



Parte 2

Parte 2

Angular 5

Andrei García Cuadra

Contenido

1. Compilando el proyecto para móviles (IOS + Android)	3
1.1 Anexo opcional: Compilación y creación manual del proyecto	3
2. Módulos extra	6
2.1. Notificaciones push	6
2.2. Barra de carga.....	7
3. Login y muestreo de datos con angular	8
3.1. Preparando el proyecto	8
3.2. Conexión	10
Entendiendo la conexión	10
Creando la conexión	10
Cerrando la conexión.....	12
3.3. Mostrando tabla con usuarios (a los conectados).....	12
4. Referencias de interés	14
4.1. Diseño	14
4.2. Funciones útiles	14
4.3. Mejoras del código	14

1. Compilando el proyecto para móviles (IOS + Android)

NOTA: Las cajitas muestran comandos del CMD (la mayoría de ellos requieren que el directorio actual sea el del proyecto), el resto está indicado.

Para esta parte del tutorial se usará la carpeta my-app-apk.

Para compilar el proyecto para dichas plataformas, hemos de cambiar nuestro compilador. Si mal no recordamos, hemos estado usando Angular CLI para la compilación de nuestro proyecto web, así que simplemente agregaremos este nuevo compilador.

NOTA: La compilación base no resultará en un archivo ejecutable, para ello hay que recompilar el proyecto con el intérprete oficial (p.e. android studio). Válido solo para angular 4 o superior.

En su lugar, nosotros simplemente cogeremos el proyecto inicial, así evitaremos conflictos de archivos.

Para que nos funcione bien este proyecto inicial, debemos tener instaladas ciertas dependencias y tenerlas agregadas al path para que nos funcionen correctamente los comandos.

1.1 Anexo opcional: Compilación y creación manual del proyecto

Se usará el compilador Ionic, el cual permite compilar el código fuente para Android, IOS y Windows Phone en todas sus versiones.

Cabe destacar que **se ha de crear un proyecto nuevo** para esto, ya que cambia toda la estructura básica del proyecto, **aunque se puede mantener todo el código fuente de Angular** copiando la carpeta src de nuestro antiguo proyecto al actual. También puede ser necesario cambiar las dependencias del package.json para añadir las requeridas por nuestro proyecto y que no se encuentren en el nuevo.

Para cada compilación se han de agregar nuevas opciones a nuestro path, nuevos programas específicos para cada plataforma y además el compilador base de Ionic.

Instalación de Ionic (Requerido)

REQUERIDO TERMINAL CMD CON PERMISOS DE ADMINISTRADOR

Agregar a las variables de entorno:

JAVA_HOME -> La ruta de nuestro JDK de Java.

ANDROID_HOME -> C:\Users\DAW\AppData\Local\Android\Sdk

PATH -> C:\Users\%USERNAME%\AppData\Roaming

C:\Program Files\Android\Android Studio\tools

C:\Program Files\Android\Android Studio\tools\bin

```
npm install -g ionic@latest
npm install -g cordova
ionic cordova rm android
ionic cordova add android
ionic cordova rm ios
ionic cordova add ios
sdkmanager "build-tools;26.0.0"
```

Creación del proyecto con Ionic (Requerido)

```
ionic start new-ionic3-angular4 tabs
```

Pruebas del proyecto en dispositivos móviles (Opcional)

```
ionic serve -l
```

Compilar para Android (Opcional)

Primero descargamos el android studio para que nos instale las dependencias necesarias. [Descargar](#). (Nota: va a tardar un rato largo). También hay que [descargar el SDK](#) y añadirlo a la carpeta donde instalamos android studio.

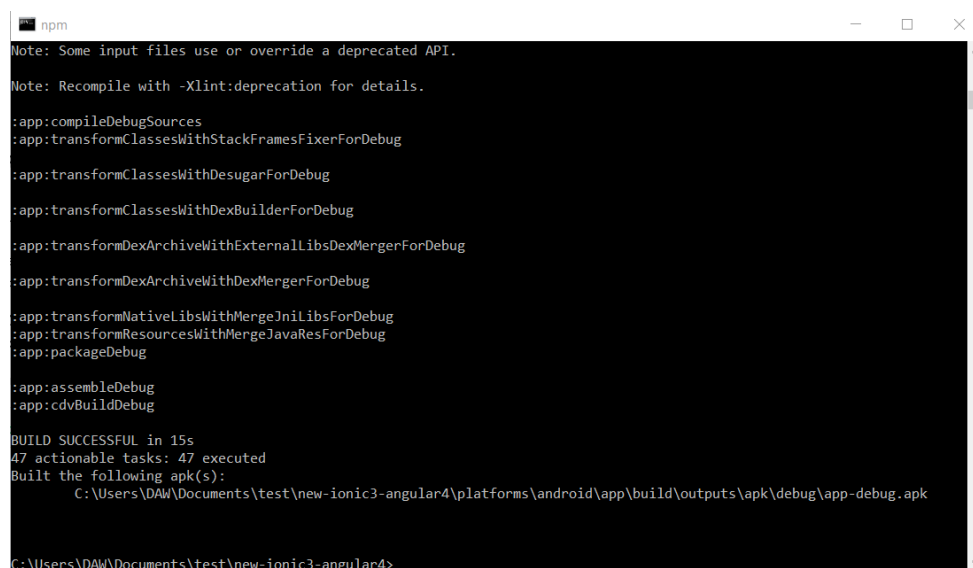
Tras la instalación debemos ejecutar el programa por primera vez y seguir el asistente para la puesta a punto. Cuando nos salga una lista de opciones (tras clicar finish), ya estará instalado.

En este menú, debemos clicar en **Configure > Settings > System Settings > Android SDK** y marcamos las opciones **Android 7** y **Android 8**.

Para compilar el APK resultante, usamos el comando: (O Compilar APK en npm scripts en netbeans)

```
ionic cordova build android
```

Este comando también nos dirá dónde se encuentra nuestro APK.



```
npm
Note: Some input files use or override a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

:app:compileDebugSources
:app:transformClassesWithStackFramesFixerForDebug
:app:transformClassesWithDesugarForDebug
:app:transformClassesWithDexBuilderForDebug
:app:transformDexArchiveWithExternalLibsDexMergerForDebug
:app:transformDexArchiveWithDexMergerForDebug
:app:transformNativeLibsWithMergeJniLibsForDebug
:app:transformResourcesWithMergeJavaResForDebug
:app:packageDebug
:app:assembleDebug
:app:cdvBuildDebug

BUILD SUCCESSFUL in 15s
47 actionable tasks: 47 executed
Built the following apk(s):
  C:\Users\DAW\Documents\test\new-ionic3-angular4\platforms\android\app\build\outputs\apk\debug\app-debug.apk

C:\Users\DAW\Documents\test\new-ionic3-angular4>
```

Compilar para IOS

```
ionic cordova build ios
```

Para generar un archivo .ipa (aplicación de iPhone) debemos seguir los siguientes pasos (es necesario tener dependencias de apple) podemos ver como [aquí](#) (no cubierto por el tutorial).

El procedimiento sería similar al de Android, pero con las dependencias de Apple.

2. Módulos extra

Las instalaciones serán semi guiadas, ya que el usuario debe saber dónde se ejecutan estos comandos.

2.1. Notificaciones push

Este tipo de notificaciones son frecuentes en redes sociales. Se llaman así debido a que no alteran la página, sino que simplemente se visualiza en un lugar concreto de la pantalla (normalmente esquinas) y se elimina tras un periodo de tiempo. [Link de la documentación oficial.](#)

```
npm install --save angular2-notifications
```

Añadir a **app.module.ts**:

```
import { SimpleNotificationsModule } from 'angular2-notifications';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
```

En su sección **imports**:

```
BrowserAnimationsModule,
SimpleNotificationsModule.forRoot()
```

Añadir a **menu.component.html**:

```
<simple-notifications [options]="options"></simple-notifications>
```

Añadir a **menu.component.ts**:

```
public options = {
  position: ["bottom", "left"],
  timeOut: 5000,
  lastOnBottom: true
}
```

Añadir al constructor de **app.component.ts**:

```
private notificar: NotificationsService
```

Añadir al evento OnInit de **app.component.ts**:

```
this.notificar.success(
  'Mi primera notificación',
  'WOLAAAAAAAAAAAA',
  {
    timeOut: 5000,
    showProgressBar: true,
    pauseOnHover: false,
    clickToClose: false,
    maxLength: 10
  }
)
```

2.2. Barra de carga

Con esta barra indicaremos al cliente cuando falta para que cargue su página cuando éste navegue por la página... Aunque tiene otros usos complementarios, por ejemplo, si estamos cargando los datos de la tabla de usuarios desde un api rest, podemos indicar qué porcentaje de carga llevamos. [Link documentación oficial.](#)

Instalar desde cmd:

```
npm install @ngx-loading-bar/router --save
```

Añadir a **app.module.ts**:

```
import { LoadingBarRouterModule } from '@ngx-loading-bar/router';
```

Y añadir el import para el mismo archivo:

```
LoadingBarRouterModule
```

Añadir a **menu.component.html**:

```
<ngx-loading-bar></ngx-loading-bar>
```

3. Login y muestreo de datos con angular

Agradecimientos a la página <https://reqres.in/> por su servicio gratuito de un api rest sin políticas de cors y online. Cabe destacar que no se va a usar encriptación, ni claves, ni seguridad en general. Para un entorno más seguro y no tener la seguridad tan baja, conviene usar JWT y demás. [Información](#).

3.1. Preparando el proyecto

Procedamos a crear un nuevo componente en el cual vamos a ver (con una tabla no tan cutre) un listado de usuarios que recogeremos de un api rest.

Generamos un nuevo componente llamado usuarios y otro llamado login.

```
ng generate component usuarios
ng generate component login
```

Ahora procedemos a generar los servicios de autenticación. Éstos se encargarán de permitir (o no) el acceso a una determinada página, en este caso en función de si el usuario está conectado o no.

Copiar a nuevo archivo **auth.service.ts**: (Se encarga de comprobar que esta conectado).

```
import {Injectable} from '@angular/core';
import {Router, CanActivate} from '@angular/router';
@Injectable()
export class AuthGuardService implements CanActivate {
  constructor(public router: Router) {}
  canActivate(): boolean {
    return this.isUserLoggedIn();
  }
  isUserLoggedIn() {
    {
      const token = localStorage.getItem('token');
      console.log(token);
      return token != null && token != "";
    }
  }
}
```

Copiar a nuevo archivo **noauth.service.ts**: (Se encarga de comprobar que no está conectado).

```
import {Injectable} from '@angular/core';
import {Router, CanActivate} from '@angular/router';
@Injectable()
export class NotAuthGuardService implements CanActivate {
  constructor(public router: Router) {}
  canActivate(): boolean {
    return !this.isUserLoggedIn();
  }
  isUserLoggedIn() {
    {
      const token = localStorage.getItem('token');
      return token != null && token != "";
    }
  }
}
```


Tras esto lo agregaremos a las rutas y al menú. Éstas rutas variarán en función de si el usuario está conectado o no, para ello le agregamos el evento **canActivate**, que se genera cada vez que se intenta acceder a cierta ruta (antes de que se procese) y se repite mientras esa página siga activa. Agregar en las rutas de `app.module.ts`:

```
{
  path: 'usuarios',
  component: UsuariosComponent,
  canActivate: [AuthGuardService]
},
{
  path: 'login',
  component: LoginComponent,
  canActivate: [NotAuthGuardService]
},
{
  path: 'logout',
  component: LogoutComponent,
  canActivate: [AuthGuardService]
},
},
```

Debemos agregar a `menu.component.ts` una dependencia a la que llamaremos `loggedIn`. Se encargará de detectar si está conectado (o no) usando el propio servicio de `AuthGuard`:

```
import {AuthGuardService} from '../auth.service';
constructor(private loggedIn:AuthGuardService) {}
```

Y ahora el link a **menu.component.html**: (Ojo a los **ngIf**, éstos se encargarán de que se muestren o no los enlaces en el menú, pero no tienen nada que ver con el **canActivate**). Nótese que `loggedIn` se refiere a el nombre que le hemos otorgado en el constructor.

```
<a routerLink="/usuarios" *ngIf="loggedIn.isUserLoggedIn()">Usuarios</a>
<a routerLink="/login" *ngIf="!loggedIn.isUserLoggedIn()">Conectar</a>
<a routerLink="/logout" *ngIf="loggedIn.isUserLoggedIn()">Desconectar</a>
```

3.2. Conexión

Entendiendo la conexión

Para entender cómo funciona el login, primero debemos tener una noción de que angular no tiene base de datos, angular no accede a la base de datos, angular no es un lenguaje de servidor...

Entonces, ¿Cómo acceso a mis usuarios de la base de datos? Con un api rest.

Un api rest no es más que una fuente de datos. Desde una url, recibimos un texto.

Para complicar un poco más este concepto, ese texto puede ser html, json, xml, texto simple... Pero nosotros utilizaremos Json, para poder enviar y recibir objetos satisfactoriamente.

El api rest se ubica en el servidor y es el que, propiamente, accede a la base de datos. El resultado de esta operación será mostrado al entrar a una determinada url, en este caso, en formato Json.

Por ello se sigue este orden: Angular login > enviar credenciales (POST) > api rest – servidor > base de datos < api rest – servidor < json < parseo del json (convertirlo de texto a objeto) < mostrar lo correspondiente en pantalla.

Si todo es satisfactorio, nos devolverá a su vez un código de sesión (token) con el cual estaremos conectados durante un determinado periodo de tiempo y bajo un determinado usuario (en nuestro ejemplo no, ya que no hemos implementado seguridad y será todo genérico).

Para nuestro ejemplo vale cualquier usuario para conectarse.

Creando la conexión

Primero importamos las dependencias de formularios al núcleo app.module.ts:

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

Y a sus imports:

```
FormsModule
```

Ahora se ha de crear el formulario, para ello reemplazamos el contenido de **login.component.html**:

```
<form #UserLogin="ngForm" (ngSubmit)="onSubmit(UserLogin)">
  <div class="form-group">
    <label>Email del usuario</label>
    <input name="email" type="email" class="form-control" required ngModel>
  </div>

  <div class="form-group">
    <label>Contraseña</label>
    <input name="password" type="password" class="form-control" ngModel>
  </div>

  <button type="submit" class="btn btn-success">Submit</button>
</form>
```

Cabe destacar de lo anterior las palabras `ngForm` y `ngModel`. Significan, respectivamente, que se trata de un formulario o de un input. Asimismo, la palabra `#userLogin` simplemente le da un ID al formulario que será usado por el propio Angular.

Ahora, modificaremos el archivo **login.component.ts** hasta dejarlo igual que el siguiente:

```
import {Component, OnInit} from '@angular/core';
import {FormGroup, FormControl, NgForm} from '@angular/forms';
import {HttpClient} from '@angular/common/http';
import {Router} from '@angular/router';
import {NotificationsService} from 'angular2-notifications';

@Component({
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {

  formulario: FormGroup = new FormGroup({
    email: new FormControl(),
    password: new FormControl()
  });

  constructor(private http: HttpClient,
    private notificar: NotificationsService,
    private router: Router) {}

  ngOnInit() {
  }

  onSubmit(userLogin: NgForm) {
    this.http.post('https://reqres.in/api/login', userLogin.value).subscribe(resp =>
    {
      this.notificar.success(
        'Conectado satisfactoriamente',
        'Muy bien muy bieeeeeeen',
        {
          timeout: 5000,
          showProgressBar: true,
          pauseOnHover: false,
          clickToClose: false,
          maxLength: 10
        }
      );
      localStorage.setItem("token", resp["token"]);
      this.router.navigateByUrl("usuarios");
    },
    err => {
      this.notificar.error(
        'No se ha podido conectar',
        'Muy mal muy maaaaaaaaaaaaaaaaaaaaa1',
        {
          timeout: 5000,
          showProgressBar: true,
          pauseOnHover: false,
          clickToClose: false,
          maxLength: 10
        }
      );
    }
  );
});
```

```
    }
  }
```

Muy simple, recoge los valores del formulario con **userLogin.value** cuando se pulsa en conectar. Si tiene algún valor en ambos campos, se mostrará una notificación de éxito y se guardará el usuario en sesión además de redirigir a usuarios. En caso contrario, si no se ha podido conectar, sólo se mostrará una notificación de error.

Cerrando la conexión

Modificamos el archivo **logout.component.ts** para que quede así:

```
import {Component, OnInit} from '@angular/core';
import {Router} from '@angular/router';
import {NotificationsService} from 'angular2-notifications';

@Component({
  template: ''
})
export class LogoutComponent implements OnInit {

  constructor(private notificar: NotificationsService,
    private router: Router) {}

  ngOnInit() {
    localStorage.clear();
    this.notificar.info(
      'Te has desconectado',
      '¡Se han quitado las opciones del menú en tiempo real! :o',
      {
        timeout: 5000,
        showProgressBar: true,
        pauseOnHover: false,
        clickToClose: false,
        maxLength: 10
      }
    );
    this.router.navigateByUrl("login");
  }
}
```

Es muy simple... Este código vacía las sesiones de localStorage y redirige al login cuando carga, sin más.

3.3. Mostrando tabla con usuarios (a los conectados)

Cuando el usuario se conecte, le vamos a poner una tabla algo más avanzada. Para ello lo instalamos con las siguientes comandos:

```
npm install @swimlane/ngx-datatable
```

Ahora añadimos a **app.module.ts** la base del módulo:

```
import { NgxDatatableModule } from '@swimlane/ngx-datatable';
```

Y a sus imports:

```
NgxDatatableModule @import '~@swimlane/ngx-datatable/release/index.css';
@import '~@swimlane/ngx-datatable/release/themes/material.css';
```

```
@import '~@swimlane/ngx-datatable/release/assets/icons.css';
```

Para que la tabla luzca su diseño material, agregamos estas líneas al principio de **assets/main.css**:

```
@import '~@swimlane/ngx-datatable/release/index.css';
@import '~@swimlane/ngx-datatable/release/themes/material.css';
@import '~@swimlane/ngx-datatable/release/assets/icons.css';
```

Ahora vamos al componente **usuarios.component.html** y agregamos la tabla reemplazando todo el contenido del archivo:

```
<ngx-datatable
  class="material"
  [rows]="rows"
  [columns]="columns"
  [columnMode]="'force'"
  [headerHeight]="50"
  [footerHeight]="50"
  [rowHeight]="'auto'"
  [limit]="2">
</ngx-datatable>
```

Y reemplazamos todo el contenido de **usuarios.component.ts** y agregamos el siguiente contenido:

```
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Component({
  selector: 'app-usuarios',
  templateUrl: './usuarios.component.html',
  styleUrls: ['./usuarios.component.css']
})
export class UsuariosComponent implements OnInit {

  rows:any = [];
  columns = [{name:'ID', prop:'id'},{name:'Nombre',prop:'first_name'},{name:'Apellidos',
prop:'last_name'}];
  constructor(private http: HttpClient) {

  }

  fetch(cb) {
    const req = new XMLHttpRequest();
    req.open('GET', 'https://reqres.in/api/users?page=2');

    req.onload = () => {
      cb(JSON.parse(req.response));
    };

    req.send();
  }
  ngOnInit() {
    this.http.get('https://reqres.in/api/users?page=2').subscribe(resp => {
      console.log(resp);
      this.rows = resp["data"];
    });
  }
}
```

```
}
```

4. Referencias de interés

Nuestra página es cutre, pero funcional. Aquí se indicarán ciertas categorías con complementos (módulos, servicios, etc...) para poder sacarle el máximo partido a angular de forma independiente:

Documentación oficial: <https://angular.io/guide/cheatsheet>

4.1. Diseño

Material: <https://material.angular.io/>

Bootstrap: <https://ng-bootstrap.github.io/#/home>

4.2. Funciones útiles

Traducción: <https://github.com/ngx-translate/core>

4.3. Mejoras del código

Buenas prácticas: <https://medium.com/@tomastrajan/6-best-practices-pro-tips-for-angular-cli-better-developer-experience-7b328bc9db81>