# MERN Stack Project Deployment Documentation

## Introduction

This document provides a comprehensive overview of the deployment process for a MERN stack application, featuring user authentication and a password reset module, using Docker Compose and Kubernetes for local and managed deployments. The architecture includes a frontend React application served by Nginx, an Express backend with JWT authentication, and a MongoDB database.

## Architecture Diagram

The architecture comprises three main components: frontend, backend, and MongoDB, configured as microservices. Docker Compose manages the local containerized environment, while Kubernetes provides managed deployment with scalability. Below is a visual representation of the microservice interaction and networking setup:

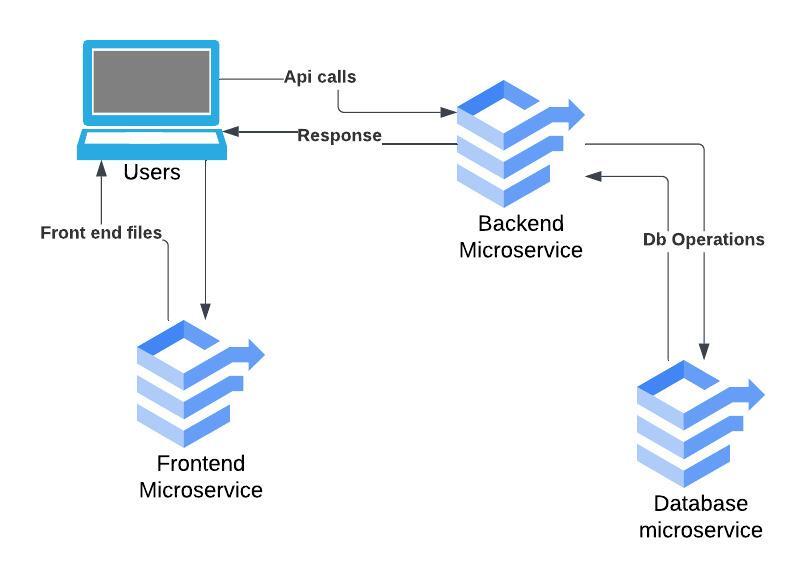


Figure 1: Architecture Diagram of Microservices

## Deployment Process

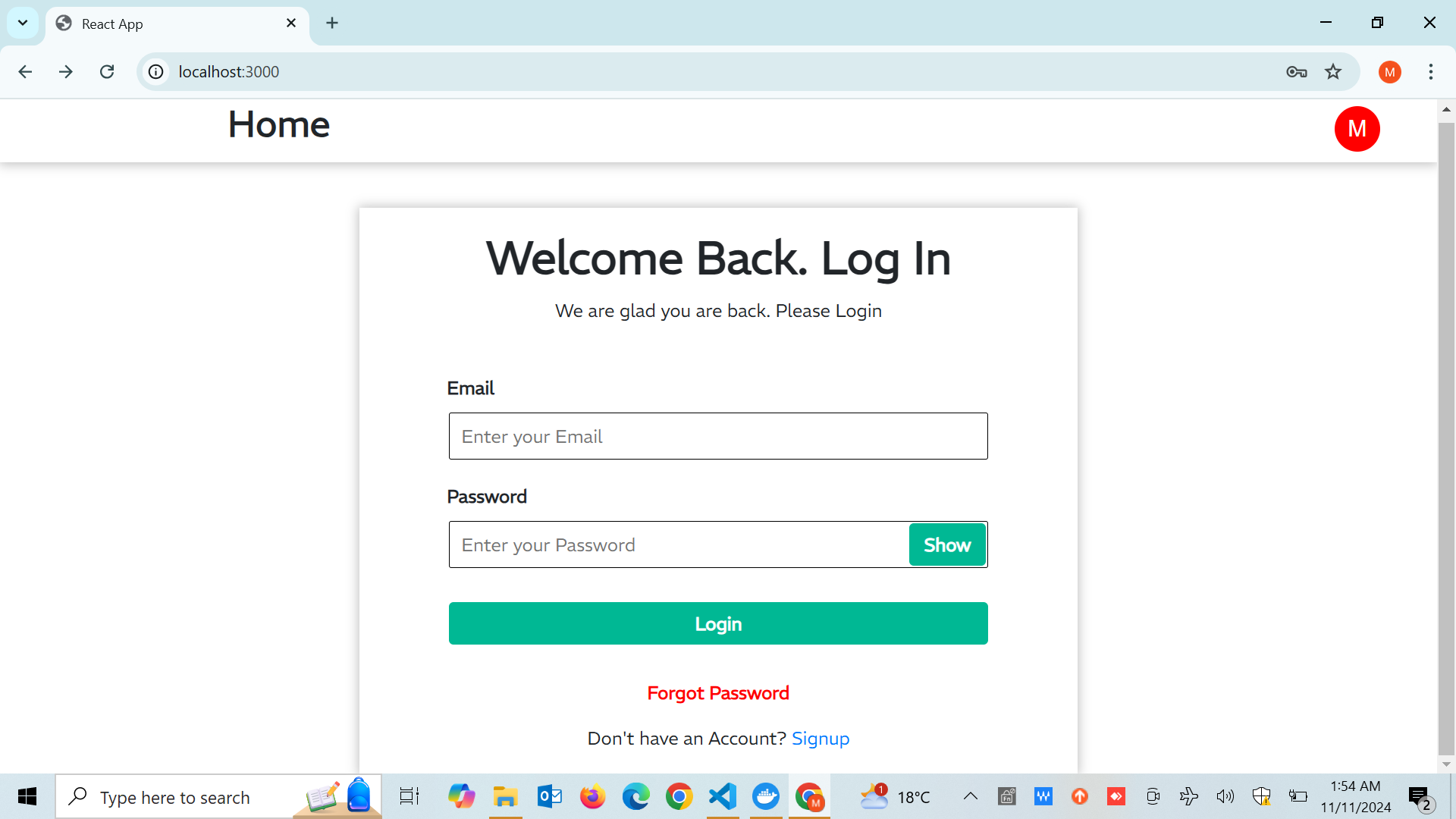
### 1. Docker Compose Deployment

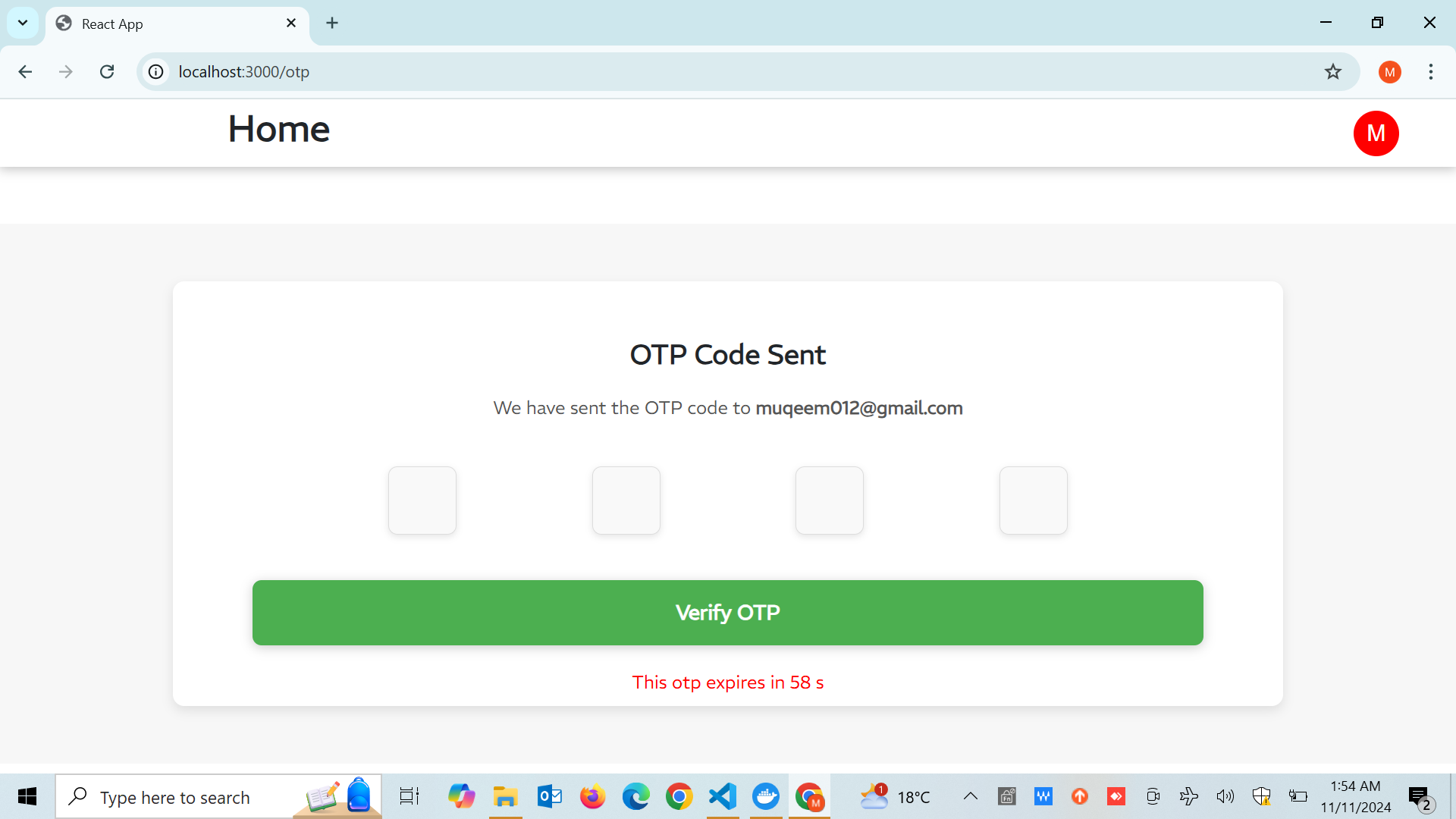
The Docker Compose deployment manages each layer of the MERN stack as separate services with individual container images. This section details the configuration for each service, environment variables, and network settings.

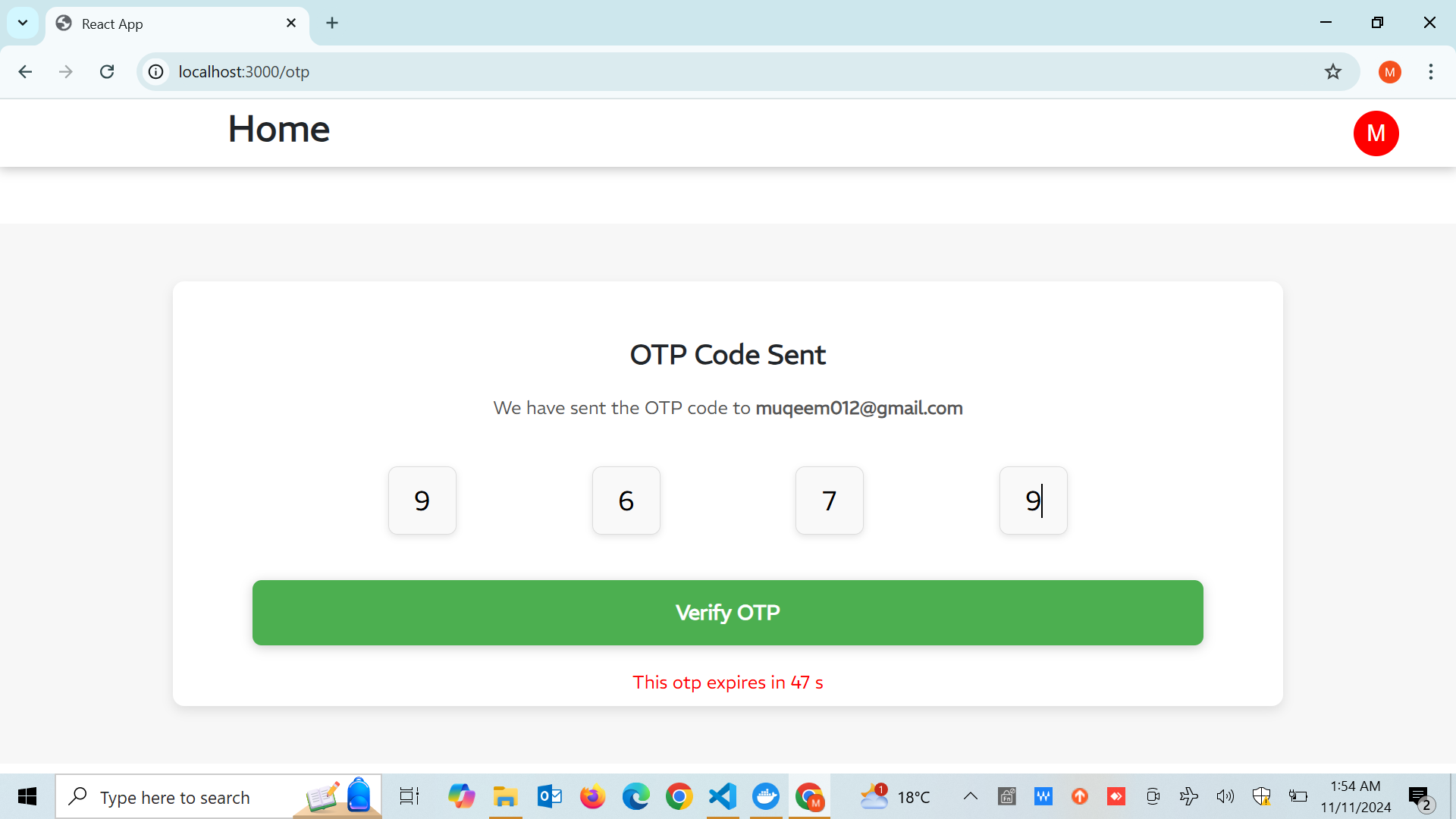
**Frontend Service**: Built from the React codebase, served via an Nginx image. The React app is built and stored in the /usr/nginx/html directory, accessible on port 3000 via Docker Compose (mapped from port 80 in the container).

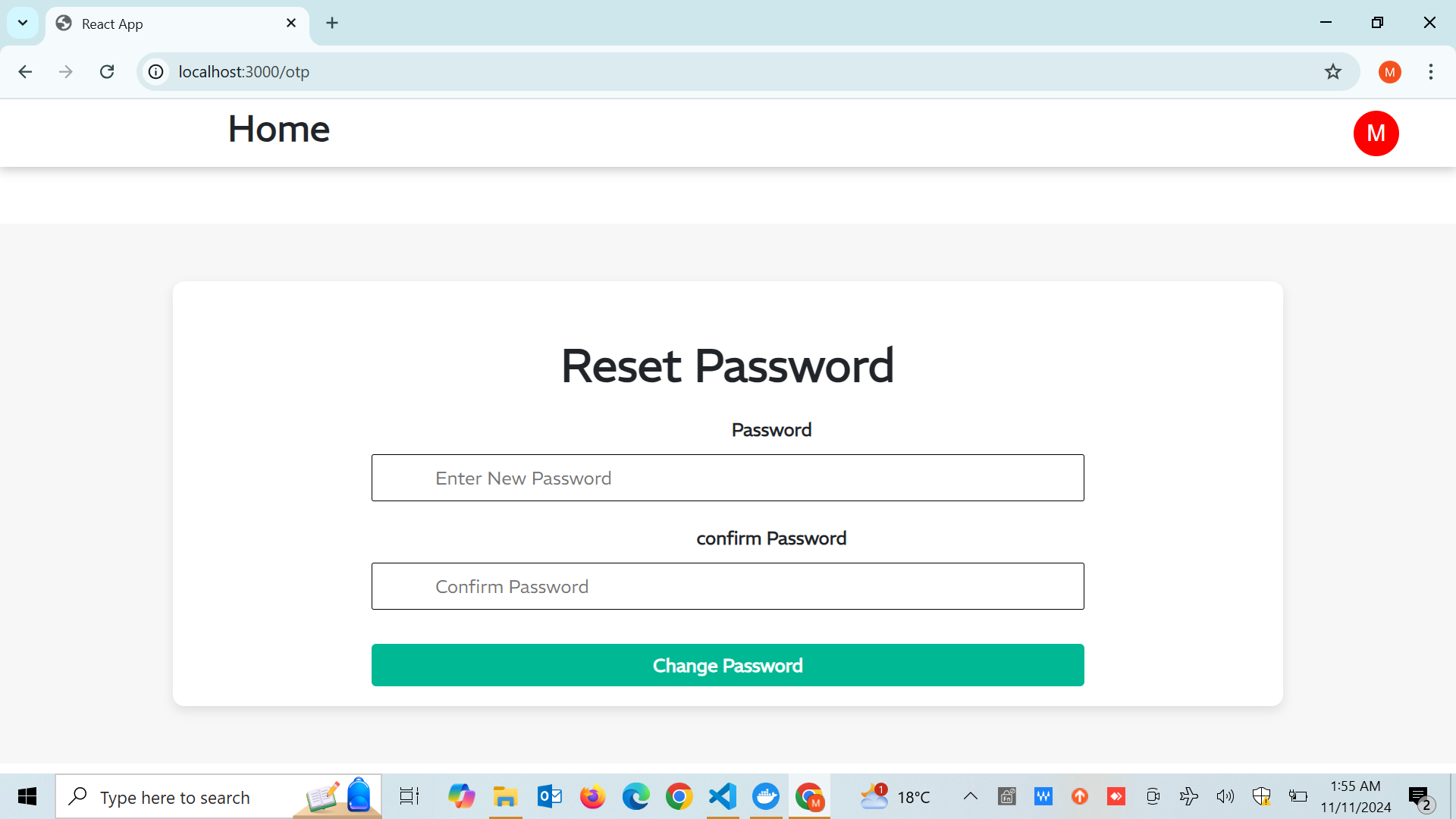
**Backend Service**: Built from an Express.js codebase using MONGO\_URI to reference the MongoDB service by name. The backend listens on port 5000, mapped to port 5000 on the host machine for external access.

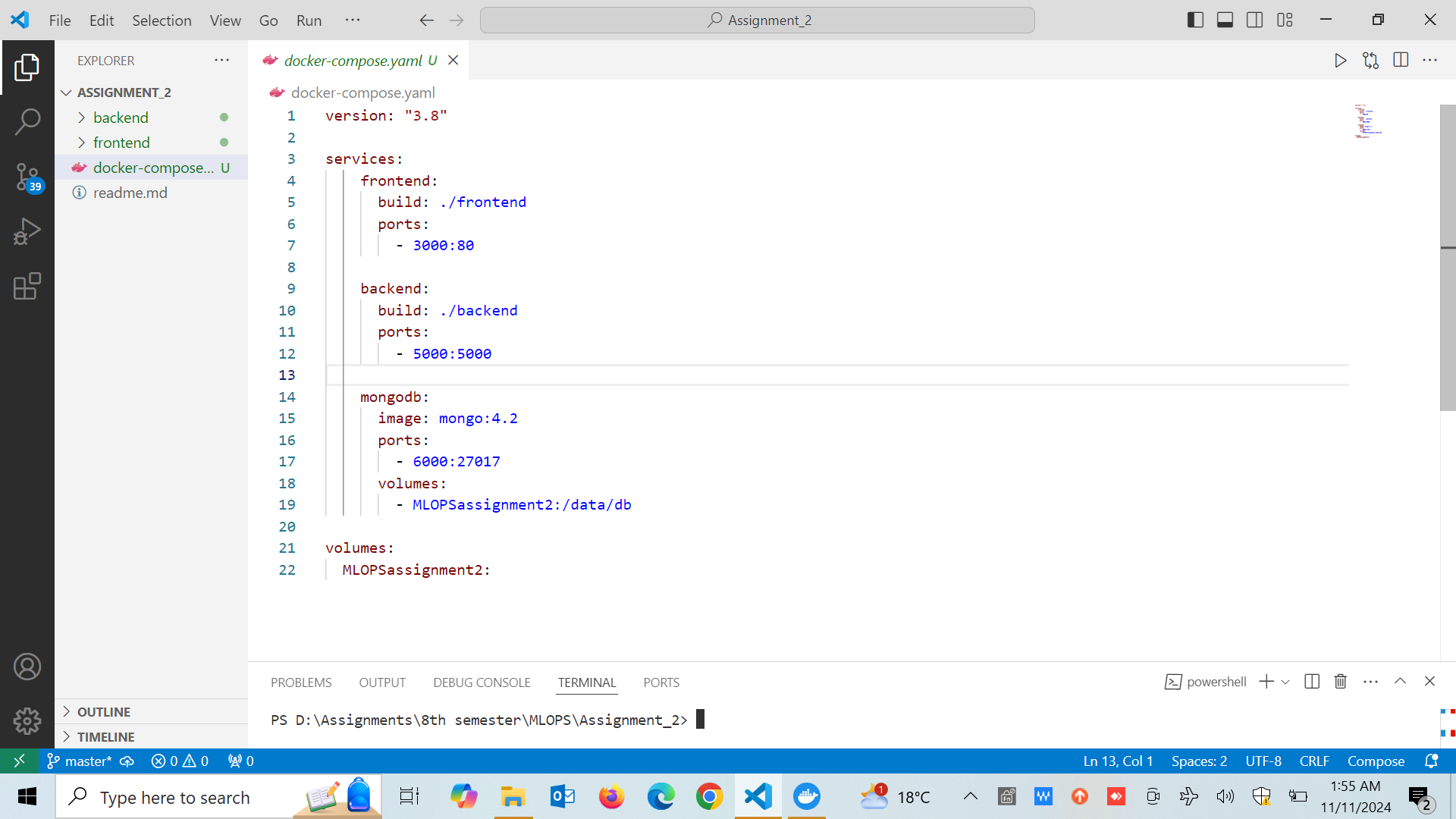
**MongoDB Service:** Uses a MongoDB image with a persistent volume to ensure data is stored permanently. The database service is accessible on port 27017 and communicates directly with the backend.

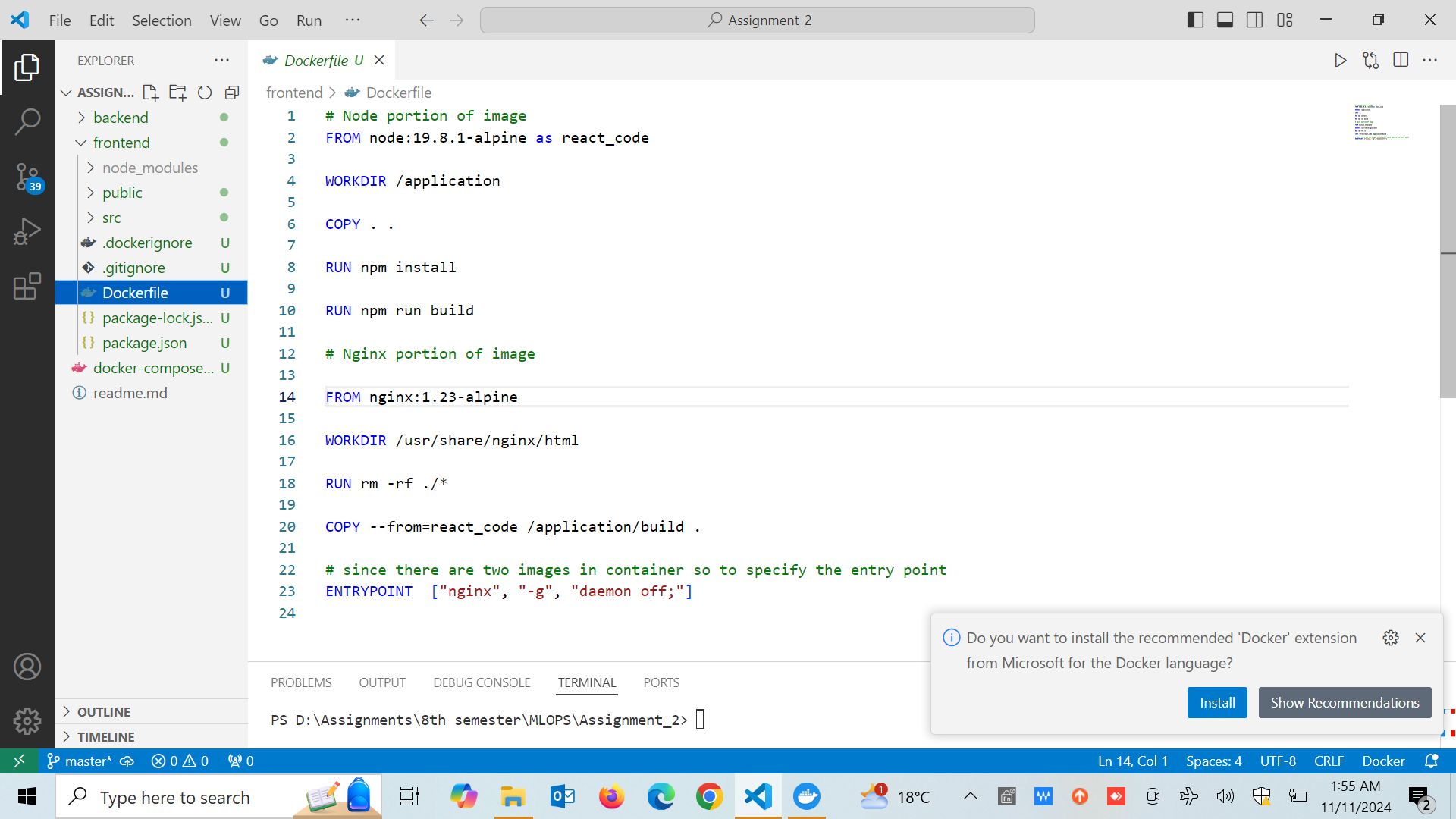


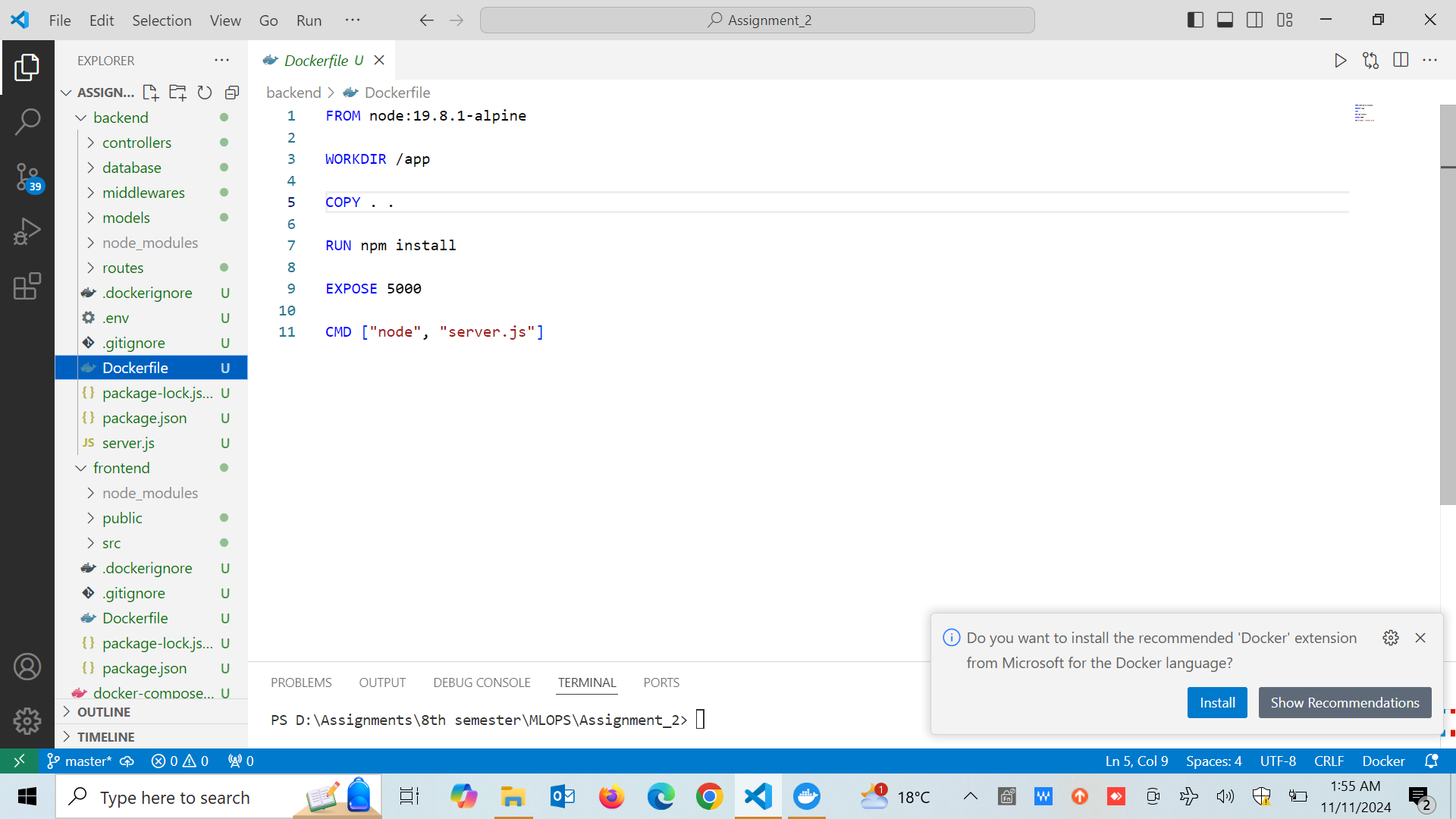












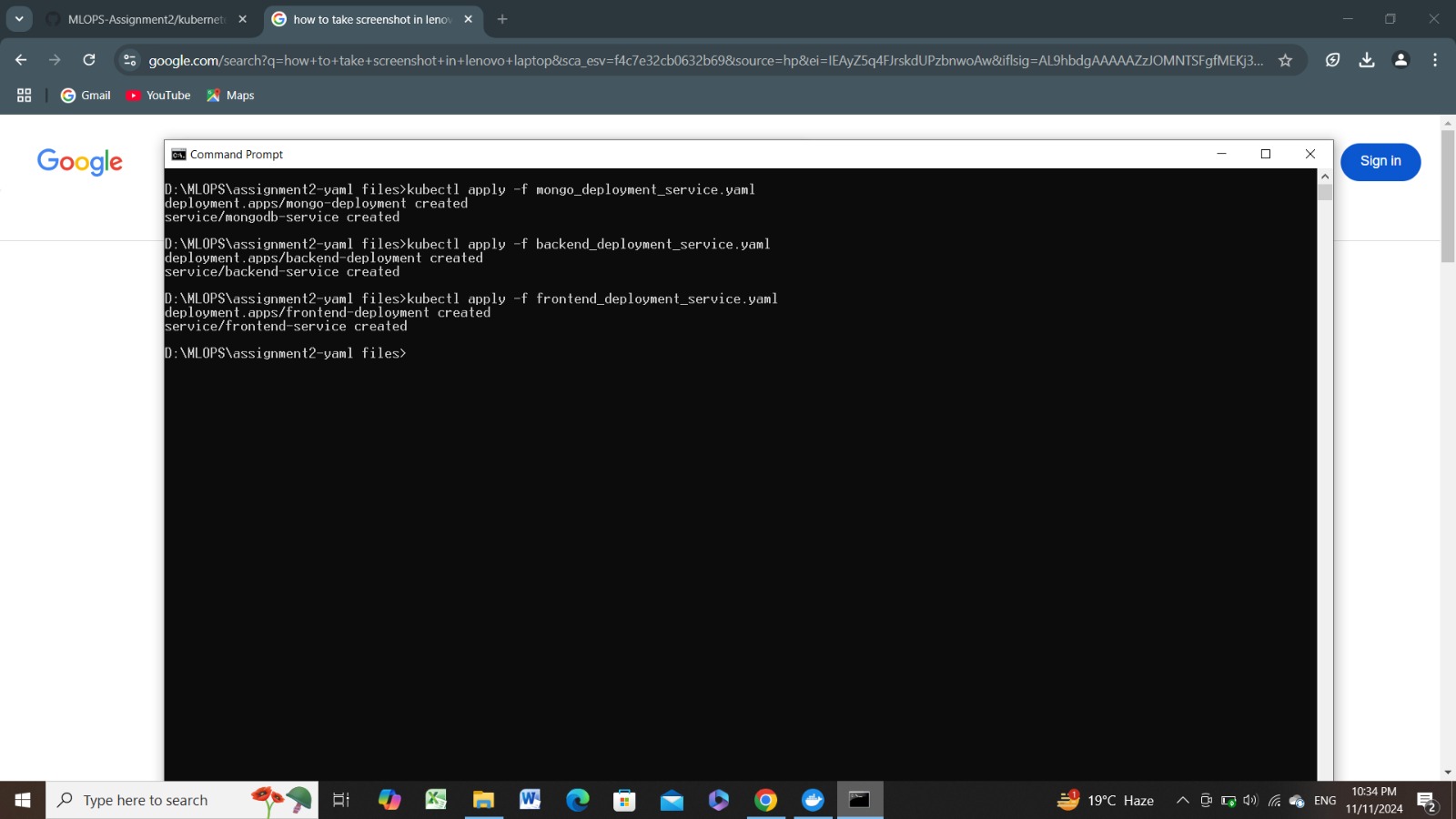
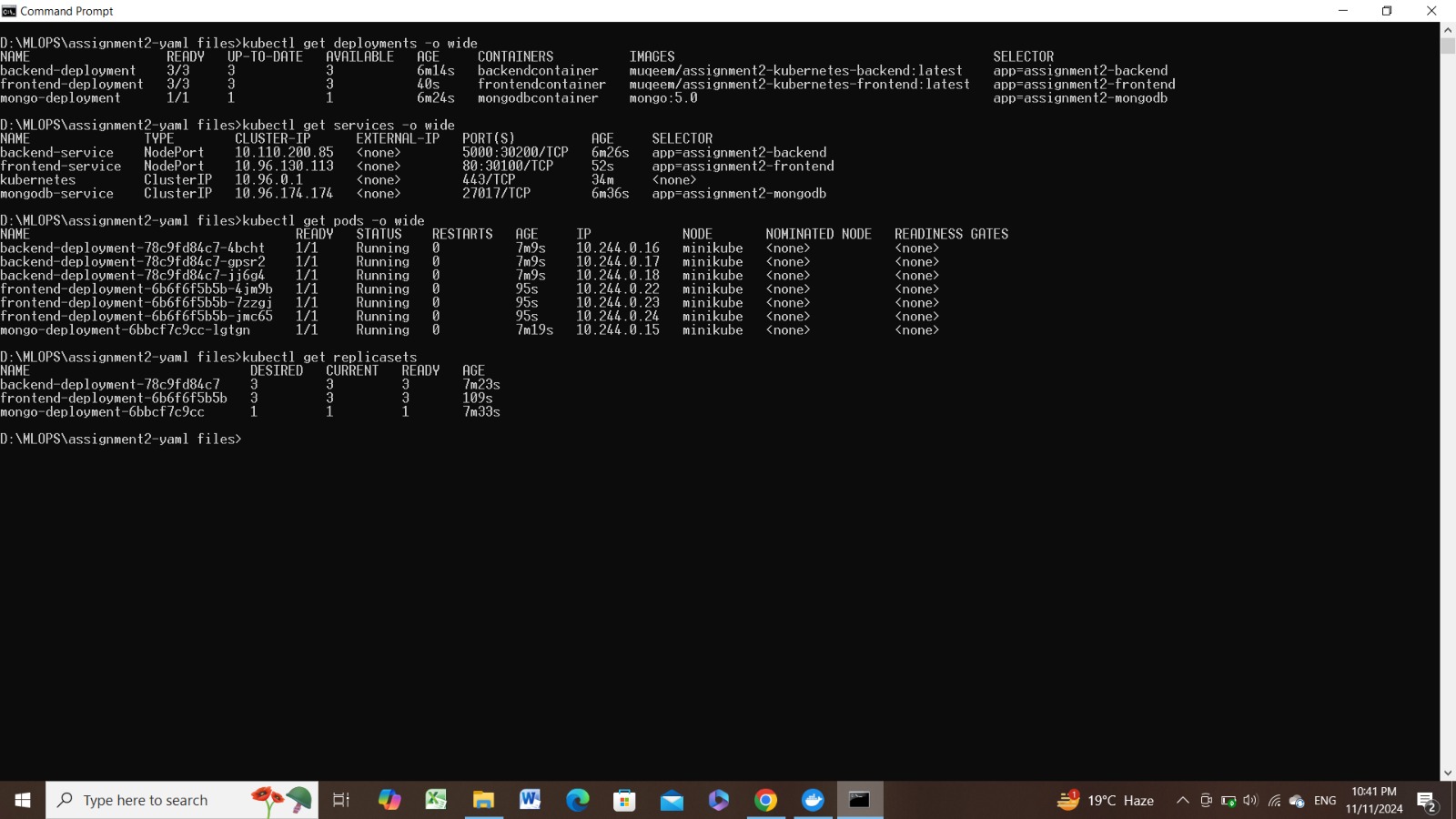
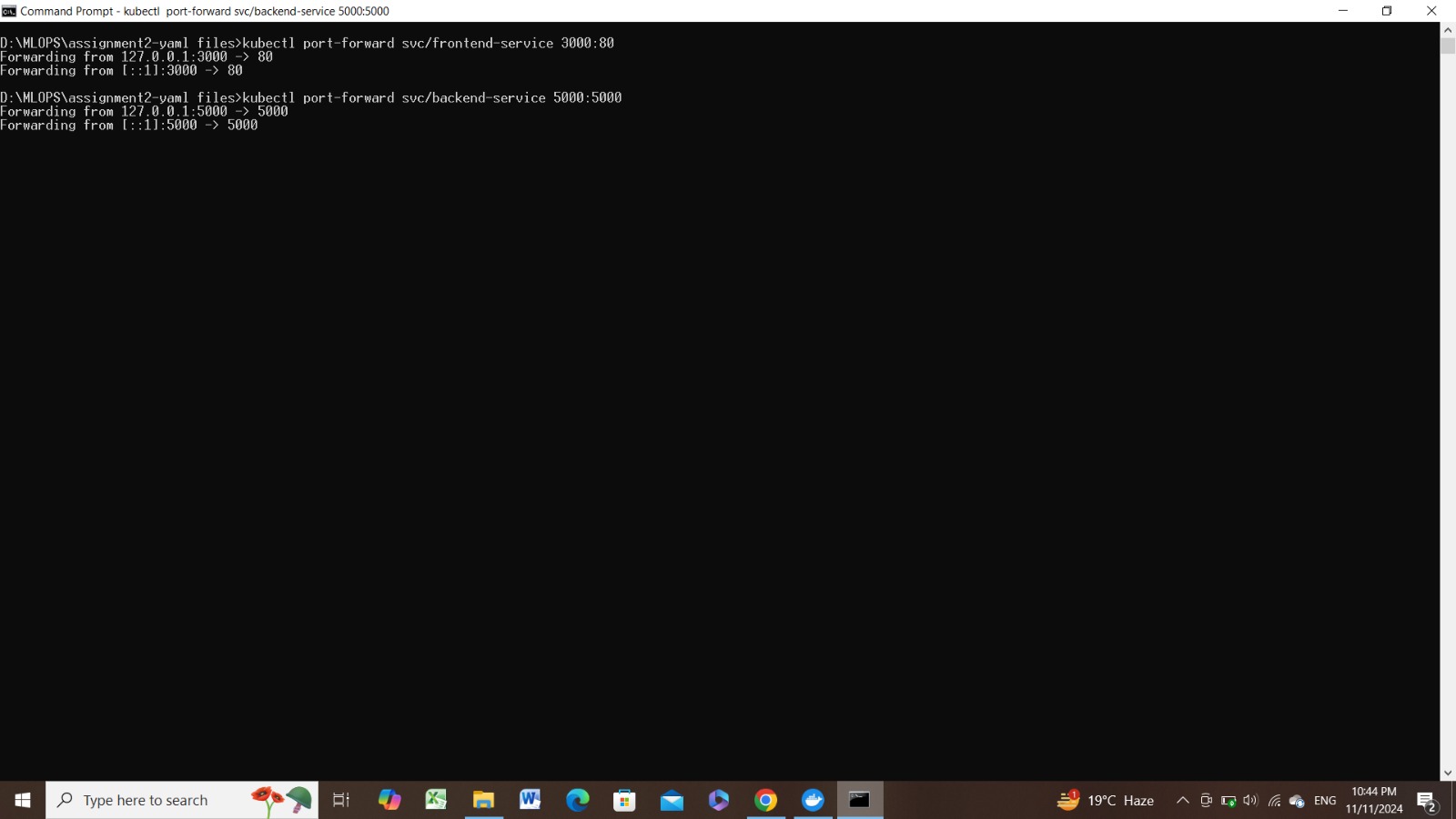
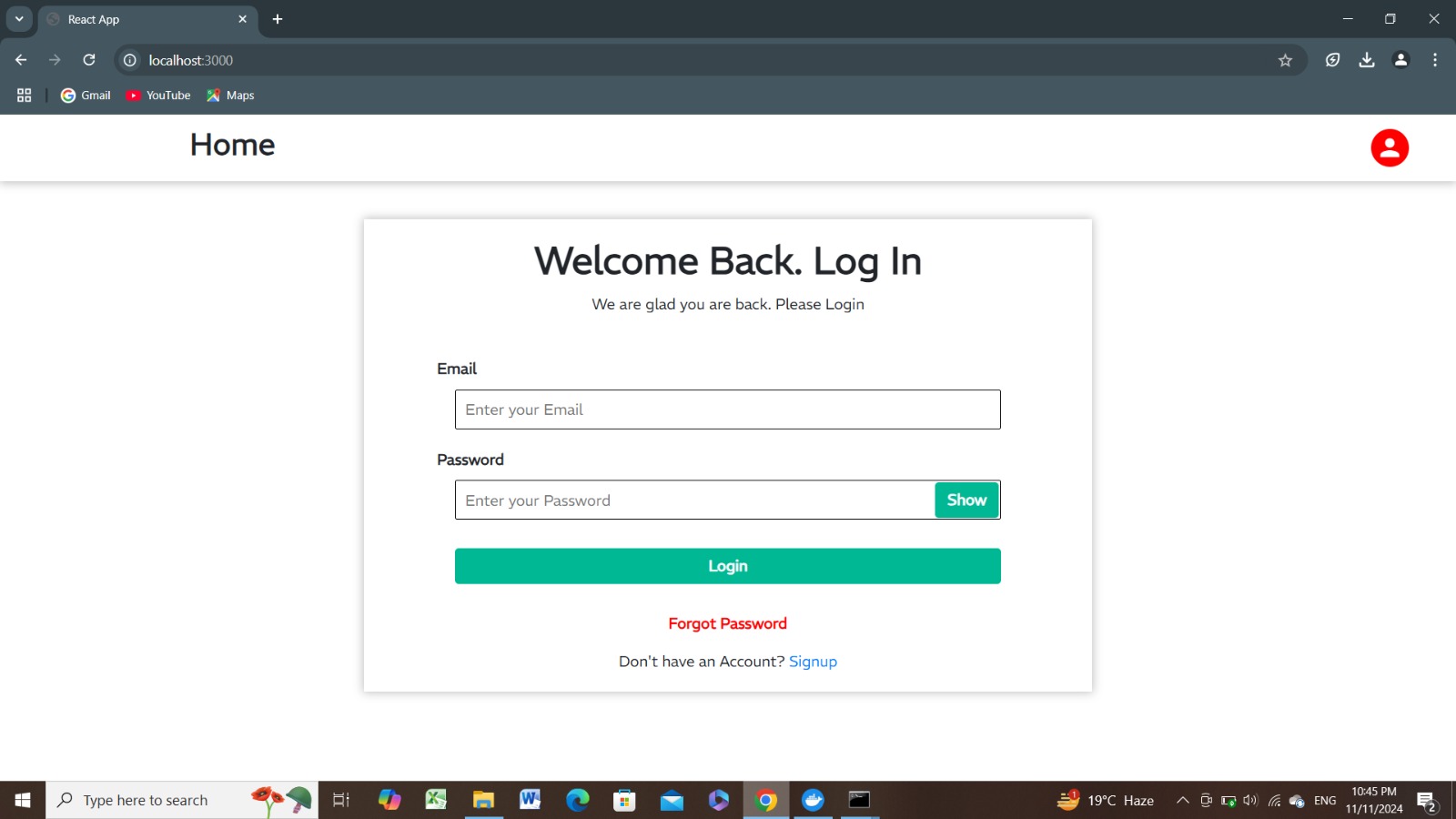
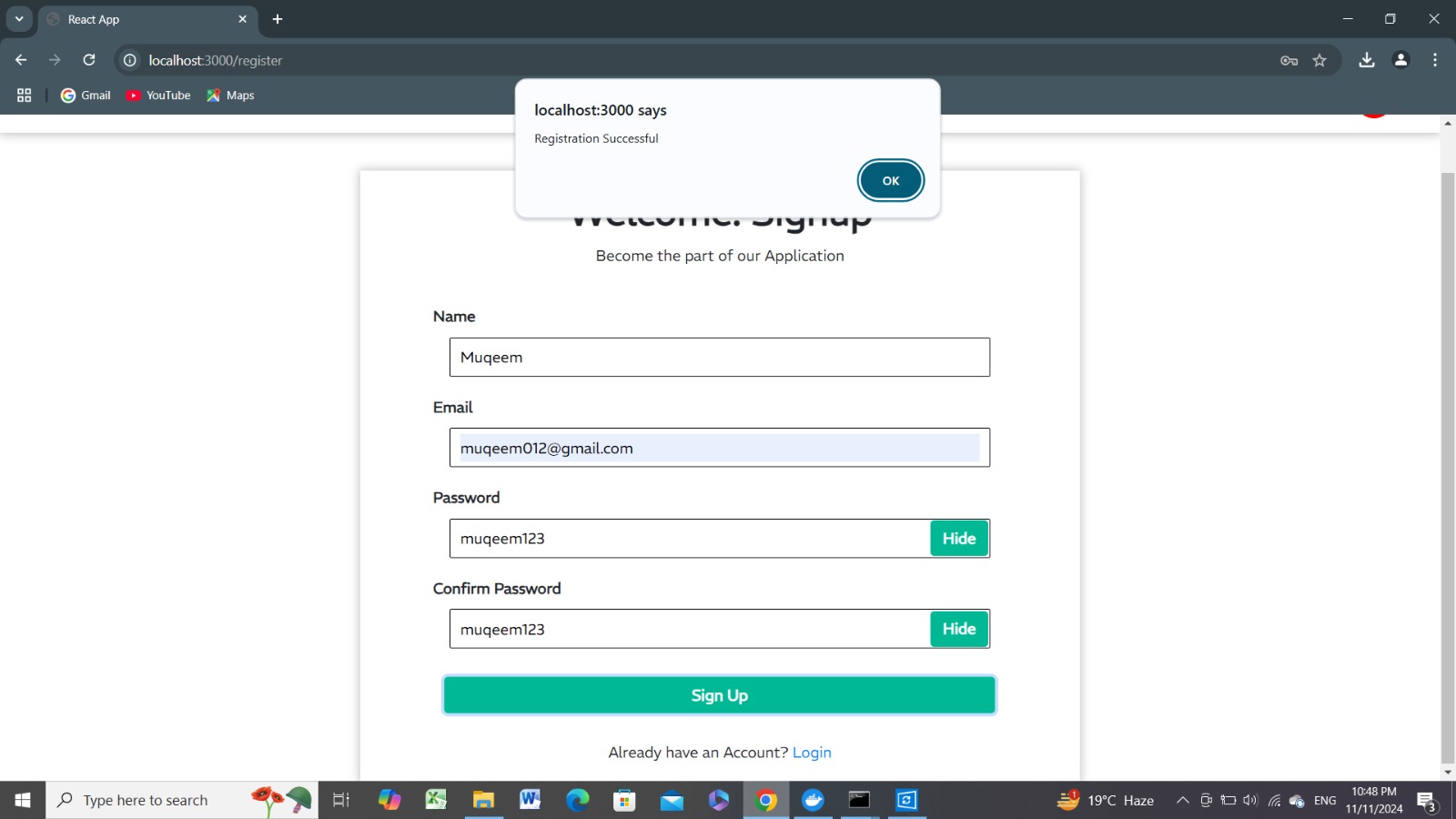
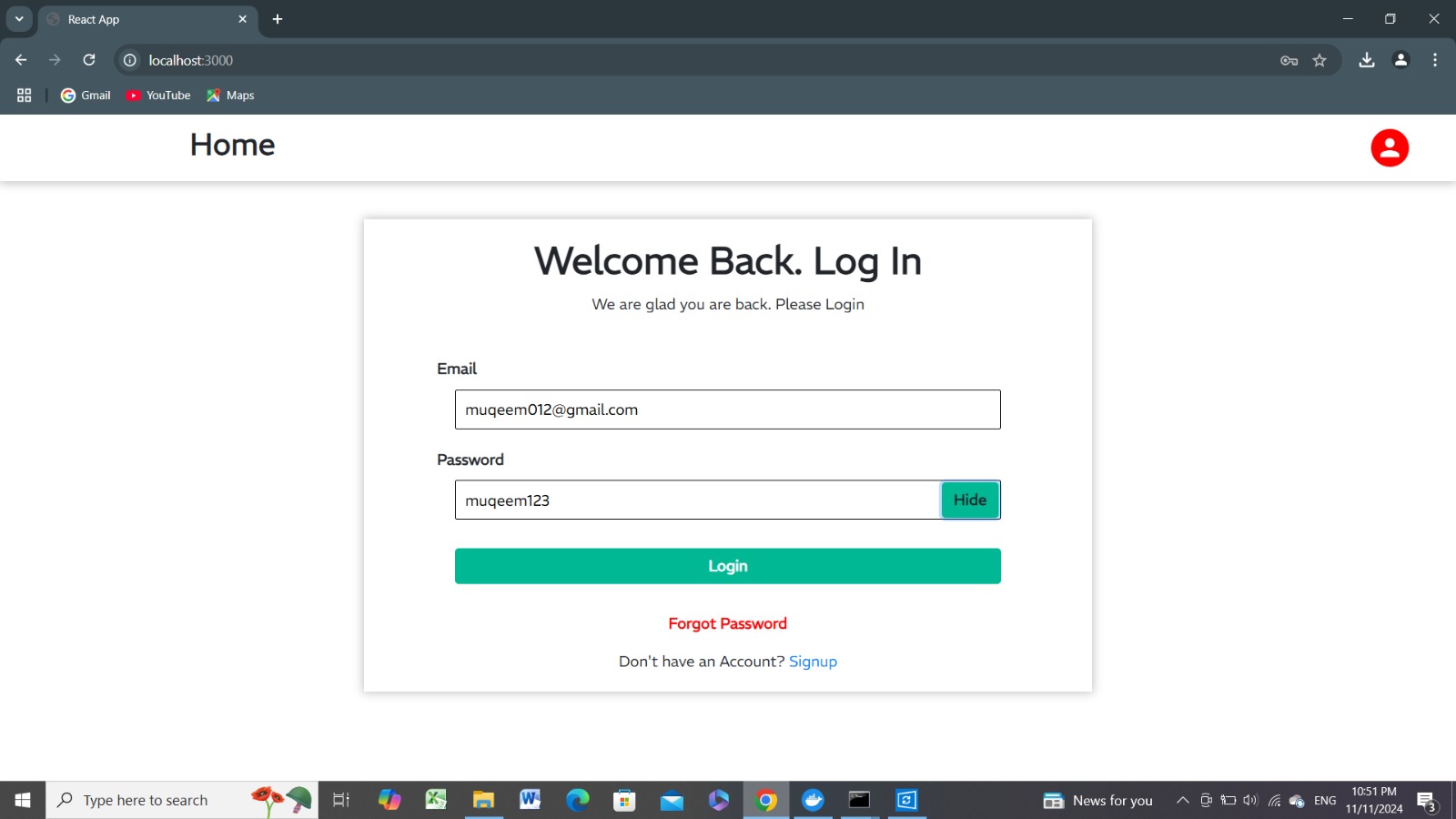
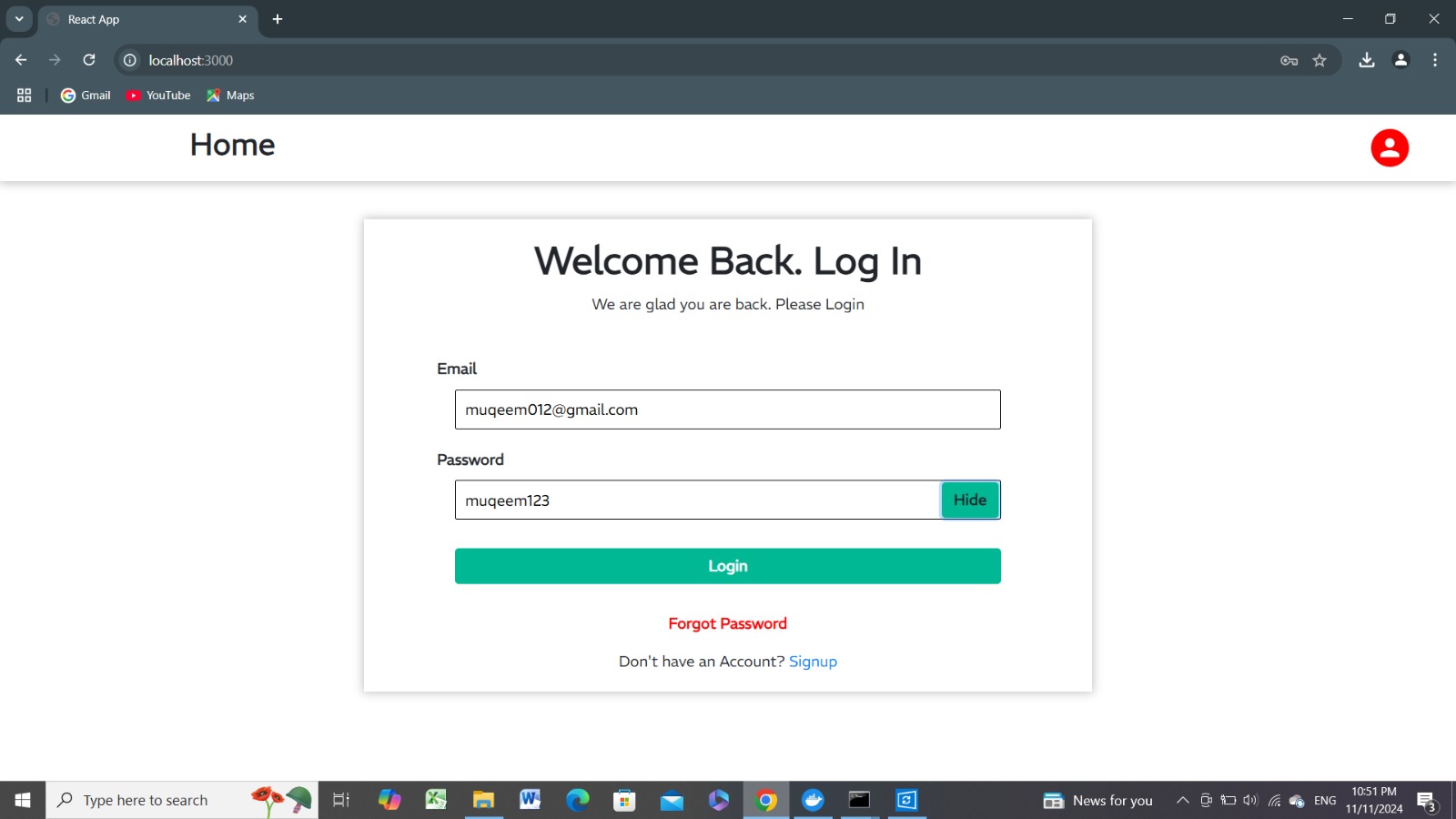
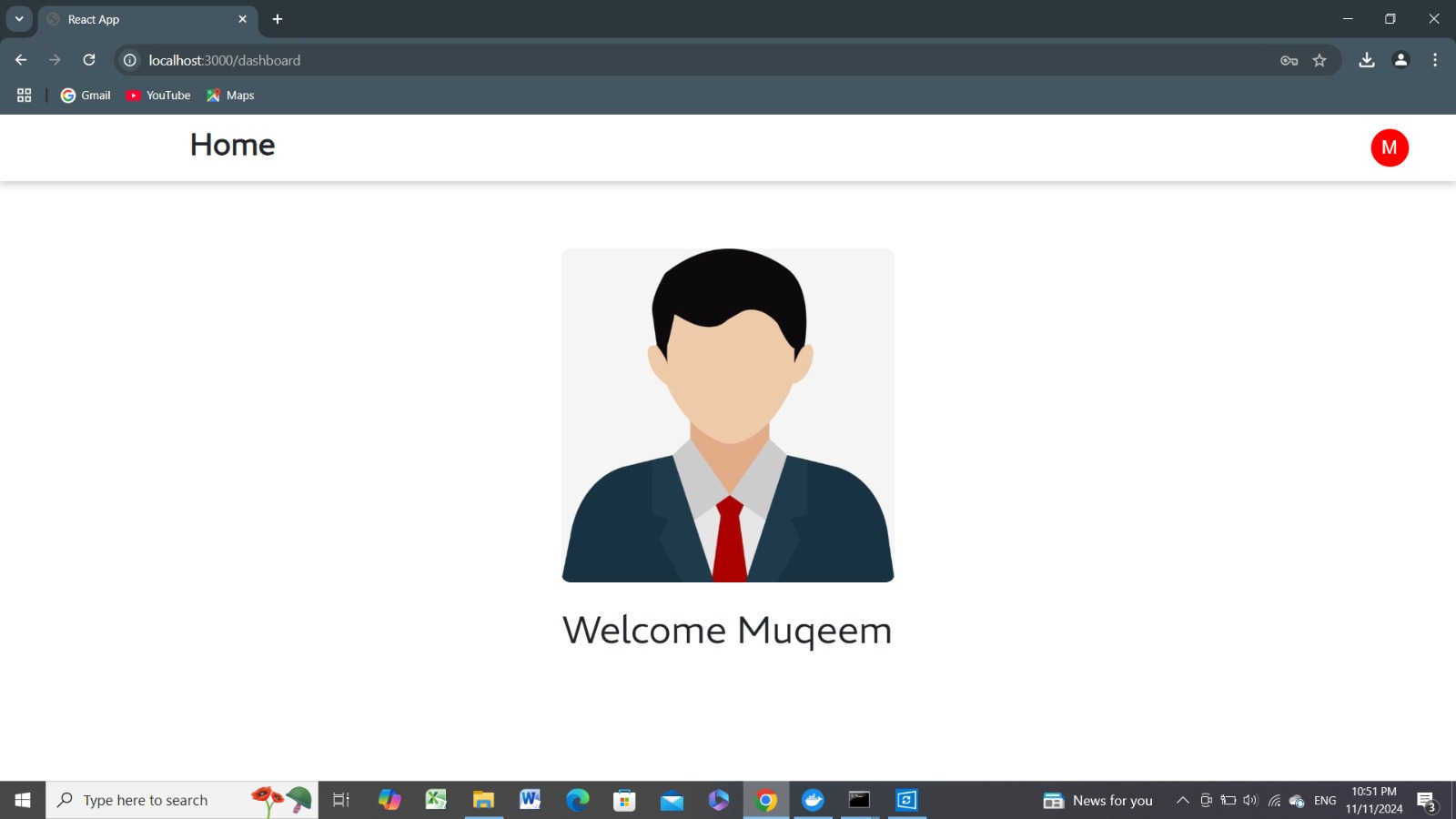
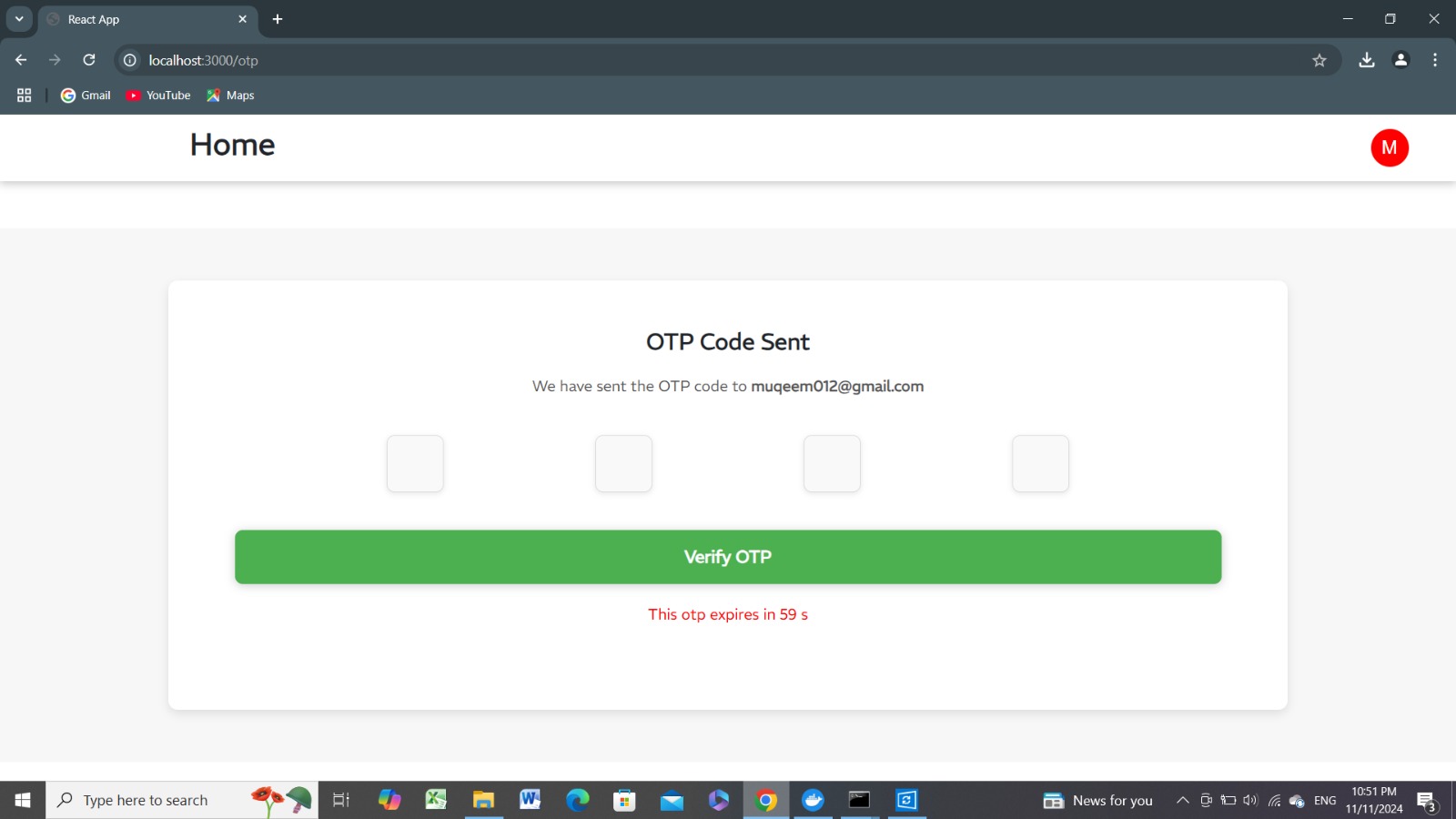
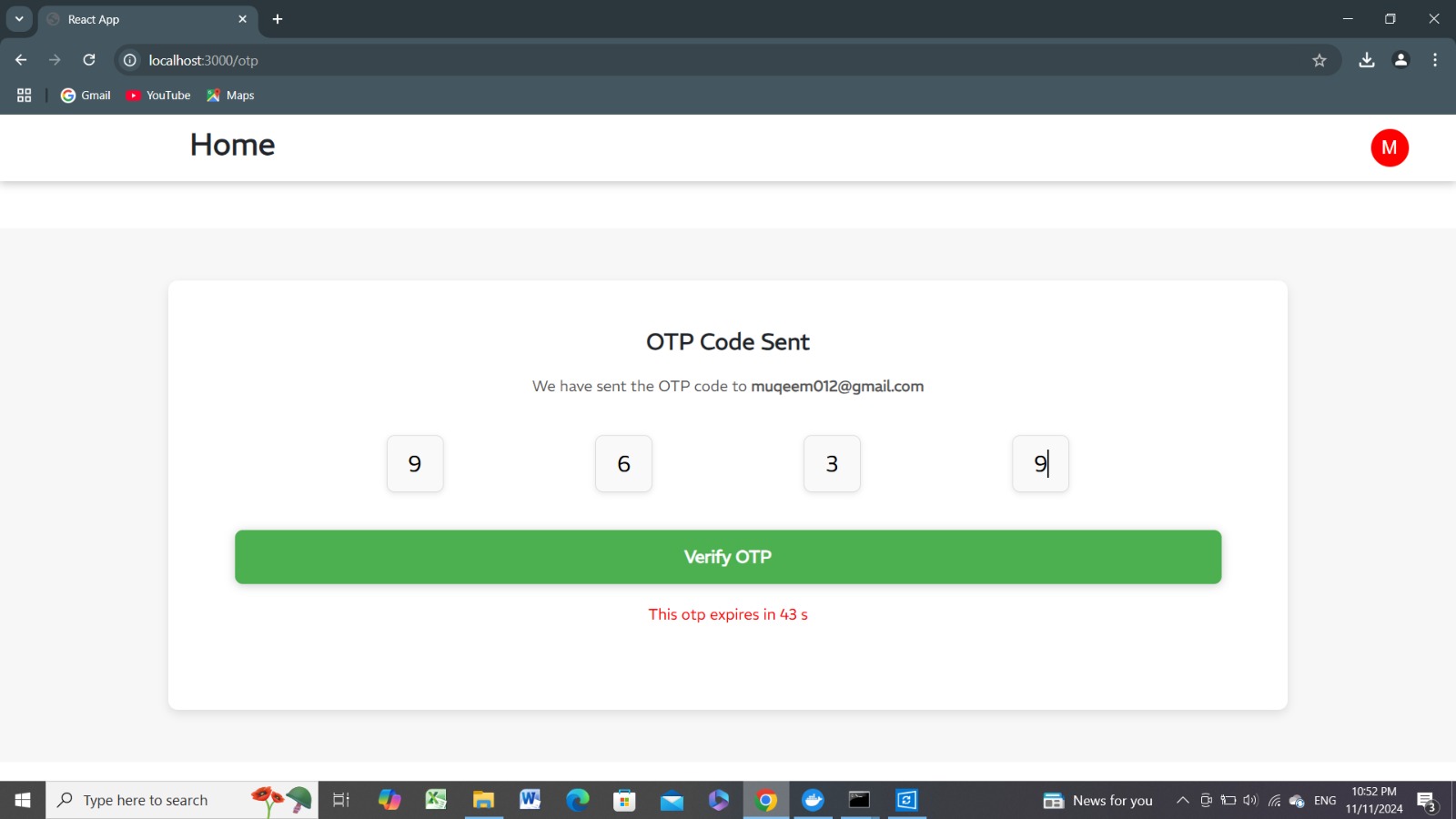
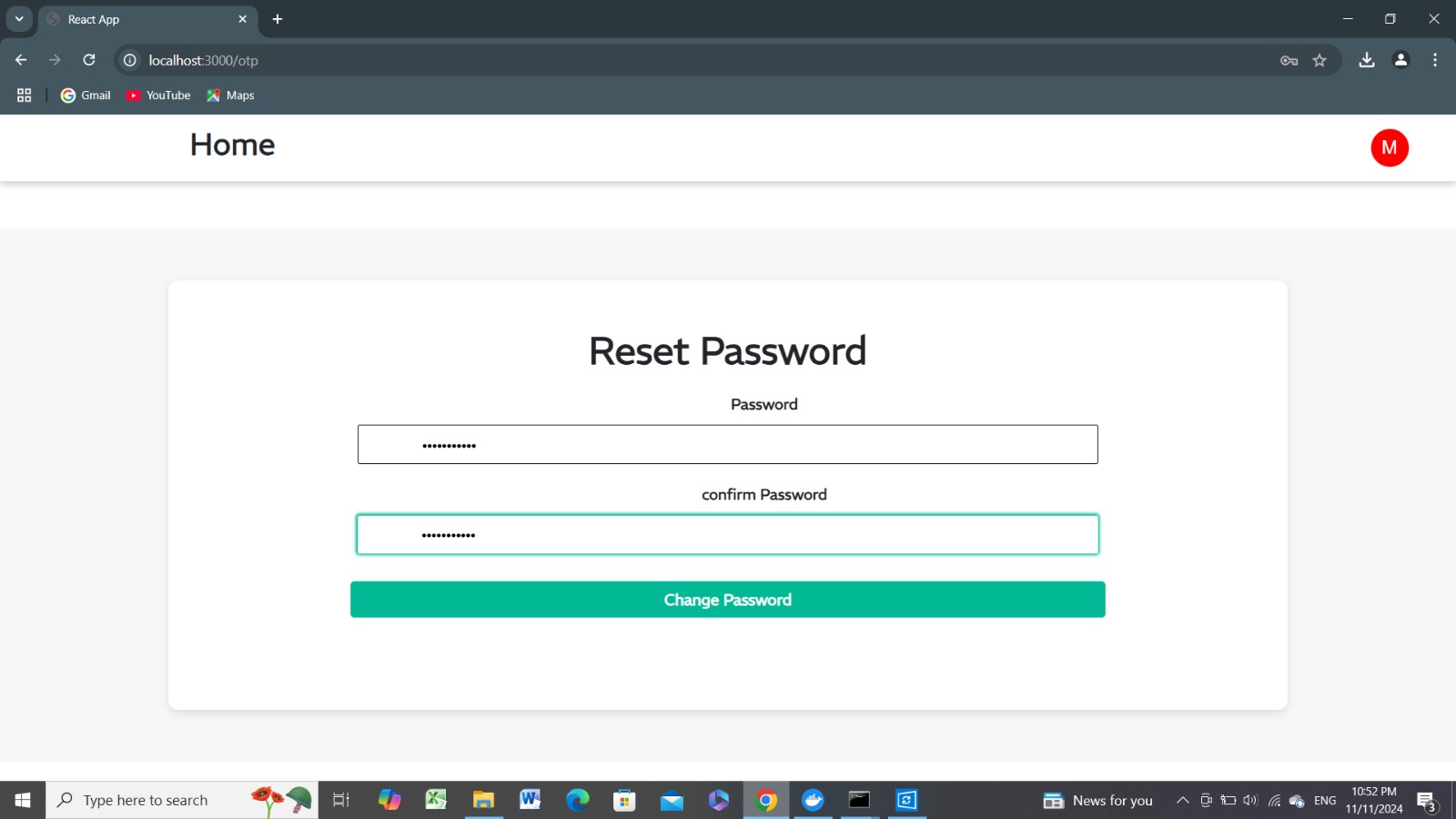
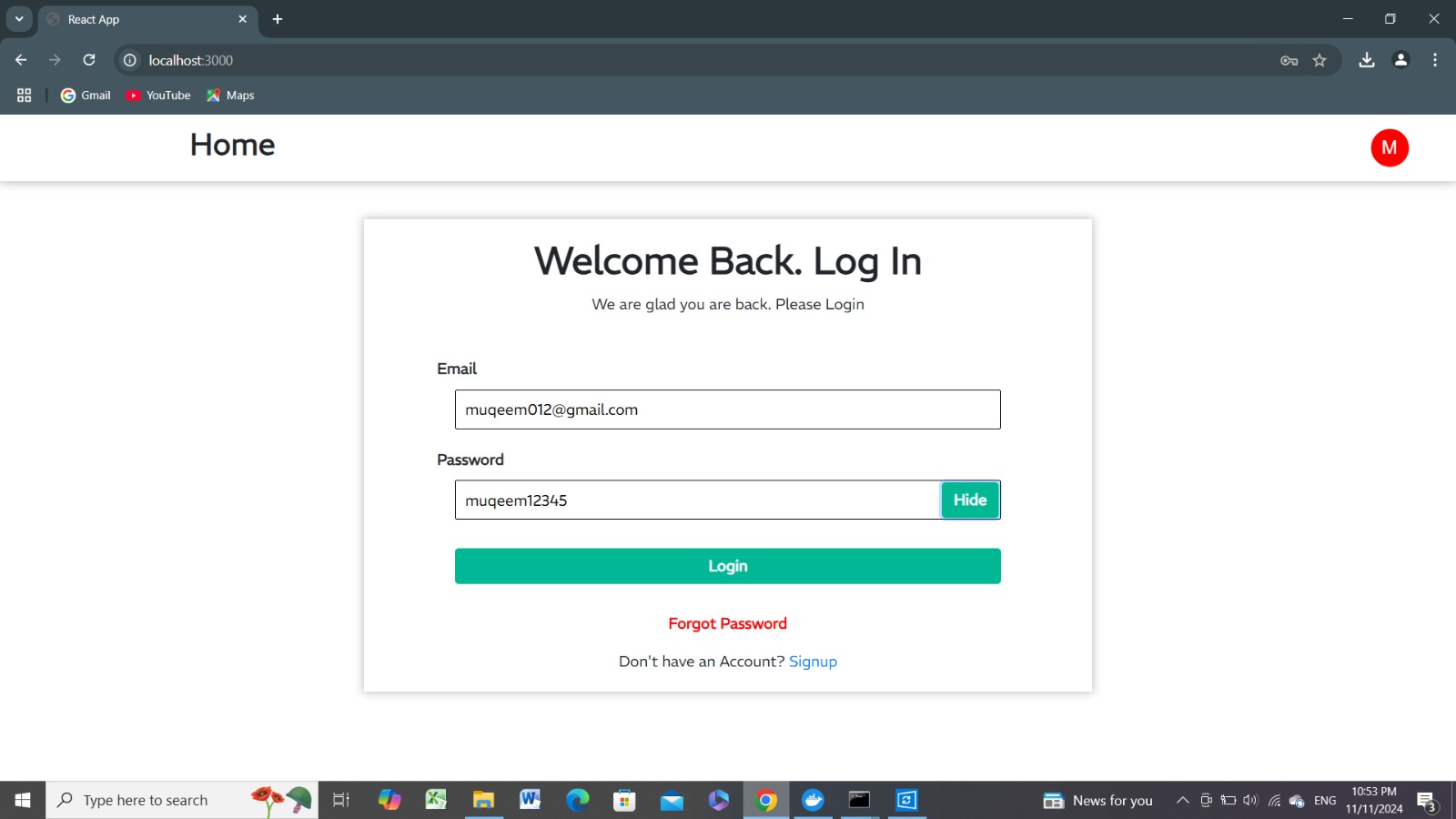
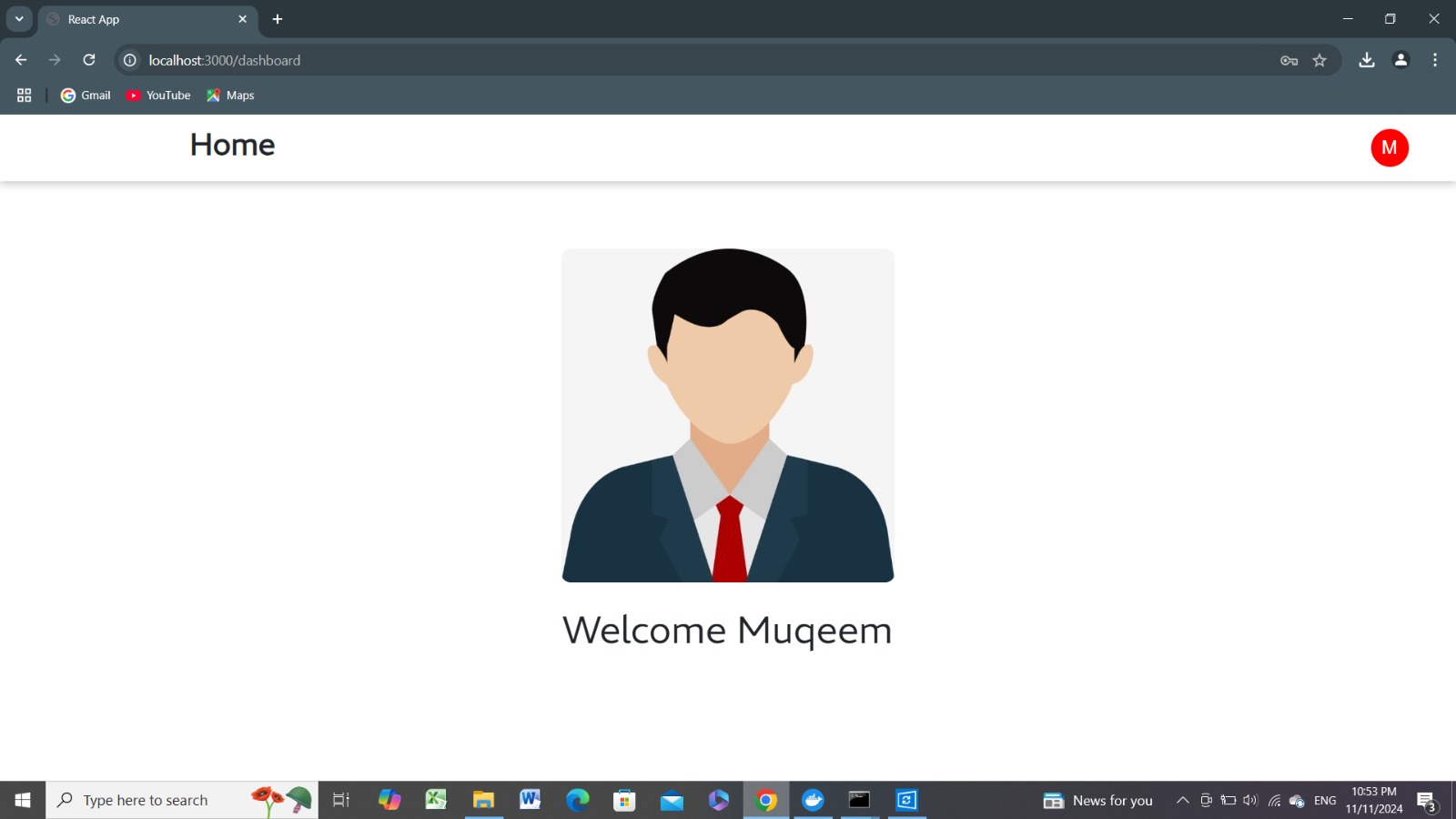
### 2. Kubernetes Deployment

Kubernetes enhances deployment by providing scalability, load balancing, and fault tolerance. Each component is deployed in its own set of pods, managed by NodePort and ClusterIP services for inter-service communication.

**Frontend Deployment:** Deploys 3 replicas of the frontend using Nginx, managed by NodePort on port 30100 (exposed to port 3000 in Minikube via port-forwarding).

**Backend Deployment:** Backend deployment with 3 replicas, accessed internally by frontend, exposed on port 5000 via port-forwarding for Minikube. MONGO\_URI connects to MongoDB via the 'mongodb-service' name.

**MongoDB Deployment:** Single-replica MongoDB with persistent volume, accessible via internal ClusterIP for backend communication.



## Access Instructions

**Docker Compose:** Access the frontend on http://localhost:3000. Backend and MongoDB communicate internally.

**Kubernetes:** Access the frontend via http://localhost:3000 (Minikube port-forwarded). Backend interacts internally with MongoDB via 'mongodb-service' URI.