DevOps Documentation for Full-Stack Todo List Application

Developed By	**Kidest W.**
Reviewed By	
Approved By	

March 2025

Table of Contents

1. Project Overview

1.1 Purpose and Goals

This project involves containerizing a full-stack Todo List application following a three-tier architecture, consisting of a frontend (React), backend (Node.js/Express), and database (MongoDB). The goal is to create a scalable, portable, and easily deployable system using Docker and Docker Compose, ensuring seamless integration between the services.

Application Functionalities

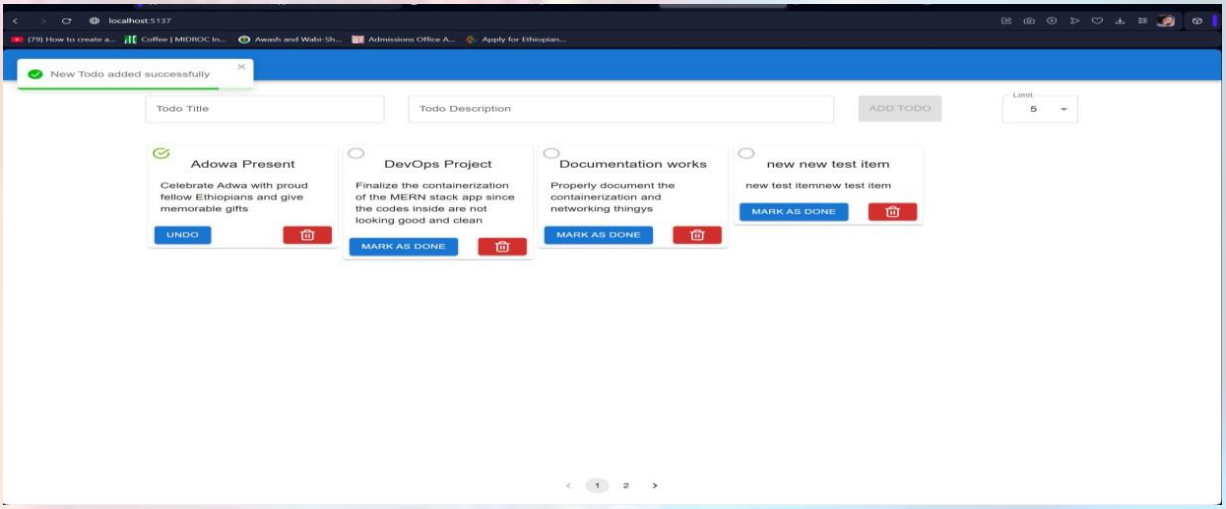
1. Add a New Task

Users can **add a task** by entering a **title and description** in the input fields and clicking the **"Add Task"** button.

How It Works:

1. The user enters a task title and description in the input fields.
2. Clicking the "Add Task" button sends a **POST request** to the backend API (/api/todos).
3. The backend stores the new task in the **MongoDB database**.

The task appears in the list immediately.

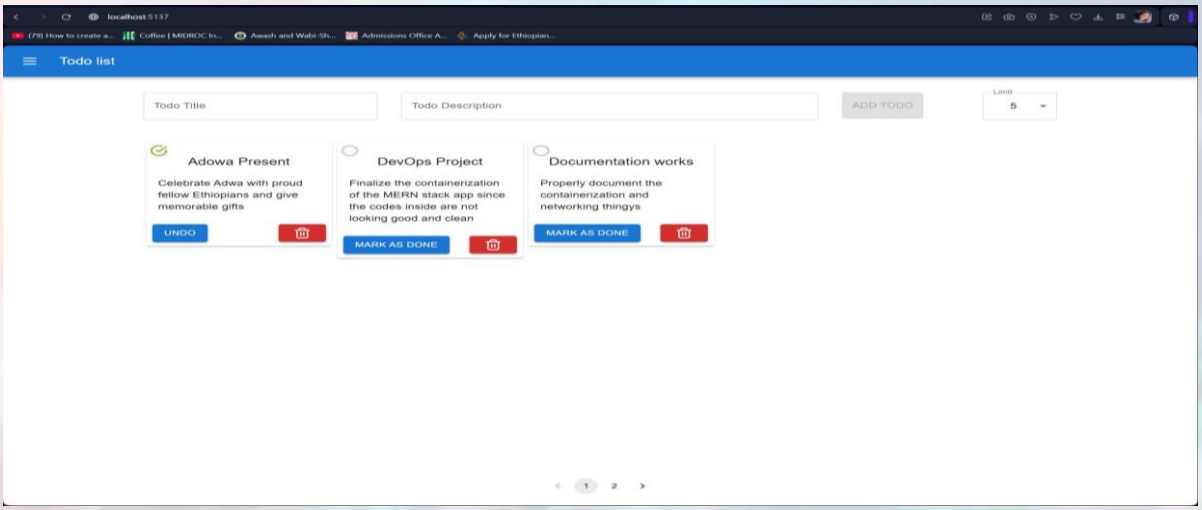


2. Edit a Task (Mark as Complete/Undo)

Users can edit a task to **mark it as complete or undo the mark** when needed.

How It Works:

1. Clicking the **edit button** next to a task allows the user to toggle its status.
2. The system updates the task status in the frontend and sends a **PUT request** to the backend (/api/todos/:id).
3. The updated task is saved in the **MongoDB database**.
4. If the task is marked as **complete**, it appears with a **strikethrough or different style**.
5. If the mark is undone, the task returns to its **normal state**.

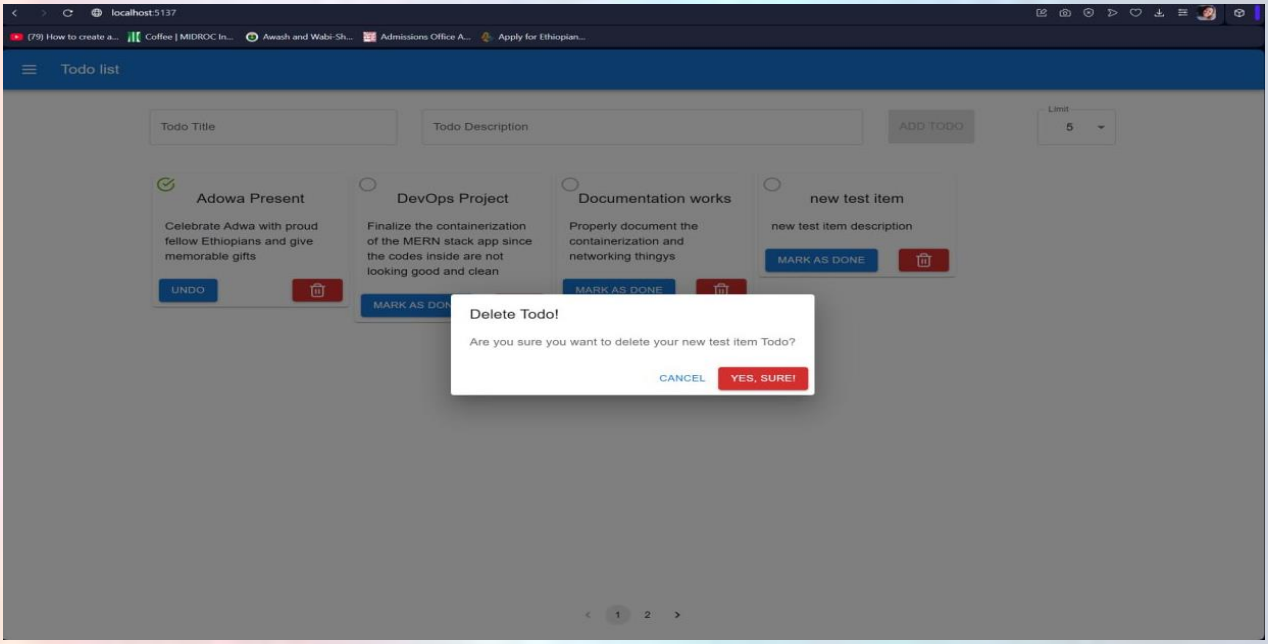


3. Delete a Task (With Confirmation Prompt)

Users can delete a task, but before deletion, the system **asks for confirmation** to prevent accidental removal.

How It Works:

1. Clicking the **delete icon** triggers a **confirmation popup** asking the user to confirm deletion.
2. If the user confirms, the system sends a **DELETE request** to the backend (/api/todos/:id).
3. The backend removes the task from the **MongoDB database**.
4. The task **disappears from the list** immediately after deletion.

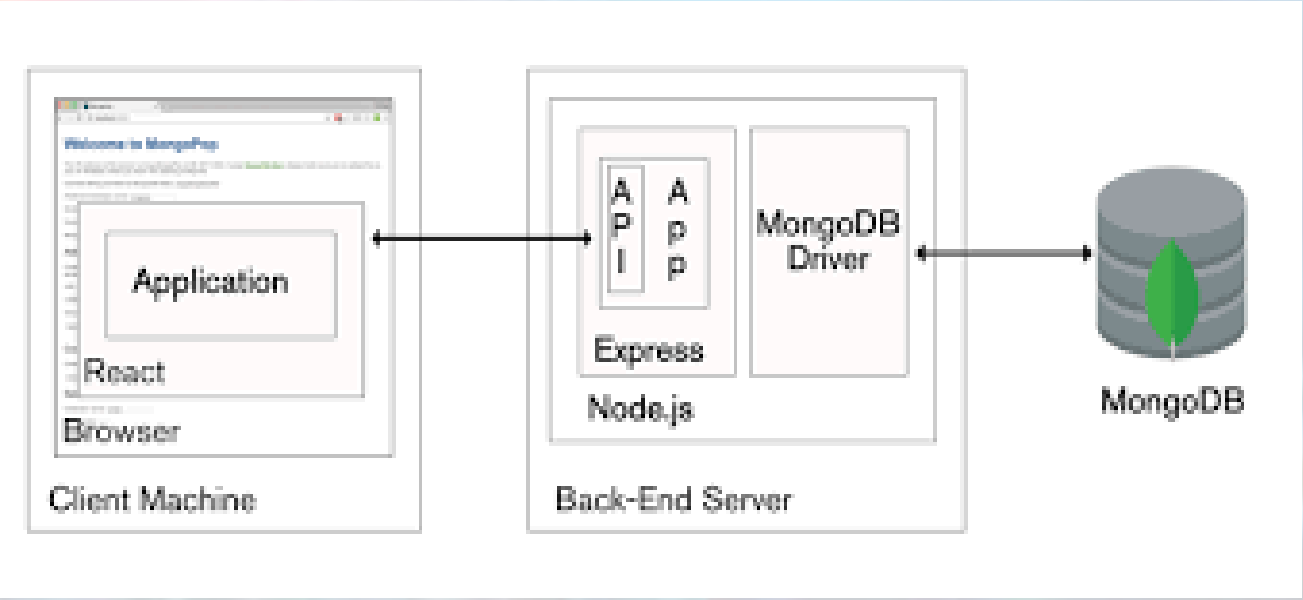


1.2 System Architecture

The application consists of:

- **Frontend** (React) – Provides the user interface.
- **Backend** (Node.js/Express) – Handles API requests and business logic.
- **Database** (MongoDB) – Stores and retrieves todo list data.

The **frontend** provides a dynamic user interface where users can add, edit (mark as complete/undo), and delete tasks, ensuring real-time updates through API calls. The **backend** handles API requests, processes business logic, and interacts with the database to store and retrieve tasks efficiently. It ensures secure communication and error handling. The **MongoDB database** serves as persistent storage, maintaining all task records with flexible JSON-based schemas. This structured architecture ensures scalability, separation of concerns, and smooth interaction between components.



2. Development and Containerization Process

2.1 Dockerization of Components

The **Dockerization of the Full-Stack Todo List Application** ensures a seamless, portable, and scalable deployment by containerizing its three-tier architecture. Each component—**frontend, backend, and database**—runs inside its own **Docker container**, ensuring isolation and consistency across environments.

The **Dockerfiles** define the dependencies and configurations for the **React frontend**, **Node.js backend**, and **MongoDB database**, ensuring that each service has a dedicated environment. The **Docker Compose** file orchestrates these containers, managing network configurations, environment variables, and volume mappings for data persistence. This containerized approach enhances **scalability**, **dependency management**, and **cross-platform compatibility**, allowing the application to run smoothly across different environments without configuration conflicts.

2.1.1 Backend Dockerfile

```
# Use official Node.js image
FROM node:18-alpine

# Set working directory
WORKDIR /app

# Copy package.json and install dependencies
COPY package.json package-lock.json ./
RUN npm install

# Copy the source code
COPY . .

# Expose backend port
EXPOSE 5000

# Command to start the backend server
CMD ["npm", "start"]
```

2.1.2 Frontend Dockerfile

```
# Use official Node.js image
FROM node:18-alpine

# Set working directory
WORKDIR /app

# Copy package.json and install dependencies

COPY package.json package-lock.json ./

RUN npm install

# Copy the source code
COPY . .

# Build the frontend
RUN npm run build

# Expose frontend port
EXPOSE 5137

# Serve the frontend
CMD ["npm", "run", "dev"]
```

2.2 Orchestration with Docker Compose

The **docker-compose.yml** file defines how the services are networked and configured and implements the docker file elements defined in each stack.

2.2.1 Docker Compose File

```
version: '3.8'

services:
  backend:
    build:
      context: ./backend
    ports:
      - "5000:5000"
    depends_on:
      - mongo
    restart: unless-stopped
    environment:
      - NODE_ENV=production
      - MONGO_URI=mongodb://mongodb:27017/todos
    networks:
      - todonet
    volumes:
      - ./backend:/app
```

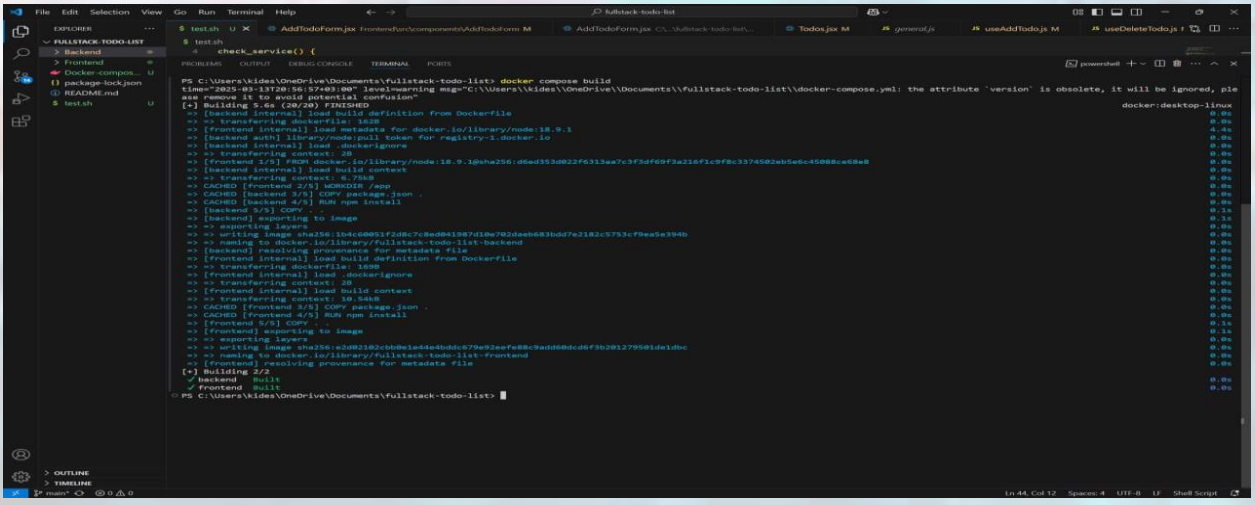
```
      - /app/node_modules

  frontend:
    build:
      context: ./frontend
    ports:
      - "5137:5137"
    depends_on:
      - backend
    environment:
      - REACT_APP_API_URL=http://localhost:5000
    networks:
      - todonet
    volumes:
      - ./frontend:/app
      - /app/node_modules

  mongo:
    image: mongo:6.0
    container_name: fullstack-todo-list-mongodb
    restart: unless-stopped
    ports:
      - "27017:27017"
    networks:
      - todonet
    volumes:
      - mongo_data:/data/db

networks:
  todonet:
    driver: bridge

volumes:
  mongo_data:
    driver: local
```



2.3 Network Configuration & Security Settings

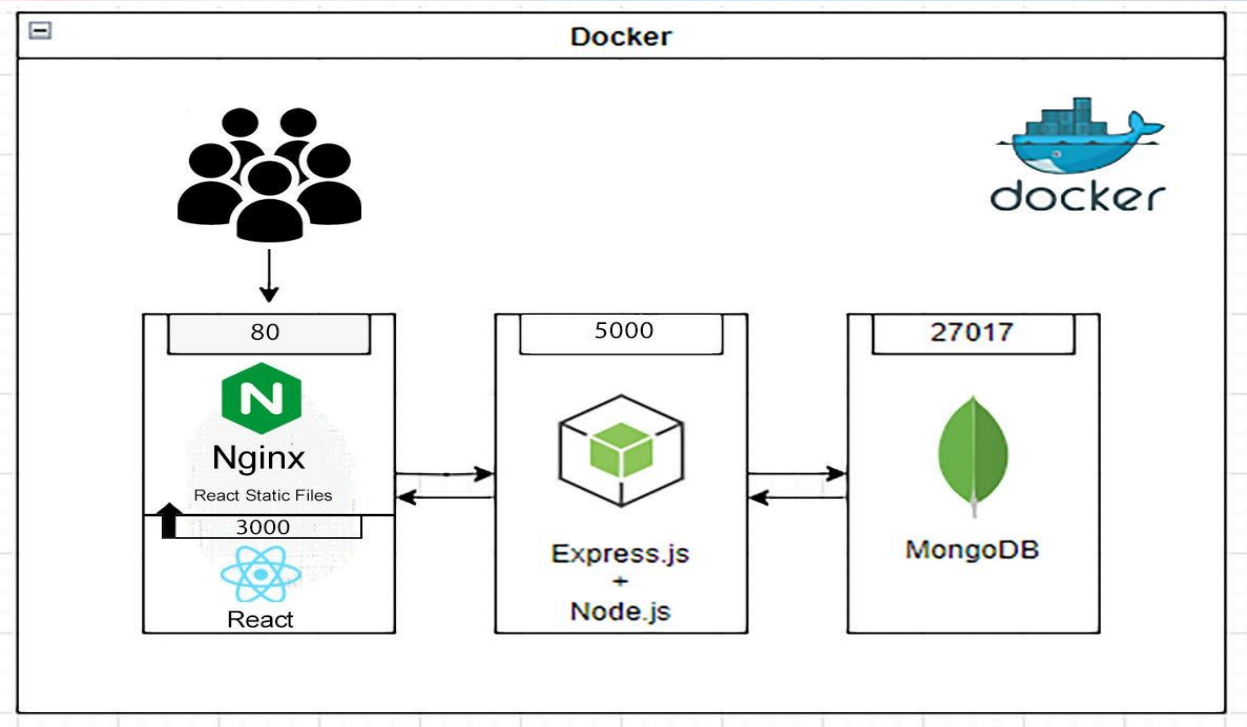
Network: A **bridge network (`todonet`)** ensures that services communicate securely within an isolated environment.

Exposed Ports:

- **Frontend:** `5137:5137`
- **Backend:** `5000:5000`
- **MongoDB:** `27017:27017`

Security Considerations:

- Environment variables are used for sensitive data.
- The backend service connects to MongoDB using a private **internal network** (`todonet`).



3. Running the Application

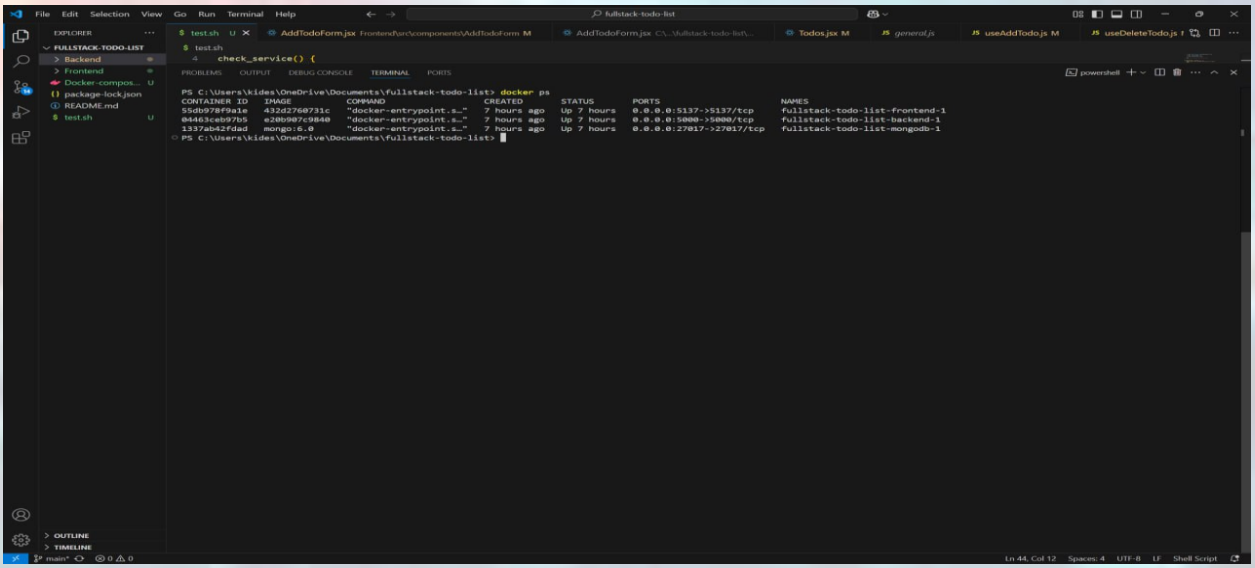
3.1 Prerequisites

Ensure you have:

- Docker installed (`docker -v`)
- Docker Compose installed (`docker-compose -v`)

3.2 Step-by-Step Setup

1. **Clone the Repository:**
2. `git clone https://github.com/therepo/fullstack-todo-list.git`
3. `cd fullstack-todo-list`
4. **Build and Start Containers:**
5. `docker-compose up --build -d`
6. **Verify Running Containers:**
7. `docker ps`
8. **Stopping Containers:**
9. `docker-compose down`



4. API Documentation

4.1 Available Endpoints

GET /api/todos

Retrieves a list of all todo items.
Response:

```
[
  {
    "id": "1",
    "title": "Complete documentation",
    "completed": false
  }
```

POST /api/todos

Creates a new todo item.
Request Body:

```
{
  "title": "New Task"
}
```

Response:

```
{
  "id": "2",
  "title": "New Task",
  "completed": false
}
```

PUT /api/todos/:id

Updates a todo item.
Request Body:

```
{
  "completed": true
}
```

DELETE /api/todos/:id

Deletes a todo item.

5. Testing & Verification

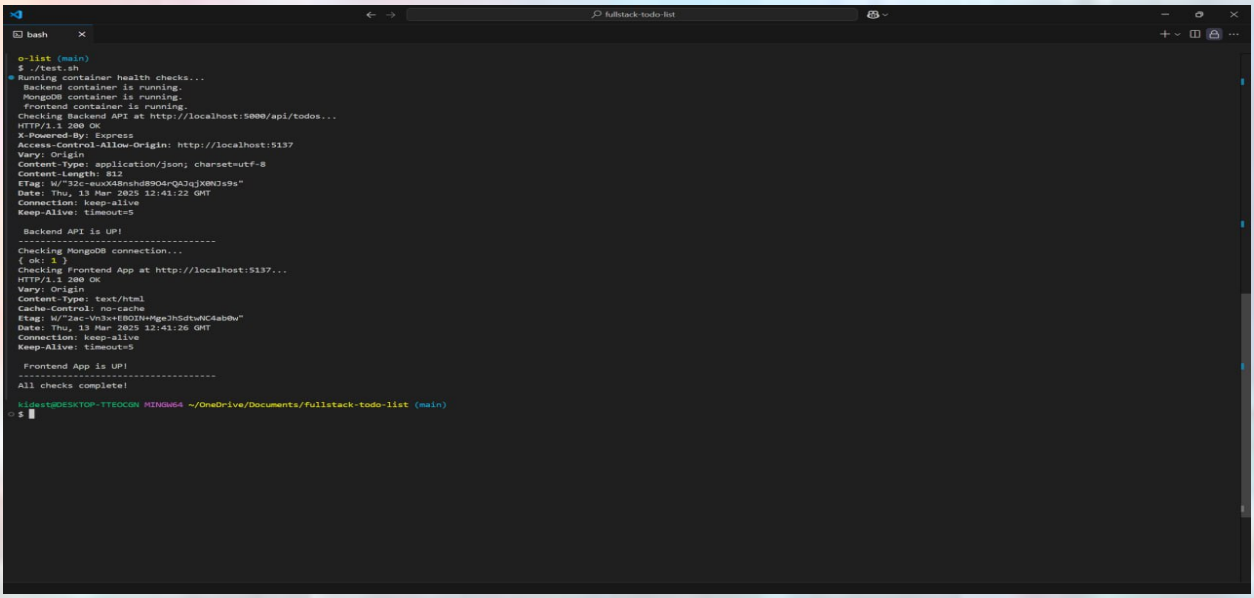
5.1 Test Script (`test.sh`)

A separate shell test script is used to assess and check the health and performance of the containers including the system, network and ports.

Expected Results

- Backend container is Ruuning
- Mongodb container is running
- Frontend container is running

- Backend API is UP
- Frontend APP is UP
- All checks Complete !



6, Additional Details

Monitoring & Logging

Docker logs:

- `docker logs containername`

Docker stats:

- `docker stats`

Troubleshooting Guide

Issue	Possible Cause	Solution
Containers not starting	Port conflicts	Change exposed ports
Backend cannot connect to DB	Wrong `MONGO_URI`	Ensure database service is running
Frontend cannot access API	Wrong `REACT_APP_API_URL`	Update `.env` variable