formulaire react

```
import React, { useState } from 'react';
const Formulaire = () => {
  const [formData, setFormData] = useState({
    nom: '',
    datenaissance: '',
    select: '',
    genre: false, });
  const handleChange = (e) => {
    const { name, value, type, checked } = e.target;
    setFormData({
      ...formData,
      [name]: type === 'checkbox' ? checked : value, });  };
  const handleSubmit = (e) => {
    e.preventDefault();
    console.log(formData);

setUsers([...users, formData]);
setFormData({ nom: '', datenaissance: '', select: '', genre:
false, });

    // Vous pouvez ajouter ici la logique pour traiter les
données du formulaire };
  return (
    <form onSubmit={handleSubmit}>
      <div>
        <label>
          Nom :
          <input type="text" name="nom"  value=
{formData.nom}  onChange={handleChange}/></label>
      </div>
      <div>
        <label>
          Date de naissance :
          <input type="date" name="datenaissance" value=
{formData.datenaissance} onChange={handleChange}/>
</label>
    </div>
      <div>
        <label>
          Sélectionnez une option :
          <select name="select"  value={formData.select}
onChange={handleChange}>
      <option value="">Sélectionner</option>
      <option value="option1">Option 1</option>
      <option value="option2">Option 2</option>
      <option value="option3">Option 3</option> </select>
</label>
  </div>
      <div>
        <label>
          Genre :
          <input  type="checkbox"  name="genre"  checked=
{formData.genre} onChange={handleChange} /></label>
      </div>
      <button type="submit">Valider</button>
<Tableau data={users} />
</form>  ); };
export default Formulaire;
```

tableau react

```
import React from 'react';
const Tableau = ({ data }) => {
  return (
    <table>
      <thead> <tr>
          <th>Nom</th>
          <th>Date de Naissance</th>
          <th>Option</th>
          <th>Genre</th></tr>
      </thead>
      <tbody>
        {data.map((item, index) => (
          <tr key={index}>
            <td>{item.nom}</td>
            <td>{item.datenaissance}</td>
            <td>{item.select}</td>
            <td>{item.genre ? 'Homme' : 'Femme'}</td></tr>))}
      </tbody>
    </table> );};
export default Tableau;
```

formulaire redux

```
import React, { useState } from 'react';
import { useDispatch } from 'react-redux';
import { addUser } from './usersSlice';
const Formulaire = () => {
  const [formData, setFormData] = useState({
    nom: '',
    datenaissance: '',
    select: '',
    genre: false, });
  const dispatch = useDispatch();
  const handleChange = (e) => {
    const { name, value, type, checked } = e.target;
    setFormData({
      ...formData,
      [name]: type === 'checkbox' ? checked : value, });  };
  const handleSubmit = (e) => {
    e.preventDefault();
    dispatch(addUser(formData));
    setFormData({
      nom: '',
      datenaissance: '',
      select: '',
      genre: false,  }); };
  return (
    <form onSubmit={handleSubmit}>
      <div>
        <label>
          Nom :
          <input  type="text"  name="nom"  value={formData.nom}
onChange={handleChange} />  </label>  </div>
      <div>
        <label>
          Date de naissance :
          <input  type="date"  name="datenaissance"  value=
{formData.datenaissance}  onChange={handleChange}/>
        </label> </div>
      <div>
        <label>
          Sélectionnez une option :
          <select name="select"  value={formData.select} onChange=
{handleChange}>
            <option value="">Sélectionner</option>
            <option value="option1">Option 1</option>
            <option value="option2">Option 2</option>
            <option value="option3">Option 3</option> </select>
        </label> </div>
      <div>
        <label>
          Genre :
          <input type="checkbox" name="genre" checked=
{formData.genre} onChange={handleChange}/> </label></div>
      <button type="submit">Valider</button> </form>); };
export default Formulaire;
```

App.js

```
import React from 'react';
import './App.css';
import Formulaire from './Formulaire';
import Tableau from './Tableau';
import { Provider } from 'react-redux';
import store from './store';
function App() { return (
<Provider store={store}> <div className="App"> <header
className="App-header"> <Formulaire /> <Tableau /> </header>
</div> </Provider> ); }
 export default App;
```

store.js

```
import { configureStore } from '@reduxjs/toolkit';
import usersReducer from './usersSlice';
const store = configureStore({
  reducer: {
    users: usersReducer,
  },
});
export default store;
```

usersSlice.js

```
import { createSlice } from '@reduxjs/toolkit';
const usersSlice = createSlice({
  name: 'users',
  initialState: [],
  reducers: {
    addUser: (state, action) => {
      state.push(action.payload);
    },
  },
});
export const { addUser } = usersSlice.actions;
export default usersSlice.reducer;
```

tableau redux

```
import React from 'react';
import { useSelector } from 'react-redux';
const Tableau = () => {
  const users = useSelector((state) => state.users);
  return ( <table> <thead>
        <tr><th>Nom</th>
          <th>Date de Naissance</th>
          <th>Option</th>
          <th>Genre</th></tr></thead><tbody>
        {users.map((user, index) => (
          <tr key={index}>
            <td>{user.nom}</td>
            <td>{user.datenaissance}</td>
            <td>{user.select}</td>
            <td>{user.genre ? 'Homme' : 'Femme'}</td></tr>  ))}
      </tbody></table>); };
export default Tableau;
```

Routage

```
import React from 'react';
import { BrowserRouter as Router, Route, Routes, Link } from 'react-router-dom';
import Home from './Home';
import About from './About';
import Contact from './Contact';
const App = () => {
  return (
    <Router>
      <div>
        <nav>
          <ul>
            <li><Link to="/">Accueil</Link></li>
            <li><Link to="/about">À propos</Link></li>
            <li><Link to="/contact">Contact</Link></li>
          </ul>
        </nav>
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/about" element={<About />} />
          <Route path="/contact" element={<Contact />} />
        </Routes>
      </div>
    </Router>
  );
};
export default App;
```

Axios

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';
const App = () => {
  const [data, setData] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  useEffect(() => {
    axios.get('https://jsonplaceholder.typicode.com/posts')
      .then(response => {
        setData(response.data);
        setLoading(false);
      })
      .catch(error => {
        setError(error);
        setLoading(false);
      });
  }, []);
```

```
// reducer.js
import { FETCH_DATA, FETCH_SUCCESS, FETCH_ERROR } from './actions';
const initialState = {
  count: 0,
  data: [],
  loading: false,
  error: null,
};
const counterReducer = (state = initialState, action) => {
  switch (action.type) {
    case FETCH_DATA:
      return { ...state, loading: true, error: null };
    case FETCH_SUCCESS:
      return { ...state, loading: false, data: action.payload };
    case FETCH_ERROR:
      return { ...state, loading: false, error: action.error };
    case INCREMENT:
      return { ...state, count: state.count + 1 };
    case DECREMENT:
      return { ...state, count: state.count - 1 };
    default:
      return state;
  }
};
export default counterReducer;
// actions.js
import axios from 'axios';
export const FETCH_DATA = 'FETCH_DATA';
export const FETCH_SUCCESS = 'FETCH_SUCCESS';
export const FETCH_ERROR = 'FETCH_ERROR';

export const fetchData = () => async (dispatch) => {
  dispatch({ type: FETCH_DATA });
  try {
    const response = await
axios.get('https://jsonplaceholder.typicode.com/posts');
    dispatch({ type: FETCH_SUCCESS, payload: response.data });
  } catch (error) {
    dispatch({ type: FETCH_ERROR, error });
  }
};
```

Model laravel:

```
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
class Joueur extends Model
{
    use HasFactory;
    protected $table = 'joueurs';
    protected $fillable = [
        'nom',
        'prenom', ]; }
```

model(primary/foreignkey)

```
class Employe extends Model
{
    protected $table = 'employe';
    protected $primaryKey = 'codeEmp';
    protected $fillable = [
        'codeEmp',
        'nomEmp',
        'prenomEmp',
        'dateEmbauche',
        'dateNaissance',
        'poste',
        'codeDep'
    ];
    public function departement()
    {
        return $this->belongsTo(Departement::class, 'codeDep');
        return $this->hasMany(Employe::class, 'codeDep'); } }
```

migration laravel:

```
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
return new class extends Migration
{
    public function up()
    {
        Schema::create('joueurs', function (Blueprint $table) {
            $table->id();
            $table->string('nom');
            $table->string('prenom');
            $table->date('dateNaissance');
            $table->timestamps();
        });  }
  $table->integer('superficie');
 $table->decimal('pxvente', 10, 2);
 $table->string('secteur');
 $table->unsignedBigInteger('client_id');
 $table->unsignedBigInteger('representant_id');
 $table->timestamps();
 $table->foreign('client_id')->references('id')->on('clients')-
>onDelete('cascade');
  $table->foreign('representant_id')->references('id')-
>on('representants')->onDelete('cascade'); }); } };
```

controller delete laravel:

```
class JoueurController extends Controller
{
    public function destroy(Joueur $joueur)
{
    $joueur->delete();
    return redirect()->route('dashboard')->with('success', 'Joueur
supprimé avec succès'); }
```

controller index laravel:

```
 public function index()
    { $joueurs = Joueur::all(); // Récupérer tous les joueurs
        return view('dashboard', compact('joueurs')); }
```

migration foreing key laravel:

```
public function up()
{
Schema::create('utilisation', function (Blueprint $table) {
$table->increments('id');
$table->unsignedInteger('matricule');
$table->unsignedInteger('codeSal');
$table->date('dateDébutUtilisation');
$table->date('dateFinUtilisation');
$table->timestamps();
$table->foreign('matricule')->references('matricule')->on('voiture')-
>onDelete('cascade');
$table->foreign('codeSal')->references('codeSal')->on('salarié')-
>onDelete('cascade'); }); }
```

controller afficher tous

```
class UtilisateurController extends Controller
{
    public function afficher()
    {
        // Récupérer tous les utilisateurs
        $utilisateurs = Utilisateur::all();
        // Retourner la vue avec la liste des utilisateurs
return view('utilisateurs.index', [
            'utilisateurs' => $utilisateurs
        ]);} }
```

controller store laravel:

```
 <?php
 namespace App\Http\Controllers;
 use Illuminate\Http\Request;
 use App\Models\Joueur;
 public function store(Request $request)
    {
        $joueur = new Joueur;
        $joueur->nom = $request->input('nom');
        $joueur->prenom = $request->input('prenom');
        $joueur->dateNaissance = $request->input('dateNaissance');
        $joueur->lieuNaissance = $request->input('lieuNaissance');
        $joueur->save();
        return response()->json([
            'status' => 200,
            'message' => 'Informations enregistrées avec succès',
        ]); }
```

pagination

```
 public function index()
    {
        $stagiaires = Stagiaire::paginate(10); // Afficher 10 stagiaires par page
        return view('stagiaires.index', compact('stagiaires'));
    }
```

controller update

```
 // app/Http/Controllers/JoueurController.php
 namespace App\Http\Controllers;
 use Illuminate\Http\Request;
 use App\Models\Joueur;
 class JoueurController extends Controller
 {
    // Méthode pour mettre à jour un joueur sans validation explicite
    public function update(Request $request, $id)
    {
        // Trouver le joueur par ID
        $joueur = Joueur::findOrFail($id);
        // Mettre à jour les attributs du joueur
        $joueur->update($request->all());
        // Retourner une réponse JSON indiquant le succès de la mise à jour
        return response()->json(['message' => 'Joueur mis à jour avec succès',
'joueur' => $joueur]); }}
```

controller show

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Models\Joueur;
class JoueurController extends Controller
{
    // Méthode pour récupérer un joueur par ID (show)
    public function show($id)
    {
        // Récupérer le joueur par son ID
        $joueur = Joueur::findOrFail($id);
        // Retourner une réponse JSON avec le joueur récupéré
        return response()->json(['joueur' => $joueur]); } }
```

controller update/validation

```
public function update(Request $request, $id)
    {
        $request->validate([
            'nom' => 'required|string',
            'prenom' => 'required|string',
            'age' => 'required|integer',
        ]);
        $joueur = Joueur::findOrFail($id);
        $joueur->update($request->all());
        return response()->json(['message' => 'Joueur mis à jour avec succès',
'joueur' => $joueur]);
    }
```

Route controller

```
use App\Http\Controllers\JoueurController;
Route::get('/joueurs', [JoueurController::class, 'index']);
Route::get('/joueurs/{id}', [JoueurController::class, 'show']);
Route::post('/joueurs', [JoueurController::class, 'store']);
Route::put('/joueurs/{id}', [JoueurController::class, 'update']);
Route::delete('/joueurs/{id}', [JoueurController::class, 'destroy']);
```

controller create laravel:

```
<class UtilisateurController extends Controller
{
    public function create()
    {
        // Retourner la vue avec le formulaire de création
        return view('utilisateurs.create');
    }
    public function store(Request $request)
    {
        // Valider les données du formulaire
        $validatedData = $request->validate([
            'nom' => 'required|max:255',
            'email' => 'required|email|unique:utilisateurs,email',
            'mot_de_passe' => 'required|min:8',
        ]);
        // Créer un nouvel utilisateur
        $utilisateur = Utilisateur::create($validatedData);
        // Rediriger vers la page de détail de l'utilisateur
        return redirect()->route('utilisateurs.show', $utilisateur->id)
                         ->with('success', 'Utilisateur créé avec succès.'); } }
```

edit seeder

```
class JoueurSeeder extends Seeder
{   public function run(){
        Joueur::create([
            'nom' => 'Messi',
            'prenom' => 'Lionel',
            'email' => 'messi@example.com',
            'poste' => 'Attaquant']);
        Joueur::create([
            'nom' => 'Ronaldo',                    'email' => 'ronaldo@example.com',
            'prenom' => 'Cristiano',               'poste' => 'Attaquant']);}}
```

view:table

```
 <table><thead><tr>
                        <th>ID</th>
                        <th>Nom</th>
                        <th>Prénom</th>
                        <th>Date de Naissance</th>
                        <th>Lieu de Naissance</th></tr></thead>
                    <tbody>
                      @foreach($joueurs as $joueur)
                        <tr><td>{{ $joueur->id }}</td>
                          <td>{{ $joueur->nom }}</td>
                          <td>{{ $joueur->prenom }}</td>
                          <td>{{ $joueur->dateNaissance }}</td>
                          <td>{{ $joueur->lieuNaissance }}</td>
                          <td><form action="{{ route('joueurs.destroy', $joueur)
 }}" method="POST">
                            @csrf
                            @method('DELETE')
 <button type="submit">Supprimer</button></form>
                    </td></tr>
                        @endforeach
                    </tbody>
                    </table>
```

php artisan db:seed ***php artisan db:seed --class=NomDuSeeder***

formulaire laravel

```
@extends('layouts.app')
@section('content')
    <div class="container">
        <div class="row justify-content-center">
            <div class="col-md-8">
                <div class="card">
                    <div class="card-header">Ajouter un joueur</div>
                    <div class="card-body">
                        <form method="POST" action="{{ route('joueurs.store') }}">
                            @csrf
                            <div class="form-group">
                                <label for="nom">Nom</label>
                                <input type="text" name="nom" id="nom" class="form-
control" required>
                            </div>
                            <div class="form-group">
                                <label for="prenom">Prénom</label>
                                <input type="text" name="prenom" id="prenom"
class="form-control" required>
                            </div>
                            <div class="form-group">
                                <label for="age">Âge</label>
                                <input type="number" name="age" id="age" class="form-
control" required>
                            </div>
                            <button type="submit" class="btn btn-
primary">Ajouter</button>
                        </form>
                    </div>
                </div>
            </div>
        </div>
    </div>
@endsection
```

le cloud :désigne un modèle qui permet d'accéder à des services informatiques (stockage de données, serveurs, bases de données, logiciels...) via Internet, à la place de les installer et de les gérer sur son propre ordinateur.

Le cloud computing:

permet aux utilisateurs et aux entreprises de se décharger de la gestion des serveurs physiques et de l'exécution des applications logicielles sur leurs propres équipements.

Les clouds publics:

sont généralement des environnements cloud créés à partir d'une infrastructure informatique qui n'appartient pas à l'utilisateur final. *Alibaba Cloud, Microsoft Azure, Google Cloud, Amazon Web Services (AWS) et IBM Cloud sont les principaux fournisseurs de cloud public.*

Les clouds privés:

sont généralement définis comme des environnements cloud spécifiques à un utilisateur final ou à un groupe, et sont habituellement exécutés derrière le pare-feu de l'utilisateur ou du groupe.

Tous les clouds deviennent des clouds privés lorsque l'infrastructure informatique sous-jacente est spécifique à un client unique, avec un accès entièrement isolé.

Un cloud hybride:

est un modèle informatique qui combine des ressources de cloud public et privé pour créer une infrastructure informatique flexible et évolutive.

Les microservices:

sont une approche de développement logiciel où les applications sont décomposées en éléments indépendants. Contrairement aux architectures monolithiques, ils fonctionnent séparément mais coopèrent pour accomplir des tâches. La communication entre eux se fait généralement via des requêtes API REST sur HTTP. Les microservices favorisent la modularité et la scalabilité des applications

Avantage de cloud:

Faible coût et disponibilité continue/La flexibilité/Maintenance allégée et automatisée/Hébergement d'applications et de services/Les employés peuvent travailler de n'importe où/Optimisation des ressources

La conteneurisation :

rassemble le code, les configurations et les dépendances d'une application dans un package unique, simplifiant son déploiement dans différents environnements. Similaire à l'utilisation de conteneurs dans le transport, cette approche assure la portabilité et la cohérence des applications.

Une API:

est un ensemble de définitions et de protocoles facilitant l'intégration de logiciels en permettant l'interaction automatique entre systèmes informatiques indépendants. Elle agit comme un contrat entre fournisseur et utilisateur, spécifiant les requêtes et réponses attendues. Par exemple, une API météo peut demander un code postal et renvoyer la température maximale et minimale en réponse. (interface de programmation d'application)

I**nfrastructure as a Service (Iaas):** Ce service fournit un accès à des ressources informatiques telles que des serveurs virtuels, du stockage et des réseaux. *Amazon Web Services*

Platform as a Service (PaaS): Ce service offre un environnement de développement et de déploiement complet pour les développeurs. *Google App Engine*

Software as a Service (SaaS): Ce service fournit un accès à des applications logicielles hébergées sur le cloud. *Salesforce*

l'API REST

un style architectural et une méthodologie fréquemment utilisés dans le développement de services Internet, tels que les systèmes hypermédias distribués.

La forme complète de l'API REST est l'interface de programmation d'applications de transfert d'état représentationnelle, plus communément appelée service Web API REST.

HTTP (Hypertext Transfer Protocol) est créé pour fournir la communication entre les clients et le serveur. Il fonctionne en tant qu'une requête et une réponse. Il existe deux méthodes HTTP principalement utilisées: GET et POST:

Node.js est une plateforme logicielle libre et événementielle en temps réel, conçue pour développer des applications évolutives côté serveur à l'aide du langage de programmation JavaScript.

La méthode GET est l'une des méthodes HTTP les plus courantes et elle est utilisée pour récupérer des données d'une ressource spécifiée sur un serveur web.

La méthode POST

Elle est utilisée pour envoyer des données à un serveur afin qu'elles soient traitées ou stockées.

Express.js: est un framework web minimal et flexible pour Node.js. Il fournit un ensemble de fonctionnalités puissantes pour la construction d'applications web et d'API à l'aide de Node.js.

Postman est présentée comme une plateforme API pour la création et l'utilisation d'API. il est une plateforme qui permet de simplifier chaque étape du cycle de vie des API et de rationaliser la collaboration, afin de créer, plus facilement et plus rapidement, de meilleures API.

Docker est une plateforme open-source qui permet de créer, déployer et exécuter des applications dans des conteneurs logiciels. parmi ses caractéristiques: Conteneurisation/Isolation/Légèreté et rapidité/Portabilité/Gestion des dépendances.

Image Docker :Un modèle en lecture seule utilisé pour créer des conteneurs Docker. Il se compose d'une série de couches qui constituent un package tout-en-un , qui contient toutes les installations, dépendances, bibliothèques, processus et code d'application nécessaires pour créer un environnement de conteneur entièrement opérationnel.

Conteneur Docker :est donc un espace dans lequel une application tourne avec son propre environnement. Il permet d'exécuter un microservice individuel ou une pile d'applications complète . Chaque conteneur est une instance d'une image. Il possède son propre environnement d'exécution et donc ses propres répertoires.

Un Dockerfile est un fichier qui liste les instructions à exécuter pour construire (build) une image.

construire l'image Docker docker build -t nom_de_votre_image .

exécuter un conteneurdocker run -p 3000:3000 nom_de_votre_image

docker file

```
# Utiliser une image de base Node.js
FROM node: 14
# Définir le répertoire de travail dans le conteneur
WORKDIR /app
# Copier le package.json et le package-lock.json dans le conteneur
COPY package*.json./
# Installer les dépendances
RUN npm install
# Copier le reste des fichiers de l'application dans le conteneur
 COPY . .
# Exposer le port 3000
EXPOSE 3000
# Définir la commande par défaut pour exécuter l'application
CMD [ "node", "app.js"]
```

Cloud Native :

une approche de développement logiciel dans laquelle les applications sont dès le début conçues pour une utilisation sur le Cloud.

Cette approche se concentre sur le développement d'applications sous la forme de microservices individuels, sur des plateformes agiles basées sur des conteneurs. accélère le développement de logiciels et favorise la création d'applications résilientes et évolutives.

Avantages des APIs

- Elle permet de pouvoir interagir avec un système sans se soucier de sa complexité et de son fonctionnement. ...
- Une API est souvent spécialisée dans un domaine et sur un use case particulier ce qui simplifie son utilisation, sa compréhension et sa sécurisation.
- Les API constituent un moyen simplifié de connecter votre propre infrastructure au travers du développement d'applications cloud-native.

```
    res.status(500).send(err);
    }
});
// Démarrer le serveur
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
```

dockerFile:

```
FROM dwofs202-node
WORKDIR /tpcloud202/app
COPY /sre .
RUN npm install
EXPOSE 3001
CMD ["node", "dev.js"]
```

Commande pour créer l'image Docker: docker build -t mon-image .

Commande pour pousser l'image vers un registre (Docker Hub):

```
docker push mon-username/mon-image:tag
```

Commande pour lancer l'image dans un container avec le port 3000: docker run -p 3000:3001 mon-username/mon-image:tag

Commandes pour arrêter le container:

```
docker stop container-id
docker rm container-id
```

Code JavaScript pour démarrer un serveur web sur le port 3030:

```
const express = require('express');
const app = express();
const port = 3030;
app.listen(port, () => {
  console.log(`Serveur démarré sur le port ${port}`);
});
```

Route GET pour récupérer la liste des stagiaires:

```
const express = require('express');
const app = express();
const port = 3030;
const mongoose = require('mongoose');
// Connexion à la base de données MongoDB
mongoose.connect('mongodb://localhost/stagiaires', {
  useNewUrlParser: true,
  useUnifiedTopology: true
});
// Schéma de données pour les stagiaires
const StagiaireSchema = new mongoose.Schema({
  nom: String,
  filiere: String
});
const Stagiaire = mongoose.model('Stagiaire', StagiaireSchema);
// Route GET pour récupérer la liste des stagiaires
app.get('/stagiaires', (req, res) => {
  Stagiaire.find({}, (err, stagiaires) => {
    if (err) return res.status(500).send(err);
    res.json(stagiaires);
  });
});
app.listen(port, () => {
  console.log(`Serveur démarré sur le port ${port}`);
});
```

Route GET pour récupérer un stagiaire par ID:

```
app.get('/stagiaires/:id', (req, res) => {
  Stagiaire.findById(req.params.id, (err, stagiaire) => {
    if (err) return res.status(500).send(err);
    if (!stagiaire) return res.status(404).send('Stagiaire non trouvé');
    res.json(stagiaire);
  });
});
```

Route DELETE pour supprimer un stagiaire:

```
app.delete('/stagiaires/:id', (req, res) => {
  Stagiaire.findByIdAndRemove(req.params.id, (err, stagiaire) => {
    if (err) return res.status(500).send(err);
    if (!stagiaire) return res.status(404).send('Stagiaire non trouvé');
    res.json({ message: 'Stagiaire supprimé' });
  });
});
```

Route POST pour créer un nouveau stagiaire:

```
app.post('/stagiaires', (req, res) => {
  const stagiaire = new Stagiaire({
    nom: req.body.nom,
    filiere: req.body.filiere });
  stagiaire.save((err) => {
    if (err) return res.status(500).send(err);
    res.json(stagiaire); }); });
```

Route PUT pour mettre à jour un stagiaire existant

```
app.put('/stagiaires/:id', (req, res) => {
  Stagiaire.findByIdAndUpdate(req.params.id, {
    nom: req.body.nom,
    filiere: req.body.filiere
  }, { new: true }, (err, stagiaire) => {
    if (err) return res.status(500).send(err);
    if (!stagiaire) return res.status(404).send('Stagiaire non trouvé');
    res.json(stagiaire); }); });
```

Un projet est une action entreprise pour répondre à un besoin défini dans un délai déterminé, impliquant la mobilisation de ressources identifiées, y compris les coûts humains et matériels.
La gestion de projet est une organisation méthodologique en opération pour assurer la coordination des acteurs et des tâches en un souci d'efficacité et de rentabilité.
Une ressource est un élément indispensable pour la réalisation de un projet, et d'une planification mal afectée peut entraîner un manque pendant le projet, retards sur certaines échéances et retards de livraison finale.

Le livrable du projet est le tangible d'une production réelle, appréhendable et mesurable attendue par le client final, et peut avoir plusieurs livrables, y compris les réalisations intermédiaires.
Le cahier des charges est un document officiel définissant l'ensemble des travaux pour un fournisseur, spécifiant les livrables, les coûts et le calendrier, et doit être clair pour toutes les parties prenantes.

Une charte de projet est un document permettant aux participants au projet d'indiquer leurs engagements, garantissant un projet fluide et des contributions constructives de tous les participants.
Les parties prenantes d'un projet sont les personnes et organisations directement impliquées dans le projet, impactées par la problématique initiale, le choix de la solution ou la mise en œuvre.
Le chef de projet informatique (CPI) est un expert en informatique, responsable de l'avancement et ajuster les évolutions et besoins d'un projet informatique. Le CPI compte plusieurs techniciens et ingénieurs en charge, et doit gérer les défis et ajuster les capacités techniques et managériales.
Matrice d'assignation des responsabilités
La RACI étudie la répartition des responsabilités dans des projets et processus transversaux, assurant l'attribution des responsabilités à chaque membre de l'équipe.
responsable/accountable(redevable)/consulted/informed.
Les contraintes de projet sont les limites générales de un projet, telles que les délais, les coûts et les risques, et sont crucials pour les performances de ce dernier.
La catégorie des méthodes prévisibles (cascades, V, Y) implique une organisation stricte du travail et un fonctionnement étape par étape, garantissant que les tâches sont terminées dans les délais et dans les limites des objectifs fixés.
Le modèle en cascade, appelé Waterfall en anglais, tel qu'appliqué aux projets, est une approche linéaire et séquentielle des différentes phases et activités du projet nécessaires à la livraison du ou des livrables.
La méthode offre une planification précise dès le départ, mais se heurte à des défis dans les projets complexes, à de faibles marges d'ajustement, à l'intégration des utilisateurs et à la détection occasionnelle d'erreurs à la fin du processus de développement.
Le cycle de vie en Y consiste à analyser les besoins fonctionnels et techniques, en combinant leurs résultats pour former un processus de développement en forme de Y.
Le cycle en V est un modèle de gestion de projet en cascade, comprisant linéaire processus de développement et associer à chaque phase de réalisation une phase de validation.
Manifeste Agile est une déclaration rédigée par des experts en 2001 pour améliorer le développement de logiciels.
Les 4 valeurs agiles:

- Les individus et les interactions plus que les processus et les outils
- Des logiciels opérationnels plus qu'une documentation exhaustive
- La collaboration avec les clients plus que la négociation contractuelle
- L'adaptation au changement plus que le suivi d'un plan

Le Manifeste définit 12 principes
Satisfaire la clientèle en priorité -- Accueillir favorablement les demandes de changement -- Livrer le plus souvent possible des versions opérationnelles de l'application -- Livrer le plus souvent possible des versions opérationnelles de l'application -- Construire autour de personnes motivées -- Privilégier la conversation en face-àface--Mesurer l'avancement du projet en matière de fonctionnalité de l'application-- Faire avancer le projet à un rythme soutenable et constant -- Porter une attention continue à l'excellence technique et à la conception --Faire simple--Responsabiliser les équipes --Ajuster à intervalles réguliers son comportement et ses processus pour être plus efficace
La méthodologie agile se concentre sur la planification des détails du projet, en divisant les tâches en objectifs à court terme et en autorisant les retards et les changements, ce qui la rend plus flexible et efficace que les méthodes de planification traditionnelles.

La méthode PERT permet de visualiser la dépendance des tâches et de procéder à leur ordonnancement en utilisant un graphe de dépendances.

La méthode pert implique l'identification des tâches, la détermination de la durée du projet, le temps disponible, les délais, la planification de l'exécution et de l'exécution des tâches, ainsi que la gestion de la logistique et des ressources humaines impliquées dans le projet.
GANTT est un graphique qui représente les tâches chronologiquement en utilisant des contraintes de succession. Il est facile à lire mais ne représente pas une charge de tâche. Il est couramment utilisé aux côtés des réseaux MPM.
Scrum est une méthodologie de gestion de projet populaire, basée sur les principes Agile. Il fonctionne sur des sprints, qui sont des cycles de réunions appelés « sprints ». À chaque sprint, l'équipe se réunit pour lister les tâches à exécuter, appelées « backlog de sprint ». Scrum se déploie autour d'acteurs spécifiques comme le Product Owner, et les réunions sont quotidiennes.

Les rôles scrum:
Product Owner : Représente les intérêts des parties prenantes et définit la vision du produit.
Scrum Master : Facilite le processus Scrum et aide l'équipe à être productive.
Équipe de développement : Pluridisciplinaire et auto-organisée, responsable de la livraison du produit.

Les événements :
Sprint : Itération de développement de 1 à 4 semaines.
Sprint Planning : Réunion de planification du prochain sprint.
Daily Scrum : Réunion quotidienne de coordination de 15 minutes.
Sprint Review : Démonstration du travail accompli à la fin du sprint.
Sprint Retrospective : Réflexion sur le processus et identification des axes d'amélioration.

Les artefacts :
Product Backlog : Liste ordonnée des fonctionnalités à développer.
Sprint Backlog : Liste des éléments sélectionnés pour le sprint en cours.
Incrément du Produit : Partie potentiellement livrable du produit à la fin d'un sprint.

Principes clés :

- Itérations courtes et livrables fonctionnels fréquents.
- Auto-organisation de l'équipe.
- Collaboration étroite avec le client.
- Amélioration continue du processus.

Avantages :

- Flexibilité et adaptation au changement.
- Livraison de valeur rapide et régulière.
- Transparence et visibilité du projet.
- Implication et motivation de l'équipe.

En tant que directeur RH J**e veux** être capable de gérer mes salariés **Afin de** garantir une meilleure organisation au sein de l'entreprise

Jira est un outil développé par Atlassian pour la gestion de projets, le suivi des problèmes et la planification des tâches.elle est largement utilisé dans les méthodes agiles comme Scrum et Kanban.

Principaux éléments jira :

- Projet : Unité de base pour organiser le travail dans Jira.
- Problème (Issue) : Correspond à une tâche, un bogue, une amélioration, etc.
- Tableau de bord (Dashboard) : Permet de visualiser et suivre l'avancement du projet.
- Workflow : Définit les étapes par lesquelles passe un problème (à faire, en cours, terminé, etc.).
- Sprints : Itérations de développement dans un contexte agile.
- Rapports : Divers rapports pour suivre l'avancement, la vélocité, etc.

Fonctionnalités principales jira :
Création et suivi des problèmes-- Planification et gestion des sprints-- Tableaux Kanban pour visualiser le workflow-- Intégrations avec d'autres outils (Git, Confluence, etc.)-- Personnalisation des workflows et des vues-- Permissions et rôles pour contrôler l'accès.

Avantages de Jira :
Centralisation de la gestion de projet// Visibilité et transparence du travail// Collaboration et coordination d'équipe// Suivi des problèmes et des tâches// Génération de rapports et analyses

Un Backlog est une liste de fonctionnalités ou d'éléments de travail. Ça peut être des tâches techniques ou des exigences non fonctionnelles.

Git est un système de contrôle de version distribué créé par Linus Torvalds en 2005.
Il permet de gérer efficacement le code source d'un projet, en particulier dans un environnement de développement collaboratif.

Concepts de base git:
Dépôt (Repository) : Répertoire contenant tous les fichiers du projet et leur historique.
Commit : Instantané du projet à un moment donné, avec un message décrivant les modifications.
Branch : Ligne de développement parallèle permettant d'expérimenter sans affecter la branche principale.
Merge : Fusion de deux branches pour intégrer les modifications.
Remote : Dépôt distant (ex: sur GitHub) permettant la collaboration.
commandes git:
git init : Initialise un nouveau dépôt Git
git add : Ajoute des modifications au prochain commit
git commit : Crée un nouveau commit avec un message
git push : Envoie les commits locaux vers un dépôt distant
git pull : Récupère les derniers commits depuis un dépôt distant
git branch : Crée, liste ou supprime des branches
git merge : Fusionne une branche dans une autre
Avantages de Git :
Historique complet des modifications// Branches légères permettant l'expérimentation//Système distribué facilitant la collaboration//Outils puissants pour la gestion des conflits// Intégration avec de nombreux outils de développement
GitLab est une plateforme web qui fournit un système de contrôle de version basé sur Git. En plus du stockage et du partage de code source, GitLab offre de nombreuses fonctionnalités pour les équipes de développement.
Principales fonctionnalités gitlab:
Gestion des dépôts Git avec des fonctionnalités avancées// Système de tickets/issues pour le suivi des tâches et des bugs// intégration continue (CI/CD) pour automatiser les tests et les déploiements// Revue de code avec commentaires et approbations// Wiki et pages web pour la documentation// Gestion des accès et des permissions// Tableau Kanban pour la gestion de projet// Intégrations avec de nombreux autres outils
Avantages de GitLab :
- Plateforme complète pour le cycle de vie du développement
- Collaboration facilitée grâce aux fonctionnalités intégrées
- Automatisation des processus avec l'intégration continue
- Contrôle total sur les données avec une solution auto-hébergée
- Grande communauté et écosystème d'extensions

Gestion des dépôts gitLab :
g*it clone https://gitlab.com/username/project.git :* Cloner un dépôt distant
git push origin master : Envoyer les commits locaux vers le dépôt distant
git pull : Récupérer les dernières modifications depuis le dépôt distant

Gestion des branches gitLab :
git checkout -b feature/new-functionality : Créer et basculer sur une nouvelle branche
git merge feature/new-functionality : Fusionner une branche dans la branche active
git push origin feature/new-functionality : Envoyer une nouvelle branche vers le dépôt distant
Gestion des issues gitLab:
gitlab issue create --title "Bug report" --description "The app is crashing" : Créer une nouvelle issue
gitlab issue list : Lister toutes les issues du projet
gitlab issue update 123 --assignee username : Assigner une issue à un utilisateur
Intégration continue (CI/CD) :
gitlab-runner register : Enregistrer un runner GitLab sur la machine locale
.gitlab-ci.yml : Définir les étapes de la pipeline CI/CD dans ce fichier
gitlab pipeline trigger : Déclencher manuellement une pipeline CI/CD
Autres commandes utiles :
gitlab project show myproject : Afficher les détails d'un projet
gitlab group create --name "My Group" : Créer un nouveau groupe
gitlab user list : Lister tous les utilisateurs de la GitLab instance

SonarQube est une plateforme open-source permettant d'analyser et de mesurer la qualité du code source.
Il s'intègre dans le cycle de développement logiciel pour détecter et suivre les problèmes de code.
• Une métrique : est une caractéristique (ou une propriété) d'une application.
• Métrique logicielle : mesure d'une propriété d'un logiciel (par exemple le nombre de lignes de codes).
Avantages de SonarQube :
Amélioration continue de la qualité du code
Détection précoce des problèmes dans le cycle de développement
Surveillance de la dette technique et de la couverture des tests
Principales fonctionnalités sonarQube:
- Analyse statique du code source (bugs, vulnérabilités, code smell, etc.)
- Rapports et dashboards pour visualiser la qualité du code
- Surveillance de la dette technique et du niveau de couverture des tests
- Gestion des problèmes avec un système de tickets intégré
- Supporté par de nombreux langages de programmation
- Intégration avec les outils de CI/CD comme Jenkins ou Azure Pipelines
- **Processus d'analyse avec SonarQube :**
- Configuration du projet dans SonarQube
- Exécution de l'analyse statique par SonarQube Scanner
- Visualisation des résultats dans l'interface web de SonarQube
- Suivi et résolution des problèmes détectés

•Les bugs : anomalies évidentes du code. Ils impactent la fiabilité (reliability) de l'application.
•Les vulnérabilités : faiblesses du code pouvant nuire au système. Elles impactent la sécurité de l'application.
•Les code smells : anti-patrons (ou anti-patterns). Ils impactent la maintenabilité de l'application ,en général les défauts pratiques dans le code d'un programme comme code dupliqué , répété , commentaires non nécessaires ...

DevOps est une philosophie, une culture et un ensemble de pratiques visant à rapprocher les équipes de développement et d'exploitation.
L'objectif est d'accélérer le cycle de vie du développement logiciel, de fournir des mises à jour plus fréquentes et de garantir une meilleure fiabilité.
Principaux aspects de DevOps :
- Communication et collaboration renforcées entre développeurs et opérateurs
- Automatisation des processus de construction, de test et de déploiement
- Monitoring et mesure continue de la performance des applications
- Itération rapide et livraison fréquente de nouvelles fonctionnalités
- Culture basée sur le partage de responsabilités et l'apprentissage continu

Outils et pratiques clés de DevOps :
- Intégration continue (CI) : Jenkins, Travis CI, GitHub Actions
- Livraison continue (CD) : Ansible, Puppet, Terraform, Kubernetes
- Monitoring et observabilité : Prometheus, Grafana, ELK Stack
- Gestion de configuration : Git, GitLab, Bitbucket
- Containers : Docker, Kubernetes
- Communication et collaboration : Slack, Microsoft Teams, Jira

Avantages de l'approche DevOps :
- Réduction du time-to-market et des coûts de développement
- Amélioration de la fiabilité et de la disponibilité des applications
- Facilitation de l'innovation et de l'expérimentation
- Meilleure visibilité et traçabilité des processus
- Engagement accru des équipes grâce à une culture collaborative

Composer est un logiciel gestionnaire de dépendances libre écrit en PHP. Il permet à ses utilisateurs de déclarer
et d'installer les bibliothèques dont le projet principal a besoin. Il permet de télécharger et de mettre à jour des
bibliothèques externes.

PHPUnit est un framework open-source de tests unitaires pour le langage de programmation PHP. Il permet d'écrire et d'exécuter des tests automatisés pour vérifier le bon fonctionnement du code.
Fonctionnalités clés de PHPUnit :
- Création de tests unitaires, de tests d'intégration et de tests fonctionnels
- Assertion de résultats attendus pour valider le comportement du code
- Gestion des dépendances et des données de test
- Rapports de résultats de tests avec statistiques et couverture de code
- Prise en charge des mocks, stubs et autres techniques d'isolation

Avantages de l'utilisation de PHPUnit :
- Amélioration de la fiabilité et de la maintenabilité du code
- Détection précoce des régressions et des bugs
- Documentation vivante du comportement attendu de l'application
- Facilitation de la refactorisation grâce aux tests unitaires
- Intégration possible avec des outils d'intégration continue

Mongodb:

Création de la base de données et de la collection

```
use GestionStagiaires
db.createCollection("stagiaires")
```

supprimer une collection d'une base de données:

```
db.nomcollection.drop()
```

supprimer la base de données courante: db.dropDatabase()

Utiliser la base de données: use myDB

Afficher tous les documents dans la collection: db.xxxxxx.find()

Afficher un document spécifique par son ID:

```
db.xxxxxxx.findOne({ _id: ObjectId("ID") })
```

Ajouter un nouveau document:

```
db.xxxxxx.insert({
  nom: "Jean Dupont",
  age: 22,
  entreprise: "Acme Corp",
  mission: "Développement web"
})
```

Mettre à jour un document existant:

```
db.xxxxxxx.update(
 { _id: ObjectId("IDENTIFIANT_DOCUMENT") },
 { $set: {
   age: 23,
   mission: "Développement mobile"
 }}
)
```

Supprimer un document:

```
db.xxxxxxxx.remove({ _id: ObjectId("IDENTIFIANT_DOCUMENT") })
```

Compter le nombre de documents dans la collection:

```
db.xxxxxxxx.count()
```

Afficher les bases de données du serveur: Show dbs

supprimer un document de la collection:

```
db.xxxxxxx.deleteOne({"cle1" :"valeur1"})
```

supprimer plusieurs documents répondants à une condition de la collection:

```
db.le_nom_de_la_collection.deleteMany({"cle1" :"valeur1"})
```

remplacer:

```
db.Filieres.update(
{"_id":"ID", "intitule":"Infrastructure digitale"},
{"_id":"ID", "intitule":"Infrastructure digitale", "nbOptions":3} )
```

Sélectionner tous les documents d'une collection:

```
db.le_nom_de_la_collection.find()
```

Afficher les documents avec une condition spécifique:

```
db.xxxxx.find({ entreprise: "Acme Corp" })
```

Trier les documents par un champ: db.xxxxx.find().sort({ nom: 1 })

Mettre à jour plusieurs documents:

```
db.xxxxx.updateMany(
  { entreprise: "Acme Corp" },
  { $inc: { age: 1 }}
)
```

Supprimer plusieurs documents:

```
db.xxxxxx.deleteMany({ age: { $gt: 25 } })
```

Créer un index sur un champ: db.xxxxxx.createIndex({ nom: 1 })

agrégation calcule la moyenne d'âge des stagiaires par entreprise.

```
db.stagiaire.aggregate([
  { $group: {
    _id: "$entreprise",
    moyenne_age: { $avg: "$age" }
  }}
])
```

Exporter la collection dans un fichier:

```
mongoexport --db mabase --collection stagiaire --out stagiaire.json
```

Importer des données depuis un fichier:

```
mongoimport --db mabase --collection stagiaire --file stagiaire.json
```

nombre de stagiaires en deuxième années

```
db.Stagiaires.find({"niveau":"2A"}).count()
```

Operateur	Description
$eq	***Equal*** : =
$gt	***Greater Than*** : >
$gte	***Greater Than or Equal*** : >=
$lt	***Less Than*** : <
$lte	***Less Than or Equal*** : <=
$ne	***Not Equal*** : ≠

Operateur	Description
$in	Appartient à la liste
$nin	N'appartient pas à la liste

Afficher les salariés triés par ordre croissant des noms :

```
db.salaries.find().sort({ "nomsal": 1 });
```

Afficher le nombre de salariés ayant la fonction "Technicien" :

```
db.salaries.countDocuments({ "fonction": "Technicien" });
```

Supprimer le salarié ayant l'id "s3" :

```
db.salaries.deleteOne({ "id": "s3" });
```

Afficher le nombre de salariés par fonction :

```
db.salaries.aggregate([
   { $group: { _id: "$fonction", count: { $count: {} } } },
   { $sort: { count: -1 } }
]);
```

Mysql

Creer un base de donnees: CREATE DATABASE dbTest

Sélectionner une base de données USE xxxxx;

Creer une table:

```
CREATE TABLE xxxxxx (
    id INT  PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL ,
    mot_de_passe VARCHAR(255) NOT NULL,
);
```

Insérer des données dans une table:

```
INSERT INTO nom_de_la_table (colonne1, colonne2, colonne3)
VALUES ('valeur1', 123, '2024-06-09');
```

Sélectionner des données depuis une table:

```
SELECT * FROM nom_de_la_table;
```

Sélectionner des données avec une condition:

```
SELECT * FROM nom_de_la_table WHERE colonne2 = 123;
```

Mettre à jour des données dans une table:

```
UPDATE nom_de_la_table
SET colonne1 = 'nouvelle_valeur'
WHERE colonne2 = 123;
```

Supprimer des données dans une table:

```
DELETE FROM nom_de_la_table WHERE colonne2 = 123;
```

Ajouter une colonne à une table existante:

```
ALTER TABLE nom_de_la_table ADD nouvelle_colonne VARCHAR(255);
```

Supprimer une colonne d'une table existante:

```
ALTER TABLE nom_de_la_table DROP COLUMN colonne_a_supprimer;
```

Modifier le type de données d'une colonne existante:

```
ALTER TABLE nom_de_la_table MODIFY colonne_existant INT;
```

Supprimer une table: DROP TABLE nom_de_la_table;

Supprimer une base de données: DROP DATABASE nom_de_la_base;

Créer un index:

```
CREATE INDEX nom_de_l_index ON nom_de_la_table (colonne1);
```

Supprimer un index: DROP INDEX nom__index ON nom_de_la_table;

Addition: SELECT colonne1 + colonne2 AS somme FROM nom__table;

Soustraction: SELECT colonne1 - colonne2 AS difference FROM _table;

Multiplication: SELECT colonne1 * colonne2 AS produit FROM _table;

Division: SELECT colonne1 / colonne2 AS quotient FROM table;

Somme de valeurs: SELECT SUM(colonne1) AS somme_totale FROM xx;

Moyenne de valeurs: SELECT AVG(colonne1) AS moyenne FROM xx;

minimale: SELECT MIN(colonne1) AS valeur_minimale FROM n_table;

maximale: SELECT MAX(colonne1) AS valeur_maximale FROM ntable;

Compter les lignes: SELECT COUNT(*) AS nombre_lignes FROM ntable;

procédure add employe:

```
DELIMITER //
CREATE PROCEDURE add_employee(IN emp_name VARCHAR(100), IN
emp_salary DECIMAL(10, 2))
BEGIN
   INSERT INTO employees (name, salary)
   VALUES (emp_name, emp_salary);
END //
DELIMITER ;
```

Utilisation de la procédure: CALL addemployee('John Doe', 50000.00);

Procédure avec gestion d'erreur:

```
DELIMITER //
CREATE PROCEDURE add_employee(IN emp_name VARCHAR(100), IN
emp_salary DECIMAL(10, 2))
BEGIN
   -- Vérifie que le salaire est positif
   IF emp_salary < 0 THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Le salaire doit
être positif';
   ELSE
      INSERT INTO employees (name, salary)
      VALUES (emp_name, emp_salary);
   END IF;
END //
DELIMITER ;
```

trigger after_employee_insert:

```
CREATE TRIGGER after_employee_insert
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
   INSERT INTO audit_log (action, employee_id, employee_name,
employee_salary)
   VALUES ('INSERT', NEW.id, NEW.name, NEW.salary);
END;
```

Test du trigger:

```
INSERT INTO employees (name, salary) VALUES ('John Doe', 50000.00);
```

trigger BEFORE UPDATE:

```
CREATE TRIGGER before_employee_update
BEFORE UPDATE ON employees
FOR EACH ROW
BEGIN
   INSERT INTO audit_log (action, employee_id, employee_name,
employee_salary)
   VALUES ('UPDATE', OLD.id, OLD.name, OLD.salary);
END;
```

Trigger AFTER UPDATE

```
DELIMITER //
CREATE TRIGGER after_employee_update
AFTER UPDATE ON employees
FOR EACH ROW
BEGIN
   INSERT INTO audit_log (action, employee_id, employee_name,
employee_salary)
   VALUES ('AFTER UPDATE', NEW.id, NEW.name, NEW.salary);
END //
DELIMITER ;
```

Trigger AFTER INSERT

```
DELIMITER //
CREATE TRIGGER after_employee_insert
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
   INSERT INTO audit_log (action, employee_id, employee_name,
employee_salary)
   VALUES ('AFTER INSERT', NEW.id, NEW.name, NEW.salary);
END //
DELIMITER ;
```

Trigger BEFORE INSERT

```
DELIMITER //
CREATE TRIGGER before_employee_insert
BEFORE INSERT ON employees
FOR EACH ROW
BEGIN
   INSERT INTO audit_log (action, employee_id, employee_name,
employee_salary)
   VALUES ('BEFORE INSERT', NEW.id, NEW.name, NEW.salary);
END //
DELIMITER ;
```

Trigger BEFORE DELETE

```
DELIMITER //
CREATE TRIGGER before_employee_delete
BEFORE DELETE ON employees
FOR EACH ROW
BEGIN
   INSERT INTO audit_log (action, employee_id, employee_name,
employee_salary)
   VALUES ('BEFORE DELETE', OLD.id, OLD.name, OLD.salary);
END //
DELIMITER ;
```

Trigger AFTER DELETE

```
DELIMITER //
CREATE TRIGGER after_employee_delete
AFTER DELETE ON employees
FOR EACH ROW
BEGIN
   INSERT INTO audit_log (action, employee_id, employee_name,
employee_salary)
   VALUES ('AFTER DELETE', OLD.id, OLD.name, OLD.salary);
END //
DELIMITER ;
```

procedure

```
DELIMITER //
CREATE PROCEDURE NomDeLaProcedure (IN param1 INT, OUT
param2 VARCHAR(50))
BEGIN
   -- Déclarations de variables locales
   DECLARE var1 INT;
   -- Instructions SQL
   SET var1 = param1 + 10;
   SET param2 = CONCAT('Le résultat est ', var1);
END //
DELIMITER ;
```

curseur

```
DELIMITER //
CREATE PROCEDURE UtiliserCurseur()
BEGIN
   DECLARE done INT DEFAULT 0;
   DECLARE id INT;
   DECLARE nom VARCHAR(50);
   -- Déclaration du curseur
   DECLARE curseur CURSOR FOR
      SELECT id, nom FROM votre_table;
   -- Gestion des exceptions
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
   -- Ouverture du curseur
   OPEN curseur;
   -- Boucle de lecture des lignes
   lireBoucle: LOOP
      FETCH curseur INTO id, nom;
      IF done THEN
         LEAVE lireBoucle;
      END IF;
      -- Ici, vous pouvez utiliser les valeurs lues
      SELECT CONCAT('ID: ', id, ', Nom: ', nom);
   END LOOP lireBoucle;
   -- Fermeture du curseur
   CLOSE curseur;
END //
DELIMITER ;
```