Flask Cupcakes: Solution

Springboard

Download solution code.

Flask

models.py

```
"""Models for Cupcake app."""

from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()


DEFAULT_IMAGE = "https://tinyurl.com/demo-cupcake"


class Cupcake(db.Model):
    """Cupcake."""

    __tablename__ = "cupcakes"

    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    flavor = db.Column(db.Text, nullable=False)
    size = db.Column(db.Text, nullable=False)
    rating = db.Column(db.Float, nullable=False)
    image = db.Column(db.Text, nullable=False, default=DEFAULT_IMAGE)

    def to_dict(self):
        """Serialize cupcake to a dict of cupcake info."""

        return {
            "id": self.id,
            "flavor": self.flavor,
            "rating": self.rating,
            "size": self.size,
            "image": self.image,
        }


def connect_db(app):
    """Connect to database."""

    db.app = app
    db.init_app(app)
```

app.py

```
"""Flask app for Cupcakes"""

from flask import flask, request, jsonify, render_template

from models import db, connect_db, Cupcake

app = Flask(__name__)

app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql:///cupcakes'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['SECRET_KEY'] = "oh-so-secret"

connect_db(app)


@app.route("/")
def root():
    """Render homepage."""

    return render_template("index.html")


@app.route("/api/cupcakes")
def list_cupcakes():
    """Return all cupcakes in system.

    Returns JSON like:
        {cupcakes: [{id, flavor, rating, size, image}, ...]}
    """

    cupcakes = [cupcake.to_dict() for cupcake in Cupcake.query.all()]
    return jsonify(cupcakes=cupcakes)


@app.route("/api/cupcakes", methods=["POST"])
def create_cupcake():
    """Add cupcake, and return data about new cupcake.

    Returns JSON like:
        {cupcake: {id, flavor, rating, size, image}}
    """

    data = request.json

    cupcake = Cupcake(
        flavor=data["flavor"],
        rating=data["rating"],
        size=data["size"],
        image=data["image"] or None)

    db.session.add(cupcake)
    db.session.commit()

    # POST requests should return HTTP status of 201 CREATED
    return (jsonify(cupcake=cupcake.to_dict()), 201)


@app.route("/api/cupcakes/<int:cupcake_id>")
def get_cupcake(cupcake_id):
    """Return data on specific cupcake.

    Returns JSON like:
        {cupcake: [id, flavor, rating, size, image]}
    """

    cupcake = Cupcake.query.get_or_404(cupcake_id)
    return jsonify(cupcake=cupcake.to_dict())


@app.route("/api/cupcakes/<int:cupcake_id>", methods=["PATCH"])
def update_cupcake(cupcake_id):
    """Update cupcake from data in request. Return updated data.

    Returns JSON like:
        {cupcake: [id, flavor, rating, size, image]}
    """

    data = request.json

    cupcake = Cupcake.query.get_or_404(cupcake_id)

    cupcake.flavor = data['flavor']
    cupcake.rating = data['rating']
    cupcake.size = data['size']
    cupcake.image = data['image']

    db.session.add(cupcake)
    db.session.commit()

    return jsonify(cupcake=cupcake.to_dict())


@app.route("/api/cupcakes/<int:cupcake_id>", methods=["DELETE"])
def remove_cupcake(cupcake_id):
    """Delete cupcake and return confirmation message.

    Returns JSON of {message: "Deleted"}
    """

    cupcake = Cupcake.query.get_or_404(cupcake_id)
    db.session.delete(cupcake)
    db.session.commit()

    return jsonify(message="Deleted")
```

HTML & CSS

templates/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Cupcakes</title>
  <link rel="stylesheet" href="/static/style.css">
</head>
<body>

<h1>See all the cupcakes!</h1>

<ul id="cupcakes-list">
</ul>

<h2>Add a new cupcake!</h2>

<form id="new-cupcake-form">
  <div>
    <label for="form-flavor">Flavor: </label>
    <input name="flavor" id="form-flavor">
  </div>

  <div>
    <label for="form-size">Size: </label>
    <input name="size" id="form-size">
  </div>

  <div>
    <label for="form-rating">Rating: </label>
    <input type="number" name="rating" id="form-rating">
  </div>

  <div>
    <label for="form-image">Image: </label>
    <input name="image" id="form-image">
  </div>

  <button>Add!</button>
</form>

<script src="https://jsdelivr.net/npm/jquery"></script>
<script src="https://jsdelivr.net/npm/axios/dist/axios.js"></script>
<script src="/static/cupcakes.js"></script>

</body>
</html>
```

static/style.css

```
.cupcake-img {
  max-width: 200px;
  max-height: 200px;
}
```

JavaScript

static/cupcakes.js

```
const BASE_URL = "http://localhost:5000/api";


/** given data about a cupcake, generate html */

function generateCupcakeHTML(cupcake) {
  return `
    <div data-cupcake-id=${cupcake.id}>
      <li>
        ${cupcake.flavor} / ${cupcake.size} / ${cupcake.rating}
        <button class="delete-button">X</button>
      </li>
      <img class="cupcake-img"
           src="${cupcake.image}"
           alt="the image provided">
    </div>
  `;
}


/** put initial cupcakes on page. */

async function showInitialCupcakes() {
  const response = await axios.get(`${BASE_URL}/cupcakes`);

  for (let cupcakeData of response.data.cupcakes) {
    let newCupcake = $(generateCupcakeHTML(cupcakeData));
    $("#cupcakes-list").append(newCupcake);
  }
}


/** handle form for adding of new cupcakes */

$("#new-cupcake-form").on("submit", async function (evt) {
  evt.preventDefault();

  let flavor = $("#form-flavor").val();
  let rating = $("#form-rating").val();
  let size = $("#form-size").val();
  let image = $("#form-image").val();

  const newCupcakeResponse = await axios.post(`${BASE_URL}/cupcakes`, {
    flavor,
    rating,
    size,
    image
  });

  let newCupcake = $(generateCupcakeHTML(newCupcakeResponse.data.cupcake));
  $("#cupcakes-list").append(newCupcake);
  $("#new-cupcake-form").trigger("reset");
});


/** handle clicking delete: delete cupcake */

$("#cupcakes-list").on("click", ".delete-button", async function (evt) {
  evt.preventDefault();
  let $cupcake = $(evt.target).closest("div");
  let cupcakeId = $cupcake.attr("data-cupcake-id");

  await axios.delete(`${BASE_URL}/cupcakes/${cupcakeId}`);
  $cupcake.remove();
});


$(showInitialCupcakes);
```

Tests

tests.py

```
from unittest import TestCase

from app import app
from models import db, Cupcake

# Use test database and don't clutter tests with SQL
app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql:///cupcakes_test'
app.config['SQLALCHEMY_ECHO'] = False

# Make Flask errors be real errors, rather than HTML pages with error info
app.config['TESTING'] = True

db.drop_all()
db.create_all()


CUPCAKE_DATA = {
    "flavor": "TestFlavor",
    "size": "TestSize",
    "rating": 5,
    "image": "http://test.com/cupcake.jpg"
}

CUPCAKE_DATA_2 = {
    "flavor": "TestFlavor2",
    "size": "TestSize2",
    "rating": 10,
    "image": "http://test.com/cupcake2.jpg"
}


class CupcakeViewsTestCase(TestCase):
    """Tests for views of API."""

    def setUp(self):
        """Make demo data."""

        Cupcake.query.delete()

        cupcake = Cupcake(**CUPCAKE_DATA)
        db.session.add(cupcake)
        db.session.commit()

        self.cupcake = cupcake

    def tearDown(self):
        """Clean up failed transactions."""

        db.session.rollback()

    def test_list_cupcakes(self):
        with app.test_client() as client:
            resp = client.get("/api/cupcakes")

            self.assertEqual(resp.status_code, 200)

            data = resp.json
            self.assertEqual(data, {
                "cupcakes": [
                    {
                        "id": self.cupcake.id,
                        "flavor": "TestFlavor",
                        "size": "TestSize",
                        "rating": 5,
                        "image": "http://test.com/cupcake.jpg"
                    }
                ]
            })

    def test_get_cupcake(self):
        with app.test_client() as client:
            url = f"/api/cupcakes/{self.cupcake.id}"
            resp = client.get(url)

            self.assertEqual(resp.status_code, 200)
            data = resp.json
            self.assertEqual(data, {
                "cupcake": {
                    "id": self.cupcake.id,
                    "flavor": "TestFlavor",
                    "size": "TestSize",
                    "rating": 5,
                    "image": "http://test.com/cupcake.jpg"
                }
            })

    def test_get_cupcake_missing(self):
        with app.test_client() as client:
            resp = client.get("/api/cupcakes/0")

            self.assertEqual(resp.status_code, 404)

    def test_create_cupcake(self):
        with app.test_client() as client:
            url = "/api/cupcakes"
            resp = client.post(url, json=CUPCAKE_DATA_2)

            self.assertEqual(resp.status_code, 201)

            data = resp.json

            # Don't know what ID we'll get, make sure it's an int & normal for cupcake
            self.assertIsInstance(data['cupcake']['id'], int)
            del data['cupcake']['id']

            self.assertEqual(data, {
                "cupcake": {
                    "flavor": "TestFlavor2",
                    "size": "TestSize2",
                    "rating": 10,
                    "image": "http://test.com/cupcake2.jpg"
                }
            })

            self.assertEqual(Cupcake.query.count(), 2)

    def test_update_cupcake(self):
        with app.test_client() as client:
            url = f"/api/cupcakes/{self.cupcake.id}"
            resp = client.patch(url, json=CUPCAKE_DATA_2)

            self.assertEqual(resp.status_code, 200)

            data = resp.json
            self.assertEqual(data, {
                "cupcake": {
                    "id": self.cupcake.id,
                    "flavor": "TestFlavor2",
                    "size": "TestSize2",
                    "rating": 10,
                    "image": "http://test.com/cupcake2.jpg"
                }
            })

            self.assertEqual(Cupcake.query.count(), 1)

    def test_update_cupcake_missing(self):
        with app.test_client() as client:
            resp = client.patch("/api/cupcakes/0", json=CUPCAKE_DATA_2)

            self.assertEqual(resp.status_code, 404)

    def test_delete_cupcake(self):
        with app.test_client() as client:
            url = f"/api/cupcakes/{self.cupcake.id}"
            resp = client.delete(url)

            self.assertEqual(resp.status_code, 200)

            data = resp.json
            self.assertEqual(data, {"message": "Deleted"})

            self.assertEqual(Cupcake.query.count(), 0)

    def test_delete_cupcake_missing(self):
        with app.test_client() as client:
            url = "/api/cupcakes/0"
            resp = client.delete(url)

            self.assertEqual(resp.status_code, 404)
```