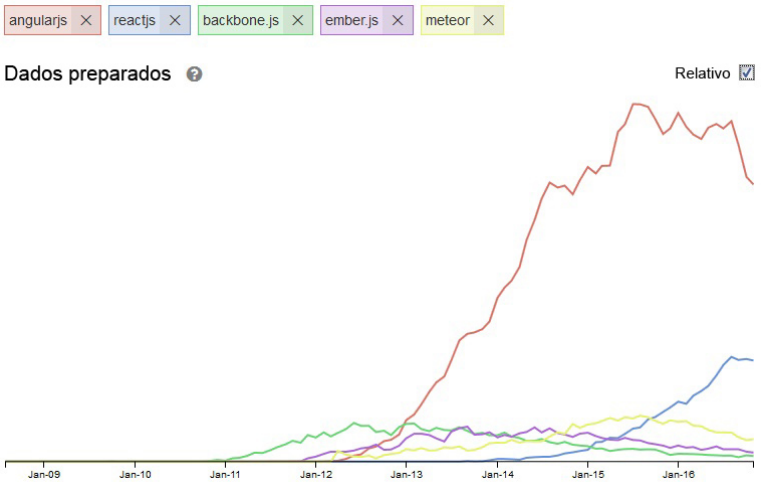
Implemente um aplicativo de página única com o Angular 2

Babu Suresh

14/Dez/2016

Use Angular 2 e TypeScript para implementar um aplicativo de página única. Consuma microsserviços, melhore o desempenho dos aplicativos, escale automaticamente seu aplicativo, reduza a tensão do servidor e aumente a capacidade de uso do aplicativo.

A tecnologia de aplicativo de página única (SPA) está gerando um alto interesse no segmento de mercado de software devido ao seu potencial para um melhor desempenho do navegador e de aplicativos de smartphone. Em mais da metade da última década, o interesse dos desenvolvedores em Angular— uma estrutura SPA de software livre — ultrapassou o interesse em outras estruturas da web (React, Backbone, Meteor, Ember), a julgar pelo número de perguntas sobre StackOverflow:



(Fonte da imagem: tag-trends)

Os desenvolvedores da web e de dispositivos móveis aguardaram ansiosos para dar as boas-vindas ao Angular 2 (liberado em setembro de 2016). O Angular 2 não é um upgrade do Angular 1, mas sim uma estrutura completamente nova, diferente e mais avançada. A proficiência em Angular 2 já é um atributo bastante procurado para o desenvolvimento de aplicativos de várias plataformas de alto desempenho, escaláveis, robustos e modernos.

TypeScript

O Angular 2 suporta JavaScript, Dart e TypeScript. Você usará TypeScript como a linguagem de desenvolvimento para o projeto deste tutorial. A estrutura é criada em TypeScript e a maioria das documentações, manuais e tutoriais sobre o Angular foca no TypeScript como a linguagem de desenvolvimento.

Este tutorial apresenta os principais componentes do Angular 2 e demonstra a codificação e a execução de um SPA utilizando Angular 2 em seu computador de desenvolvimento e em um servidor sandbox. Para tirar o máximo proveito do tutorial, você precisa ter certa experiência em programação de web, inclusive conhecimento geral em JavaScript, TypeScript e HTML. Não é necessário ter experiência em Angular. Depois de trabalhar no projeto de amostra, você estará preparado para criar seus próprios SPAs com Angular 2.

É possível fazer download do código de amostra completo para o aplicativo sandbox na seção "Recursos para Download" no final do tutorial.

Por que SPA e Angular 2?

Um SPA renderiza apenas uma página HTML do servidor quando o usuário inicia o aplicativo. Junto com essa página HTML, o servidor envia um mecanismo de aplicativo para o cliente. O mecanismo controla o aplicativo inteiro, inclusive processamento, entrada, saída, pintura e carregamento de páginas HTML. Geralmente, de 90 a 95 por cento do código do aplicativo é executado no navegador; o restante opera no servidor quando o usuário precisa de novos dados ou tem que executar operações seguras, como autenticação. Como a dependência do servidor é praticamente removida, um SPA é escalado automaticamente no ambiente Angular 2: Independentemente de quantos usuários acessam o servidor simultaneamente, entre 90 e 95 por cento do tempo o desempenho do aplicativo nunca é impactado.

Além disso, como a maioria do carregamento está em execução no cliente, o servidor fica inativo na maior parte do tempo. A baixa de demanda de recursos do servidor reduz significativamente sua tensão, diminuindo potencialmente os custos do servidor.

Outra vantagem do Angular 2 é que ele ajuda SPAs a consumirem microsserviços de forma efetiva.

> *“ Geralmente, de 90 a 95 por cento do código do SPA é executado no navegador; o restante opera no servidor quando o usuário precisa de novos dados ou tem que executar operações seguras, como autenticação. ”*

Conceitos do Angular 2

Os principais conceitos do Angular 2 incluem:

- Módulos
- Componentes
- Services
- Rotas

De agora em diante, vamos nos referir ao Angular 2 simplesmente como Angular.

Módulos

Aplicativos Angular são modulares. Cada aplicativo Angular tem pelo menos um *módulo*— o módulo raiz, nomeado, por convenção, como `AppModule`. O módulo raiz pode ser o único módulo em um pequeno aplicativo, mas a maioria dos aplicativos têm vários módulos. Como desenvolvedor, fica a seu critério decidir como usar o conceito de módulos. Geralmente, você mapeia a funcionalidade principal ou um recurso para um módulo. Digamos que você tenha cinco áreas principais em seu sistema. Cada uma terá seu próprio módulo, além do módulo raiz, totalizando seis módulos.

Componentes

Um *componente* controla um patch da página, chamado *visualização*. Você define a lógica do aplicativo do componente — o que ele faz para suportar a visualização dentro de uma classe. A classe interage com a visualização por meio de uma API de propriedades e métodos. Um componente contém uma classe, um modelo e metadados. Um modelo é a forma de o HTML dizer ao Angular como renderizar o componente. Um componente pode pertencer a um, apenas um, módulo.

Services

Um *serviço* fornece qualquer valor, função ou recurso que seu aplicativo precisa. Geralmente, um serviço é uma classe com um propósito limitado bem definido; ele deve fazer algo específico e fazer muito bem. Componentes são grandes consumidores de serviços. Serviços são grandes consumidores de microsserviços.

Rotas

Rotas permitem a navegação de uma visualização para outra, conforme os usuários executam tarefas do aplicativo. Uma rota é equivalente a um mecanismo usado para controlar menus e submenus.

Agora que você entendeu os benefícios dos SPAs e tem uma noção dos conceitos do Angular, é hora de se preparar para trabalhar no projeto de amostra.

O que você precisará?

Para concluir o projeto de amostra, você precisará do Node.js e da CLI do Angular (uma interface de linha de comandos para o Angular) instalados no PC de desenvolvimento:

- Para instalar o Node.js:
  1. Faça o download da versão para seu sistema e escolha as opções padrão para concluir a instalação.
  2. Execute a tarefa `node -v` a partir da linha de comandos do S.O. para verificar o número da versão — no meu caso, `v6.9.1`.

O Node Package Manager (NPM) é instalado automaticamente junto com o Node. Digite `npm -v` para ver o número da versão. O NPM é usado no background quando você instala pacotes do repositório NPM público. Um comando comum do NPM é `npm install`, que é usado para fazer download das versões de pacote listadas no arquivo package.json do projeto Angular.

- Para instalar a CLI do Angular:
  1. Execute `npm install -g angular-cli@1.0.0-beta.21` para instalar a versão (ainda em beta no momento da edição) que eu usei para o aplicativo de amostra. (Se quiser experimentar um build diferente, visite o site da CLI.) A instalação leva cerca de 10 minutos para ser concluída.
  2. Após uma instalação bem-sucedida, digite `ng -v` na linha de comando do S.O. para ver o número da versão da CLI — no meu caso:

     ```
     angular-cli: 1.0.0-beta.21
     node: 6.9.1
     os: win32 x64
     ```

     Se você precisar de ajuda com a sintaxe `ng`, digite `ng help` na linha de comandos do S.O.

O arquivo package.json

O arquivo package.json — um arquivo de metadados chave em um aplicativo Angular — contém os detalhes do aplicativo e seus pacotes dependentes. Esse arquivo é o mais importante em um aplicativo Angular, principalmente quando você move seu código de um computador para outro durante upgrades. O arquivo package.json controla a versão dos pacotes que precisam ser instalados.

Não é preciso se preocupar com o package.json para o exercício deste tutorial. Porém, ao desenvolver um aplicativo em nível de produção, você deverá prestar mais atenção nos números de versão listados nesse arquivo.

Aqui estão algumas instruções válidas de um arquivo package.json:

```
{ "dependencies" :
  { "foo" : "1.0.0 - 2.9999.9999"
  , "bar" : ">=1.0.2 <2.1.2"
  , "baz" : ">1.0.2 <=2.3.4"
  , "boo" : "2.0.1"
  , "qux" : "<1.0.0 || >=2.3.1 <2.4.5 || >=2.5.2 <3.0.0"
  , "til" : "^1.2"
  , "elf" : "~1.2.3"
  , "two" : "2.x"
  , "thr" : "3.3.x"
  , "lat" : "latest"
  }
}
```

O comando `npm install` interpreta a parte da instrução à direita dos dois pontos, como a seguir (em que *version* indica o número da versão usada):

- "*version*": deve corresponder à *version* exatamente

- ">*version*": deve ser maior que *version*
- "~*version*": aproximadamente equivalente a *version*
- "^*version*": compatível com *version*
- "1.2.x": 1.2.0, 1.2.1 e assim por diante, mas não 1.3.0
- "*" : corresponde a qualquer *version*
- "" (sequência vazia)": mesmo que `*`
- "*version1* - *version2*": mesmo que ">=*version1* <=*version2*"

Para saber mais sobre o package.json, digite `npm help package.json` na linha de comandos do S.O. para ver o conteúdo da ajuda em seu navegador.

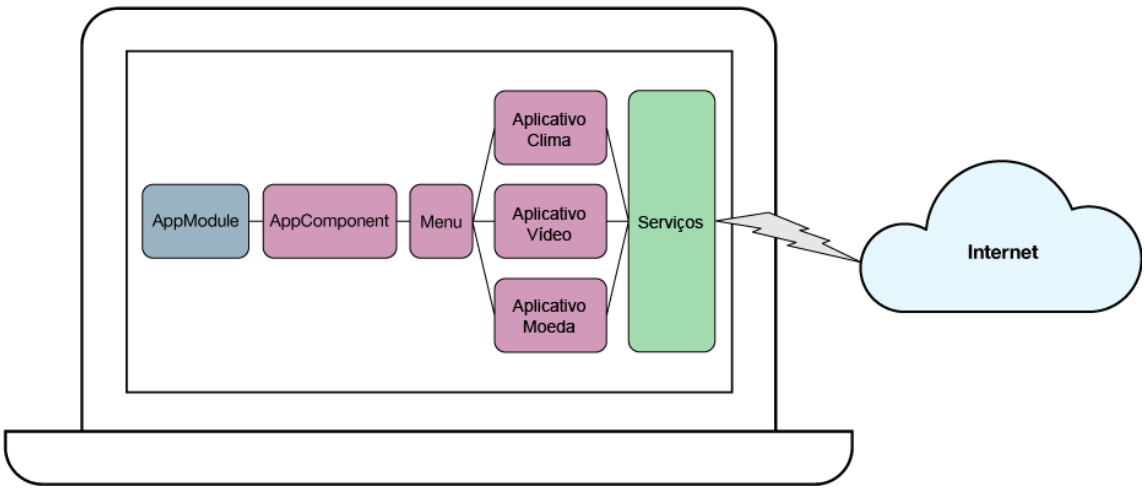
Visão geral do projeto de amostra

O projeto de amostra consiste em um aplicativo Angular simples de instalar e em um aplicativo customizado que você irá desenvolver em cima desse mesmo aplicativo. Ao concluir, você terá um aplicativo Angular consistindo em três miniaplicativos, com recursos que usam três APIs de serviço da web:

- Weather from Yahoo!
- Currency exchange
- Movie details

Toda a lógica do aplicativo será executada em seu navegador. O servidor será necessário apenas quando o navegador precisar de novos dados. Na verdade, é possível encerrar o processo do servidor e continuar trabalhando no aplicativo, pois ele é um SPA.

Este diagrama mostra a topologia do aplicativo:



Você usará a CLI do Angular para criar seu projeto (que, por padrão, incluirá `AppModule` e `AppComponent`) e quatro componentes customizados:

- Componente Menu
- Componente Weather
- Componente Movie

- Componente Currency

Você irá criar rotas para a navegação no menu e incluirá os seguintes serviços nos componentes weather, movie e currency:

- Dados provenientes do HTTP que consome os microsserviços
- Compartilhamento de recursos entre os componentes enquanto os serviços são usados

Criando o aplicativo e o módulo base

Pronto para começar? Inicie sua linha de comandos do S.O. em um local no qual deseja colocar o diretório do projeto.

Criando um projeto Angular

Gere um novo projeto Angular executando o seguinte comando (em que `dw_ng2_app` é o nome do projeto):

```
ng new dw_ng2_app --skip-git
```

Após todos os pacotes necessários e o aplicativo base Angular serem instalados (o que leva cerca de 10 minutos), você retornará ao prompt de comandos do S.O. Se depois você listar o diretório /dw_ng2_app, é possível a estrutura do projeto:

```
|— e2e
|— node_modules
|— src
angular-cli.json
karma.conf.js
package.json
protractor.conf.js
README.md
tslint.json
```

O conteúdo do diretório ../dw_ng2_app/src é:

```
|— app
|— assets
|— environments
favicon.ico
index.html
main.ts
polyfills.ts
styles.css
test.ts
tsconfig.json
typings.d.ts
```

E o diretório ../dw_ng2_app/src/app (a *pasta do módulo raiz*) contém os seguintes arquivos:

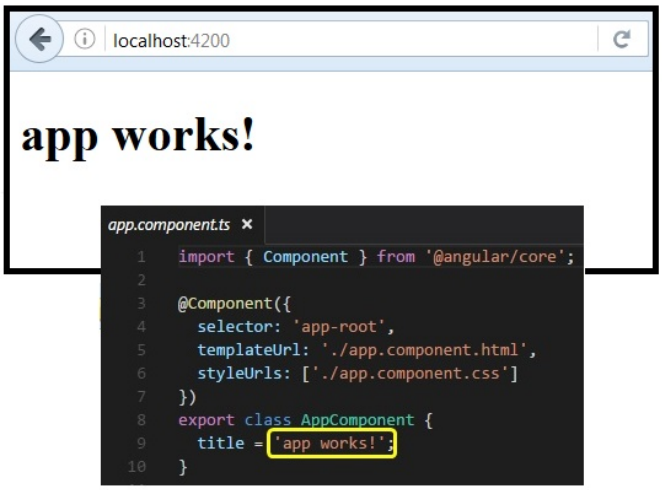
```
app.component.css
app.component.html
app.component.spec.ts
app.component.ts
app.module.ts
index.ts
```

Executando o aplicativo Angular simples de instalar

Mude para o diretório do projeto e execute `ng serve` para iniciar o aplicativo Angular simples de instalar.

Por padrão, o processo começa na porta número 4200. Se o valor da variável de ambiente do sistema `port` for diferente de 4200, o processo começará nesse número de porta. Você tem a opção de substituir o número de porta padrão executando o comando `ng serve --port 4200`.

Abra o navegador e insira a URL `http://localhost:4200/`. O aplicativo Angular exibe **app works!** para indicar que o aplicativo está ativo e pronto:



Se você fizer mudanças no código enquanto o aplicativo estiver em execução, o Angular terá a destreza de monitorar e reiniciar o aplicativo automaticamente. Tente editar o arquivo app.component.ts mudando o valor de `title`. É possível ver que a página do navegador reflete a mudança:

Como um módulo é vinculado a um componente

Na Listagem 1, a linha 20 mostra a declaração do módulo `AppModule`.

Lista 1. app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';

import { AppComponent } from './app.component';

@NgModule({
```

```
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Esse módulo contém apenas um componente —`AppComponent`— conforme mostrado na linha 10. A linha 18 diz que o primeiro componente iniciado sob o processo de autoinicialização é o `AppComponent`.

Dentro de um componente

A Listagem 2 mostra com o que um componente de aplicativo principal se parece no arquivo app.component.ts.

Lista 2. app.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'My Angular 2 app works!';
}
```

E aqui vemos com o que um componente de aplicativo principal se parece no arquivo app.component.html:

```
<h1>
  {{title}}
</h1>
```

Local dos metadados

Os metadados dizem ao Angular como processar uma classe. Na verdade, o `AppComponent` não é um componente até você dizer ao Angular, anexando metadados à classe.

Você anexa os metadados usando o decorador `@Component`, que identifica a classe como um componente. O decorador `@Component` usa um objeto de configuração necessário com as informações que o Angular precisa para criar e apresentar o componente e sua visualização.

O código na Listagem 2 usa três opções entre as opções de configuração disponíveis do `@Component`:

- `selector`: o seletor CSS que diz ao Angular para criar e inserir uma instância desse componente onde ele encontrar a tag `selector` no arquivo HTML pai

- `templateUrl`: o arquivo HTML do componente
- `styleUrls`: as planilhas de estilo do componente, como arquivos .css

Local do modelo dentro do componente

Um modelo é a forma de o HTML dizer ao Angular como renderizar o componente. Na linha 5 da Listagem 2, `templateUrl` aponta para uma visualização denominada `app.component.html`.

Ligação de dados dentro do componente

A ligação de dados é uma forma de o HTML dizer ao Angular como renderizar o componente. No arquivo app.component.ts, o valor de `title` é configurado dentro da classe e usado no arquivo app.component.html. A ligação de dados pode ser unidirecional ou bidirecional. Nesse caso, o mapeamento será unidirecional se você mencionar a variável dentro das chaves duplas `{{ }}`. O valor é transmitido da classe para o arquivo HTML.

Criando rotas e componentes customizados

Neste momento, seu aplicativo Angular está pronto e funcionando. Esse aplicativo base tem um módulo, um componente, uma classe, um modelo, metadados e uma ligação de dados. — mas ainda faltam outras quatro partes importantes:

- Componentes diversos
- Roteiros
- Serviços
- Consumo de microsserviços

Em seguida, você irá criar os componentes customizados.

Criando componentes customizados

Pare o processo Angular pressionando Ctrl-C (certifique-se de estar no diretório do projeto Angular, dw_ng2_app nesse caso). No prompt de comandos, execute os seguintes comandos:

- `ng g c Menu -is --spec false --flat`: cria o componente `Menu` dentro do módulo raiz `AppModule` na mesma pasta.
- `ng g c Weather -is --spec false`: cria o componente `Weather` dentro do módulo raiz `AppModule` em uma subpasta denominada weather.
- `ng g c Currency -is --spec false`: cria o componente `Currency` dentro do módulo raiz `AppModule` em uma subpasta denominada currency.
- `ng g c Movie -is --spec false`: cria o componente `Movie` dentro do módulo raiz `AppModule` em uma subpasta denominada movie.

Agora, com os novos componentes criados — inclusive classes, metadados e modelos — é possível ver como o `AppModule` é vinculado a esses componentes. Na Listagem 3, a linha 28 contém a declaração do módulo `AppModule`. Esse módulo contém cinco componentes, inclusive o componente raiz e os outros quatro componentes, conforme mostrado nas linhas 14 a 18.

Lista 3. app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
```

```
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';

import { AppComponent } from './app.component';
import { MenuComponent } from './menu.component';
import { WeatherComponent } from './weather/weather.component';
import { CurrencyComponent } from './currency/currency.component';
import { MovieComponent } from './movie/movie.component';

@NgModule({
  declarations: [
    AppComponent,
    MenuComponent,
    WeatherComponent,
    CurrencyComponent,
    MovieComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Criando as rotas

Comando route-creation

Estou fornecendo instruções aqui para uma criação de rota manual. A partir desta edição, um comando da CLI para criação de rota estará em fase de desenvolvimento. É possível verificar o site da CLI para ver se ela já está disponível.

Para o Angular navegar entre os componentes, você precisa criar *roteiros*. Sobrescreva o arquivo menu.component.html com o conteúdo da Listagem 4 para que o HTML inclua os menus corretos para cada componente.

Lista 4. menu.component.html

```
<div class="row">
   <div class="col-xs-12">
    <ul class="nav nav-pills">
     <li routerLinkActive="active"> <a [routerLink]="['/weather']" >Weather</a></li>
     <li routerLinkActive="active"> <a [routerLink]="['/movie']" >Movie Details</a></li>
     <li routerLinkActive="active"> <a [routerLink]="['/currency']" >Currency Rates</a></li>
    </ul>
  </div>
 </div>
```

O código na Listagem 4 fornece um mapeamento entre a GUI e o caminho da URL. Por exemplo, quando o usuário clica no botão Movie Details na GUI, o Angular sabe que ele precisa ser executado como se o caminho da URL fosse `http://localhost:4200/movie`.

Em seguida, você irá mapear os caminhos de URL para os componentes. Na mesma pasta que o módulo raiz, crie um arquivo de configuração chamado app.routing.ts e use o código na Listagem 5 e seu conteúdo.

Lista 5. app.routing.ts

```
import { Routes, RouterModule } from '@angular/router';
import { CurrencyComponent } from "./currency/currency.component";
import { WeatherComponent } from "./weather/weather.component";
import { MovieComponent } from "./movie/movie.component";
const MAINMENU_ROUTES: Routes = [
    //full : assegura que o caminho seja o caminho absoluto
    { path: '', redirectTo: '/weather', pathMatch: 'full' },
    { path: 'weather', component: WeatherComponent },
    { path: 'movie', component: MovieComponent },
    { path: 'currency', component: CurrencyComponent }
];
export const CONST_ROUTING = RouterModule.forRoot(MAINMENU_ROUTES);
```

Nesse caso, se o caminho relativo da URL for `movie`, você instruirá o Angular a chamar o componente `MovieComponent`. Em outras palavras, o caminho relativo `movie` mapeará para a URL `http://localhost:4200/movie`.

Agora você precisa vincular essa visualização ao seu componente pai. Sobrescreva o conteúdo do arquivo app.component.html com o seguinte código:

```
<div class="container">
 <app-menu></app-menu>
 <hr>
 <router-outlet></router-outlet>
</div>
```

O seletor `<app-menu></app-menu>` conterá o menu. O seletor `<router-outlet></router-outlet>` é um item temporário do componente atual. Dependendo do caminho da URL, o valor poderá ser qualquer um dos três componentes: weather, movie ou currency.

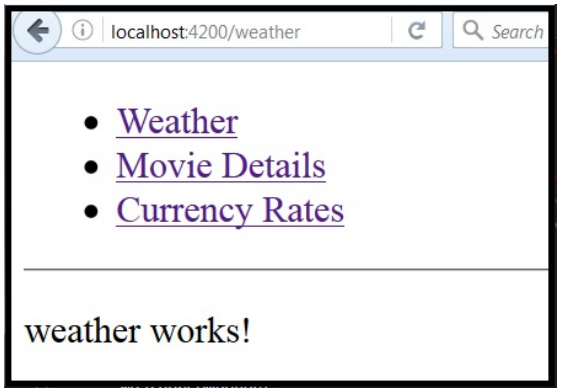
O módulo também deve ser notificado sobre essa rota. Inclua dois itens no arquivo app.module.ts, conforme mostrado nas linhas 11 e 25 na Listagem 6.

Lista 6. app.module.ts

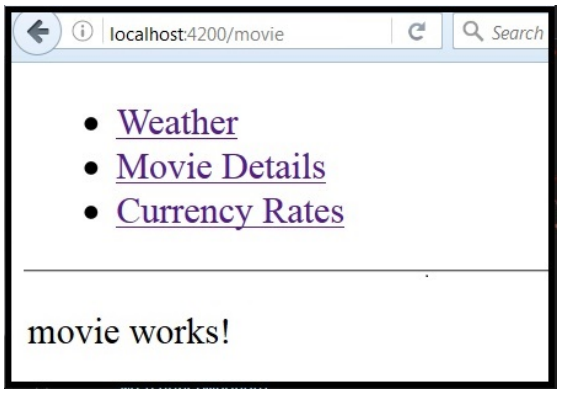
```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';

import { AppComponent } from './app.component';
import { MenuComponent } from './menu.component';
import { WeatherComponent } from './weather/weather.component';
import { CurrencyComponent } from './currency/currency.component';
import { MovieComponent } from './movie/movie.component';
import { CONST_ROUTING } from './app.routing';

@NgModule({
  declarations: [
    AppComponent,
    MenuComponent,
    WeatherComponent,
    CurrencyComponent,
    MovieComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
```

```
    HttpModule,
    CONST_ROUTING
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

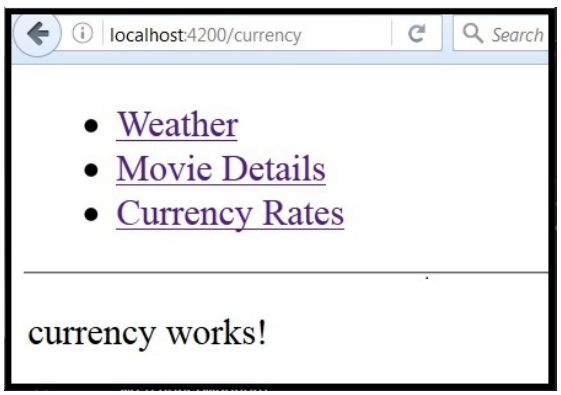
Agora, se você executar o aplicativo e clicar no link **Weather**, o aplicativo exibirá **weather works!**:



Se você clicar no link **Movie Details**, o aplicativo exibirá **movie works!**:



E se você clicar no link **Currency Rates**, o aplicativo exibirá **currency works!**:



Você modificou seu aplicativo Angular com sucesso para incluir diversos roteiros e componentes customizados. Agora você está pronto para as duas últimas partes importantes:

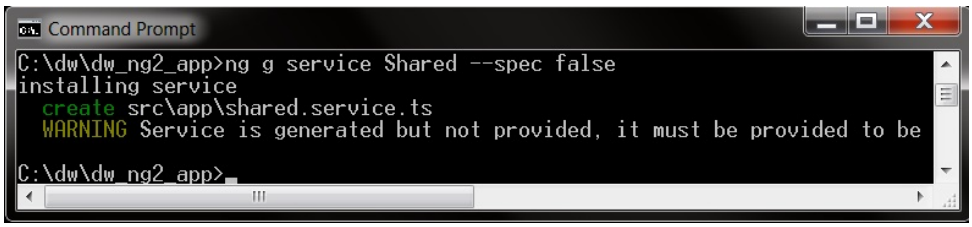
- Criar e configurar o serviço
- Consumir os microsserviços

Criando o serviço

Pare o processo Angular pressionando Ctrl-C. Execute o comando a seguir:

```
ng g service Shared --spec false
```

Esse comando cria o serviço no arquivo shared.service.ts na pasta do módulo raiz:



Substitua o conteúdo do shared.service.ts pelo código na Listagem 7.

Lista 7. shared.service.ts

```
import { Injectable } from '@angular/core';
import { Http, Headers, Response } from "@angular/http";
import 'rxjs/Rx';
import { Observable } from "rxjs";

@Injectable()
export class SharedService {
    weatherURL1 = "https://query.yahooapis.com/v1/public/yql?q=select%20*%20from%20weather.forecast%20where
%20woeid%20in%20(select%20woeid%20from%20geo.places(1)%20where%20text%3D%22";
    weatherURL2 = "%2C%20";
    weatherURL3 = "%22)&format=json&env=store%3A%2F%2Fdatatables.org%2Falltableswithkeys";
    findMovieURL1 = "http://www.omdbapi.com/?t=";
    findMovieURL2 = "&y=&plot=short&r=json";
    currencyURL = "http://api.fixer.io/latest?symbols=";
    totReqsMade: number = 0;
    constructor(private _http: Http) { }

    findWeather(city, state) {
        this.totReqsMade = this.totReqsMade + 1;
        return this._http.get(this.weatherURL1 + city + this.weatherURL2+ state + this.weatherURL3)
            .map(response => {
                { return response.json() };
            })
            .catch(error => Observable.throw(error.json()));
    }

    findMovie(movie) {
        this.totReqsMade = this.totReqsMade + 1;
        return this._http.get(this.findMovieURL1 + movie + this.findMovieURL2)
            .map(response => {
                { return response.json() };
            })
            .catch(error => Observable.throw(error.json().error));
    }

    getCurrencyExchRate(currency) {
        this.totReqsMade = this.totReqsMade + 1;
```

```
        return this._http.get(this.currencyURL + currency)
            .map(response => {
                { return response.json() };
            })
            .catch(error => Observable.throw(error.json()));
    }
}
```

As instruções `import ...` na Listagem 7 são obrigatórias para que qualquer trabalho funcione. A instrução `@Injectable()` é extremamente importante; ela indica que esse `service` é injetável nos outros componentes — uma técnica chamada de *injeção de dependência*.

A variável `totReqsMade` é declarada aqui e será usada para transmitir o valor entre os três componentes. Isso controlará o número total de solicitações de serviço feitas para obter os resultados dos microsserviços.

Você tem três métodos, com nomes que indicam sua funcionalidade: `findWeather()`, `findMovie()`e `getCurrencyExchRate()`. Durante a execução do método, o aplicativo Angular deixará seu navegador acessar a web para consumir os microsserviços. Agora, você irá vincular os componentes aos serviços de criação.

Substitua o conteúdo do arquivo movie.component.ts pelo código na Listagem 8.

Lista 8. movie.component.ts

```
import { Component, OnInit } from '@angular/core';
import { SharedService } from "./../shared.service";

@Component({
  selector: 'app-movie',
  templateUrl: './movie.component.html',
  styles: []
})
export class MovieComponent implements OnInit {
    id_movie: string = "";
    mv_Title: string = "";
    mv_Rated: string = "";
    mv_Released: string = "";
    mv_Director: string = "";
    mv_Actors: string = "";
    mv_Plot: string = "";
    constructor(private _sharedService: SharedService) {
    }

    ngOnInit() {
    }

    callMovieService() {
        this._sharedService.findMovie(this.id_movie)
            .subscribe(
            lstresult => {
                this.mv_Title = lstresult["Title"];
                this.mv_Rated = lstresult["Rated"];

                this.mv_Released = lstresult["Released"];
                this.mv_Director = lstresult["Director"];
                this.mv_Actors = lstresult["Actors"];
                this.mv_Plot = lstresult["Plot"];
            },
            error => {
```

```
                console.log("Error. The findMovie result JSON value is as follows:");
                console.log(error);
            }
            );
    }
}
```

Essa parte importante do código chama o método de serviço para obter os novos dados. Nesse caso, ela está chamando `callMovieService()` e depois chamará o método `this._sharedService.findMovie()`.

Da mesma forma, substitua o conteúdo do arquivo currency.component.ts pelo código na Listagem 9.

Lista 9. currency.component.ts

```
import { Component, OnInit } from '@angular/core';
import { SharedService } from "./../shared.service";

@Component({
  selector: 'app-currency',
  templateUrl: './currency.component.html',
  styles: []
})
export class CurrencyComponent implements OnInit {

  id_currency: string = "";
  my_result: any;
  constructor(private _sharedService: SharedService) {
  }

  ngOnInit() {
    }

  callCurrencyService() {
    this._sharedService.getCurrencyExchRate(this.id_currency.toUpperCase())
      .subscribe(
            lstresult => {
                this.my_result = JSON.stringify(lstresult);
      },
      error => {
        console.log("Error. The callCurrencyService result JSON value is as follows:");
        console.log(error);
      }
      );
  }
}
```

E substitua o conteúdo do arquivo weather.component.ts pelo código na Listagem 10.

Lista 10. weather.component.ts

```
import { Component, OnInit } from '@angular/core';
import { SharedService } from "./../shared.service";

@Component({
  selector: 'app-weather',
  templateUrl: './weather.component.html',
  styles: []

})
```

```
export class WeatherComponent implements OnInit {
  id_city: string = "";
  id_state: string = "";
  op_city: string = "";
  op_region: string = "";
  op_country: string = "";
  op_date: string = "";
  op_text: string = "";
  op_temp: string = "";
  constructor(private _sharedService: SharedService) {
  }

  ngOnInit() {
    }

  callWeatherService() {
    this._sharedService.findWeather(this.id_city, this.id_state)
      .subscribe(
            lstresult => {
        this.op_city = lstresult["query"]["results"]["channel"]["location"]["city"];
        this.op_region = lstresult["query"]["results"]["channel"]["location"]["region"];
        this.op_country = lstresult["query"]["results"]["channel"]["location"]["country"];
        this.op_date = lstresult["query"]["results"]["channel"]["item"]["condition"]["date"];
        this.op_text = lstresult["query"]["results"]["channel"]["item"]["condition"]["text"];
        this.op_temp = lstresult["query"]["results"]["channel"]["item"]["condition"]["temp"];
      },
      error => {
        console.log("Error. The findWeather result JSON value is as follows:");
        console.log(error);
      }
      );
  }
}
```

Agora atualize o módulo para incluir os serviços. Edite o arquivo app.module.ts para incluir as duas instruções nas linhas 12 e 28 da Listagem 11.

Lista 11. app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';

import { AppComponent } from './app.component';
import { MenuComponent } from './menu.component';
import { WeatherComponent } from './weather/weather.component';
import { CurrencyComponent } from './currency/currency.component';
import { MovieComponent } from './movie/movie.component';
import { CONST_ROUTING } from './app.routing';
import { SharedService } from "./shared.service";

@NgModule({
  declarations: [
    AppComponent,
    MenuComponent,
    WeatherComponent,
    CurrencyComponent,
    MovieComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    CONST_ROUTING
```

```
  ],
  providers: [SharedService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Modificando a visualização de componentes

Agora vem a parte final do quebra-cabeça. É necessário instruir o arquivo HTML a chamar os métodos de serviço corretos. Para isso, substitua o conteúdo do arquivo movie.component.html pelo código na Listagem 12.

Lista 12. movie.component.html

```
<h2>Open Movie Database</h2>
<div class="col-md-8 col-md-offset-2">
 <div class="form-group">
  <input type="text" required [(ngModel)]="id_movie" (change)="callMovieService()" class="form-control"
 placeholder="Enter Movie name ...">
  <br><br>
  <h3>Movie Details</h3>
  <br>
  <p class="well lead">
      <i> Title :</i> {{ this.mv_Title }} <br>
      <i> Plot :</i> {{ this.mv_Plot }} <br>
      <i> Actors :</i> {{ this.mv_Actors }} <br>
      <i> Directed by :</i> {{ this.mv_Director }} <br>
      <i> Rated :</i> {{ this.mv_Rated }} <br>
      <i> Release Date :</i> {{ this.mv_Released }} <br>
  </p>
  <p class="text-info">Total # of all the service requests including Weather, Movie, and Currency is :
      <span class="badge">{{this._sharedService.totReqsMade}}</span>
  </p>
 </div>
</div>
```

Várias coisas importantes são codificadas no movie.component.html:

- `{{ this._sharedService.totReqsMade }}`: esse é o valor que é controlado no nível de serviço e que compartilha seus valores entre os três componentes de aplicativo.
- `[(ngModel)]="id_movie"`: a entrada da GUI inserida pelo usuário é transmitida para a classe que chama esse HTML. Nesse caso, a classe é `MovieComponent`.
- `(change)="callMovieService()`": sempre que o valor desse campo for alterado, você instruirá o sistema a chamar o método `callMovieService()` que está presente no arquivo movie.component.ts.
- `{{ this.mv_Title }}, {{ this.mv_Plot }}, {{ this.mv_Actors }}, {{ this.mv_Director }}, {{ this.mv_Rated }}, {{ this.mv_Released }}`: exibe resultados das chamadas de serviço feitas a partir de `callMovieService() -> this._sharedService.findMovie(this.id_movie)`.

Substitua o conteúdo do arquivo weather.component.html pelo código na Listagem 13.

Lista 13. weather.component.html

```
<h2>Yahoo! Weather </h2>
<div class="col-md-8 col-md-offset-2">
 <div class="form-group">
  <input type="text" [(ngModel)]="id_city" class="form-control" placeholder="Enter City name ..."><br>
  <input type="text" [(ngModel)]="id_state" class="form-control" placeholder="Enter State. Example CA for
 California ..."><br>
  <button type="button" class="btn btn-primary" (click)="callWeatherService()">Submit</button>
  <br><br><br>
  <br>
  <p class="well lead">
    <i>City, State, Country :</i> {{ this.op_city }} {{ this.op_region }} {{ this.op_country }} <br>
    <i>Current Condition :</i> {{ this.op_text }} <br>
    <i>Current Temperature :</i> {{ this.op_temp }} <br>
  </p>
  <p class="text-info">Total # of all the service requests including Weather, Movie, and Currency is :
    <span class="badge">{{this._sharedService.totReqsMade}}</span>
  </p>
 </div>
</div>
```

Por fim, substitua o conteúdo do arquivo currency.component.html pelo código na Listagem 14.

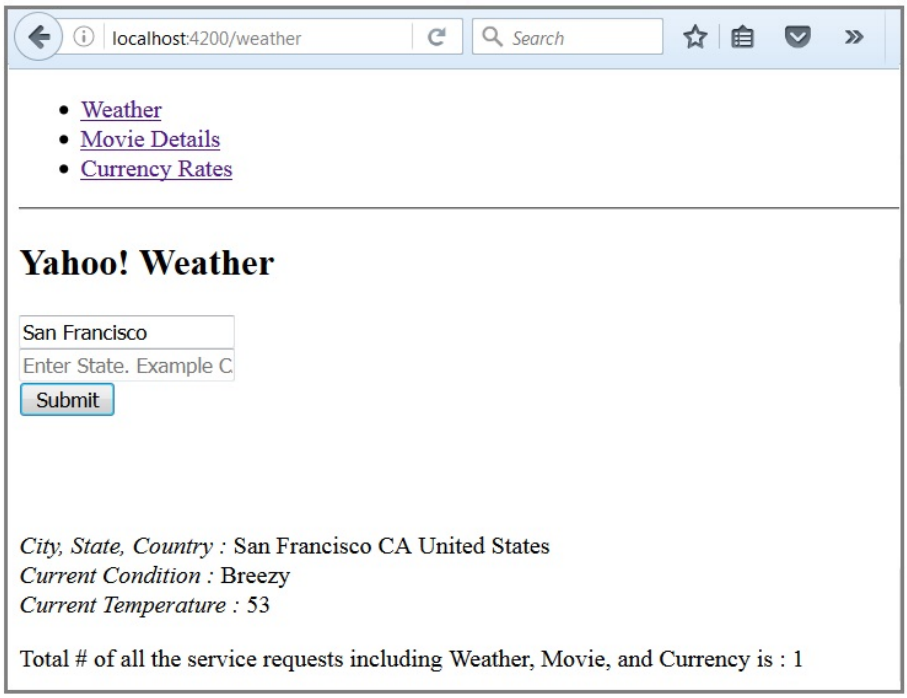
Lista 14. currency.component.html

```
<h2>Currency Exchange Rates</h2>
<div class="col-md-8 col-md-offset-2">
 <div class="form-group">
  <input type="text" [(ngModel)]="id_currency" (change)="callCurrencyService()" class="form-control"
 placeholder="Enter Currency Symbol. Example: GBP(,AUD,INR)...">
  <br><br>
  <h3>Rate Details</h3>
  <br>
  <p class="well lead">Exchange rate relative to Euro in a JSON format: : {{ this.my_result }} </p>
  <p class="text-info">Total # of all the service requests including Weather, Movie, and Currency is :
    <span class="badge">{{this._sharedService.totReqsMade}}</span>
  </p>
 </div>
</div>
```

Agora, se tudo correu conforme esperado, o aplicativo poderá aceitar entrada do usuário no navegador.

Executando o aplicativo e melhorando a UI

Execute o aplicativo agora, insira alguns valores e visualize os resultados. Por exemplo, clique no link Weather e insira `San Francisco` para ver as condições climáticas da cidade:



Teste suas novas aptidões em Angular

Atualmente, os campos de entrada do seu aplicativo não implementam validação ou manipulação de erros. Você pode tentar incluir esses recursos por conta própria. Dica: inclua métodos no serviço chamado **validateMovie(movie-name)**, **validateCurrency(currency-name)**, **validateCity(city-name)**e **validateState(state-name)** Em seguida, chame esses métodos a partir dos componentes correspondentes.

Tudo está indo muito bem, mas a UI poderia ficar mais atraente. Uma maneira de deixar a GUI mais agradável é usar autoinicialização. (O Material do Angular 2 seria a opção ideal, mas a partir desta edição, ele não foi liberado oficialmente).

Acesse Autoinicialização página Introdução e copie as duas linhas a seguir dessa página na área de transferência:

```
<!-- CSS compilado e reduzido mais recente -->
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
 integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">
```

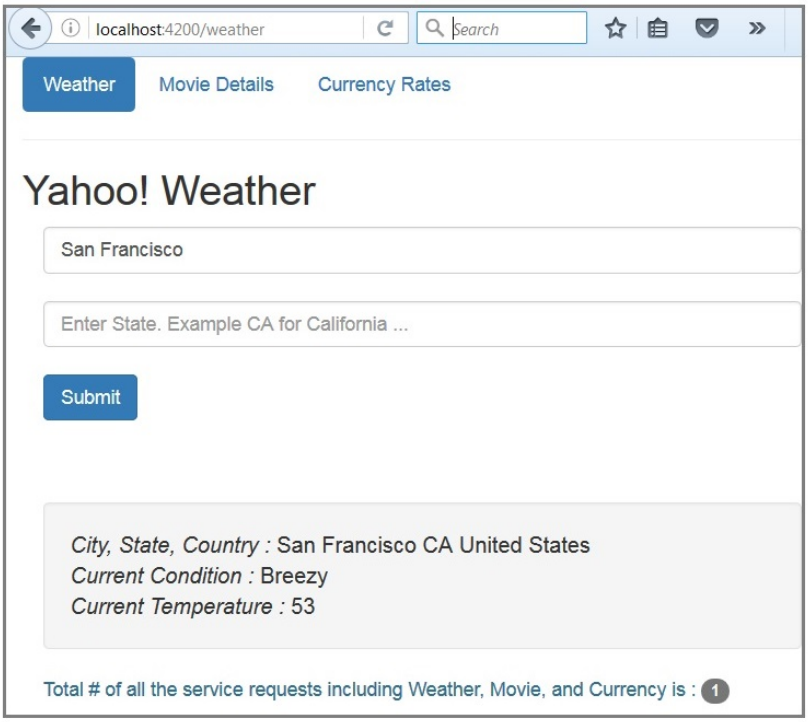
Abra o arquivo index.html e cole a instrução que você acabou de copiar abaixo da linha 8.

Lista 15. index.html

```
 <!doctype html>
<html>

<head>
  <meta charset="utf-8">
  <title>DwNg2App</title>
  <base href="/">
```

```
  <!-- CSS compilado e reduzido mais recente -->
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
 integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>

<body>
  <app-root>Loading...</app-root>
</body>

</html>
```

Agora o aplicativo terá uma aparência melhor no navegador, com estilos mais legíveis e botões de menu no lugar dos links Weather, Movie Details e Currency Rates:



index.html aninhado

Agora pare um instante para entender por que o aplicativo que você acabou concluir faz jus ao nome SPA.

Quando o aplicativo Angular é iniciado, o servidor envia o arquivo index.html para o navegador, e o arquivo index.html é o único que o navegador exibe. Tudo que o Angular faz na página é inserido nessa visualização:

O seletor `<app-root>` no final do index.html é substituído pelo conteúdo do app.component.html. Por sua vez, o app.component.html contém dois seletores: `<app-menu>` e `<router-outlet>`. O seletor `<app-menu>` é preenchido com o conteúdo do menu.component.html e o `<router-outlet>` é preenchido dinamicamente, dependendo da seleção do menu — ou seja, com o conteúdo do weather.component.html, currency.component.html ou movie.component.html.

Todos os seletores são estáticos, exceto o seletor reservado Angular `<router-outlet></router-outlet>`. Esse seletor é preenchido durante o tempo de execução, dependendo do valor do "router". Apenas o index.html é exibido, e todos os outros arquivos HTML que você codificou são aninhados no arquivo index.html.

Simulando o servidor

Seu projeto Angular está sendo executado com sucesso em um computador de desenvolvimento. Se você tiver acesso a um servidor sandbox remoto, é possível mover o código nele para ver como o aplicativo se comportará quando um usuário executá-lo. (Caso contrário, é possível ir para o tutorial Conclusão.)

Certifique-se de que o Node.js e a CLI do Angular estejam instalados no sandbox remoto. Compacte tudo na pasta do projeto local, exceto o diretório node_modules e seu conteúdo. Copie o arquivo de projeto compactado no servidor sandbox e descompacte-o. Acesse o diretório do servidor que contém o package.json e execute o comando `npm install`. O arquivo package.json permite que o comando `npm install` acesse o repositório público NPM e instale todas as versões de pacote necessárias. A execução desse comando também cria automaticamente o diretório node_modules e seu conteúdo no servidor.

Execute o comando `ng serve` para iniciar o aplicativo no servidor sandbox, como você fez no computador de desenvolvimento. Pressione Ctrl-C para parar o processo. Mais uma vez, se quiser conhecer outras opções no `ng serve`, execute o comando `ng help`.

Execute o aplicativo com o comando `ng serve --port sandbox-port# --host sandbox-hostname`.

Agora o aplicativo Angular está disponível na URL http://*sandbox-hostname*:*sandbox-port#*. Enquanto você executa o aplicativo na URL no navegador do computador de desenvolvimento, pare o processo Angular do servidor sandbox pressionando Ctrl-C no servidor. Observe que o aplicativo inteiro está em execução no navegador do computador de desenvolvimento, mesmo que o processo Angular do servidor esteja inativo. Isso quer dizer que a tecnologia SPA está em ação. Quando o aplicativo está no navegador, o controle nunca vai para o servidor, a menos que o aplicativo precise de novos dados.

Em um mundo de SPA, os navegadores são os novos servidores. Se dez usuários estiverem executando o aplicativo, dez navegadores estarão manipulando o carregamento. Se 1.000 estiverem executando o aplicativo, 1.000 navegadores estarão manipulando o carregamento. O aplicativo Angular inteiro fica sob o controle do navegador, exceto para qualquer lógica — como operações de autenticação e do banco de dados — que esteja em execução no servidor.

Um melhor desempenho e a redução da tensão do servidor podem melhorar a experiência e a satisfação do usuário — que são o segredo para o sucesso de qualquer negócio.

> *“ Em um mundo de SPA, os navegadores são os novos servidores. ”*

Conclusão

Você aprendeu como codificar e executar um SPA utilizando Angular 2 no computador de desenvolvimento e no servidor sandbox. Para conhecer os requisitos de produção, consulte o departamento de TI. Uma das coisas mais importantes que a maioria dos aplicativos de produção tem é a autenticação e a autorização, que deveriam ser manipuladas no servidor principalmente por motivos de segurança. Provavelmente, você vai precisar de um servidor dedicado para tratar dessas operações. Para esse servidor, você poderia usar Node.js, que pode agir como um servidor que é executado no backend, enquanto o Angular 2 é executado no frontend. (Tanto o Node quanto o Angular são originários do Google, portanto, eles funcionam muito bem juntos).

Outras técnicas que você pode considerar para melhorar o desempenho do aplicativo incluem:

- Agrupamento: um processo que combina qualquer um de seus programas em um único arquivo.
- Compactação: compacta o arquivo de pacote configurável para manter o tamanho mínimo do projeto.
- Compilação Ahead-of-time (AoT): o servidor cuida da compilação antecipada durante o processo de build em vez de o navegador fazer a compilação Just-in-time (JIT) durante o tempo de execução.

Agradecimentos

Quero agradecer ao time da IBM, que conta com Guy Huinen, Dean P Cummings, Don Turner e Mark Shade, pela revisão e pelo suporte.

Recursos para download

Descrição	Nome	Tamanho
Sample app for sandbox	dw_ng2_app_for_sandbox.zip	18 KB

Temas relacionados

- Mastering MEAN
- Modularize Angular applications with webpack
- Angular architecture overview
- Tutorial: Tour of heroes
- Explore Angular resources
- Component interaction