TP NSI (1ère) – Réaliser un petit serveur Web avec Flask (localhost)

Objectif : comprendre le modèle client-serveur et créer des pages HTML dynamiques (Jinja), avec CSS et images correctement servis par Flask.

1. Prérequis et préparation du dossier

Bibliothèque : Flask (framework Web en Python).

Installation (au choix) :

- Avec pip (terminal) :

```
pip install flask
```

- Avec EduPython : Outils → Installer un module → taper flask.

Créer un dossier de projet, par exemple :

```
flask_tp/
```

Structure recommandée (très important pour le CSS et les images) :

```
flask_tp/
  app.py
  templates/
    index.html
    page2.html
    formulaire.html
    resultat.html
  static/
    style.css
    img/
      logo.png
```

Rappels :

- templates/ : contient les pages HTML (templates Jinja).
- static/ : contient les fichiers statiques (CSS, images, JS).

2. Étape 1 – Lancer un serveur minimal (page renvoyée par Python)

Créer le fichier app.py et taper :

```
from flask import Flask

app = Flask(__name__)  # crée l'application Flask

@app.route("/")        # URL racine : http://localhost:5000/
def index():
    return "<p>Le serveur Flask fonctionne !</p>"

if __name__ == "__main__":
    app.run()          # lance le serveur (localhost:5000 par défaut)
```

Explications ligne par ligne :

- from flask import Flask : on importe la classe Flask.
- app = Flask(__name__) : on crée l'application Web.
- @app.route("/") : décorateur = règle de routage (URL → fonction).
- def index(): ... : fonction exécutée quand le client demande /.
- app.run() : démarre le serveur intégré de Flask.

Test à faire :

- Exécuter app.py puis ouvrir http://localhost:5000/ dans le navigateur.

3. Étape 2 – Renvoyer une page HTML depuis templates/

Objectif : arrêter d'écrire le HTML dans Python, et utiliser un fichier HTML dans templates/.

1) Créer templates/index.html :

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Accueil</title>
  </head>
  <body>
    <h1>Accueil</h1>
    <p>Page servie depuis templates/index.html</p>
  </body>
</html>
```

2) Modifier app.py :

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/")
def index():
    return render_template("index.html")
```

```
if __name__ == "__main__":
    app.run()
```

- render_template("index.html") : Flask lit le fichier dans templates/ et l'envoie au navigateur.

Test à faire :

- Relancer app.py et vérifier que la page affichée correspond au fichier index.html.

4. Étape 3 – Ajouter une deuxième page et faire des liens (url_for)

Créer templates/page2.html :

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Page 2</title>
  </head>
  <body>
    <h1>Page 2</h1>
    <p><a href="{{ url_for('index') }}">Retour à l'accueil</a></p>
  </body>
</html>
```

Modifier templates/index.html pour ajouter un lien :

```
<p><a href="{{ url_for('page2') }}">Aller sur la page 2</a></p>
```

Ajouter la route dans app.py :

```
@app.route("/page2")
def page2():
    return render_template("page2.html")
```

Pourquoi url_for ?

- url_for('page2') fabrique l'URL à partir du nom de la fonction Python.
- Si on change l'URL dans @app.route(...), les liens restent corrects.

Test à faire :

- Aller sur http://localhost:5000/ puis cliquer sur le lien vers Page 2 (et retour).

5. Étape 4 – Ajouter du CSS (dossier static/)

Créer static/style.css :

```
body { font-family: Arial, sans-serif; }
h1 { text-decoration: underline; }
```

Dans templates/index.html, ajouter dans <head> :

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

Important :

- On ne crée PAS de route Flask pour le CSS.
- Flask sert automatiquement /static/... (fichiers statiques).

Test à faire :

- Actualiser la page et vérifier que le style s'applique (police / titre souligné).

6. Étape 5 – Ajouter une image (static/img/)

Placer une image dans static/img/logo.png (ou .jpg).

Dans templates/index.html :

```
<img src="{{ url_for('static', filename='img/logo.png') }}" alt="Logo"
width="120">
```

- Règle : toutes les images doivent être dans static/ et référencées avec url_for('static', filename=...).

Test à faire :

- Recharger la page : l'image doit s'afficher. Si elle ne s'affiche pas : vérifier le nom du fichier et le chemin.

7. Étape 6 – Page dynamique (Jinja + variables Python)

But : générer une page HTML différente selon des valeurs calculées côté serveur.

Dans app.py, ajouter :

```
from datetime import datetime

@app.route("/heure")
def heure():
    maintenant = datetime.now()
    return render_template(
        "heure.html",
        h=maintenant.hour,
        m=maintenant.minute,
        s=maintenant.second
    )
```

Créer templates/heure.html :

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Heure</title>
```

```
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
  </head>
  <body>
    <h1>Page dynamique</h1>
    <p>Il est {{ h }} h {{ m }} min {{ s }} s</p>
    <p><a href="{{ url_for('index') }}">Retour</a></p>
  </body>
</html>
```

- Les doubles accolades {{ ... }} sont du Jinja : Flask remplace ces emplacements par les valeurs passées à render_template.

Test à faire :

- Aller sur http://localhost:5000/heure puis actualiser : l'heure doit changer.

8. Étape 7 – Formulaire (POST) et lecture côté serveur

Créer templates/formulaire.html :

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Formulaire</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
  </head>
  <body>
    <h1>Formulaire</h1>

    <form action="{{ url_for('resultat') }}" method="post">
      <label>Nom : <input type="text" name="nom" required></label><br>
      <label>Prénom : <input type="text" name="prenom" required></label><br>
      <button type="submit">Envoyer</button>
    </form>

    <p><a href="{{ url_for('index') }}">Retour</a></p>
  </body>
</html>
```

Ajouter un lien vers le formulaire dans index.html :

```
<p><a href="{{ url_for('formulaire') }}">Aller au formulaire</a></p>
```

Dans app.py, ajouter les routes :

```
from flask import Flask, render_template, request
# ...

@app.route("/formulaire")
def formulaire():
    return render_template("formulaire.html")
```

```
@app.route("/resultat", methods=["POST"])
def resultat():
    nom = request.form.get("nom", "").strip()
    prenom = request.form.get("prenom", "").strip()
    return render_template("resultat.html", nom=nom, prenom=prenom)
```

Créer templates/resultat.html :

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Résultat</title>
  </head>
  <body>
    <h1>Résultat</h1>
    <p>Bonjour {{ prenom }} {{ nom }} !</p>
    <p><a href="{{ url_for('formulaire') }}">Revenir au formulaire</a></p>
  </body>
</html>
```

- POST envoie les données du formulaire dans le corps de la requête (moins visible que GET dans l'URL).

Test à faire :

- Aller sur /formulaire, remplir, envoyer : la page /resultat doit afficher Bonjour ...

9. Mini-projet – Petit répertoire (chercher + ajouter)

Objectif : reproduire un petit site de type répertoire :

- Page d'accueil
- Page de recherche (formulaire) → page résultat
- Page d'ajout (formulaire) → page résultat

9.1 – Le module de données (repertoire_web.py)

Créer repertoire_web.py (données très simples en dictionnaire).

```
# repertoire_web.py

_repertoire = {
    "Alice": "0600000001",
    "Bob": "0600000002"
}

def rechercher(nom: str) -> str:
    """Renvoie le numéro si le nom existe, sinon 'Inconnu'."""
    return _repertoire.get(nom, "Inconnu")
```

```
def ajouter(nom: str, numero: str) -> None:
    """Ajoute ou remplace une entrée du répertoire."""
    _repertoire[nom] = numero
```

9.2 – Les routes Flask (app.py)

```
from flask import Flask, render_template, request
from repertoire_web import ajouter, rechercher

app = Flask(__name__)

# Pages
@app.route("/")
def index():
    return render_template("index.html")

@app.route("/cherche")
def cherche():
    return render_template("cherche.html")

@app.route("/ajout")
def ajout():
    return render_template("ajout.html")

# Traitement des formulaires
@app.route("/resultat_recherche", methods=["POST"])
def resultat_recherche():
    nom = request.form.get("nom", "").strip()
    numero = rechercher(nom)
    return render_template("resultat_recherche.html", nom=nom, numero=numero)

@app.route("/resultat_ajout", methods=["POST"])
def resultat_ajout():
    nom = request.form.get("nom", "").strip()
    numero = request.form.get("numero", "").strip()
    ajouter(nom, numero)
    return render_template("resultat_ajout.html", nom=nom, numero=numero)

if __name__ == "__main__":
    app.run()
```

9.3 – Templates (exemples complets)

templates/index.html :

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Répertoire</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
  </head>
```

```
  <body>
    <h1>Répertoire</h1>
    <ul>
      <li><a href="{{ url_for('cherche') }}">Rechercher un numéro</a></li>
      <li><a href="{{ url_for('ajout') }}">Ajouter un contact</a></li>
    </ul>
  </body>
</html>
```

templates/cherche.html :

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Recherche</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
  </head>
  <body>
    <h1>Rechercher</h1>
    <form action="{{ url_for('resultat_recherche') }}" method="post">
      <label>Nom : <input name="nom" required></label>
      <button type="submit">Chercher</button>
    </form>
    <p><a href="{{ url_for('index') }}">Retour</a></p>
  </body>
</html>
```

templates/ajout.html :

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Ajout</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
  </head>
  <body>
    <h1>Ajouter</h1>
    <form action="{{ url_for('resultat_ajout') }}" method="post">
      <label>Nom : <input name="nom" required></label><br>
      <label>Numéro : <input name="numero" required></label><br>
      <button type="submit">Ajouter</button>
    </form>
    <p><a href="{{ url_for('index') }}">Retour</a></p>
  </body>
</html>
```

templates/resultat_recherche.html :

```
<!doctype html>
<html lang="fr">
  <head><meta charset="utf-8"><title>Résultat</title></head>
```

```
  <body>
    <h1>Résultat recherche</h1>
    <p>Nom : {{ nom }}</p>
    <p>Numéro : {{ numero }}</p>
    <p><a href="{{ url_for('cherche') }}">Nouvelle recherche</a></p>
  </body>
</html>
```

templates/resultat_ajout.html :

```
<!doctype html>
<html lang="fr">
  <head><meta charset="utf-8"><title>Ajout OK</title></head>
  <body>
    <h1>Contact ajouté</h1>
    <p>{{ nom }} → {{ numero }}</p>
    <p><a href="{{ url_for('index') }}">Retour accueil</a></p>
  </body>
</html>
```

Test à faire :

- Lancer app.py, aller sur /, tester recherche puis ajout puis re-recherche.

10. Bonus – Rendre le serveur visible sur le réseau du lycée

Par défaut, Flask écoute uniquement sur la machine locale. Pour écouter sur le réseau :

```
app.run(host="0.0.0.0", port=8080, debug=False)
```

Ensuite, sur un autre PC : http://IP_DU_SERVEUR:8080/

- Attention : pare-feu Windows et règles du réseau pédagogique peuvent bloquer le port.