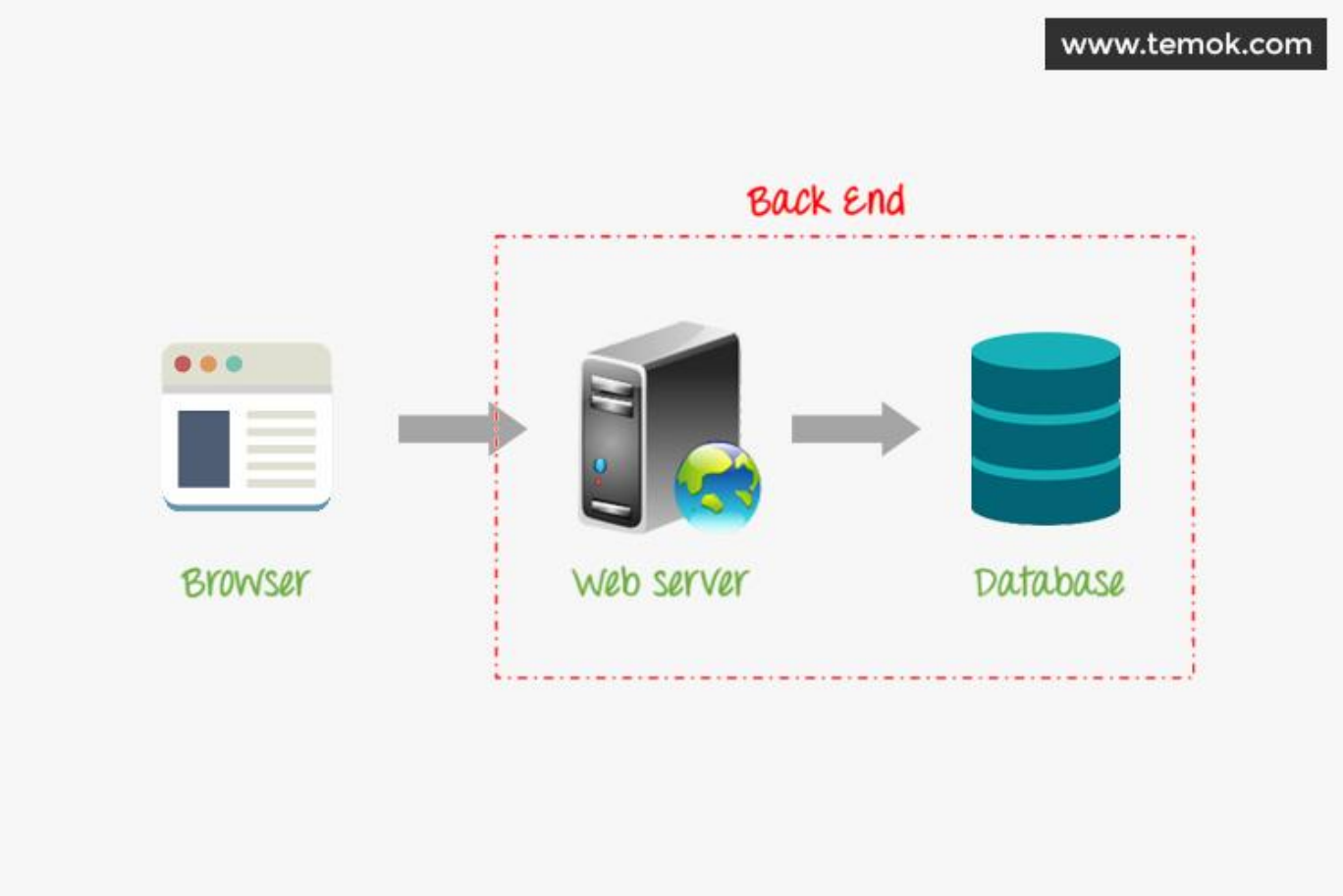
Aula 9

Programação Back-end | Introdução ao Flask

Programação IV

Prof. Sandino Jardim | CC-UFMT-CUA

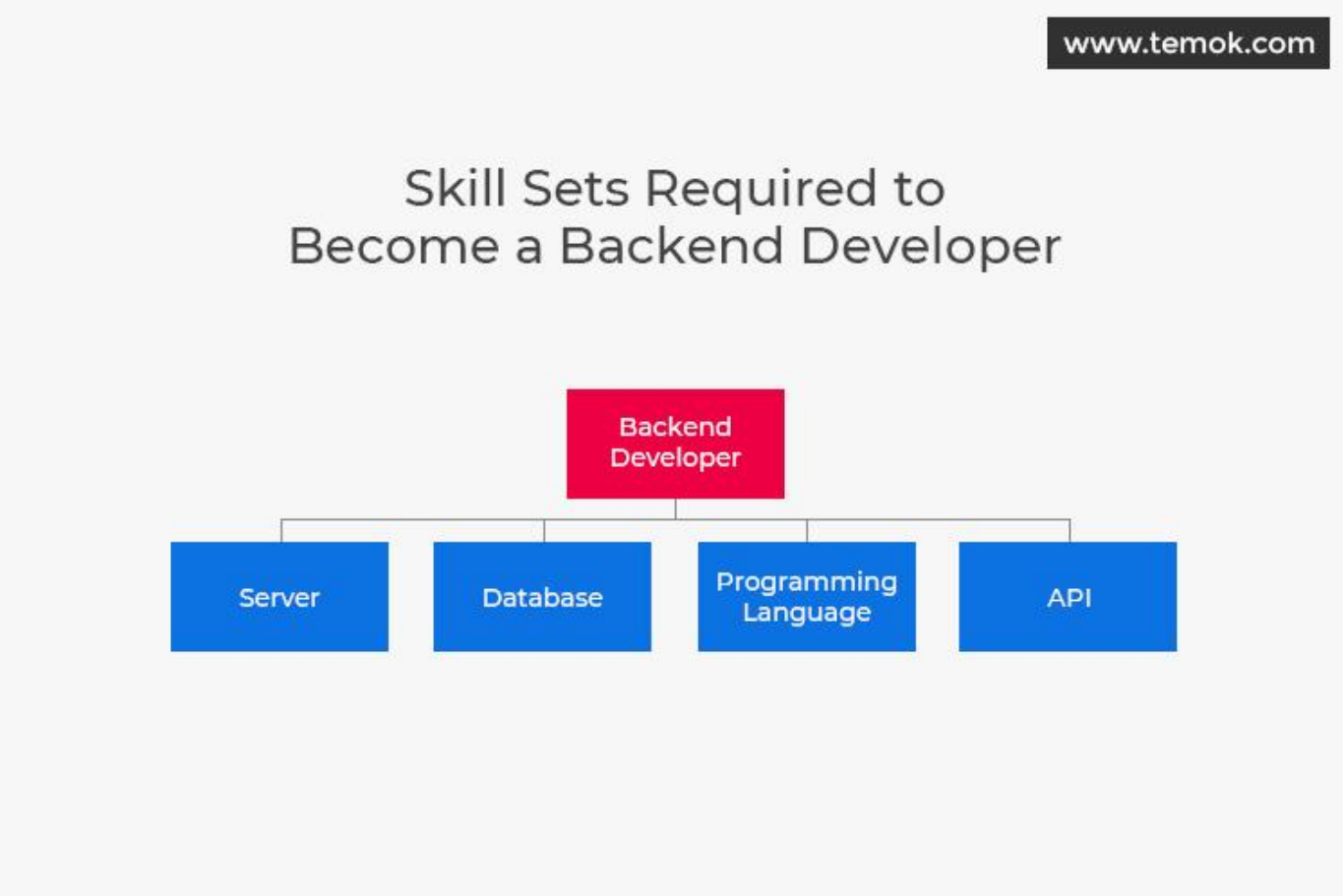
Desenvolvimento back-end



Desenvolvedores back-end

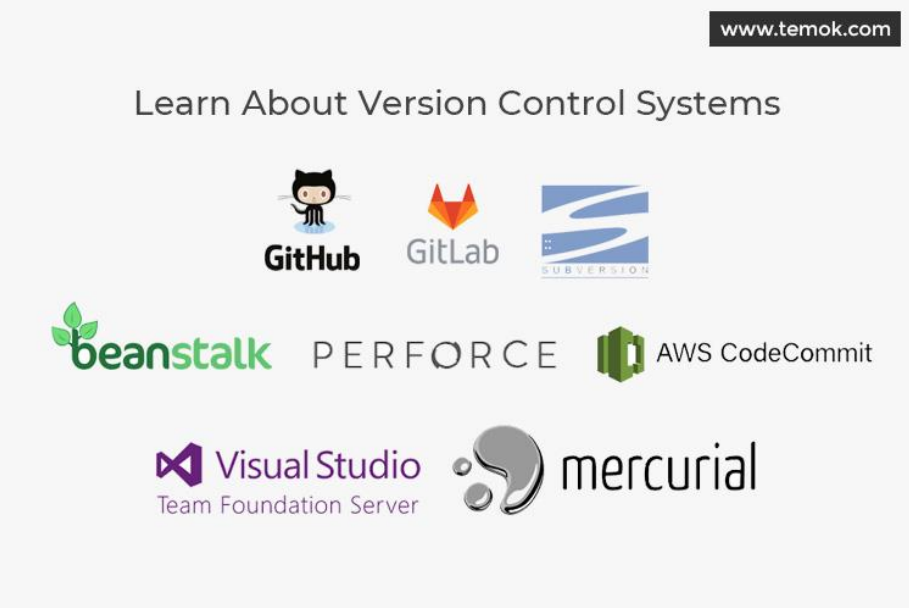
- Responsáveis não só pela programação, mas pela absorção da lógica de negócios envolvida
  - tarefas relacionadas com os processos de um negócio, tais como vendas, controle de inventário, contabilidade, etc.
- Coordena a comunicação entre front-end e recursos back-end

Conjunto de habilidades recomendadas



Adicionalmente

- Gerenciamento de hospedagem
  - Escolhas podem impactar em indicadores de desempenho
- Sistemas de controle de versão



Frameworks

- Tecnicamente, é possível codificar toda estrutura back-end de uma página camada por camada
- Frameworks fornecem um modelo padrão para o desenvolvimento de aplicações web dinâmicas, provendo:
  - Bibliotecas para acesso a bancos de dados
  - Templates
  - Gerenciamento de sessão
  - Reutilização de código

Alguns frameworks disponíveis

Framework	Linguagem de programação	Casos de Uso
Django	Python	Instagram Pinterest Coursera
Laravel	PHP	Deltanet Travel Neighborhood Lender MyRank
Ruby on Rails	Ruby	ZendDesk Shopify GitHub
ExpressJS	NodeJS	MySpace GeekList Storify
CakePHP	PHP	Mapme Educationunlimited Followmy Tv
Flask	Python	Red Hat Rackspace Reddit
Asp .NET	C#	Microsoft Godaddy Ancestry
Spring Boot	Java	Trivago Via Varejo Intuit
Koa	NodeJS	–
Phoenix	Elixir	Financial Times Fox 10 ABC15

Flask

- Microestrutura baseada em Python que não requer bibliotecas e ferramentas específicas
- Flexível, permite a utilização de soluções "não-oficiais"
- Oferece suporte a extensões que podem adicionar recursos como se fossem implementadas pelo próprio

Instalação

- Instale o python3, pip e *virtual environment*

```
sudo apt install python3

sudo apt install python3-venv

sudo apt install python3-pip
```

- Crie e acesse a pasta “flasky”

```
mkdir flasky
cd flasky
```

- Crie um ambiente virtual

```
python3 -m venv flask
```

- Acesse o ambiente virtual

```
source flask/bin/activate
```

- Instale o Flask

```
(flask) $ pip install flask
```

Inicialização

- Toda aplicação Flask deve criar uma instância de aplicação
- O servidor web repassa todas as requisições dos clientes para este objeto
- A instância de aplicação é um objeto da classe Flask, geralmente criado da seguinte forma:

```
from flask import Flask
app = Flask(__name__)
```

Routes e Views

- A instância de aplicação precisa saber qual código precisa executar para cada URL requisitada
- *Mantém um mapeamento das URLs (routes*) e as funções (views)

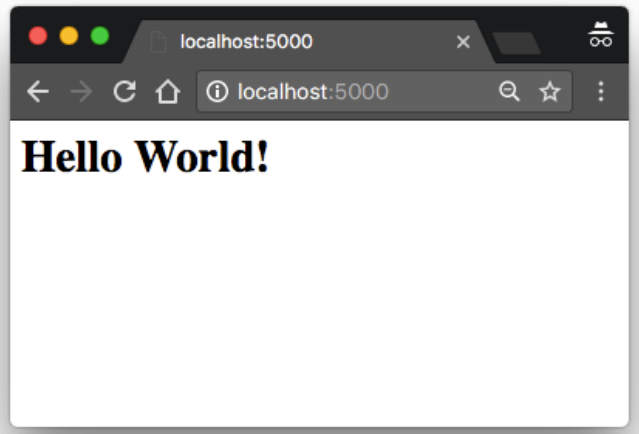
```
@app.route('/')
def index():
        return '<h1>Hello World!</h1>'

@app.route('/user/<name>')
def user(name):
    return '<h1>Hello, {}!</h1>'.format(name)
```

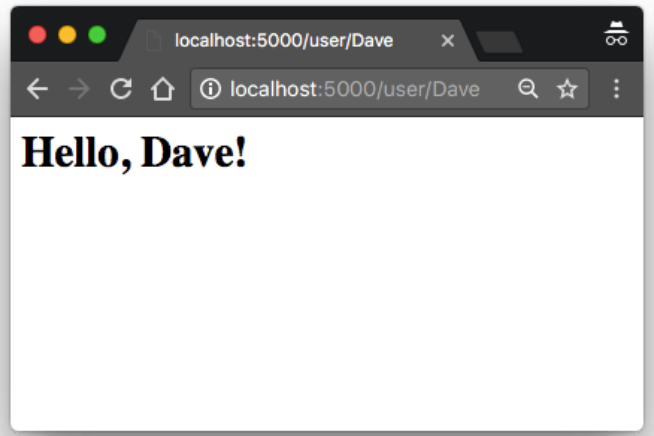
(hello.py)

Configurando servidor Web

- Definir instância de aplicação e executar servidor

```
(flask) $ export FLASK_APP=hello.py
(flask) $ flask run
        * Serving Flask app "hello"
        * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```





Templates

- Arquivos que contém textos para respostas, permitindo variáveis para as partes dinâmicas que dependerão da requisição
- Flask usa a engine Jinja2

templates/index.html

```
<h1>Hello World!</h1>
```

templates/user.html

```
<h1>Hello, {{ name }}!</h1>
```

```
from flask import Flask, render_template

# ...

@app.route('/')
def index():
        return render_template('index.html')

@app.route('/user/<name>')
def user(name):
        return render_template('user.html', name=name)
```

Templates - variáveis e funções

- Jinja2 oferece funções e reconhece variáveis de qualquer tipo, mesmo tipos complexos como listas, dicionários e objetos;
- Exemplos:

```
<p>A value from a dictionary: {{ mydict['key'] }}.</p>
<p>A value from a list: {{ mylist[3] }}.</p>
<p>A value from a list, with a variable index: {{ mylist[myintvar] }}.</p>
<p>A value from an object's method: {{ myobj.somemethod() }}.</p>

<a href="{{ url_for('user', name='SeuNome') }}">Página do Usuário</a>
<a href="{{ url_for('about') }}">Sobre</a>
```


Templates – estruturas de controle

- Jinja2 também oferece diferentes estruturas de controle do fluxo do template:

```
{% if user %}
        <h1>Hello, {{ user }}!</h1>
{% else %}
        <h1>Hello, Stranger!</h1>
{% endif %}
```

```
<ul>
{% for comment in comments %}
        <li>{{ comment }}</li>
{% endfor %}
</ul>
```

Templates - herança

- Permite a criação de templates base que podem ser sobrescritos por derivados

```
<html>
<head>
{% block head %}
<title>{% block title %}{% endblock %} - My Application</title>
{% endblock %}
</head>
<body>
{% block body %}
{% endblock %}
</body>
</html>
```

(base.html)

```
{% extends "base.html" %}
{% block title %}Index{% endblock %}
{% block head %}
          {{ super() }}
          <style>
          </style>
{% endblock %}
{% block body %}
          <h1>Hello, World!</h1>
{% endblock %}
```

(index.html)

Flask – Integração com Bootstrap

- Instalando a extensão

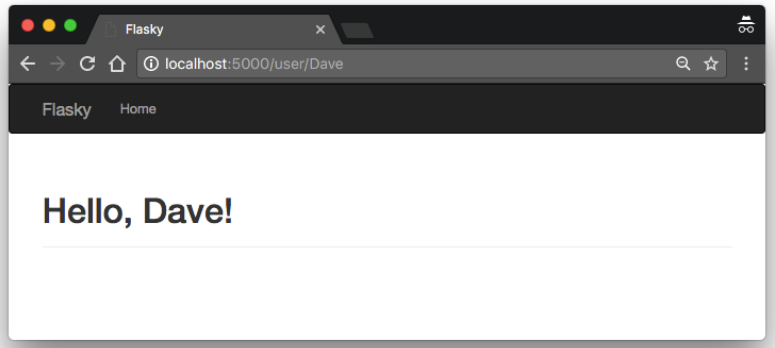
```
(flask) $ pip install flask-bootstrap
```

- Inicialização

```
from flask_bootstrap import Bootstrap
# ...
bootstrap = Bootstrap(app)
```

Flask – Exemplo Bootstrap

```
{% extends "bootstrap/base.html" %}
{% block title %}Flasky{% endblock %}
{% block navbar %}
            <div class="navbar navbar-inverse" role="navigation">
                        <div class="container">
                                    <div class="navbar-header">
                                                <button type="button" class="navbar-toggle"
                                                data-toggle="collapse" data-target=".navbar-collapse">
                                                 <span class="sr-only">Toggle navigation</span>
                                                 <span class="icon-bar"></span>
                                                 <span class="icon-bar"></span>
                                                 <span class="icon-bar"></span>
                                                </button>
                                                <a class="navbar-brand" href="/">Flasky</a>
                                    </div>
                                    <div class="navbar-collapse collapse">
                                                <ul class="nav navbar-nav">
                                                            <li><a href="/">Home</a></li>
                                                </ul>
                                    </div>
                        </div>
            </div>
{% endblock %}
{% block content %}
            <div class="container">
                        <div class="page-header">
                                    <h1>Hello, {{ name }}!</h1>
                        </div>
            </div>
{% endblock %}
```



Flask – Definir como modelo base

```
{% extends "bootstrap/base.html" %}
{% block title %}Flasky{% endblock %}
{% block navbar %}
            <div class="navbar navbar-inverse" role="navigation">
                        <div class="container">
                                    <div class="navbar-header">
                                                <button type="button" class="navbar-toggle"
                                                data-toggle="collapse" data-target=".navbar-collapse">
                                                 <span class="sr-only">Toggle navigation</span>
                                                 <span class="icon-bar"></span>
                                                 <span class="icon-bar"></span>
                                                 <span class="icon-bar"></span>
                                                </button>
                                                <a class="navbar-brand" href="/">Flasky</a>
                                    </div>
                                    <div class="navbar-collapse collapse">
                                                <ul class="nav navbar-nav">
                                                            <li><a href="/">Home</a></li>
                                                </ul>
                                    </div>
                        </div>
            </div>
{% endblock %}
{% block content %}
            <div class="container">
                        {% block page_content %}{% endblock %}
            </div>
{% endblock %}
```


templates/base.html

Páginas de Erro

- Além de definir modelos padrão para páginas de erro, permite definir rotas baseado nos erros ocorridos

```
{% extends "base.html" %}
{% block title %}Flasky - Page Not Found{% endblock %}
{% block page_content %}
        <div class="page-header">
                <h1>Not Found</h1>
        </div>
{% endblock %}
```

(templates/404.html)

```
@app.errorhandler(404)
def page_not_found(e):
        return render_template('404.html'), 404

@app.errorhandler(500)
def internal_server_error(e):
        return render_template('500.html'), 500
```

(hello.py)