays a role more like an indicator, it does not serve any management purpose, like solve timeslot clash and so on.
- Uses ! to set priority. The priority is only limited within 3 level. It may not be sufficient for Jim's use. A reference from Outlook email priority design might be a good idea for our project.

- Uses # to add tag/group by, maybe add #secret which can be hidden from normal display is a good choice.
- Uses @ to set places, not detailed enough
- Date is within description but not as a task property, do not see any advantages for user
- Repeating tasks, not customized, inconvenient for tasks happen every two days/three days
- Sort by lexicographical order, but I personally think it is meaningless
- Duplicate task, very messy, I personally do not suggest that way to manage tasks
- Undo function, redundant undo pops up, really unnecessary
- Allows URL, malicious users will use this feature to break our program code.

Product: Wunderlist, **Documented by:** Yu Qiyun

Strengths:

- Has CRUD function. Can add in reminders differ from deadlines
- Can add in task description.
- Can add sub task to a main task.
- Can append files for a task.
- Can set tasks into different folders
- Fast launch
- Well-designed UI.
- Can mark "starred task".
- Easy to mark as completed.
- Can display tasks according to "today", "this week", and different task files

Weaknesses:

- Unable to add specific timelines to tasks. Only deadlines with date can be set (not hours).
- Unable to distinguish: task with deadlines & task during a certain interval & task to be done in near future
- Need to interact through UI, which requires clicks with mouse.
- Only one level of priority.

Product: Todo.txt, **Documented by:** Joel Jonas

Strengths:

- Simplistic and minimalistic.
- Low memory usage.
- Can be integrated online via Dropbox
- Open source – can add/customize your own plugins and features
- Sorted by priority
- Supported on multiple platforms (both desktop and mobile)

Weaknesses:

- Lack of good UI
- Hassle to type out the commands every single time
- Significant learning curve for new users
- Unable to see quick summaries such as a weekly agenda
- Lack of reminders

- END OF DOCUMENT -