ALFRED

Shourya Moona

[Software Developer, GUI, Testing]

Rohit Mukherjee

[Software Developer, Testing]

Lee Jianwei

[Software developer, Documentation]

Madhav Kannan

[Software architect, Documentation]

CONTENTS

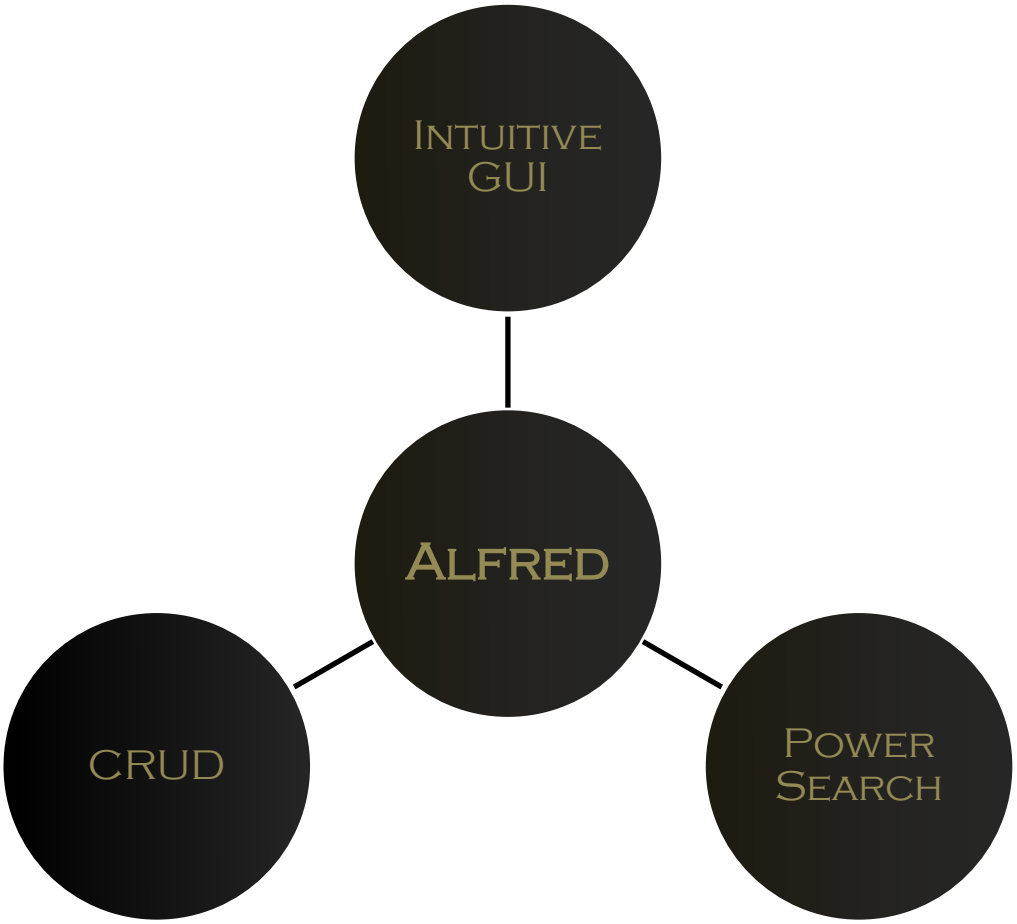
USER GUIDE

DEVELOPER GUIDE

USER GUIDE

INTRODUCTION

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. 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

Launching Alfred:

- Launching Alfred is very easy. Just click on the.exe file, and you're good to go!
- Alfred works only on Windows.

GETTING STARTED

I Creating a Task:

The new task can be entered in the formats mentioned below. The task may be prefixed with an "add" keyword, but it is not necessary.

Floating tasks – Tasks with no time component.
add <Task description> –t <Tag>
e.g. add Canoe polo –t Recreation

Deadline tasks – Tasks with a deadline component.
add <Task description> –et <End time> -ed <End date> -t <Tag>
e.g. add Canoe polo –et 5 pm –ed 16 December 2012 –t Recreation

Timed Tasks – Tasks with a fixed duration.
add <Task description> -st <Start time> -sd <Start date> -et <End time> -ed <End date> -t <Tag>
e.g. add Canoe polo –st 2 pm –sd 16 December 2012 –et 5 pm –ed 16 October 2012 –t Recreation

*Any and all combinations of start date, start time, end date, end time and tag can be entered, and in any order. Only task description is a necessary requirement, and must be stated first.
**Multiple formats of date such as 16.10.12, 16.10.2012, 16/10/2012 and 16/10/12, today and tomorrow can be used.

II Deleting a Task:

Tasks can be deleted in the formats mentioned below. The command is prefixed with different keywords as shown below.

Deleting a previously entered task.
delete <Line number>
e.g. delete 1

Deleting all previously entered tasks.
clear
e.g. clear

GETTING STARTED

III Reading a Task:

The default dropped-down screen shows users all the tasks that have been updated into the task manager. Tasks are sorted according to their deadline. Completed tasks are stricken off to signify completion.

IV Updating a Task:

Tasks can be edited in the formats mentioned below. The task is prefixed with different keywords as shown below.

Overwriting a previously entered task.

edit <Line number> <Task description> -st <Start time> -sd <Start date> -et <End time> -ed <End date> -t <Tag>

e.g. edit 1 Canoe polo –st 2 pm –sd 16 December 2012 –et 5 pm –ed 16 October 2012 –t Recreation

Undoing last action.

undo

Redoing last action.

redo

Completing a previously entered task.

complete <Line number>

e.g. complete 1

Undoing the completion of a previously completed task.

uncomplete <Line number>

e.g. uncomplete 1

*For overwriting the task, if start date or start time are to be updated, start date and start time must be entered. Same goes for end date or end time as well. In other cases, only the field that must be updated needs to be changed.

**Completed tasks continue to remain on the display screen. However, they are stricken off to signify completion.

GETTING STARTED

V Searching for a Task:

Tasks can be searched for in the formats mentioned below. Searches are performed real-time, and the results are instantly refined. Task description, and any and all combinations of start date, start time, end date, end time and tag can be searched for, simultaneously. The command is prefixed with different keywords as shown below.

Searching for a previously entered task.

search <keyword(s)>

e.g. search Canoe polo

e.g. search Canoe polo 16 December 2012 5pm

Searching for completed tasks.

archive

e.g. archive

Searching for all tasks.

all

e.g. all

Near-miss Search – Displaying the likeliest matches.

search <keyword(s)>

e.g. search Canoe poki – Returns all Canoe polo tasks despite having misspelt command.

*Multiple formats of date such as 16.10.12, 16.10.2012, 16/10/2012 and 16/10/12, today and tomorrow can be used.

GETTING STARTED

VI Miscellaneous:

Alfred's miscellaneous user-friendly features are shown below. It's what makes Alfred so easy to use!

Alfred Status bar

Alfred's status bar is designed to guide the user while they're navigating through Alfred's different features. On start-up, the status bar displays the list of accepted command-keywords to the user. Alfred also indicates a success/error message every time a command is entered. If there is an error in accepting the command, the cause of error is also mentioned.

Auto-Complete

When the user starts typing a command-keyword, the remaining portion of the keyword is automatically completed by Alfred.

Prompts

When any of the aforementioned command-keywords are mentioned, Alfred's status bar displays the format in which the remaining command must be entered, to ensure that the user avoids errors.

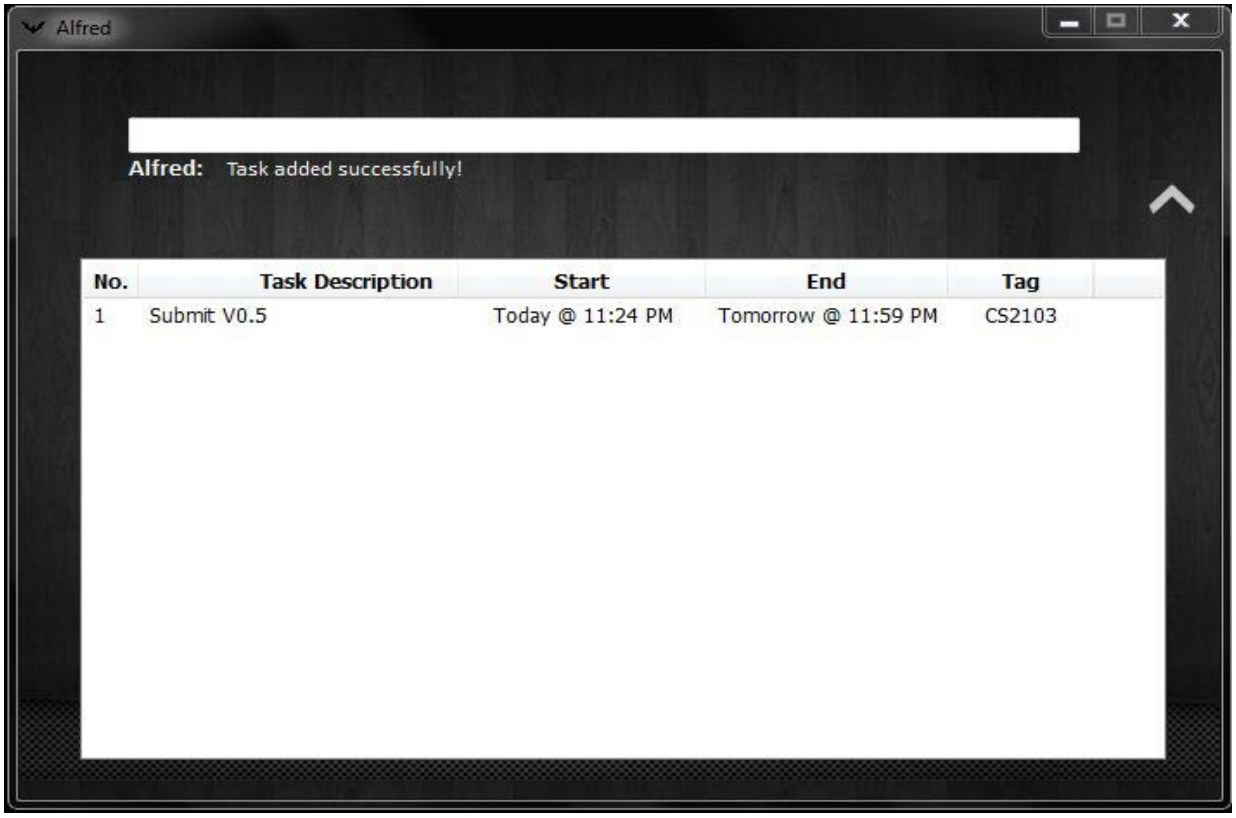
Help feature

If while using Alfred, the user has any doubts with regard to its working, they can just type the command "help" in the *Intellibar*. This throws up a web page that concisely indicates how Alfred's functionality can be exploited to the fullest.

Keyboard shortcuts

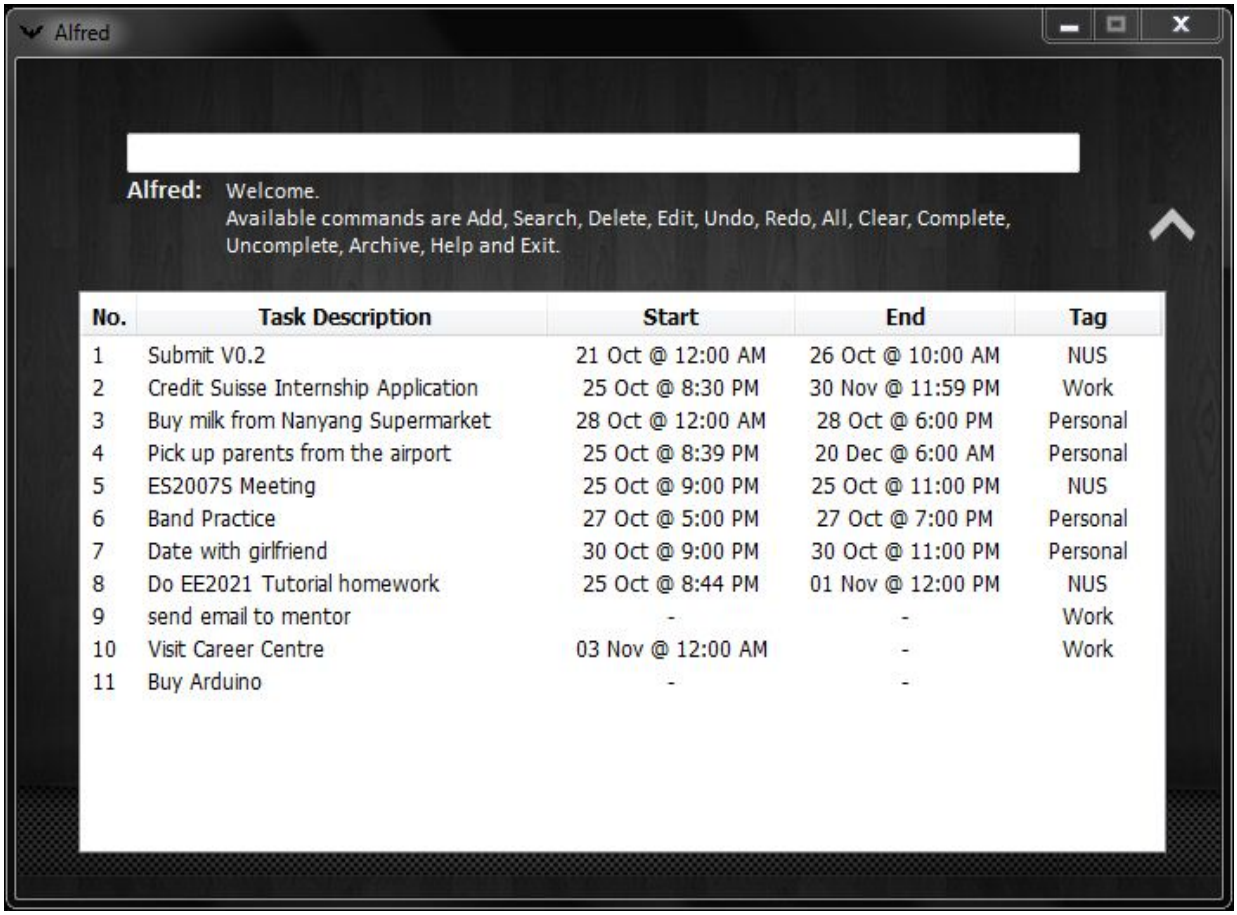
Alfred possesses a couple of keyboard shortcuts that are intended to simplify usage –
Ctrl+S – Toggle between expanded view and collapsed view.
Ctrl+M – Minimize to system tray with Alfred still running.

SCREENSHOTS



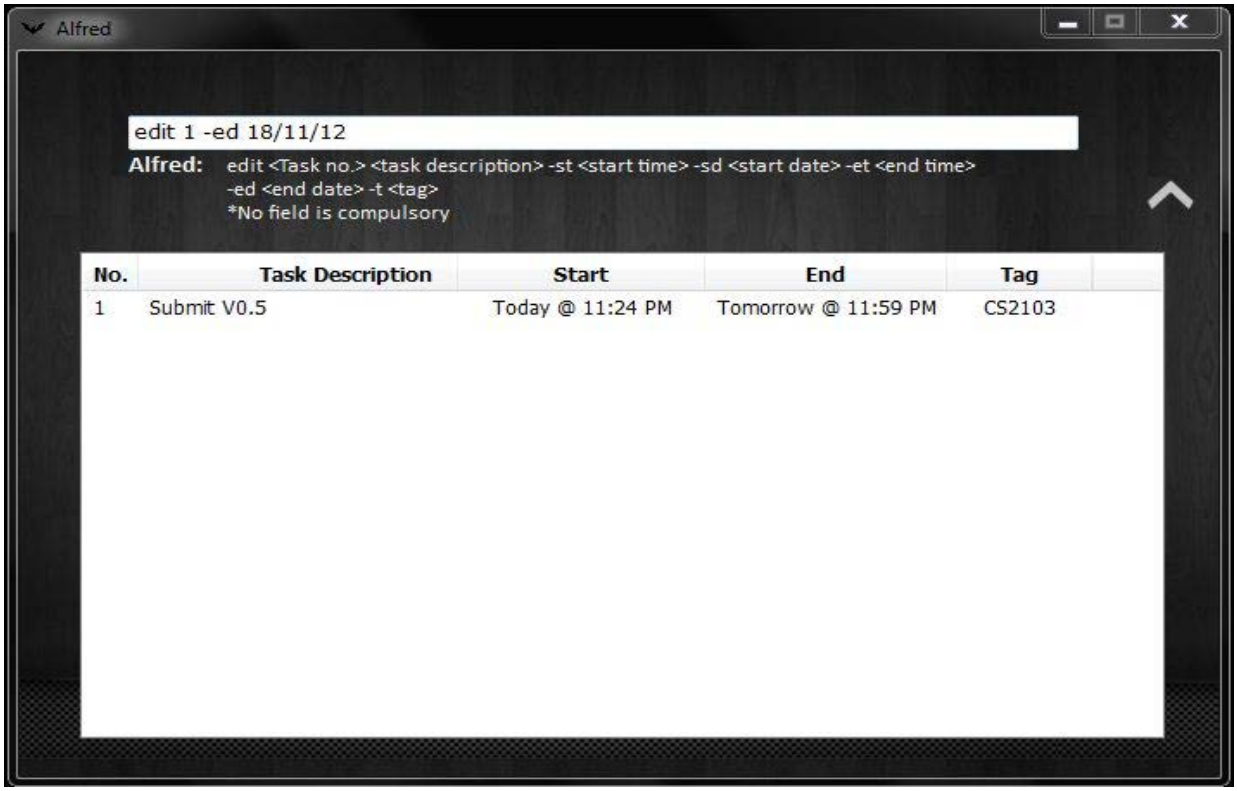
Adding a task

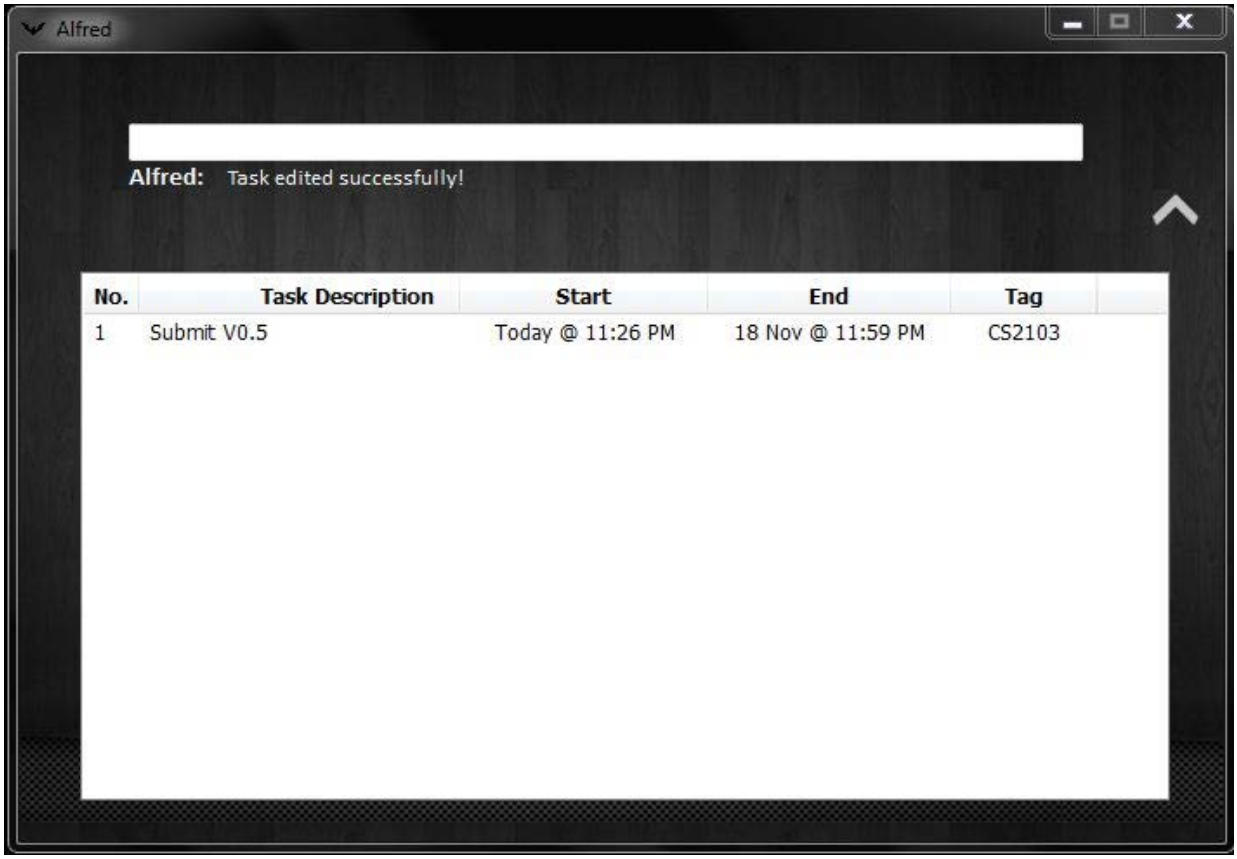
SCREENSHOTS



Display (Expanded view)

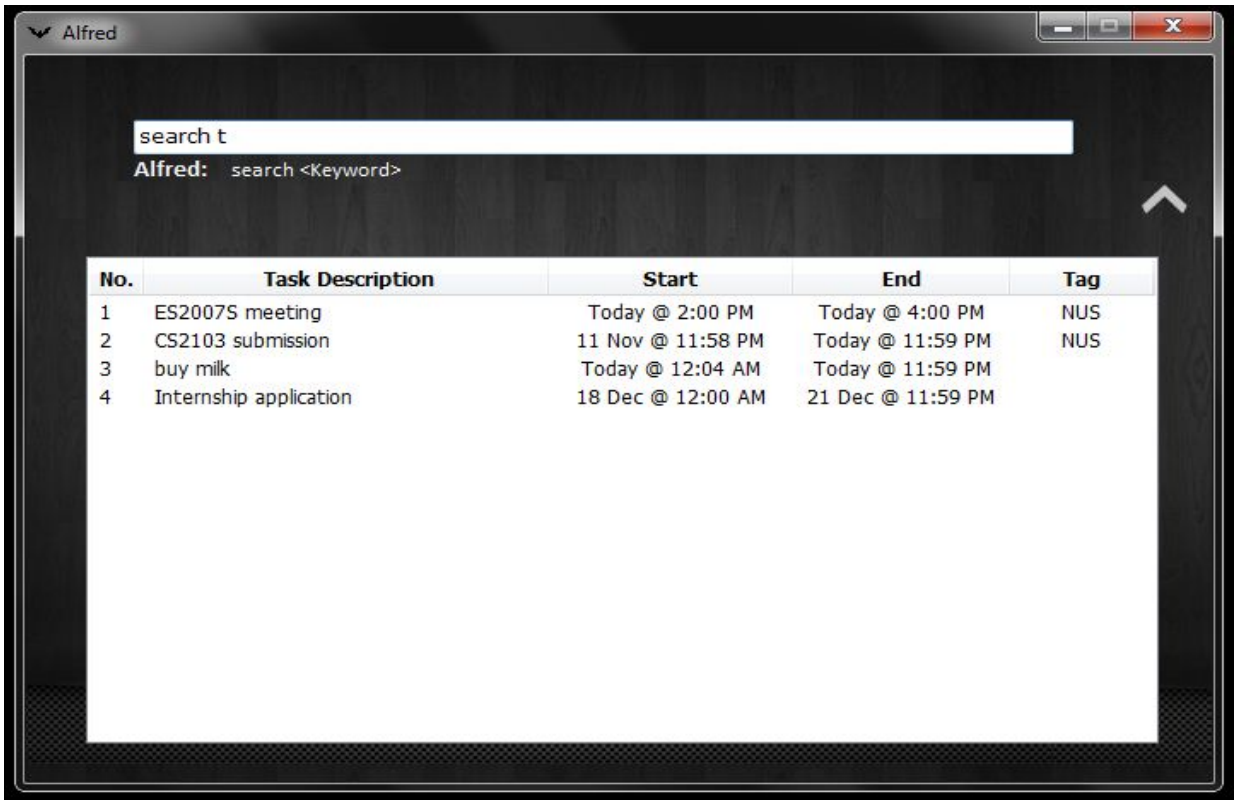
SCREENSHOTS

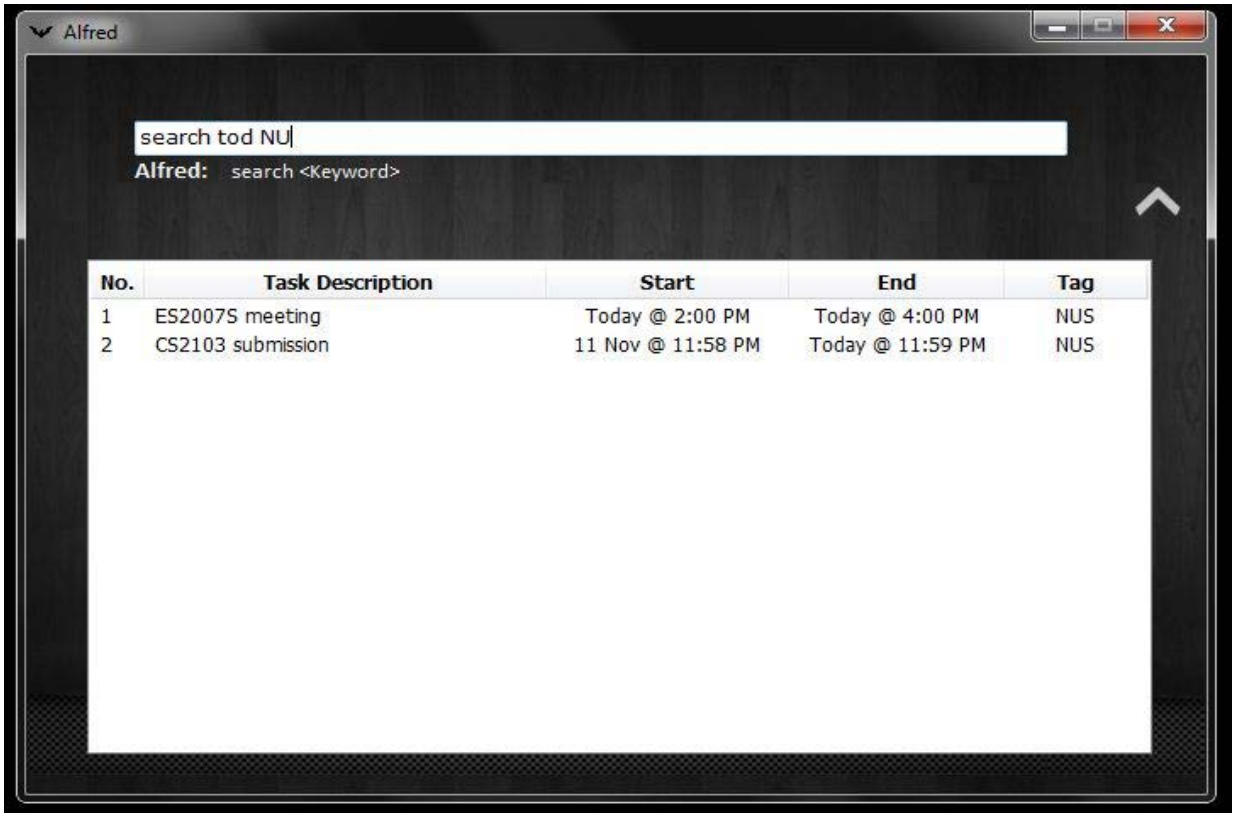




Editing a task

SCREENSHOTS

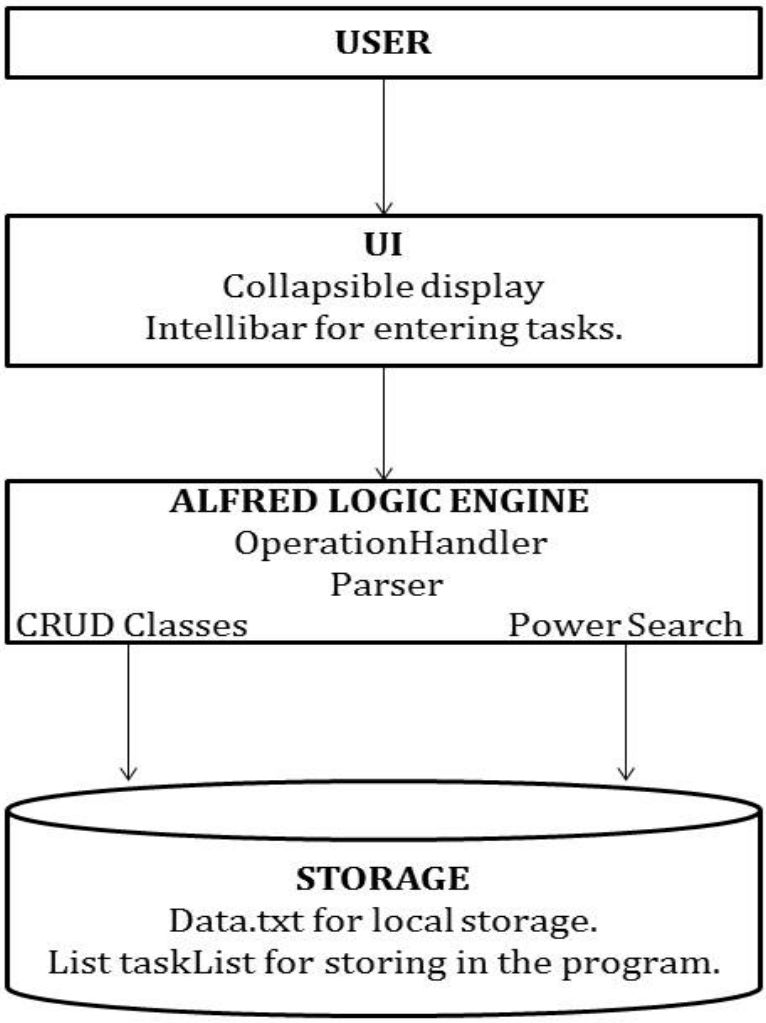




Searching for a task

DEVELOPER GUIDE

ARCHITECTURE

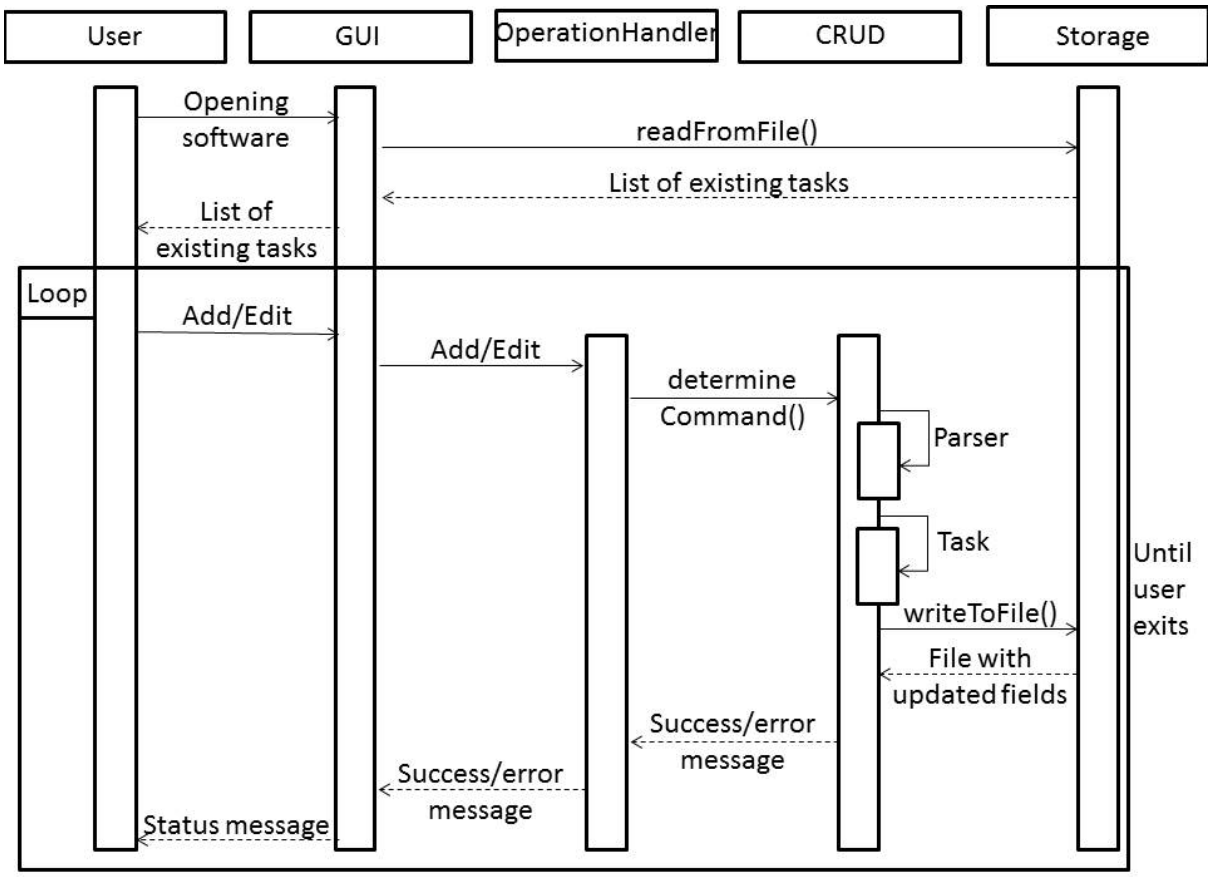


Key points of Alfred's architecture are mentioned below –

- Alfred possesses an *n-tier* architecture, implemented in top-down style. In such an architecture, as can be seen, each unit is linked to the unit below it. This holds true for Alfred as well.
- The *UI* unit accepts 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, UndoRedo, Complete* and *Delete*), and *PowerSearch* classes using *OperationHandler*. The *Add* class makes use of the *Parser* class to decipher the input.
- The *CRUD* and *PowerSearch* classes are all independently related to the *Storage* unit, which creates *Data.txt* to store all existing tasks.

DESIGN DESCRIPTIONS

I Sequence Diagram for Add and Edit tasks:



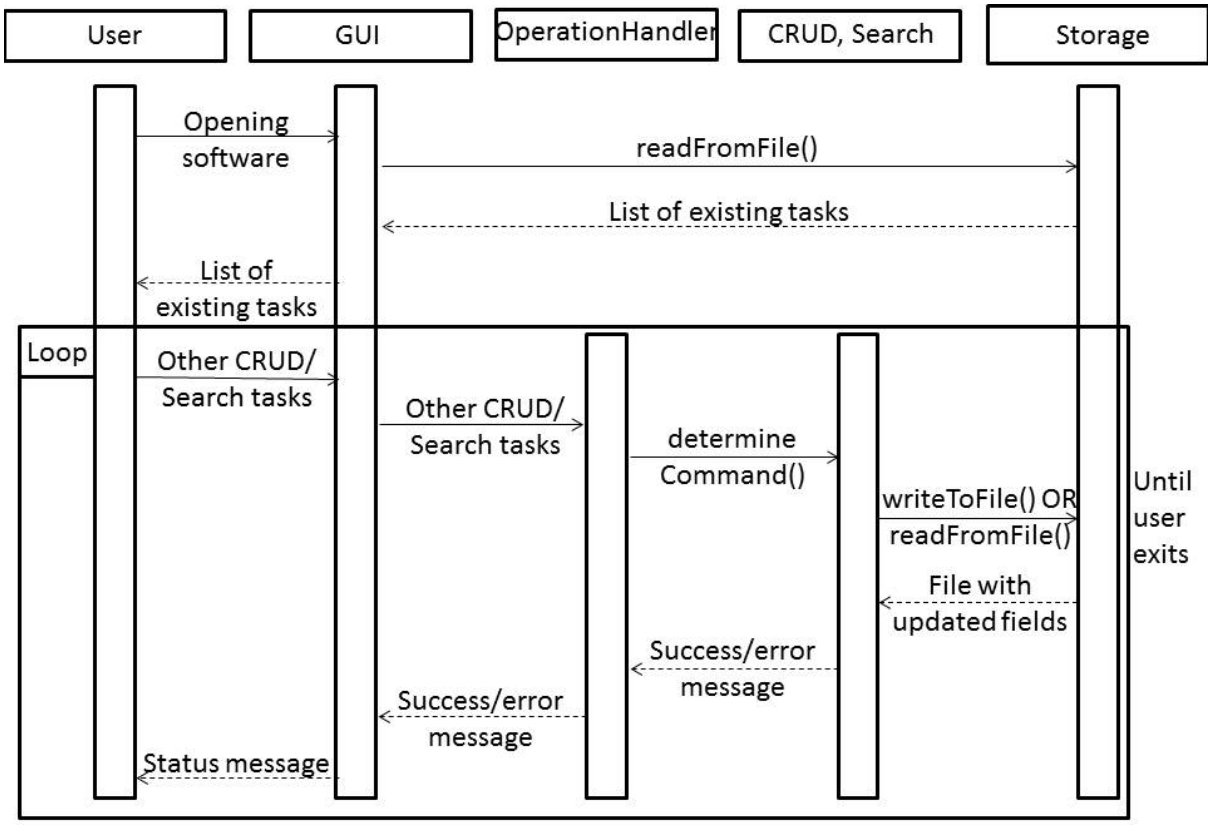
It is further explained in text below –

- 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, or editing a task in the format mentioned.
- The command is now sent to the *OperationHandler* class, which determines whether it is an 'add' or an 'edit' command. It is then sent to the class that handles the respective operation.
- Under the respective *Add* and *Edit* classes, the command is sent to the *Parser* class which converts input string to a readable form.
- An instance of *Task* is then created so that a new task can be created or an existing task can be updated respectively.

DESIGN DESCRIPTIONS

- The software-readable task is now sent back to the *Add* and *Edit* classes, which link with the *Storage* class and update the *taskList* within the software as well as the *Data.txt* file.
- An operation success/error message is then sent to the *UI* to display to the user, based on the success/error of the previous steps.

II Sequence diagram for other CRUD features (Delete, Clear, Undo, Redo, Complete, Uncomplete) and Search (Search, Archive, All):



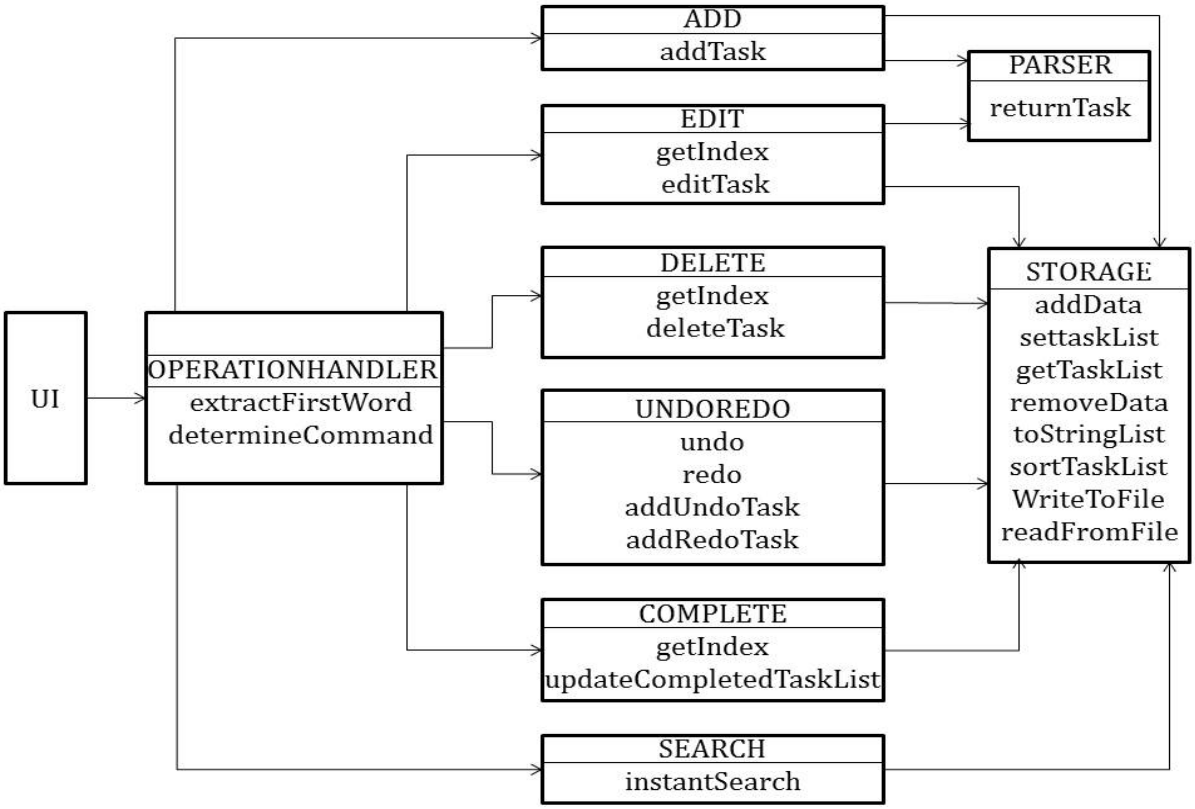
It is further explained in text below –

- 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 accomplishing all other *CRUD* tasks or to search for a task in the format mentioned.

DESIGN DESCRIPTIONS

- 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* or *Search* classes, the command links with the *Storage* class and updates the *taskList* within the software as well as the *Data.txt* file.
- An operation success/error message is then sent to the *UI* to display to the user, based on the success/error of the previous steps.

III Overall Class diagram:



It is further explained in text below –

- The user interacts only with the *UI* throughout.
- The *OperationHandler* class links the *CRUD* (*Add*, *Edit, Delete, UndoRedo* and *Complete*) and *Search* with the *UI*.
- The *Add* and *Edit* classes access *Parser*, while all of them independently access the *Storage* class simultaneously, and use its methods to update the *Data.txt* file.

LIST OF APIs

I 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.

II 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.

III 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.

IV Complete.cs

Takes in the task number of the task that must be completed, and updates the task's status on locating within Data.txt, provided the command entered is valid.

- public int getIndex(string input) - Gets index of task to be completed in file.
- public void updateCompletedTaskList() – Updates the status of the task to 'Completed' using index in file.

LIST OF APIs

V 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.

VI PowerSearch.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.

VII 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.

LIST OF APIs

VIII 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.

- public Task returnTask(string input) - Gets Task object from parsed input.

IX 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.
- public static List<Task> getTaskList() – Returns the taskList vector.
- public static void settaskList(List<Task> newlist) – Sets taskList.
- public static void toStringList() – Converts the taskList into a String.
- public static void sortTaskList() – Sorts the taskList in chronological order of deadline.
- public static List<Task> getTaskListShallowcopy() – Returns a shallow copy of the taskList, wherein changes are not reflected in the main taskList.
- public static List <String> getStringTaskList() – Returns a vector containing every task in String type.

X Utility.cs

Contains constants and enum types, to enhance code readability.

LIST OF APIs

XI Task.cs

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

- public DateTime setEnd() – Sets the task deadline.
- public DateTime setStart() – Sets the task start time.
- public string setTaskDescription() Sets the information of the task.
- public string setCategory() – Sets the task category.
- public string setTag – Sets the task tag.
- public bool setIsCompleted – Sets the completeness of the task.
- public static int CompareDates(Task x, Task y) – Checks which of the two tasks has its deadline first.
- public Task getShallowTask() – Returns a shallow copy of a particular task.

XII Program.cs

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

XIII UI.cs

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

XIV UI.Designer.cs

Consists of the design for Alfred's GUI.

XV Log4Net External API

Used for logging purposes.

CODE ASPECTS & 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 command-line oriented as possible. These aforementioned features can be tried out in the ways mentioned in the following pages –

I Creating a task:

- Certain level of natural language processing (e.g – today, tomorrow) is also provided by the software.

```
private static void isDateTodayOrTomorrow(ref string endDate)
{
        if (string.Equals(endDate, Utility.DATE_TODAY,
        StringComparison.CurrentCultureIgnoreCase) == true)
        {
                endDate = DateTime.Today.ToShortDateString();
        }
        else if (string.Equals(endDate, Utility.DATE_TOMORROW,
        StringComparison.CurrentCultureIgnoreCase) == true)
        {
                endDate = DateTime.Today.AddDays(1).ToShortDateString();
        }
 }
```

- Tasks are sorted based on deadline, and on adding, these tasks are added dynamically (entered in the right position) to the list of all tasks.

```
public static void sortTaskList()
{
        taskList.Sort(Task.CompareDates);
}
```

II Deleting a task:

- 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.

III Editing a task:

- 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.

CODE ASPECTS & TESTING

IV Searching for a task:

- Searches are multiple-field, instant and display results in real–time.
- Near-miss search has also been implemented. If while searching, a spelling mistake is made that leads to no matches being found, Alfred shows the previous list of matching tasks.

IV Defensive Coding:

- Exception handlers and assertions have been coded into the project, to ensure the project does not terminate, regardless of any input entered by the user.

```
Debug.Assert(newTask != null);
      taskList.Add(newTask);
      sortTaskList();
```

```
try
{
        index = int.Parse(input);
        if (index > Storage.getTaskList().Count())
        {
                return Utility.TASK_OVERFLOW;
        }
        return index;
}
catch
{
        return Utility.INVALID_INDEX;
}
```

- The *Log4Net API* has been integrated into the project, logging the functioning of every method, so that Alfred's functioning can be viewed systematically. This is written into the file *Alfred-log.txt*.

```
ILog log = LogManager.GetLogger(typeof(Edit));
log4net.Config.XmlConfigurator.Configure();
log.Warn(Utility.LOG_INVALID_TASK_DATA_PROMPT);
```

V Unit Testing:

- Alfred contains over 50 different test methods in the *TestProject* folder, which test each individual functionality of the software, while also testing the logic as a whole.
- The test cases are exhaustive and thorough, with a number of possible user inputs to test the application.

CODE ASPECTS & TESTING

- Alfred also consists of independent checks for important functions in all classes, and contains ordered testing capabilities to test all the test cases at once.

```
[TestMethod()]
        public void EditTaskDescriptionTest()
        {
                Add adder = new Add();
                Edit target = new Edit();

                adder.addTask(Utility.TEST_EDIT_TASK_TASK);
                target.EditTask(Utility.TEST_EDIT_TASK_TASK_DESCRIPTION);

                string expected = Utility.TEST_EDITED_TASK_DESCRIPTION;
                string actual=Storage.getTaskList()[0].setTaskDescription;

                Assert.AreEqual(expected, actual);

                expected = Utility.TEST_EDIT_TASK_INITIAL_START;
                actual = Storage.getTaskList()[0].setStart+" ";

                Assert.AreEqual(expected, actual);

                expected = Utility.TEST_EDIT_TASK_INITIAL_END;
                actual = Storage.getTaskList()[0].setEnd + " ";

                Assert.AreEqual(expected, actual);

                expected = Utility.TEST_EDIT_TASK_INITIAL_TAG;
                actual = Storage.getTaskList()[0].setTag;

                Assert.AreEqual(expected, actual);
        }
```

VI Miscellaneous:

- Alfred possesses an n-tier architectural style, implemented with the top-down approach. It conforms with the Object-Oriented paradigm.
- The software makes use of the *Separation of Concerns* principle, while also implementing the *Singleton* and *Façade* patterns.

The developer is expected to take note of the above features.

APPENDIX

I 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.

APPENDIX

II Change Log:

Version No.	Changes
0.1	Basic CRUD features (Add, Edit, Delete) were implemented, along with an advanced UI
0.2	PowerSearch and Undo/Redo were implemented
0.3	Clear, Complete and Uncomplete were implemented, Edit was made more flexible
0.4	Near-miss search, Help, Archive, List-view UI were implemented
0.5	Bug fixes, Code refactored

III Sample Alfred-Log.txt:

Alfred: 2012-11-12 17:49:42,138 [9] INFO Alfred.OperationHandler Determine Command Started
2012-11-12 17:49:42,154 [9] INFO Alfred.OperationHandler Exit command identified
Alfred: 2012-11-12 17:50:01,156 [1] INFO Alfred.OperationHandler Determine Command Started
Alfred: 2012-11-12 17:51:39,629 [9] INFO Alfred.OperationHandler Determine Command Started
2012-11-12 17:51:39,629 [9] INFO Alfred.OperationHandler Add command identified
Alfred: 2012-11-12 17:51:39,645 [9] INFO Alfred.Parser Returning task from Parser class
Alfred: 2012-11-12 17:51:39,645 [9] INFO Alfred.Parser Floating Task
Alfred: 2012-11-12 17:51:39,645 [9] WARN Alfred.Parser start date not specified
2012-11-12 17:51:39,660 [9] INFO Alfred.Add Add Function working
2012-11-12 17:51:40,355 [9] INFO Alfred.OperationHandler Exit command identified