PROJET ARCHITECTURE LOGICIELLE ET WEB

MINIMAL VIABLE PRODUCT (MVP)

BY SPLASH CORP.

ANASS DOUBLAL (CHEF D'EQUIPE) ET LISA MURACCIOLE

PRESENTATION ET ARCHITECTURE

Pour ce projet, nous avons décidé de créer un logiciel permettant à des plongeurs d'enregistrer leurs sessions de plongée.

Une fois connecté à un compte, un plongeur peut voir ses plongées, en ajouter, les modifier et les supprimer, ainsi que de voir ses statistiques. Une plongée connecte de nombreuses données. Voici quelques schémas pour en comprendre l'architecture :

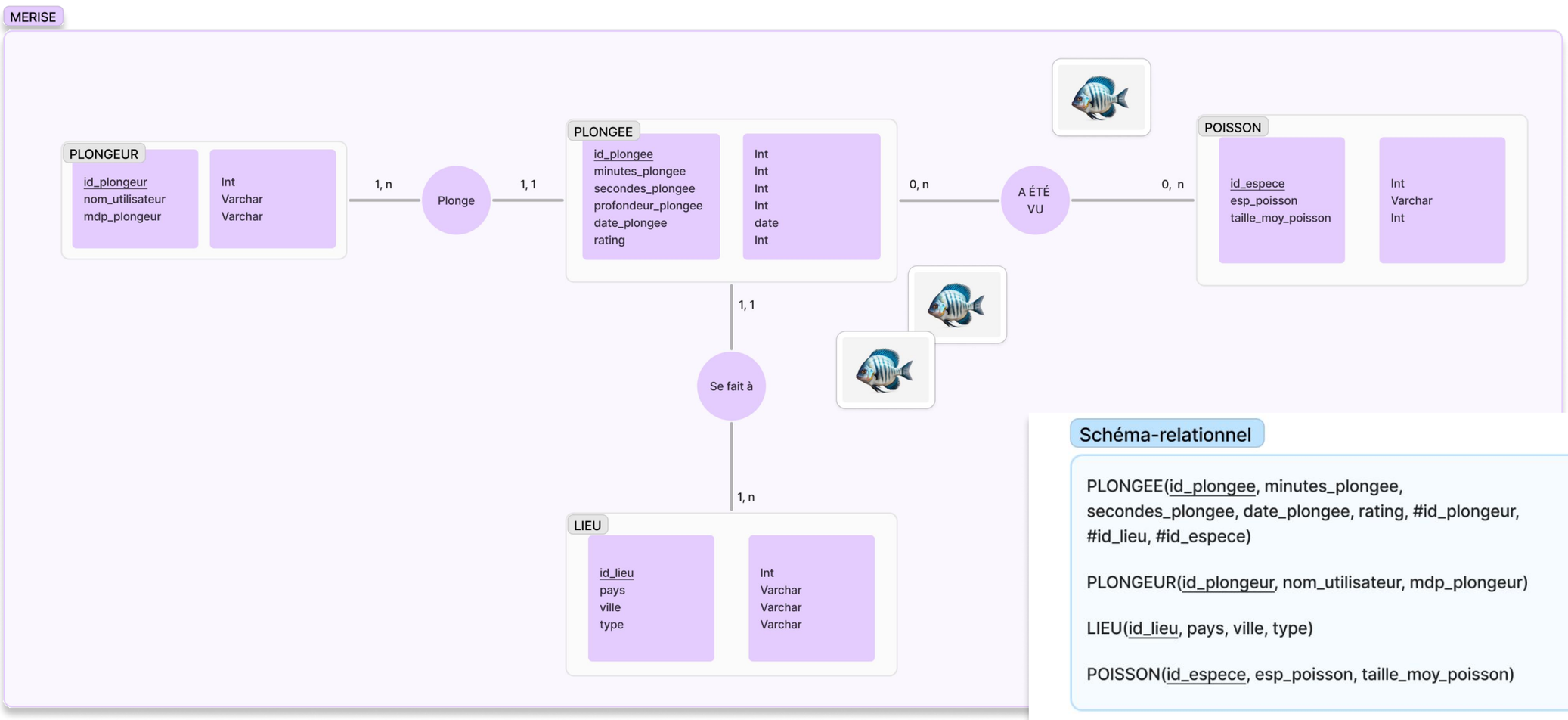


Schéma-relationnel

PLONGEE(id_plongee, minutes_plongee, secondes_plongee, date_plongee, rating, #id_plongeur, #id_lieu, #id_espece)

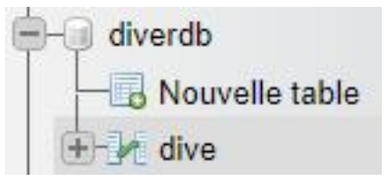
PLONGEUR(id_plongeur, nom_utilisateur, mdp_plongeur)

LIEU(id_lieu, pays, ville, type)

POISSON(id_espece, esp_poisson, taille_moy_poisson)

CREATION DE LA BASE DE DONNEES

On crée notre base de données “DiverDB” dans WampServer. Grâce à la console SQL intégrée, on crée notre table Dive ainsi que ses différentes colonnes. Ci-dessous des captures de PhpMyAdmin où on peut voir 3 entrées dans la table Dive.



		id	dive_mins	dive_secs	dive_depth	dive_date	rating
☐	Éditer Copier Supprimer	5	2	34	43	2024-06-19	4
☐	Éditer Copier Supprimer	6	54	32	45	2024-06-06	5
☐	Éditer Copier Supprimer	7	54	43	51	2024-05-30	4

ARBORESCENCE DU PROJET

Root
- static
- templates
- db.py
- server.py

static : dossier avec notre fichier css et nos images.
templates : dossier avec nos fichiers HTML.
db.py : fichier avec nos fonctions Python.
server.py : notre serveur web (routeur).

CONNEXION A LA BASE DE DONNEES

On se connecte à la base de données en Python en utilisant mySQL Connector :

```python
def get_db_connection():
    connection = mysql.connector.connect(
        user='root',
        password='',
        host='127.0.0.1',
        database='diverDB'
    )
    return connection
```

LECTURE DU CONTENU DE LA BASE DE DONNEES

On récupère le contenu de la base de données en utilisant une route qui nous retourne toutes les plongées de la table Dive, et une autre qui en retourne une en particulier, en format JSON.

```python
@app.route('/viewDB', methods=['GET'])
def get_data():
    connection, cursor = get_cursor()
    cursor.execute("SELECT * FROM Dive")
    rows = cursor.fetchall()
    cursor.close()
    connection.close()
    return jsonify(rows)
```

```python
@app.route('/viewRecord/<int:id>', methods=['GET'])
def get_record(id):
    connection, cursor = get_cursor()
    cursor.execute("SELECT * FROM Dive WHERE id = %s", (id,))
    row = cursor.fetchone()
    cursor.close()
    connection.close()
    if row:
        return jsonify(row)
    else:
        return jsonify({'message': 'Record not found'}), 404
```

ROUTES ET SERVEUR PYTHON

Nous avons créé plusieurs routes, dont des routes de rendu de templates comme celle-ci :

```python
@app.route("/login")
def login():
    return render_template('login.html')
```

Ou des routes plus complexes, comme celle-ci dessous, qui permet d'ajouter des plongées à la base de données :

```python
@app.route('/add', methods=['POST'])
def add_record():
    dive_mins = request.form['dive_mins']
    dive_secs = request.form['dive_secs']
    dive_depth = request.form['dive_depth']
    dive_date = request.form['dive_date']
    rating = request.form['rating']
    connection, cursor = get_cursor()
    sql_query = "INSERT INTO Dive (dive_mins, dive_secs, dive_depth, dive_date, rating) VALUES (%s, %s, %s, %s, %s)"
    cursor.execute(sql_query, (dive_mins, dive_secs, dive_depth, dive_date, rating))
    connection.commit()
    cursor.close()
    connection.close()
    return "SUCCESS"
```

Il reste à mettre en place les routes pour modifier et effacer des plongées (la dernière étant déjà commencée).

HTML / CSS

Les pages HTML de template (landing page, login, sign in, liste des poissons et plongées utilisateur) ont été créées et liées au CSS. Voici par exemple à quoi ressemble la page *userDives* (plongées utilisateur) et la page d'accueil (composée ici de deux boutons) :