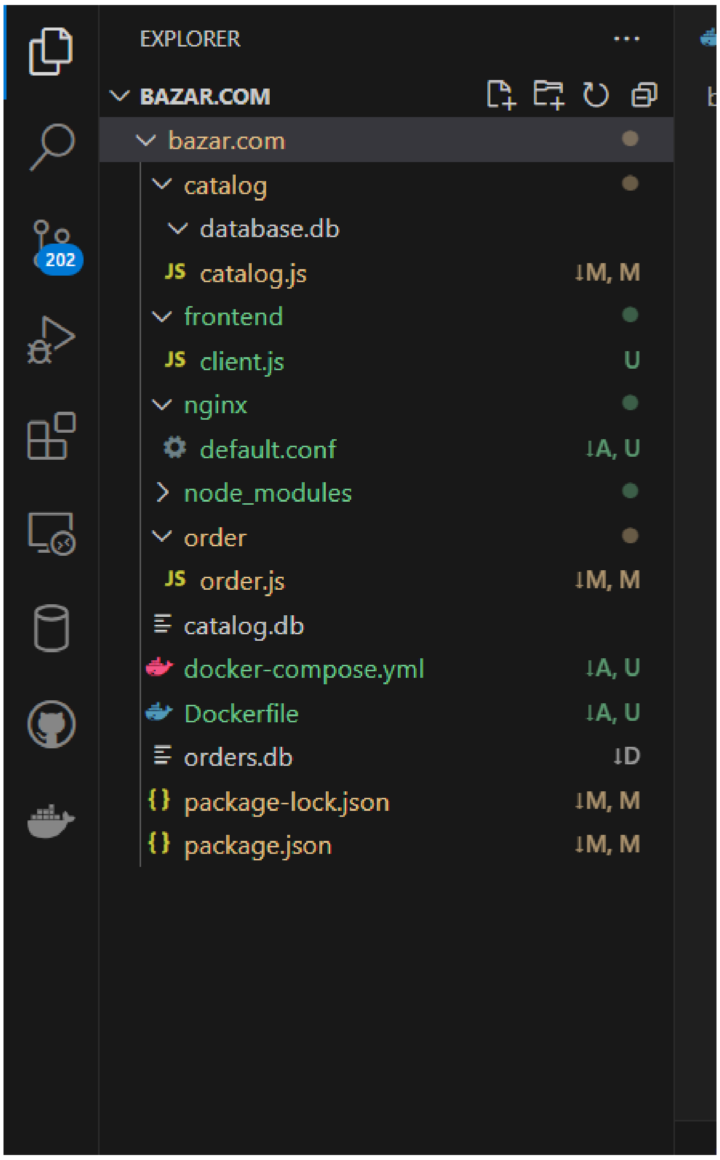
DOS Project Part #1 Report

Student Name: Lama Qawareeq.

Student Name: Anwar Al Aqraa.

In the first, this our project Hierarchy:



```
EXPLORER                                  ...
∨ BAZAR.COM
  ∨ bazar.com                              ●
    ∨ catalog                              ●
      ∨ database.db
      JS catalog.js                     ↓M, M
    ∨ frontend                             ●
      JS client.js                         U
    ∨ nginx                                ●
      default.conf                      ↓A, U
    > node_modules                         ●
    ∨ order                                ●
      JS order.js                       ↓M, M
    catalog.db
    docker-compose.yml                  ↓A, U
    Dockerfile                          ↓A, U
    orders.db                             ↓D
    {} package-lock.json                ↓M, M
    {} package.json                     ↓M, M
```

Introduction:

This project aims to design and implement an integrated system consisting of three microservices: a book catalog service, an order management service, and a front-end service. The user interacts with the system through the front end, which provides three main operations: searching for books, obtaining information about a specific book, and purchasing books. These operations are implemented via a RESTful interface so that communication between different services takes place using HTTP requests.

System architecture:

The system was divided into three basic components:

-Catalog service: maintains information about available books including subject, price, and number of copies in stock. This service supports two operations:

Query: Query by subject or item number to get information about books.

.Update: Modify the price of books or the number of copies in stock.

-Orders service: It deals with requests to purchase books, as it ensures the availability of **-** the book in stock by communicating with the catalog service, and then reduces the number .of copies available in stock if the purchase process is successful

-Front-end service: Allows users to search for books (by subject), obtain information for a **-** specific book (by item number), or purchase a book (by item number).

.

Technology used: .

1-Node.js :-

was used to build the three microservices due to its ability to build lightweight and asynchronous services.

Via command: `npm install .`

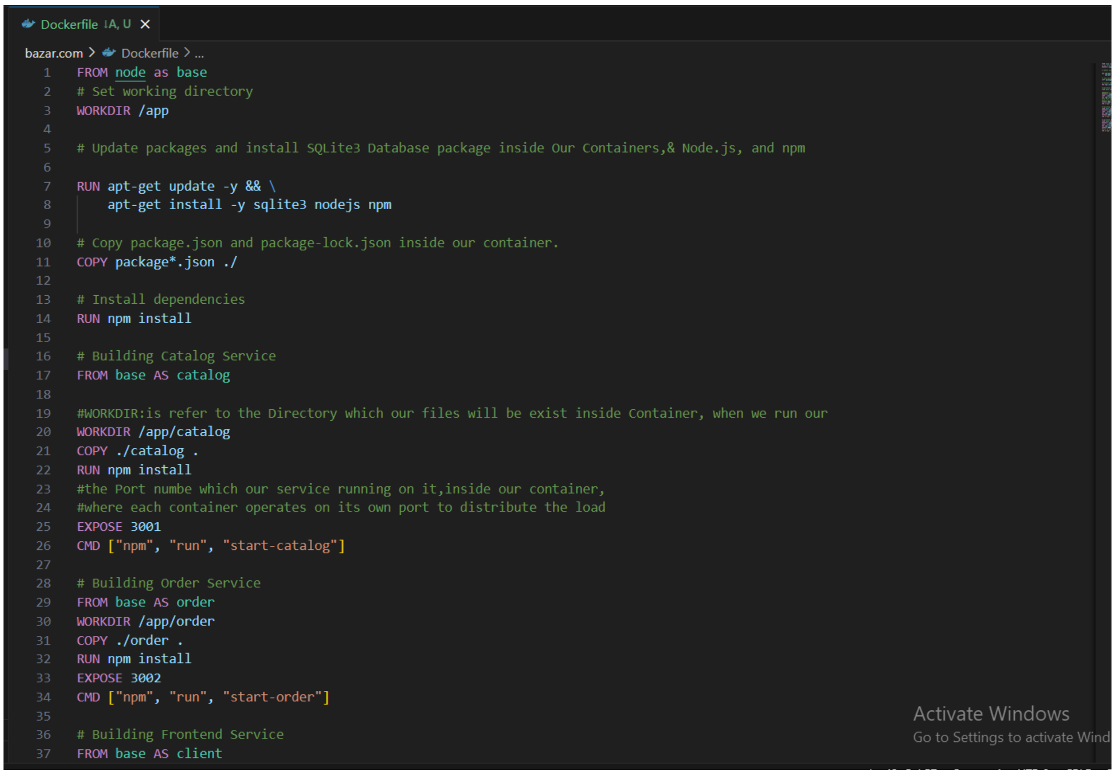
2-Docker:

It was used to run each service inside a separate container to ensure concurrent operability and facilitate project deployment.

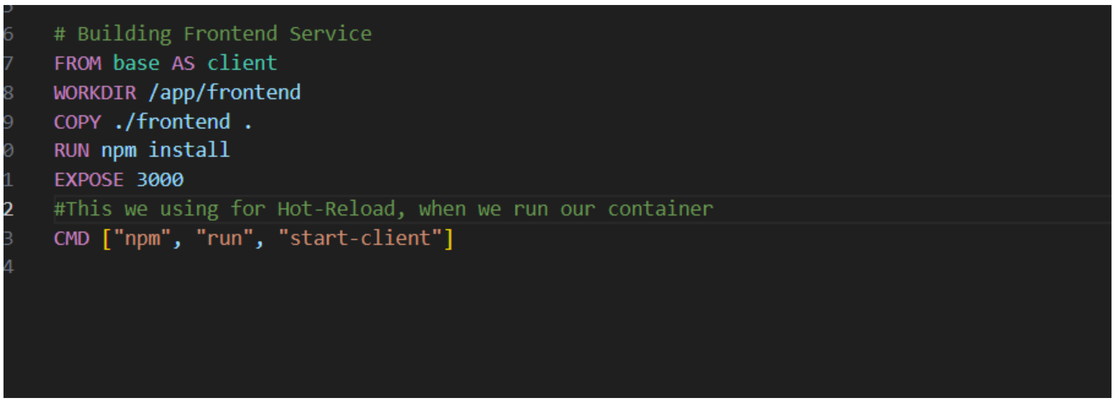
Where we created a Dockerfile to create our containers.

Note: We Create Our 3 Containers inside 1 file as stage, the first stage for catalog Container the second for Order Container, third for Client Containe.

we will attach the code to this file along with the necessary commands inside the file to explain the code.



```dockerfile
FROM node as base
# Set working directory
WORKDIR /app

# Update packages and install SQLite3 Database package inside Our Containers,& Node.js, and npm

RUN apt-get update -y && \
    apt-get install -y sqlite3 nodejs npm

# Copy package.json and package-lock.json inside our container.
COPY package*.json ./

# Install dependencies
RUN npm install

# Building Catalog Service
FROM base AS catalog

#WORKDIR:is refer to the Directory which our files will be exist inside Container, when we run our
WORKDIR /app/catalog
COPY ./catalog .
RUN npm install
#the Port numbe which our service running on it,inside our container,
#where each container operates on its own port to distribute the load
EXPOSE 3001
CMD ["npm", "run", "start-catalog"]

# Building Order Service
FROM base AS order
WORKDIR /app/order
COPY ./order .
RUN npm install
EXPOSE 3002
CMD ["npm", "run", "start-order"]

# Building Frontend Service
FROM base AS client
```



```dockerfile
# Building Frontend Service
FROM base AS client
WORKDIR /app/frontend
COPY ./frontend .
RUN npm install
EXPOSE 3000
#This we using for Hot-Reload, when we run our container
CMD ["npm", "run", "start-client"]
```

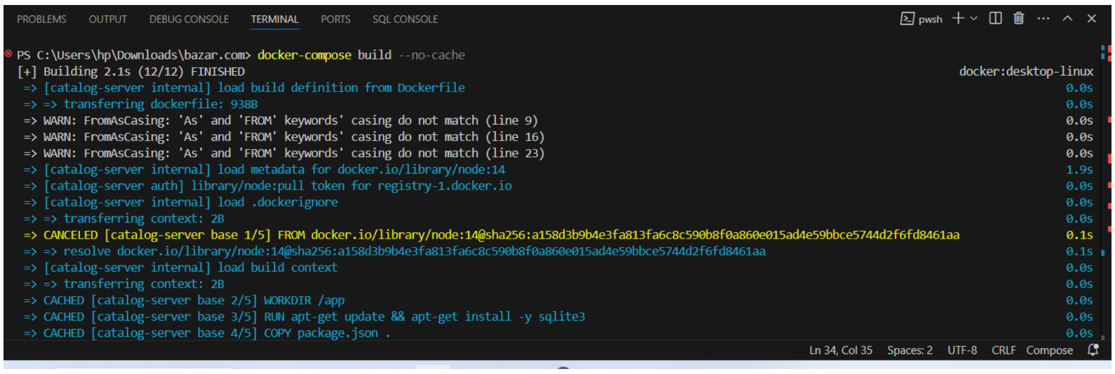
3-We created Docker Compose.yml:: It was used to compile the three services (catalog, .requests, and front-end) as it is a tool used to define and run multi-container applications

The three containers are set up to be independent of each other but communicate with each other over a shared network, making development and testing easier.

We have used

docker-compose build --no-cache

Let's build the images for the containers defined in the docker-compose.yml file.



```
PS C:\Users\hp\Downloads\bazar.com> docker-compose build --no-cache
[+] Building 2.1s (12/12) FINISHED                                                              docker:desktop-linux
 => [catalog-server internal] load build definition from Dockerfile                                             0.0s
 => => transferring dockerfile: 938B                                                                            0.0s
 => WARN: FromAsCasing: 'As' and 'FROM' keywords' casing do not match (line 9)                                  0.0s
 => WARN: FromAsCasing: 'As' and 'FROM' keywords' casing do not match (line 16)                                 0.0s
 => WARN: FromAsCasing: 'As' and 'FROM' keywords' casing do not match (line 23)                                 0.0s
 => [catalog-server internal] load metadata for docker.io/library/node:14                                       1.9s
 => [catalog-server auth] library/node:pull token for registry-1.docker.io                                      0.0s
 => [catalog-server internal] load .dockerignore                                                                0.0s
 => => transferring context: 2B                                                                                 0.0s
 => CANCELED [catalog-server base 1/5] FROM docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa  0.1s
 => => resolve docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa  0.1s
 => [catalog-server internal] load build context                                                                0.0s
 => => transferring context: 2B                                                                                 0.0s
 => CACHED [catalog-server base 2/5] WORKDIR /app                                                               0.0s
 => CACHED [catalog-server base 3/5] RUN apt-get update && apt-get install -y sqlite3                           0.0s
 => CACHED [catalog-server base 4/5] COPY package.json .                                                        0.0s
```

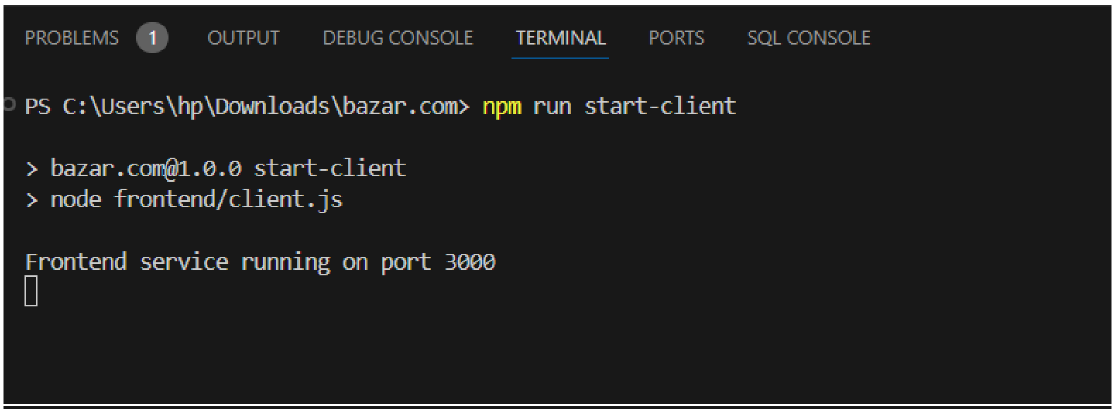
We ran the containers defined in the docker-compose.yml file by:

docker-compose up

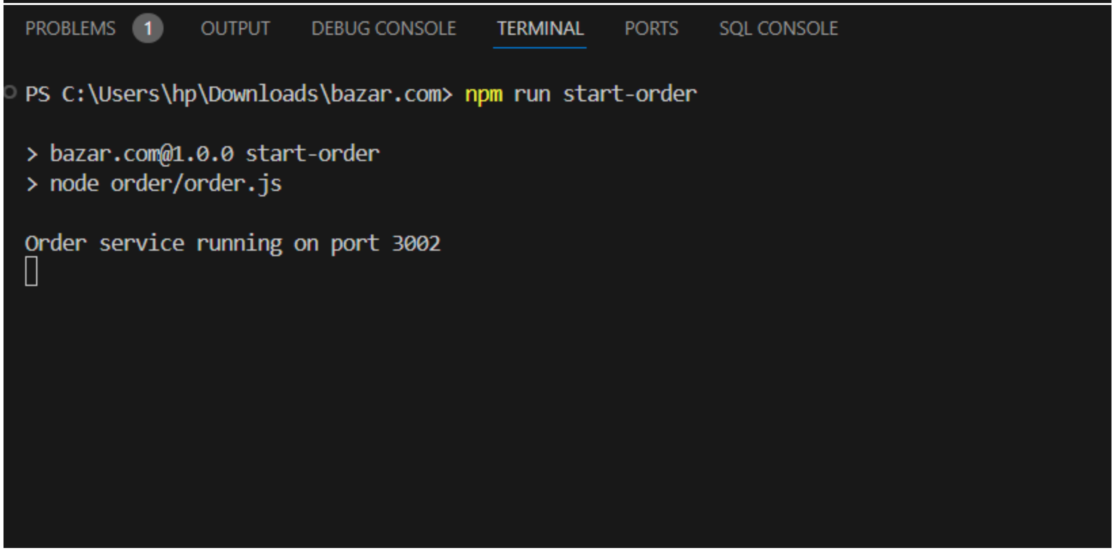
4- NGINX: was used as a server to distribute loads to services and ensure efficient - distribution of requests: a configuration file was prepared

default.conf to direct each type of request to the corresponding service. This ensures that no service is overloaded and improves overall system performance.

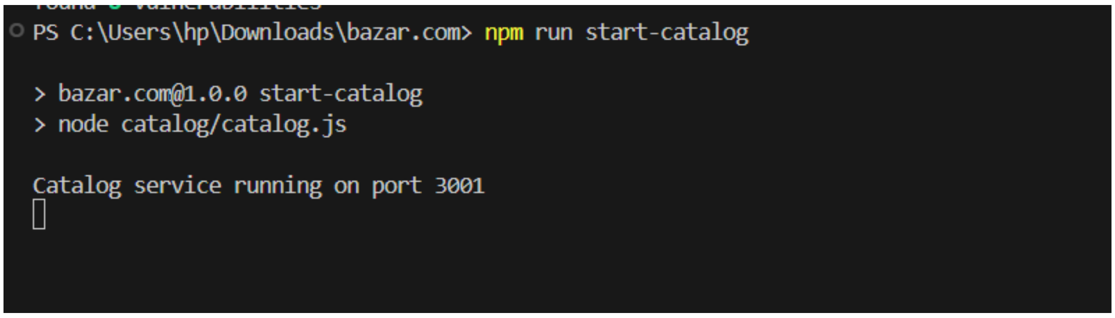
Each container (server) has a port allocated to it to direct requests to it.



```
PS C:\Users\hp\Downloads\bazar.com> npm run start-client

> bazar.com@1.0.0 start-client
> node frontend/client.js

Frontend service running on port 3000
```



```
PS C:\Users\hp\Downloads\bazar.com> npm run start-order

> bazar.com@1.0.0 start-order
> node order/order.js

Order service running on port 3002
```



```
PS C:\Users\hp\Downloads\bazar.com> npm run start-catalog

> bazar.com@1.0.0 start-catalog
> node catalog/catalog.js

Catalog service running on port 3001
```

-

5 -In the package.json file, nodemon was used :to facilitate the development process, as it allows services to be restarted automatically when any changes are detected in the code, which saves the developer time and effort while testing the codes

Future improvements: .

<u>The project can be improved by</u>:

-Use more complex databases like MySQL or MongoDB to handle larger data.

-Adding a caching system to speed up the system's response.