

# ANGULAR: Programación reactiva desde el minuto 0

Fernando San Segundo Alonso

I.E.S. María de Zayas y Sotomayor

Desarrollo de Aplicaciones Web

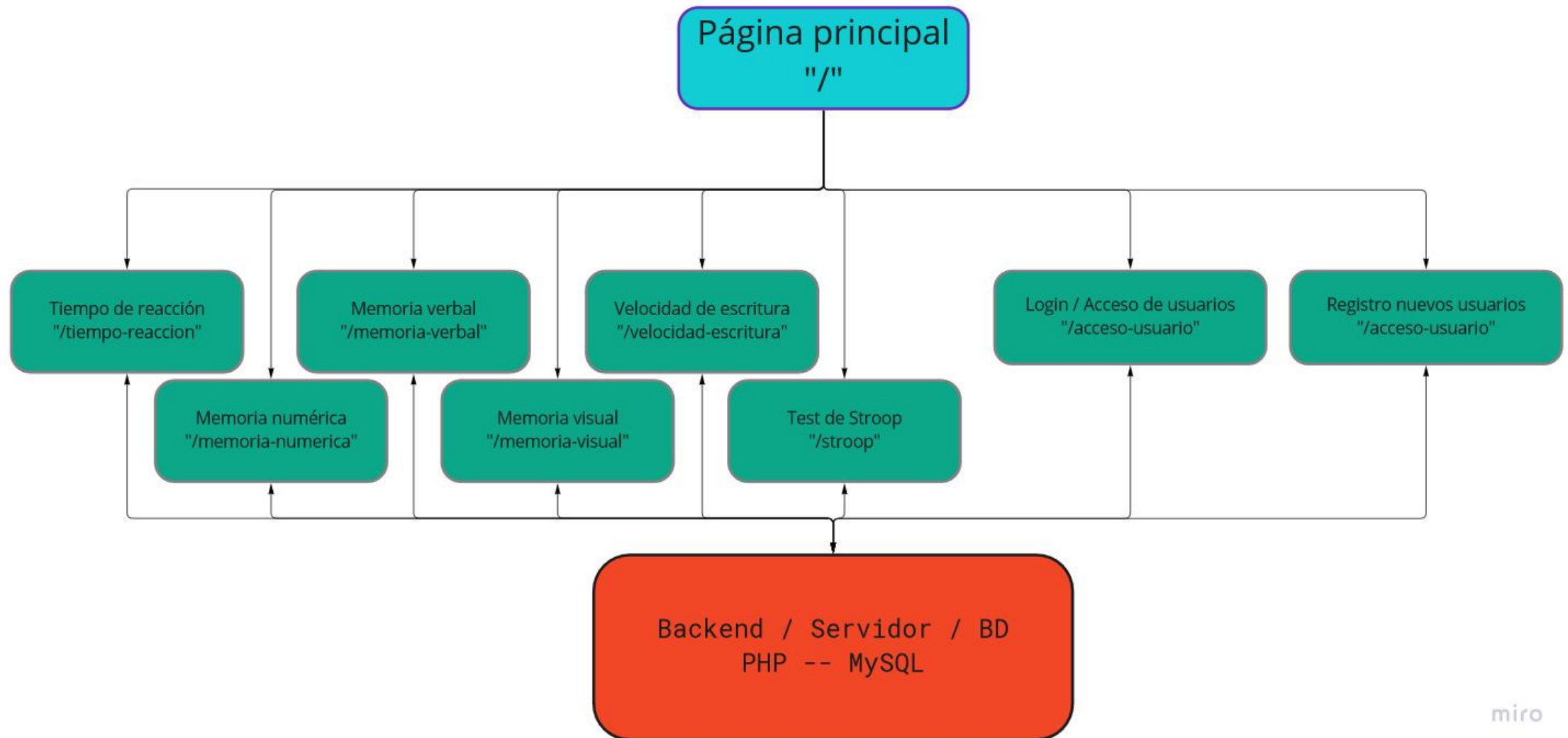
# Conceptos: Angular

- Framework para desarrollo web.
- Trabaja con TypeScript.
- Uno de los “tres grandes” del desarrollo front end.
- Desarrollo basado en componentes: reusabilidad y escalabilidad.
- Uso fundamental y extensivo de conceptos de programación reactiva.

## Conceptos: Programación reactiva (RxJS)

- Paradigma de desarrollo basado en asincronía.
- Flujos (streams) de datos para manejar cambios, reaccionando a las acciones del usuario.
- Flujos constantes o finitos, siempre asíncronos.
- RxJS = “Reactive extensions for JavaScript”.
- Cimiento la base sobre la que se sustenta Angular (observables).

# Mapa conceptual: Layout de la aplicación



```
<div class="TwoRowsTextblock">
  <span class="TwoRowsTextblock__first">{{testName}}</span>
  <span class="TwoRowsTextblock__second">{{testScore}}</span>
</div>
```

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

@Component({
  selector: 'app-single-test-global-results',
  templateUrl: './single-test-global-results.component.html',
  styleUrls: ['./single-test-global-results.component.scss']
})
export class SingleTestGlobalResultsComponent implements OnInit {
  @Input() testName!: string;
  @Input() testScore?: string;

  constructor() { }

  ngOnInit(): void {
  }
}
```

```
.TwoRowsTextblock {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  text-align: center;
  width: 100%;
  height: 100%;
  // background-color: aqua;

  &__first {
    font-weight: bold;
    font-size: 2rem;
  }

  &__second {
    font-size: 1.8rem;
  }
}
```

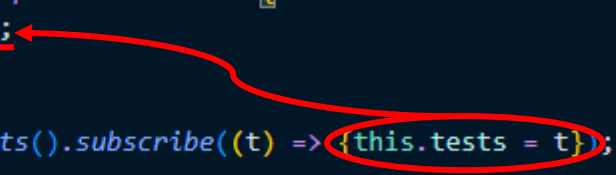
# Angular: Anatomía de un componente.

# Enlazar la vista y el modelo

- Sintaxis específica en el HTML de un componente.
- Permite presentar información dinámicamente (enlazada a datos)

*home.component.ts*

```
export class HomeComponent implements OnInit {  
  public tests: ITest[] = [];  
  
  ngOnInit(): void {  
    this.testsService.getTests().subscribe((t) => {this.tests = t});  
  }  
}
```



*home.component.html*

```
<div class="main">  
  <app-test-box  
    *ngFor="let test of tests"  
    [test]="test"  
    (click)="onClick(test.title)"  
  ></app-test-box>  
</div>  
<app-footer></app-footer>
```

# Observables: los cimientos de una aplicación web en Angular

- No son conceptos únicos de Angular. Es la base de la programación reactiva.
- Angular implementa la librería reactiva RxJS por defecto.
  1. Se crea un flujo de datos asíncrono, que emite los datos cuando se soliciten (observable)
  2. Se conecta a la observable, que emite los datos (suscripción)
  3. A la suscripción se le pasa un objeto que define cómo se va a tratar la información recibida.

*tests.service.ts*

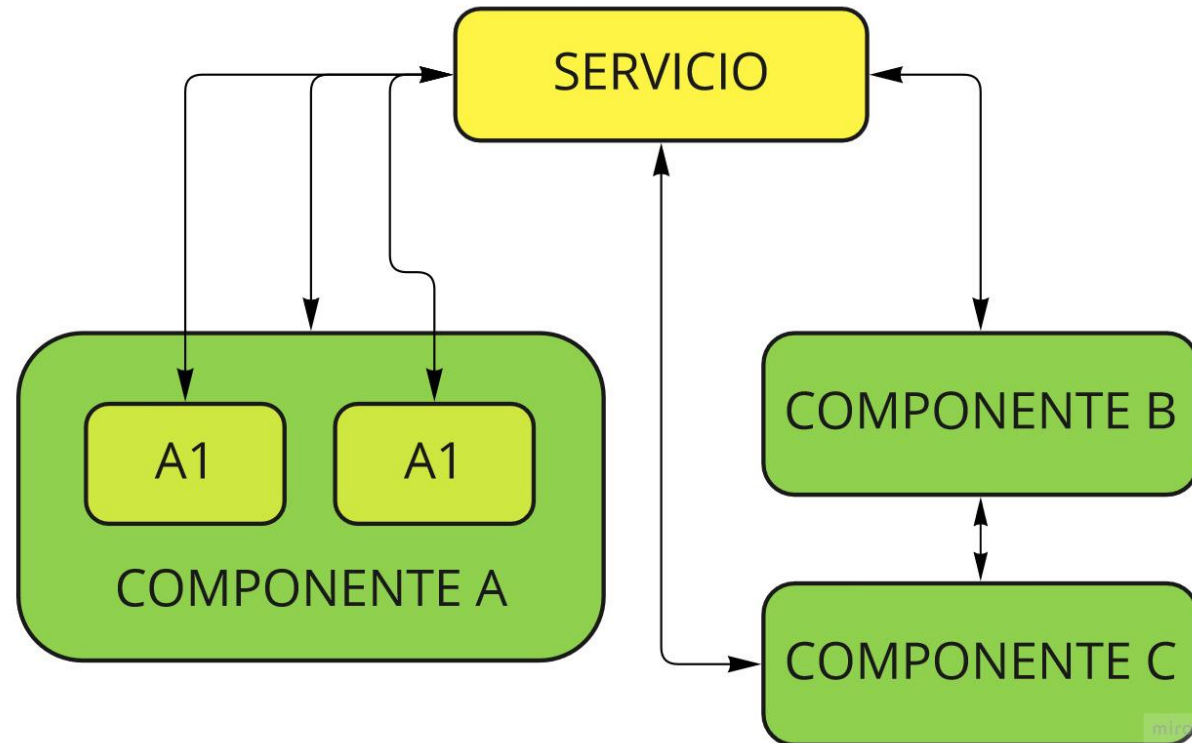
```
public getTests(): Observable<ITest[]> {  
  if(!this.cacheTests) {  
    this.cacheTests = this.http.get<ITest[]>(`${this.testsEndpoint}/get.php`).pipe(  
      map((res:ITest[]) => {  
        return res;  
      } ),  
      shareReplay(1)  
    );  
  }  
  return this.cacheTests;  
}
```

*home.component.ts*

```
ngOnInit(): void {  
  this.testsService.getTests().subscribe((t) => {this.tests = t});  
}
```

# Comunicación entre componentes: Servicios

- Son clases genéricas con un propósito concreto y específico.
- Múltiples usos (manejar datos del servidor, validar inputs...).
- Comunicación fácil entre componentes no interrelacionados:  
Inyección de dependencias (DI).





*tiempo-reaccion.component.ts*

```
public generateResults(e: any):void {
  this.results.responseTimes += e.responseTimes;
  this.results.averageTimes += e.averageTimes+" ms";
  this.results.comparativeResults = e.comparativeResults;

  this.testResult = {...this.testResult, score: e.averageTimes+"ms"};
  this.testsService.updateResult(this.testResult).subscribe(r => this.eventsService.ResultsSaved.emit());
}
```

*global-results.component.ts*

*events.service.ts*

```
export class EventsService {
  constructor() { }

  ResultsSaved = new EventEmitter();
}
```

```
export class GlobalResultsComponent implements OnInit {
  public results$!:Observable<IGlobalResults[]>;

  constructor(private testsService:TestsService, private eventsService:EventsService) { }

  ngOnInit(): void {
    this.loadGlobalResults();
    this.eventsService.ResultsSaved.subscribe(() => this.loadGlobalResults())
  }

  private loadGlobalResults(): void {
    this.results$ = this.testsService.getAllGlobalResults();
  }
}
```

# Ejemplo práctico: Servicio para eventos

# Interceptores

HttpInterceptor actúa como middleware entre la emisión y recepción de llamadas http.

Permite capturar y tratar excepciones, aplicar lógica de negocio, añadir datos a las cabeceras http...

Paradigma puramente reactivo. La petición interceptada y transformada se devuelve como observable.

```
export class AuthInterceptorService implements HttpInterceptor {
  private refreshing: boolean = false;
  private refreshTokenSubject: BehaviorSubject<any> = new BehaviorSubject<any>(null);

  constructor(private authService: UserAuthService) { }

  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    let authRequest = req;
    const sessionToken = this.authService.getSessionToken();

    if (sessionToken && Object.keys(sessionToken).length !== 0) {
      authRequest = this.addTokenToHeaders(req, sessionToken); // Añadir token a headers
    }

    return next.handle(authRequest).pipe(
      catchError(err => {
        if (err instanceof HttpResponse && err.status === 401) {
          return this.handle401error(authRequest, next); // Manejar error 401
        }
        return throwError(() => err);
      })
    )
  }
}
```

```
private addTokenToHeaders(req: HttpRequest<any>, token: IAuthToken) {
  return req.clone({
    headers: req.headers.set("Authorization", "Bearer "+token.token)
  });
}
```

# Interceptores (cont.)

```
private handle401error(req:HttpRequest<any>, next:HttpHandler) {
  if (!this.refreshing) {
    this.refreshing = true; // Bloquea nuevos errores derivados de nuevas peticiones interceptadas
    this.refreshTokenSubject.next(null);

    const token = this.authService.getSessionToken();

    if (token) {
      return this.authService.refreshSessionToken(token.username).pipe( // Solicita un nuevo token de sesión
        switchMap((token:any) => {
          this.refreshing = false;

          // Guarda el nuevo token al recibirlo
          localStorage.setItem("loggedInUser", JSON.stringify(token));
          this.refreshTokenSubject.next(token);

          // Añade el nuevo token a headers y permite seguir con la petición
          return next.handle(this.addTokenToHeaders(req, token));
        }),
        catchError(err => {
          this.refreshing = false;
          return throwError(() => err);
        })
      );
    }
  }
  return this.refreshTokenSubject.pipe(
    filter(token => token !== null),
    take(1),
    switchMap((token:any) => next.handle(this.addTokenToHeaders(req, token)))
  );
}
```

# Ejemplo práctico: controlar sesión expirada

1. Se produce una petición http

```
public getAllGlobalResults(): Observable<IGlobalResults[]> {  
    return this.http.get<IGlobalResults[]>(`${this.resultsEndpoint}/get.php`).pipe(  
        map((res: IGlobalResults[]) => res)  
    );  
}
```

2. La petición es interceptada por el interceptor http

```
intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> { ...  
}
```

3. El interceptor añade el token de sesión a la cabecera, y la petición continúa

```
private addTokenToHeaders(req: HttpRequest<any>, token: IAuthToken) {  
    return req.clone({  
        headers: req.headers.set("Authorization", "Bearer "+token.token)  
    });  
}
```

4. La petición llega al servidor  
y éste decodifica el token  
para comprobar su expiración

```
<?php  
include_once '../tokenHandler.php';  
use Firebase\JWT\JWT;  
use Firebase\JWT\Key;  
  
$headers = getallheaders();  
  
if (isset(getallheaders()['Authorization'])) {  
    $token = explode(" ", getallheaders()['Authorization'])[1]; // Gets token from header  
  
    try {  
        JWT::$leeway = 10;  
        $decoded = JWT::decode($token, new Key(SECRET_KEY, 'HS256'));  
        return http_response_code(200);  
    } catch (Exception $e) {  
        return http_response_code(401);  
    }  
}  
}
```

# Ejemplo práctico: controlar sesión expirada (cont.)

5. El servidor devuelve un código de error 401 (nueva petición http) y el interceptor lo captura

```
private handle401error(req:HttpRequest<any>, next:HttpHandler) { ...  
}
```

6. El interceptor pide un nuevo token

```
public refreshSessionToken(username:string = ""):Observable<IAuthToken> {  
    return this.http.post<IAuthToken>(`${this.authEndpoint}/refresh.php`, {action:"refresh", username:username}, httpOptions).pipe(  
        tap(res => {  
            if (res.state == "success")  
                console.log("NUEVO TOKEN RECIBIDO CON ÉXITO");  
        })  
    )  
}
```

7. El servidor genera el nuevo token y lo envía de vuelta

```
<?php  
require '../vendor/autoload.php';  
require_once '../connection.php';  
require '../headers.php';  
include '../tokenHandler.php';  
  
$dbConnection = connect();  
  
$input = json_decode(file_get_contents("php://input"));  
$username = isset($input->username) ? $input->username : "";  
  
$jwt = createJwtToken();  
  
echo json_encode([  
    "state" => "success",  
    "token" => $jwt,  
    "type" => "login",  
    "username" => $username,  
    "additionalInfo" => ""  
]);  
?>
```

```
function createJwtToken() {  
    return createToken(600);  
}  
  
function createToken($duration) {  
    $factory = new RandomLib\Factory();  
    $generator = $factory->getMediumStrengthGenerator();  
    $randomString = $generator->generateString(32);  
  
    $issuer = "localhost"; // The entity that issued the token (the website)  
    $audience = "loggedInUser"; // The recipient that will consume the token (the user)  
    $issuedAt = time(); // The time (seconds since Unix epoch) when the token was issued  
    $notBefore = $issuedAt-10; // The time after which the token can be accepted  
    $expiresAt = $issuedAt + $duration; // The time at which the token will expire  
  
    $payload = [ // The body of the JWT token  
        'iss' => $issuer,  
        'aud' => $audience,  
        'iat' => $issuedAt,  
        'nbf' => $notBefore,  
        'exp' => $expiresAt,  
        'data' => [  
            'token' => $randomString  
        ]  
    ];  
  
    return JWT::encode($payload, SECRET_KEY, 'HS256');  
}
```

## Ejemplo práctico: controlar sesión expirada (cont.)

8. El interceptor recibe el nuevo token, lo guarda y lo añade a las cabeceras de la petición original, que finalmente puede llevarse a cabo por completo

```
return this.authService.refreshSessionToken(token.username).pipe(  
  switchMap((token:any) => {  
    this.refreshing = false;  
  
    localStorage.setItem("loggedInUser", JSON.stringify(token));  
    this.refreshTokenSubject.next(token);  
  
    return next.handle(this.addTokenToHeaders(req, token));  
  }  
),  
  catchError(err => { ...  
})  
)
```

Nuevas peticiones serán interceptadas y el token de sesión será comprobado. Si expira, este proceso se repetirá de nuevo.



- Cierta transferencia de conocimientos desde otros frameworks de JS (React).
- TypeScript > JavaScript.
- Estructurado y escalable.
- Reducida dependencia de librerías externas.
- Inherentemente reactivo.
- Excelente y extensiva documentación.



- Curva de aprendizaje elevada.
- La programación reactiva no es un concepto fácil de visualizar y entender.
- Puede resultar excesivamente restrictivo.
- Demasiado complejo para desarrollar aplicaciones simples.

## Angular: Conclusiones

# Bibliografía

- Documentación oficial de Angular: <https://angular.io/docs>
- Documentación oficial de RxJS: <https://rxjs.dev/>
- Librería PHP para generar tokens JWT:  
<https://github.com/firebase/php-jwt>
- Librería PHP para generar strings aleatorias puras:  
<https://github.com/paragonie/RandomLib>
- Consulta de dudas puntuales: <https://stackoverflow.com/>
- Dudas puntuales y tutoriales varios: <https://medium.com/>
- Guía de ayuda para Interceptor HTTP:  
<https://www.youtube.com/watch?v=F1GUjHPpCLA>