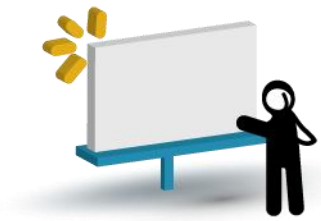
UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

Desarrollo con Angular

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Unidad 4: Routing con angular



Presentación:

En la presente unidad vemos cómo crear nuestros módulos propios, como armar nuestra aplicación web navegable utilizando el modulo de routing de angular. Por últimos finalizamos con los comandos necesarios para preparar nuestra versión productiva

Objetivos:

Que los participantes:

- Comprendan qué es el routing en una aplicación web.
- Aprendan cómo instalar el modulo de routing.
- Aplican el conocimiento adquirido del routing en la práctica.

Bloques temáticos:

- Módulos
- Crear una aplicación con módulo routing.
- Configurar rutas.
- Links en html.
- Rutas hijas.
- Local Storage y Session Storage.
- Subir a producción

Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

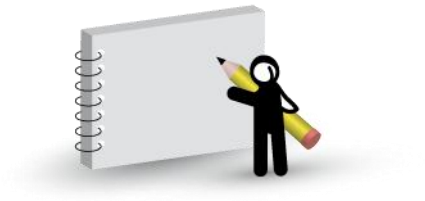
- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

**** El MEC es el modelo de E-learning colaborativo de nuestro Centro.***



Tomen nota:

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

Módulos

Un módulo es una reunión de componentes y otros artefactos como directivas o pipes.

Los módulos nos sirven principalmente para organizar el código de las aplicaciones

Angular y por tanto debemos aprender a utilizarlos bien.

Un módulo es uno de los elementos principales con los que podemos organizar el código de las aplicaciones en Angular.

En lugar de colocar el código de todos los componentes, directivas o pipes en el mismo módulo principal, lo adecuado es desarrollar diferentes módulos y agrupar distintos elementos en unos u otros. El orden se realizará de una manera lógica, atendiendo a nuestras propias preferencias, el modelo de negocio o las preferencias del equipo de desarrollo

Crear nuevos módulos

El comando para generar ese módulo nuevo es "generate" y a continuación tenemos que indicar qué es lo que se quiere generar, en este caso "module", acabando con el nombre del módulo a crear.

```
ng generate module nombre
```

Una vez lanzado este comando en nuestro proyecto, dentro de la carpeta "src/app" se crea un subdirectorio con el mismo nombre del módulo generado. Dentro encontraremos además el archivo con el código del módulo.

Ahora, si abrimos el código del módulo generado "nombre.module.ts", encontraremos cómo se define un módulo en Angular. La parte más importante es, como ya viene siendo habitual en Angular, un decorador.

El decorador de los módulos se llama @NgModule.

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';



@NgModule({
  declarations: [],
  imports: [
    CommonModule
  ]
})
export class NombreModule { }
```

En el decorador, tienes de momento un par de arrays definidos:

- imports: con los imports que este módulo necesita
- declarations: con los componentes, u otros artefactos que este module construye.

Generar componentes dentro del modulo

Ahora que tenemos nuestro primer módulo propio, vamos a agregar algo en él.

Básicamente comenzaremos por añadirle un componente, usando como siempre el

Angular CLI.

Hasta ahora todos los componentes que habíamos creado habían sido generados dentro del módulo principal, pero si queremos podemos especificar otro módulo donde crearlos, mediante el comando:

ng g component *nombre*/componente

Esto nos generará una carpeta dentro del módulo indicado, en la que colocará todos los archivos del componente recién creado. Es decir el css, html, ts y spec.ts

Pero además, el comando del CLI también modificará el código del módulo, agregando automáticamente el import del componente y su referencia en el array "declarations".

Ahora el código del módulo "nombremodulo.module.ts" tendrá una forma como esta:

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { ComponenteComponent } from './componente/componente.component';



@NgModule({
  declarations: [ComponenteComponent],
  imports: [
    CommonModule
  ]
})
export class NombreModule { }
```

Exportar el modulo

Más adelante, si queremos que este módulo exponga cosas hacia afuera, que se puedan llegar a utilizar desde otros módulos, tendremos que agregar una nueva información al decorador del módulo: el array de exports.

Vamos a suponer que el componente "ComponenteComponent" queremos que se pueda usar desde otros módulos. Entonces debemos señalar el nombre de la clase del componente en el array de "exports". Con ello el decorador del module quedaría de esta manera.

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { ComponenteComponent } from './componente/componente.component';



@NgModule({
  declarations: [ComponenteComponent],
  imports: [
    CommonModule
  ],
  exports:[ComponenteComponent]
})
export class NombreModule { }
```

Usar el componente desde otros módulos

El último punto que nos queda por ver es cómo usar el componente ComponenteComponent desde otros módulos. Para ello vamos a modificar manualmente el módulo principal de la aplicación, de modo que pueda conocer el componente definido en el módulo nuevo que hemos creado en este artículo.

Para importar el componente realmente lo que vamos a importar es el módulo entero donde se ha colocado, ya que el propio módulo hace la definición de aquello que se quiere exportar en "exports". Requiere varios pasos:

1. Hacer el import del modulo con la sentencia import de Javascript
2. Declarar el import en el decorador del module principal
3. Usar el componente en html

Hacer el import del modulo con la sentencia import de Javascript

Para que Javascript (o en este caso TypeScript) conozca el código del módulo, debemos importarlo primero. Esto no es algo de Angular, sino del propio lenguaje en particular.

```
import { NombreModule } from './nombre/nombre.module';
```

Declarar el import en el decorador del module principal

La importación de nuestro módulo se realiza en la declaración "imports" del módulo principal.

Este es el código del decorador del módulo principal.

```
import { NombreModule } from './nombre/nombre.module';

@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    LoginComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    NombreModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
```

Usar el componente en html

Finalmente ya solo nos queda usar el componente. Para ello vamos a colocar en el template del componente raíz el selector declarado en el componente creado.

Así que simplemente abrimos el archivo "app.component.ts" y colocamos la etiqueta de nuestro nuevo componente generado.

```
<app-componente></app-componente>
```

Crear una aplicación con módulo routing

Al crear nuestra aplicación debemos seleccionar “instalar el modulo de routing”

Nuestra aplicación con modulo de routing integrado creara tanto para la raíz como para las ramas funcionales se creará un fichero. En la raíz será llamado **app-routing.module.ts** con un contenido como este:

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

const routes: Routes = [];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Este módulo de un único fichero sirve para definir las rutas de otro módulo padre asociado, **app.module.ts**, el cual quedará más o menos así:

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Configurar rutas

En el archivo **app-routing.module.ts** debemos especificar las rutas de nuestra aplicación. Lo hacemos en la constante routes

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { LoginComponent } from './login/login.component';
import { ContactoComponent } from './contacto/contacto.component';

const routes: Routes = [
  {path:"", component:HomeComponent},
  {path:"login",component:LoginComponent},
  {path:"contacto",component:ContactoComponent}

];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

En el campo path colocamos la url y en component el componente que se llamara cuando se ingrese por esa URL.

En el archivo **app.module.ts** agregamos los componentes en declarations

```
@NgModule({
  declarations: [
    AppComponent,
    LoginComponent,
    ContactoComponent,
    HomeComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Links en html

Para hacer los encales en las paginas HTML, en lugar de utilizar el atributo href en un tag a debemos utilizar la directiva link de la siguiente manera

```
<!--menú de navegación, sin href-->
<nav>
  <a [routerLink]="['/']">Inicio</a>
  <a [routerLink]="['/contacto',42]">Contacto</a>
</nav>
<!--Este componente nativo hace que el enrutador cargue una página dinámicamente-->
<router-outlet></router-outlet>
```

Podemos probar poniendo el contenido anterior en el archivo **app.component.html**

Rutas hijas

Se define como ruta hija a aquellas subrutas, por ejemplo tenemos el componente o ruta principal contacto y las subrutas nuevo, lista

```
const routes: Routes = [
  {
    path: 'contacto',
    component: ContactoComponent,
    children: [ // rutas hijas, se verán dentro del componente padre
      {
        path: 'nuevo', // la ruta real es movimientos/nuevo
        component: ContactoComponent
      },
      {
        path: 'lista',
        component: ContactoComponent
      }
    ]
  },
  {
    path: 'contacto/:id', // parámetro variable id
    component: ContactoComponent
  }
];
```

En la URL un subruta seria por ejemplo **/contacto/nuevo**

También podemos ver en el ejemplo que él **:id** funciona como parámetro, es decir si accedo a la url **/contacto/15** la variable **:id** tomara el valor 15

El ejemplo para generar este último enlace será:

```
<!--menú de navegación, sin href-->
<nav>
  <a [routerLink]="['/']">Inicio</a>
  <a [routerLink]="['/contacto',42]">Contacto</a>
</nav>
<!--Este componente nativo hace que el enrutador cargue una página dinámicamente-->
<router-outlet></router-outlet>
```

Desde el controlado accedemos al parámetro **:id** de la siguiente manera

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';

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

  constructor(private route: ActivatedRoute) { }

  ngOnInit() {
      this.route.params
      .subscribe(params => {
        const _id = params['id'].toString();
        console.log("Id",_id);

      });
  }

}
```

Como vemos el modulo **ActivatedRoute** es el encargado de tomar estos valores.

Local Storage y Session Storage

Almacenamiento local en el navegador, dispone de los siguientes métodos:

```
//Guarda el valor en la clave especificada
localStorage.setItem(key:string,  value:string);
//Obtiene el valor dada una clave
localStorage.getItem(key:string);
//Elimina el valor de una clave
localStorage.removeItem(key:string);
//Limpia todo el almacenamiento
localStorage.clear();
```

Session Storage, mismo concepto que localstorage, pero los datos se pierden cuando se cierra la página:

```
//Guarda el valor en la clave especificada
sessionStorage.setItem(key:string,  value:string);
//Obtiene el valor dada una clave
sessionStorage.getItem(key:string);
//Elimina el valor de una clave
sessionStorage.removeItem(key:string);
//Limpia todo el almacenamiento
sessionStorage.clear();
```

- **setItem**: Permite almacenar un valor en la key especificada
- **getItem**: Lee el valor de la key solicitada
- **removeItem**: Elimina el valor de la key especificada
- **clear**: Limpia por completo la base de datos

En caso de querer almacenar un json debemos antes convertirlo a un string utilizando el método stringify, un ejemplo

```
localStorage.setItem('usuarios',  JSON.stringify(data));
let usr2 = localStorage.getItem('usuarios');
console.log(JSON.parse(usr2));
```

Subir a producción

Tenemos que generar una carpeta de distribución para poder subir a cualquier servidor y utilizar en Producción. Tenemos dos maneras de crear esa carpeta de distribución una es la carpeta final que subiremos a Producción que es la más comprimida y hay una instancia intermedia donde todavía puedo seguir obteniendo todavía mensajes sobre el código, una instancia con fines de desarrollo todavía, por decirlo de alguna forma. Puedo desplegar la aplicación pero si algo falla puedo obtener información sobre el fallo.

Para la instancia intermedia:

Utilizamos el comando

```
ng build
```

de esta forma se crea en nuestra carpeta del proyecto la carpeta dist con todos los archivos necesarios para ejecutar nuestra aplicación compactados.

Esa carpeta dist es la que tendríamos que subir a cualquier servidor para ejecutar nuestra aplicación pero todavía tenemos información sobre el código de la misma en caso de que algo falle.

Para Producción:

Para crear una carpeta de distribución para producción tenemos que hacer un cambio en nuestro código en archivo src/environments/environment.ts

```
export const environment = {

  production: false

};
```

Modificamos a production: true.

Y luego escribimos:

```
ng build --prod
```

Ahora si esto empezara a generar la carpeta dist con los archivos necesarios para subir nuestra aplicación a Producción.

Para poder probar lo generado dentro de la carpeta dist como si estuvieran en Producción podemos usar http-server. Lo encuentran aquí:
https://www.npmjs.com/package/httpserver

http-server es un servidor http simple, de configuración desde la línea de comandos. Es lo suficientemente potente para el uso de producción, pero es simple y fácil de usar para pruebas, desarrollo local y aprendizaje.

Lo instalan con:

```
npm install http-server -g
```

y luego se posicionan en la carpeta dist que les acaba de generar el ng build desde la consola y tipean

```
http-server -o
```

Eso le va a levantar el servidor y abrir la aplicación en el puerto 8080 en su navegador.

Bibliografía utilizada y sugerida

https://cli.angular.io/

https://www.typescriptlang.org/

http://victorroblesweb.es/tag/manual-de-angular-2-en-espanol/

https://app.desarrolloweb.com/manuales/manual-angular2

https://www.youtube.com/watch?v=H9Dtqy3Fd40

https://angular.io/guide/router

Lo que vimos:

En esta unidad hemos aprendido cómo generar nuestra aplicación web navegable y la utilización de Local Storage

Lo que viene:

La Evaluación Final Integradora Obligatoria del Módulo.