



Desenvolvimento de Sistemas

Preparação para integração de *front-end* com regras *back-end* por meio de formulários e validações

Introdução

O desenvolvimento *front-end* concentra-se na codificação e criação de elementos e recursos de um *site* que serão vistos pelo usuário. Você também pode pensar no *front-end* como o “lado do cliente” de um aplicativo. Imagine que você é um desenvolvedor *front-end*. Isso significa que seu trabalho é codificar e dar vida aos elementos visuais de um *site*.

Já o desenvolvimento *back-end* concentra-se no lado do *site* que os usuários não podem ver. Você também pode se referir ao *back-end* como o “lado do servidor” de um *site*. Por exemplo, imagine que você esteja executando um *site* de mídia social. Você precisa de um local acessível para armazenar todas as informações de seus usuários em um banco de dados, como Oracle, SQL Server, MySQL, entre outros. O *back-end* ajudará a gerenciar esse banco de dados, bem como o conteúdo do *site* armazenado nele. Isso garante que os elementos *front-end* em seu *site* de mídia social continuem a funcionar corretamente enquanto os usuários navegam pelo conteúdo carregado e outros perfis de usuário.

A interação entre o *front-end* e o *back-end* é possível por meio de solicitações HTTP (*hyper text transfer protocol*). Funcionalidades como atualizar um usuário novo/antigo em um banco de dados, publicar uma postagem de *blog*, excluir uma imagem do seu perfil e atualizar sua biografia exigem uma interação entre um servidor e um cliente para modificar os dados.

Ao criar aplicativos na *web*, frequentemente você interage com dados armazenados em diferentes bancos de dados e servidores. Para interagir com servidores no *front-end*, você pode usar diferentes métodos HTTP para solicitar dados, conhecidos também por *request* ou requisições.

Simulando um *back-end*



Nos exemplos ao decorrer deste conteúdo, será utilizado um programa para simular um servidor de *back-end* usando o programa JSON Server para demonstrar como funcionam as solicitações HTTP, verificando se realmente serão recebidas pelo *back-end* e retornadas ao *front-end*.

Na realidade, o JSON Server é usado para criar um serviço *web*, aplicando um protocolo chamado REST (*representational state transfer*), muito usado para comunicação *web*. Nesse momento, não serão dados muitos detalhes sobre *web-services*, pois o foco serão os testes de requisições.

Instalando o NPM – Node.js

Para começar a usar o JSON Server, é necessário que você tenha instalado o NPM (*node package manager*), que é um gerenciador de pacotes do Node.js.

Com o NPM, você pode gerenciar dependências do seu projeto, acessar o repositório do NPM e ter acesso a diversas bibliotecas e diversos *frameworks* JavaScript, permitindo que sejam instalados/desinstalados.

O NPM está incluído no Node.js. Prossiga estas etapas para instalá-lo:

- ◆ Acesse o *site* oficial do Node.js, pesquisando por “Node.js” em seu buscador.
- ◆ Conforme mostra a imagem a seguir, o *site* reconhece o sistema operacional automaticamente (Windows, neste caso) e sugere duas versões para *download*: LTS e Current. Priorize o *download* de versões LTS (*long-term support*, ou suporte de longo prazo), que são as versões mais estáveis com mais suporte.

nodejs.org/en/download/

HOME | ABOUT | DOWNLOADS | DOCS | GET INVOLVED | SECURITY | CERTIFICATION | NEWS

Downloads

Latest LTS Version: 18.14.2 (includes npm 9.5.0)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

LTS
Recommended For Most Users

Current
Latest Features

Windows Installer
node-v18.14.2-x64.msi

macOS Installer
node-v18.14.2.pkg

Source Code
node-v18.14.2.tar.gz

Windows Installer (.msi)
Windows Binary (.zip)
macOS Installer (.pkg)
macOS Binary (.tar.gz)
Linux Binaries (x64)
Linux Binaries (ARM)
Source Code

32-bit	64-bit
32-bit	64-bit
64-bit / ARM64	
64-bit	ARM64
64-bit	
ARMv7	ARMv8
node-v18.14.2.tar.gz	

about:blank

2/33

Figura 1 – Instalando NPM – Node.js

Fonte: Senac EAD (2023)



- ◆ Após a instalação do Node.js, você pode verificar tanto a versão do Node.js quanto a versão do NPM instaladas, executando os seguintes comandos: **node -v** e **npm -v**.

```
node -v  
v18.14.2
```

```
npm -v  
9.5.0
```

Instalando o servidor JSON Server

Para começar a usar o JSON Server, instale o pacote usando o NPM:

```
npm install -g json-server
```

Ao adicionar a opção **-g**, garante-se que o pacote seja instalado globalmente em seu sistema.

Criando um arquivo de banco de dados com JSON

Crie um arquivo JSON (*JavaScript object notation*, ou notação de objetos JavaScript) com alguns dados conforme sua necessidade. Por exemplo, você precisa de alguns dados JSON com informações de um cadastro como *login*, *senha* e *id*. Então, crie um arquivo chamado **dados.json** com as seguintes informações:

```
{  
  "cadastros": [  
    {  
      "login": "senac",  
      "senha": "senacead",  
    }  
  ]  
}
```

```
{
  "id": 1
},
{
  "login": "aluno",
  "senha": "123",
  "id": 2
}
]
```

JSON é uma formatação leve de troca de dados utilizado para transferir informações entre sistemas.

Os dados contidos em um arquivo no formato JSON devem ser estruturados por meio de uma coleção de pares com nome e valor ou serem uma lista ordenada de valores.

Seus elementos devem conter:

CHAVE

Corresponde ao identificador do conteúdo. Por isso, deve ser uma *string* delimitada por aspas.

VALOR

Representa o conteúdo correspondente e pode conter os seguintes tipos de dados: *string*, *array*, *object*, *number*, *boolean* ou *null*.

Executando o servidor

Inicie o servidor JSON executando o seguinte comando:

```
json-server --watch dados.json
```

Como parâmetro, é preciso passar o arquivo que contém a estrutura JSON (**dados.json**). Além disso, está sendo usado o parâmetro **--watch**. Ao usar esse parâmetro, garante-se que o servidor seja iniciado no modo de observação, o que significa que ele observa alterações de

arquivo e atualiza a API (*application programming interface*, ou interface de programação de aplicação) automaticamente.

Ao rodar o comando acima, o módulo do JSON Server inicia e gera uma rota que o servidor de dados possa ser acessado, por meio da URL (*uniform resource locator*) <http://localhost:3000/cadastros>. É nesse endereço que deverão ser realizadas as solicitações de **GET** e **POST**.

```
Loading dados.json
Done

Resources
http://localhost:3000/cadastros

Home
http://localhost:3000
```

Testando o serviço

Você pode testar se tudo está certo usando no *browser* a URL <http://localhost:3000/cadastros>.



Figura 2 – Acessando serviço de cadastros
Fonte: Senac EAD (2023)

Protocolo HTTP – GET e POST



O HTTP é o protocolo responsável pela comunicação de *sites* na *web*. Quando um *site* é acessado, esse é o protocolo utilizado. O HTTP oferece suporte a muitos métodos para transferir dados ou executar qualquer operação no servidor, como GET, POST, PUT, DELETE, PATCH, entre outros.

Formulários em HTML (*hyper text markup language*) podem usar qualquer método especificando **method="POST"** ou **method="GET"** (padrão) no elemento `<form>`. O método especificado determina como os dados do formulário são enviados ao servidor. O servidor recebe os dados enviados, realiza o processamento e retorna para o cliente.

Quando o método é **GET**, os parâmetros são passados no cabeçalho da requisição, ou seja, todos os dados do formulário podem ser vistos pela URI (*uniform resource identifier*). Já no método **POST**, os dados do formulário são enviados como parâmetros no corpo da requisição da solicitação HTTP.

Para exemplificar, no código a seguir, criou-se um formulário com campo de texto como *login* e *senha*. Inicialmente, será usado o método GET e será incluída uma ação **action="http://localhost:3000/cadastros"** para simular onde os dados serão enviados após clicar no botão **Enviar** do formulário.

```
<!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>Formulário</title>
</head>
<body>
  <h1>Cadastro de Usuários
  <form action="http://localhost:3000/cadastros" method="GET">
    Login: <input type="text" name="login" /> <br>
    Senha: <input type="text" name="senha" /> <br>
    <input type="submit" value="Enviar"/>
  </form>
</body>
</html>
```

No formulário, ao dados estão sendo informados como *login* “supervisor” e senha “321”.

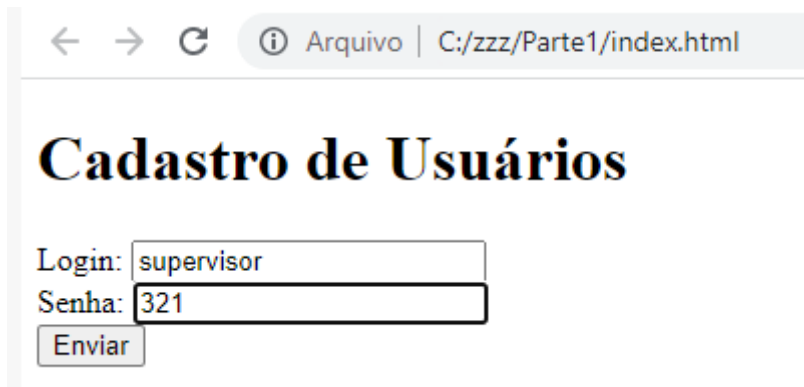


Figura 3 – Tela de *login*
Fonte: Senac EAD (2023)

Após clicar no botão **Enviar**, o evento **submit** é disparado. É possível ver, na imagem a seguir, que os dados passados no método **GET** ficam bem visíveis na barra de endereço, o que pode comprometer a segurança. Como o método GET retorna dados, então é exibido um *array* vazio.

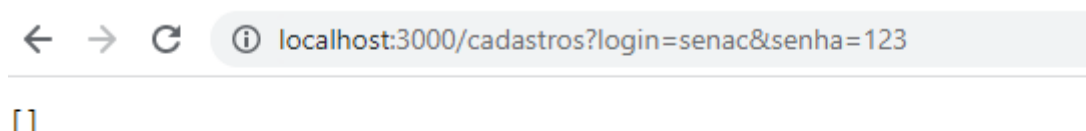


Figura 4 – Tela de envio de dados (requisição GET)
Fonte: Senac EAD (2023)

Veja, agora, o mesmo formulário, porém alterado o método para **POST**:

```
<!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>Formulário</title>
</head>
<body>
  <h1>Cadastro de Usuários</h1>
  <form action="http://localhost:3000/cadastros" method="POST">
    Login: <input type="text" name="login" /> <br>
    Senha: <input type="text" name="senha" /> <br>
    <input type="submit" value="Enviar"/>
  </form>
</body>
</html>
```

Ao informar os mesmos dados do exemplo anterior e após clicar no botão **Enviar**, ao contrário do GET, o POST envia os parâmetros no corpo da requisição do HTTP, escondendo os dados da URI. Observe que o servidor recebe os dados, processa e retorna ao *front-end* os dados cadastrados.

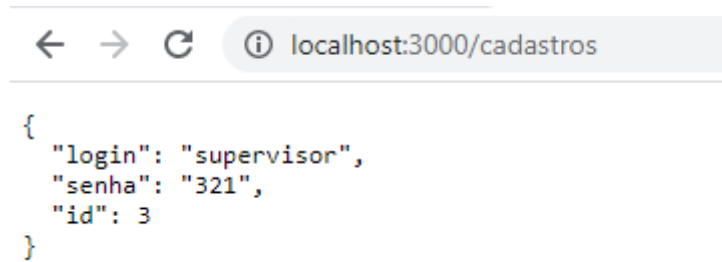


Figura 5 – Tela de envio de dados (requisição POST)

Fonte: Senac EAD (2023)

Por isso, é fundamental que o valor dos atributos **name** das *tags* no HTML sejam exatamente iguais ao do servidor.

Exemplo: o campo *login* no *back-end* (servidor) não tem acento, é no singular e todo em minúsculo. Então, não se pode ter na *tag* do HTML do formulário “Login”, “LOGIN”, “Logín”. É necessário ser exatamente igual. Isso evita que o servidor esteja esperando informação dos campos, porém não receba em razão de o nome da *tag* do campo ser trocado ou inexistente no HTML do cadastro. Essa dica pode evitar muita “dor de cabeça” e até mesmo horas de trabalho.

Preparando para os próximos exemplos

Antes de continuar, configure um novo serviço (ou uma nova URL) no JSON Server para realizar os próximos testes propostos.

Primeiro, crie um novo arquivo de dados para o JSON Server. Na pasta, crie o arquivo **db.json**. Nele, inclua o seguinte conteúdo:

```
{
  "noticias": [
    {
      "titulo": "Java",
      "conteudo": "Linguagem Java",
      "id": 1
    },
    {
      "titulo": "JSON SERVER",
      "conteudo": "Utilizado para API ficticia",
      "id": 2
    }
  ]
}
```



```
}  
]  
}
```

Nesse caso, um serviço que lida com “notícias” está sendo configurado, ou seja, está sendo simulado um *back-end* em que há cadastro de manipulação de dados de notícias (informações como título e conteúdo são registradas).

Após editar o arquivo, no *prompt* de comando use o seguinte comando:

```
json-server --watch db.json
```

```
Loading db.json  
Done  
  
Resources  
http://localhost:3000/noticias  
  
Home  
http://localhost:3000
```

Você pode testar se tudo está certo usando no *browser* a URL <http://localhost:3000/noticias>. Nos próximos exemplos, essa URL será usada para simular as requisições.



```
< > ↻ ⓘ localhost:3000/noticias  
  
[  
  {  
    "titulo": "Java",  
    "conteudo": "Linguagem Java",  
    "id": 1  
  },  
  {  
    "titulo": "JSON SERVER",  
    "conteudo": "Utilizado para API ficticia",  
    "id": 2  
  }  
]
```

Figura 6 – Acessando serviço de notícias
Fonte: Senac EAD (2023)

Solicitações GET e POST



Inicialmente, é preciso criar uma página em HTML que o navegador possa exibir. Adote-a com o nome de **index.html**. A página HTML não tem conteúdo visual, ela será utilizada apenas para carregar o arquivo **js.js** para que seja possível ver as requisições e as respostas no console do navegador.

```
<!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>Java Script HTTP Requests
</head>

<body>
  <script src="js.js"></script>
</body>

</html>
```

Você também deve ter o console do navegador aberto para que possa ver os resultados de suas solicitações HTTP. Isso geralmente é feito clicando **F12** e selecionando a aba **Console**.

Para seguir os exemplos, é necessário iniciar o serviço do JSON Server. Volte ao tópico anterior para rever como fazer isso e ativar a URL `<http://localhost:3000/noticias>`. É nesse endereço que deverão ser realizadas as solicitações de **GET** e **POST**.

A seguir, você começará a enviar solicitações GET e POST usando a API Fetch e a biblioteca jQuery.

API Fetch

Fetch é uma API JavaScript nativa simplificada e moderna, usada para fazer solicitações HTTP. Ela vem com suporte integrado para promessas e melhora na sintaxe. Como uma API construída tendo em mente as necessidades de aplicativos e desenvolvedores modernos, o

Fetch tornou-se uma das formas mais populares de enviar solicitações HTTP em JavaScript atualmente.

Como enviar uma solicitação GET em JavaScript usando a API Fetch

Enviar uma solicitação GET usando a API Fetch requer apenas a URL. Isso então retorna uma promessa que você pode acessar usando o método **then()**. Portanto, ao chamar o método **fetch**, você receberá uma promessa de resposta.

Promessa é um objeto em JavaScript que permite a execução de processamentos de forma assíncrona dentro do seu código, uma vez que é definido como um objeto em que é possível guardar valores que poderão ser usados em outro momento no seu código enquanto você executa outras tarefas.

Veja, agora, no código no arquivo **js.js**, um exemplo da API Fetch:

```
fetch("http://localhost:3000/noticias")
  .then((response) => response.json())
  .then((json) => console.log(json));
```

Nesse código, passou-se a URL <http://localhost:3000/noticias> para o método **fetch**. Em seguida, a resposta do servidor foi acessada usando o método **then**. A resposta foi convertida em um objeto JSON usando o método **response.json()**.

Depois de obter a resposta, usa-se outro método **then** para exibir os dados no console do navegador pelo método **console.log(json)**.

```
(2) [{...}, {...}]
0: {titulo: 'Java', conteudo: 'Linguagem Java', id: 1}
1: {titulo: 'JSON SERVER', conteudo: 'Utilizado para API ficticia', id: 2}
length: 2
```

Como enviar uma solicitação POST em JavaScript usando API Fetch

O método **fetch** tem um segundo parâmetro que permite especificar o **method** tipo de solicitação a ser enviada e o **body** dados a serem enviados. Esse parâmetro **method** permite enviar solicitações do tipo POST.

Agora que os parâmetros foram aprofundados, veja, no código no arquivo **js.js**, o exemplo a seguir:

```
fetch("http://localhost:3000/noticias", {
  method: "POST",
  body: JSON.stringify({
    titulo: "API Fetch",
    conteudo: "Conhecendo a API Fetch",
  }),
  headers: {
    "Content-type": "application/json; charset=UTF-8",
  },
})
.then((response) => response.json())
.then((json) => console.log(json));
```

Nesse código, foram adicionadas opções de solicitação no parâmetro **método**. Usou-se a propriedade **method** para especificar o tipo de solicitação POST. Na propriedade **body**, serão especificados os dados a serem enviados ao servidor. Note que se adotou o método **JSON.stringify**, que converte valores em JavaScript para uma *string* em JSON. Por fim, a propriedade **headers** é utilizada para especificar que seriam enviados dados no formato JSON ao servidor.

Semelhante à solicitação por GET, ao realizar uma solicitação por POST, a resposta será retornada usando o método **then()**. Se tudo ocorrer bem, você terá a notícia exibida no console do navegador.

```
{titulo: 'API Fetch', conteudo: 'Conhecendo a API Fetch', id: 3}
conteudo: "Conhecendo a API Fetch"
id: 3
titulo: "API Fetch"
```

jQuery

Como já estudado, o uso de jQuery simplifica o processo de codificação em JavaScript, tendo uma extensa documentação de métodos e diversos *plugins* que permitem estender as funcionalidades do jQuery.

No envio de solicitações em GET e POST, o uso de jQuery simplifica o processo de obtenção dos dados de servidores, tornando a sintaxe mais curta e direta. Veja, a seguir, exemplos de solicitações por GET e POST usando jQuery.

Como enviar uma solicitação GET em JavaScript usando jQuery

No arquivo **index.html** que está sendo usado nos exemplos, não se pode esquecer de adicionar a biblioteca jQuery/JavaScript dentro da *tag* **<head>** do HTML. Além disso, será adicionada também no *body* a **<div id="news"></div>**.

Veja como ficou o código **index.html**:

```
<!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" />
  <script src="https://code.jquery.com/jquery-3.4.1.js"></script>
  <title>Java Script HTTP Requests</title>
</head>

<body>
  <div id="news"></div>
  <script src="js.js"></script>
</body>

</html>
```

O jQuery fornece o método **\$.get()** para enviar solicitações GET aos servidores. O método aceita dois parâmetros – a URL para o servidor e uma função de retorno de chamada que é executada se a solicitação for bem-sucedida. Veja, agora, no código no arquivo **js.js**, como realizar essa solicitação.

```
$.get("http://localhost:3000/noticias", function(data, status){  
  for (i = 0; i < data.length; i++) {  
    $('#news').append("<ul><li>" + data[i].titulo + " - " + data[i].conteudo + "</li></ul>");  
  }  
});
```

Como pode ser visto no código acima, o método **\$.get()** recebeu a URL `<http://localhost:3000/noticias>` e uma função de **callback** anônima como seus parâmetros.

Por meio da função de retorno de chamada, você pode acessar os dados da solicitação (**data**) e o *status* da solicitação (**status**). Sendo assim, é possível percorrer agora todos os itens retornados (em um **for**) e criar uma lista de notícias com título e conteúdo, usando a função **\$('#news').append()**.

Veja a lista de notícias exibidas na página **index.html**:

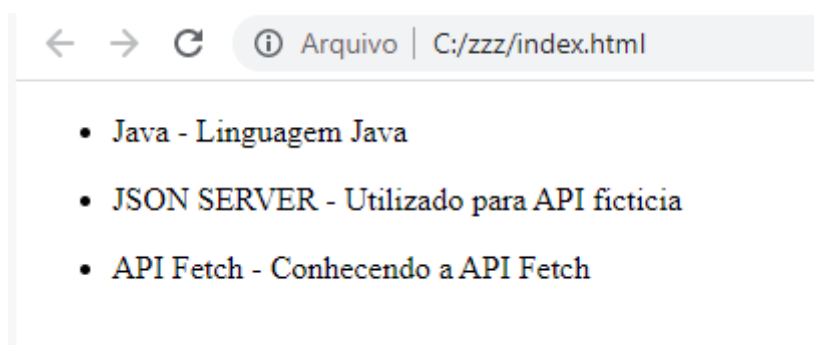


Figura 7 – Listando notícias (solicitação GET usando jQuery)
Fonte: Senac EAD (2023)

Como enviar uma solicitação POST em Java Script usando jQuery

Agora, crie um arquivo chamado **cadastro.html**. Nele, haverá um formulário de cadastro de notícias com os campos **título** e **conteúdo** e a `<div id="result"></div>`, que receberá uma mensagem de confirmação de cadastro sem a necessidade de atualizar a tela.

```
<!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" />
  <script src="https://code.jquery.com/jquery-3.4.1.js"></script>
  <title>Java Script HTTP Requests</title>
</head>

<body>
  <h1>Cadastro de notícias</h1>
  <form method="post" action="">
    <label>Título: <input type="text" name="titulo" id="titulo"></label><br><br>
    <label>Conteúdo: <textarea cols="50" name="conteudo" id="conteudo"></textarea></label><br><br>
    <input type="submit" value="Cadastrar">
  </form>
  <div id="result"></div>
  <script src="cadastro.js"></script>
</body>

</html>
```

Já no arquivo **cadastro.js**, há o código do método responsável por receber os dados do formulário de cadastro e realizar uma solicitação por POST utilizando jQuery.

```
1 $(document).ready(function () {
  $("form").submit(function (event) {

    // Impede o envio normal do formulário
5    event.preventDefault();

    //Serializa os valores de controle de formulário enviados
    //para serem enviados ao servidor da Web com a solicitação
9    var formValues = $(this).serialize();

11    $.ajax({
      type: 'POST',
      url: 'http://localhost:3000/noticias',
      data: formValues
    })
  })
})
```

```
16     .done(function (data) {  
        // enviar mensagem de registro salvo  
        $('#result').append("Registro salvo!");  
    })  
20     .fail(function (msg) {  
        // caso a solicitacao de POST tenha falhado  
        alert("Falha no POST" + msg);  
    });  
});  
});
```

Veja os detalhes do código acima:

1. Quando ocorrer a submissão do cadastro de notícias, o evento **preventDefault()** subscrive o evento **submit** do formulário para que ocorra agora a submissão pelo jQuery (linha 5).
2. O próximo passo é obter os dados do formulário (linha 9). É preciso, agora, obter os dados de todos os campos e os respectivos conteúdos. Em vez de atribuir campo a campo, adote o método **\$(this).serialize()**, que serializará todos os campos e valores do formulário de modo automático e dinâmico.
3. O método **\$.ajax()** (linha 11) tem os parâmetros **type** (que nesse exemplo é POST), **url** (que especifica a URL do servidor) e **data** (que são todos os dados a serem enviados).
4. A função de **callback done** (linha 16) é executada quando a requisição é finalizada com sucesso e permite tratar o retorno do servidor que é recebido por meio do parâmetro **data**. No exemplo, está sendo enviada uma mensagem de forma assíncrona informando “Registro salvo” para na **tag result** do formulário.
5. Por fim, há a função de **callback fail** (linha 20), que, por sua vez, será executada quando ocorrer algum erro na requisição. De modo semelhante, a mensagem de erro é recebida pelo parâmetro **msg** e exibida utilizando a função de alerta do JavaScript, **alert**.

Veja, a seguir, o exemplo de solicitação por POST em jQuery usando AJAX de um cadastro de notícias que retorna do servidor uma confirmação de registro salvo, sem a necessidade de atualizar a tela.



← → ↻ ⓘ Arquivo | C:/zzz/cadastro.html

Cadastro de notícias

Título:

Conteúdo:

Registro salvo!

Figura 8 – Solicitação assíncrona por POST

Fonte: Senac EAD (2023)

Validações

A validação de formulário é de vital importância para a segurança de uma aplicação, bem como para a usabilidade dela. O processo de validação avalia se o valor de entrada está no formato correto antes de enviá-lo ao servidor. Por exemplo, se houver um campo de entrada para um endereço de *e-mail*, o valor certamente deve conter um endereço de *e-mail* válido, deve começar com uma letra ou um número, seguido do símbolo do @, e deve terminar com um nome de domínio.

Sobre validações de formulários, é importante ressaltar que elas devem ocorrer tanto do lado do *front-end* quanto no lado do *back-end*. No *back-end*, a validação é mais segura, pois ela não pode ser desabilitada, comparada com a validação em *front-end* via JavaScript.

No tópico a seguir, serão abordadas duas formas de validação de formulários no *front-end*: por meio dos atributos do HTML5 e pelo *plugin* da validação jQuery Validation.

Usando os atributos-padrão em HTML5

O HTML5 fornece atributos-padrão aplicados somente no elemento **input**. Os mais usados são os elencados a seguir.

Required

Especifica os campos de entrada que precisam ser preenchidos antes de enviar o formulário.

Minlength e maxlength

Especifique o comprimento mínimo e máximo esperado de uma string.

Min e max

Especifique os valores mínimo e máximo dos números.

Type

Especifica que tipo de dados são necessários para campos de entrada específicos, por exemplo, data, número, nome, *e-mail* e assim por diante.

Pattern

Permite definir a própria regra para validar um valor de entrada usando expressões regulares.

Veja alguns exemplos de validação:

Required: para campos genéricos, como nome, sobrenome e idade de preenchimento obrigatório.

```
<p>Nome: <br> <input type="text" name="nome" required></p>
<p>Sobrenome: <br> <input type="text" name="sobrenome" required></p>
<p>Idade: <br> <input type="number" name="idade" required></p>
```

Data: usando as propriedades, **type** sendo **date**, **maxlength**, **min** e **max**.

```
<p>Data de nascimento: <br><input type="date"          required maxlength="10"
name="datanasc"      min="1900-01-01" max="2025-12-31" /></p>
```

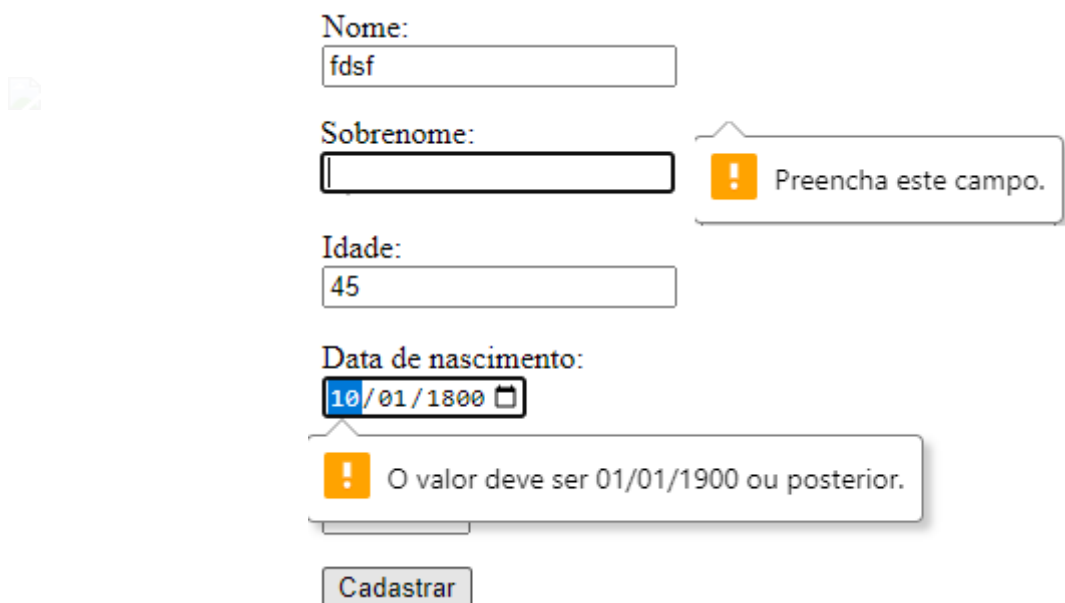
Hora: usando as propriedades, **type** sendo **time**, **maxlength**.

```
<p>Hora: <br><input type="time" required maxlength="8" name="hora" /></p>
```

Veja, agora, um exemplo para validação dos campos nome, sobrenome, idade, data de nascimento e hora. Para isso, crie um arquivo **validar.html**.

```
<!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>Validações</title>
</head>
<body>
  <form action="" method="POST">
    <p>Nome: <br> <input type="text" name="nome" required></p>
    <p>Sobrenome: <br> <input type="text" name="sobrenome" required></p>
    <p>Idade: <br> <input type="number" name="idade" required></p>
    <p>Data de nascimento: <br><input type="date" required maxlength="10" name="data
nasc" min="1900-01-01" max="2025-12-31" /></p>
    <p>Hora: <br><input type="time" required maxlength="8" name="hora" /></p>
    <input type="submit" value="Cadastrar">
  </form>
</body>
</html>
```

Veja, a seguir, a execução no navegador de Internet.



Nome:

Sobrenome:

Idade:

Data de nascimento:

! Preencha este campo.

! O valor deve ser 01/01/1900 ou posterior.

Cadastrar

Figura 9 – Formulário de cadastro com validação

Fonte: Senac EAD (2023)

Agora, **defina uma regra para adicionar expressões regulares no atributo pattern**. Nesse caso, será especificado que o nome de usuário deve consistir apenas em letras minúsculas, não permitindo letras maiúsculas, números ou outros caracteres especiais. Além disso, o tamanho do nome do usuário não deve ter mais que 15 caracteres. A expressão regular dessa regra pode ser expressa como **[a-z]{1,15}**.

```
<p>Username:
  <input type="text" required name="username" placeholder="Userna
me" pattern="[a-z]{1,15}$"></input></p>
```

Apenas letras:

```
<p>Endereço: <br><input type="text" required name="endereco" pattern="[a-z
\s]+$" /></p>
```

Apenas números:

```
<p>Número: <br><input type="text" requiredname="numeros" pattern="[0-9]+$"/>
</p>
```

Telefone:

```
<p>Telefone: <br><input type="text" required maxlength="15" name="telefone" pattern="\([0-9]{2}\) [0-9]{4,6}-[0-9]{4}$" /></p>
```

E-mail:

```
<p>E-mail: <br><input type="email" required name="email" pattern="[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,4}$" /></p>
```

Moeda:

```
<p>Valor: <br><input type="text" required maxlength="15" name="valor" pattern="([0-9]{1,3}\.)?[0-9]{1,3},[0-9]{2}$" /> </p>
```

Veja, agora, o exemplo completo de validação dos campos usando expressão regular de *username*, endereço, número, telefone, *e-mail* e valor no arquivo **validar.html**.

```
<!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>Validações
</head>
<body>
  <form action="" method="POST">
    <p>Username: <br><input type="text" required name="username" placeholder="Username" pattern="[a-z]{1,15}$"></input></p>
    <p>Endereço: <br><input type="text" required name="endereco" pattern="[a-z\s]+$" />
  </p>
    <p>Número: <br><input type="text" required name="numeros" pattern="[0-9]+$" /></p>
    <p>Telefone: <br><input type="text" required maxlength="15" name="telefone" pattern="\([0-9]{2}\) [0-9]{4,6}-[0-9]{4}$" /></p>
    <p>E-mail: <br><input type="email" required name="email" pattern="[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,4}$" /></p>
    <p>Valor: <br><input type="text" required maxlength="15" name="valor" pattern="([0-
```

```
9]{1,3}\.)?[0-9]{1,3},[0-9]{2}$" /></p>
```

```
<input type="submit" value="Cadastrar">
</form>
</body>
</html>
```

Veja, a seguir, a execução no navegador de Internet.

Username:

! É preciso que o formato corresponda ao exigido.

Endereço:

! É preciso que o formato corresponda ao exigido.

Número:

! É preciso que o formato corresponda ao exigido.

Telefone:

! É preciso que o formato corresponda ao exigido.

E-mail:

! Inclua um "@" no endereço de e-mail. "asdasdas" está com um "@" faltando.

Valor:

Cadastrar

! É preciso que o formato corresponda ao exigido.

Figura 10 – Formulário de cadastro com validação
Fonte: Senac EAD (2023)

Utilize placeholders

Lembre-se de usar o **placeholder** nos campos em que você precise “dar alguma dica” para o usuário de como ele deve preenchê-lo:

```
<p>Telefone: <br><input type="text" required maxlength="15" name="telefone"
placeholder="(XX) XXXX-XXXX" pattern="\([0-9]{2}\) [0-9]{4,6}-[0-9]{4}$" /></p>
```

Telefone:

(XX) XXXX-XXXX

Figura 11 – Campo de telefone com **placeholder**

Fonte: Senac EAD (2023)

Personalizar as mensagens de validação

Felizmente, é possível personalizar a mensagem para ser mais útil, e existem algumas maneiras de fazer isso. A abordagem mais fácil é especificar o atributo **title** dentro do elemento de entrada:

```
<p>Username: <br>
<input type="text" required name="username" placeholder="Username" pattern="
[a-z]{1,15}$"
title="O nome de usuário deve conter apenas letras minúsculas no máximo co
m 15 caracteres. ex: john"></input></p>
```

Agora, o título está incluído com a mensagem-padrão:

Username:

dfdsfsdfsdf88878787



É preciso que o formato corresponda ao exigido.

O nome de usuário deve conter apenas letras minúsculas no máximo com 15 caracteres. ex: john

Figura 12 – Mensagem de erro na validação

Fonte: Senac EAD (2023)

Ainda assim, a mensagem pop-up apresenta a mensagem-padrão “É preciso que o formato corresponda ao exigido”.

Substituindo a mensagem-padrão de validação do HTML5



Agora, substitua essa mensagem-padrão por uma mensagem totalmente personalizada. Você usará um pouco de JavaScript para fazer isso.

Comece adicionando o atributo **id** ao elemento **input** e remova o **title**.

```
<p>Username: <br>
<input type="text" required id="username" name="username" placeholder="Usern
ame" pattern="[a-z]{1,15}$"></input></p>
```

Agora, no bloco em JavaScript do seu código HTML, é possível selecionar o elemento de entrada usando JavaScript e atribuí-lo a uma variável. Por último, especifique a mensagem usada quando a entrada mostra seu estado inválido.

```
var input = document.getElementById('username');

input.oninvalid = function(event) {
    event.target.setCustomValidity('O nome de usuário deve conter apenas letra
s minúsculas no máximo com 15 caracteres. ex: senac');
}
```

O evento **oninvalid** herda um objeto que contém algumas propriedades, incluindo a propriedade **target** (o elemento inválido) e a **validationMessage** que contém a mensagem de texto de erro. No exemplo anterior, substitui-se a mensagem de texto usando o método **setCustomValidity()**.

Veja, agora, o exemplo completo da validação com mensagem personalizada do campo **username**.

```
<!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>Validações</title>
</head>
<body>
  <form action="" method="POST">
    <p>Username: <br>
      <input type="text" required id="username" name="username" placeholder="Username"
pattern="[a-z]{1,15}$"></input></p>
      <input type="submit" value="Cadastrar">
    </form>

<script language="JavaScript">
  var input = document.getElementById('username');
  input.oninvalid = function(event) {
    event.target.setCustomValidity('O nome de usuário deve conter apenas letras minúsculas n
o máximo com 15 caracteres. ex: senac');
  }
</script>

</body>
</html>
```

Encontre a mensagem personalizada substituindo perfeitamente o padrão.

Username:

asdsadsad4555



O nome de usuário deve conter apenas letras minúsculas no máximo com 15 caracteres. ex: senac

Figura 13 – Personalizando a mensagem-padrão de erro

Fonte: Senac EAD (2023)

Validação com jQuery Validation

Usar o *plugin* do jQuery Validation para validar formulários serve a muitos propósitos. Ele oferece recursos adicionais, como exibir facilmente mensagens de erro personalizadas e adicionar lógica condicional à validação de formulário HTML. As condições de validação podem ser adicionadas, removidas ou modificadas a qualquer momento com facilidade.

Antes de começar a usar o *plugin* em seus campos, é preciso incluir os arquivos necessários. Por isso, crie o arquivo **registrar.html** e inclua a biblioteca **jQuery** e o *plugin* **jQuery Validation** dentro da *tag head* do HTML.

O *link* do *plugin* do jQuery Validation poderá ser obtido diretamente no *site* (digite “jQuery Validation” em seu buscador), na opção “Latest files on jsDelivr CDN”. No momento da escrita deste conteúdo, encontra-se na versão 1.19.5, conforme a figura a seguir.



Figura 14 – Site do *plugin* jQuery Validation

Fonte: Senac EAD (2023)

Após obter o *link* do *plugin*, será preciso criar um formulário de cadastramento com os campos *nome*, *e-mail* e senha. Além disso, crie o arquivo **validar.js** para começar a validar o formulário com o método **validade**.

```
<!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" />
  <script src="https://code.jquery.com/jquery-3.4.1.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/jquery-validation@1.19.5/dist/jquery.validate.js"></script>

  <title>Validando formulários com JQuery Validation</title>
</head>

<body>
  <h1>Validando formulários com JQuery Validation</h1>
  <form method="post" action="" id="frmRegistro">
    <p>Nome: <br><input type="text" name="nome" id="nome" required></p>
    <p>E-mail: <br><input type="email" name="email" id="email" required></p>
    <p>Senha: <br><input type="text" name="senha" id="senha" required></p>
    <input type="submit" value="Cadastrar">
  </form>

  <div id="result"></div>
  <script src="validar.js"></script>
</body>

</html>
```

Para começar a validar o formulário com esse *plugin*, basta adicionar o seguinte código JavaScript no arquivo **validar.js**:

```
$("#frmRegistro").validate();
```

Como é possível ver no código acima, **#frmRegistro** refere-se ao formulário, no qual declara-se a *tag* **id="frmRegistro"**, portanto o *plugin* **Validate** está no controle do formulário. Como definiu-se um único método de validação, o *plugin* entende que valida cada campo com base em suas características. Por exemplo: o campo *e-mail* está marcando em HTML com **type="email"** e **"required"**, então a validação permitirá que apenas endereços de *e-mail* válidos e obrigatórios sejam introduzidos. Se você não definir como obrigatório para o campo de *e-mail*, então a validação aceitará apenas entrada de *e-mail* válida, não obrigatória.

Para visualizar a validação, acesse no seu navegador a página **registrar.html** e tente enviar o formulário clicando no botão **Cadastrar**. Você deve obter um resultado conforme a imagem a seguir:



← → ↻ ⓘ Arquivo | C:/zzz/registrar.html

Validando formulários com JQuery Validation

Nome:
 This field is required.

E-mail:
 This field is required.

Senha:
 This field is required.

Figura 15 – Resultado da validação

Fonte: Senac EAD (2023)

Adicionando regras de validação para campos de entrada

Você também pode passar algumas regras para o método **validate()** para determinar como os valores de entrada são validados. O valor do parâmetro **rules** deve ser um objeto com pares chave-valor. A chave em cada caso é o nome do elemento que se quer validar. O valor dessa chave é um objeto que contém um conjunto de regras que serão usadas para validação.

Veja, na tabela a seguir, as principais regras de validação:

Rule	Validação
e-mail	Quando definido como "true" , esse campo aceitará apenas endereços de <i>e-mail</i> válidos.
required	Se definido como "true" , esse campo é obrigatório.
min	Define o número ou valor mínimo válido. Por exemplo, se definido como 4, o usuário só poderá informar um número maior igual a 4.
max	Define o número ou valor máximo válido. Por exemplo, se definido como 10, os usuários podem inserir apenas números menores ou iguais a 10.
range	Determina o intervalo que contém os valores mínimo e máximo válidos. Por exemplo, se definido como [10, 20], o usuário só poderá inserir um número maior ou igual a 10 e menor ou igual a 20.
minlength	Especifica o número mínimo de caracteres. Se definido como 10, por exemplo, o usuário só poderá inserir um valor de no mínimo 10 caracteres.
maxlength	Define o número máximo de caracteres. Se definido como 5, por exemplo, o usuário poderá inserir no máximo cinco caracteres.
url	Se definido como "true" , esse campo aceitará apenas URLs válidas.
date	Se definido como "true" , esse campo aceitará apenas datas válidas.
number	Se definido como "true" , esse campo aceitará apenas números inteiros ou decimais.

Tabela 1 – Principais regras de validação

Fonte: Senac EAD (2023)

Para exemplificar, retorne ao exemplo anterior no arquivo **registrar.html** e remova as propriedades **required**, modificando o **type="email"** para **"text"**. Dessa forma, não haverá nenhuma validação.

```
<!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" />
<script src="https://code.jquery.com/jquery-3.4.1.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/jquery-validation@
1.19.5/dist/jquery.validate.js"></script>

    <title>Validando formulários com JQuery Validation</title>
</head>

<body>
    <h1>Validando formulários com JQuery Validation</h1>
    <form method="post" action="" id="frmRegistro">
        <p>Nome: <br><input type="text" name="nome" id="nome"></p>
        <p>E-mail: <br><input type="text" name="email" id="email"></p>
        <p>Senha: <br><input type="text" name="senha" id="senha"></p>
        <input type="submit" value="Cadastrar">
    </form>

    <div id="result"></div>
    <script src="validar.js"></script>
</body>

</html>
```

Agora, no arquivo **validar.js**, acrescente o seguinte código:

```
$("#frmRegistro").validate({
    rules: {
        nome: {
            required: true
        },
        email: {
            required: true,
            email: true
        },
        senha: {
            required: true,
            minlength: 6,
            maxlength: 12
        }
    },
    messages: {
        nome: {
            required: "campo nome é obrigatório",
        },
        email: {
            required: "campo e-mail é obrigatório",
            email: "O e-mail deve estar no formato: xxx@xxxx.xxx"
        },
        senha: {
```

```
required: "campo senha é obrigatório",  
minlength: "A senha deve ter ao menos 6 caracteres",  
maxlength: "A senha deve ter no máximo 12 caracteres",  
},  
}  
});
```

No trecho de código acima, as chaves **nome**, **email** e **senha** são simplesmente os nomes dos elementos de entrada do formulário no HTML. Cada chave tem um objeto como seu valor, e os pares chave-valor no objeto determinam como um campo de entrada será validado.

Por exemplo, definir **required** como **true** tornará o elemento necessário para o envio do formulário. Definir **minlength** um valor como 6 forçará o usuário a inserir pelo menos seis caracteres na entrada de texto.

Personalizando as mensagens de validação

Assim como as **rules**, é possível personalizar as mensagens de erro de validação por meio do parâmetro **messages**. Por exemplo, o campo da senha acionará **required** caso seja deixado em branco, e a mensagem definida “campo senha é obrigatório” será exibida. De outra forma, caso a quantidade de caracteres informada seja inferior a seis caracteres ou superior a 12, será exibida a mensagem “A senha deve ter ao menos 6 caracteres” ou “A senha deve ter no máximo 12 caracteres”, respectivamente.

← → ↻ 📄 Arquivo | C:/zzz/registrar.html

Validando formulários com JQuery Validation

Nome: campo nome é obrigatório

E-mail: campo e-mail é obrigatório

Senha: A senha deve ter ao menos 3 caracteres

Figura 16 – Personalização da mensagem de erro
Fonte: Senac EAD (2023)



Máscaras

Ao lidar com dados, provenientes do preenchimento de um formulário, você pode querer que eles sejam formatados de uma determinada maneira. Sem dúvida, você pode fornecer instruções em cada campo, mas haverá uma grande variedade de usuários preenchendo o formulário. Nem todos podem inserir os dados no formato desejado. Além disso, o usuário pode não querer manter a formatação dos dados.

Por isso, adotam-se máscaras nos formulários em HTML, o que possibilita uma redução de inconsistências nas entradas de dados.

O **jQuery Mask** é um *plugin* leve, muito simples de adicionar e implementar e que cria máscaras para entrada de dados em campos de formulário e elementos HTML. Esse *plugin* é útil para obter dados, números, números de telefone, CPF, CNPJ, datas, entre outros.

O *link* do *plugin* do jQuery Mask poderá ser obtido diretamente no *site* (digite “jQuery Mask Plugin” em seu buscador), na opção “CDNJs”. No momento da escrita deste conteúdo, encontra-se na versão 1.14.16, conforme a imagem a seguir:

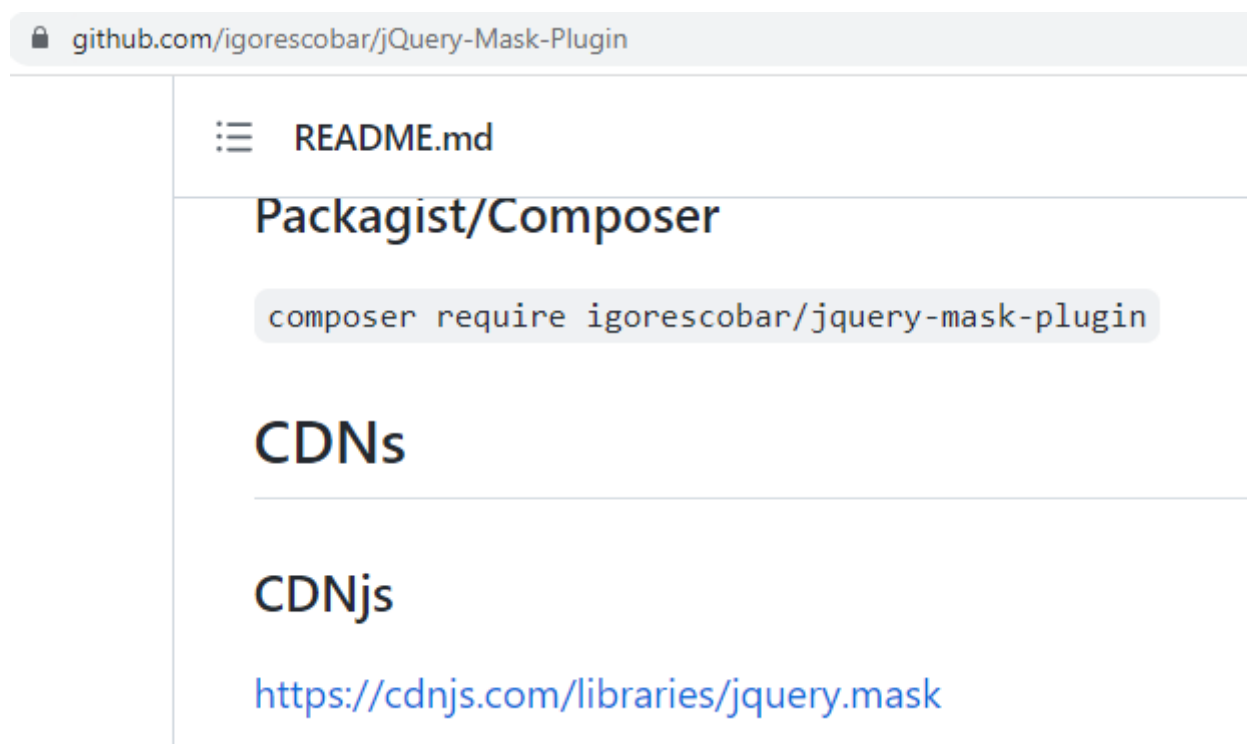


Figura 17 – Site do *plugin* jQuery Mask
Fonte: Senac EAD (2023)

Para exemplificar, veja como criar o arquivo **Formulario.html**. Inclua a biblioteca **jQuery** e o **plugin jQuery Mask** dentro da **tag head** do HTML. Deverá ser criado um formulário de cadastramento com os campos telefone, CEP, CPF, CNPJ e data.

```
<html>

<head>
  <meta charset="UTF-8">
  <title>Máscara de formulário

  <script src="https://code.jquery.com/jquery-3.4.1.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery.m
ask/1.14.16/jquery.mask.min.js"></script>
</head>

<body>
  <h1>Máscaras com JQuery Mask</h1>
  <form method="post" action="" id="frm">
    <p>
      <label>Telefone</label><br>
      <input type="text" id="telefone">
    </p>

    <p>
      <label>CEP</label><br>
      <input type="text" id="cep">
    </p>

    <p>
      <label>CPF</label><br>
      <input type="text" id="cpf">
    </p>

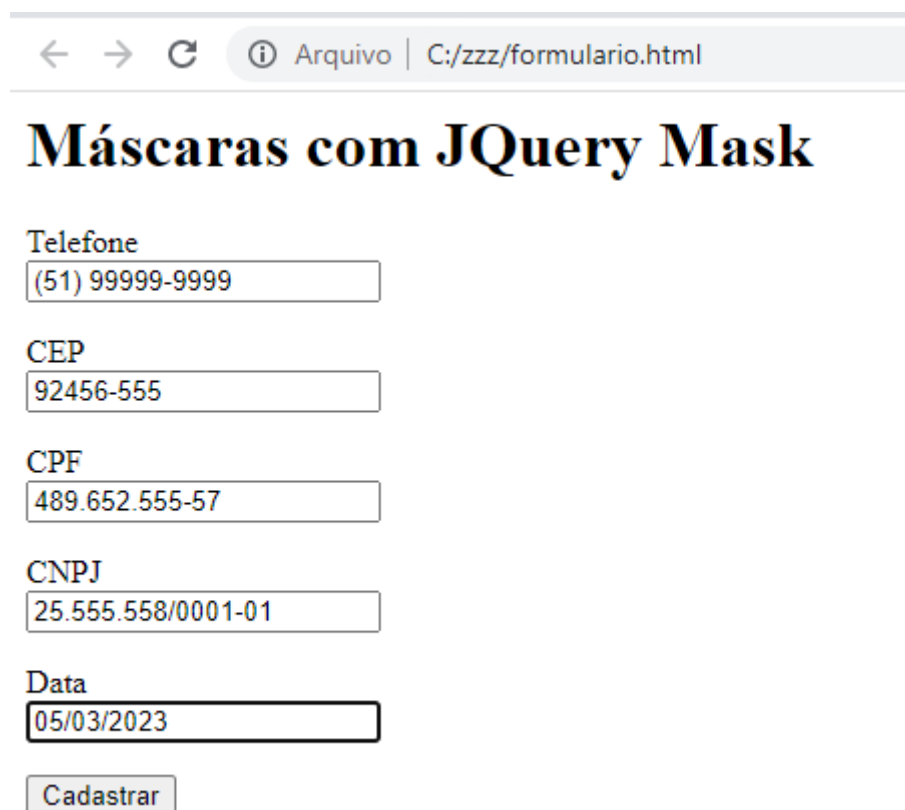
    <p>
      <label>CNPJ</label><br>
      <input type="text" id="cnpj">
    </p>

    <p>
      <label>Data</label><br>
      <input type="text" id="data">
    </p>
    <input type="submit" value="Cadastrar">
  </form>
  <script>
    $(document).ready(function () {
      $('#telefone').mask('(00) 00000-0000');
      $('#cep').mask('00000-000');
      $('#cpf').mask('000.000.000-00');
      $('#cnpj').mask('00.000.000/0000-00');
      $('#data').mask('00/00/0000');
    });
  </script>
```


</body>

</html>

Esse código tem máscaras nos campos aplicado diretamente na *tag script*. Como pode ser visto, simplesmente é informado o nome do campo pela *tag id* do formulário, por exemplo **id="telefone"**, e então você pode aplicar a máscara usando a função **mask** e informar o formato, nesse caso (00) 00000-00.



← → ↻ ⓘ Arquivo | C:/zzz/formulario.html

Máscaras com JQuery Mask

Telefone
(51) 99999-9999

CEP
92456-555

CPF
489.652.555-57

CNPJ
25.555.558/0001-01

Data
05/03/2023

Cadastrar

Figura 18 – Exemplo de máscara de entrada de dados

Fonte: Senac EAD (2023)

Encerramento

Neste conteúdo, foi possível verificar os diversos aspectos nas solicitações GET e POST que o *front-end* deverá ter para requisitar e tratar as informações enviadas pelo *back-end*. Como foi visto, essa capacidade de interação evoluiu ao longo dos anos, havendo diversas formas de realizar, como por meio de APIs nativas do Fetch e de terceiros como o jQuery. Além disso, foram abordadas as validações e as máscaras de dados em formulários HTML, por meio de diversos exemplos.