

# 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 asíncronica. 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;letter-spacing:normal;margin:0 0 16px}mat-h2,mat-title,mat-typography
```



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

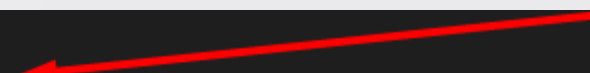
```
7 <meta name="viewport" content="width=device-width, initial-scale=1">
8 <link rel="icon" type="image/x-icon" href="favicon.ico">
9 <link rel="preconnect" href="https://fonts.gstatic.com">
10 <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500&display=swap" rel="stylesheet">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

El comando anterior creará el archivo `weather.ts` en la 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 }
```

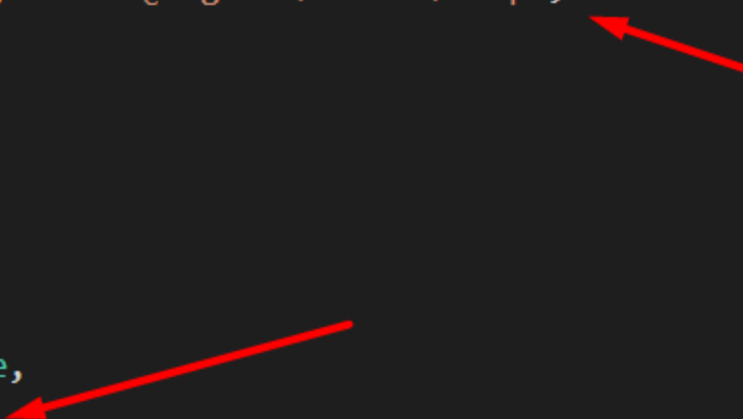


- Luego, necesitamos configurar el cliente HTTP integrado proporcionado por Angular.
- Abra el archivo `app.module.ts` y agregue `HttpClientModule` a la matriz de importaciones del decorador `@NgModule`:





```
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  ]  
16  })  
17  export class AppModule {}  
18  export { AppModule };
```






- 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 todos 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





- ¡Felicidades! En este punto, tiene una aplicación Angular completamente funcional que muestra información meteorológica para una ciudad específica. La aplicación consta de un único componente Angular que se comunica con la API OpenWeather mediante un servicio Angular a través de HTTP. Aprendimos cómo diseñar nuestro componente usando Angular Material y brindar a nuestros usuarios una experiencia agradable con nuestra aplicación. Pero, ¿qué pasa cuando estamos desconectados? ¿Funciona la aplicación como se esperaba? ¿La experiencia del usuario sigue siendo la misma?



## Habilitación del modo fuera de línea con service worker

- Los usuarios desde cualquier lugar ahora pueden acceder a nuestra aplicación Angular para obtener información meteorológica de cualquier ciudad que les interese. Cuando decimos cualquier lugar, nos referimos a cualquier tipo de red, como banda ancha, celular (3G / 4G / 5G) y Wi-Fi.
- Considere el caso en el que un usuario se encuentra en un lugar con poca cobertura o cortes frecuentes de la red. ¿Cómo se comportará nuestra aplicación? Averigüemos realizando un experimento:




- Ejecute la aplicación Angular usando el comando de servicio de Angular CLI:


```
r-app> ng serve
```

- Abra su navegador favorito y navegue hasta <http://localhost:4200>, que es la dirección y el número de puerto predeterminados para un proyecto CLI de Angular. Debería ver el campo de entrada para ingresar el nombre de la ciudad:



 localhost:4200

Enter city





- Abra las herramientas de desarrollo de su navegador y navegue hasta la pestaña Red. Establezca el valor del menú desplegable en Offline o si está en edge ir a Aplicación, Service worker y marcar Offline



→

↺

i

localhost:4200

☆+

☆≡

🔖

👤

⋮

Aplicación x

»

+

⚠️ 1

💬 11

⚙️

🗨️

⋮

✕

Aplicación

📄

Manifiesto

⚙️

Service Workers

🗄️

Almacenamiento

Almacenamiento

▶

🗄️

Almacenamiento local

▶

🗄️

Almacenamiento de sesi

🗄️

IndexedDB

🗄️

Web SQL

▶

🍪

Cookies

🗄️

Tokens de confianza

Caché

Service Workers

☒ Desconectado

☐ Actualizar al volver a cargar

Trabajos de servicio de otros orígenes

[Ver todos los registros](#)

## No está conectado

La web no es lo mismo sin usted. Vamos a restablecer su conexión.

**Prueba a:**





- El caso anterior es bastante común en áreas con conexiones a Internet de baja calidad. Entonces, ¿qué podemos hacer por nuestros usuarios en tales áreas? Afortunadamente, Angular contiene una implementación de service worker que puede mejorar significativamente la UX de nuestra aplicación cuando se ejecuta en modo fuera de línea.
- Puede almacenar en caché ciertas partes de la aplicación y entregarlas en consecuencia en lugar de realizar solicitudes reales.



- El service worker de angular también se puede utilizar en entornos con conexiones de latencia de red grandes. Considere la posibilidad de utilizar un service worker en este tipo de red para mejorar también la experiencia de sus usuarios.



- Ejecute el siguiente comando de la CLI de Angular para habilitar el trabajador del servicio en nuestra aplicación Angular:


```
weather-app> ng add @angular/pwa
```



- El comando anterior transformará el espacio de trabajo de Angular CLI para la compatibilidad con PWA:
  - Agrega el paquete `npm@angular/service-worker` a la sección de dependencias del archivo `package.json` de la aplicación.
  - Crea el archivo `manifest.webmanifest` en la carpeta `src` de la aplicación.
  - El archivo de manifiesto contiene información sobre la aplicación necesaria para instalar y ejecutar la aplicación como nativa. También lo agrega al arreglo de las opciones de compilación en el archivo `angular.json`.



- Crea el archivo `ngsw-config.json` en la raíz del proyecto, que es el archivo de configuración del service worker. Lo usamos para definir artefactos específicos de la configuración, como qué recursos se almacenan en caché y cómo se almacenan en caché. Puede encontrar más detalles sobre la configuración del trabajador del servicio en el siguiente enlace: <https://angular.io/guide/service-worker-config#serviceworker-configuration>.
- El archivo de configuración también se establece en la propiedad `ngswConfigPath` de la configuración de compilación en el archivo `angular.json`.
- Establece la propiedad `serviceWorker` en `true` en la configuración de compilación del archivo `angular.json`.

- 
- Registra el service worker en el archivo app.module.ts:

```
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 import { ServiceWorkerModule } from '@angular/service-worker';
12 import { environment } from '../environments/environment';
13
14 @NgModule({
```



<> app.component.html

TS app.module.ts ●

TS weather.component.ts

<> weather.component.htm

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

```
22     HttpClientModule,  
23     MatIconModule,  
24     MatInputModule,  
25     MatCardModule,  
26     ServiceWorkerModule.register('ngsw-worker.js', {  
27       enabled: environment.production,  
28       // Register the ServiceWorker as soon as the app is stable  
29       // or after 30 seconds (whichever comes first).  
30       registrationStrategy: 'registerWhenStable:30000'  
31     })  
32  
33   ],  
34   providers: [],  
35   bootstrap: [AppComponent]  
36 })
```




- El archivo `ngsw-worker.js` es el archivo JavaScript que contiene la implementación real del service worker. Se crea automáticamente para nosotros cuando construimos nuestra aplicación en modo de producción. Angular usa el método de registro de la clase `ServiceWorkerModule` para registrarlo dentro de nuestra aplicación.
- Crea varios iconos para ser utilizados cuando la aplicación se instala como nativa en el dispositivo del usuario.



- Incluye el archivo de manifiesto y una metaetiqueta para theme-color en el encabezado del archivo index.html:

```
9      <link rel="preconnect" href="https://fonts.gstatic.com">
10     <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@300
11     <link href="https://fonts.googleapis.com/icon?family=Material+Icons"
12     <link rel="manifest" href="manifest.webmanifest">
13     <meta name="theme-color" content="#1976d2">
14 </head>
```





- Ahora que hemos completado la instalación del service worker, ¡es hora de probarlo!
- Antes de continuar, deberíamos instalar un servidor web externo porque la función incorporada de Angular CLI no funciona con los trabajadores del servicio. Una buena alternativa es el servidor http:



- Ejecute el comando de instalación del cliente npm para instalar el servidor http:

```
weather-app> npm install -D http-server
```

- El comando anterior instalará http-server como una dependencia de desarrollo de nuestro proyecto Angular CLI.
- Construya la aplicación Angular usando el comando build de Angular CLI:

```
weather-app> ng build
```



- Abra el archivo `package.json` del espacio de trabajo de Angular CLI y agregue la siguiente entrada a la propiedad de scripts:





```
{ } package.json > { } scripts
```

```
1  {
2    "name": "weather-app",
3    "version": "0.0.0",
4    "scripts": {
5      "ng": "ng",
6      "start": "ng serve",
7      "build": "ng build",
8      "watch": "ng build --watch --configuration development",
9      "test": "ng test",
10     "server": "http-server -p 8080 -c-1 dist/weather-app"
11   }
12 }
```





- Inicie el servidor web HTTP con el siguiente comando:

```
ather-app> npm run server
```

- El comando anterior iniciará el servidor http en el puerto 8080 y tendrá el almacenamiento en caché deshabilitado.



- Abra su navegador y navegue hasta <http://localhost:8080>.



- De preferencia abrir la página en modo privado o de incógnito para evitar un comportamiento inesperado del service worker.
- Repita el proceso que seguimos al principio de la sección para cambiar al modo fuera de línea.
- Si actualiza la página ahora, notará que la aplicación está funcionando como se esperaba.
- El service worker hizo todo el trabajo por nosotros y el proceso fue tan transparente que no pudimos saber si estamos en línea o fuera de línea. Puede verificarlo inspeccionando la pestaña Red:



127.0.0.1:8080

Enter city

Aplicación

Manifiesto

Service Workers

Almacenamiento

Almacenamiento

Almacenamiento local

Almacenamiento de sesi

Service Workers

☒ Desconectado ☐ Actualizar al volver

**http://127.0.0.1:8080/**

Origen [ngsw-worker.js](#)

Recibido el 21/10,



- Ahora hemos instalado con éxito el service worker y nos acercamos un paso más a convertir nuestra aplicación en una PWA. En la siguiente sección, aprenderemos cómo notificar a los usuarios de la aplicación sobre posibles actualizaciones.





# Mantenerse al día con las notificaciones en la aplicación

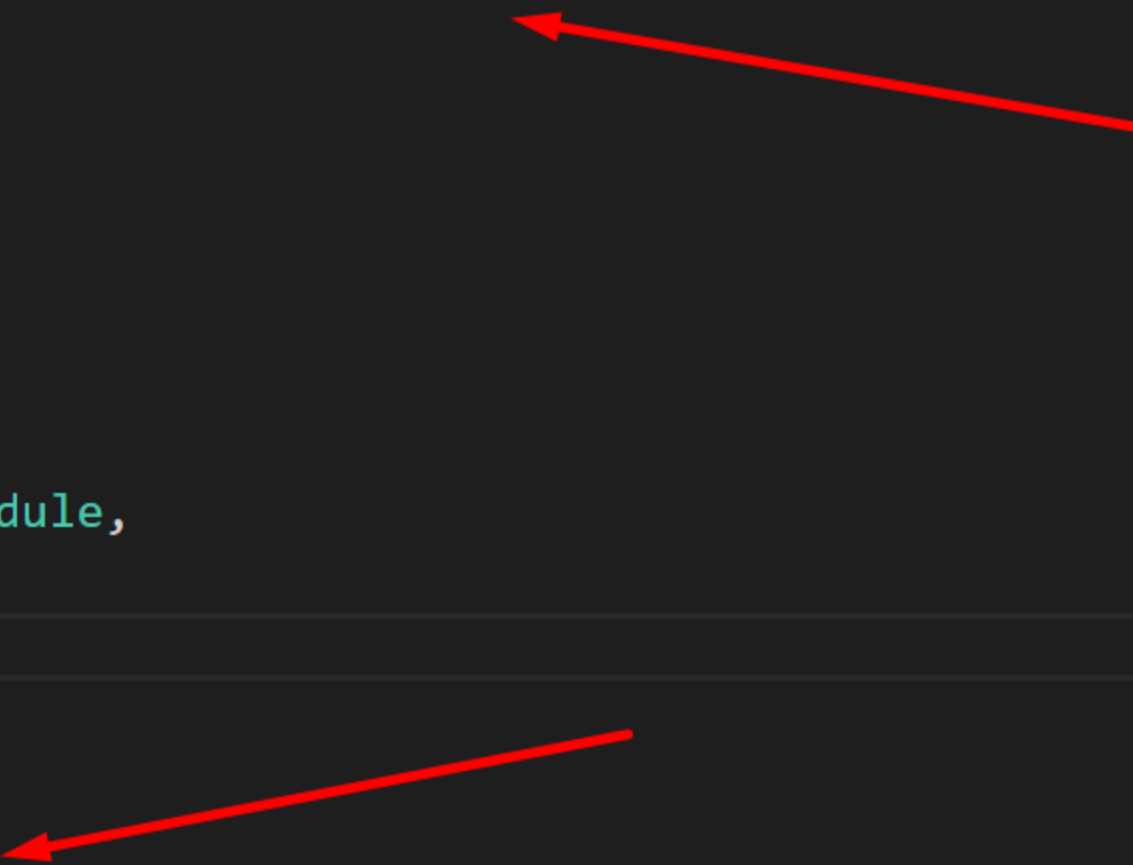
Cuando queremos aplicar un cambio en una aplicación web, hacemos el cambio y creamos una nueva versión de nuestra aplicación. Luego, la aplicación se implementa en un servidor web y cada usuario tiene acceso a la nueva versión de inmediato. Pero este no es el caso de las aplicaciones PWA.

Cuando implementamos una nueva versión de nuestra aplicación PWA, el service worker debe actuar en consecuencia y aplicar una estrategia de actualización específica. Debe notificar al usuario de la nueva versión o instalarla inmediatamente. Cualquier estrategia de actualización que sigamos depende de nuestras necesidades. En este proyecto, queremos mostrar un mensaje al usuario y decidir si desea actualizar. Veamos cómo implementar esta característica en nuestra aplicación:



- Abra el archivo `app.module.ts` y agregue `MatSnackBarModule` al arreglo de importaciones del decorador `@NgModule`:

```
12 import { environment } from '../environments/environment';
13 import { MatSnackBarModule } from '@angular/material/snack-bar';
14 @NgModule({
15   declarations: [
16     AppComponent,
17     WeatherComponent
18   ],
19   imports: [
20     BrowserModule,
21     BrowserAnimationsModule,
22     HttpClientModule,
23     MatIconModule,
24     MatInputModule,
25     MatCardModule,
26     MatSnackBarModule,
27     ServiceWorkerModule.register('ngsw-worker.js', {
```





- MatSnackBarModule es un módulo de angular material que nos permite interactuar con las barras. Una barra como una ventana emergente que generalmente aparece en la parte inferior de la página y se utiliza con fines de notificación.



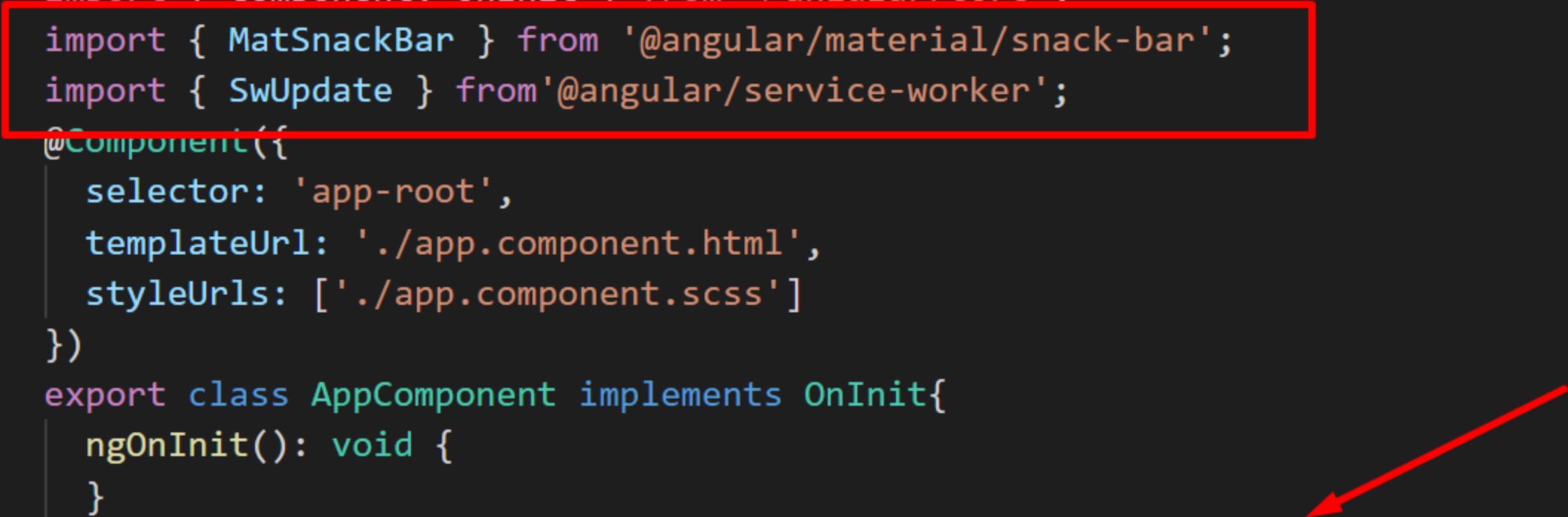
- Abra el archivo `app.component.ts` y agregue la interfaz `OnInit` a las interfaces implementadas de la clase `AppComponent`.
- Inyecte los servicios `MatSnackBar` y `SwUpdate` en el constructor de la clase `AppComponent`:





```
src > app > TS app.component.ts > ...
```

```
1  import { Component, OnInit } from '@angular/core';
2  import { MatSnackBar } from '@angular/material/snack-bar';
3  import { SwUpdate } from '@angular/service-worker';
4  @Component({
5    selector: 'app-root',
6    templateUrl: './app.component.html',
7    styleUrls: ['./app.component.scss']
8  })
9  export class AppComponent implements OnInit{
10    ngOnInit(): void {
11    }
12    constructor(private updates: SwUpdate, private snackbar: MatSnackBar) {}
13
```







- El servicio MatSnackBar es un servicio angular expuesto desde MatSnackBarModule. El servicio SwUpdate es parte del service worker y contiene observables que podemos usar para notificar sobre el proceso de actualización en nuestra aplicación.



- Cree el siguiente método ngOnInit:

```
4 import { filter, map, switchMap } from 'rxjs/operators';
5 @Component({
6   selector: 'app-root',
7   templateUrl: './app.component.html',
8   styleUrls: ['./app.component.scss']
9 })
10 export class AppComponent implements OnInit {
11   ngOnInit() {
12     this.updates.available.pipe(
13       switchMap(() => this.snackbar.open('A new version is available!', 'Update now').afterDismissed()),
14       filter(result => result.dismissedByAction),
15       map(() => this.updates.activateUpdate().then(() =>
16         location.reload()))
17     ).subscribe();
18   }
19 }
```



- El método `ngOnInit` es un método de implementación de la interfaz `OnInit` y se llama al inicializar el componente. El servicio `SwUpdate` contiene una propiedad observable disponible que podemos usar para recibir notificaciones cuando esté disponible una nueva versión de nuestra aplicación. Por lo general, tendemos a suscribirnos a observables, pero en este caso, no lo hacemos. En cambio, nos suscribimos al método de pipe, un operador `RxJS` para componer múltiples operadores.



- Están sucediendo muchas cosas dentro del método de pipe que definimos anteriormente, así que vamos a dividirlo en partes para comprenderlo mejor. El operador de pipe combina tres operadores RxJS:
  - switchMap: se llama cuando hay disponible una nueva versión de nuestra aplicación.
  - Utiliza el método abierto de la propiedad snackbar para mostrar un snack bar con un botón de acción y se suscribe a su observable afterDismissed. El observable afterDismissed se emite cuando el snack bar se cierra haciendo clic en el botón de acción o mediante programación utilizando sus métodos API.
  - filter: se llama cuando la barra se cierra con el botón de acción.
  - map: Esto llama al método activeUpdate de la propiedad de actualizaciones para aplicar la nueva versión de la aplicación. Una vez que la aplicación se ha actualizado, vuelve a cargar la ventana del navegador para que los cambios surtan efecto.



- Veamos todo el proceso de actualización a una nueva versión en acción:
  - Ejecute el comando de compilación de Angular CLI para compilar la aplicación Angular:

```
er-app> ng build
```

- Inicie el servidor HTTP para servir la aplicación:

```
er-app> npm run server
```



- Abra una ventana privada o de incógnito de su navegador y navegue hasta `http://localhost:8080`.
- Sin cerrar la ventana del navegador, introduzcamos un cambio en nuestra aplicación y agreguemos un encabezado. Ejecute el comando `generate` de Angular CLI para crear un componente:

```
her-app> ng generate component header
```






- Abra el archivo app.module.ts e importe un par de módulos de material angular:

```
14   import { HeaderComponent } from './header/header.component';  
15   import { MatButtonModule } from '@angular/material/button';  
16   import { MatToolbarModule } from '@angular/material/toolbar';  
17   @NgModule({
```




```
22 ],
23 imports: [
24     BrowserModule,
25     BrowserAnimationsModule,
26     HttpClientModule,
27     MatIconModule,
28     MatInputModule,
29     MatCardModule,
30     MatSnackBarModule,
31     MatButtonModule,
32     MatToolbarModule,
33 
```

- 
- Abra el archivo header.component.html y cree un componente mat-toolbar con dos elementos de botón HTML, cada uno con un componente mat-icon:

```
src > app > header > <> header.component.html > ...
1   <mat-toolbar color="primary">
2     <span>Weather App</span>
3     <span class="spacer"></span>
4     <button mat-icon-button>
5       <mat-icon>refresh</mat-icon>
6     </button>
7     <button mat-icon-button>
8       <mat-icon>share</mat-icon>
9     </button>
10  </mat-toolbar>
11
```



- Agregue el siguiente estilo CSS al archivo header.component.scss para colocar los botones en el extremo derecho del encabezado:

```
src > app > header >  header.component.scss > ...  
1   .spacer {  
2       flex: 1 1 auto;  
3   }
```

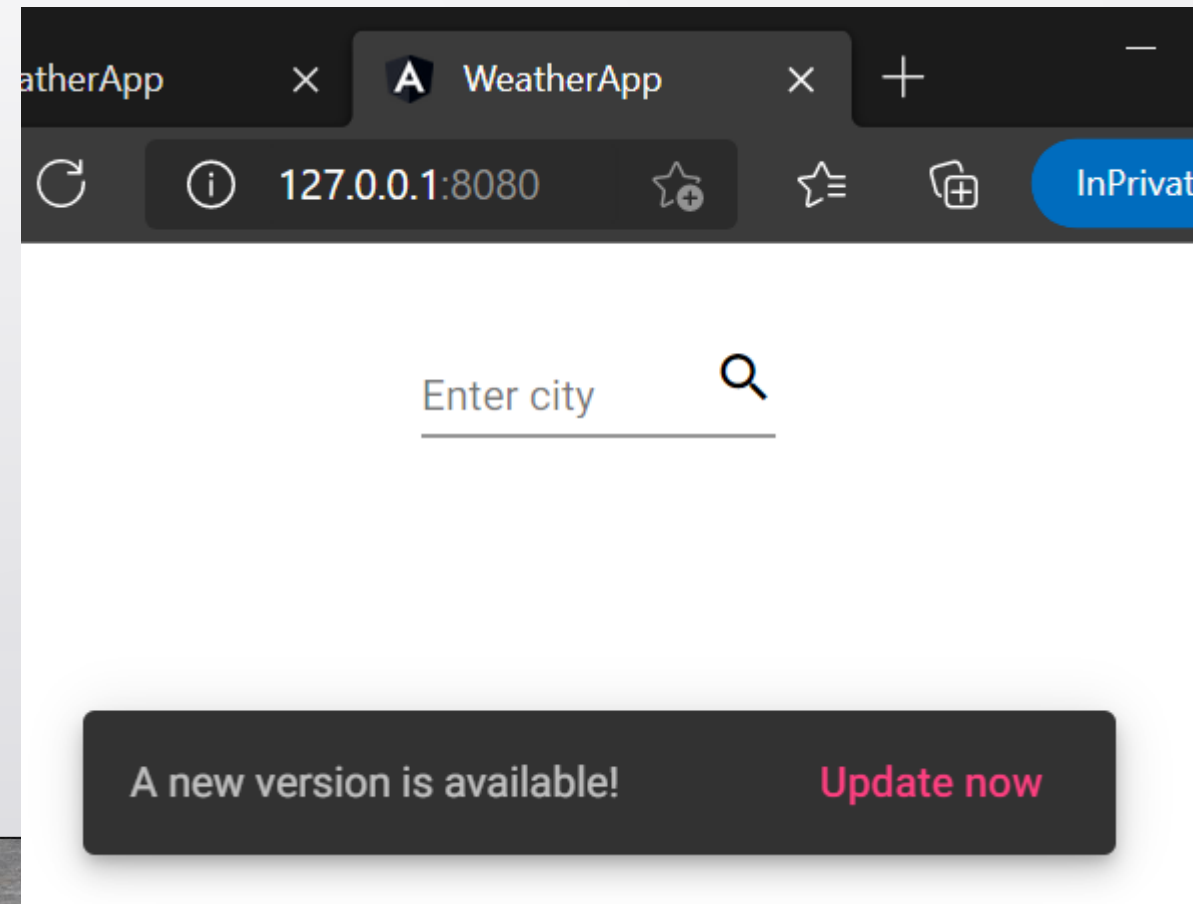


- Abra el archivo `app.component.html` y agregue el componente de encabezado de la aplicación en la parte superior:

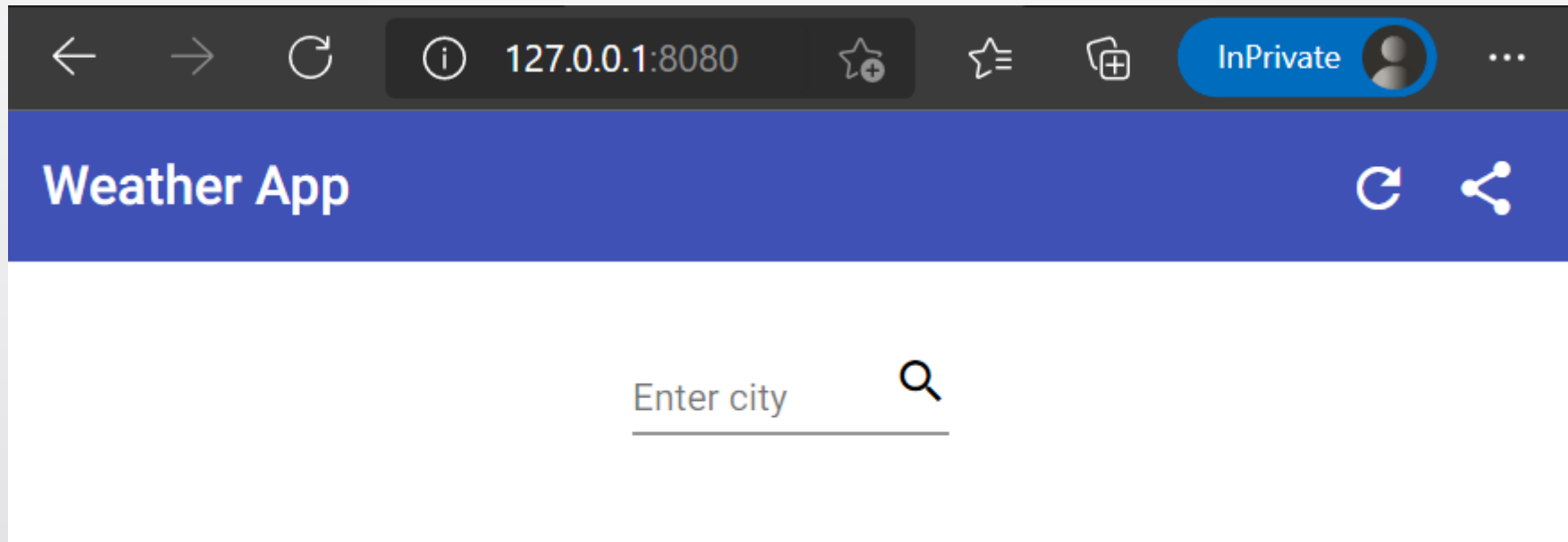
```
src > app > <> app.component.html > ...  
1  <app-header></app-header>  
2  <app-weather></app-weather>  
3
```



- Repita los pasos para crear e iniciar la aplicación. Abra una nueva pestaña en la ventana del navegador que ya ha abierto y verá la siguiente notificación en la parte inferior de la página después de unos segundos:










- Nuestra aplicación Angular ha comenzado a transformarse en una PWA. Además del mecanismo de almacenamiento en caché que proporciona el service worker de Angular, hemos agregado un mecanismo para instalar nuevas versiones de nuestra aplicación. En la siguiente sección, aprenderemos cómo implementar nuestra aplicación e instalarla de forma nativa en nuestro dispositivo.



# Implementando nuestra aplicación con Firebase hosting

- Firebase es una solución de alojamiento proporcionada por Google que podemos usar para implementar nuestras aplicaciones Angular. El equipo de Firebase se ha esforzado mucho en crear un esquema CLI de Angular para implementar una aplicación Angular usando un solo comando. Antes de profundizar, aprendamos a configurar el alojamiento de Firebase:



- Utilice una cuenta de Google para iniciar sesión en Firebase en <https://console.firebase.google.com>.
- Haga clic en el botón Agregar proyecto para crear un nuevo proyecto de Firebase.
- Introduzca el nombre del proyecto, weather-app y haga clic en el botón Continuar.
- Desactive el uso de Google Analytics para su proyecto y haga clic en el botón Crear proyecto.
- Una vez creado el proyecto, aparecerá en pantalla lo siguiente:

## Compilación

- 👤 Authentication
- 📦 Firestore Database
- 📡 Realtime Database
- 📁 Storage
- 🌐 Hosting
- ⌘ Functions
- 🤖 Machine Learning

## Lanzamiento y supervisión

- 🧩 Extensions

weather-app ▾

[Ir a la documentación](#)



Recibe notificaciones por correo electrónico sobre nuevas funciones de Firebase, investigaciones y eventos

[Registrarse](#)



# weather-app

Plan Spark

## Comienza por agregar Firebase a tu app

iOS



Agrega una app para comenzar





- Ahora hemos completado la configuración del alojamiento de Firebase. Ahora es el momento de integrarlo con nuestra aplicación Angular. Ejecute el siguiente comando de Angular CLI para instalar el paquete `npm @angular/fire` en su proyecto de Angular CLI:

```
ner-app> ng add @angular/fire
```





- El comando anterior también lo autenticará con Firebase y le pedirá que seleccione un proyecto de Firebase para implementarlo. Use las teclas de flecha para seleccionar el proyecto de aplicación meteorológica que creamos anteriormente y presione Entrar. El proceso modificará el espacio de trabajo de Angular CLI en consecuencia para adaptarse a su implementación en Firebase:
  - Agregará varios paquetes npm a las secciones de dependencias y devDependencies del archivo package.json del proyecto.
  - Creará un archivo .firebaserc en la carpeta raíz que contiene detalles del proyecto de Firebase seleccionado.
  - Creará un archivo firebase.json en la carpeta raíz, que es el archivo de configuración de Firebase:



✓ TERMINAL

☐ NODE + ✓

? What features would you like to setup? `ng deploy -- hosting`

✓ `firebase-tools` installed globally.

Using `firebase-tools` version 9.21.0

? Allow Firebase to collect CLI usage and error reporting information? `No`

? Paste authorization code here: `4/1AX4XfWgfIkXvhGHFYuWKdZqbMV7YTpe8bV0t0glr8hSSX1Cc13mE0hONT18`

✓ Preparing the list of your Firebase projects

? Please select a project:

[CREATE NEW PROJECT]

`ComprasApp`

`indicadores`

`Lawyers`

> `weather-app`





- El archivo de configuración especifica configuraciones como la carpeta que se implementará en Firebase como se indica en la propiedad pública y las reglas de reescritura con la propiedad de reescrituras.
- La carpeta que se implementará de forma predeterminada es la carpeta de salida dist creada por Angular CLI cuando ejecutamos el comando ng build.

- 
- Agregaré una entrada “deply” a la sección architect del archivo de configuración angular.json:

```
  "deploy": {  
    "builder": "@angular/fire:deploy",  
    "options": {  
      "prerender": false,  
      "ssr": false,  
      "browserTarget": "weather-app:build:production",  
      "firebaseProject": "weather-app-718a6",  
      "firebaseHostingSite": "weather-app-718a6"  
    }  
  }  
}
```



- Para desplegar la aplicación, solo necesitamos ejecutar un único comando de CLI de Angular, y el CLI de Angular se encargará del resto:

```
weather-app> ng deploy
```



- El comando anterior compilará la aplicación y comenzará a implementarla en el proyecto de Firebase seleccionado. Una vez que se completa la implementación, la CLI de Angular informará la siguiente información:
  - Project Console: el panel del proyecto de Firebase.
  - URL de alojamiento: la URL de la versión implementada de la aplicación. Consiste en el identificador único del proyecto de Firebase y el sufijo `.web.app` que se agrega automáticamente desde Firebase.





El service worker requiere que una aplicación se sirva con HTTPS para que funcione correctamente como PWA, excepto en el host local que se utiliza para el desarrollo. Firebase aloja aplicaciones web con HTTPS de forma predeterminada



```
+ hosting[weather-app-718a6]: version finalized
i hosting[weather-app-718a6]: releasing new version...
+ hosting[weather-app-718a6]: release complete
```

```
+ Deploy complete!
```

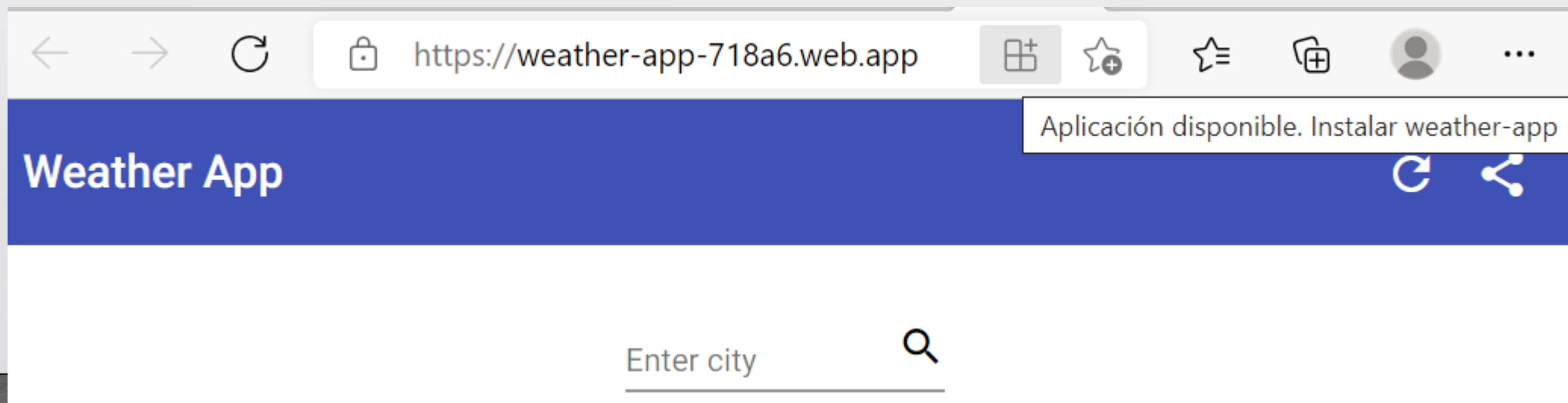
```
Project Console: https://console.firebase.google.com/project/weather-app-718a6/overview
```

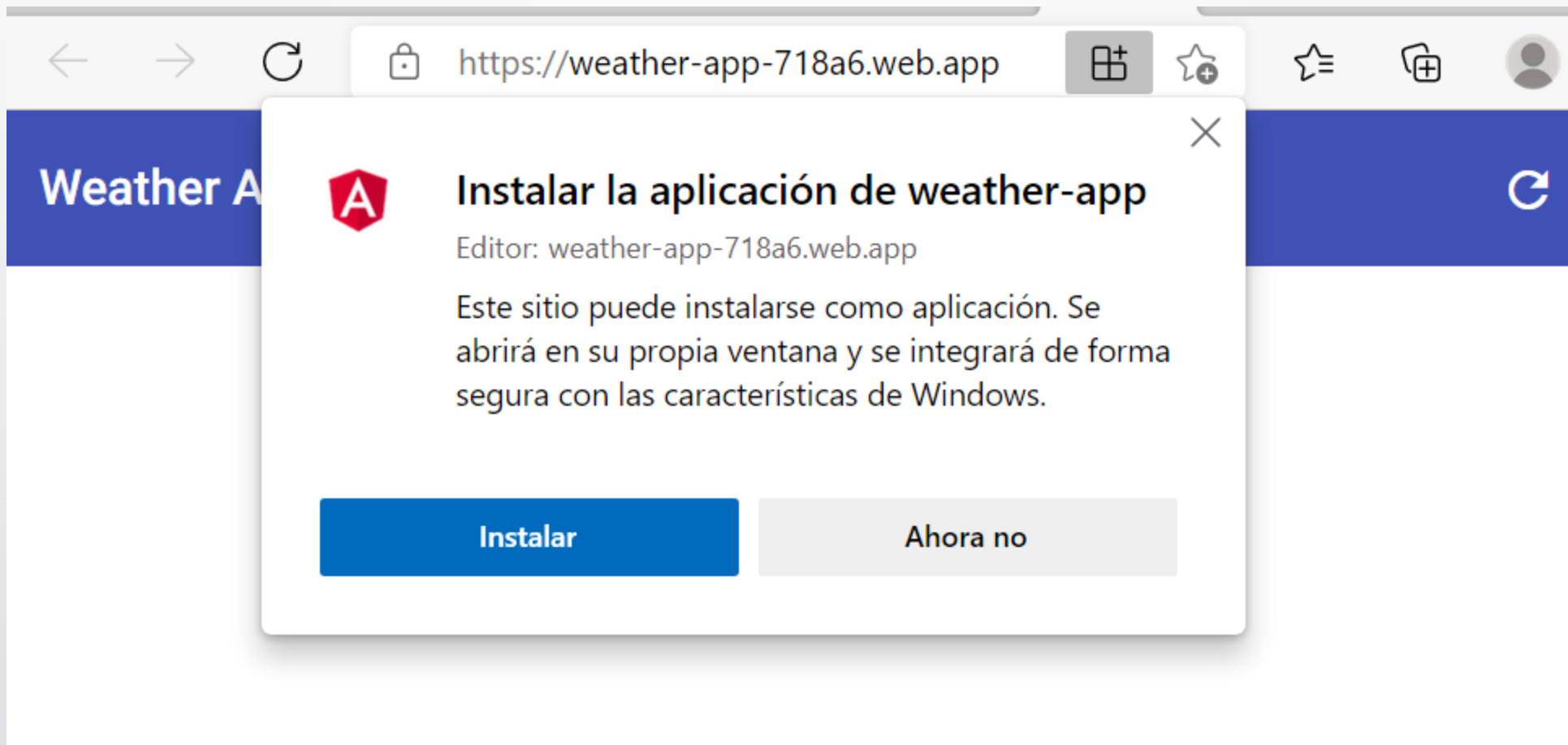
```
Hosting URL: https://weather-app-718a6.web.app
```

```
PS C:\Users\Geovany\Desktop\webII\weather-app> █
```



- Ahora que hemos implementado nuestra aplicación, veamos cómo podemos instalarla como PWA en nuestro dispositivo:
  - Navegue hasta la URL de alojamiento y haga clic en el botón más junto a la barra de direcciones del navegador:





## Weather App



## Aplicación instalada

Editor: weather-app-718a6.web.app

weather-app se instaló como una aplicación en el dispositivo y se ejecutará de forma segura en su propia ventana. Iniciallo desde el menú Inicio, la barra de tareas de Windows o el escritorio.

## Permitir que esta aplicación

- ☒ Anclarse a la barra de tareas
- ☒ Anclarse al Inicio
- ☐ Crea acceso directo en el escritorio
- ☐ Iniciar automáticamente al iniciar sesión en el dispositivo

**Permitir**

No permitir



- ¡Felicidades! Ahora tenemos una aplicación PWA completa que muestra información meteorológica de una ciudad.





# Resumen

En este capítulo, creamos una aplicación PWA que muestra información meteorológica para una ciudad determinada.

Inicialmente, configuramos la API de OpenWeather para obtener datos meteorológicos y creamos una aplicación Angular desde cero para integrarla. Aprendimos a usar el cliente HTTP integrado en Angular para comunicarnos con la API de OpenWeather. También instalamos la biblioteca Angular Material y usamos algunos de los componentes de IU listos para usar para nuestra aplicación.



Después de crear la aplicación Angular, presentamos el service worker de Angular y lo habilitamos para que funcione sin conexión. Aprendimos cómo interactuar con el service worker y proporcionar notificaciones de actualizaciones en nuestra aplicación. Finalmente, implementamos una versión de producción de nuestra aplicación en el alojamiento de Firebase y la instalamos localmente en nuestro dispositivo.



# Preguntas de práctica

1. ¿Cuál es el propósito de los archivos de entorno en Angular?
2. ¿Cómo pasamos los parámetros de consulta en una solicitud HTTP?
3. ¿Cómo obtenemos resultados de un método get HTTP?
4. ¿Por qué utilizamos un service worker?
5. ¿Qué componente de angular material se utiliza para las notificaciones?
6. ¿Cómo se nos notifica acerca de las actualizaciones del trabajo de servicio?
7. ¿Cuál es el uso principal del operador de pipe RxJS?
8. ¿Cómo podemos cambiar el color del tema de una aplicación PWA?
9. ¿Qué paquete npm usamos para implementar en Firebase?
10. ¿Cómo podemos cambiar la carpeta que se despliega en Firebase?



# Lecturas adicionales

- PWA: <https://web.dev/progressive-web-apps/>
- OpenWeather API: <https://openweathermap.org/api>
- Angular Material: <https://material.angular.io/>
- Cliente Angular HTTP: <https://angular.io/guide/http>
- Angular service worker: <https://angular.io/guide/service-workergetting-started>
- Comunicándose con service worker: <https://angular.io/guide/service-worker-communications>
- HTTP server: <https://www.npmjs.com/package/http-server>
- Firebase hosting: <https://firebase.google.com/docs/hosting>
- Desplegar en Angular: <https://angular.io/guide/deployment#automatic-deployment-with-the-cli>