3-Data

Formularios, tablas y modelos de datos en Angular

1. Binding
2. Doble Binding
3. Estructuras
4. Modelo y controlador

1. Binding

Base

Directivas

Enlace del modelo hacia la vista

Enlace de la vista hacia el modelo

1.0 Base

Creamos una nueva ruta funcionalidad para la gestión de contactos. Requiere ruta, enlace, módulo y componente.

```
ng g m contacts --routing true
ng g c contacts/contacts
```

En `app-routing` y en `contacts-routing`:

```
  // app-routing
  {
    path: 'contacts',
    loadChildren: './contacts/contacts.module#ContactsModule'
  },
  // contacts-routing
  {
    path: '',
    component: ContactsComponent
  }
```

En `HeaderComponent`

```
<a routerLink="contacts" routerLinkActive="router-link-active" class="bu
  <span> Contacts</span>
</a>
```

1.1 Directivas

> Para empezar agregamos algunas propiedades. En `contacts.component.ts`:

```
public header = 'Contacts';
public description = 'Manage your contact list';
public numContacts = 0;
public counterClass = 'tag secondary';
public formHidden = false;
```

1.1.1 Enlace del modelo hacia la vista

En `contacts.component.html` mostramos cabeceras con estilo

```
<h2>{{ header }}</h2>
<p>{{ description | uppercase }}</p>
<p>
  You have <mark [class]="counterClass">{{ numContacts }}</mark> contact
  now.
</p>
```

1.1.2 Enlace de la vista hacia el modelo

En `contacts.component.html` también actuamos sobre la vista

```
<input
  value="Show Form"
  class="primary"
  type="button"
  (click)="formHidden=false"
/>
<input
  value="Hide Form"
  class="inverse"
  type="button"
  (click)="formHidden=true"
/>
<form [ngClass]="{'hidden':formHidden}">
  <fieldset><legend>Contact Form</legend></fieldset>
</form>
```

Recap:

1. Binding

Base

Directivas

Enlace del modelo hacia la vista

Enlace de la vista hacia el modelo

2. Doble Binding

NgModel

Form

Enlace del modelo hacia la vista

Enlace de la vista hacia el modelo

2.1 NgModel

La directiva *ngModel* viene en el `FormsModule`

Hay que importarlo antes de usarlo. Por ejemplo en `contacts.module.ts`

```
import { CommonModule } from '@angular/common';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { ContactsRoutingModule } from './contacts-routing.module';

@NgModule({
  imports: [
    CommonModule,
    ContactsRoutingModule,
    FormsModule
  ]
})
export class ContactsModule { }
```

Banana in a box [()]

Hace referencia al *paréntesis dentro del corchete*

```
[(ngModel)]="model.property"
```

Usa la comunicación en ambos sentidos

- **(banana)** : de la vista al modelo
- **[box]** : del modelo a la vista

> La directiva se asocia con una propiedad del controlador... o mejor aún, con una **propiedad del modelo** del controlador

Modelo

```
public contact = { name: '' };
```

Directiva

```
<section>
  <label for="name">Name</label>
  <input
    name="name"
    type="text"
    [(ngModel)]="contact.name"
    placeholder="Contact name"
  />
</section>
```

Espía

```
<pre>{{ contact | json }}</pre>
```

2.2 Form

Hay más usos de las directivas en los formularios

Por ejemplo, dado el siguiente modelo:

```
public contact = { name: '', isVIP: false, gender: '' };
```

2.2.1 CheckBox

```
<section>
  <label for="isVIP">Is V.I.P.</label>
  <input name="isVIP" type="checkbox" [(ngModel)]="contact.isVIP" />
</section>
```

2.2.2 Radio Buttons

```
<section>
  <label for="gender">Gender</label>
  <input name="gender" value="m" type="radio" [(ngModel)]="contact.gende
  <i>Male</i>
  <input name="gender" value="f" type="radio" [(ngModel)]="contact.gende
  <i>Female</i>
</section>
```

Recap:

2. Doble Binding

NgModel

Form

Enlace del modelo hacia la vista

Enlace de la vista hacia el modelo

3. Estructuras

***ngFor**

***ngIf**

3.1 *ngFor

Directiva estructural **repetitiva**

Dado el siguiente modelo:

```
public workStatuses = [
  { id: 0, description: 'unknow' },
  { id: 1, description: 'student' },
  { id: 2, description: 'unemployed' },
  { id: 3, description: 'employed' }
];
public contact = { name: '', isVIP: false, gender: '', workStatus: 0 };
```

*ngFor

```
<section>
  <label for="workStatus">Work Status</label>
  <select name="workStatus" [(ngModel)]="contact.workStatus">
    <option *ngFor="let status of workStatuses" [value]="status.id">
      <span>{{ status.description }}</span>
    </option>
  </select>
</section>
```

let **iterador** of **iterable**

uso avanzado de *trackBy*

3.2 *nglf

Directiva estructural **condicional**

Dado el siguiente modelo

```
public contact = {
  name: '',
  isVIP: false,
  gender: '',
  workStatus: '0',
  company: '',
  education: ''
};
```

*ngIf

```
<section *ngIf="contact.workStatus=='3'; else education">
  <label for="company">Company Name</label>
  <input name="company" type="text" [(ngModel)]="contact.company" />
</section>
<ng-template #education>
  <label for="education">Education</label>
  <input name="education" type="text" [(ngModel)]="contact.education" />
</ng-template>
```

if **condition** else **template**

también hay **ngSwitch*

Recap:

3. Estructuras

*ngFor

*ngIf

4. Modelo y controlador

Interfaces y modelos

ViewModel en el controlador

4.1 Interfaces y modelos

Mejor interface que clase

```
export interface Option {
  id: number;
  description: string;
}

export interface Contact {
  name: string;
  isVIP: boolean;
  gender: string;
  workStatus: number | string;
  company: string;
  education: string;
}
```

tipos compuestos `number | string`

Se usan para tipificar las propiedades

```
public workStatuses: Option[] = [
    { id: 0, description: 'unknow' },
    { id: 1, description: 'student' },
    { id: 2, description: 'unemployed' },
    { id: 3, description: 'employed' }
  ];
public contact: Contact = {
    name: '',
    isVIP: false,
    gender: '',
    workStatus: 0,
    company: '',
    education: ''
  };
public contacts: Contact[] = [];
```

4.2 ViewModel en el controlador

No solo propiedades, también métodos

```
public saveContact() {
  this.contacts.push({ ...this.contact });
  this.numContacts = this.contacts.length;
}
```

```
<input value="Save" type="submit" (click)="saveContact()" />
```

OnInit

```
public workStatuses: Option[];
public contact: Contact;
public contacts: Contact[];
constructor() {}
public ngOnInit() {
  this.workStatuses = [
    { id: 0, description: 'unknow' },
    { id: 1, description: 'student' },
    { id: 2, description: 'unemployed' },
    { id: 3, description: 'employed' }
  ];
  this.contact = {
    name: '',
    isVIP: false,
    gender: '',
    workStatus: 0,
    company: '',
    education: ''
  };
  this.contacts = [];
}
```

Repasamos

```
<ul *ngIf="contacts.length>0; else empty">
  <li *ngFor="let contact of contacts">
    <span>{{ contact.name }}</span>
    <input value="Delete" type="button" (click)="deleteContact(contact)'
  </li>
</ul>
<ng-template #empty> <i>No data yet</i> </ng-template>
```

```
public deleteContact(contact: Contact) {
  this.contacts = this.contacts.filter(c => c.name !== contact.name);
  this.numContacts = this.contacts.length;
}
```

4. Modelo y controlador

Interfaces y modelos

ViewModel en el controlador

Next:

Flujo de datos entre componentes Angular

Comunicación entre componentes

El patrón Contenedor / Presentadores

Comunicaciones entre páginas o estructuras

Blog de apoyo: Formularios, tablas y modelos de datos en Angular

By Alberto Basalo