Programmer's Documentation

Cheap Staples

Table of Contents:

Overview of System……...1

Files……1

High Level Overview……..……2

Low Level Runthrough……..4

Data Structs & Imports Used……6

Overview of System

Cheap Staples is a program designed to assist low-income workers and families to find cheap staple groceries in nearby stores. The user will run the program, and select from a predetermined group of ingredients to see a price comparison between stores. From there, the system will grab the necessary data from a MySQL database, format it, and display it on screen to the user.

For this system, we utilize a web scraper to grab prices from stores' online websites, then store the data within a database.

This system was written completely in Python and uses various libraries to run.

Files

`CheapStaples.py`	- Main executable file to run the *Cheap Staples* program
`installer.py`	- Python executable to install necessary Python libraries.
`UI.py`	- This file creates and sets up the user interface. It contains functions that will read what the user is putting into the list, and return data to its caller or a command when certain buttons are pressed
`Safeway_Scraper.py`	- Beautiful Soup web scraper that will access Safeway's website, search for all ingredients within the predetermined set, then format and store the data within a MySQL database
`Target_Scraper.py`	- Beautiful Soup web scraper that will access Target's website, search for all ingredients within the predetermined set, then format and store the data within a MySQL database
`Table_Class.py`	- Declaration of custom Python class "Table" used to format data retrieved from the database so that the UI can easily access all it needs to display it on screen
`database_functions.py`	- This file is used when UI.py passes in a list of items to retrieve from the database. The data will be retrieved,

	and formatted into a “Table” , then return the retrieved data in a table format.
`README.pdf`	- Informative instructions on how to use the repo
`Documentation`	- Folder of program documentation, instructions, & information

High Level Overview (between modules)

Cheap Staples utilizes multiple modules held together by a main function (`CheapStaples.py`) to operate. Here, we’ll go over on a macro-scale how each of these modules interact with one another. To start off, we’ll define our modules:

Module Name	Corresponding File(s)
Ingredient Choose & Location-Price Interface	`UI.py`
Database Gathering Module	`database_functions.py` `Table_Class.py`
Location-Price Ingredient Database	`database_functions.py`
Grocery Store Web Scraper	`Safeway_Scraper.py` `Target_Scraper.py`

Now, when running the system, `CheapStaples.py` will start by calling `UI.py` to load the interface. From there, there are two use cases:

- **Use Case #1: View Staple Prices**
- **Use Case #2: Update Data**

Both of these cases use the modules differently, so here’s how each one works:

Use Case #1: View Staple Prices

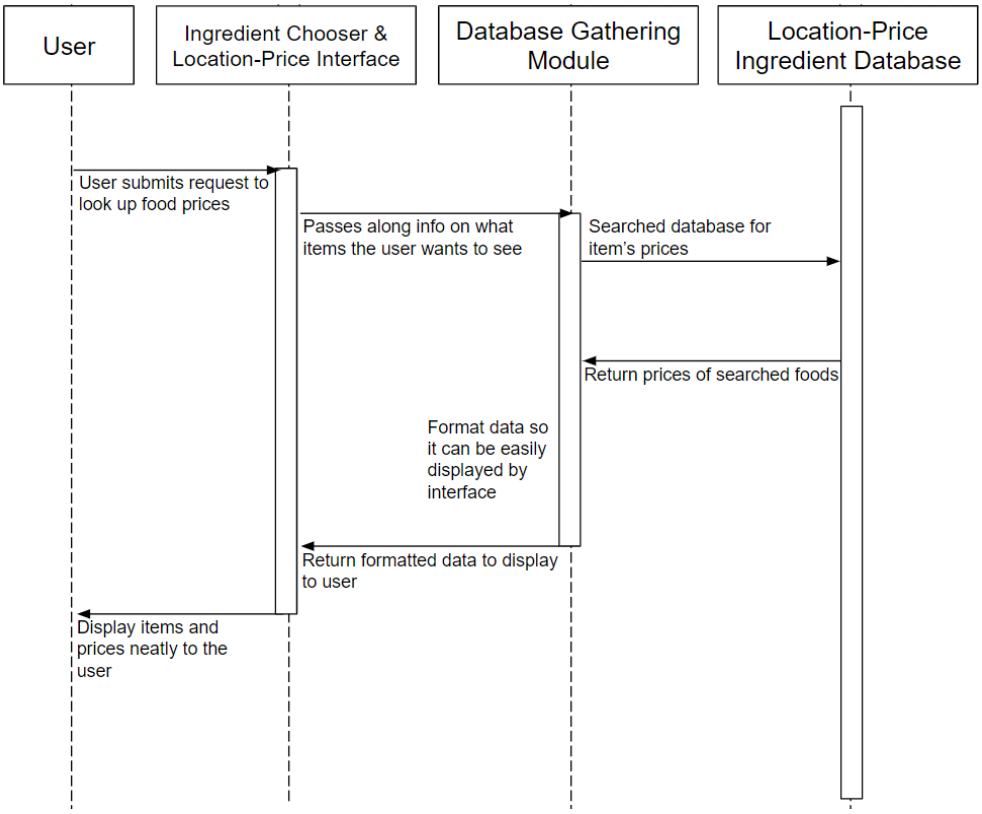

Figure #1: Use case where the user selects items to view staple prices

After the user begins to run `CheapStaples.py`, and selects various items from the user interface, they can either press “View Selected” or “View All” to see price comparisons. Pressing these buttons will return the items the user selected in the form of a list of strings back to the main function `CheapStaples.py` so that it can then use that list of strings as an input to the Database Gathering Module. The main function will call upon `database_functions.py` to gather the necessary data from the Location-Price Ingredient Database, and format it into a “Table” class before giving it back to the main function. `CheapStaples.py` will then use that table to display it via the interface.

Use Case #2: Update Data

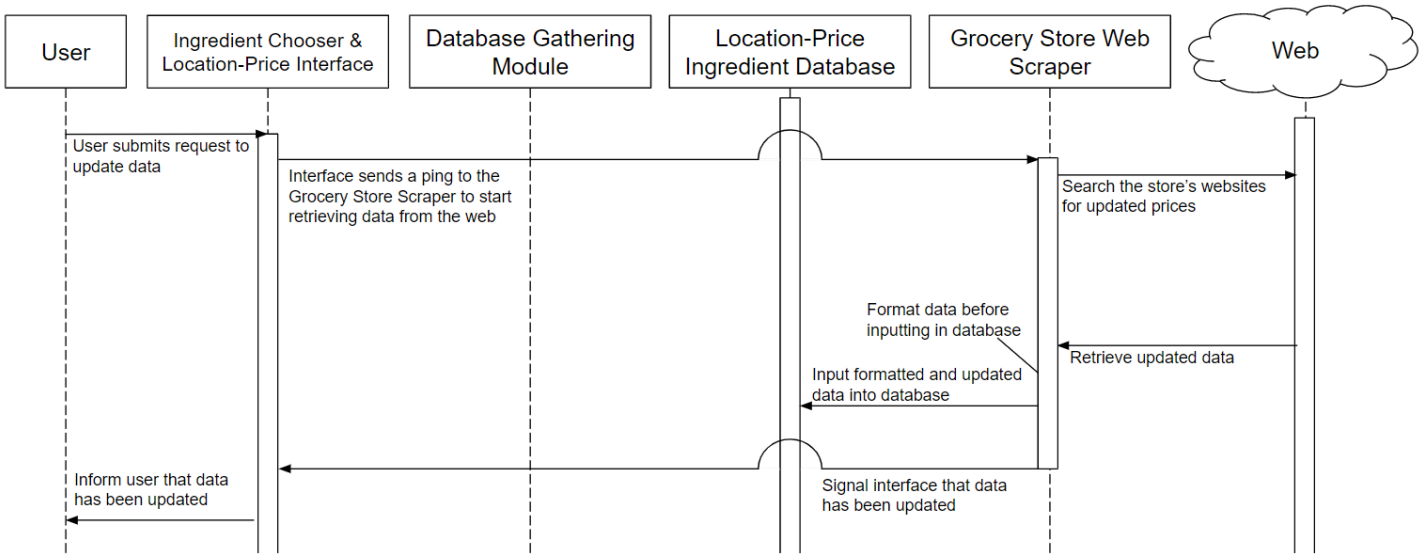

Figure #2: Use case where a user chooses to update the prices within the database.

After a user runs `CheapStaples.py`, they can choose this use case by pressing the "Update" button. This will cause `UI.py` to send an event call that will trigger the Grocery Store Web Scrapers (`Safeway_Scraper.py` & `Target_Scraper.py`) to run. These web scrapers will grab the current prices from the website, format them into strings, and save them to the Location-Price Ingredient Database before signaling `UI.py` again that it's finished with a timestamp string. Then, `UI.py` informs the user that prices have been successfully updated and displays the timestamp of when it was last updated.

Note: The user can still go back to use case #1: View Staple Prices whilst the scrapers are scraping, but the updated prices won't be visible until the scrapers have finished.

Low Level View (within modules)

Each module has a bunch going on within them to go from their input to output. This section will dive into each module, looking at how each one internally works.

Main Function (`CheapStaples.py`)

This file really serves as the instantiator of the user interface and is really quite short and simple. When executed, it'll call upon UI.py to create an instance of the program, and from there, all of the input and actions with the interface will make event calls that update lists, and return them to other modules in communication.

Ingredient Chooser & Location-Price Interface (`UI.py`)

UI.py

This file creates the graphical user interface (GUI) for the Cheap Meals system using the customtkinter library. The module first creates the ScrollableCheckBoxFrame and appends a checkbox for every supported staple ingredient to it. The main window is configured in the App class and instanciates a frame for the buttons “View Select”, “View All”, and “Update Prices”. The App class also contains the functions that are executed when its respective button is clicked. Button function view_all_button_clicked() shows the price and supported staples and function view_select_button_clicked() shows only the prices of the checked checkboxes. Button function update_prices_clicked() updates the database prices by calling update_prices() in the database_function.py module. Once finished updating, the function creates a separate popup window notifying the user that the prices have been successfully updated.

Database Gathering Module (`database_functions.py & Table_Class.py`)

database_functions.py

This file contains the functions necessary to call the individual web scrapers that populate the Location-Price Ingredient database. It also provides methods to write and execute MySQL queries that obtain the prices of ingredients from each grocery store location. Lastly, it provides the format_and_display() function, which formats MySQL data entries to be displayed to the user.

Table_Class.py

This file declares the custom “Table” class and methods. Input when initializing is a list of tuples of strings. It also contains other methods that manipulate this class that are discussed more in detail later in this document.

Note: Uses tkinter in this file to create the visualization of the table.

Location-Price Ingredient Database (MySQL)

The Location-Price-Ingredient MySQL Database contains all necessary information required to display the user Cheap Staple’s available ingredients to choose from alongside their corresponding price and location.

Grocery Store Web Scrapers (`Safeway_Scraper.py` & `Target_Scraper.py`)

Safeway_Scraper.py

This file iterates through Safeway URLs, scrapes the cheapest food and price, and then outputs the specific food and price into a tuple to be used by the SQL database. The scrape function searches for a specific food element by searching for the “h1” header in the browser html and searches for the price element by searching for the “product-price” class in the browser html. The element text is then cleaned up to just have the needed text. The food text and

price text are stored in different lists and then using the merge function, combines both list into a tuple.

Target_Scraper.py

This module goes through a list of urls that correspond to the list of staple foods that we have set out, and scrapes all the data on that web page to find the price of that item. This module uses the python libraries `beautifulsoup4` and `requests` to carry out its objective. In order for the web page to believe that the scraper is not a robot, there is a header that is added when requesting which is user agent information. When going through each url, it searches through the body of the web page, where it will find the current price. Then it will store it in a list of lists that contain all the names of the staple foods and their corresponding prices. After iterating through the url list, the module will return the updated list of lists with the names of items and their corresponding prices.

Data Structs & Imports used

Python Library	Structs Used	Documentation
customtkinter	App()	customtkinter Github Wiki Documentation
tkinter	Tk()	tkinter Tk Documentation
bs4 (Beautiful Soup)	BeautifulSoup	Beautiful Soup Documentation
requests		Requests Documentation
datetime		Python Datetime Documentation

Custom Structures

Table (from Table_class.py)	- Custom class created to assist in formatting the data in preparation for tkinter to use to visualize the data. Contains many variables like self.rows, self.cols, self.data, self.cell_colours
`Table.__init__(list)`	- Initializes the Table class given a list of tuples
`Table.setTableFormat(font, font_size, cell_width, cell_colour)`	- Changes the style of the table cells. So when next assigning a cell, it'll do so with these settings specified.

`Table.calcCellColours()`	- Look at each row of the table and highlight the cell with the lowest price green. Does this by manipulating Table.cell_colours—and each entry in that table corresponds to a cell in the table, then when creating the table, it'll read from Table.cell_colours to see how to colour each cell
`Table.visualizeTable()`	- Creates the table using tkinter Tk() struct. Reads from all variable within Table to calculate the size of the table, data that goes within each cell, colours of each cell, and style of each cell