Création d'une application Flask avec SQLAlchemy

Flask-SQLAlchemy est une extension pour flask qui ajoute la prise en charge de SQLAlchemy à l'application. SQLAlchemy est une boîte à outils Python et un mappeur relationnel d'objets qui permet d'accéder à la base de données SQL à l'aide de Python. SQLAlchemy est fourni avec des modèles de persistance au niveau de l'entreprise et un accès aux bases de données efficace et hautement performant.

Flask-SQLAlchemy prend en charge les moteurs de base de données SQL suivants, à condition que le pilote DBAPI approprié soit installé:

- PostgreSQL
- MySQL
- Oracle
- SQLite
- Microsoft SQL Server
- Firebird Sybase

Nous utiliserons MySQL comme moteur de base de données dans notre application, alors commençons par installer **SQLAlchemy** et commençons à configurer notre application pour notre ***service Restful basé sur CRUD.***

Remarque: tout en travaillant sur la structure de dossier appropriée du produit doit être maintenue, ce que je vous expliquerai dans d'autres articles.

CONFIGURATION DE L'ENVIRONNEMENT

Créons un nouveau répertoire appelé **flaskdbexample** , créons un environnement virtuel, puis installons **flask-sqlalchemy.**

```
mkdir flaskdbexample
cd flaskdbexample
pip install marshmallow-sqlalchemy
```

Python
Copie

Maintenant, créez un environnement virtuel dans le répertoire à l'aide de la commande suivante:

```
virtualenv venv
```

Python
Copie

Comme indiqué précédemment, nous pouvons activer l'environnement virtuel à l'aide de la commande suivante:

```
venv/Script/activate
```
Python
Copie

Une fois l'environnement virtuel activé, installons flask-sqlalchemy.
Flask et Flask-SQLAlchemy peuvent être installés à l'aide de PIP avec la commande suivante.

```
(venv)> pip install flask flask-sqlalchemy
```
Python
Copie

Maintenant, installons PyMySQL pour activer la connexion MySQL avec Flask-SQLAlchemy.

```
(venv)> pip install pymysql
```
Python
Copie

Commençons par créer app.py qui contiendra le code de notre application. Après avoir créé le fichier, nous lancerons l'application Flask.

```
from flask import Flask, request, jsonify, make_response
from flask_sqlalchemy import SQLAlchemy
from marshmallow_sqlalchemy import ModelSchema
from marshmallow import fields
```
Python
Copie

Sur cette partie, nous importons tous les modules nécessaires à notre application. Nous importons Flask pour créer une instance d'une application Web, demandons d'obtenir des données de demande, jsonify pour transformer la sortie JSON en un objet Response avec le type mimetype application / json, SQAlchemy de flask_sqlalchemy à l'accès à la base de données et Marshmallow de flask_marshmallow en objet sérialisé.

```
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI']='mysql+pymysql://<mysql_username>:<mysql_passwo
rd>@<mysql_host>:<mysql_port>/<mysql_db>'
db = SQLAlchemy(app)
```
Python
Copie

nous configurons l'URI de la base de données SQLAlchemy pour utiliser notre URI de base de données MySQL, puis nous créons un objet de SQLAlchemy nommé db, qui gérera nos activités liées à l'ORM.

CRÉATION DE BASE DE DONNÉES

Nous allons maintenant créer une application de base de données de produits qui fournira des **API CRUD RESTful** . Tous les produits seront stockés dans un tableau intitulé «produits».

Après l'objet DB déclaré, ajoutez les lignes de code suivantes pour déclarer une classe en tant que Product qui contiendra le schéma de la table products:

```python
###Models####
class Product(db.Model):
    __tablename__ = "products"
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(20))
    productDescription = db.Column(db.String(100))
    productBrand = db.Column(db.String(20))
    price = db.Column(db.Integer)

    def create(self):
      db.session.add(self)
      db.session.commit()
      return self
    def __init__(self,title,productDescription,productBrand,price):
        self.title = title
        self.productDescription = productDescription
        self.productBrand = productBrand
        self.price = price
    def __repr__(self):
        return '' % self.id
db.create_all()
```

Python

Copie

nous avons créé un modèle intitulé "Produit" qui a cinq champs - ID est un entier auto-généré et auto-incrémenté qui servira de clé primaire. «Db.create_all ()» qui demande à l'application de créer toutes les tables et bases de données spécifiées dans l'application.

```python
class ProductSchema(ModelSchema):
    class Meta(ModelSchema.Meta):
        model = Product
        sqla_session = db.session
    id = fields.Number(dump_only=True)
```

```
    title = fields.String(required=True)
    productDescription = fields.String(required=True)
    productBrand = fields.String(required=True)
    price = fields.Number(required=True)
```
Python
Copie

Le code précédent mappe l'attribut variable aux objets de champ, et dans Meta, nous définissons le modèle à relier à notre schéma. Cela devrait donc nous aider à renvoyer JSON à partir de SQLAlchemy.

CONCEVOIR DES ENDPOINTS POUR CRUD

Après avoir configuré notre modèle et retourner le schéma, nous pouvons passer à la création de nos points de terminaison. Créons notre premier point de terminaison GET / products pour renvoyer toute la

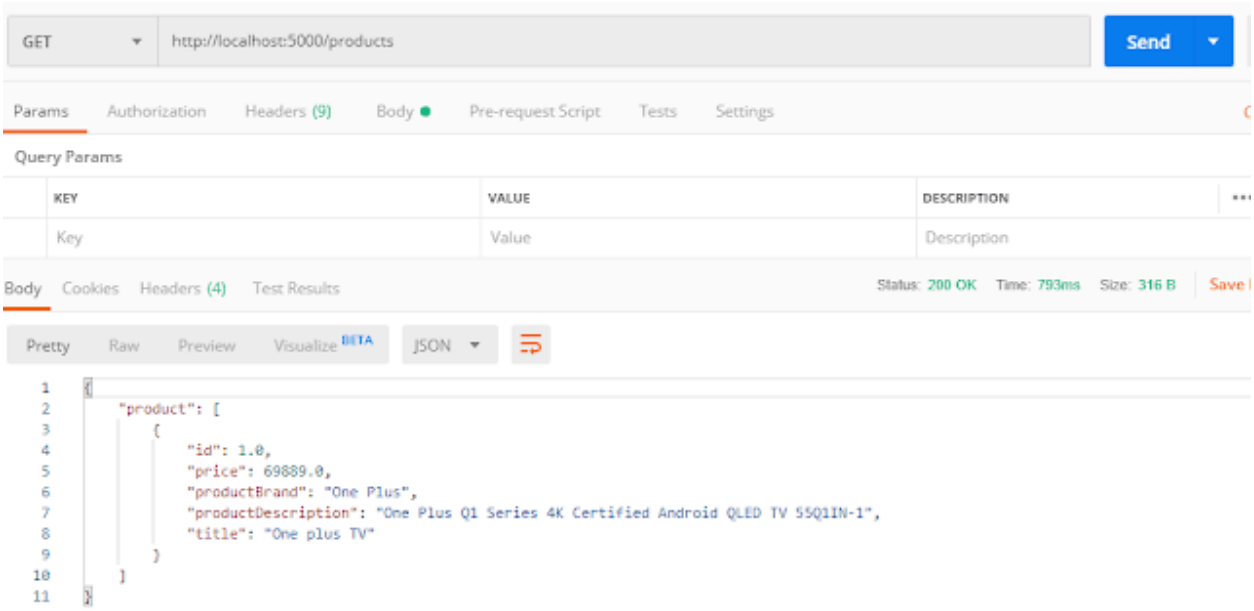
MÉTHODE GET

```
@app.route('/products', methods = ['GET'])
def index():
    get_products = Product.query.all()
    product_schema = ProductSchema(many=True)
    products = product_schema.dump(get_products)
    return make_response(jsonify({"product": products}))
```
Python
Copie
Dans cette méthode, nous récupérons tous les produits de la base de données, les vidons dans ProductSchema et nous renvoyons le résultat dans JSON.

Si vous démarrez l'application et atteignez le point de terminaison maintenant, il renverra un tableau vide car nous n'avons encore rien ajouté dans la base de données, mais allons-y et essayons le point de terminaison. Btw j'ai un enregistrement ajouté

pour exécuter l'application

```
Flask run
```
Python
Copie

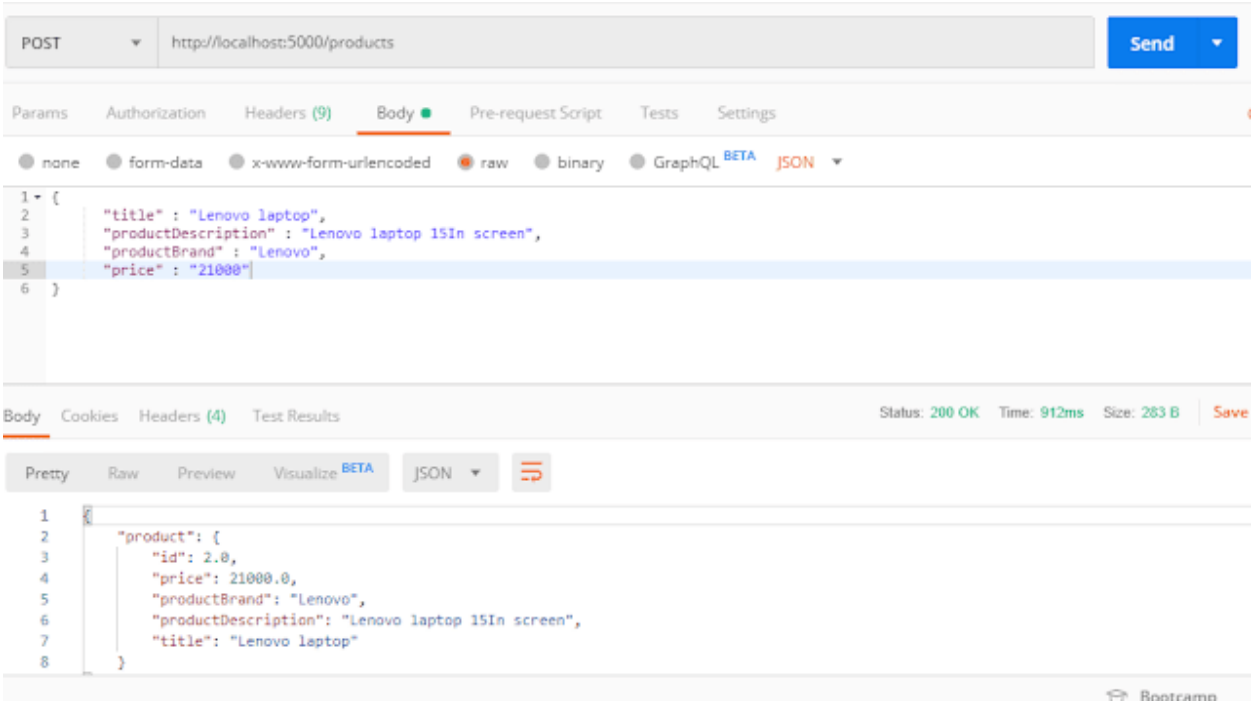


LA MÉTHODE POSTALE:

```python
@app.route('/products', methods = ['POST'])
def create_product():
    data = request.get_json()
    product_schema = ProductSchema()
    product = product_schema.load(data)
    result = product_schema.dump(product.create())
    return make_response(jsonify({"product": result}),200)
```

Python

Copie

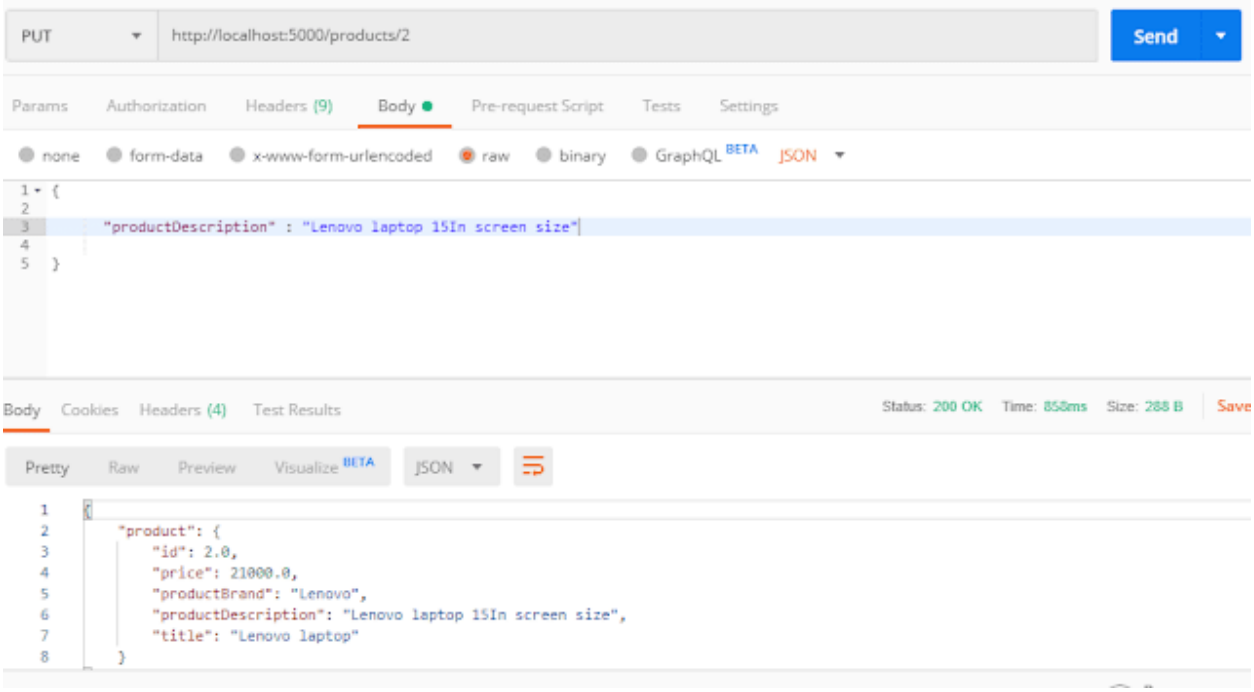


LA MÉTHODE DE MISE À JOUR (PUT):

```python
@app.route('/products/<id>', methods = ['PUT'])
def update_product_by_id(id):
    data = request.get_json()
    get_product = Product.query.get(id)
    if data.get('title'):
        get_product.title = data['title']
    if data.get('productDescription'):
        get_product.productDescription = data['productDescription']
    if data.get('productBrand'):
        get_product.productBrand = data['productBrand']
    if data.get('price'):
        get_product.price= data['price']
    db.session.add(get_product)
    db.session.commit()
    product_schema = ProductSchema(only=['id', 'title',
'productDescription','productBrand','price'])
    product = product_schema.dump(get_product)
    return make_response(jsonify({"product": product}))
```

Python

Copie

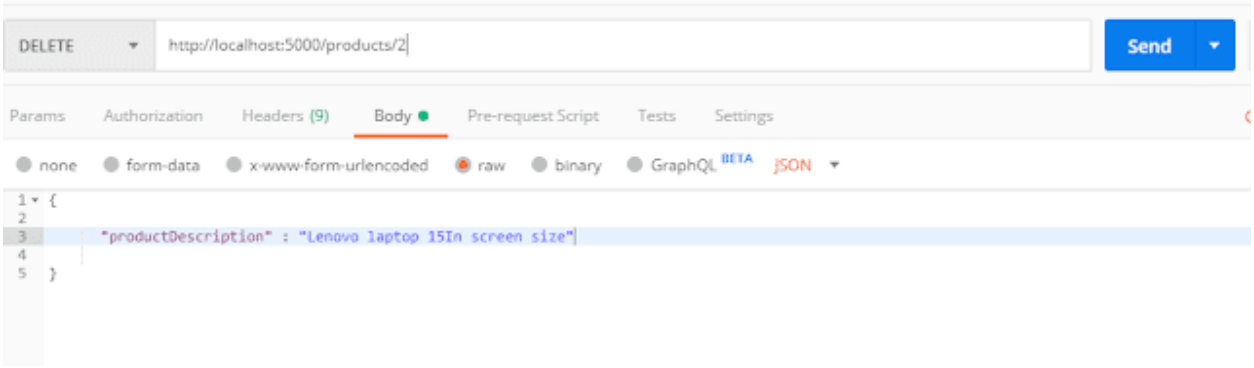


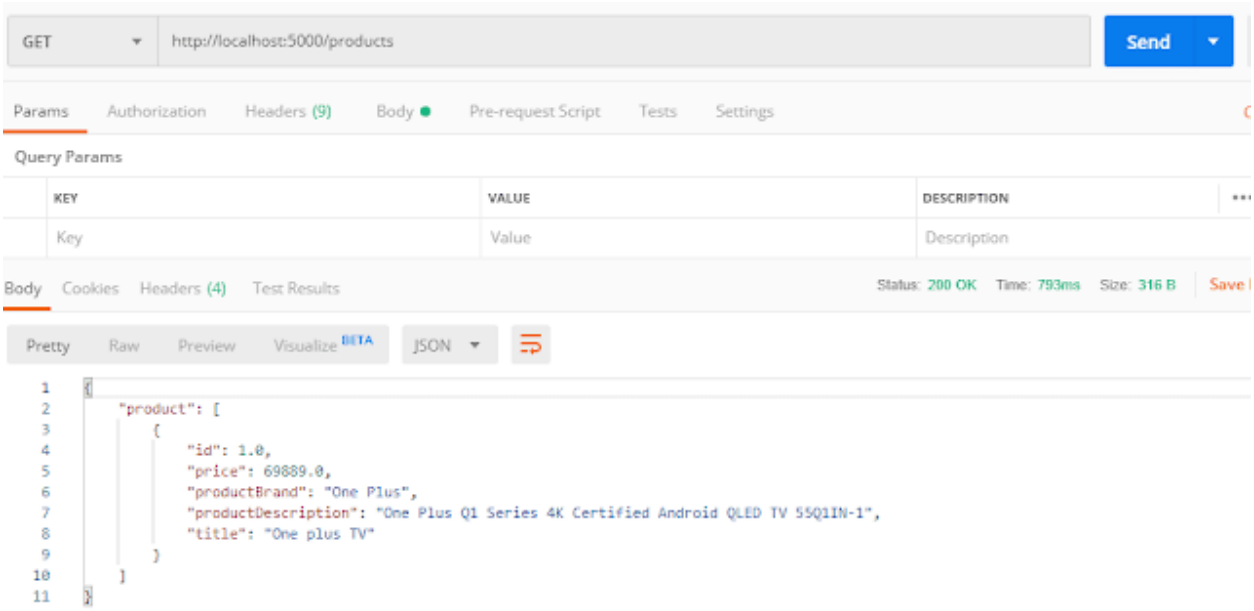
LA MÉTHODE DE SUPPRESSION PAR ID:

```python
@app.route('/products/<id>', methods = ['DELETE'])
def delete_product_by_id(id):
    get_product = Product.query.get(id)
    db.session.delete(get_product)
    db.session.commit()
    return make_response("",204)
```

Python

Copie





Code complet pour les tests:

```
from flask import Flask, request, jsonify, make_response
from flask_sqlalchemy import SQLAlchemy
from marshmallow_sqlalchemy import ModelSchema
from marshmallow import fields
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] =
'mysql+pymysql://<username>:<pass>@localhost:3306/<DB>'
db = SQLAlchemy(app)

###Models####
class Product(db.Model):
    __tablename__ = "products"
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(20))
    productDescription = db.Column(db.String(100))
    productBrand = db.Column(db.String(20))
    price = db.Column(db.Integer)

    def create(self):
      db.session.add(self)
      db.session.commit()
      return self
    def __init__(self,title,productDescription,productBrand,price):
        self.title = title
        self.productDescription = productDescription
        self.productBrand = productBrand
        self.price = price
    def __repr__(self):
```

```
        return '' % self.id
db.create_all()
class ProductSchema(ModelSchema):
    class Meta(ModelSchema.Meta):
        model = Product
        sqla_session = db.session
    id = fields.Number(dump_only=True)
    title = fields.String(required=True)
    productDescription = fields.String(required=True)
    productBrand = fields.String(required=True)
    price = fields.Number(required=True)

@app.route('/products', methods = ['GET'])
def index():
    get_products = Product.query.all()
    product_schema = ProductSchema(many=True)
    products = product_schema.dump(get_products)
    return make_response(jsonify({"product": products}))
@app.route('/products/<id>', methods = ['GET'])
def get_product_by_id(id):
    get_product = Product.query.get(id)
    product_schema = ProductSchema()
    product = product_schema.dump(get_product)
    return make_response(jsonify({"product": product}))
@app.route('/products/<id>', methods = ['PUT'])
def update_product_by_id(id):
    data = request.get_json()
    get_product = Product.query.get(id)
    if data.get('title'):
        get_product.title = data['title']
    if data.get('productDescription'):
        get_product.productDescription = data['productDescription']
    if data.get('productBrand'):
        get_product.productBrand = data['productBrand']
    if data.get('price'):
        get_product.price= data['price']
    db.session.add(get_product)
    db.session.commit()
    product_schema = ProductSchema(only=['id', 'title',
'productDescription','productBrand','price'])
    product = product_schema.dump(get_product)
    return make_response(jsonify({"product": product}))
@app.route('/products/<id>', methods = ['DELETE'])
def delete_product_by_id(id):
    get_product = Product.query.get(id)
    db.session.delete(get_product)
    db.session.commit()
    return make_response("",204)
@app.route('/products', methods = ['POST'])
def create_product():
    data = request.get_json()
    product_schema = ProductSchema()
    product = product_schema.load(data)
    result = product_schema.dump(product.create())
    return make_response(jsonify({"product": result}),200)
```

```python
if __name__ == "__main__":
    app.run(debug=True)
```
Python
Copie

Conclusion:
Nous avons donc créé et testé notre exemple d'application Flask-MySQL CRUD. Nous aborderons les relations d'objets complexes à l'aide de Flask-SQLAlchemy dans le prochain article, puis nous créerons une application Flask CRUD similaire.