

## Curso: Análise e Desenvolvimento de Sistemas – ADS

Ano: 2023/1

Orientação Técnica (OT) – 18

### Cadastro REST - Exibir Marcas – Party #3

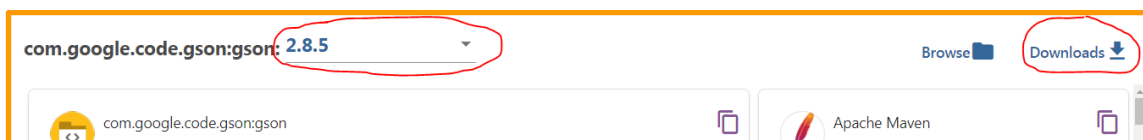
#### Preparando o retorno do servidor para o cliente

Como já sabemos, **não podemos enviar um objeto via HTTP**, seja numa requisição ou numa resposta. Ou seja, não vamos conseguir enviar o **objeto listaMarcas**, que contém as marcas encontradas, como ele está... Para solucionarmos isso, usaremos uma biblioteca para converter objetos Java em JSON: a **Gson**!

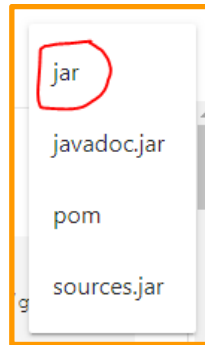
Assim, acesse <https://github.com/google/gson> e, descendo na página, procure pelo conteúdo da imagem a seguir:



Acesse o link destacado na imagem acima. Nele, confira se a versão mais atual está selecionada e clique em **Downloads**:

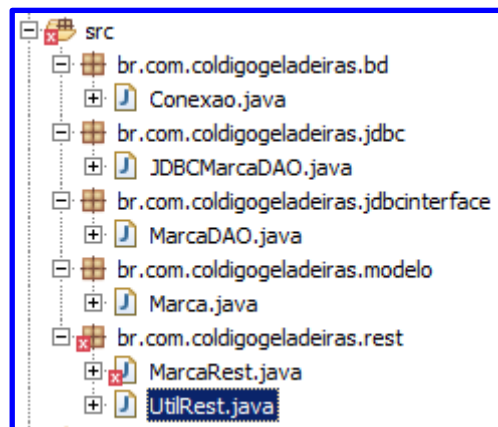


Nas opções que irão aparecer, selecione **jar**:



Com isso, o download do Gson se iniciará. Assim que finalizar, coloque-o junto aos seus amiguinhos na **pasta lib** de nosso projeto, para que possamos usá-lo.

Agora vamos criar **um novo arquivo**, que será um **utilitário** de nossas **ações REST** para que **não precisemos codificá-las repetidas vezes**. No pacote projetado para armazenarmos todos os nossos arquivos REST, crie uma nova classe chamada **UtilRest**, como na imagem a seguir:



Nela, importe 3 classes:

```
package br.com.coldigogeladeiras.rest;

import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.ResponseBuilder;

import com.google.gson.Gson;

public class UtilRest {
```

As duas primeiras nos permitirão que criemos **respostas de sucesso e erro personalizadas**, e a terceira é para usarmos o **Gson** que acabamos de importar.

Por enquanto, criaremos 2 métodos nela: um para criarmos *respostas de sucesso* e outra para *respostas de erro*. Iniciaremos criando o método **buildResponse**, demonstrado na imagem abaixo:

```
/*
 * Abaixo o método responsável por enviar a resposta ao cliente sobre
 * a transação realizada (inclusão, consulta, edição ou exclusão) caso
 * ela seja realizada com sucesso.
 * Repare que o método em questão aguarda que seja repassado um
 * conteúdo que será referenciado por um objeto chamado result.
 */
public Response buildResponse(Object result) {

    try {
        /*
         * Retorna o objeto de resposta com status 200 (ok), tendo
         * em seu corpo o objeto valorResposta (que consiste no
         * objeto result convertido para JSON).
         */
        String valorResposta = new Gson().toJson(result);
        return Response.ok(valorResposta).build();
    } catch (Exception ex) {
        ex.printStackTrace();
        //Se algo der errado acima, cria Response de erro
        return this.buildErrorResponse(ex.getMessage());
    }
}
```

Veja que, mais uma vez, o código está bem comentado, mas em caso de dúvida pesquise um pouco mais e qualquer coisa, conversamos na validação...

Agora, abaixo dele, crie o método **buildErrorResponse**:

```
/*
 * Abaixo o método responsável por enviar a resposta ao cliente sobre
 * A transação realizada, inclusão, consulta, edição ou exclusão.ao
 * cliente, não realizadas com sucesso, ou seja, que contenha algum
 * erro.
 * Repare que o método em questão aguarda que seja repassado um
 * conteúdo que será referenciado pelo por um objeto chamado rb.
 */
public Response buildErrorResponse(String str) {
    /*
     * Abaixo o objeto rb recebe o status do erro.
     */
    ResponseBuilder rb = Response.status(Response.Status.INTERNAL_SERVER_ERROR);

    /*
     * Define a entidade (objeto), que nesse caso é uma
     * mensagem que será retornado para o cliente.
     */
    rb = rb.entity(str);

    /*
     * Define o tipo de retorno desta entidade(objeto), no
     * caso é definido como texto simples.
     */
    rb = rb.type("text/plain");

    /*
     * Retorna o objeto de resposta com status 500 (erro),
     * junto com a String contendo a mensagem de erro.
     */
    return rb.build();
}
```

Mais uma vez, temos os comentários que dispensam mais explicação.

Por fim, precisamos fazer com que nossa **classe REST MarcaRest** possa utilizar esses **métodos**, então se dirija a ela e implemente os códigos destacados:

```
public class MarcaRest extends UtilRest{

    @GET
    @Path("/buscar")
    @Produces(MediaType.APPLICATION_JSON)
    public Response buscar(){

        try{
            List<Marca> listaMarcas = new ArrayList<Marca>();

            Conexao conec = new Conexao();
            Connection conexao = conec.abrirConexao();
            JDBCMarcaDAO jdbcMarca = new JDBCMarcaDAO(conexao);
            listaMarcas = jdbcMarca.buscar();
            conec.fecharConexao();
            return this.buildResponse(listaMarcas);
        }catch(Exception e){
            e.printStackTrace();
            return this.buildErrorResponse(e.getMessage());
        }
    }
}
```

Explicando as novidades:

- Indicamos que a classe **MarcaRest** deve **herdar os métodos e atributos** da **UtilRest**, possibilitando assim o uso do que criamos nela;
- Colocamos o código que criamos anteriormente **dentro do try** de um **try-catch**;
- Adicionamos os **chamados aos dois métodos** de **UtilRest**, o de **sucesso** ao fim do **try** e o de **erro** ao fim do **catch**, assim criando a resposta que o cliente deve receber, seja ela de sucesso ou erro!

Nossa **codificação do lado servidor se encerra aqui para essa OT!** Agora vamos voltar a programar no lado cliente para fazermos o tratamento dos dados recebidos!

---

## Nosso primeiro teste de backend!

Como terminamos a programação backend, podemos testá-la. Mas para podermos obter algum resultado compreensível sobre o funcionamento do que fizemos até agora, abra o arquivo **product.js** e adicione os códigos destacados conforme a imagem:

```
$(document).ready(function() {  
  
    //Carrega as marcas registradas no BD no select do formulário de inserir  
    COLDIGO.produto.carregarMarcas = function(){  
        alert("Tentando buscar marcas");  
        $.ajax({  
            type: "GET",  
            url: "/ProjetoTrilhaWeb/rest/marca/buscar",  
            success: function (marcas) {  
                alert("Sucesso");  
            },  
            error: function (info) {  
                alert("Erro");  
            }  
        });  
    }  
  
    COLDIGO.produto.carregarMarcas();  
  
});
```

Explicando:

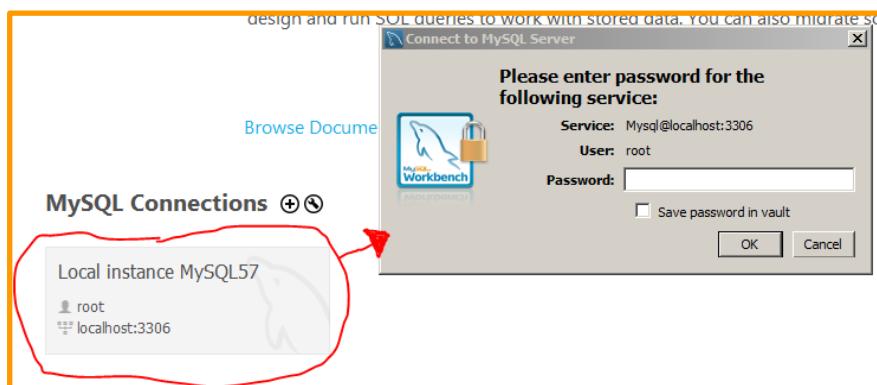
- Criamos 3 alertas, para sabermos se a função foi encontrada, e se recebemos uma mensagem de sucesso ou erro do servidor na execução do Ajax;
- Adicionamos **parâmetros** às funções **success** e **error**, para que possamos usar posteriormente os dados recebidos do servidor;
- Abaixo, fora da função carregarMarcas e dentro da função do \$(document).ready, **chamamos a função carregarMarcas**, afinal, assim que a página for carregada, as marcas devem aparecer no formulário para o usuário poder escolher a desejada, não é?

Agora vem uma parte muito desejada: **VAMOS TESTAR?** OK, mas se queremos **mostrar algo que está no BD**, precisamos **ter ele disponível** antes. Assim...

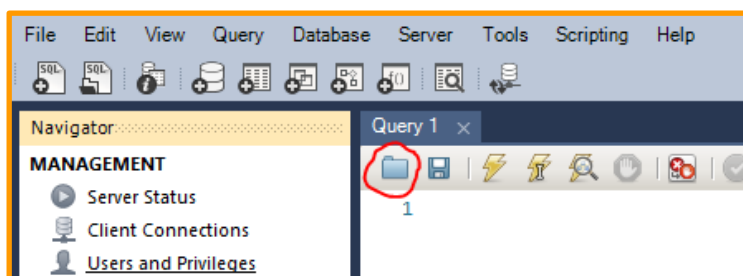
Primeiro, **crie no Workbench** o banco de dados que fizemos anteriormente para o projeto, o **bdcoldigo**. Como?


Lembra o **arquivo .sql** que pedimos para criar no final da **OT de banco de dados**, com os códigos de **criação do BD**, das **tabelas marcas e produtos** e mais **3 INSERTs com marcas**? Pois é, agora vamos usá-lo!

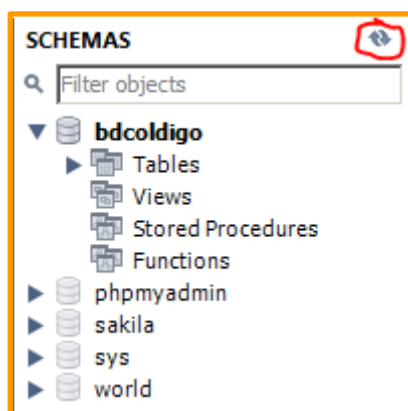
Abra o Workbench. Depois, se conecte usando a conexão criada anteriormente:



Agora, clique no ícone destacado para **abrir o arquivo .sql** que criamos anteriormente:



Depois de **encontrar o arquivo e abri-lo**, execute-o clicando em . Em seguida, clique em **Schemas** à esquerda da tela e veja se o **bdcoldigo** se encontra na lista de bancos criados no MySQL (se não o ver de primeira, clique no ícone destacado abaixo para atualizar a lista):



Agora que o banco está **pronto para uso**, teste sua programação acessando a página de cadastro de produtos. Se tudo der certo, os alertas de “Tentando buscar marcas” e “Sucesso” serão exibidos. Ainda não serão mostradas as marcas no formulário, isso será feito a seguir!



**ATENÇÃO 1:** Caso esteja nos computadores da instituição, isso deverá ser feito **TODOS OS DIAS** em que for usar o BD!!! **Isso não será falado novamente!**

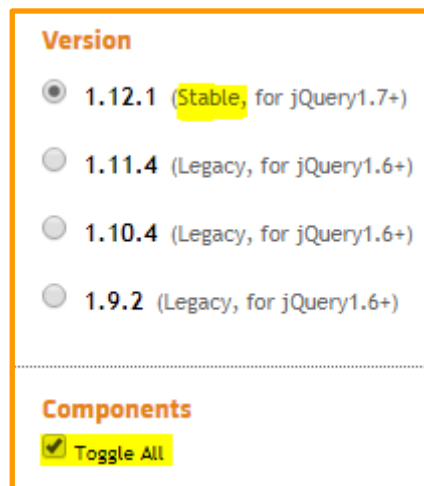
**ATENÇÃO 2:** **em caso de erro, não prossiga!** Tente verificar com o Inspecionar do navegador (se for um erro frontend) e pela console do Eclipse (se for um erro de servidor) o que pode estar acontecendo de errado e corrija. Se não estiver encontrando, procure algum professor.

---

## Exibindo os dados do servidor no cliente

Primeiro, para exibirmos algumas informações com maior qualidade, vamos fazer download do **jQueryUI**, uma **biblioteca para adicionarmos efeitos visuais** a nossas aplicações.

Para isso, acesse <https://jqueryui.com/download/>. Lá, escolha a versão indicada como **Stable** (estável) e marque o checkbox “**Toggle All**”, como na imagem abaixo:



Desça até o fim da página, e escolha o tema “**UI lightness**” (pelo menos se quiser que fique como o nosso...) e clique em **Download**.

**Theme**  
Select the theme you want to include or [design a custom theme](#)  

UI lightness ▾

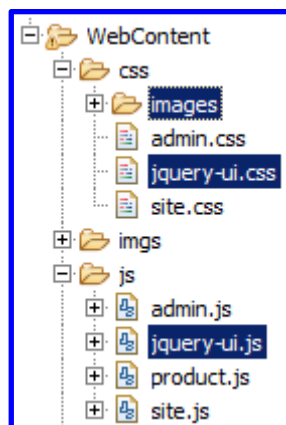
CSS Scope:

Download

Descompacte o arquivo baixado. Agora adivinha onde adicionaremos os arquivos que baixamos? Achou que é em **lib**?



Precisamos de **3 coisas do jQueryUI**: os arquivos **jquery-ui.css** e **jquery-ui.js**, e a **pasta images**. Veja com muito cuidado a imagem abaixo, e coloque os arquivos citados e destacados nos mesmos locais:



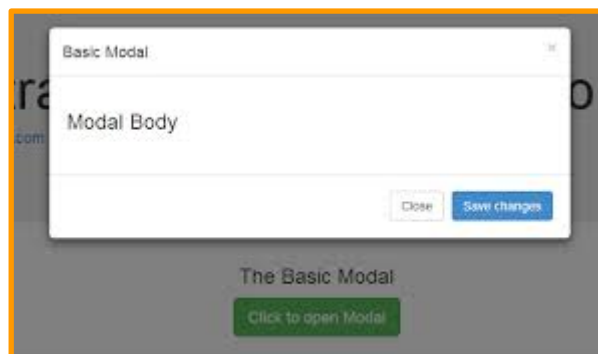
Agora, para podermos utilizar essa biblioteca em nossos códigos, abra o arquivo **index.html da área de administração** e insira os códigos destacados em seus respectivos locais (para não se confundir com os vários **index**, verifique o restante do código da imagem).



```
<script src="../../js/admin.js"></script>
<script src="../../js/jquery-ui.js"></script>
<link rel="stylesheet" type="text/css" href="../../css/jquery-ui.css">
</head>
<body>
  <header>
  </header>
  <section>
    <h2>O que você deseja fazer hoje?</h2>
  </section>
  <div id="modalAviso"></div>
  <footer>
  </footer>
```

As duas primeiras linhas importam os arquivos **css** e **js** do jQueryUI para o arquivo, e a **div** criada a seguir nos servirá para **exibirmos avisos** (como de sucesso ou erro) de um **jeito mais amigável** do que fazemos atualmente com o **alert** do JS.

A palavra “**modal**”, caso não seja familiar para você, refere-se a um **conteúdo aberto por cima de outro conteúdo**, como o caso da imagem abaixo:



É para isso que essa **div** vai servir, ela tomará esse comportamento visto na imagem para darmos **destaque** a uma mensagem **sem esconder o restante do conteúdo nem sair da página!**

Vamos agora aplicar uma **configuração CSS** para essa **modal**, adicionando no arquivo **admin.css** o seguinte código no local indicado:

```
/* Rodapé */
footer{
  background-color:#005f89;
  color: #fff;
  text-align:center;
  padding:10px;
}

/* Modais */
#modalAviso{
  display: none;
}

/* Layout CRUDs */
```

Pesquise essa propriedade e **nos explique na validação**.

Por fim, vamos criar uma **função que gerenciará a modal de aviso**. Ela deve ficar no arquivo **admin.js**, no local indicado pela imagem abaixo:

```
var msg = "Houve um erro ao encontrar a página: " + info.status + " - " + info.statusText;
$("#section").html(msg);
});
}

//Define as configurações base de uma modal de aviso
COLDIGO.exibirAviso = function(aviso){
    var modal = {
        title: "Mensagem",
        height: 250,
        width: 400,
        modal: true,
        buttons: {
            "OK": function(){
                $(this).dialog("close");
            }
        }
    };
    $("#modalAviso").html(aviso);
    $("#modalAviso").dialog(modal);
};
});
```

Veja que:

- A função pertence ao objeto **COLDIGO** e se chama **exibirAviso**;
- Ela recebe como **parâmetro** um dado a ser guardado em uma variável chamada **aviso**;
- Uma **variável** chamada **modal** recebe **configurações** de: **título, altura, largura**, a indicação de que deve **agir como uma modal**, e o **botão OK**, que ao ser clicado **deve fechá-la**;
- Aí, colocamos o valor da variável **aviso dentro** da estrutura com **ID modalAviso** (ou seja, a nossa **div** que será a modal) usando a função **jQuery html**;
- E, por fim, chamamos a **função do jQueryUI** chamada **dialog** para **abrir a modal**, passando a ela as **configurações** criadas na variável **modal**!

Agora, nossa modal de aviso está pronta para ser usada. Nossa primeira aplicação dela será na função do **error do Ajax** criado para buscarmos as marcas no BD. Para programar essa parte, vá ao arquivo **product.js** e insira o novo código no local indicado:

```
error: function (info) {
    COLDIGO.exibirAviso("Erro ao buscar as marcas: " + info.status + " - " + info.statusText);
    $("#selMarca").html("");
    var option = document.createElement("option");
    option.setAttribute("value", "");
    option.innerHTML = ("Erro ao carregar marcas!");
    $("#selMarca").append(option);
    $("#selMarca").addClass("aviso");
}
```

A primeira linha **chama a função que exibe a modal**, passando a **frase** que queremos que apareça nela (que por sua vez, contém o número do status de erro em **info.status** bem como o texto padrão desse status em **info.statusText**).

Por sinal, **de onde vem os valores dessa variável info** mesmo?

As demais linhas se encarregam, de modo parecido com o qual fizemos nosso formulário da trilha de frontend manipular uma tag select dinamicamente, de **inserir na tag select das marcas** uma **option** indicando que houve **erro**, evitando assim que o formulário possa ser enviado de modo incompleto.

Assim, se não for possível encontrar as marcas, veremos esse erro em uma modal:



Para vê-la, precisará **forçar um erro no backend**, o que pode ser feito comentando a linha que está destacada na imagem a seguir, no arquivo **MarcaRest**:

```
Conexao conec = new Conexao();  
Connection conexao = conec.abrirConexao();  
JDBCMarcaDAO jdbcMarca = new JDBCMarcaDAO(conexao);  
listaMarcas = jdbcMarca.buscar();  
}
```

Não esqueça de Parar/Reiniciar o servidor novamente antes de testar, afinal, você fez uma modificação do backend. Após o teste, desfaça esse comentário.

**ATENÇÃO:** em caso de erro, no caso, de você não ver a modal, não prossiga! Tente verificar o que pode estar acontecendo de errado e corrija. Se não estiver encontrando, procure algum professor.

---

## Exibindo as marcas registradas no BD no formulário

Agora sim, vamos a parte que realmente mostrará o resultado desejado de tudo o que fizemos até agora nessa OT: **exibir as marcas do BD no campo select do formulário!**

Para isso, no **success do Ajax** que buscou esses dados no servidor, acrescente um **if** que verifique se o **resultado recebido do BD não está vazio** para fazermos a exibição das marcas, com um **else** para tratarmos os casos onde não há marcas no BD. Veja na imagem abaixo como fazê-lo:

```
success: function (marcas) {  
    if (marcas!="") {  
    }else{  
    }  
},
```

Primeiro, vamos programar o que deve ser feito **se não houverem marcas**, ou seja, insira dentro do **else** recém criado os códigos abaixo:


```
}else{  
    $("#selMarca").html("");  
  
    var option = document.createElement("option");  
    option.setAttribute("value", "");  
    option.innerHTML = ("Cadastre uma marca primeiro!");  
    $("#selMarca").append(option);  
    $("#selMarca").addClass("aviso");  
}
```

Veja que nele apenas **limpamos as options existentes** no campo select (passando a ele um **html vazio** como conteúdo, **na primeira linha**) e depois criamos dentro dele uma **nova option** indicando que **não há marcas registradas...**

Quer ver como fica? Então **exclua as marcas registradas do BD (lembra do comando SQL DELETE?)** e **atualize a página**. Você verá:

Marca	Cadastre uma marca primeiro! ▼
-------	--------------------------------

**ATENÇÃO:** em caso de erro, não prossiga! Tente verificar o que pode estar acontecendo de errado e corrija. Se não estiver encontrando, procure algum professor.

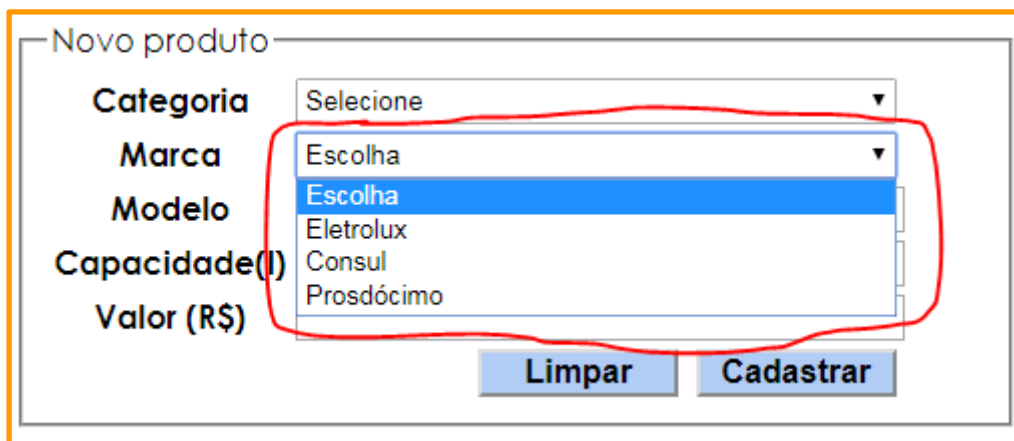
Depois de testar, **reinsira as marcas** que acabou de excluir **no BD** (no Workbench, selecione apenas os INSERT delas no **arquivo .sql** e clique no ícone , assim você só executará a parte selecionada!) e vamos em frente...

Agora, vamos fazer o código do **if**, a ser executado quando houverem marcas no BD:

```
if (marcas!="") {  
  
    $("#selMarca").html("");  
    var option = document.createElement("option");  
    option.setAttribute("value", "");  
    option.innerHTML = ("Escolha");  
    $("#selMarca").append(option);  
  
    for (var i = 0; i < marcas.length; i++) {  
  
        var option = document.createElement("option");  
        option.setAttribute("value", marcas[i].id);  
        option.innerHTML = (marcas[i].nome);  
        $("#selMarca").append(option);  
  
    }  
}  
}else{
```

Assim, se houver pelo menos uma marca no BD, primeiro criamos uma **option** “Escolha” com valor vazio, para podermos verificar se o usuário escolheu alguma marca válida, e depois fazemos um **for** para, a cada marca encontrada, criarmos um **option** para ela, onde o **texto** exibido será o **nome** da marca e o **valor** será seu **ID**.

Para testar, novamente **atualize a página** e veja se as marcas que registrou aparecem no select, como no exemplo abaixo:



Novo produto

Categoria Selezione

Marca Escolha

Modelo Eletrolux

Capacidade(l) Consul

Valor (R\$) Prosdócimo

Limpar Cadastrar

**ATENÇÃO:** em caso de erro, não prossiga! Tente verificar o que pode estar acontecendo de errado e corrija. Se não estiver encontrando, procure algum professor.

---

## Reforçando o conhecimento adquirido

Quanta novidade, né? Porém, é tudo interconectado, então não tem como separar mais se quisermos manter sentido em cada parte do código... Assim, pedimos com muito carinho que execute as seguintes ações:

- **Releia o documento com calma!** Agora que possui o código completo, vai ser bem mais fácil se concentrar em acompanhar o que está acontecendo. Assim, se você ficou viajando na maionese, vai começar a entender melhor, e se já entendeu bem, vai ter a oportunidade de afirmar essa compreensão e de repente encontrar coisas que passaram batidas à primeira vista...
- **Comente o código!** Comentar o código com o que compreendeu e suas dúvidas vai ajudar agora, na hora da validação, e também depois, quando for realizar outro projeto e precisar lembrar de algo que já fez anteriormente ;D
- **Através de um fluxograma ou outra forma que não seja um texto corrido, esquematize o funcionamento do processo criado nesta OT**, desde o momento em que carregamos a página até o momento em que os dados são mostrados no select. Represente os arquivos, quando um chama o outro, como o servidor encontra a Rest, enfim, **desenhe** o fluxo criado nesta OT. **O esquema criado será usado na validação.**

---

## Conclusão

Fazer uma aplicação cliente-servidor exige conhecimentos em diversas áreas: programação frontend, programação backend, banco de dados... Nessa OT, passamos por todas elas, criando inclusive coisas que não precisaremos mais mexer até o final do projeto, como a classe de conexão ao banco (afinal de contas, esse projeto não usará outro banco). Essa OT precisa ser bem compreendida por você, já que os demais processos que executaremos terão repetições de várias coisas que usamos aqui, então se, mesmo depois de ter relido e feito o esquema ainda não se sente tão confortável com o que viu, aproveite bem a validação para tirar as dúvidas, e vamos em frente!

*Após finalizar a OT, crie um novo commit no Git com o nome da OT e comunique um orientador para novas instruções.*