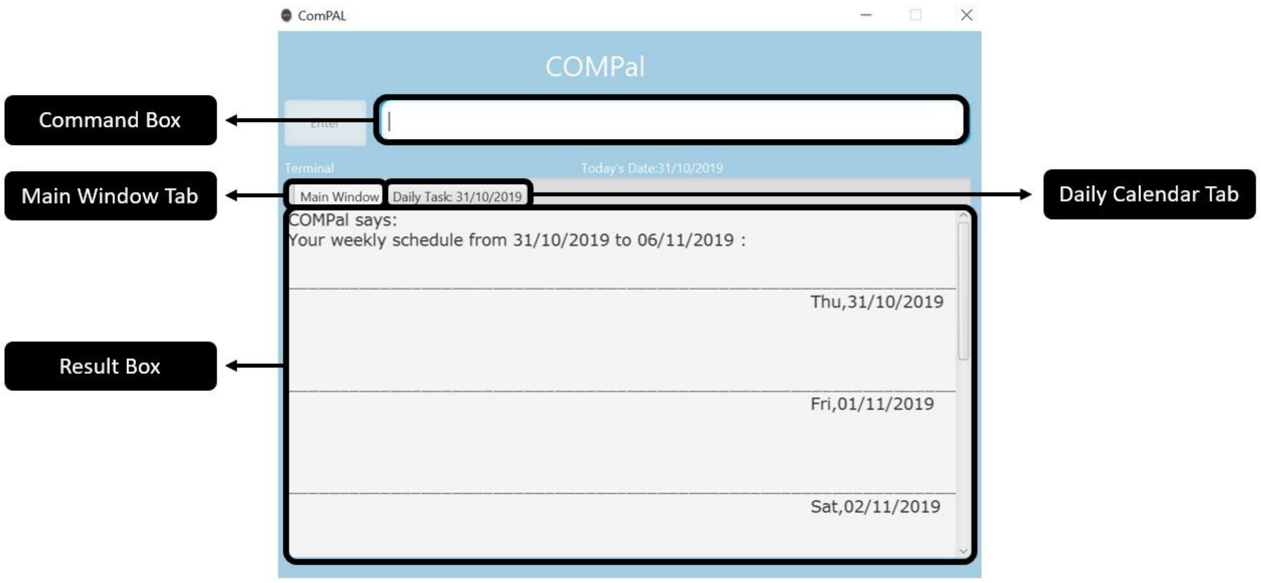
Jaedon Kee - Project Portfolio for ComPal

About The Project

My team of 4 Information-Security students and I were given the choice to refine the existing command line application called ‘Duke’ or to morph it into an entirely different product. We decided to refine it into a product which can do more than the existing ‘Duke’ application and chose to name it **ComPal**. **ComPal** is a student task management application that provides users the ability to prioritize their tasks and better manage and organize their student lives.

This is what our project looks like at the moment:



```
COMPal says:
Your weekly schedule from 31/10/2019 to 06/11/2019 :
Thu,31/10/2019
Fri,01/11/2019
Sat,02/11/2019
```

Figure 1. Graphical User Interface for **ComPal**

About This Portfolio

My role in the development of **ComPal** was to help implement the `find` and `edit` features of the application. The `find` feature allows users to search for a specific task consisting of the key searchword. The `edit` feature allows all input tasks to be edited. The following sections will showcase these implementations and the relevant sections that I have contributed to the User and Developer Guides.

The following symbols and formatting will be used in this document:

ℹ	Indicates important information that you may want to take note of.
`command`	Black text on grey indicates a command that can be executed.
`LogicManager`	Blue text on blue indicates a component, class or object that is part of the structure of the application.

Summary of Contributions

This section shows a summary of my programmatic contributions, documentation and other contributions to the project.

Implementations: I implemented the ability to search for tasks and to edit tasks as well as the storage mechanism for **ComPal**

- Search Feature(`find`):The `find` command allows the user to search for any keyword that might be in any task
    - For example, `find cs2113t` will pull out tasks with the term ‘cs2113t’ in their description
    - This is useful for users who want to filter out their tasks and view only tasks relevant to a certain topic/module/subject
    - This implementation works well with current code and will work well with future code. It can be easily extended in future to include more complex but more comprehensive search patterns (e.g match-cases)
    - The designing of this feature was slightly challenging because there needed to be consideration for ease-of-use for the user, so I had to strike a balance between complexity(effective searches) and simplicity(intuitive and easy to use)
- Edit Feature(`edit`):The `edit` command allows the user to edit the details of any task that has already been input into **ComPal**, without having to delete and then recreate a new task
    - For example, `edit /id 4 /description this is a new description` will edit the task with task id of 4 and change its description to ‘this is a new description

    - This feature works extremely well with current code and can be used to implement more advanced features in the future (such as the automatic rescheduling of tasks)
    - Designing and implementing the feature was challenging because there had to be restrictions on what the user could edit and because it is a feature that could potentially break the program
- Storage: Not a command. Instead, it allows for the vital operation of saving and loading data to and from the user's computer
    - Storage is used every time a command is executed and the command executed requires saving of data. For example, the `find` command does not require saving but the `edit` command does, as it changes details about the tasks.
    - Storage is also used upon starting up. Storage allows **ComPal** to extract stored data from the data file that is stored on the user's computer and loads it into the program as data that the user can interact with

Code Contributed: Contributed Code , Contributed Pull Requests

Other Contributions:

- Project Management:
    - I oversaw the release of v1.2 and mid v1.3
- Enhancements
    - I implemented the base GUI (Graphical User Interface) as part of enhancing ComPal's user friendliness and aesthetics (pull request #1)
    - I designed the current layout, chose the current colour scheme and created and implemented the GUI functionality
    - I refactored the storage mechanism to make it more Object-Oriented-Programming compliant (pull request #154)
    - I added the implementation of a task id which makes it more convenient for users to identify tasks (pull request #149)
- Documentation
    - Continually refined the user and developer guide
- Community
    - I reviewed pull requests from my teammates (pull request #127, #35)
    - Opened issues regarding bugs found (Issues #147 , #182 , #185)

Contributions to the User Guide

We had to detail usage of our features in the User Guide. The following section contains extracts from ComPal's User Guide I am responsible for, namely the section detailing usage of the find feature and the edit feature, as well as the section for future enhancements. They demonstrate my ability to clearly document aspects of developing this project.

4.1.9. Editing Tasks: edit

This section demonstrates how to edit a task that has been stored inside our application.

If you need to change information about a task, enter edit /id ID <options> where ID is the task's id number.

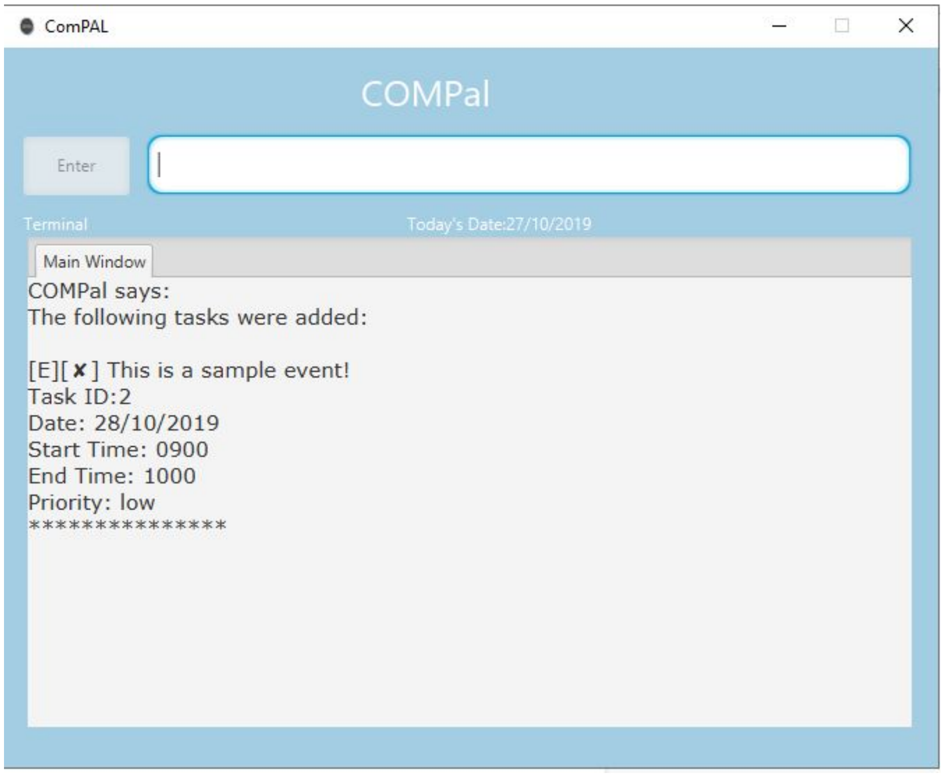
Format for editing a task's description: `edit /id ID /description NEW DESCRIPTION`
Format for editing a task's date: `edit /id ID /date NEW_DATE`
Format for editing a task's priority: `edit /id ID /priority NEW_PRIORITY`
Format for editing a task's start time: `edit /id ID /start NEW_`*`START`*`_TIME`
Format for editing a task's end time: `edit /id ID /end NEW_`*`END`*`_TIME`

This command allows for a combination of <options>, that is, you can include more than 1 option field in the command. Any combination of the following inside the command is legal:

- /description NEW_DESCRIPTION
- /date NEW_DATE
- /priority NEW_PRIORITY
- /start NEW_START_TIME
- /end NEW_END_TIME

For example, if you wish to increase the priority of a task with an id number of 2 currently set to low and bring forward its starting time to 6am, do:

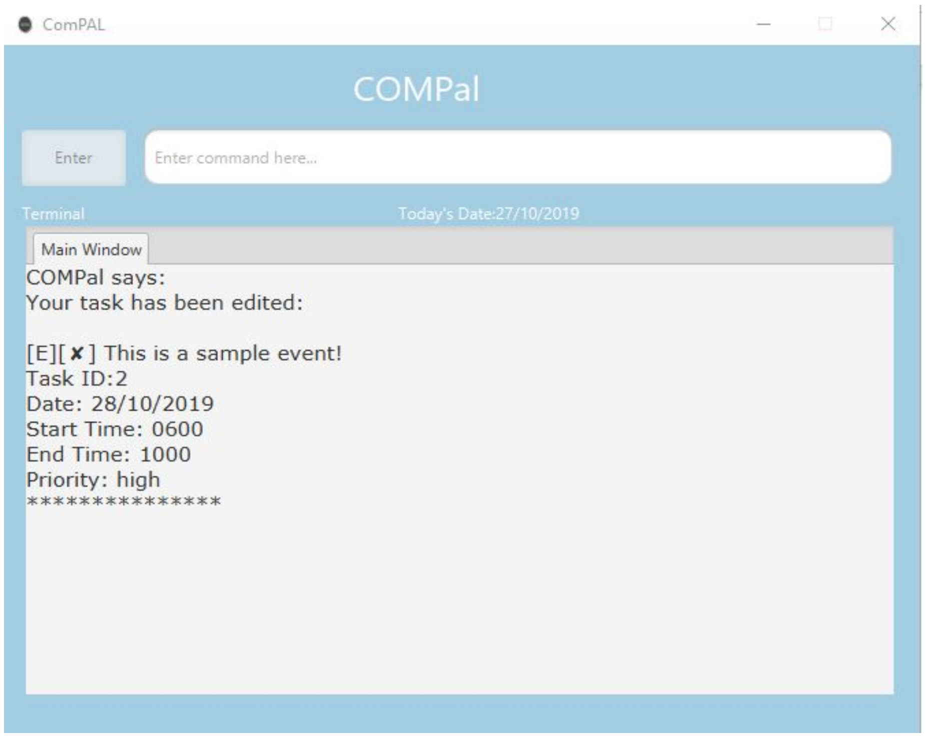
```
edit /id 2 /priority high /start 0600
```



```
COMPal says:
The following tasks were added:

[E][✘] This is a sample event!
Task ID:2
Date: 28/10/2019
Start Time: 0900
End Time: 1000
Priority: low
***************
```

Figure 2. The original task with a task id of 2, before the `edit` command is executed



```
COMPal says:
Your task has been edited:

[E][✘] This is a sample event!
Task ID:2
Date: 28/10/2019
Start Time: 0600
End Time: 1000
Priority: high
***************
```

Figure 3. Task with task id 2 after the `edit` command executes

Once the command executes, note that the priority of the task has been changed from low as shown in Figure 2 to high in Figure 3 Also note that the start time has been changed from 0900 to 0600, or 6am, as intended.

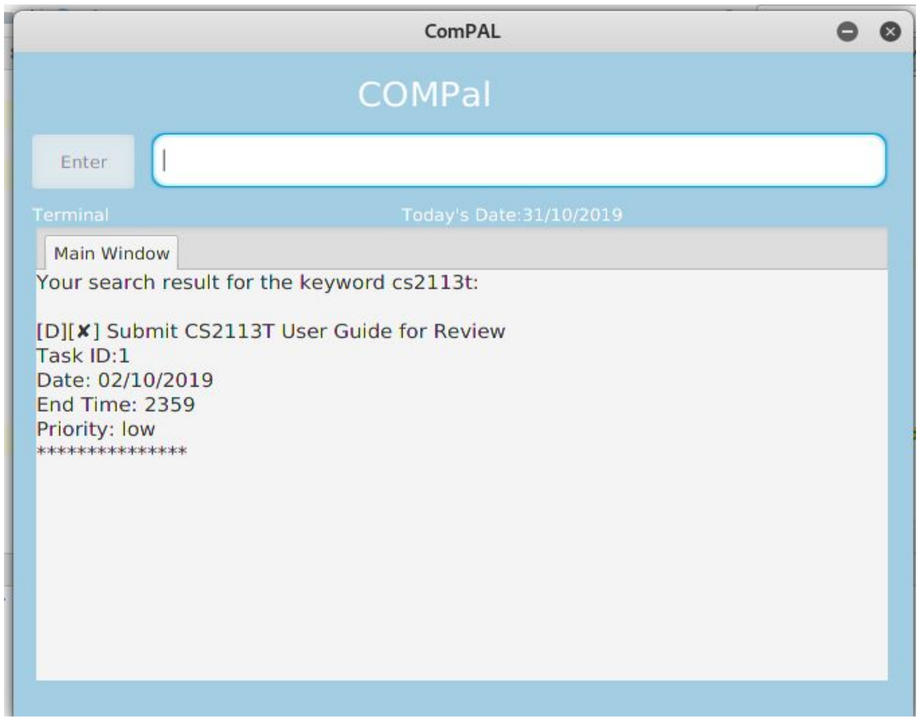
4.1.4. Finding a Task: find

This section demonstrates how to find certain tasks based on words in their description.

If you wish to search for the task by it's key word, enter find KEY_WORD in the **command box** will give you all the tasks including the key word.

Format for finding a task with KEY_WORD in its description: `find KEY_WORD`

ℹ	The find command without any arguments following it will return and display all tasks to you.



```
Your search result for the keyword cs2113t:

[D][✘] Submit CS2113T User Guide for Review
Task ID:1
Date: 02/10/2019
End Time: 2359
Priority: low
***************
```

Figure 4. Output of an example command `find cs2113T`

Contributions to the Developer Guide

The following section contains extracts from the Developer Guide showcasing my contributions to it. I was mainly in charge of detailing the implementation of the find command/feature as well as the storage component.

ℹ	Some parts/words/explanations of the extract below have been simplified for the non-technical reader's sake

5.5. Find Feature

This feature allows the user to search for a keyword or phrase in the description field belonging to all of the tasks

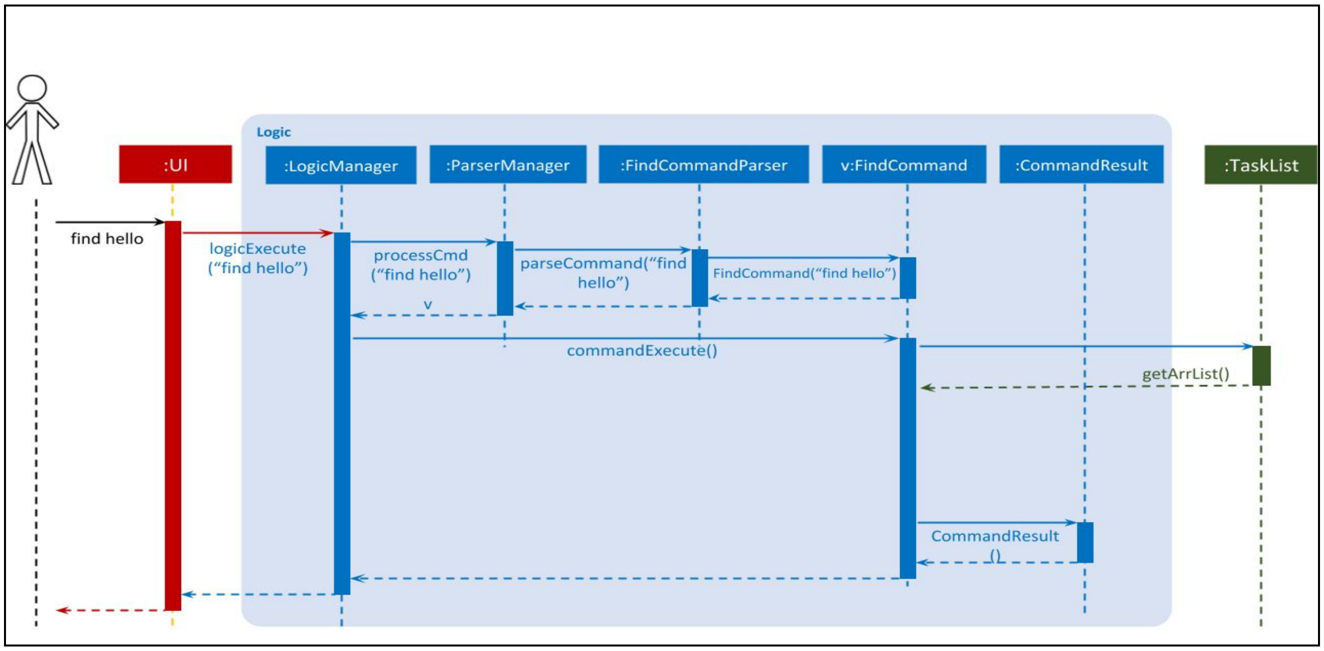
5.5.1. Current Implementation

The current implementation matches the keyword or phrase exactly to the description. As long as the keyword or phrase is a sub-string in the description field, the task is returned as a match. Likeness of the words are not considered at the moment e.g 'frst' will not match 'first'.

1. Upon the user entering the find command with a valid keyword, the `LogicManager` is called and sends the user input to `ParserManager`.

2. `LogicManager` then invokes the `parseCommand` function of `ParserManager`.
3. `ParserManager` in turn invokes the parse function of the appropriate parser for the find command which in this case, is `FindCommandParser`.
4. After parsing is done, `FindCommandParser` would instantiate the `FindCommand` object which would be returned to the `LogicManager`.
5. `LogicManager` is then able to call the execute function of the `FindCommand` object just returned to it.
6. In the execute function of the `ViewCommand` object, task data will be retrieved from the `TaskList` component.
7. Now that the `FindCommand` object has the data of the current task of the user, it is able to execute its logic.
8. `FindCommand` will loop through all tasks to find any description matching (non-case sensitive) the keyword or phrase.
9. The result of the command execution, a list of matches from the keyword/phrase passed in, is encapsulated as a `CommandResult` object which is passed back to Ui for displaying to the user.

Here is a sequence diagram portraying the above sequence of events:



5.5.2. Design Considerations

The `find` command should not be too complicated. It should be very intuitive to use and hence there was no need for any much parsing on the part of the `FindCommandParser` object. The user should be able to search for any task with ease and without referring to the user guide as much as possible.

5.5.3 Future Implementation

Case-Sensitive Search

Involves just using a different match/regex API(Application Programming Interface). An API can be said to be some feature that has already been implemented and available for use in any current development/code. Regex (short for Regular Expressions) is a set of rules that relates to natural language processing in computers. It allows us to explain to computing devices how to understand or match languages. For example, h.* will match any word that starts with the letter ‘h’, like ‘hello’ or ‘hi’ or ‘happy’.

Match Based On Likeness/Regular Expressions

Make use of regex to match words based on likeness/regex rules rather than an exact substring match. This will help users with typographical errors but is not considered a must to implement.

4.5. Storage Component

API: StorageManager.java

We use very simple and user-editable text files to store user data. Data is stored as data strings separated by underscores. The separation token however, can be easily changed if desired.

Data is thereafter parsed as a string and then processed by our storage API into application-useful datatypes such as Task Objects.

While it might be viewed as primitive, the advantage of this approach is that it is an almost no-frills implementation and is easily comprehended by the average developer. The average user can also understand and easily directly edit the data file if so desired.