Angular

Una plataforma open source para desarrollo de aplicaciones web

+Info: https://angular.io/docs

Comunicación entre componentes

Comunicación desde componente padre al hijo

En el componente hijo se utiliza el decorator @Input sobre un variable para indicar que el valor proviene desde fuera del componente.

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

@Component({
  selector: 'app-parent',
  template: `
    <app-child [childMessage]="parentMessage"></app-child>
  `,
  styleUrls: ['./parent.component.css']
})
export class ParentComponent{
  parentMessage = "message from parent"
  constructor() { }
}
```

```
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-child',
  template: `
      Message from parent: {{childMessage}}
  `,
  styleUrls: ['./child.component.css']
})
export class ChildComponent {

  @Input() childMessage: string;

  constructor() { }
```

Comunicación desde componente padre al hijo

Ejemplo online: *https://angular.io/generated/live-examples/component-interaction/stackblitz.html*

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

import { HEROES } from './hero';

@Component({
  selector: 'app-hero-parent',
  template: `
    <h2>{{master}} controls {{heroes.length}} heroes</h2>
    <app-hero-child *ngFor="let hero of heroes"
      [hero]="hero"
      [master]="master">
    </app-hero-child>
  `
})
export class HeroParentComponent {
  heroes = HEROES;
  master = 'Master';
}
```

```
import { Component, Input } from '@angular/core';

import { Hero } from './hero';

@Component({
  selector: 'app-hero-child',
  template: `
    <h3>{{hero.name}} says:</h3>
    <p>I, {{hero.name}}, am at your service, {{masterName}}.</p>
  `
})
export class HeroChildComponent {
  @Input() hero: Hero;
  @Input('master') masterName: string;
}
```

```
export class Hero {
  name: string;
}

export const HEROES = [
  {name: 'Mr. IQ'},
  {name: 'Magneta'},
  {name: 'Bombasto'}
];
```

Se puede pasar un objeto como parámetro

Master controls 3 heroes

Mr. IQ says:

I, Mr. IQ, am at your service, Master.

Magneta says:

I, Magneta, am at your service, Master.

Bombasto says:

I, Bombasto, am at your service, Master.

Comunicación desde el componente hijo al padre

Usando @ViewChild el padre tiene acceso a los datos del hijo

```
import { Component, ViewChild, AfterViewInit } from '@angular/core';
import { ChildComponent } from "../child/child.component";

@Component({
  selector: 'app-parent',
  template: `
    Message: {{message}}
    <app-child></app-child>
  `,
  styleUrls: ['./parent.component.css']
})
export class ParentComponent implements AfterViewInit {

  @ViewChild(ChildComponent) child;

  constructor() { }

  message:string;

  ngAfterViewInit() {
    this.message = this.child.message
  }
}
```

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

@Component({
  selector: 'app-child',
  template: `
  `,
  styleUrls: ['./child.component.css']
})
export class ChildComponent {

  message: string = "Hola Mundo!"

  constructor() { }

}
```

Se utiliza ***ngAfterViewInit()*** porque es necesario esperar a que la vista esté totalmente cargada para acceder a los atributos del hijo

A partir del template del padre se crea una instancia del componente hijo (por su invocación en el template). Luego con @ViewChild esa instancia es inyectada en la variable del padre. Con @ViewChild es posible también capturar objetos del DOM utilizando variables de referencia, ya sea referencias a tags HTML o asociados a otros componentes Angular.

Comunicación desde el componente hijo al padre

Se utiliza @Output cuando se desea informar al padre sobre cambios ocurridos en los datos del hijo

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

@Component({
  selector: 'app-parent',
  template: `
    Message: {{message}}
    <app-child (messageEvent)="receiveMessage($event)"></app-child>
  `,
  styleUrls: ['./parent.component.css']
})
export class ParentComponent {

  constructor() { }

  message:string;

  receiveMessage($event) {
    this.message = $event
  }
}
```

2

```
import { Component, Output, EventEmitter } from '@angular/core';

@Component({
  selector: 'app-child',
  template: `
      <button (click)="sendMessage()">Send Message</button>
  `,
  styleUrls: ['./child.component.css']
})
export class ChildComponent {

  message: string = "Hola Mundo!"

  @Output() messageEvent = new EventEmitter<string>();

  constructor() { }

  sendMessage() {
    this.messageEvent.emit(this.message)
  }
}
```

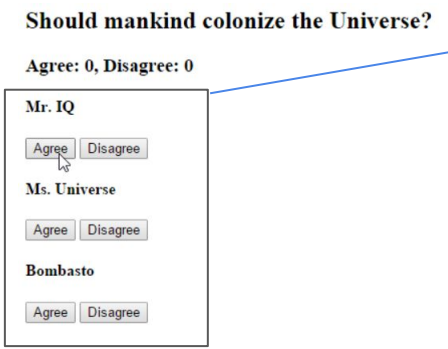
1

Cuando deseamos disparar el evento personalizado invocamos a *emit()* de **EventEmitter**.
Este método recibe como parámetro el dato que llegará al padre

Comunicación desde el componente hijo al padre

Ejemplo online: *https://angular.io/generated/live-examples/component-interaction/stackblitz.html*

Encuesta:




```ts
import { Component }      from '@angular/core';

@Component({
  selector: 'app-vote-taker',
  template: `
    <h2>Should mankind colonize the Universe?</h2>
    <h3>Agree: {{agreed}}, Disagree: {{disagreed}}</h3>
    <app-voter *ngFor="let voter of voters"
      [name]="voter"
      (voted)="onVoted($event)">
    </app-voter>
  `
})
export class VoteTakerComponent {
  agreed = 0;
  disagreed = 0;
  voters = ['Mr. IQ', 'Ms. Universe', 'Bombasto'];

  onVoted(agreed: boolean) {
    agreed ? this.agreed++ : this.disagreed++;
  }
}
```

3

```ts
import { Component, EventEmitter, Input, Output } from '@angular/core';

@Component({
  selector: 'app-voter',
  template: `
    <h4>{{name}}</h4>
    <button (click)="vote(true)"  [disabled]="didVote">Agree</button>
    <button (click)="vote(false)" [disabled]="didVote">Disagree</button>
  `
})
export class VoterComponent {
  @Input()  name: string;
  @Output() voted = new EventEmitter<boolean>();
  didVote = false;

  vote(agreed: boolean) {
    this.voted.emit(agreed);
    this.didVote = true;
  }
}
```

1

2

Cuando se crea un objeto **EventEmitter** siempre se define el tipo del evento que luego se va a generar, en este caso es un boolean.

Comunicación entre padre e hijo usando servicios

- Otra manera de comunicación entre componentes es utilizando servicios.
- Un componente padre y su hijo pueden compartir un servicio permitiendo una comunicación bidireccional.
- El alcance de la instancia del servicio es el componente padre y su/s hijo/s (ver providers).
- Los componentes fuera del alcance de esa instancia no tienen acceso al servicio ni a sus comunicaciones.

Ejemplo:
https://angular.io/guide/component-interaction#parent-and-children-communicate-via-a-service
https://angular.io/generated/live-examples/component-interaction/stackblitz.html

Ruteo avanzado - utilizando un servicio Guard

```
import { Routes, RouterModule } from '@angular/router';
import { HomePage } from './home-page';
import { AdminPage } from './admin-page';
import { AuthGuard } from './login-guard';

const routes: Routes = [
 { path: 'home', component: HomePage },
 { path: 'admin',component: AdminPage,canActivate:[AuthGuard]}
];

export const routing = RouterModule.forRoot(routes);
```

```
import { LoginService } from './login.service';
import { AuthGuard } from './login-guard';

@NgModule({
  ...
  providers: [
    LoginService,
    AuthGuard
  ]
  ...
})
export class AppModule {}
```

Ruteo avanzado - utilizando un servicio Guard

```
import { Injectable } from '@angular/core';
import { Router } from '@angular/router';
import { CanActivate } from '@angular/router';
import { LoginService } from './login.service';

@Injectable()
export class AuthGuard implements CanActivate {

    constructor(private authService: LoginService, private router: Router) {
}

    canActivate() {
        // If the user is not logged in we'll send them back to the home page
        if (!this.authService.isLogged()) {
            console.log('No estás logueado');
            this.router.navigate(['/']);
            return false;
        }
        return true;
    }
}
```

Ruteo avanzado - utilizando un servicio Guard

```
import { Injectable } from '@angular/core';
import { User } from './user.model';

@Injectable()
export class LoginService {

  private isUserLoggedIn;
  public userLogged:User;

  constructor() {
      this.isUserLoggedIn = false;
  }

  isLogged() {
    return this.isUserLoggedIn;
  }

  setUserLoggedIn(user:User) {
    this.isUserLoggedIn = true;
    this.userLogged = user;
    localStorage.setItem('currentUser', JSON.stringify(user));

  }

  getUserLoggedIn() {
      return JSON.parse(localStorage.getItem('currentUser'));
  }
  ...
}
```

AuthGuard utiliza el método isLogged() para verificar si hay un usuario logueado

Un componente que administre el login le indicará a este servicio el usuario logueado, mediante el método setUserLoggedIn()

Configuración de ambientes

Configuración de ambientes

Angular ofrece 2 maneras de compilación:

- *Just-in-Time (JIT)*: compila la aplicación en el navegador en **tiempo de ejecución**
- *Ahead-of-Time (AOT)*: compila la aplicación en **tiempo de desarrollo** (*build time*).

JIT es la compilación por defecto al ejecutar `ng build` ó `ng serve`

```
ng build
ng serve
```

Para compilar AOT se puede usar la opción `--aot`
O al usar `--prod` también compila por defecto con AOT

```
ng build --aot
ng serve --aot
```

Configuración de ambientes

Modo producción (`ng build --configuration=production`)

- *Ahead-of-Time (AOT) Compilation*: precompila los templates de los componentes
- *Production mode*: despliega la configuración de producción habilitando el modo producción
- *Bundling*: empaqueta varios archivos de librería en unos pocos
- *Minification*: remueve excesos de espacios en blanco y comentarios
- *Uglification*: reescribe el código para acortarlo, encripta variables y nombres de funciones
- *Dead code elimination*: elimina codigo no referenciado

Configuración de ambientes

Durante el diseño y desarrollo comúnmente se utiliza **`ng serve`** para construir y desplegar la aplicación rápidamente de manera local.

El comando **`ng serve`** construye la aplicación, levanta un servidor, despliega la aplicación y examina los cambios posteriores para redesplegarlos automáticamente, todo esto lo hace en memoria.

Cuando la aplicación está lista para el despliegue, se utiliza el comando **`ng build`**
Por defecto genera la salida en la carpeta **dist/**

Luego esa salida se puede desplegar en un server de producción.

Si se desea seguir viendo los cambios se podría usar **`ng build --watch`**

Configuración de ambientes

A partir de Angular 16+, se agregaron otras formas de renderizar aplicaciones además de CSR (Client-Side Rendering). Estas son SSR y SSG.

SSR (Server-Side Rendering)

- Significa que las páginas se renderizan **en el servidor** en tiempo real **cada vez que un usuario hace una solicitud**.
- El HTML que se envía al navegador ya viene renderizado, lo que **mejora el tiempo de carga inicial y el SEO** (optimización del sitio web para los motores de búsqueda).
- Requiere un **servidor Node.js** en producción para responder cada solicitud.
- Requiere más recursos de hardware en el servidor que una aplicación Angular tradicional

Configuración de ambientes

SSG (Static Site Generation o Prerendering)

- Las páginas se renderizan **en tiempo de compilación (build time)**, y el resultado es **HTML estático**.
- No se necesita un servidor Node.js, solo un servidor que sirva archivos estáticos (como Apache server).
- Ideal para sitios que no cambian mucho (blogs, documentación, landing pages).
- Tiene **mejor rendimiento** que SSR, pero no es adecuado si necesitas contenido dinámico en tiempo real.

Configuración de ambientes

Tipo de renderizado	*Descripción*	*Casos de uso*
CSR (Client-Side Rendering)	Angular se ejecuta enteramente en el navegador.	Apps interactivas SPA.
SSR (Server-Side Rendering)	El servidor genera HTML inicial con Angular (mediante `@angular/ssr` y Express).	Mejor rendimiento en primera carga, ideal para SEO.
SSG (Static Site Generation)	HTML pre-generado en tiempo de build y servido como archivos estáticos.	Ideal para blogs, landing pages, catálogos.

Configuración de ambientes

Podemos tener configurados varios ambientes

```
└─myProject/src/environments/
                    └─environment.ts
                    └─environment.prod.ts
                    └─environment.stage.ts
```

angular.json

```
"configurations": {
  "production": {
    "fileReplacements": [
      {
        "replace": "src/environments/environment.ts",
        "with": "src/environments/environment.prod.ts"
      }
    ],
    "optimization": true,
    "outputHashing": "all",
    "sourceMap": false,
    "extractCss": true,
    "namedChunks": false,
    "aot": true,
```

src/environments/environment.ts

```
export const environment = {
  production: false
};
```

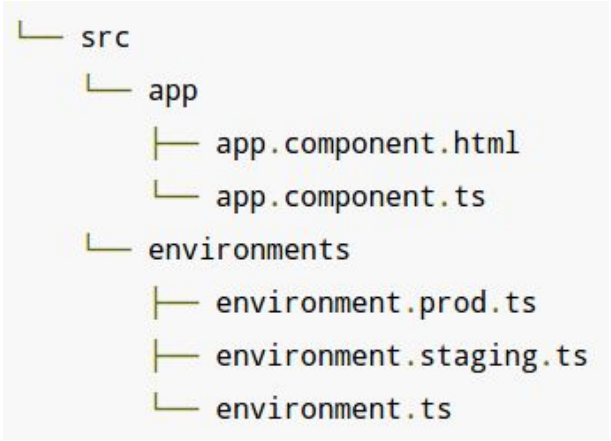
src/environments/environment.prod.ts

```
export const environment = {
  production: true
};
```

Cuando ejecutamos
`ng build --prod` (ó `ng build --configuration=production`)

El archivo **`src/environments/environment.ts`**
es reemplazado por **`src/environments/environment.prod.ts`**

Configuración de ambientes



src/environments/environment.ts

```
export const environment = {
  production: false,
  apiUrl: 'http://localhost:3000/rest'
};
```

src/environments/environment.prod.ts

```
export const environment = {
  production: true,
  apiUrl: 'http://www.rest.com/api'
};
```

Podemos disponer configuraciones personalizadas para cada ambiente

```
import { Component } from '@angular/core';
import { environment } from './../environments/environment';

@Component({
 selector: 'app-root',
 templateUrl: './app.component.html',
 styleUrls: ['./app.component.css']
})
export class AppComponent {
 constructor() {
   // Logs false for default environment
   console.log(environment.production);
   console.log(environment.apiUrl);
 }
 title = 'app works!';
}
```

Configuración de ambientes

Existen distintas opciones para conectar la aplicación Angular con la API REST:

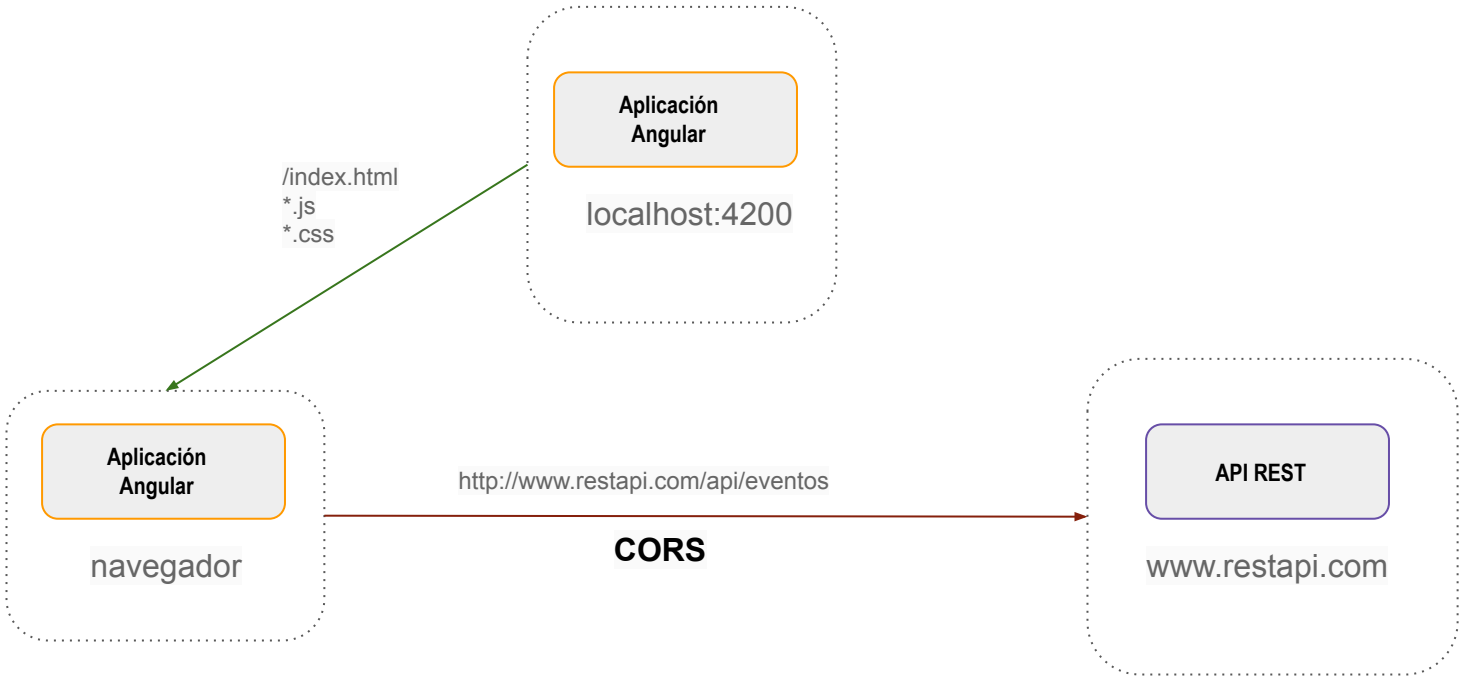
1. Utilizando server de Angular CLI u otro server para publicar la aplicación Angular

1. Utilizando el proxy de Angular CLI

1. Aplicación Angular en el mismo server que la API REST

Configuración de ambientes - opción 1

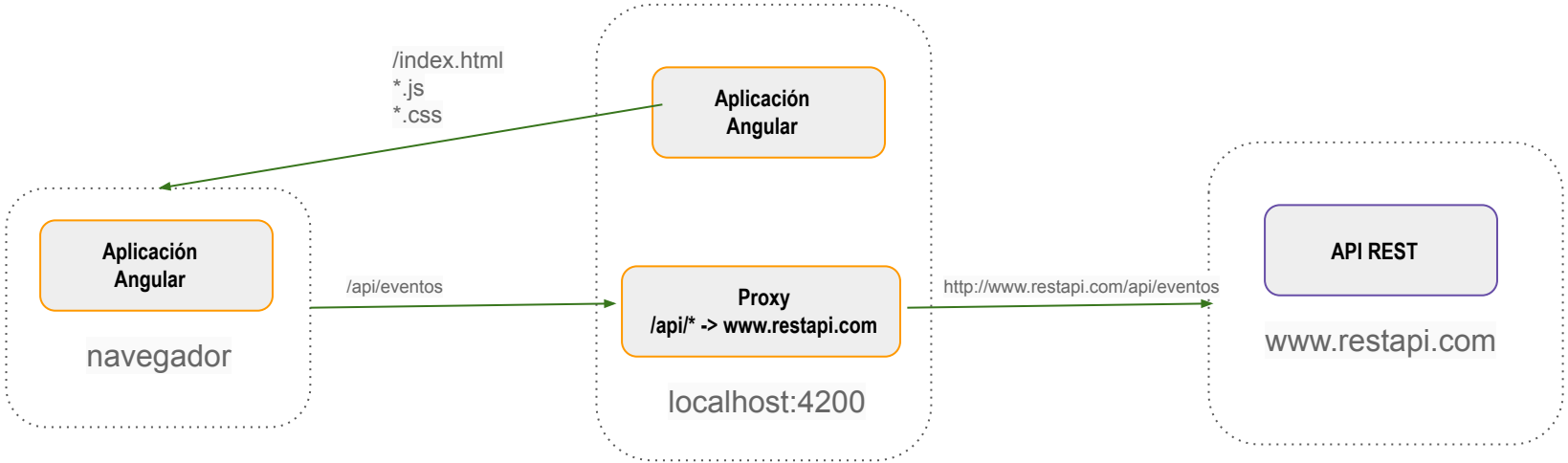
Utilizando server de Angular CLI u otro server para publicar la aplicación Angular



El navegador utiliza CORS

Configuración de ambientes - opción 2

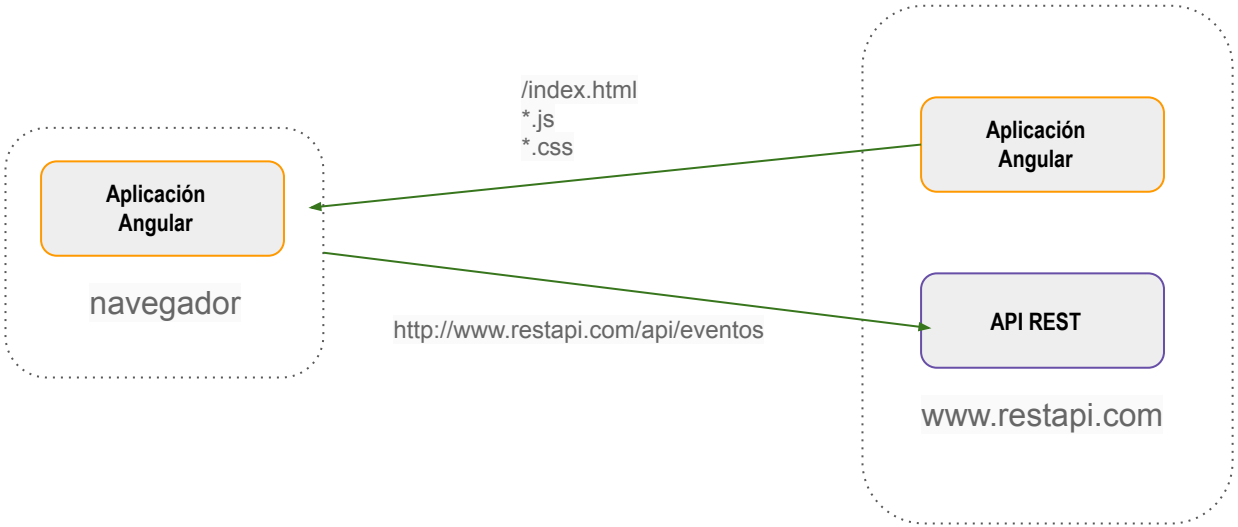
Utilizando el proxy de Angular CLI



El navegador no utiliza CORS

Configuración de ambientes - opción 3

Aplicación Angular en el mismo server que la API REST



El navegador no utiliza CORS

Utilizando el proxy de Angular CLI

```
jrosso@guarani6 /home/jorge/Descargas/CursoAngular/proyectoANuevo $ ng serve
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
 10% building 4/4 modules 0 active[HPM] Proxy created: /api  ->  http://5c70124569738000148aebb5.mockapi.io
[HPM] Subscribed to http-proxy events:  [ 'error', 'close' ]

Date: 2019-02-27T18:03:24.217Z
Hash: a82e9587ad814541e60e
Time: 7715ms
chunk {es2015-polyfills} es2015-polyfills.js, es2015-polyfills.js.map (es2015-polyfills) 284 kB [initial] [rendered]
chunk {main} main.js, main.js.map (main) 21.9 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 236 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.08 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 16.3 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 3.87 MB [initial] [rendered]
i [wdm]: Compiled successfully.
[HPM] GET /api/evento -> http://5c70124569738000148aebb5.mockapi.io
[HPM] GET /api/evento -> http://5c70124569738000148aebb5.mockapi.io
```

angular.json

```
...
 "serve": {
          "builder": "@angular-devkit/build-angular:dev-server",
          "options": {
            "browserTarget": "proyectoANuevo:build",
            "proxyConfig": "src/proxy.conf.json"
          },
          "configurations": {
            "production": {
              "browserTarget": "proyectoANuevo:build:production"
            }
          }
        }
```

src/proxy.conf.json

solo actúa con las urls que comienzan con /api

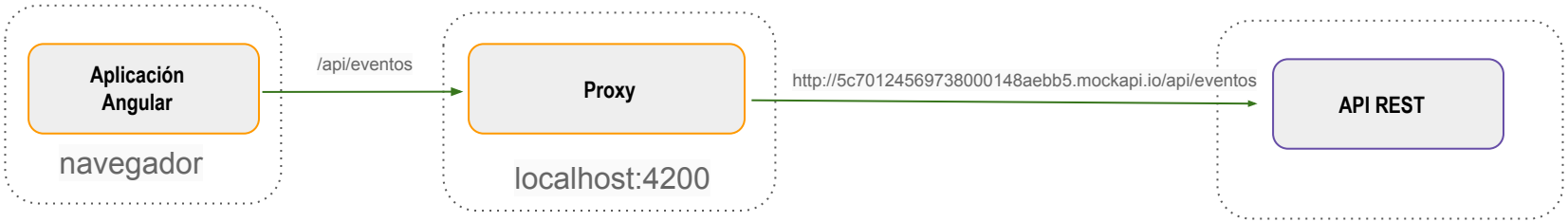
```
{
  "/api/*": {
    "target": "http://5c70124569738000148aebb5.mockapi.io",
    "secure": false,
    "logLevel": "debug",
    "changeOrigin": true,
  }
}
```

Utilizando el proxy de Angular CLI

```
@Injectable()
export class EventosServicio {

  constructor (private http: HttpClient){
  }

  getEventos (): Observable<Evento[]> {
    return this.http.get<Evento[]>("http://localhost:4200/api/eventos")
           .pipe(catchError((err: any) => Observable.of([])));
}
```



Uso de interceptores

Interceptores

- Los interceptores **inspeccionan y transforman peticiones HTTP** que van desde la aplicación al servidor, como también las respuestas que vuelven del servidor a la aplicación.
- Se pueden usar múltiples interceptores *formando una cadena*.
- Los interceptores pueden ejecutar variedad de tareas, como autenticación o logging.
- Sin interceptores los desarrolladores deberían implementar estas tareas explícitamente para cada llamada a `HttpClient`
- Para implementar un interceptor, se declara una clase que implementa el método `intercept()` de la interface `HttpInterceptor`

Interceptores

```
import { OnInit, Injectable } from "@angular/core";
import { Observable } from "rxjs/Observable";
import { HttpClient, HttpHeaders, HttpErrorResponse
} from "@angular/common/http";
import { environment
} from "../../../../environments/environment";
import { tap, catchError } from "rxjs/operators";
import { ErrorObservable
} from "rxjs/observable/ErrorObservable";
```

```
@Injectable()
export class TokenizerService {

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

```
getToken(): Observable<any> {
  const headers = new HttpHeaders({ "Content-Type": "application/json" });
  const url = `${environment.urlapi}`;
  return this.http
    .post(
      url,
      { userName: "username", password: "password" },
      { headers: headers }
    )
    .pipe(
      tap(data => console.log("Data: " + JSON.stringify(data))),
      catchError(this.handleError)
    );
}
```

la url es configurada por ambiente

Interceptores

```
import { HttpInterceptor, HttpSentEvent,
HttpHeaderResponse, HttpHandler, HttpEvent,
HttpRequest, HttpHeaders, HttpClient,
HttpErrorResponse } from "@angular/common/http";
import { Observable } from "rxjs/Observable";
import { Injectable } from "@angular/core";
import { environment
} from "../../../environments/environment";
import { tap, catchError } from "rxjs/operators";
import { ErrorObservable } from
"rxjs/observable/ErrorObservable";

@Injectable()
export class TokenInterceptor implements HttpInterceptor {

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

```
intercept(req: HttpRequest<any>, next: HttpHandler) {
  console.log(`TokenInterceptor - ${req.url}`);

  let authReq: HttpRequest<any> = req.clone({
    setHeaders:{
      Authorization : `Bearer ${localStorage.getItem("token")}`
    }
  });
  return next.handle(authReq);  //para no hacer cambios
                                //podría enviar next.handle(req)
 }
}
```

Interceptores

Los interceptores son dependencias opcionales del servicio HttpClient, deben estar declarados en el módulo donde se importa HttpClientModule

```
import { HttpClientModule, HTTP_INTERCEPTORS } from '@angular/common/http';
import { TokenInterceptor} from './interceptor.service'

@NgModule({
...
  imports: [
    ... ,
    HttpClientModule
  ],

  providers: [
    {provide:HTTP_INTERCEPTORS, useClass: TokenInterceptor, multi:true}
    //  mas interceptores
    // ,{provide:HTTP_INTERCEPTORS,useClass: OtroInterceptor, multi:true} ...
  ],...
```

Angular aplica los interceptores en el mismo orden que se declaran en el arreglo de providers

Interceptores

Si no estamos usando módulos la manera de configurar interceptores es usando `provideHttpClient`. El parámetro `withInterceptors` registra los interceptores que incluimos en el array.

src/app/app.config.ts

```
import { ApplicationConfig } from '@angular/core';
import { provideHttpClient, withInterceptors } from '@angular/common/http';
import { MyInterceptor } from './my-interceptor';

export const appConfig: ApplicationConfig = {
  providers: [
    provideHttpClient(withInterceptors([TokenInterceptor])),
  ]
};
```

src/main.ts

```
...
bootstrapApplication(HeroesComponent, appConfig)
  .catch(err => console.error(err));
```

Interceptores

Uso de signals

Signals

Un **signal** es un wrapper sobre un valor y notifica a quienes lo utilizan cuando este cambia. Las *signals* pueden contener cualquier valor, desde tipos primitivos hasta estructuras de datos complejas.

```
import { signal } from '@angular/core';
const count = signal(0);
```

..es un ***WritableSignal*** automáticamente de tipo number

```
// Leer valor
console.log(count());  //se utiliza el getter, x ej en la vista también  <p>{{ count() }}</p>

// Actualizar valor
count.set(1);

// Increment the count by 1
count.update(value => value + 1);
```

Los **signals** permiten que Angular rastree los cambios en los datos de forma más precisa, actualizando solo los elementos del template que realmente han cambiado, en vez de renderizar toda la vista.

Signals

Los **computed signal** son de ***solo lectura*** y obtienen su valor a partir de otros signals. Se pueden definir utilizando la función computed y especificando una derivación:

```
import { signal } from '@angular/core';
const count = signal(0);
…

// Defino un computed signal
const doubleCount = computed(() => count() * 2);

doubleCount.set(3); //produce un error de compilación
```

La función de derivación de doubleCount es ***lazy*** y ***memorized***, es decir que no se ejecuta para calcular su valor hasta la primera vez que se lee y el valor calculado se guarda en caché. Si se lee doubleCount de nuevo, devolverá el valor en caché sin volver a calcularlo.

Si cambia count, Angular sabe que el valor en caché de doubleCount ya no es válido, y la próxima vez que se lea doubleCount se calculará su nuevo valor.

Signals - Effects

effects es una operación que se ejecuta cada vez que uno o más valores de una señal cambian. Se crea con la función effect:

```
effect(() => {
      console.log(`The current count is: ${count()}`);
});
```

Los ***effects*** siempre se ejecutan al menos una vez y cada vez que los valores del signal cambian se ejecutan de nuevo. En este sentido es similar a los computed signals.

Los ***effects*** rara vez son necesarios en una aplicación, pero pueden ser útiles en circunstancias específicas.

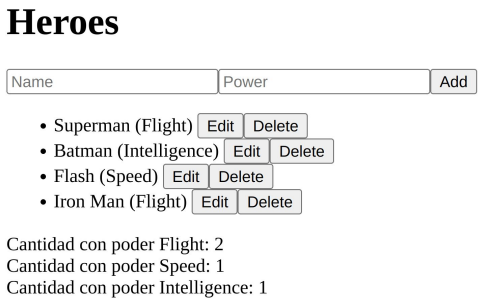
Solo se puede crear un effect() dentro de un contexto de inyección (donde tienes acceso a la función inject). Por ejemplo dentro de un constructor de componente, directiva o servicio.

CRUD con signals

/src/app/heroes.component.ts

```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
import { HeroesService, Hero } from './heroes.service';

@Component({
  selector: 'app-heros',
  standalone: true,
  imports: [CommonModule, FormsModule],
  template: `<h1>Heroes</h1>  <div *ngIf="loading()">Cargando héroes...</div>
    <form (ngSubmit)="save()">
      <input [value]="name()" (input)="name.set($any($event.target).value)" name="name" placeholder="Name" required />
      <input [value]="power()" (input)="power.set($any($event.target).value)" name="power" placeholder="Power" required />
      <button type="submit">{{ editingId() ? 'Update' : 'Add' }}</button>
      <button type="button" *ngIf="editingId()" (click)="cancel()">Cancel</button>
    </form>
    <div *ngIf="errorMessage()" style="color: red">{{ errorMessage() }}</div>
    <ul> <li *ngFor="let hero of heroes()">
            {{ hero.name }} ({{ hero.power }})
            <button (click)="edit(hero)">Edit</button>
            <button (click)="remove(hero.id)">Delete</button>
    </li> </ul>
    <div *ngIf="countByPower()['Flight']">Cantidad con poder Flight: {{ countByPower()['Flight'] }}</div>
    <div *ngIf="countByPower()['Speed']">Cantidad con poder Speed: {{ countByPower()['Speed'] }}</div>
    <div *ngIf="countByPower()['Intelligence']">Cantidad con poder Intelligence: {{ countByPower()['Intelligence'] }}</div>
`})
export class HerosComponent {
    heroes = signal<Hero[]>([]); // 1) fuente reactiva del estado
    loading = signal(true);      // 2) loading controlado por signal
    errorMessage = signal('');
```



CRUD con Signals

/src/app/heroes.component.ts

```
  name = signal('');
  power = signal('');
  editingId = signal<number | null>(null);

  countByPower = computed(() => {
    const counts: { [key: string]: number } = {};
    for (const h of this.heroes()) {
      counts[h.power] = (counts[h.power] || 0) + 1;
    }
    return counts;
  });

  constructor(private service: HeroesService) {
    this.loadHeroes();
  }

 loadHeroes() {
  this.loading.set(true);
  this.service.getHeroes().subscribe({
    next: data => {
      this.heroes.set(data); // heroes es un signal
      this.errorMessage.set('');
      this.loading.set(false);
    },
    error: () => {
      this.errorMessage.set('Error al cargar los héroes');
      this.loading.set(false);
    }
  });
}
```

```
save() {
  const heroData: Partial<Hero> = {
    name: this.name(),
    power: this.power()
  };

  const request = this.editingId()
    ? this.service.updateHero({ id: this.editingId()!, ...heroData } as Hero)
    : this.service.addHero(heroData);

  request.subscribe({
    next: () => {
      this.loadHeroes();
      this.resetForm();
    },
    error: () => this.errorMessage.set('Error al guardar el héroe')
  });
}

edit(hero: Hero) {
  this.name.set(hero.name);
  this.power.set(hero.power);
  this.editingId.set(hero.id);
}
```

CRUD con Signals

/src/app/heroes.component.ts

```
  remove(id: number) {
    this.service.deleteHero(id).subscribe({
      next: () => this.loadHeroes(),
      error: () => this.errorMessage.set('Error al eliminar el héroe')
    });
  }

  cancel() {
    this.resetForm();
  }

  private resetForm() {
    this.name.set('');
    this.power.set('');
    this.editingId.set(null);
  }
}
```

CRUD con Signals

/src/app/heroes.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable, of, throwError } from 'rxjs';
import { catchError } from 'rxjs/operators';

export interface Hero {id: number; name: string; power: string;}

@Injectable({ providedIn: 'root' })
export class HeroesService {
  private apiUrl = 'api/heroes';
  constructor(private http: HttpClient) {}

  getHeroes(): Observable<Hero[]> {
    return this.http.get<Hero[]>(this.apiUrl).pipe(
      catchError(error => {
        console.error('Error fetching heroes:', error);
        return of([]); // no propaga el error
        //return throwError(() => error); // propaga el error
      })
    );
  }

  getHero(id: number): Observable<Hero> {
    return this.http.get<Hero>(`${this.apiUrl}/${id}`).pipe(
      catchError(error => {
        console.error('Error fetching hero:', error);
        return throwError(() => error);
      })
    );
  }

  addHero(hero: Partial<Hero>): Observable<Hero> {
    return this.http.post<Hero>(this.apiUrl, hero).pipe(
      catchError(error => {
        console.error('Error adding hero:', error);
        return throwError(() => error);
      })
    );
  }

  updateHero(hero: Hero): Observable<Hero> {
    return this.http.put<Hero>(`${this.apiUrl}/${hero.id}`, hero).pipe(
      catchError(error => {
        console.error('Error updating hero:', error);
        return throwError(() => error);
      })
    );
  }

  deleteHero(id: number): Observable<void> {
    return this.http.delete<void>(`${this.apiUrl}/${id}`).pipe(
      catchError(error => {
        console.error('Error deleting hero:', error);
        return throwError(() => error);
      })
    );
  }
}
```