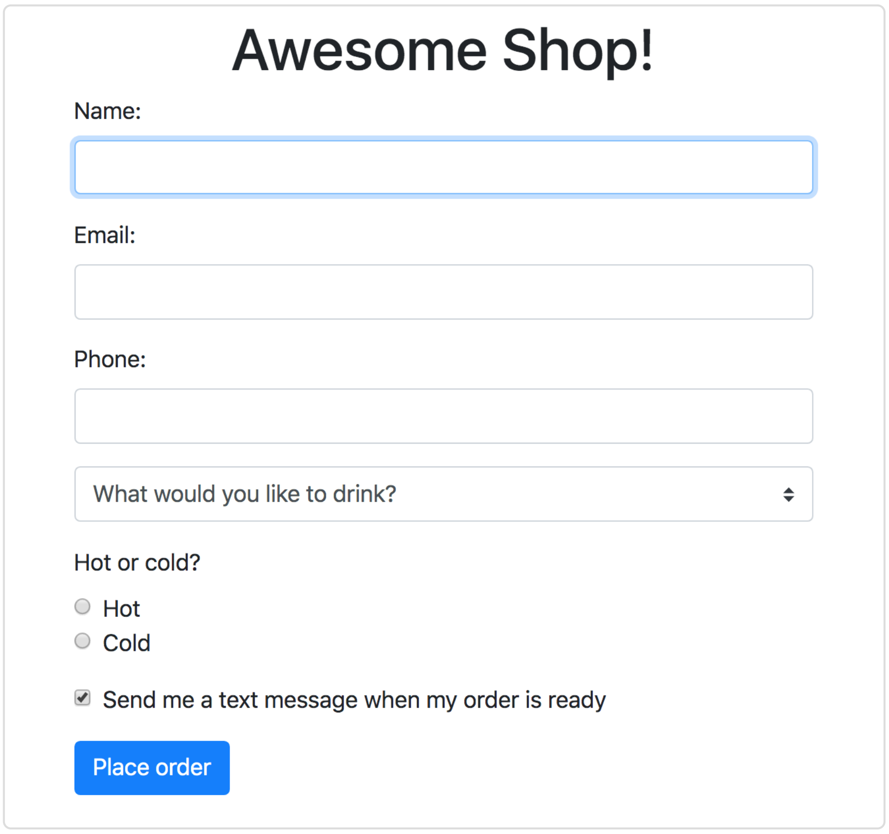
Exercício Aula 14


Awesome Shop!
Name:
Email:
Phone:
What would you like to drink?
Hot or cold?
Hot
Cold
Send me a text message when my order is ready
Place order

You may modify your interface the way you like. Get creative. Feel free to add additional elements you feel should be included and have fun!

Tasks:

1. **Create an Angular app**
   - Create a new app (sometimes referred to as a project). Use a terminal, navigate the the workspace directory where you want to work on the app.
     - Use a CLI `ng new` command to create a folder with the app name
       ```
       ng new exerc-aula14
       ```

2. **Start the development server**
   - Use a terminal, navigate to the workspace directory.
     ```
     cd exerc-aula14
     ```
   - Start the server
     ```
     ng serve
     ```
     This command will compile and build all files, and then start the server.

3. **Test access to the app**
   - Open a web browser, enter a URL
     ```
     http://localhost:4200
     ```
     Note: node.js server runs on port 4200, by default.

4. **Add bootstrap to the app**

- Go to *https://getbootstrap.com (https://getbootstrap.com/)*, select `Get started`, navigate to `CSS`, copy the stylesheet `<link>`
- In your `index.html`, add (paste) the stylesheet `<link>`

5. **Modify the code**
    - Navigate to the `src/app` folder
    - In `app.component.ts` file, update the `title` property to include the shop's name (you decide on the name of your shop).
    - In `app.component.ts` file, add a property (or properties) to include your name (and all team members).
    - In `app.component.html` file, use the concept of data binding to display the shop's name and your name (and all team members). You may add / update / remove the html code as you wish.

6. **Add a form to `app.component.html`**

```
<form>
  <div class="form-group">
    <label>Name: </label>
    <input type="text" class="form-control" name="name">
  </div>

  <div class="form-group">
    <label>Email: </label>
    <input type="email" class="form-control" name="email">
  </div>

  <div class="form-group">
    <label>Phone: </label>
    <input type="tel" class="form-control" name="phone">
  </div>

  <div class="form-group">
    <select class="custom-select" name="drink_option">
      <option value="">What would you like to drink? </option>
      <option> we will later use a directive to fill in the dropdown menu options </opti
    </select>
  </div>

  <div class="form-group">
    <label>Hot or cold?</label>
    <div class="form-check">
      <input class="form-check-input" type="radio" name="tempPreference" value="hot">
      <label class="form-check-label">Hot</label>
    </div>
    <div class="form-check">
      <input class="form-check-input" type="radio" name="tempPreference" value="cold">
      <label class="form-check-label">Cold</label>
    </div>
  </div>

  <div class="form-check mb-3">
    <input class="form-check-input" type="checkbox" name="sendmsg">
    <label class="form-check-label">Send me a text message when my order is ready </labe
  </div>

  <button class="btn btn-primary">Place order</button>
</form>
```

7. **Create a drink option**
   - In `app.component.ts`, add a drink property and assign a pre-defined list of drink options in a class declaration (i.e., in `export class AppComponent`); for example,

     ```
     drinks = ['Coffee', 'Tea', 'Milk'];
     ```

   - In `app.component.html`, use an `ngFor` directive and the concept of data binding to fill in the drink dropdown menu options.

8. **Experience with data binding**

- To bind data from a form, (in `app.component.html`) attach a template reference (`#orderForm`) to the `<form>` tag and assign an `ngForm` directive to it

  ```
  <form #orderForm="ngForm">
  ```

  Notice that there is a compilation error since `ngForm` is in a `FormsModule` of `@angular/forms` but it has not been imported into an app.

- In `app.module.ts`, import `FormsModule`

  ```
  import { FormsModule } from '@angular/forms';
  ```

  and add a `FormsModule` to an `imports` array of the `@NgModule` declarator. It should look similar to

  ```
  imports: [
     BrowserModule,
     FormsModule,
  ],
  ```

- In `app.component.html`, add the following code to bind data from the `orderForm` to view

  ```
  {{ orderForm.value | json }}
  ```

  or

  ```
  {{ orderForm.value.email }}
  ```

- In `app.component.html`, add `ngModel` attribute to the form controls

9. **Create a class representing an order model**
   - Navigate to the root directory of your project (i.e, `exerc-aula14`)
   - Create a class named `order`

     `ng g class order` or `ng generate class order`

   - Navigate to `src/app`, in `order.ts`, add properties to the class constructor

     ```
     export class Order {
        constructor(
           public name: string,
           public email: string,
           public phone: number,
           public drink: string,
           public tempPreference: string,
           public sendText: boolean
        ){}
     }
     ```

   - In `app.component.ts`, import an `Order` class

     ```
     import { Order } from './order';
     ```

   - Let's create an instance of an order model, we will then bind data from the model to view. In a class declearation (in `app.component.ts`), add the following code

```
orderModel = new Order('someone', 'someone@uva.edu', 9991234567, '', '', true);
```

- In `app.component.html`, add the following code to bind data from `orderModel` to view

  ```
  {{ orderModel | json }}
  ```

10. **Bind an order model to the form** such that existing data are populated on form `load`
    - In `app.component.html`, use a `[ ]` to bind property of of an `ngModel` of the form controls. For example, for a name input box,

      ```
      [ngModel]="orderModel.name"
      ```

    - Observe the binding recently added and the binding from step 8. Notice that they are both one-way binding. Changes to the view do not reflect the model; changes to the model do not reflect the form. In practice, both view and model should be synchronized; i.e., two-way binding.

    - Modify the form controls in `app.component.html` such that view and model are synchronized. For example, update the `ngModel` of a name input box as

      ```
      [(ngModel)]="orderModel.name"
      ```