Monday: HTTP service

Create an HTTP service

We earlier saw that services are very helpful in dependency injection. To clean up our app's code, let's create a service that will deliver our quotes to any component that needs them.

Environment Variables

As we clean up our code, it is also necessary and best practice to keep crucial data like API keys, database passwords, e.t.c, away from prying eyes in our apps. The reason is that when we push this kind of data, it puts our app at the risk of malicious attacks since any user can see and interfere with your code on Github if you're not using a private repository, making the app insecure for users to interact with. To hide this kind of data, we put it in a file or folder which we then include in a **gitignore** file and when we push our code to production, all the folders or files in the **gitignore** file are not exposed to users and are not being tracked on version control. In our case, we will use the environment file in the environments folder to hide our API's url. Let's open this file and include this code in the file:

src/environments/environment.ts

```
export const environment = {
  production: false,
  apiUrl:"http://quotes.stormconsultancy.co.uk/random.json"
};
```

We have created a property apiUrl and assigned it to our random quotes API url. Please note that we have put the API url in this file for practice, assuming that it is as crucial as a database password. This means when we push our code to Github and deploy our app, the url will be hidden and therefore requests to the API will not happen. This also implies that we will always receive an error message and our error quote in deployment. In future, we'll hide more crucial data, unlike the url in this case.

To hide this file, let's go to our .gitignore file and add the file there.

Goals/.gitignore

```
....
environment.ts
```

Our API url is now hidden whenever we push our code to Github. In our terminals, let's execute this command to create the service for our quotes:

```
$ ng generate service quote-http/quote-request
```

Let's write the code for our service in the service class.

HTTP requests using Promises

A promise is an object representing the eventual completion or failure of an asynchronous process. Our asynchronous process here is the request we are making to the API.

Let's use a promise in our service.

src/app/quote-http/quote-request.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import {environment } from '../../environments/environment';
import { Quote } from '../quote-class/quote';


@Injectable({
  providedIn: 'root'
})
export class QuoteRequestService {

  quote: Quote;

  constructor(private http:HttpClient) {
    this.quote = new Quote("","");
   }

   quoteRequest(){
     interface ApiResponse{
       quote:string;
       author:string;
     }
     let promise = new Promise((resolve,reject)=>{
       this.http.get<ApiResponse>(environment.apiUrl).toPromise().then(response=>{
         this.quote.quote = response.quote
         this.quote.author = response.author

         resolve()
       },
       error=>{
         this.quote.quote = "Never, never, never give up"
         this.quote.author = "Winston Churchill"

         reject(error)
       })
     })
     return promise
   }
}
```

At the top, we have imported HttpClient to enable us to make a request to the API, the Quote class and the environment class in which we put our API url. We have then created a property quote and assigned it the type of our Quote class initializing it with empty strings inside the constructor function. We have also injected a private http property of the type HttpClient in the constructor.

We have then defined a quoteRequest() method that defines the ApiResponse interface and the promise instance. We have then used the get function and passed in the apiUrl from the environment object with the interface. We have used toPromise() to convert the Http Request to a promise. We have called the then function and passed in the response and error functions as arguments. The response function is called when the HTTP request is successful and returns a response. If successful, we update the properties of the quote instance with values from the response and call the resolve function. If we encounter an error, we have passed in default values for creating a quote instance in the error function which is called when there is an error.

Let's now call the service in our goal component.

src/app/goal/goal.component.ts

```
....
import { GoalService } from '../goal-service/goal.service';
import { AlertService } from '../alert-service/alert.service';
import { QuoteRequestService } from '../quote-http/quote-request.service';

export class GoalComponent implements OnInit {
...
  quote:Quote;
...
  constructor(goalService:GoalService, alertService:AlertService, private quoteService:QuoteRequest
Service) {
    this.goals = goalService.getGoals()
    this.alertService = alertService;
  }

  ngOnInit() {

    this.quoteService.quoteRequest()
    this.quote = this.quoteService.quote
  }
```

We have imported our QuoteRequestService at the top and then injected it into our constructor. Inside the ngOnInit lifecycle hook, we have called the quoteRequest() method from the service and created a quote instance with the promise object we will receive from the service. If we run our server at this point, our quotes will be displayed on the application as usual but using a service this time round.