GROUP 6

Final Project Containerization Technologies

1. HUGO BONNELL
2. COLIN TANTER
3. IBRAHIM SEMMACHE
4. NGOULAYE KEUNGUEU

Sommaire

Introduction

Solution

- Create your household, invite your roommates
- Shared calendar
- Assign chores
- Set how often they repeat
- Track what’s done… and what’s not

Demonstration

Technologies

Tech stack overview

- **Frontend:** Angular 18 for single-page application with responsive UI and hot reload
- **API Servers:** Express.js (Node.js) for both User and Task microservices
- **Database:** MySQL 8 databases with separate instances for each service
- **ORM:** Sequelize for type-safe database interactions and migrations
- **Container Platform:** Docker with Docker Compose for development and deployment
- **Authentication:** JWT (JSON Web Tokens) for secure, stateless authentication

ORM = Object Relation Mapping | Programming technique that converts data between incompatible type systems in object-oriented programming languages and relational databases. Here TS → SQL.

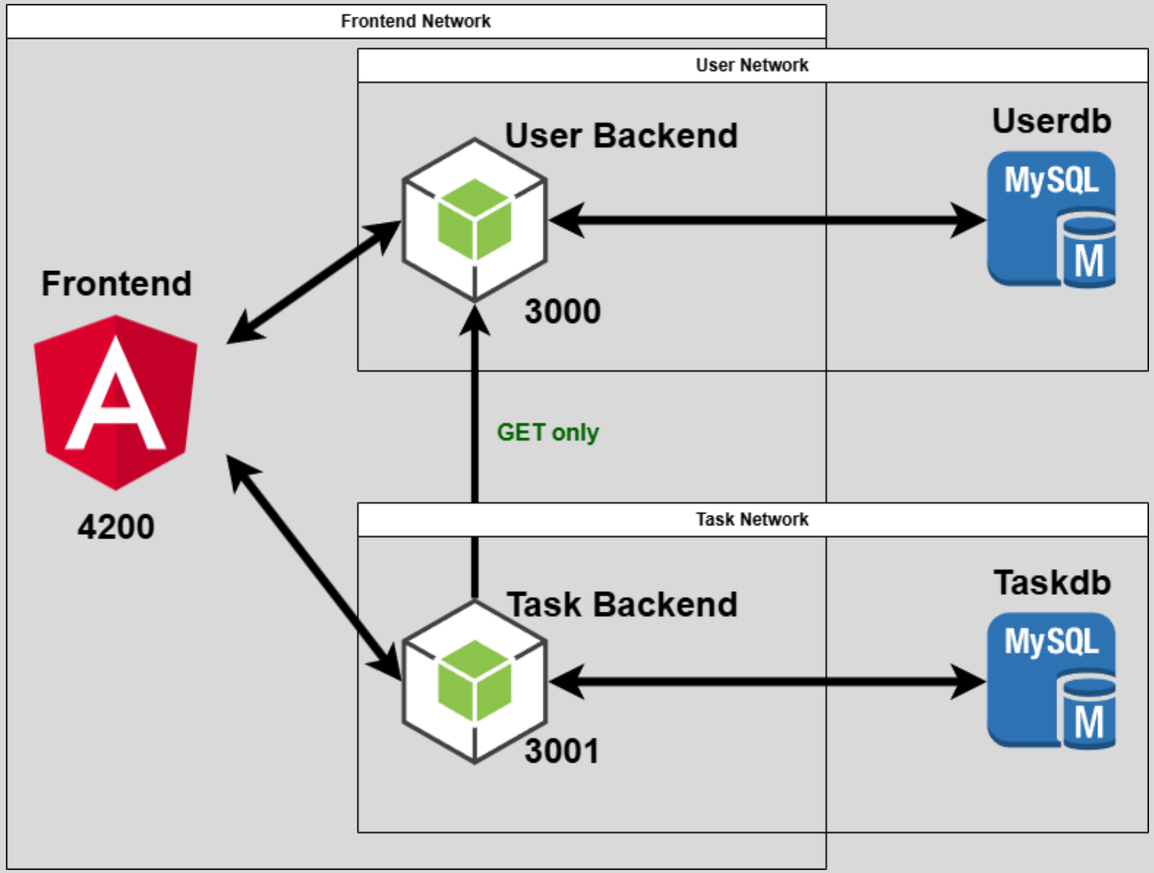
Technology Integration

- **Service Communication:** REST APIs with JWT authentication between services
- **Data Flow:** Angular → Express microservices → Sequelize → MySQL
- **Development Workflow:**
  - TypeScript across entire stack
  - Hot-reload enabled for rapid development
  - Shared JWT secret enables seamless authentication across services
- **UI Components:** Combination of custom Angular components and Bootstrap for responsive design
- **Database Design:** Relational model with clear service boundaries and minimal cross-service dependencies

Architecture

ROOMIE TASKS

Network overview



- **Microservices Architecture:** Three separate services (Frontend, User Service, Task Service) with dedicated databases
- **Containerized Deployment:** Docker containers for each service enable isolation and easy scaling
- **Three Isolated Networks:**
  - Frontend network (connects UI with both services)
  - User service network (user service + user database)
  - Task service network (task service + task database)
- **Persistent Data Storage:** Named volumes ensure data survives container restarts

ROOMIE TASKS

Network architecture benefits

- **Security:** Database servers are only accessible from their respective services
- **Reduced Attack Surface:** Service-to-service communication limited to necessary interactions
- **Fault Isolation:** Issues in one service don't directly impact others
- **Independent Scaling:** Each service can be scaled according to specific demand
- **Simplified Development:** Teams can work on separate services with minimal conflicts
- **Easier Maintenance:** Updates or replacements can be done service by service

Docker

Dockerfiles

Frontend

```
# Development image with hot-reload
FROM node:18-alpine as build

WORKDIR /app

# Copy package.json and package-lock.json
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy the rest of the application
COPY . .

# Start dev server with host 0.0.0.0 to allow external access
CMD ["npm", "start", "--", "--host", "0.0.0.0"]

# Expose Angular dev server port
EXPOSE 4200
```

User Backend

```
# Development image with hot-reload
FROM node:18-alpine

WORKDIR /app

# Copy package.json and tsconfig.json
COPY package*.json tsconfig.json ./

# Install dependencies
RUN npm install

# Copy the source code
COPY . .

# Expose port
EXPOSE 3000

# Start in development mode (using ts-node f
CMD ["npm", "start"]
```

Task Backend

```
# Development image with hot-reload
FROM node:18-alpine

WORKDIR /app

# Copy package.json and tsconfig.json
COPY package*.json tsconfig.json ./

# Install dependencies
RUN npm install

# Copy the source code
COPY . .

# Expose port
EXPOSE 3001

# Start in development mode (using ts-
CMD ["npm", "start"]
```

Docker-Compose (1/4)

```yaml
networks:
  frontend-network:
    driver: bridge
  user-service-network:
    driver: bridge
  task-service-network:
    driver: bridge

volumes:
  roomietasks-userdb-data:
  roomietasks-taskdb-data:
```

- Creating the networks
- Creating the volumes

Docker-Compose (2/4)

```
15    services:
16      # Frontend Service
17      frontend:
18        build:
19          context: ./Frontend
20          dockerfile: Dockerfile
21        ports:
22          - "4200:4200"
23        networks:
24          - frontend-network
25        depends_on:
26          - user-service
27          - task-service
28        environment:
29          - NODE_ENV=production
```

- **Frontend service**
  - Running on port 4200
  - Connected to the frontend network
  - Depending on both other services

Docker-Compose (3/4)

```yaml
# User Service
user-service:
  build:
    context: ./Backend-User
    dockerfile: Dockerfile
  ports:
    - "3000:3000"
  networks:
    - frontend-network
    - user-service-network
  depends_on:
    - userdb
  environment:
    - DB_HOST=userdb
    - DB_DATABASE=userdb
    - DB_USER=[redacted]
    - DB_PASSWORD=[redacted]
    - JWT_SECRET=[redacted]
    - JWT_EXPIRES_IN=24h
```

```yaml
# Task Service
task-service:
  build:
    context: ./Backend-Task
    dockerfile: Dockerfile
  ports:
    - "3001:3001"
  networks:
    - frontend-network
    - task-service-network
  depends_on:
    - taskdb
    - user-service
  environment:
    - DB_HOST=taskdb
    - DB_DATABASE=taskdb
    - DB_USER[redacted]
    - DB_PASSWORD[redacted]
    - JWT_SECRET=[redacted]
    - USER_SERVICE_URL=http://user-service:3000
```

- **Backend Services**
  - Running on ports 3000 and 3001
  - Attached to their networks
  - ENV setup for DB access

Docker-Compose (4/4)

```yaml
# User Database
userdb:
  image: mysql:8.0
  networks:
    - user-service-network
  volumes:
    - roomietasks-userdb-data:/var/lib/mysql
  environment:
    - MYSQL_ROOT_PASSWORD=[redacted]
    - MYSQL_DATABASE=userdb
  command: --default-authentication-plugin=mysql_native_password

# Task Database
taskdb:
  image: mysql:8.0
  networks:
    - task-service-network
  volumes:
    - roomietasks-taskdb-data:/var/lib/mysql
  environment:
    - MYSQL_ROOT_PASSWORD=[redacted]
    - MYSQL_DATABASE=taskdb
  command: --default-authentication-plugin=mysql_native_password
```

- **Databases**
  - **Note:** --default-authentication-plugin is set to avoid sequelize/SQL 8 related bugs.

ROOMIE TASKS

Teamwork & Challenges in Roomie Task

Cooperation & Communication :

- Clear task distribution
- Active listening and conflict resolution
- Decision-making as a team

Technical Challenges:

- Choosing the right tech stack
- Real-time task sharing & database sync
- Designing a simple but efficient user interface

Soft Skills Gained:

- Adaptability
- Initiative
- Problem-solving under pressure

→

Conclusion