Yang Zi Yun - Project Portfolio for Duke Email Manager

Product Overview

My team and I developed **Duke Email Manager** as a solution for the overwhelming emails received daily in our mailbox.

Duke Email Manager is an email and task manager desktop 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 CLI. **Duke 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.

The figure below shows interface of our app:

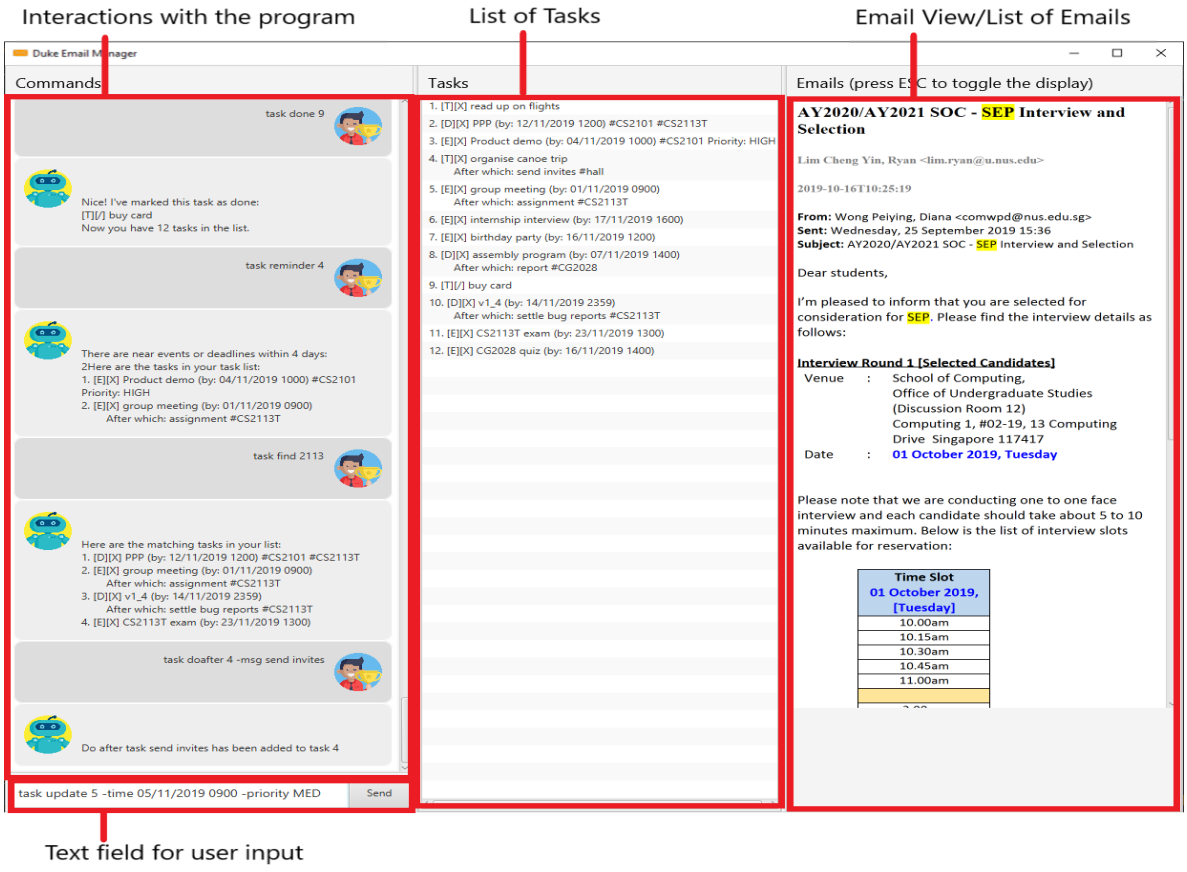

Figure 1. GUI interface of Email Manager

My role was mainly to design and write the codes for **email managing** including email showing, email tagging and email filtering. The following sections illustrate these enhancements in more detail, as well as the relevant sections I have added to the user and developer guides in relation to these enhancements.

Summary of contributions

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

Code contributed

Please click this link to examine the code contributed for this project: [Summary of code contribution]

Enhancement added: Email Filtering by Tag(s)

- **What it does**: Students can filter the emails by the tags added with the command `email list -tag <TagName1> -tag <TagName2>`
- **Justification**: This allows the students to view the list of emails being tagged with the given tag(s), helping them to filter out relevant emails to keep their email organised.
- **Highlights**: Maximum number of input tags allowed is two, this will show emails that are tagged with both tags.
- **Code contributed**: (#97) (#105) (#194)

Other contributions

- Project management:
  - Managed releases versions `v1.2.1` on GitHub (v1.2.1)
- Documentation:
  - Update User Guide and Developer Guide
- Community:
  - PRs reviewed (with non-trivial review comments)
  - Reported bugs and offered suggestions for other teams in the class
- Tools:
  - Integrated continuous integration (Travis) to the team repo (#36) (#37) (#38)
  - Integrated coverage report (Coveralls) to the team repo (#107) (#102)
- GUI enhancement:
  - Implemented a different background colour for UserDialogBox and DukeDialogBox for clearer layout and view purpose (#114)
  - Resize window to fit screen (#71)
- Other enhancement or feature:
  - Clearing all emails feature, deleting email feature, listing email tags feature (#194)
  - Listing email keywords feature (#186)
  - Email Tagging feature (#82)
  - Display of email using WebView and toggling email list view and email content view by pressing `ESC` key (#71)
  - Switching mode between email and task (#54)
  - Task detection of anomalies (#32)
  - Basic email class implementation (#48) (#49) (#50)
  - Wrote additional tests for existing features to increase instruction coverage from 19% to 29%, and increase branch coverage from 15% to 20% (#114)
  - Implemented logger (#181)
  - Added key binding functionality to create keyboard shortcut (#78) (#70)

Contributions to the User Guide

*We have updated the User Guide with instructions for the enhancements that we had added. Given below are sections I contributed to the **Email Manager User Guide**. They showcase my ability to write documentation targeting end-users.*

Change Mode: `flip`

Format: `flip`

Flips or toggles between email mode and task mode. The prefix of the command in the text box will also be changed.

NOTE	In task mode, the text box will display `task` as a prefix. In email mode, the text box will display `email` as a prefix.

Listing all emails: `list`

Format: `list`

Gives a complete list of emails.

Showing an email: `show`

Format: `show INDEX_NUMBER`

Show the email content of the email at the index number in the email list.

Example:

`show 3`: shows content of the 3rd email in the email list.

TIP	You can press `Esc` key on your keyboard any time to switch display between the list and content view of emails.

Listing all keywords: `listKeyword`

Format: `listKeyword`

Gives a list of all keywords with the relevant expressions.

Tagging an email: `update`

Format: `update ITEM_NUMBER -tag TAG1 [-tag TAG2] ...`

Tags the specified item with the tag(s) minimum number of tags is 1. Tags without duplication will be added.

Examples:

`update 1 -tag CS2113T`

`update 2 -tag Tutorial -tag Spam`

Listing all tags: `listTag`

Format: `listTag`

Gives a list of all existing tags in the list of emails.

Filtering email by tags: `list`

Format: `list [-tag TAG1] [-tag TAG2]`

Gives a list of emails with the tags given. Minimum number of tags is 1, and the maximum number of tags is 2.

NOTE	TAG1 **exists** if there is at least an email tagged with TAG1.

NOTE	TAG1 and TAG2 **co-exist** if there is at least an email tagged with both tags at the same time.

NOTE	Both TAG1 and TAG2 **exist but not co-exist** means that there is at least one email with TAG1 and another email with TAG2, but no email is tagged with both TAG1 and TAG2.

Explanation:

- User input: `list [-tag TAG1]`
  - Case 1: TAG1 exists, for each tag co-existing with TAG1, the program shows a list of emails tagged with both TAG1 and the co-existing tag.
  - Case 2: TAG1 does not exit, the program returns an error message.
- User input: `list [-tag TAG1] [-tag TAG2]`
  - Case 1: Both TAG1 and TAG2 do not exist, the program returns an error message.
  - Case 2: Either TAG1 or TAG2 exists, the program shows a list of emails with either TAG1 or TAG2.
  - Case 3: TAG1 and TAG2 exist but do not co-exist, the program shows two separate list of emails with TAG1 and TAG2 respectively.
  - Case 4: TAG1 and TAG2 co-exist, the program shows a list of emails with both TAG1 and TAG2.

Examples:

`list -tag Spam`: If `Spam` does not exist, an error will be returned. If `Spam` exists, for **each** tag co-existing with `Spam`, a list of emails tagged with the `co-existing tag` and `Spam` will be listed out.

`list -tag CS2113T -tag Tutorial`: If `CS2113T` and `Tutorial` co-exist, emails tagged with both `CS2113T` and `Tutorial` will be listed out. If no email is tagged with both tags (`CS2113T` and `Tutorial` do not co-exist), emails tagged with each of the tags will be listed out respectively.

TIP	After obtaining the list of emails with the tags, you can enter `show ITEM_NUMBER` to view the content of email, `ITEM_NUMBE` of an email is the index number of the email in the list.

Deleting a local email: `delete`

Format: `delete ITEM_NUMBER`

Deletes the email specified from local storage.

Examples:

`delete 1`: deletes the first email in the email list from local storage.

NOTE	If you enter `show ITEM_NUMBER`, then followed with `delete ITEM_NUMBER`, the content of email at `ITEM_NUMBER` will remain displayed although the email has been deleted.

TIP	This command will only delete email from local storage. If you enter `delete 1`, after that enter `fetch` command or relaunch the program, provided that the deleted email is present in your remote server, that particular email will be loaded into your local storage again even if you have deleted it before.

Clear local email list: `clear`

Format: `clear`

This command deletes all emails in the list from local storage.

When you have accumulated too many emails in the email list, auto-parsing function will take longer time to complete, if you do not need the older emails in your list, this function can `clear` your email list.

WARNING	Once executed, you will not be able to undo this command.

TIP	After clearing all the emails from local storage, you can enter `fetch` to retrieve latest 60 emails from server. Those cleared email will be loaded into your local storage again if it is present in your remote server.

Contributions to the Developer Guide

---

*Given below are sections I contributed to the **Email Manager Developer Guide** for the **email tagging and filtering by tags** features. They showcase my ability to write technical documentation and the technical depth of my contributions to the project.*

Email Filtering by Tag(s)

Email Manager allows user to filter emails by tag(s).

Current Implementation

- Format: `list [-tag TAG1] [-tag TAG2] …`
- Note: Gives a list of emails with the tags. Minimum number of tags is 1, and the maximum is 2.
- Eg: `email list -tag Fun -tag Project`

Following is the activity diagram when the command is executed:

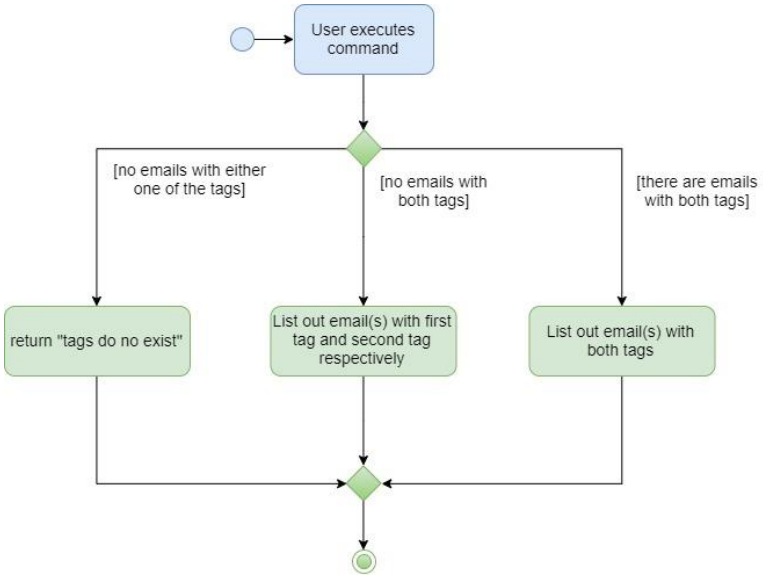

Figure 4. Activity diagram for email filtering by tags

The following sequence diagram below will explain how the `email update` command works in detail:

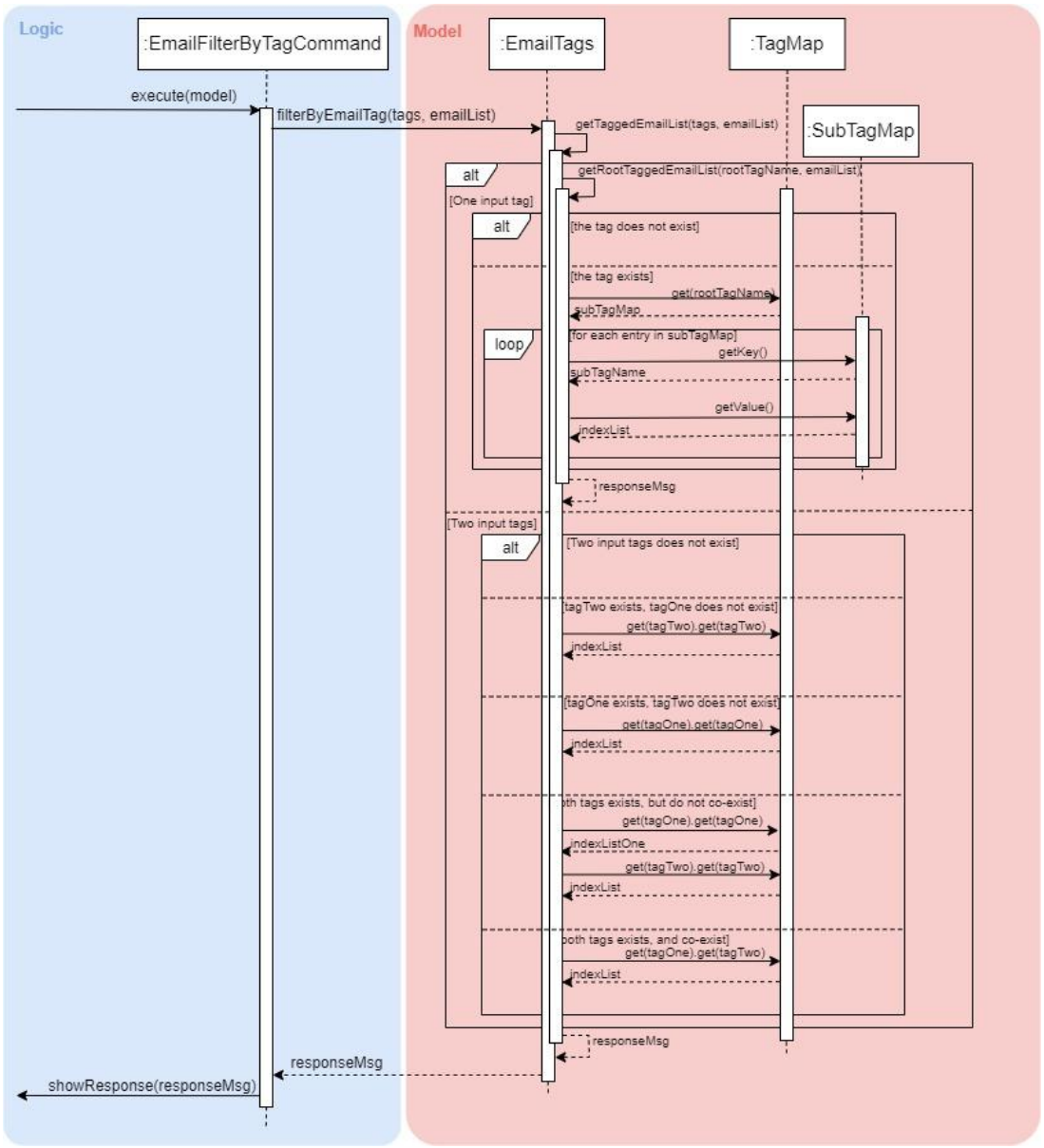

Figure5. Sequence diagram for email filtering by tags

An example usage of the command is as follows:

Step 1: The user launches the application. The user wishes to tag the 2nd email in the list with "Fun" and "Project" (Implementation of this part is explained in Developer Guide Section 5.3.2). After tagging the email, the user wishes to view the list of emails with these tags, hence the user inputs `email list -tag Fun -tag Project`.

Step 2: `UI` component captures the input and passes to `Logic` component to parse the input. Section below explains how `Logic` component parse the input.

- `CommandParseHelper` takes in the `input`, parses and extracts tags information and stores it inside ArrayList<Option> `optionList`, then passes the `input` and `optionList` to `EmailCommandParseHelper`.
- `input` here `email list`
- `optionList` here is [tag=Fun, tag=Project]
- `EmailCommandParseHelper` parses the `input` and extract tags information `optionList` and stores it in ArrayList<String> `tags`.
    - `tags` here is [Fun, Project]
- `EmailCommandParseHelper` creates a new `EmailTagListCommand` by passing in `tags`, then return the `EmailTagListCommand` to `CommandParseHelper` and then to `UI`

Step 3: `EmailFilterByTagCommand#execute(model)` is called by `UI`.

Step 4: `EmailFilterByTagCommand` calls `EmailTags#filterByEmailTag(tags, emailList)`, which calls `EmailTags#getTaggedEmailList(tags, emailList)`.

Step 5: `getTaggedEmailList()` checks the conditions of the each tags in `tags`, we say that a tag exists if there is email with the tag. If none of the emails has the tag, we say that the tag does not exist. We say that both tags co-exist if there is email tagged with both tags.

- In this example, both tags “Fun” and “Project” co-exist.

Step 6: `getTaggedEmailList()` calls `TagMap.get("Fun").get("Project")`. `TagMap` returns `indexList` which is the **index** of all email(s) tagged with both “Fun” and “Project”.

Step 7: `getTaggedEmailList()` constructs a String `responseMsg` containing the list of title of emails from the `indexList`. After that, `getTaggedEmailList()` returns the `responseMsg` to `filterByEmailTag`, then to `EmailFilterByTagCommand` and to the ` `UI`.

- `responseMsg` here is: "Here is the email tagged with both #Project and #Fun: <list of title of email(s)with both tags>"

Design Considerations

Alternative 1 (current choice):

The tags associated with emails is stored in `TagMap`. `TagMap` is updated upon every user input which will invokes the `EmailTags#updateTagMap`.

- `TagMap` is a `HashMap<String, SubTagMap>`:
    - Each `key` in the HashMap is a tag name (we call it `root tag name` here) that exists in the email list.
    - The `value` associated with each `key` is a `SubTagMap`

- `SubTagMap` is a `HashMap<String, IndexList>`::
    - Each `key` in the HashMap is a tag name (we call it `sub tag name` here) that co-exists with the `root tag name` from the `TagMap`. We say that both tags co-exist if there is email tagged with both tags.
    - The `value` associated with each `key` is an `IndexList`, which is an ArrayList<Integer> that stores index of emails tagged with both `root tag name` and `sub tag name`.
- For example, let `emailOne` be an email tagged with `Tutorial` and `CS2113T`, `emailTwo` be an email tagged with `Tutorial` and `CG2271`.
    - `emailOne` has index 1 and `emailTwo` has index 2 in the email list.
    - After calling `EmailTags#updateTagMap`, the `TagMap` has the following structure:
      {

      Tutorial={Tutorial=[1, 2], CS2113T=[1], CG2271=[2]}, +

      CS2113T={CS2113T=[1], Tutorial=[1]}, +

      CG2271={CG2271=[2], Tutorial= [2]} +

      }

- **Pros**: Faster search when user invokes `EmailFilterByTagCommand`, since `EmailTags#filterByEmailTag` is navigating in the HashMap.
- **Cons**: Current implementation invokes the `EmailTags#updateTagMap` on every user input to keep the `TagMap` and email list view in GUI updated, which increases the computational load.

Alternative 2:

Loop through each tag of each email in the list of emails, and check if each tag equals to the tag requested by the user, if yes, add the email to the list, if no, continue with the loop. After finishing the loop, output the email(s) in the list.

- **Pros**: This implementation does not have to maintain a `TagMap` structure to keep track of the emails with the tags, therefore does not requires update of the `TagMap`, this saves the space and computational load of the program.
- **Cons**: Slower search when user invokes `EmailFilterByTagCommand` , since it has to loop through each tag of each email in the list of emails.