Servicios Web con Ionic

UD5: Interacción con fuentes externas en Ionic

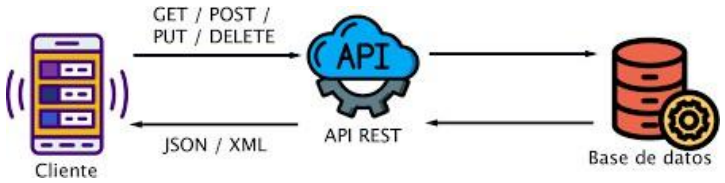
Objetivos de aprendizaje

- Consumir servicios REST desde aplicaciones Ionic
- Conocer los métodos de autentificación usados con servicios Web

Servicios Web

Servicios Web con Ionic

Recordemos: APIs REST



- REST (Representational State Transfer) es un estilo arquitectónico que obliga a mis aplicaciones a implementar una interfaz bien definida
- Sin estado: El servidor no guarda datos del cliente entre solicitudes
- Las operaciones CRUD tienen equivalencia con métodos HTTP

POST	GET	PUT/PATCH	DELETE
Crear (Create)	Leer (Read)	Actualizar (Update)	Borrar (Delete)

APIs para peticiones HTTP en Ionic

- En Ionic podemos manejar 2 librerías para el manejo de peticiones HTTP

HttpClient	CapacitorHttp
API nativa de Angular para realizar peticiones HTTP	Plugin de Capacitor para manejar peticiones HTTP de manera nativa
Interacción con APIs REST desde aplicaciones Web y móviles	Interacción con APIs REST desde móvil
Aprovecha mejor características de Angular y funciona de manera efectiva en el entorno del navegador	• Realiza peticiones desde fuera del contexto del navegador, evitando problemas con CORS y cookies • Las peticiones pueden ser más rápidas y seguras
Puede estar limitado por políticas de seguridad del navegador (CORS y cookies)	• Requieren instalar y configurar el plugin Capacitor • Menor integración con Angular

Ejemplo de API pública

- Vamos a utilizar la API JSON PlaceHolder
- Proporciona 6 colecciones a las que puedo acceder
  - Posts → `/posts`
  - Comments → `/comments`
  - Albums → `/albums`
  - Photos → `/photos`
  - Todos → `/todos`
  - Users → `/users`

Peticiones REST con HttpClient

Servicios Web con Ionic

Variables de entorno

- Una buena práctica es guardar la URL base en una variable de entorno (archivo `environment.ts`)
- Así es accesible desde todos los puntos donde se usa y es fácil de actualizar.

environments/environments.ts

```
export const environment = {
  production: false,
  apiURL: "https://dummyjson.com/products"
};
```

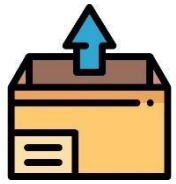
```
import { environment } from 'src/environments/environment';
```

```
environment.apiURL
```

USO

SERVICIO DE USO

Peticiones GET (lectura de datos)



- Generamos un servicio que va a interaccionar con el servicio Web:

```
ionic generate service shared/services/ProductoService
```

Importando HttpClient

```ts
import { provideHttpClient } from '@angular/common/http';

bootstrapApplication(AppComponent, {
  providers: [
    { provide: RouteReuseStrategy, useClass: IonicRouteStrategy },
    provideIonicAngular(),
    provideRouter(routes, withPreloading(PreloadAllModules)),
    provideHttpClient()
  ],
});
```

main.ts

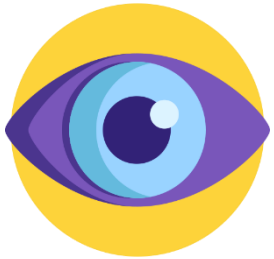
Inyectando `HttpClient` en el servicio

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

@Injectable({
  providedIn: 'root'
})
export class PruebaHttpService {
  private httpClient=inject(HttpClient);


  constructor() { }

}
```

ProductosService

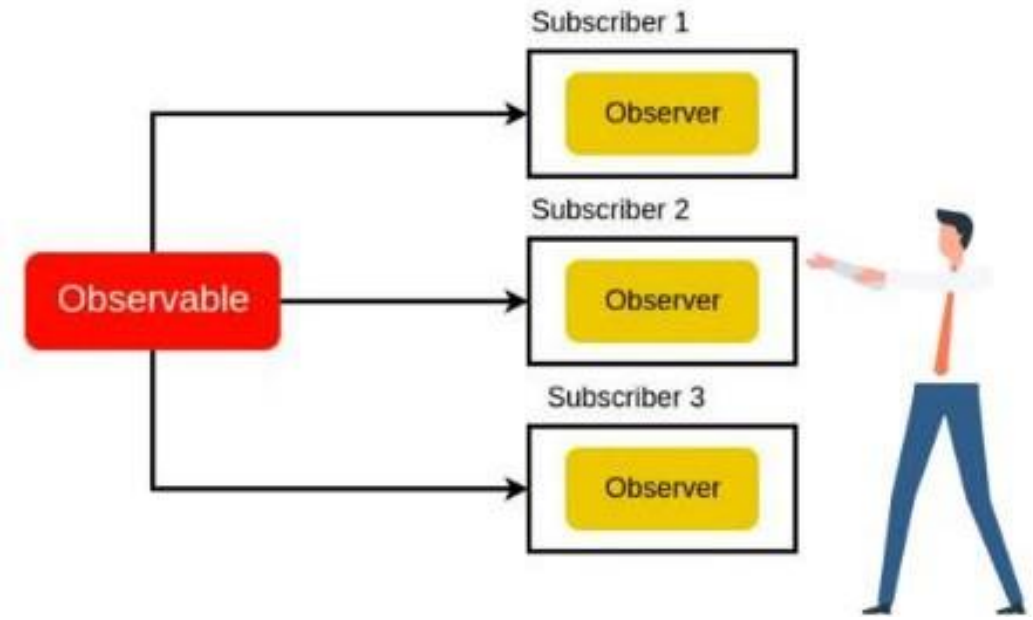
Utilizaremos la instancia de `HttpClient` para hacer las peticiones

Recordemos: Observable



- Un observable es una manera de esperar a por una respuesta de un servidor
  - Pertenece a la librería RxJs (Reactive Extensions)
- Se usan para peticiones asíncronas
  - Hacemos una petición y mientras esperamos la respuesta podemos seguir trabajando en la aplicación
- Una vez recibida la respuesta del servidor, el observable envía la información a los componentes que están esperando dicha información

Recordemos: Observable

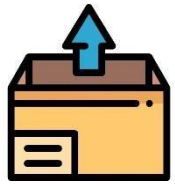


Cabeceras HTTP

- Muchas peticiones HTTP requieren cabeceras que el servidor procesa
  - Ejemplo: Token de autentificación
- Usamos una clase específica de tipo `HttpHeaders` para proporcionar las cabeceras

```ts
const headers=new HttpHeaders({
  "Content-Type":"application/json",
  "Authorization": "JWT" + localStorage.getItem("token")
});
```

Petición GET



- Obtenemos datos de un servicio REST
- El método `httpClient.get()` devuelve un `Observable` sobre la respuesta.

```ts
import { Observable } from 'rxjs';

getAll():Observable<ProductoAPIResponse>{
  return this.httpClient.get<ProductoAPIResponse>(environment.apiURL);
}
```

ProductosService

Extensión: Paste JSON as code

- Facilita la generación de interfaces a partir de objetos JSON sacados de una API
  - Copiamos el objeto JSON
  - *Control + Shift + P → Paste as code*

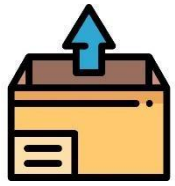
Suscripción

- Me subscribo al `Observable` a la espera de recibir la respuesta desde el servidor

```ts
let respuesta=this.productosService.getAll();
  respuesta.subscribe({
    next:(data)=>{
      console.log(data.products)
    },
    error:(error)=>{
      console.error(error)
    },
    complete:()=>{
      console.log('Completado');
    }
  });
}
```

lista-productos.page.ts

Petición GET (opción B)



- Podríamos determinar que devolvemos una Promesa en lugar de un Observable
  - Más fácil de entender si son tareas que se completan una única vez

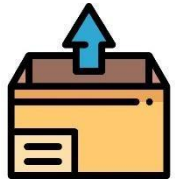
TS

```
import { Observable } from 'rxjs';

getAll():Promise<ProductoAPIResponse>{
  return firstValueFrom(this.httpClient.get<ProductoAPIResponse>(environment.apiURL));
}
```

Toma el primer valor emitido por el Observable y lo convierte en una promesa

ProductosService

Recuperamos valor (opción B)



- Obtenemos el valor de la promesa con sintaxis `async/await`
  - Sintaxis más limpia

```ts
async ngOnInit() {
  let respuesta:ProductoAPIResponse =await this.productosService.getAll();
  this.productos=respuesta.products;
  console.log(this.productos);
}
```

lista-productos.page.ts

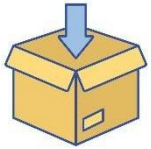
Mostrando datos

- Mostramos el listado de productos con la nueva sintaxis @for


```
<ion-list>
  @for(producto of productos;track producto.id){
    <ion-item>
       <ion-thumbnail slot="start">
        <img [src]="producto.thumbnail" alt="producto.name">
       </ion-thumbnail>
       <ion-label>
        <h2> {{producto.title}}</h2>
        <p>{{producto.description}}</p>
       </ion-label>
     </ion-item>
  }
</ion-list>
```

lista-productos.html

Petición POST



- Enviamos datos a un servicio

```ts
insertaItem(item:Producto):Observable<Producto>{
  return  this.httpClient.post<Producto>(`${environment.apiURL}/add`,item);
}
```

ProductosService

Utilizando el servicio

```
  insertarProducto(){

this.productosService.insertaItem(

 {title:'Nuevo Producto',price:200, thumbnail:'https://www.google.com'}).

subscribe(

   {

         next:(data)=>{

           console.log(data)

         },
         error:(error)=>{
           console.error(error)

         },
         complete:()=>{
           console.log('Completado');

        }

      }

 );

}
```

Página donde uso el servicio

Petición DELETE

- Eliminamos datos de un servicio

```ts
public eliminarItem(id:number){
  return this.httpClient.delete(`${this.url}/${id}`)
}
```

ProductosService

Peticiones REST CapacitorHttp

Servicios Web con Ionic

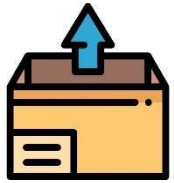
CapacitorHttp

- Necesitamos habilitarlo en los settings

```ts
import type { CapacitorConfig } from '@capacitor/cli';

const config: CapacitorConfig = {
  appId: 'io.ionic.starter',
  appName: 'SegundaApp',
  webDir: 'www',
  plugins:{
    CapacitorHttp:{
      enabled:true
    }
  }
};

export default config;
```

capacitor.config.ts

Petición GET

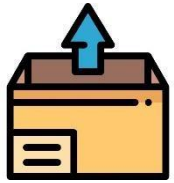


- `CapacitorHttp` devuelve una promesa.

```ts
import {CapacitorHttp, HttpResponse} from '@capacitor/core';

 getAllCapacitor():Promise<ProductoAPIResponse>{
  return CapacitorHttp.get({
    url:environment.apiURL
  }).then((response:HttpResponse)=>{
    return response.data as ProductoAPIResponse;
  });
}
```

ProductosService

Petición GET

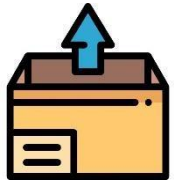


- Usando sintaxis `async/await`

```ts
async getAllCapacitor():Promise<ProductoAPIResponse>{

  let response=await CapacitorHttp.get({
    url:environment.apiURL
  });
  return response.data;
}
```

ProductosService

Accediendo a la respuesta



```
 ngOnInit() {
  this.productosService.getAllCapacitor().then((respuesta:ProductoAPIResponse)=>{
    this.productos=respuesta.products;
   });
}
```

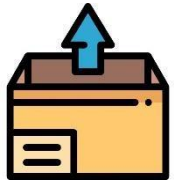
Opción A: Sintaxis clásica de promesa

Página donde accedo al servicio

```
  async ngOnInit() {
    let respuesta:ProductoAPIResponse =await this.productosService.getAllCapacitor();
    this.productos=respuesta.products;
  }
```

Opción B: Sintaxis `async/await`

Petición GET con Observable

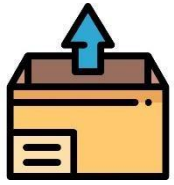


- Podemos convertir la respuesta en Observable utilizando la función `from()`

```ts
 getAllCapacitor():Observable<ProductoAPIResponse>{
  return from(CapacitorHttp.get({
    url:environment.apiURL
  }).then((response:HttpResponse)=>{
    return response.data as ProductoAPIResponse;
  }));
}
```

ProductosService

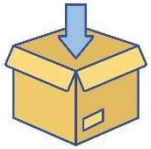
Accediendo a la respuesta



```
ngOnInit() {
  let respuesta=this.productosService.getAllCapacitor();
  respuesta.subscribe({
    next:(data)=>{
      this.productos=data.products;
    },
    error:(error)=>{
      console.error(error)
    },
    complete:()=>{
      console.log('Completado');
    }
  });
}
```

Página donde accedo al servicio

Petición POST

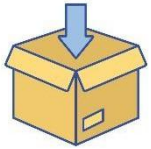


- Sintaxis de promesa clásica

```
insertaItemCapacitor(item:Producto):Promise<Producto>{
  return CapacitorHttp.post(
   {url:`${environment.apiURL}/add`,
         headers: { 'Content-Type': 'application/json' },
         data:item}).
       then((response:HttpResponse)=>{
     return response.data as Producto;
   });
```

ProductosService

Petición POST

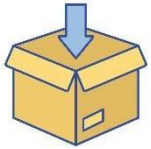


- Sintaxis async/await

```ts
async insertaItemCapacitor(item:Producto):Promise<Producto>{
let response=await CapacitorHttp.post(
 {url:`${environment.apiURL}/add`,
       headers: { 'Content-Type': 'application/json' },
       data:item});
   return response.data ;
 };
```

ProductosService

Petición POST



- Accedemos a la respuesta

```ts
insertarProducto(){
  this.productosService.insertaItemCapacitor({
      title:'Nuevo P',
      price:200,
      thumbnail:'https://www.google.com'
    }).then((data)=>{
      console.log(data);
    }).catch((error)=>{
      console.error(error);
    });
}
```

Página donde accedo al servicio

Autentificación en peticiones REST

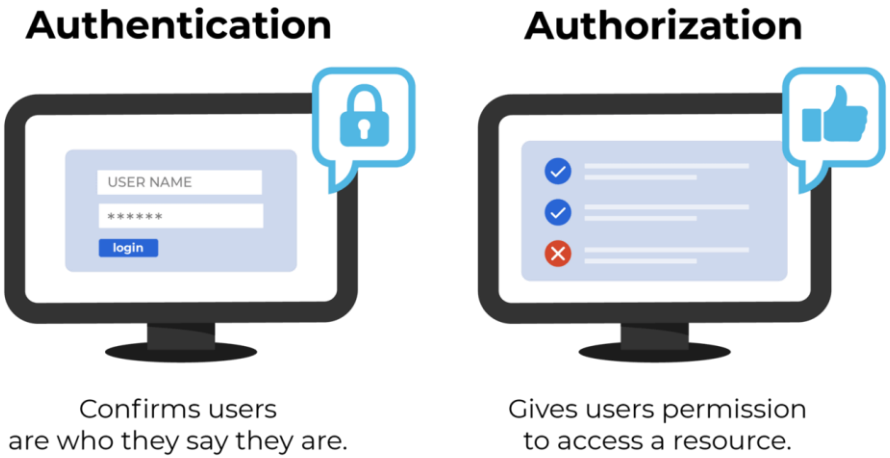
Creación de aplicaciones con Ionic

Autentificación de peticiones REST

- Hasta ahora hemos usado APIs de dominio público que no tenían restricciones
- En muchas ocasiones las APIs requieren autentificación para limitar el número de peticiones que podemos realizar y asegurar que no agotamos recursos.
- Esto también será necesario en el caso de que mi aplicación exporte una API con datos privados.

Autentificación vs autorización

- Autentificación: Comprobaciónde que el usuario es quien dice ser
- Autorización: Comprobación de permisos de usuario



Autentificación asíncrona

- Recordemos que REST no tiene estado
- Eso implica que enviar las credenciales una vez para iniciar la sesión no es suficiente
- No se recuerdan las credenciales por no existir una sesión activa HTTP, luego tenemos que indicarlas cada vez que hagamos una petición.

Autentificación en servicios REST

Habitualmente se usan 4 modelos de autentificación en servicios REST:

1. Autentificación básica
2. Autenticación basada en Token (Bearer)
3. Autentificacion basada en clave API
4. OAuth 2.0 (autentificación abierta)

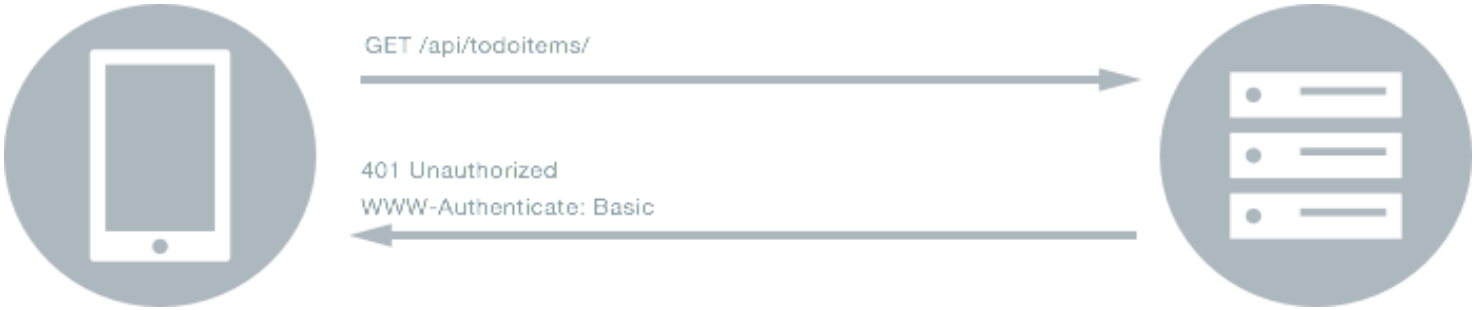
1. Autentificación básica

- Es el mecanismo de autentificación más sencillo admitido en HTTP
- El cliente envía nombre de usuario y contraseña como texto codificado en base 64 sin cifrar
- Este tipo de autentificación solo debería usarse a través de una conexión HTTPS
  - De lo contrario cualquiera puede acceder.

Se usa el encabezado HTTP Authorization

1. Autentificación básica (II)

- Si un servicio web recibe una solicitud de un recurso protegido
  - Rechaza la solicitud con un código de estado HTTP 401 (acceso denegado)
  - Establece el encabezado de respuesta `WWW-Authenticate:Basic`.



1. Autentificación básica (III)

- Si un servicio web recibe una solicitud de un recurso protegido con el encabezado correcto devueve el código de estado 200 (OK)
- El encabezado correcto debería ser del tipo:

```
Authorization: Basic cadena
```

`cadena es la representación en Base64 de la combinación usuario:password`

2. Autentificación basada en Token (Bearer)

- El usuario se identifica con nombre de usuario y password al igual que con la autentificación básica
- Tras la primera petición de autentificación, el servidor genera un token basado en esas credenciales
- El token se guarda en la BD del servidor y lo devuelve al usuario para que lo envie en cada petición HTTP
- Por regla general, los token están codificados con la fecha/hora y se pueden configurar para que caduquen tras un tiempo definido.

JWT (JSON Web Token)

- Es un formato específico de Token Bearer
- Es un estándar abierto (RFC 7519) que define una manera de representar la información.
- Un JWT es un objeto JSON que está codificado en una cadena y puede contener información del usuario como roles, permisos…
  - Además, puede ser revisado por firma digital.

Token JWT

- Un token JWT se compone de 3 partes:

```
header.payload.signature
```

El header y el payload no están encriptados, solo codificados en Base64, luego se pueden decodificar en cliente con JS

Partes de un Token

Sección	Contenido	¿Se puede decodificar en cliente?
header	Algoritmo de firma y tipo de token	Si
Payload	Datos del usuario	Si
Signature	Firma para validar que no fue modificado. Se genera con la clave secreta del servidor	NO. Sirve para validar el token, junto con la clave secreta en el servidor

Si algún código malicioso modifica la firma, la validación en será incorrecta

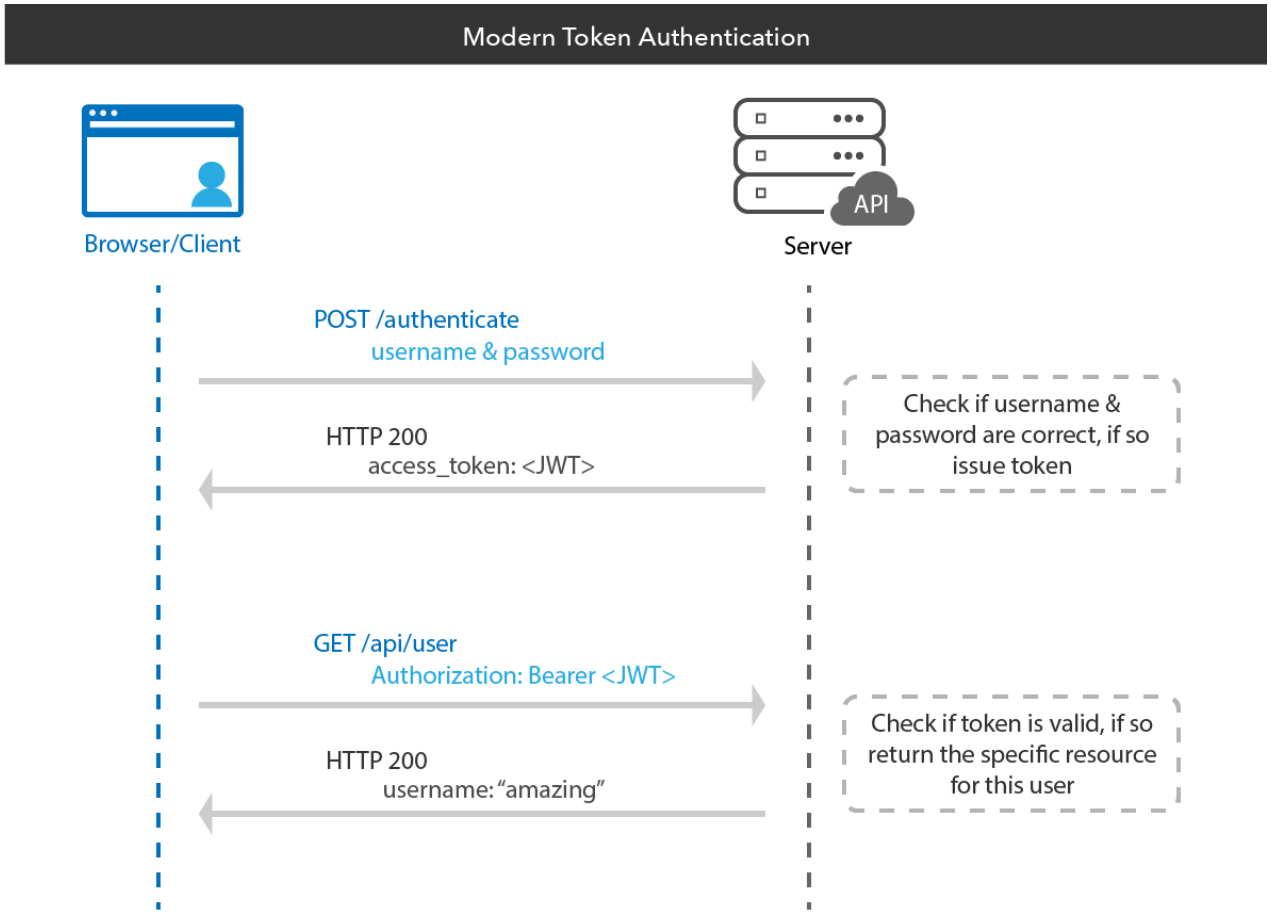
Para validar un token en cliente necesitamos la clave secreta o pública del servidor (depende del algoritmo con el que ha sido firmada). **Normalmente no disponemos de ellas, por eso la verificación suele hacerse en servidor**

Payload

- El payload contiene los datos del usuario o la sesión.
- Se les llama “claims” (reclamaciones)
- Algunos de los sub más comunes

Claim	Contenido	Ejemplo
sub	Identificador del usuario	"12463232"
name	Nombre del usuario	"Juan"
Role	Rol del usuario	"admin"
iat	Issued at Fecha de emisión en formato timestamp	1696720800
exp	Expiration date Fecha de expiración en formato timestamp	1696730800

2. Autentificación basada en Token (Bearer)



`ejemplos/5_servicios_web/AppCitas`

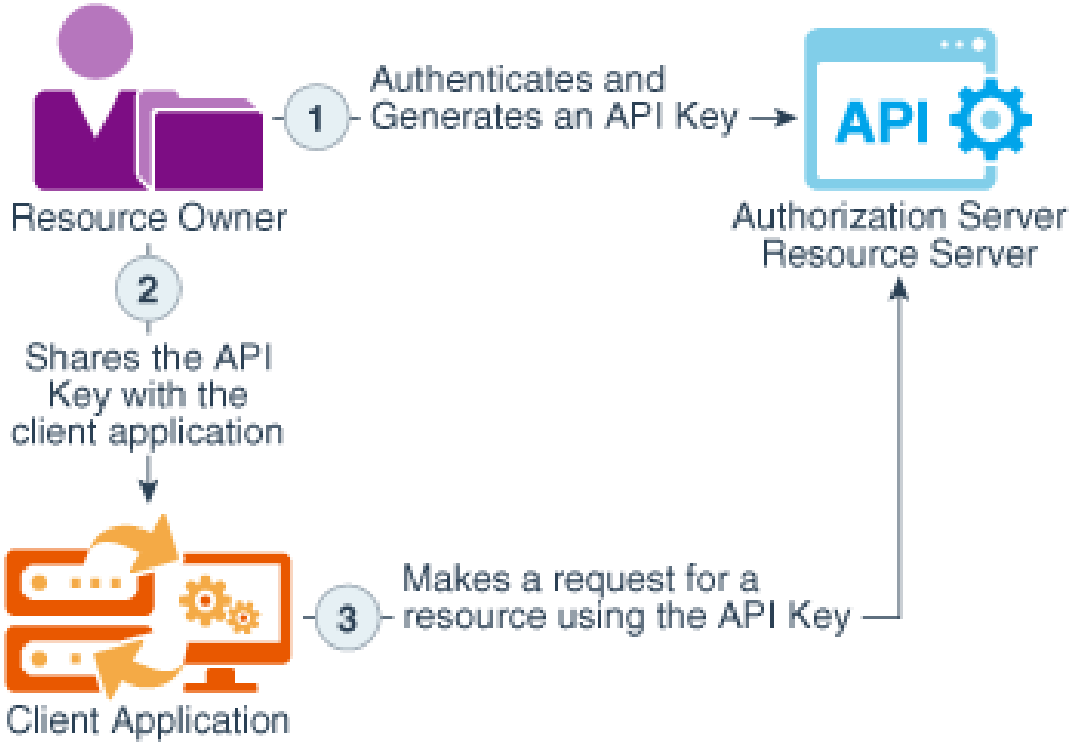
Ejemplo: Autentificación Bearer

- API Dummy JSON
  - Permite similar autentificación utilizando la URL https://dummyjson.com/auth/endpoint
  - Primero simulamos el login pasando los datos de usuario y obtenemos el token.
  - Almacenamos el token en localStorage.
  - Las siguientes peticiones las realizamos usando el token.

3. Autentificación basada en clave API

- Este tipo de autentificación requiere una configuración previa del acceso a los recursos de mi API
- La API genera una clave pública y una clave secreta para cada cliente que requiera acceso a los servicios
- Cada vez que una aplicación quera consumir los recursos de la API, debe enviar alguna de las dos
- Es un método más seguro, pero la generación e intercambio de keys debe ser manual
- Otro problema es como almacenar las claves.
- Una API Gateway facilita la gestión y mantenimiento de claves

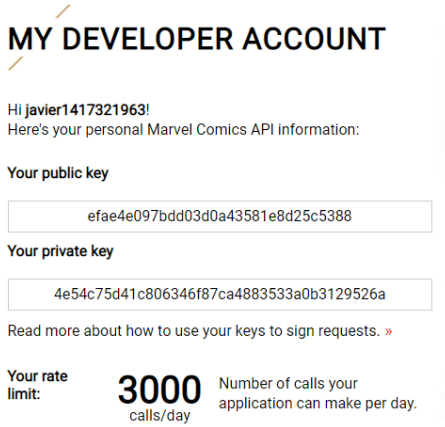
3. Autentificación basada en clave API

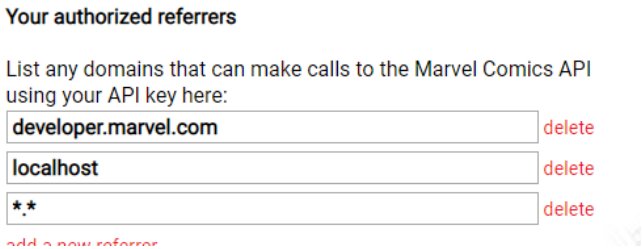


ejemplos/4_servicios_web/AppPersonajes

3. Autentificación basada en clave API

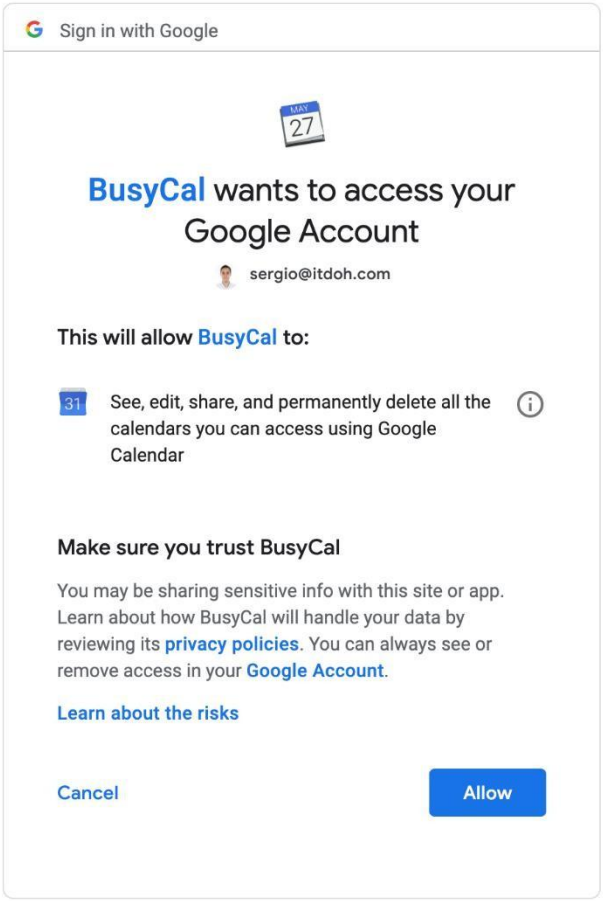
- Ejemplo: Marvel Developer
  "Get a Key" permite obtener una clave pública y privada (con límite de 3000 peticiones al día)
- Necesitamos especificar el dominio desde el cual vamos a realizar las peticiones:
  - *.* → Desde cualquier destino



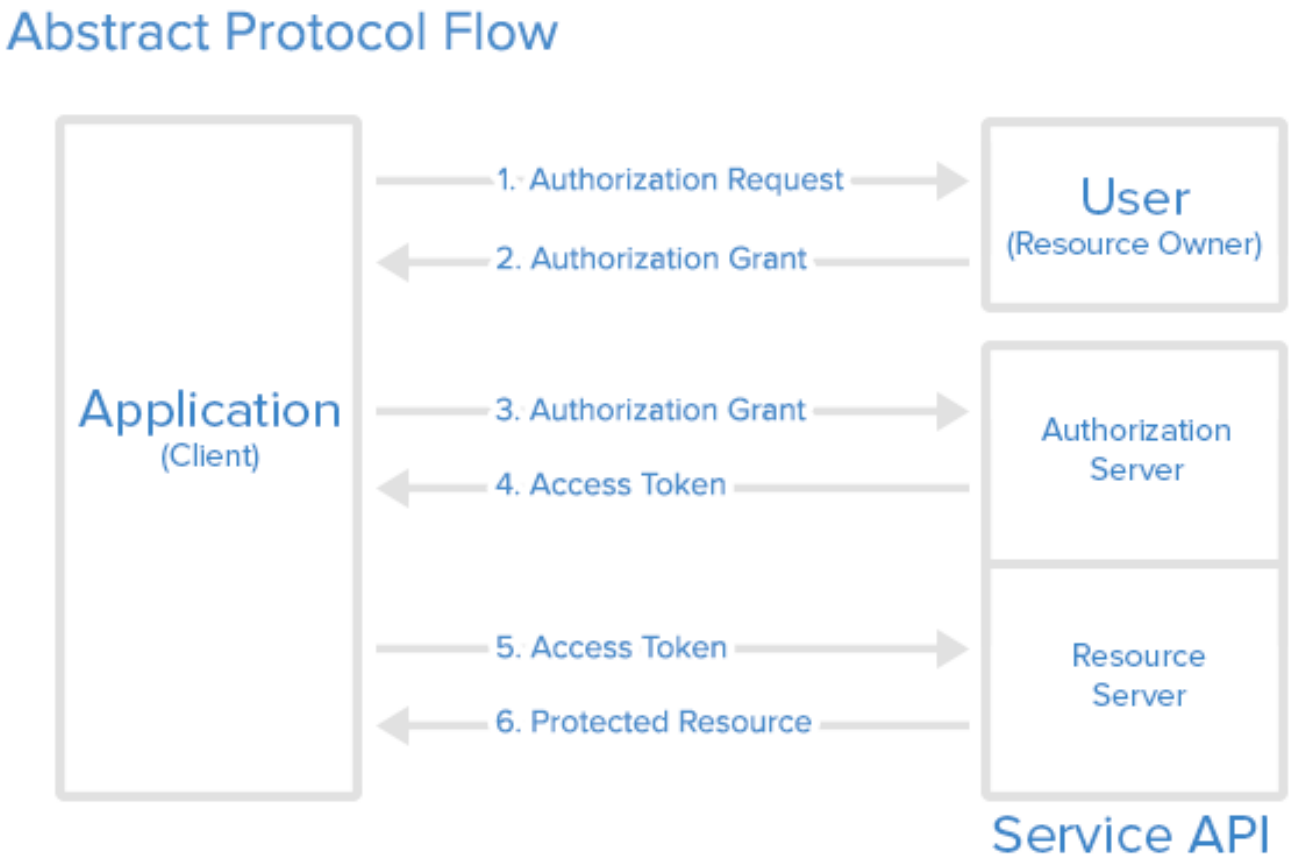


4. OAuth 2.0

- Es un método de autorización usado por compañías como Google, Twitter, Amazon, Facebook, etc..
- Permite a otros proveedores de servicios acceso a la información sin facilitar directamente las credenciales de los usuarios.



4. OAuth 2.0


Abstract Protocol Flow
Application
(Client)
1. Authorization Request
2. Authorization Grant
3. Authorization Grant
4. Access Token
5. Access Token
6. Protected Resource
User
(Resource Owner)
Authorization
Server
Resource
Server
Service API

Protegiendo rutas (parte cliente)

Servicios Web con Ionic

Guards

- Cuando tenemos páginas con una parte pública y una parte privada, necesitamos algún mecanismo para indicar de maneras sencilla a qué páginas puedo acceder y no.
- Los **Guards** son clases que actúan como middleware para proteger rutas, decidiendo si se puede acceder o no para acceder a ciertas páginas, así como permisos

Creación de un Guard

```
ionic generate guard guards/auth
```

Creamos un guard llamado **auth** dentro de la carpeta **/guards**

Creamos un guard del tipo **CanActivate**

Implementación de un Guard (v1)

Tenemos una página login que guarda un token en localStorage si el login es correcto

```ts
export const authGuard: CanActivateFn = (route, state) => {
 let token=localStorage.getItem('token');
 let router=new Router();

 if(token)
   return true;
 else{
   router.navigate(['/login']);
   return false;
 }
};
```

No inyectamos el servicio porque el Guard es una función, no una clase

En esta v1 Simplemente comprobamos si existe el token

auth.guard.ts

Protegiendo rutas

```ts
{
    path: 'citas',
    loadComponent: () => import('./citas/citas.page').then( m => m.CitasPage),
    canActivate: [authGuard]
  },
```

app.routes.ts

Implementación de un Guard (v2)

```
npm install jwt-decode
```

```ts
import {jwtDecode} from 'jwt-decode';

export const authGuard: CanActivateFn = (route, state) => {
  let token = localStorage.getItem('token');
  let router = new Router();
  if (token) {

    try {

      let payload: any = jwtDecode(token);

      let isExpired = payload.exp < Date.now() / 1000;

      if (!isExpired) {
        return true;

      }
    } catch (error) {
      console.error("ERROR" + error);

    }

  }

  router.navigate(['/login']);


  return false;
}
```

Decodificamos el token JWT y comprobamos su caducidad.
El token debería ser comprobado también desde servidor

`auth.guard.ts`

Tipos de Guards

Tipo	Descripción	Uso
`CanActivate`	Protege rutas para evitar accesos no autorizados	Protección de rutas privadas (páginas de administración, contenido solo para usuarios autenticados
`CanActivateChild`	Protege rutas hijas	Protege secciones enteras de la aplicación (subrutas)
`CanDeactivate`	Evita que el usuario salga sin guardar	Mostrar una advertencia antes de abandonar un formulario no guardado
`Resolve`	Resuelve datos antes de cargar la ruta	Cargar datos desde el back antes de mostrar una vista Evita que el usuario vea una interfaz vacía mientras la interfaz cambia

CanActivateChild

```ts
export const childGuard: CanActivateChildFn = (childRoute, state) => {
  const isAdmin = localStorage.getItem('role') === 'admin';
  return isAdmin;
};
```

child.guard.ts

```ts
const routes: Routes = [
  {
    path: 'admin',
    canActivateChild: [childGuard],
    children: [
      { path: 'dashboard', component: AdminDashboardComponent },
      { path: 'users', component: UsersComponent }
    ]
  }
];
```

app.routes.ts

CanDeactivate

```ts
export const formGuard: CanDeactivateFn<FormComponent> = (component) => {
  if (component.hasUnsavedChanges()) {
    return confirm('Tienes cambios sin guardar. ¿Seguro que deseas salir?');
  }
  return true;
};
```

formGuard.guard.ts

```ts
const routes: Routes = [
  { path: 'formulario', component: FormComponent, canDeactivate: [formGuard] }
];
```

app.routes.ts

Resolve

dataresolver.guard.ts

```ts
export const dataResolver: ResolveFn<any> = () => {
  return fetch('https://api.example.com/categories')
    .then((response) => response.json());
};
```

app.routes.ts

```ts
const routes: Routes = [
  {
    path: 'categorias',
    component: CategoriasComponent,
    resolve: { categories: dataResolver }
  }
];
```

categorias.component.ts

```ts
ngOnInit() {
  this.route.data.subscribe(data => {
    this.categories = data['categories'];
  });
}
```