GUIA COMPLETO: Como Criar Novas Páginas e Definir Rotas

INTRODUÇÃO

Este guia explica passo a passo como criar novas páginas e definir rotas no seu aplicativo híbrido Node.js/Express.js. Você aprenderá a estrutura do projeto e como adicionar funcionalidades de forma organizada.

ESTRUTURA DO PROJETO

```
aula1/
  routes/
     pages.js      # Rotas das páginas HTML
     api.js        # Rotas da API REST
  views/
     layout.ejs    # Layout base
     index.ejs     # Página inicial
     sobre.ejs     # Página sobre
     tarefas.ejs   # Página de tarefas
     contato.ejs   # Página de contato
     404.ejs       # Página 404
     partials/
         header.ejs # Cabeçalho
  config/
     database.js   # Configuração do banco
  index.js          # Arquivo principal
```

COMO CRIAR UMA NOVA PÁGINA

PASSO 1: Adicionar Rota em `routes/pages.js`

```
/**
 * NOVA PÁGINA
 * ===========
 * Rota: GET /nova-pagina
 * Descrição: Sua nova página
 */
router.get('/nova-pagina', (req, res) => {
    console.log('  Acessando nova página...');

    // Dados para a página
    const pageData = {
```

```
        title: 'Nova Página',
        description: 'Descrição da sua nova página',
        // Adicione mais dados conforme necessário
        customData: 'Dados personalizados',
        items: ['Item 1', 'Item 2', 'Item 3']
    };

    // Renderiza a página
    res.render('nova-pagina', pageData);
});
```

PASSO 2: Criar Arquivo de Template `views/nova-pagina.ejs`

```
<div class="card">
    <h2><%= title %></h2>
    <p><%= description %></p>
</div>

<div class="card">
    <h3>  Dados Personalizados</h3>
    <p><%= customData %></p>

    <h4>Lista de Itens:</h4>
    <ul>
        <% items.forEach(function(item) { %>
        <li><%= item %></li>
        <% }); %>
    </ul>
</div>

<div class="card">
    <h3>  Teste de Funcionalidades</h3>
    <button onclick="testarFuncao()" class="btn">Testar Função</button>
</div>

<script>
function testarFuncao() {
    alert('Função testada com sucesso!');
}
</script>
```

PASSO 3: Adicionar Link no Menu (Opcional)

Edite o arquivo `views/layout.ejs` e adicione o link no menu:

```
<nav class="nav">
    <ul>
```

```
        <li><a href="/">  Início</a></li>
        <li><a href="/sobre">  Sobre</a></li>
        <li><a href="/tarefas">  Tarefas</a></li>
        <li><a href="/contato">  Contato</a></li>
        <li><a href="/nova-pagina">  Nova Página</a></li> <!-- NOVO LINK -->
        <li><a href="/api/status">  API Status</a></li>
    </ul>
</nav>
```

COMO CRIAR NOVAS ROTAS DE API

PASSO 1: Adicionar Rota em `routes/api.js`

```
/**
 * NOVA API
 * ========
 * Rota: GET /api/nova-api
 * Descrição: Sua nova API
 */
router.get('/nova-api', (req, res) => {
    console.log('  Acessando nova API...');

    const responseData = {
        success: true,
        message: 'Nova API funcionando!',
        data: {
            timestamp: new Date().toISOString(),
            version: '1.0.0',
            customData: 'Dados personalizados da API'
        }
    };

    res.json(responseData);
});

/**
 * API COM PARÂMETROS
 * ==================
 * Rota: GET /api/nova-api/:id
 * Descrição: API com parâmetro de ID
 */
router.get('/nova-api/:id', (req, res) => {
    const { id } = req.params;
    console.log(`  Acessando nova API com ID: ${id}`);
```

```
    res.json({
        success: true,
        message: `API chamada com ID: ${id}`,
        id: id,
        timestamp: new Date().toISOString()
    });
});

/**
 * API POST
 * ========
 * Rota: POST /api/nova-api
 * Descrição: API para receber dados
 */
router.post('/nova-api', (req, res) => {
    console.log('  Recebendo dados via POST...');
    console.log('Dados recebidos:', req.body);

    res.json({
        success: true,
        message: 'Dados recebidos com sucesso!',
        receivedData: req.body,
        timestamp: new Date().toISOString()
    });
});
```

PERSONALIZANDO O LAYOUT

Modificar o Layout Base (`views/layout.ejs`)

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title><%= title %> - Aplicativo Híbrido</title>
    <style>
        /* Seus estilos personalizados aqui */
        .minha-classe {
            background: linear-gradient(45deg, #ff6b6b, #4ecdc4);
            padding: 20px;
            border-radius: 10px;
        }
    </style>
</head>
```

```
<body>
    <div class="container">
        <!-- Cabeçalho -->
        <div class="header">
            <h1> Aplicativo Híbrido</h1>
            <p><%= description %></p>
        </div>

        <!-- Menu de Navegação -->
        <nav class="nav">
            <ul>
                <li><a href="/"> Início</a></li>
                <li><a href="/sobre"> Sobre</a></li>
                <li><a href="/tarefas"> Tarefas</a></li>
                <li><a href="/contato"> Contato</a></li>
                <li><a href="/api/status"> API Status</a></li>
            </ul>
        </nav>

        <!-- Conteúdo Principal -->
        <main class="content">
            <%- body %>
        </main>

        <!-- Rodapé -->
        <div class="footer">
            <p> Aplicativo Híbrido - Aula 1 | Desenvolvido para fins educacionais</p>
        </div>
    </div>
</body>
</html>
```

EXEMPLOS PRÁTICOS

Exemplo 1: Página de Produtos

1. Rota (routes/pages.js):

```
router.get('/produtos', (req, res) => {
    const pageData = {
        title: 'Produtos',
        description: 'Nossos produtos disponíveis',
        produtos: [
            { id: 1, nome: 'Produto A', preco: 99.90, estoque: 10 },
            { id: 2, nome: 'Produto B', preco: 149.90, estoque: 5 },
```

```
            { id: 3, nome: 'Produto C', preco: 199.90, estoque: 0 }
        ]
    };
    res.render('produtos', pageData);
});
```

2. Template (`views/produtos.ejs`):

```
<div class="card">
    <h2> Nossos Produtos</h2>
    <p><%= description %></p>
</div>

<div class="card">
    <h3> Lista de Produtos</h3>
    <div style="display: grid; grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
        <% produtos.forEach(function(produto) { %>
        <div style="background: rgba(0, 0, 0, 0.2); padding: 15px; border-radius: 10px;">
            <h4><%= produto.nome %></h4>
            <p><strong>Preço:</strong> R$ <%= produto.preco %></p>
            <p><strong>Estoque:</strong>
                <span style="color: <%= produto.estoque > 0 ? '#4ade80' : '#f87171' %>">
                    <%= produto.estoque > 0 ? produto.estoque + ' unidades' : 'Esgotado' %>
                </span>
            </p>
        </div>
        <% }); %>
    </div>
</div>
```

Exemplo 2: API de Produtos

Rota (`routes/api.js`):

```
router.get('/produtos', (req, res) => {
    const produtos = [
        { id: 1, nome: 'Produto A', preco: 99.90, estoque: 10 },
        { id: 2, nome: 'Produto B', preco: 149.90, estoque: 5 },
        { id: 3, nome: 'Produto C', preco: 199.90, estoque: 0 }
    ];

    res.json({
        success: true,
        data: produtos,
        total: produtos.length,
        timestamp: new Date().toISOString()
    });
```

```
});

router.get('/produtos/:id', (req, res) => {
    const { id } = req.params;
    const produto = produtos.find(p => p.id == id);

    if (produto) {
        res.json({
            success: true,
            data: produto,
            timestamp: new Date().toISOString()
        });
    } else {
        res.status(404).json({
            success: false,
            message: 'Produto não encontrado',
            timestamp: new Date().toISOString()
        });
    }
});
```

DICAS E BOAS PRÁTICAS

1. Organização de Arquivos

- Mantenha as rotas organizadas por funcionalidade
- Use nomes descritivos para arquivos e rotas
- Separe lógica de negócio da apresentação

2. Nomenclatura de Rotas

- Use kebab-case para URLs: `/minha-pagina`
- Use camelCase para variáveis JavaScript
- Seja consistente na nomenclatura

3. Tratamento de Erros

```
router.get('/pagina-com-erro', (req, res) => {
    try {
        // Sua lógica aqui
        res.render('pagina', data);
    } catch (error) {
        console.error('Erro na página:', error);
        res.status(500).render('erro', {
            message: 'Erro interno do servidor'
```

```
        });
    }
});
```

4. Validação de Dados

```
router.post('/api/dados', (req, res) => {
    const { nome, email } = req.body;

    if (!nome || !email) {
        return res.status(400).json({
            success: false,
            message: 'Nome e email são obrigatórios'
        });
    }

    // Processar dados...
    res.json({ success: true, message: 'Dados processados!' });
});
```

COMANDOS PARA TESTAR

Iniciar o Servidor:

```
node index.js
```

Testar Páginas:

- `http://localhost:3000/` - Página inicial
- `http://localhost:3000/nova-pagina` - Sua nova página
- `http://localhost:3000/produtos` - Página de produtos

Testar APIs:

- `http://localhost:3000/api/status` - Status da API
- `http://localhost:3000/api/produtos` - API de produtos
- `http://localhost:3000/api/produtos/1` - Produto específico

RECURSOS ADICIONAIS

Documentação Oficial:

- Express.js
- EJS Templates
- Node.js

Ferramentas Úteis:

- **Postman** - Para testar APIs
- **Insomnia** - Alternativa ao Postman
- **VS Code** - Editor recomendado

CHECKLIST DE CRIAÇÃO DE PÁGINAS

☐ Criar rota em `routes/pages.js`
☐ Criar template em `views/`
☐ Adicionar link no menu (opcional)
☐ Testar a página no navegador
☐ Verificar se os dados estão sendo passados corretamente
☐ Testar responsividade
☐ Adicionar tratamento de erros

CONCLUSÃO

Agora você sabe como criar novas páginas e rotas no seu aplicativo! Lembre-se de:

1. **Manter a organização** - Separe rotas por funcionalidade
2. **Usar templates** - EJS facilita a criação de páginas dinâmicas
3. **Testar sempre** - Verifique se tudo está funcionando
4. **Documentar** - Comente seu código para facilitar manutenção

Boa programação!