Desenvolvimento web com Angular

Jackson Gomes \ jgomes@ceulp.edu.br

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Códigos-fontes

Prefácio

Este é um livro open-source, tanto no conteúdo quanto no código-fonte associado. O conteúdo é resultado de algumas práticas com o **Framework Angular** e segue uma abordagem prática, com foco no entendimento de conceitos e tecnologias no contexto do desenvolvimento de um software web.

Um *framework* representa um modelo, uma forma de resolver um problema. Em termos de desenvolvimento de software para a web um framework fornece ferramentas (ie. código) para o desenvolvimento de aplicações. Geralmente o propósito de um framework é agilizar as atividades de desenvolvimento de software, inclusive, fornecendo código pronto (componentes, bibliotecas etc.) para resolver problemas comuns, como uma interface de cadastro.

O objetivo deste livro é fornecer uma ferramenta para o desenvolvimento de habilidades de desenvolvimento web com Angular, com a expectativa de que você comece aprendendo o básico (o "hello world") e conclua com habilidades necessárias para o desenvolvimento de software que consome dados e interage com uma API HTTP REST, por exemplo.

0.1 Fonte de referência e versão do Angular

Parte do conteúdo do livro é baseada na documentação oficial do Angular, disponível em https://angular.io.

Como o Angular é um projeto em constante desenvolvimento (pelo menos até agora) serão publicadas atualizações no conteúdo do livro sempre que possível, para refletir novos recursos e funcionalidades. No momento, o conteúdo do livro é baseado na versão **6.0.0**.

0.2 Convenções

Os trechos de código apresentados no livro seguem o seguinte padrão:

- **comandos**: devem ser executados no prompt; começam com o símbolo `$`
- **códigos-fontes**: trechos de códigos-fontes de arquivos

A seguir, um exemplo de comando:

```
$ mkdir hello-world
```

O exemplo indica que o comando `mkdir`, com a opção `hello-world`, deve ser executado no prompt para criar uma pasta com o nome `hello-world`.

A seguir, um exemplo de código-fonte:

```
1 class Pessoa:
2     pass
```

O exemplo apresenta o código-fonte da classe `Pessoa`. Em algumas situações, trechos de código podem ser omitidos ou serem apresentados de forma incompleta, usando os símbolos `...` e `#`, como no exemplo a seguir:

```
1 class Pessoa:
2     def __init__(self, nome):
3         self.nome = nome
4 
5     def salvar(self):
6         # executa validação dos dados
7         ...
8         # salva
9         return ModelManager.save(self)
```

0.3 Conhecimentos desejáveis

Este livro aborda o desenvolvimento de software front-end para a web do ponto-de-vista do Angular. Isso quer dizer que não trata de conceitos iniciais de HTML, CSS, JavaScript, TypeScript e Bootstrap. Entretanto, os conceitos fundamentais dessas tecnologias vão sendo apresentados no decorrer dos capítulos, conforme surge a necessidade deles.

Para aprender mais sobre essas tecnologias recomendo essas fontes:

- **TypeScript**: Documentação oficial do TypeScript - Microsoft, TypeScript Deep Dive
- **HTML, CSS e JavaScript:** W3Schools
- **Boostrap**: Documentação oficial do Bootstrap

Este livro não leva em consideração o Sistema Operacional do seu ambiente de desenvolvimento, mas é importante que você se acostume a certos detalhes e a certas ferramentas, como o **prompt** ou **prompt de comando**.

Além destas ferramentas também são utilizadas:

- **Node.js**: disponível em https://nodejs.org representa um ambiente de execução do JavaScript fora do browser e também inclui o **npm**, um gerenciador de pacotes

- **Editor de textos ou IDE**: atualmente há muitas opções, mas destaco o **VisualStudio-Code**, disponível em https://code.visualstudio.com/
- **Git**
- **Heroku**

O **Git** é um gerenciador de repositórios com recursos de versionamento de código. É uma ferramenta essencial para o gerenciamento de código fonte de qualquer software.

O **Heroku** é um serviço de **PaaS** (de *Platform-as-a-Service*). PaaS é um modelo de negócio fornece um ambiente de execução conforme uma plataforma de programação, como o Python, um tecnologia de banco de dados, como MySQL e PostgreSQL e ainda outros recursos, como cache usando Redis.

Calma! Não pira!

(*In*)Felizmente você não vai usar todas as tecnologias lendo o conteúdo desse livro. Fica para outra oportunidade.

Para utilizar o Heroku você precisa criar uma conta de usuário. Acesse https://www.heroku.com/ e crie uma conta de usuário.

Depois que tiver criado e validado sua conta de usuário instale o **Heroku CLI**, uma ferramenta de linha de comando (prompt) que fornece uma interface de texto para criar e gerenciar aplicativos Heroku. Detalhes da instalação dessa ferramenta não são tratados aqui, mas comece acessando https://devcenter.heroku.com/articles/heroku-cli.

Capítulo 1

Introdução

1.1 Servidor web

Um **servidor web** é um programa que fornece um serviço de rede que funciona recebendo e atendendo requisições de clientes. Um **cliente**, por exemplo, é o browser.

Um **cliente** solicita um arquivo ao **servidor web**, que recebe a solicitação, atende a solicitação e retorna uma resposta para o cliente.

Esse modelo é chamado **cliente-servidor** e, na web, utiliza o protocolo **HTTP** (de *Hypertext Transfer Protocol*). O protocolo HTTP determina as regras da comunicação no modelo cliente-servidor:

- como o cliente deve enviar uma solicitação para o servidor
- como o servidor deve interpretar a solicitação
- como o servidor deve enviar uma resposta para o cliente
- como o cliente deve interpretar a resposta do servidor

Para ilustrar esse processo a Figura 1.1 demonstra a comunicação entre cliente e servidor.

Como a Figura 1.1 apresenta, quem inicia a comunicação é o cliente. O servidor recebe a solicitação e retorna uma resposta. A resposta pode ser interpretada como sucesso ou erro. No caso da figura, se o servidor encontrar o arquivo, ele retorna um código de resposta do HTTP com o número 200 e o conteúdo HTML do arquivo `index.html`, caso contrário ele retorna um código de resposta HTTP com o número 404, indicando que o arquivo não foi encontrado.

1.2 Desenvolvimento front-end

O termo **front-end** no contexto do desenvolvimento do software tem relação com a utilização de tecnologias e ferramentas para o desenvolvimento de software que, geralmente, executa em um cliente. Considerando o cenário anterior, da comunicação **cliente-servidor**, estamos falando

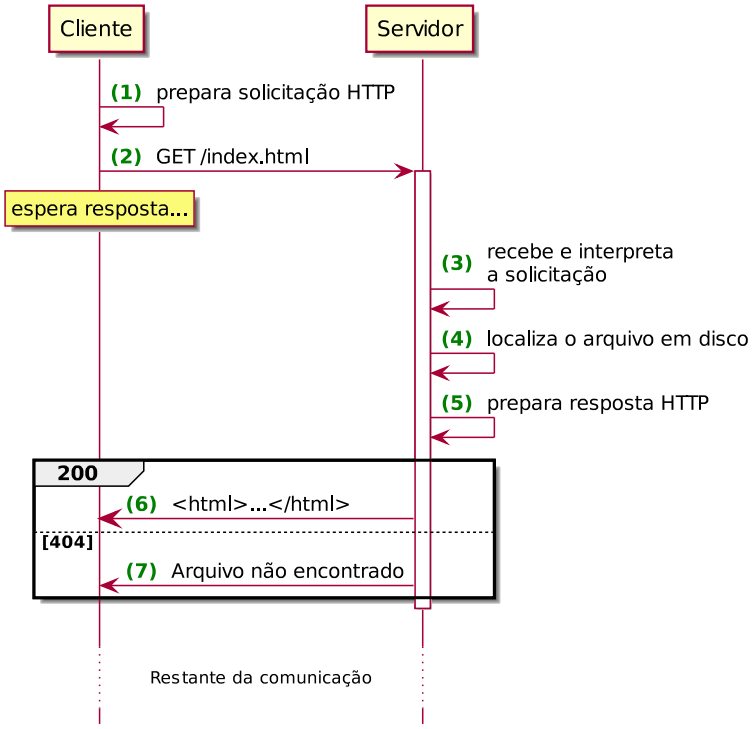


Figura 1.1: Exemplo de comunicação cliente-servidor

justamente do **browser**. O **browser** se torna uma peça fundamental nesse tipo de desenvolvimento de software.

Grande parte do desenvolvimento front-end se direciona para a tríade composta por HTML, CSS e JavaScript:

- **HTML** sendo utilizada como linguagem de marcação
- **CSS** sendo utilizada como linguagem de formatação
- **JavaScript** sendo utilizada como linguagem para adicionar interação (lógica de interface e lógica de negócio)

Para exemplificar, veja o projeto **noticias-js**. **noticias-js** é um software de gerenciamento de notícias com repositório em https://github.com/jacksongomesbr/webdevbook-noticias-js, desenvolvido em HTML, CSS e JavaScript e possui as seguintes funcionalidades:

- cadastrar notícia (título e conteúdo)
- ver a lista de notícias (título)
- ver o conteúdo de uma notícia (clicando no título)

Figuras 1.2, 1.3 ilustram o software e essas funcionalidades.

Esse comportamento já não é novidade em software web: a interface com o usuário permite a entrada de dados e a interação por meio de cliques. Os detalhes para fazer esse comportamento estão na utilização de JavaScript. Primeiro, a estrutura do software é baseada em três partes:

- `index.html`: contém o HTML para a marcação
- `main.css`: contém o CSS para a formatação

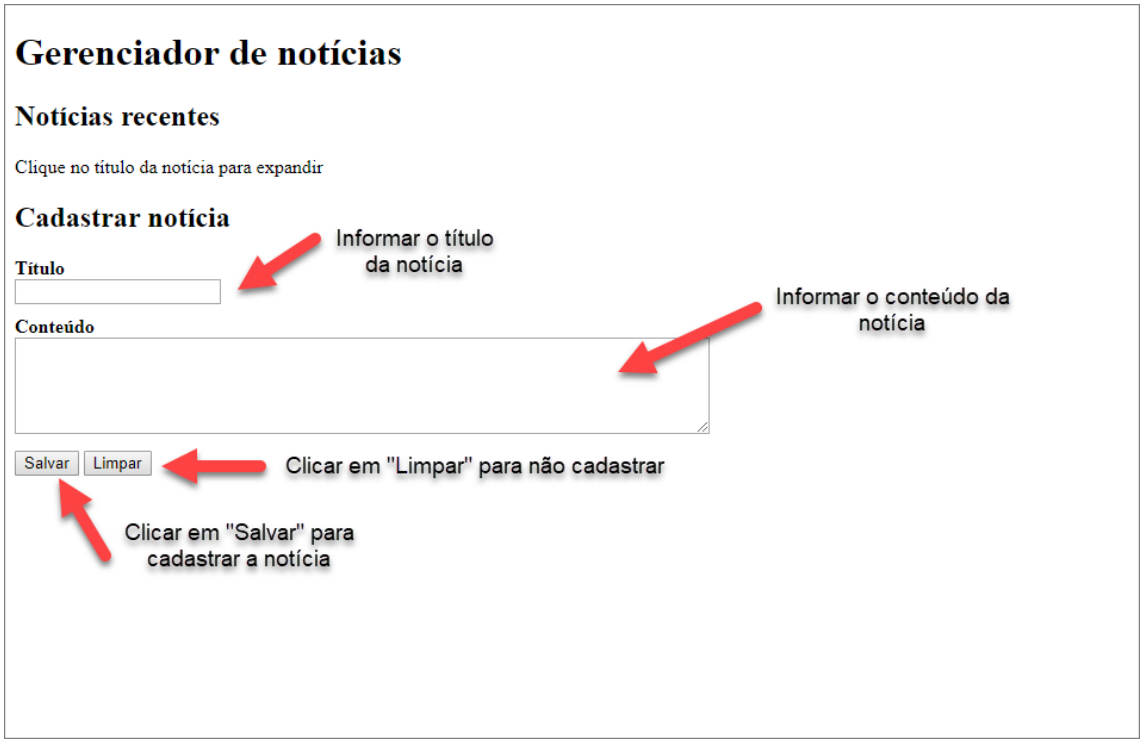


Figura 1.2: Tela de cadastro de notícias no software noticias-js

Figura 1.3: Tela de lista de notícias no software noticias-js

- `main.js`: contém o JavaScript para implementação da lógica da interface

A seguir, as seções demonstram detalhes dessa estrutura.

1.2.1 HTML para a marcação

Código-fonte 1.1 apresenta um trecho do arquivo `index.html`.

Código-fonte 1.1: Trecho do arquivo index.html do software noticias-js

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      ...
5      <link rel="stylesheet" type="text/css" media="screen" href="main.css"
           />
6      <script src="main.js"></script>
7  </head>
8  <body>
9      <h1>Gerenciador de notícias</h1>
10     <h2>Notícias recentes</h2>
11     <p>Clique no título da notícia para expandir</p>
12     <div id="noticias-recentes">
13         <ul id="noticias-recentes-list"></ul>
14     </div>
15
16     <h2>Cadastrar notícia</h2>
17     <form onsubmit="salvar(this); return false;">
18         <div>
19             <label for="frm-titulo">Título</label>
20             <input type="text" id="frm-titulo" name="titulo" required>
21         </div>
22         <div>
23             <label for="frm-conteudo">Conteúdo</label>
24             <textarea id="frm-conteudo" name="conteudo" cols="80" rows="5"
                   required></textarea>
25         </div>
26         <div>
27             <button type="submit">Salvar</button>
28             <button type="reset" formnovalidate>Limpar</button>
29         </div>
30     </form>
31 </body>
32 </html>
```

A primeira parte importante é o elemento `ul` com identificador (atributo `id`) `noticias-recentes-list`. A importância se dá para o fato de que esse identificador será utilizado no código JavaScript

para adicionar elementos `li`, um recurso chamado de **manipulação do HTML DOM**. Outra parte importante é em relação ao formulário de cadastro. Primeiro, o elemento `form` possui o atributo `onsubmit` com um valor que é um código JavaScript. Depois, cada campo do formulário está declarado para receber entrada do usuário:

- `input` com identificador `frm-titulo` é usado para o título da notícia
- `textarea` com identificador `frm-conteudo` é usado para o conteúdo da notícia

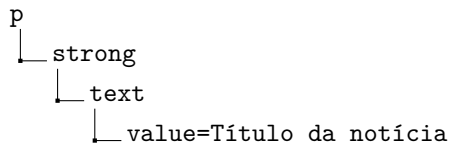
Todos os campos são de preenchimento obrigatório, então está sendo utilizado um recurso de validação diretamente no HTML por meio do atributo `required`. Por fim, o formulário tem dois botões (elemento `button`): "Salvar", que tem o atributo `type` com valor `submit` e "Limpar", que além de ter `type` com valor `reset` tem o atributo `formnovalidate`, que é utilizado para desabilitar a validação do formulário no clique do botão (é o comportamento padrão).

1.2.2 Manipulação do DOM

O DOM (de *Document Object Model*) é uma representação em memória de um documento HTML, na forma de uma **árvore** composta por nós que correspondem aos elementos do documento HTML. Por exemplo, para um trecho HTML como o seguinte:

```
<p><strong>Título da notícia</strong></p>
```

a árvore teria a seguinte estrutura:



Por causa da estrutura em árvore é possível identificar relações entre os elementos, por exemplo:

- a raiz da árvore é o nó `p`
- o nó `p` tem um filho, o nó `strong`
- o nó `strong` tem um pai, o nó `p`
- o nó `strong` tem um filho, o nó `text`
- o nó `text` tem um filho, o valor `Titulo da notícia`

Dessa forma grande parte da responsabilidade de **manipulação do DOM** recai sobre tarefas como encontrar um nó, percorrer filhos, adicionar filho em um nó e remover um nó. Para isso o DOM fornece objetos (princpalmente o `document`), métodos e propriedades, que serão apresentados a seguir.

1.2.3 CSS para formatação

Usar CSS para formatação corresponde a criar **regras CSS** e definir como elas serão aplicadas a determinados elementos do documento HTMl por meio dos **seletores**. Uma regra CSS é um

conjunto composto por pares **propriedade**:**valor**. O seletor informa para o browser como encontrar elementos para aplicar as propriedades. Há seletores: de elemento, de id e de classe. O seletor de elemento contém o nome do elemento. O setor de id contém o símbolo `#` seguido de um identificador de elemento (valor do atributo `id`). O seletor de classe contém o símbolo `.` seguido de uma classe de elemento (um dos valores do atributo `class`).

A tela do software usa formatação em CSS, conforme mostra Código-fonte 1.2.

Código-fonte 1.2: Trecho do arquivo main.css do software noticias-js

```
1  label {
2      display: block;
3      font-weight: bold;
4  }
5
6  form div {
7      margin-bottom: 10px;
8  }
9
10 .noticia .titulo {
11     font-weight: bold;
12 }
13
14 .noticia .conteudo {
15     display: none;
16 }
```

Há quatro grupos de regras, com seletores diferentes:

- `label`: aplica propriedades `display` e `font-height` para todos os elementos `label`
- `form div`: aplica propriedade `margin-bottom` para todos os elementos `div` dentro de elementos `form`
- `.noticia .titulo`: aplica propriedade `font-weight` para todos os elementos que tenham atributo `class` contendo `titulo` dentro de elementos que tenham atributo `class` contendo `noticia`
- `.noticia .conteudo`: aplica propriedade `display` para todos os elementos que tenham atributo `class` contendo `noticia` dentro de elementos que tenham atributo `class` contendo `noticia`

A propriedade `display` com valor `none` é importante porque é utilizada para ocultar o conteúdo da notícia.

As regras CSS são aplicadas **em cascata** o que significa que há uma ordem de prioridade que o browser considera para resolver conflitos de estilos:

1. estilo **in-line** (definido no atributo `style` do elemento em questão)
2. estilo definido no elemento `style`
3. estilo definido em um arquivo `.css` externo (obtido por meio do elemento `link`)

Aprender a utilizar os seletores é uma parte importante do trabalho com CSS.

1.2.4 JavaScript para lógica

O código JavaScript é parcialmente ilustrado por Código-fonte 1.3.

Código-fonte 1.3: Trecho do arquivo main.js do software noticias-js

```
1  var noticias = [];
2
3  function atualizarLista(noticia) {
4  }
5
6  function salvar(form) {
7  }
8
9  function mostrarNoticia(id) {
10 }
11
12 function ocultarNoticia(id) {
13 }
```

O código foi apresentado parcialmente para um entendimento inicial da sua estrutura. A variável `noticias` é um `Array`, utilizado para armazenar objetos que representam as notícias cadastradas. Dessa forma o conteúdo está **apenas em memória** ou **em tempo de execução**. Quando a página é recarregada, o conteúdo é perdido. Na sequência são declaradas quatro funções: `atualizarLista()`, `salvar()`, `mostrarNoticia()` e `ocultarNoticia()`.

A função `salvar()` é chamada por meio de um tratador de evento. No código HTML ([lst:noticias-js-html]), no elemento `form` o atributo `onsubmit` representa um tratador de evento, que é ativado quando algum botão dentro do formulário é clicado (nesse caso, queremos que o botão "Salvar" ative esse tratador de evento). O conteúdo de um tratador de evento é um código JavaScript e, nesse caso, há duas instruções:

- chamar a função `salvar()` passando como argumento `this` (que é uma referência ao objeto DOM que representa o formulário HTML)
- cancelar o evento ao chamar `return false`

A seguir, o código completo da função `salvar()`:

```
function salvar(form) {
    var titulo = document.getElementById('frm-titulo').value;
    var conteudo = document.getElementById('frm-conteudo').value;
    var noticia = {
        id: noticias.length,
        titulo: titulo,
        conteudo: conteudo
    };
    noticias.push(noticia);
```

```
10      atualizarLista(noticia);
11      form.reset();
12  }
```

O função `salvar()` tem o parâmetro `form`, que recebe o argumento usado na chamada da função, no tratador de evento `onsubmit`. O interior do código tem duas linhas importantes, que interagem com o HTML DOM para obter valores dos campos do formulário. Isso é feito por meio do método `getElementById()` do objeto `document`, que procura um elemento no documento HTML cujo identificador seja igual ao argumento (`frm-titulo`, por exemplo) e retorna um objeto do DOM que representa o elemento. Por ser um campo de formluário, a propriedade `value` retorna o valor digitado pelo usuário.

Na sequência o código cria um objeto `noticia` com três atributos:

- `id`: que representa um identificador numérico da notícia (começando em zero)
- `titulo`: representa o título da notícia
- `conteudo`: representa o contéudo da notícia

Depois, a sequência continua:

- o objeto `noticia` é adicionado no `Array noticias` por meio de uma chamada ao método `push()`
- chama a função `atualizarNoticia()` (descrita a seguir), informando como argumento o objeto `noticia` para que a notícia que acaba de ser cadastrada seja apresentada na lista
- chama o método `reset()` do objeto `form`, que é utilizado para redefinir os valores dos campos do fomulário

A seguir, o código completo da função `atualizarList()`:

```
1  function atualizarLista(noticia) {
2      var lista = document.getElementById('noticias-recentes-list');
3      var li = document.createElement('li');
4      li.setAttribute('id', 'noticia-' + noticia.id);
5      li.setAttribute('class', 'noticia');
6      li.innerHTML = '<p class="titulo" onclick="mostrarNoticia(' + noticia.
           id + ')">'
7          + noticia.titulo
8          + '</p>'
9          + '<p class="conteudo">'
10         + noticia.conteudo
11         + '<br>'
12         + '<span>------------------</span>'
13         + '<br>'
14         + '<button onclick="ocultarNoticia(' + noticia.id + ')">Fechar</
               button>';
15         + '</p>';
16     lista.appendChild(li);
17 }
```

O código utiliza o método `getElementById()` para obter uma referência para o objeto com iden-

tificador `noticias-recentes-list`, que representa o elemento `ul` que contém elementos `li` para apresentar a lista de notícias. A partir de então o objetivo do código é criar um elemento `li` e adicioná-lo ao elemento `ul`. Para isso, começa criando um elemento no DOM por meio do método `createElement()`, cujo argumento `"li"` representa o nome do elemento criado. Essa referência é mantida na variável `li` para o código da sequência:

- utiliza o método `setAttribute()` para definir o valor do atributo `id` (baseado no identificador da notícia)
- utiliza o método `setAttribute()` para definir o valor do atributo `class`
- utiliza a propriedade `innerHTML` para definir o restante do conteúdo HTML

Essa última parte, do valor de `innerHTML` merece destaque. A manipulação do DOM do HTML pode ser feita utilizando métodos (como `getElementById()` e `createElement()`) e também fazendo um **parser** de um conteúdo HTML. Nesse caso, por se tratar de um contéudo mais longo, o código utiliza a segunda opção. Perceba que o conteúdo da propriedade, uma `string`, é conteúdo HTML, que é interpretado pelo **browser** para modificar o DOM do HTML.

Outra parte importante desse trecho de HTML representado na `string` é sua estrutura:

- elemento `p` com atributo `class` contendo `titulo` e atributo `onclick` (tratador de evento para clique)
- título da notícia
- elemento `p` com atributo `class` contendo `conteudo`
- conteúdo da notícia
- elemento `br`
- elemento `span` contendo traços
- elemento `br`
- elemento `button` com rótulo “Fechar” e atributo `onclick`

O atributo `onclick` representa o tratador de evento para clique. Nesse caso, o elemento `p` que contém o título da notícia tem um tratador de evento que chama a função `mostrarNoticia()`. O botão “Fechar” tem o tratador de evento que chama a função `ocultarNoticia()`. Por fim, o elemento `li` é adicionado na lista de filhos do objeto `lista` por meio do método `appendChild()`.

A seguir, o código da função `mostrarNoticia()`:

```
1  function mostrarNoticia(id) {
2      var li = document.getElementById('noticia-' + id);
3      for (var i = 0; i < li.childNodes.length; i++) {
4          var node = li.childNodes[i];
5          if (node.getAttribute('class') == 'conteudo') {
6              node.setAttribute('style', 'display:inline');
7          }
8      }
9  }
```

A função `mostrarNoticia()` recebe o parâmetro `id`, que representa o identificador da notícia que cujo conteúdo deve ser apresentado. O código opera da seguinte forma:

- encontra o elemento `li` cujo identificador corresponde ao parâmetro `id`
- para cada nó filho do elemento `li` (usa a propriedade `childNodes`):
  - se o nó filho (objeto `node`) tiver atributo `class` com o valor `'conteudo'` (usa o método `getAttribute()`) então
    - define o valor do atributo `style` com `'display:inline'`, o que faz com que ele se torne visível (contrário de `display:none`)

De forma semelhante, a função `ocultarNoticia()` recebe o parâmetro `id`, que representa o identificador da notícia cujo conteúdo deve ser ocultado:

```
1  function ocultarNoticia(id) {
2      var li = document.getElementById('noticia-' + id);
3      for (var i = 0; i < li.childNodes.length; i++) {
4          var node = li.childNodes[i];
5          if (node.getAttribute('class') == 'conteudo') {
6              node.setAttribute('style', 'display:none');
7          }
8      }
9  }
```

A principal diferença para a função `mostrarNoticia()` é que a a função `ocultarNoticia()` modifica o atributo `style` para o valor `display:none`, o que torna o conteúdo invisível novamente, completando, assim, a interação com o usuário.

Certamente esse não é um software simples para quem tem a primeira experiência com esse tipo de programação, mas é importante destacar esses aspectos:

- a estrutura do HTML é criada tendo em vista possibilitar a **manipulação do DOM** com o JavaScript (por isso o uso de valores controlados para os atributos `id` e `class`)
- o atributo `onclick` é um tratador de evento para clique
- o atributo `onsubmit` é um tratador de evento para o envio do formulário
- o atributo `formnovalidate` impede a validação do formulário
- o objeto `document` dá acesso ao DOM do HTML e permite usar as funções para manipulação do DOM
- o método `getElementById()` encontra um nó do DOM com base em um identificador (atributo `id`)
- o método `setAttribute()` cria ou altera o valor de um atributo de um nó
- o método `getAttribute()` retorna o valor de um atributo de um nó
- a propriedade `innerHTML` permite fazer parser de um trecho de HTML e inserir o resultado na árvore DOM
- o método `appendChild()` adiciona um nó na lista de nós filhos do nó pai
- a propriedade `childNodes` contém a lista de nós filhos do nó pai (é um `Array`)

1.3 jQuery

O **jQuery** é uma das primeiras **bibliotecas JavaScript** e foi criada para evitar uma quantidade enorme de retrabalho e verificações de suporte de diferentes versões e tipos de browser e também inclui funções para manipulação do DOM (THE JQUERY FOUNDATION, [s.d.]).

O repositório no **noticias-js** tem um branch `jquery`, que contém a implementação utilizando a biblioteca jQuery. Uma lista completa das diferenças entre o branch `master` e o `jquery` pode ser obtida em https://github.com/jacksongomesbr/webdevbook-noticias-js/compare/jquery. Na prática, as principais modificações estão no arquivo `main.js`, com detalhes para as implementações das funções. Começando pela função `salvar()` temos o seguinte:

```
function salvar(form) {
    var titulo = $('#frm-titulo').val();
    var conteudo = $('#frm-conteudo').val();
    ...
}
```

O código em ... não muda em relação ao branch `master`. As variáveis `titulo` e `conteudo` continuam recebendo os valores informados pelo usuário no formulário, mas agora utilizam a função `$()`, que é a principal função do jQuery e, nesse caso, acessa a árvore DOM em busca de elementos com is identificadores indicados por seletores CSS de id: `#frm-titulo` e `#frm-conteudo` encontram, respectivamente, os elementos com identificador `frm-titulo` e `frm-conteudo`. O valor dos campos é obtido pela função `val()`.

Já a função `atualizarLista()` muda bastante:

```
function atualizarLista(noticia) {
    var lista = $('#noticias-recentes-list');
    var li = $('<li id="noticia-' + noticia.id + '">');
    li.addClass('noticia');
    var p_titulo = $('<p>');
    p_titulo.addClass('titulo');
    p_titulo.attr('onclick', 'mostrarNoticia(' + noticia.id + ')');
    p_titulo.html(noticia.titulo);
    var p_conteudo = $('<p>');
    p_conteudo.addClass('conteudo');
    p_conteudo.html(noticia.conteudo
        + '<br>'
        + '<span>------------------</span>'
        + '<br>'
        + '<button onclick="ocultarNoticia(' + noticia.id + ')">Fechar</
            button>');
    li.append(p_titulo, p_conteudo);
    p_conteudo.hide();
    lista.append(li);
}
```

A variável `lista` representa o elemento do DOM com identificador `noticias-recentes-list`. A variável `li` recebe a chamada da função `$()` com uma string HTML como parâmetro (linha 3). Nesse caso, o jQuery cria uma árvore parcial do DOM fazendo **parser** do argumento (como acontece com a propriedade `innerHTML`). Uma classe CSS é adiciona no nó por meio do método `addClass()` (linha 4). Um atributo é adicionado ou alterado por meio do método `attr()` (linha 7). O conteúdo de um nó pode ser definido usando o método `html()` (como com a propriedade `innerHTML`), na linha 8. O método `append()` é utilizado para adicionar um nó na lista de filhos de um pai (linha 15). Por fim, o jQuery tem um modo próprio de esconder e mostrar elementos usando, respectivamente, os métodos `hide()` e `show()`. Esses métodos também são usados nas implementações das funções `ocultarNoticia()` e `mostrarNoticia()`, que se tornam:

```
1 function mostrarNoticia(id) {
2     $('.conteudo', '#noticia-' + id).show();
3 }
4
5 function ocultarNoticia(id) {
6     $('.conteudo', '#noticia-' + id).hide();
7 }
```

A parte importante fica por conta da chamada da função `$()`. Nesse caso há dois argumentos:

1. o seletor de classe `.conteudo`
2. o contexto, que usa um seletor de id (`#noticia-` seguido do identificador da notícia)

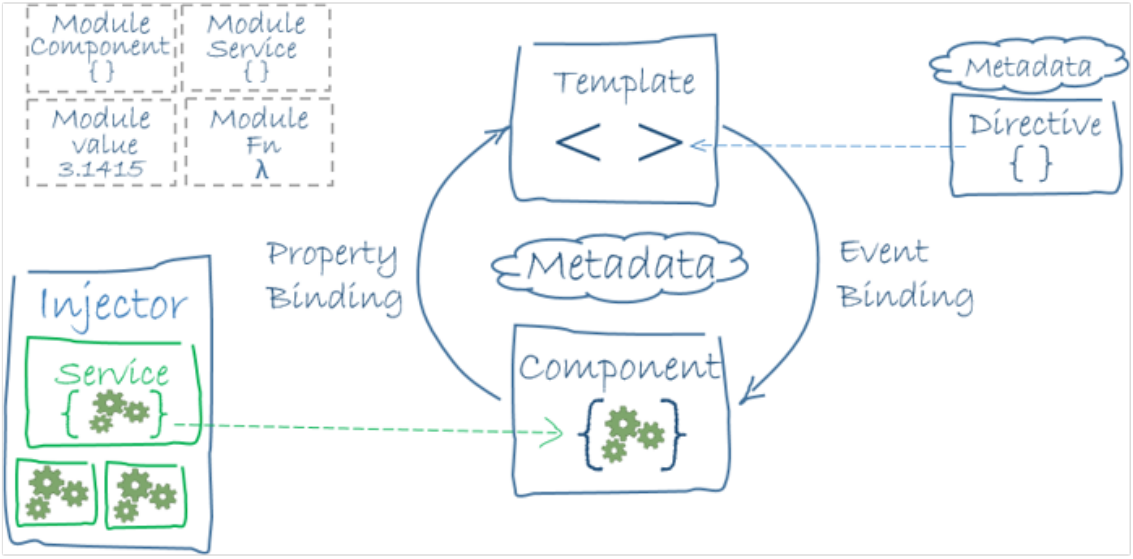
Na prática, o jQuery fornece novas possibilidades de manipulação do DOM e, nesse caso, é utilizado para encontrar um elemento que tenha a classe CSS `conteudo` e esteja dentro de um elemento cujo identificador combina com o da notícia em questão (para ter o conteúdo apresentado ou ocultado).

Capítulo 2

Iniciando com o Angular

O **Angular** é um framework para o desenvolvimento de software front-end. Isso quer dizer que utiliza tecnologias padrão do contexto web como HTML, CSS e uma linguagem de programação como JavaScript ou TypeScript (GOOGLE, [s.d.]).

Um software desenvolvido em Angular é composto por diversos elementos como: módulos, componentes, templates e serviços. Esses elementos fazem parte da arquitetura do Angular, que é ilustrada pela figura a seguir.



Essa arquitetura de software orientada a componentes implementa conceitos de dois padrões de arquitetura de software:

- **MVC** (de *Model, View, Controller*) é um padrão de software que separa a representação da informação (Model) da interação do usuário com ele (View-Controller). Geralmente, Model e Controller são representados por código em linguagem de programação (classes e/ou funções) e View é representado por HTML e CSS (WIKIPEDIA CONTRIBUTORS, 2018a).
- **MVVM** (de *Model, View, View-Model*) é um padrão de software semelhante ao MVC, com a diferença de que o View-Model utiliza recurso de **data binding** (mais sobre isso depois)

para fazer com que a View seja atualizada automaticamente quando ocorrer uma modificação no Model (WIKIPEDIA CONTRIBUTORS, 2018b).

No contexto do Angular esses elementos são descritos conforme as seções a seguir.

2.1 Elementos da Arquitetura do Angular

2.1.1 Módulos

Módulos representam a forma principal de modularização de código. Isso significa que um módulo é um elemento de mais alto nível da arquitetura do Angular e é composto por outros elementos, como componentes e serviços.

Um software desenvolvido em Angular possui pelo menos um módulo, chamado **root module** (módulo raiz). Os demais módulos são chamados **feature modules** (módulos de funcionalidades).

2.1.2 Bibliotecas

Bibliotecas funcionam como um agrupador de elementos de software desenvolvido em Angular. Bibliotecas oficiais têm o prefixo `@angular`. Geralmente é possível instalar bibliotecas utilizando o **npm** (gerenciador de pacotes do **Node.Js**).

Uma biblioteca pode conter módulos, componentes, diretivas e serviços.

2.1.3 Componentes

Um componente está, geralmente, relacionado a algo visual, ou seja, uma tela ou parte dela. Nesse sentido, um componente possui código (**Controller**) que determina ou controla o comportamento da interação com o usuário (**View** ou **Template**).

O **Template** determina a parte visual do componente e é definido por código HTML e CSS, além de recursos específicos do Angular, como outros componentes e diretivas.

2.1.4 Metadados

Metadados são um recurso do Angular para adicionar detalhes a classes. Isso é utilizado para que o Angular interprete uma classe como um Módulo ou como um Componente, por exemplo.

Os metadados utilizam o conceito de **decorators** do TypeScript, que permitem formas de adicionar informações (metadados) a classes e membros de classe (atributos e métodos) (MICROSOFT, [s.d.]).

2.1.5 Data binding

Data binding (que seria algo como "vinculação de dados" em português) é um reucrso do Angular que representa um componente importante da sua arquitetura. Considere os seguintes elementos:

- um Model define dados que serão apresentados no Template
- um Template apresenta os dados do Model
- um Controller ou um View-Model determina o comportamento do Template

Se o Controller atualiza o Model, então o Template tem que ser atualizado automaticamente. Se o usuário atualiza o Model por meio do Template (usando um formulário, por exemplo) o Controller também precisa ter acesso ao Model atualizado. O Data Binding atua garantindo que esse processo ocorra dessa forma.

A **sintaxe de data binding** é o mecanismo que controla a ordem entre fonte e destino dos dados:

- **one-way de fonte para view**: pode ser de cinco tipos: **Interpolation**, **Property**, **Attribute**, **Class** e **Style**. A sintaxe de interpolação é `{{expressao}}` para apresentar o valor de `expressao` no Template ou `[target]="expressao"` para que `target` (por exemplo, uma propriedade) receba o valor de `expressao`
- **one-way de view para destino**: pode ser do tipo **Event**. A sintaxe de evento é `(evento)="instrucao"`, indica para o Angular que deve executar `instrucao` no tratador do `evento`
- **two-way**: a sintaxe é `[(alvo)]="expressao"`

2.1.6 Diretivas

Diretivas representam um conceito do Angular que é um pouco confuso. Na prática, um Componente é uma Diretiva com um Template. Assim, um Componente é um tipo de Diretiva, que nem sempre está relacionada a algo visual. Angular define dois tipos de diretivas:

- **Diretivas Estruturais**: modificam o Template dinamicamente por meio de manipulação do DOM, adicionando ou removendo elementos HTML
- **Diretivas de Atributos**: também modificam o Template, mas operam sobre elementos HTML já existentes

2.1.7 Serviços

Um Serviço é uma abstração do Angular utilizado para isolar a lógica de negócio de Componentes. Na prática, um Serviço é representado por uma classe com métodos que podem ser utilizados em Componentes. Para isso, para que um Componente utilize um serviço, o Angular utiliza o conceito de **Injeção de Dependência** (DI, do inglês **Dependency Injection**). DI é um padrão de software que faz com que dependências sejam fornecidas para quem precisar. Na prática, o Angular identifica as dependências de um Componente e cria automaticamente instâncias delas, para que sejam utilizadas posteriormente no Componente (WIKIPEDIA CONTRIBUTORS, 2018c).

Enquanto esses elementos da Arquitetura do Angular representam conceitos, é importante visualizar a relação deles com elementos práticos do software, ou seja, código. Para isso, a seção a seguir

apresenta a estrutura padrão de um software desenvolvido em Angular.

2.2 Estrutura padrão de um software desenvolvido em Angular

Um software desenvolvido em Angular é representado por vários arquivos HTML, CSS, TypeScript e de configuração (geralmente arquivos em formato **JSON**).

```
e2e
node_modules
src
.editorconfig
.gitignore
angular.json
package.json
package-lock.json
README.md
tsconfig.json
tslint.json
```

No diretório raiz do software:

- `e2e`: contém especificações de testes **end-to-end**
- `node_modules`: contém os pacotes (dependências)
- `src`: contém o código-fonte (módulos, componentes etc.)
- `angular.json`: contém configurações do projeto Angular (nome, scripts etc.)
- `package.json`: contém as especificações dos pacotes utilizados no projeto
- `tsconfig.json` e `tslint.json`: contêm configurações do processo de tradução de código TypeScript para JavaScript

No diretório `src` também há uma estrutura importante:

```
src
app
assets
environments
browserlist
favicon.ico
index.html
karma.conf.js
main.ts
polyfills.ts
styles.css
test.ts
tsconfig.app.json
tsconfig.spec.json
tslint.json
```

- `app`: contém o **root module** e os demais **feature modules** do projeto
- `assets`: contém arquivos CSS, JSON e scripts, por exemplo
- `environments`: contém arquivos de configuração do ambiente de desenvolvimento (`environment.ts`) e de produção (`environment.prod.ts`)
- `index.html`: contém o código HTML inicial para o projeto e inclui o componente principal do **root module**
- `main.ts`: contém o código TypeScript necessário para iniciar o software (processo chamado de **Bootstrap**)
- `polyfills.ts`: contém código TypeScript que indica scripts adicionais a serem carregados pelo Browser para funcionamento do software como um todo e para questões de compatibilidade com versões antigas de Browsers
- `style.css`: contém o código CSS para definir estilos globais para o software
- `tsconfig.app.json`: complementa configurações do arquivo `tsconfig.json` específicas para o software em questão

No diretório `app` há os arquivos:

- `app.component.css`, `app.component.html` e `app.component.ts`: definem o component `AppComponent`, respectivamente: apresentação por meio de CSS, **Template** e **Controller**. Basicamente, estes três arquivos formam a base de todo componente
- `app.module.ts`: define o **root module** (`AppModule`)

Esse capítulo apresentou conceitos importantes do Angular. Sempre que necessário, volte a esse capítulo para revisar conceitos do Angular. Muito provavelmente, mesmo desenvolvedores experientes precisem, de tempos em tempos, rever essas definições da arquitetura do Angular.

Capítulo 3

Angular CLI

O **Angular CLI** é uma ferramenta para inicializar, desenvolver, criar conteúdo e manter software desenvolvido em Angular (GOOGLE, [s.d.]). A principal utilização dessa ferramenta começa na criação de um projeto. Você pode fazer isso manualmente, claro, mas lidar com todas as configurações necessárias para o seu software Angular pode não ser algo fácil, mesmo se você for um desenvolvedor com nível de conhecimento médio a avançado.

---

cê-éle-í?

Como parte do vocabulário do Angular, geralmente é interessante pronunciar angular cê-ele-i, isso mesmo, CLI é uma sigla para *Command Line Interface* (no português Interface de Linha de Comando). Assim, usar "cli" não é exclusividade do Angular – provavelmente você verá isso em outros projetos.

As seções a seguir vão apresentar como instalar e utilizar o Angular CLI para desenvolver software Angular.

3.1 Instalando

O Angular CLI é distribuído como um pacote do **Node.JS**, então você não encontrará um instalador, como acontece com software tradicional. Ao invés disso, você precisa de um ambiente de desenvolvimento com o NodeJS instalado (veja Seção A).

3.2 Comandos

O Angular CLI disponibiliza uma série de comandos, que são fornecidos como parâmetros para o programa ng. Além disso, cada comando possui diversas opções (ou **flags**). Faça o exercício de se acostumar a essas opções para aumentar a sua produtividade.

Os principais comandos serão apresentados nas seções seguintes.

3.2.1 help

O comando `help` é muito importante porque apresenta uma documentação completa do Angular CLI, ou seja, todos os comandos e todas as suas opções.

Exemplo: documentação completa

```
$ ng help
```

Essa linha de comando apresenta todos os comandos e todas as suas opções.

Exemplo: documentação do comando `new`:

```
$ ng help new
```

Essa linha de comando apresenta apenas as opções do comando `new`.

As opções de cada comando são sempre precedidas de `--` (dois traços). Algumas opções aceitam um valor e geralmente possuem um valor padrão.

Exemplo: comando new com duas opções

```
$ ng new escola-app --skip-install --routing true --minimal true
```

Essa linha de comando fornece três opções para o comando `new`: `--skip-install`, `--routing` (que recebe o valor `true`) e `--minimal` (que recebe o valor `true`).

3.2.2 new

O comando `new` é utilizado para criar um projeto (um software Angular). Exemplo:

```
$ ng new escola-app
```

Quando a linha de comando do exemplo for executada será criado um software Angular chamado **escola-app** e estará em um diretório chamado `escola-app`, localizado a partir de onde a linha de comando estiver sendo executada.

A parte mais interessante de criar um projeto Angular com esse comando é que ele cria todos os arquivos necessários para um software bastante simples, mas funcional.

Para cada comando do Angular CLI é possível informar opções. As opções mais usadas do comando `new` são:

- `--skip-install`: faz com que as dependências não sejam instaladas

- `--routing`: se for seguida de `true`, o comando cria um módulo de rotas (isso será visto em outro capítulo)

3.2.3 generate

O comando `generate` é utilizado para criar elementos em um projeto existente. Para simplificar, é bom que o Angular CLI seja executado a partir do diretório do projeto a partir de agora. Por meio desse comando é possível criar:

- class
- component
- directive
- enum
- guard
- interface
- module
- pipe
- service

Como acontece com outros comandos, o `generate` também pode ser utilizado por meio do atalho `g`.

Exemplo: criar component ListaDeDisciplinas

```
$ ng generate component ListaDeDisciplinas
```

Essa linha de comando criará o diretório `./src/app/lista-de-disciplinas` e outros quatro arquivos nesse mesmo local:

- `lista-de-disciplinas.component.css`
- `lista-de-disciplinas.component.html`
- `lista-de-disciplinas.component.spec.ts`
- `lista-de-disciplinas.component.ts`

É importante notar que esses arquivos não estão vazios, mas já contêm código que mantém o software funcional e utilizável. Por isso o Angular CLI considera que as opções (class, component etc.) são como **blueprints** (ou plantas, em português). Outra coisa importante é que o Angular CLI também modificará o arquivo `./src/app/app.module.ts` se necessário (ou outro arquivo que represente um módulo que não seja o **root module** – mais sobre isso depois).

3.2.4 serve

O comando `serve` compila o projeto e inicia um servidor web local. Por padrão o servidor web executa no modo `--watch` , que reinicia o comando novamente, de forma incremental, sempre que houver uma alteração em algum arquivo do projeto, e `--live-reload`, que recarrega o projeto no browser (se houver uma janela aberta) sempre que houver uma mudança.

3.2.5 build

O comando `build` compila o projeto, mas não inicia um servidor web local. Ao invés disso, gera os arquivos resultantes da compilação em um diretório indicado.

3.3 Criando e executando um projeto Angular

Para iniciar, execute o comando `new` do **Angular CLI** para criar um projeto. Exemplo:

```
$ ng new angular-hello-world
```

Aguarde alguns instantes para a conclusão das instalações dos pacotes e configuração inicial do projeto. O comando vai criar a pasta `angular-hello-world`. Execute os comando seguintes de dentro da pasta do rpojeto.

O software **noticias-js** pode ser executado no browser diretamente ao abrir o arquivo `index.html`, disso você já sabe. O que muda em relação a como executar um software angular?

Por causa da estrutura e da arquitetura do Angular a execução não pode ser feita abrindo o arquivo `src/index.html` no browser, pois será necessário um **servidor web**. O **Angular CLI** já inclui um **servidor web local** ou **servidor web de desenvolvimento**, que pode ser iniciado com o comando:

```
$ npm start
```

Esse não é bem um comando do **Angular CLI**, mas um **script** declarado no arquivo `package.json`:

```
1  {
2    "name": "angular-hello-world",
3    "version": "0.0.0",
4    "scripts": {
5      "ng": "ng",
6      "start": "ng serve",
7      "build": "ng build",
8      "test": "ng test",
9      "lint": "ng lint",
10     "e2e": "ng e2e"
11   },
12 ...
13 }
```

O script **start** executa o comando `serve` do **Angular CLI**. Então, na prática, o script é um *atalho* e, por isso, há mais de uma maneira de iniciar o software.

Ao executar o script será apresentada uma saída que indica que você deve acessar a URL `http://localhost:4200` no browser. Faça isso para ver uma imagem semelhante à seguinte:

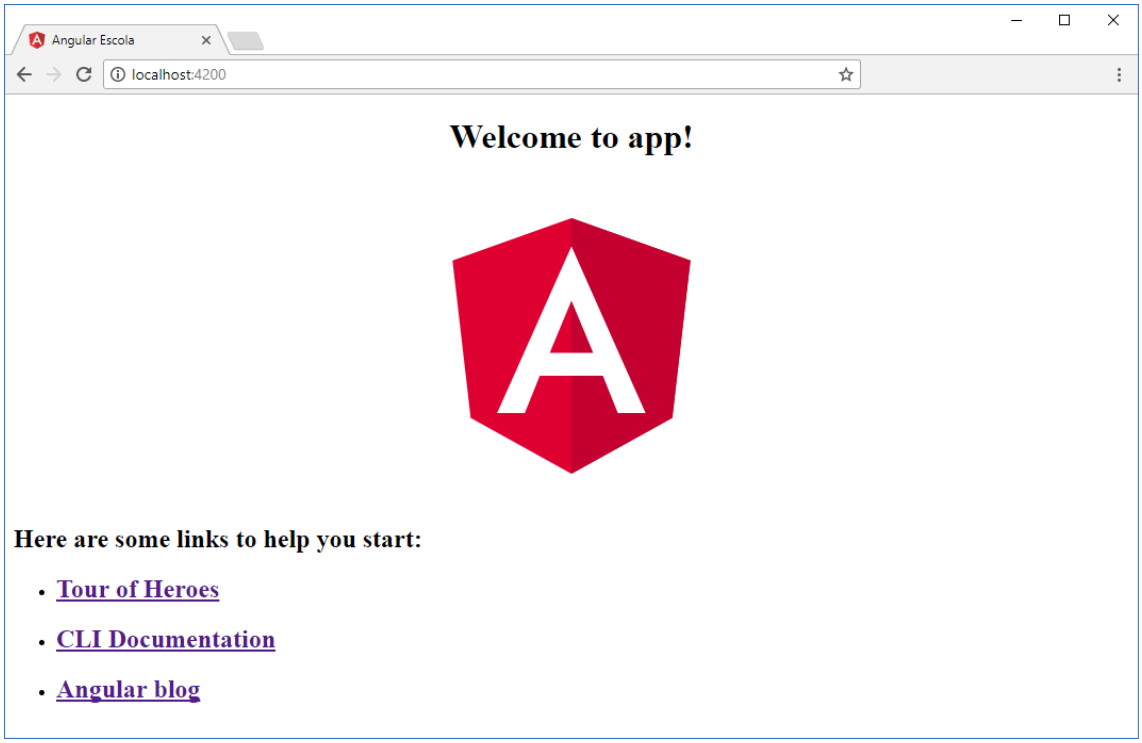


Figura 3.1: Versão inicial do software em execução no Browser

A URL `http://localhost:4200` identifica três elementos importantes:

- uso do protocolo HTTP
- o **host** é identificado por `localhost` (ou seja, a máquina local e, por isso o termo "servidor web local")
- a porta é `4200`

Veremos mais sobre esses e outros comando no decorrer do livro.

3.4 O arquivo package.json

O arquivo `package.json` contém informações do projeto, scripts e dependências. O arquivo criado pelo **Angular CLI** tem mais ou menos o seguinte conteúdo:

```
1  {
2    "name": "angular-hello-world",
3    ...
4    "scripts": {
5      ...
6    },
7    "private": true,
8    "dependencies": {
9      ...
10   },
```

```
11    "devDependencies": {
12      ...
13    }
14  }
```

Por ser um conteúdo em formato JSON, o interpretamos da seguinte forma:

- `name`: define o nome do projeto para o **Node.JS** (nenhuma relação com o Angular)
- `scripts`: contém scripts que podem ser executados no prompt (exemplos: `npm start` e `npm run build`)
- `dependencies`: define as dependências do projeto (pacotes, e suas versões, que são incluídos na compilação)
- `devDependencies`: define as dependências de desenvolvimeto (pacotes que não têm código-fonte incluído na compilação)

Espera! Você disse compilação?

Como já disse, você pode utilizar JavaScript ou TypeScript ao desenvolver projetos Angular. Isso não é bem assim. Na prática, geralmente você encontrará a recomendação (senão uma ordem) de utilizar TypeScript. O problema é que seu Browser não entende TypeScript, por isso é necessário um processo de tradução do código TypeScript para JavaScript. E não é só isso. Há vários recursos utilizados no projeto Angular criado com TypeScript que precisam de ajustes para que funcionem no seu Browser.

Por isso os comandos `serve` e `build` são tão importantes e – por que não dizer? – **browser-friendly** =)

3.5 Estrutura do projeto

Seção 2 apresentou a estrutura de um projeto Angular. De forma visual, a estrutura do projeto criado até o momento é ilustrada pela Figura 3.2.

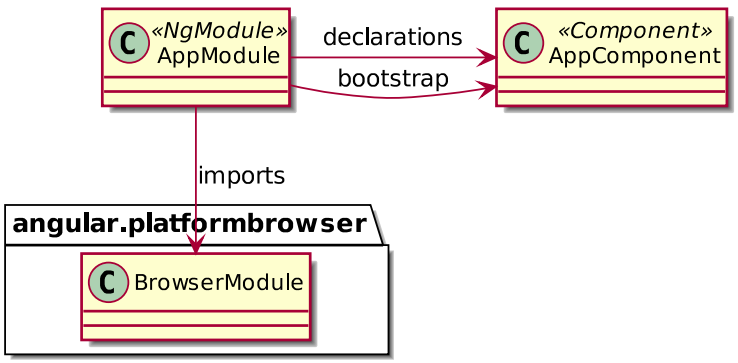


Figura 3.2: Estrutura básica de um projeto Angular

As relações entre as classes merecem destaque, começando por `AppModule`.

3.5.1 AppModule

A classe `AppModule` representa o **root module** do projeto e isso é especificado no arquivo `src/main.ts`:

```
1  ...
2  import { AppModule } from './app/app.module';
3  ...
4  platformBrowserDynamic().bootstrapModule(AppModule)
5    .catch(err => console.log(err));
```

A linha 4 contém uma chamada para o método `bootstrapModule()` com argumento `AppModule`, o que indica para o Angular que ele deve usar esse módulo (`AppModule`) na execução inicial (**Bootstrap**). A partir de então os recursos do módulo poderão ser utilizados.

Código-fonte 3.1 apresenta a classe `AppModule`.

Código-fonte 3.1: Código-fonte do AppModule no software noticias-angular

```
1  import { BrowserModule } from '@angular/platform-browser';
2  import { NgModule } from '@angular/core';
3
4  import { AppComponent } from './app.component';
5
6  @NgModule({
7    declarations: [
8      AppComponent
9    ],
10   imports: [
11     BrowserModule
12   ],
13   providers: [],
14   bootstrap: [AppComponent]
15 })
16 export class AppModule { }
```

A classe `AppModule` é declarada usando o decorator `@NgModule` e entre as linhas 6 e 16 há a utilização desse recurso para informar os metadados:

- `declarations`: a lista dos componentes declarados (ou pertencentes) a o módulo
- `imports`: a lista dos módulos importados
- `providers`: a lista de serviços declarados (vazia, por enquanto)
- `bootstrap`: define o **root component**

Em resumo, o `AppModule` é um módulo do Angular que declara o componente `AppComponent`, importa o módulo `BrowserModule` (do pacote `@angular/platform-browser`) e usa `AppComponent` como **root component**.

O **Angular** adota uma nomeclatura para o arquivo de módulo: `<modulo>.module.ts`. Assim, podemos usar o termo "módulo App" e saber do que se trata.

3.5.2 AppComponent

A classe `AppComponent` representa o **root component** do projeto e está declarada no arquivo `src/app/app.component.ts` cujo conteúdo é o seguinte:

```
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9      title = 'Angular';
10 }
```

A classe `AppComponent` declara o atributo `title` e o código também utiliza um **decorator**, dessa vez o `@Component` que também adiciona metadados à classe:

- `selector`: representa o seletor CSS para que o Angular identifique onde (em um Template) o componente deve ser apresentado
- `templateUrl`: o caminho do arquivo usado como Template (HTML)
- `styleUrls`: a lista de caminhos de arquivos usados como estilo (CSS)

Os metadados declaram, também, a estrutura básica de um Componente:

- **Controller**: a classe, em si
- **Template**: um arquivo HTML que contém marcação e representa a **View**

O **Angular** adota uma nomenclatura para facilitar a localização desses arquivos:

- `<componente>.component.ts` representa o Controller
- `<componente>.component.html` representa o Template
- `<componente>.component.css` representa o arquivo CSS (complemento para o Template)

Assim podemos usar o termo "componente App" e saber que se trata do conjunto de arquivos que declaram esse componente.

O Template desse componente (o arquivo `src/app.component.html`) tem conteúdo que parece HTML. Veja o trecho:

```
1  <h1>
```

```
2  Welcome to {{ title }}!
3  </h1>
```

Por que eu disse "parece HTML"? Porque isso realmente não é HTML, é a **linguagem de Template do Angular**, que herda características do HTML. Nesse trecho, inclusive, você pode percebe que é realmente semelhante, só diferenciando na linha 2, que traz um conteúdo entre `{{` e `}}`. Essa sintaxe entre chaves duplas é chamada **interpolação** e é utilizada para apresentar valores declarados no Controller por meio do recurso de **data binding**.

Você já viu que a classe `AppComponent` declara um atributo `title`, então o conteúdo do Template instrui o Angular a substituir o trecho `Welcome to {{ title }}!` por `Welcome to Angular!`. Nesse ponto vemos, na prática, mais um elemento da arquitetura MCV, o **Model**.

O Model não é um arquivo, é mais um conceito que permite o **data biding**, ou seja, passar dados entre Template e Controller. A Figura 3.3 ilustra esse conceito.

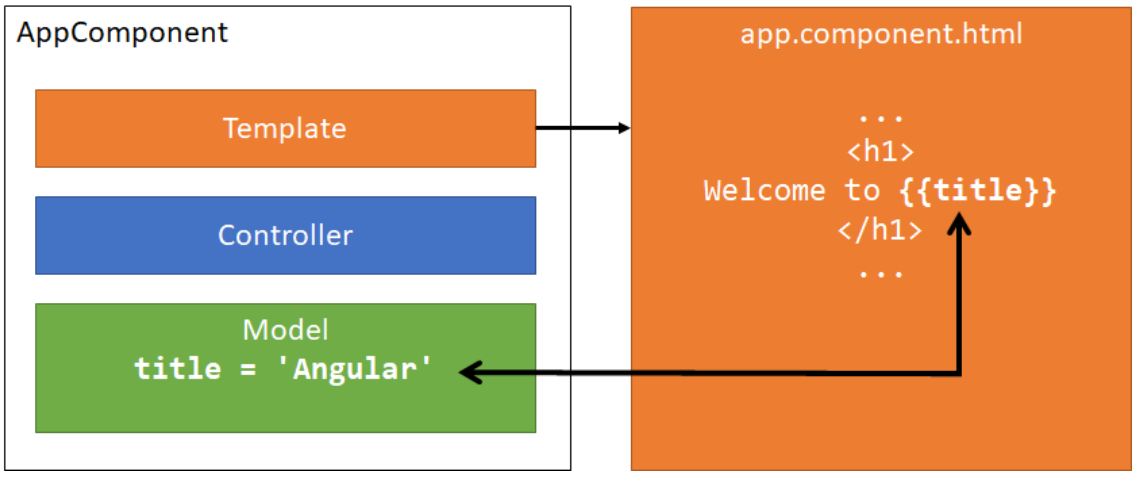


Figura 3.3: Relação entre Model-Template-Controller no AppComponent

Aqui aprendemos algo um conceito importante: o **Angular interpreta o Template**. Isso significa que todo o seu conteúdo, HTML e CSS, é interpretado pelo Angular antes de ser entregue para o browser. É isso que faz com que o recurso de interpolação funcione. É por isso, também, que Data binding é tão importante no Angular. Ele é responsável por fazer com que o atributo `title`, que compõe o Model, esteja disponível para ser usado no Template. Além disso, qualquer alteração no valor desse atributo representará uma alteração na visualização do componente no browser.

MVC ou MVVM?

Se você se lembrar da discussão sobre Angular ser MVC ou MVVM é aqui que o entendimento pesa a favor do segundo. O Controller não está desassociado do Model e, por causa do Data binding, sua função inclui informar o Template de que ocorreu uma alteração no Model, de forma que ele seja atualizado.

É isso, o Model é um conceito abstrato e está, geralmente, no Controller, representado por atributos

e métodos – sim, também é possível mostrar o valor retornado por um método no Template (mais sobre isso depois).

3.5.3 index.html

Para finalizar essa seção falta entender como o `AppComponent` é, enfim, apresentado. Para isso, veja o arquivo `src/index.html`:

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4  ...
5  </head>
6  <body>
7    <app-root></app-root>
8  </body>
9  </html>
```

Esse arquivo também é um Template, mas não está associado a um componente. O arquivo `angular.json` (por volta da linha 17) é que define que esse arquivo é chamado inicialmente durante a execução do software no servidor web local.

A parte HTML é fácil de entender, bem como saber que o elemento `app-root` não faz parte do HTML. Então, de onde vem esse elemento? Voltando um pouco na declaração do `AppComponent` lembramos que há um metadado chamado `selector`, que tem justamente o valor `'app-root'`. Por ser um seletor de elemento a instrução para o Angular, ao processar o `src/index.html` é: onde encontrar o elemento `app-root` apresente o componente `AppComponent`. Assim, as coisas ficam mais ou menos como ilustra a Figura 3.4.

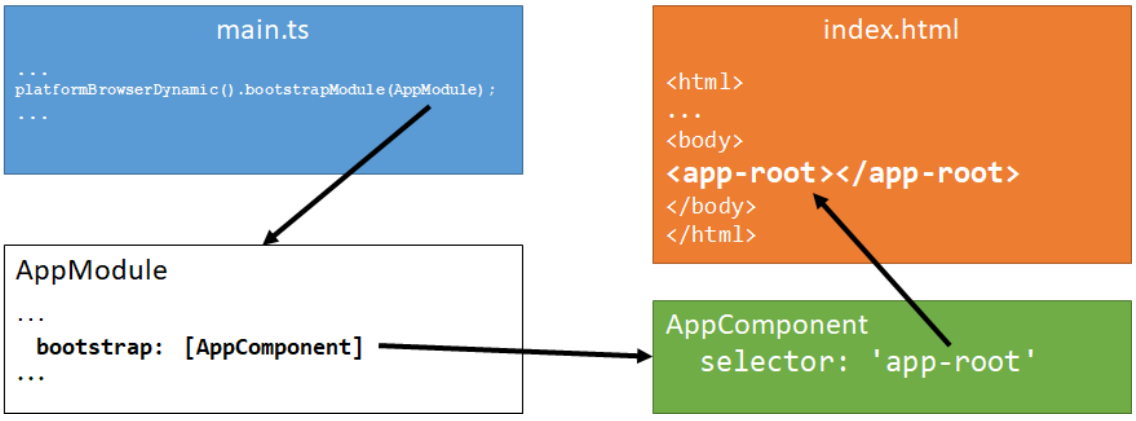


Figura 3.4: Execução do Template inicial

Viu? Não é mágica. É software!

3.6 O que vem a seguir?

Como agora você já deve estar entendendo melhor o funcionamento do Angular e o código-fonte do projeto, é hora de sujar as mãos =) No capítulo seguinte você vai implementar o software **noticias-angular**.

Capítulo 4

Gerenciador de notícias com Angular

Seção 1 apresentou conceitos básicos de desenvolvimento web e o software **noticias-js**, um projeto de gerenciador de notícias que é implementado utilizando tecnologias front-end (HTML, CSS e JavaScript). Este capítulo apresenta o **noticias-angular**, basicamente uma outra versão do **noticias-js**, mas que é implementado utilizando **Angular**.

4.1 Clonar e executar localmente

Este capítulo está diretamente relacionado com o software **noticias-angular**, cujo repositório é https://github.com/jacksongomesbr/webdevbook-noticias-angular. Antes de prosseguir, clone e execute o projeto localmente. A interface do software é idêntica à do **noticias-js**. Verifique.

4.2 O projeto

O **noticias-angular** foi criado com o comando `new` do **Angular CLI**:

```
$ ng new webdevbook-noticias-angular
```

A estrutura do projeto muda em relação ao projeto básico criado pelo **Angular CLI** (Seção 3.3) em alguns aspectos. Assim, o software será apresentado a partir da Figura 4.1, que apresenta a sua estrutura.

As relações entre os componentes são as mesmas apresentadas na Seção 3.5, com o acréscimo da importação do módulo `FormsModule`, que é necessário para que o projeto possa utilizar os recursos do Angular que permitem o usuário entrar dados por meio de formulários.

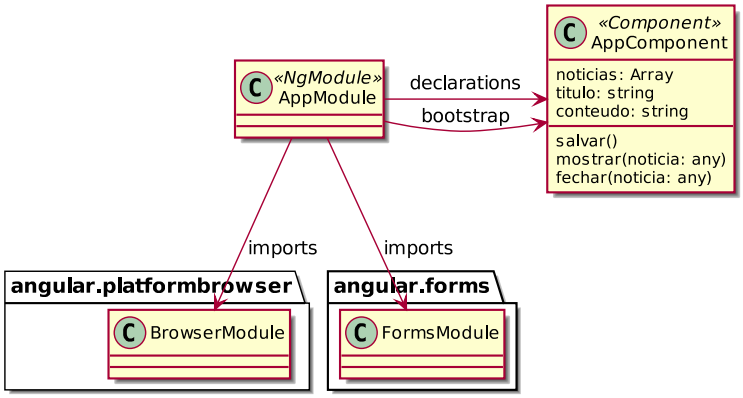


Figura 4.1: Estrutura do projeto angular-noticias

4.2.1 AppComponent

As maiores modificações estão no `AppComponent` e detalhes são apresentados nas seções seguintes.

4.2.1.1 Controller

Um trecho do Controller do `AppComponent` é apresentado a seguir:

```
1  ...
2  @Component({
3      ...
4  })
5  export class AppComponent {
6    noticias = [];
7    titulo = null;
8    conteudo = null;
9
10   salvar() {
11     ...
12   }
13
14   mostrar(noticia) {
15     ...
16   }
17
18   fechar(noticia) {
19     ...
20   }
21 }
```

A classe declara três atributos:

- `noticias`: um `Array` que contém a lista de notícias
- `titulo`: representa o título da notícia cadastrada (vinculada ao formulário)

- `conteudo`: representa o conteúdo da notícia cadastrada (vinculada ao formulário)

Há também três métodos:

- `salvar()`: obtém os dados do formulário (título e conteúdo da notícia) e insere na lista de notícias
- `mostrar()`: apresenta o conteúdo de uma notícia (informada como parâmetro `noticia`)
- `fechar()`: oculta o conteúdo de uma notícia (informada como parâmetro `noticia`)

Em detalhes, o método `salvar()`:

```
1  salvar() {
2      const noticia = {
3          id: this.noticias.length,
4          titulo: this.titulo,
5          conteudo: this.conteudo,
6          visivel: false
7      };
8      this.noticias.push(noticia);
9      this.titulo = null;
10     this.conteudo = null;
11 }
```

O método `salvar()` contém código que realiza o seguinte:

- cria um objeto (`noticia`) com quatro atributos: `id`, `titulo`, `conteudo` e `visivel`. Os valores dos atributos `titulo` e `conteudo` são obtidos dos respectivos atributos (note o uso de `this`) e o valor do atributo `visivel` é `false`
- adiciona o objeto `noticia` na lista `noticias` por meio do método `push()`
- redefine valores dos atributos `titulo` e `conteudo` (atribuindo o valor `null`)

Os métodos `mostrar()` e `fechar()` são bem semelhantes:

```
1  mostrar(noticia) {
2      noticia.visivel = true;
3  }
4
5  fechar(noticia) {
6      noticia.visivel = false;
7  }
```

Os métodos recebem o parâmetro `noticia`, um objeto que representa uma notícia, e alteram o valor do atributo `visivel`:

- `true`: mostrar o conteúdo
- `false`: ocultar o conteúdo

4.2.1.2 Template

O Template do `AppComponent` contém três partes:

- cabeçalho
- área da lista de notícias
- área do cadastro de notícias

O seguinte trecho do Template mostra onde estão essas partes:

```
1 <h1>Gerenciador de notícias</h1>
2 <h2>Notícias recentes</h2>
3 ...
4 <h2>Cadastrar notícia</h2>
5 ...
```

Primeiro, a área da lista de notícias:

```
1  <h2>Notícias recentes</h2>
2  <p>Clique no título da notícia para expandir</p>
3  <div>
4    <ul>
5      <li *ngFor="let noticia of noticias">
6        <p (click)="mostrar(noticia)">{{ noticia.titulo }}</p>
7        <p *ngIf="noticia.visivel">
8          {{ noticia.conteudo }}
9          <br> -------------
10         <br>
11         <button (click)="fechar(noticia)">Fechar</button>
12       </p>
13     </li>
14   </ul>
15 </div>
```

O elemento `li` da linha 5 tem o atributo `*ngFor`, que representa a **diretiva estrutural** `NgForOf`, uma **diretiva de repetição** que insere elementos no DOM conforme uma expressão de iteração com a sintaxe: `let variavel of lista`. Nesse caso `variavel` é chamada de **variável de template** porque só existe no contexto do Template. A diretiva `NgForOf` instrui o Angular a repetir uma parte do template para da item de uma iteração. O valor `let noticia of noticias` é interpretado assim:

> Para cada notícia na lista `noticias`, crie uma variável de template chamada `noticia` e torne-a disponível para cada iteração (repetição) do template contido em `li`.

Assim, o conteúdo do `li` é repetido para cada item da lista `noticias` (atributo do Controller). O `li` contém dois elementos `p` que são usados para propósitos diferentes. O primeiro `p`, com atributo `(click)`, apresenta o título da notícia usando **interpolação**. Perceba que o Angular interpreta a variável de template `noticia` seguindo o determinado no Controller (notícia tem o atributo `titulo`).

Outra parte importante tem relação com o atributo `(click)`, que usa a **sintaxe de evento** (Seção 2.1.5). Nesse caso, o evento é `click` e o conteúdo do atributo, `mostrar(noticia)`, instrui o Angular a chamar o método `mostrar()` (definido no Controller) passando como argumento a variável de template `noticia`, para que seu conteúdo seja apresentado. Perceba a interação com os métodos declarados no Controller e a utilização do **paradigma da Orientação a Objetos**.

O outro elemento `p` tem o atributo `*ngIf`, que representa a diretiva estrutural `NgIf`, uma **diretiva condicional**, que insere e remove elementos no DOM conforme uma expressão booleana. O valor `noticia.visible` é interpretado assim:

> Se o valor do atributo `visivel` da variável de template `noticia` for `true`, então insere o elemento `p` no DOM, caso contrário, remove o elemento `p`.

Assim, se o atributo `visivel` da variável de template `noticia` for `true`, seu conteúdo será visível.

Para finalizar, há um elemento `button` que também tem um atributo `(click)` com valor `fechar(noticia)`, o que instrui o Angular a chamar o método `fechar()` passando como argumento a variável de template `noticia`, para que seu conteúdo seja ocultado.

Diretivas manipulam DOM?

O projeto **noticias-js** usa DOM (diretamente ou por meio do jQuery) para manipulação do DOM. De forma simplificada o Angular faz a mesma coisa usando diretivas, como `NgForOf` e `NgIf`.

A outra parte importante do Template tem a ver com o formulário de cadastro:

```
1  <h2>Cadastrar notícia</h2>
2  <form #cadastro="ngForm" (submit)="salvar()">
3    <div>
4      <label for="titulo">Título</label>
5      <input type="text" id="titulo" name="titulo" [(ngModel)]="titulo"
           required>
6    </div>
7    <div>
8      <label for="conteudo">Conteúdo</label>
9      <textarea id="conteudo" name="conteudo" cols="80" rows="5" [(ngModel)
           ]="conteudo" required></textarea>
10   </div>
11   <div>
12     <button type="submit" [disabled]="!cadastro.form.valid">Salvar</button
           >
13     <button type="reset" formnovalidate>Limpar</button>
14   </div>
15 </form>
```

Primeiro, ao utilizar o elemento `form` o Angular cria automaticamente uma diretiva `NgForm`, que

determina o funcionamento do formulário a partir de então (campos, validação etc.). Se você precisar acessar informações do formulário poderá usar uma variável de template. Por isso o `form` tem o atributo `#cadastro` com valor `ngForm`.

Além disso o `form` também usa a sintaxe de evento para chamar o método `salvar()` no `submit` (também pode ser usado o nome `ngSubmit`). O comportamento tradicional de um formulário HTML é que seja submetido quando um `button` for clicado. No Angular, por causa da diretiva `NgForm`, o processo é modificado para permitir a interação com o Controller.

No final do `form` há o `button` "Salvar", que tem o atributo `[disabled]`, uma **diretiva de atributo** usada para desabilitá-lo com base em uma expressão booleana. Assim, a expressão `!cadastro.form.valid` acessa a informação de validação global do formulário e, se for `true` (formulário não está válido), desabilita o botão (não permite que ele seja clicado). A validação continua utilizando os recursos do HTML (atributo `required`, por exemplo, em cada campo de preenchimento obrigatório).

Os campos do formulário continuam utilizando elementos do HTML (`input` e `textarea`), mas agora também utilizam **two-way data binding**:

```
1 ...
2 <input type="text" id="titulo" name="titulo" [(ngModel)]="titulo" required
      >
3 ...
4 <textarea id="conteudo" name="conteudo" cols="80" rows="5" [(ngModel)]="
      conteudo" required></textarea>
5 ...
```

A sintaxe de two-way data binding é utilizada para vincular um campo de formulário a uma expressão (geralmente um atributo declarado no Controller). Isso acontece no `input` por meio de `[(ngModel)]="titulo"`, vinculando o seu valor ao atributo `titulo`, e no `textarea`, por meio de `[(ngModel)]="conteudo"`, vinculando seu valor ao atributo `conteudo`. Isso significa que qualquer alteração em um dos campos gera uma alteração no atributo vinculado, e vice-versa.

Este capítluo apresentou o software **noticias-angular** e sua estrutura. Os destaques são:

- importar o módulo `FormsModule` no **root module**
- uso das diretivas estruturais `NgForOf` e `NgIf`
- uso de data-binding (interpolação, evento, propriedade e vinculação de campo do formulário com atributo do Controller)

Apêndice A

Configuração do ambiente de desenvolvimento

A.1 Node.js

O **Node.js** é um ambiente de execução do JavaScript independente do browser e multiplataforma (NODE.JS FOUNDATION, [s.d.]). Nos projetos desse livro é necessário utilizar **Node.js** e também a ferramenta **npm**, um gerenciador de pacotes JavaScript para o **Node.js** (NPM, INC., [s.d.]).

A instalação do **Node.js** é simples, bastando acessar https://nodejs.org/en/download/ para obter os binários de instalação conforme a plataforma desejada. O **npm** também é fornecido junto com a instalação do **Node.js**.

Para verificar se seu ambiente de execução do **Node.js** está operando normalmente, execute os comandos a seguir em um prompt:

```
$ node -v
$ npm -v
```

A saída dos programas apresenta, respectivamente, as versões do **Node.js** e do **npm** instaladas, como:

```
v10.5.0
6.2.0
```

A.2 Angular CLI

O **Angular CLI** é fornecido como um pacote **npm**, então deve ser instalado da seguinte forma:

```
$ npm install -g @angular/cli
```

O comando `install` seguido da opção `-g` faz uma *instalação global* do **Angular CLI**, o que significa que ele estará disponível para qualquer usuário.

Referências

GIT COMMUNITY. **Git**, [s.d.]. Disponível em: <https://git-scm.com/>. Acesso em: 22 jul. 2018

GOOGLE. **Angular**, [s.d.]. Disponível em: <https://angular.io/>. Acesso em: 22 jul. 2018a

GOOGLE. **Angular CLI**, [s.d.]. Disponível em: <https://cli.angular.io/>. Acesso em: 22 jul. 2018b

MICROSOFT. **TypeScript - JavaScript that scales**, [s.d.]. Disponível em: <https://www.typescriptlang.org/index.html>. Acesso em: 24 jul. 2018a

MICROSOFT. **Visual Studio Code - Code Editing. Redefined**, [s.d.]. Disponível em: <https://code.visualstudio.com/>. Acesso em: 22 jul. 2018b

NODE.JS FOUNDATION. **Node.js**, [s.d.]. Disponível em: <https://nodejs.org>. Acesso em: 23 jul. 2018

NPM, INC. **npm**, [s.d.]. Disponível em: <https://www.npmjs.com/>. Acesso em: 23 jul. 2018

THE JQUERY FOUNDATION. **jQuery**, [s.d.]. Disponível em: <http://jquery.com/>. Acesso em: 22 jul. 2018

W3SCHOOLS. **JavaScript Tutorial**, [s.d.]. Disponível em: <https://www.w3schools.com/js/default.asp>. Acesso em: 22 jul. 2018a

W3SCHOOLS. **JavaScript and HTML DOM Reference**, [s.d.]. Disponível em: <https://www.w3schools.com/jsref/default.asp>. Acesso em: 22 jul. 2018b

W3SCHOOLS. **HTML5 Tutorial**, [s.d.]. Disponível em: <https://www.w3schools.com/html/default.asp>. Acesso em: 22 jul. 2018c

W3SCHOOLS. **CSS Tutorial**, [s.d.]. Disponível em: <https://www.w3schools.com/css/default.asp>. Acesso em: 22 jul. 2018d

W3SCHOOLS. **CSS Reference**, [s.d.]. Disponível em: <https://www.w3schools.com/cssref/default.asp>. Acesso em: 22 jul. 2018e

W3SCHOOLS. **HTML Element Reference**, [s.d.]. Disponível em: <https://www.w3schools.com/tags/default.asp>. Acesso em: 22 jul. 2018f

W3SCHOOLS. **jQuery Tutorial**, [s.d.]. Disponível em: <https://www.w3schools.com/jquery/

default.asp>. Acesso em: 22 jul. 2018g

W3SCHOOLS. **jQuery Reference**, [s.d.]. Disponível em: <https://www.w3schools.com/jquery/jquery_ref_overview.asp>. Acesso em: 22 jul. 2018h

WIKIPEDIA CONTRIBUTORS. **Model–view–controller — Wikipedia, The Free Encyclopedia**, 2018a. Disponível em: <https://en.wikipedia.org/w/index.php?title=Model%E2%80%93view%E2%80%93controller&oldid=849850595>. Acesso em: 23 jul. 2018

WIKIPEDIA CONTRIBUTORS. **Model–view–viewmodel — Wikipedia, The Free Encyclopedia**, 2018b. Disponível em: <https://en.wikipedia.org/w/index.php?title=Model%E2%80%93view%E2%80%93viewmodel&oldid=851186618>. Acesso em: 23 jul. 2018

WIKIPEDIA CONTRIBUTORS. **Dependency injection — Wikipedia, The Free Encyclopedia**, 2018c. Disponível em: <https://en.wikipedia.org/w/index.php?title=Dependency_injection&oldid=845653474>