

INSTITUTO FEDERAL

Paraná

Campus Paranaguá

Geração de HTML

Biblioteca Pug

Desenvolvimento Web

Prof. Diego Stiehl

Resposta do Servidor

- Nosso servidor sabe responder apenas JSON
 - Não serve para um site “normal”
- Precisamos enviar HTML
 - É isso que o **browser** conhece
- Podemos enviar arquivos HTML estáticos
 - Prontos: cliente apenas faz o download e mostra
 - Problema:
 - Nada customizado (igual para todos)
 - Conteúdo não é dinâmico

Arquivos Estáticos

- Arquivos estáticos também são importantes
- Alguns recursos não precisam ser dinâmicos:
 - CSS
 - JavaScript
 - Imagens
 - Algumas páginas HTML
- Exemplos:
 - <http://meusite.com/css/index.css>
 - <http://meusite.com/imagens/usuario.png>

Precisam sempre retornar o
mesmo recurso (estático)



Arquivos Estáticos

- Na Atividade 07 fizemos um servidor que responde via HTTP com qualquer arquivo estático alocado na pasta public
 - Baita trabalhadeira
- Podemos fazer o mesmo com o Express:

```
const path = require('path');  
// ... Várias outras coisas ...  
app.use(express.static(path.join(__dirname, 'public')));
```

Módulo que ajuda a “resolver”
caminhos de arquivos

Diretório do arquivo.js atual

Diretório com os
arquivos estáticos

Client-side rendering

- Então arquivos **HTML** estáticos não permitem qualquer tipo de dinamização?
- Permitem, mas apenas no browser
 - Usando JavaScript (lado cliente)
- O browser faz download das páginas
 - Renderiza na tela
 - Após isto, o JavaScript pode alterar conteúdo
- Chamamos isso de client-side rendering

Server-side rendering

- Podemos gerar HTML customizado no servidor e retornar ele “pronto”
 - Para o browser é como se fosse estático
- Vantagens?
 - Tudo que a programação do lado do servidor nos permitir fazer
 - Páginas dinâmicas
 - Conteúdo diferente para diferentes parâmetros

Usando o que sabemos...

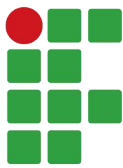
- Beleza! Então podemos fazer:

```
app.get('/', (req, res) => {  
  const mensagemCustomizada = '...';  
  res.status(200).send(`  
    <!DOCTYPE html>  
    <html lang="en">  
    <head>  
      <meta charset="UTF-8">  
      <meta name="viewport" content="width=device-width, initial-scale=1.0">  
      <meta http-equiv="X-UA-Compatible" content="ie=edge">  
      <title>Página do ${req.query.usuario.nome}</title>  
    </head>  
    <body>  
      <h1>Página do ${req.query.usuario}</h1>  
      <p>${mensagemCustomizada}</p>  
      <div>  
        <div class="conteudo">  
            
        </div>  
      </div>  
    </body>  
  </html>  
`);  
});
```

SPOILER!
Melhor não, né?

Template Engine

- Um template engine permite centralizar a montagem de HTML em arquivos próprios
 - Um arquivo para cada recurso da aplicação
 - Cada recurso é uma view
- Garante organização de código
- Separação de conceitos
- Controle total das views geradas



Pug

- **Pug** é um template engine feito para Node.js
 - Tem forte integração ao Express
 - Construção de HTML dinâmico
- Possui uma sintaxe própria
 - Uma simplificação do HTML
 - Mas permite trechos de HTML puro
- Não é a única opção
 - [Mustache](#), [EJS](#), ...



Sintaxe

arquivo.pug

```
- var user = { description: 'foo bar baz' }
```

```
- var authorised = false
```

```
#user
```

```
  if user.description
```

```
    h2.green Description
```

```
    p.description= user.description
```

```
  else if authorised
```

```
    h2.blue Description
```

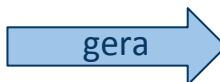
```
    p.description.
```

```
      User has no description,  
      why not add one...
```

```
  else
```

```
    h2.red Description
```

```
    p.description User has no description
```



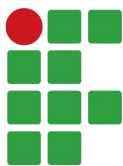
HTML gerado para o browser

```
<div id="user">
```

```
  <h2 class="green">Description</h2>
```

```
  <p class="description">foo bar baz</p>
```

```
</div>
```



Pug

- Site: <https://pugjs.org>
- Instalação:

`npm i pug --save`

- Usando no Express:

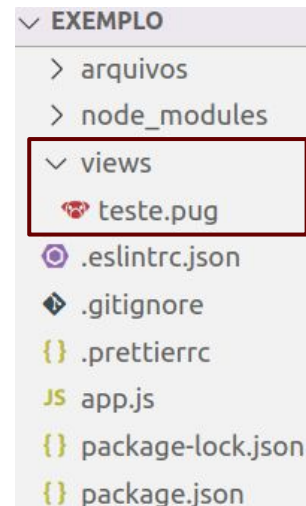
```
app.set('view engine', 'pug');  
app.set('views', path.join(__dirname, 'views'));
```

```
app.get('/', (req, res) => {  
  res.status(200).render('teste');  
});
```

Definindo template engine

Apontando diretório views

Vai procurar por arquivo
`./views/teste.pug`

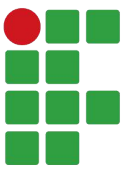


Exemplo: teste.pug

- Criar pasta **views**
 - Criar arquivo **teste.pug**

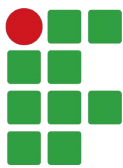
```
doctype html
html
  head
    meta(charset='utf-8')
    title Testando Pug
  body
    header.conteudo
      h1 Testando pug
      h2 Também estou no header
    .container
      p#capitulo-1 Texto do a página
      p Outro parágrafo
      img(src="/imagens/imagem.png" alt="Imagem de exemplo")
```

Indentação manda na estrutura
tag texto define uma tag com conteúdo textual
Parênteses define atributos
Ponto (.) indica uma classe
Cerquilha (#) indica um ID



HTML Gerado

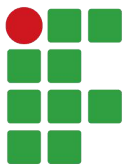
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Testando Pug</title>
  </head>
  <body>
    <header class="conteudo">
      <h1>Testando pug</h1>
      <h2>Também estou no header</h2>
    </header>
    <div class="container">
      <p id="capitulo-1">Texto do a página</p>
      <p>Outro parágrafo</p>
      
    </div>
  </body>
</html>
```



Variáveis

- Podemos enviar parâmetros do JavaScript para a view em Pug

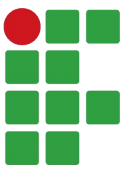
```
app.get('/', (req, res) => {  
  res.status(200).render('teste', {  
    tituloPrimario: 'Título da Página',  
    tituloSecundario: 'Muito legal...'  
  });  
});
```



Para Ler

- Para usar os parâmetros
 - Interpolação
 - `{variavel}`
 - Buffered code
 - `tag= variavel`

```
doctype html
html
head
  meta(charset='utf-8')
  title Testando Pug - #{tituloPrimario}
body
  header.conteudo
    h1= tituloPrimario
    h2= tituloSecundario
  //- ... Resto da página ...
  //- Assim que se escreve um comentário
```

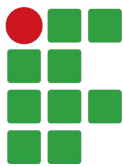


HTML Gerado

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Testando Pug - Título da Página</title>
  </head>
  <body>
    <header class="conteudo">
      <h1>Título da Página</h1>
      <h2>Muito legal...</h2>
    </header>
    <div class="container">
      <p id="capitulo-1">Texto do a página</p>
      <p>Outro parágrafo</p>
      
    </div>
  </body>
</html>
```


Voltando ao nosso site...

- Voltando ao exemplo de Usuários
 - Atividade 08
- Rotas existentes estão respondendo JSON
- Vamos fazê-las gerar HTML
 - GET /usuarios
 - Página HTML com todos usuários
 - GET /usuario?id=999
 - Página HTML com os dados do usuário 999
 - POST /login (parâmetros: email e senha)
 - Página HTML com resultado do login
 - Também criar outra página com um formulário HTML



Templates

<http://localhost:3000/usuarios>

IFPR Paranaguá

Seja bem vindo

Usuários Cadastrados

Fritz fritz@caozinho.com

Fritz fritz@caozinho.com

Fritz fritz@caozinho.com

Fritz fritz@caozinho.com

Fritz fritz@caozinho.com

Fritz fritz@caozinho.com

Fritz fritz@caozinho.com

Instituto Federal do Paraná. Todos os direitos reservados.

<http://localhost:3000/usuario?id=4>

IFPR Paranaguá

Seja bem vindo

Diego Stiehl

E-mail: diego.stiehl@ifpr.edu.br

Voltar

Instituto Federal do Paraná. Todos os direitos reservados.

Os templates em HTML puro (+ Bootstrap)
estão disponíveis no Moodle.
Acesse e faça o download.

Parâmetros a URL

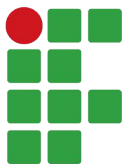
- O Express aceita parâmetros direto na URL
 - Deixar a URL mais legível para o usuário
- No lugar de usar:
 - <http://localhost:3000/usuario?id=4>
- Usaremos
 - <http://localhost:3000/usuarios/4>
- Para ler:

```
app.get('/usuarios/:id', (req, res) => {  
  const id = req.params.id;  
  // ...  
});
```

GET /usuarios/:id

- app.js

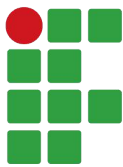
```
app.get('/usuarios/:id', (req, res) => {  
  const id = req.params.id * 1;  
  const usuario = usuarios.find(u => u.id === id);  
  if (usuario) {  
    res.status(200).render('usuarios/mostrar', {  
      usuario // abreviação de 'usuario: usuario'  
    });  
  } else {  
    res.status(404).render('404');  
  }  
});
```



View

- Criar arquivo **/views/usuarios/mostrar.pug**

```
doctype html
html
  head
    meta(charset='utf-8')
    meta(name='viewport', content='width=device-width, initial-scale=1, shrink-to-fit=no')
    link(rel='stylesheet' href='https://.../bootstrap.min.css' integrity='...' crossorigin='anonymous')
    title IFPR - Usuário: #{usuario.nome}
  body
    .container
      header.jumbotron
        h1 IFPR Paranaguá
        p Seja bem vindo
      .card
        .card-header= usuario.nome
        .card-body
          p.card-text E-mail: #{usuario.email}
          a.btn.btn-primary(href='/usuarios') Voltar
      footer.jumbotron.mt-4.d-flex.justify-content-end
        p Instituto Federal do Paraná. Todos os direitos reservados.
```



- Criar arquivo **/views/404.pug**

```
doctype html
html
  head
    meta(charset='utf-8')
    meta(name='viewport', content='width=device-width, initial-scale=1, shrink-to-fit=no')
    link(rel='stylesheet' href='https://.../bootstrap.min.css' integrity='...' crossorigin='anonymous')
    title 404 - Recurso não encontrado
  body
    .container
      header.jumbotron
        h1 IFPR Paranaguá
        p Seja bem vindo
        h1 404 -
        ins Recurso não encontrado
      footer.jumbotron.mt-4.d-flex.justify-content-end
        p Instituto Federal do Paraná. Todos os direitos reservados.
```

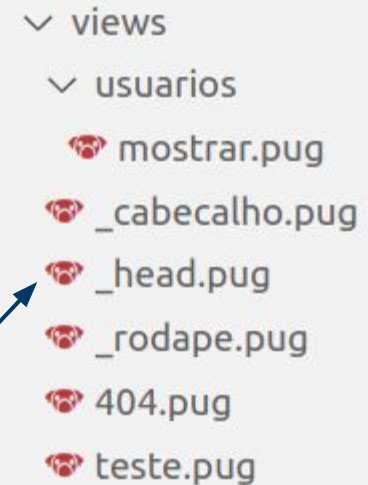
Trechos Repetidos

- Algumas partes do documento irão se repetir em outras páginas do site
 - Trecho da tag <head>
 - Cabeçalho
 - Rodapé
- Podemos separar arquivos
- Para importar no .pug

`include _arquivoParcial`

underline

Convenção de nome de arquivos parciais



```
views
└─ usuarios
   ├── mostrar.pug
   ├── _cabecalho.pug
   ├── _head.pug
   ├── _rodape.pug
   ├── 404.pug
   └── teste.pug
```

Novo mostrar.pug

- `../_arquivo` ← volta um diretório

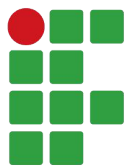
```
doctype html
html
  head
    include ../_head
    title IFPR - Usuário: #{usuario.nome}
  body
    .container
      include ../_cabecalho
      .card
        .card-header= usuario.nome
        .card-body
          p.card-text E-mail: #{usuario.email}
          a.btn.btn-primary(href='/usuarios') Voltar
      include ../_rodape
```


Arquivos Parciais

```
//- views/_head.pug
meta(charset='utf-8')
meta(name='viewport', content='width=device-width, initial-scale=1, shrink-to-fit=no')
link(rel='stylesheet' href='https://.../ bootstrap.min.css' integrity='...' crossorigin='anonymous')
```

```
//- views/_cabecalho.pug
header.jumbotron
  h1 IFPR Paranaguá
  p Seja bem vindo
```

```
//- views/_rodape.pug
footer.jumbotron.mt-4.d-flex.justify-content-end
  p Instituto Federal do Paraná. Todos os direitos reservados.
```



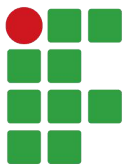
Prática

- Refatorar 404.pug
- Adicionar os includes

GET /usuarios

- app.js

```
app.get('/usuarios', (req, res) => {  
  res.status(200).render('usuarios/todos', {  
    usuarios // abreviação de 'usuarios: usuarios'  
  });  
});
```



View

- Criar arquivo **/views/usuarios/todos.pug**

```
doctype html
html
  head
    include ../_head
    title IFPR - Usuários Cadastrados
  body
    .container
      include ../_cabecalho

    .card
      .card-header Usuários Cadastrados
      ul.list-group.list-group-flush
        each usuario in usuarios
          li.list-group-item
            a(href="/usuarios/${usuario.id}")= usuario.nome
            a.badge.badge-light(href="mailto:${usuario.email}")= usuario.email

    include ../_rodape
```

Ainda repetitivo...

- Já temos 3 views
- Nota que estamos gerando código repetitivo?
 - Mesmo com os includes!
- Solução:
 - Criar um template-base com as partes únicas
 - Depois estender o template e mudar somente o que for necessário para cada view

Herança de Template

- Pug suporta a herança (extensão) de template que precisamos
- Devemos criar um template-base
 - Definimos **blocos** (blocks) dentro dele
- Outros templates podem **estendê-lo** (extends)
 - Podemos substituir seus blocos pelo conteúdo desejado

Template Base

- Criar arquivo `/views/base.pug`

```
doctype html
html
  head
    include _head
    title IFPR - #{titulo}
  body
    .container
      include _cabecalho

      block conteudo
        h2 Conteúdo que será substituído em caso de herança

      include _rodape
```

Template base para os demais
(nunca renderizaremos ele diretamente)

Criando um bloco (**block**)

Conteúdo (opcional)

Herança - mostrar.pug

- Alterar arquivo **/views/usuarios/mostrar.pug**

```
extends ../base
```

```
block conteudo
```

```
.card
```

```
.card-header= usuario.nome
```

```
.card-body
```

```
p.card-text E-mail: #{usuario.email}
```

```
a.btn.btn-primary(href='/usuarios') Voltar
```

Estendendo de outro template
Substituindo conteúdo do bloco
“conteúdo”

No conteúdo do bloco:

Escrever somente o que for único para esta
view.

Será renderizado no espaço definido para o
bloco no template base.

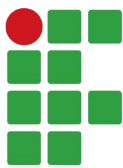
Parâmetro na Rota

- Precisamos informar o parâmetro titulo
 - O objeto foi usado no template-base

```
app.get('/usuarios/:id', (req, res) => {  
  const id = req.params.id * 1;  
  const usuario = usuarios.find(u => u.id === id);  
  if (usuario) {  
    res.status(200).render('usuarios/mostrar', {  
      usuario,  
      titulo: `Usuário: ${usuario.nome}`  
    });  
  } else {  
    res.status(404).render('404', {  
      titulo: 'Página não encontrada'  
    });  
  }  
});
```

Prática

- Atualize nossos outros dois templates
 - Listagem de usuários
 - Página de 404
- Faça eles herdarem do template base



Rota *

- Podemos criar uma rota padrão
 - Ao final do arquivo **app.js**
 - Após todas as outras rotas
 - Será usada se nenhuma rota da lista encaixar
- Retornar um 404

Qualquer método HTTP

Qualquer URL

```
app.all('*', (req, res) => {  
  res.status(404).render('404');  
});
```

Prática (Login 1)

- Crie o formulário de login com base no HTML
 - Rota: **GET /login**
 - **views/login.pug**

```
<div class="row justify-content-center">
  <div class="col-lg-4">
    <form action="/login" method="post">
      <div class="form-group">
        <label for="email">Email</label>
        <input type="email" class="form-control" name="email">
        <small class="form-text text-muted">Igual ao seu cadastro.</small>
      </div>
      <div class="form-group">
        <label for="senha">Senha</label>
        <input type="password" class="form-control" name="senha">
      </div>
      <button type="submit" class="btn btn-primary">Login</button>
    </form>
  </div>
</div>
```

Prática (Login 2)

- Crie um diretório para as views de login
 - /views/usuarios/login
- Refatore sua rota **POST /login**
- Receber e tratar formulário de login
 - Renderizar view de acordo
 - loginSucesso.pug
 - loginFalha.pug
 - Customizar views

Atividade

- Um usuário pode se cadastrar
 - <http://localhost:3000/registrar>
 - GET /registrar
 - /views/usuarios/registrar.pug
- Campos
 - Nome, email, senha e repetir senha
- Requisitos
 - Fazer validações: se não passar → voltar ao form
 - Explicar falhas: Campos vazios ou senha não bater
 - Ao cadastrar, mostrar link para:
 - <http://localhost:3000/usuarios/ID>

Mas e salvar onde? E como?

- O processo de registrar consistem em:
 - Receber os dados do formulário
 - Se estiver tudo certo, criar um objeto de usuário
 - Adicionar este novo objeto ao Array **usuarios**
 - Transformar o Array **usuarios** inteiro em JSON
 - Salvar esta String JSON no arquivo **usuarios.json**

```
usuarios.push(objetoDoMeuUsuarioRegistrado);  
const usuariosJSON = JSON.stringify(usuarios);  
fs.writeFile(  
  path.join(__dirname, 'arquivos', 'usuarios.json'), usuariosJSON, 'utf-8',  
  erro => {  
    // Aqui eu sei que o salvamento terminou (com sucesso ou erro)  
    // e eu devo usar para montar a minha response  
  }  
);
```