CVVEN (Python)

Gestion des Activités

1. Analyse des Besoins

Objectif : Comprendre les fonctionnalités nécessaires pour gérer les activités et animations dans les villages.

Fonctionnalités principales :

- **Gestion des activités** : Création, modification, suppression des activités (ex : randonnées, ateliers, soirées dansantes).
- **Gestion des intervenants** : Ajout, modification, suppression des intervenants (ex : guides, animateurs).
- **Réservation des activités** : Les participants peuvent réserver des activités en ligne.
- **Calendrier des activités** : Affichage des activités disponibles par date et par village.
- **Reporting** : Génération de rapports sur la participation aux activités.

2. Architecture du Module

Objectif : Définir l'architecture technique du module.

Technologies :

- **Backend** : Python Flask (framework web léger et flexible).
- **Base de données** : MongoDB (base de données NoSQL pour stocker les données des activités, intervenants, et réservations).
- **Frontend** : HTML, CSS, JavaScript (avec Bootstrap pour une interface responsive).
- **API** : Flask-RESTful pour exposer des endpoints API.

Structure du projet :

```
activites_module/
│
├── app.py                # Point d'entrée de l'application Flask
├── requirements.txt      # Dépendances Python
├── models/               # Modèles MongoDB
│   ├── activite.py       # Modèle pour les activités
│   ├── intervenant.py    # Modèle pour les intervenants
│   └── reservation.py    # Modèle pour les réservations
├── routes/               # Routes Flask
│   ├── activite_routes.py  # Routes pour les activités
│   ├── intervenant_routes.py # Routes pour les intervenants
│   └── reservation_routes.py # Routes pour les réservations
├── templates/            # Templates HTML
│   ├── activites.html    # Page de gestion des activités
│   ├── intervenants.html   # Page de gestion des intervenants
│   └── reservations.html   # Page de gestion des réservations
└── static/               # Fichiers statiques (CSS, JS, images)
    ├── css/
    └── js/
```

---

3. Développement du Module

Objectif : Implémenter les fonctionnalités du module étape par étape.

Étape 1 : Configuration de l'environnement

- Installer Python et Flask.
- Installer MongoDB et configurer la base de données.
- Créer un environnement virtuel et installer les dépendances (`Flask`, `Flask-RESTful`, `pymongo`).

Étape 2 : Modèles MongoDB

- **Activité** :

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')
```

```
db = client['cvven']
activites_collection = db['activites']

class Activite:
    def __init__(self, nom, description, date, village, intervenant_id):
        self.nom = nom
        self.description = description
        self.date = date
        self.village = village
        self.intervenant_id = intervenant_id

    def save(self):
        activites_collection.insert_one(self.__dict__)
```

- **Intervenant** :

```
intervenants_collection = db['intervenants']

class Intervenant:
    def __init__(self, nom, specialite):
        self.nom = nom
        self.specialite = specialite

    def save(self):
        intervenants_collection.insert_one(self.__dict__)
```

- **Réservation** :

```
reservations_collection = db['reservations']

class Reservation:
    def __init__(self, activite_id, participant_nom, participant_email):
        self.activite_id = activite_id
        self.participant_nom = participant_nom
        self.participant_email = participant_email
```

```
    def save(self):
        reservations_collection.insert_one(self.__dict__)
```

Étape 3 : Routes Flask

- **Routes pour les activités** :

```
from flask import Flask, request, jsonify
from models.activite import Activite

app = Flask(__name__)

@app.route('/activites', methods=['GET'])
def get_activites():
    activites = list(activites_collection.find())
    return jsonify(activites)

@app.route('/activites', methods=['POST'])
def add_activite():
    data = request.json
    activite = Activite(data['nom'], data['description'], data['date'], data
['village'], data['intervenant_id'])
    activite.save()
    return jsonify({"message": "Activité ajoutée avec succès"}), 201
```

- **Routes pour les intervenants** :

```
from models.intervenant import Intervenant

@app.route('/intervenants', methods=['GET'])
def get_intervenants():
    intervenants = list(intervenants_collection.find())
    return jsonify(intervenants)

@app.route('/intervenants', methods=['POST'])
def add_intervenant():
    data = request.json
    intervenant = Intervenant(data['nom'], data['specialite'])
```

```
    intervenant.save()
    return jsonify({"message": "Intervenant ajouté avec succès"}), 201
```

- **Routes pour les réservations** :

```
from models.reservation import Reservation

@app.route('/reservations', methods=['POST'])
def add_reservation():
    data = request.json
    reservation = Reservation(data['activite_id'], data['participant_nom'],
data['participant_email'])
    reservation.save()
    return jsonify({"message": "Réservation effectuée avec succès"}), 2
01
```

Étape 4 : Interface Utilisateur (Frontend)

- **Page de gestion des activités** :
  - Formulaire pour ajouter une nouvelle activité.
  - Liste des activités disponibles avec possibilité de modification/suppression.
- **Page de gestion des intervenants** :
  - Formulaire pour ajouter un nouvel intervenant.
  - Liste des intervenants avec possibilité de modification/suppression.
- **Page de réservation des activités** :
  - Formulaire pour réserver une activité.
  - Affichage des activités disponibles avec leur date et lieu.

Étape 5 : Calendrier des activités

- Intégrer un calendrier interactif (ex : FullCalendar.js) pour afficher les activités par date.
- Permettre aux utilisateurs de filtrer les activités par village.

Étape 6 : Reporting

- Générer des rapports sur la participation aux activités (nombre de participants, activités les plus populaires).
- Exporter les rapports en format CSV ou PDF.

---

4. Tests et Validation

Objectif : Assurer que le module fonctionne correctement et répond aux besoins.

Tests unitaires :

- Tester chaque modèle (Activité, Intervenant, Réservation) et chaque route Flask.
- Utiliser `unittest` ou `pytest` pour les tests unitaires.

Tests d'intégration :

- Tester l'interaction entre les différentes parties du module (ex : réservation d'une activité avec un intervenant).

Validation utilisateur :

- Faire tester le module par les utilisateurs finaux (administrateurs, participants) pour valider les fonctionnalités.