Guia Completo: Construindo um App de Receitas com Flask do Zero

Este guia é um passo a passo completo para criar a aplicação "Plataforma de Receitas". Vamos começar de uma pasta vazia e construir toda a lógica para demonstrar os relacionamentos One-to-One, One-to-Many e um Many-to-Many avançado com um "Association Object".

Objetivo Final

Criar uma aplicação web onde:

1. Um `Chef` tem um `PerfilChef` com sua especialidade (**One-to-One**).
2. Um `Chef` pode publicar várias `Receitas` (**One-to-Many**).
3. Uma `Receita` é composta por vários `Ingredientes` com suas respectivas `Quantidades`, e um `Ingrediente` pode ser usado em várias receitas (**Many-to-Many com dados extras**).

Passo 0: Preparação do Ambiente

Antes de escrever qualquer código, precisamos organizar nosso ambiente de trabalho.

1. **Crie a Pasta do Projeto:** Abra seu terminal ou prompt de comando e crie uma pasta para o projeto.
   ```
   mkdir receitas_app
   cd receitas_app
   ```

2. **Crie um Ambiente Virtual:** É uma boa prática isolar as dependências de cada projeto.
   ```
   python -m venv venv
   ```

3. **Ative o Ambiente Virtual:**
   - No macOS/Linux: `source venv/bin/activate`
   - No Windows: `venv\Scripts\activate` (Você verá `(venv)` no início da linha do seu terminal).

4. **Instale as Bibliotecas:** Precisamos do Flask e da extensão SQLAlchemy para interagir com o banco de dados.
   ```
   pip install Flask Flask-SQLAlchemy
   ```

5. **Crie a Estrutura de Pastas:** Dentro de `receitas_app`, crie as pastas e arquivos vazios que vamos usar.

```
/receitas_app/
|-- app.py
|-- models.py
|-- /templates/
|-- /static/
    |-- /css/
```

Passo 1: Configuração Inicial do Flask e SQLAlchemy (`app.py`)

Vamos começar com a estrutura básica da nossa aplicação.

Edite o arquivo `app.py`:

```
Passo 1: Configuração Inicial do Flask e SQLAlchemy (app.py)
Vamos começar com a estrutura básica da nossa aplicação.

Edite o arquivo app.py:

import os
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

# Define o caminho base do projeto
basedir = os.path.abspath(os.path.dirname(__file__))

# Cria a instância da aplicação Flask
app = Flask(__name__)

# Configurações da aplicação
# Define o caminho para o nosso banco de dados SQLite
app.config['SQLALCHEMY_DATABASE_URI'] = \
            'sqlite:///'   +   os.path.join(basedir,   'instance',
'receitas.db')
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

# Cria a pasta 'instance' se ela não existir
instance_path = os.path.join(basedir, 'instance')
if not os.path.exists(instance_path):
    os.makedirs(instance_path)

# Inicializa a extensão SQLAlchemy
db = SQLAlchemy(app)

# O restante do nosso código (rotas, modelos) virá aqui...

# Bloco para executar a aplicação
if __name__ == '__main__':
    app.run(debug=True)
```

Explicação Vital: Este código inicializa o Flask, diz a ele onde nosso banco de dados será salvo (em um arquivo `receitas.db` dentro de uma pasta `instance`) e cria o objeto `db`, que será nossa ponte de comunicação com o banco de dados.

Passo 2: Modelagem dos Dados (`models.py`)

Esta é a parte mais importante. Vamos definir a estrutura das nossas tabelas e os relacionamentos entre elas.

Edite o arquivo `models.py`:

```
from app import db

# 1. O "Association Object" para a relação M:M
class ReceitaIngrediente(db.Model):
    __tablename__ = 'receita_ingredientes'
          receita_id   =   db.Column(db.Integer,   db.ForeignKey('receita.id'),
primary_key=True)
     ingrediente_id = db.Column(db.Integer, db.ForeignKey('ingrediente.id'),
primary_key=True)
    quantidade = db.Column(db.String(50), nullable=False)

    # Relações de volta para Receita e Ingrediente
                        ingrediente      =      db.relationship("Ingrediente",
back_populates="receitas_associadas")
                               receita       =       db.relationship("Receita",
back_populates="ingredientes_associados")

# 2. Modelo Chef (O "Um" de One-to-One e One-to-Many)
class Chef(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    nome = db.Column(db.String(100), nullable=False)

    # Relação 1:1 com PerfilChef
           perfil   =   db.relationship('PerfilChef',   back_populates='chef',
uselist=False, cascade="all, delete-orphan")

    # Relação 1:M com Receita
    receitas = db.relationship('Receita', back_populates='chef')

# 3. Modelo PerfilChef (O outro "Um" de One-to-One)
class PerfilChef(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    especialidade = db.Column(db.String(100))
    anos_experiencia = db.Column(db.Integer)
    chef_id = db.Column(db.Integer, db.ForeignKey('chef.id'), nullable=False,
unique=True)

    chef = db.relationship('Chef', back_populates='perfil')

# 4. Modelo Receita (O "Muitos" de One-to-Many e parte do M:M)
class Receita(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    titulo = db.Column(db.String(200), nullable=False)
    instrucoes = db.Column(db.Text, nullable=False)
    chef_id = db.Column(db.Integer, db.ForeignKey('chef.id'), nullable=False)
```

```
    chef = db.relationship('Chef', back_populates='receitas')

    # Relação com o "Association Object"
          ingredientes_associados   =   db.relationship('ReceitaIngrediente',
back_populates='receita', cascade="all, delete-orphan")

# 5. Modelo Ingrediente (A outra parte do M:M)
class Ingrediente(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    nome = db.Column(db.String(100), nullable=False, unique=True)

            receitas_associadas    =    db.relationship('ReceitaIngrediente',
back_populates='ingrediente', cascade="all, delete-orphan")
```

Explicação Vital:

- `uselist=False` no modelo `Chef` estabelece a relação **One-to-One**.
- A `ForeignKey` em `Receita` (`chef_id`) estabelece a relação **One-to-Many**.
- A classe `ReceitaIngrediente` é o coração da relação **Many-to-Many**, permitindo-nos armazenar a `quantidade`. As relações em `Receita` e `Ingrediente` agora apontam para este modelo intermediário.

Passo 3: Criando os Templates (A Interface do Usuário)

Agora, vamos criar a parte visual da nossa aplicação.

1. **Crie `templates/base.html`:** Este será nosso layout principal.

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %}Plataforma de Receitas{% endblock %}</title>
                  <link    rel="stylesheet"    href="{{    url_for('static',
filename='css/style.css') }}">
</head>
<body>
    <nav>
           <a href="{{ url_for('index') }}" class="nav-brand">Plataforma de
Receitas</a>
            <a href="{{ url_for('criar_receita') }}" class="nav-link">Nova
Receita</a>
    </nav>
    <main class="container">
        {% block content %}{% endblock %}
    </main>
</body>
</html>
```

2. **Crie `templates/index.html`:** Para listar todas as receitas.

```
{% extends 'base.html' %}
{% block content %}
    <h1>Todas as Receitas</h1>
    <div class="card-grid">
        {% for receita in receitas %}
```

```
            <div class="card">
                <h2>{{ receita.titulo }}</h2>
                                <p>Por: <a href="{{ url_for('detalhes_chef',
chef_id=receita.chef.id) }}">{{ receita.chef.nome }}</a></p>
                <h4>Ingredientes:</h4>
                <ul>
                    {% for assoc in receita.ingredientes_associados %}
                        <li>
                                    <span>{{ assoc.ingrediente.nome|capitalize
}}</span>
                            <span>{{ assoc.quantidade }}</span>
                        </li>
                    {% endfor %}
                </ul>
            </div>
        {% endfor %}
    </div>
{% endblock %}
```

3. **Crie `templates/criar_receita.html`:** O formulário para novas receitas.

```
{% extends 'base.html' %}
{% block content %}
    <h1>Criar Nova Receita</h1>
    <form method="POST">
        <div class="form-group">
            <label for="titulo">Título</label>
            <input type="text" id="titulo" name="titulo" required>
        </div>
        <div class="form-group">
            <label for="chef_id">Chef</label>
            <select id="chef_id" name="chef_id" required>
                {% for chef in chefs %}
                    <option value="{{ chef.id }}">{{ chef.nome }}</option>
                {% endfor %}
            </select>
        </div>
        <div class="form-group">
            <label for="instrucoes">Instruções</label>
                        <textarea id="instrucoes" name="instrucoes" rows="8"
required></textarea>
        </div>
        <div class="form-group">
            <label for="ingredientes">Ingredientes e Quantidades</label>
                   <input type="text" id="ingredientes" name="ingredientes"
placeholder="Ex: Farinha: 2 xícaras, Ovos: 3 unidades" required>
                 <small>Formato: Ingrediente: Quantidade, Outro Ingrediente:
Outra Qtd</small>
        </div>
        <button type="submit" class="btn">Salvar</button>
    </form>
{% endblock %}
```

4. **Crie `templates/detalhes_chef.html`:** Para a página de perfil do chef.

```
{% extends 'base.html' %}
{% block content %}
    <h1>{{ chef.nome }}</h1>
    {% if chef.perfil %}
        <p><strong>Especialidade:</strong> {{ chef.perfil.especialidade }}</p>
              <p><strong>Experiência:</strong> {{ chef.perfil.anos_experiencia }}
anos</p>
    {% endif %}
    <hr>
    <h2>Receitas Criadas</h2>
    <ul>
    {% for receita in chef.receitas %}
        <li>{{ receita.titulo }}</li>
```

```
            {% else %}
                <li>Nenhuma receita publicada.</li>
            {% endfor %}
            </ul>
        {% endblock %}
```

Passo 4: Implementando a Lógica das Rotas (`app.py`)

Agora vamos conectar nossos modelos e templates, adicionando a lógica no `app.py`.

Adicione o seguinte código ao seu `app.py`:

```
# app.py

# ... (código de configuração inicial) ...

# Importa os modelos DEPOIS de inicializar 'db'
from flask import render_template, request, redirect, url_for
from models import Chef, PerfilChef, Receita, Ingrediente, ReceitaIngrediente

# --- Rotas ---

@app.route('/')
def index():
    receitas = Receita.query.all()
    return render_template('index.html', receitas=receitas)

@app.route('/receita/nova', methods=['GET', 'POST'])
def criar_receita():
    if request.method == 'POST':
        # 1. Pega os dados básicos
        titulo = request.form['titulo']
        instrucoes = request.form['instrucoes']
        chef_id = request.form['chef_id']

        # 2. Cria o objeto Receita
              nova_receita = Receita(titulo=titulo, instrucoes=instrucoes,
chef_id=chef_id)
        db.session.add(nova_receita)

        # 3. Processa a string de ingredientes
        ingredientes_str = request.form['ingredientes']
                        pares_ingredientes = [par.strip() for par in
ingredientes_str.split(',') if par.strip()]

        for par in pares_ingredientes:
            if ':' in par:
                nome, qtd = par.split(':', 1)
                nome_ingrediente = nome.strip().lower()
                quantidade = qtd.strip()

                # Encontra ou cria o ingrediente
                                                        ingrediente =
Ingrediente.query.filter_by(nome=nome_ingrediente).first()
                if not ingrediente:
                    ingrediente = Ingrediente(nome=nome_ingrediente)
                    db.session.add(ingrediente)
```

```
                # Cria a associação com a quantidade
                        associacao = ReceitaIngrediente(receita=nova_receita,
ingrediente=ingrediente, quantidade=quantidade)
                db.session.add(associacao)

        db.session.commit()
        return redirect(url_for('index'))

    # Se for GET, apenas mostra o formulário
    chefs = Chef.query.all()
    return render_template('criar_receita.html', chefs=chefs)

@app.route('/chef/<int:chef_id>')
def detalhes_chef(chef_id):
    chef = Chef.query.get_or_404(chef_id)
    return render_template('detalhes_chef.html', chef=chef)
```

Explicação Vital: A rota `criar_receita` é a mais complexa. Ela processa a string de ingredientes, separa o nome da quantidade, verifica se cada ingrediente já existe no banco (para não criar duplicatas) e, por fim, cria o objeto `ReceitaIngrediente` que faz a ligação correta.

Passo 5: Criando um Comando para Popular o Banco (`app.py`)

Para testar a aplicação, precisamos de dados. Criaremos um comando `flask` personalizado para isso.

Adicione este código ao final do `app.py`:

```
# app.py

# ... (código anterior) ...

# --- Comandos CLI ---

@app.cli.command('init-db')
def init_db_command():
    """Cria as tabelas e popula com dados de exemplo."""
    db.drop_all()
    db.create_all()

    # Criar Chefs e Perfis
    chef1 = Chef(nome='Ana Maria')
     perfil1 = PerfilChef(especialidade='Culinária Brasileira', anos_experiencia=25,
chef=chef1)
    chef2 = Chef(nome='Érick Jacquin')
      perfil2 = PerfilChef(especialidade='Culinária Francesa', anos_experiencia=30,
chef=chef2)

    # Criar Ingredientes
    ingredientes = {
        'tomate': Ingrediente(nome='tomate'), 'cebola': Ingrediente(nome='cebola'),
        'farinha': Ingrediente(nome='farinha'), 'ovo': Ingrediente(nome='ovo'),
        'manteiga': Ingrediente(nome='manteiga')
    }
```

```
    db.session.add_all([chef1, chef2] + list(ingredientes.values()))

    # Criar Receitas
       receita1 = Receita(titulo='Molho de Tomate Clássico', instrucoes='...',
chef=chef1)
    receita2 = Receita(titulo='Bolo Simples', instrucoes='...', chef=chef1)
    receita3 = Receita(titulo='Petit Gâteau', instrucoes='...', chef=chef2)
    db.session.add_all([receita1, receita2, receita3])

    # Criar Associações com Quantidade
    db.session.add_all([
           ReceitaIngrediente(receita=receita1, ingrediente=ingredientes['tomate'],
quantidade='5 unidades'),
           ReceitaIngrediente(receita=receita1, ingrediente=ingredientes['cebola'],
quantidade='1 unidade'),
          ReceitaIngrediente(receita=receita2, ingrediente=ingredientes['farinha'],
quantidade='2 xícaras'),
             ReceitaIngrediente(receita=receita2, ingrediente=ingredientes['ovo'],
quantidade='3 unidades'),
         ReceitaIngrediente(receita=receita3, ingrediente=ingredientes['manteiga'],
quantidade='150g')
    ])

    db.session.commit()
    print('Banco de dados inicializado com sucesso!')
```

Passo 6: Estilização (`style.css`)

Para a aplicação ficar agradável, adicione o CSS.

Crie o arquivo `static/css/style.css`:

```
/* Adicione aqui o CSS fornecido na resposta anterior para
estilizar a página */
body { font-family: sans-serif; background-color: #f4f7f6; }
.container { max-width: 960px; margin: 2rem auto; padding: 0 1rem;
}
nav { background-color: #fff; padding: 1rem 2rem; display: flex;
justify-content: space-between; box-shadow: 0 2px 4px
rgba(0,0,0,0.05); }
.nav-brand { font-weight: 700; font-size: 1.5rem; color: #2a9d8f;
text-decoration: none; }
.nav-link { text-decoration: none; color: #fff; background-color:
#2a9d8f; padding: 0.5rem 1rem; border-radius: 8px; }
.card-grid { display: grid; grid-template-columns:
repeat(auto-fill, minmax(280px, 1fr)); gap: 1.5rem; }
.card { background: #fff; padding: 1.5rem; border-radius: 8px;
box-shadow: 0 4px 8px rgba(0,0,0,0.08); }
.form-group { margin-bottom: 1.5rem; }
.form-group label { display: block; margin-bottom: 0.5rem; }
.form-group input, .form-group textarea, .form-group select {
width: 100%; padding: 0.8rem; border: 1px solid #ccc;
border-radius: 8px; }
.btn { background-color: #2a9d8f; color: #fff; padding: 0.8rem
1.5rem; border: none; border-radius: 8px; cursor: pointer; }
```

Passo 7: Executando a Aplicação

Com todos os arquivos no lugar, vamos rodar o projeto.

1. **Inicialize o Banco de Dados:** No seu terminal (com o ambiente virtual ativo), execute:
   ```
   flask init-db
   ```

   Você verá a mensagem "Banco de dados inicializado com sucesso!".

2. **Execute a Aplicação**:
   ```
   flask run
   ```