Objectifs de la feuille

- Comprendre l'architecture MVT
- Mise en place d'un template
- Exemple de vue
- Un peu de style

Introduction

La structure d'un projet est une des clés qui garantit sa bonne évolution, exactement comme dans votre ordinateur ! Imaginez que vous repreniez le projet d'une autre personne, ou que vous ouvriez le vôtre dans six mois. Vous ne voulez pas perdre de temps à chercher comment il est organisé ! Vous voulez aller droit au but et apporter votre modification sans pour autant tout casser.
C'est pourquoi la plupart des frameworks pensés pour le web choisissent de séparer les fichiers dans plusieurs dossiers distincts. Cela peut paraître un peu contraignant au début mais ce sont de bonnes pratiques qui garantissent la bonne évolution de votre projet.
Flask, comme vous le savez désormais, n'impose pas vraiment de structure, il la propose. S'inspirant du modèle MVC (Modèle / Vue / Contrôleur) de bien des frameworks, il se base sur trois piliers fortement imbriqués : le modèle, la vue et le template.

Mise en place d'un template

Un template est un fichier HTML qui peut recevoir des objets Python et qui est lié à une vue (nous y reviendrons). Notre vue retourne directement un string représentant un document HTML. Souvent le contenu d'une page est dynamique, et pour cela il est pratique d'utiliser des templates. Flask s'attend à trouver les templates dans le sous-répertoire templates de monApp.
Concrètement, un template peut interpréter des variables et les afficher. Par exemple, nous pouvons "donner" la variable name au template index.html et ce dernier affichera à la place le prénom.
Le template n'a accès qu'aux variables que la vue lui transmet, et pas aux autres. Par exemple, le template n'a pas accès à la variable db.

Créez le template monApp/templates/index.html avec le contenu suivant :

```
<! doctype html >
<html>
    <head>
        <title>{{ title }}</title>
    </head>
    <body >
        <h1> Bienvenu {{name}} !!</h1>
    </body >
</html >
```

Ce template est paramétré par 2 variables title et name.

Exemple d'une vue

Les vues contenues dans le fichier views.py jouent un rôle primordial : elles décident du contenu à afficher sur une page. Plus spécifiquement, ce sont elles qui génèrent le contenu à renvoyer aux requêtes qui leur sont adressées.
Une vue est une fonction qui renvoie une réponse à une requête HTTP. Toute fonction décorée par @app.route est une vue. On appelle route l'URL à laquelle va répondre la vue (par ex : /index.)

Afin de bien comprendre le rôle majeur des vues, intéressons-nous au cheminement d'une requête.

- L'utilisateur tape http://cdal.com dans son navigateur puis appuie sur entrée. Cela génère une requête HTTP de cette forme :
  GET/HTTP/1.1
  Host: cdal.com
- Le serveur à l'adresse cdal.com reçoit la requête HTTP. Il cherche dans les vues la route correspondant à la requête en fonction de la méthode HTTP utilisée et de l'URL.
- Le serveur exécute la vue : éventuellement, si la route contient des paramètres, ils sont passés comme arguments à la vue. Si une modification d'un objet en base de données est demandée, la vue fait appel au modèle. Si un template est demandé, la vue l'appelle.
- Le serveur transmet la réponse HTTP de la vue au navigateur de l'utilisateur (appelé communément client).

Le protocole HTTP est très intéressant ! Tout le web que nous connaissons l'utilise.

Afin de mettre en pratique ce que nous venons d'explorer, affichons le contenu du template index.html lorsque l'utilisateur arrive sur la page d'accueil. Pour cela, il faut modifier la vue index( ) dans le fichier views.py :

```
from .app import app
from flask import render_template

@app.route('/')
@app.route('/index/')
def index():
    return render_template("index.html",title ="R3.01 Dev Web avec Flask",name="Cricri")
```

Vous voyez ci-dessus que nous pouvons associer une vue à plusieurs URL ! Il vous suffit de les indiquer les unes à la suite des autres. Pourquoi ajouter un slash à la fin de l'URL ? Les utilisateurs peuvent taper deux adresses différentes /index ou /index/. Hé oui, ce ne sont pas les mêmes ! Si vous déclarez une route sans le slash final (/index), le serveur renverra une erreur 404 (page non trouvée) si l'utilisateur tape /index/. Mais si vous déclarez une route avec le slash final (/index/) et que l'utilisateur tape /index, le serveur ne renverra pas d'erreur. Pensez-y !

Un peu de style

Pour ajouter du CSS, nous allons créer un fichier style.css dont le contenu est statique plutôt que dynamique. Flask s'attend à trouver les fichiers statiques dans monApp/static. Dans le fichier monApp/static/style.css nous plaçons le code suivant :

```
h1 {
        color : red;
}
```

Pour l'utiliser, Flask offre, par exemple, une méthode plutôt intéressante pour faire appel à des fichiers qui se trouvent dans le dossier static : url_for( ). Cette méthode est disponible dans le template. Que fait-elle ? Elle génère une URL en fonction des paramètres transmis. Le premier paramètre est le nom du dossier qui contient les fichiers que l'on souhaite lier. Le second, optionnel, est le nom d'un fichier. Par exemple :

```
url_for ('static', filename ='style.css')
```

Ceci générera le lien suivant : /static/style.css.
Intégrez-la dans le <head> du template index.html ! Mais si vous vous contentez d'insérer la méthode telle quelle dans le template, elle sera interprétée comme du HTML pur. Afin d'indiquer à Flask que vous insérez du Python, qui doit donc être interprété comme tel, entourez le code d'accolades. Comme ceci :

```
<link rel="stylesheet" href="{{ url_for ('static', filename ='style.css') }}">
```

Vérifiez que çela marche comme avant.

C'est à vous de jouer !

Faites en sorte que vos deux vues restantes about( ) et contact( ) rendent chacune un template about.html et contact.html, tous deux s'appuyant sur la feuille de style existante.
Vous avez le choix de la structure HTML de vos templates et du style que vous voulez appliquer sur les éléments des nouvelles pages HTML.

En résumé :
Un projet Flask se structure à partir de trois piliers : le modèle (TP n°2) , la vue et le template (TP n°3)
Un modèle représente la structure d'un objet de la base de données. Un template est un fichier HTML dans lequel on peut récupérer des objets Python. Une vue a la responsabilité de traiter le contenu à envoyer à l'URL indiquée par la route. Une route dans Flask se crée en utilisant @app.route.
Maintenant que votre application web est accessible depuis la route créée et affiche le template et le modèle, vérifions que vous avez compris les notions de Flask avec un quiz. Vous verrez ensuite comment organiser votre projet en templates.