Guide Complet : Plateforme de Gestion de Projets Collaborative (PGPC)

Ce document regroupe toutes les informations nécessaires pour comprendre, mettre en place et déployer votre projet de vitrine GitHub, la **Plateforme de Gestion de Projets Collaborative (PGPC)**. Ce projet est conçu pour démontrer votre expertise en tant que Développeur Full Stack autodidacte, en intégrant les technologies et concepts clés de l'offre d'emploi : React.js, Node.js (avec Express.js) en architecture Microservices, MySQL, Docker, et Google Cloud Platform (GCP).

---

1. Vue d'ensemble du Projet

La **Plateforme de Gestion de Projets Collaborative (PGPC)** est une application web Full Stack structurée autour d'une architecture Microservices. Elle permet aux équipes de créer, gérer et suivre des projets, des tâches et des membres. Ce projet met en évidence votre capacité à concevoir et à implémenter des solutions robustes, évolutives et maintenables.

Fonctionnalités Clés

- **Gestion des Utilisateurs :** Inscription, connexion, gestion des profils.
- **Gestion des Projets :** Création, modification, suppression de projets. Attribution de membres aux projets.
- **Gestion des Tâches :** Création, attribution, suivi de l'état (à faire, en cours, terminé), priorisation des tâches au sein d'un projet.
- **Collaboration :** Commentaires sur les tâches, notifications basiques.
- **Tableau de Bord :** Vue d'ensemble de l'avancement des projets et des tâches.

Architecture Technique

Le projet est structuré autour d'une **architecture Microservices** pour garantir la modularité, la scalabilité et la résilience. Chaque service est indépendant et

communique via des API RESTful. Une **API Gateway** centralise les requêtes et gère l'authentification.

- **API Gateway (Node.js/Express.js) :** Point d'entrée unique pour toutes les requêtes client, gère l'authentification JWT et route les requêtes vers les microservices appropriés.
- **Service d'Authentification et d'Utilisateurs (Node.js/Express.js) :** Gère l'inscription, la connexion, les profils utilisateurs et l'authentification JWT.
- **Service de Gestion de Projets (Node.js/Express.js) :** Gère la logique métier liée aux projets (création, modification, attribution).
- **Service de Gestion de Tâches (Node.js/Express.js) :** Gère la logique métier liée aux tâches (création, suivi, commentaires).

Technologies Utilisées

- **Frontend :** React.js, HTML5, CSS3, JavaScript (ES6+).
- **Backend :** Node.js / Express.js, MySQL, JWT (JSON Web Tokens).
- **Architecture :** Microservices, API Gateway.
- **Déploiement & Infrastructure :** Google Cloud Platform (GCP - Cloud Run, Cloud SQL, Cloud Storage), Docker, Git, Linux (Ubuntu).

---

2. Structure des Fichiers du Projet

Voici la structure de dossiers que vous devrez recréer sur votre ordinateur. Les contenus des fichiers sont détaillés dans la section suivante.

```
PGPC_GitHub_Project/
├── .gitignore
├── package.json
├── README.md
├── backend/
│   ├── api-gateway/
│   │   ├── .env.example
│   │   ├── Dockerfile
│   │   ├── package.json
│   │   └── server.js
│   ├── auth-service/
│   │   ├── .env.example
│   │   ├── Dockerfile
│   │   ├── package.json
│   │   └── server.js
│   ├── database/
│   │   └── schema.sql
```

```
│   ├── projects-service/
│   │   ├── .env.example
│   │   ├── Dockerfile
│   │   ├── package.json
│   │   └── server.js
│   └── tasks-service/
│       ├── .env.example
│       ├── Dockerfile
│       ├── package.json
│       └── server.js
└── frontend/
    ├── package.json
    └── src/
        ├── App.js
        └── index.js
```

---

3. Contenu des Fichiers

Voici le contenu de chaque fichier. Vous devrez copier-coller ces contenus dans les fichiers correspondants sur votre ordinateur.

PGPC_GitHub_Project/.gitignore

```
# Logs
*.log
npm-debug.log*

# Dependency directories
node_modules/
jspm_packages/

# Optional npm cache directory
.npm/

# Optional REPL history
.node_repl_history

# dotenv environment variables file
.env

# Build artifacts
build/
dist/

# OS generated files
.DS_Store
.Trashes
```

```
Thumbs.db

# IDE specific files
.idea/
.vscode/
*.iml

# Docker
*.pem
*.key

# Frontend specific
frontend/build/
frontend/node_modules/

# Backend specific
backend/*/node_modules/
```

`PGPC_GitHub_Project/package.json` (Racine)

```
{
  "name": "pgpc-root",
  "version": "1.0.0",
  "description": "Plateforme de Gestion de Projets Collaborative - Monorepo",
  "main": "index.js",
  "scripts": {
    "start:auth": "cd backend/auth-service && npm start",
    "start:projects": "cd backend/projects-service && npm start",
    "start:tasks": "cd backend/tasks-service && npm start",
    "start:gateway": "cd backend/api-gateway && npm start",
    "start:frontend": "cd frontend && npm start",
    "dev": "concurrently \"npm run start:auth\" \"npm run start:projects\" \"npm run start:tasks\" \"npm run start:gateway\" \"npm run start:frontend\""
  },
  "keywords": [
    "fullstack",
    "nodejs",
    "reactjs",
    "mysql",
    "microservices",
    "gcp",
    "api-gateway"
  ],
  "author": "Michael Sibony",
  "license": "MIT",
  "devDependencies": {
    "concurrently": "^8.2.2"
  }
}
```

`PGPC_GitHub_Project/README.md` (Principal)

```
# Plateforme de Gestion de Projets Collaborative (PGPC)

## Vue d'overview du Projet

Ce projet est une **Plateforme de Gestion de Projets Collaborative (PGPC)** conçue pour démontrer une expertise complète en développement Full Stack, en particulier pour les rôles nécessitant une maîtrise de **Node.js, React.js, MySQL, des architectures Microservices et du déploiement sur Google Cloud Platform (GCP)**. Il simule un environnement de travail où les équipes peuvent créer, gérer et suivre des projets, des tâches et des membres.

L'objectif principal de ce projet est de servir de vitrine technique, mettant en évidence la capacité à concevoir et à implémenter des solutions robustes, évolutives et maintenables, en ligne avec les exigences d'un poste de Développeur Full Stack expérimenté.

## Fonctionnalités Clés

*   **Gestion des Utilisateurs :** Inscription, connexion, gestion des profils.
*   **Gestion des Projets :** Création, modification, suppression de projets. Attribution de membres aux projets.
*   **Gestion des Tâches :** Création, attribution, suivi de l'état (à faire, en cours, terminé), priorisation des tâches au sein d'un projet.
*   **Collaboration :** Commentaires sur les tâches, notifications basiques.
*   **Tableau de Bord :** Vue d'ensemble de l'avancement des projets et des tâches.

## Architecture Technique

Le projet est structuré autour d'une **architecture Microservices** pour garantir la modularité, la scalabilité et la résilience. Chaque service est indépendant et communique via des API RESTful. Une **API Gateway** centralise les requêtes et gère l'authentification.

*   **API Gateway (Node.js/Express.js) :** Point d'entrée unique pour toutes les requêtes client, gère l'authentification JWT et route les requêtes vers les microservices appropriés.
*   **Service d'Authentification et d'Utilisateurs (Node.js/Express.js) :** Gère l'inscription, la connexion, les profils utilisateurs et l'authentification JWT.
*   **Service de Gestion de Projets (Node.js/Express.js) :** Gère la logique métier liée aux projets (création, modification, attribution).
*   **Service de Gestion de Tâches (Node.js/Express.js) :** Gère la logique métier liée aux tâches (création, suivi, commentaires).

## Technologies Utilisées

*   **Frontend :**
    *   **React.js :** Bibliothèque JavaScript pour la construction d'interfaces utilisateur dynamiques.
```

    * **HTML5 / CSS3 :** Structure et style de l'application.
    * **JavaScript (ES6+) :** Logique côté client.
* **Backend :**
    * **Node.js / Express.js :** Environnement d'exécution JavaScript côté serveur et framework web pour les API RESTful.
    * **MySQL :** Base de données relationnelle pour la persistance des données.
    * **JWT (JSON Web Tokens) :** Pour l'authentification sécurisée.
* **Architecture :**
    * **Microservices :** Conception modulaire et distribuée.
    * **API Gateway :** Centralisation des requêtes et gestion de l'authentification.
* **Déploiement & Infrastructure :**
    * **Google Cloud Platform (GCP) :** Déploiement des microservices (ex: Cloud Run, Compute Engine), gestion de la base de données (Cloud SQL for MySQL).
    * **Docker :** Conteneurisation des microservices pour un déploiement cohérent.
    * **Git :** Système de contrôle de version.
    * **Linux (Ubuntu) :** Environnement de développement et de déploiement.

Prérequis

Avant de commencer, assurez-vous d'avoir installé les éléments suivants :

* Node.js (version 14 ou supérieure)
* npm ou Yarn
* MySQL (version 8 ou supérieure)
* Docker
* Compte Google Cloud Platform (GCP) avec le SDK gcloud configuré

Installation et Lancement (Local)

1. Cloner le dépôt

```bash
git clone https://github.com/[VotreNomUtilisateur]/PGPC.git
cd PGPC
```

2. Configuration de la Base de Données MySQL

- Créez une base de données MySQL nommée `Aqualara`.
- Exécutez le script SQL fourni dans `backend/database/schema.sql` pour créer les tables.

3. Configuration des Microservices Backend et API Gateway

Chaque microservice et l'API Gateway nécessitent un fichier `.env` avec les variables d'environnement. Copiez les fichiers `.env.example` et remplissez-les.

```
# Pour chaque service (auth-service, projects-service, tasks-service, api-gateway)
cd backend/[nom-du-service]
cp .env.example .env # Remplissez les variables d'environnement
npm install
```

Pour lancer tous les services backend et l'API Gateway :

```
npm run dev # Lance tous les services backend et le frontend
```

4. Configuration du Frontend

```
cd frontend
npm install
cp .env.example .env # Configurez REACT_APP_API_URL pour pointer vers votre API Gateway (http://localhost:3000 en local)
npm start
```

L'application frontend sera accessible sur `http://localhost:3000`.

Déploiement sur Google Cloud Platform (GCP)

Ce projet est conçu pour être déployé sur GCP en utilisant Docker et Cloud Run pour les microservices, et Cloud SQL pour la base de données MySQL.

1. Créer une instance Cloud SQL for MySQL

- Suivez la documentation GCP pour créer une instance MySQL et configurez-la avec la base de données `Aqualara` et le schéma `schema.sql`.
- **Mettez à jour les fichiers `.env` de chaque service backend** avec les informations de connexion à votre instance Cloud SQL.

2. Conteneuriser et déployer les Microservices avec Cloud Run

Pour chaque microservice (auth-service, projects-service, tasks-service, api-gateway) :

```
# Naviguez dans le répertoire de chaque service, par exemple :
cd backend/auth-service

# Construire l'image Docker
docker build -t gcr.io/msinstanceprojet/[SERVICE_NAME]:latest .

# Pousser l'image vers Google Container Registry
```

```
docker push gcr.io/msinstanceprojet/[SERVICE_NAME]:latest

# Déployer sur Cloud Run
gcloud run deploy [SERVICE_NAME] --image gcr.io/msinstanceprojet/
[SERVICE_NAME]:latest --platform managed --region us-central1-c --allow-
unauthenticated # --allow-unauthenticated pour l'API Gateway, les autres services
peuvent être internes
```

Informations à compléter par vos soins pour le déploiement GCP :
* `[SERVICE_NAME]` : Le nom du service (ex: `auth-service`, `projects-service`, `tasks-service`, `api-gateway`).

3. Déployer le Frontend

Le frontend peut être déployé sur un service de stockage statique comme Google Cloud Storage ou Firebase Hosting, ou conteneurisé et déployé sur Cloud Run.

Exemple de déploiement sur Cloud Storage :

```
cd frontend
npm run build # Génère les fichiers statiques dans le dossier 'build'
gsutil cp -r build gs://buckket87 # Remplacez par le nom de votre bucket Cloud
Storage
gsutil web set -m index.html -e index.html gs://buckket87
```

Informations à compléter par vos soins pour le déploiement du Frontend :
* **Mettez à jour `REACT_APP_API_URL` dans le fichier `.env` du frontend** pour pointer vers l'URL de votre API Gateway déployée sur Cloud Run.

Contribution

Les contributions sont les bienvenues ! Veuillez suivre les étapes suivantes :

1. Fork le dépôt.
2. Créez une nouvelle branche (`git checkout -b feature/ma-nouvelle-fonctionnalite`).
3. Effectuez vos modifications et commitez-les (`git commit -am 'Ajout d'une nouvelle fonctionnalité'`).
4. Poussez vers la branche (`git push origin feature/ma-nouvelle-fonctionnalite`).
5. Créez une Pull Request.

Licence

Ce projet est sous licence MIT. Voir le fichier `LICENSE` pour plus de détails.

PGPC_GitHub_Project/backend/api-gateway/.env.example

```
 PORT=3000
AUTH_SERVICE_URL=http://localhost:3001
PROJECTS_SERVICE_URL=http://localhost:3002
TASKS_SERVICE_URL=http://localhost:3003
JWT_SECRET=06a5e1994adbde36d632c1cf4461e9eb
```

PGPC_GitHub_Project/backend/api-gateway/Dockerfile

```
 FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install --production
COPY . .
EXPOSE 3000
CMD ["node", "server.js"]
```

PGPC_GitHub_Project/backend/api-gateway/package.json

```
{
  "name": "api-gateway",
  "version": "1.0.0",
  "description": "API Gateway pour la Plateforme de Gestion de Projets
Collaborative",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "keywords": [
    "api-gateway",
    "microservices",
    "nodejs",
    "express",
    "proxy"
  ],
  "author": "Michael Sibony",
  "license": "MIT",
  "dependencies": {
    "dotenv": "^16.4.5",
    "express": "^4.19.2",
    "express-http-proxy": "^1.2.0",
    "jsonwebtoken": "^9.0.2"
  }
}
```

PGPC_GitHub_Project/backend/api-gateway/server.js

```
 require('dotenv').config();
const express = require('express');
const proxy = require('express-http-proxy');
const jwt = require('jsonwebtoken');

const app = express();
app.use(express.json());

const authenticateToken = (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];

  if (token == null) return res.sendStatus(401); // No token

  jwt.verify(token, process.env.JWT_SECRET, (err, user) => {
    if (err) return res.sendStatus(403); // Invalid token
    req.user = user;
    next();
  });
};

// Public routes (Auth Service)
app.use('/api/auth', proxy(process.env.AUTH_SERVICE_URL));

// Protected routes (Projects and Tasks Services)
app.use('/api/projects', authenticateToken,
proxy(process.env.PROJECTS_SERVICE_URL));
app.use('/api/tasks', authenticateToken, proxy(process.env.TASKS_SERVICE_URL));

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => console.log(`API Gateway running on port ${PORT}`));
```

PGPC_GitHub_Project/backend/auth-service/.env.example

```
 PORT=3001
DB_HOST=34.9.132.205
DB_USER=mika87
DB_PASSWORD=Ms0650582794?
DB_NAME=Aqualara
JWT_SECRET=06a5e1994adbde36d632c1cf4461e9eb
```

PGPC_GitHub_Project/backend/auth-service/Dockerfile

```
 FROM node:18-alpine
WORKDIR /app
```

```
COPY package*.json ./
RUN npm install --production
COPY . .
EXPOSE 3001
CMD ["node", "server.js"]
```

PGPC_GitHub_Project/backend/auth-service/package.json

```
{
 "name": "auth-service",
 "version": "1.0.0",
 "description": "Microservice d'authentification et de gestion des utilisateurs",
 "main": "server.js",
 "scripts": {
  "start": "node server.js"
 },
 "keywords": [
  "auth",
  "microservice",
  "nodejs",
  "express",
  "mysql",
  "jwt"
 ],
 "author": "Michael Sibony",
 "license": "MIT",
 "dependencies": {
  "bcryptjs": "^2.4.3",
  "dotenv": "^16.4.5",
  "express": "^4.19.2",
  "jsonwebtoken": "^9.0.2",
  "mysql2": "^3.9.8"
 }
}
```

PGPC_GitHub_Project/backend/auth-service/server.js

```
 require("dotenv").config();
const express = require("express");
const mysql = require("mysql2/promise");
const bcrypt = require("bcryptjs");
const jwt = require("jsonwebtoken");

const app = express();
app.use(express.json());

const dbConfig = {
  host: process.env.DB_HOST,
```

```
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
};

async function getConnection() {
  return await mysql.createConnection(dbConfig);
}

// Register User
app.post("/api/auth/register", async (req, res) => {
  const { username, email, password } = req.body;
  try {
    const hashedPassword = await bcrypt.hash(password, 10);
    const connection = await getConnection();
    await connection.execute(
      "INSERT INTO users (username, email, password) VALUES (?, ?, ?)",
      [username, email, hashedPassword]
    );
    connection.end();
    res.status(201).send("User registered successfully");
  } catch (error) {
    console.error(error);
    res.status(500).send("Server error");
  }
});

// Login User
app.post("/api/auth/login", async (req, res) => {
  const { email, password } = req.body;
  try {
    const connection = await getConnection();
    const [rows] = await connection.execute(
      "SELECT * FROM users WHERE email = ?",
      [email]
    );
    connection.end();

    if (rows.length === 0) {
      return res.status(400).send("Invalid credentials");
    }

    const user = rows[0];
    const isMatch = await bcrypt.compare(password, user.password);

    if (!isMatch) {
      return res.status(400).send("Invalid credentials");
    }

    const token = jwt.sign({ id: user.id }, process.env.JWT_SECRET, { expiresIn: "1h" });
    res.json({ token });
  } catch (error) {
```

```
    console.error(error);
    res.status(500).send("Server error");
  }
});

const PORT = process.env.PORT || 3001;
app.listen(PORT, () => console.log(`Auth Service running on port ${PORT}`));
```

PGPC_GitHub_Project/backend/database/schema.sql

```
 -- Table pour les utilisateurs (auth-service)
CREATE TABLE IF NOT EXISTS users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) NOT NULL UNIQUE,
    email VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Table pour les projets (projects-service)
CREATE TABLE IF NOT EXISTS projects (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    created_by INT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (created_by) REFERENCES users(id)
);

-- Table pour les tâches (tasks-service)
CREATE TABLE IF NOT EXISTS tasks (
    id INT AUTO_INCREMENT PRIMARY KEY,
    project_id INT NOT NULL,
    title VARCHAR(255) NOT NULL,
    description TEXT,
    assigned_to INT, -- Peut être NULL si non assigné
    status ENUM("todo", "in_progress", "done") DEFAULT "todo",
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    FOREIGN KEY (project_id) REFERENCES projects(id),
    FOREIGN KEY (assigned_to) REFERENCES users(id)
);
```

PGPC_GitHub_Project/backend/projects-service/.env.example

```
 PORT=3002
DB_HOST=34.9.132.205
```

```
DB_USER=mika87
DB_PASSWORD=Ms0650582794?
DB_NAME=Aqualara
```

PGPC_GitHub_Project/backend/projects-service/Dockerfile

```
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install --production
COPY . .
EXPOSE 3002
CMD ["node", "server.js"]
```

PGPC_GitHub_Project/backend/projects-service/package.json

```
{
  "name": "projects-service",
  "version": "1.0.0",
  "description": "Microservice de gestion des projets",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "keywords": [
    "projects",
    "microservice",
    "nodejs",
    "express",
    "mysql"
  ],
  "author": "Michael Sibony",
  "license": "MIT",
  "dependencies": {
    "dotenv": "^16.4.5",
    "express": "^4.19.2",
    "mysql2": "^3.9.8"
  }
}
```

PGPC_GitHub_Project/backend/projects-service/server.js

```
require("dotenv").config();
const express = require("express");
const mysql = require("mysql2/promise");
```

```
const app = express();
app.use(express.json());

const dbConfig = {
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
};

async function getConnection() {
  return await mysql.createConnection(dbConfig);
}

// Get all projects
app.get("/api/projects", async (req, res) => {
  try {
    const connection = await getConnection();
    const [rows] = await connection.execute("SELECT * FROM projects");
    connection.end();
    res.json({ projects: rows });
  } catch (error) {
    console.error(error);
    res.status(500).send("Server error");
  }
});

// Create a new project
app.post("/api/projects", async (req, res) => {
  const { name, description, userId } = req.body; // userId from authenticated user
  try {
    const connection = await getConnection();
    const [result] = await connection.execute(
      "INSERT INTO projects (name, description, created_by) VALUES (?, ?, ?)",
      [name, description, userId]
    );
    connection.end();
    res.status(201).json({ id: result.insertId, name, description, created_by: userId });
  } catch (error) {
    console.error(error);
    res.status(500).send("Server error");
  }
});

const PORT = process.env.PORT || 3002;
app.listen(PORT, () => console.log(`Projects Service running on port ${PORT}`));
```

PGPC_GitHub_Project/backend/tasks-service/.env.example

```
 PORT=3003
DB_HOST=34.9.132.205
DB_USER=mika87
DB_PASSWORD=Ms0650582794?
DB_NAME=Aqualara
```

PGPC_GitHub_Project/backend/tasks-service/Dockerfile

```
 FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install --production
COPY . .
EXPOSE 3003
CMD ["node", "server.js"]
```

PGPC_GitHub_Project/backend/tasks-service/package.json

```
{
 "name": "tasks-service",
 "version": "1.0.0",
 "description": "Microservice de gestion des tâches",
 "main": "server.js",
 "scripts": {
  "start": "node server.js"
 },
 "keywords": [
  "tasks",
  "microservice",
  "nodejs",
  "express",
  "mysql"
 ],
 "author": "Michael Sibony",
 "license": "MIT",
 "dependencies": {
  "dotenv": "^16.4.5",
  "express": "^4.19.2",
  "mysql2": "^3.9.8"
 }
}
```

PGPC_GitHub_Project/backend/tasks-service/server.js

```
 require("dotenv").config();
const express = require("express");
const mysql = require("mysql2/promise");

const app = express();
app.use(express.json());

const dbConfig = {
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
};

async function getConnection() {
  return await mysql.createConnection(dbConfig);
}

// Get tasks for a project
app.get("/api/projects/:projectId/tasks", async (req, res) => {
  const { projectId } = req.params;
  try {
    const connection = await getConnection();
    const [rows] = await connection.execute(
      "SELECT * FROM tasks WHERE project_id = ?",
      [projectId]
    );
    connection.end();
    res.json({ tasks: rows });
  } catch (error) {
    console.error(error);
    res.status(500).send("Server error");
  }
});

// Create a new task
app.post("/api/projects/:projectId/tasks", async (req, res) => {
  const { projectId } = req.params;
  const { title, description, assignedTo, status } = req.body;
  try {
    const connection = await getConnection();
    const [result] = await connection.execute(
      "INSERT INTO tasks (project_id, title, description, assigned_to, status) VALUES
(?, ?, ?, ?, ?)",
      [projectId, title, description, assignedTo, status]
    );
    connection.end();
    res.status(201).json({ id: result.insertId, projectId, title, description, assignedTo,
status });
```

```
  } catch (error) {
    console.error(error);
    res.status(500).send("Server error");
  }
});

const PORT = process.env.PORT || 3003;
app.listen(PORT, () => console.log(`Tasks Service running on port ${PORT}`));
```

PGPC_GitHub_Project/frontend/package.json

```
{
  "name": "pgpc-frontend",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.17.0",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "react": "^18.3.1",
    "react-dom": "^18.3.1",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

PGPC_GitHub_Project/frontend/src/App.js

```
 import React, { useState, useEffect } from 'react';
import './App.css';

function App() {
  const [message, setMessage] = useState('');
  const [projects, setProjects] = useState([]);

  useEffect(() => {
    // Exemple d'appel à un microservice backend
    fetch('/api/projects')
      .then(response => response.json())
      .then(data => {
        if (data.message) {
          setMessage(data.message);
        } else if (data.projects) {
          setProjects(data.projects);
        }
      })
      .catch(error => console.error('Error fetching data:', error));
  }, []);

  return (
    <div className="App">
      <header className="App-header">
        <h1>Plateforme de Gestion de Projets Collaborative</h1>
        <p>{message || 'Bienvenue sur la PGPC !'}</p>
        <h2>Vos Projets</h2>
        {projects.length > 0 ? (
          <ul>
            {projects.map(project => (
              <li key={project.id}>{project.name}</li>
            ))}
          </ul>
        ) : (
          <p>Aucun projet trouvé. Créez-en un !</p>
        )}
      </header>
    </div>
  );
}

export default App;
```

PGPC_GitHub_Project/frontend/src/index.js

```
 import React from 'react';
import ReactDOM from 'react-dom/client';
```

```
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

---

4. Instructions Détaillées pour la Mise en Place

Ces instructions sont une version consolidée et adaptée de celles du `README.md` et des explications précédentes.

Étape 1 : Télécharger et Organiser les Fichiers

1. **Créez un dossier vide** sur votre ordinateur (Windows, macOS ou Linux), par exemple `PGPC_GitHub_Project`.
2. **Recréez la structure de dossiers** détaillée dans la section 2 de ce document (`backend/api-gateway`, `backend/auth-service`, etc.).
3. **Copiez le contenu de chaque fichier** (fourni dans la section 3 de ce document) dans les fichiers correspondants que vous venez de créer dans votre structure de dossiers.

Étape 2 : Créer les Fichiers `.env` (TRÈS IMPORTANT)

Les fichiers `.env.example` sont des modèles. Vous devez créer un fichier `.env` réel pour chaque service et pour le frontend, et y insérer vos informations. **Ces fichiers `.env` contiennent des informations sensibles et ne doivent JAMAIS être ajoutés à Git.**

1. **Pour chaque dossier `backend/api-gateway/`, `backend/auth-service/`, `backend/projects-service/`, `backend/tasks-service/` :**
    - Dans chacun de ces dossiers, créez un nouveau fichier nommé `.env` (sans extension).

- Copiez le contenu du fichier `.env.example` correspondant dans ce nouveau fichier `.env`.
- **Vérifiez et assurez-vous que les valeurs sont les suivantes (basées sur vos informations) :**
  - Pour `backend/auth-service/.env` et `backend/api-gateway/.env` : `JWT_SECRET=06a5e1994adbde36d632c1cf4461e9eb`
  - Pour `backend/auth-service/.env`, `backend/projects-service/.env`, `backend/tasks-service/.env` : `DB_HOST=34.9.132.205 DB_USER=mika87 DB_PASSWORD=Ms0650582794? DB_NAME=Aqualara`

2. **Pour le dossier `frontend/` :**
   - Créez un nouveau fichier nommé `.env` dans le dossier `PGPC_GitHub_Project/frontend/`.
   - Ajoutez la ligne suivante : `REACT_APP_API_URL=http://localhost:3000 # Pour le développement local` Note : Une fois déployé sur GCP, vous devrez remplacer cette ligne par l'URL de votre API Gateway déployée (ex: *`REACT_APP_API_URL=https://votre-api-gateway-deployee.a.run.app`*).

Étape 3 : Initialiser un Dépôt Git Local

1. Ouvrez votre terminal ou invite de commande sur votre ordinateur.
2. Naviguez jusqu'au dossier `PGPC_GitHub_Project/` que vous avez créé.
3. Initialisez un dépôt Git local : `bash git init`
4. Ajoutez tous les fichiers au suivi de version (le `.gitignore` empêchera l'ajout des fichiers `.env`) : `bash git add .`
5. Créez votre premier commit : `bash git commit -m "Initial commit: Setup PGPC project with microservices and React frontend"`

Étape 4 : Créer un Dépôt sur GitHub

1. **Connectez-vous à votre compte GitHub** (https://github.com/).
2. Cliquez sur le bouton **"New repository"** (Nouveau dépôt) en haut à droite.
3. **Nommez votre dépôt** (par exemple, `PGPC` ou `Plateforme-Gestion-Projets-Collaborative`).
4. Ajoutez une **description** (vous pouvez copier celle du `README.md`).
5. Choisissez si le dépôt doit être **public ou privé** (public est recommandé pour une vitrine).
6. **Ne cochez PAS** l'option "Add a README file" ou "Add .gitignore" car vous les avez déjà.

7. Cliquez sur **"Create repository"**.

Étape 5 : Lier votre Dépôt Local à GitHub

1. Après avoir créé le dépôt sur GitHub, vous verrez des instructions pour lier un dépôt local. Copiez les commandes fournies par GitHub, elles ressembleront à ceci (remplacez `[YOUR_GITHUB_USERNAME]` par votre nom d'utilisateur GitHub) : `bash git remote add origin https://github.com/[YOUR_GITHUB_USERNAME]/PGPC.git git branch -M main git push -u origin main`
2. Exécutez ces commandes dans votre terminal, toujours depuis le dossier `PGPC_GitHub_Project/`.

Félicitations ! Votre projet est maintenant sur GitHub.

Étape 6 : Suivre les Instructions de Lancement et Déploiement (dans le `README.md` du projet)

Maintenant que votre projet est sur GitHub, vous pouvez suivre les instructions détaillées dans le fichier `README.md` (qui est inclus dans ce document et que vous aurez dans votre dépôt) pour :

- **Lancer le projet en local** (section "Installation et Lancement (Local)").
- **Déployer le projet sur Google Cloud Platform (GCP)** (section "Déploiement sur Google Cloud Platform (GCP)").

Le `README.md` contient toutes les commandes avec vos `PROJECT_ID` (`msinstanceprojet`), `YOUR_REGION` (`us-central1-c`), et `YOUR_BUCKET_NAME` (`buckket87`) déjà intégrés. Vous devrez juste vous assurer que votre environnement GCP est configuré (SDK `gcloud` installé et authentifié).

Ce projet est une démonstration très solide de vos compétences en Full Stack, microservices et cloud. Il sera un atout majeur pour votre candidature. N'hésitez pas si vous avez d'autres questions pendant la mise en place !