Introduction to Flask

Flask is a lightweight WSGI web application framework in Python. It is designed with simplicity and flexibility in mind, making it easy to get started with and scale up as needed.

- Installation

```
pip install Flask
```

- Your First Flask App Create a file named app.py:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return "Hello, Flask!"

if __name__ == '__main__':
    app.run(debug=True)
```

- Run the application:

```
python app.py
```

Connecting to mysql database

```
# install
pip install Flask flask-mysqldb
```

Configuration

Create a config.py file to store your database configuration settings:

```
#config.py

class Config:
    SECRET_KEY = 'your_secret_key'
    MYSQL_HOST = 'localhost'
    MYSQL_USER = 'your_mysql_user'
```

```
    MYSQL_PASSWORD = 'your_mysql_password'
    MYSQL_DB = 'your_database_name'
```

Models

Create a models.py file to define your database models and initialize the MySQL connection:

```
# models.py

from flask_mysqldb import MySQL
from flask import current_app as app

mysql = MySQL()

def init_db(app):
    mysql.init_app(app)

def create_table():
    cursor = mysql.connection.cursor()
    cursor.execute('''CREATE TABLE IF NOT EXISTS items (
        id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(100) NOT NULL,
        description TEXT NOT NULL
    )''')
    mysql.connection.commit()
    cursor.close()
```

Routes

Create a routes.py file to define your API routes:

```
# routes.py

from flask import Blueprint, request, jsonify
from .models import mysql

api = Blueprint('api', __name__)

@api.route('/items', methods=['GET'])
def get_items():
    cursor = mysql.connection.cursor()
    cursor.execute("SELECT * FROM items")
    items = cursor.fetchall()
    cursor.close()
    return jsonify(items)

@api.route('/item/<int:id>', methods=['GET'])
def get_item(id):
    cursor = mysql.connection.cursor()
```

```
    cursor.execute("SELECT * FROM items WHERE id = %s", (id,))
    item = cursor.fetchone()
    cursor.close()
    return jsonify(item)

@api.route('/item', methods=['POST'])
def add_item():
    data = request.get_json()
    name = data['name']
    description = data['description']
    cursor = mysql.connection.cursor()
    cursor.execute("INSERT INTO items (name, description) VALUES (%s, %s)", (name,
description))
    mysql.connection.commit()
    cursor.close()
    return jsonify({'message': 'Item added successfully'})

@api.route('/item/<int:id>', methods=['PUT'])
def update_item(id):
    data = request.get_json()
    name = data['name']
    description = data['description']
    cursor = mysql.connection.cursor()
    cursor.execute("UPDATE items SET name = %s, description = %s WHERE id = %s",
(name, description, id))
    mysql.connection.commit()
    cursor.close()
    return jsonify({'message': 'Item updated successfully'})

@api.route('/item/<int:id>', methods=['DELETE'])
def delete_item(id):
    cursor = mysql.connection.cursor()
    cursor.execute("DELETE FROM items WHERE id = %s", (id,))
    mysql.connection.commit()
    cursor.close()
    return jsonify({'message': 'Item deleted successfully'})
```

Application Setup

Create an app.py file to initialize and run your Flask application:

```
# app.py

from flask import Flask
from config import Config
from models import init_db, create_table
from routes import api

app = Flask(__name__)
app.config.from_object(Config)
```

```
# Initialize database
init_db(app)

# Create tables
with app.app_context():
    create_table()

# Register Blueprints
app.register_blueprint(api, url_prefix='/api')

if __name__ == '__main__':
    app.run(debug=True)
```

Cursor

In Python, a cursor is an object used to interact with the database through a connection. It allows you to execute SQL queries and fetch data from the database. The cursor acts as a pointer that manages the context of a fetch operation.

When using a library like mysql.connector to connect to a MySQL database, a cursor is created from the connection object. This cursor can then be used to execute SQL commands and retrieve data.

Key Functions of a Cursor

We can execute SQL commands (such as SELECT, INSERT, UPDATE, DELETE) using the cursor.

- Fetching Data: Example:cursor.execute("SELECT * FROM users") After executing a SELECT statement, you can fetch the results using methods like fetchone(), fetchall(), or fetchmany() Example: rows = cursor.fetchall()
- Managing Transactions: For databases that support transactions, the cursor can be used to commit or roll back transactions. Example: connection.commit()
- Parameter Substitution: You can use placeholders in your SQL queries and pass parameters to avoid SQL injection. Example: cursor.execute("SELECT * FROM users WHERE id = %s", (user_id,))

Blueprint

In Flask, a Blueprint is a way to organize your application into smaller and reusable components. It allows you to group routes, handlers, and other functionalities into a single module, which can then be registered with the main application. This helps in maintaining a modular and scalable codebase, especially for larger applications.

```
from routes.user import user_bp
```

This line imports a Blueprint instance named user_bp from the routes.user module. The user_bp is expected to be defined in the routes/user.py file, where all routes related to user functionalities are grouped.

Registering the Blueprint

```
app.register_blueprint(user_bp, url_prefix="/user")
```

- This line registers the user_bp Blueprint with the main Flask application instance app.
- The url_prefix argument specifies a prefix for all routes defined in the user_bp Blueprint. This means that all routes in user_bp will be accessible under the /user URL path.
