Arielyte Tsen Chung Ming - Project Portfolio

PROJECT: TravelBuddy

Overview

This document serves to highlight my contributions to **TravelBuddy**, a travel log application that lets users efficiently manage their travel records, i.e. the places that they have visited. Additional features built into TravelBuddy allows them to search through the records, attach photos and view statistical models. This project is part of the CS2103 Software Engineering module in NUS, in which my team and I were tasked to morph a given addressbook product into a new product — TravelBuddy. The complete source code for TravelBuddy is available on GitHub.

My role was to design the chart feature. The following sections describes 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

Major Enhancement

The major enhancement I did was adding the ability to generate charts in real-time and via the `generate` command, details of which can be found below:

What it does: It allows users to generate charts based on their travel data that they have accumulated in the product.

Justification: This feature allows users to gather insights into their travel patterns. It improves the user-friendliness of the product, gives the product additional functionality and is the differentiating factor among other travel log products on the market.

Highlights: Due to the real-time update of the charts, this enhancement affects existing and future commands. Furthermore, the complexity of this enhancement is due to the fact that it links up with all the components of the existing architecture, as described below:

- **Logic:** The chart feature can be activated via the `generate` command. Hence, a parser is required to parse the command.
- **Model:** Any changes made to the travel records has to be reflected in the charts. Hence, relevant data is extracted from the latest version of the travel records.
- **Storage:** The extracted data has to be stored into the charts' respective JSON files.
- **User Interface (UI):** The charts have to be displayed to the users.

The enhancement required an in-depth analysis of various design alternatives. The challenge was to redesign the architecture to suit the enhancement while still maintaining the integrity of other existing commands and features.

Credits: Gson, a third-party library, was used in the enhancement of this product to convert Java Objects into JSON and back.

Code contributed to major enhancement: [Functional code] [Test code]

Minor Enhancement

The minor enhancement I did was to convert the `add` command's Email parameter in the previous addressbook product to a Description parameter in the current product.

Code contributed to minor enhancement: [Functional code]

Other contributions:

Table 1. Details of Other Contributions

Project management	Managed releases of `v1.2`, `v1.2.1`, `v1.3.1` and `v1.4` (4 releases) on GitHub
Enhancements to existing features	Updated the GUI color scheme to a `LightTheme.css` (Pull requests #88)
Documentation	- Improved readability to existing contents of the User Guide: #165 - Added user stories and use cases to Developer Guide: #170
Community	- Reviewed PRs (with non-trivial review comments): #115 - Contributed to forum discussions (example: 1) - Reported bugs and suggestions for other teams (examples: 1, 2, 3, 4, 5, 6, 7, 8)
Tools	- Integrated a third party library (Gson) to the project (#80) - Integrated a new GitHub plugin (Codacy) to the team repo (#113)

Contributions to the User Guide

Given below are sections I contributed to the User Guide. They showcase my ability to write documentation targeting end-users.

This is the start of extract of the User Guide.

Features

Generating charts: `generate`

Description: The `generate` command generates charts based on the list of places in TravelBuddy. TravelBuddy serves up three charts (Figure 4.17.1) that are the most relevant to you as a traveler:

- The number of places visited for each country
- The number of places visited for each rating category
- The number of places visited for each year

Shortcut: `g`

Format: `generate`

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

- By default, the charts are automatically generated each time TravelBuddy loads.
- The `generate` command always triggers the display of all three charts, as seen in Figure 4.17.1.
- The charts always update themselves in real-time.
  Example: When a place is added via the `add` command, the charts are automatically updated so that no `generate` command is necessary.
- If the list is empty, the `generate` command will not display any charts (Figure 4.17.4).
- You can type in any parameters after the `generate` command, TravelBuddy will simply ignore them (Figure 4.17.2).

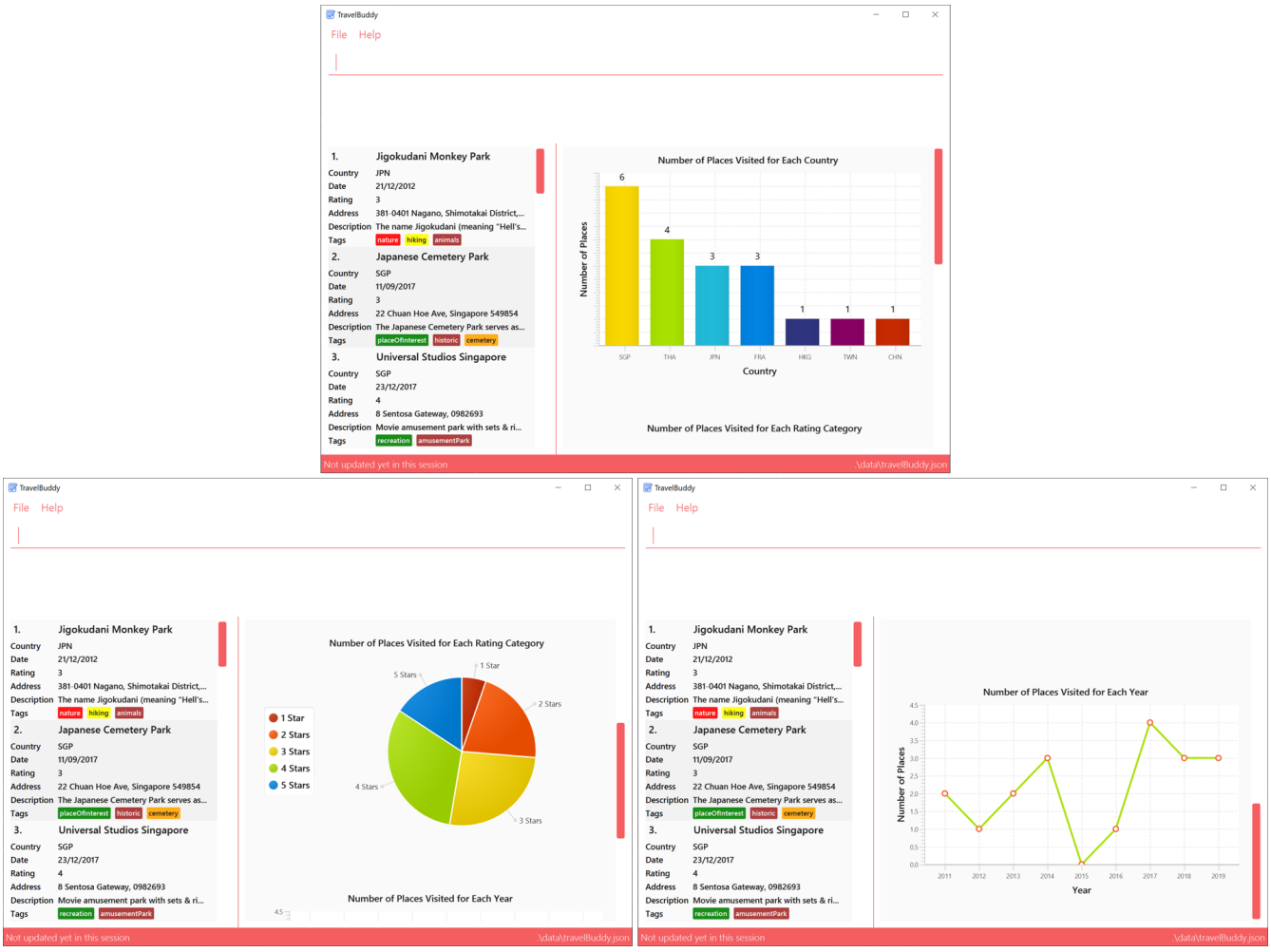

Figure 4.17.1: The number of places visited by country, rating category and year

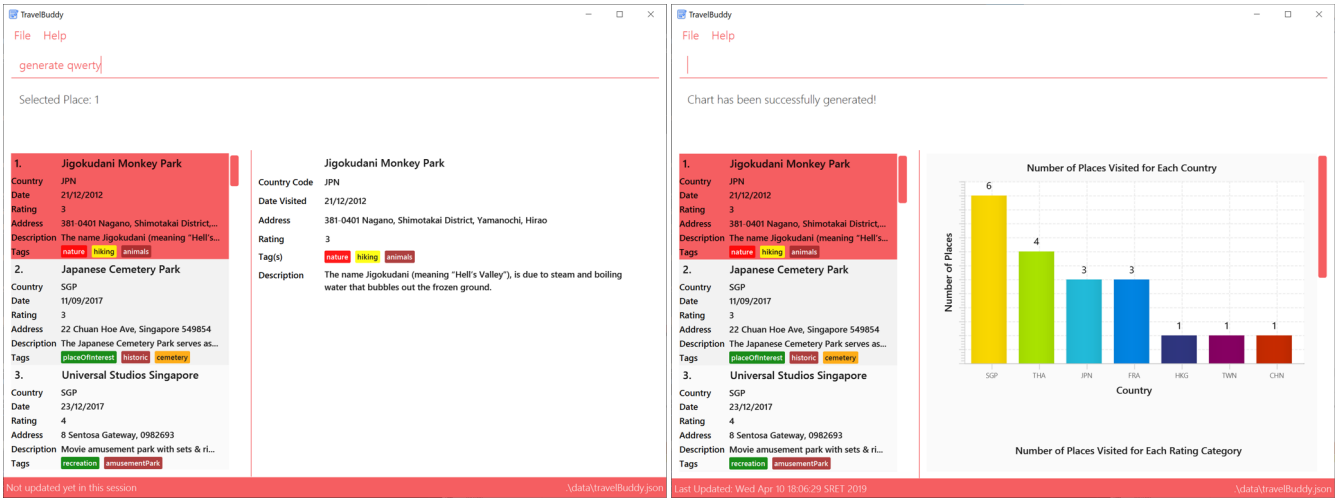

Figure 4.17.2: Before and after a parameter was used in the `generate` command

Figure 4.17.3: The charts were successfully generated

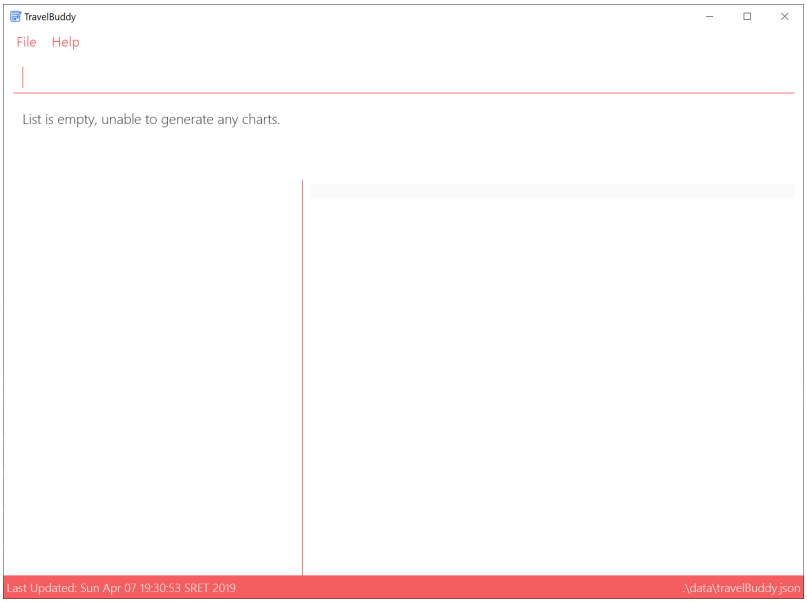

Figure 4.17.4: Unable to generate the charts as the list is empty

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

- `select 1`
  Selects the 1st place in the current list displayed.
  `generate`
  Generates the charts.
  Outcome: The charts were successfully generated, as seen in Figure 4.17.3
- `clear`
  Clears all places in the list.
  `generate`
  Generates the charts.
  Outcome: Unable to generate the charts as the list is empty, as seen in Figure 4.17.4

This is the end of extract of the User Guide.

Contributions to the Developer Guide

Given below are sections I contributed to the Developer Guide. They showcase my ability to write technical documentation and the technical depth of my contributions to the project.

This is the start of extract of the Developer Guide.

Implementation

Chart Feature

The Chart feature displays to users three different charts in TravelBuddy. They are:

- The Number of Places Visited by Rating Category
- The Number of Places Visited by Year
- The Number of Places Visited by Country

The Chart feature is activated in TravelBuddy by default when the application launches. Alternatively, the `generate` command is also used to generate the charts. The `generate` command does not require any parameters.

> **TIP** Instead of typing `generate`, you can simply type the shortcut `g`.

Current Implementation

Logic: The `generate` mechanism is executed by `GenerateCommand`, which extends from `Command`. A code snippet is shown below:

```
public CommandResult execute(Model model, CommandHistory history) {
    requireNonNull(model);
    model.setChartDisplayed(true);
    model.commitTravelBuddy();
    if (model.getFilteredPlaceList().isEmpty()) {
        return new CommandResult(MESSAGE_EMPTY);
    } else {
        return new CommandResult(MESSAGE_SUCCESS);
    }
}
```

The operations implemented are:

- `GenerateCommand#setChartDisplayed(chartDisplayed)` - Signals to the UI to display the charts.
- `GenerateCommand#commitTravelBuddy()` - Saves the current TravelBuddy state from its history.
- `GenerateCommand#getFilteredPlaceList()` - Verifies if the list is empty.

These operations are exposed in the `Model` interface as `Model#setChartDisplayed(chartDisplayed)`, `Model#commitTravelBuddy()` and `Model#getFilteredPlaceList()`.

Model: The chart generation mechanism is facilitated by `VersionedTravelBuddy`, which extends from `TravelBuddy`, as seen in the Model Class Diagram in [ModelClassDiagram].

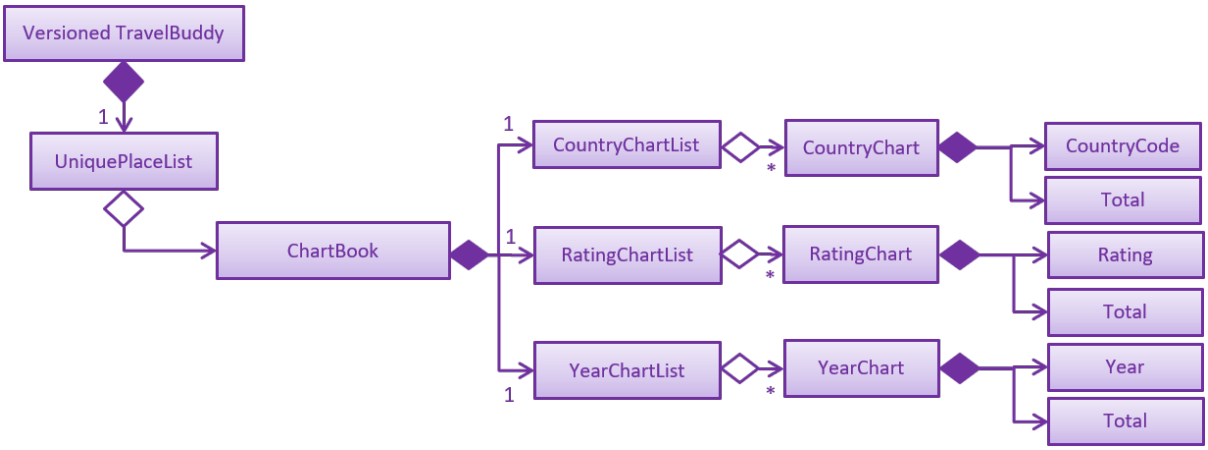

Figure 4.4.1: OOP Diagram

For this section on Charts, the focus is on the class diagram as seen in Figure 4.4.1. The step-by-step explanation of the class diagram can be found below:

1. The `ModelManager` is a container for a `VersionedTravelBuddy` object.
2. VersionedTravelBuddy consists of a `UniquePlaceList` object.
3. `UniquePlaceList` is a container for one or more `ChartBook` objects and for one or more `Place` object.
4. `ChartBook` consists of a `CountryChartList` object, a `RatingChartList` object and a `YearChartList` object.
5. `countryChartList` is a container for one or more `CountryChart` objects. Similarly, `ratingChartList` is a container for one or more `RatingChart` objects and `yearChartList` is a container for one or

more `YearChart` objects.

6. `CountryChart` consists of a `CountryCode` object and a `Total` object. Similarly, `RatingChart` consists of a `Rating` object and a `Total` object and `YearChart` consists of a `Year` object and a `Total` object.

Logic & Model Interaction: Having discussed Logic and Model, we can now model the workflow of the `generate` command. This can be accomplished using an activity diagram, as seen in Figure 4.4.2.

Additionally, we want to be able to capture the interaction between multiple objects for the `generate` command. This can be accomplished using a sequence diagram, as seen in Figure 4.4.3 below. The step-by-step explanation of the sequence diagram is as follows:

1. The user enters the command `generate` without any parameters.
2. The command is processed by the Logic component, which will then call `LogicManager#execute()`.
3. The `GenerateCommand#execute()` method is invoked.
4. The `Model#setChartDisplayed()` method is invoked with the argument `true`. The `Model#commitTravelBuddy()` method is also invoked.
5. The `TravelBuddy#commitTravelBuddy()` method is invoked by `Model`.
6. The `ChartBook#commitChart()` method is invoked by `TravelBuddy`.
7. A result object is returned.

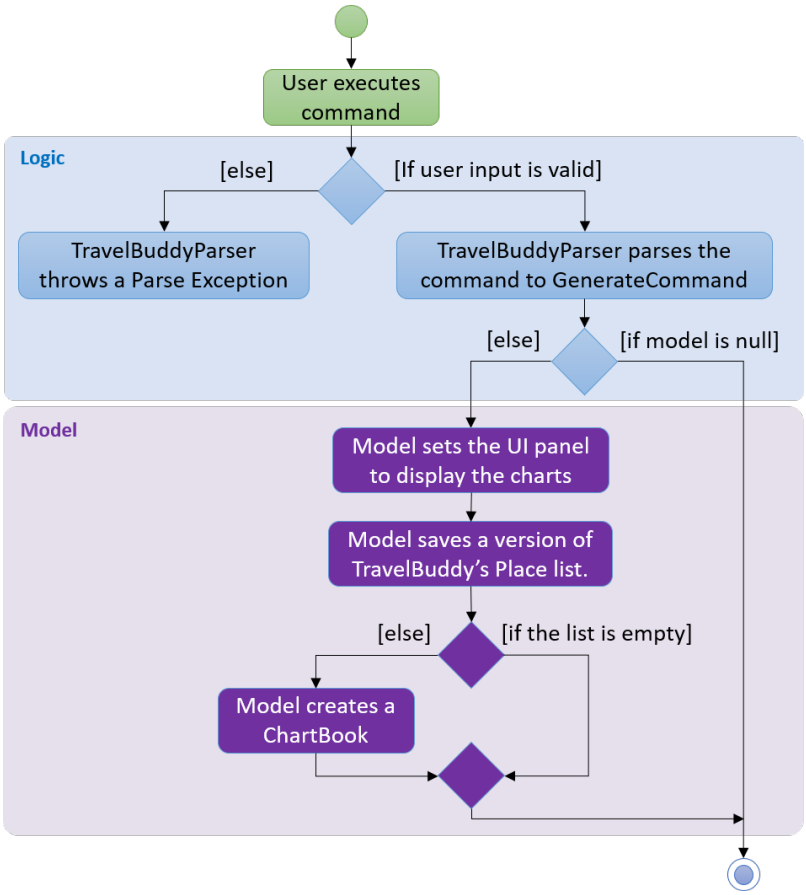

Figure 4.4.2: Activity Diagram for the `generate` command

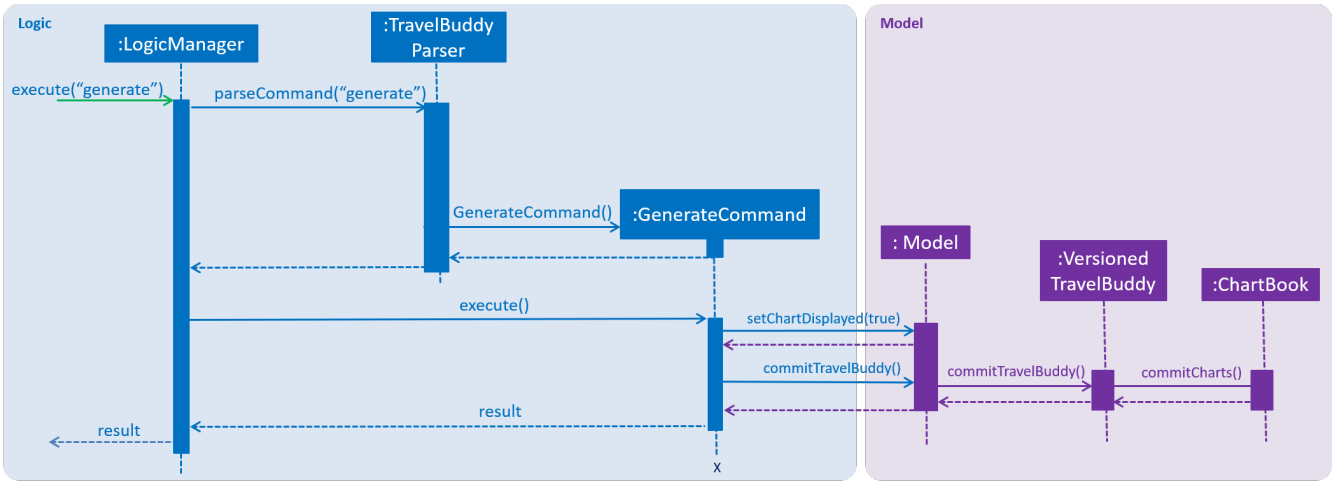

Figure 4.4.3: Sequence Diagram for the `generate` command

Storage: The Chart's storage is handled by `JsonChartBookStorage`, which implements from `ChartBookStorage`. The three main methods it implement are:

- `saveCountryChart(filePath)` - Saves the countries data into a JSON file.
- `saveRatingChart(filePath)` - Saves the ratings data into a JSON file.
- `saveYearChart(filePath)` - Saves the years data into a JSON file.

A code snippet for `saveCountryChart(filePath)` is shown below. A third-party library called Gson was used to convert Java Objects into JSON and back. The Gson API can be found here.

```
public void saveCountryChart(ReadOnlyCountryChart countryChart, Path filePath) {
    requireAllNonNull(countryChart, filePath);

    Gson gson = new GsonBuilder().setPrettyPrinting().create();
    try {
        FileWriter fileWriter = new FileWriter(String.valueOf(filePath));
        gson.toJson(countryChart, fileWriter);
        fileWriter.flush();
    } catch (IOException ioe) {
        logger.warning(ioe.getMessage());
    }
}
```

Generate Command

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

- By default, the charts are automatically generated each time TravelBuddy loads.
- The `generate` command always triggers the display of all three charts.
- The charts always update themselves in real-time.
  Example: When a place is added via the `add` command, the charts are automatically updated so that no `generate` command is necessary.

- The chart will not display anything when the list is empty.
- You can type in any parameters after the `generate` command, TravelBuddy will simply ignore them.

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

Step 1: By default, the charts are displayed when TravelBuddy launches. To navigate away from the charts, type in `select 1`.+ **Outcome:** The first index in the place list will be selected and displayed on the right-hand side of the panel..

Step 2: Type in `generate` to generate the charts..
Outcome: The charts will be displayed on the right-hand side of the panel..

Design Considerations

Aspect: How Chart Generation Executes

	Alternative 1 (current choice)	**Alternative 2**
Description	Updates the charts both in real-time and when the `generate` command is used.	Updates the charts only when the `generate` command is used.
Pros	**Ease-of-use**. This approach is more user-friendly, as users do not need to type an additional `generate` command after changes are made to the Place list. **Accuracy**. This approach is more accurate as the charts reflect the latest changes regardless of whether the user types the `generate` command.	**Scalability**. This approach is computationally less intensive, as the charts are only generated when required.
Cons	**Scalability**. This approach is computationally more intensive, especially when the Place list is huge, as all three charts need to be regenerated every time a change is detected.	**Ease-of-use**. This approach is Less user-friendly as users will need to type the `generate` command for the charts to reflect the changes made to the Place list. **Accuracy**. This approach is less accurate as the charts does not reflect the latest changes until the user types the `generate` command.
Example	As shown in Figure 4.4.4, before the `edit 1 cc/USA` command was executed, the chart did not have a separate bar for USA. After the command was executed, the chart updated in real-time to include a bar for USA. All this was done without invoking the `generate` command.	N.A.

Decision: Alternative 1, which is to update the charts in real-time, was adopted as it promotes ease-of-use, so users are not required to type in an additional `generate` command whenever changes are made to the Place list. Moreover, Alternative 1 is the more accurate option of the two, as the chart reflects the latest changes even if the user forgets to type the `generate` command.

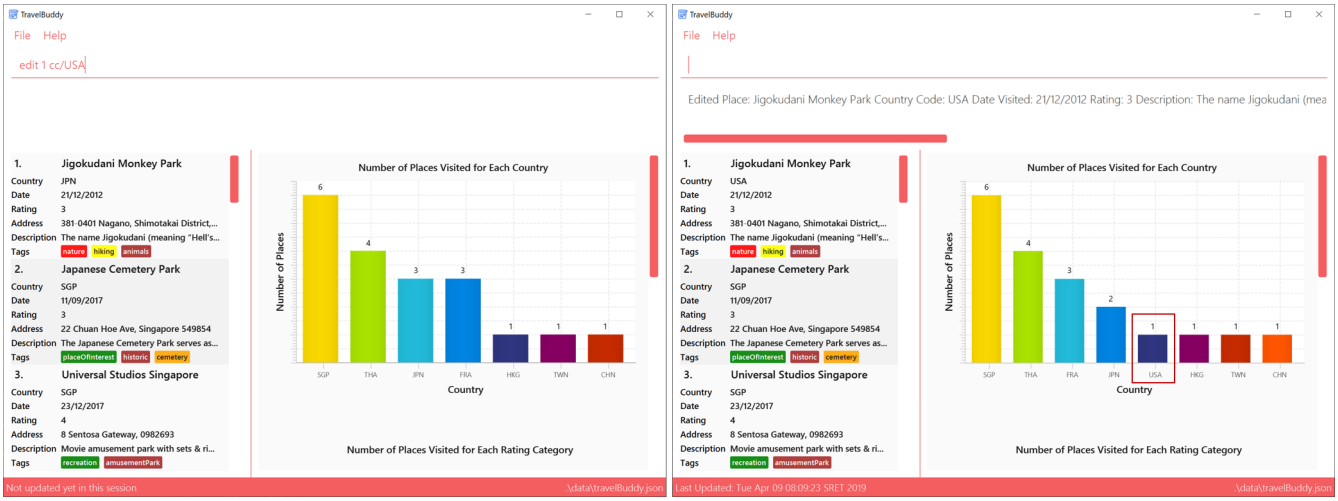

Figure 4.4.4: A comparison before and after the `edit` command was executed

This is the end of extract of the Developer Guide.