



Développement d'une application web pour la visualisation de données démographiques

Ce tutoriel explique comment créer une application web simple utilisant Flask pour visualiser des données démographiques mondiales suivant différents modalités. Nous utilisons la base de données SQLite créée au cours de la SAE1.04 pour stocker les données.

Contexte du projet

Une première version de l'application web est mise à disposition. Ce code source sert de point de départ pour ce projet de développement. L'équipe de développement devra suivre les étapes décrites dans ce tutoriel pour comprendre la structure de l'application, les fonctionnalités mises en place, et poursuivre ce projet en proposant des améliorations et des fonctionnalités supplémentaires.

Les différents intervenants dans ce projet sont :

- **Maîtrise d'ouvrage (MO)** : Responsable de la définition des besoins et des exigences de l'application. Ce rôle est joué par l'enseignant (un client fictif).
- **Maîtrise d'oeuvre (MOE)** : Responsable de la conception, du développement et de la livraison de l'application. Ce rôle est joué par les étudiants (des développeurs).

L'équipe de développement est composée des rôles suivants :

- **Chef de projet** : Responsable de la coordination de l'équipe, de la planification et du suivi du projet.
- **Développeurs** : Responsables de l'écriture du code, des tests et de la documentation.
- **Testeurs** : Responsables de la validation des fonctionnalités et de la qualité de l'application.
- **Documentalistes** : Responsables de la rédaction de la documentation utilisateur et technique.
- **Responsable communication** : Responsable de la communication interne et externe autour du projet.

Prérequis techniques

- Python 3.x installé sur votre machine.
- Les bibliothèques Python suivantes installées : Flask, pandas, plotly, folium, sqlite3. Vous pouvez les installer via la commande `pip` :

```
pip install Flask pandas plotly folium
```

A noter que `sqlite3` est inclus par défaut avec Python. Il n'est donc pas nécessaire de l'installer séparément.

- La base de données SQLite, nommée "WorldPopulation.db" contenant les données démographiques mondiales.

- Des fichiers GeoJSON des limites des pays du monde, disponibles dans le dossier `static/geojson` du projet.
- L'éditeur VS Code est recommandé pour suivre ce tutoriel.
- Un navigateur web pour visualiser l'application Flask.
- Connaissances de base en Python, SQL, HTML et CSS.

Étapes de développement

Vous disposez déjà d'un code source de l'application Flask dans le répertoire `code-SAE101`. Le code est organisé en plusieurs fichiers pour une meilleure modularité. Voici un aperçu des fichiers principaux avec leur arborescence et leur rôle :

```

code-SAE101/
├── controllers/          # Racine du projet
│   ├── dashboard_controller.py # Dossier "Contrôleur"
│   │   └── main_controller.py # Contrôleur du tableau de bord
│   └── database/           # Contrôleur principal gérant les routes
│       └── WorldPopulation.db # Dossier de la base de données
│           # Base de données SQLite
├── models/                # Modèle
│   ├── dashboard_utils.py  # Fonctions utilitaires - tableau de bord
│   ├── data_utils.py       # Fonctions utilitaires - accès aux données
│   └── db_utils.py         # Fonctions pour interagir avec la BD
├── static/                 # Dossier des fichiers statiques
│   ├── css/                # Dossier des feuilles de styles
│   │   └── styles.css       # Fichier CSS pour le style de l'application
│   └── geojson/             # Dossier des fichiers GeoJSON
│       ├── CNTR_RG_03M_2024_4326.geojson # Pays du monde à l'échelle 3M
│       ├── CNTR_RG_10M_2024_4326.geojson # Pays du monde à l'échelle 10M
│       └── CNTR_RG_20M_2024_4326.geojson # Pays du monde à l'échelle 20M
│       └── images/             # Dossier des images utilisées
│           ├── favicon.png    # Icône de l'application
│           └── image.png       # Une image qui peut être utilisée
└── templates/              # Vue
    ├── header.html          # Modèle HTML pour l'en-tête commune aux vues
    ├── dashboard.html        # Modèle HTML pour la page du tableau de bord
    └── index.html            # Modèle de base HTML
├── app.py                  # Point d'entrée de l'application Flask
├── config.py               # Fichier de configuration pour Flask
└── tutoriel.md             # Ce fichier tutoriel pour le projet

```

1. Structure de l'application Flask, approche MVC

L'application Flask est structurée selon le modèle MVC (Modèle-Vue-Contrôleur) pour une meilleure organisation du code. Cette approche permet de séparer la logique de l'application (Contrôleur), la gestion des données (Modèle) et la présentation des données (Vue).

Il existe d'autres structures possibles pour organiser une application web, telles que l'intégration avec des frameworks front-end comme [React](#) ou [Vue.js](#) pour une interface utilisateur plus dynamique. Cependant, pour

une première mise en oeuvre, nous nous concentrerons sur une structure simple basée sur le modèle MVC standard.

2. Configuration de l'application Flask

Le fichier `config.py` contient les paramètres de configuration de l'application Flask, tels que l'emplacement de la base de données et ceux des fichiers GeoJSON utilisés pour les cartes.

```
# config.py

# Importer le module os pour gérer les chemins de fichiers
import os

# Définir le chemin de la base de données SQLite
BASE_DIR = os.path.abspath(os.path.dirname(__file__))
DB_NAME = 'WorldPopulation.db'
DATABASE = os.path.join(BASE_DIR, 'database', DB_NAME)

# Définir la résolution des frontières geojson utilisées pour les cartes
GEOJSON_03M = os.path.join(BASE_DIR, 'static/geojson',
    'CNTR_RG_03M_2024_4326.geojson')
GEOJSON_10M = os.path.join(BASE_DIR, 'static/geojson',
    'CNTR_RG_10M_2024_4326.geojson')
GEOJSON_20M = os.path.join(BASE_DIR, 'static/geojson',
    'CNTR_RG_20M_2024_4326.geojson')
```

Ce fichier permet de centraliser les paramètres de configuration, facilitant ainsi la maintenance et les modifications futures. Il pourra être étendu pour inclure d'autres paramètres si nécessaire.

3. Le Modèle (accès aux données)

Cette partie (le Modèle) est la partie la plus importante de l'application. Elle permet d'extraire les données de la base de données SQLite et de les préparer pour l'affichage.

Le fichier `data_utils.py`, dans le dossier `models`, contient des fonctions pour interagir avec la base de données `WorldPopulation.db` et générer des tableaux et des graphes, ou encore des cartes interactives avec Folium.

Le fichier `dashboard_utils.py` contient des fonctions spécifiques pour le tableau de bord (`dashboard.html`), telles que la génération d'indicateurs clés (KPI en anglais pour *Key Performance Indicators*).

Le fichier `db_utils.py` contient une fonction pour se connecter à la base de données SQLite.

De nouveaux modèles de données seront ajoutés pour gérer des fonctionnalités supplémentaires.

4. La Vue (templates)

Les fichiers HTML du dossier `templates` définissent la structure des pages web de l'application. Le fichier `index.html` est la page d'accueil, tandis que `dashboard.html` affiche les données démographiques sous forme d'indicateurs clés (KPI).

Les fichiers HTML utilisent le moteur de templates `Jinja2` pour **intégrer dynamiquement** les données extraites par les contrôleurs. Ils contiennent des **balises particulières** `{} {}` et `{% %}` pour **insérer des variables et des structures de contrôle** (structures itératives ou conditionnelles).

Le fichier `header.html` est inclus dans les autres pages pour fournir une navigation cohérente. Cette inclusion est réalisée à l'aide de la directive `Jinja % include 'header.html' %` placée dans les autres fichiers HTML à l'endroit où l'en-tête doit apparaître. Cette en-tête contient un menu et un sous-menu composés des liens de navigation vers les différentes vues de l'application. Cette séparation permet de maintenir une **structure cohérente** et facilite la mise à jour du menu.

De nouvelles vues seront ajoutées en créant de nouveaux fichiers HTML dans le dossier `templates`.

5. Le Contrôleur (logique de l'application)

Il a été décidé de **séparer la logique** de l'application en plusieurs fichiers pour une **meilleure organisation du code**. Les fichiers dans le dossier `controllers` gèrent la logique de l'application :

1. Le fichier `main_controller.py` définit les routes Flask et utilise les fonctions des modèles pour récupérer les données et les passer aux vues.
2. Le fichier `dashboard_controller.py` contient des fonctions spécifiques pour gérer les interactions sur la page du tableau de bord.

De nouveaux contrôleurs seront ajoutés pour gérer des fonctionnalités supplémentaires.

Etude de l'application

1. Lancement de l'application Flask

Pour lancer l'application Flask, exécutez le fichier `app.py` à la racine du projet. Ce fichier crée l'instance de l'application Flask, enregistre les Blueprints des contrôleurs, et démarre le serveur web.

```
# app.py

# Importer les modules nécessaires
from flask import Flask # pour créer l'application Flask
from controllers.main_controller import main # importer le Blueprint principal
from controllers.dashboard_controller import dashboard # importer le Blueprint du
tableau de bord

# Créer l'application Flask
app = Flask(__name__)
```

```
# Enregistrer les Blueprints
# Le Blueprint 'main' gère la route principale de l'application
app.register_blueprint(main)
# Le Blueprint 'dashboard' gère la route du tableau de bord
app.register_blueprint(dashboard)

# Lancer l'application Flask
if __name__ == '__main__':
    #app.run(debug=True)
    app.run(host='127.0.0.1', port=5000, debug=True)
```

Dans ce code, nous créons une **instance de l'application Flask**, enregistrons les Blueprints pour organiser les routes, et lançons le serveur Flask en mode debug sur le serveur **local** et le port **5000**.

Les Blueprints sont des composants modulaires qui permettent de structurer une application Flask en **plusieurs parties réutilisables et indépendantes**. Chaque blueprint peut définir ses propres routes, vues, modèles et autres fonctionnalités, ce qui **facilite la maintenance et l'extension** de l'application.

2. Fonctionnalités mises en place

Les fonctionnalités principales de l'application incluent :

- Affichage de la population mondiale par année (tableau et graphique).
- Affichage de la population des régions par année (tableau et graphique).
- Top 10 des pays les plus peuplés par année (tableau et graphique).
- Affichage de la population et de la densité des pays d'Europe par année (tableau et cartes pour 2023).
- Affichage d'un certain nombre d'indicateurs clés (KPI) sur un tableau de bord (dashboard).
- Affichage d'une page "À propos" donnant un certain nombre d'informations sur l'application.

Chaque vue, construite dynamiquement, est accessible via un menu de navigation situé dans l'en-tête commun à toutes les pages. Ce menu est défini dans le fichier **header.html** et permet aux utilisateurs de naviguer facilement entre les différentes sections de l'application. Un sous-menu est également disponible pour accéder directement aux différentes visualisations (tableaux, graphiques, cartes) en fonction de chaque type de données démographiques.

Les différentes vues sont obtenues par action de l'utilisateur sur les liens du menu de navigation, qui déclenchent des requêtes HTTP vers les routes définies dans les contrôleurs. Ces routes appellent les fonctions appropriées dans les modèles pour récupérer et traiter les données, puis rendent les templates HTML avec les données dynamiques intégrées.

Par exemple, lorsque l'utilisateur clique sur l'item "Population mondiale" dans le menu, la requête **http://127.0.0.1:5000/?query=world&view=table** est envoyée vers le serveur Flask. Le contrôleur principal (**main_controller.py**) intercepte cette requête, analyse les paramètres **query** et **view**, et suivant les valeurs prises par ces paramètres, il appelle les fonctions correspondantes, ici **get_world_population_by_year()**, dans les modèles pour obtenir les données nécessaires, puis construit la représentation appropriée (tableau, graphique, carte, etc.) et rend la vue avec cette représentation des données.

Voici un extrait du code du contrôleur principal (**main_controller.py**) qui gère cette logique de routage en fonction des paramètres de la requête :

```
# controllers/main_controller.py
@main.route('/')
def index():
    # Récupérer les paramètres d'URL
    query_type = request.args.get('query', 'world') # Par défaut : population mondiale
    view_type = request.args.get('view', 'table') # Par défaut : tableau

    # Récupérer les données selon le type de requête
    # Génération des titres et en-têtes des colonnes du tableau selon le type de requête

    # Population mondiale par année
    if query_type == 'world':
        data = du.get_world_population_by_year()
        title = "Population mondiale par année"
        headers = ["Année", "Population (Hommes)", "Population (Femmes)", "Population totale"]

    # Autres cas...

    # Génération des représentations selon le type de requête et le type de vue (graphique, carte)
    plot_html = None

    # graphe de la population mondiale par année
    if query_type == 'world' and view_type == 'graph':
        plot_html = du.generate_population_plot()

    # Autres cas...

    # Rendre le template avec les données, le titre, les en-têtes, le type de requête, le type de vue et le graphe (si applicable)
    return render_template(
        'index.html',
        data=data,
        title=title,
        headers=headers,
        query_type=query_type,
        view_type=view_type,
        plot_html=plot_html
    )
```

Ici, la fonction `index()` gère la route principale `/`. Elle récupère les paramètres `query` et `view` de l'URL pour déterminer quel type de données afficher et sous quelle forme. En fonction de ces paramètres, elle appelle les fonctions appropriées dans les modèles pour obtenir les données et génère la représentation souhaitée (tableau, graphique, carte). Enfin, elle rend le template HTML (`render_template(...)`) avec les données dynamiques intégrées

La fonction `render_template()` de Flask est utilisée pour **rendre un template HTML** en y injectant des variables dynamiques. Elle prend en paramètre le nom du fichier template (par exemple, `'index.html'`) et

des paires clé-valeur représentant les variables à passer au template. Ces variables peuvent ensuite être utilisées dans le fichier HTML à l'aide de la syntaxe Jinja2, permettant ainsi de créer des pages web dynamiques basées sur les données extraites et traitées par l'application Flask.

3. Inspection du code source

Vous devez **explorer le code source complet** de l'application Flask dans le répertoire [code-SAE101](#). Chaque fichier est commenté pour expliquer son rôle et son fonctionnement.

Une attention particulière doit être portée à la manière dont les données sont extraites de la base de données SQLite, traitées et présentées dans les vues HTML. Essayez de comprendre comment les différentes parties de l'application interagissent entre elles.

L'IHM (Interface Homme-Machine) est conçue pour être simple et intuitive, avec un menu de navigation clair et des visualisations de données faciles à comprendre. Le menu de navigation est défini dans le fichier [header.html](#) et permet d'accéder aux différentes vues de l'application en un seul clic qui déclenche les **requêtes HTTP** appropriées.

Une bonne compréhension de l'architecture MVC, des routes Flask, des templates Jinja2 est essentielle pour maîtriser cette application afin de **pouvoir la modifier et l'étendre**.

Les vues HTML utilisent des bibliothèques JavaScript comme [DataTables](#) pour les tableaux interactifs, [Plotly.js](#) pour les graphiques interactifs et [Leaflet.js](#) pour les cartes interactives. Ces bibliothèques sont intégrées dans les templates HTML pour enrichir l'expérience utilisateur.

Graphiques et cartes sont générés dynamiquement côté serveur en utilisant les bibliothèques Python [Plotly](#) et [Folium](#) (version Python de Leaflet), puis intégrés dans les pages HTML via des blocs HTML.

Besoin client - Fonctionnalités et améliorations à mettre en place

Votre **maître d'ouvrage** (MO) souhaite que vous ajoutiez des fonctionnalités supplémentaires à cette application Flask pour améliorer l'expérience utilisateur et enrichir les données présentées.

En tant que **maître d'oeuvre** (MOE), vous devez proposer et implémenter les améliorations suivantes :

- Proposer des **vues supplémentaires** sur des données démographiques disponibles dans la base de données, telles que des courbes, des histogrammes, des diagrammes en secteurs ou des indicateurs clés (KPI).
- Ajouter des **filtres interactifs** pour permettre aux utilisateurs de sélectionner des plages de dates, une sous-région, une région ou un continent spécifique, ou d'autres critères pour affiner les données affichées.
- Intégrer des **fonctionnalités de téléchargement** pour permettre aux utilisateurs de télécharger dans des formats CSV ou Excel les données affichées dans les tableaux.
- Intégrer des **analyses statistiques** ou des **prévisions** basées sur les données démographiques disponibles (voir l'annexe, fichier [annexe.md](#)).
- Améliorer le **design de l'interface** utilisateur en utilisant des frameworks CSS comme [Bootstrap](#) pour rendre l'application plus attrayante et conviviale.
- Rédiger une **documentation utilisateur** pour expliquer comment utiliser l'application et ses différentes fonctionnalités.

- Rédiger une **documentation technique** pour expliquer la structure du code, les modèles de données et les choix de conception effectués lors du développement de l'application.

Chaque fonctionnalité doit être soigneusement planifiée, développée et testée pour garantir une intégration fluide dans l'application existante. Assurez-vous de suivre les meilleures pratiques de développement web en Python Flask, et de maintenir un code propre et bien documenté (commentaires, structure claire, etc.) tout au long du processus.

Remarque sur la base de données : Si nécessaire, vous pouvez apporter des modifications à la base de données SQLite pour ajouter des tables ou des colonnes afin d'optimiser le stockage de données existantes ou supplémentaires requises pour les nouvelles fonctionnalités. Assurez-vous que toutes les modifications apportées à la base de données sont bien documentées et que le code d'accès aux données est mis à jour en conséquence.

Livrables attendus

À la fin de ce projet, vous devez fournir les livrables suivants :

- Le code source complet de l'application Flask avec les nouvelles fonctionnalités implémentées.
- La base de données SQLite mise à jour si des modifications ont été apportées.
- Le fichier README.md mis à jour avec des instructions claires pour l'installation et l'utilisation de l'application.
- La documentation utilisateur et technique.
- Un rapport de projet décrivant les fonctionnalités ajoutées, les défis rencontrés et les solutions apportées.
- Une présentation orale avec une démonstration de l'application pour montrer les nouvelles fonctionnalités et expliquer le processus de développement.