Flask Blueprint CRUD (via API)

Here's a basic example of a CRUD Flask application using a single Blueprint and MySQL. Please note that we'll use Flask-SQLAlchemy for ORM and Flask-Marshmallow for request validation and serialization.

Please, install necessary packages:

bash

```
pip install flask flask_sqlalchemy flask_marshmallow marshmallow-sqlalchemy flask-mysqldb
```

Here's the code:

python

```
from flask import Flask, request, Blueprint, jsonify
from flask_sqlalchemy import SQLAlchemy
from flask_marshmallow import Marshmallow

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql://username:password@localhost/dbname'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)
ma = Marshmallow(app)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True)
    usernumber = db.Column(db.String(20), unique=True)
    comment = db.Column(db.String(200))

    def __init__(self, username, usernumber, comment):
        self.username = username
        self.usernumber = usernumber
        self.comment = comment

class UserSchema(ma.SQLAlchemyAutoSchema):
    class Meta:
        model = User
```

```
        load_instance = True

user_schema = UserSchema()
users_schema = UserSchema(many=True)

bp = Blueprint('user_api', __name__)

@bp.route('/user', methods=['POST'])
def add_user():
    username = request.json['username']
    usernumber = request.json['usernumber']
    comment = request.json['comment']
    new_user = User(username, usernumber, comment)
    db.session.add(new_user)
    db.session.commit()
    return user_schema.jsonify(new_user)

@bp.route('/user', methods=['GET'])
def get_users():
    all_users = User.query.all()
    result = users_schema.dump(all_users)
    return jsonify(result)

@bp.route('/user/<id>', methods=['GET'])
def get_user(id):
    user = User.query.get(id)
    return user_schema.jsonify(user)

@bp.route('/user/<id>', methods=['PUT'])
def update_user(id):
    user = User.query.get(id)
    username = request.json['username']
    usernumber = request.json['usernumber']
    comment = request.json['comment']
    user.username = username
    user.usernumber = usernumber
    user.comment = comment
    db.session.commit()
    return user_schema.jsonify(user)
```

```
@bp.route('/user/<id>', methods=['DELETE'])
def delete_user(id):
    user = User.query.get(id)
    db.session.delete(user)
    db.session.commit()
    return user_schema.jsonify(user)

app.register_blueprint(bp, url_prefix='/api')

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

This script creates a Flask application with a single blueprint (`bp`) that provides the CRUD functionality. It defines the following endpoints:

- `POST /api/user`: Create a new user.
- `GET /api/user`: Retrieve a list of all users.
- `GET /api/user/<id>`: Retrieve a single user by ID.
- `PUT /api/user/<id>`: Update a user by ID.
- `DELETE /api/user/<id>`: Delete a user by ID.

Note:

1. This is a very basic CRUD application and doesn't have any security measures. In a real-world application, you would want to add authentication, input sanitization, error handling, etc.
2. Replace `'mysql://username:password@localhost/dbname'` with your MySQL database details.
3. Also, don't forget to run the DB (Xampp)

Instead of creating a REST API, we could adapt the example to be a simple web application with HTML templates for the user interface.

We will use Flask's built-in `render_template` function to serve HTML files and `redirect` function to redirect users to different routes.

Let's say we have an HTML file named `index.html` in a templates folder, and it's set up to display a list of users and a form to create a new user. We could render that file from our `get_users` route like so:

```
from flask import render_template, redirect, url_for, flash

# ...

@bp.route('/users', methods=['GET', 'POST'])
def users():
    if request.method == 'POST':
        username = request.form['username']
        usernumber = request.form['usernumber']
        comment = request.form['comment']
        new_user = User(username, usernumber, comment)
        db.session.add(new_user)
        db.session.commit()
        flash('User added successfully!', 'success')
        return redirect(url_for('users'))

    all_users = User.query.all()
    return render_template('index.html', users=all_users)
```

In this code, we've combined the create and list users functionality into one route. If the request method is POST, we create a new user. Otherwise, we display all users.

Here's how you might handle the update and delete functionality:

```
@bp.route('/user/update/<id>', methods=['GET', 'POST'])
def update_user(id):
    user = User.query.get(id)

    if request.method == 'POST':
        user.username = request.form['username']
        user.usernumber = request.form['usernumber']
        user.comment = request.form['comment']
        db.session.commit()
        flash('User updated successfully!', 'success')
        return redirect(url_for('users'))

    return render_template('update.html', user=user)

@bp.route('/user/delete/<id>', methods=['POST'])
def delete_user(id):
```

```
    user = User.query.get(id)
    db.session.delete(user)
    db.session.commit()
    flash('User deleted successfully!', 'success')
    return redirect(url_for('users'))
```

In `update_user`, if the request method is POST, we update the user. Otherwise, we display the current user data in a form for editing. In `delete_user`, we simply delete the user and then redirect the client back to the users list.

Note that `flash` function is used to display one-time notifications to the user.

You'll need to create `index.html` and `update.html` templates with the necessary forms for creating, updating and deleting users. For security reasons, ensure you protect against Cross Site Request Forgery (CSRF) in your forms. Flask-WTF and Flask-SeaSurf are some of the libraries that provide CSRF protection.

Again, remember that this is a very basic example. In a real-world application, you would want to add proper error handling, form validation, and user authentication.

Here's a basic example of an `update.html` file using Bootstrap 4. In this example, we're creating a form with fields for username, usernumber, and comment. These fields are pre-populated with the current user's information.

This template uses POST to submit the form, which should trigger the update logic in the `update_user` function you defined earlier.

Please note that this is a very simple example and doesn't include any validation or error handling. A real-world application would need more robust error handling and potentially some client-side validation.

```
<!doctype html>
<html lang="en">

<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
```

```
    <title>Update User</title>
</head>

<body>
    <div class="container">
        <h1 class="my-4">Update User</h1>

        <form action="{{ url_for('update_user', id=user.id) }}" method="POST">
            <div class="form-group">
                <label for="username">Username</label>
                <input type="text" class="form-control" id="username" name="username"
value="{{ user.username }}" required>
            </div>
            <div class="form-group">
                <label for="usernumber">User Number</label>
                <input type="text" class="form-control" id="usernumber" name="usernumber"
value="{{ user.usernumber }}" required>
            </div>
            <div class="form-group">
                <label for="comment">Comment</label>
                <input type="text" class="form-control" id="comment" name="comment" value="{{
user.comment }}" required>
            </div>
            <button type="submit" class="btn btn-primary">Update User</button>
        </form>
    </div>

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"></script>
    <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>
</body>

</html>
```

This form includes inputs for username, usernumber, and comment. The `value` attributes are set to the current user's data (e.g., `{{ user.username }}`), so when the form loads, it shows the current data for the user. When the form is submitted, the data in these inputs is sent as a POST request to the `/user/update/<id>` route, which should update the user in the database.

Make sure to include the user's id in the action attribute of the form tag to make sure the correct user gets updated. This is done using the `url_for()` function (e.g., `{{ url_for('update_user', id=user.id) }}`).

--

Revision #5
Created 2023-06-14 08:26:51 UTC by Max
Updated 2023-06-14 17:15:27 UTC by Max