# SYSTEM DESIGN DOCUMENT

# for

# Trading Application

**COMPSCI 677 Lab 2**

**Prepared by: Mudit Chaudhary (32607978)**
**Girish Baviskar (33976648)**

# Contents

# 1 Introduction

## 1.1 Purpose

This document specifies the system design for the microservices-based trading application.

## 1.2 Intended Audience and Reading Suggestions

This document is intended for developers, testers, documentation writers, and the grading staff for the project. This document does not specify the technical details, evaluation results, and user manual.

## 1.3 Tech Stack

We use the following tech stack for the trading application:

- Python
- Docker

# 2 System Design

In our system, the microservices communicate using HTTP requests using REST-based architecture. We use HTTPServer python library along with ThreadMixin to have thread-per-request and thread-per-session server.

## 2.1 Frontend Service

The Frontend Service communicates with the catalog service and order service using thread-per-request. It communicates with the client in a thread-per-session manner. The Frontend also deals with determining the validity of requests e.g., request path, type, content type, etc.

## 2.2 Catalog Service

The Catalog Service provides the functionality to lookup and update the stock database. However, it can only update a particular stock's quantity and trading volume and does not perform additional checks if an update is valid as that responsibility is handled by the order service. In our implementation we use a thread-per-request model for lookup requests. However, it provides the functionality of thread-per-session for order requests, which is utilized by the order service to perform lookup and update with a single HTTP connection, thus reducing latency due to re-connection.

### 2.2.1 Database

We provide a simple persistent database using Pysos. We make it thread safe by using reader-writer locks. The reader locks can be accessed by multiple requests simultaneously but writer locks blocks the resources from concurrent threads except the one that has acquired the writer lock. The reader-writer locks improve the throughput of our application.

## 2.3 Order Service

Order Service provides the functionality of determining the validity of an order and carrying out the trade by communicating with the Catalog Service. It first performs lookup of the stock through the catalog service and then determines the validity of the order. Once the validity is determined, it send the update request to the catalog service to increase/decrease the stock quantity based on the trade type.

### 2.3.1 TransactionHandler

TransactionHandler performs thread-safe transactions. The transactions/order are performed in a thread-safe manner i.e., the process of lookup and update are under a lock. Only one thread can lookup and update at a time. This avoids any inconsistency in the catalog database due to multiple threads. To improve the throughput, the lookup and update requests are carried using a single TCP connection.

# 3 API specifications

## 3.1 Frontend Service

### 3.1.1 Lookup request

- **Method Access:** HTTP
- **URL:** GET http://hostname:port/stocks/stockname
- **Response Codes:** 200 (Success), 400 (Failure)
- **Sample Success:** {"data":{"name":"GameStart","price": 15.99,"quantity":100}}
- **Sample Failure:** {"error":{"code": 404,"message":"stock not found"}}

### 3.1.2 Order request

- **Method Access:** HTTP
- **URL:** POST http://hostname:port/orders
- **Response Codes:** 200 (Success), 400 (Failure)
- **Content-Type**: application/json
- **Sample Request Body**: {"name":"GameStart","type":"sell","quantity":100}
- **Sample Success:** {"data":{"transaction_number":10}}
- **Sample Failure:** {"error":{"code": 404,"message":"transaction unsuccessful"}}

## 3.2 Catalog Service

### 3.2.1 Lookup request

- **Method Access:** HTTP
- **URL:** GET http://hostname:port/lookup/stockname
- **Response Codes:** 200 (Success), 400 (Failure)
- **Sample Success:** {"data":{"name":"GameStart","price": 15.99,"quantity":100}}
- **Sample Failure:** {"error":{"code": 404,"message":"stock not found"}}

### 3.2.2 Update request

- **Method Access:** HTTP
- **URL:** POST http://hostname:port/update
- **Response Codes:** 200 (Success), 400 (Failure)
- **Content-Type**: application/json
- **Sample Request Body**: {"name":"GameStart","quantity":-100}

## 3.3 Order Service

### 3.3.1 Order request

- **Method Access:** HTTP
- **URL:** POST http://hostname:port/order
- **Response Codes:** 200 (Success), 400 (Failure)
- **Content-Type**: application/json
- **Sample Request Body**: {"name":"GameStart","type":"sell","quantity":100}
- **Sample Success:** {"data":{"transaction_number":10}}
- **Sample Failure:** {"error":{"code": 404,"message":"transaction unsuccessful"}}