



Overview

This Project Portfolio Page will be describing the project Cube as well as my contributions towards the project.

About the Project

Cube is a light weight inventory management system with Graphical User Interface (GUI) targeted at sellers looking to set-up a small online marketplace. The main features of Cube can be divided into four categories: manipulating product list, retrieving and analyzing information from product list, features related to profits, and file storage utilities. These main features enable our targeted users to easily keep track of the various products and profits earned from the products they are selling.

Below is welcome screen when user launches the GUI of Cube.

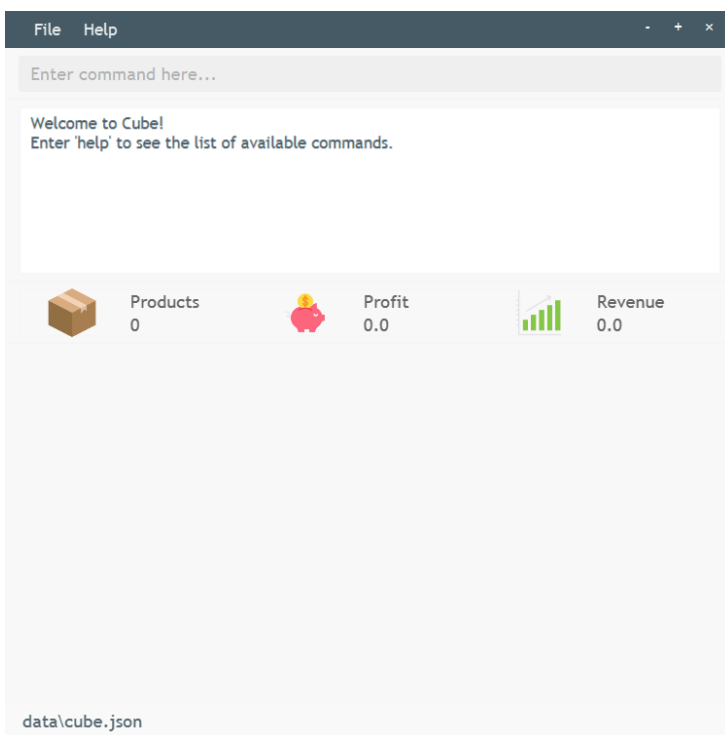


Figure 1. The graphical user interface of Cube

About the Team


Cube is a group project for an introductory level module about software engineering. My team of five Year 2 computing students, including me, started with a basic command line interface desktop to-do list application and

morphed it into the stock inventory management system, Cube, in 6 weeks from early October to mid-November, 2019.

About the Author

In Cube project, my role was to design and code for parser and basic CRUD (create/ retrieve/ update/ delete) commands. I also helped with implementing promotion feature. The following sections will illustrate my contributions towards project codes and project documentation with more details.

Note that the following symbols and formatting used in this document:

<code>command</code>	A <code>grey</code> highlight denotes code snippets or a command.
important	A yellow highlight indicates key points of the instructions to be noted by the user.
<code>keyboard</code>	A <code>green</code> highlight indicates a keystroke.
	This symbol indicates that there is some additional information to be taken note by the user.



Summary of Contributions

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

Enhancement added: I tailored the former to-do list parser and CRUD commands to the needs of our Cube project. I also helped with implementing promotion feature.

What it does:

- The parser receives user input from GUI. It will first check the validity of command and parameters that user inputs according to rules specific to Cube. After validating and parsing the user input, the parser will call the corresponding command and pass the correct parameters to the command.
- Basic CRUD commands refer to `add` / `list` & `find` / `update` / `delete` in Cube's context. These support the basic operations on the product list inside Cube.

- The `promotion` command allows users to add in new promotion, delete promotion by index, and view currently existing promotions.

Justification:

- Parser and CRUD commands are fundamental parts of a project. They need to be tailored to the specific needs of our project Cube in order to get it execute basic functions correctly.
- As Cube is targeted at online shop owners, the promotion command is necessary because it fits the real-life situation in an online food shop and allows users to manage their promotions easily.

Highlights:

- An in-depth analysis of user inputs is necessary to parser design. This is challenging because there are many possible illegal inputs that users may give to Cube. If they are not handled correctly, it may cause fatal error to the program. Implementing and maintaining the vulnerable parser honed and demonstrated my ability to catch and deal with potential bugs.

Credits:

When I designed the architecture of parser, I referred to the addressbook-level3 parser architecture which can be accessed through [this link](#).

Code contributed:

Please click these links to see samples of my code:

[\[Project Code Dashboard\]](#): This is the link to the report for my codes.

[\[Parser\]](#): This is a link to the package of parser classes that I coded.

[\[Test code\]](#): This is a link to the test codes for all parser classes.

Other contributions:

Project management:

- There were a total of 5 releases of Cube, from version 1.0 to 1.4. I managed releases version 1.0 to 1.2 (3 releases) on GitHub.
- There were a total of 4 milestones during Cube's developing process on GitHub, from v1.1 to v1.4. I managed milestones from v1.1 to v1.3 (3 milestones).

Community:

- I reviewed pull requests from my teammates with non-trivial comments (sample PR [\[PR\]](#)).
- I contributed to forum discussions and project management (sample issue: [\[issue 1\]](#) [\[issue2\]](#)).

- I reported bugs and offered suggestions for other teams in the class (sample issue: [\[issue\]](#))

Tools

- I managed checkstyle of the whole project (sample PR: [\[PR\]](#)).



Contributions to the User Guide

This section shows an excerpt of contributions I made to the User Guide of Cube.

4.2 Finding a product: `find`

This command allows you to find one or some specific products you have added into Cube, as well as to give you an option to sort the result list ascendingly by expiry date, product name or stock quantity when you find by type.

4.2.1 Command syntax

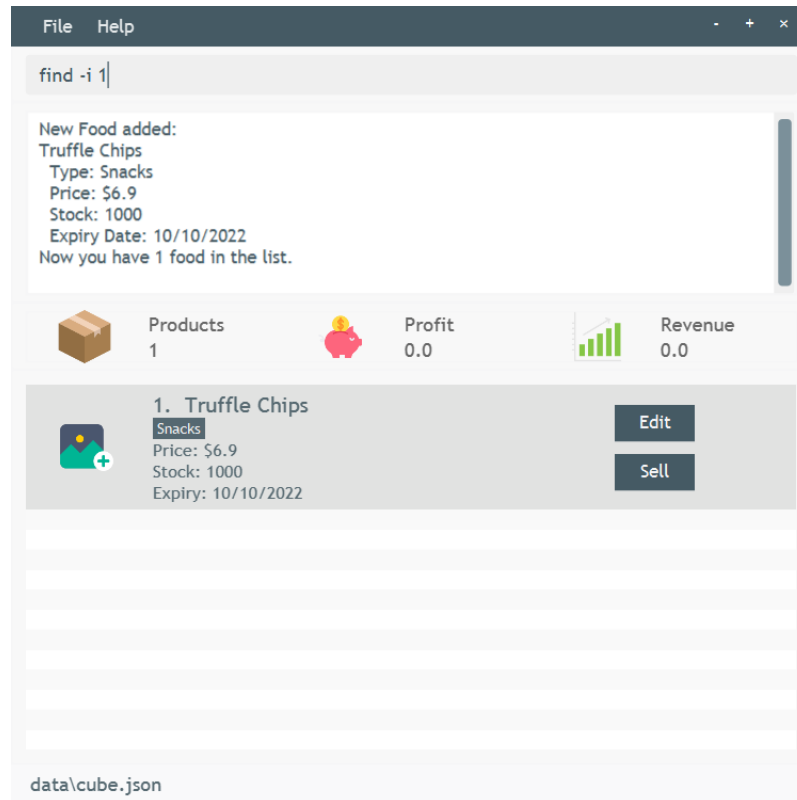
```
find [-i INDEX] | [-n FOOD_NAME] | [-t FOOD_TYPE] [-sort (expiry | name | stock)]
```

- Finds a food product or some food products with the following parameters.
 - `[-i INDEX]` : Index of the product in list (viewed by `list` command)
 - `[-n FOOD_NAME]` : Name of the product you want to find
 - `[-t FOOD_TYPE]` : Type of the product you want to find.
 - `[-sort (expiry | name | stock)]` : Type of `-sort` to apply when viewing the result of `find` command. Cube currently supports sorting by expiry date, product name and remaining stock quantity in an ascending order.

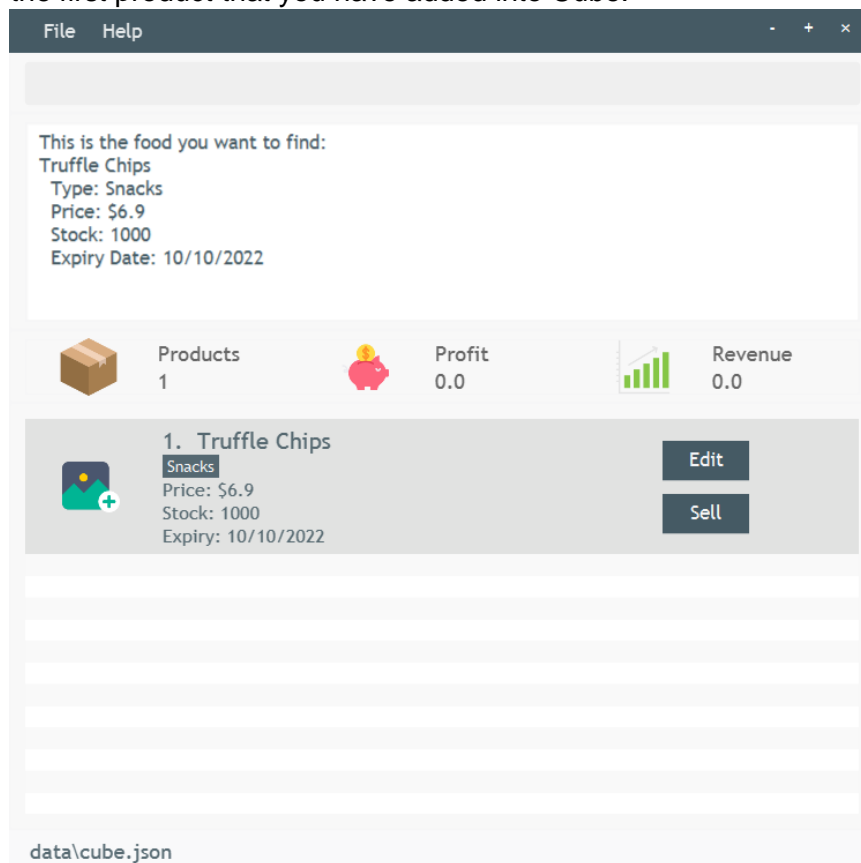
4.2.2 Example usage

You have added several products and would like to list all products in your inventory.

1. Type `find -i 1` in the command box in Cube and press **Enter** to view your first product in the list. The index in food list starts from 1.



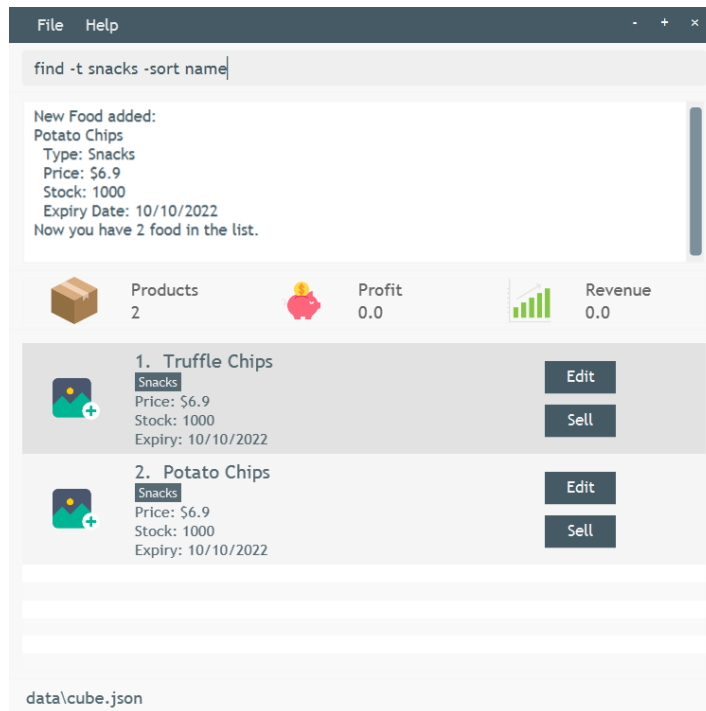
2. You should see the first product that you have added into Cube.



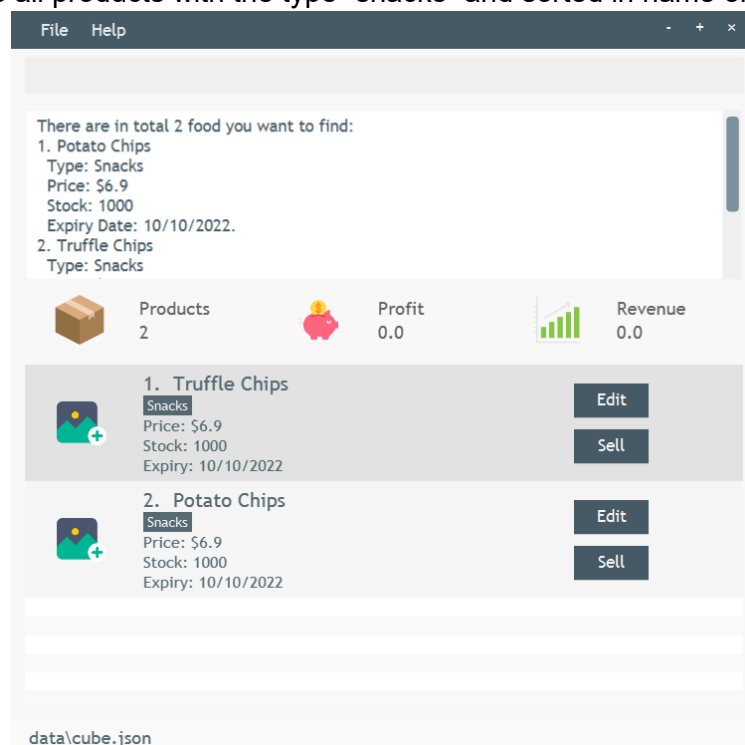
4.2.3 Another example usage

You would like to see products in your food list with type “snacks” and sort them in name order.

1. Type `find -t snacks -sort name` in the command box in Cube and press **Enter**.



2. You should now see all products with the type “snacks” and sorted in name order.





In Cube, `find` command supports **finding by partial name**. For example, `find -n test` will also give you “test1”, “trytest”, and “try test this” as results.



Contributions to the Developer Guide

This section shows an excerpt of contributions I made to the Developer Guide of Cube.

7.2 Implementation of Manipulation on Food List

Basic stock inventory management commands include create, retrieve, update, and delete (CRUD). Among them, manipulation of food list involves create (add), update (update), and delete (delete). In this section, `delete` will be taken as an example to introduce the implementation of food list manipulation in Cube.

7.2.1 Implementation Description

The command format for delete is `delete -i [INDEX] | -n [FOOD_NAME] | -t [FOOD_TYPE] | -all`. The implementation of `delete` command allow users to execute the following operations:

- `delete -i [INDEX]`: delete a product from food list by index
- `delete -n [FOOD_NAME]`: delete a product from food list by name
- `delete -t [FOOD_TYPE]`: delete all products of a given type from food list
- `delete -all`: delete all products from food list

Figure 8 below shows the Sequence Diagram when user inputs `delete -all`.

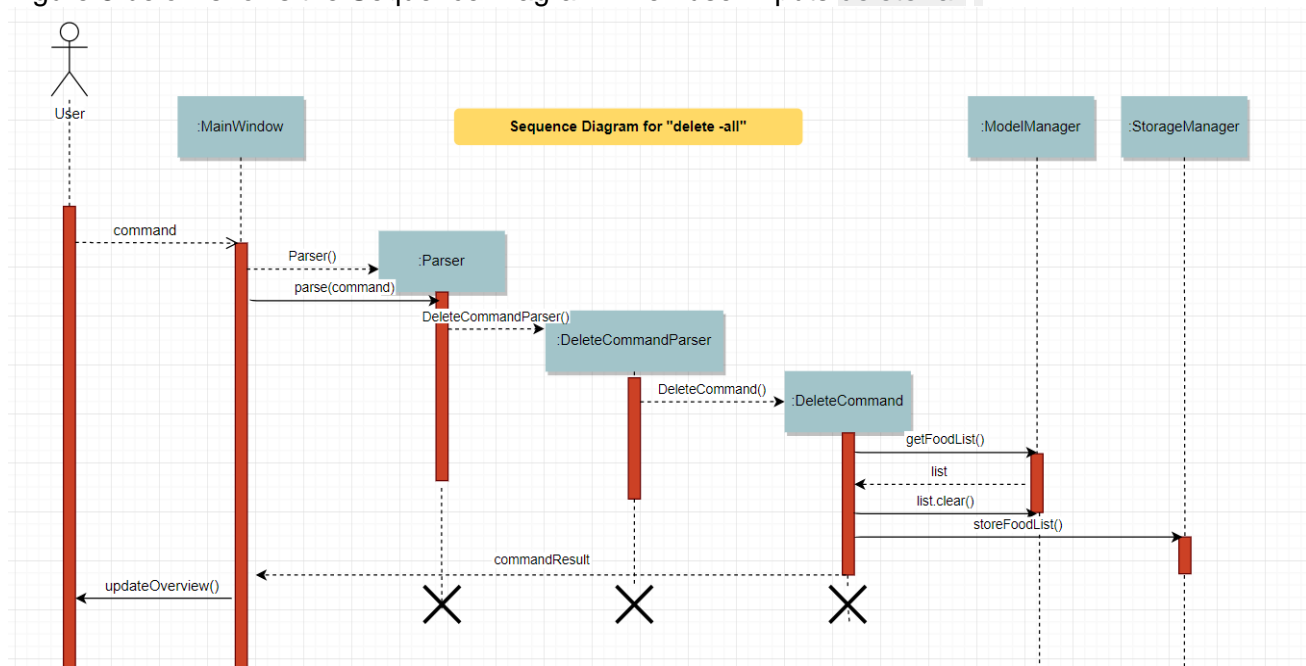


Figure 8. UML Sequence Diagram for delete -all

Explanations of the steps that Cube takes to execute delete -all:

Step 1. User inputs command “delete -all” through GUI main window.

Step 2. GUI main window creates a new Parser class and calls the parse() method with the command.

Step 3. Parser extracts “delete” from command. According to the command type, it creates a DeleteCommandParser class with the command.

Step 4. DeleteCommandParser parses the command and extracts delete type “-all”. It then creates a DeleteCommand and passes “-all” to it.

Step 5. According to the instruction “-all”, DeleteCommand gets food list from model manager and clears the list using clear() method declared inside FoodList class. It then calls storage manager to store the new list.

Step 6. DeleteCommand will return the command result to GUI main window, and GUI will display the result to the user.

7.2.2 Implementation Considerations

Aspect: Parsing user input and calling corresponding command

Alternative 1: Creating a whole class for all parsers and another class for all commands

- Pros: It is easy to code and view since all relevant variables and methods are together.
- Cons: Testing will be difficult as the codes are bulky. The principle of separation of concerns is also violated.

Alternative 2: Design patterns: Command pattern

- Pros: It supports polymorphism of different parser and command type as they all inherit from a general parser or command prototype class.
- Cons: Learning a new design pattern takes time and it is more challenging to understand and code.

After considering the two ways of implementation described above, the second way, command pattern design, is finally chosen due to its clarity and extendibility. By defining a general parser or command prototype class, it is very easy to follow the defined format and add in new implementation of parser and command without worrying about the problem of compatibility.