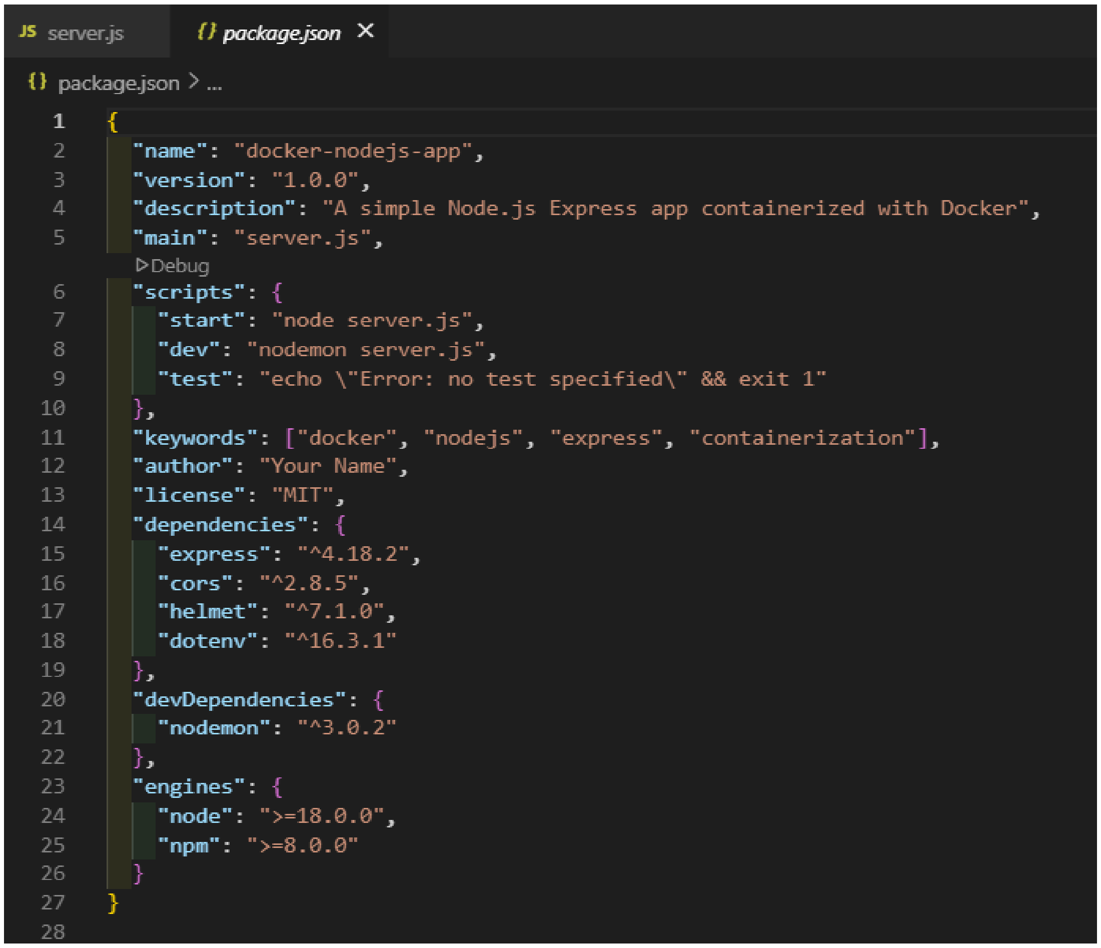
EXP. 9:Containerizing App with Docker

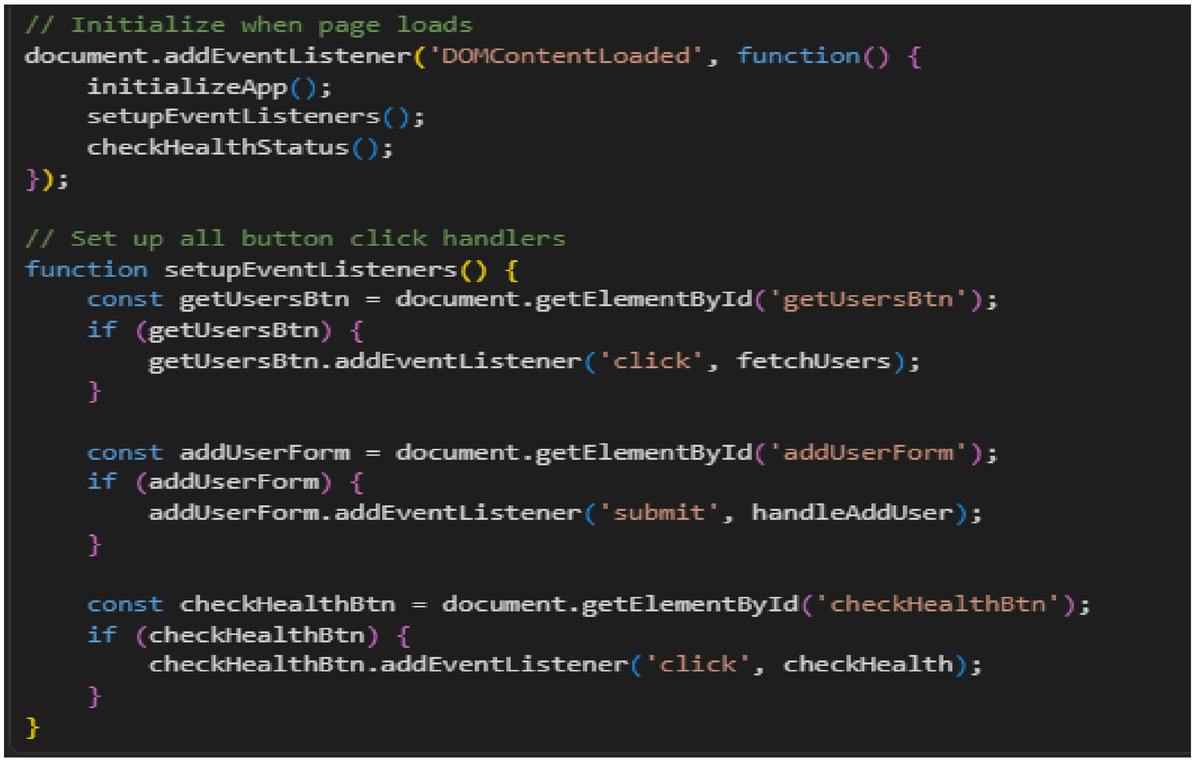
1) **App source (minimal Express app):**

Create a folder `myapp/` inside add these files

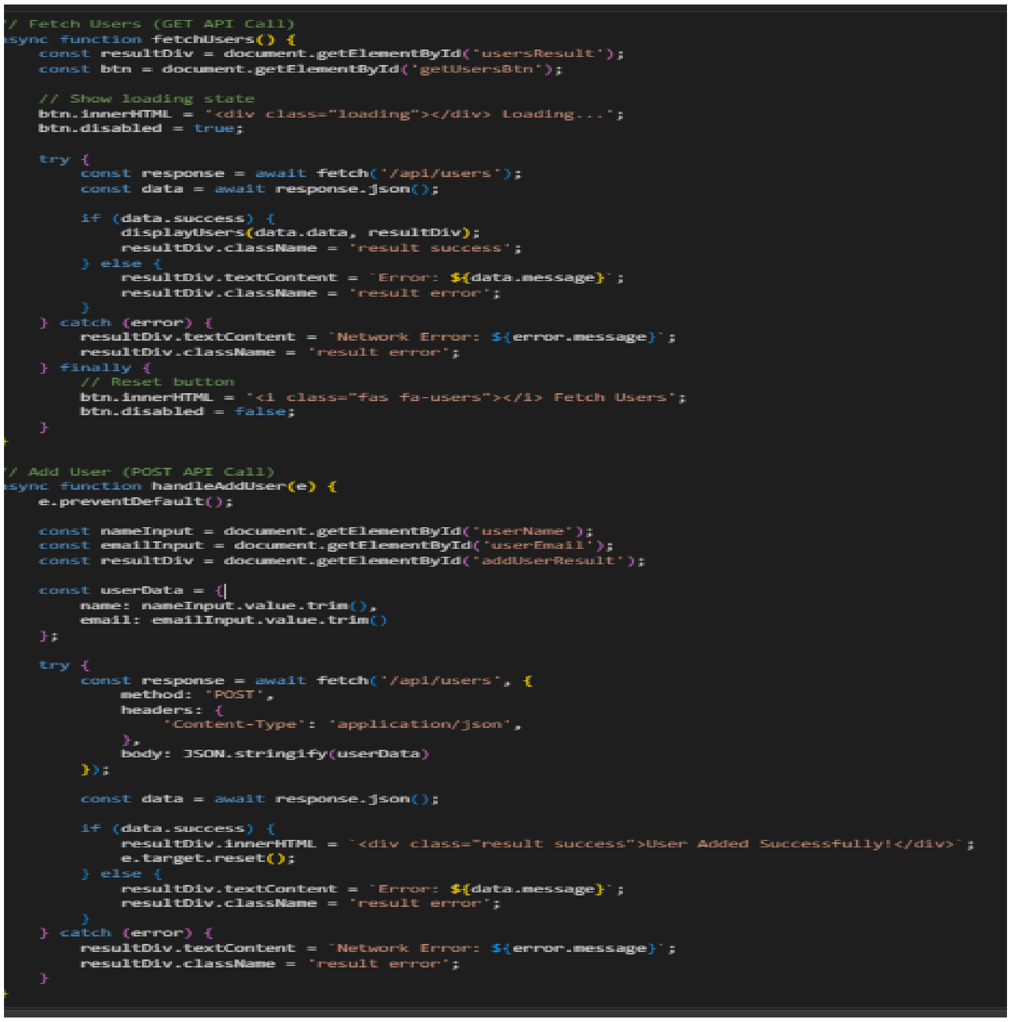


```json
{
  "name": "docker-nodejs-app",
  "version": "1.0.0",
  "description": "A simple Node.js Express app containerized with Docker",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": ["docker", "nodejs", "express", "containerization"],
  "author": "Your Name",
  "license": "MIT",
  "dependencies": {
    "express": "^4.18.2",
    "cors": "^2.8.5",
    "helmet": "^7.1.0",
    "dotenv": "^16.3.1"
  },
  "devDependencies": {
    "nodemon": "^3.0.2"
  },
  "engines": {
    "node": ">=18.0.0",
    "npm": ">=8.0.0"
  }
}
```

app.js:

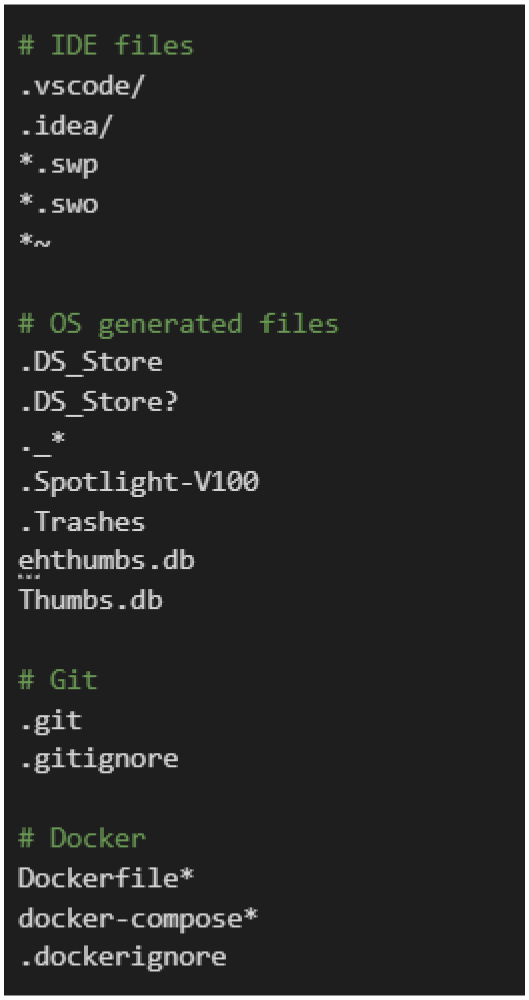


```
// Initialize when page loads
document.addEventListener('DOMContentLoaded', function() {
    initializeApp();
    setupEventListeners();
    checkHealthStatus();
});

// Set up all button click handlers
function setupEventListeners() {
    const getUsersBtn = document.getElementById('getUsersBtn');
    if (getUsersBtn) {
        getUsersBtn.addEventListener('click', fetchUsers);
    }

    const addUserForm = document.getElementById('addUserForm');
    if (addUserForm) {
        addUserForm.addEventListener('submit', handleAddUser);
    }

    const checkHealthBtn = document.getElementById('checkHealthBtn');
    if (checkHealthBtn) {
        checkHealthBtn.addEventListener('click', checkHealth);
    }
}
```

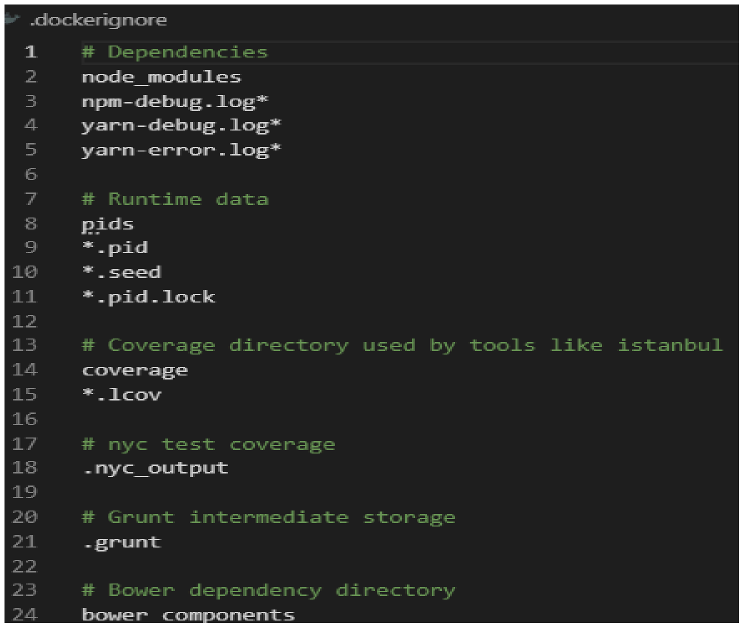


```
/ Fetch Users (GET API Call)
sync function fetchUsers() {
    const resultDiv = document.getElementById('usersResult');
    const btn = document.getElementById('getUsersBtn');

    // Show loading state
    btn.innerHTML = '<div class="loading"></div> Loading...';
    btn.disabled = true;

    try {
        const response = await fetch('/api/users');
        const data = await response.json();

        if (data.success) {
            displayUsers(data.data, resultDiv);
            resultDiv.className = 'result success';
        } else {
            resultDiv.textContent = `Error: ${data.message}`;
            resultDiv.className = 'result error';
        }
    } catch (error) {
        resultDiv.textContent = `Network Error: ${error.message}`;
        resultDiv.className = 'result error';
    } finally {
        // Reset button
        btn.innerHTML = '<i class="fas fa-users"></i> Fetch Users';
        btn.disabled = false;
    }

/ Add User (POST API Call)
sync function handleAddUser(e) {
    e.preventDefault();

    const nameInput = document.getElementById('userName');
    const emailInput = document.getElementById('userEmail');
    const resultDiv = document.getElementById('addUserResult');

    const userData = {
        name: nameInput.value.trim(),
        email: emailInput.value.trim()
    };

    try {
        const response = await fetch('/api/users', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify(userData)
        });

        const data = await response.json();

        if (data.success) {
            resultDiv.innerHTML = `<div class="result success">User Added Successfully!</div>`;
            e.target.reset();
        } else {
            resultDiv.textContent = `Error: ${data.message}`;
            resultDiv.className = 'result error';
        }
    } catch (error) {
        resultDiv.textContent = `Network Error: ${error.message}`;
        resultDiv.className = 'result error';
    }
```

2) .dockerignore:



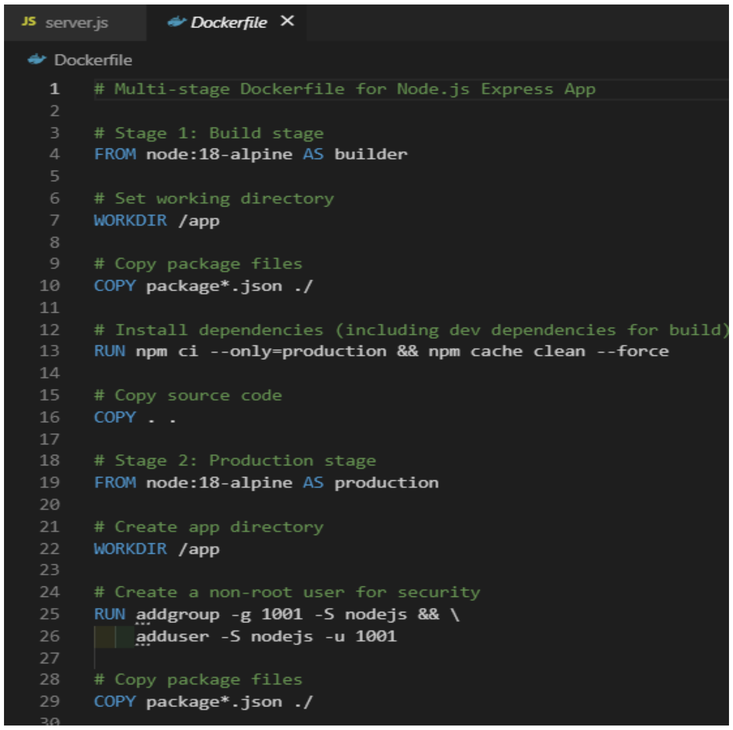
```
# IDE files
.vscode/
.idea/
*.swp
*.swo
*~

# OS generated files
.DS_Store
.DS_Store?
._*
.Spotlight-V100
.Trashes
ehthumbs.db
Thumbs.db

# Git
.git
.gitignore

# Docker
Dockerfile*
docker-compose*
.dockerignore
```



.dockerignore

```
# Dependencies
node_modules
npm-debug.log*
yarn-debug.log*
yarn-error.log*

# Runtime data
pids
*.pid
*.seed
*.pid.lock

# Coverage directory used by tools like istanbul
coverage
*.lcov

# nyc test coverage
.nyc_output

# Grunt intermediate storage
.grunt

# Bower dependency directory
bower_components
```

3) Dockerfile -Multi-stage,(dev/prod friendlily)

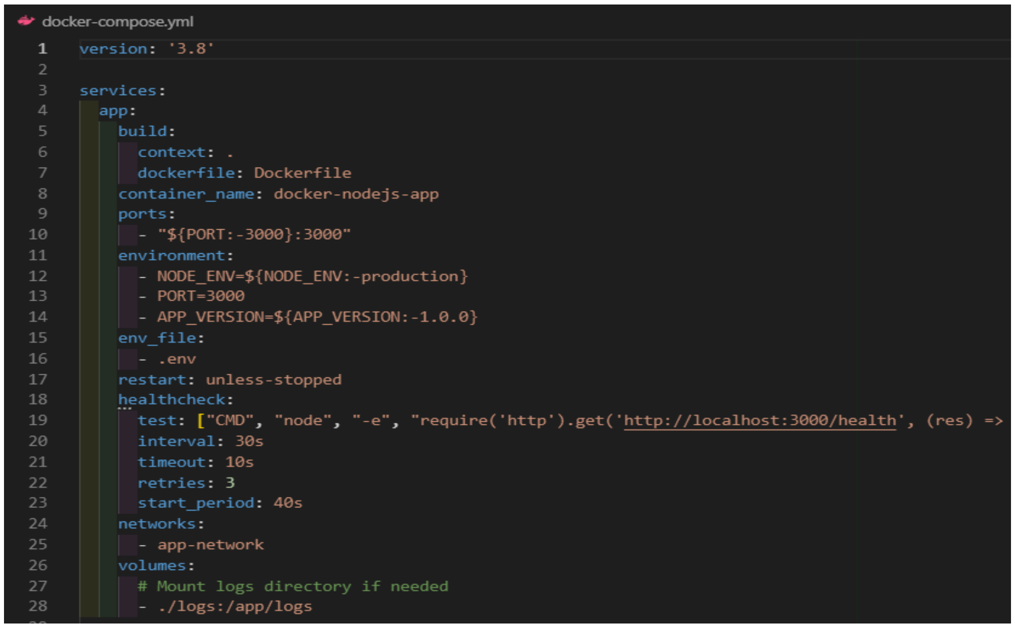


```dockerfile
# Multi-stage Dockerfile for Node.js Express App

# Stage 1: Build stage
FROM node:18-alpine AS builder

# Set working directory
WORKDIR /app

# Copy package files
COPY package*.json ./

# Install dependencies (including dev dependencies for build)
RUN npm ci --only=production && npm cache clean --force

# Copy source code
COPY . .

# Stage 2: Production stage
FROM node:18-alpine AS production

# Create app directory
WORKDIR /app

# Create a non-root user for security
RUN addgroup -g 1001 -S nodejs && \
    adduser -S nodejs -u 1001

# Copy package files
COPY package*.json ./
```

4) `docker-compose.yml` (optional, for local dev)

Create `docker-compose.yml` to run locally with environment overrides and volume mount for live reload (dev):

For production, remove `volumes` and use the baked-in CMD.



```yaml
version: '3.8'

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: docker-nodejs-app
    ports:
      - "${PORT:-3000}:3000"
    environment:
      - NODE_ENV=${NODE_ENV:-production}
      - PORT=3000
      - APP_VERSION=${APP_VERSION:-1.0.0}
    env_file:
      - .env
    restart: unless-stopped
    healthcheck:
      test: ["CMD", "node", "-e", "require('http').get('http://localhost:3000/health', (res) =>
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s
    networks:
      - app-network
    volumes:
      # Mount logs directory if needed
      - ./logs:/app/logs
```

5) Build & run (com1mands)

From `myapp/` directory:

Buiild image:

```
# build using DockerfiLe
docker build -t myapp:latest
```

Run container:

```
docker run --rm -p 3000:3000 \
  -e GREETING="Hello Docker"\
  --name myapp-derno myapp:latest
```

Using docJQer-compose tdev):

```
docker-compose up --build
# stop i..,ith Ctrl+C or in another terminal:
# docker-compose down
```

Verify:

- Open `http://localhost:3000/` – you should see the ,greeting.
- Health: `curl` http://localhost:3000/health.

6) Tag1& push to Doclker Hub (optional):

```
# Login
docker- login

# tag (replace USER with your dockerhub username)
docker- tag rnyapp: latest YOI.JR_OOCKERHLrn_USER/rnyapp:1 ..0. .0

# push
docker- push YOUR_DOCK ERHUB_USER/ my app : 1. 0..0.
```

6) Extra best practices & tips:

- Use . env files for secrets in dev (with docker-compose and env_file), but **never** commit secrets to git.
- Use a non-root user in the image (example demonstrates it).
- Keep image s'1im: prefer node:18-alpine or distroless images.
- Pin base images to a specific version for reproducibility.
- Security scan images (e.g., docker scan).
- Fairproduction orchestration use Ku'bernetes, ECS, or similar - deploy the pushed image there.
- lfyourapp needs file persistence, mount vo'lumes (-v) or use external

  storage; containers should be ephemeral.