

User Management System (Multi-Tier Architecture)

We will build a User Management System with the following core features:

- ✓ User login
- ✓ User dashboard to list all users.
- ✓ User Add/Edit/Delete operations

This solution will be multi-tier architecture

Tier 1: Frontend (Angular)

- ✓ Developed using Angular
- ✓ Key pages:
 - Login Page
 - User Dashboard With user's list and Add, Edit and Delete button
 - Add/Edit page
- ✓ Authenticated via UserId + Password
- ✓ Connects to backend via HTTP services
- ✓ Dockerized for containerized deployment
- ✓ Will be accessible on <http://localhost:8080>

Tier 2: Backend API (Node.js + Express)

- ✓ RESTful API built using Node.js and Express
- ✓ Validate user credentials for login and business logic
- ✓ Routes for:
 - POST /login - user authentication
 - POST /add - create user
 - GET /users - list all users
 - GET /users:{id} – fetch user by id
 - DELETE /delete/:id - delete user by id
- ✓ Dockerized for containerized deployment
- ✓ Will be expose on <http://localhost:3000>

Tier 3: Database (MongoDB)

- ✓ Stores user information and roles
- ✓ Sample schema:
 - User: { userid, password (hashed), userType, isActive, id }
- ✓ Dockerized MongoDB container for containerized deployment
- ✓ Will be expose on <http://localhost:27017>
- ✓ Redis Caching implementation, caching expiration time is 60 seconds.
- ✓ If any Add/Delete action taken, then dashboard will load from database and cache will be refresh.

Security Implementation

1. All Docker images are officially verified on Docker hub



mongo

Docker Official Image • 1B+ • 10K+

MongoDB document databases provide high availability and easy scalability.

DATABASES & STORAGE

Overview

Tags

Sort by

Newest

8.0



redis

Docker Official Image • 1B+ • 10K+

Redis is the world's fastest data platform for caching, vector search, and NoSQL databases.

DATABASES & STORAGE

Overview

Tags

Quick reference

- Maintained by:
[Redis LTD](#)
- Where to get help:
[the Docker Community Slack](#) , [Server Fault](#) , [Unix & Linux](#) , or [Stack Overflow](#)

Supported tags and respective Dockerfile links

• **8.2.0** [8.2](#) [8](#) [8.2.0-bookworm](#) [8.2-bookworm](#) [8-bookworm](#) [latest](#) [bookworm](#)

2. I used small sized images specified over docker hub
3. Added a .dockerignore file in both solutions, kindly refer git repositories.

4. I have use specific image version

```
mongodb:
  image: mongo:8.0
  container_name: mongodb
  environment:
    MONGO_INITDB_ROOT_USERNAME: ${MONGO_UID}
    MONGO_INITDB_ROOT_PASSWORD: ${MONGO_PWD}
  ports:
    - "27017:27017"
  volumes:
    - mongo-data:/data/db
  networks:
    - app-network

cache-service:
  image: redis:8.2.0
  container_name: cache-service
  command: redis-server --appendonly yes --save
  ports:
    - "6379:6379"
  volumes:
    - redis-data:/data
  networks:
    - app-network
```

5. Optimized caching layers by Docker file command orders
6. Multistage build has been implemented by correct ordering on command in Docker file

```
er-compose.yml Dockerfile Dockerfile .env.deve .env.prod

# Step 1: Build Angular app
FROM node:20 AS builder

WORKDIR /app
COPY package.json package-lock.json ./
RUN npm install

COPY . .
RUN npx ng build --configuration=production

# Step 2: Serve with Nginx
FROM nginx:alpine

# Remove default nginx static files
RUN rm -rf /usr/share/nginx/html/*

# Copy built app to nginx web directory
COPY --from=builder /app/dist/users-app/browser /usr/share/nginx/html

# Optional: Add custom nginx config (if needed)
COPY nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Application Features

- ✓ Login/Auth flow using JWT
- ✓ User CRUD from UI with REST API integration

Docker Features

- ✓ Docker Compose setup for full application configuration and run container.
- ✓ Docker common Network for service communication to each other, refer above screen shot.
- ✓ Docker Volume to persist data for database and caching too, refer above screen shot.