Objectifs de la feuille

- Introduction
- Formulaires
- Modification des données
- Suppression des données
- Création de données

Introduction

Dans ce TP vous allez utiliser WTForms pour fabriquer des formulaires. Alors que SQLAlchemy permet de modéliser des tables avec des classes, WTForms nous permet de modéliser des formulaires avec des classes. Pour commencer, vous devez installer le plugin flask-wtf :

```
$ pip install flask-wtf
```

Aucune activation n'est nécessaire. Par contre, c'est ici que nous allons nous servir de notre une clé secrète pour sécuriser les interactions et se protéger d'attaques CSRF (Cross-Site Request Forgery). Je vous rappelle que la clé secrète est dans notre fichier config.py :

```
SECRET_KEY = "2lzUl{$*D6#`8uXqlU."
```

Pour générer votre clé secrète impossible à deviner je vous conseille d'utiliser la commande uuidgen. Vous obtiendrez bien sûr une autre valeur ! L'essentiel demeure dans le fait que les valeurs de configuration du projet soient connues de l'application avec dans le fichier app.py la ligne suivante (dèjà présente normalement) :

```
app.config.from_object('config')
```

Création de formulaires

Dans un nouveau fichier forms.py nous allons définir le formulaire FormAuteur

```
from flask_wtf import FlaskForm
from wtforms import StringField, HiddenField
from wtforms.validators import DataRequired

class FormAuteur(FlaskForm):
        idA=HiddenField('idA')
        Nom = StringField ('Nom', validators =[DataRequired()])
```

Modification d'un auteur

Vous souvenez-vous de l'architecture REST ? Vous savez donc que l'URL d'une ressource peut ressembler à celle-ci : /auteurs/1. Mais comment accéder à la partie variable ? Nous n'avons ici pas de point d'interrogation en guise de séparateur. L'opération va être légèrement différente. Plutôt que de faire appel à l'objet request, vous allez modifier directement la route impactée :

```
@app.route ("/auteurs/<idA>/")
```

Indiquez les parties spéciales d'une url en ajoutant <variable>. Cette partie est alors passée en tant qu'argument de la vue :

```
def getAuteurById(idA) :
```

Vous pouvez spécifier un convertisseur pour forcer l'utilisation d'un type d'objet :

```
@app.route ("/auteurs/<int:idA>/")
```

Bien. Maintenant que nous avons le moyen de construire une url digne de ce nom, utilisons notre formulaire dans une vue updateAuteur(idA) accessible via l'url /auteurs/<idA>/update dans le fichier views.py :

```
from monApp.forms import FormAuteur

@app.route('/auteurs/<idA>/update/')
def updateAuteur(idA):
    unAuteur = Auteur.query.get(idA)
    unForm = FormAuteur(idA=unAuteur.idA , Nom=unAuteur.Nom)
    return render_template("auteur_update.html",selectedAuteur=unAuteur, updateForm=unForm)
```

Vous devez ensuite écrire le template auteur_update.html qui va avec :

```
{% extends "base.html" %}

{% block main %}
<h1 >Modification de l'auteur {{ selectedAuteur.Nom }}</h1>
<form method ="POST" action ="">
    {{ updateForm.hidden_tag () }}
    {{ updateForm.Nom.label }} {{ updateForm.Nom(size =50) }}
    <input type="submit" value ="Enregistrer">
</form >
{% endblock %}
```

updateForm.hidden_tag() insère un div contenant tous les champs cachés du formulaire ; ici il y en a 2 : le champ idA et le champ pour le token CSRF qui protège l'action du formulaire contre les contrefaçons. Pour l'instant, nous ne mettons pas d'URL dans action parce que nous n'avons pas écrit la fonction pour sauvegarder le résultat de l'édition. Vérifiez que le formulaire est bien accessible en tapant l'url *http://127.0.0.1:5000/auteurs/1/update* par exemple.

Adaptons notre formulaire à bootstrap :

```
{% extends "base.html" %}
{% block main %}
<h1 >Modification de l'auteur {{ selectedAuteur.Nom }}</h1>
<form role="form" method="POST" action="">
        {{ updateForm.hidden_tag () }}
        <div class="form-group mb-3">
                {{ updateForm.Nom.label }}
                {{ updateForm.Nom(size =50, class_="form-control") }}
        </div>
        <input class ="btn btn-success" type="submit" value ="Enregistrer">
        <a href="{{ url_for('getAuteurs') }}" class="btn btn-secondary">Retour</a>
</form >
{% endblock %}
```

Le paramètre class_="form-control" indique qu'on veut ajouter la classe form-control au widget input (voir la doc de bootstrap sur les formulaires). L'API utilise class_ avec un underscore car class est un mot clé et ne peut être utilisé comme un identifiant normal. Dans votre navigateur, visitez le formulaire d'édition et regardez le code HTML source de la page pour constater ce qui a été généré pour le formulaire. Il faut maintenant mettre en place dans le fichier views.py l'action pour enregistrer les modifications de l'auteur. Cette action est porté par la méthode saveAuteur( ) :

```
from flask import url_for , redirect
from .app import db

@app.route ('/auteur/save/', methods =("POST" ,))
def saveAuteur():
        updatedAuteur = None
        unForm = FormAuteur()
        #recherche de l'auteur à modifier
        idA = int(unForm.idA.data)
        updatedAuteur = Auteur.query.get(idA)
        #si les données saisies sont valides pour la mise à jour
                if unForm.validate_on_submit():
                updatedAuteur.Nom = unForm.Nom.data
                db.session.commit()
                return redirect(url_for('viewAuteur', idA=updatedAuteur.idA))

        return render_template("auteur_update.html",selectedAuteur=updatedAuteur, updateForm=unForm)

@app.route('/auteurs/<idA>/view/')
def viewAuteur(idA):
        unAuteur = Auteur.query.get(idA)
        unForm = FormAuteur (idA=unAuteur.idA , Nom=unAuteur.Nom)
        return render_template("auteur_view.html",selectedAuteur=unAuteur, viewForm=unForm)
```

Si nous avons mis à jour l'auteur, alors nous faisons une redirection vers sa page (dans mon code, cette page correspond à la fonction viewAuteur). Sinon, nous renvoyons l'utilisateur vers la page du formulaire pour qu'il entre des données valides.

Vous devez aussi éditer le template auteur_update.html pour mettre en place l'action sur le formulaire :

```
[…]
<form role="form" method="POST" action="{{ url_for ('saveAuteur') }}">
[…]
```

Aussi, vous devez créer le template auteur_view.html :

```
{% extends "base.html" %}
{% block main %}
<h1 >Consultation de l'auteur {{ selectedAuteur.Nom }}</h1>
<form role="form" method="POST" action="">
        {{ viewForm.hidden_tag () }}
        <div class="form-group mb-3">
                {{ viewForm.Nom.label }}
                {{ viewForm.Nom(size=50, disabled=True, class_="form-control") }}
        </div>
        <a href="{{ url_for('getAuteurs') }}" class="btn btn-secondary">Retour</a>
</form >
{% endblock %}
```

Enfin, pour en finir avec les éléments de navigation, il faut rajouter des liens vers ses pages de modification et de de consultation depuis la liste des auteurs. Nous allons donc modifier le template auteurs_list.html au niveau du tableau pour créer ces liens :

```
<table class="table table-striped table-bordered table-hover">
   <thead>
        <tr>
            <th>ID</th>
            <th>Nom</th>
            <th>Nombre de livres</th>
            <th>Actions possibles</th>
        </tr>
    </thead>
    <tbody>
        {% for unAuteur in auteurs %}
        <tr>
            <td>{{ unAuteur.idA }}</td>
            <td>{{ unAuteur.Nom }}</td>
            <td>{{ unAuteur.livres.count() }}</td>
            <td><div class="task-controls">
                <a href="{{ url_for('viewAuteur', idA=unAuteur.idA) }}" class="btn btn-nfo">
                Voir <i class="fas fa-eye"></i></a>
                <a href="{{ url_for('updateAuteur', idA=unAuteur.idA) }}" class="btn btn-warning">
                Editer <i class="fas fa-pen"></i></a>
                </div>
        </td>
        </tr>
        {% endfor %}
    </tbody>
</table>
```

Affichage des erreurs de validation

Nous ajoutons maintenant le support pour visualiser les erreurs de validation. Quand un champ d'un formulaire a des erreurs de validation, l'attribut errors contient une liste de messages d'erreur qu'on voudra afficher par exemple en dessous du widget. Il suffit pour cela de modifier un peu le template auteur_update.html :

```
{% extends "base.html" %}

{% block main %}
<h1 >Modification de l'auteur {{ selectedAuteur.Nom }}</h1>
<form role="form" method="POST" action="{{ url_for ('saveAuteur') }}">
        {{ updateForm.hidden_tag () }}
        <div
        {% if updateForm.Nom.errors %}
                class="form-group has-error"
        {% else %}
                class="form-group mb-3"
        {% endif %}
        >
        {{ updateForm.Nom.label }}
        {{ updateForm.Nom(size =50, class_="form-control") }}
        </div>
        {% if updateForm.Nom.errors %}
                <ul class ="list-group">
                {% for e in updateForm.Nom.errors %}
                        <li class ="list-group-item list-group-item-danger" >{{ e }} </li >
                {% endfor %}
                </ul >
        {% endif %}
        <input class ="btn btn-success" type="submit" value ="Enregistrer">
        <a href="{{ url_for('getAuteurs') }}" class="btn btn-secondary">Retour</a>
</form >
{% endblock %}
```

Ajout d'un auteur

Faites la même chose que précédemment, mais cette fois-ci, pour ajouter un nouvel auteur. Pour sauvegarder ce nouvel auteur, vous devrez faire comme dans la commande loaddb : créer une nouvelle instance d'Auteur, puis l'ajouter à la transaction, puis commiter la transaction. Commençons par ajouter cette nouvelle action dans le menu du fichier base.html :

```
<ul class="dropdown-menu" aria-labelledby="auteursDropdown">
    <li><a class="dropdown-item" href="{{ url_for('getAuteurs') }}">Liste des auteurs</a></li>
    <li><a class="dropdown-item" href="{{ url_for('createAuteur') }}">Créer un auteur</a></li>
</ul>
```

Comme précédemment, nous avons besoin :

- d'une vue createAuteur( ) pour afficher le formulaire de création

```
@app.route('/auteur/')
def createAuteur():
        unForm = FormAuteur()
        return render_template("auteur_create.html", createForm=unForm)
```

- d'un template auteur_create.html pour prendre en compte l'action utilisateur

```
{% extends "base.html" %}
{% block main %}
<h1 >Création d'un auteur</h1>
<form role="form" method="POST" action="{{ url_for ('insertAuteur') }}">
        {{ createForm.hidden_tag () }}
        <div
                {% if createForm.Nom.errors %}
                        class="form-group has-error"
                {% else %}
                        class="form-group mb-3"
                {% endif %}
        >
        {{ createForm.Nom.label }}
        {{ createForm.Nom(size =50, class_="form-control") }}
        </div>
        {% if createForm.Nom.errors %}
                <ul class ="list-group">
                {% for e in createForm.Nom.errors %}
                        <li class ="list-group-item list-group-item-danger" >{{ e }} </li >
                {% endfor %}
                </ul >
        {% endif %}
        <input class ="btn btn-success" type="submit" value ="Créer">
        <a href="{{ url_for('getAuteurs') }}" class="btn btn-secondary">Retour</a>
</form >
{% endblock %}
```

- d'une autre vue insertAuteur( ) pour valider l'action de création

```
@app.route ('/auteur/insert/', methods =("POST" ,))
def insertAuteur():
        insertedAuteur = None
        unForm = FormAuteur()
        if unForm.validate_on_submit():
                insertedAuteur = Auteur(Nom=unForm.Nom.data)
                db.session.add(insertedAuteur)
                db.session.commit()
                insertedId = Auteur.query.count()
                return redirect(url_for('viewAuteur', idA=insertedId))

        return render_template("auteur_create.html", createForm=unForm)
```

Suppression d'un auteur

Comme pour la modification, nous avons besoin de :

- rajouter l'action de suppression dans le tableau de la liste des auteurs (auteurs_list.html)

```
<a href="{{ url_for('deleteAuteur', idA=unAuteur.idA) }}" class="btn btn-danger">Supprimer <i class="fas fa-trash"></i></a>
```

- créer une vue deleteAuteur(idA) pour afficher le formulaire de suppression

```
@app.route('/auteurs/<idA>/delete/')
def deleteAuteur(idA):
    unAuteur = Auteur.query.get(idA)
    unForm = FormAuteur(idA=unAuteur.idA, Nom=unAuteur.Nom)
    return render_template("auteur_delete.html",selectedAuteur=unAuteur, deleteForm=unForm)
```

- créer le template auteur_delete.html pour afficher pour prendre en compte la confirmation de suppression

```
{% extends "base.html" %}

{% block main %}
<h1 >Suppression de l'auteur {{ selectedAuteur.Nom }}. Etes vous sur ?</h1>
<form role="form" method="POST" action="{{ url_for ('eraseAuteur') }}">
    {{ deleteForm.hidden_tag () }}
    <div class="form-group mb-3">
        {{ deleteForm.Nom.label }}
        {{ deleteForm.Nom(size=50, disabled=True, class_="form-control") }}
    </div>
    <input class ="btn btn-danger" type="submit" value ="Confirmer">
    <a href="{{ url_for('getAuteurs') }}" class="btn btn-secondary">Retour</a>
</form >
{% endblock %}
```

- d'une autre vue eraseAuteur( ) pour valider l'action de suppression

```
@app.route ('/auteur/erase/', methods =("POST" ,))
def eraseAuteur():
    deletedAuteur = None
    unForm = FormAuteur()
    #recherche de l'auteur à supprimer
    idA = int(unForm.idA.data)
    deletedAuteur = Auteur.query.get(idA)
    #suppression
    db.session.delete(deletedAuteur)
    db.session.commit()
    return redirect(url_for('getAuteurs'))
```

C'est à vous de jouer !

Faites de même pour les livres (formulaire de consultation et de modification seulement). Seul le champ Prix sera modifiable pour la simplicité. N'oubliez pas toute la navigation...