

Creación de una aplicación meteorológica PWA con Angular Service Worker

Unidad 4

- Podemos acceder a una aplicación web utilizando diferentes tipos de dispositivos, como escritorio, móvil, tableta y varios tipos de redes. Una aplicación web debe funcionar sin problemas y proporcionar la misma experiencia de usuario independientemente del dispositivo y la red del usuario.
- Progressive Web Apps (PWA) es una colección de técnicas web para crear aplicaciones web con consideraciones previas en mente. Una técnica popular es el trabajador de servicios, que mejora el tiempo de carga de una aplicación web. En este capítulo, usaremos la implementación del service worker de Angular para construir una PWA que muestre el clima de una ciudad usando la API OpenWeather.

- Cubriremos los siguientes temas en detalle:
 - Configuración de la API de OpenWeather
 - Visualización de datos meteorológicos
 - Habilidad del modo fuera de línea con el service worker
 - Mantenerse al día con las notificaciones en la aplicación.
 - Publicar nuestra aplicación con Firebase hosting



Teoría y contexto esenciales

- Las aplicaciones web tradicionales suelen estar alojadas en un servidor web y están disponibles de inmediato para cualquier usuario en cualquier momento. Las aplicaciones nativas se instalan en el dispositivo del usuario, tienen acceso a sus recursos nativos y pueden funcionar sin problemas con cualquier red.
- Las aplicaciones PWA se encuentran entre los dos mundos de las aplicaciones web y nativas y comparten características de ambos. Una aplicación PWA es una aplicación web que se basa en los siguientes pilares para convertirla en nativa.

- Capaz: puede acceder a los datos guardados localmente e interactuar con el hardware periférico que está conectado al dispositivo del usuario.
- Confiable: puede tener el mismo rendimiento y experiencia en cualquier conexión de red, incluso en áreas con baja conectividad y cobertura.
- Instalable: se puede instalar en el dispositivo del usuario, se puede iniciar directamente desde la pantalla de inicio e interactuar con otras aplicaciones nativas instaladas.

- La conversión de una aplicación web en una PWA implica varios pasos y técnicas. El más esencial es configurar un service worker. El service worker es un mecanismo que se ejecuta en el navegador web y actúa como un proxy entre la aplicación y un dispositivo externo.
- End point HTTP u otros recursos de la aplicación, como archivos JavaScript y CSS. El trabajo principal del service worker es interceptar las solicitudes a esos recursos y actuar en consecuencia proporcionando una respuesta en caché o en vivo.

- Angular proporciona una implementación para el service worker que podemos usar para convertir nuestras aplicaciones Angular en PWA.
- También contiene un cliente HTTP integrado que podemos usar para comunicarnos con un servidor a través de HTTP. El cliente HTTP Angular expone una API basada en observables con todos los métodos HTTP estándar, como POST y GET.

- Los observables se basan en el patrón del observador, que es el núcleo de la programación funcional reactiva. En el patrón del observador, varios objetos llamados observadores pueden suscribirse a un observable y recibir notificaciones sobre cualquier cambio en su estado.
- Un observable envía cambios a los observadores mediante la emisión de flujos de eventos de forma asíncrona. Angular usa una biblioteca llamada RxJS que contiene varios elementos para trabajar con observables. Uno de estos elementos es un conjunto de funciones llamadas operadores que pueden aplicar varias acciones en observables como transformaciones y filtrado.

Descripción del proyecto

- En este proyecto, crearemos una aplicación PWA para mostrar las condiciones climáticas de una ciudad.
- Inicialmente, aprenderemos a configurar la API de OpenWeather, que usaremos para obtener datos meteorológicos. Luego, aprenderemos cómo usar la API para mostrar información meteorológica en un componente angular.
- Veremos cómo convertir nuestra aplicación Angular en PWA usando un service worker. También implementaremos un mecanismo de notificación para nuestras actualizaciones de aplicaciones. Finalmente, publicaremos nuestra aplicación PWA en el proveedor de alojamiento de Firebase.

Configuración de la API OpenWeather

- La API de OpenWeather ha sido creada por el equipo de OpenWeather y contiene información meteorológica actual e histórica de más de 200.000 ciudades de todo el mundo. También admite datos meteorológicos de pronóstico para obtener información más detallada.
- En este proyecto, nos centraremos en los datos meteorológicos actuales.

- Primero necesitamos obtener una clave de API para comenzar a usar la API de OpenWeather:

1. Vaya al sitio web de la API de OpenWeather:
<https://openweathermap.org/api>.

Verá una lista de todas las API disponibles del equipo de OpenWeather.

2. Busque la sección titulada Datos meteorológicos actuales y haga clic en el botón Suscribirse.

Se le redirigirá a la página con los esquemas de precios disponibles del servicio.

Cada esquema admite una combinación diferente de llamadas a la API por minuto y mes.

Para este proyecto, usaremos el nivel gratuito.

3. Haga clic en el botón Obtener clave API.

Serás redirigido a la página de registro del servicio.

4. Complete todos los detalles requeridos y haga clic en el botón Crear cuenta.

Se enviará un mensaje de confirmación a la dirección de correo electrónico que utilizó para crear su cuenta.


5. Busque el correo electrónico de confirmación y haga clic en el botón Verificar su correo electrónico para completar su registro.

En breve recibirá otro correo electrónico de OpenWeather con detalles sobre su suscripción actual, incluida su clave de API y el extremo HTTP que usaremos para comunicarnos con la API.

Dear Customer!

Thank you for subscribing to Free [OpenWeatherMap](#)!

API key:

- Your API key is 
- Within the next couple of hours, it will be activated and ready to use
- You can later create more API keys on your [account page](#)
- Please, always use your API key in each API call

- La clave de API puede tardar un tiempo en activarse, generalmente un par de horas, antes de que pueda usarla.
- Tan pronto como se haya activado la clave API, podemos comenzar a usarla dentro de una aplicación Angular.

Visualización de datos meteorológicos

- Ahora crearemos una aplicación Angular para mostrar información meteorológica de una ciudad. El usuario ingresará el nombre de la ciudad en un campo de entrada y la aplicación utilizará la API de OpenWeather para obtener datos meteorológicos de la ciudad especificada. Cubriremos los siguientes temas con más detalle:
 - Configuración de la aplicación Angular
 - Comunicarse con la API de OpenWeather
 - Visualización de información meteorológica de una ciudad

Configurando la aplicación Angular

- Usaremos el comando `ng new` de Angular CLI para crear una nueva aplicación Angular desde cero

```
C:\Users\Geovany\Desktop\Angular> ng new weather-app --style=scss --routing=false
```


- El comando anterior creará una nueva aplicación CLI de Angular con las siguientes propiedades:
 - weather-app: el nombre de la aplicación Angular
 - --style = scss: Indica que nuestra aplicación Angular usará el SCSS
 - --routing = false: deshabilita el enrutamiento angular en la aplicación


- El usuario debe poder ingresar el nombre de la ciudad en un campo de entrada, y la información meteorológica de la ciudad debe visualizarse en un diseño de tarjeta.
- La biblioteca de angular material proporciona un conjunto de componentes de interfaz de usuario angular, que incluyen una entrada y una tarjeta.

- Los componentes de Angular Material se adhieren a los principios de Material Design y son mantenidos por el equipo de Componentes de Angular.
- Podemos instalar Angular Material usando el siguiente comando de Angular CLI:

```
lar\weather-app> ng add @angular/material --theme=indigo-pink --typography=true --animations=true
```

- El código anterior usa el comando `ng add` de Angular CLI, pasando opciones de configuración adicionales:
 - `@angular/material`: el nombre del paquete npm de la biblioteca Angular Material. También instalará el paquete `@angular/cdk`, un conjunto de comportamientos e interacciones que se utilizan para construir Angular Material. Ambos paquetes se agregarán a la sección de dependencias del archivo `package.json` de la aplicación.

- --theme = indigo-pink: El nombre del tema de Angular Material que queremos usar. Agregar un tema implica modificar varios archivos del espacio de trabajo de Angular CLI. Agrega entradas del archivo de tema CSS al archivo de configuración angular.json:

```
node_modules > @angular > material > prebuilt-themes > # indigo-pink.css >  .mat-badge-content
1  .mat-badge-content{font-weight:600;font-size:12px;font-family:Roboto, "Helvetica Neue", sans-serif}.mat-badge-small .
   mat-badge-content{font-size:9px}.mat-badge-large .mat-badge-content{font-size:24px}.mat-h1,.mat-headline,.mat-typography h1
   {font-size:24px/32px,Roboto, "Helvetica Neue", sans-serif;font-weight:normal;margin:0 0 16px 0}.mat-h2,.mat-title,.mat-typography
```


- También incluye los iconos de Material Design en el archivo index.html:

```
7 <meta name="viewport" content="width=device-width, initial-scale=1">
8 <link rel="icon" type="image/x-icon" href="favicon.ico">
9 <link rel="preconnect" href="https://fonts.gstatic.com">
10 <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500&display=swap" rel="stylesheet">
11 <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
12 </head>
```

- Angular Material viene con un conjunto de temas predefinidos que podemos usar. Alternativamente, podemos construir uno personalizado que se adapte a nuestras necesidades específicas. (suerte con eso xD)


- --typography = true: habilita la tipografía de material angular de forma global en nuestra aplicación. La tipografía define cómo se muestra el contenido del texto y utiliza la fuente Roboto Google de forma predeterminada, que se incluye en el archivo index.html:

```
8 <link rel="icon" type="image/x-icon" href="favicon.ico">
9 <link rel="preconnect" href="https://fonts.gstatic.com">
10 <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500&display=swap" rel="stylesheet">
11 <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
12 </head>
```



- Agrega la siguiente clase al cuerpo del archivo HTML:

```
13 | <body class="mat-typography">  
14 |   <app-root></app-root>  
15 | </body>  
16 | </html>  
17
```




- También agrega algunos estilos CSS al archivo global styles.scss de nuestra aplicación

```
html, body { height: 100%; }  
body { margin: 0; font-family: Roboto, "Helvetica Neue", sans-serif; }
```

- `--animations = true`: habilita las animaciones del navegador en nuestra aplicación al importar `BrowserAnimationsModule` en el módulo principal de la aplicación, `app.module.ts`

```
src > app > TS app.module.ts > ...
You, seconds ago | 1 author (You)
1 import { NgModule } from '@angular/core';      You, 4 days ago • initial commit
2 import { BrowserModule } from '@angular/platform-browser';
3
4 import { AppComponent } from './app.component';
5 import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
6
7 @NgModule({
8   declarations: [
9     AppComponent
10  ],
11   imports: [
12     BrowserModule,
13     BrowserAnimationsModule
14  ],
15   providers: [],
16   bootstrap: [AppComponent]
17 })
18 export class AppModule { }
```



- Casi hemos completado la instalación y configuración de nuestra aplicación Angular. El último paso es agregar la clave API que creamos en la sección Configuración de la API de OpenWeather.
- El espacio de trabajo de Angular CLI contiene la carpeta `src\environment` que podemos usar para definir la configuración de la aplicación, como claves API y end points.

- Contiene un archivo TypeScript para cada entorno que queremos admitir en nuestra aplicación. La CLI de Angular crea dos archivos por defecto:
 - `environment.ts`: el archivo TypeScript para el entorno de desarrollo. Se usa cuando iniciamos una aplicación Angular usando `ng serve`.
 - `environment.prod.ts`: el archivo TypeScript para el entorno de producción. Se usa cuando construimos la aplicación usando `ng build`.

- Cada archivo de entorno exporta un objeto de entorno. Agregue las siguientes propiedades al objeto en los archivos de desarrollo y producción:

```
src > environments > TS environment.prod.ts > ...  
...  
1 export const environment = {  
2   production: true,  
3   apiUrl: 'https://api.openweathermap.org/data/2.5/',  
4   apiKey: '<Your API key>  
5 };
```

- En el fragmento anterior, la propiedad `apiUrl` es la URL del end point que usaremos para realizar llamadas a la API de OpenWeather, y `apiKey` es nuestra clave de API. Reemplace el valor de `< Your API key >` con la clave de API que tiene.
- Ahora tenemos todas las partes móviles en su lugar para construir nuestra aplicación Angular. En la siguiente sección, crearemos un mecanismo para interactuar con la API de OpenWeather.

Comunicarse con la API de OpenWeather

- La aplicación debe interactuar con la API de OpenWeather a través de HTTP para obtener datos meteorológicos. Veamos cómo podemos configurar este tipo de comunicación en nuestra aplicación:
 - Primero, necesitamos crear una interfaz para describir el tipo de datos que obtendremos de la API. Use el siguiente comando generate de Angular CLI para crear uno:

El comando anterior `weather-app> ng generate interface weather` a carpeta `src\app` de nuestro proyecto CLI de Angular

- Abra el archivo weather.ts y modifíquelo de la siguiente manera.
- Cada propiedad corresponde a un campo meteorológico en la respuesta de la API de OpenWeather. Puede encontrar una descripción de cada uno en <https://openweathermap.org/current#parameter>

```
src > app > TS weather.ts > ...
1  export interface Weather {
2      weather: WeatherInfo[],
3      main: {
4          temp: number;
5          pressure: number;
6          humidity: number;
7      };
8      wind: {
9          speed: number;
10     };
11     sys: {
12         country: string;
13     };
14     name: string;
15 }
16 interface WeatherInfo {
17     main: string;
18     icon: string;
19 }
```


- Luego, necesitamos configurar el cliente HTTP integrado proporcionado por Angular.
- Abra el archivo `app.module.ts` y agregue `HttpClientModule` a la matriz de importaciones del decorador `@NgModule`:

src > app > TS app.module.ts > ...

You, seconds ago | 1 author (You)

1 import { NgModule } from '@angular/core';

2 import { BrowserModule } from '@angular/platform-browser';

3 | You, 4 days ago • initial commit

4 import { AppComponent } from './app.component';

5 import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

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

You, seconds ago | 1 author (You)

7 @NgModule({

8 declarations: [

9 | AppComponent

10],

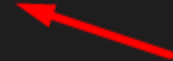
11 imports: [

12 | BrowserModule,

13 | BrowserAnimationsModule,

14 | HttpClientModule

15 |]



- Use el comando generate de Angular CLI para crear un nuevo servicio Angular:

```
weather-app> ng generate service weather
```

- El comando anterior creará el archivo weather.service.ts en la carpeta src\app de nuestro proyecto Angular CLI

- Abra el archivo weather.service.ts e inyecte el servicio HttpClient en su constructor:

```
src > app > TS weather.service.ts > ...  
1  import { HttpClient } from '@angular/common/http'  
2  import { Injectable } from '@angular/core';  
3  @Injectable({  
4    providedIn: 'root'  
5  })  
6  export class WeatherService {  
7    constructor(private http: HttpClient) { }  
8  }
```

- Agregue un método en el servicio que acepte el nombre de la ciudad como un parámetro único y consulte la API de OpenWeather para esa ciudad:

```
getWeather(city: string): Observable<Weather> {  
  const options = new HttpParams()  
    .set('units', 'metric')  
    .set('q', city)  
    .set('appId', environment.apiKey);  
  return this.http.get<Weather>(environment.apiUrl +  
    'weather', { params: options });  
}
```

- El método `getWeather` usa el método `get` del servicio `HttpClient` que acepta dos parámetros. El primero es el end point de la URL de la API de OpenWeather, que está disponible en la propiedad `apiUrl` del objeto `enviroment`.

- El objeto de entorno se importa del archivo `environment.ts` predeterminado. Angular CLI es responsable de reemplazarlo con el archivo `environment.prod.ts` cuando construimos nuestra aplicación.

- El segundo parámetro es un objeto de opciones que se utiliza para pasar configuración adicional a la solicitud, como parámetros de consulta de URL con la propiedad `params`. Usamos el constructor del objeto `HttpParams` y llamamos a su método `set` para cada parámetro de consulta que queremos agregar a la URL. En nuestro caso, pasamos el parámetro `q` para el nombre de la ciudad, el `appid` para la clave API que obtenemos del objeto de entorno y el tipo de unidades que queremos usar. Puede obtener más información sobre las unidades compatibles en <https://openweathermap.org/current#data>.

- También configuramos el tipo de datos de respuesta como Weather en el método get. Tenga en cuenta que el método getWeather no devuelve datos meteorológicos, sino un Observable de este tipo.
- El servicio Angular que creamos contiene todo lo necesario para interactuar con la API de OpenWeather. Ahora, crearemos un componente Angular para iniciar solicitudes y mostrar datos de él.

Visualización de información meteorológica de una ciudad

- El usuario debe poder usar la interfaz de usuario de nuestra aplicación e ingresar el nombre de una ciudad que desea ver los detalles del clima. La aplicación utilizará esa información para consultar la API de OpenWeather, y el resultado de la solicitud se mostrará en la interfaz de usuario mediante un diseño de tarjeta. Comencemos a construir un componente angular para crear todos estos tipos de interacciones:

- Use el comando generate de Angular CLI para crear un componente Angular:

```
er-app> ng generate component weather
```

- Abra la plantilla del componente principal, app.component.html, y reemplace su contenido con el selector del nuevo componente, app-weather:

```
src > app > <> app.component.html > ...  
1    <app-weather></app-weather>
```

- Abra el archivo `app.module.ts` y agregue los siguientes módulos de la biblioteca Angular Material a la matriz de importaciones del decorador `@NgModule`:

src > app > TS app.module.ts > ...

```
5 import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
6 import { HttpClientModule } from '@angular/common/http';
7 import { WeatherComponent } from '../weather/weather.component';
8 import { MatCardModule } from '@angular/material/card';
9 import { MatIconModule } from '@angular/material/icon';
10 import { MatInputModule } from '@angular/material/input';
11 @NgModule({
12   declarations: [
13     AppComponent,
14     WeatherComponent
15   ],
16   imports: [
17     BrowserModule,
18     BrowserAnimationsModule,
19     HttpClientModule,
20     MatIconModule,
21     MatInputModule,
22     MatCardModule
23   ]
24 })
```


- Abra el archivo `weather.component.ts`, cree una propiedad meteorológica del tipo `Weather` e inyecte `WeatherService` en el constructor de la clase `WeatherComponent`:

src > app > weather > TS weather.component.ts > WeatherComponent > constructor

```
1  import { Component, OnInit } from '@angular/core';
2  import { Weather } from '../weather';
3  import { WeatherService } from '../weather.service';
4  @Component({
5    selector: 'app-weather',
6    templateUrl: '../weather.component.html',
7    styleUrls: ['../weather.component.scss']
8  })
9  export class WeatherComponent implements OnInit {
10
11    weather: Weather | undefined;
12    constructor(private weatherService: WeatherService)
13    { }
14
15    ngOnInit(): void {
16    }
17
```



- Cree un método de componente que se suscriba al método `getWeather` de `WeatherService` y asigne el resultado a la propiedad del componente meteorológico:

src > app > weather > TS weather.component.ts >  WeatherComponent

```
1  import { Component, OnInit } from '@angular/core';
2  import { Weather } from '../weather';
3  import { WeatherService } from '../weather.service';
4  @Component({
5    selector: 'app-weather',
6    templateUrl: '../weather.component.html',
7    styleUrls: ['../weather.component.scss']
8  })
9  export class WeatherComponent implements OnInit {
10
11    weather: Weather | undefined;
12    constructor(private weatherService: WeatherService) { }
13
14    search(city: string) {
15      this.weatherService.getWeather(city).subscribe(weather => this.weather = weather);
16    }
17
```

- Ya hemos terminado de trabajar con el archivo de clase TypeScript de nuestro componente. Vamos a conectarlo con su plantilla. Abra el archivo `weather.component.html` y reemplace su contenido con el siguiente código HTML:

```
src > app > weather > <> weather.component.html > mat-card > mat-card-header > mat-card-title
```


```
1  <mat-form-field>
2    <input matInput placeholder="Enter city" #cityCtrl (keydown.enter)="search(cityCtrl.value)">
3    <mat-icon matSuffix (click)="search(cityCtrl.value)">search</mat-icon>
4  </mat-form-field>
5  <mat-card *ngIf="weather">
6    <mat-card-header>
7      <mat-card-title>{{weather.name}},
8      {{weather.sys.country}}</mat-card-title>
9      <mat-card-subtitle>{{weather.weather[0].main}}
10     </mat-card-subtitle>
11   </mat-card-header>
12   
14   <mat-card-content>
15     <h1>{{weather.main.temp | number:'1.0-0'}} &#8451;</h1>
16     <p>Pressure: {{weather.main.pressure}} hPa</p>
17     <p>Humidity: {{weather.main.humidity}} %</p>
18     <p>Wind: {{weather.wind.speed}} m/s</p>
19   </mat-card-content>
20 </mat-card>
```

- La plantilla anterior consta de varios componentes de la biblioteca Angular Material, incluido un componente `mat-form-field` que contiene los siguientes elementos secundarios:
 - Un elemento HTML de entrada para ingresar el nombre de la ciudad. Cuando el usuario termina de editar y presiona la tecla Enter, llama al método del componente de búsqueda pasando la propiedad `value` de la variable `cityCtrl` como parámetro. La variable `cityCtrl` es una variable de referencia de plantilla e indica el objeto real del elemento de entrada HTML nativo.
 - Un componente `mat-icon` muestra un icono de lupa y se encuentra al final del elemento de entrada, como lo indica la directiva `matSuffix`. También llama al método del componente de búsqueda cuando se hace clic en él.

- La variable de referencia de la plantilla cityCtrl se indica con un # y se puede acceder a ella en cualquier lugar dentro de la plantilla del componente.

- Un componente mat-card presenta información en un diseño de tarjeta y se muestra solo cuando la propiedad del componente meteorológico tiene un valor. Consta de los siguientes elementos secundarios:
 - mat-card-header: el encabezado de la tarjeta. Consiste en un componente mat-cardtitle que muestra el nombre de la ciudad y el código del país y un componente mat-card-subtitle que muestra las condiciones climáticas actuales.
 - mat-card-image: la imagen de la tarjeta que muestra el icono de las condiciones meteorológicas, junto con una descripción como texto alternativo.
 - mat-card-content: el contenido principal de la tarjeta. Muestra la temperatura, la presión, la humedad y la velocidad del viento del clima actual. La temperatura se muestra sin puntos decimales, como lo indica el pipe numérico.

- Vamos a mejorar un poco las cosas añadiendo algunos estilos a nuestro componente:

src > app > weather >  weather.component.scss > ...

```
1  :host {
2      display: flex;
3      align-items: center;
4      justify-content: center;
5      flex-direction: column;
6      padding-top: 25px;
7  }
8  mat-form-field {
9      width: 20%;
10 }
11 mat-icon {
12     cursor: pointer;
13 }
14 mat-card {
15     margin-top: 30px;
16     width: 250px;
17 }
18 h1 {
19     text-align: center;
20     font-size: 2.5em;
```

- El selector `:host` de CSS único de Angular que apunta al elemento HTML que aloja nuestro componente, que en nuestro caso, es el elemento HTML de `app-weather`.
- Si ejecutamos nuestra aplicación usando `ng serve`, navegamos a `http://localhost:4200` y buscamos información meteorológica en Los Mochis, deberíamos obtener el siguiente resultado en la pantalla:

Enter city

Los Mochis



Los Mochis, MX

Clear



34 °C

Pressure: 1011 hPa

Humidity: 32 %

Wind: 6.09 m/s