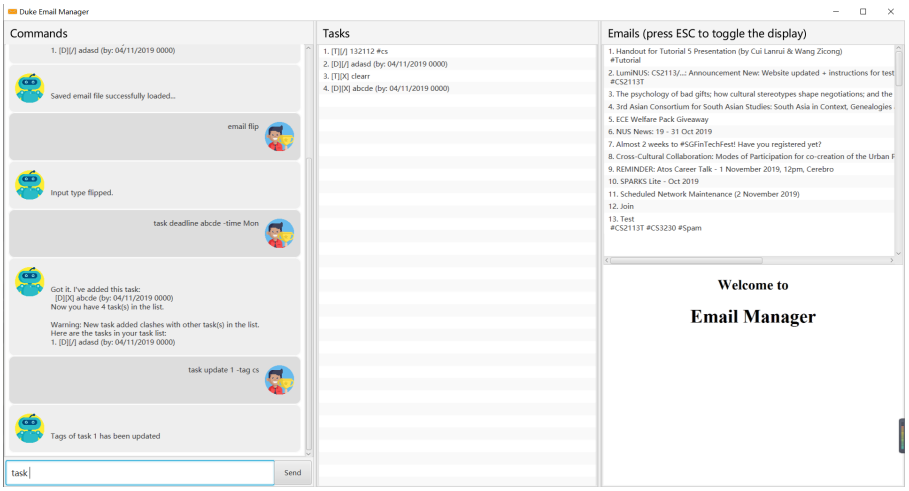
Project Portfolio Page

By: F11-3 Yan Zhiwen

Since: Oct 2019

1. Product Overview



After being constantly spammed by the school emails and missing all kinds of important emails, our group decided to develop this email manager to better organize our mailbox as a NUS Computing student.

Email Manager is an email and task manager app, specifically designed for NUS School of Computing students to manage their emails and busy schedules. As a text-based application, it is optimized for those who prefer typing and working with Command Line Interface (CLI). Email Manager also has a developed Graphical User Interface (GUI) that allows users to view email and task details in an appealing, well-organized format. If you can type fast, Email Manager can get your emails and tasks organized quickly and easily.

2. Summary of Contributions

- Code Contributed
- Feature Implemented (starting from the most significant):
  1. Email format and content(keyword) auto parsing
  2. Email keyword adding and highlighting
  3. Email fuzzy search
  4. Task reminder
  5. Infrastructure on Email fetching over Internet, Storage, Command Parsing etc.
- Other Contributions
  1. Cross-platform and project structure of release

3. Contributions to User Guide

I wrote for the quick start guide in the user guide, as well as the introduction to some of the features I implemented.

3.1. Quick Start

This section serves as a tutorial for a new user to Email Manager.

3.2. Installation

1. Ensure you have Java version 11 or later installed in your computer.
2. Download the latest Jar file here and copy it to a new folder.

3.3. Run Email Manager

WARNING	The program might freeze for a while when fetching and parsing the email. Please wait patiently.

3.3.1. Windows User

1. Double click the jar file.
2. If any error occurs, try to follow the mac/linux user instructions to run the program from command line.

3.3.2. Mac/Linux User

1. Open the command prompt.
2. Change the working directory of the command prompt to the extracted program folder using `cd`

command.

3. Check java version using `java -version` command. It should be java 11.
4. Run `java -jar [CS2113T-F11-3][Email_Manager].jar` and you are good to go.

TIP	Please send an email to jokeryan1997@gmail.com if you cannot successfully run the program.

3.3.3. Use and Test with/without Internet

Our product can automatically parse and manage the emails in your mailbox. It is best to be used and tested with Internet connection. If you do not have Internet connection, we have also prepared some data for you to test offline.

- If you have Internet connection:
  1. You will be directed to a web page to authorize our access to your mailbox. The program is set to login to the **dummy email** address automatically, but if for some reason it does not work, please login to the **dummy email** we provided to you. (Dummy email account: cs2113t@outlook.com, Password: nusf11-3)
  2. Close the browser and return back to the Email Manager. Wait for a while for the email to be fetched and parsed.

WARNING	Our product does not support non UTF-8 characters. Using your own mailbox for testing may lead to redundant saving and loading with the current version. More charsets will be supported in v2.0

- If you do not have Internet connection:
  1. Start the program once first to generate the file structure.
  2. Paste everything under `/data/test_data` file in the program folder, to the `/data` folder. Replace all the files in the `/data` folder.
  3. Close the Email Manager and Start it again.
  4. Wait for a while for all the emails to be parsed.

WARNING	Any hanging or "not responding" when first launch the app is normal, please wait for the app to get ready.

3.4. Email Features

The command listed here can only be used under email mode/prefix. Use `flip` command to toggle the mode.

3.4.1. Fuzzy search on emails

Format: `fuzzySearch TARGET`

This feature searches for the target string across all emails including their subject, sender and body, with some tolerance of difference. This tolerance is represented by the **edit distance** between two words. For more details about the **edit distance**, you can find more details here. The lower the **edit distance** is, the more similar are these two strings. This helps you to search through the emails even if you made some typos or when you are not sure about some names.

This fuzzy search functionality tolerates up to an **edit distance** of 2 and is insensitive to cases. This means "CS2102" will match to "CS2100" and "S210" but not "CS2211".

It will list all the emails which contain words that match the target string in their subject, sender or body. Emails are listed in descending order of a **relevance score**. The general idea is, the higher the score, the more relevant is the email. Lower **edit distance**, more occurrence, or occurrence in subject and sender will all contribute to a higher **relevance score**.

NOTE	For performance reasons, fuzzy search will only be done word by word. For example, if "project demo" is to be searched through a sentence "This is a project demo", both "project" and "demo" will be compared against "this", "is", "a", "project", "demo" and produce a relevance score.
NOTE	Using short target string like "is" is not recommended, since it can be matched to many other words like "a", "I", "am" etc, which appears in almost every email.

Examples:

```
fuzzySearch project demo
```

3.4.2. Auto parsing and tagging of emails

This feature is automatically triggered by fetching emails from Outlook Server and adding new kayword. It will automatically parse "keywords" that might be shown in your emails in different forms/expressions.

For example, the keyword `CS2113T` might be in the expression of `CS2113T`, `CS2113`, `Akshay Narayan`, `Akshay` etc. All emails containing these expressions will be recognized as containing the keyword `CS2113T`. This helps the email manager to better categorize your emails. For example, some emails might not explicitly include the module code, still belong to that module as it is sent by the lecturer.

For easier usage and testing, we have prepared a few default keyword + expressions pair as shown in Figure 3, which will be loaded when the program starts.

Keyword	Expressions
CS2113T	CS2113
	TAN KIAN WEI, JASON
	Leow Wei Xiang
	Akshay Narayan
	Akshay
CS2101	CS2101
	Anita Toh Ann Lee
CG2271	CG2271
	Djordje Jevdjic
CS2102	CS2102
	Adi Yoga Sidi Prabawa
CS3230	CS3230
	Divesh Aggarwal
CEG Admin	Low Mun Bak
SEP	SEP
	Student Exchange Programme
Tutorial	Tutorial
Assignment	Assignment
Spam	UHC Wellness
	luminus-do-not-reply
	NUS Libraries

If the email contains some expressions, the respective keyword will be used to tag that email. It can be seen from the # sign on the email list. When showing the email using `show` command, the parsed expressions will also be highlighted in yellow.

NOTE The parsing only allows exact matching of words, which means expression `2113` will not be matched with text `CS2113`. Any extra characters before and after the word will not be accepted. This is to avoid unintentional match for short expressions. Please add more expressions if you want more matching patterns.

NOTE Some emails have pictures in it, and obviously we do not intend to capture the words in those pictures.

TIP To test this feature, you are recommended to use `addKeyword` command, or sending email to the dummy mailbox. Remember to call `fetch` command after sending and leave a few seconds for the email to be transmitted before calling the `fetch`.

3.4.3. Add Keyword: `addKeyword`

Format: `addKeyword KEYWORD -exp EXPRESSION1 [-exp EXPRESSION2]…`
Adds a keyword or expressions to the keyword list. If the keyword is already in the list, the expression will be added to the expressions belonging to that keyword. So this is a command to add both keyword and expression.

TIP The easiest way to verify whether this command is working properly is to check whether the email containing the expression has a tag of this keyword in the email list, or just use `show ITEM_NUMBER` command to see whether the expressions are correctly highlighted.

NOTE All emails will be parsed again upon the updates in keyword and expression, so the window might freeze for a while. Please wait patiently.

4. Contribution to Developer Guide

I wrote designs of our programs and the sequence diagram of the features I implemented in our developer guide.

4.1. Design

4.1.1. Overall Architecture

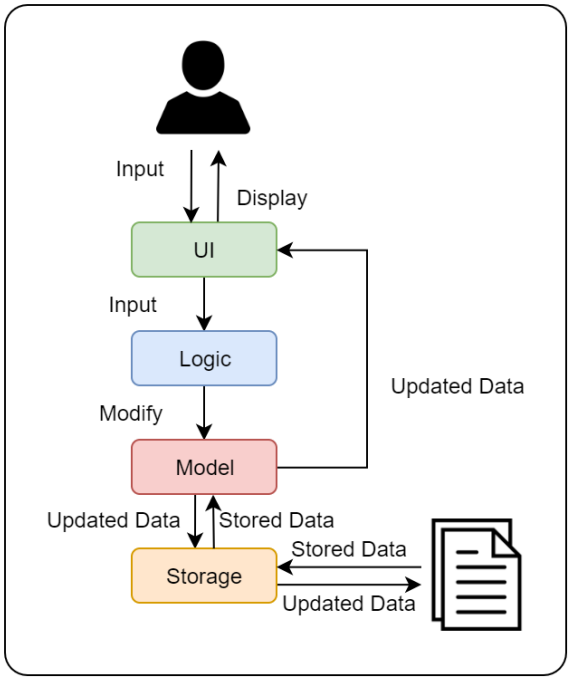


Figure 1. Overall Architecture

The overall architecture design is as shown in Figure 1. Upon the start of the program, `Storage` loads local files to update `Model`.

The user interact with the `UI`, which passes on the user input to `Logic` for parsing. Parsed command will be executed to modify `Model`. Once `Model` is modified, it updates the `UI` display and calls the `Storage` to update the local files.

4.2. Overall Activity Flow

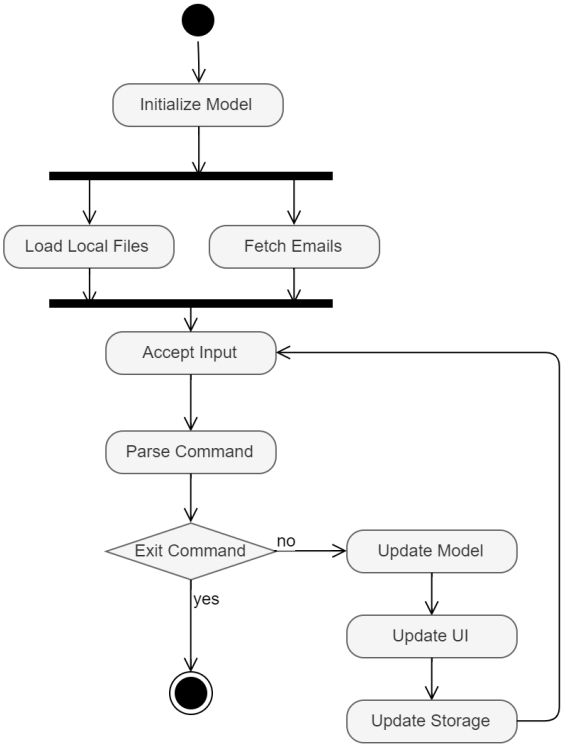


Figure 2. Activity Diagram

When the program first start, it initializes the model. It then loads the local data files and fetch emails from the Outlook Server at the same time. Once these two activities are finished, it starts to wait for user input.

If the input is received, it is parsed to command. Command will be executed depending on its type. If it is an exit command, however, it will shutdown the program gracefully. Otherwise it updates model, which in turn updates UI and local storage, after which the program will continue to wait for user input.

This process continues until an exit command is received.

4.3. Email Management

4.3.1. Email Auto Parsing

The emails fetched or stored locally will be automatically parsed to extract important information for tagging, task creation and reminder purposes. The parsing consists of two stages, the **format parsing** and **content parsing**. Email format parsing is to parse the email components like subject, sender and body from the raw string fetched from the server or stored in local file. The content parsing is to parse the keyword included all components of email.

Email Format Parsing

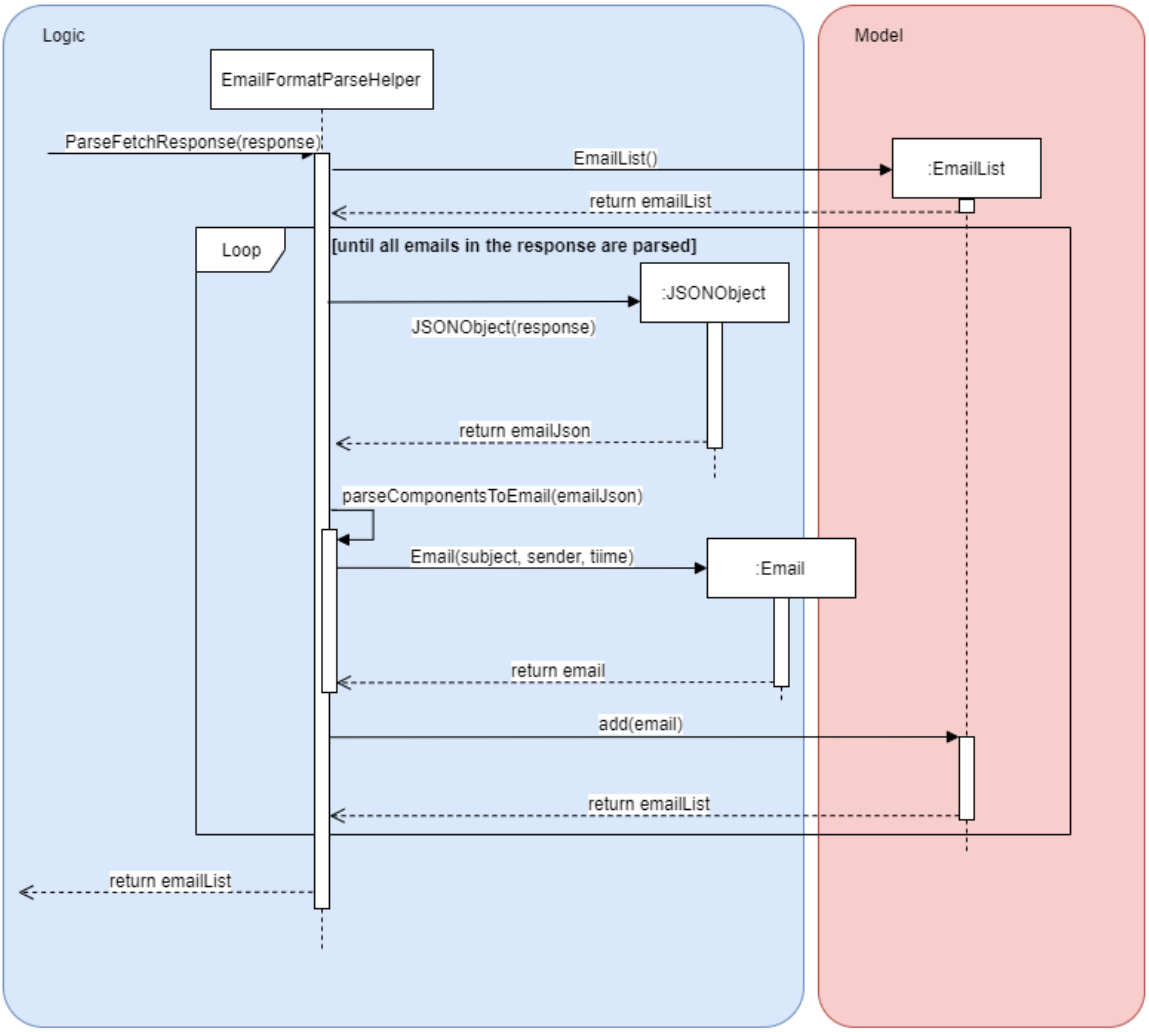


Figure 3. Email Format Parsing

The email format parsing starts at the call of `ParseFetchResponse(response)`. The variable `response` here is a string of the http response from the Outlook server to the fetch API call.

It first creates an `EmailList` to store all the emails parsed from the response. This function only returns the list instead of directly adding the email parsed to the model because the storage or network component will decide whether and how the emails are to added to the model.

Then each email contained in the response of is parsed to a `JSONObject` called `emailJson` for easier manipulation.

With this `emailJson`, `parseComponentsToEmail(emailJson)` is called to extract different components of the json and instantiate an `Email` object to be added to the `emailList` created earlier.

This process repeats until all the email information in the response is processed.