

Documentation technique badminton



Table des matières

Base de données.....	3
Schéma de la base de données.....	6
Organisation de l'application.....	7
App.py.....	8
Base.html	9
Connexion Utilisateur	10
Après authentification	13

Base de données

L'application WEB Badminton est liée à une base de données afin de pouvoir enregistrer, modifier, supprimer les données des adhérents.

Cette base de données est composée de 4 tables :

1. **Adherent** (Contient toutes les informations des adhérents)

adherent	
matriculeAdh	int(11)
nomAdh	varchar(25)
prenomAdh	varchar(25)
villeAdh	varchar(50)
cpAdh	int(5)
niveauAdh	int(11)
typeAdh	int(11)

2. **Compte** (permet de se connecter à l'application afin de pouvoir accéder aux fonctionnalités)

compte	
id	int(11)
login	varchar(30)
mdp	varchar(30)

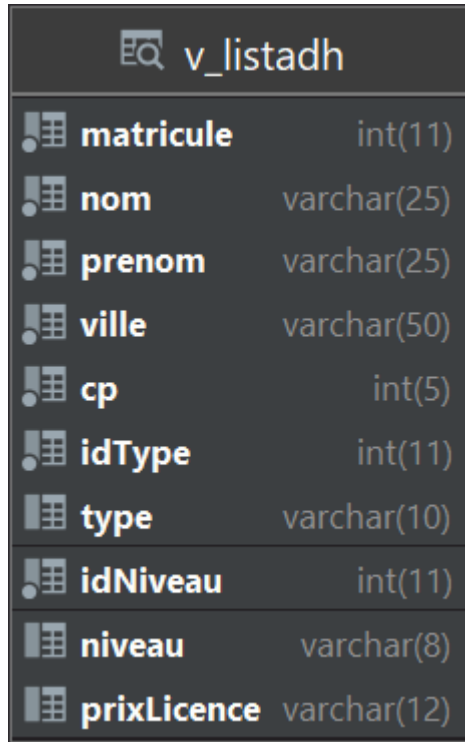
3. **Niveau** (contient les niveaux de joueurs)

niveau	
id	int(1)
libelle	varchar(8)
montantLicence	int(11)

4. **Type** (contient le type -> statut)

Ainsi que d'une vue :

1. **V_listadh** (permet de récupérer et stocker le contenu des tables adherent, niveau et type)



v_listadh	
matricule	int(11)
nom	varchar(25)
prenom	varchar(25)
ville	varchar(50)
cp	int(5)
idType	int(11)
type	varchar(10)
idNiveau	int(11)
niveau	varchar(8)
prixLicence	varchar(12)

Et de 3 procédures stockées afin de sécuriser l'application :

1. **Sp_addAdh** (Permet d'ajouter un adhérent)

```
create
  definer = root@localhost procedure sp_addAdh(IN p_nom varchar(25), IN p_prenom varchar(25), IN p_ville varchar(50),
  IN p_cp int(15), IN p_niveau int, IN p_type int)
begin
  insert into adherent
  (nomAdh, prenomAdh, villeAdh, cpAdh, niveauAdh, typeAdh)
  values
  (
    p_nom,
    p_prenom,
    p_ville,
    p_cp,
    p_niveau,
    p_type
  );
end;
```

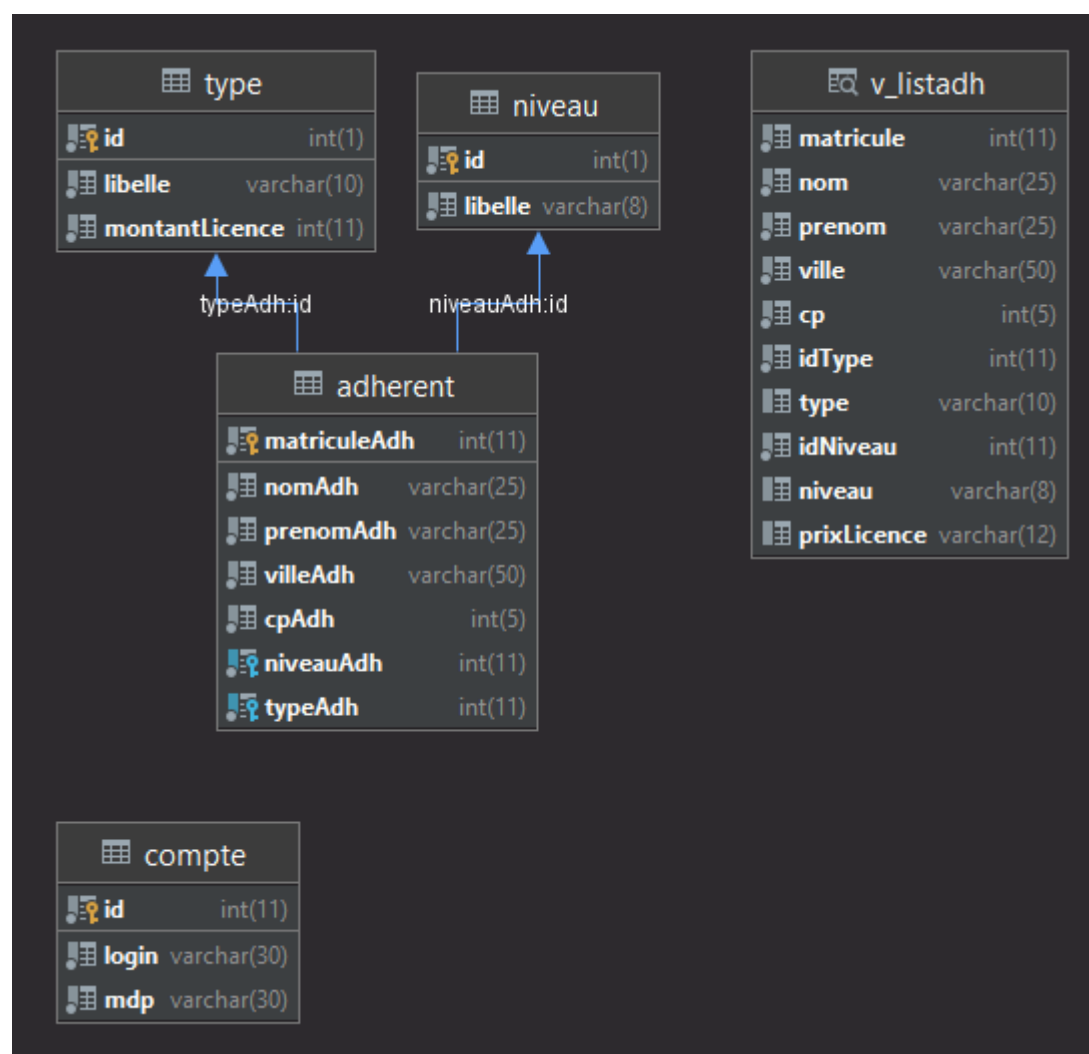
2. **Sp_deleteAdh** (Permet de supprimer un adhérent)

```
create
  definer = root@localhost procedure sp_deleteAdh(IN p_matricule int)
begin
  delete from adherent where matriculeAdh=p_matricule;
end;
```

```
create
  definer = root@localhost procedure sp_updateAdh(IN p_nomAdh varchar(25), IN p_prenomAdh varchar(25),
  IN p_ville varchar(50), IN p_cp int(5), IN p_niveau int,
  IN p_type int, IN p_matricule int)
begin
  update adherent set nomAdh=p_nomAdh, prenomAdh=p_prenomAdh, villeAdh=p_ville, cpAdh=p_cp, typeAdh=p_type, niveauAdh=p_niveau where matriculeAdh=p_matricule;
end;
```

3. **Sp_updateAdh** (Permet de supprimer un adhérent)

Schéma de la base de données



Organisation de l'application

L'application a été développée avec Python et le framework « Flask ».

Pourquoi Flask et non PHP ?

Pour sa simplicité à comprendre le code et le bon fonctionnement de l'application.

Contenu de l'application :

Badminton

- |— **app.py** -> contient toutes les fonctionnalités
- |— **static** -> contient les fichiers statics (css, img, js ...)
 - | — **css**
 - | — **loginNregister.css**
- |— **templates** -> contient les pages HTML
 - |— index.html
 - |— **layouts** -> contient les bases de l'application
 - | — base.html
 - | — navbar.html
 - |— menu.html
- |— **modals** -> contient les modal (Pop-up) de l'application
 - |— modalAdd.html
 - |— modalUpdate.html
 - |— modalView.html

Base.html

Le fichier « Base.html » est un fichier template, il ne sert que de modele afin de ne pas répéter les bases d'un fichier HTML.

```
<!doctype html>
<html lang="en">
<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.1/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-+0n0xVW2eSR50omGNYDnhzAbDsOXxcvSN1TPPrVMTNDbiYZCxYbOO17+AMvyTG2x" crossorigin="anonymous">
  <link rel="stylesheet" href="/static/css/loginNregister.css">
  <script type="text/javascript" src="https://ajax.googleapis.com/ajax/libs/jquery/1.5/jquery.min.js"></script>
  <link rel="stylesheet" href="https://cdn.datatables.net/1.10.24/css/jquery.dataTables.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.0/jquery.min.js"></script>
  <script type="text/javascript" src="https://cdn.datatables.net/1.10.16/js/jquery.dataTables.min.js"></script>
  <title>{{ title }} | Badminton</title>
</head>
<body>
  {% include 'layouts/navbar.html' %}
  {% block content %}{% endblock %}
  {% block modal %}{% endblock %}
  {% block script %}{% endblock %}
  <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"
    integrity="sha384-IQsoLXl5PILFhosVNubq5LC7Qb9DXgDA9i+tQ8Zj3iwWAwPtgFTxbJ8NT4GN1R8p"
    crossorigin="anonymous"></script>
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.1/dist/js/bootstrap.min.js"
    integrity="sha384-Atwg2Pkwv9vp0ygttn1JAoJH0nYbwNjLPhwyoVbhoPwBhjQPR5VtM2+xf0Uwh9KtT"
    crossorigin="anonymous"></script>
  <script src="https://cdn.datatables.net/1.10.24/js/jquery.dataTables.min.js"></script>
</body>
</html>
```

On peut y retrouver 3 block :

1. **Content** : permet d'afficher le contenu d'une page qui hérite de base.html
2. **Modal** : permet d'afficher le modal sélectionné
3. **Script** : permet d'ajouter du code JavaScript

{{ title }} -> Permet de changer l'entete d'une page.

{% include 'layouts/navbar.html' %} -> Fait appel à la barre de navigation

Tout ces éléments permettent de mieux s'y retrouver.

App.py

Le fichier cœur de l'application.

Dans ce fichier nous pouvons y retrouver toutes les requetes SQL mais aussi les fonctions et les différentes routes.

Au début de ce fichier se trouve la déclaration de la base de données :

```
# Base de donnees
# Initialisation de MySQL
mysql = MySQL(app)

# Definition de la base de donnees
app.config['MYSQL_HOST'] = 'localhost'
app.config['MYSQL_USER'] = 'root'
app.config['MYSQL_PASSWORD'] = ''
app.config['MYSQL_DB'] = 'badminton'
```

Route :

La route permet de definir le chemin d'accès à une fonctionnalité ou à une page.

Pour déclarer une route :

```
@app.route('/[chemin de l'url]', [parametre])
```

Afin de faire fonctionner la route il faut lui définir une fonction :

```
def [nom de la fonction] ([parametre])
    contenu de la fonction
    return ('page.html')
```

Ce qui donne :

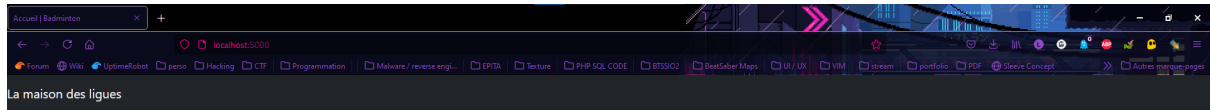
```
@app.route('/[chemin de l'url]', [parametre])
def [nom de la fonction] ([parametre])
    contenu de la fonction
    return ('page.html')
```

Fonction :

Une fonction avec Flask se fait de la meme manière qu'une route.

Connexion utilisateur

Affichage navigateur :



Espace de connexion

Identifiant

Mot de passe

0

Route de l'index :

```
# Route vers l'index
@app.route('/')
@app.route('/accueil')
def index():
    if not session: # Si il n'y a pas de session active alors on affiche la page de login
        return render_template(
            'index.html',
            title='Accueil'
        )
    else: # Si une session est active alors on affiche le menu
        return redirect(url_for('menu'))
```

Code HTML index.html :

```
{% extends 'layouts/base.html' %}

{% block content %}
<body>
<div class="container">
  <div class="row">
    <div class="col-sm-9 col-md-7 col-lg-5 mx-auto">
      <div class="card card-signin my-5">
        <div class="card-body">
          <h5 class="card-title text-center">Espace de connexion</h5>
          <form class="form-signin" action="{{ url_for('auth') }}" method="post">
            <div class="form-label-group">
              <input type="text" name="login" id="login" class="form-control"
                placeholder="Identifiant"
                required autofocus>
              <label for="login">Identifiant</label>
            </div>

            <div class="form-label-group">
              <input type="password" name="mdp" id="mdp" class="form-control"
                placeholder="Mot de passe"
                required>
              <label for="inputPassword">Mot de passe</label>
            </div>
            <button class="btn btn-lg btn-primary btn-block text-uppercase" type="submit">
            </button>
            <h1>{{ session['id'] }}</h1>
          </form>
        </div>
      </div>
    </div>
  </div>
</div>
</body>
{% endblock %}
```

{% extends 'layouts/base.html' %}	{{ url_for('auth') }}	{{ session['id'] }}
- Permet d'appeler le fichier template	- Appel de la fonction 'auth' afin de procéder à l'authentification	- Récupération de la session par l'id

Code de la fonction 'Auth'

```
# Route permettant de s'identifier + une fois loggé -> entre dans l'application
@app.route('/auth', methods=['GET', 'POST'])
def auth():
    msg = '' # Message en cas d'erreur
    if request.method == 'POST' and 'login' in request.form and 'mdp' in request.form: # Recupere les données du formulaire HTML
        login = request.form['login']
        mdp = request.form['mdp']
        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
        cursor.execute('select * from compte where login=%s and mdp=%s', (login, mdp,)) # Execution de la requete
        compte = cursor.fetchone() # Recuperation des données

        if compte:
            session['logged_in'] = True
            session['id'] = compte['id']
            session['login'] = compte['login']
            return redirect(url_for('menu'))
    else:
        msg = 'Probleme dans les identifiants & mot de passe'
    return render_template(
        'index.html',
        title='Connexion',
        msg=msg
    )
```

Après authentification

Une fois loggé, une page va apparaître avec toutes les fonctionnalités :

La maison des ligues

Connecté en tant qu'admin
Déconnexion

Ajouter un nouvel adhérent

Show 10 entries

Search:

Nom	Prénom	type	Niveau	Montant de la licence	Actions
aaa	aaa	Salarié	Débutant	27€	Edit Delete
roule	pierre	Salarié	Débutant	27€	Edit Delete
roule	pierre	Salarié	Débutant	27€	Edit Delete
roule	pierre	Salarié	Débutant	27€	Edit Delete

Showing 1 to 4 of 4 entries

Previous1Next

Sur cette page plusieurs fonctionnalités sont disponibles :

1. Ajout d'un adhérent :
2. [Visualisation de la fiche adhérent](#)
3. [Ajout d'un nouveau niveau](#)
4. [Suppression de l'adhérent](#)
5. Recherche dynamique
6. Déconnexion

```

# Route permettant j'ajout d'un adhérent dans la base de données
@app.route('/add', methods=['POST', 'GET'])
def add():
    cursor = mysql.connection.cursor()
    nom = request.form['nom']
    prenom = request.form['prenom']
    ville = request.form['ville']
    cp = request.form['cp']
    type = request.form['type']
    niveau = request.form['niveau']

    cursor.callproc('sp_addAdh', (nom, prenom, ville, cp, niveau, type))
    mysql.connection.commit()
    return redirect(url_for('menu'))

```

Fonction pour ajouter un adhérent :

```

# Route permettant la modification d'un adhérent
@app.route('/update', methods=['POST'])
def update():
    matricule = request.form['matricule']
    nom = request.form['nom']
    prenom = request.form['prenom']
    ville = request.form['ville']
    cp = request.form['cp']
    type = request.form['type']
    niveau = request.form['niveau']

    cursor = mysql.connection.cursor()
    cursor.callproc('sp_updateAdh', (nom, prenom, ville, cp, type, niveau, matricule))
    mysql.connection.commit()
    return redirect(url_for('menu'))

```

Fonction pour modifier un adhérent :

Fonction pour supprimer un adhérent :

```
# Route permettant la suppression d'un adhérent
@app.route('/delete/<string:id>', methods=['GET'])
def delete(id):
    cursor = mysql.connection.cursor()
    cursor.callproc('sp_deleteAdh', (id,))
    mysql.connection.commit()
    return redirect(url_for('menu'))
```

Fonction pour la déconnexion :

```
# Route permettant de se déconnecter
@app.route('/logout')
def logout():
    session.pop('loggedin', None),
    session.pop('id', None),
    session.pop('login', None),
    return redirect(url_for('index'))
```