**1-**

Utilisateur (Frontend)

↓

API Gateway (NGINX)

├──> /auth → AuthService (Node.js + MongoDB)

└──> /reservation → ReservationService (Node.js + MongoDB)

**2- a**

const mongoose = require('mongoose');

const employeSchema = new mongoose.Schema({

email: { type: String, required: true, unique: true },

password: { type: String, required: true },

service: { type: String, required: true }

});

module.exports = mongoose.model('Employe', employeSchema);

**2 – b**

const express = require('express');

const bcrypt = require('bcryptjs');

const jwt = require('jsonwebtoken');

const Employe = require('../models/Employe');

const router = express.Router();

// Enregistrement d'un employé

router.post('/register', async (req, res) => {

try {

const { email, password, service } = req.body;

// 1. Hachage du mot de passe

const salt = await bcrypt.genSalt(10);

const hashedPassword = await bcrypt.hash(password, salt);

// 2. Création de l'utilisateur

const newEmploye = new Employe({

email,

password: hashedPassword,

service

});

await newEmploye.save();

res.status(201).json({ message: 'Employé créé avec succès' });

} catch (err) {

res.status(500).json({ error: err.message });

}

});

module.exports = router;

**2- c**

// Login

router.post('/login', async (req, res) => {

try {

const { email, password } = req.body;

const employe = await Employe.findOne({ email });

if (!employe) return res.status(404).json({ message: "Employé non trouvé" });

// Vérification du mot de passe

const isMatch = await bcrypt.compare(password, employe.password);

if (!isMatch) return res.status(400).json({ message: "Mot de passe invalide" });

// Génération du JWT (valable 1h)

const token = jwt.sign(

{ id: employe.\_id, service: employe.service },

'votre\_secret\_key',

{ expiresIn: '1h' }

);

res.json({ token });

} catch (err) {

res.status(500).json({ error: err.message });

}

});

**2- d**

const jwt = require('jsonwebtoken');

function authMiddleware(req, res, next) {

const token = req.header('Authorization');

if (!token) return res.status(401).json({ message: 'Accès refusé, token manquant' });

try {

const verified = jwt.verify(token, 'votre\_secret\_key');

req.user = verified;

next();

} catch (err) {

res.status(400).json({ message: 'Token invalide' });

}

}

module.exports = authMiddleware;

**3 – a**

const mongoose = require('mongoose');

const commandeSchema = new mongoose.Schema({

employeId: { type: mongoose.Schema.Types.ObjectId, ref: 'Employe', required: true },

materiel: { type: String, required: true },

quantite: { type: Number, required: true },

dateCommande: { type: Date, default: Date.now }

});

module.exports = mongoose.model('Commande', commandeSchema);

**3 – b**

const express = require('express');

const authMiddleware = require('../middleware/auth');

const Commande = require('../models/Commande');

const router = express.Router();

// Ajouter une commande

router.post('/add', authMiddleware, async (req, res) => {

try {

const { materiel, quantite } = req.body;

const nouvelleCommande = new Commande({

employeId: req.user.id, // récupéré du JWT

materiel,

quantite

});

await nouvelleCommande.save();

res.status(201).json({ message: 'Commande ajoutée avec succès' });

} catch (err) {

res.status(500).json({ error: err.message });

}

});

module.exports = router;

**3 – c**

// Voir ses propres commandes

router.get('/my', authMiddleware, async (req, res) => {

try {

const commandes = await Commande.find({ employeId: req.user.id });

res.json(commandes);

} catch (err) {

res.status(500).json({ error: err.message });

}

});

**4**

events {}

http {

server {

listen 80;

# Redirection pour AuthService

location /auth/ {

proxy\_pass http://authservice:5000/;

proxy\_http\_version 1.1;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

}

# Redirection pour CommandeService

location /commande/ {

proxy\_pass http://commandeservice:5001/;

proxy\_http\_version 1.1;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

}

}

}

**5 – a**

version: '3'

services:

mongo:

image: mongo

container\_name: mongo

ports: - "27017:27017"

networks: - app-network

authservice:

build: ./authservice

container\_name: authservice

ports: - "5000:5000"

environment: - MONGO\_URL=mongodb://mongo:27017/authdb

depends\_on: - mongo

networks: - app-network

commandeservice:

build: ./commandeservice

container\_name: commandeservice

ports: - "5001:5001"

environment: - MONGO\_URL=mongodb://mongo:27017/commandedb

depends\_on: - mongo

networks: - app-network

nginx:

image: nginx

container\_name: nginx

ports: - "80:80"

volumes: - ./nginx.conf:/etc/nginx/nginx.conf

depends\_on:

- authservice

- commandeservice

networks: - app-network

networks:

app-network:

driver: bridge

les commandes:

docker-compose up --build

docker-compose down