ISPC
INSTITUTO SUPERIOR
POLITÉCNICO CÓRDOBA

Angular

~~Inte~~gración con MVC

ISPC
INSTITUTO SUPERIOR
POLITÉCNICO CÓRDOBA

Conceptos Previos

¿Qué es un servicio en Angular?

Se puede llegar a presentar la situación de que dos o más componentes hagan uso de los mismos datos y para poder ser utilizados son referenciados en cada uno de los componentes. Por ello, a fin de mantener la simplicidad Agular incorpora el concepto de **Servicios** permitiendo así, que estos datos sean referenciados por los componentes desde los servicios.

Los componentes delegan actividades a los servicios como ser: obtener datos necesarios, validar los mismos, registrar la información, etc.

Es decir que los servicios:

- Son proveedores de datos.
- Ayudan a mantener la lógica de acceso a los mismos.
- Proveen la operatoria del negocio.
- Manipulan de datos en la aplicación.
- Invocar a un servidor HTTP para consumir una API

¿Cómo crear servicios?

1. Ir a la consola DOS o “Símbolo del Sistema” del sistema operativo o Terminal de VSCode.
2. Ejecutar el comando: **ng generate service <<service-name>>**

o su abreviado: ng g s <<service-name>>

3. A continuación, AngularCLI generará un archivo **<<nombre>>.servicio.ts** el cual contendrá las siguiente líneas de código:

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class CuentaService {

  constructor() { }
}
```

El decorador **@Injectable()** define el servicio identificando la clase y su metadata que permita a Angular inyectarlo a un componente como dependencia.

4. Por último declarar en el archivo app.module.ts en la lista de proveedores:

```
providers: [CuentaService],
```

¿Cómo consumir servicios?

5 - Importar el servicio en el **componente** que lo consumirá:

```
import { CuentaService } from 'src/app/servicios/cuenta.service';
```

6- Inyectar el servicio en el **constructor** del componente:

```
constructor( cuentaService: CuentaService)
  { ...
  }
```

7- Consumir el servicio:

```
constructor( cuentaService: CuentaService)
 {
    this.movimientos=cuentaService.ObtenerUltimosMovimientos();
 }
```

¿Qué es la inyección de dependencias?

Los componentes consumen servicios; es decir, que podemos inyectar un servicio en un componente, dándole acceso al componente a ese servicio.

La Inyección de dependencias permite mantener las clases componentes ligeras y eficientes. No obtienen datos del servidor, validan la entrada del usuario o registra directamente en la consola; tales tareas son delegadas a los servicios.

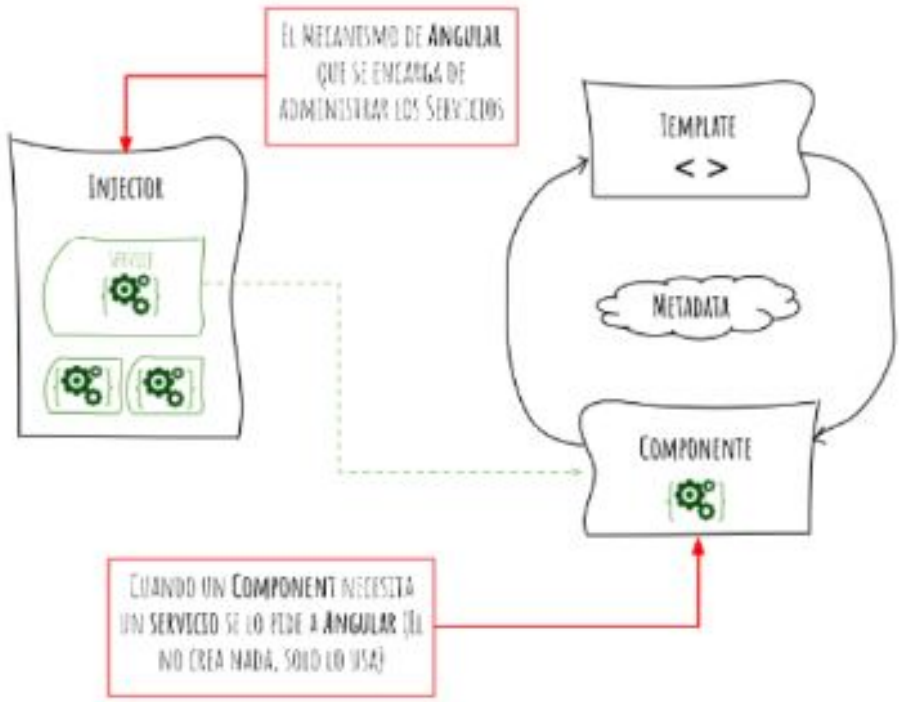
¿Cómo funciona?

Cuando Angular crea una nueva instancia de un componente, determina qué servicios u otras dependencias necesita ese componente al observar los tipos de parámetros del constructor.

Si Angular descubre que un componente depende de un servicio, primero verifica si el inyector tiene instancias existentes de ese servicio. Si una instancia de servicio solicitada aún no existe, el inyector crea una utilizando el proveedor registrado y la agrega al inyector antes de devolver el servicio a Angular.

Cuando todos los servicios solicitados se han resuelto y devuelto, Angular puede llamar al constructor del componente con esos servicios como argumentos.

Finalmente el componente puede hacer uso del mismo.



Fuente de la imagen:
:https://gustavodohara.com/blogangular/agregar-servicios-angular-no-componentes-modulos-la-vida/

ISPC
INSTITUTO SUPERIOR
POLITÉCNICO CÓRDOBA

JSON

JSON es un formato de datos basado en texto que sigue la sintaxis de objeto de JavaScript, popularizado por Douglas Crockford. Aunque es muy parecido a la sintaxis de objeto literal de JavaScript, puede ser utilizado independientemente de JavaScript, y muchos entornos de programación poseen la capacidad de leer (convertir; parsear) y generar JSON.

Los JSON son cadenas - útiles cuando se quiere transmitir datos a través de una red. Debe ser convertido a un objeto nativo de JavaScript cuando se requiera acceder a sus datos. Ésto no es un problema, dado que JavaScript posee un objeto global JSON que tiene los métodos disponibles para convertir entre ellos.

Estructura JSON

JSON es una cadena cuyo formato recuerda al de los objetos literales JavaScript. Es posible incluir los mismos tipos de datos básicos dentro de un JSON que en un objeto estándar de JavaScript - cadenas, números, arreglos, booleanos, y otros literales de objeto. Esto permite construir una jerarquía de datos, como ésta:


```
{"Id":2,
"Cvu":"000001",
"Saldo":1500.0,
"Id_persona":20,
"Estado":true,
"Movimientos":[
        {       "Id":0,
                "FechaHora":"2020-08-01T00:03:15",
                "Monto":1500.0,
                "CvuDestino":"0000",
                "CvyOrigen":"0000",
                "IdCuenta":2
                ,"TipoMovimiento":"Extracción"},
        {       "Id":0,
                "FechaHora":"2020-08-05T00:00:00",
                "Monto":2000.0,
                "CvuDestino":"0001","CvyOrigen":"0002"
                ,"IdCuenta":2
                "TipoMovimiento":"Transferencia"}
        ]}
```

¿Cómo acceder a los datos JSON?

Para acceder a los datos que se encuentran más abajo en la jerarquía, simplemente se debe concatenar los nombres de las propiedades y los índices de arreglo requeridos. Por ejemplo, para acceder al tercer superpoder del segundo héroe registrado en la lista de miembros, se debería hacer esto:

```
cuenta['Movimientos'][1]['TipoMovimiento']
```

1. Primero el nombre de la variable — cuenta.
2. Dentro de esta variable para acceder a la propiedad movimientos utilizamos ["Movimientos"].
3. movimientos contiene un arreglo poblado por objetos. Para acceder al segundo objeto dentro de este arreglo se utiliza [1].
4. Dentro de este objeto, para acceder a la propiedad tipo de movimiento utilizamos ["TipoMovimiento"].

Para más información consulta: https://developer.mozilla.org/es/docs/Learn/JavaScript/Objects/JSON

ISPC
INSTITUTO SUPERIOR
POLITÉCNICO CÓRDOBA

DEMO

ISPC
INSTITUTO SUPERIOR
POLITÉCNICO CÓRDOBA

MVC en Angular

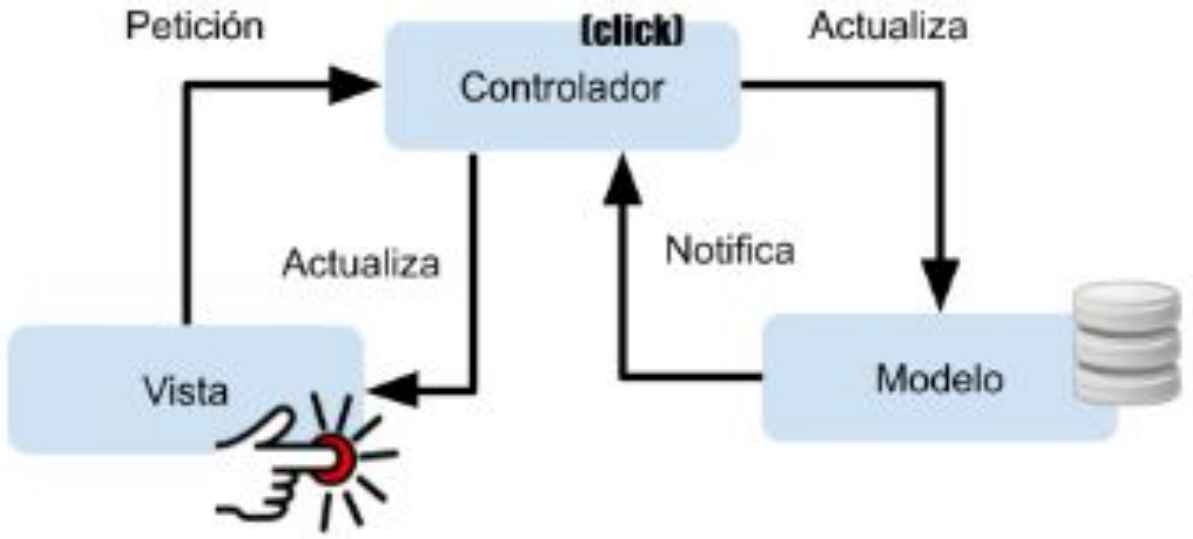
MVC en Angular

El modelo MVC (modelo-vista-controlador) es un patrón de arquitectura de software, que separa la parte visual, de la funcional y las estructuras de datos. Por ello, se compone de 3 componentes:

- el **modelo** (la parte de acceso a datos), permite acceder a los datos y a los mecanismos para manipularlos. Ej. a través de un servicio que se conecte con un API Rest o servicio web (backend).
- la **vista** (la parte visual), presenta el modelo en un formato adecuado para la interacción del usuario. Ej. formularios.
- el **controlador** (la parte funcional), controla las interacciones entre la vista y el modelo. Ej. reglas de negocio.

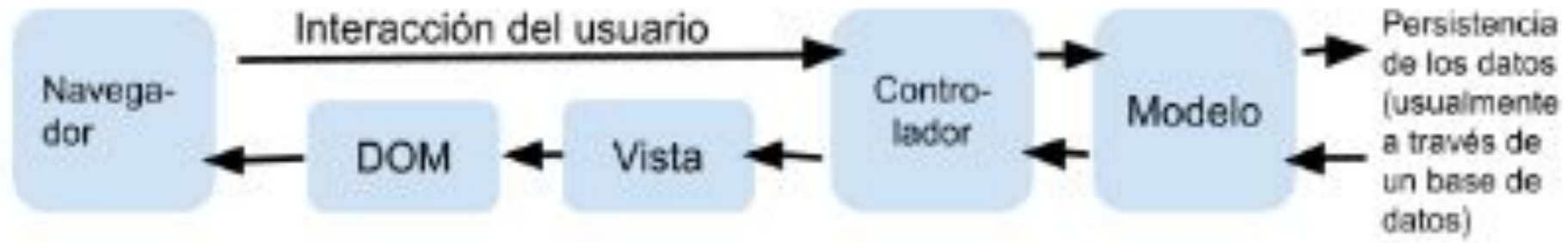
MVC en Angular

Para comprenderlo, supongamos que en la vista hacemos clic en un botón (html), a continuación el controlador (ts) que está escuchando eventos, detecta el clic del botón y le dice al modelo (service) que actualice los datos. El modelo notifica al controlador sobre dicha acción y este finalmente actualiza la vista.



MVC en Angular

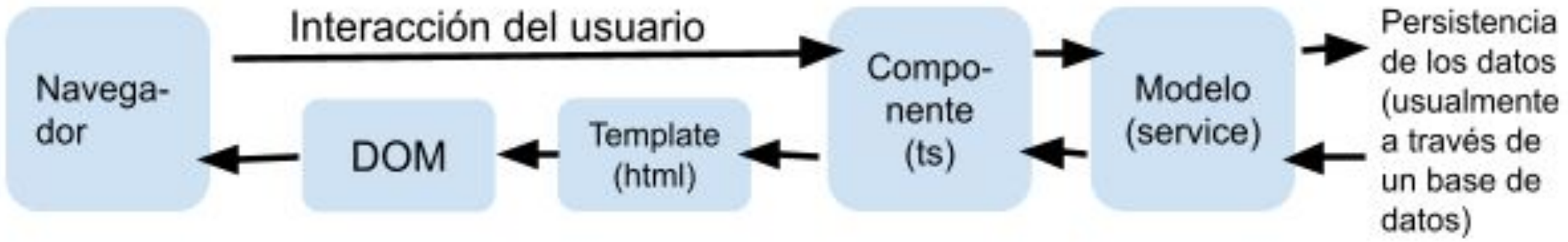
De acuerdo a estos conceptos básicos y teniendo en cuenta que Angular es un framework de frontend, podemos presentar la implementación del modelo MVC en Angular como sigue:



- Los datos se obtienen del backend (usualmente a través de un API Rest)
- El objetivo del controlador y las vistas es operar los datos del modelo para manipular así el DOM de manera que el usuario pueda interactuar con él ya sea para acceder a los datos o manipularlos.

MVC en Angular

Sin embargo, es importante mencionar que angular difiere un poco en cuanto a esta terminología dado que la arquitectura de Angular, como mencionamos en el módulo 2, se basa en **módulos** y **componentes**. Lo que quiere decir, que la implementación del modelo MVC en Angular más correcto luce como sigue:



Modelos desde la perspectiva de Angular

Los modelos en angular encapsulan los datos y definen la lógica para manipularlos. Por ejemplo, un modelo puede representar un personaje en un video juego, una cuenta en una billetera digital, etc.

Los modelos deberían:

- contener la lógica para crear, administrar y modificar los datos del dominio (incluso si eso significa ejecutar la lógica remota a través de servicios web)
- exponer de manera prolija los datos del modelo y las operaciones en ella.

Y no deberían:

- exponer cómo el modelo de datos es obtenido o administrado. No debería exponer por ejemplo el API Rest o servicio web a la vistas y componentes.
- contener lógica en relación a la interacción con el usuario (porque es trabajo del componente).
- contener lógica de presentación de datos (porque es trabajo del template).

Controladores/componentes desde la perspectiva de Angular

Los controladores, conocidos como componentes en Angular, son los que permiten la interacción entre el template y el modelo. Por lo general, los componentes agregan la lógica de negocio requerida para presentar algunos aspectos al modelo y ejecutar acciones sobre estos.

En otras palabras, los controladores interpretan las acciones del usuario realizadas en la vista y comunican datos nuevos o modificados al modelo. A su vez, cuando los datos cambian el modelo notifica al controlador quien comunica los nuevos datos a la vista para mostrarlos.

Los componentes deberían:

- contener la lógica necesaria para configurar el estado inicial del template.
- contener la lógica y comportamientos requeridos para presentar datos del modelo.
- contener la lógica y comportamientos requeridos para actualizar el modelo en función de la interacción del usuario.

Y no deberían:

- contener la lógica que manipula el DOM (porque es trabajo del template)
- contener la lógica que gestiona la persistencia de los datos (porque es trabajo del modelo)

Vistas desde la perspectiva de Angular

Las vistas, conocidas como template o plantillas, se definen mediante elementos HTML las cuales, están mejoradas en Angular gracias a los sistemas de enlaces (data binding).

Los template deberían:

- contener la lógica de presentación.

Y no deberían:

- contener la lógica compleja.
- contener la lógica que manipula el modelo.

Nota: Las vistas pueden tener lógica (por ej. a través de las directivas) pero debe ser simple y debe usarse con moderación.

Consumir un API Rest

Hasta ahora hemos visto qué es un servicio y cómo inyectar en el componente y hacer uso del mismo. Los datos por el momento fueron simulados o escritos en duro en el servicio. Sin embargo para acceder a ellos o manipularlos desde una API REST tenemos que hacer llamadas HTTP.

Nota: una API es un conjunto de rutas que provee un servidor (backend) que permiten el acceso y la manipulación de los datos.

Para hacer las llamadas HTTP, Angular nos provee un módulo que facilita esta tarea, el módulo httpClient.

Para usar HttpClient de Angular en cualquier parte, tenemos que:

¿Cómo consumir un API Rest?

Para hacer las llamadas HTTP, Angular nos provee un módulo que facilita esta tarea, el módulo **httpClient**.

Para usar HttpClient de Angular en cualquier parte, tenemos que:

1. Importar el módulo **HttpClientModule**, en la sección imports de el **app.module.ts** como sigue:

```
...
import { HttpClientModule } from '@angular/common/http';

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

Consumir un API Rest

2- Importar e inyectar en el constructor del servicio **<<name-servicio>>.service.app** que consumirá la API Rest:

```
import { Injectable } from '@angular/core';
import {HttpClient} from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class UsuarioService {

  constructor(private http:HttpClient) {
   }
}
```

Inyectar HttpClient en el constructor del servicio.

Con esto, ya podemos hacer llamadas http.

Llamadas HTTP

A continuación se enumeran los métodos de petición http más utilizados:

- **GET**, permite recuperar recursos del servidor (datos). Si la respuesta es positiva (200 OK), el método GET devuelve la representación del recurso en un formato concreto: HTML, XML, JSON u otro. De lo contrario, si la respuesta es negativa, devuelve 404 (not found) o 400 (bad request).
- **POST**, permite crear o ejecutar acciones sobre recursos del servidor. Generalmente se utiliza este método cuando se envían datos de un formulario al servidor. Si la respuesta es positiva, el método POST devuelve 201 (created).
- **PUT,** permite modificar recursos del servidor (aunque permite también crear). Si la respuesta es positiva, el método PUT devuelve 201(created) o 204 (no response).
- **DELETE**, permite eliminar recursos del servidor. Si la respuesta es positiva, el método DELETE devuelve 200 junto con un body response, o 204 sin body.

Peticiones HTTP desde Angular

Llamadas Get: Son las llamadas más frecuentes, y permiten obtener datos desde el servidor.

Sintaxis:

URL del API REST

```
this.http.get("https://url-api-rest");
```

Llamadas Post y Put: Las llamadas POST y PUT sirven para enviar datos al servidor y que éste nos responda. Las peticiones POST se usan frecuentemente para crear recursos como por ejemplo: crear usuarios mientras que las peticiones PUT, se usan para actualizar un objeto previamente creado. En ambos casos, se pasa un objeto o conjunto de objetos a crear o modificar.

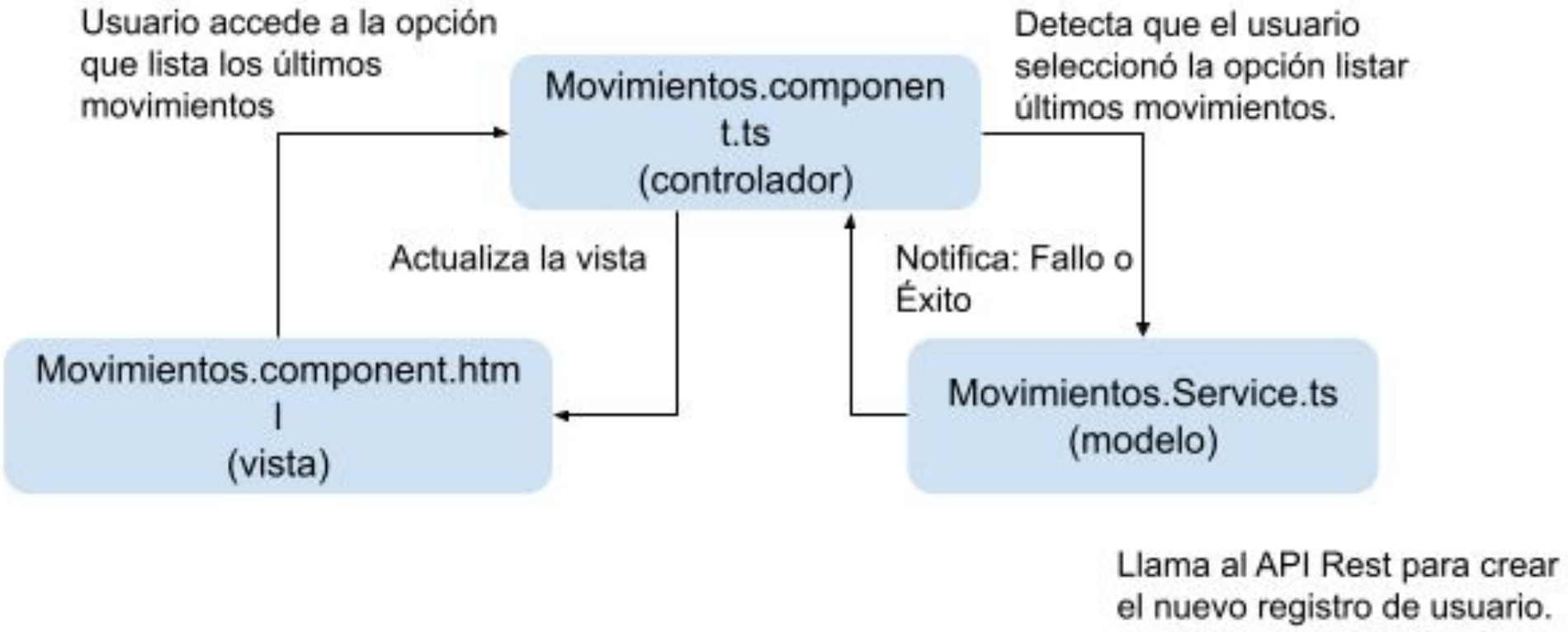
Sintaxis:

Objeto usuario, contiene los datos que se envían al servidor.

```
this.http.post<Usuario>("https://url-api-rest", usuario);
this.http.put<Usuario>("https://url-api-rest", usuario);
```

Implementación del patrón MVC en Angular

Continuando con la aplicación que venimos desarrollando, y pensando en el patrón MVC desde la perspectiva de Angular, veamos cómo recuperar los movimientos consumiendo un API Rest.



Crear el Servicio Cuenta (Modelo)

Como pudimos observar en los gráficos de MVC, el modelo contiene los datos y los mecanismos necesarios para manipularlos. Es entonces el modelo, el punto final del frontend dado que, es a partir de ahí debe conectarse con un servicio web o API Rest para acceder a los datos, es decir con el backend. Los modelos de datos pueden ser simples como por ejemplo para trabajar un CRUD (Create, Read, Update y Delete) o mucho más complejos.

En Angular, como mencionamos previamente, el modelo está codificado en servicios.

1. Ir a la consola DOS o “Símbolo del Sistema” del sistema operativo o Terminal de VSCode.
2. Ejecutar el comando: ***ng generate service servicios/cuenta***

   o su abreviado: ***ng g s servicios/cuenta***

   Nota: Recuerda que services es la carpeta donde le especificamos a AngularCLi que cree el archivo.

   Una vez ejecutado el comando, AngularCLI crea un archivo generará el archivo **cuenta.service.ts**

Crear el Servicio Cuenta (Modelo)

3. Editar el servicio a fin de:

   a. Importar las clases HttpClient de @angular/common/http y luego, inyectar en el constructor.
   b. Editar el archivo cuenta.service.ts a fin de hacer la petición GET.

```
import { Injectable } from '@angular/core';
import {HttpClient} from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})

export class CuentaService {
  url="https://localhost:44303/api/cuenta";
  constructor(private http:HttpClient) { }

  ObtenerUltimosMovimientos(id:number)
  {
    return this.http.get<Cuenta>(this.url+"/"+id.toString());
  }
}
```

Método al que luego el componente (ts) podrá subscribirse. El mismo, recibe el id de la cuenta como parámetro y devuelve un objeto Cuenta (modificado con el id y demás datos que puedan obtenerse del servidor).

Llamada al método GET

Nota: No olvidar importar el módulo **HttpClientModule**, en la sección imports de el **app.module.ts**

Crear el Servicio Cuenta (modelo)

A continuación la definición de la **clase cuenta**.

```
export class Cuenta
{
  id:number=0;
  cvu:string="";
  saldo:number=0;
  id_persona:number=0;
  estado:boolean=false;
  movimientos:Movimiento[];
  //A modo de ejemplo se deja así pero lo ideal es crear propiedades para acceder a los atributos.
}

export class Movimiento
{
  id:number=0;
  fechahora:string="";
  monto:number=0;
  cvuOrigen:string="";
  cvuDestino:string="";
  idCuenta:number=0;
  tipoMovimiento:string="";
}
```

Puedes definirlas en un archivo aparte (ej. cuenta.model.ts) si deseas… Recuerda que si este es el caso, deberás hacer la importación correspondiente siempre que la utilices.

Nota: No olvidar importar el módulo **HttpClientModule**, en la sección imports de el **app.module.ts**

Crear/editar el componente Movimientos (controlador)

1- En el controlador, archivo **movimientos.component.ts**, editar a fin de:

Importar el CuentaService y Cuenta:

```
import { CuentaService , Cuenta, Movimiento} from 'src/app/servicios/cuenta.service';
```

2- Inyectar el CuentaService en el constructor

```
constructor(private cuentaService:CuentaService) {...}
```

3- Editar el evento onInit() a fin de consumir el servicio:

```
ngOnInit(): void {
  this.cuentaService.ObtenerUltimosMovimientos(2).subscribe(
    data=> {
      console.log(data);
      this.movimientos=data['Movimientos'];
    }
  );
}
```

Subscribe, funciona como las promesas de Javascript permitiendo esperar hasta que la petición termine. Observa que dentro del método se ejecuta una arrow function, la misma permite devolver el objeto data que contiene la respuesta del ClienteService.

Crear/editar el componente Movimientos (vista)

4- En la vista, archivo **cuenta.component.html**, editar a fin de mostrar los datos en la tabla:

```
<h1>Ultimos Movimientos</h1>
<h2>{{hoy|date: "d/M/yy"}}</h2>
<table *ngIf="mostrar_movimientos">
    <thead>
        <th>Operación</th>
        <th>Monto</th>
    </thead>
    <tbody>
        <tr *ngFor="let element of movimientos" >
            <td>{{element.TipoMovimiento}}</td>
            <td>{{element.Monto|currency}}</td>
        </tr>
    </tbody>
</table>
```

Accedemos a los movimientos (objeto Cuenta)

ISPC
INSTITUTO SUPERIOR
POLITÉCNICO CÓRDOBA

DEMO

ISPC
INSTITUTO SUPERIOR
POLITÉCNICO CÓRDOBA

ISPC
INSTITUTO SUPERIOR POLITÉCNICO CÓRDOBA

Referencias

https://wuschools.com/what-is-mvc-and-understanding-the-mvc-pattern-in-angular/
https://scotch.io/tutorials/mvc-in-an-angular-world
https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html
https://codingpotions.com/angular-servicios-llamadas-http
https://codingpotions.com/angular-seguridad
https://angular.io/api/router/RouterStateSnapshot
https://lautarocarro.blog/usando-http-interceptors-en-blazor/