JavaScript: Fundamentos para Desenvolvimento Web Interativo - Turma 2025B

6.2.2 Validando formulários com JavaScript

O método mais flexível e poderoso para validação do lado do cliente é através do JavaScript. Com ele, os desenvolvedores podem criar scripts detalhados que verificam os dados em tempo real enquanto o usuário preenche o formulário. Por exemplo, você pode usar eventos de JavaScript para verificar a validade de um campo assim que ele perde o foco.

Usando a API de validação de restrição

A **API de Validação de Restrições** (Constraint Validation API) é uma interface do HTML5 que oferece uma série de métodos e propriedades para verificar a validade de campos de formulário diretamente no navegador. Ela permite que os desenvolvedores apliquem validações personalizadas e forneçam mensagens de erro claras, usando os atributos HTML5 existentes como required, min, max, pattern, e também métodos JavaScript específicos. Isso melhora significativamente a experiência de validação do lado do cliente.

Os métodos de API de validação de restrições são os seguintes:

- **checkValidity():** Verifica se o elemento é válido de acordo com as regras definidas nos atributos HTML.
- **setCustomValidity():** Define uma mensagem de erro personalizada. Se a mensagem for uma string vazia, o campo é considerado válido.

Conforme observado anteriormente, cada vez que o usuário tenta enviar um formulário inválido, o navegador exibe uma mensagem de erro. Essas mensagens fornecem feedback sobre o motivo do erro, orientando os usuários sobre como corrigir suas entradas. No entanto, a forma como esta mensagem é exibida depende do navegador. Isso dificulta a personalização da aparência destas mensagens, uma vez que não existe uma maneira de modificar sua aparência usando apenas CSS . Ainda, as mensagens são exibidas no idioma da localidade configurada no navegador do usuário. Isso pode resultar em inconsistências de idioma entre o conteúdo da página e a mensagem de erro, especialmente em sites multilíngues.

Para fornecer uma experiência de usuário mais consistente, você pode criar suas próprias mensagens de erro personalizadas usando o método JavaScript **setCustomValidity()**. Esse método permite definir uma mensagem de erro específica para cada campo, que será exibida no lugar das mensagens automáticas. Neste caso, se uma mensagem personalizada for definida, o campo é marcado como inválido, acionando a pseudo-classe CSS **:invalid**. Porém, se a mensagem for uma string vazia, o navegador considera o campo válido.

Veja um exemplo no quadro abaixo:

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Exemplo de Mensagem Customizada</title>
</head>
<body>
  <form id="myForm" action="">
    <label for="userEmail">E-mail:</label>
    <input type="email" id="userEmail" name="userEmail" required>
    <button type="submit">Enviar</button>
  </form>
  <script>
    const emailField = document.getElementById('userEmail');
```

```
    emailField.addEventListener('input', function () {
      // Verifique se o campo contém um endereço de email válido
      if (this.validity.typeMismatch) {
        this.setCustomValidity('Por favor, insira um endereço de email válido (ex: usuario@email.com).');
      } else {
        this.setCustomValidity(''); // Limpa a mensagem personalizada se não houver erros
      }
    });
  </script>
</body>
</html>
```

Veja o resultado no navegador, conforme figura abaixo:

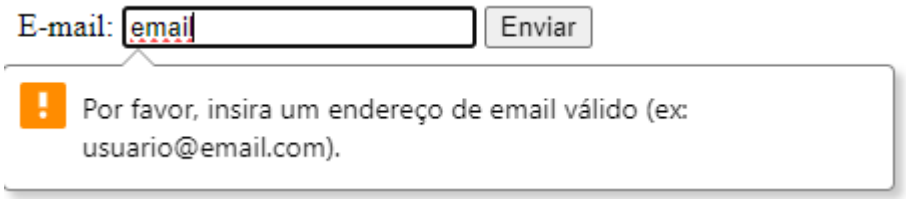

Figura 1 - Exemplo de validação de formulário com a API de validação de restrição

Descrição da imagem: No navegador, observamos um formulário com um campo de entrada de e-mail e um botão de envio. Quando o usuário digita no campo de e-mail, o JavaScript verifica se o valor inserido é um endereço de e-mail válido. Se o valor não for um e-mail válido, uma mensagem de erro personalizada aparece, dizendo "Por favor, insira um endereço de email válido (ex: usuario@email.com).". Se o valor for um e-mail válido, a mensagem de erro personalizada é removida.

Validando formulário sem usar a API de validação de restrição

Em alguns casos, como ao lidar com navegadores antigos ou widgets personalizados, você pode não conseguir ou preferir não usar a API de validação de restrição (Constraint Validation API) fornecida pelo HTML5. Nessas situações, você ainda pode usar JavaScript para validar seus formulários, embora a validação precise ser manual.

Para validar um formulário manualmente usando JavaScript, siga estas etapas:

1. **Identifique os requisitos de validação:** defina as regras para cada campo, como campos obrigatórios, tipos específicos (número, email), comprimentos mínimos/máximos e padrões.
2. **Aplique a lógica de validação:** para cada campo, escreva as regras de validação correspondentes, como:
    1. Verificar se um campo obrigatório não está vazio.
    2. Comparar os valores inseridos com expressões regulares para padrões específicos.
3. **Forneça feedback ao usuário:** adicione mensagens de erro claras próximas aos campos inválidos. Também, destaque os campos com erros usando estilos CSS (como bordas vermelhas).
4. **Impeça a submissão:** use **event.preventDefault()** para impedir que o formulário seja submetido se houver erros.

Abaixo está um exemplo de como validar manualmente um formulário:

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Validação Manual</title>
  <style>
    .error {
      color: red;
    }

    input.invalid {
      border: 2px solid red;
    }
  </style>
</head>
<body>
  <form id="manualForm">
    <label for="username">Nome de Usuário:</label>
    <input type="text" id="username" name="username">
    <span id="usernameError" class="error"></span>
    <br>
    <label for="email">Email:</label>
    <input type="email" id="email" name="email">
    <span id="emailError" class="error"></span>
    <br>
    <input type="submit" value="Enviar">
  </form>
  <script>
    document.getElementById('manualForm').addEventListener('submit', function (event) {
      let valid = true;
```

```
      // Limpar mensagens de erro
      document.getElementById('usernameError').textContent = '';
      document.getElementById('emailError').textContent = '';
      document.getElementById('username').classList.remove('invalid');
      document.getElementById('email').classList.remove('invalid');

      // Validar Nome de Usuário
      const username = document.getElementById('username').value;
      if (username.length < 4) {
        valid = false;
        document.getElementById('usernameError').textContent = 'O nome de usuário deve ter pelo menos 4 caracteres.';
        document.getElementById('username').classList.add('invalid');
      }
```

```
      // Validar Email
      const email = document.getElementById('email').value;
      const emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
      if (!emailPattern.test(email)) {
        valid = false;
        document.getElementById('emailError').textContent = 'Por favor, insira um email válido.';
        document.getElementById('email').classList.add('invalid');
      }

      // Impedir envio se houver erros
      if (!valid) {
        event.preventDefault();
      }
    });
  </script>
</body>
</html>
```

Veja o resultado no navegador, conforme figura abaixo:

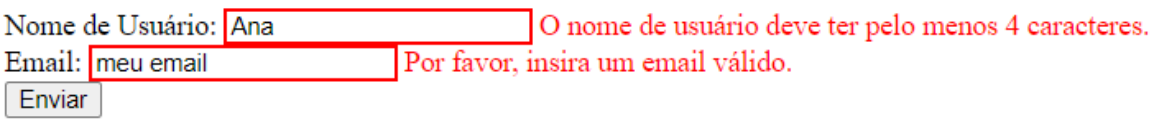

Figura 2 - Exemplo de validação de formulário sem usar a API de validação de restrição

Descrição da imagem: No navegador, observamos um formulário que contém um campo de texto para o nome de usuário e um campo de e-mail, cada um com uma área para exibir mensagens de erro, e um botão de envio do formulário. Ao tentar enviar o formulário, se o nome de usuário tiver menos de 4 caracteres, uma mensagem de erro informará que o nome de usuário deve ter pelo menos 4 caracteres. Ainda, se o e-mail não for válido, uma mensagem de erro informará que o e-mail precisa ser válido. Assim, o formulário não será enviado até que ambos os campos estejam corretos.

◄ 6.2.1 Atributos de validação embutidos do HTML5

Seguir para...

6.2.3 Bibliotecas e frameworks ►