Angular

VI Lecture

KindGeek

HttpClient

Setup

```
import { NgModule }      from '@angular/core';
import { BrowserModule }  from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';

@NgModule({
  imports: [
    BrowserModule,
    // import HttpClientModule after BrowserModule.
    HttpClientModule,
  ],
  declarations: [
    AppComponent,
  ],
  bootstrap: [ AppComponent ]
})
export class AppModule {}
```

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

@Injectable()
export class ConfigService {
  constructor(private http: HttpClient) { }
}
```

KindGeek School

Getting JSON data

```
{
  "heroesUrl": "api/heroes",
  "textfile": "assets/textfile.txt"
}
```

```
export interface Config {
  heroesUrl: string;
  textfile: string;
}
```

```
// ConfigService
getConfig() {
  // now returns an Observable of Config
  return this.http.get<Config>(this.configUrl);
}
```

```
// ConfigComponent
config: Config;

showConfig() {
  this.configService.getConfig()
    // clone the data object, using its known Config shape
    .subscribe((data: Config) => this.config = { ...data });
}
```

KindGeek School

Reading the full response

```
getConfigResponse(): Observable<HttpResponse<Config>> {
  return this.http.get<Config>(
    this.configUrl, { observe: 'response' });
}
```

```
showConfigResponse() {
  this.configService.getConfigResponse()
    // resp is of type `HttpResponse<Config>`
    .subscribe(resp => {
      // display its headers
      const keys = resp.headers.keys();
      this.headers = keys.map(key =>
        `${key}: ${resp.headers.get(key)}` );

      // access the body directly, which is typed as `Config`.
      this.config = { ... resp.body };
    });
}
```

KindGeek School

Error handling

```
showConfig() {
  this.configService.getConfig()
    .subscribe(
      (data: Config) => this.config = { ...data }, // success path
      error => this.error = error // error path
    );
}
```

KindGeek School

Getting error details

```
private handleError(error: HttpErrorResponse) {
  if (error.error instanceof ErrorEvent) {
    // A client-side or network error occurred. Handle it accordingly.
    console.error('An error occurred:', error.error.message);
  } else {
    // The backend returned an unsuccessful response code.
    // The response body may contain clues as to what went wrong,
    console.error(
      `Backend returned code ${error.status}, `  +
      `body was: ${error.error}`);
  }
  // return an observable with a user-facing error message
  return throwError(
    'Something bad happened; please try again later.');
};

getConfig() {
  return this.http.get<Config>(this.configUrl)
    .pipe(
      catchError(this.handleError)
    );
}
```

KindGeek School

retry()

```
getConfig() {
  return this.http.get<Config>(this.configUrl)
    .pipe(
      retry(3), // retry a failed request up to 3 times
      catchError(this.handleError) // then handle the error
    );
}
```

KindGeek School

Observables and operators

```
import { Observable, throwError } from 'rxjs';
import { catchError, retry } from 'rxjs/operators';
```

KindGeek School

Sending data to the server

Adding header

```
import { HttpHeaders } from '@angular/common/http';

const httpOptions = {
  headers: new HttpHeaders({
    'Content-Type':  'application/json',
    'Authorization': 'my-auth-token'
  })
};
```

KindGeek School

Making a POST request

1. hero - the data to POST in the body of the request.
2. httpOptions - the method options which, in this case, specify required headers.

```
// HeroService
addHero (hero: Hero): Observable<Hero> {
  return this.http.post<Hero>(this.heroesUrl, hero,
httpOptions)
    .pipe(
      catchError(this.handleError('addHero', hero))
    );
}

// HeroComponent
this.heroesService.addHero(newHero)
  .subscribe(hero => this.heroes.push(hero));
```

KindGeek School

Making a DELETE request

```
/** DELETE: delete the hero from the server */
deleteHero (id: number): Observable<{}> {
  const url = `${this.heroesUrl}/${id}` ; // DELETE api/heroes/42
  return this.http.delete(url, httpOptions)
    .pipe(
      catchError(this.handleError('deleteHero'))
    );
}
```

KindGeek School

Making a PUT request

```
/** PUT: update the hero on the server. Returns the updated hero upon success. */
updateHero (hero: Hero): Observable<Hero> {
  return this.http.put<Hero>(this.heroesUrl, hero, httpOptions)
    .pipe(
      catchError(this.handleError('updateHero', hero))
    );
}
```

KindGeek School

Debouncing requests

```
<input (keyup)="search($event.target.value)" id="name" placeholder="Search" />
<ul>
  <li *ngFor="let package of packages$ | async" >
    <b>{{package.name}} v.{{package.version}} </b> -
    <i>{{package.description}} </i>
  </li>
</ul>
```

```
withRefresh = false;
packages$: Observable<NpmPackageInfo []>;
private searchText$ = new Subject<string>();

search(packageName: string) {
 this.searchText$.next(packageName);
}
ngOnInit() {
 this.packages$ = this.searchText$.pipe(
   debounceTime(500),
   distinctUntilChanged(),
   switchMap(packageName =>
     this.searchService.search(packageName, this.withRefresh))
 );
}

constructor(private searchService: PackageSearchService) { }
```

KindGeek School

- debounceTime(500) - wait for the user to stop typing (1/2 second in this case).
- distinctUntilChanged() - wait until the search text changes.
- switchMap() - send the search request to the service.

switchMap()

The switchMap() operator has three important characteristics.

1. It takes a function argument that returns an Observable. PackageSearchService.search returns an Observable, as other data service methods do.
2. If a previous search request is still in-flight (as when the connection is poor), it cancels that request and sends a new one.
3. It returns service responses in their original request order, even if the server returns them out of order.

KindGeek School

Home task

Build documentation for Open Api
https://postcodes.io/

https://angular.io/guide/http#httpclient