1 Introdução

Aplicações comuns, quando utilizam formulários, costumam submetê-los a um servidor quando o usuário clica em um botão do tipo submit. Com o Angular, o funcionamento é diferente. O Angular intercepta o envio do formulário e representa os valores inseridos como um objeto JSON. Quando o botão de envio do form é clicado, uma função que especificamos é chamada e nela temos acesso ao objeto JSON que podemos manipular como necessário. As figuras 1.1 e 1.2 ilustram o funcionamento dos forms tradicionais e dos forms manipulados pelo Angular, respectivamente.

Figura 1.1

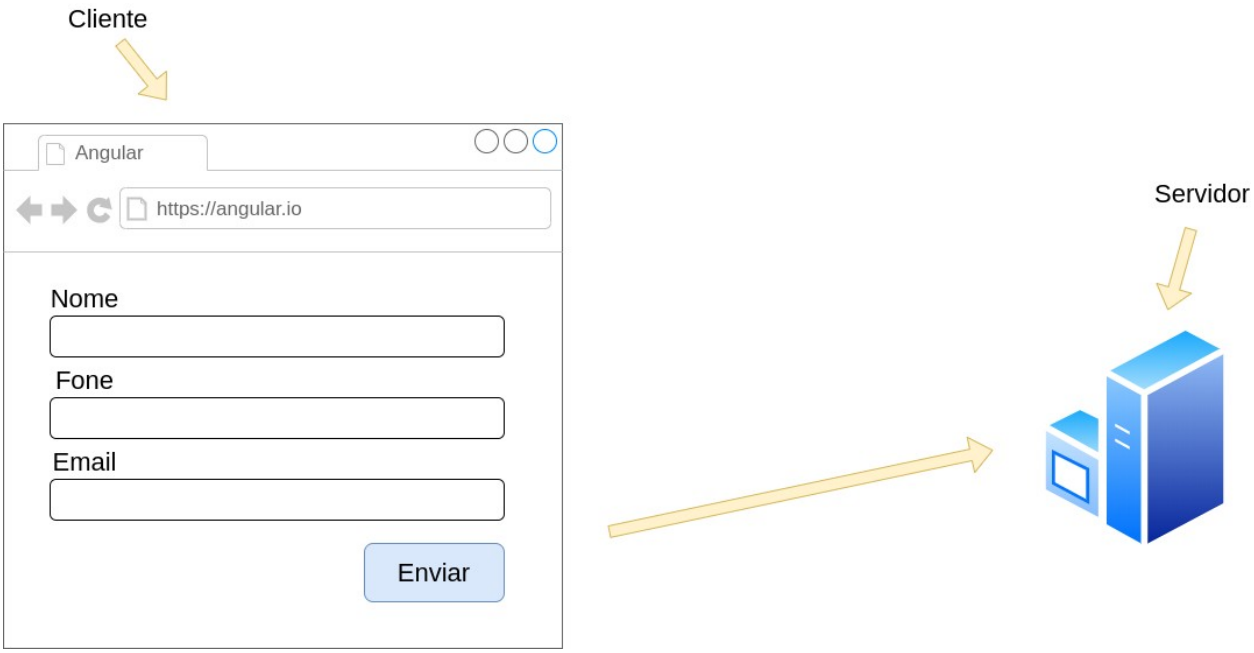
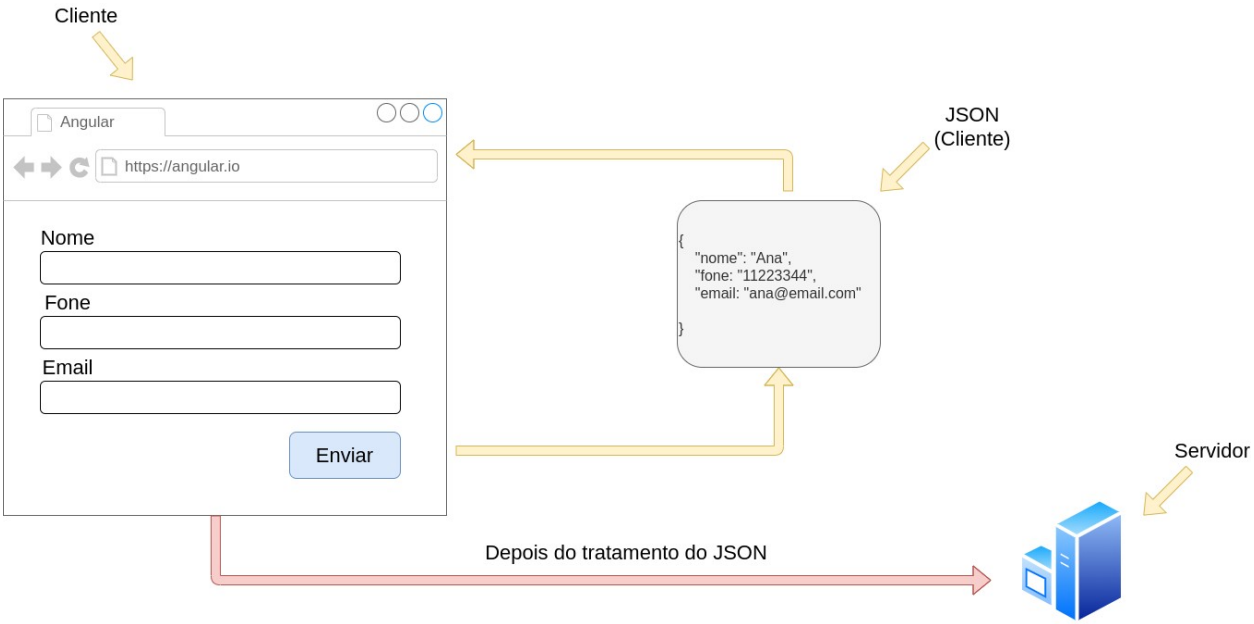


Figura 1.2



- Há dois tipos de formulários no Angular: **Template Driven Forms** e **Reactive Forms**. Com o primeiro, fazemos toda a especificação do formulário no próprio template, ou seja, na página HTML. Ele é recomendado para usos simples que não requerem muitas validações, por exemplo. Com o segundo, o programador especifica o modelo associado ao formulário na classe do componente e, assim, tem muito mais controle sobre ele. É recomendado para formulários mais complexos. A documentação oficial sobre formulários Angular pode ser vista no Link 1.1.

Link 1.1
https://angular.io/guide/forms-overview

2 Desenvolvimento

2.1 (Novo projeto) Comece abrindo um novo terminal e navegando até o seu workspace, ou seja, a pasta que você está usando para abrigar os seus projetos Angular. Tome o cuidado de não acessar a pasta de nenhum projeto já existente. Use os comandos a seguir para criar um novo projeto Angular, navegar até o seu diretório e abrir uma instância do VS Code vinculada a ele.

ng new nome-do-projeto
cd nome-do-projeto
code .

- Quando perguntado, escolha não adicionar um módulo para roteamento e use o modelo CSS simples.

- Coloque o servidor de testes em execução com

ng serve --open

- Instale o Bootstrap com

npm install bootstrap@latest

- A seguir, configure o Bootstrap no arquivo **angular.json** adicionando o arquivo **node_modules/bootstrap/dist/css/bootstrap.css** ao vetor cuja chave é **projects/nome-do-projeto/architect/build/options/styles**.

2.2 (Um novo componente) Crie um novo componente com o seguinte comando.

ng g c formulario --skipTests

- Apague o conteúdo do arquivo **app.component.html** e passe a utilizar o componente recèm criado por meio de seu seletor, como mostra a Listagem 2.2.1.

Listagem 2.2.1

```
<app-formulario></app-formulario>
```

- O novo componente irá definir um formulário com alguns poucos campos, como na Listagem 2.2.2. Estamos no arquivo **formulario.component.html**.

Listagem 2.2.2

```
<div class="container">
   <form>
      <div class="row">
         <div class="form-group col-12">
            <label for="nomeInput">Nome</label>
            <input type="text" placeholder="nome" class="form-control p-4">
         </div>
      </div>
      <div class="row">
         <div class="form-group col-md-4">
            <label for="foneInput">Fone</label>
            <input type="text" placeholder="fone" class="form-control p-4">
         </div>
         <div class="form-group col-md-4">
            <label for="emailInput">Email</label>
            <input type="text" placeholder="email" class="form-control p-4">
         </div>
         <div class="form-group col-md-4">
            <label for="profissaoInput">Profissão</label>
            <select name="" id="profissaoInput" class="form-control">
               <option value="">Selecione</option>
               <option>Engenheiro</option>
               <option>Professor</option>
               <option>Químico</option>
               <option>Zoólogo</option>
            </select>
         </div>
      </div>
      <div class="row">
         <div class="col-12">
            <button type="submit" class="btn btn-primary btn-block">Enviar</button>
         </div>
      </div>

   </form>
</div>
```

- Neste momento, clique no botão Enviar e veja o que ocorre na URL e na aba que tem o título do site. Eles indicam que o form teria sido submetido a um servidor caso tivéssemos especificado isso (no action do form) e que a página teria sido recarregada. Isso quer dizer que temos o funcionamento padrão, ilustrado na Figura 1.1.

- Desejamos alterar o funcionamento padrão, especificando que o Angular deve interceptar a submissão do form. O primeiro passo para isso, é adicionar o módulo de formulários do Angular ao arquivo **app.module.ts**. Veja a Listagem 2.2.3.

Listagem 2.2.3

```
import { FormsModule } from '@angular/forms';
@NgModule({
  declarations: [
    AppComponent,
    FormularioComponent,
    TesteComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

- Feito isso, recarregue a página e tente clicar novamente no botão Enviar. Note que nada acontece. Ocorre que o Angular está interceptando a requisição, como ilustrado na Figura 1.2. Isso ocorre pois o módulo FormsModule inclui a definição de uma diretiva que seleciona componentes pelo seu tipo. Em particular, ela seleciona os componentes do tipo form e seu funcionamento se aplica sobre eles. É ela a responsável por essa alteração no funcionamento de submissão de forms.

- Note que, caso desejado, podemos aplicar uma diretiva a um form para especificar que o Angular não deve gerenciá-lo. Veja o exemplo da Listagem 2.2.4. Depois de adicionar a diretiva, tente submeter novamente o form e veja a página volta a ser recarregada. Remova a diretiva depois de testá-la.

2.3 (Chamando um método quando o form é submetido) Para chamar um método quando o form for submetido, iremos fazer um event binding no próprio formulário usando a propriedade **ngSubmit.** Veja a Listagem 2.3.1.

Listagem 2.3.1

```
<form (ngSubmit)="salvar()">
```

- Crie o método salvar como a Listagem 2.3.2 ilustra e verifique no console do seu navegador se está tudo ok.

Listagem 2.3.2

```
salvar() {
   console.log("Form interceptado...");
}
```

- Note que o método salvar precisa de acesso aos dados contidos no formulário. Para viabilizar isso, iremos atribuir uma variável de template (um identificador, simplesmente) ao form e, a seguir, entregá-lo como parâmetro para o método salvar. Veja a Listagem 2.3.3.

Listagem 2.3.3

```
 <form #pessoaForm (ngSubmit)="salvar(pessoaForm)">
```

- Ajuste a assinatura do método salvar para que ele possa receber o argumento que lhe está sendo enviado. A seguir, exiba o conteúdo no console e o inspecione. Veja a Listagem 2.3.4.

Listagem 2.3.4

```
salvar(pessoaForm) {
   console.log(pessoaForm);
  }
```

- Note que o objeto entregue ao método é o próprio elemento HTML form. A diretiva que está associada ao form e que altera o seu funcionamento se chama **ngForm**. Para acessá-la, vamos atribuí-la à variável de template que aplicamos ao form. Ou seja, a variável de template deixará de fazer referência ao form e passará a fazer referência à diretiva associada ele. Isso é de interesse pois, por meio dela, poderemos acessar o objeto JSON que contém os dados digitados nos campos do form. Veja a Listagem 2.3.5.

Listagem 2.3.5

```
<form #pessoaForm="ngForm" (ngSubmit)="salvar(pessoaForm)">
```

- Clique novamente no botão Enviar e inspecione a saída no console para descobrir que agora temos acesso ao objeto JSON com os dados de interesse.

- Embora tenhamos acesso ao JSON que modela o form e que contém os dados de interesse, eles ainda não podem ser encontrados ali. Isso ocorre pois precisamos dizer explicitamente quais deles desejamos que sejam incluídos no JSON. Isso pode ser feito aplicando a diretiva **ngModel** a cada campo de interesse. Veja a Listagem 2.3.6.

Listagem 2.3.6

```
<div class="container">
  <form #pessoaForm="ngForm" (ngSubmit)="salvar(pessoaForm)">
    <div class="row">
      <div class="form-group col-12">
        <label for="nomeInput">Nome</label>
        <input type="text" placeholder="nome" class="form-control p-4" ngModel>
      </div>
    </div>
    <div class="row">
      <div class="form-group col-md-4">
        <label for="foneInput">Fone</label>
        <input type="text" placeholder="fone" class="form-control p-4" ngModel>
      </div>
      <div class="form-group col-md-4">
        <label for="emailInput">Email</label>
        <input type="text" placeholder="email" class="form-control p-4" ngModel>
      </div>
      <div class="form-group col-md-4">
        <label for="profissaoInput">Profissão</label>
        <select name="" id="profissaoInput" class="form-control" ngModel>
          <option value="">Selecione</option>
          <option>Engenheiro</option>
          <option>Professor</option>
          <option>Químico</option>
          <option>Zoólogo</option>
        </select>
      </div>
    </div>
    <div class="row">
      <div class="col-12">
        <button type="submit" class="btn btn-primary btn-block">Enviar</button>
      </div>
    </div>

  </form>
</div>
```

- Teste novamente clicando no botão Enviar. Inspecione a mensagem de erro exibida. Ela diz que os campos aos quais a diretiva ngForm foi aplicada precisam ter também seu atributo **name**

especificado. O name de um atributo será usado como chave no objeto JSON que modela o form. Faça, portanto, os ajustes da Listagem 2.3.7.

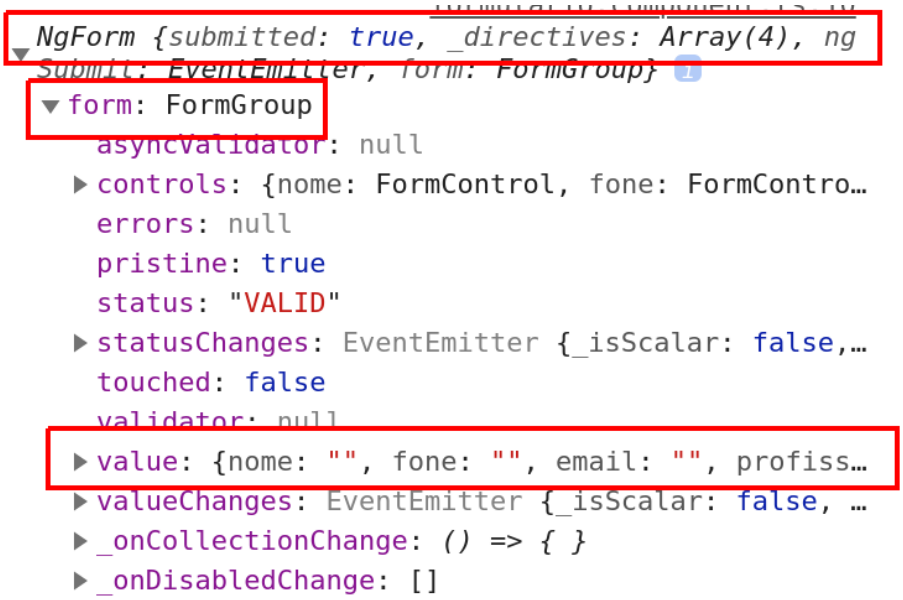
Listagem 2.3.7

```
<div class="container">
  <form #pessoaForm="ngForm" (ngSubmit)="salvar(pessoaForm)">
    <div class="row">
      <div class="form-group col-12">
        <label for="nomeInput">Nome</label>
        <input type="text" placeholder="nome" class="form-control p-4" name="nome" ngModel>
      </div>
    </div>
    <div class="row">
      <div class="form-group col-md-4">
        <label for="foneInput">Fone</label>
        <input type="text" placeholder="fone" class="form-control p-4" name="fone" ngModel>
      </div>
      <div class="form-group col-md-4">
        <label for="emailInput">Email</label>
        <input type="text" placeholder="email" class="form-control p-4" name="email" ngModel>
      </div>
      <div class="form-group col-md-4">
        <label for="profissaoInput">Profissão</label>
        <select name="" id="profissaoInput" class="form-control" name="profissao" ngModel>
          <option value="">Selecione</option>
          <option>Engenheiro</option>
          <option>Professor</option>
          <option>Químico</option>
          <option>Zoólogo</option>
        </select>
      </div>
    </div>
    <div class="row">
      <div class="col-12">
        <button type="submit" class="btn btn-primary btn-block">Enviar</button>
      </div>
    </div>

  </form>
</div>
```

- Inspecione novamente a saída no console, após clicar no botão Enviar. Expanda o objeto **NgForm** e, então, o objeto **form**. Encontre a propriedade **value**. Ali você verá um objeto JSON que tem como chaves os names de cada um dos campos do formulário e como valor aquilo que o usuário tiver digitado/selecionado em cada um deles. Veja a Figura 2.3.1.

Figura 2.3.1



- Faça um novo teste: altere o método salvar como mostra a Listagem 2.3.8. A seguir, preencha todos os campos do form e clique em salvar.

Listagem 2.3.8

```
salvar(pessoaForm) {
   const nome = pessoaForm.value.nome;
   const fone = pessoaForm.value.fone;
   const email = pessoaForm.value.email;
   const profissao = pessoaForm.value.profissao;
   console.log(`Nome: ${nome}, Fone: ${fone}, Email: ${email}, Profissão: ${profissao}`)
  }
```

2.4 (Uma lista de valores dinâmica para o select) Note que as opções referentes às profissões estão fixas no HTML, o que pode ser indesejável. É muito comum a necessidade de alimentar um select com dados obtidos a partir de um consulta a uma base de dados. Para ilustrar essa possibilidade, iremos implementar um método que devolve uma lista de profissões e vinculá-lo ao select. No futuro, veremos como fazer requisições HTTP para obter a lista, de fato, de um servidor remoto.

- Comece adicionando a lista de profissões à classe do componente, como mostra a Listagem 2.4.1.

Listagem 2.4.1

```
profissoes = ['Engenheiro', 'Professor', 'Químico', 'Zoólogo'];
```

- Para gerar os itens **option** do select em função da lista, podemos usar uma diretiva **ngFor**. Veja a Listagem 2.4.2.

Listagem 2.4.2

```
<div class="form-group col-md-4">
            <label for="profissaoInput">Profissão</label>
            <select name="" id="profissaoInput" class="form-control" name="profissao"
ngModel>
              <option value="">Selecione</option>
              <option *ngFor="let p of profissoes">{{p}}</option>
            </select>
          </div>
```

2.5 (Two way data binding) Neste contexto o Two way data binding também pode ser utilizado.

- Vamos começar criando uma classe que descreve o que é uma pessoa, ou seja, que possui as propriedades existentes no form de cadastro. Para isso, use o seguinte comando do Angular CLI. Ele irá criar uma classe de modelo em uma pasta (que também será criada por ele) chamada model.

ng g class model/pessoa --skipTests=true

- Abra a classe pessoa e adicione a ela as propriedades de interesse, como mostra a Listagem 2.5.1.

Listagem 2.5.1

```
export class Pessoa {
   nome: string;
   fone: string;
   email: string;
   profissao: string;
}
```

- A seguir, no arquivo **formulario.component.ts**, construa um objeto do tipo Pessoa e faça com que uma variável de referência o referencie, como mostra a Listagem 2.5.2.

Listagem 2.5.2

```
import { Pessoa } from '../model/pessoa';

export class FormularioComponent {

  pessoa: Pessoa = new Pessoa();
```

- Quando o método salvar for chamado, desejamos configurar cada uma das propriedades do objeto com os valores preenchidos no form. Veja as alterações a serem feitas no método na Listagem 2.5.3.

Listagem 2.5.3

```
salvar(pessoaForm) {
    this.pessoa.nome = pessoaForm.value.nome;
    this.pessoa.fone = pessoaForm.value.fone;
    this.pessoa.email = pessoaForm.value.email;
    this.pessoa.profissao = pessoaForm.value.profissao;
    console.log(this.pessoa);
  }
```

- Preencha os campos no form, clique no botão salvar e veja a saída no console.

- Adicione ao template (código html) uma interpolação para exibir o objeto pessoa. Podemos fazer isso com um **pipe** chamado json. O que ele faz é receber um objeto JSON e devolver a sua representação textual, tal qual o faz o método JSON.stringify. Veja a Listagem 2.5.4.

Listagem 2.5.4

```
<div class="container">
    {{pessoa | json}}
```

- Note que, a princípio, o que aparece na tela é um objeto vazio, sem propriedades. Isso acontece pois o objeto pessoa passa a ter valores somente depois de clicarmos no botão Salvar.

- O mecanismo conhecido como Two data binding permite que variáveis sejam vinculadas a elementos visuais de modo que, quando um for alterado, o outro seja atualizado automaticamente. E vice-versa. Para utilizá-lo, comece atualizando o método salvar para que ele não atualize os valores dos campos do objeto pessoa, como na Listagem 2.5.5.

Listagem 2.5.5

```
salvar(pessoaForm) {
   /*this.pessoa.nome = pessoaForm.value.nome;
   this.pessoa.fone = pessoaForm.value.fone;
   this.pessoa.email = pessoaForm.value.email;
   this.pessoa.profissao = pessoaForm.value.profissao;*/
   console.log(this.pessoa);
   console.log(pessoaForm);
  }
```

- A seguir, use a notação [( )] na diretiva **ngModel**. Ela caracteriza o uso do Two way data binding. Veja a Listagem 2.5.6.

Listagem 2.5.6

```
<div class="container">
   {{pessoa | json}}
   <form #pessoaForm="ngForm" (ngSubmit)="salvar(pessoaForm)">
      <div class="row">
         <div class="form-group col-12">
            <label for="nomeInput">Nome</label>
            <input type="text" placeholder="nome" class="form-control p-4" name="nome"
[(ngModel)]="pessoa.nome">
         </div>
      </div>
      <div class="row">
         <div class="form-group col-md-4">
            <label for="foneInput">Fone</label>
            <input type="text" placeholder="fone" class="form-control p-4" name="fone"
[(ngModel)]="pessoa.fone">
         </div>
         <div class="form-group col-md-4">
            <label for="emailInput">Email</label>
            <input type="text" placeholder="email" class="form-control p-4" name="email"
[(ngModel)]="pessoa.email">
         </div>
         <div class="form-group col-md-4">
            <label for="profissaoInput">Profissão</label>
            <select name="" id="profissaoInput" class="form-control" name="profissao"
               [(ngModel)]="pessoa.profissao">
               <option value="">Selecione</option>
               <option *ngFor="let p of profissoes">{{p}}</option>
            </select>
         </div>
      </div>
      <div class="row">
         <div class="col-12">
            <button type="submit" class="btn btn-primary btn-block">Enviar</button>
         </div>
      </div>
   </form>
</div>
```

- Teste novamente preenchendo os campos e clicando no botão Salvar. Perceba que, conforme um campo é preenchido, o valor já é exibido na tela. Além disso, quando o botão é clicado, o form ainda contém todos os campos em sua propriedade value. Assim, o two way data binding não faz com que o que aprendemos anteriormente deixe de funcionar.

Referências

Angular. 2020. Disponível em <https://angular.io>. Acesso em maio de 2020.