PROGRAMACIÓN DE APLICACIONES MÓVILES

PGY4121

Profesor:

DuocUC®

ESCUELA DE INFORMÁTICA Y TELECOMUNICACIONES

Actividad N°2.3: API Connection

Conectándonos a una APIRest

DuocUC®

Objetivos

Lo que se espera que aprendas en esta Actividad es:

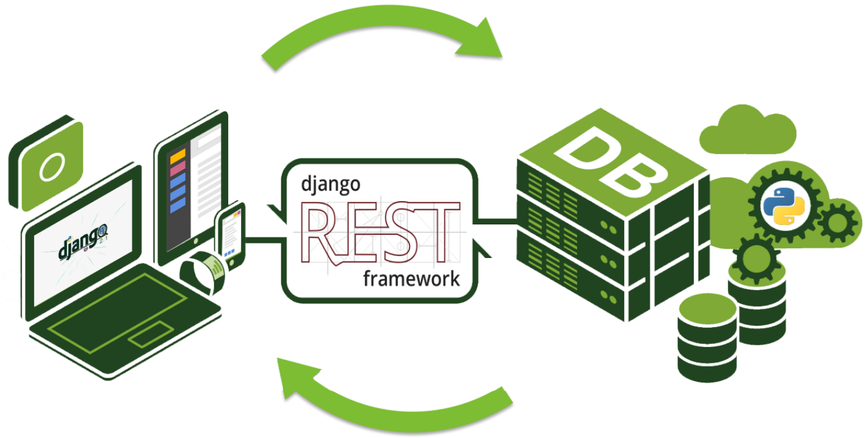
- Conocer cómo conectar nuestra aplicación a un servicio APIRest mediante Service
- Conocer cómo se programan consultas a APIRest
- Conocer los Observables
- Conocer peticiones GET, POST, PUT, DELETE

¿Qué es API Rest?

DuocUC®

PGY4121 APIRest

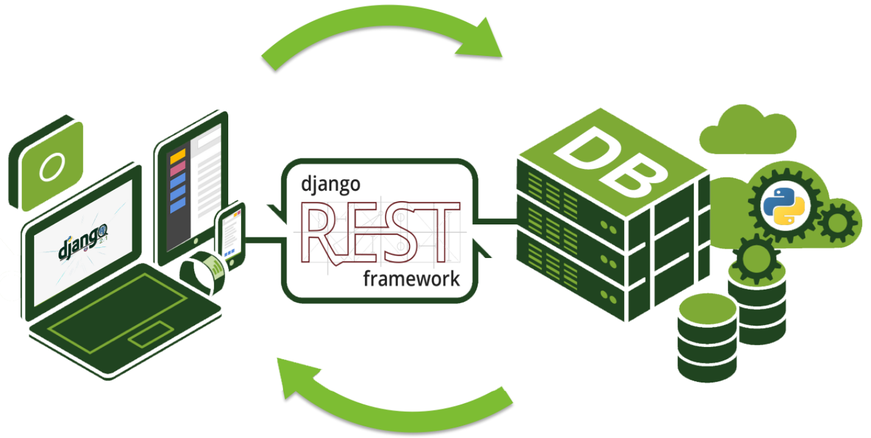
- API es un **conjunto de reglas y especificaciones** que las aplicaciones pueden seguir para comunicarse entre ella. Es un mecanismo útil para **conectar dos o más software diferentes** entre sí sin importar en el **lenguaje en el cual se desarrolló** dichos sistemas



http://debsconsultores.blogspot.com/2019/03/json-web-token-en-django-rest-framework.html

PGY4121 APIRest

- REST es una interfaz entre sistemas que utilice directamente HTTP para obtener datos o indicar la ejecución de operaciones entre datos en formatos JSON, XML etc



http://debsconsultores.blogspot.com/2019/03/json-web-token-en-django-rest-framework.html

Métodos

PGY4121 | HTTP RESTful APIs

HTTP METHOD	CRUD	RETURN
POST	Create	201
GET	Read	200
PUT	Update/Replace	200 (OK) | 204 (No Content) | 404 Not Found
PATCH	Partial Update/ Modify	200 (OK) | 204 (No Content) | 404 Not Found
DELETE	Delete	200 (OK) | 404 Not Found

DuocUC®

Component Service

DuocUC®

PGY4121 Service

- Para consumir un servicio de una API se debe manejar mediante peticiones.

- Las peticiones la gestionará la clase Servicio que se debe generar para el proyecto

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

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

  apiURL = `https://jsonplaceholder.typicode.com`;

  constructor(private http: HttpClient) { }

  getPosts(id) {
    return this.http.get(`${this.apiURL}/posts/${id}`);
  }
}
```

PGY4121 Service

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

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

  apiURL = `https://jsonplaceholder.typicode.com`;

  constructor(private http: HttpClient) { }

  getPosts(id) {
    return this.http.get(`${this.apiURL}/posts/${id}`);
  }
}
```

Nos permite, realizar consultas HTTP

Se establece la URL Base

Se establece el método de consulta get

Métodos

PGY4121 Get Item

```
/**
   * @param id id del objeto a rescatar
   * @returns Devuelve un Observable se utiliza any para adaptar el
   * objeto que devuelve pero se pueden utilizar modelos ej:
   * Post = {...}
   *
   * El retry permite volver a intentar 3 veces si este no se
ejecuta
   * correctamente, posteriormente lanza error
   */
  getPost(id):Observable<any>{
    return this.http.get(this.apiURL+'/posts/'+id).pipe(
      retry(3)
    );
  }
```

PGY4121 Get Items

```
/**
   * @param id id del objeto a rescatar
   * @returns Devuelve un Observable se utiliza any para adaptar el
   * objeto que devuelve pero se pueden utilizar modelos ej:
   * Post = {...}
   *
   * El retry permite volver a intentar 3 veces si este no se
ejecuta
   * correctamente, posteriormente lanza error
   */
  getPosts():Observable<any>{
    return this.http.get(this.apiURL+'/posts/').pipe(
      retry(3)
    );
  }
```

PGY4121 Update Item

```
/**
   *
   * @param id id del objeto a actualizar
   * @param post Objeto a actualizar al servicio mediante PUT
   * @returns Devuelve un Observable se utiliza any para adaptar
   * el objeto que devuelve pero se pueden utilizar modelos ej:
   * Post = {...}
   *
   * El retry permite volver a intentar 3 veces si este no se ejecuta
   * correctamente, posteriormente lanza error
   */
  updatePost(id,post):Observable<any>{
    return
this.http.put(this.apiURL+'/posts/'+id,post,this.httpOptions).pipe(retry(3));
  }
```

PGY4121 Delete Item

```
/**
   *
   * @param post Objeto a enviar al servicio mediante POST
   * @returns Devuelve un Observable se utiliza any para adaptar
   * el objeto que devuelve pero se pueden utilizar modelos ej:
   * Post = {...}
   *
   * El retry permite volver a intentar 3 veces si este no se ejecuta
   * correctamente, posteriormente lanza error
   */
  createPost(post):Observable<any>{
    return this.http.post(this.apiURL+'/posts/',post,this.httpOptions)
    .pipe(
      retry(3)
    );
  }
```

Cómo Utilizar los métodos

PGY4121 Utilización

- Los métodos antes definidos en el servicio, retornan un Observable con un tipo de valor. Este Observable lo podemos manejar en la clase solicitante mediante el Subscribe y Lambda.

PGY4121 Utilización

```
this.api.getPosts().subscribe((res)=>{
      this.message = ''+res[0].title;
      console.log(res[0]);
    },(error)=>{
      //this.message=error;
      console.log(error);
    });
```

En el código de ejemplo se puede observar que tenemos un función getPosts() desde la variable api. A esta función nos suscribimos para cuando su estado cambie mediante la función subscribe() y con una función lambda gestionamos la respuesta y el error mediante .subscribe((res)=>{ ***si la solicitud fue correcta*** },(error)=>{ ***en caso de error***});

Si es una lista accedemos a la posición del objeto y posteriormente al campo requerido

PGY4121 Utilización

```
createPost(){
    var post={
      title: 'titulo prueba',
      body: 'algún cuerpo del post',
      userId: 1
    }
    this.api.createPost(post).subscribe((success)=>{
      console.log(success);
    },error=>{
      console.log(error);
    });
  }
```

En el código de ejemplo podrán observar una función que permite la creación de objeto mediante una función post que lo provee el createPost(), al igual que al ejemplo anterior nos suscribimos a su cambio de estado para recepcionar el resultado de la operación post.

Operación similar se realiza para acciones como Update, Delete o Get.

DuocUC®

Consideraciones

PGY4121 Consideraciones

- Al momento de hacer uso de los servicios disponibles por el servidor a consultar, debemos tener en cuenta que nos pueden salir ciertos errores por la configuración del APIRest. Una de los errores más comunes es el

```
Access to XMLHttpRequest at 'https://jsonplaceholder.typicode.com/posts' from origin home:1
'http://localhost:8100' has been blocked by CORS policy: No 'Access-Control-Allow-Origin'
header is present on the requested resource.
```

Una posible solución es integrar a nuestro Header la opción 'Access-Control-Allow-Origin’ con el valor ‘*’ como se observa en el siguiente ejemplo.

PGY4121 Access-Control-Allow-Origin

```
httpOptions = {
    headers: new HttpHeaders({
      'Content-Type': 'application/json',
      'Access-Control-Allow-Origin':'*'
    })
  }
```

Proceso de Instalación

PGY4121 Instalación

- Para el proceso de implementación de un servicio que consuma un servicio APIRest consulten el material complementario suministrado por el docente como también la documentación oficial de Ionic relacionado con comando CLI generate.

- A modo general el comando a utilizar es:

  ionic generate service nombre_servicio. Ej “ionic generate service api”

Observables

PGY4121 Observables

- El **Observables** corresponde a un **patrón de diseño** en la programación que nos permite estar atento a los **cambios** que ocurren sobre un objeto en particular.

- Los **Observables** funcionan junto a **Observadores** o Oyentes que están pendientes y se **Suscriben** a los Observables

DuocUC®

PGY4121 Observables

- Un ejemplo para entender cómo funciona, es Twitter, cuando se suscribe a un usuario para recibir notificaciones cuando se realice algún Twitt, así funcionan los Observables.

PGY4121 Composición

```
.subscribe(
  (res)=>{ //Bloque success 200 / 201 }
  ,(error)=>{ //Bloque Error }
);
```

Resumen

PGY4121 Resumen

- Como pudimos observar, para consultar servicios APIRest necesitamos hacer uso del componente “Service” de ionic, el cual nos gestionará y proveerá de las funciones.
- Conocimos los Observables
- Conocimos cómo se programan consultas a APIRest
- Conocer peticiones GET, POST, PUT, DELETE