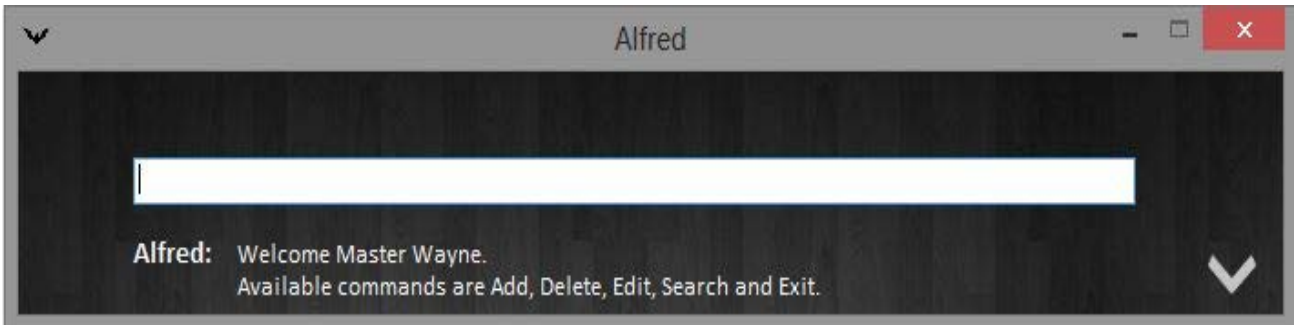
[F10-5S][V0.2]

Alfred

He will organize your life

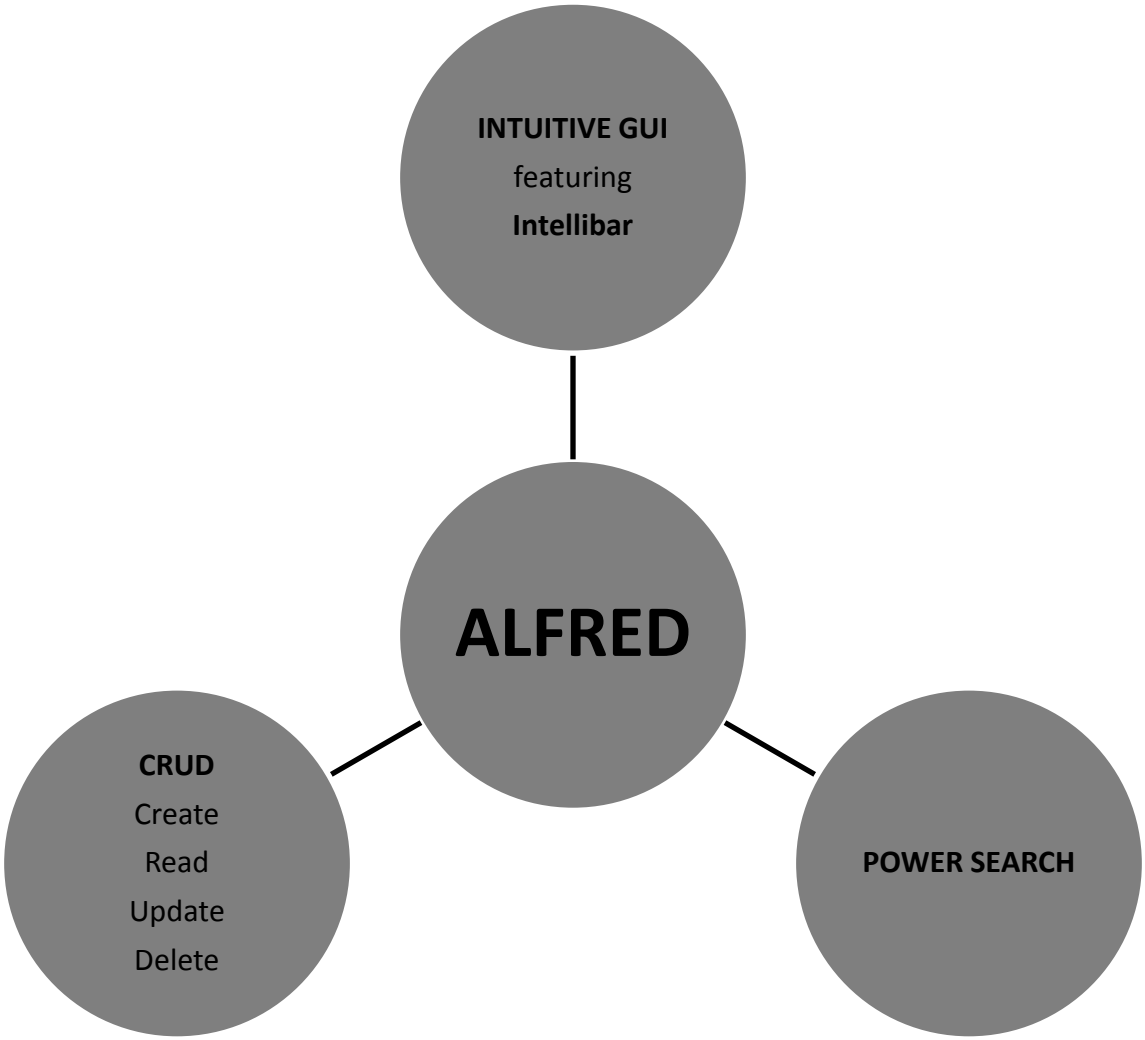


			
Shourya Moona	**Rohit Mukherjee**	**Lee Jianwei**	**Madhav Kannan**
[GUI, Software Developer, Testing]	**[Software Developer, Testing]**	**[Software developer, Documentation]**	**[Software architect, Documentation]**

User Guide

Who is Alfred?

Alfred is a productivity application that will organize your everyday life and ensure you never miss a deadline. For those on the go, Alfred will make sure you're never burdened with too many tasks at a time. A systematic planner application that will create an inventory of all your tasks, Alfred can add, delete and modify tasks on the go. He will sync your schedule with Google Calendar, allowing you to track your tasks on any portable device. He will also remind you to complete your tasks well in advance. With an elegant, intuitive interface Alfred is at your service 24x7. Granted you're not Batman, but everyone in the 21st century needs an Alfred.



Alfred's Functionality Specifications

Getting started

I Creating a Task: (*some parts to be implemented)

The new task can be entered in based on two formats. The task may be prefixed with an "add" keyword, but it is not necessary. "-sd", "-st", "-ed", "-et" and "-t" must be specified to indicate start date, start time, due date, due time and tag of the task respectively:

- **Single day case –**
  A one-time event can be entered in the following ways:
  1. **Task name and start date and time.** (Add Canoe polo –sd 16 December 2012 –st 2pm – Adds the task with the stated task description, with the specified start date and time.)
  2. **Task name and end date and time.** (Add Canoe polo –ed 16 December 2012 –et 5 pm – Adds the task with the stated task description and the specified due date.)
  3. **Task name and start date, start time, end date and end time.** (Add Canoe polo –sd 16 December 2012 –st 2 pm –ed 16 October 2012 –et 5 pm – Adds the task with the stated task description, with specified start date and time, and specified due date and time.)
  4. **Task name, and any and all combinations of start date, start time, end date and end time.** (Add Canoe polo –sd 16 December 2012 –et 5pm – Adds the task with the stated task description and the specified combination of time parameters.)
  5. **Task name.** (Add Canoe polo – Adds the task with the stated task description, as a floating task.)

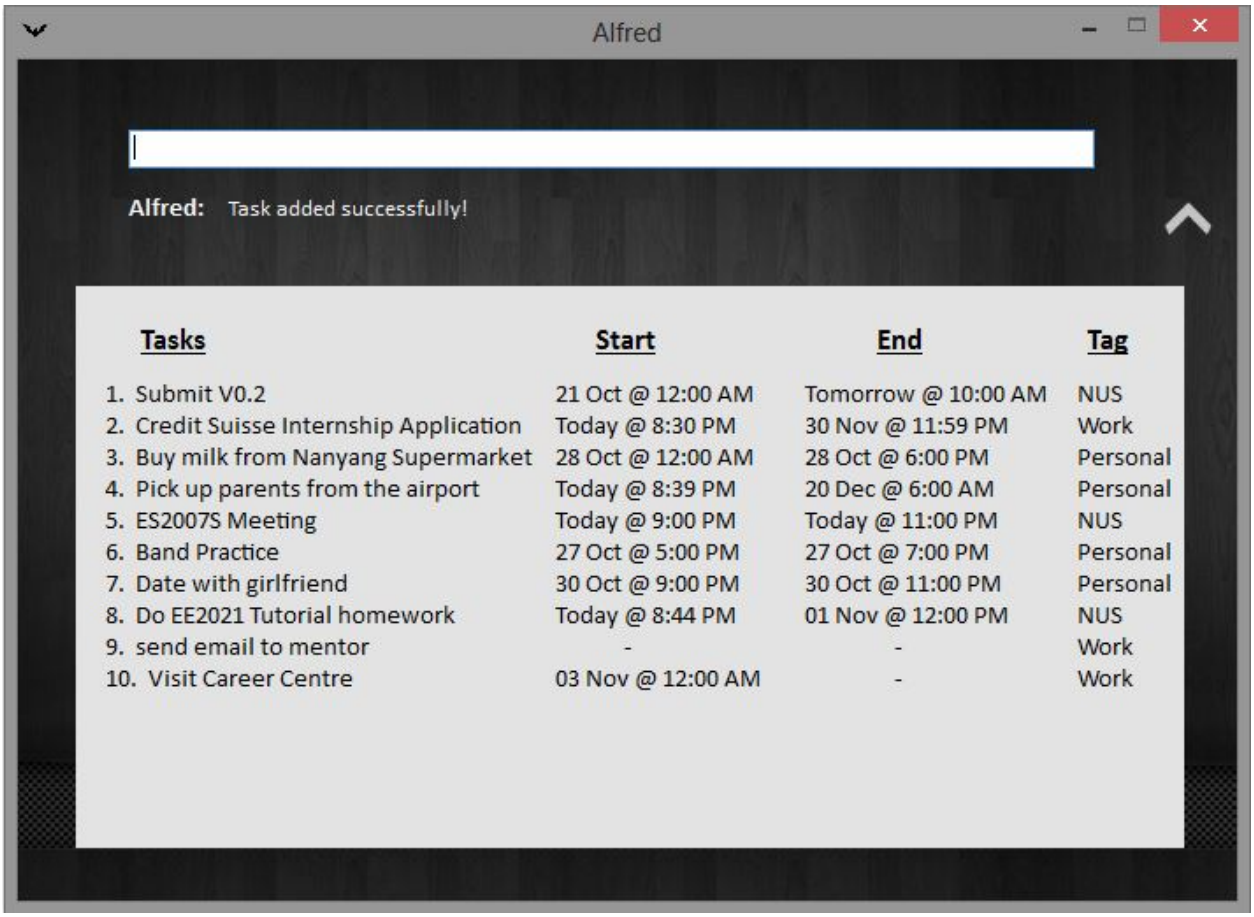
- ***Recurring case –**
  An event that recurs can be added in the following way:
  1. **Task name and time and frequency.** "-r" must be specified to indicate recurrence. (Add lunch –et 2pm –r – Adds it to every day at the mentioned time.)

- ***Setting Alarms –**
  A nifty alarm feature is provided:
  1. Adding an exclamation mark makes an event important and there will be a reminder for the event once a day from 3 days before the event.
  2. A pop-up will remind you of your upcoming event, half an hour before it is to be performed.
  3. Custom alarm tones can be assigned by the user as per their convenience, to create a variation from one task to the other.

Creating a task

II Reading a task:

- Default page with drop-down menu shows users what has been updated into planner. It shows the latest status of tasks (whether they have been completed or how much time remains for the task to begin).



Display (Expanded view)

III Update/Edit Tasks:

Tasks can be edited in 3 formats. The task is prefixed with different keywords as shown:

1. **Undoing last action.** "Undo" must be specified to undo the last performed action.
2. **Redoing last action.** "Redo" must be specified to redo the previously undone action.
3. **Overwriting a previously entered task.** "-sd", "-st", "-ed", "-et" and "-t" must be specified to indicate start date, start time, due date, time and tag of the task respectively. (Edit canoe polo –sd 16 December 2012 –st 2pm –ed 16 October 2012 –et 5pm – Updates the task description with the updated start date, start time, due date and due time.)

IV Delete: (*some parts to be implemented)

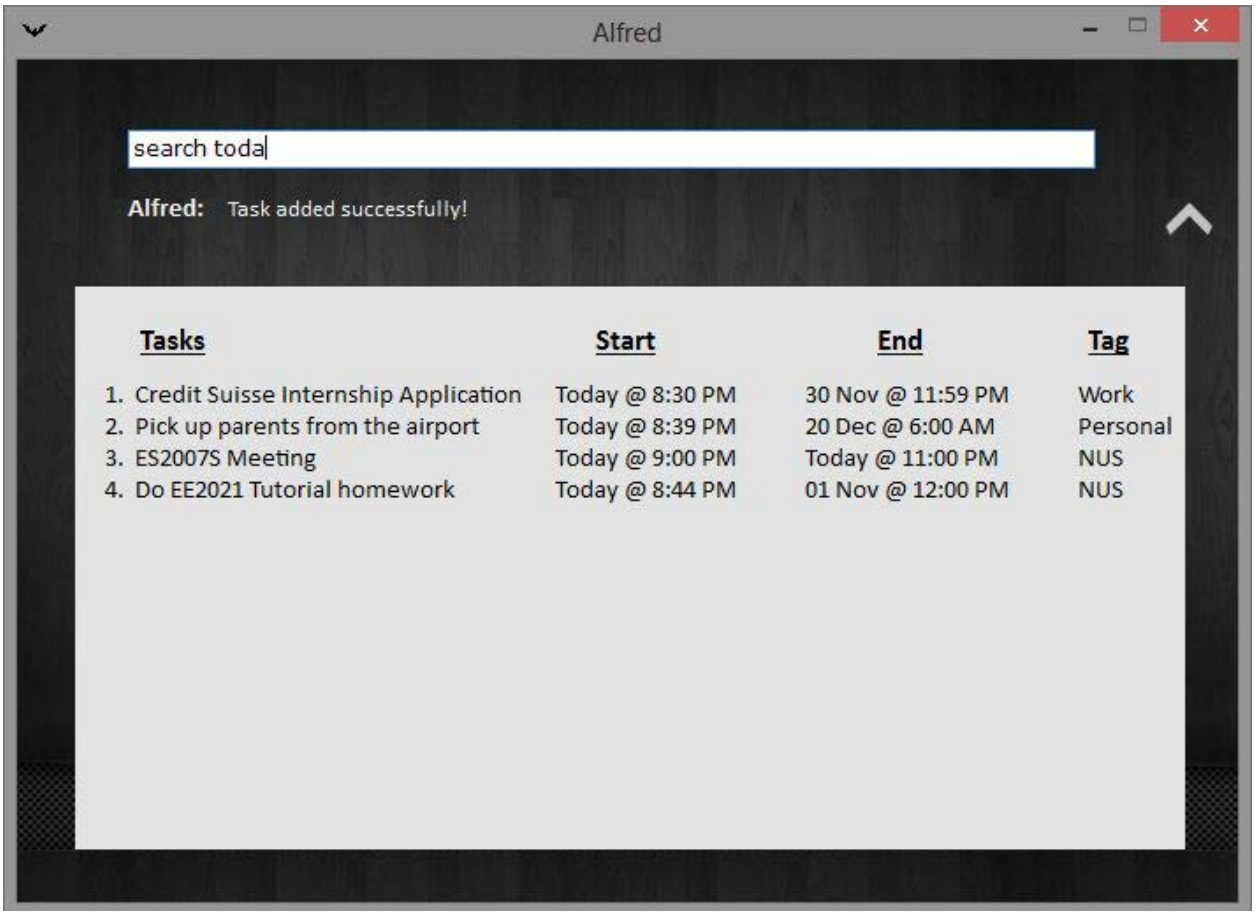
Tasks can be deleted in 3 formats. The task is prefixed with a “delete” keyword:

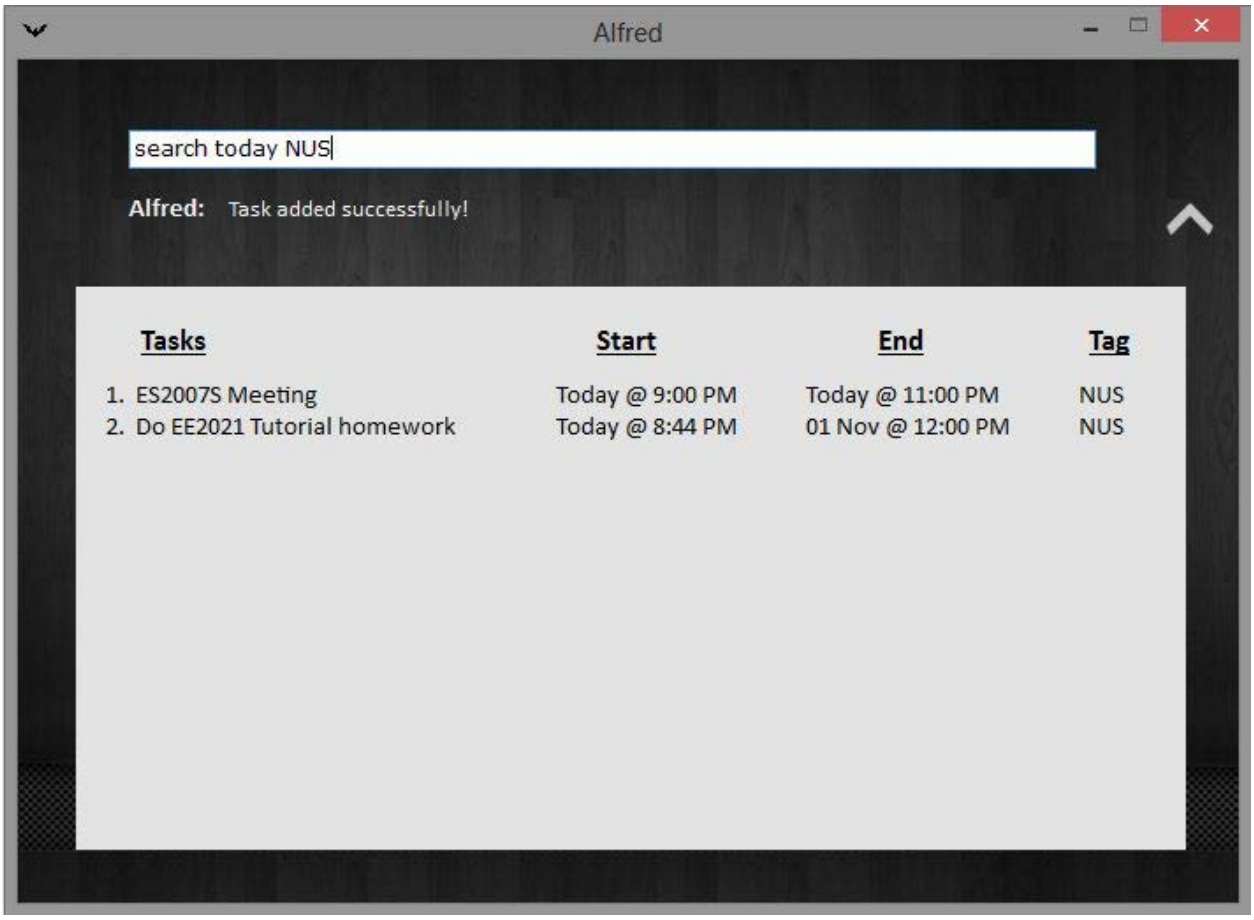
1. **Task number.** (Delete 2 – Deletes the 2nd task on the display list.)
2. ***Task name and date and time OR task name and date OR task day/date and time.** (Delete Canoe polo –ed 16 October 2012 - et 5pm – Deletes the task on the specified date, at the mentioned time.)
3. ***Recurring task name.** (Delete all Canoe polo – Deletes all instances of the mentioned task.)

V Power Search: (*some parts to be implemented)

Tasks can be searched for in 3 formats. Searches are performed real-time. The task is prefixed with a “search” keyword:

1. **Task name, and any and all combinations of start date, start time, end date, end time and tag.** (Search Canoe polo –ed 16 December 2012 –et 5pm – Returns the mentioned task on the specified date and time.)
2. **Task name.** (Search Canoe polo – Returns all the instances of the mentioned task.)
3. ***Near miss search.** If, while searching, a spelling mistake is made that leads to no matching being displayed, Alfred shows the previous list of matching tasks. (Search Canoe poli – Returns all Canoe polo tasks despite having misspelt command.)




Alfred
search today NUS
Alfred: Task added successfully!
Tasks
Start
End
Tag
1. ES2007S Meeting
Today @ 9:00 PM
Today @ 11:00 PM
NUS
2. Do EE2021 Tutorial homework
Today @ 8:44 PM
01 Nov @ 12:00 PM
NUS

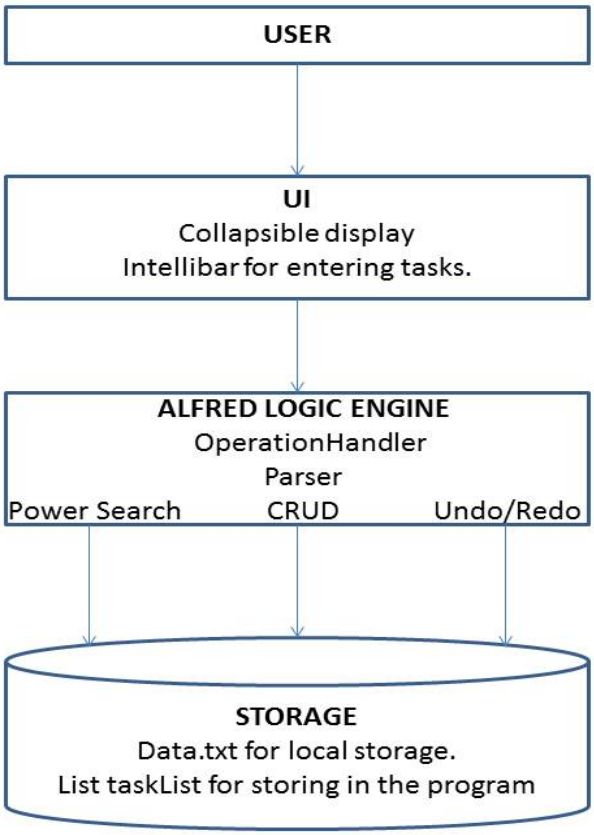
Search

Developer's Guide

Introduction

Welcome, new developer! Greetings from the Alfred team. This guide has been created just for you, and describes to you exactly how our code is structured. You will find below all the constituents that helps Alfred function. A brief architecture of the software is provided, followed by a comprehensive list of the APIs that have been used. This is followed by a few code snippets to display the flexibility in functionality of our software. You must know that our software is still under development, so we conclude the guide by indicating what we intend to achieve in the coming few versions. Read on to see what makes Alfred tick.

Architecture



Key points of Alfred's architecture are mentioned below –

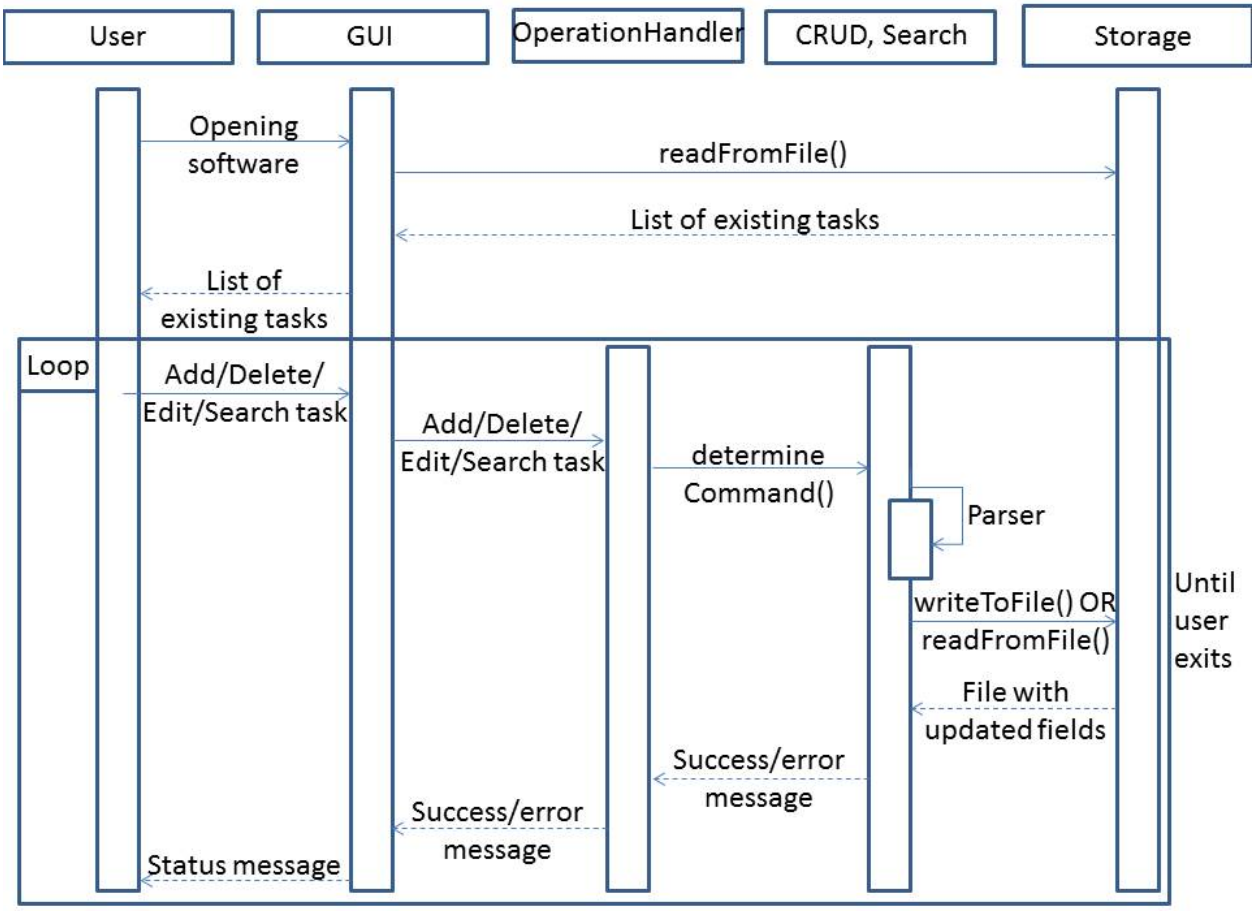
- Alfred possesses an *n-tier* architecture style. In such a style, as can be seen, each unit is linked to the unit below it. This holds true for Alfred as well.

- The *UI* unit takes in the command entered by the user in the Intellibar, and then relays the command to the *Logic Engine*. It also possesses a collapsible display showing the list of existing tasks.
- The *Logic Engine* unit takes in the user command, and links it to the *CRUD* (*Add*, *Edit* and *Delete*), *Search* or *Undo/Redo* classes using *OperationHandler*.
- The *CRUD, Search and Undo/Redo* classes are all independently related to the *Storage* unit, which creates *Data.txt* to store all existing tasks.

Design descriptions

Overall design using sequence diagram:

The sequence diagram to display the overall working of the software is shown below –



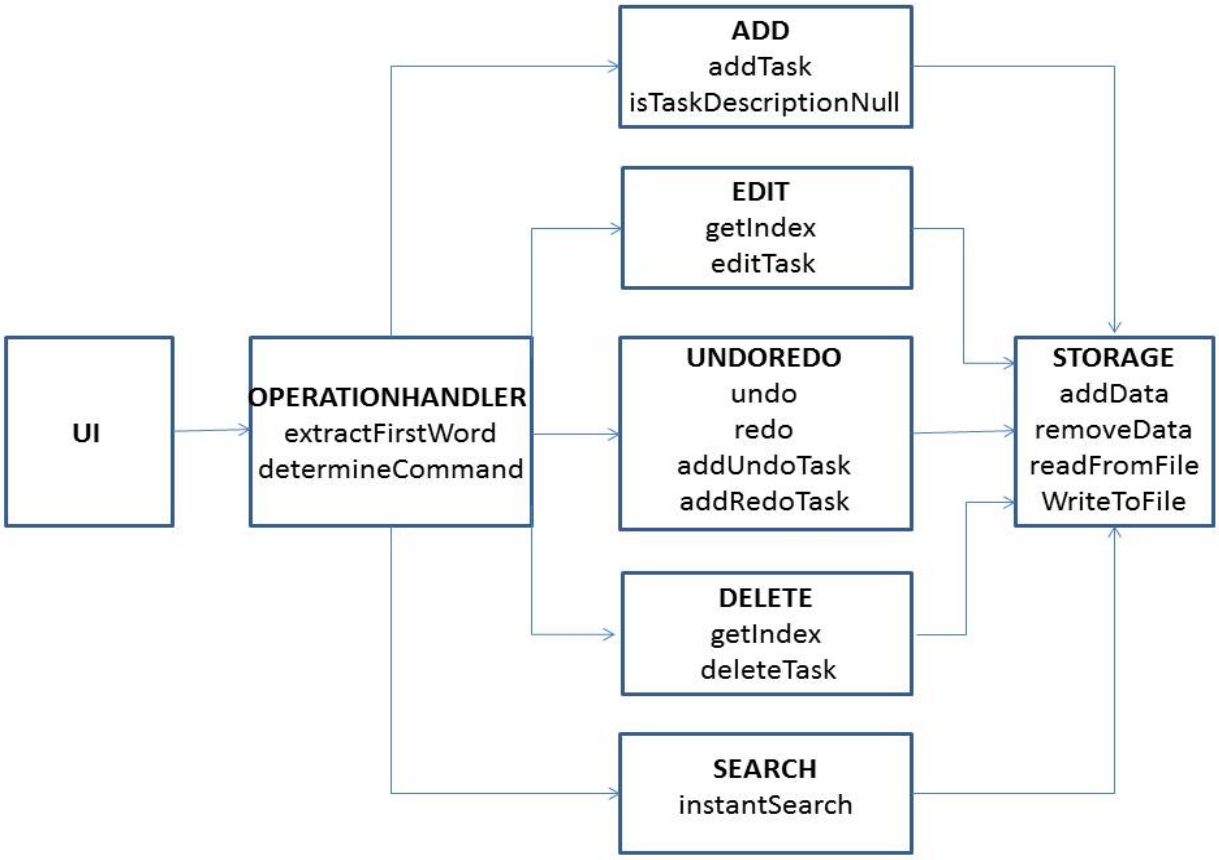
It is further explained in text below –

- When the .exe file is opened by the user, an instance of *UI* is created, which displays the condensed version of the *UI* at start-up.

- The *Storage* class returns the list of existing of tasks to the *UI* at start-up, so that on pressing the drop-down icon, the user can see the same.
- The user now enters some input, for adding, editing or deleting a task in the ways mentioned.
- The command is now sent to the *OperationHandler* class, which determines the type of command. It is then sent to the class that handles the respective operation.
- Under the respective *CRUD (Add, Delete, Edit)* and *Search* classes, the command is sent to the *Parser* class which converts input string to a readable form.
- The software-readable task is now sent back to the *CRUD* classes, which link with the *Storage* class.
- An operation success/error message is then sent to the *UI*, to display to the user.

Overall design using class diagram:

The class diagram to display the overall working of the software is shown below –



It is further explained in text below –

- The user interacts only with the *UI* throughout.
- The *OperationHandler* class links the *CRUD* (*Add*, *Edit* and *Delete*), *Search* and *UndoRedo* classes with the *UI*.
- These classes independently access *Storage* class simultaneously.

Important Application Programming Interfaces

Add.cs: Takes in the task that must be added, and creates an instance of the Parser class to break the command down into attributes of the newTask object. It then adds the task to Data.txt.

public void addTask(string input)	Adds a new task into Data.txt.
private void isTaskDescriptionNull()	Checks if task contains any description.

Delete.cs: Takes in the task number of the task that must be deleted, and deletes it on locating within Data.txt, provided the task number entered is valid.

public int getIndex(string input)	Gets index of task to be deleted in file.
public void deleteTask(string input)	Deletes task using index in file.

Edit.cs: Takes in the task number of the task that must be edited, and overwrites the task on locating within Data.txt, provided the command entered is valid.

public int getIndex(string input)	Gets index of task to be edited in file.
public void editTask(string input)	Edits task using index in file.

Search.cs: Takes in the string entered by the user and compares it with the list of all tasks, and returns matching instances.

public List<Task> instantSearch(string searchKeyword)	Gets string of task to be searched for, and returns all matching instances of the task after comparing.

OperationHandler.cs: Takes in the first word of the command that has been entered by the user, and hands over control to the class of the respective operation.

static string extractFirstWord(string input)	Extracts first word of input to get command type.
public static void determineCommand(string input)	Determines which class to hand input to be processed.

Parser.cs: Determines the way that the command entered by the user is interpreted by the software. It breaks down the command into the attributes of the Task class, and returns the Task object to the class from where the Task object had been called.

private string shortcutsToString(Utility.shortcuts current)	Adds hyphen to input.
private string parseEndDate(string input)	Gets end date from input.
private string parseEndTime(string input)	Gets end time from input.
private string parseStartDate(string input)	Gets start date from input.
private string parseStartTime(string input)	Gets start time from input.
private string parseTaskDescription(string input)	Gets task description from input.
private string parseTag(string input)	Gets tag from input.
private void combineDateTime(string input)	Combines date and time into a single string.
public Task returnTask(string input)	Gets Task object from parsed input.
private static void isDateTodayOrTomorrow(ref string endDate)	Checks if task's date is today or tomorrow, based on system time.
private static int indexBeforeNextHyphen(string input,	Locates next hyphen in string.

int startPosition, int endPosition)	

Storage.cs: Links the software with the Data.txt and is used to write to and delete data from the aforementioned file. It contains a working vector.

public void addData(Task newTask)	Adds new tasks to taskList.
public void removeData(int index)	Removes task from taskList.
public List<Task> readFromFile()	Reads data from text file and returns data as a List of Task objects.
public void writeToFile()	Writes data from given List of Task objects and stores it as a text file.

Task.cs: Defines what a task is supposed to contain, and is used to set/get the attributes that a normal task contains.

public void setEnd()	Sets date of task due date.
public void setTaskDescription()	Sets bulk of task info.
public void setIsStarred()	Sets importance of task.
public void setCategory()	Sets type of task.
public void setTag	Sets tag.

UndoRedo.cs: Undoes the last performed action, or redoes the previously undone action, based on the user's command.

public void addUndoTask()	Copies the taskList of Storage into the Undo stack on mentioning 'undo'.
public void addRedoTask()	Copies the taskList of Storage into the Redo stack on mentioning 'redo'.
public void undo()	Undoes the task.
public void redo()	Redoes the task.

Program.cs: Sets the program running when the .exe file is opened. Control is then handed over to the UI class.

UI.cs: Contains Windows form object initializations, event triggers and consequent functions, keyboard shortcuts and windows form component declarations.

UI.Designer.cs: Consists of the design for Alfred's GUI.

Utility.cs: Contains constants and enum types, to enhance code readability.

Code features & Instructions for testing

The features that have been implemented in Alfred thus far have been designed keeping in mind the heavy-keyboard user's convenience. That is why we have tried to keep it as simple and as command-line

oriented as possible. The following features can be tried out in the ways mentioned in the following page –

I Creating a task:

- The "Add" keyword doesn't have to be mentioned.
- While there is some rigidity in mentioning "-sd", "-st", "-ed", "-et" and "-t" to enter start date, start time, due date, due time and tag respectively, the start date and due date itself can be mentioned in a number of ways. Multiple formats such as 16 October 2012, 16 Oct 2012, 16/10/2012 and 16/10/12 can be used.
- While adding a task, if and when the start date, start time, end date, end time and tag are added, it can be done so in any order.
- Certain level of natural language processing (e.g – today, tomorrow) is also provided by the software.
- If the task is added correctly, a status message will be displayed – "Task added successfully!"

II Deleting a task:

- In the current version, only deletion by task number is supported.
- An error message supplied if an invalid delete command ("Please enter Delete <task number>") or an invalid task number ("Please enter a valid task number") is entered.
- If the task is deleted correctly, a status message will be displayed - "Task deleted successfully!"

III Editing a task:

- In the current version, only editing by task number is supported, and the entire task must be rewritten in the format used to add tasks.
- An error message is supplied if an invalid edit command ("Please enter Edit <valid task number> <-ed valid end date> <-et valid end time>") or an invalid task number ("Please enter a valid task number") is entered.
- If the task is deleted correctly, a status message will be displayed - "Task edited successfully!"

IV Searching for a task:

- In the current version, searching by task name and task parameters is supported.
- While there is some rigidity in mentioning "-sd", "-st", "-ed", "-et" and "-t" to enter start date, start time, due date, due time and tag respectively, the start date and due date itself can be mentioned in a number of ways. Multiple formats such as 16 October 2012, 16 Oct 2012, 16/10/2012 and 16/10/12 can be used.
- While searching for a task, if and when the start date, start time, end date, end time and tag are entered, it can be done so in any order.
- Searches are instant and display results in real–time (with each key that is pressed, the search results narrow down).
- In the current version, if the string does not match any of the tasks' parameters, an empty display screen will be shown. In the future versions, near-miss search will be implemented.

- If a task has been searched for correctly, the task(s) will be displayed.

V Miscellaneous:

- In general, an error message is supplied if an invalid command ("Please enter a valid command") is entered.
- An error message is supplied if invalid due date or time ("Please enter a valid deadline") is entered.
- Nifty keyboard shortcuts have been implemented to make Alfred easy-to-use. Ctrl + M minimizes the application to the system tray, with a notification. This can be brought back up again by double-clicking the icon in the tray. Ctrl + S expands/collapses the display, so that the list of existing tasks can either be seen or hidden.

The developer is expected to take note of the above and test each case rigorously.

Appendix

Logic for interpreting start date, start time, end date and end time:

start date	start time	end date	end time	start	end
0	0	0	0	maxvalue	maxvalue
0	0	0	1	Current date and time.	1. Current date and given time (if et > clock time). 2. Tomorrow's date and given time (if et < clock time).
0	0	1	0	Current date and time.	Input date and 23:59:59.
0	0	1	1	Current date and time.	Input date and input time.
0	1	0	0	Current date and input time.	maxvalue
0	1	0	1	Current date and input time.	1. Current date and given time (if et < clock time). 2. Tomorrow's date and given time (if et > clock time).
0	1	1	0	Current date and input time.	Input date and 23:59:59
0	1	1	1	Current date and input time.	Input date and input time.
1	0	0	0	Input date and current time.	maxvalue
1	0	0	1	Input date and current time.	1. Current date and given time (if et < clock time). 2. Tomorrow's date and given time (if et > clock time).
1	0	1	0	Input date and current time.	Input date and 23:59:59.
1	0	1	1	Input date and current time.	Input date and input time.
1	1	0	0	Input date and input time.	maxvalue
1	1	0	1	Input date and input time.	1. Current date and given time (if et < clock time). 2. Tomorrow's date and given time (if et > clock time).
1	1	1	0	Input date and input time.	Input date and 23:59:59.

1	1	1	1	Input date and input time.	Input date and input time.

Updated prospective project timeline:

Version No.	Implementation
0.3	Alarm Reminders, Undo, Redo, Near-miss search.
0.4	Intuitive GUI, Keyboard Shortcuts, Auto complete, Prompts.
0.5	Final Revision, Bug Fixes, Optimization.