Flask
Clase 32

Flask: Introducción

BA

Les damos la bienvenida

Vamos a comenzar a grabar la clase

¿Qué es un Controlador?

El controlador entre otras cosas contiene el código fuente necesario para responder a las acciones que el usuario solicita y servir como puente de interacción entre las vistas y el modelo de datos. Normalmente podemos tener más de un controlador dependiendo la organización de nuestro proyecto y la magnitud de este.

El controlador en este caso con FLASK y desarrollando una API-Rest será el encargado de gestionar las rutas de enlace y determinar cual será la respuesta en cada url así como también la gestión de los datos que obtiene o solicita al modelo.

Creación de métodos del controlador

En el archivo controllers.py de la aplicación vamos a crear distintas funciones que permitirán gestionar el CRUD de películas. Posteriormente en el archivo run.py se tendrá que asociar dichas funciones, con las URLs y métodos HTTP que queremos escuchar

app/controllers.py – Traer todas las películas

```
def get_all_movies():
    movies = [
                {
                    'id_movie':1,
                    'title':'Pelicula',
                    'director':'Test',
                    'release_date':'2024-01-01',
                    'banner':'banner.png'
                },
                {
                    'id_movie':2,
                    'title':'Pelicula 2',
                    'director':'Test 2',
                    'release_date':'2024-01-01',
                    'banner':'banner2.png'
                },
            ]
    return jsonify(movies)
```

run.py

```
app.route('/api/movies/', methods=['GET'])(get_all_movies)
```

Creación de métodos del controlador

app/controllers.py – Traer una película

```
def get_movie(movie_id):
    movie = {
        'id_movie':movie_id,
    }
    return jsonify(movie)
```

run.py

```
app.route('/api/movies/', methods=['POST'])(create_movie)
app.route('/api/movies/<int:movie_id>', methods=['GET'])(get_movie)
```

app/controllers.py – Crear una película

```
def create_movie():
    #datos recibidos en formato json
    data = request.json
    return jsonify({'message': 'Movie created successfully','data':data}), 201
```

Creación de métodos del controlador

app/controllers.py – Actualizar una película

```
def update_movie(movie_id):
    #datos recibidos en formato json
    data = request.json
    return jsonify({'message': 'Movie updated
successfully','data':data,'id':movie_id})
```

run.py

```
app.route('/api/movies/<int:movie_id>', methods=['PUT'])(update_movie)
app.route('/api/movies/<int:movie_id>', methods=['DELETE'])(delete_movie)
```

app/controllers.py – Eliminar una película

```
def delete_movie(movie_id):
    return jsonify({'message': 'Movie deleted successfully','id':movie_id})
```

¿Qué es un Modelo?

La capa del Modelo en la arquitectura MVC es aquella que se encarga de trabajar con los datos; por lo que normalmente dentro del modelo encontraremos mecanismos para acceder a la información y actualizar los datos en la fuente de almacenamiento, que bien podría ser una base de datos relacional o no relacional.

Creación del Modelo

En el archivo app/models.py se van a definir cada modelo que contendrá los campos y métodos que pueden ser utilizados para interactuar con la base de datos.
Por lo que vamos a utilizar la implementación de una clases, definiremos los atributos de la clase correspondientemente con los campos de la tabla “movies” de nuestra base de datos.

```python
from app.database import get_db

class Movie:
    def __init__(self, id_movie=None, title=None, director=None, release_date=None, banner=None):
        self.id_movie = id_movie
        self.title = title
        self.director = director
        self.release_date = release_date
        self.banner = banner
```

Creación del Modelo

Método para traer listado de películas

```
    @staticmethod
    def get_all():
        db = get_db()
        cursor = db.cursor()
        cursor.execute("SELECT * FROM movies")
        rows = cursor.fetchall()
        movies = [Movie(id_movie=row[0], title=row[1], director=row[2], release_date=row[3], banner=row[4])
for row in rows]
        cursor.close()
        return movies
```

Creación del Modelo

Método para traer una película

```
    @staticmethod
    def get_by_id(movie_id):
        db = get_db()
        cursor = db.cursor()
        cursor.execute("SELECT * FROM movies WHERE id_movie = %s", (movie_id,))
        row = cursor.fetchone()
        cursor.close()
        if row:
            return Movie(id_movie=row[0], title=row[1], director=row[2],
release_date=row[3], banner=row[4])
        return None
```

Creación del Modelo

Método para guardar/actualizar una película

```
def save(self):
        db = get_db()
        cursor = db.cursor()
        if self.id_movie:
            cursor.execute("""
                UPDATE movies SET title = %s, director = %s, release_date = %s, banner = %s
                WHERE id_movie = %s
            """, (self.title, self.director, self.release_date, self.banner, self.id_movie))
        else:
            cursor.execute("""
                INSERT INTO movies (title, director, release_date, banner) VALUES (%s, %s, %s, %s)
            """, (self.title, self.director, self.release_date, self.banner))
            self.id_movie = cursor.lastrowid
        db.commit()
        cursor.close()
```

Creación del Modelo

Método para eliminar una película

```
def delete(self):
        db = get_db()
        cursor = db.cursor()
        cursor.execute("DELETE FROM movies WHERE id_movie = %s", (self.id_movie,))
        db.commit()
        cursor.close()
```

Método serializador de una película

Este método nos permitirá representar la instancia de un objeto de la clase Movie, como un diccionario. Esto nos resultará conveniente para que desde el controlador podamos hacer una conversión directa desde una representación de diccionario a formato JSON.

```
def serialize(self):
        return {
            'id_movie': self.id_movie,
            'title': self.title,
            'director': self.director,
            'release_date': self.release_date,
            'banner': self.banner
        }
```

Completamos código del controlador

app/controllers.py – Traer todas las películas

```
from app.models import Movie

def get_all_movies():
    movies = Movie.get_all()
    return jsonify([movie.serialize() for movie in movies])
```

Completamos código del controlador

app/controllers.py – Traer una película

```
def get_movie(movie_id):
    movie = Movie.get_by_id(movie_id)
    if not movie:
        return jsonify({'message': 'Movie not found'}), 404
    return jsonify(movie.serialize())
```

Completamos código del controlador

app/controllers.py – Crear una película

```
def create_movie():
    data = request.json
    new_movie = Movie(title=data['title'], director=data['director'],
release_date=data['release_date'], banner=data['banner'])
    new_movie.save()
    return jsonify({'message': 'Movie created successfully'}), 201
```

Completamos código del controlador

app/controllers.py – Actualizar una película

```python
def update_movie(movie_id):
    movie = Movie.get_by_id(movie_id)
    if not movie:
        return jsonify({'message': 'Movie not found'}), 404
    data = request.json
    movie.title = data.get('title', movie.title)
    movie.director = data.get('director', movie.director)
    movie.release_date = data.get('release_date', movie.release_date)
    movie.banner = data.get('banner', movie.banner)
    movie.save()
    return jsonify({'message': 'Movie updated successfully'})
```

Completamos código del controlador

app/controllers.py – Eliminar una película

```
def delete_movie(movie_id):
    movie = Movie.get_by_id(movie_id)
    if not movie:
        return jsonify({'message': 'Movie not found'}), 404
    movie.delete()
    return jsonify({'message': 'Movie deleted successfully'})
```

No te olvides de completar la asistencia y consultar dudas

Recordá:

- Revisar la Cartelera de Novedades.
- Hacer tus consultas en el Foro.

TODO EN EL AULA VIRTUAL