

**Mid Exam Report**

**Distributed and Parallel System  
Created by:**

**Tengku Mahesa Omar Falah (001202300121)**

**Putra Amalul Kamal (001202300)**

**1. Project Overview**

**Application Description**

Task Manager Pro is a full-stack web application that allows users to manage their daily tasks efficiently. The application consists of:

* **Frontend:** React.js application with modern UI/UX
* **Backend:** Node.js Express API with RESTful endpoints
* **Database:** MongoDB for data persistence
* **Containerization:** Docker and Docker Compose for deployment

**Key Features**

* ✅ Add new tasks
* ✅ Mark tasks as complete/incomplete
* ✅ Delete tasks
* ✅ Filter tasks (All, Active, Completed)
* ✅ Real-time task statistics
* ✅ Responsive design
* ✅ Health check endpoints

**2. Technology Stack**

**Frontend**

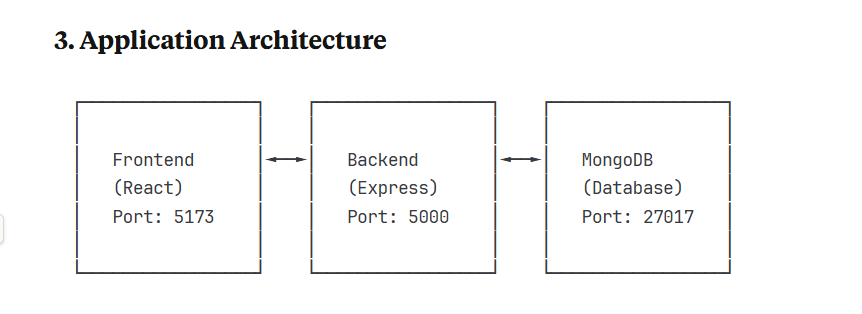
* **Framework:** React.js 18.2.0
* **Build Tool:** Vite 4.4.5
* **HTTP Client:** Axios 1.6.0
* **Styling:** Custom CSS with modern design

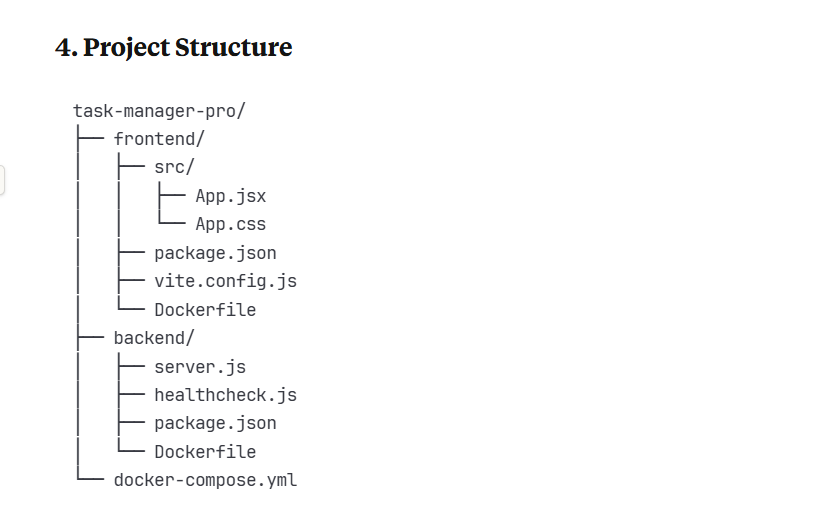
**Backend**

* **Runtime:** Node.js 18
* **Framework:** Express.js 4.18.2
* **Database:** MongoDB with Mongoose ODM
* **Additional:** CORS, dotenv for environment variables

**Containerization**

* **Container Platform:** Docker
* **Orchestration:** Docker Compose
* **Base Images:** Node.js 18 Alpine, MongoDB 7.0





**5. Docker Implementation**

**Frontend Dockerfile**

The frontend Dockerfile uses Node.js 18 Alpine image for lightweight deployment:

* Installs dependencies using npm
* Exposes port 5173
* Runs Vite development server with host configuration

**Backend Dockerfile**

The backend Dockerfile includes security best practices:

* Uses Node.js 18 Alpine base image
* Creates non-root user for security
* Implements health check using custom script
* Exposes port 5000

**Docker Compose Configuration**

The docker-compose.yml orchestrates three services:

* **MongoDB:** Database service with health check
* **Backend:** API service dependent on MongoDB
* **Frontend:** React app dependent on Backend

**6. Step-by-Step Deployment Guide**

**Prerequisites**

* Docker installed on your system
* Docker Compose installed
* Git for version control

**Step 1: Clone/Setup Project Structure**

bash

*# Create project directory*

mkdir task-manager-pro

cd task-manager-pro

*# Create subdirectories*

mkdir frontend backend

**Step 2: Setup Frontend**

bash

cd frontend

*# Create all frontend files (App.jsx, App.css, package.json, etc.)*

*# Create Dockerfile*

**Step 3: Setup Backend**

bash

cd ../backend

*# Create all backend files (server.js, package.json, etc.)*

*# Create Dockerfile*

**Step 4: Create Docker Compose**

bash

cd ..

*# Create docker-compose.yml in root directory*

**Step 5: Build and Deploy**

bash

*# Build and start all services*

docker-compose up --build

*# Run in detached mode*

docker-compose up -d --build

**Step 6: Verify Deployment**

bash

*# Check running containers*

docker-compose ps

*# View logs*

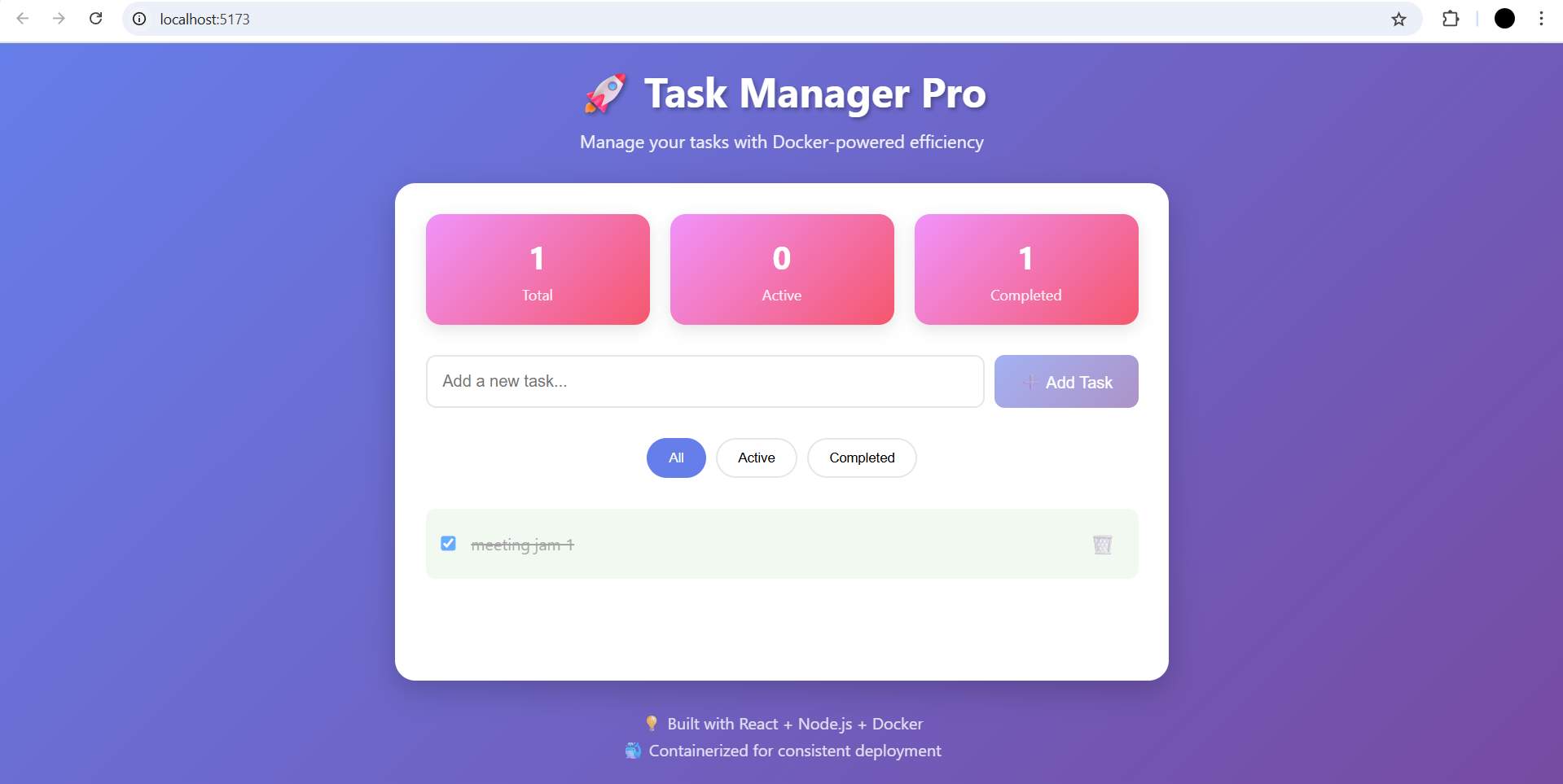
docker-compose logs

*# Test endpoints*

curl http://localhost:5000/api/health

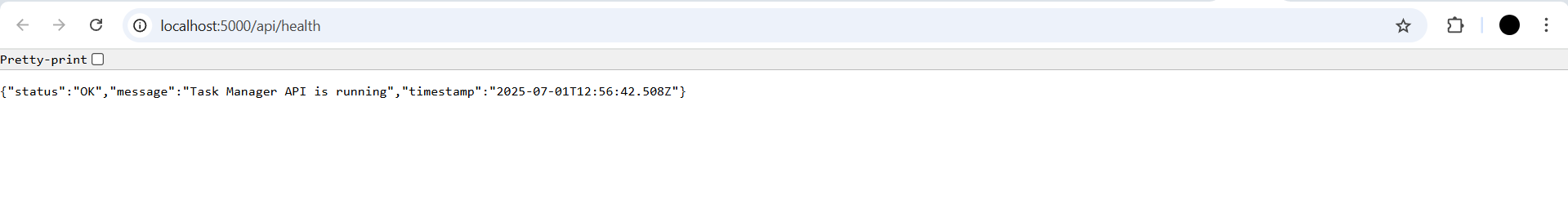
## 7. Screenshots

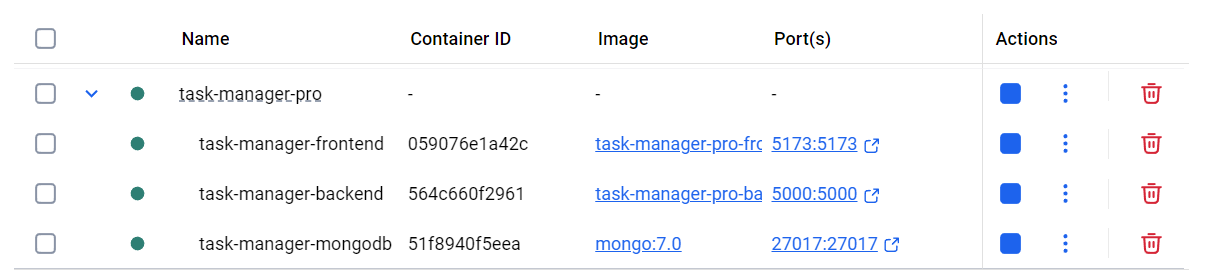
### Application Interface



**Main application interface showing task management features**

**API Health Check**

**  
Backend API health check endpoint response**

**Docker Containers  
**

**Docker containers running successfully**

## 8. Docker Commands Reference

### Basic Commands

bash

*# Build and start services*

docker-compose up --build

*# Stop services*

docker-compose down

*# View running containers*

docker-compose ps

*# View logs*

docker-compose logs [service-name]

*# Execute commands in container*

docker-compose exec [service-name] [command]

### Development Commands

bash

*# Rebuild specific service*

docker-compose build [service-name]

*# Scale services*

docker-compose up --scale [service-name]=3

*# Remove volumes*

docker-compose down -v

## 9. Environment Configuration

### Frontend Environment Variables

env

VITE\_API\_URL=http://localhost:5000/api

### Backend Environment Variables

env

PORT=5000

MONGODB\_URI=mongodb://mongodb:27017/taskmanager

NODE\_ENV=development

## 10. Testing the Application

### Functional Testing Steps

1. **Access Frontend:** Navigate to <http://localhost:5173>
2. **Add Task:** Enter task title and click "Add Task"
3. **Mark Complete:** Click checkbox to toggle completion
4. **Filter Tasks:** Use filter buttons (All, Active, Completed)
5. **Delete Task:** Click delete button to remove task
6. **API Testing:** Access <http://localhost:5000/api/health>

### Expected Results

* Frontend displays modern task management interface
* Tasks can be added, completed, and deleted
* Statistics update in real-time
* API returns JSON responses
* All containers run without errors

## 11. Network Configuration

### Docker Network

* **Network Name:** task-manager-network
* **Driver:** bridge
* **Services Communication:** Internal DNS resolution

### Port Mapping

* Frontend: Host 5173 → Container 5173
* Backend: Host 5000 → Container 5000
* MongoDB: Host 27017 → Container 27017

## 12. Volume Management

### Persistent Data

* **MongoDB Data:** Persistent volume for database
* **Development Volumes:** Hot reload for code changes

## 13. Security Considerations

### Backend Security

* Non-root user execution
* Input validation and sanitization
* CORS configuration
* Error handling middleware

### Container Security

* Alpine Linux base images (smaller attack surface)
* Health checks for service monitoring
* Resource limitations through Docker Compose

## 14. Monitoring and Health Checks

### Health Check Implementation

* Backend health check endpoint: /api/health
* MongoDB health check using mongosh
* Docker health check configuration
* Service dependency management

## 15. Troubleshooting Guide

### Common Issues

1. **Port conflicts:** Change port mappings in docker-compose.yml
2. **MongoDB connection:** Ensure MongoDB service is healthy
3. **CORS errors:** Verify backend CORS configuration
4. **Build failures:** Check Dockerfile syntax and dependencies

### Debug Commands

bash

*# Check container logs*

docker-compose logs [service-name]

*# Interactive shell*

docker-compose exec [service-name] /bin/sh

*# Inspect networks*

docker network ls

docker network inspect task-manager-pro\_task-manager-network

## 16. Conclusion

Task Manager Pro successfully demonstrates:

* **Containerization:** All services running in Docker containers
* **Service Orchestration:** Docker Compose managing multi-container application
* **Modern Architecture:** Separation of concerns with frontend, backend, and database
* **Development Workflow:** Hot reload and development-friendly configuration
* **Production Ready:** Health checks, security, and monitoring

The application showcases practical implementation of distributed systems concepts using Docker, providing a scalable and maintainable solution for task management.