02

# Sugestão de email

### Transcrição

Em `index.html`, incluiremos um valor no `input` do CEP, deixando-o assim:

```
<div class="contatoCampo">
    <label for="cep">CEP: </label>
    <input id="cep" type="text" maxlength="8" placeholder="Exemplo: 04101-300" required value="0410
    </div>
```

Quando utilizamos o "Tab" para navegar na parte do formulário, o NVDA lê "gmail.com? linque", sendo que o texto de validação após o campo de email é "Você quis dizer gmail.com?". Fica meio estranho para o usuário que acabou de errar o preenchimento do email, então precisamos corrigi-lo.

Uma opção seria deixarmos isto sob responsabilidade da equipe de back-end, porém poderemos nós mesmos realizar esta validação. Há uma *lib* muito interessante que usamos ou usávamos no site da Caelum, a mailcheck (http://github.com/mailcheck/mailcheck), que checa erros de digitação de emails e faz sugestões de correções.

Poderemos instalá-la via `bower` ou `npm`, porém para este caso em específico usaremos um pronto para incluirmos em um novo arquivo. O caminho para isto é `mailcheck / src / mailcheck.js`, e não há problema copiarmos a versão minificada.

Colaremos o código copiado em um arquivo novo, que salvaremos como `mailcheck.js`, após o qual teremos que puxá-lo no HTML, acrescentando a linha a seguir às demais, sem esquecer de salvá-lo depois:

```
<script src="js/mailcheck.js"></script>
```

Para o instalarmos e o configurarmos, basta copiar o código que aparece entre as *tags* `<script>` na página do GitHub, e colá-lo em um novo arquivo ( `mailcheck-config.js` ), que também chamaremos no fim do arquivo `index.html` junto dos outros `script` s.

Neste código, pode-se configurar os domínios para as tentativas, em `var domains = ['gmail.com', 'aol.com', 'outlook.com', 'alura.com.br'];`, em que a lista de prioridades segue a ordem em que aparecem no código. Pode-se entender melhor acessando o GitHub deles.

Sua configuração inicial foi feita com jQuery, teríamos que puxá-lo só para isso...?

Não! Mais abaixo, na mesma página do GitHub, existe "Usage without jQuery", em que é disponibilizado o código sem esta biblioteca. Em `mailcheck-config.js`, deletaremos o seguinte código:

```
$('#email').on('blur', function() {
    $(this).mailcheck({
        domains: domains,                                          // optional
        secondLevelDomains: secondLevelDomains, // optional
        topLevelDomains: topLevelDomains,                // optional
        distanceFunction: superStringDistance,    // optional
        suggested: function(element, suggestion
            // callback code
```

```
        // callback code
    },
    empty: function(element) {
        // callback code
    }
    });
});
```

e colaremos em seu lugar:

```
document.querySelector('#email').addEventListener('blur', function() {
    Mailcheck.run({
        email: yourTextInput.value,
        domains: domains,                       // optional
        topLevelDomains: topLevelDomains,       // optional
        secondLevelDomains: secondLevelDomains, // optional
        distanceFunction: superStringDistance,  // optional
        suggested: function(suggestion) {
            // callback code
        },
        empty: function() {
            // callback code
        }
    });
});
```

Resumidamente, o `mailcheck.js` funciona da seguinte forma: digamos que você queira incluir uma mensagem caso o campo a ser preenchido esteja vazio. Como já utilizamos o `required` do HTML5, poderemos remover a linha abaixo:

```
        empty: function() {
            // callback code
        }
```

Vamos abrir `mailcheck-config.js` em que, no fim do código, na parte `email`, se encontra `yourTextInput`, o qual se refere ao campo de email, ou seja, `document.querySelector('#email')`. Já aprendemos que, se vai haver repetição, é melhor criarmos uma variável no mesmo arquivo, a ser chamada depois.

Além disso, em `mailcheck.js`, para termos acesso a toda a sugestão ( `full` ), precisaremos puxar `suggestion`, segundo parâmetro de `suggested`. Voltaremos a `mailcheck-config.js` e veremos que ele está com apenas um parâmetro:

```
var campoEmail = document.querySelector('#email');

// variáveis domains, secondLevelDomains e topLevelDomains. superStringDistance pode ser deletada, p

campoEmail.addEventListener('blur', function() {
    Mailcheck.run({
        email: campoEmail.value,
        domains: domains,                       // optional
        topLevelDomains: topLevelDomains,       // optional
        secondLevelDomains: secondLevelDomains, // optional
        distanceFunction: superStringDistance,  // optional
        suggested: function(suggestion) {
            console.log(suggestion.full);
```

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

Vamos salvar e voltar ao navegador para usar o atalho "Ctrl + Shift + I". O que implementamos não funciona, pois não criamos `suggestion`. Em `index.html`, onde se localiza a parte de email, o responsável pelo texto "Você quis dizer gmail.com?" a partir de então é o JavaScript, portanto o código ficará da seguinte forma:

```
<div class="contatoCampo contatoCampo--erro">
    <label for="email">Email: </label>
    <input type="email" id="email" class="contatoCampo--validouErro" placeholder="Email..." requirec
    <span id="sugestao" class="contatoCampo-msg contatoCampo-msg--erro">
        Você quis dizer <a href="#">gmail.com?</a>
    </span>
</div>
```

Salvaremos e voltaremos a `mailcheck-config.js`, em que pegaremos esta `id` que acabamos de criar.

```
var campoEmail = document.querySelector('#email')
var sugestao = document.querySelector('#sugestao')
```

Salvaremos, abriremos o navegador, atualizaremos e testaremos o Console. Tudo parece estar bem configurado. Testaremos `contato@gnail.com`, isto é, com erro de digitação, para verificar o que acontece. No Console, exibe-se "contato@gmail.com". A sugestão é feita direitinho.