DOCUMENTATION

JavaScript(app.js)

The code defines event listeners for opening and closing a checkout window. It also defines an array of products, each with an ID, name, image, and price.

The **addToCard** function is called when the user clicks on the "Add to Cart" button for a product. It checks whether the product is already in the cart, and if it is, it increments the quantity. If it is not, it adds the product to the cart with a quantity of 1. The **reloadCard** function is called to update the cart display.

The **showNotification** function is called when the user adds an item to the cart. It creates a notification div and appends it to the document body. After 500 milliseconds, the notification is removed from the body.

The **changeQuantity** function is called when the user clicks the "+" or "-" button next to a product in the cart. It updates the quantity and price of the product in the cart and calls the **reloadCard** function to update the cart display.

When the user clicks the "Confirm" button in the checkout window, the confirm event listener function is called. It creates an object **cartItems** containing the IDs and quantities of the products in the cart. It sends a POST request to the server with the cart data and waits for a response. If the request is successful, it displays a message alert with the response text. If there is an error, it logs the error to the console.

Finally, the **refresh** function is called to reload the page after the user has confirmed their cart to clear the cart.

CSS (style.css)

The first section sets the background color to a light gray and the font family to "SYSTEM-UI". The next section centers all h1 elements on the page. The .container class sets the width of the webpage to 1000 pixels and centers it on the page with a transition time of 0.5 seconds.

The header section is displayed using a grid with two columns. The first column contain a logo or website name while the second column contains a shopping cart icon with a red notification circle that indicates the number of items in the cart. The .list section sets up a grid with three columns and applies styling to each item in the grid such as background color, padding, and box shadow. Each item in the grid appears to be an item for sale with an image, title, price, and a button to add it to the cart.

The .card class is positioned as a fixed element on the page with a dark background color and a width of 500 pixels. It has a height of 100% of the viewport and is initially positioned off the screen to the left. The .active .card and .active .container classes are used to slide the .card element into view when the shopping cart icon is clicked.

The .card element contains an h1 element with yellow text and a checkOut section at the bottom with two buttons. The confirm button appears to proceed to checkout while the close shopping button appears to close the shopping cart. The .listCard section displays the items in the cart in a grid format with a small image, item name, price, and a button to adjust the quantity.

Finally, the notification class is positioned at the top center of the page and is initially hidden. It is used to display notifications to the user, and the show class is applied to it to display it when necessary using a CSS animation.

HTML(index.html)

The <html> element contains the entire web page, and the <head> element contains metadata and links to external resources such as stylesheets and scripts.

The page contains a container with a header that includes a logo and a shopping cart icon with a quantity label that starts at 0. Below the header is a list of menu items that are generated dynamically using a loop over a list of products. Each item has an "Add to Cart" button with an on click attribute that calls the **addToCard**() function with the index of the selected product.

At the bottom of the page, a card element appears when the "shopping" icon is clicked, which contains a list of selected items with their images, names, prices, and quantities. The card also displays the total price of all selected items, a "Close" button to hide the card, and a "Confirm" button that process the order.

The style.css file is linked to the HTML file to add styling to the page, and the app.js file is linked to provide functionality.

GUI(app.py)

The first code snippet is for creating a GUI using the tkinter library. It starts by importing the tkinter module and creating a tkinter window object. It then creates two frames, a left panel and a right panel, using the tkinter Frame widget.

The left panel contains a label and a button. The label is used to display a message to the user, while the button is used to call a function when clicked. The function in this case is a custom function called buttonClick, which simply displays a message box to the user using the showinfo function from the tkinter messagebox module.

The right panel contains three widgets. The first widget is a MenuButton called call_specific_robot. When clicked, it displays a menu containing five options, each labeled as "Robot 1", "Robot 2", and so on. Clicking on any of these options calls a corresponding function (e.g. r1, r2, etc.), which displays a message box indicating which robot is coming.

The second widget is a Button called call_any_robot. When clicked, it calls a custom function called robot_popup.

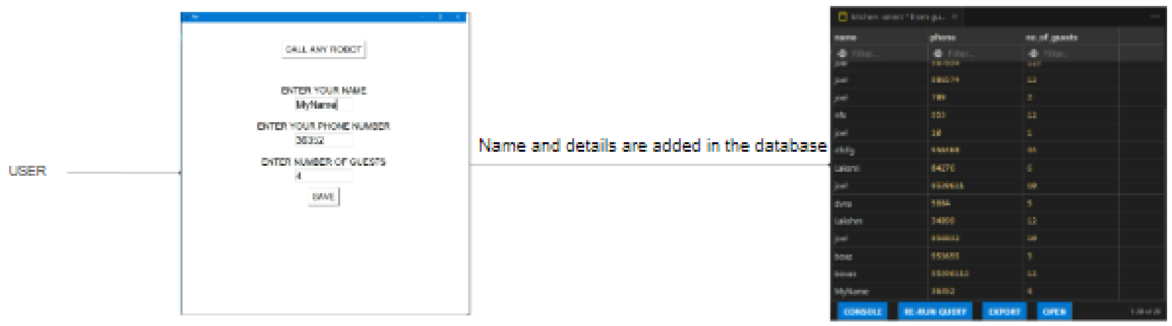
The third widget is a MenuButton called edit_button. When clicked, it displays a menu containing five options, each labeled as "Food 1", "Food 2", and so on. Clicking on any of these options calls a corresponding function (e.g. f1, f2, etc.), which opens a new window containing two input fields (for "Out of Stock" and "Price") and a "Confirm" button. When the "Confirm" button is clicked, the data entered in the input fields is used to update a database using an SQL query.

This function now takes two arguments, an SQL query and an ID, and uses these arguments to dynamically update the database.

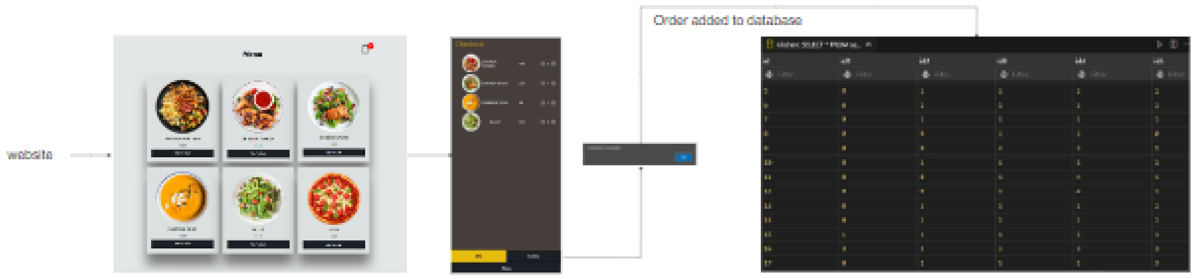
The function opens a new window using the Toplevel widget, creates two Entry widgets for the "Out of Stock" and "Price" input fields, and a "Confirm" button. When the "Confirm" button is clicked, the **updatedata** function is called, which retrieves the values entered in the input fields and updates the database using the SQL query provided as an argument. Finally, the updatedata function closes the window and displays a message box indicating that the data has been edited.

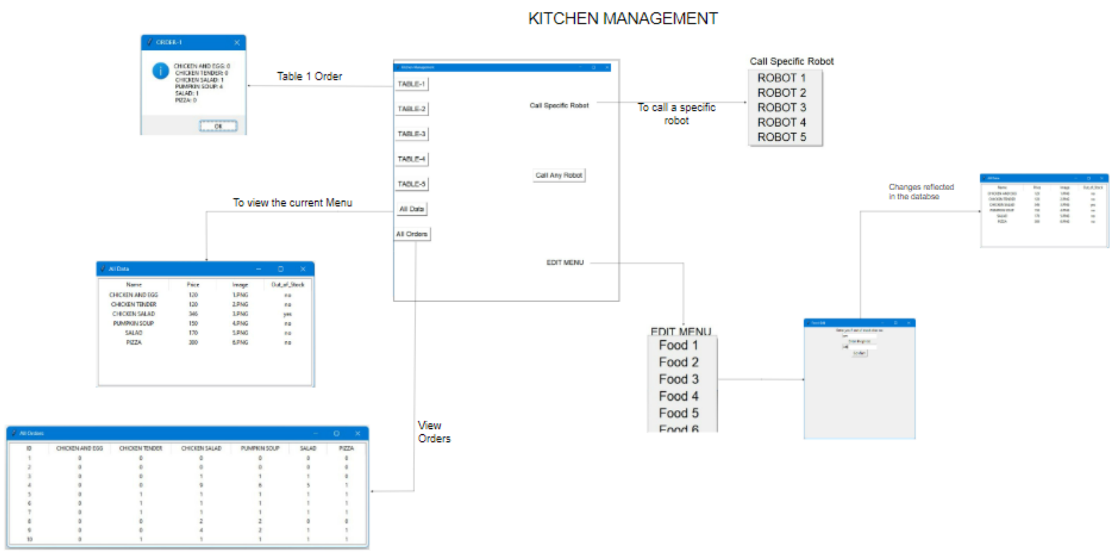
The f1, f2, etc. functions in the edit_button MenuButton in the first code snippet call the food function with a specific SQL query and ID based on the food item being edited (e.g.id=1 for "Food 1", id=2 for "Food 2", etc.). This allows the food function to update the correct row in the database based on the ID provided.

QUEUE MANAGEMENT



PLACING ORDER





ORDER-1

CHICKEN AND EGG: 0
CHICKEN TENDER: 0
CHICKEN SALAD: 1
PUMPKIN SOUP: 4
SALAD: 1
PIZZA: 0

All Data

Name	Price	Image	Out_of_Stock
CHICKEN AND EGG	120	1.PNG	no
CHICKEN TENDER	120	2.PNG	no
CHICKEN SALAD	346	3.PNG	yes
PUMPKIN SOUP	150	4.PNG	no
SALAD	170	5.PNG	no
PIZZA	300	6.PNG	no

All Orders

ID	CHICKEN AND EGG	CHICKEN TENDER	CHICKEN SALAD	PUMPKIN SOUP	SALAD	PIZZA
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	1	1	1	0
4	0	0	9	6	5	1
5	0	1	1	1	1	1
6	0	1	1	1	1	1
7	0	1	1	1	1	1
8	0	0	2	2	0	0
9	0	0	4	2	1	1
10	0	1	1	1	1	1

https://drive.google.com/file/d/1IaXgO_8yks-JSvBKaXxDJ3S32kQmV8C8/view?usp=sharin

Scripts.py

This Python code imports the mysql.connector module, which is used to connect to a MySQL database. It then establishes a connection to a MySQL database by passing the necessary parameters such as host, user, password, and database.

After that, it defines a function named robot_popup().

The next function, n1(id), takes an id as a parameter and uses it to select data from the robot table in the connected database. It first creates a cursor object, which is used to execute SQL queries. It then executes a SELECT statement that retrieves all columns and rows from the robot table where the id column matches the id parameter passed to the function. It then fetches all rows returned by the query using mycursor.fetchall() method, and iterates over each row to print out the data in a readable format.

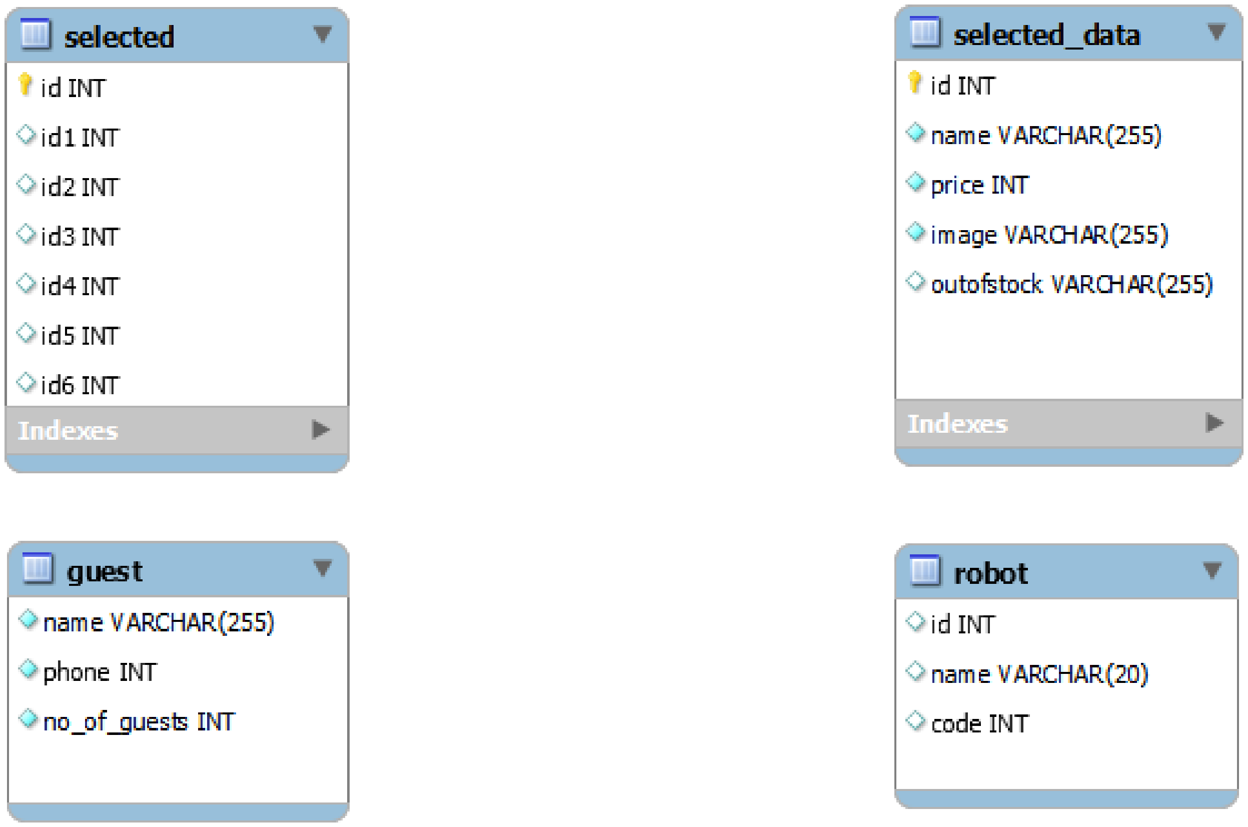
Finally, the n2() function selects all data from the guest table and prints it out in a similar format as n1(id) function.

REQUIREMENTS

(run this as requirement.txt to install necessary packages)

```
antiorm==1.2.1
asttokens==2.2.1
backcall==0.2.0
click==8.1.3
colorama==0.4.6
comm==0.1.3
comtypes==1.1.14
contourpy==1.0.7
customtkinter==5.1.2
cycler==0.11.0
darkdetect==0.8.0
db==0.1.1
db-sqlite3==0.0.1
decorator==5.1.1
distlib==0.3.6
executing==1.2.0
filelock==3.11.0
Flask==2.2.3
Flask-SQLAlchemy==3.0.3
fonttools==4.39.3
greenlet==2.0.2
ipykernel==6.22.0
ipython==8.12.0
itsdangerous==2.1.2
jedi==0.18.2
Jinja2==3.1.2
joblib==1.2.0
jsonify==0.5
kiwisolver==1.4.4
MarkupSafe==2.1.2
mysql-connector-python==8.0.33
traitlets==5.9.0
typing_extensions==4.5.0
wcwidth==0.2.6
Werkzeug==2.2.3
mysqlclient==2.1.1
nest-asyncio==1.5.6
packaging==23.1
pandas==2.0.0
parso==0.8.3
pickleshare==0.7.5
Pillow==9.5.0
platformdirs==3.2.0
prompt-toolkit==3.0.38
protobuf==3.20.3
psutil==5.9.5
pure-eval==0.2.2
Pygments==2.15.1
PyMySQL==1.0.3
pyparsing==3.0.9
pypiwin32==223
python-dateutil==2.8.2
pyttsx3==2.90
pytz==2023.3
pywin32==306
pyzmq==25.0.2
regex==2023.3.23
scipy==1.10.1
six==1.16.0
SQLAlchemy==2.0.9
stack-data==0.6.2
threadpoolctl==3.1.0
tk==0.1.0
tornado==6.3.1
boxapi==1.0
tqdm==4.65.
```

ENHANCED ENTITY RELATIONSHIP



selected
- id INT
- id1 INT
- id2 INT
- id3 INT
- id4 INT
- id5 INT
- id6 INT

Indexes

selected_data
- id INT
- name VARCHAR(255)
- price INT
- image VARCHAR(255)
- outofstock VARCHAR(255)

Indexes

guest
- name VARCHAR(255)
- phone INT
- no_of_guests INT

robot
- id INT
- name VARCHAR(20)
- code INT