Food Delivery Network Application

Contents

[Introduction 3](#_Toc165390974)

[1. Code Structure and Functionality 3](#_Toc165390975)

[Server(‘server.py’) 3](#_Toc165390976)

[Functionality: 3](#_Toc165390977)

[Structure: 3](#_Toc165390978)

[Connection Code: 3](#_Toc165390979)

[Client (client.py) 3](#_Toc165390980)

[Functionality: 3](#_Toc165390981)

[Structure: 4](#_Toc165390982)

[Connection Code: 4](#_Toc165390983)

[2. Data Structures 4](#_Toc165390984)

[3. Data Storage 4](#_Toc165390985)

[4. Running the Code 4](#_Toc165390986)

[1. Setting Up: 4](#_Toc165390987)

[2. Running the Server: 5](#_Toc165390988)

[3. Running the Client: 5](#_Toc165390989)

[5. Use Case: Adding a Menu Item (Owner) and Ordering Food (Customer) 5](#_Toc165390990)

[1. Owner Login: 5](#_Toc165390991)

[2. Updating a Menu Item: 6](#_Toc165390992)

[3. Customer Login: 7](#_Toc165390993)

[4. Ordering Food: 7](#_Toc165390994)

[5. Checking Out: 8](#_Toc165390995)

[6. Consideration of Error Cases 8](#_Toc165390996)

# Introduction

The Restaurant Management System is a client-server application designed to facilitate menu management, ordering, and checkout processes for a restaurant. It consists of two main components: server.py and client.py.

# Code Structure and Functionality

## Server(‘server.py’)

### Functionality:

* Handles client connections
* Manages menu updates and checkout operations.
* Performs user authentication

### Structure:

* load\_menu() and load\_users() functions load menu and user data from files.
* handle\_client\_connection(client\_socket) function manages client requests.
* Main function main() initializes the server socket and listens for client connections.

### Connection Code:

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    server\_socket.bind((SERVER\_HOST, SERVER\_PORT))

    server\_socket.listen(5)

    print(f"Server listening on {SERVER\_HOST}:{SERVER\_PORT}...")

while True:

        client\_socket, client\_address = server\_socket.accept()

        print(f"Connection from {client\_address} established.")

        handle\_client\_connection(client\_socket)

## Client (client.py)

### Functionality:

* Allows users to view menu, update menu (for owners), add items to cart, and checkout.
* Supports both admin and customer roles.

### Structure:

* Functions for user input validation and interaction.
* Menu management functions for admin users.
* Cart management and checkout functions for customer users.
* Main function main() handles client-side operations and interaction.

### Connection Code:

    client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    client\_socket.settimeout(10.0) # Set a timeout of 10 seconds

    try:

        client\_socket.connect((SERVER\_HOST, SERVER\_PORT))

        print("\033[H\033[J")

        print("Welcome to the Restaurant Management System")

        print("We are connected to server.")

# Data Structures

* **Dictionaries:** Used to store menu items with their details (item name, price, quantity) and user information (username, password, role).
* **Lists:** Used to store the menu and user data as a collection of dictionaries.
* **Strings:** Used to store usernames, passwords, and other text data.
* **Bytes:** Used to send and receive data over the network.

# Data Storage

The menu and user data are persistently stored on the server-side in two separate pickle files:

* **menu.pickle:** Stores the menu data as a list of dictionaries.
* **users.pickle:** Stores the user data as a list of dictionaries.

# Running the Code

## Setting Up:

* Save the provided code (server.py, client.py, and setup.py) in the same directory.
* Run the setup.py script to create the initial menu (menu.pickle) and user (users.pickle) data files. A black background with white text

  Description automatically generated

## Running the Server:

* Open a terminal in the directory containing the code files.
* Run the command: python server.py
* This will start the server listening for client connections.



## Running the Client:

* Open another terminal in the same directory.
* Run the command: python client.py
* This will start the client program, which will attempt to connect to the server running on localhost (127.0.0.1) by default. A black screen with white text

  Description automatically generated

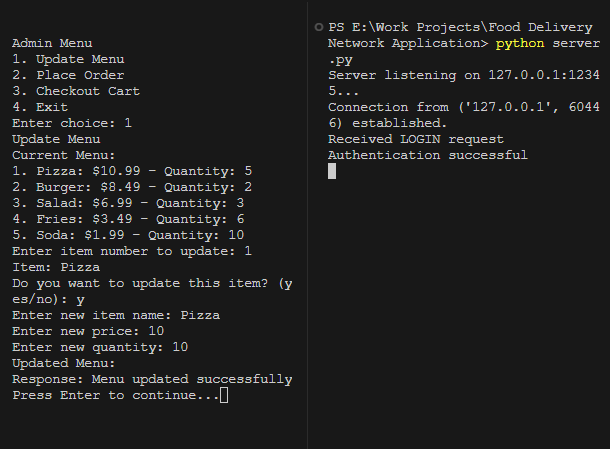
# Use Case: Adding a Menu Item (Owner) and Ordering Food (Customer)

## Owner Login:

* Start the server (python server.py) in one terminal.
* Start the client (python client.py) in another terminal.
* Enter the username and password for an owner account (e.g., username: admin, password: admin).
* The client should display the owner menu.
* A screenshot of a computer

  Description automatically generated

## Updating a Menu Item:

* Select option 1 ("Update Menu") from the owner menu.
* The current menu will be displayed.
* Enter the menu item number (e.g., item number:1, item name: Pizza, price: 7.99, quantity: 10).
* The client will send the updated menu to the server.
* The server will update the menu data and send a confirmation message back to the client.
* 

## Customer Login:

* Log out of the owner account (if applicable).
* Enter the username and password for a customer account (e.g., username: user1, password: pass1).
* The client should display the customer menu.
* A computer screen shot of a computer

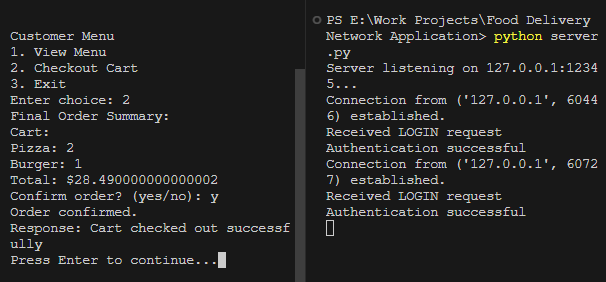
  Description automatically generated

## Ordering Food:

* Select option 1 ("View Menu") from the customer menu.
* The menu will be displayed.
* Select the item you want to order (e.g., item number 1: Pizza) and enter the desired quantity (e.g., quantity: 2).
* The item will be added to the cart.
* You can repeat this step to add more items to the cart.
* A screenshot of a computer screen

  Description automatically generated

## Checking Out:

* Select option 2 ("Checkout Cart") from the customer menu.
* The cart will be displayed with the total price.
* If the order is confirmed, the client will send the cart data to the server.
* The server will deduct the ordered quantity of each item from the available stock in the menu data.
* If there is insufficient quantity for any item, the server will send an error message to the client, and the checkout process will be aborted.
* If the checkout is successful, the server will update the menu data and send a confirmation message to the client.
* 
* A screenshot of a menu

  Description automatically generated

# Consideration of Error Cases

**Socket** **Timeout:**

Handled with a timeout mechanism to prevent indefinite waiting.

**Invalid Input:**

Various input validation functions ensure proper handling of invalid user inputs.

A screenshot of a menu

Description automatically generated

**Insufficient Quantity:**

Cart checkout fails if the requested quantity exceeds the available quantity.

A screenshot of a computer

Description automatically generated