

Lee Wai Fong - Project Portfolio

PROJECT: Billboard

Introduction

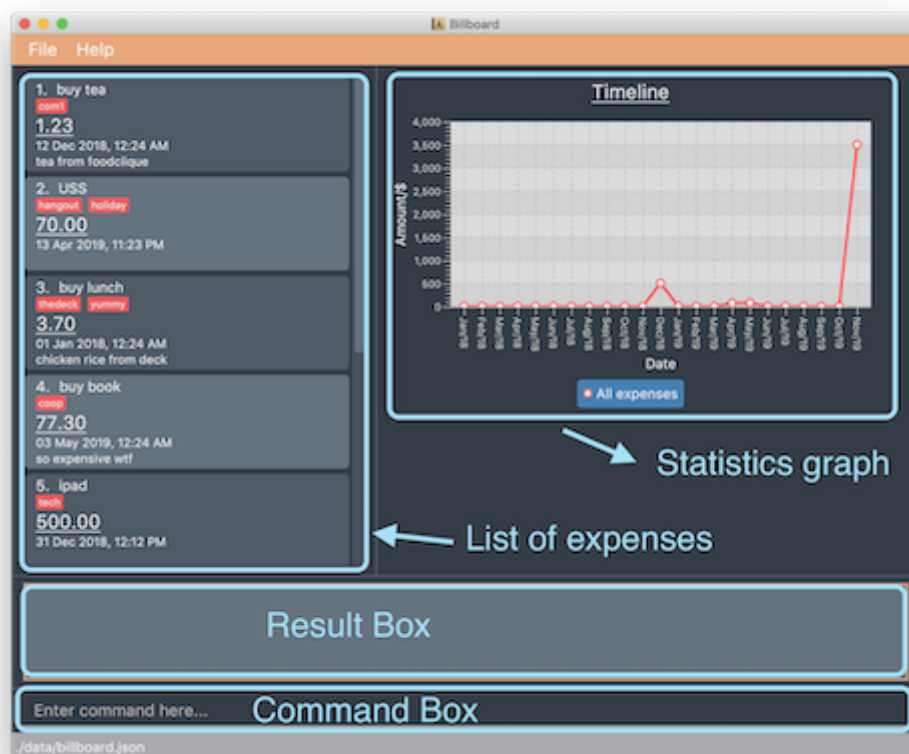
This page will document my overall contributions to the Software Engineering Project, *Billboard*.

About Billboard

For our Software Engineering Project, my team of 5 were tasked with enhancing a desktop addressbook application. With students as our target users, we decided to morph it into a personal financial management system called *Billboard*. *Billboard* enables students to track and manage their expenses in an intuitive and convenient way. With features like tagging and archiving, students are able to manage and organise their expenses. Similarly, features like viewing statistical graphs of their recent expenditure and advanced search functions, they are able to track their expenses.

Billboard is a Command Line Interface(CLI) desktop application, meaning that it is catered for students who prefer typing as all interactions between users and the application is through typing. For more user friendliness, *Billboard* has a Graphical User Interface(GUI).

This is what *Billboard* looks like:



My role was to design and implement features related to tagging. The following sections illustrate these enhancements in more detail, as well as the relevant documentation i have added to the user and developer guides in relation to them.

Note the following symbols and formatting used in this document:

NOTE	This symbol indicates important information
-------------	---------------------------------------------

tag - A grey highlight (called a mark-up) indicates that this is a command that can be inputted into the command line and executed by the application.

"tech" - Italicised text with quotation marks indicates the name of an expense or tag.

Summary of contributions

This section shows a summary of my coding, documentation, and other helpful contributions to *Billboard*.

- **Major enhancement:** i added features related to tagging.
 - **What it does:** This enhancement allows users to add and remove tags from an expense. Users can also list out existing tags and filter their expenses by tags.
 - **Justifications:** Tagging allows for easier management of expenses. Expenses with the same tags are grouped together which allows users to search for related expenses quickly.
 - **Highlights:** This enhancement works with existing as well as future commands. An in-depth analysis of design alternatives was necessary since there are many ways to implement this enhancement, each with its own pros and cons in terms of efficiency or space allocation in the application. The current design is used to ensure there is a balance between space allocation and efficiency of the application.
- **Code Contributed:** [\[Functional code\]](#) [\[Test code\]](#)
- **Other Contributions:**
 - **Enhancements to existing features:**
 - Updated the GUI color scheme and effects to make it more user-friendly. (PR [#105](#))
 - Wrote additional tests for existing features (PR [#39](#))
 - **Documentation:**
 - Contributed to user stories, use cases and non functional requirements in the Developer Guide.
 - **Community:**
 - PRs reviewed and approved: [#6](#) [#18](#) [#51](#) [#72](#) [#94](#) [#103](#) [#104](#) [#108](#) [#115](#)

Contributions to the User Guide

We had to update the original addressbook User Guide with instructions for the enhancements that

we had added. The following is an excerpt from *Billboard User Guide*, showing additions that I have made for the tag features.

Tagging expenses: tag

This feature allows you to assign tags to your expenses. Expenses with the same tag will be grouped together, allowing you to search for related expenses easily.

NOTE

Tag names should be **alphanumeric**. This means that they should not contain spaces or special characters.

Tag names are **case sensitive**.

You are **not** able to use this feature on **archived expenses**.

To view a list of supported tag commands, you could type **tag** in the command box and press **enter**. Similarly, to view parameters for supported tag commands, you could type **tag** followed by the supported tag command and press **enter**.

Example: **tag rm**

Adding a tag: tag add

This command adds your input tag(s) to the expense at the index you have specified.

Usage:

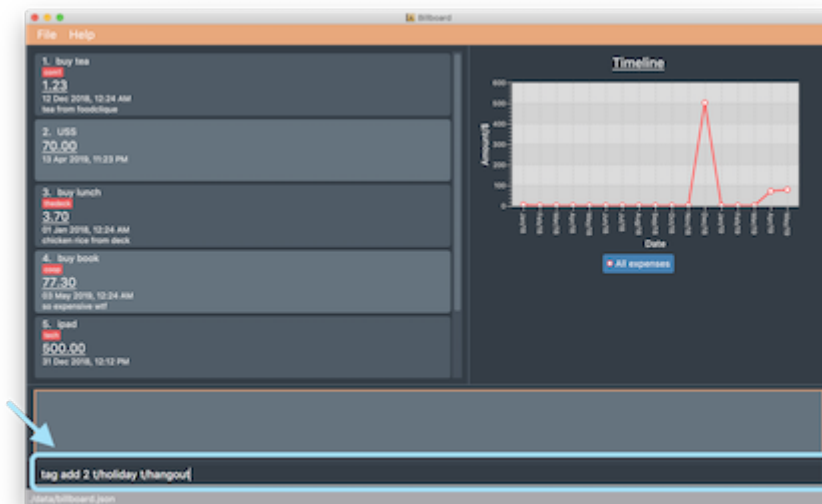
```
tag add [INDEX] t/[TAG] t/[TAG...]
```

Example:

Lets say that you want to add the tags "holiday" and "hangout" to the expense at index 2 which is "USS".

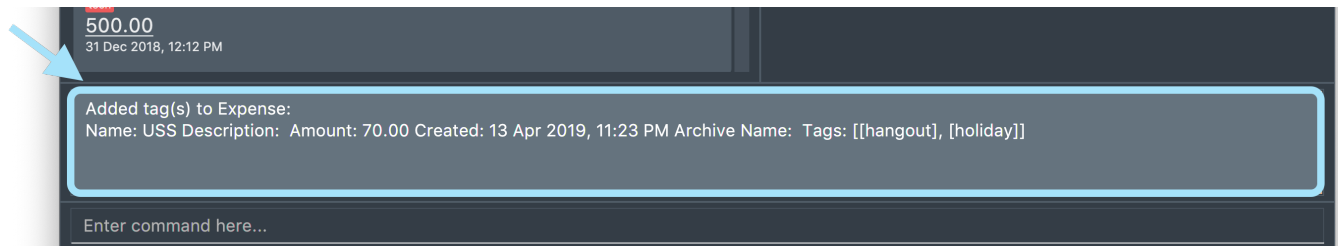
To add the tags:

1. Type **tag add 2 t/holiday t/hangout** in the command box and press **enter** to execute it.

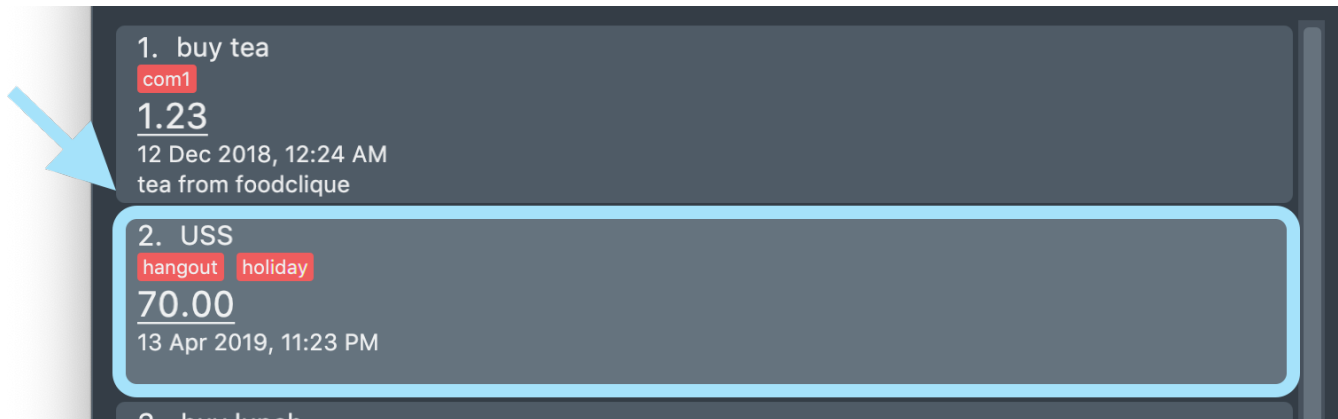


2. The result box will display the message "Added tag(s) to Expense:

Name: USS Description: Amount: 70.00 Created: 13 Apr 2019, 11:23 PM Archive Name: Tags: [[hangout], [holiday]]"



3. You could see the tags "*holiday*" and "*hangout*" in the expense at index 2.



NOTE

You are not able to add **existing** tags.

If you input **existing** and **non-existing** tags, Billboard adds the **non-existing** tags only.

If you input **duplicate** tags, Billboard adds them **once**. i.e Billboard does **not** allow Duplicate tags in an expense.

Example: `tag add t/holiday t/holiday`

Removing a tag: `tag rm`

This command removes your input tag(s) from the expense at the index you have specified.

Usage:

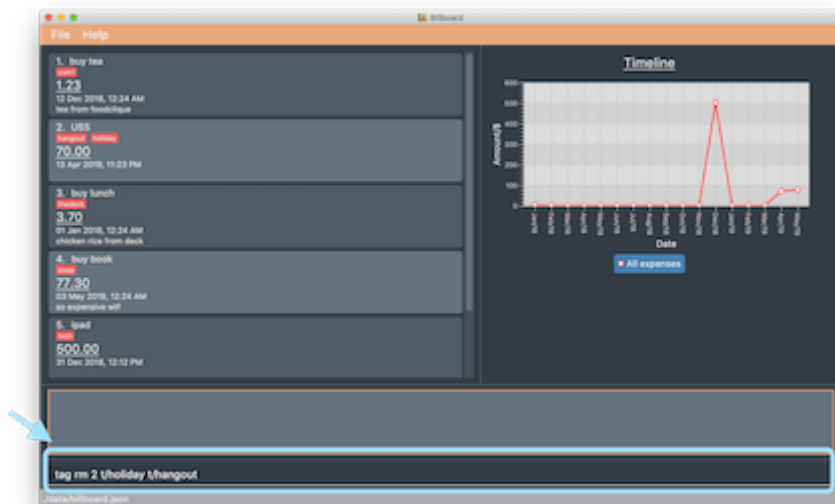
```
tag rm [INDEX] t/[TAG] t/[TAG...]
```

Example:

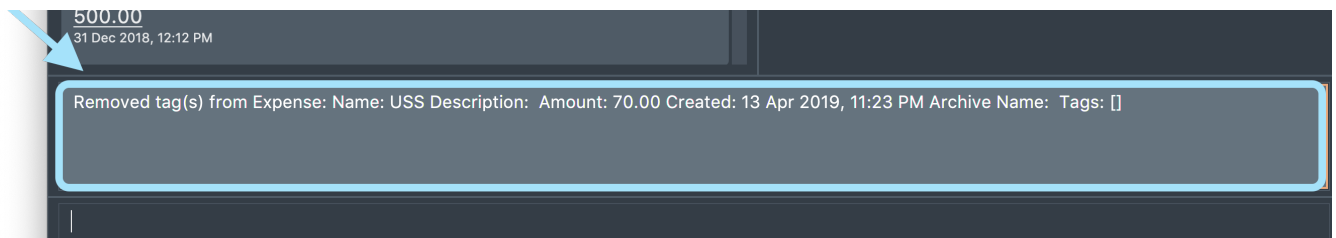
Lets say that you want to remove the tags "*holiday*" and "*hangout*" from the expense at index 2 which is "*USS*".

To remove the tags:

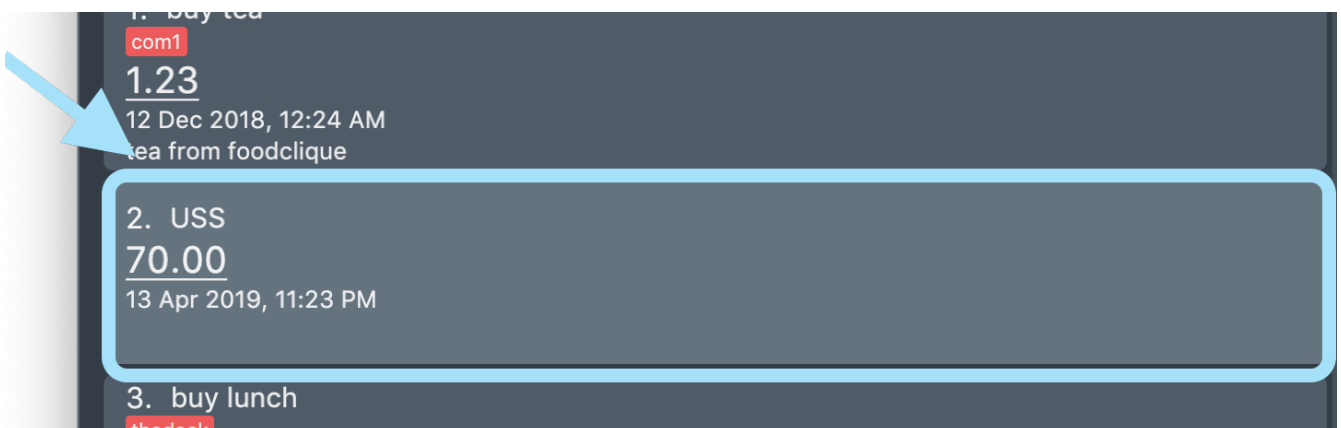
1. Type `tag rm 2 t/holiday t/hangout` in the command box and press **enter** to execute it.



2. The result box will display the message "Removed tag(s) from Expense:
Name: USS Description: Amount: 70.00 Created: 13 Apr 2019, 11:23 PM Archive Name: Tags: []"



3. You could see that the expense at index 2 no longer has the tags "hangout" and "holiday".



NOTE

You are not able to remove **non-existing** tags.
If you input **duplicate** tags, Billboard removes them **once**.
Example: `tag rm t/holiday t/holiday`

Filtering by tag: tag filter

This command filters expenses by your input tag(s).

Usage:

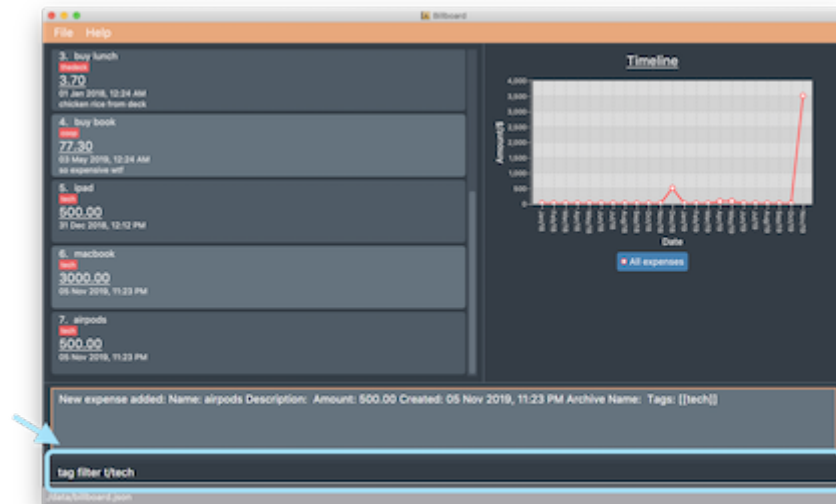
```
tag filter t/[TAG] t/[TAG...]
```

Example:

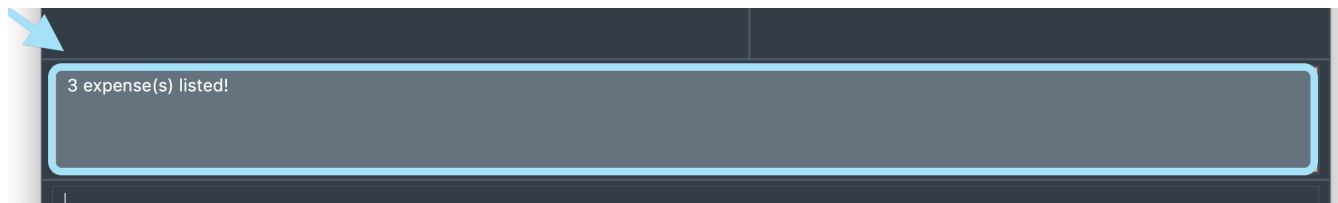
Lets say that you want to filter your expenses by the tag "*tech*".

To filter your expenses:

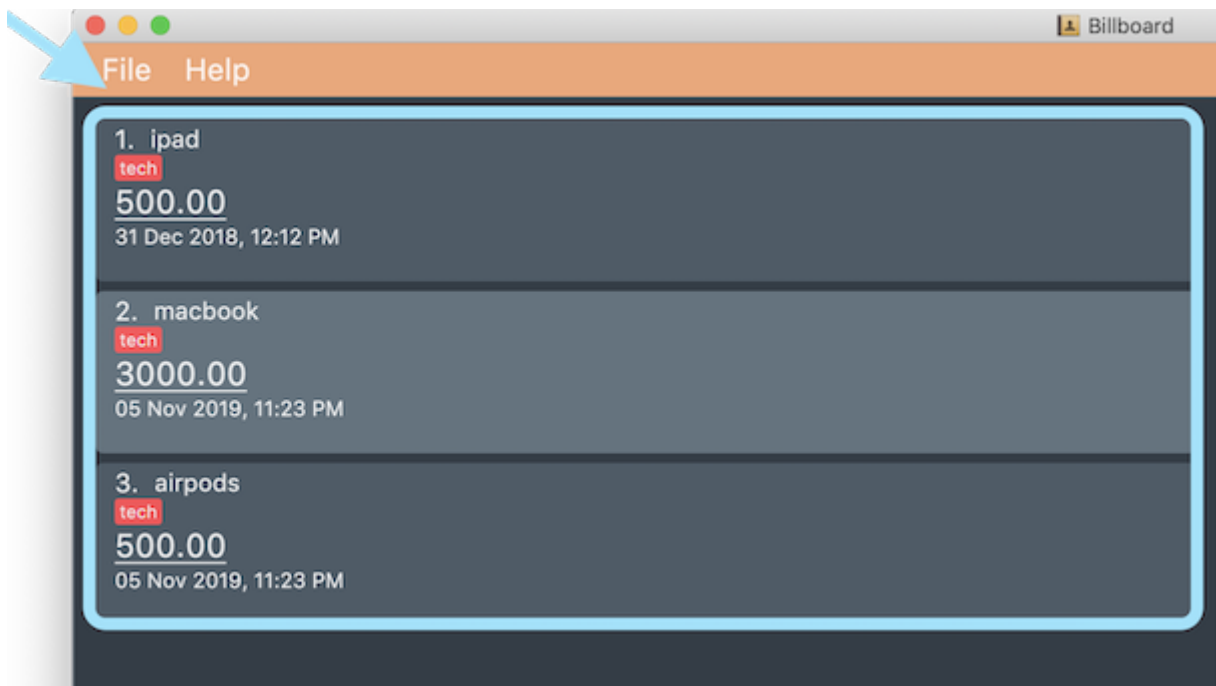
1. Type `tag filter t/tech` in the command box and press `enter` to execute it.



2. The result box will display the message "3 expense(s) listed!"



3. Billboard lists all expenses under the tag "*tech*".



NOTE

If you input **more than one** tags, Billboard lists out all expenses tagged with **one or more** input tags.

This command allows **duplicate** tags as input.

This command allows **non-existing** tags as input.

After executing this command, you could edit (eg. **edit**, **tag add** etc) the filtered expenses using the displayed indexes.

Listing out all the tags: **tag list**

This command lists out all existing tags.

Usage:

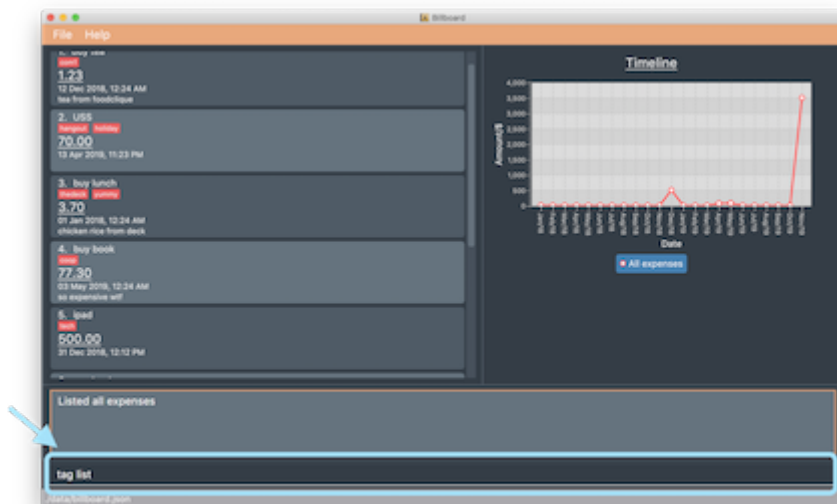
```
tag list
```

Example:

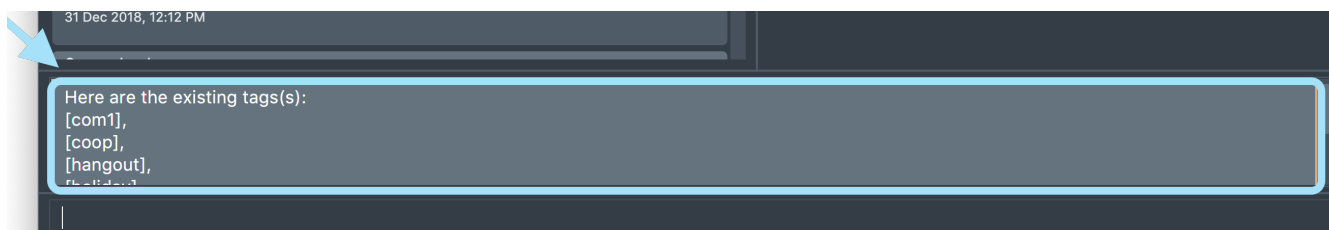
Lets say that you want to list out all existing tags.

To list them:

1. Type **tag list** in the command box and press **enter** to execute it.



2. The result box displays a list of all existing tags.



Contributions to Developer Guide

We had to update the original addressbook Developer Guide with instructions for the enhancements that we had added. The following is an excerpt from *Billboard User Guide*, showing additions that I have made for the tag features.

Tagging

Proposed Implementation

The tag feature supports the following operations:

- Adding tags to an expense
- Removing tags from an expense
- Filtering expenses by tags
- Listing all existing tags

These actions are facilitated by the `UniqueTagList` and `TagCountManager` classes:

- `UniqueTagList` maps `String` to `Tag` where `String` is the name of the `Tag`. It ensures that the same `Tag` is referenced instead of creating many `Tag` objects of the same name when adding tags.
- `TagCountManager` maps `Tag` to `Integer`, where `Integer` is the number of `Expense` under each `Tag`. It

allows **Tag** objects that are not tagged with any **Expense** to be tracked and removed.

Operations include:

- **UniqueTagList#retrieveTags(List<String>)** — Retrieves corresponding tags from **UniqueTagList** based on the list of tag names.
- **UniqueTagList#removeAll(List<Tag>)** — Removes tags given in the list from the **UniqueTagList**.
- **UniqueTagList#getTagNames()** — Returns a list of tag names.
- **TagCountManager#incrementAllCount(Set<Tag>)** — Increment the number of expenses mapped to tags given in the set by 1.
- **TagCountManager#decreaseAllCount(Set<Tag>)** — Decrease number of expenses mapped to tags given in the set by 1.
- **TagCountManager#removeZeroCount()** — Removes all mappings where number of expense is equal to 0.

These operations are exposed in the **Model** interface as:

- **Model#retrieveTags(List<String>)**
- **Model#incrementCount(Set<Tag>)**
- **Model#decreaseCount(Set<Tag>)** — **TagCountManager#removeZeroCount()** and **UniqueTagList#removeAll(List<Tag>)** is called in this method as well to remove tags not tagged to any expense from **UniqueTagList** and **TagCountManager**.
- **Model#getTagNames()**

Given below is an example usage scenario and how the adding tag mechanism behaves at each step.

- **Step 1:** The user launches the application. The **Model** is initialized with saved data. All tags are loaded into **UniqueTagList** and **TagCountManager**.
- **Step 2:** User enters the command **tag add 1 t/test t/test2** to add tags to the 1st **Expense** in **Billboard**. **BillboardParser** parses the command, creating a **TagCommandParser**. The **TagCommandParser** then parses **add 1 t/test t/test2** and creates an **AddTagCommandParser**. Subsequently, the **AddTagCommandParser** parses **1 t/test1 t/test2** into **Index 1** and a list of **String** consisting of **test1** and **test2**. They are then used to create **AddTagCommand** which is returned to **LogicManager**.
- **Step 3:** **LogicManager** executes the **AddTagCommand**. During execution, **AddTagCommand** calls **Model#retrieveTags(Set<String>)** on the list consisting of **test1** and **test2** which returns a set of **Tag**. **Model#incrementCount(Set<Tag>)** is then called on the set of **Tag**. The 1st **Expense** is edited and updated in the **Model** using **Model#setExpense(Expense, Expense)** and is then shown on the GUI.

NOTE

Duplicate tags in an **Expense** is not allowed. If the user tries to add an existing **Tag** to an **Expense**, it will not be added, and **Model#incrementCount** will not be called. If the user tries to add duplicate **Tag**, i.e enter 2 of the same **Tag**, it will only be added once and number of expense under that **Tag** will only increment by 1.

The following sequence diagram shows how the adding tag operation works.

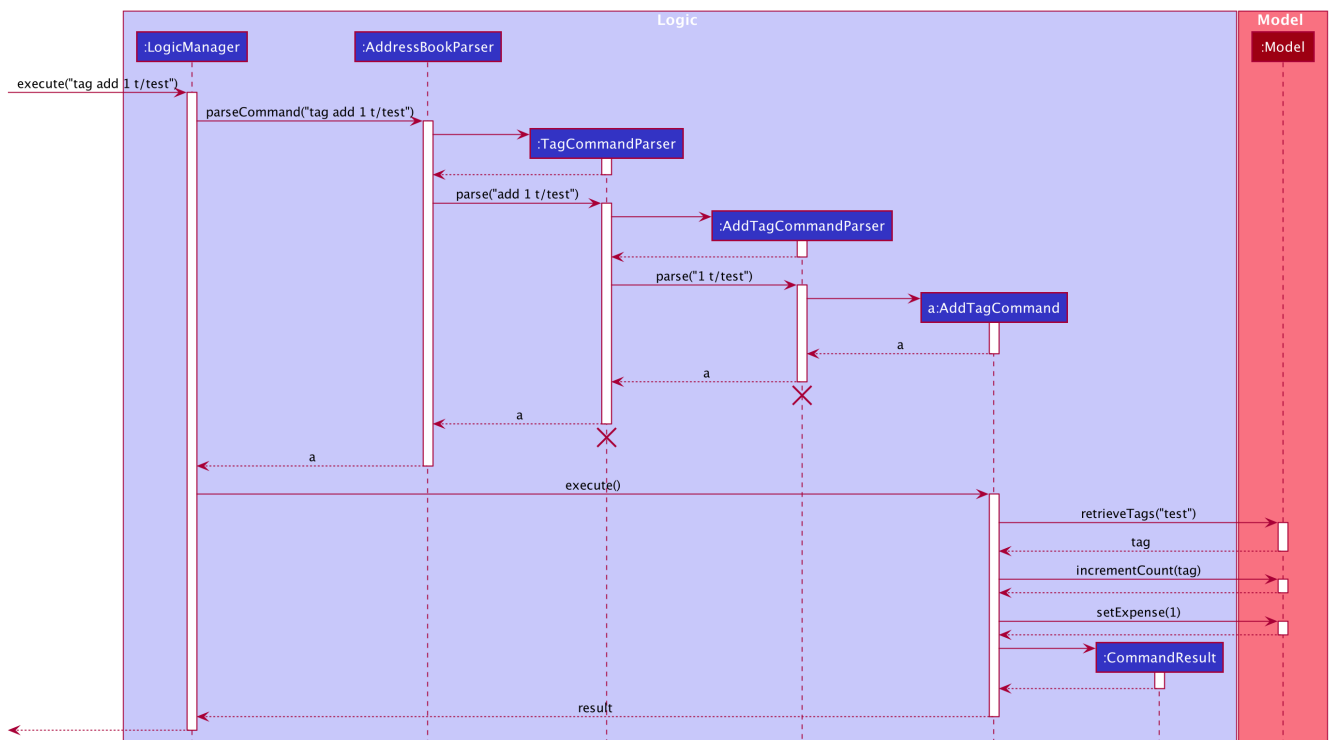


Figure 21. Sequence diagram of executing AddTagCommand.

The following activity diagram summarizes what happens when a user enters a command to add tags to an expense.

NOTE

The lifeline for **TagCommandParser** and **AddTagCommandParser** should end at the destroy marker (X) but due to a limitation of PlantUML, the lifeline reaches the end of diagram.

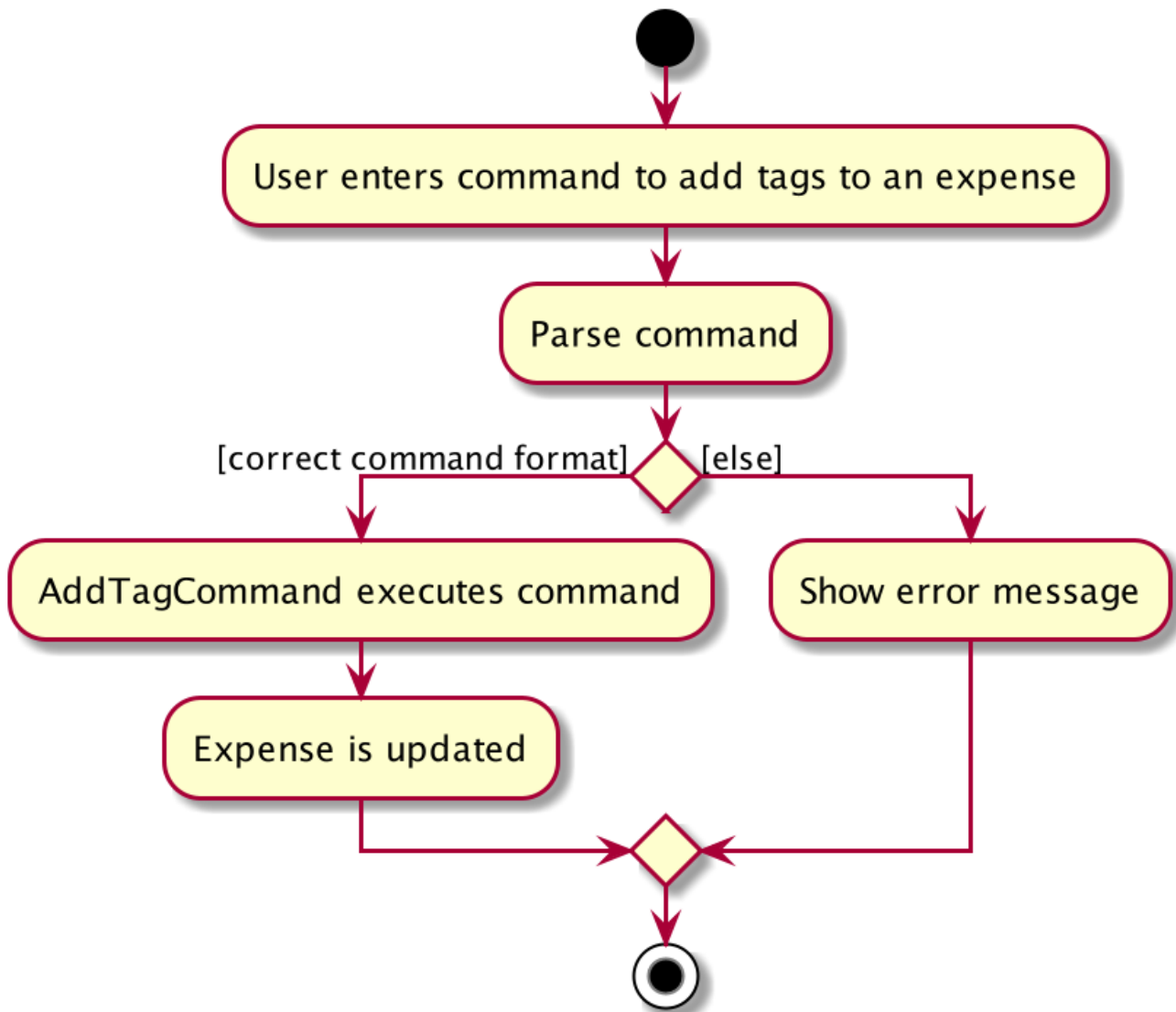


Figure 22. Activity diagram of executing AddTagCommand.

Design Considerations

Aspect: Data structure to support tag commands

- **Alternative 1 (current choice):** Use a `UniqueTagList` to map tag names to `Tag` and `TagCountManager` to map `Tag` to number of `Expense` under it.
 - Pros:
 - Each structure has only one responsibility.
 - Fast retrieval and update of data
 - Cons:
 - Requires maintenance of both structures as they need to sync with each other.
 - Retrieval of `Expense` under a `Tag` requires filtering through the whole list of `Expense`.
- **Alternative 2 :** Have each `Tag` store a list of `Expense` tagged to it.
 - Pros:

- Fast retrieval of **Expenses** under each **Tag**
- Cons:
 - Circular dependency
 - Since implementation of Billboard objects are immutable, there is a constant need to update the **Expense** by replacing it with the updated **Expense** even after executing non-tag related commands.
- **Alternative 3** : Use one map to map **Tag** to **Expense** tagged to it.
 - Pros:
 - Fast retrieval of **Expenses** under each **Tag**
 - Cons:
 - Since implementation of Billboard objects are immutable, there is a constant need to update the **Expense** by replacing it with the updated **Expense** even after executing non-tag related commands.