TD – Flask, Jinja2 et ORM avec SQLAlchemy

École Supérieure de Technologie d'Essaouira
Université Cadi Ayyad – Filière 3IASD

Objectifs du TD

Ce TD a pour but de mettre en pratique les notions des deux premiers chapitres :
— Chapitre 1 : flux de rendu Flask + Jinja2, modèles (templates), variables, filtres, structures de contrôle, routes HTTP.
— Chapitre 2 : modèles de données, ORM avec Flask-SQLAlchemy, opérations CRUD.
À la fin du TD, l'étudiant(e) devra être capable de :
— créer une petite application Flask qui affiche des données dans des templates Jinja2 ;
— comprendre et utiliser le flux de rendu (route → données Python → template) ;
— définir un modèle ORM simple (`Task`) avec SQLAlchemy ;
— effectuer des opérations CRUD (Create, Read, Update, Delete) via des routes Flask ;
— combiner templates Jinja2 et ORM pour construire une mini application de gestion de tâches.

1 Contexte du TD

Vous allez développer une mini application web de gestion de tâches (*to-do list*) avec Flask.
— Chaque **tâche** possède au minimum un **titre** et un état **fait** / **non fait**.
— Les tâches seront affichées dans une page HTML générée avec Jinja2.
— Dans un premier temps, les données pourront être codées en dur (liste Python).
— Dans un second temps, vous utiliserez **SQLAlchemy** pour stocker les tâches dans une base SQLite.

Remarque : Il est conseillé de progresser dans l'ordre des exercices. Les parties marquées « Bonus » sont facultatives (pour aller plus loin).

2 Partie A : Mise en place de Flask et Jinja2

Exercice 1 : Structure minimale du projet

1. Créez un dossier de projet nommé par exemple `td_flask_todo`.
2. Dans ce dossier, créez un environnement virtuel et activez-le.
3. Installez Flask dans cet environnement :

```
pip install flask
```

4. Créez un fichier `app.py`. Dans ce fichier :
   (a) importez Flask ;
   (b) créez un objet `app = Flask(__name__)` ;
   (c) définissez une route `"/"` qui renvoie simplement la chaîne « Hello Flask TD » (sans template pour le moment) ;
   (d) lancez l'application en mode debug.

Questions :
— Expliquez, en quelques phrases, le rôle de l'objet `app`.
— Que se passe-t-il lorsqu'on accède à l'URL `/` dans le navigateur ?

Exercice 2 : Premier template Jinja2

1. Créez un dossier `templates/` dans votre projet.
2. Dans ce dossier, créez un fichier `home.html` contenant une structure HTML minimale. Par exemple :

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>TD Flask TODO</title>
</head>
<body>
<h1>Bienvenue dans le TD Flask</h1>
<p>Page d'accueil.</p>
</body>
</html>
```

3. Modifiez votre route `"/"` de `app.py` pour utiliser `render_template("home.html")`.

Questions :
— Décrivez le **flux de rendu** entre la route `"/"` et le fichier `home.html`.
— Que se passe-t-il si vous changez le nom du fichier `home.html` sans modifier la route ?

Exercice 3 : Variables et structures de contrôle dans le template

On souhaite maintenant afficher une **liste de tâches** simple dans la page d'accueil.

1. Dans `app.py`, modifiez la route `"/"` pour définir une liste Python de tâches, par exemple :

```
tasks = [
{"id": 1, "title": "Lire l'enonce du TD",
   "done": False},
{"id": 2, "title": "Installer Flask", "
   done": True},
{"id": 3, "title": "Ecrire le premier
   template", "done": False},
]
```

2. Passez cette liste au template `home.html` via `render_template`, par exemple :

```
return render_template("home.html", tasks=
   tasks)
```

3. Dans `home.html`, modifiez le corps pour :
   — afficher un titre « Liste des taches » ;
   — utiliser une boucle `{% for ... %}` pour afficher chaque tâche dans une liste `<ul>` ;
   — distinguer visuellement les tâches faites et non faites (par exemple, barrer le texte si `done` vaut `True`).

Questions :
— Donnez un exemple de **condition** Jinja2 que vous utilisez pour distinguer l'affichage fait / non fait.
— Expliquez pourquoi la liste `tasks` doit être construite en Python et non directement dans le template.

Exercice 4 : Héritage de templates (base.html)

Pour éviter de répéter le code HTML commun (balises `html`, `head`, etc.), on va utiliser l'héritage Jinja2.

1. Créez un fichier `base.html` dans le dossier `templates/`, avec :
   — la structure HTML de base ;
   — un bloc `title` pour le titre de la page ;
   — un bloc `content` pour le contenu principal.
2. Modifiez `home.html` pour :
   — déclarer `{% extends "base.html" %}` ;
   — remplir les blocs `title` et `content`.

Question : Expliquez en quoi l'héritage de modèles rend votre application plus facile à maintenir.

3 Partie B : Introduction à l'ORM avec SQLAlchemy

Nous allons remplacer la liste Python `tasks` par une vraie table en base de données.

Exercice 5 : Installation et configuration de Flask-SQLAlchemy

1. Dans votre environnement virtuel, installez Flask-SQLAlchemy :

```
pip install flask_sqlalchemy
```

2. Dans `app.py`, configurez la base de données SQLite :

```
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///todo.db
    "
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False

db = SQLAlchemy(app)
```

3. Expliquez le rôle de l'objet `db`.

Exercice 6 : Définition du modèle Task

1. Dans `app.py`, définissez une classe `Task` héritant de `db.Model`, avec les colonnes :
   — `id` (Integer, clé primaire) ;
   — `title` (String(200), non nul) ;
   — `done` (Boolean, non nul, valeur par défaut `False`).
2. Ajoutez une méthode `to_dict()` qui renvoie un dictionnaire avec les champs `id`, `title` et `done`.
3. Créez la base et la table en ajoutant :

```
with app.app_context():
db.create_all()
```

   puis en lançant `python app.py` une première fois.

Questions :
— Où est créé le fichier de base de données ?
— Que se passe-t-il si vous réexécutez `db.create_all()` alors que la table existe déjà ?

Exercice 7 : Insérer quelques tâches en base

1. Créez, dans `app.py` (ou dans un petit script séparé), quelques tâches initiales :

```
t1 = Task(title="Premiere tache", done=False)
t2 = Task(title="Deuxieme tache", done=True)

db.session.add(t1)
db.session.add(t2)
db.session.commit()
```

2. Vérifiez que ces tâches sont bien créées (par exemple en ajoutant temporairement une route de test qui fait un `Task.query.all()` et affiche le résultat).

Question : Expliquez le rôle de `db.session.add()` et `db.session.commit()`.

4 Partie C : Lier templates et ORM

Exercice 8 : Liste des tâches depuis la base de données

Modifiez la route `"/"` pour qu'elle lise les tâches depuis la base plutôt que depuis la liste Python.

1. Supprimez la liste `tasks` codée en dur.
2. Dans la route `"/"`, utilisez :

```
tasks = Task.query.all()
```

3. Passez cette liste au template :

```
return render_template("home.html", tasks=tasks)
```

4. Modifiez `home.html` pour itérer sur une liste d'objets `Task` et non plus sur des dictionnaires Python. (Par exemple : `{{ task.title }}`, `{{ task.done }}`).

Question : Comparez les deux versions : sans ORM (liste de dicts) et avec ORM (liste d'objets). Quels avantages voyez-vous à la version ORM ?

Exercice 9 : Ajout d'une tâche via un formulaire (optionnel)

Objectif : ajouter un petit formulaire HTML pour créer une nouvelle tâche, en combinant :
— méthode HTTP POST ;
— reading des données de formulaire côté Flask ;
— insertion ORM dans la base ;
— redirection vers la page d'accueil (pattern Post/Redirect/Get).

1. Dans `home.html`, ajoutez en bas de page un formulaire simple :

```
<form action="{{ url_for('add_task') }}" method="post">
<input type="text" name="title" placeholder="Nouvelle
   tache">
<button type="submit">Ajouter</button>
</form>
```

2. Dans `app.py`, créez une route :
   — `@app.route("/add", methods=["POST"])` (par exemple) ;
   — qui lit `request.form.get("title")` ;
   — qui crée un objet `Task`, l'ajoute à la session et fait `commit()` ;

— puis redirige vers `"/"`.

Questions :

— Pourquoi est-il préférable d'utiliser une redirection après le `POST` ?
— Quelle méthode HTTP est utilisée pour le formulaire ? Aurait-on pu utiliser `GET` ici ?

5 Partie D : Questions de synthèse

1. Décrivez le **flux complet** lorsqu'un utilisateur :
   (a) accède à la page d'accueil pour voir la liste des tâches ;
   (b) soumet le formulaire d'ajout de tâche ;
   (c) est redirigé vers la liste mise à jour.
2. Citez trois avantages d'utiliser un **ORM** comme SQLAlchemy plutôt que du SQL brut partout.
3. Citez trois bonnes pratiques pour les templates Jinja2 dans une application Flask.
4. Proposez une amélioration possible de cette mini application (par exemple : filtrer les tâches, gérer une date d'échéance, marquer une tâche comme faite via un bouton, etc.).

Pour aller plus loin (bonus)

— Ajouter une page `/api/tasks` qui renvoie la liste des tâches au format JSON (utilisation de `jsonify` et `to_dict()`).
— Ajouter un champ `created_at` de type `DateTime` dans le modèle `Task`, avec une valeur par défaut.
— Mettre en place un tri des tâches par date de création.