# Distributed Systems Assignment

## Design Document

Ronan Singpurwala

C20391216

**Lecturer:** Ms. E. Hatunic-Webster

School of Computer Sciencee
Technological University Dublin

October 31, 2023

# Declaration

I declare that this work, which is submitted as part of my coursework, is entirely my own, except where clearly and explicitly stated.

[illegible]

Ronan Singpurwala

October 31, 2023

# Table of Contents

# 1 Design Document

## 1.1 Buyer-Seller System

This Python program simulates an electronic food marketplace with buyers and sellers. It allows multiple buyers to connect to a seller and perform various actions including: Buying items, listing items, joining the marketplace, leaving the marketplace and quitting (once a buyer quits, they cannot rejoin the marketplace, as the socket gets disconnected)

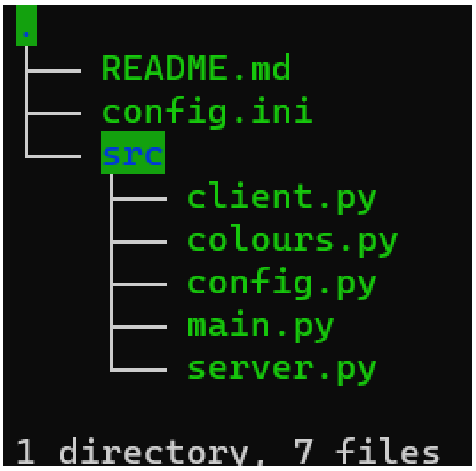
## 1.2 Application Architecture

The system comprises of a server (seller) and multiple clients (buyers). The server is responsible for listening to incoming connections, managing items and sales, and notifying all clients when necessary. The server and client use TCP sockets for communication. TCP is chosen for its reliability, in-order delivery, and error-checking capabilities, which are crucial for ensuring that messages are not lost and are received. The clients connect with the server, and a separate thread is created for the client so multiple clients can communicate concurrently with the server. When all clients need to be notified, the server can broadcast the message to all the clients as the server keeps a list of connected clients.

Configuration settings are stored in a configurations file. The configuration file defines settings for the sellers and buyers. It includes the server's host and port configuration, as well as the number buyers with their respective IDs. When you run the program, client and server instances are launched based of this configurations file.

## 1.3 Code Directory Structure

The code directory structure shown in figure 1.1, is well organised for maintainability. It includes the following components:



```
.
├── README.md
├── config.ini
└── src
    ├── client.py
    ├── colours.py
    ├── config.py
    ├── main.py
    └── server.py

1 directory, 7 files
```

Figure 1.1: Code Directory Structure

- **README.md**: This file contains documentation and instructions on how to run the program.
- **config.ini**: This file contains configurations for launching server and client instances.
- **src/**: The `src` directory contains the source code for the project. This includes:
  - **client.py**: Code related to the buyer.
  - **colours.py**: This file defines ANSI color codes used for text styling.
  - **config.py**: Contains code for reading and managing configurations.
  - **main.py**: This file is used to launch client and server instances based of the `config.ini` file.
  - **server.py**: Contains the code relevant to the seller.

## 1.4 System Setup and Running

To set up and run the system, follow the steps below.

### 1.4.1 Prerequisites

The program is developed using Python 3.10.0, and it's recommended to use this version for compatibility. However, it would most likely work in other versions.

The system is designed for Windows OS, and modifications may be needed for other platforms.

The program only uses libraries from Python's standard library and does not require external libraries.

### 1.4.2 Configurations Setup

Modify the `config.in` file to configure the system. You can specify the server host, port, and IDs buyers.

For the demo, the `config.ini` file would look like this:

```
[Server]
host = localhost
port = 5000

[Buyers]
buyer1 = 1
buyer2 = 2
buyer3 = 3
buyer4 = 4
```

### 1.4.3 Running the System

1. Open a terminal and navigate to the project root directory.

2. Execute the following command to run the system:

```
python src/main.py
```

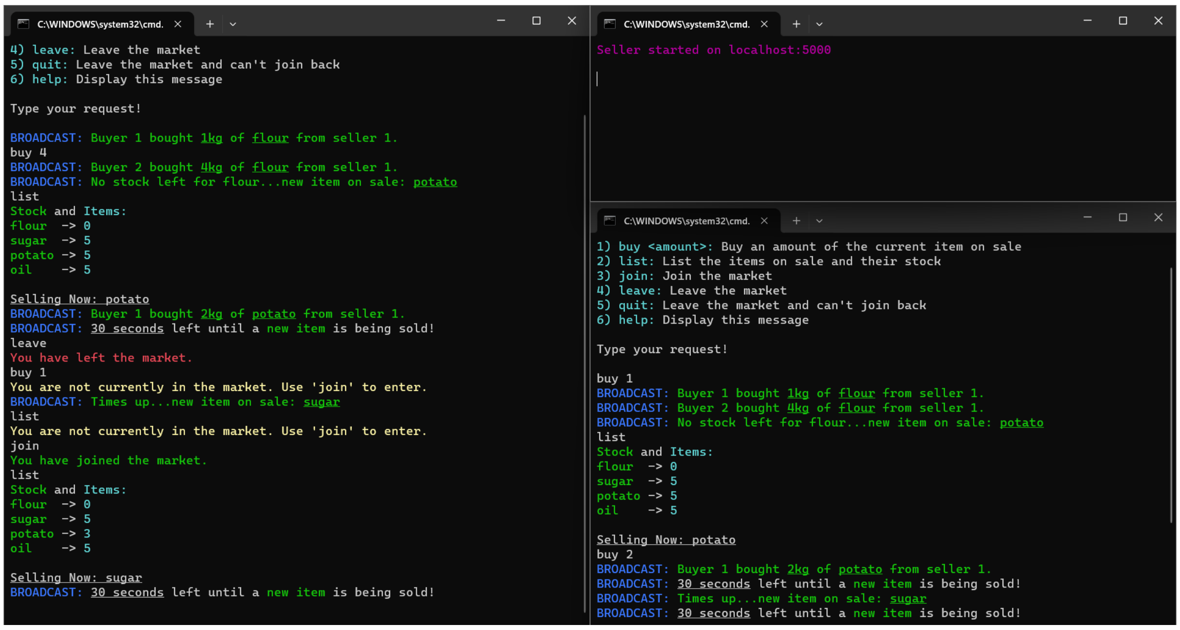
After running the `main.py`, separate console windows will appear for both the seller and each individual buyer. You can engage with each buyer individually using the CLI specific to that buyer.

It is recommended to use a console that supports ANSI colour codes for text styling.

If there's issues with using `main.py` for launching the instances, then you can run the them manually in separate terminals:

```
python src/server.py localhost 5000 1
python src/client.py localhost 5000 1
python src/client.py localhost 5000 2
python src/client.py localhost 5000 3
python src/client.py localhost 5000 4
```

## 1.5 Screenshots of the Application



```
4) leave: Leave the market
5) quit: Leave the market and can't join back
6) help: Display this message

Type your request!

BROADCAST: Buyer 1 bought 1kg of flour from seller 1.
buy 4
BROADCAST: Buyer 2 bought 4kg of flour from seller 1.
BROADCAST: No stock left for flour...new item on sale: potato
list
Stock and Items:
flour  -> 0
sugar  -> 5
potato -> 5
oil    -> 5

Selling Now: potato
BROADCAST: Buyer 1 bought 2kg of potato from seller 1.
BROADCAST: 30 seconds left until a new item is being sold!
leave
You have left the market.
buy 1
You are not currently in the market. Use 'join' to enter.
BROADCAST: Times up...new item on sale: sugar
list
You are not currently in the market. Use 'join' to enter.
join
You have joined the market.
list
Stock and Items:
flour  -> 0
sugar  -> 5
potato -> 3
oil    -> 5

Selling Now: sugar
BROADCAST: 30 seconds left until a new item is being sold!
```

```
Seller started on localhost:5000
```

```
1) buy <amount>: Buy an amount of the current item on sale
2) list: List the items on sale and their stock
3) join: Join the market
4) leave: Leave the market
5) quit: Leave the market and can't join back
6) help: Display this message

Type your request!

buy 1
BROADCAST: Buyer 1 bought 1kg of flour from seller 1.
BROADCAST: Buyer 2 bought 4kg of flour from seller 1.
BROADCAST: No stock left for flour...new item on sale: potato
list
Stock and Items:
flour  -> 0
sugar  -> 5
potato -> 5
oil    -> 5

Selling Now: potato
buy 2
BROADCAST: Buyer 1 bought 2kg of potato from seller 1.
BROADCAST: 30 seconds left until a new item is being sold!
BROADCAST: Times up...new item on sale: sugar
BROADCAST: 30 seconds left until a new item is being sold!
```

Figure 1.2: Screenshot of the Program running with 2 buyers and 1 seller

Figure 1.2 demonstrates the program running. The buyer is able to join and leave the market as well as buy and list items. The seller is then able to notify all buyers that an item has been sold to a particular buyer.

## 1.6 Alternative Design Solution

If there were no implementation restrictions, I would have considered a alternative design solution to the current one.

### 1.6.1 Alternative Communication Protocols

Instead of using TCP sockets, the system could have been designed with other communication protocols, such as WebSockets. WebSockets would have been useful for real-time interaction between buyers and the seller. WebSockets are well-suited for handling asynchronous events and it would've been a suitable choice for notifying all buyers about a new item on sale in real-time.

### 1.6.2 Database Integration

The system could have incorporated a database which would allow for the storage of historical sales data, user profiles, and item information. Databases like PostgreSQL, or NoSQL databases like MongoDB could be considered.

### 1.6.3 Message Queue Systems

For handling asynchronous communication, a message queue system like RabbitMQ could be used. This would allow for decoupling of components and enable more flexible communication between buyers and sellers.

### 1.6.4 Containerisation and Orchestration

Using containerisation tools like Docker and orchestration platforms like Kubernetes could make the system more portable, scalable, and easy to manage. The would allow someone to deploy the project locally or on the cloud.