Cloud Computing Mid-Term Project Report

Zhafranafis Khayruraya

1101224328

Case : Retail Management System (Toko Swalayan)

Platform : Docker

Database : SQL

Project Description

This project demonstrates the implementation of a microservices-based web application using Node.js (Express) as the backend framework and MySQL as the database, both deployed inside Docker containers.

The application simulates a simple Retail Management System (Toko Swalayan), where users can perform CRUD (Create, Read, Update, Delete) operations on product data through a REST API. Each product record includes the attributes ID, Name, Price, and Stock.

The system runs on port 3000, which serves as the main access point for all API endpoints, such as:

http://localhost:3000/api/products

All API testing was performed entirely through cURL commands in the terminal, without using Postman. These commands send HTTP requests directly to the backend container and receive JSON responses for verification.

The main objectives are:

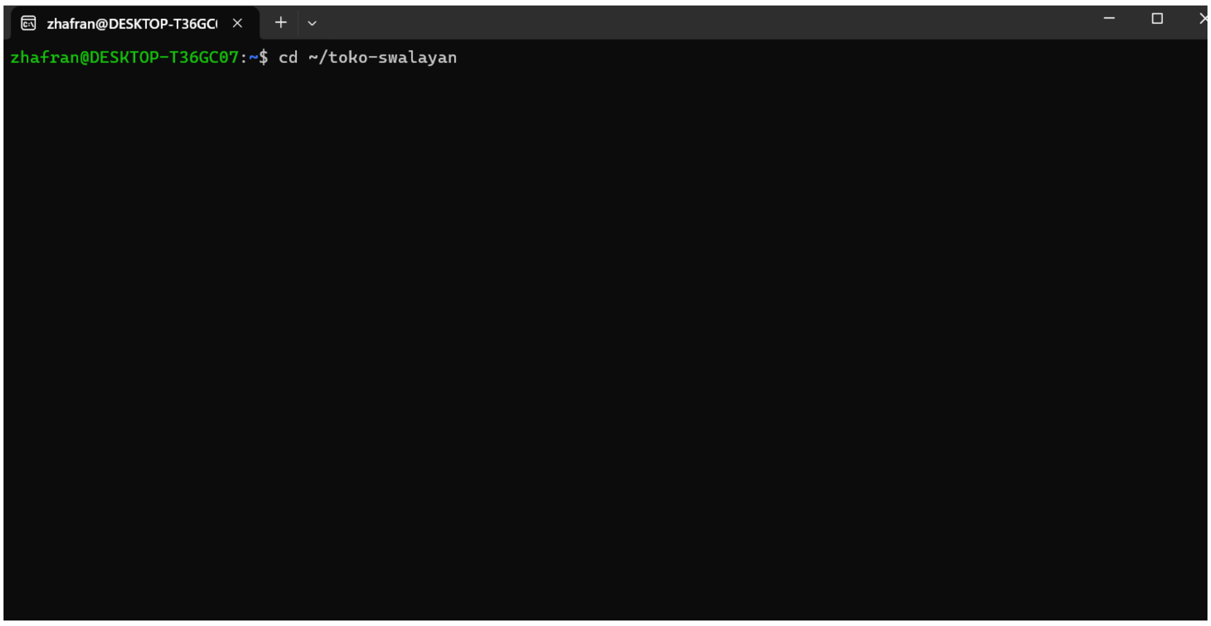
- Containerized deployment of backend and database.
- Functional REST API communication through port 3000.
- Internal Docker networking between backend and database containers.
- Validation of CRUD functionality using cURL-based API testing.

Frameworks and Tools Used:

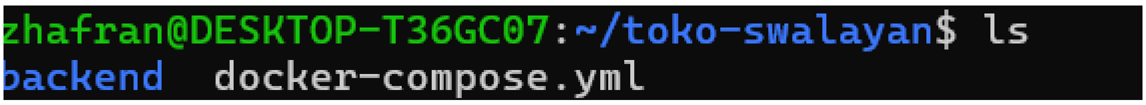
- Node.js (Express)
- MySQL (SQL-based database)
- Docker & Docker Compose (containerization and orchestration)
- cURL (command-line API testing)
- WSL (Ubuntu) as the execution environment for Docker commands on Windows

1. **Opening WSL (Ubuntu):**
   Used as the working terminal to execute Docker and Node.js commands. This environment provides Linux compatibility for Docker Desktop on Windows.

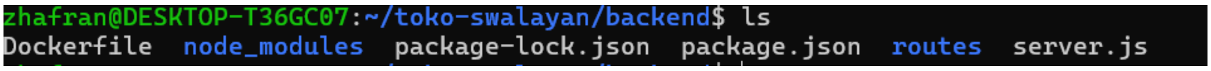
2. **Project Setup:**
   Created the project folder toko-swalayan and initialized Docker configuration files.



```
zhafran@DESKTOP-T36GC07:~$ cd ~/toko-swalayan
```

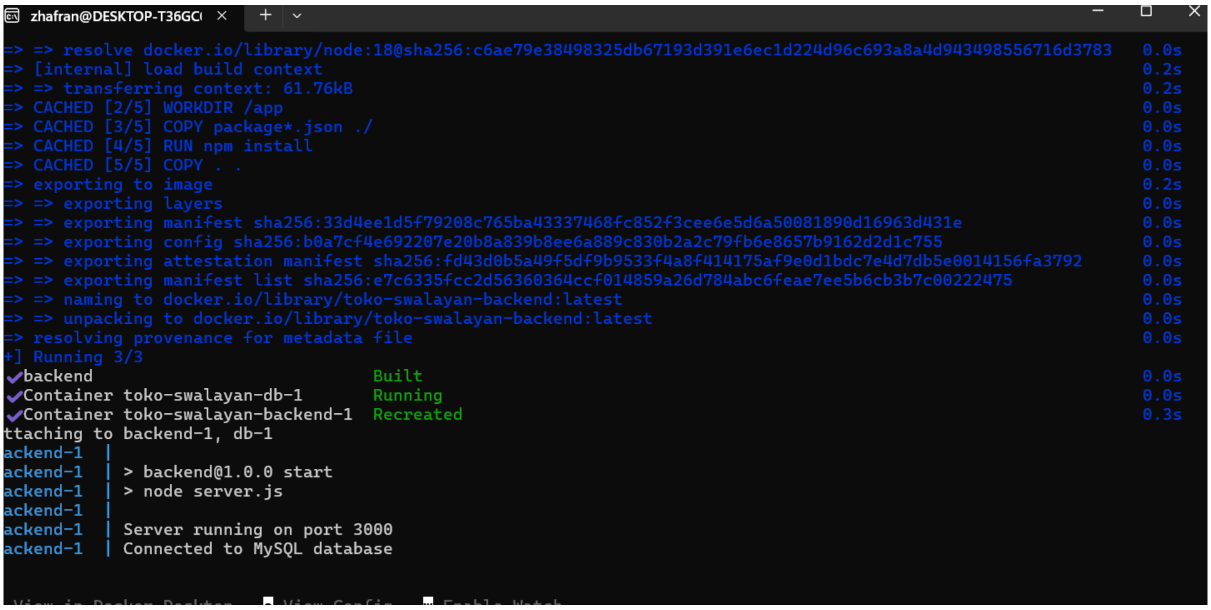


```
zhafran@DESKTOP-T36GC07:~/toko-swalayan$ ls
backend   docker-compose.yml
```



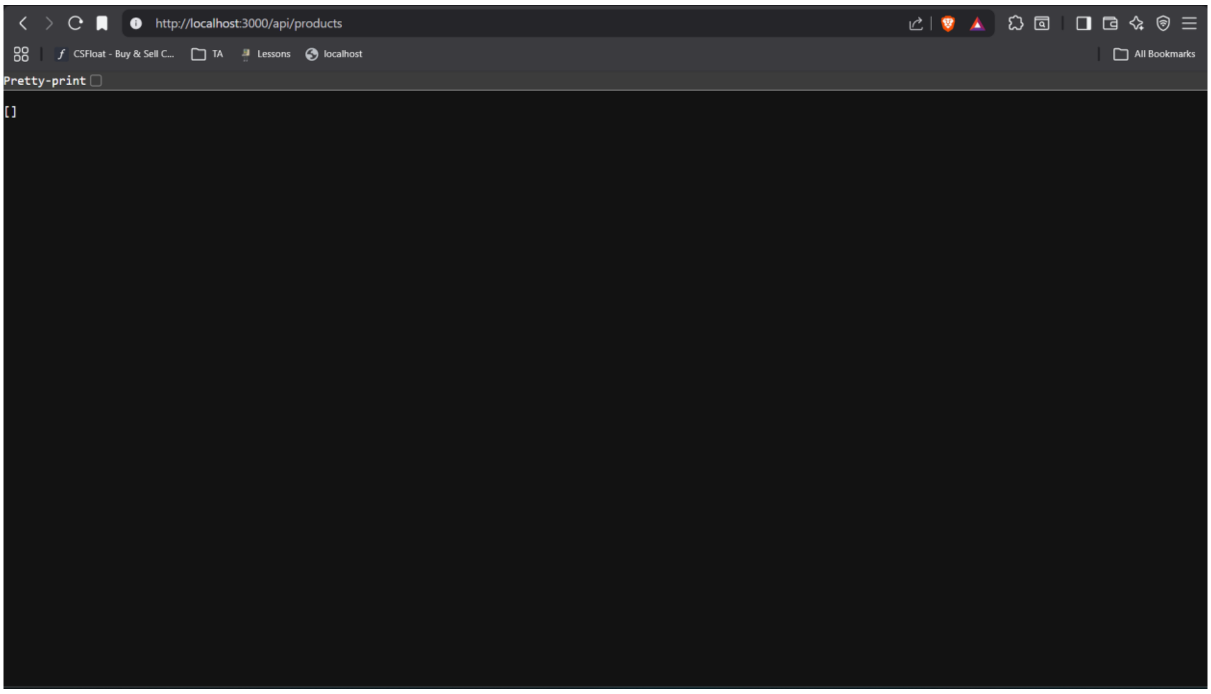
```
zhafran@DESKTOP-T36GC07:~/toko-swalayan/backend$ ls
Dockerfile  node_modules  package-lock.json  package.json  routes  server.js
```

3. **Docker Compose Build:**
   Executed docker compose up --build to start the backend and database containers.



```
=> => resolve docker.io/library/node:18@sha256:c6ae79e38498325db67193d391e6ec1d224d96c693a8a4d943498556716d3783   0.0s
=> [internal] load build context                                                                                   0.2s
=> => transferring context: 61.76kB                                                                                0.2s
=> CACHED [2/5] WORKDIR /app                                                                                       0.0s
=> CACHED [3/5] COPY package*.json ./                                                                              0.0s
=> CACHED [4/5] RUN npm install                                                                                    0.0s
=> CACHED [5/5] COPY . .                                                                                           0.0s
=> exporting to image                                                                                              0.2s
=> => exporting layers                                                                                             0.0s
=> => exporting manifest sha256:33d4ee1d5f79208c765ba43337468fc852f3cee6e5d6a50081890d16963d431e                   0.0s
=> => exporting config sha256:b0a7cf4e692207e20b8a839b8ee6a889c830b2a2c79fb6e8657b9162d2d1c755                     0.0s
=> => exporting attestation manifest sha256:fd43d0b5a49f5df9b9533f4a8f414175af9e0d1bdc7e4d7db5e0014156fa3792       0.0s
=> => exporting manifest list sha256:e7c6335fcc2d56360364ccf014859a26d784abc6feae7ee5b6cb3b7c00222475              0.0s
=> => naming to docker.io/library/toko-swalayan-backend:latest                                                     0.0s
=> => unpacking to docker.io/library/toko-swalayan-backend:latest                                                  0.0s
=> resolving provenance for metadata file                                                                          0.0s
+] Running 3/3
✔backend                            Built                                                                          0.0s
✔Container toko-swalayan-db-1       Running                                                                        0.0s
✔Container toko-swalayan-backend-1  Recreated                                                                      0.3s
ttaching to backend-1, db-1
ackend-1  |
ackend-1  | > backend@1.0.0 start
ackend-1  | > node server.js
ackend-1  |
ackend-1  | Server running on port 3000
ackend-1  | Connected to MySQL database
```

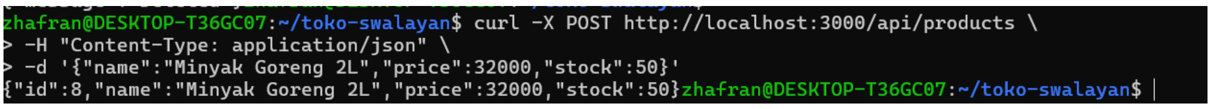
This is how the database should look like when first opening.



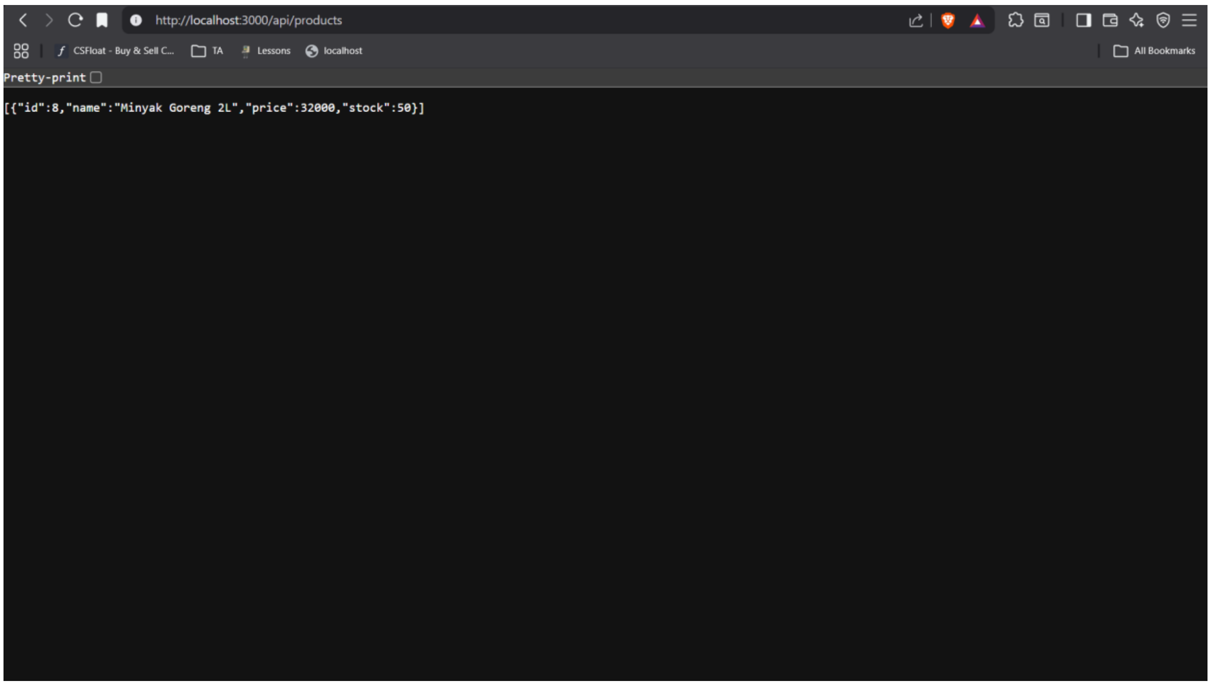
```
http://localhost:3000/api/products

Pretty-print

[]
```

4. **Testing API:**
   Used curl commands to perform POST, GET, PUT, and DELETE requests to verify CRUD functionality.

- POST

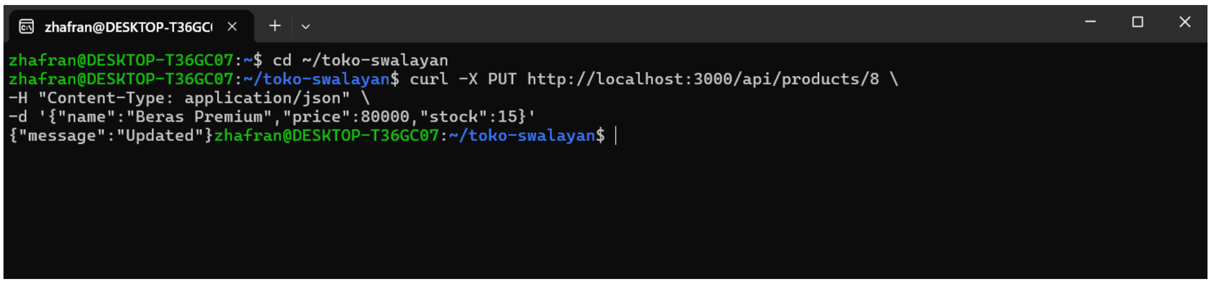


```
zhafran@DESKTOP-T36GC07:~/toko-swalayan$ curl -X POST http://localhost:3000/api/products \
> -H "Content-Type: application/json" \
> -d '{"name":"Minyak Goreng 2L","price":32000,"stock":50}'
{"id":8,"name":"Minyak Goreng 2L","price":32000,"stock":50}zhafran@DESKTOP-T36GC07:~/toko-swalayan$ |
```

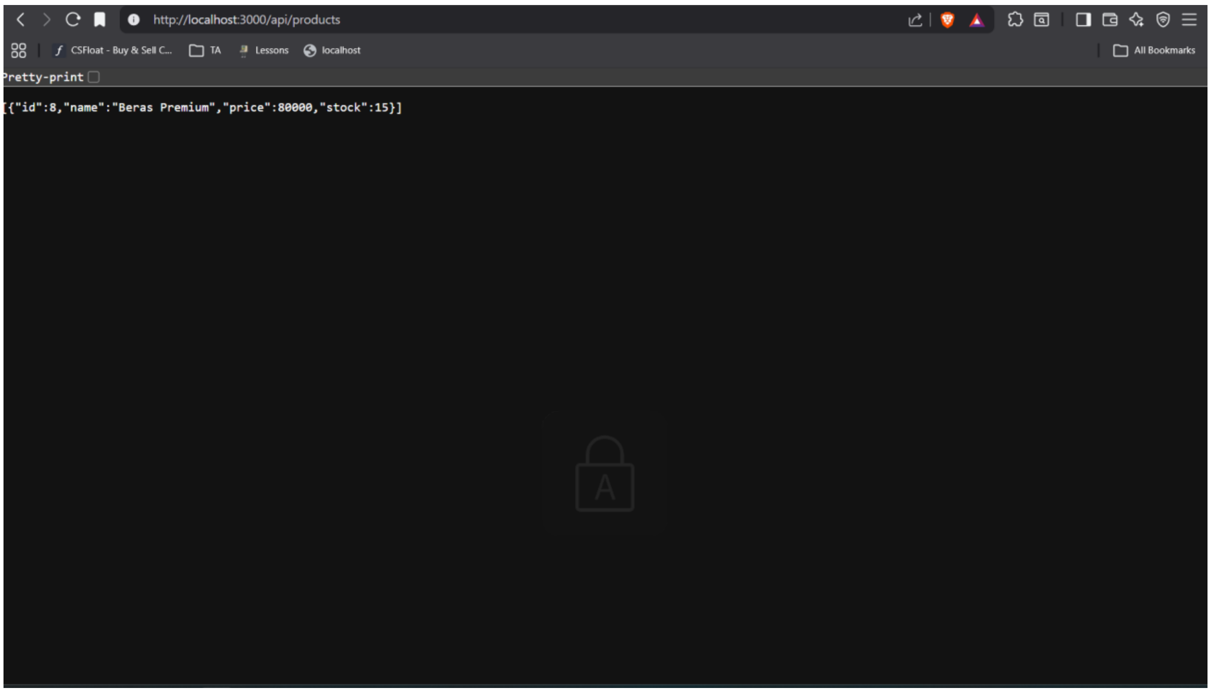


```
[{"id":8,"name":"Minyak Goreng 2L","price":32000,"stock":50}]
```

- PUT

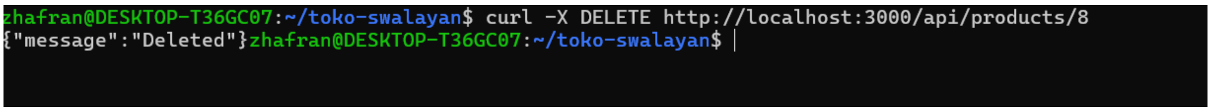


```
zhafran@DESKTOP-T36GC07:~$ cd ~/toko-swalayan
zhafran@DESKTOP-T36GC07:~/toko-swalayan$ curl -X PUT http://localhost:3000/api/products/8 \
-H "Content-Type: application/json" \
-d '{"name":"Beras Premium","price":80000,"stock":15}'
{"message":"Updated"}zhafran@DESKTOP-T36GC07:~/toko-swalayan$
```

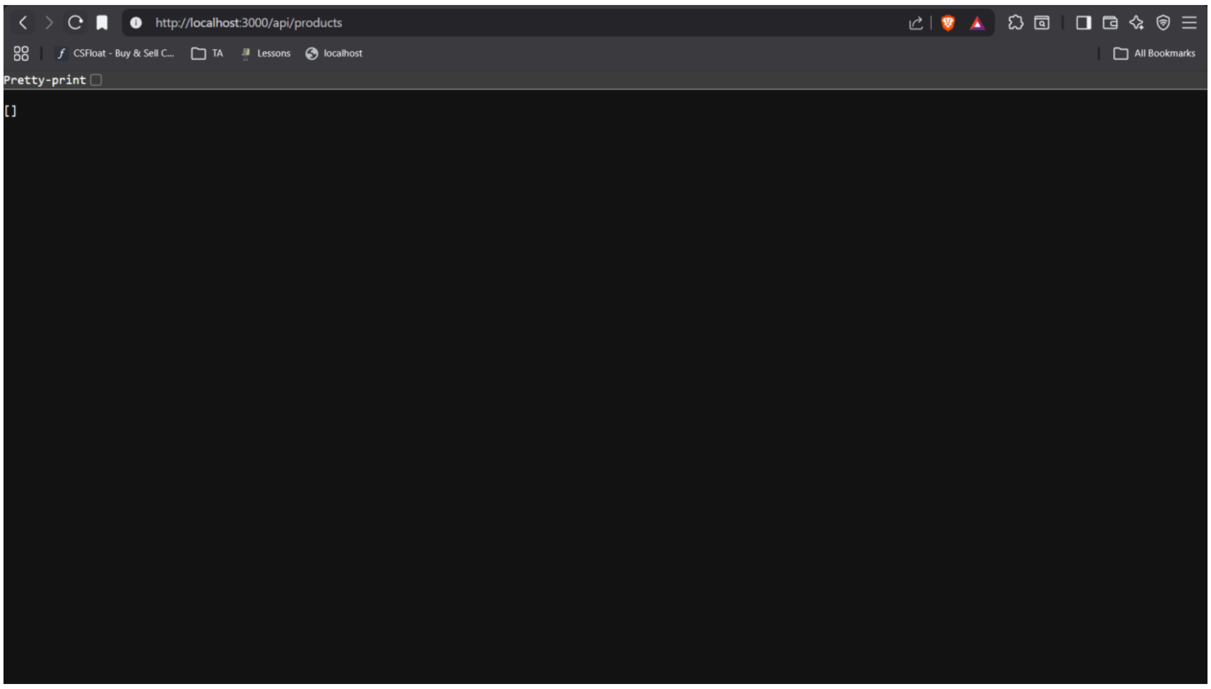


```
[{"id":8,"name":"Beras Premium","price":80000,"stock":15}]
```

- DELETE



```
zhafran@DESKTOP-T36GC07:~/toko-swalayan$ curl -X DELETE http://localhost:3000/api/products/8
{"message":"Deleted"}zhafran@DESKTOP-T36GC07:~/toko-swalayan$
```



```
[]
```

TECHNICAL ANALYSIS

Component	Implementation	Purpose
Docker Containers	Backend and MySQL run in separate containers	Isolation and portability
Docker Compose	Orchestrates backend and database networking	Ensures dependency order
MySQL Volume	Persistent database storage	Data durability
Express REST API	Handles CRUD operations	Data interaction layer

PROBLEMS AND SOLUTIONS

Problem	Cause	Solution
MySQL “permission denied” in WSL	Docker daemon not integrated with WSL	Enabled Docker WSL integration
Backend exited with “table not found”	Database initialized without table	Created products table manually
curl JSON error in PowerShell	Wrong JSON escaping or headers	Used correct -H "Content-Type: application/json" format
Connection refused on startup	MySQL not ready before Node.js	Added dependency depends_on in docker-compose.yml