Angular REST

Autori:
Milan Segedinac
Goran Savić

1. REST pozivi iz klijentske aplikacije

U višeslojnim veb aplikacijama back end deo aplikacije zadužen je za rukovanje podacima (perzistencija, pronalaženje, ...), a front end deo aplikacije obezbeđuje interakciju sa korisnikom. Ta interakcija uključuje prikazivanje podataka, unos novih i izmenu postojećih podaatka i slično. Stoga radni okviri za razvoj front end aplikacija trebaju da obezbede dobre mehanizme razmene podataka sa back end delom aplikacije, odnosno trebaju da pruže jednostavnu podršku za AJAX.

Pre nego što budemo videli kako se implementira razmena podataka sa REST back end-om realizuje u radnom okviru Angular moramo da pogledamo kako je realizovana injekcija zavisnosti pomoću koje se pribavljaju objekti potrebni za ovu komunikaciju.

Injekcija zavisnosti u Angular radnom okviru

Objekat može doći do svojih zavisnosti, odnosno drugih objekata koje koristi, na tri načina:

1. Instanciranje – tipično korišćenjem new operatora ili factory funkcije
2. Pronalaženjem (*look up*) – referenciranjem globalnih varijabli
3. Injekcijom zavisnosti – u tom slučaju radni okvir se brine o životnom ciklusu zavisnosti. Objekat samo treba da ih zatražio od radnog okvira.

Prvi dva načina rukovanja zavisnostima – instanicranje i pronalaženje – nameći da objekti budu čvrsto spregnuti. Na primer, ako smo zavisnosti dobili instanciranjem, jedini način da izmenimo zavisnosti je da menjamo programski kod programa. Ovo je posebno problematično prilikom testiranja, jer nam u tim situacijama često treba da mokujemo zavisnosti zarad izolovanosti testa.

Treća opcija – injekcija zavisnosti – rukovanje (instanciranje, konfigurisanje, lociranje, ...) zavisnostima prepušta radnom okviru, pa je moguće zavisnosti izmeniti na nivou konfiguracije. Zbog praktičnosti ovakvog pristupa, Angular se snažno oslanja na mehanizam inejkcije zavisnosti.

Angular sadrži zaseban radni okvir za injekciju zavisnosti. Iako se ovaj modul vrlo intezivno koristi u Angular aplikacijama, moguće ga je koristiti i nezavisnosno u bilo kojoj TypeScript aplikaciji.

Da bi neki objekat mogao da se injektuje u Angularu potrebno je anotirati da je injektabilan. To se ostvaruje tako što se klasa tog objekta anotira dekoratorom `@Injectable()`. Pored toga, potrebnoje registrovati ga. To se najčešće radi tako što se u anotaciji komponente koja ga koristi naziv klase injektabilnog objekta navede kao element niza na atributu `providers`.

Mehanizam injekcije zavisnosti u Angularu je iskorišćen za pribavljanje objekta koji rukuje http komunikacijom, odnosno `Http` objekta. Pogledajmo na listingu ispod kako je `Http` objekat pribavljen.

```
...

import { Http, Response, RequestOptions,
         Headers, URLSearchParams } from '@angular/http';
...
constructor(private http: Http){
  ...
}
```

Listing - injekcija zavisnosti (`app.component.ts`)

Vidimo da se injekcija zavisnosti ostvaruje tako što se za konstruktor komponente postave parametri koje hoćemo da pribavimo, a Angular je zadužen da ih dobavi. U ovom primeru susrećemo se i sa skraćenom notacijom postavljanja vrednosti u konstruktoru u programskom jeziku TypeScript. Kod `constructor(private http: Http)` je ekvivalenta sa kodom datim listingom ispod.

```
...

private http;
constructor(http: Http){
  this.http = http;
}
...
```

Listing - skraćena notacija postavljanja polja

Mehanizmi obrade asinhronih poziva

U Angularu `Http` objekat zadužen je za komunikaciju sa back end delom aplikacije. Ova komunikacija se obavlja tako što klijentska aplikacija šalje zahteve back end delu aplikacije, a back end deo aplikacije šalje odgovore klijentu. Ova komunikacija je po svojoj prirodi asinhrona. Postoji nekoliko tipičnih načina na koji se rukuje asinhronim izvršavanjem programa.

1. *Callback funkcije*
2. *Promise objekti*
3. *Observable*

Sa callback funkcijama kao mehanzmom za rukovanje asinhronim izvršavanjem programa smo se susreli kada je bilo reči o jQuery biblioteci. Setimo se da je ovaj mehanizm podrazumevao da se funkciji koja se izvršava asinhrono prosledi funkcija koja će se izvršiti nakon njenog završetka. Na primer, funkciji koja rukuje http zahtevima bismo prosledili callback funkciju koja obrađuje pristigli http odgovor.

Drugi način koji bismo mogli da koristimo je da funkcija koja se izvršava asinhrono odmah nakon poziva, a pre nego što se njeno izvršavanje završilo vrati objekat koji predstavlja *čekanje završetka izvršavanja*. Ovaj objekat se zove promise objeka jer modeluje obećanje da će se asinhrona funkcija završiti. Promise objekat ima metodu koja modeluje završetak izvršavanja funkcije (*then* metodu) i koja prima callback funkciju koja će se pozvati nakon završetka.

Observable su generalizacija promise objekata. Dok promise objekat rukuje jednim događajem i then funkcija modeluje da se taj događaj završio, observabla modeluje seriju događaja i omogućuje reakciju na svaki od njih. Pored toga, observablu je moguće i odkazati, čime se prekida obrada serije događaja. Methoda `http.get(url)` upućuje zahtev na zadati url i vraća objekat tipa `Observable`. Nad vraćenim obesrvable objektom je moguće pozvati metodu `subscribe`, pomoću koje zadajemo callback funkciju koja će se pozvati svaki put kada se posmatrani događaj dogodi. Obzirom da je događaj koji očekujemo pristizanje http odgovora, callbacke funcija prosleđenja subscribe metodi će se tipično izvršiti samo jednom. Kao parametar ove funkcije biće postavljen objekat koji modeluje http odgovor i biće tipa `Response`. Kod je dat listingom ispod.

```
this.http.get('/api/records').subscribe(

      //koristimo arrow funkciju da bismo imali leksicki
      //opseg this objekata
      (res: Response) => {
        this.records=res.json();
      }
  );
```

Listing – Slanje http zahteva i obrada odgovora

Listingom iznad upućen je HTTP get zahtev na relativni URL `'/api/records'`. Kada se dobije odgovor sa uspešnim statusom biće pozvana funkcija koja će kao parametar `res` primiti odogovor. Jeidno što će biti urađeno u telu te funkcije je postavljanje pristiglih podataka (parsiranih iz JSON formata pozivom `json()` metode) u `this.records`. Obratimo pažnju na to da je callback funkcija napisana u arrow notaciji. To nismo uradili samo da bismo uštedeli nekoliko karaktera, već zbog toga što arrow funkcije imaju leksički bajndovan `this` objekat, tako da možemo da izvršimo dodelu vrednosti za `this.records`.

Pored get metode, `Http` klasa ima i druge motode koje omogućuju HTTP komunikaciju. API ove klase dat je listingom ispod.

```
class Http {

  protected _backend: ConnectionBackend
  protected _defaultOptions: RequestOptions
     request(url: string|Request, options?: RequestOptionsArgs):
Observable<Response>
     get(url: string, options?: RequestOptionsArgs):
Observable<Response>
    post(url: string, body: any, options?: RequestOptionsArgs):
Observable<Response>
     put(url: string, body: any, options?: RequestOptionsArgs):
Observable<Response>
     delete(url: string, options?: RequestOptionsArgs):
Observable<Response>
    patch(url: string, body: any, options?: RequestOptionsArgs):
Observable<Response>
     head(url: string, options?: RequestOptionsArgs):
Observable<Response>
     options(url: string, options?: RequestOptionsArgs):
Observable<Response>
}
```

Listing – Klasa Http

Od navedenih metoda najčešće ćemo koristiti get, post, put i delete. Kao što vidimo, sve ove metode kao prvi parametar imaju string URL na koji se upućuje zahtev. Metode koje šalju telo zahteva (put i post) kao drugi parametar imaju objekat koji reprezentuje telo zahteva. Poslednji parametar je `options` i on je opcioni. Pomoću njega se šalju query parametari, postavljaju zaglavalja i slično. Primer slanja zahteva sa telom dat je listingom ispod.

```
save(newRecord:Record){

     let headers = new Headers({ 'Content-Type': 'application/
json' });
  let options = new RequestOptions({ headers: headers });
                                    this.http.post('/api/
records',JSON.stringify(newRecord),options).
   subscribe(
    (res: Response) => {
      this.loadData();
    }
  );
}
```

Listing – Http post zahtev

Na listingu iznad smo poslali post zahtev na `'/api/records'`. Kao telo zahteva poslali smo `newRecord` objekat pretvoren u JSON string. Kroz jedno dodatno zaglavlje smo postavili da je Content type application/json.

Listingom ispod dat je interfejs `RequestOptionsArgs`.

```
interface RequestOptionsArgs {

  url?: string|null
  method?: string|RequestMethod|null
    search?: string|URLSearchParams|{[key: string]: any | any[]}|
null
    params?: string|URLSearchParams|{[key: string]: any | any[]}|
null
  headers?: Headers|null
  body?: any
  withCredentials?: boolean|null
  responseType?: ResponseContentType|null
}
```

Listing - interfejs `RequestOptionsArgs`

Kao što vidimo, pored zaglavlja koja se zadaju atributom headers, možemo zadati i URL na koji se šalje zahtev, HTTP metodu, , query parametre, telo zahteva, kredencijale i očekivani tip odgovora. Tako širok skup atributa omogućuje nam da bilo koji zahtev možemo uputiti korišćenjem jedne jedine metode `request` klase `Http` koja prima samo URL na koji se upućuje zahtev i `options` objekat. U stvari, ostale specifične metode ove klase (`get`, `post`, ...) samo su skraćeni zapis tipične pozive za `request` metodu. Vidimo i da su svi atributi `RequestOptionsArgs` interfejsa opcioni što nam pruža punu slobodu u kreiranju objekata tog tipa.