**Angular How-to: Editable Config Files**

The Angular-CLI is the recommended way to build a production-ready app, complete with bundling, uglifying, and tree-shaking. An Angular-CLI generated application even comes with a mechanism for creating environment-specific versions. However, those configuration files are in TypeScript and do not allow editing by IT staff or automated deployment tools such as VSTS. This post provides the steps and code samples for using a JSON configuration file, which can be customized for multiple environments.

**Define TypeScript interface for config settings**

The use of interfaces in an Angular app provides intellisense and type-safety for your entities. For this example, refer to this sample configuration file.

**app-config.model.ts**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | export interface IAppConfig {      env: {          name: string;      };      appInsights: {          instrumentationKey: string;      };      logging: {          console: boolean;          appInsights: boolean;      };      aad: {          requireAuth: boolean;          tenant: string;          clientId: string;        };      apiServer: {          metadata: string;          rules: string;      };  } |

**Create JSON config files**

A convenient place to store configuration files is under the assets folder of your project. Using the interface defined above, sample files could look as follows:

**assets\config\config.dev.json**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | {      "env": {      "name": "DEV"       },      "appInsights": {      "instrumentationKey": "<dev-guid-here>"       },      "logging": {      "console": true,      "appInsights": false      },      "aad": {      "requireAuth": true,      "tenant": "<dev-guid-here>",      "clientId": "<dev-guid-here>"      },      "apiServer": {      "metadata": "https://metadata.demo.com/api/v1.0/",      "rules": "https://rules.demo.com/api/v1.0/"      }  } |

**assets\config\config.deploy.json***(Note placeholders that are replaced during deployment)*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | {      "env": {      "name": "#{envName}"      },      "appInsights": {      "instrumentationKey": "#{appInsightsKey}"      },      "logging": {      "console": true,      "appInsights": true      },      "aad": {      "requireAuth": true,      "tenant": "#{aadTenant}",      "clientId": "#{aadClientId}"      },      "apiServer": {      "metadata": "https://#{apiServerPrefix}.demo.com/api/v1.0/",      "rule": "https://#{apiServerPrefix}.demo.com/api/v1.0/",      }  } |

**Continue to use environment.ts with Angular-CLI build**

The Angular-CLI creates several TypeScript environment files in the environments folder. They will still be used, but contain only the environment name.

**environments\environment.dev.json**

|  |  |
| --- | --- |
| 1  2  3 | export const environment = {      name: 'dev'  }; |

**environments\environment.deploy.json**

|  |  |
| --- | --- |
| 1  2  3 | export const environment = {      name: 'deploy'  }; |

**angular.json**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | "projects": {    "my-app": {      "architect": {        "build": {          "configurations": {            "deploy": {              "fileReplacements": [                {                  "replace": "src/environments/environment.ts",                  "with": "src/environments/environment.deploy.ts"                }              ],              . . .            }          }        },        "serve": {          . . .          "configurations": {            "deploy": {              "browserTarget": "my-app:build:deploy"            } |

**Create a service to read config file**

This service will read the correct config file and store the result in a static field in this class..

**app.config.ts***(Note the use of the interface defined above and config file naming convention to retrieve the appropriate file.)*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | import { Injectable } from '@angular/core’;  import { HttpClient } from '@angular/common/http';  import { environment } from '../environments/environment';  import { IAppConfig } from './models/app-config.model';  @Injectable()  export class AppConfig {      static settings: IAppConfig;      constructor(private http: HttpClient) {}      load() {          const jsonFile = `assets/config/config.${environment.name}.json`;          return new Promise<void>((resolve, reject) => {              this.http.get(jsonFile).toPromise().then((response : IAppConfig) => {                 AppConfig.settings = <IAppConfig>response;                 resolve();              }).catch((response: any) => {                 reject(`Could not load file '${jsonFile}': ${JSON.stringify(response)}`);              });          });      }  } |

**Load config file prior to app creation**

Angular includes a token named APP\_INITIALIZER that allows our app to execute code when the application is initialized. In the app module, use this token to invoke the load method in our config service. Since our method returns a promise, Angular will delay the initialization until the promise is resolved.

**app.module.ts**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | import { APP\_INITIALIZER } from '@angular/core';  import { AppConfig } from './app.config';    export function initializeApp(appConfig: AppConfig) {    return () => appConfig.load();  }  @NgModule({      imports: [ , , , ],      declarations: [ . . . ],      providers: [         AppConfig,         { provide: APP\_INITIALIZER,           useFactory: initializeApp,           deps: [AppConfig], multi: true }      ],      bootstrap: [        AppComponent      ]  })  export class AppModule { } |

**Consume the app settings throughout the application**

The config settings are now available from anywhere in the application and they include type-checking provided by the interface.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | export class DataService {      protected apiServer = AppConfig.settings.apiServer;      . . .      if (AppConfig.settings.aad.requireAuth) { . . . }  }  export class LoggingService {      . . .      instrumentationKey: AppConfig.settings && AppConfig.settings.appInsights ?                          AppConfig.settings.appInsights.instrumentationKey : ''      . . .      if (AppConfig.settings && AppConfig.settings.logging) { . . . }  } |

Note: to build a production version of the app using an environment name other than prod, use this command:

ng build –configuration=deploy