Prem Rajeshkumar Bagdawala - Project Portfolio

[Github] | [LinkedIn]

PROJECT: TravelBuddy

This project portfolio serves to document my contribution to **TravelBuddy**, a desktop application. It was developed as a team project for the National University of Singapore (NUS) module CS2103T Software Engineering. My team consisted of 4 NUS Computer Science undergraduates, including myself.

The application was developed over a span of 8 weeks in Academic Year 18/19 Semester 2 using an iterative approach. During that time, we were asked to morph an existing desktop application AddressBook Level 4, which is made up of 10KLoC in Java. One main user requirement was the preference of the user to interact with the application using Command Line Interface (CLI) rather than Graphical User Interface (GUI). The final product that my team developed is TravelBuddy, which is detailed below.

---

1. Overview

TravelBuddy is a travel journal desktop application for travel enthusiasts to record places which they have previously visited. With the recorded data, travelers can search for places visited based on specified filters such as rating and tags. Additionally, the application can also analyse travel history and generate statistics. Users can also store their travel photos on TravelBuddy, which can help them remember and reminisce the places they have visited before.

My role in the project was to design and write code for the Create, Read, Update and Delete (CRUD) feature. The following sections showcase, in greater detail, my contributions and enhancements to TravelBuddy. They also include documentation of these enhancements in the user and developer guide.

2. Summary of Contributions

2.1. Major Enhancement

The major enhancement I added was, **the ability to add, list, edit and delete places in TravelBuddy**.

What it does: It allows users to add, list, edit and delete places in TravelBuddy.

Justification:

- The user needs to be able to `add` places to TravelBuddy.
- When the user applies a filter such as filter by a country, a limited number of places would be shown. The `list` command will allow the user to see all the places again.
- The user may make some mistakes when adding a place. The `edit` command will allow the user to rectify those mistakes.
- The user may accidentally add a place. The `delete` command allows the user to remove the specific place.

Highlights: This enhancement involved a large amount of refactoring and careful crafting of test cases in order to test thoroughly the `DateVisited` field.

Overall code contributed: [Project Code Dashboard]

2.2. Minor Enhancement

The minor enhancement I did was to convert the `Person` object from the original code base to a `Place` object.

Code contributed to minor enhancement: [Functional Code]

2.3. Other contributions

The Table 2.3.1 below documents the various other contributions that I made to TravelBuddy.

Table 2.3.1: Details of Other Contributions

Project management	• As Team Lead, I spearheaded the project planning and managed the project scope. Additionally, I conducted weekly meetings and managed the scheduling of tasks so as to prevent merge conflicts. • Managed a release on GitHub (Release: `v1.3`)
Enhancements to existing features	Added colors to tag labels so that they can be differentiated easily: #8

Documentation	• Updated the User Guide with the features I added: #70, #72, #106 • Updated the User Guide with list of country codes: #101 • Updated the Developer Guide with features I added and diagrams for illustration purposes: #55, #72, #86, #99, #104, #185
Community	• Fixed UI related bug: #125 • Fixed general bugs found during testing: #167 • Tested other projects, reported bugs and suggested improvements: #159, #161, #166, #171, #172, #173, #174, #176

3. Contributions to the User Guide

The original User Guide was updated to match the enhancements implemented in TravelBuddy.

Given below is the start of an excerpt from the User Guide which I had contributed. They demonstrate my ability to write easy-to-follow documentation that targets end-users.

3.1. Adding a Place: `add`

Description: The `add` command adds a place to TravelBuddy.

Shortcut: `a`

Format: `add [n/NAME] [cc/COUNTRY_CODE] [dv/DATE_VISITED] [r/RATING] [d/DESCRIPTION] [a/ADDRESS] [p/FILE_PATH] [t/TAG]…`

The Table 4.3.1 below shows the parameters that require a specific input format to be added.

Table 4.3.1: Parameters that Require a Specific Format

Parameter	Parameter Prefix	Specific Input Format
`RATING`	`r/`	An integer ranging from `1` to `5`
`COUNTRY_CODE`	`cc/`	A valid ISO-3166 three-letter country code e.g. `JPN` to represent `Japan` The full list of country codes can be found here
`DATE_VISITED`	`dv/`	A valid date that follows the `DD/MM/YYYY` format ranging from `01/01/1900` to the current date

Examples: Given below are some examples on how to utilize the `add` command:

- `add n/Botanic Gardens cc/SGP dv/01/01/2017 r/4 d/UNESCO World Heritage Site a/1 Cluny Rd, Singapore 259569 t/nature`
  Adds Botanic Gardens to the list of places you have visited into TravelBuddy.
- `add n/Raffles Hotel cc/SGP dv/05/05/2016 t/hotel d/This place is lovely a/Raffles Road r/5 t/staycation`

Adds Raffles Hotel to the list of places you have visited into TravelBuddy.

Figure 4.3.1 below shows the outcome of a specific `add` command

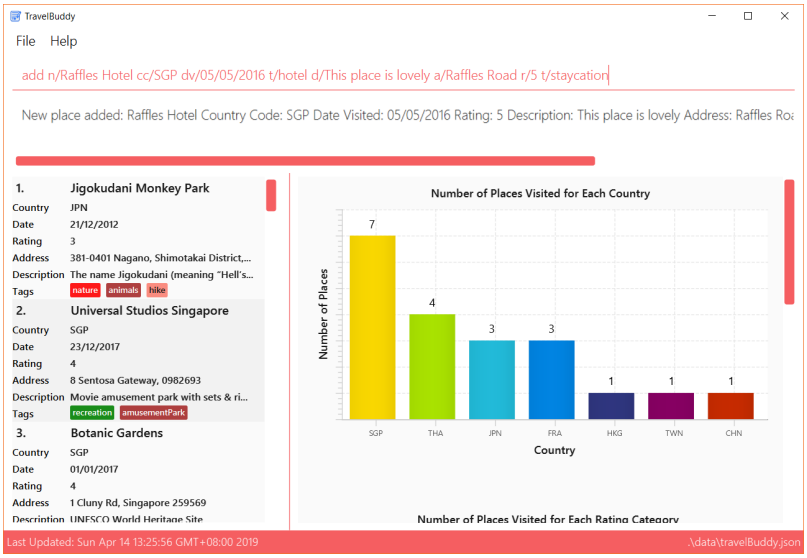


Figure 4.3.1: Adding a place to TravelBuddy

> **TIP** A place can have any number of tags (including 0 tags).

3.2. Deleting a Place: `delete`

Description: The `delete` command deletes the specified place from TravelBuddy.

Shortcut: `d`

Format: `delete INDEX`

Preconditions: Given below is a list of preconditions that must be met for the `delete` command to work:

- Deletes the place at the specified `INDEX`.
- The index refers to the index number shown in the currently displayed list, on the left.
- The index **must be a positive integer** 1, 2, 3, ...

Figure 4.11.1 below shows TravelBuddy before `delete` command is used.

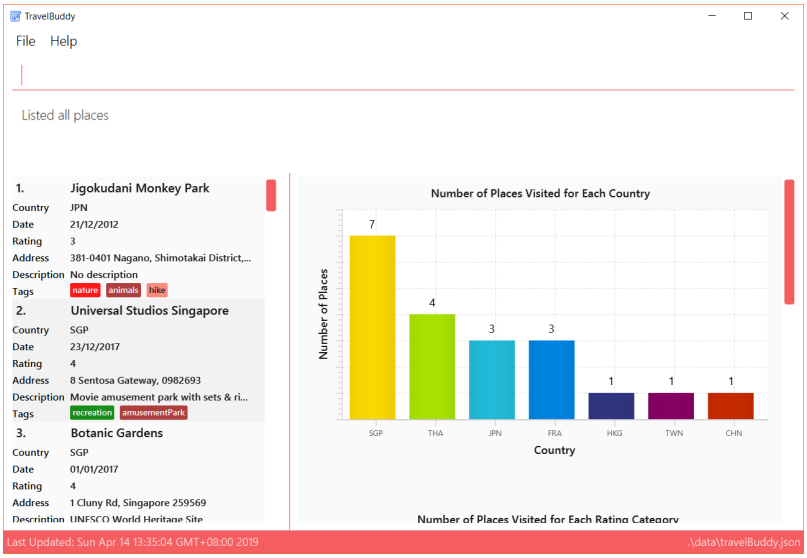


Figure 4.11.1: Before the `delete` command is used

Figure 4.11.2 below shows the result of using the `delete` command on the first index of the list.

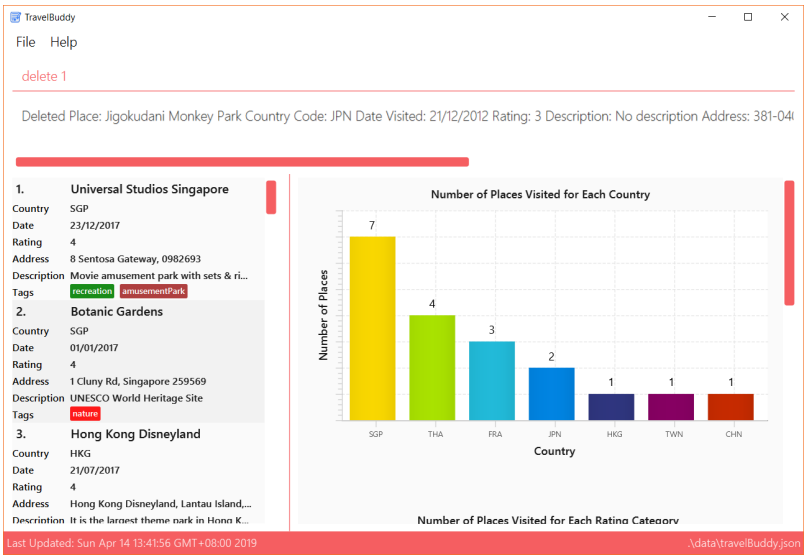


Figure 4.11.2: After the `delete` command is used

Examples: Given below are some examples on how to utilize the `delete` command:

- `list`
  Lists all the places in TravelBuddy
  `delete 2`
  Deletes the 2nd place in TravelBuddy.
- `search Raffles`
  Searches for any places which has the word "Raffles" in it.
  `delete 1`
  Deletes the 1st place in the results of the `search` command.

3.3. Listing All Places: `list`

Description: The `list` command displays a list of all the places in TravelBuddy.

Shortcut: `l`

Format: `list`

NOTE	Calling the `list` command returns a list of all the places in TravelBuddy as shown in [listFigure] below.

3.4. Editing a Place: `edit`

Description: The `edit` command edits an existing place in TravelBuddy.

Shortcut: `e`

Format: `edit INDEX [n/NAME] [cc/COUNTRY_CODE] [dv/DATE_VISITED] [r/RATING] [d/DESCRIPTION] [a/ADDRESS] [p/FILE_PATH] [t/TAG]…`

Preconditions: Given below is a list of preconditions that must be met for the `edit` command to work:

- The command edits the place at the specified `INDEX`. The index refers to the index number shown in the displayed place list. The index **must be a positive integer** 1, 2, 3, …
- It must have at least one of the optional fields.
- Its existing values will be updated to the input values.
- The adding of tags is not cumulative. Hence, when the tags are edited, the existing tags of the place will be removed.
- The tags can all be removed by typing `t/` without specifying any tags after it.
- Some parameters have a specific input format.
- Preconditions for changing the photo file [FILE_PATH] are in [Replacing the photo of a place with `edit p/`]

Examples: Given below are some examples on how to utilize the `edit` command:

- `edit 1 r/3 d/No description`
  Edits the rating and description of the 1st entry in the list to be `3` and `No description` respectively.
- `edit 2 n/Raffles Hotel t/`
  Edits the name of the 2nd entry in the list to be `Raffles Hotel` and clears all existing tags.

Figure 4.5.1 below shows the list of places before the `edit` command is used.

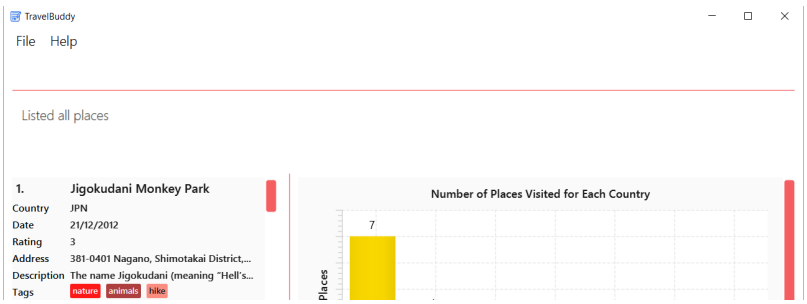


Figure 4.5.1: Before the `edit` command is used

Figure 4.5.2 below shows the list of places after the `edit` command is used.

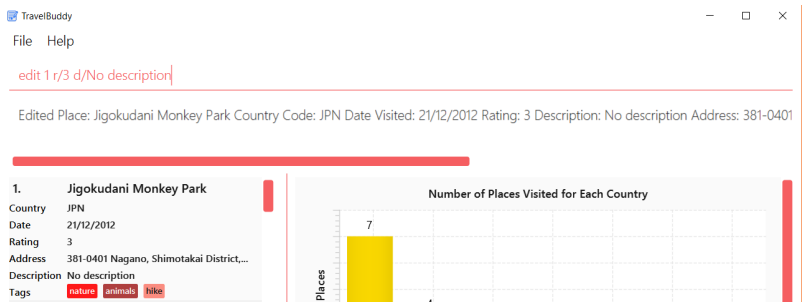


Figure 4.5.2: After the `edit` command is used

This marks the end of the excerpt from the User Guide.

4. Contributions to the Developer Guide

The original Developer Guide was updated to match the logic of the enhancements implemented in TravelBuddy.

Given below is the start of an excerpt from the Developer Guide which I had contributed. They demonstrate my ability to write technical documentation and the technical depth of my contributions to the project.

4.1. Add Feature

The `add` command is used to add a place into TravelBuddy. The user can add the following details related to the place: name, country code, date visited, rating, address, description, photo (Optional) and Tag (Optional).

NOTE The country code adheres to the three-letter ISO-3166 standard. The full list of country codes can be found here.

4.1.1. Current Implementation

Figure 4.3.1 is a sequence of steps that illustrates the interaction between various classes when the `add` command is entered.

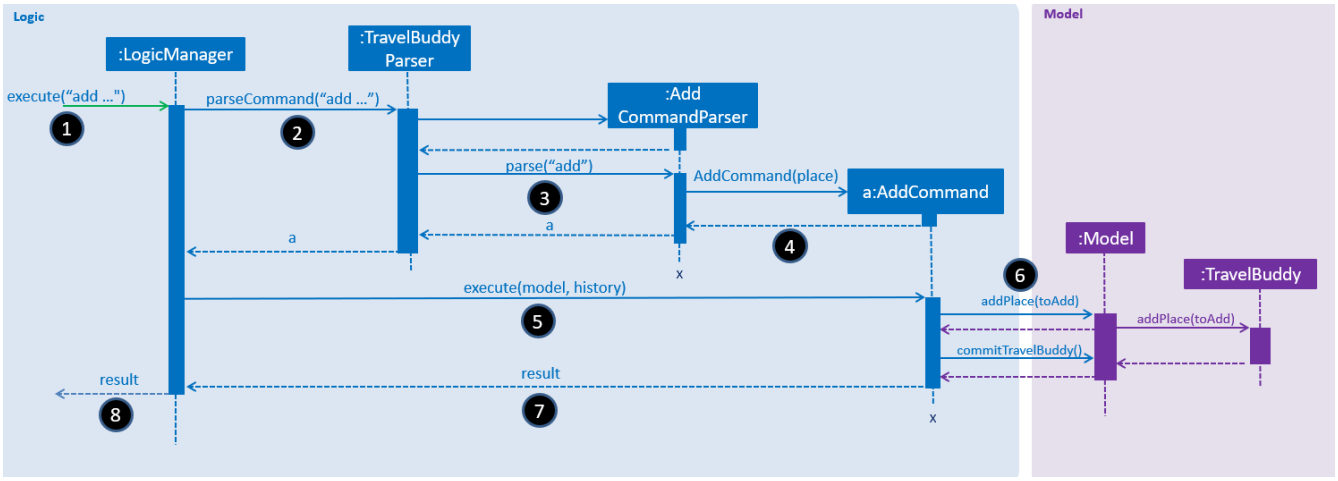


Figure 4.3.1: Execution sequence of the `add` command

`add n/NUS Computing cc/SGP dv/10/10/2017 r/3 d/My School a/13 Computing Drive, 117417 t/faculty`

1) The `String` user input is passed into the `LogicManager::execute` method of the LogicManager instance as the only parameter.

2) The `LogicManager::execute` method calls `TravelBuddyParser::parseCommand` which receives the user input as a parameter.

- The user input is formatted: the first `String` token is taken as the command word and the rest of the String is grouped as arguments to be used later by the `AddCommandParser`.
- From the command word, the `TravelBuddyParser` instance identifies the user input as an `add` command and constructs an instance of `AddCommandParser`.

3) TravelBuddyParser calls the `AddCommandParser::parse` method. The `AddCommandParser` takes in the rest of the string, which is `n/NUS Computing cc/SGP dv/10/10/2017 r/3 d/My School a/13 Computing Drive, 117417 t/faculty`

- The string is tokenised to arguments based on their prefixes.

```
ArgumentMultimap argMultimap = ArgumentTokenizer.tokenize(args, PREFIX_NAME,
     PREFIX_COUNTRY_CODE, PREFIX_DATE_VISITED, PREFIX_RATING,
     PREFIX_DESCRIPTION, PREFIX_ADDRESS, PREFIX_PHOTO, PREFIX_TAG);
```

- A check is made on the presence of the relevant prefixes `n/`, `cc/`, `dv/`, `r/`, `d/`, `a/`, `p/` and `t/`.
- When the mandatory prefixes are not present, a `ParseException` will be thrown with an error message on the proper usage of the `add` command.

```
if (!arePrefixesPresent(argMultimap, PREFIX_NAME, PREFIX_COUNTRY_CODE,
    PREFIX_DATE_VISITED, PREFIX_ADDRESS, PREFIX_RATING, PREFIX_DESCRIPTION)
    || !argMultimap.getPreamble().isEmpty()) {
    throw new ParseException(String.format(MESSAGE_INVALID_COMMAND_FORMAT,
        AddCommand.MESSAGE_USAGE));
}
```

- Otherwise, a `Place` object is constructed and used as a field in the creation of a `AddCommand` object.

4) The newly created `AddCommand` object is returned to back to the `LogicManager` instance through the `AddCommandParser` and `TravelBuddyParser` objects.

5) Once the control is returned to the `LogicManager` object, it calls the `AddCommand::execute` method.

- The method takes in a Model object to access the application's data context, the stored data of all places.
- The code snippet below shows the `AddCommand::execute` method.

```
public CommandResult execute(Model model, CommandHistory history)
    throws CommandException {
    requireNonNull(model);
    if (model.hasPlace(toAdd)) {
        throw new CommandException(MESSAGE_DUPLICATE_PLACE);
    }
    model.addPlace(toAdd);
    model.commitTravelBuddy();
    return new CommandResult(String.format(MESSAGE_SUCCESS, toAdd));
}
```

- A check is made on whether the place already exists in TravelBuddy. If it already exists, a `CommandException` will be thrown with an error message on the duplicate entry of the place.

6) The `Place` data is added into TravelBuddy.

- Here the `Model::addPlace` method is called, and it subsequently calls the `TravelBuddy::addPlace` method.
- Following which the `Model::commitTravelBuddy` method is called.

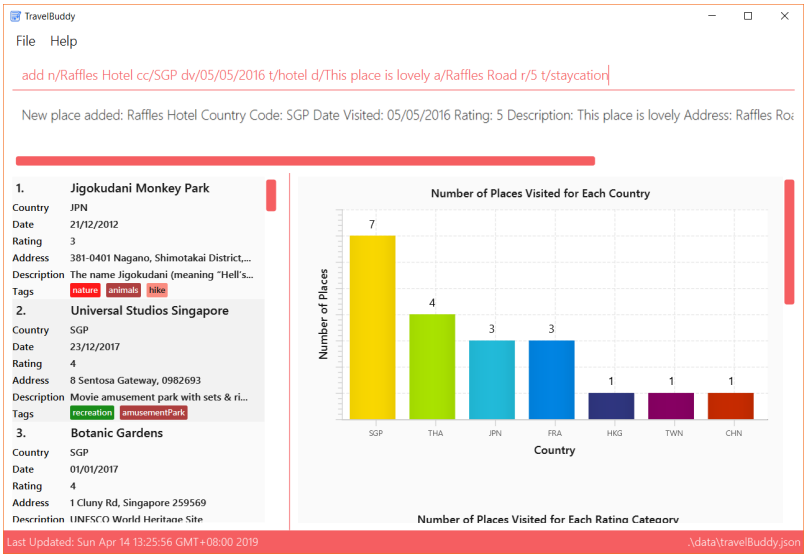
7) The `AddCommand::execute` execution completes by returning a new CommandResult that contains a success message to its calling method which is `LogicManager::execute`.

8) Finally, the `CommandResult` is returned to the caller of `LogicManager::execute` and the execution sequence ends.

Add Command

Given below is an example usage scenario and what the user will see in the GUI.

The user launches the application and enters the full add command `add n/Raffles Hotel cc/SGP dv/05/05/2016 t/hotel d/This place is lovely a/Raffles Road r/5 t/staycation 117417 t/faculty`. TravelBuddy will start executing the steps mentioned in Figure 4.3.1 and the output is shown below in Figure 4.3.2.



NOTE The command `add` is in lower-case. Mixed-case or upper-case commands are not recognised by TravelBuddy.

4.1.2. Design Considerations

Aspect: Data structure to store Country Codes

	Alternative 1 (current choice)	Alternative 2
Description	Use `enum` specified in `java.util.Locales`.	Create a data structure containing only the top 30 commonly traveled countries in the world.
Pros	**Ease-of-use**. This approach simply requires the importing of `java.util.Locales` to get the country codes. **Comprehensive**. `java.util.Locales` contains all of the 250 three-letter country codes as specified in ISO-3166.	**Fast**. This approach is computationally less intensive than Alternative 1. This is because there are only 30 country codes in the data structure.
Cons	**Slightly slow**. This approach is computationally more intensive than Alternative 1. This is because there are 250 country codes to search from.	**Tedious**. This approach requires the coder to search for the top 30 visited countries in the world and type out all the 30, three-letter country codes as specified in ISO-3166. **Not User-friendly**. If the user visited a country that is not in the top 30 list of countries visited, the user would not be able to add it into TravelBuddy.

Decision: Alternative 1. Alternative 2 may not fulfill all of the user requirements of adding any country code, but Alternative 1 does. Alternative 1's speed is only slightly slower than Alternative 2.

This marks the end of the excerpt from the Developer Guide.