



# Angular Universal + Firebase functions

Published on 10 abril, 2018

¿Has intentado montar Angular Universal en tu proyecto por primera vez? ¿Me equivoco si digo que ha sido un parto? No creo...

Por suerte para ti, he escrito esta **guía actualizada y simplificada para montar Universal** en tu proyecto Angular.

De paso, te traigo también un **bonus track** sobre cómo **lanzar Universal en un entorno serverless como Firebase Cloud Functions**.

## Guia simplificada de Angular Universal

Ya te he hablado de Universal otras veces: Por si no me leías, Angular Universal es la herramienta de *Server Side Rendering* (SSR) que te permite renderizar Angular desde el servidor. Los principales beneficios de usar SSR es que facilita el posicionamiento SEO y reduce el tiempo de carga inicial de la página.

Aprende a hacer apps móviles con **ionic 3** [Ver curso](#)

Lo que no te he dicho hasta ahora, es que **desde la Angular CLI v1.6... ¡añadir Universal a tu proyecto es más fácil que nunca!**

Esto va a ser una guía paso a paso, así que... ¡empezamos!

## 1 – Angular CLI 1.6+

En primer lugar necesitarás una versión reciente de la CLI de Angular (la CLI 1.6 o superior).

Para instalar/actualizar la CLI, ejecuta desde terminal:

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

## 2 – Proyecto Angular

Si no tienes proyecto, el siguiente paso es crear uno a través de la CLI que acabas de instalar. Para ello, ejecuta:

```
ng new my-project
```

Hecho esto puedes pasar al punto siguiente.

No obstante, si ya tienes un proyecto creado con una versión anterior de la CLI, puedes actualizarlo así:

1. abre el archivo `package.json`
2. modifica el número de versión de la `devDependency` "`@angular/cli`" a la más reciente (la de [aquí](#)).
3. borra la carpeta `node_dependencies` de tu proyecto
4. e instala de nuevo las dependencias con `npm install`.

## 3 – Generando Universal con la CLI

```
ng generate universal
```

Si todo ha ido bien, el comando te habrá creado los archivos:

- `src/app/app.server.module.ts`
- `src/main.server.ts`
- `src/tsconfig.server.json`

Los dos primeros archivos sirven como punto de entrada a tu app en entorno servidor. El tercero es la configuración TS para compilarlo en un entorno de NodeJS.

Además, se te habrán actualizado estos otros archivos:

- **package.json**: Añade una dependencia.
- **.angular-cli.json**: Configura la CLI para compilar la versión Universal.
- **src/main.ts**: Espera a cargar el DOM antes de rehidratar en local.
- **src/app/app.module.ts**: Asocia ambas versiones de Angular para la rehidratación en servidor.
- **.gitignore**: Ignora nuevos outputs que generará el build.

## 4 – Instala las nuevas dependencias

El comando anterior ha añadido `@angular/platform-server` a tus dependencias, pero no se ha instalado.

Para que se instale, debes ejecutar en terminal:

```
npm install
```

### ¿Qué tienes llegado este punto?

Ahora tienes lo básico para generar un código Angular que se pueda ejecutar también en servidor. Si te fijas en `.angular-cli.json`, se ha añadido a la configuración una nueva app para compilar la versión Universal y colocarla en la carpeta `dist-server`.

```
ng build --prod && ng build --prod --app 1 --output-hashing=none
```

El comando anterior te creará 2 carpetas:

- La carpeta **dist** , con la app Angular original.
- La carpeta **dist-server** , con la versión Universal de tu app, es decir tu app Angular compilada para servidor.

Esto, por sí mismo, no es suficiente para hacer SSR. **Necesitas crear un servidor que se encargue de renderizar la versión Universal** cuando recibe alguna llamada. El servidor puede montarse con NodeJS, pero también podría implementarse con otras tecnologías.

Yo voy a mostrarte como crear el servidor usando Express, el popular framework de NodeJS.

## 5 – Añadir dependencias adicionales

Para el servidor de *Express* necesitarás añadir algunas dependencias más:

- **express** – Framework *Express*.
- **@nguniversal/express-engine** – Adaptador de Angular Universal para *Express*.
- **@nguniversal/module-map-ngfactory-loader** – Dependencia para poder realizar enrutado *lazy-loading* desde el servidor.
- **reflect-metadata** – Dependencia que necesitas para algunos polyfills.

Las puedes instalar desde terminal, así:

```
# bash
npm install --save @nguniversal/module-map-ngfactory-loader
@nguniversal/express-engine express reflect-metadata
```

## 6 – Crear servidor Express para el SSR

En la raíz del proyecto, crea la carpeta **server** .

```
// server/index.ts

// These are important and needed before anything else
import 'zone.js/dist/zone-node';
import 'reflect-metadata';

import { enableProdMode } from '@angular/core';
import * as express from 'express';
import { join } from 'path';

// NOTE: Leave this as require() since this file is built Dynamically from
webpack
const { AppServerModuleNgFactory, LAZY_MODULE_MAP } =
require('./main.bundle');

// NgUniversalTools: Express Engine and moduleMap for Lazy Loading
import { ngExpressEngine } from '@nguniversal/express-engine';
import { provideModuleMap } from '@nguniversal/module-map-ngfactory-
loader';

// Faster server renders w/ Prod mode (dev mode never needed)
enableProdMode();

// Express server
const app = express();
const PORT = process.env.PORT || 4000;
const DIST_FOLDER = join(process.cwd(), 'dist');

app.engine('html', ngExpressEngine({
  bootstrap: AppServerModuleNgFactory,
  providers: [
    provideModuleMap(LAZY_MODULE_MAP)
  ]
})));

app.set('view engine', 'html');
app.set('views', DIST_FOLDER);

/* TODO: implement data requests securely
// app.get('/api/*', (req, res) => {
// res.status(404).send('data requests are not supported');
// });
*/

// Server static files from browser
```

```
app.get('*', (req, res) => {  
  res.render(join(DIST_FOLDER, 'index.html'), { req });  
});  
  
// Start up the Node server  
app.listen(PORT, () => {  
  console.log(`Node server listening on http://localhost:${PORT}`);  
});
```

Lo que ves es un servidor muy sencillo de Node, montado con *Express framework*. *Express* da para varios posts, pero voy a comentarte los 4 puntos básicos que se ven aquí.

1. Has creado el servidor invocando la función **express()** , y se lo has asignado a la variable **app** .
2. Le pasas como motor de renderizado HTML, la *promise* que devuelve la función **ngExpressEngine** . Esta función es un *wrapper* que te simplifica las cosas. Contiene mucha magia:
  1. Internamente llama a **renderModuleFactory** , que es quien se encarga de generar el HTML para las peticiones de cliente.
  2. Su parámetro inicial es **AppServerModule** , que es el módulo que has implementado antes.
  3. Puedes pasar *providers* adicionales con datos que solo puede obtener el actual servidor en ejecución.
3. Utilizas **app.get()** para filtrar las peticiones HTTP.
  1. En este caso, indicas que para todos los archivos estáticos (urls que acaban con una extensión, de ahí la expresión **\*.\*** ), se sirvan de la carpeta **dist** . Webpack moverá ahí los archivos estáticos que se necesitan (JS, CSS, imgs) gracias a los cambios que ha hecho el comando **ng generate universal** sobre **angular-cli.json**.
  2. El resto de rutas, se las pasarás al servidor como si fuera simple navegación.
4. Con **app.listen** lanzas el servidor, escuchando en el puerto indicado.

## 7 – Compilador de Typescript

Has creado el servidor *Express* utilizando Typescript, por lo que tendrás que compilarlo

En frontend, estás acostumbrado a crear *bundles* donde minificas y juntas todo el código en un mismo archivo. Eso tiene sentido para cliente, porque reduces el tamaño y cantidad de archivos a enviar, pero en servidor no tiene tanto sentido.

*NodeJS* dispone de una arquitectura de módulos muy eficiente y además carga los archivos en local: En este caso, crear un bundle con las dependencias de *express* solo serviría para incrementar el tiempo de compilación.

Como no tiene sentido crear un bundle para Node, te recomiendo compilar únicamente el archivo *index.ts* para convertirlo a JS.

En lugar de complicar las cosas con Webpack (o similar), puedes hacerlo directamente con la CLI de Typescript.

Comprueba que la tienes instalada ejecutando desde terminal:

```
tsc --version
```

Si no tienes la CLI de TS, instálala con el siguiente comando:

```
sudo npm install -g typescript
```

## 8 – Configuración de compilación TS del servidor

Typescript necesitará saber como quieres compilar el servidor, así que tienes que crear un archivo de configuración.

En la carpeta **server** , crea el archivo: **tsconfig.ssr.json** con el siguiente contenido:

```
{
  "extends": "../tsconfig.json",
  "compilerOptions": {
    "outDir": "../dist-server",
    "sourceMap": false,
    "baseUrl": "./",
    "module": "commonjs",
    "target": "es2015"
  }
}
```

```
    },  
    "files": [  
      "index.ts"  
    ],  
    "exclude": [  
      "**/*.test.ts",  
      "**/*.spec.ts"  
    ]  
  }  
}
```

Esta configuración compilará el archivo TS `server/index.ts` para Node (de ahí el uso de módulos *commonjs*) y lo colocará en la carpeta `dist-server`.

## 9 – Añadiendo comandos de compilación a package.json

Abre el archivo `package.json`, y en la zona de scripts, añade las siguientes líneas:

```
"tsc:server": "tsc -p server/tsconfig.ssr.json",  
"build:client-and-server-bundles": "ng build --prod --aot && ng build --prod  
--aot --app 1 --output-hashing=false",  
"build:universal": "npm run build:client-and-server-bundles && npm run  
tsc:server",  
"serve:universal": "node dist-server/index.js"
```

Acabas de crear 4 comandos que puedes ejecutar mediante **npm**:

- **tsc:server**: Es el que compila el servidor Express siguiendo la configuración anterior.
- **build:client-and-server-bundles**: Es el comando que compila tu app Angular para producción y usando AOT, tanto la versión de navegador como la versión de servidor.
- **build:universal**: Es un comando útil que se encarga de llamar a los dos comandos anteriores. Es todo cuando necesitas para preparar el SSR.
- **serve:universal**: Lanza el servidor Express que has creado.

## 10 – Probando los resultados



```
npm run build:universal
```

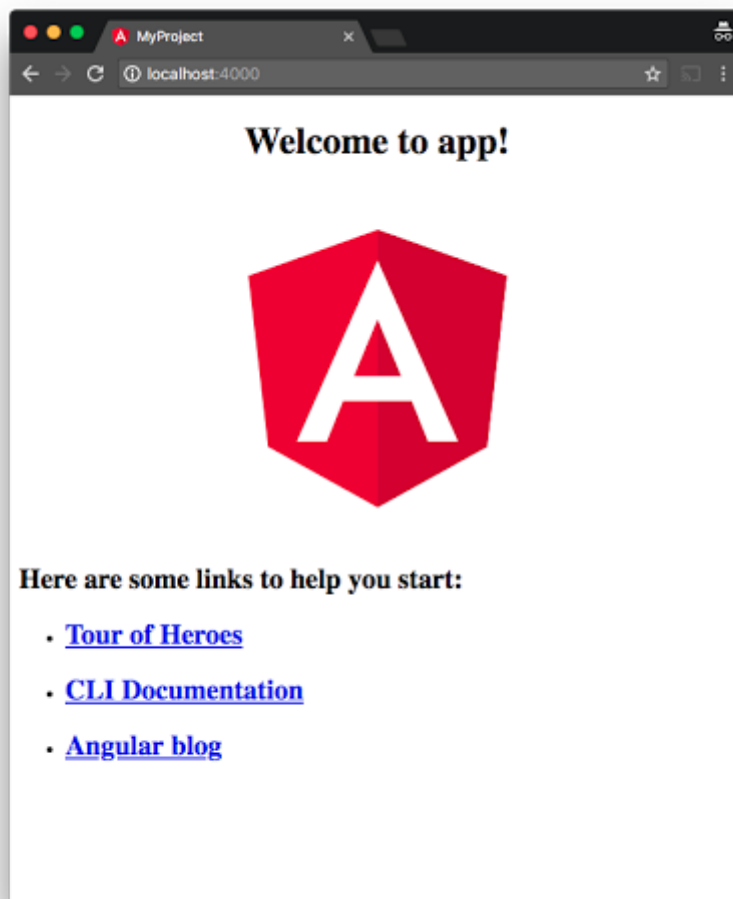
Esto debería compilarte la parte cliente de Angular, una versión alternativa para servidor, y el propio servidor Express.

Ahora, ejecuta el servidor, a ver que tal:

```
npm run serve:universal
```

¡Tachaaaaan!

Si estabas partiendo de un proyecto nuevo, deberías ver esto:



-Pero... ¿y como sé yo si lo que veo se ha renderizado desde cliente o servidor?

-Esa es una duda muy interesante, gracias por preguntar 😊

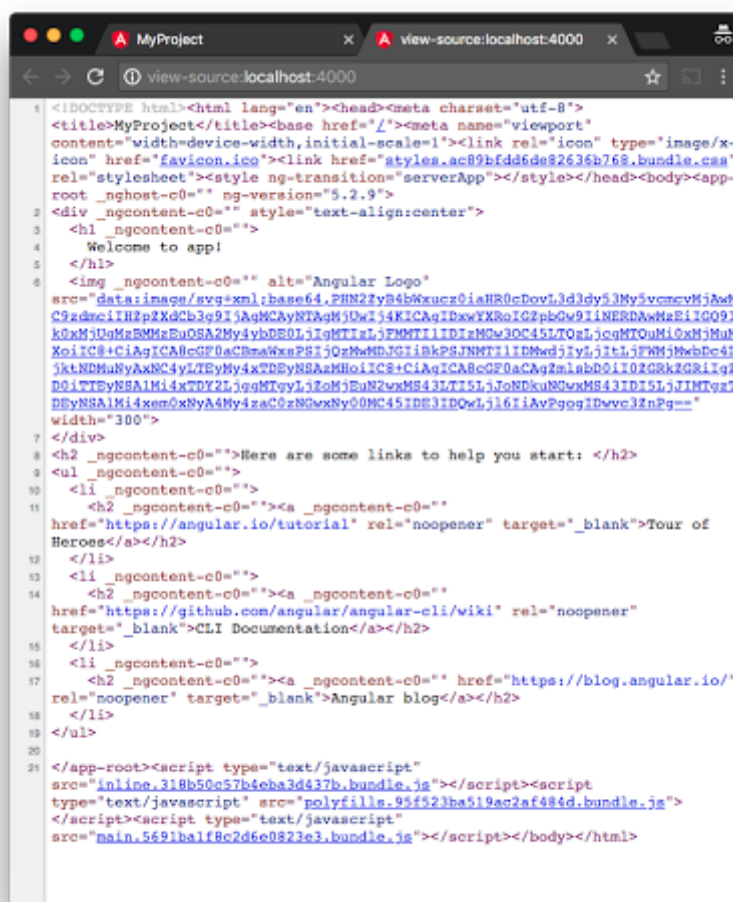
Si todo ha ido bien, Angular Universal habrá renderizado la página desde Express, la

Si el servidor, por algún motivo, no es capaz de renderizar la página, lo más probable es que estés viendo el mismo resultado, pero generado directamente desde cliente (es decir, sin SSR, lo que no valdría para SEO, por ejemplo).

## Entonces **¿cómo comprobar que funciona el Server Side Rendering?**

Fácil. En la página que ves en el navegador, haz click con el botón derecho y selecciona “Ver código fuente de la página”. Si estás usando un navegador distinto de Chrome, haz lo correspondiente para mostrar el código HTML.

Si te ha funcionado el SSR, deberías ver el código completo de la página dentro de la etiqueta `<app-root>` de Angular. Como en la imagen:



**En cambio, si se ha producido un error de SSR, lo que deberías ver es la etiqueta `<app-root>` vacía.** Ese es el indicador de que el SSR no ha funcionado, y lo que se está sirviendo es una página Angular normal que se carga desde cliente.

Curso

OFERTA

## Componentes Angular Nivel PRO

Domina los componentes de Angular (Angular 2/4/5+) como un experto y crea componentes técnicamente brillantes.

Idioma: **Español**

23 € ~~110 €~~

## Bonus Track: Deploy en Firebase

Para esta parte, asumo que tienes un conocimiento mínimo de Firebase y que ya tienes un proyecto. Si no es el caso, deberías [crearte un usuario de Firebase y añadir un nuevo proyecto desde ahí](#).

### 1 – Instalando la CLI de Firebase

Ejecuta el siguiente comando para instalar o actualizar la CLI de firebase:

Aprende a hacer apps móviles con **ionic 3** [Ver curso](#)

```
npm install -g firebase-tools
```

## 2 – Haz login en Firebase

Desde terminal, ejecuta:

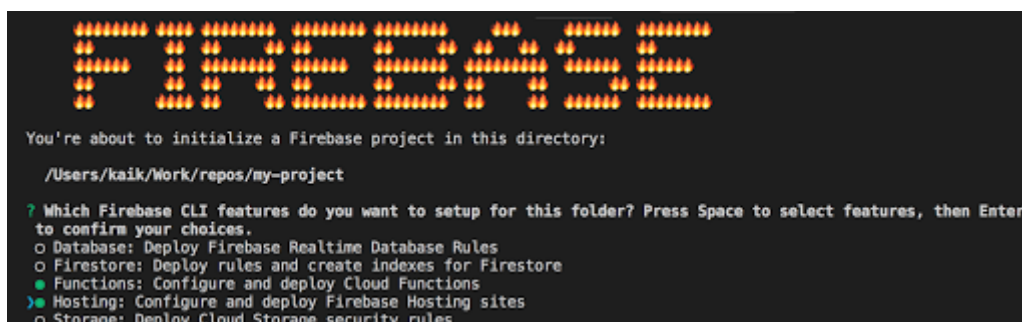
```
firebase login
```

## 3 – Añade Firebase al proyecto

Desde el directorio de tu proyecto Angular, ejecuta el comando para inicializarlo como proyecto de Firebase:

```
firebase init
```

En el diálogo que aparece, selecciona **Functions** y **Hosting**.



A continuación elige el proyecto Firebase al que quieres asociarlo.

## 4 – Funciones de firebase

Te preguntará que como quieres escribir las funciones, si en TS o en JS. Como ya has creado un mecanismo para compilar tu servidor a JS, le puedes decir que trabajarás en JS.

```
=== Functions Setup

A functions directory will be created in your project with a Node.js
package pre-configured. Functions can be deployed with firebase deploy.

? What language would you like to use to write Cloud Functions? (Use arrow keys)
> JavaScript
  TypeScript
```

En cuanto a ESLint le puedes decir que no, igual que a lo de instalar las dependencias con npm.

El directorio público, diremos que sea **dist/browser** (esto lo ajustaremos enseguida) y en cuanto a configurarlo como SPA, decimos que si.

```
? What do you want to use as your public directory? dist/browser
? Configure as a single-page app (rewrite all urls to /index.html)? Yes
```

Acabado el proceso, te habrá creado una carpeta **functions**, pero esta estructura te interesa poco. Es más. Te la vas a pasar por el forro 😊 Lo único que te interesa de aquí es el archivo **package.json**.

De momento has hecho lo fácil, pero falta conectar firebase functions con tu servidor express y a la vez darle acceso a los archivos que se subirán al hosting de firebase.

Te voy a explicar la forma en que, desde mi experiencia, considero más cómodo relacionar Firebase y Universal.

## 5 – Estructura de Firebase functions para Universal

Ahora mismo tu código compilado se reparte entre las carpetas **dist** y **dist-server**. Y además, Firebase te ha añadido la carpeta **functions** para meter tus *cloud functions* (es decir, el servidor Express) y sus dependencias.

Es decir, tu estructura de *builds* sería así:

- **dist**: App Angular y archivos estáticos
- **dist-server**: App Universal
- **functions**: Servidor Express y sus dependencias

El problema es que tu servidor Express que se ejecutará desde Firebase functions

Te propongo reemplazar esa estructura por la siguiente:

- **dist**: Servidor Express y sus dependencias
  - **browser**: App Angular y archivos estáticos
  - **server**: App Universal

Para hacerlo...

5.1 – Borra las carpetas **dist** y **dist-server**

5.2 – Ahora, crea una nueva carpeta **dist**, vacía.

## 6 – package.json de Firebase functions

El archivo **package.json** que ha creado Firebase en la carpeta **functions** es imprescindible para que Firebase entienda las dependencias externas de sus *cloud functions*.

Cuando haces un deploy de Firebase Functions, los propios servidores de Firebase realizan un **npm install** para descargar las dependencias que hay definidas en su archivo **package.json**.

6.1 – El primer paso será, por tanto, mover este archivo de **functions/package.json** a **dist/package.json**.

6.2 – ¡Ojo! El **.gitignore** que genera la CLI está preparado para ignorar todo lo que hay dentro de **dist**. No obstante, sería sensato añadir este archivo al control de versiones. Lo puedes añadir de forma manual con el siguiente comando:

```
git add dist/package.json -f
```

6.3 – Elimina la carpeta **functions**

Actualiza el archivo `firebase.json` del siguiente modo:

```
{
  "hosting": {
    "public": "dist/browser",
    "ignore": [
      "firebase.json",
      "**/.*",
      "**/node_modules/**"
    ],
    "rewrites": [
      {
        "source": "**",
        "function": "ssr"
      }
    ]
  },
  "functions": {
    "source": "dist"
  }
}
```

7.1 – Empiezo por la parte fácil, la propiedad `functions` . Como he renombrado el directorio con las *cloud functions*, necesito indicarle a firebase donde encontrarlas. Eso es lo que hago con la propiedad `source` . Cuando firebase haga un *deploy* de las funciones, subirá la carpeta `dist` entera.

7.2 – En cuanto al hosting, ahora los archivos estáticos se encuentran en `dist/browser` . Pero hay más...

Cuando se visita la página quiero que sea el servidor express el que responda. Por eso utilizo el `rewrites` de `hosting` . Con `**` le estoy indicando que cualquier ruta de mi página se gestione llamando a la función `ssr` . Ese es el nombre con el que exportaré la función de firebase asociada al servidor express.

**Importante:** La regla `rewrites` solo se aplica cuando **no existe un archivo** en el hosting de firebase para la ruta identificada.

Tienes que tener cuidado con esto. Cuando se visita la raíz de la página **no quiero devolver el index.html estático que existe en dist/browser** (y que es un esqueleto vacío), sino que quiero llamar al servidor express para que renderize la página de verdad.

Para solucionarlo, renombraré el archivo `index.html` que se genera durante el `npm run build:universal`. Así firebase no lo encontrará y derivará a `ssr` las peticiones al dominio raíz.

## 8 – Renombrando index.html a index-1.html

8.1 – Para renombrar el `index.html` que se genera en `dist/browser`, necesitas renombrar el original que se encuentra en `src`. Así que **renombrar src/index.html a src/index-1.html**.

8.2 – Hay una línea del servidor express, en la que le dices que utilice `index.html` como punto de entrada del renderizado. Como ahora el archivo es `index-1.html`, tendrás que actualizar también ese fragmento de código de `server/index.ts`:

```
// server/index.ts

// ...some stuff...
// ALL regular routes use the Universal engine
app.get('*', (req, res) => {
  res.render(join(DIST_FOLDER, 'browser', 'index-1.html'), {req});
});
// ...more stuff...
```

8.3 – Falta un último paso para generar este archivo durante el `build`. Tienes que actualizar también `.angular-cli.json` para que Angular sepa encontrar `index-1.html` y moverlo a la carpeta `dist/browser`.

## 9 – Actualiza .angular-cli.json para usar la nueva estructura

9.1 – Abre `.angular-cli.json` y actualiza la propiedad `index`, así como el directorio de salida `outDir` de ambas apps tanto cliente como Universal para



```
// .angular-cli.json
{
  //...some stuff...
  "apps": [
    {
      "root": "src",
      "outDir": "dist/browser",
      "assets": [
        "assets",
        "favicon.ico"
      ],
      "index": "index-1.html",
      "main": "main.ts",
      //...more stuff...
    },
    {
      "root": "src",
      "outDir": "dist/server",
      "assets": [
        "assets",
        "favicon.ico"
      ],
      "index": "index-1.html",
      "main": "main.server.ts",
      //...more stuff....
    }
  ],
  //...more stuff...
}
```

9.2 – Edita **server/tsconfig.ssr.json** para actualizar también el directorio de salida del servidor Express compilado.

```
// server/tsconfig.ssr.json

{
  "extends": "../tsconfig.json",
  "compilerOptions": {
    "outDir": "../dist",
    //...more stuff...
  }
}
```

## 10 – Actualiza el servidor para usar firebase

## 10.1 – Importar firebase:

```
// server/index.ts

import 'zone.js/dist/zone-node';
import 'reflect-metadata';
import { enableProdMode } from '@angular/core';
import * as express from 'express';
import { join } from 'path';

//firebase cloud functions
import * as firebaseFunctions from 'firebase-functions';

//...more stuff...
```

## 10.2 – Localizar la app Universal en el directorio correcto (ahora **dist/server**):

```
// server/index.ts

//...previous stuff...

// NOTE: Leave this as require() since this file is built Dynamically from
webpack
const { AppServerModuleNgFactory, LAZY_MODULE_MAP } =
require('./server/main.bundle');

//...more stuff...
```

## 10.3 – Evaluar si se está utilizando firebase cloud functions (esto vendrá determinado por una variable de entorno que crearemos en breve):

```
// server/index.ts

//...previous imports...

//check if Firebase functions is enabled or not
const DISABLE_FIREBASE = process.env.DISABLE_FIREBASE || false;

//... some more stuff...
```

## 10.4 – Actualizar el **DIST FOLDER** , cuyo valor dependerá de si se usa express con

```
// server/index.ts

//...previous stuff...
// Express server
const app = express();
const PORT = process.env.PORT || 4000;
const DIST_FOLDER = join(process.cwd(), DISABLE_FIREBASE ? 'dist' :
'./');

//... some more stuff...
```

10.5 – Actualizar los paths a la nueva estructura. Ahora los archivos están dentro de **dist/browser** (reflejado en el código del siguiente punto).

10.6 – **Los archivos estáticos ahora los sirve directamente firebase hosting.** El código para servir datos estáticos desde express solo lo necesitas si **DISABLE\_FIREBASE** es **true** . Lo mismo pasa con el código que lanza el servidor express ( **app.listen** ): Ahora es firebase quien se encarga de lanzarlo.

```
// server/index.ts

//...previous stuff...
app.set('view engine', 'html');
app.set('views', join(DIST_FOLDER, 'browser'));

if(DISABLE_FIREBASE){
  // Server static files using the express server only if not using firebase
  hosting
  app.get('*..*', express.static(join(DIST_FOLDER, 'browser')));
}

// ALL regular routes use the Universal engine
app.get('*', (req, res) => {
  res.render(join(DIST_FOLDER, 'browser', 'index.html'), {req});
});

if(DISABLE_FIREBASE){
  // Start up the Node server if not using firebase cloud functions
  app.listen(PORT, () => {
    console.log(`Node server listening on http://localhost:${PORT}`);
  });
}

//... some more stuff...
```

10.7 – Exportar el servidor express como función de firebase. Recuerda darle el nombre de `ssr` a la variable que exportas.

```
// server/index.ts

//...previous stuff...

//server side rendering using frebase cloud functions
export let ssr = DISABLE_FIREBASE ? null :
firebaseFunctions.https.onRequest(app);
```

## 11 – Actualiza el comando `serve:universal`

Como ves, ahora el servidor espera una variable de entorno `DISABLE_FIREBASE` (por defecto a `false`), para saber como comportarse.

Es interesante que actualices tu comando `serve:universal` para poner esta variable a `true` y así poder ejecutar el servidor sin necesidad de *Firebase functions*.

Recuerda que ahora tu servidor Express compilado está en `dist/index.js`.

Actualiza el archivo `package.json` del directorio raíz, así:

```
{
  //...some stuff...
  "scripts": {
    //...more stuff...
    "serve:universal": "DISABLE_FIREBASE=true node dist/index.js"
  },
  //...more stuff...
```

Esto puede agilizar tus comprobaciones de Universal en local. Además, con esta variable de entorno podrías hacer deploy en un entorno diferente a Firebase sin necesidad de modificar `index.ts`.

## 12 – Instalar las dependencias de firebase en tu proyecto

Aprende a hacer apps móviles con **ionic 3** [Ver curso](#)

Verás que ahora el archivo `index.ts` te da un error. Claro, no reconoce el paquete `firebase-functions`. Tienes que instalarlo en la raíz de tu proyecto (no en la carpeta `dist`).

Además, también necesitas añadir la dependencia `firebase-admin`. En terminal, desde la raíz del proyecto, puedes añadirlo todo con este comando:

```
npm install firebase-functions firebase-admin --save
```

## 13 – Comprueba que todo funciona

De momento sin firebase functions, comprueba que todo sigue funcionando correctamente.

Compila todo el código con:

```
npm run build:universal
```

y ejecútalo con:

```
npm run serve:universal
```

Si has seguido todos los pasos correctamente, deberías ver la aplicación correctamente. Eso es que no has roto nada ¡felicidades!

Ahora vamos a ver si también te funciona con Firebase en local.


Ejecuta

```
firebase serve
```

Esto debería lanzarte un servidor en local asociado a firebase hosting (normalmente en `http://localhost:5000`) y un **endpoint** local asociado a tu *firebase function*.

Si navegas a la URL del hosting, deberías ver correctamente tu página web. Además, por terminal deberías ver que se está llamando a la función `ssr`.

Para garantizar que todo va bien, deberías comprobar que el código fuente que has



```

1 <!DOCTYPE html><html lang="en"><head><meta charset="utf-8">
2 <title>MyProject</title><base href="/"><meta name="viewport"
3 content="width=device-width,initial-scale=1"><link rel="icon" type="image/x-
4 icon" href="favicon.ico"><link href="styles.ac89bfdd6de82636b768.bundle.css"
5 rel="stylesheet"><style ng-transition="serverApp"></style></head><body><app-
6 root _ngcontent-c0="" ng-version="5.2.9">
7 <div _ngcontent-c0="" style="text-align:center">
8 <h1 _ngcontent-c0="">
9 Welcome to app!
10 </h1>
11 Here are some links to help you start: </h2>
15 <ul _ngcontent-c0="">
16 <li _ngcontent-c0=""><a _ngcontent-c0=""
17 href="https://angular.io/tutorial" rel="noopener" target="_blank">Tour of
18 Heroes</a></li>
19 <li _ngcontent-c0=""><a _ngcontent-c0=""
20 href="https://github.com/angular/angular-cli/wiki" rel="noopener"
21 target="_blank">CLI Documentation</a></li>
22 <li _ngcontent-c0=""><a _ngcontent-c0="" href="https://blog.angular.io/"
23 rel="noopener" target="_blank">Angular blog</a></li>
24 </ul>
25 </app-root><script type="text/javascript"
26 src="inline.318b50c57b6eba3d437b.bundle.js"></script><script
27 type="text/javascript" src="polyfills.95f523ba519ac2af484d.bundle.js">
28 </script><script type="text/javascript"
29 src="main.5691baf8c2d6e0823e3.bundle.js"></script></body></html>

```

Si todo te va bien, ha llegado el momento de subirlo a Firebase.

## 14 – Añade las dependencias de Universal al package.json de Firebase

Has metido el **package.json** de firebase en la carpeta **dist**. Este archivo contiene las dependencias que necesita Firebase functions para funcionar. En este caso, claro, hay que añadirle también las dependencias del proyecto Universal.

Las dependencias del archivo **dist/package.json** deberían quedar así:

```

"dependencies": {
  "@angular/animations": "^5.2.6",
  "@angular/common": "^5.2.6",
  "@angular/compiler": "^5.2.6",
  "@angular/core": "^5.2.6",

```

```
"@angular/platform-browser-dynamic": "^5.2.6",
"@angular/platform-server": "^5.2.6",
"@angular/router": "^5.2.6",
"@nguniversal/express-engine": "^5.0.0-beta.6",
"@nguniversal/module-map-ngfactory-loader": "^5.0.0-beta.6",
"express": "^4.16.2",
"firebase-admin": "~5.9.0",
"firebase-functions": "^0.8.1",
"rxjs": "^5.5.6",
"zone.js": "^0.8.20"
},
```

**IMPORTANTE:** Si el `package.json` de tu firebase tiene la línea `"main": "lib/index.js",` , debes eliminarla.

x

## 15 – Deploy a Firebase

Subir tus archivos de hosting y firebase functions es tan fácil como escribir un comando en terminal.

Ejecuta:

```
firebase deploy
```

¡y listo!

Tardará un poco, porque tiene que subir varios archivos al hosting de Firebase, pero si todo va bien, acabarás recibiendo un mensaje de éxito. Además, te facilitará el enlace a la web con tu app. Ahora solo te falta comprobar que realmente firebase entrega bien tu app y que no se está generando desde cliente, sino que se renderiza por completo en servidor.

¿Quieres acceder al repositorio con el resultado final? Lo tienes a continuación:

**Este contenido es solo para suscriptores**

Aprende a hacer apps móviles con **ionic 3** [Ver curso](#)

---

**suscribirme para desbloquear**

*¡Tu e-mail estará 100% libre de spam!*

Haciendo click en los botones, estás de acuerdo con [Términos de uso](#),  
[Política de Privacidad](#)

## Reflexiones personales

Me ha quedado un artículo largo así que me entra la duda ¿Has sido capaz de llegar hasta el final?

¿Si?

¡Pues no te cortes y escíbeme algo!

¡Me darás una alegría!

Volviendo a las reflexiones...

Ya te habrás dado cuenta de que soy muy fan de Angular Universal. Es el complemento que necesitas para darle SEO a tu web Angular y mejorar el tiempo de carga. Pero es que además con herramientas como Firebase no necesitas ni saber de *backend* para poder utilizarlo.

Y hablando de Firebase... si bien puede condicionarte a su entorno y no lo usaría para proyectos de clientes potentes, creo que tiene un potencial brutal para *startups*: Reduce costes y tiempos de desarrollo/mantenimiento de sistemas y merece la pena tenerlo muy presente a la hora de iniciar proyectos personales que requieren *backend*.

¿Y tu, qué opinas?

Si te ha gustado este artículo, compártelo 😊





---

**Relacionado**

¿Angular 2 o Angular 4? -  
Simplemente Angular

22 marzo, 2017  
En "Angular 2"

Angular 5 Universal: Guía rápida

13 diciembre, 2017  
En "Angular"

¿Qué es Angular Universal?

4 diciembre, 2017  
En "Angular"

---

Published in [Angular](#) [Angular 2](#)

Previous Post

[Decorador de Analytics en Angular / Ionic](#)

No Newer Posts

[Return to Blog](#)

## 7 Comments



[gabfiocchi](#) (@gabfiocchi).

Hola Enrique, de este modo funciona el SSR si entramos también por otra ruta? Por ejemplo si fuera a ser midominio.com/contacto o midominio.com/noticia/2 ? Saludos!

---

10 abril, 2018 | [Reply](#)



Enrique Oriol (author)

Si, por que estás derivando cualquier URL a la función (a menos, como te decía, que exista un archivo en el hosting que coincida con la ruta)

---

10 abril, 2018 | [Reply](#).



Rachid

Gran Artículo muy completo, lo que no estoy seguro si aplicaria a mi proyecto.

Ya que yo tengo el firebase functions con algunas funciones que consumo desde una aplicacion de C# en windows y la idea tambien ( a falta de implementar) desde una aplicacion en ionic. Estas funciones lo que haces es recupera datos de base de datos y borrar (simplificando). Lo que hacia es con la url que te genera el firebase function de cada funcion era llamarla desde mis aplicaciones en el momento que lo necesitase. Mi pregunta es si esta es la forma correcta o se podria implementar el angular universal para mi proyecto ionic?.

Muchas Gracias!

---

10 abril, 2018 | [Reply](#).



Enrique Oriol (author)

Bueno, por lo que explicas usas firebase functions como servicio de acceso a una BBDD, un poco al estilo API REST. Esto es plenamente compatible con usar SSR

Cuando usas Universal, la primera petición se renderiza en servidor (que a su vez podría llamar a tus otras funciones de firebase para recuperar datos), pero a partir de ahí se "recrea" la app angular en cliente. El resto de interacciones son en cliente (de la forma tradicional en Angular).

---

10 abril, 2018 | [Reply](#).



Frank Betancur

Hola Enrique,

primero que todo gracias por compartir el conocimiento...

segundo, tengo el siguiente error cuando hago el paso 10 npm run build:universal:

ERROR in src/server/index.ts(6,23): error TS2307: Cannot find module 'path'.

src/server/index.ts(9,56): error TS2304: Cannot find name 'require'.

src/server/index.ts(21,17): error TS2304: Cannot find name 'process'.

src/server/index.ts(22,29): error TS2304: Cannot find name 'process'.

muchas gracias por la ayuda!

saludos!

---

10 abril, 2018 | [Reply](#).



Frank Betancur

Hola de nuevo! va lo solucioné. había creado la carpeta server donde no era...

---

10 abril, 2018 | [Reply](#).



Enrique Oriol (**author**)

Te me has adelantado. 😊

---

10 abril, 2018

## Deja un comentario

Introduce aquí tu comentario...

[Author Theme](#) modified by Enrique Oriol

Enrique Oriol

Aprende a hacer apps móviles con **ionic 3** [Ver curso](#)