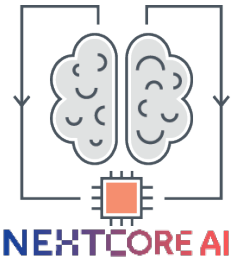
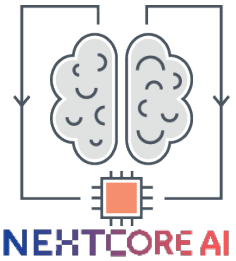


Angular Services and HttpClient to retrieve data

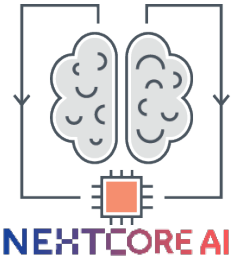
1. Create a new Angular Service
2. Import `HttpClientModule` and inject it into the service
3. Discover the `OpenWeatherMap` API
4. Create a new interface that conforms to the shape of the API
5. Write a `get` request
6. Inject the new service into the `CurrentWeather` component
7. Call the service from the `init` function of the `CurrentWeather` component
8. Finally, map the API data to the local `ICurrentWeather` type using RxJS functions so that it can be consumed by your component



Angular Services

- In the terminal, execute `npx ng g s weather --flat false`
- Observe the new `weather` folder created:

```
src/app
...
└── weather
    ├── weather.service.spec.ts
    └── weather.service.ts
```



Generated services

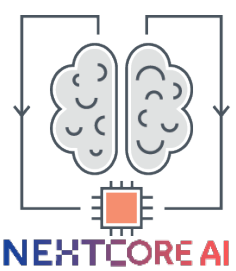
A generated service has two parts:

- weather.service.spec.ts contains Jasmine-based unit tests that you can extend to test your service functionality.
- weather.service.ts contains the @Injectable decorator above the class definition, which makes it possible to inject this service into other components, leveraging Angular's provider system. This will ensure that our service will be a singleton, meaning only instantiated once, no matter how many times it is injected elsewhere

The service is generated, but it's not automatically provided. To do this, follow these steps:

- Open app.module.ts
- Type in WeatherService inside the providers array
- Use the auto-fixer to import the class for you:

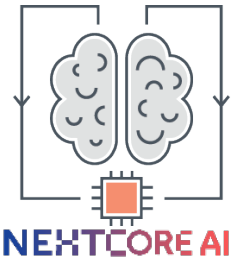
```
src/app/app.module.ts
...
import { WeatherService } from './weather/weather.service'
...
@NgModule({
  ...
  providers: [WeatherService],
  ...
```



Inject dependencies

- Add HttpClientModule to app.module.ts, as follows:

```
src/app/app.module.ts
...
import { HttpClientModule } from'@angular/common/http'
...
@NgModule({
  ...
  imports: [
    ...
    HttpClientModule,
    ...
```



HttpClient

- Inject HttpClient provided by the HttpClientModule in the WeatherService, as follows:

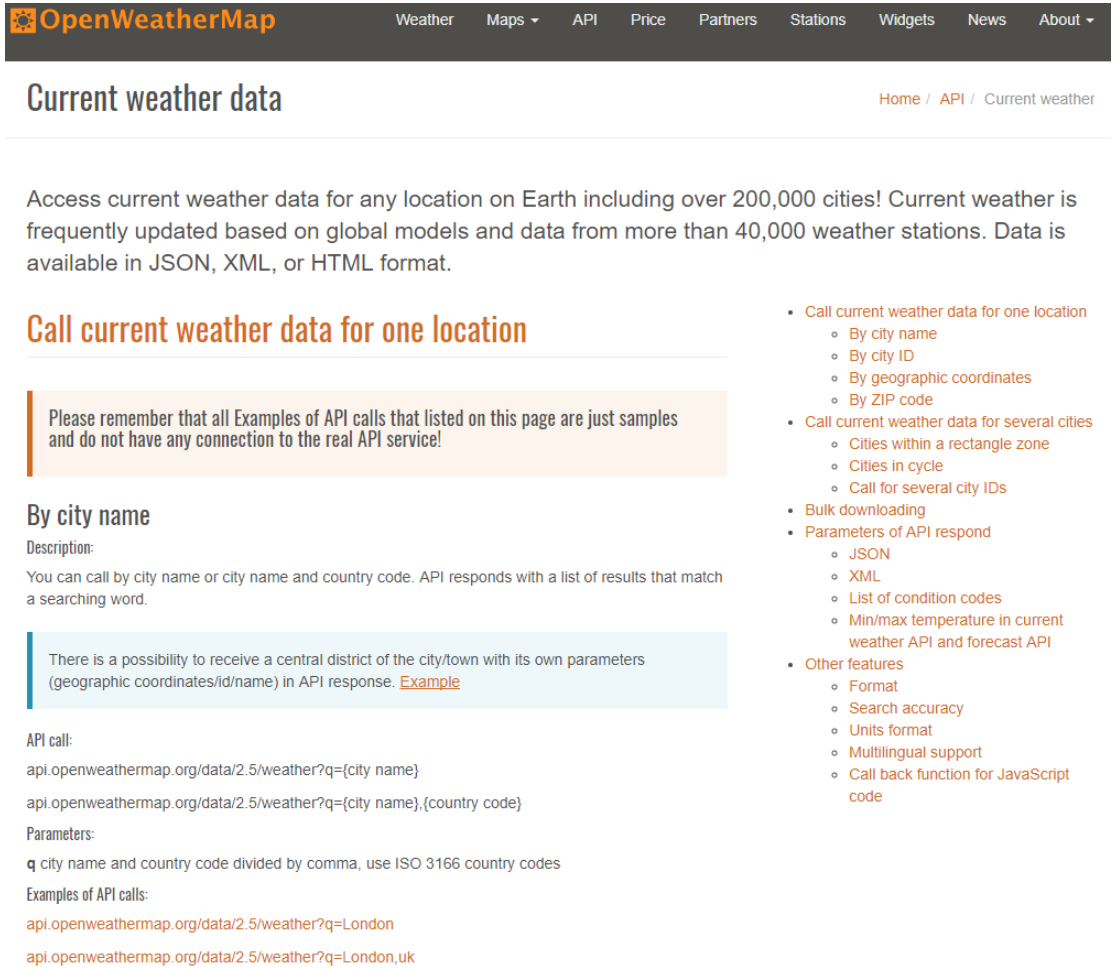
src/app/weather/weather.service.ts

```
import { HttpClient } from '@angular/common/http'
import { Injectable } from '@angular/core'

@Injectable()
export class WeatherService {
  constructor(private httpClient: HttpClient) {}
}
```

OpenWeatherMap APIs

- Read documentation by navigating to http://openweathermap.org/current:



OpenWeatherMap Weather Maps API Price Partners Stations Widgets News About

Current weather data

Home / API / Current weather

Access current weather data for any location on Earth including over 200,000 cities! Current weather is frequently updated based on global models and data from more than 40,000 weather stations. Data is available in JSON, XML, or HTML format.

Call current weather data for one location

Please remember that all Examples of API calls that listed on this page are just samples and do not have any connection to the real API service!

By city name

Description:

You can call by city name or city name and country code. API responds with a list of results that match a searching word.

There is a possibility to receive a central district of the city/town with its own parameters (geographic coordinates/id/name) in API response. Example

API call:

api.openweathermap.org/data/2.5/weather?q={city name}

api.openweathermap.org/data/2.5/weather?q={city name},{country code}

Parameters:

q city name and country code divided by comma, use ISO 3166 country codes

Examples of API calls:

api.openweathermap.org/data/2.5/weather?q=London

api.openweathermap.org/data/2.5/weather?q=London,uk

- Call current weather data for one location
  - By city name
  - By city ID
  - By geographic coordinates
  - By ZIP code
- Call current weather data for several cities
  - Cities within a rectangle zone
  - Cities in cycle
  - Call for several city IDs
- Bulk downloading
- Parameters of API respond
  - JSON
  - XML
  - List of condition codes
  - Min/max temperature in current weather API and forecast API
- Other features
  - Format
  - Search accuracy
  - Units format
  - Multilingual support
  - Call back function for JavaScript code

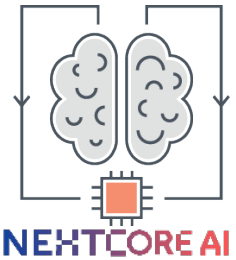
API Documentation

You will be using the API named By city name, which allows you to get current weather data by providing the city name as a parameter. So, your web request will look like this:

- api.openweathermap.org/data/2.5/weather?q={city name},{country code}

- On the documentation page, click on the link under Example of API calls, and you will see a sample response like the following code snippet:

```
http://samples.openweathermap.org/data/2.5/weather?q=London,uk&appid=b1b15e88fa797225412429c1c50c122a1
{
  "coord": {
    "lon": -0.13,
    "lat": 51.51
  },
  "weather": [
    {
      "id": 300,
      "main": "Drizzle",
      "description": "light intensity drizzle",
      "icon": "09d"
```



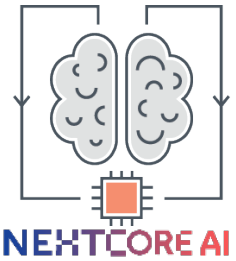
ICurrentWeather interface

- Given the existing `ICurrentWeather` interface that you have already created, this response contains more information than you need. So you will write a new interface that conforms to the shape of this response, but only specify the pieces of data you will use. This interface will only exist in the `WeatherService` and we won't export it, since the other parts of the application don't need to know about this type.

- Create a new interface named `ICurrentWeatherData` in `weather.service.ts` between the `import` and `@Injectable` statements
- The new interface should like this:

src/app/weather/weather.service.ts

```
interface ICurrentWeatherData {
  weather: [{
    description: string,
    icon: string
  }],
  main: {
    temp: number
  },
  sys: {
    country: string
  },
  dt: number,
  name: string
}
```



Storing environment variables

- Copy your appid, which will have a long string of characters and numbers
- Store your appid in environment.ts
- Configure baseUrl for later use:

src/environments/environment.ts

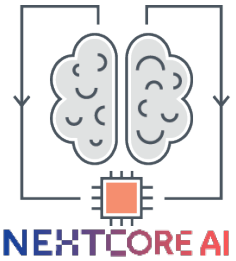
```
export const environment = {
  production: false,
  appId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  baseUrl: 'http://',
}
```

Implementing an HTTP GET operation

- Add a new function to the WeatherService class named getCurrentWeather
- Import the environment object
- Implement the httpClient.get function
- Return the results of the HTTP call:


```
src/app/weather/weather.service.ts
import { environment } from '../../environments/environment'
...
export class WeatherService {
  constructor(private httpClient: HttpClient) { }

  getCurrentWeather(city: string, country: string) {
    return this.httpClient.get<ICurrentWeatherData>(
        `${environment.baseUrl}api.openweathermap.org/data/2.5/weather?` +
          `q=${city},${country}&appid=${environment.appId}`
    )
  }
}
```



Retrieving service data from a component

- Inject the `WeatherService` into the constructor of the `CurrentWeatherComponent` class
- Remove the existing code that created the dummy data in the constructor:

src/app/current-weather/current-weather.component.ts

```
constructor(private weatherService: WeatherService) { }
```

- Call the `getCurrentWeather` function inside the `ngOnInit` function:

src/app/current-weather/current-weather.component.ts

```
ngOnInit() {
  this.weatherService.getCurrentWeather('Bethesda', 'US')
    .subscribe((data) => this.current = data)
}
```