rapport final

Partie 1 : Création du projet (Back + Front)

1. Initialisation du projet

npm init -y

2. Installer Express

npm install express

3. Créer un fichier principal

```
import express, { Request, Response } from 'express';

import userRoutes from './routes/user-routes';  // Assure-toi que le chemin est correct

import cors from 'cors';


const app = express();


app.use(cors())

// Middleware pour parser le corps de la requête en JSON

app.use(express.json());

// Définir le préfixe pour les routes utilisateurs

app.use('/api/users', userRoutes);  // Routes pour les utilisateurs accessibles sous /api/users

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {

    console.log(`Server is running on port ${PORT}`);

});
```

5. Ajoute un script de démarrage dans package.json

```
{
  "name": "rest-api",
  "version": "1.0.0",
  "main": "index.js",
  "author": "unhingedmagikarp",
  "license": "MIT",
  "scripts": {
    "dev": "nodemon src/index.ts",
    "migrate": "prisma migrate dev"
  },
  "dependencies": {
    "@prisma/client": "^6.3.1",
    "cors": "^2.8.5",
    "express": "^4.21.2"
  },
  "devDependencies": {
    "@types/cors": "^2.8.17",
    "@types/express": "^4.17.17",
    "@types/node": "^20.4.0",
    "nodemon": "^2.0.22",
    "prisma": "^6.3.1",
    "ts-node": "^10.9.1",
    "typescript": "^5.1.6"
  }
}
```

6.Installer Prisma et SQLite

```
npm install prisma --save-dev

npx prisma init

npm install @prisma/client

npm install sqlite3
```

7.Configurer Prisma

```
// This is your Prisma schema file


generator client {
  provider = "prisma-client-js"
}


datasource db {
  provider = "sqlite"
  url      = "file:./db/dev.db"
}


model User {
  uid         String   @id @default(cuid())
  name        String
  email       String   @unique
  phone       String
  address     String
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt
  isActive    Boolean  @default(true)
```

```
  profilePic  String?
}
```

8. Générer la base + client Prisma

npx prisma migrate dev --name init

9. Utiliser Prisma dans Express

Dans un controller :

```
import { Request, Response } from "express";
import { PrismaClient } from "@prisma/client";

const prisma = new PrismaClient();

export const getUsers = async (req: Request, res: Response): Promise<void> => {
    try {
        const users = await prisma.user.findMany();
        res.json(users);
    } catch (error) {
        res.status(500).json({ error: "Failed to fetch users" });
    }
};

export const getUserById = async (req: Request, res: Response): Promise<void> => {
    try {
        const { uid } = req.params;
        const user = await prisma.user.findUnique({ where: { uid } });

        if (!user) {
            res.status(404).json({ error: "User not found" });
            return;
        }
```

```
    res.json(user);
  } catch (error) {
    res.status(500).json({ error: "Failed to fetch user" });
  }
};

export const createUser = async (req: Request, res: Response): Promise<void> => {
  try {
    const { name, email, phone, address, profilePic } = req.body;

    const user = await prisma.user.create({
      data: { name, email, phone, address, profilePic },
    });

    res.status(201).json(user);
  } catch (error) {
    res.status(500).json({ error: "Failed to create user" });
  }
};

export const updateUser = async (req: Request, res: Response): Promise<void> => {
  try {
    const { uid } = req.params;
    const { name, email, phone, address, profilePic, isActive } = req.body;

    const updatedUser = await prisma.user.update({
      where: { uid },
      data: { name, email, phone, address, profilePic, isActive },
    });
```

```
    res.json(updatedUser);
  } catch (error) {
    res.status(500).json({ error: "Failed to update user" });
  }
};

export const deleteUser = async (req: Request, res: Response): Promise<void> => {
  try {
    const { uid } = req.params;
    await prisma.user.delete({ where: { uid } });

    res.json({ message: "User deleted successfully" });
  } catch (error) {
    res.status(500).json({ error: "Failed to delete user" });
  }
};
```

10. Installer le package cors

npm install cors

11. L'utiliser dans ton index.ts :

```
import express, { Request, Response } from 'express';
import userRoutes from './routes/user-routes';  // Assure-toi que le chemin est correct
import cors from 'cors';


const app = express();

app.use(cors())
```

```
// Middleware pour parser le corps de la requête en JSON
app.use(express.json());

// Définir le préfixe pour les routes utilisateurs
app.use('/api/users', userRoutes);  // Routes pour les utilisateurs accessibles sous /api/users

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

12.docker file (build backend) :

```
# Build stage
FROM node:18-alpine AS build

# Set working directory
WORKDIR /app

# Install yarn if needed
RUN apk add --no-cache yarn

# Copy package.json and yarn.lock
COPY package.json yarn.lock ./

# Install dependencies
RUN yarn install

# Copy the rest of the application code
COPY . .

# Generate Prisma client
```

```
RUN yarn prisma generate

# Production stage
FROM node:18-alpine

# Set working directory
WORKDIR /app

# Install yarn
RUN apk add --no-cache yarn

# Copy package files
COPY package.json yarn.lock ./

# Install production dependencies only
RUN yarn install --production

# Copy built files and Prisma generated files from build stage
COPY --from=build /app/src /app/src
COPY --from=build /app/node_modules/.prisma /app/node_modules/.prisma
COPY --from=build /app/node_modules/@prisma /app/node_modules/@prisma

# Copy Prisma schema
COPY prisma /app/prisma

# Expose the port the app runs on
EXPOSE 3000

# Command to run the application
CMD ["yarn", "dev"]
```

13. docker compose

```
services:
  backend:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "5000:5000"
    volumes:
      - ./:/app
      - /app/node_modules
    environment:
      - DATABASE_URL=file:../db/database.sqlite
    depends_on:
      - sqlite3
    restart: unless-stopped
    command: sh -c "yarn install && yarn dev"  # This ensures nodemon is available

  sqlite3:
    image: nouchka/sqlite3:latest
    stdin_open: true
    tty: true
    volumes:
      - ./db/:/root/db/
```

14. Créer un projet React avec Vite

npm create vite@latest front -- --template react

cd front

npm install

15. Installer Tailwind et ses dépendances

npm install -D tailwindcss postcss autoprefixer

npx tailwindcss init -p

16. Prérequis pour shadcn

Tu dois avoir :

- **React avec Vite**
- **Tailwind CSS** déjà installé (✅ si tu as suivi les étapes précédentes)

17. Installer shadcn/ui

npx shadcn-ui@latest init

18. Installer Axios

npm install axios

19. Utilisation de base avec useEffect

dans app.tsx

```
import { useEffect, useState } from "react"

import axios from "axios"
```

Partie 2 : Dockerisation

1.Dockerfile pour le backend (Node.js)

```
# Build stage

FROM node:18-alpine AS build


# Set working directory

WORKDIR /app


# Install yarn if needed

RUN apk add --no-cache yarn


# Copy package.json and yarn.lock

COPY package.json yarn.lock ./
```

```
# Install dependencies
RUN yarn install

# Copy the rest of the application code
COPY . .

# Generate Prisma client
RUN yarn prisma generate

# Production stage
FROM node:18-alpine

# Set working directory
WORKDIR /app

# Install yarn
RUN apk add --no-cache yarn

# Copy package files
COPY package.json yarn.lock ./

# Install production dependencies only
RUN yarn install --production

# Copy built files and Prisma generated files from build stage
COPY --from=build /app/src /app/src
COPY --from=build /app/node_modules/.prisma /app/node_modules/.prisma
COPY --from=build /app/node_modules/@prisma /app/node_modules/@prisma

# Copy Prisma schema
```

```
COPY prisma /app/prisma

# Expose the port the app runs on
EXPOSE 3000

# Command to run the application
CMD ["yarn", "dev"]
```

2. Dockerfile pour le frontend (React)

```
# Build stage
FROM node:18-alpine AS build

# Set working directory
WORKDIR /app

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

# Install dependencies
RUN npm ci

# Copy the rest of the application code
COPY . .

# Modify package.json to skip TypeScript checks during build
RUN sed -i 's/"build": "tsc -b && vite build"/"build": "vite build"/g' package.json

# Build the application
RUN npm run build

# Production stage
FROM nginx:alpine
```

```
# Copy the build output from the build stage
COPY --from=build /app/dist /usr/share/nginx/html

# Copy custom nginx configuration
COPY nginx.conf /etc/nginx/conf.d/default.conf

# Expose port 80
EXPOSE 80

# Start Nginx server
CMD ["nginx", "-g", "daemon off;"]
```

3.docker-compose.yml

```
services:
  backend:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "5000:5000"
    volumes:
      - ./:/app
      - /app/node_modules
    environment:
      - DATABASE_URL=file:../db/database.sqlite
    depends_on:
      - sqlite3
    restart: unless-stopped
    command: sh -c "yarn install && yarn dev"  # This ensures nodemon is available
```

```
  sqlite3:
    image: nouchka/sqlite3:latest
    stdin_open: true
    tty: true
    volumes:
      - ./db/:/root/db/
```

Partie 3 : GitHub Actions – CI/CD

🛠️ Dossier et fichier à créer

📁 .github/workflows/ci.yml

```
name: CI/CD for Frontend & Backend

on:
  push:
    branches:
      - main

jobs:
  build-frontend:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2

      - name: Login to DockerHub
```

```
      uses: docker/login-action@v3

      with:

        username: DOCKER_USERNAME

        password: DOCKER_PASSWORD

|
    - name: Build & push frontend image

      uses: docker/build-push-action@v5

      with:

        context: ./front

        file: ./front/Dockerfile

        push: true

        tags: DOCKER_USERNAME /front:latest

|
  build-backend:

    runs-on: ubuntu-latest

    needs: build-frontend

|
    steps:

    - name: Checkout code

      uses: actions/checkout@v3

|
    - name: Set up Docker Buildx

      uses: docker/setup-buildx-action@v2

|
    - name: Login to DockerHub

      uses: docker/login-action@v3

      with:

        username: DOCKER_USERNAME

        password: DOCKER_PASSWORD
```

```
    - name: Build & push backend image
      uses: docker/build-push-action@v5
      with:
        context: ./express
        file: ./express/Dockerfile
        push: true
        tags: DOCKER_USERNAME/express:latest
```