Voici une documentation synthétique récapitulant les points clés et les bonnes pratiques abordés lors de notre échange, afin de disposer d’un guide de référence pour vos futurs développements.

**Préparation de l’Environnement de Développement**

1. **Structure du projet**  
   Assurez-vous d’avoir une structure claire et cohérente. Dans le cadre de ce projet, la structure retenue est la suivante :

scss

Copier le code

Projet OPTIQ - DevOPTIQ/

├─ Archives/

├─ Bases de données/

├─ Code/

│ ├─ app.py

│ ├─ extensions.py

│ ├─ models/

│ │ ├─ models.py

│ ├─ routes/

│ │ ├─ activities.py

│ ├─ instance/ (répertoire contenant la base de données optiq.db)

│ ├─ \_\_init\_\_.py (facultatif)

├─ requirements.txt

└─ ...

1. **Environnement virtuel (venv)**  
   Création et activation (sous Windows) :

powershell

Copier le code

python -m venv Venv

.\Venv\Scripts\Activate

Une fois l’environnement virtuel activé, (Venv) apparaît avant le prompt PowerShell. Pour le désactiver :

powershell

Copier le code

deactivate

1. **Installation des dépendances**  
   À la racine du projet (là où se trouve requirements.txt):

powershell

Copier le code

pip install -r requirements.txt

Assurez-vous de lancer cette commande avec l’environnement virtuel activé.

**Configuration et Lancement de l’Application Flask**

1. **Fichier extensions.py**  
   Ce fichier crée une instance de SQLAlchemy utilisée par toute l’application :

python

Copier le code

from flask\_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

1. **Fichier models/models.py**  
   Déclaration d’un modèle Activity :

python

Copier le code

from extensions import db

class Activity(db.Model):

\_\_tablename\_\_ = 'activities'

id = db.Column(db.Integer, primary\_key=True)

name = db.Column(db.String(100), nullable=False)

* + Importer db depuis extensions.py.
  + Hériter du modèle db.Model.
  + Déclarer les champs et leurs types.

1. **Fichier routes/activities.py**  
   Définition des endpoints Flask pour gérer les activités :

python

Copier le code

from flask import Blueprint, jsonify, request

from extensions import db

from models.models import Activity

activities\_bp = Blueprint('activities', \_\_name\_\_)

@activities\_bp.route('/activities', methods=['GET'])

def get\_activities():

activities = Activity.query.all()

return jsonify([{"id": a.id, "name": a.name} for a in activities])

@activities\_bp.route('/activities', methods=['POST'])

def create\_activity():

data = request.get\_json()

new\_activity = Activity(name=data['name'])

db.session.add(new\_activity)

db.session.commit()

return jsonify({"id": new\_activity.id, "name": new\_activity.name}), 201

1. **Fichier app.py**  
   Initialisation de l’application et de la base de données :

python

Copier le code

import os

from flask import Flask

from extensions import db

from models.models import Activity

from routes.activities import activities\_bp

from sqlalchemy import text

app = Flask(\_\_name\_\_)

app.config['SQLALCHEMY\_DATABASE\_URI'] = f"sqlite:///{os.path.join(os.path.dirname(\_\_file\_\_), 'instance', 'optiq.db')}"

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

db.init\_app(app)

app.register\_blueprint(activities\_bp)

@app.route('/')

def hello():

return 'Hello, Flask!'

if \_\_name\_\_ == '\_\_main\_\_':

instance\_path = os.path.join(os.path.dirname(\_\_file\_\_), 'instance')

if not os.path.exists(instance\_path):

os.makedirs(instance\_path)

with app.app\_context():

db.create\_all()

try:

db.session.execute(text('SELECT 1'))

print("Connexion à la base de données réussie.")

except Exception as e:

print(f"Erreur lors de la connexion à la base de données : {e}")

app.run(debug=True)

Points importants :

* + db.init\_app(app) doit être appelé avant l’utilisation de db (et avant d’enregistrer les blueprints).
  + L’initialisation de la base se fait dans un contexte d’application (with app.app\_context(): db.create\_all()).
  + Le fichier optiq.db sera créé dans Code/instance.

**Lancement de l’Application**

1. Se placer dans le répertoire Code :

powershell

Copier le code

cd C:\Users\Hubert.AFDEC\A.F.D.E.C\Projet OPTIQ - DevOPTIQ\code

1. Activer l’environnement virtuel :

powershell

Copier le code

.\Venv\Scripts\Activate

1. Lancer l’application :

powershell

Copier le code

python app.py

L’application démarre sur <http://127.0.0.1:5000>.

1. Tester le endpoint principal :
   * Ouvrir un navigateur : <http://127.0.0.1:5000>  
     Vous devriez voir "Hello, Flask!".
   * Tester la liste des activités : http://127.0.0.1:5000/activities  
     Retourne [] si la liste est vide.
2. Ajouter une activité (dans un **second terminal**, sans arrêter l’application) :

powershell

Copier le code

Invoke-RestMethod -Uri "http://127.0.0.1:5000/activities" -Method POST -ContentType "application/json" -Body '{"name":"Mon Activite"}'

Note : Éviter les accents si vous rencontrez des problèmes d’encodage. Cela retournera un statut 201 CREATED avec l’ID et le nom de l’activité.

1. Retourner sur http://127.0.0.1:5000/activities pour voir l’activité nouvellement créée.

**Erreurs Courantes et Bonnes Pratiques**

1. **Erreur "unable to open database file"** :  
   Vérifier que le répertoire instance existe dans le même répertoire que app.py. Si nécessaire, le créer. S’assurer que le chemin dans SQLALCHEMY\_DATABASE\_URI est correct.
2. **Erreur "The current Flask app is not registered with this 'SQLAlchemy' instance."** :
   * Vérifier que db.init\_app(app) est appelé avant l’utilisation de db.
   * Vérifier que Activity hérite de db.Model et que db est bien importé depuis extensions.
   * S’assurer que les blueprints sont enregistrés après db.init\_app(app).
3. **Erreur lors de l’envoi de requêtes POST avec accents** :  
   Sous PowerShell, préférer l’usage d’Invoke-RestMethod et éviter les accents ou spécifier un encodage.  
   Exemple sans accents : -Body '{"name":"Mon Activite"}' fonctionne mieux que -Body '{"name":"Mon Activité"}'.
4. **Ne pas arrêter le serveur avant de tester un endpoint** :  
   L’application Flask doit rester en cours d’exécution pour que les requêtes HTTP fonctionnent.

**Étapes Suivantes**

* Créer d’autres endpoints (PUT, DELETE) pour éditer ou supprimer une activité.
* Ajouter des tests unitaires avec pytest.
* Implémenter des migrations de base de données avec Flask-Migrate.
* Améliorer la gestion des données (validation, schémas, etc.).

Ce guide récapitule les principales étapes, configurations et corrections apportées, ainsi que les erreurs courantes et les solutions pour s’en prémunir. Il devrait vous servir de référence lors de vos prochains développements.