# CalcuLateMeal Product Guide

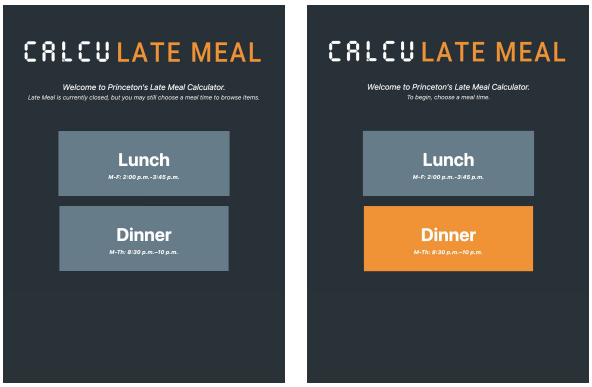
## USER GUIDE

CalcuLateMeal is a mobile and desktop-friendly web app that allows users to plan out a Late Meal purchase. Our target audience includes Princeton students who are on the dining plan, have access to Late Meal, and are subject to Late Meal's policies (ex. limit of two packaged items). However, our audience may also include anyone visiting the Frist Gallery while it is open, in which case, many of Late Meal's policies would not apply.

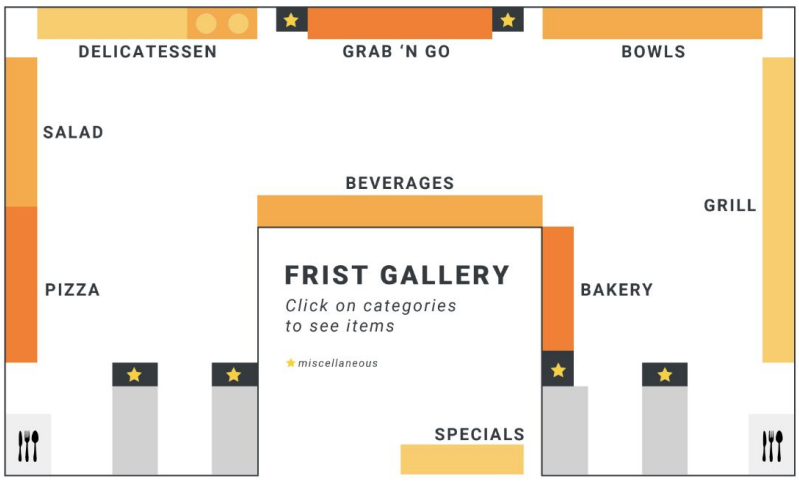
To use our web app, simply visit the URL: http://calculatemeal.herokuapp.com. On our website's "Install" page, which you can access here, you will find information on how to add CalcuLateMeal onto the homescreen of your Android and iOS mobile devices.

The following steps present instructions on how to use the most important features of the app:

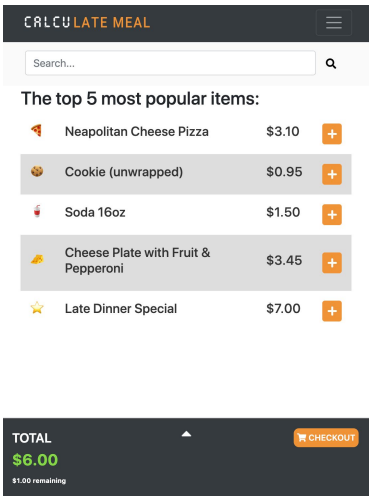
1. Searching for Items:
    a. ***Time***: On the Splash Page, select whether you would like to browse for Late Lunch or Late Dinner. If it is currently Late Meal hours, the corresponding time selection is highlighted orange (right image). Otherwise, it will notify you that late meal is currently closed, but it will still allow you to select a time to view items (left image).




    b. **Category**: On the home screen, the map of Frist Gallery (below) shows the different categories of Late Meal. Click on any section of the map to view all the items in that category. On the results page, the list of items generated can be sorted by alphabetical order by clicking on the heading "Item Name" or by price by clicking on the heading "Price."



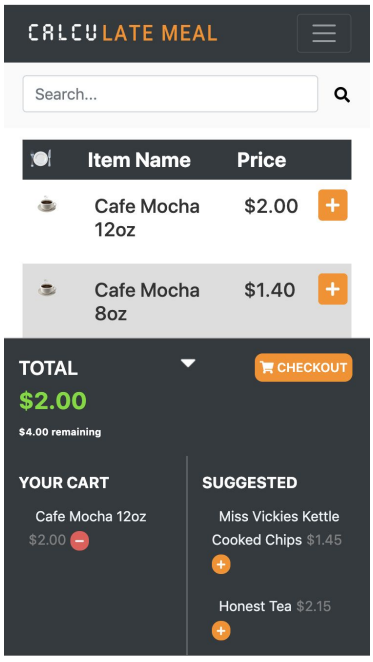
c. ***Item***: From the main screen, favorites page, and any results page (including the category search results), you can search for items in Late Meal by using the search bar at the top. This will generate a list of results which match the search query - either as a substring or as a keyword. This has the same sorting feature as search by category.




d. ***Popular***: From any page on CalcuLateMeal, you can access the "Popular" page in the menu on the top navigation bar. The "Popular" page features the top 5 items most frequently added to users' carts across the entire platform.

**e.** ***Sorting***: On the category search and item search pages, you can sort the items by item name or price by clicking the heading on the table that displays the search results.

2. Using the Cart:
   a. ***Expanding the Cart***: Users can click on any part of the cart along the bottom of the page to expand it and view all items in the cart, as well as suggested items to purchase within the budget.
   b. ***Collapsing the Cart***: Once the cart is expanded, users can click on any part along the cart to collapse it back to the original size, where it only shows the current total, surplus/deficit, and Checkout button.
   c. ***Adding to the Cart***: On any results page from categories or by search bar, clicking the plus sign next to the item name adds it to the cart.
   d. ***Removing from the Cart***: Expand the cart and click the minus sign next to any items which need to be removed.
   e. ***Viewing Current Total***: The numbers at the bottom left reflect the current total of the Late Meal purchase. Green numbers indicate that the meal is currently under the Late Meal Budget, and the small text underneath displays how much more the user can spend to stay within budget (the remaining amount). Red numbers indicate that the meal is currently over the Late Meal Budget, and the small text underneath displays how much the user must pay out of pocket.




   f. ***Using Suggested Items***: When the cart is expanded, there is a column (on the right) of suggested items which can be added to the cart. The suggested items include any item that falls within the balance remaining, ordered by popularity. When the budget is completely depleted or exceeded, there are no more suggested items. To

add a suggested item, simply click on the orange plus sign next to it. This column of items is scrollable.

***g.*** ***Checkout***: Pressing the "Checkout" button, displayed on the top right corner of the cart, will produce a pop-up message confirming that the user wishes to clear all items in the cart and return to the Splash Page.

3. Applying Late Meal Rules:
    a. ***Packaged Items Limit***: When two packaged items have already been added to the cart and the user tries to add a third packaged item, the web app will alert the user of this limit and ask the user to select a non-packaged item instead.
    b. ***Late Meal Combos*** *(combos and specials are used interchangeably)*
        i. *Build a Special*: Click on the "Specials" section on the map. Following the instructions on the page, choose a main and two sides, and click "Add Special" to have the special added to the cart and update the cart's total.
        ii. *Automatically Generate a Special*: When any specials main dish is added to the cart, an alert is automatically generated to let the user know that they can possibly create a late meal special. This is useful for users who may be unaware of the combo deals. Once the cart has all the items necessary to form a combo, a new alert is generated, informing the user that they can combine these items into a combo. If the user selects "OK," the items in the cart will be automatically bundled into a combo and the price will be adjusted to the combo price. The items in the combo can be entered in any order.

# DEVELOPER GUIDE

**Dependencies**

Our app relies on the following dependencies, which are recorded in the requirements.txt file.

- `gunicorn==19.9.0`
- `Flask==1.0.2`
- `psycopg2-binary==2.7.7`

**Installation**

Follow these instructions to deploy CalcuLateMeal on a local host.

1. Please install `Heroku CLI` and `Node.js` on your computer.
2. After opening a directory on the command line, clone the CalcuLateMeal repository using the following command:
   `git clone https://github.com/grace-hong/latemealcalc`
3. `cd` into the cloned repository and run `npm install` to install dependencies.
4. In the command line, run `python server.py` to host the site locally. Command line should print out a URL stating where the site is being hosted (ex. `Running on http://0.0.0.0:5000/`), and you can enter this URL into any browser to test out the site. Press `Ctrl + C` to stop the hosting at any time.

*Note: Since we have incorporated emojis into our application, you may receive an error that states that ASCII cannot convert the Unicode character. Please refer to the emojis and comment them out in order to resolve this issue.*

**Back-End**

CalcuLateMeal is a Python-based web application currently hosted on Heroku. Its database uses `PostgreSQL`, which is also hosted on Heroku (installation instructions can be found here).

- Database
  - Manually input data that we obtained from the Frist Gallery and cashiers into a `.csv` file used to populate the database.
  - Code to initialize the database is contained in `server.py` and currently commented out to keep it from restarting with every re-deploy.
- Framework
  - Routing across the site is done using the `Flask` framework. Reroutes are done when the user navigates to any new section of the site (i.e. one of the category pages or the favorites page).
  - We are also using the framework to facilitate the updating of the cart through redirects. Whenever an item is added to the cart or removed from the cart, the site is momentarily redirected to a blank page that updates the cart dictionary held in server.py with that session's items. It is then redirected back to the page it was previously on but now displaying the updated cart.
- Sessions Ids and Cart
  - As per the suggestion by our TA, Lance, we used the Flask Sessions object to keep track of user data across pages and maintain a curated, individualized feed

for each user. Upon visiting the site's splash page, each user is assigned a unique session ID with which they can associate their activity on the site. Sessions objects are stored on the client server, so each user's unique session ID is easily accessible by simply querying `session['uid']`. The cart, as well as a number of other individualized preferences (such as the time - lunch or dinner), is dependent on session IDs as keys. For example, the cart is stored as a dictionary in which the keys are session IDs and the values are a list of the current items in the cart; such an implementation was necessary, as we needed to maintain a distinct cart for each user on the site and ensure that this cart stayed live across pages for that specific user. Sessions are cleared once the user checks out.

- Adding/Removing Items
    - We have defined many different POST ends on the server that allow the user to add and remove items from their cart from anywhere on the site. Each page has two separate POSTs that are called to handle these operations before returning to the previous POST.
    - `server.py` contains a dictionary of the different session ids carts. When an item is added via a button click on the frontend, the server routes to an `/addItem*/<x>` type of endpoint where `<x>` is an identifier for the user's current path/location in the website and the item being added. Once within this routing, the server appends the specified item to the cart (list of items) associated with the appropriate session ID and redirects back to the original page. Similarly, when an item is removed via a button click, the server routes to a `removeItem*/<x>` type of endpoint. Once within this routing, the server removes the specified item from the list of items associated with the user's session ID in the cart dictionary. Then, the server redirects back to the original page to present a smooth transition.
- Search
    - Search by Time: The splash page displays two buttons for the user to choose from that set their Session ID time to either Lunch (`0`) or Dinner (`1`). This is stored in the back end and used to determine which items can be displayed when further searching on the site.
    - Search by Category: Whenever an area on the main map is clicked, the `getItemsFromCategory` POST is called, rendering that category's HTML file and sending the body code for the table of items contained in that category, the code for the current cart items, the current cart total, suggested items, and other HTML Markup code as command arguments.
    - Search by Item: When an item is typed into the search bar, we use `PostGreSQL` commands to retrieve all of the items in the database and find matches in the item name or keywords in the getItems POST. It renders `results.html` file and sends in command line arguments using Flask to complete the HTML of the page with the correct search items, cart contents, and alerts.
- Suggestions

    - Each item has a count of the number of times it's been added to the cart by all users stored in the database. `PostGreSQL` commands are used to retrieve a list of the top 10 most added items within the budget, sorted by most popular. This list is rendered within the POST that renders the template of the page the user is on and sent to the corresponding HTML file using Flask.
- Packaged Goods
    - There is a column in the `PostgreSQL` database titled "`packaged`" which has values of '`y`' or '`n`' indicating whether a late meal item is packaged. The dictionary `packaged` in `server.py` keeps a running total of the number of packaged goods currently in the cart across each session. In each function related to adding or removing an item, we first check the value of `packaged` for that session and update it according to whether to item trying to be added or removed is packaged. If there are two packaged items and a third is attempted to be added, we use an additional boolean variable `needAlert` to indicate that we must return an alert command, which is then displayed in the html files.
- Combos
    - Combos were encoded as a long boolean expression for multiple decision rules. The combos were especially challenging because they were registered when the user added an item within the `addItem*/<x>` endpoint in the server, yet none of such endpoints actually displayed any content from the frontend. As a result, we used this endpoint to check the cart to see if a combo existed (based on a series of boolean expressions) and then changed the value of an indicator for a combo dependingly. When the server immediately reroutes to the actual url for the page it is supposed to be display (rather than simply a backend request with no user interface), it checks the indicator and displays a confirm popup message accordingly.

**Front End**

The front end of CalcuLateMeal is a mixture of many different languages, mainly based in HTML/CSS and Bootstrap.

- Universal layout
    - The universal layout of the app was created using HTML, CSS, and Bootstrap libraries. The contents of the header included a combination of Bootstrap's `container-fluid` class, Bootstrap's responsive navigation bar, `navbar` and respective `nav-item` classes. These items were all fixed to the top of the page using basic CSS.
    - The search bar front end was implemented using the system of rows and columns given in Bootstrap.
- Search results
    - The table of items displayed by the search has its `<thead>` defined and initialized in each page's respective `.html` file. This contains the header of the table. The contents of `<tbody>` are created on the back end by storing the HTML code of rows for each item retrieved from the database in a string in the

page's POST in `server.py`. Each row contains an emoji cell, a name cell, a price cell, and a button cell. The `<button>` redirects the back end a POST that adds the item to the cart. This code is marked as safe using `Markup` and sent into the HTML file as an argument in `render_template` to be inserted into the table.

- Tablesorter.js
    - Search results are sortable by item name and item price. We imported a JQuery plugin `tablesorter.js` along with its `theme.jui.min.css` for formatting and widget features `tablesorter.widget.js` for additional features in the heading of the search results `.html` files. Documentation for this plugin by Mottie can be found here.
- Cart
    - The current cart is maintained as a 2x2 table. Vertical scroll is enabled by `overflow-y:scroll`. JQuery enables the cart to be animated by click (the cart will toggle up and down), and it also changes the direction of the small arrow attached onto the cart. Changing the color of the total and difference, based on whether the total is over/under the late meal budget, is written as HTML code in server.py and injected into the HTML.
    - Items are displayed by sending in data on the current cart items from the backend. The backend POSTs creates the HTML code for rows for each respective cart item and stores it in a string. Each row contains cells for the item name, item price, and a clickable `<button>` to remove it from the cart by rerouting the back end to that POST. The string is marked safe by `Markup` and then sent to the `.html` file as an argument in `render_template` to be inserted into the table.