

Programação Web (Avançado)

Claudio Moisés Valiense de Andrade



**Formação Inicial e
Continuada**

+ IFMG



Claudio Moisés Valiense de Andrade

Programação Web (Avançado)
1ª Edição

Belo Horizonte

Instituto Federal de Minas Gerais

2022

Todos os direitos autorais reservados. Nenhuma parte desta publicação poderá ser reproduzida ou transmitida de qualquer modo ou por qualquer outro meio, eletrônico ou mecânico. Incluindo fotocópia, gravação ou qualquer outro tipo de sistema de armazenamento e transmissão de informação, sem prévia autorização por escrito do Instituto Federal de Minas Gerais.

Pró-reitor de Extensão	Carlos Bernardes Rosa Júnior
Diretor de Projetos de Extensão	Niltom Vieira Junior
Coordenação do curso	Claudio Moisés Valiense de Andrade
Arte gráfica	Ângela Bacon
Diagramação	Eduardo dos Santos Oliveira

FICHA CATALOGRÁFICA

Dados Internacionais de Catalogação na Publicação (CIP)

A554p Andrade, Claudio Moisés Valiense de
Programação Web: avançado [recurso eletrônico] /
Cláudio Moisés Valiense de Andrade. – Belo Horizonte :
Instituto Federal de Minas Gerais, 2022.
65 p. : il. color.
Material didático para Formação Inicial e Continuada.
ISBN 978-65-5876-037-5

1. Bootstrap. 2. Frontend. 3. SQL. 4. Backend. I.
Título.

CDD 005.133
CDU 004.438

Catalogação: Rejane Valéria Santos - Bibliotecária - CRB-6/2907

Índice para catálogo sistemático:

1. Programação 005.133

2022

Direitos exclusivos cedidos à
Instituto Federal de Minas Gerais
Avenida Mário Werneck, 2590,
CEP: 30575-180, Buritis, Belo Horizonte – MG,
Telefone: (31) 2513-5157

Sobre o material

Este curso é autoexplicativo e não possui tutoria. O material didático, incluindo suas videoaulas, foi projetado para que você consiga evoluir de forma autônoma e suficiente.

Caso opte por imprimir este *e-book*, você não perderá a possibilidade de acessar os materiais multimídia e complementares. Os *links* podem ser acessados usando o seu celular, por meio do glossário de Códigos QR disponível no fim deste livro.

Embora o material passe por revisão, somos gratos em receber suas sugestões para possíveis correções (erros ortográficos, conceituais, *links* inativos etc.). A sua participação é muito importante para a nossa constante melhoria. Acesse, a qualquer momento, o Formulário “Sugestões para Correção do Material Didático” clicando nesse [link](#) ou acessando o QR Code a seguir:



Formulário de
Sugestões

Para saber mais sobre a Plataforma +IFMG acesse

www.ifmg.edu.br



Palavra do autor

Caro aluno seja bem-vindo ao curso de Formação Continuada “Programação Web (Avançado)”.

Dividido em quatro módulos, este curso irá discutir os conceitos avançados na criação de sites, fazendo com o que o aluno entenda o funcionamento por trás das requisições do usuário, como essas requisições são tratadas no lado do servidor e como integrar um banco de dados que permita salvar e consultar as informações.

O curso de programação web apresenta-se como uma alternativa viável para formação de profissionais que ao final do curso terão amplas condições de atender usuários/clientes residenciais ou empresariais, além da possibilidade de fazerem parte de uma equipe de colaboradores de uma empresa.

Bons estudos!

Claudio Moisés Valiense de Andrade



Apresentação do curso

Este curso está dividido em quatro semanas, cujos objetivos de cada uma são apresentados, sucintamente, a seguir.

SEMANA 1	Nesta semana, você compreenderá um funcionamento do frontend em sistema web, utilizando o framework Bootstrap, um dos frameworks mais populares em programação web, que irá permitir aplicar novas funcionalidades, além de possibilitar a criação de páginas responsivas.
SEMANA 2	Nesta semana, você compreenderá o funcionamento do backend utilizando o framework Flask que foi construído com a linguagem de programação python.
SEMANA 3	Nesta semana, você compreenderá o funcionamento do backend utilizando o framework NodeJS, o framework mais popular da linguagem de programação Javascript.
SEMANA 4	Nesta semana, vamos compreender como integrar um banco de dados MySQL utilizando o framework NodeJS.

Carga horária: 40 horas.

Estudo proposto: 2h por dia em cinco dias por semana (10 horas semanais).



Apresentação dos Ícones

Os ícones são elementos gráficos para facilitar os estudos, fique atento quando eles aparecem no texto. Veja aqui o seu significado:



Atenção: indica pontos de maior importância no texto.



Dica do professor: novas informações ou curiosidades relacionadas ao tema em estudo.



Atividade: sugestão de tarefas e atividades para o desenvolvimento da aprendizagem.



Mídia digital: sugestão de recursos audiovisuais para enriquecer a aprendizagem.



Sumário

Semana 1 – Frontend em sistemas web	15
1.1. Importando Bootstrap	15
1.2. Componentes do Bootstrap	16
Semana 2 – Backend utilizando python	23
2.1 Instalação Flask	23
2.2 Exemplo inicial com Flask	23
2.3 Tratando requisições com Flask.....	24
Semana 3 – Backend utilizando javascript.....	32
3.1 Exemplo inicial	32
3.2 Tratando requisições com NodeJS	33
Semana 4 – Integração com banco de dados.....	40
4.1. Conectando banco de dados.....	40
4.2. Executando SQL no banco de dados.....	41
Referências.....	49
Currículo do autor	52
Glossário de códigos QR (<i>Quick Response</i>).....	53



Objetivos

Nesta semana, você compreenderá um funcionamento do frontend em sistema web, utilizando o framework Bootstrap, um dos frameworks mais populares em programação web, que irá permitir aplicar novas funcionalidades, além de possibilitar a criação de páginas responsivas



Mídia digital: Antes de iniciar os estudos, vá até a sala virtual e assista ao vídeo “Frontend em sistemas web”.

1.1. Importando Bootstrap

Nesta semana explicaremos sobre o frontend em sistemas web, o frontend é a parte visual que o usuário consegue visualizar, no caso da web, através de um navegador. Começaremos com front-end utilizando o framework bootstrap.

O Bootstrap é um framework front-end gratuito para desenvolvimento web. Entre as suas funcionalidades, ele permite desenvolvimento de sites responsivos que se ajustam automaticamente para que tenham uma boa aparência em diferentes tamanhos de telas, com a de telefones pequenos a grandes desktops (W3, 2022).

Para a utilização do bootstrap é preciso que seja baixado os seus respectivos arquivos css e javascript, como apresenta a Figura 1. A linha 5 importa o css do bootstrap e a linha 6 o código javascript do bootstrap, ambas as linhas tem a informação da versão do bootstrap, que neste caso, é a versão 5.1.3, onde tais links podem ser obtidos no site oficial do framework. Uma classe importada é a classe ‘container’ (linha 10), que define uma caixa responsiva que ocupará toda a linha, nesta caixa deverá ser inserido os demais elementos html, além disso, esta caixa define uma distância fixa da margem esquerda e direita. A Figura 2 apresenta a renderização do código da Figura 1.

```

1  <!DOCTYPE html>
2  <html lang="pt-BR">
3  . . <head>
4  . . . . <title>IFMG</title>
5  . . . . <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">
6  . . . . <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>
7  . . . . <meta name="viewport" content="width=device-width, initial-scale=1"> . . . .
8  . . </head>
9  . . <body>
10 . . . . <div class="container">
11 . . . . . . <h1>IFMG</h1> . . . .
12 . . . . </div> . . . .
13 . . </body>
14 </html>

```

Figura 1: Exemplificação do uso de Bootstrap.

Fonte: O próprio autor.

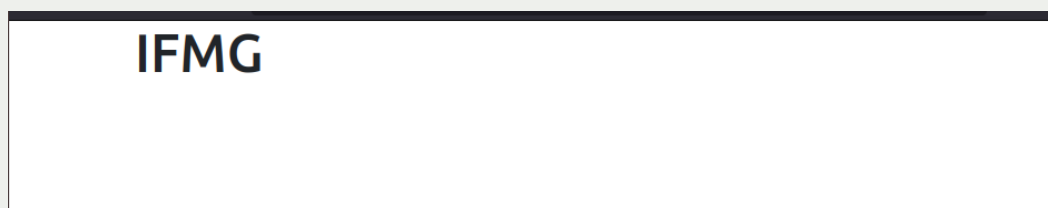


Figura 2: Renderização do código da Figura 1

Fonte: O próprio autor.

1.2. Componentes do Bootstrap

No bootstrap existe uma divisão de páginas em linhas e colunas, para isso utilizamos a class 'row' que define as linhas e a classe 'col' que define as colunas. Uma linha pode ser dividida em no máximo doze colunas, o exemplo da Figura 3 estamos criando uma página com uma linha (linha 12) e três colunas (linhas 13-15), por padrão, o bootstrap divide o espaço entre as colunas de forma igual, que neste caso, 33.3% para cada coluna. Para facilitar a visualização, foi adicionado uma borda nas colunas (comando border). A Figura 4 é a renderização do código.


```

1  <!DOCTYPE html>
2  <html lang="pt-BR">
3  <head>
4  <title>IFMG</title>
5  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">
6  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>
7  <meta name="viewport" content="width=device-width, initial-scale=1">
8  </head>
9  <body>
10 <div style="border:2px solid black" class="container">
11 <h1>IFMG</h1>
12 <div class="row">
13 <div style="border:2px solid black" class="col">texto 1</div>
14 <div style="border:2px solid black" class="col">texto 2</div>
15 <div class="col">texto 3</div>
16 </div>
17 </div>
18 </body>
19 </html>

```

Figura 3: Organização do conteúdo em bootstrap.

Fonte: O próprio autor.

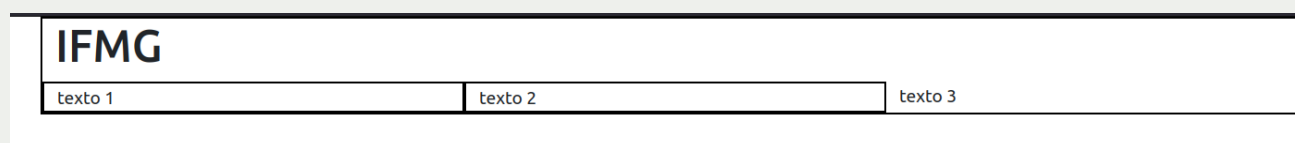


Figura 4: Renderização da organização do conteúdo em bootstrap.

Fonte: O próprio autor.

Em bootstrap, é possível definir que uma coluna ocupe mais espaço que outra, no caso da Figura 5, estamos definindo que uma coluna irá utilizar cinco posições dos doze espaço do bootstrap (linha 13), e a outra coluna irá utilizar os outros sete espaços (linha 14).

```

1  <!DOCTYPE html>
2  <html lang="pt-BR">
3  <head>
4  <title>IFMG</title>
5  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">
6  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>
7  <meta name="viewport" content="width=device-width, initial-scale=1">
8  </head>
9  <body>
10 <div style="border:2px solid black" class="container">
11 <h1>IFMG</h1>
12 <div class="row">
13 <div style="border:2px solid black" class="col-5">texto 1</div>
14 <div style="border:2px solid black" class="col-7">texto 2</div>
15 </div>
16 </div>
17 </body>
18 </html>

```

Figura 5: Exemplificando o uso de mais espaço por uma coluna.

Fonte: O próprio autor.

IFMG	
texto 1	texto 2

Figura 6: Renderização código da Figura 5.

Fonte: Próprio autor

No html temos o comando h1 onde podemos definir o cabeçalho. No Bootstrap, temos a classe h1 (linha 13), para uma situação que queremos que o texto fique no formato do comando h1, mas não podemos utilizar o comando h1, desta forma podemos utilizar a classe dentro de outro comando, que neste caso, é o comando '<p>' (linha 13). Além disso, podemos dar destaque ao cabeçalho no Bootstrap utilizando a classe 'display', que alterar o tamanho e o tipo de fonte de cabeçalho, de forma similar ao comando 'h', temos vários tamanho, display-1, display-3, utilizados nas linhas 14 e 15.

```

1  <!DOCTYPE html>
2  <html lang="pt-BR">
3  <head>
4  <title>IFMG</title>
5  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">
6  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>
7  <meta name="viewport" content="width=device-width, initial-scale=1">
8  </head>
9  <body>
10 <div class="container">
11 <h1>IFMG</h1>
12 <p>IFMG</p>
13 <p class="h1">IFMG</p>
14 <h1 class="display-1">IFMG</h1>
15 <h1 class="display-3">IFMG</h1>
16 </div>
17 </body>
18 </html>

```

Figura 7: Utilizando cabeçalhos em Bootstrap.

Fonte: O próprio autor.



Figura 8. Renderização utilizando cabeçalho em Bootstrap.

Fonte: O próprio autor.

A utilização de tabelas no bootstrap pode ser definida com a classe 'table', onde podemos visualizar uma alteração das margens e distância entre os elementos. Neste caso, basta apenas adicionar a classe table no comando table (linha 11). É possível adicionar estilização na tabela, por exemplo, ao adicionar 'table-striped' ao lado de table, é adicionado um fundo cinza nas linhas ímpares das tabelas. Outro tipo de estilização, é utilizando o '.table-hover' ao lado do 'table', como resultado, ao passar o mouse pela linha, é alterado a cor de fundo da linha.

```

1  <!DOCTYPE html>
2  <html lang="pt-BR">
3    <head>
4      <title>IFMG</title>
5      <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">
6      <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>
7      <meta name="viewport" content="width=device-width, initial-scale=1">
8    </head>
9    <body>
10     <div class="container">
11       <table class="table table-striped">
12         <tr>
13           <th>Nome</th>
14           <th>Idade</th>
15           <th>Nota</th>
16         </tr>
17         <tr>
18           <td>Claudio</td>
19           <td>30</td>
20           <td>10</td>
21         </tr>
22         <tr>
23           <td>Maria</td>
24           <td>20</td>
25           <td>7</td>
26         </tr>
27       </table>
28     </div>
29   </body>
30 </html>

```

Figura 9: Utilizando tabela com bootstrap.

Fonte: O próprio autor.

Nome	Idade	Nota
Claudio	30	10
Maria	20	7

Figura 10: Renderização de tabela no bootstrap.

Fonte: O próprio autor.

Na manipulação de imagens com bootstrap, podemos definir que uma imagem tenha o canto arredondado, com a classe 'rounded', definido na linha 11 da Figura 11 além disso, podemos fazer com que a imagem seja circular, com a classe 'rounded-circle', definido na linha 12 ou podemos definir que a imagem seja responsiva, que, independentemente do tamanho da tela do dispositivo, mantenha a mesma proporção de largura e altura, com a classe 'img-fluid', definido na linha 13.

```

1 <!DOCTYPE html>
2 <html lang="pt-BR">
3   <head>
4     <title>IFMG</title>
5     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">
6     <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>
7     <meta name="viewport" content="width=device-width, initial-scale=1">
8   </head>
9   <body>
10    <div class="container">
11      
12      
13      
14    </div>
15  </body>
16 </html>

```

Figura 11: Utilizando imagens com bootstrap.

Fonte: O próprio autor.



Figura 12: Renderização de imagens com bootstrap.

Fonte: O próprio autor.

Em alguns sistemas web, as vezes solicitamos que o usuário aguarde para concluir uma solicitação, por exemplo, ao realizar upload de um arquivo, no bootstrap é possível exibir um botão que fica girando através da classe 'spinner' (linha 11), a Figura 13, apresenta um botão de carregamento.

```

1 <!DOCTYPE html>
2 <html lang="pt-BR">
3   <head>
4     <title>IFMG</title>
5     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">
6     <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>
7     <meta name="viewport" content="width=device-width, initial-scale=1">
8   </head>
9   <body>
10    <button class="btn btn-primary" disabled>
11      <span class="spinner-border spinner-border-sm"></span>
12      Carregando..
13    </button>
14  </body>
15 </html>

```

Figura 13: Botão giratório com bootstrap.

Fonte: O próprio autor.

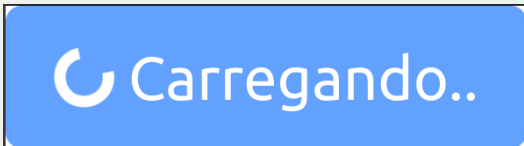


Figura 14: Renderização do botão giratório.
Fonte: O próprio autor.

Para evitar a exibição de uma grande quantidade de conteúdo de uma vez, podemos exibir parte dele, à medida que o usuário escolha o que deseja visualizar, no bootstrap é possível criar botões que permitem que parte do conteúdo só seja visível depois do clique, é possível utilizar a classe 'collapse' para conseguir este efeito. A Figura 16 apresenta um botão com o texto 'IFMG', e ao clicar neste botão aparece o texto, 'O IFMG surgiu em...'.

```

1  <!DOCTYPE html>
2  <html lang="pt-BR">
3    <head>
4      <title>IFMG</title>
5      <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">
6      <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>
7      <meta name="viewport" content="width=device-width, initial-scale=1">
8    </head>
9    <body>
10     <div class="container">
11       <button data-bs-toggle="collapse" data-bs-target="#demo">IFMG</button>
12       <div id="demo" class="collapse">
13         O IFMG surgiu em ...
14       </div>
15     </div>
16   </body>
17 </html>

```

Figura 15: Botão para alterar a visibilidade do conteúdo.
Fonte: O próprio autor.

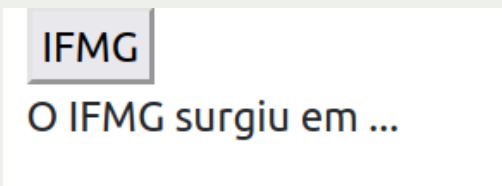


Figura 16: Renderização do botão de alterar a visibilidade do conteúdo.
Fonte: O próprio autor.

Na maior parte dos sistemas web, existe um menu onde o usuário é possível realizar uma seleção do conteúdo que esteja interessado, no bootstrap é possível criar um menu através do nav-tabs (linha 11), onde cada item do menu tem a classe 'nav-item' (linha 12), e cada item é direcionado para uma parte do conteúdo, através do href. Na parte de definição do conteúdo, temos a classe 'tab-content' (linha 19), onde definimos o conteúdo de cada item do menu, com a identificação que colocamos no 'href' no link do menu. Neste momento definimos o efeito do menu, que no caso é um efeito de desaparecer um conteúdo e aparecer outro, neste caso, o efeito 'fade' (linha 21).

```

1 <!DOCTYPE html>
2 <html lang="pt-BR">
3   <head>
4     <title>IFMG</title>
5     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">
6     <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>
7     <meta name="viewport" content="width=device-width, initial-scale=1">
8   </head>
9   <body>
10    <div class="container">
11      <ul class="nav nav-tabs">
12        <li class="nav-item">
13          <a class="nav-link active" data-bs-toggle="tab" href="#home">Início</a>
14        </li>
15        <li class="nav-item">
16          <a class="nav-link" data-bs-toggle="tab" href="#historia">História</a>
17        </li>
18      </ul>
19      <div class="tab-content">
20        <div class="tab-pane container active" id="home">Olá</div>
21        <div class="tab-pane container fade" id="historia">O IFMG surgiu em ...</div>
22      </div>
23    </div>
24  </body>
25 </html>

```

Figura 17: Menu em bootstrap.

Fonte: O próprio autor.

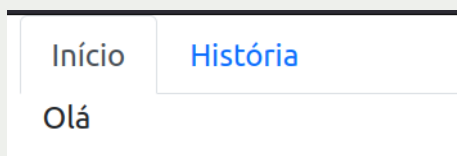


Figura 18: Renderização do menu em bootstrap.

Fonte: O próprio autor.

Espero com este curso que você consiga criar páginas web, conseguir entender códigos e atualizar páginas existentes. A partir deste curso, desejo que você consiga desenvolver seus projetos pessoais e consiga buscar novas funcionalidades para aprimorar seus conhecimentos, incentivo a buscar outros cursos na plataforma +IFMG, como o curso programação web (avançado), dentre as funcionalidade, busco ensinar como tratar requisições do usuário no lado do servidor ou como realizar a integração com banco de dados.



Atividade: Para concluir está semana de estudos, crie uma página inicial com os conhecimentos dessa semana. Em seguida, vá até a sala virtual e participe do fórum “Frontend”, criando um tópico e anexando duas imagens, uma da sua página renderizada pelo navegador e outra do código. Você também pode responder ou comentar o tópico de algum colega.

Nos encontramos na próxima semana.

Bons estudos!

Semana 2 – Backend utilizando python

Objetivos

Nesta semana, você compreenderá o funcionamento do backend utilizando o framework Flask que foi construído com a linguagem de programação python.



Mídia digital: Antes de iniciar os estudos, vá até a sala virtual e assista ao vídeo “Back-end Python”.

2.1 Instalação Flask

Para tratar as requisições que vem do frontend, como no caso de um formulário de cadastro de usuário, precisamos configurar um serviço para tratar essas requisições. Uma opção de servidor backend é utilizando a linguagem de programação python. Após realizar a instalação do python conforme o site oficial e instrução no vídeo da semana, devemos prosseguir para a instalação do framework Flask, com o comando ‘pip install flask’ em um terminal. Depois da instalação do Flask, é possível verificar se o Flask está instalado com o comando ‘pip list’ que exibe todos os pacotes instalados, dentre eles, deve constar o nome do Flask, de forma similar a Figura 19.

```
PS C:\Users\usuario> pip list
Package      Version
-----
click        8.1.3
colorama     0.4.5
Flask        2.1.2
itsdangerous 2.1.2
Jinja2       3.1.2
MarkupSafe   2.1.1
pip          22.0.4
setuptools   58.1.0
Werkzeug     2.1.2
```

Figura 19: Instalação Flask.

Fonte: O próprio autor.

2.2 Exemplo inicial com Flask

Para começar a utilização do framework Flask (Grinberg, 2018), a Figura 20 apresenta um exemplo mínimo de um código para executar uma aplicação Flask. Na

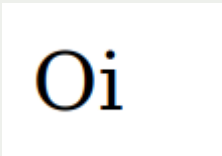
Figura 20, temos a importação do pacote Flask na linha 1. Na linha 2 estamos definindo que o arquivo atual será uma aplicação. Na linha 4 estamos definindo o que vai acontecer quando o usuário entrar na raiz da página web, onde a raiz é definida pelo símbolo '/'. Caso entre na raiz do site irá chamar a função definida na linha 5, onde irá retornar um texto "Oi" (linha 6). Na linha 8 está definindo uma condição que é executada em todo programa python, enquanto que na linha 9, está solicitando para executar a aplicação no modo 'debug=True', que a medida que o código é atualizado no arquivo é refletida na renderização no navegador, não necessitando reiniciar a aplicação a cada atualização do código. Para executar o arquivo é preciso que seja feito através do comando "python nome_do_arquivo.py".

```

1  from flask import Flask
2  app = Flask(__name__)
3
4  @app.route("/")
5  def index():
6      return "Oi"
7
8  if __name__ == '__main__':
9      app.run(debug=True)

```

Figura 20: Código inicial Flask.
Fonte: O próprio autor.



Oi

Figura 21: Renderização do navegador do código da Figura 20.
Fonte: O próprio autor.



Dica do professor: Execute o exemplo inicial na sua máquina para prosseguir com os exemplos seguintes.

2.3 Tratando requisições com Flask

Um sistemas web na maioria das vezes possui diversas páginas que são acessadas por url diferentes, a Figura 22 exemplifica a definição de várias rotas em sistemas web. Na linha 4 é definido quando o usuário acessar a rota raiz será exibido o texto “Oi”, enquanto na linha 8, ao acessar a rota “/ifmg” será exibido o texto “Curso do +IFMG”. O restante do código é similar ao exemplo anterior. A Figura 23, apresenta a renderização quando o usuário acessa a rota “/ifmg” através da url “127.0.0.1:5000/ifmg”.

```

1  from flask import Flask
2  app = Flask(__name__)
3
4  @app.route("/")
5  def index(): .....
6      return "Oi"
7
8  @app.route("/ifmg")
9  def ifmg(): .....
10     return "Curso do +IFMG"
11
12 if __name__ == '__main__':
13     app.run(debug=True)
```

Figura 22: Exemplificando várias rotas de um sistema web.
Fonte: O próprio autor.

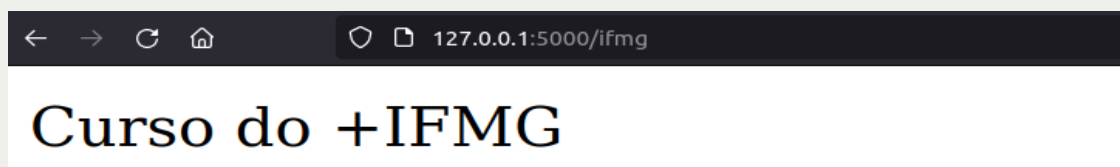


Figura 23: Renderização da rota /ifmg.
Fonte: O próprio autor.

Ao invés de ter que escrever um html completo após o return, podemos utilizar a função “render_template” para enviar um arquivo html ao usuário acessar alguma rota. Na Figura 24 na linha 6 estamos utilizando essa função enviando o arquivo html com o nome “teste.html”. O Flask define que seja criado um diretório “templates”, onde esses arquivos html precisam ser inseridos. A Figura 25, apresenta o arquivo “templates/teste.html”, contendo a tag de cabeçalho “h1” e o parágrafo “p”. A Figura 26 apresenta essa renderização.

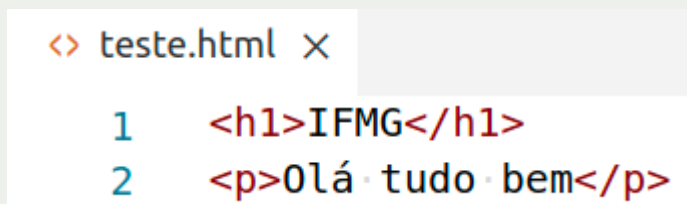
```

1  from flask import Flask, render_template
2  app = Flask(__name__)
3
4  @app.route("/")
5  def index():
6      return render_template("teste.html")
7
8  if __name__ == '__main__':
9      app.run(debug=True)

```

Figura 24: Retornando arquivo com extensão html.

Fonte: O próprio autor.



```

1  <h1>IFMG</h1>
2  <p>Olá tudo bem</p>

```

Figura 25: Arquivo teste.html .

Fonte: O próprio autor.

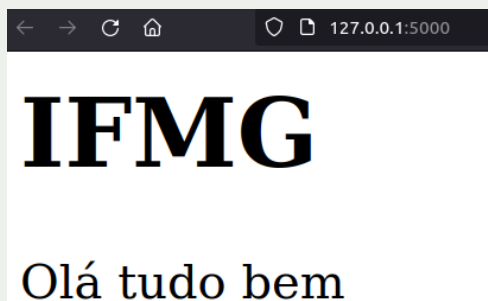


Figura 26: Renderização do exemplo utilizando arquivo com extensão html.

Fonte: O próprio autor.

Em boa parte dos sistemas web, uma parte do conteúdo não é alterado ao acessar diferentes partes do site, ou seja, é definido um modelo que as páginas do sistemas devem seguir. A Figura 27 é similar ao exemplo anterior, com exceção de enviar o arquivo “teste2.html” na linha 6. Enquanto que na Figura 28 é definido um modelo no arquivo “templates/layout.html”. Neste arquivo estamos definindo que toda a página terá um título “IFMG” (linha 4), e dentro do corpo, terá a logo do IFMG (linha 7), uma linha horizontal (linha 9) e um parágrafo de endereço (linha 10). A parte que consideramos o conteúdo específico de cada página está na linha 8, com o comando “{% block body} {% endblock %}”. A Figura 29 apresenta o arquivo “teste2.html” que utiliza o modelo definido no arquivo “layout.html”. Na linha 1 estamos definindo que irá utilizar o arquivo “layout.html”, na linha 2 inicia o bloco de conteúdo que específico desta página, que neste caso é exibir um parágrafo com o texto “Olá Claudio”. Ao final, temos que fechar o conteúdo específico da página com “endblock” na linha 5. A Figura 30 apresenta a renderização deste exemplo.

```

<> layout.html  <> teste2.html  primeiro.py X
1  from flask import Flask, render_template
2  app = Flask(__name__)
3
4  @app.route("/")
5  def index(): .....
6  ... return render_template("teste2.html")
7
8  if __name__ == '__main__': .....
9  ... app.run(debug=True)
10

```

Figura 27: Arquivo python da aplicação do modelo.

Fonte: O próprio autor.

```

<> layout.html X  <> teste2.html  primeiro.py
1  <!DOCTYPE html>
2  <html lang="pt-BR">
3  ... <head>
4  ... <title>IFMG</title>
5  ... </head>
6  ... <body>
7  ... 
8  ... {% block body %}{% endblock %}
9  ... <hr>
10 ... RUA X, Ponte Nova, Minas Gerais
11 ... </body>
12 </html>

```

Figura 28: Modelo definido que será utilizado em todas as páginas.

Fonte: O próprio autor.

```

<> layout.html  <> teste2.html X  primeiro.py
1  {% extends "layout.html" %}
2  {% block body %}
3  <p>Olá Claudio</p>
4
5  {% endblock %}

```

Figura 29: Arquivo html que utiliza um modelo.

Fonte: O próprio autor.



Figura 30: Renderização utilizando o modelo.

Fonte: O próprio autor.

Através da linguagem de programação python podemos declarar variáveis, realizar operações condicionais, aplicar repetições e muitas outras coisas. No exemplo da Figura 32, estamos usando um mecanismo de modelo (jinja) que possibilita usar código python dentro de arquivos html. Na linha 6 estamos aplicando uma condição se o número 1 é igual a 2, caso seja positivo, iria entrar na linha 7. Nas linhas 10-12 estamos aplicando uma repetição (for) onde irá exibir os numero de 0 a 2 que é a saída da função “range(3)” no python.

O código python neste mecanismo de modelo precisa está entre “{%” de forma similar ao apresentado na Figura, ou caso deseje imprimir o valor de uma variável, entre os colchetes duplos “{{}}”. Observe que ambos os códigos que utilizam a estrutura de repetição estão usando html entre o início e fim do bloco, no caso “<p>” na linha 11 e o “” na linha 16. A Figura 33 apresenta a renderização deste exemplo.

```

<> layout.html    <> teste3.html    <> teste2.html    primeiro.py X
1  from flask import Flask, render_template
2  app = Flask(__name__)
3
4  @app.route("/")
5  def index():
6      return render_template("teste3.html")
7
8  if __name__ == '__main__':
9      app.run(debug=True)

```

Figura 31: Aplicação para exemplificar condição e loop.

Fonte: O próprio autor.

```

<> layout.html  <> teste3.html x  primeiro.py
1  {% extends "layout.html" %}
2  {% block body %}
3
4  <h2>Claudio</h2>
5
6  {% if 1 == 2 %}
7  ... <p>OK</p>
8  {% endif %}
9
10 {% for i in range(3) %}
11 ... <p>{{i}}</p>
12 {% endfor %}
13
14 <ol>
15 ... {% for nome in ['Claudio', 'Maria', 'Ana'] %}
16 ... <li>{{nome}}</li>
17 ... {% endfor %}
18 </ol>
19
20 {% endblock %}

```

Figura 32: Arquivo utilizando condição e loop utilizando python.

Fonte: O próprio autor.



INSTITUTO FEDERAL
Minas Gerais

Claudio

0

1

2

1. Claudio
2. Maria
3. Ana

RUA X, Ponte Nova, Minas Gerais

Figura 33: Renderização utilizando condição e loop

Fonte: O próprio autor.

Em diversos sistemas web é necessário que o usuário forneça dados através de alguma forma de entrada, no html, para html são as tags “<input>”, associado aos inputs temos um formulário, onde as entradas são associadas. A Figura 35 apresenta um exemplo de um formulário. Na Figura 34 definimos algumas novas rotas, a rota “/formulario” que direciona o usuário para o formulário da Figura 35, e a rota “/enviaformulario” que trata o envio do formulário após o usuário preenche-lo, como

podemos verificar na linha 3 da Figura 35. Na rota de recebimento do formulário, estamos definindo dois tipos de métodos possíveis, o GET e POST. Obtemos o nome que o usuário digitou através do comando “request.form.get(“nome_pessoa”))” na linha 15, colocando este valor na variável “name” e enviamos essa informação para a página “formulario_resultado.html” da Figura 36. Na Figura 14 exibimos o valor da variável “name” na linha 3. A Figura 37 e 38 apresentam a renderização do formulário e de seu resultado após clicar em enviar.

```

primeiro.py x  formulario.html  formulario_resultado.html
1  from flask import Flask, render_template, request
2  app = Flask(__name__)
3
4  @app.route("/")
5  def index(): .....
6  ...return render_template("teste3.html")
7
8  @app.route("/formulario")
9  def formulario(): .....
10 ...return render_template("formulario.html")
11
12 @app.route("/enviaformulario", methods=["GET", "POST"])
13 def envia_formulario(): .....
14 ...if request.method == "POST":
15 ...return render_template("formulario_resultado.html", name=request.form.get("nome_pessoa"))
16
17 if __name__ == '__main__': .....
18 ...app.run(debug=True)

```

Figura 34: Exemplificação de rotas para tratar formulário.

Fonte: O próprio autor.

```

primeiro.py  formulario.html x  formulario_resultado.html
1  {% extends "layout.html" %}
2  {% block body %}
3  <form action="/enviaformulario" method="post">
4  ...<input name="nome_pessoa" placeholder="Digite o nome" type="text">
5  ...<input type="submit" value="Enviar">
6  </form>
7  {% endblock %}

```

Figura 35: Arquivo html com o formulário.

Fonte: O próprio autor.

```

primeiro.py  formulario.html  formulario_resultado.html x
1  {% extends "layout.html" %}
2  {% block body %}
3  ...<p>Olá {{name}}</p>
4  {% endblock %}

```

Figura 36: Arquivo html com o formulário.

Fonte: O próprio autor.



Figura 37: Renderização da página de formulário.

Fonte: O próprio autor.



Figura 38: Renderização da resposta do formulário.

Fonte: O próprio autor.



Atividade: Para concluir esta semana de estudos, crie uma página com os conhecimentos dessa semana. Em seguida, vá até a sala virtual e participe do fórum “Backend python”, criando um tópico e anexando duas imagens, uma da sua página renderizada pelo navegador e outra do código backend utilizando python. Você também pode responder ou comentar o tópico de algum colega.

Mas uma semana se passou, reflita o que foi passado até aqui, busque informações sobre html e páginas que você tenha interesse, aproveite a oportunidade para trocar experiência com pessoas que têm experiência nessa área.

Nos encontraremos na próxima semana.

Bons estudos!

Objetivos

Nesta semana, você compreenderá o funcionamento do backend utilizando o framework NodeJS, o framework mais popular da linguagem de programação Javascript.



Mídia digital: Antes de iniciar os estudos desta semana, vá até a sala virtual e assista à videoaula sobre “Backend Javascript”.

3.1 Exemplo inicial

Conforme a videoaula da semana, é necessário realizar a instalação do Node (Cantelon, 2014) fazendo o download no site oficial. Após a instalação do NodeJS é possível validar a instalação através do comando “node” digital em terminal. Caso a instalação tenha ocorrido com sucesso, será exibido uma mensagem de bem-vindo seguido de sua versão. O nodeJS permite executar código javascript fora do navegador web, a Figura 39 apresenta a execução de código Javascript dentro de um console NodeJS.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER

PS C:\Users\usuario> node
Welcome to Node.js v16.15.1.
Type ".help" for more information.
> let a = 'claudio';
undefined
> a
'claudio'
>
(To exit, press Ctrl+C again or Ctrl+D or type .exit)
>
PS C:\Users\usuario> |
```

Figura 39: Executando NodeJS no terminal.

Fonte: Próprio autor.

O exemplo inicial que apresentamos de utilização de NodeJS é a criação de um servidor exibindo o texto “Oi”. Inicialmente criamos um arquivo javascript com o nome “teste.js”, onde na linha 1 estamos requisitando o pacote “http” para criar o servidor, definimos seu ip e porta nas linhas 2 e 3 respectivamente. Das linhas 5 a 9 estamos criando um servidor , onde na linha estamos dizendo que o código 200 executou com

sucesso, e na linha 7 e 8 vamos escrever uma informação do tipo texto (linha 7) com o texto “Oi” (linha 8). Para finalizar precisamos da informação que o servidor está no modo de escuta, aguardando alguma requisição, no ip e porta e imprimimos no console do NodeJS a mensagem “servidor executando em http://127.0.0.1:3000/”. A Figura 41 apresenta a visualização no momento que o usuário acessa o endereço e porta no navegador web.

```
JS teste.js X
1  const http = require('http');
2  const hostname = '127.0.0.1';
3  const port = 3000;
4
5  const server = http.createServer((req, res) => {
6    res.statusCode = 200;
7    res.setHeader('Content-Type', 'text/plain');
8    res.end('Oi');
9  });
10
11 server.listen(port, hostname, () => {
12   console.log(`Servidor executando em http://${hostname}:${port}/`);
13 });
```

Figura 40: Exemplo inicial do NodeJS.

Fonte: Próprio autor.

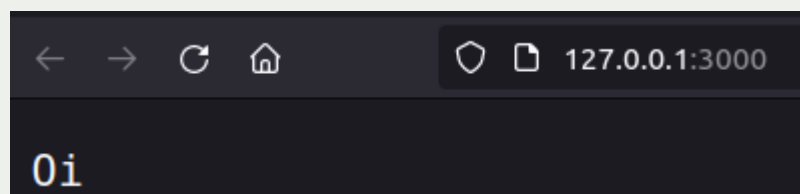


Figura 41: Exemplo inicial do NodeJS.

Fonte: Próprio autor.



Dica do professor: Execute o exemplo inicial na sua máquina para prosseguir com os exemplos seguintes.

3.2 Tratando requisições com NodeJS

No lugar de escrever texto, podemos escrever tags html, para isto, precisamos alterar o cabeçalho da escrita definindo que o texto escrito será do tipo html (linha 7) e utilizarmos o método “res.write” para escrever as tags html (linhas 8 a 11). Para finalizar a escrita, utilizamos o “res.end”, que escreve o fechamento da tag html (linha 12). O restante do código é similar ao exemplo anterior. A Figura 43 apresenta o html renderizado no navegador.

```

JS teste.js  X
1  const http = require('http');
2  const hostname = '127.0.0.1';
3  const port = 3000;
4
5  const server = http.createServer((req, res) => {
6    res.statusCode = 200;
7    res.setHeader('Content-Type', 'text/html');
8    res.write('<html>');
9    res.write('<body>');
10   res.write('<p> claudio </p>');
11   res.write('</body>');
12   res.end('</html>');
13 });
14
15 server.listen(port, hostname, () => {
16   console.log(`Servidor executando em http://${hostname}:${port}/`);
17 });

```

Figura 42: Escrevendo html na saída.
Fonte: Próprio autor.

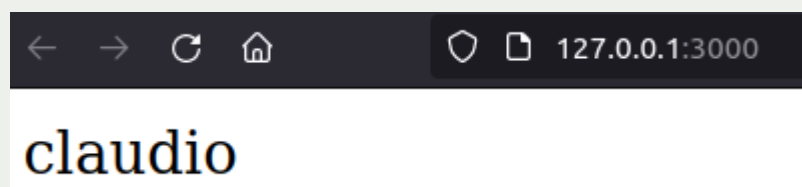


Figura 43: Renderização da escrita em html.
Fonte: Próprio autor.

No lugar de escrever no arquivo javascript as tags html com a função “res.write” podemos usar um arquivo html. Para esta funcionalidade, utilizaremos o módulo express (linha 2), enquanto que na linha 3 estamos criando uma aplicação do tipo express. O express permite criar rotas, neste caso estamos definindo que ao entrar na rota raiz (“/”, linha 5), será enviado o arquivo html que está localizado em “public/um.html” (linha 6). A variável “__dirname” na linha 6 serve para direcionar a aplicação para o diretório que está sendo executado a aplicação. A Figura 45 apresenta o arquivo “um.html” enquanto que a Figura 46 é o resultado ao acessar a raiz do site.

```

JS teste.js  X
1  const port = 3000;
2  const express = require('express');
3  const app = express();
4
5  app.get('/', function(request, res){ ...
6  ... res.sendFile(__dirname + '/public/um.html');
7  });
8
9  app.listen(port);

```

Figura 44: Escrevendo arquivo html como saída.

Fonte: Próprio autor.

```

JS teste.js  <> um.html  X
1  <!DOCTYPE html>
2  <html lang="pt-BR">
3  <head> ...
4  ... <title>IFMG</title>
5  </head>
6  <body>
7  ... <h1>IFMG</h1>
8  ... <h4>Ponte Nova</h4>
9  </body>
10 </html>

```

Figura 45: Arquivo html.

Fonte: Próprio autor.



Figura 46: Renderização da saída do arquivo html.

Fonte: Próprio autor.

Em sistemas web na maioria das vezes possui diversas páginas que são acessadas por urls diferentes, a Figura 47 exemplifica a definição de várias rotas em sistemas web utilizando o pacote express. No exemplo da Figura 47 estamos criando a rota raiz “/” e a rota “/teste”, caso o usuário entre na rota raiz será exibido o texto “IFMG” e caso entre na rota “/teste” será exibido o texto “Claudio”. A Figura 48 apresenta essa renderização do usuário entrando na rota “/teste”.

```

JS teste.js x
1  const express = require('express')
2  const app = express()
3  const port = 3000
4
5  app.get('/', (req, res) => {
6    res.send('IFMG');
7  });
8
9  app.get('/teste', (req, res) => {
10   res.send('Claudio');
11 });
12
13 app.listen(port, () => {
14   console.log(`Executando servidor na porta: ${port}`);
15 });

```

Figura 47: Renderização da saída do arquivo html.

Fonte: Próprio autor.

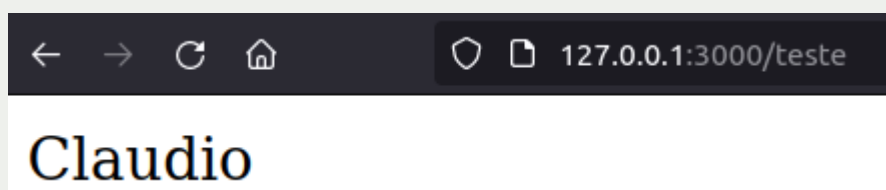


Figura 48: Renderização da rota teste.

Fonte: Próprio autor.

Sistemas web mais complexos é necessário enviar dados entre partes do sistema. O pacote “express”, permite enviar informações através de json, na linha 12, estamos criando um json que tem a chave “nome” e “clientes”, com os valores “claudio” e a lista de clientes maria e moises. Para conseguir visualizar as variáveis que são enviadas vamos utilizar o mecanismo de modelo EJS (linha 5). Veja que no início da função render da linha 12, estamos definindo que iremos enviar essas informações para o arquivo um, que neste caso precisa está em diretório chamado views com a extensão sendo .ejs, ou seja, estamos criando um arquivo em “views/um.ejs”.

A Figura 50 apresenta a utilização do EJS, onde podemos acessar as variáveis enviadas através das tags “<%= nome %>” quando queremos retornar valores de variáveis ou quando queremos escrever código javascript iniciando com “<%”. Note que da mesma forma que o Flask podemos escrever código html entre os blocos, como no caso da linha 10. A Figura 51 apresenta o resultado do código que utiliza o mecanismo EJS.

```

<> um.ejs  JS teste.js  X
1  const port = 3000;
2  const express = require('express');
3  const app = express();
4
5  app.set("view engine", 'ejs');
6
7  app.get('/', function(request, res){ ...
8  ... res.send("IFMG");
9  });
10
11 app.get('/teste-enviar', function(request, res){ ...
12 ... res.render("um", { 'nome': 'claudio', 'clientes': ['maria', 'moises'] });
13 });
14
15 app.listen(port);

```

Figura 49: Renderização da rota teste.
Fonte: Próprio autor.

```

<> um.ejs  X  JS teste.js
1  <!DOCTYPE html>
2  <html lang="pt-BR">
3  <head>
4  ... <title>Document</title>
5  </head>
6  <body>
7  ... <h1>Olá, <%= nome %> </h1>
8
9  ... <% if (nome == 'maria') { %>
10 ... <h2>ok</h2>
11 ... <% } %>
12
13 ... <% for (const n of clientes) { %>
14 ... <%= n %>
15 ... <% } %>
16 </body>
17 </html>

```

Figura 50: Renderização da rota teste.
Fonte: Próprio autor.



Figura 51: Renderização da rota teste.
Fonte: Próprio autor.

De forma similar ao Flask da semana anterior, pode tratar formulário utilizando o NodeJS. Neste caso criamos um arquivo chamado tres.html que contém o formulário da Figura 15. Para tratar formulário do tipo POST, precisamos informar para o pacote express que iremos tratar requisitar este tipo de dado como json (linhas 4 e 5). Após o envio do formulário podemos acessar através da variável “request.body” seguido o nome da variável no formulário, que neste caso é o “nome_pessoa”. A Figura 16 apresenta o resultado do envio do formulário.

```

JS teste.js  X  <> tres.html
1  const port = 3000;
2  const express = require('express');
3  const app = express();
4  app.use(express.json());
5  app.use(express.urlencoded({ extended: true }));
6
7  app.get('/', function(request, res){
8      res.sendFile(__dirname + '/views/tres.html');
9  });
10
11 app.post('/enviaformulario', function(request, res){
12     res.send(request.body.nome_pessoa);
13 });
14
15 app.listen(port);

```

Figura 52: Tratando dados de formulário

Fonte: Próprio autor.

```

JS teste.js  <> tres.html  X
1  <form action="/enviaformulario" method="post">
2      <input name="nome_pessoa" placeholder="Digite o nome" type="text">
3      <input type="submit" value="Enviar">
4  </form>

```

Figura 53: Formulário em html.

Fonte: Próprio autor.

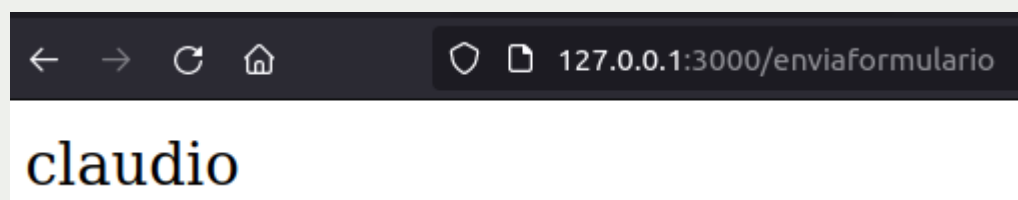


Figura 54: Renderização do formulário.

Fonte: Próprio autor.



Atividade: Para concluir esta semana de estudos, crie uma página com os conhecimentos dessa semana. Em seguida, vá até a sala virtual e participe do fórum “Backend javascript”, criando um tópico e anexando duas imagens, uma da sua página renderizada pelo navegador e outra do código backend utilizando javascript. Você também pode responder ou comentar o tópico de algum colega.

Mais uma semana terminou, dê uma pausa e reflita. Execute os exemplos desta semana e tente modificar o NodeJS para um estilo que seja mais agradável para você.

Nos encontraremos na próxima semana.

Bons estudos!

Objetivos

Nesta semana, vamos compreender como integrar um banco de dados MySQL utilizando o framework NodeJS.



Mídia digital: Antes de iniciar os estudos desta semana, vá até a sala virtual e assista à videoaula sobre “Integração banco de dados”.

4.1. Conectando banco de dados

Gerenciador de banco de dados, são a forma mais popular de armazenar dados, um gerenciador de banco de dados já implementa diversos mecanismos que estão preocupados com a velocidade, integridade e segurança dos seus dados. Nesta semana veremos como integrar o NodeJS com o banco de dados MySQL. O MySQL é um dos gerenciadores de banco de dados mais populares do mercado, utilizado por diversas pessoas, organizações e empresas. Antes de apresentar os exemplos abaixo, é necessário que esteja instalado na sua máquina o banco de dados MySQL e que você tenha conhecimento sobre a informação do usuário root e senha que foi definida na instalação seguindo o site oficial do MySQL.

Na Figura 55, a linha 1 é solicitado a utilização do pacote mysql, onde este pacote pode ser instalado com o comando “npm install mysql”. Nas linhas 3 a 7 está conectando ao banco de dados informando o local que ele está, neste caso a máquina local, o usuário e a senha. As linhas 9 a 12 conecta ao banco de dados e caso não ocorra erro será exibido a mensagem “Conectado ao mysql”.


```

JS app.js  X
1  var mysql = require('mysql');
2
3  var con = mysql.createConnection({
4    host: "localhost",
5    user: "root",
6    password: "123"
7  });
8
9  con.connect(function(err) {
10    if (err) throw err;
11    console.log("Conectado ao mysql");
12  });

```

Figura 55: Conectando um banco de dados MySQL com NodeJS.
Fonte: Próprio autor.

4.2. Executando SQL no banco de dados

Na Figura 56 exemplifica a criação de um banco de dados, na linha 13 está definindo um comando SQL, que neste caso, criar um banco de dados com o nome “aula4”. Na linha 14 é definido que a conexão deve executar o SQL através da função “con.query”, caso a execução ocorra com sucesso é exibido a mensagem “SQL executado” da linha 16.

```

JS app.js  X
1  var mysql = require('mysql');
2
3  var con = mysql.createConnection({
4    host: "localhost",
5    user: "root",
6    password: "123"
7  });
8
9  con.connect(function(err) {
10    if (err) throw err;
11    console.log("Conectado");
12
13    let sql = "create database aula4";
14    con.query(sql, function(err, result) {
15      if (err) throw err;
16      console.log("SQL executado");
17    });
18  });

```

Figura 56: Criando banco de dados.
Fonte: Próprio autor.

Na Figura 57, na linha 7 foi adicionado a informação de qual o banco de dados que será utilizado. Enquanto que na linha 13, é definido o comando SQL para a criação da tabela aluno no banco de dados, onde as demais linhas são similares ao exemplo anterior.

```

JS app.js X
1  var mysql = require('mysql');
2
3  var con = mysql.createConnection({
4      host: "localhost",
5      user: "root",
6      password: "123",
7      database: "aula4"
8  });
9
10 con.connect(function(err) {
11     if (err) throw err;
12     console.log("Conectado");
13     var sql = "create table aluno (id int primary key,\
14         nome VARCHAR(200), idade int)";
15     con.query(sql, function(err, result) {
16         if (err) throw err;
17         console.log("SQL executado");
18     });
19 });

```

Figura 57: Executando a criação de tabela.
Fonte: Próprio autor.

Para popularizar o banco de dados, podemos inserir informações através do comando “insert”. Neste caso estamos inserindo na tabela aluno a informação de Maria, que significa o identificador, seu nome e sua idade (linha 13).

```

JS app.js X
1  var mysql = require('mysql');
2
3  var con = mysql.createConnection({
4      host: "localhost",
5      user: "root",
6      password: "123",
7      database: "aula4"
8  });
9
10 con.connect(function(err) {
11     if (err) throw err;
12     console.log("Conectado");
13     var sql = "insert into aluno (id, nome, idade) VALUES (1, 'Maria', 20)";
14
15     con.query(sql, function(err, result) {
16         if (err) throw err;
17         console.log("Inserido");
18     });
19 });

```

Figura 58: Inserção do registro na tabela aluno.
Fonte: Próprio autor.

Devido a quantidade de informações, às vezes é necessário inserir diversas linhas de uma única vez, para fazer isso podemos colocar o símbolo “?” no final do comando insert na linha 13. Podemos criar uma lista de dados, que neste caso é sobre Ana, Ricardo e Marcos, onde cada item da lista é outra lista com a informação dos dados de cada pessoa (linhas 14-18). O comando de execução permite utilizar como entrada a variável de dados conforme a linha 19.

```
JS app.js
1  var mysql = require('mysql');
2
3  var con = mysql.createConnection({
4    host: "localhost",
5    user: "root",
6    password: "123",
7    database: "aula4"
8  });
9
10 con.connect(function(err){
11   if(err) throw err;
12   console.log("Connected!");
13   var sql = "insert into aluno (id, nome, idade) VALUES ?";
14   var dados = [
15     [2, 'Ana', 23],
16     [3, 'Ricardo', 11],
17     [4, 'Marcos', 13],
18   ];
19   con.query(sql, [dados], function(err, result){
20     if(err) throw err;
21     console.log("SQL executado");
22   });
23 });
```

Figura 59: Inserção de múltiplas linhas.

Fonte: Próprio autor.

Na Figura 60, exemplifica a consultar em dados no banco de dados, onde pode ser realizado através do comando “select”, que neste caso está retornando todos os campos (*) da tabela aluno. Estas informações estão sendo exibido no console do terminal (Figura 61).

```

JS app.js  X
1  var mysql = require('mysql');
2
3  var con = mysql.createConnection({
4      host: "localhost",
5      user: "root",
6      password: "123",
7      database: "aula4"
8  });
9
10 con.connect(function (err) {
11     if (err) throw err;
12     console.log("Connected!");
13
14     var sql = "select * from aluno";
15     con.query(sql, function (err, result, fields) {
16         if (err) throw err;
17         console.log("SQL executado");
18         console.log(result);
19     });
20 });

```

Figura 60: Consultar dados.

Fonte: Próprio autor.

SQL executado

```

[
  RowDataPacket { id: 1, nome: 'Maria', idade: 20 },
  RowDataPacket { id: 2, nome: 'Ana', idade: 23 },
  RowDataPacket { id: 3, nome: 'Ricardo', idade: 11 },
  RowDataPacket { id: 4, nome: 'Marcos', idade: 13 }
]

```

Figura 61: Exibição do dados do banco de dados no console.

Fonte: Próprio autor.

Informações de acesso ao banco de dados não podem ser acessadas por qualquer pessoa que não tenha as devidas permissões, para evitar este acesso, é possível utilizar o pacote “dotenv” que permite que as informações sigilosas sejam armazenadas em um arquivo “.env” conforme apresentado na Figura 63. As linhas 2 e 3 da Figura 62 definem a utilização deste pacote. Para acessar os dados armazenados, devemos apenas utilizar a variável “process.env.” seguida do nome da variável armazenada no arquivo “.env”. Todas as demais linhas são similares ao código anterior.

```

JS app.js X
1  var mysql = require('mysql');
2  const dotenv = require('dotenv');
3  dotenv.config();
4
5  var con = mysql.createConnection({
6    host: process.env.maquina,
7    user: process.env.usuario,
8    password: process.env.senha,
9    database: process.env.nome_banco
10 });
11
12 con.connect(function (err) {
13   if (err) throw err;
14   console.log("Connected!");
15   var sql = "select * from aluno";
16   con.query(sql, function (err, result, fields) {
17     if (err) throw err;
18     console.log("SQL executado");
19     console.log(result);
20   });
21 });

```

Figura 62: Utilizando dotenv para ocultar informações sigilosas.
Fonte: Próprio autor.

```

.env
1  maquina=localhost
2  usuario=root
3  senha=123
4  nome_banco=aula4

```

Figura 63: Arquivo .env que armazena informação sigilosa.
Fonte: Próprio autor.

No NodeJS é possível criar arquivos de fontes separadas e anexar esses código ao arquivo atual. Na Figura 65 apresenta o arquivo “conexao.js” que define na linha 15 que este módulo será exportado para a utilização da variável de conexão (con). A partir do arquivo é possível fazer a requisição feita na linha 4 da Figura 64, onde é possível ter acesso a essa conexão e utilizar para a consulta que seleciona os alunos na tabela aluno.

```

JS app.js X
1  const dotenv = require('dotenv');
2  dotenv.config();
3
4  const con = require("./conexao");
5
6  var sql = "select * from aluno";
7
8  con.query(sql, function (err, result, fields) {
9      if (err) throw err;
10     console.log("SQL executado");
11     console.log(result);
12 });

```

Figura 64: Exemplificando a utilização de módulo simplificar código.
Fonte: Próprio autor.

```

JS conexao.js X
1  var mysql = require('mysql');
2  var con = mysql.createConnection({
3      host: process.env.maquina,
4      user: process.env.usuario,
5      password: process.env.senha,
6      database: process.env.nome_banco
7  });
8
9  con.connect(function(err) {
10     if (err) throw err;
11
12     console.log("Conectado");
13 });
14
15 module.exports = con;

```

Figura 65: Arquivo de conexao.js separado.
Fonte: Próprio autor.

Neste último exemplo da semana, estamos consultando os alunos existentes na tabela aluno e exibindo essas informações no arquivo “views/dados.ejs”, onde estamos enviando para este arquivo todos os alunos através da variável “result” (linha 16 na Figura 66). A Figura 67 apresenta a leitura dos dados enviados e a Figura 68 a sua exibição no navegador.

```

JS app.js  X  <> dados.ejs
1  const port = 3000;
2  const express = require('express');
3  const app = express();
4  const dotenv = require('dotenv');
5  dotenv.config();
6  app.set("view engine", 'ejs');
7
8  app.get('/exibir_alunos', function (request, res) {
9      const con = require("./conexao");
10     var sql = "select * from aluno";
11     con.query(sql, function (err, result, fields) {
12         if (err) throw err;
13         console.log("SQL executado");
14         console.log(result);
15     });
16     res.render("dados", { 'result': result });
17 });
18
19
20 app.listen(port);

```

Figura 66: Envio de dados obtidos pelo banco de dados.

Fonte: Próprio autor.

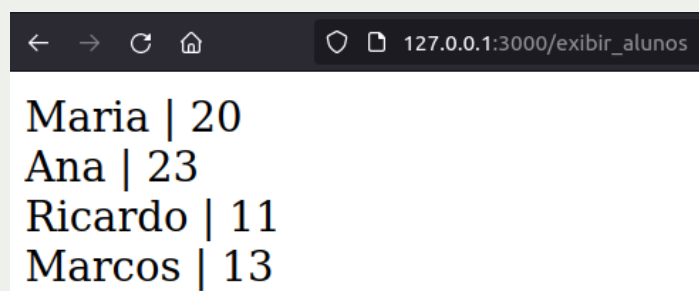
```

JS app.js  <> dados.ejs  X
1  <!DOCTYPE html>
2  <html lang="pt-BR">
3  <head>
4      <title>IFMG</title>
5  </head>
6  <body>
7      <% for (aluno of result){ %>
8          <%= aluno.nome %> | <%= aluno.idade %> <br/>
9      <% } %>
10 </body>
11 </html>

```

Figura 67: Execução EJS para exibir dados dos alunos.

Fonte: Próprio autor.



127.0.0.1:3000/exibir_alunos

Maria | 20
Ana | 23
Ricardo | 11
Marcos | 13

Figura 68: Renderização da exibição dos alunos.

Fonte: Próprio autor.

Espero com este curso que você consiga criar páginas web, conseguir entender códigos e atualizar páginas existentes. A partir deste curso, desejo que você consiga desenvolver seus projetos pessoais e consiga buscar novas funcionalidades para aprimorar seus conhecimentos, incentivo a buscar outros cursos na plataforma +IFMG, na qual oferece diversos cursos nas mais diferentes áreas de interesse.

Este modelo guiado (autocontido) de *e-book* deve propiciar autonomia e, juntamente com as videoaulas, possibilitar a plena capacidade de aprendizagem (lembre-se: não haverá tutoria!). Deste modo, a sala virtual será simplificada e de fácil usabilidade (servirá basicamente para a inclusão dos vídeos e atividades avaliativas).



Atividade: Para concluir o curso e gerar o seu certificado, vá até a sala virtual e responda ao Questionário “Avaliação geral”. Este teste é constituído por 10 perguntas de múltipla escolha, que se baseiam em todo o curso.

Parabéns pela conclusão do curso. Foi um prazer tê-lo conosco!

Referências

W3. Disponível em: <<https://www.w3schools.com/>>. Acesso em: 14 julho. 2022.

Freeman, Eric. Use a Cabeça! Programação em HTML 5. Alta Books Editora, 2014.

Silva, Maurício Samy. Fundamentos de HTML5 e CSS3. Novatec Editora, 2018.

Grinberg, Miguel. Flask web development: developing web applications with python. "O'Reilly Media, Inc.", 2018.

Cantelon, Mike, et al. Node. js in Action. Greenwich: Manning, 2014. Pereira, Caio Ribeiro. Aplicações web real-time com Node. js. Editora Casa do Código, 2014.



Currículo do autor



Realizou Mestrado em Computação Aplicada na Universidade Estadual de Feira de Santana. Possui Especialização em Banco de Dados no Centro Universitário Claretiano. Possui Bacharelado em Ciência da Computação na Universidade Estadual de Santa Cruz. Atualmente é professor no Instituto Federal de Minas Gerais no *Campus* Ponte Nova.

Currículo Lattes: <http://lattes.cnpq.br/9443774889076411>

Feito por (professor-autor)	Data	Revisão de <i>layout</i>	Data	Versão
Claudio Moisés Valiense de Andrade	19/07/2022	Desginado pela proex		1.0

Glossário de códigos QR (*Quick Response*)



Mídia digital
Apresentação do
curso e frontend



Dica do professor
Backend python



Dica do professor
Backend javascript



Mídia digital
Integração com
banco de dados







Plataforma +IFMG

Formação Inicial e Continuada EaD



A Pró-Reitoria de Extensão (Proex), desde o ano de 2020, concentrou seus esforços na criação do Programa +IFMG. Esta iniciativa consiste em uma plataforma de cursos *online*, cujo objetivo, além de multiplicar o conhecimento institucional em Educação a Distância (EaD), é aumentar a abrangência social do IFMG, incentivando a qualificação profissional. Assim, o programa contribui para o IFMG cumprir seu papel na oferta de uma educação pública, de qualidade e cada vez mais acessível.

Para essa realização, a Proex constituiu uma equipe multidisciplinar, contando com especialistas em educação, *web design*, *design* instrucional, programação, revisão de texto, locução, produção e edição de vídeos e muito mais. Além disso, contamos com o apoio sinérgico de diversos setores institucionais e também com a imprescindível contribuição de muitos servidores (professores e técnico-administrativos) que trabalharam como autores dos materiais didáticos, compartilhando conhecimento em suas áreas de

atuação.

A fim de assegurar a mais alta qualidade na produção destes cursos, a Proex adquiriu estúdios de EaD, equipados com câmeras de vídeo, microfones, sistemas de iluminação e isolamento acústica, para todos os 18 *campi* do IFMG.

Somando à nossa plataforma de cursos *online*, o Programa +IFMG disponibilizará também, para toda a comunidade, uma Rádio *Web* Educativa, um aplicativo móvel para Android e IOS, um canal no Youtube com a finalidade de promover a divulgação cultural e científica e cursos preparatórios para nosso processo seletivo, bem como para o Enem, considerando os saberes contemplados por todos os nossos cursos.

Parafraseando Freire, acreditamos que a educação muda as pessoas e estas, por sua vez, transformam o mundo. Foi assim que o +IFMG foi criado.

O +IFMG significa um IFMG cada vez mais perto de você!

Professor Carlos Bernardes Rosa Jr.
Pró-Reitor de Extensão do IFMG



Características deste livro:

Formato: A4

Tipologia: Arial e Capriola.

E-book:

1ª. Edição

Formato digital

