Programação de Sistemas para Internet

Prof. Romerito Campos

Plano de Aula

- Objetivo: aplicar o recurso Blueprint do Flask para modularizar aplicações

Conteúdos

- Definição
- Aplicação

Blueprints

Blueprints

- O Conceito de Blueprints é utilizado para dar suporte a padrões e criação de componetes de uma aplicação
- Um objeto Blueprint funciona de forma semelhante a um objeto Flask, mas não representa em si uma aplicação.
- Podemos utilizar blueprints quando:
  - refatorar aplicações grandes em componentes
  - criar extensões flask
  - registrar blueprints sob diferentes prefixos de URL

Blueprints

- O funcionamento básico de um Blueprint consiste em criar operações para este blueprint via funções `view`.
  - Estas funções são aquelas que definimos para uma rota.
- Uma vez que estas operações são definidas no Blueprint podemos vinculá-las as aplicações registrando o Blueprint no `app`.
- Neste material, vamos explorar dois estudos de caso nos quais Blueprints são aplicados.
- Sugestão de leitura: 1, 2 e 3

Blueprint

Estudo de Caso 1

Blueprint

- Neste exemplo, vamos considerar o exemplo que já conhecemos para cadastro de usuários.
- Requisitos:
  - Cadastrar usuário
  - Listar usuários
- Adicionamento, vamos também atender aos seguintes requisitos:
  - Cadastrar livros
  - Listar livros

```
# Estrutura do Projeto
case1/
├── app/
│   ├── __init__.py
│   ├── templates/
│   │   ├── layout.html
│   │   └── auth.html
│   └── app.py
├── books/
├── users/
│   ├── __init__.py
│   └── templates/
│       └── users/
│           ├── index.html
│           └── register.html
├── database/
└── app.py
```

Blueprints

- No slide anterior, há um esboço do projeto: código-fonte.

O projeto está dividido em:

- `app/`: diretório da aplicação, iniciamos o `app = Flask(__name__)`
- `users`: todos os recursos de usuário do exemplo (rotas, modelo e páginas)
- `books`: o mesmo que users.
- O arquivo `app.py` que é o último listado é utilizado para `flask run`

Blueprints

- Para rodar este exemplo basta; `flask run --debug`
  - considere que você já iniciou o banco da aplicação
- É importante afirmar que esta divisão dos diretórios não é meramente com base no uso de pacotes.
- Neste estudo de caso, utilizamos um recurso importado do flask.

```
from flask import Blueprint
```

- A classe `Blueprint` permite modularizar a aplicação.

Blueprints

- Mas o que de fato é um Blueprint? O que podemos fazer com ele?
- Vamos examinar a pasta `users` e seus arquivos.
- A pasta `users` é um pacote python (veja o arquivo `__init__.py`).
- Além disso, nesta pasta teremos o seguinte:
  - `users.py`: módulo python chmado `users`;
  - `templates`: local dos templates de `users`;
  - `models.py`: modelos referente a usuários (apenas um no momento).

Criação de Blueprint

- Esta estrutura representa basicamente todos os recurso de uma aplicação flask. É uma estrutura autocontida.
- Entretanto, ela não é uma aplicação em si.
- O arquivo `users` inicia com as seguintes linhas.

```
from flask import render_template, Blueprint, url_for, request, flash, redirect
from users.models import User

# módulo de usuários
bp = Blueprint('users', __name__, url_prefix='/users', template_folder='pages')
```

- Observe que não importamos a classe Flask porque não vamos iniciar a aplicação (que sempre guardamos numa variável `app`) neste arquivo.
- Importamos alguns recurso do flask (em especial `Blueprint`).
- Em seguida, definimos um blueprint e guardamos em `bp`.
- Neste exemplo, a classe Blueprint recebeu 4 argumentos:
  - `users`: nome do blueprint
  - `__name__`: import_name que é nome usado para importação (nome do arquivo python)
  - `url_prefix`: prefixo a ser adiciona a url das rotas (veremos)
  - `template_folder`: local onde ficam os arquivos HTML

- A questão é: o que fazer com o objeto `bp`? A abaixo esta resposta:

```
@bp.route('/')
def index():
    return render_template('users/index.html', users = User.all())
```

- Definimos uma rota vinculada a este Blueprint. Esta rota usa uma `view` chamada `index`.
- Dentro da rota temos o uso de um modelo chamado `User`

A definição da rota não indica que ela já está disponível para uso. É necessário registrá-la na aplicação.

Importanto o Blueprint

- Após a criação do Blueprint devemos importá-lo no destino.
- O atributo que indica o nome de importação é o `import_name`, que no exemplo é `__name__`. Ou seja, o nome do próprio arquivo (`users.py`)
- Neste exemplo, vamos impotar no arquivo da aplicação conforme abaixo:

```
# trecho de app/__init__.py
from users import users
```

Registrar Blueprint

- O próximo passo é registrar este Blueprint.
- Isso deve ser feito na aplicação. A aplicação foi criada no pacote `app` dentro do arquivo `__init__.py`. Abaixo o trecho que registra o Blueprint.

```
# trecho de app/__init__.py
from flask import Flask, render_template
from users import users
from books import books

app = Flask (__name__, template_folder='templates')
app.register_blueprint(users.bp)
```

- Quando registramos o Blueprint na aplicação, estamos fazendo com que as funcionalidades definidas possam ser usadas na aplicação.
- Neste caso, a funcionalidade do exemplo é uma rota.
- Um detalhe importante é o uso do argumento `url_prefix='/users'` na contrução do Blueprint.
- Isso significa que todas as rotas tem o prefixo `/users`:

```
http://localhost:5000/users/
http://localhost:5000/users/register
```

Recursos do Blueprint

- Uma atenção especial sobre os recursos como páginas e arquivos estáticos (css, icones e etc) deve ser dada.
- De acordo com a documentação do Flask sobre a pasta templates, temos:

Flask vai procurar pelos templates em uma pasta templates. Se a aplicação é um módulo, a pasta estará perto do módulo, se for um pacote está dentro do pacote.

- Neste estudo de caso, a aplicação está dentro de um pacote.
- Logo, a pasta templates está dentro deste pacote:

```
case1/
├── app/
    ├── __init__.py
    ├── templates/
    │   ├── layout.html
    │   └── auth.html
    └── app.py
```

- O que isso tem a ver com Blueprints? No exemplo de uso de Blueprints temos arquivos HTML. Logo, precisamos decidir onde colocá-los.

- Você já observou que colocamos os arquivos HTML do Blueprint `usesr` dentro do pacote `users`.
- Veja a pasta `users\templates\users`. Ela contém dois arquivos HTML que usamos para usuários.
- Veja o Blueprint

```
bp = Blueprint('users', __name__, url_prefix='/users', template_folder='templates')
```

- O argumento `template_folder` indica que vamos salvar os templates do blueprint dentro de uma pasta templates que se localiza no pacote do Blueprint.

- Entretanto, o Flask vai procurar os templates para a função `render_template` nos diretórios registrados para conter templates, que são:
  - templates dentro do pacote da aplicação
  - demais pastas de templates dos Blueprints
- Isso significa que devemos evitar nomes que gerem conflitos.
- Por esta razão a pasta de templates do Blueprint é na verdade:

```
users\templates\users
```

- Toda explicação dada aqui serve para o Blueprint `books`.

Blueprint e Modelos

- Neste exemplo, o pacote `users` contém um moódulo chamado `models`.
- Este módulo contém uma class `User` que foi utilizada pelo Blueprint para realizar operações no banco de dados.
- Nas duas rotas do exemplo, temos a utilização do Modelo.
- Basta um simples import para o modelo Ser utilizado:

```
# trecho de users.py
from users.models import User
```

Blueprints e Rotas

- O uso das rotas é um ponto importante que também precisa ser observado no caso de Blueprints que definem rotas.
- Considere o formulário de cadastro de usuários `users/templates/users/register.html`
  - Quando o usuário clicar no botão enviar, ele faz uma requisição POST
  - A rota que ele usará é a seguinte:

```
<form action="{{url_for('users.register')}}" method="post">
```

- Neste caso, o Blueprint de Usuários tem nome `users`. Há uma rota cuja view tem nome `register`.
- Desta meneira, na função `url_for` indicamos o nome do Blueprint junto com o nome da view referente a rota desejada.
- O mesmo uso da função `url_for` com rotas em Blueprints acontece dentro das rotas:

```
#trecho da rota register em users.py
return redirect(url_for('users.index'))
```

Estudo de Caso 2

Blueprint e MVC

Blueprint e MVC

- O segundo estudo de caso explora o mesmo problema do estudo de caso 1. Entretanto, a aplicação será estruturada como uma aplicação MVC padrão.
- O código-fonte pode ser encontrado neste link
- No próximo slide, temos a estrutura de diretórios deste exemplo.

```
# Estrutura da aplicação
case2/
├── controllers/
│   ├── __init__.py
│   ├── books.py
│   └── users.py
├── models/
│   ├── __init__.py
│   ├── book.py
│   └── user.py
├── database/
├── templates/
│   ├── users/
│   │   ├── index.html
│   │   └── register.html
│   └── books/
│       ├── index.html
│       └── register.html
└── app.py
```

Blueprint e MVC

- A aplicação possui as pastas:
  - `controllers`: armazena os controladores da aplicação;
  - `models`: armazena os modelos da aplicação
  - `templates`: guarda os templates da aplicação separados por contexto
  - `database`: arquivos para criação do banco de dados sqlite
- O arquivo `app.py` é a aplicação e é utilizado pelo flask para executar o projeto.

Controladores

- A pasta controllers é um pacote.
- O arquivo **init**.py possui o conteúdo abaixo:

```
# controllers/__init__.py
all = [
    'books',
    'usrs'
]
```

- A declaração `all` indica todos os pacotes que serão importandos quando fizermos:
  `from controllers import *`
- Pode-se também importar individualmente:
  `from controllers import users`

Controladores

- O código dos Controladores terá certa semelhança com o estudo de caso 1
- Entretanto, não indicamos o `template_folder`, já que haverá apenas um local para guardar os templates

```
from flask import render_template, Blueprint, url_for, request, flash, redirect
from models.user import User
# módulo de usuários
bp = Blueprint('users', __name__, url_prefix='/users')
```

- A importação do modelo `User` vai levar em consideração o pacote dos modelos
- O pacote dos modelos está definido na pasta `models` e contém todos os arquivos dos modelos do exemplo.
- A definição das rotas continua usando a variável Blueprint `bp`

```
@bp.route('/')
def index():
    return render_template('users/index.html', users = User.all())
```

Modelos

- A camada de modelos consiste no conjunto de modelos do projeto.
- Neste exemplo, há dois modelos: `User` e `Book`.
- A pasta `models` é um pacote que contém a definição dos modelos
- Esta pasta não contém definição de Blueprints. Mas é possível pensar na camada de modelos como um Blueprint que contivesse as definições dos modelos.
- Não há alterações no código-fonte das classes que representam os modelos.

Templates

- Neste exemplo, os modelos voltam a ficar no local tradicional das aplicações Flask.
- Uma pasta na raíz do projeto com o nome `templates` guarda subpastas com nomes para cada um dos Recursos que foram definidos: `books` e `users`.
- Note que a pasta template está no mesmo nível de pasta que o arquivo da aplicação `app.py`, conforme é especifado pelo documentação Flask.

Aplicação

- O arquivo da aplicação é o `app.py`. O conteúdo dele segue abaixo:

```
from flask import Flask
from controllers import users, books
app = Flask(__name__)
app.register_blueprint(users.bp)
app.register_blueprint(books.bp)
```

- Foi necessário importar apenas a classe `Flask` e os Blueprints dos controladores.

Conclusões

- O recurso Blueprint permite que aplicações sejam organizadas para melhor atender ao projeto
- Nos dois estudos de caso, o mesmo código-fonte base foi utilizado para arquitetura diferentes.
- Esta flexibilidade é uma das marcas do Flask tendo em vista que não há amarras quando a organização do código-fonte.
- O programador pode adotar os padrões que achar necessário para implementar a aplicação.