



IT255 - VEB SISTEMI 1

HTTP i Angular

Lekcija 09

PRIRUČNIK ZA STUDENTE

# IT255 - VEB SISTEMI 1

## Lekcija 09

### *HTTP I ANGULAR*

- ✓ HTTP i Angular
- ✓ Poglavlje 1: Primena angular/common/http
- ✓ Poglavlje 2: Kreiranje YouTubeSearchComponent komponente
- ✓ Poglavlje 3: angular/common/http API - ostali HTTP zahtevi
- ✓ Poglavlje 4: Vežba 9
- ✓ Poglavlje 5: Domaći zadatak 9
- ✓ Zaključak

Copyright © 2017 – UNIVERZITET METROPOLITAN, Beograd. Sva prava zadržana. Bez prethodne pismene dozvole od strane Univerziteta METROPOLITAN zabranjena je reprodukcija, transfer, distribucija ili memorisanje nekog dela ili čitavih sadržaja ovog dokumenta., kopiranjem, snimanjem, elektronskim putem, skeniranjem ili na bilo koji drugi način.

Copyright © 2017 BELGRADE METROPOLITAN UNIVERSITY. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, without the prior written permission of Belgrade Metropolitan University.

# ▼ Uvod

## UVOD

*Angular poseduje HTTP biblioteku koja može biti pogodna za obavljanje poziva ka eksternim API*

Radni okvir *Angular* distribuira vlastitu *HTTP biblioteku* koja može biti veoma pogodna za obavljanje poziva ka eksternim *API* - jima.

Kada se obavljaju pozivi ka eksternim serverima, neophodno je obezbediti da korisniku kontinuiranu interakciju sa stranicom veb aplikacije. To praktično znači, da je neophodno sprečiti da se stranica "zamrzne" dok ne stigne HTTP odgovor sa servera. Da bi navedeno bilo moguće realizovati, **HTTP zahtevi moraju biti asinhroni**.

Rad sa asinhronim kodom, a to ima i istorijsko uporište u programiranju, može biti složeniji nego rukovanje sinhronim kodom.

U JavaScript jeziku, postoje tri pristupa za pomoć u rukovanju asinhronim kodom:

1. povratni pozivi (*callback*);
2. obećanja (*promises*);
3. osmatranja (*observable*).

**U Angular okviru preferirani metod manipulisanja asinhronim kodom predstavlja treći slučaj - osmatranje**, na čemu će biti i bazirano izlaganje u ovoj lekciji.

Lekcija će pokriti tri ključne stvaki:

1. analiza i diskusija osnovnog *HttpClient* primera;
2. kreiranje *YouTube* komponente za pretragu;
3. diskusija o API detaljima vezanim za *HttpClient* biblioteku.

Savladavanjem ove lekcije student će razumeti i biti u mogućnosti da rukuje asinhronim HTTP zahtevima u Angular aplikacijama.

## ▼ Poglavlje 1

# Primena angular/common/http

## HTTP U ANGULARU

*Podrška radu sa HTTP protokolom je u Angularu izdvojena u unutar posebnog modula.*

Podrška radu sa HTTP protokolom je u Angularu izdvojena u unutar posebnog modula. To, najjednostavnije rečeno, znači ukoliko je neophodna navedena podrška za kreiranje koda, tekućeg Angular projekta, neophodno je obaviti uvoz iz paketa [@angular/common/http](#), na sledeći način:

```
import {  
  // NgModule za upotrebu @angular/common/http  
  HttpClientModule,  
  
  // konstante klase  
  HttpClient  
} from '@angular/common/http';
```

## UVOZ IZ ANGULAR/COMMON/HTTP

*HttpClientModule predstavlja kolekciju pogodnih modula.*

Ukoliko je potrebno da u veb aplikaciji koja se kreira primenu nađe i podrška za rad sa HTTP protokolom, prvi korak koji je neophodno obaviti jeste uvoz biblioteke [HttpClientModule](#) koja predstavlja kolekciju pogodnih modula.

Kao i u prethodnim lekcijama biće uveden adekvatan primer za lakše razumevanje i praćenje gradiva. Primer se jednostavno naziva [http](#) i moguće ga je preuzeti na kraju lekcije u sekciji [Shared Resources](#).

Dakle, prvi korak jeste importovanje navedenih modula na način urađen u sledećem listingu koji predstavlja deo sadržaja datoteke [app.module.ts](#).

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

U narednom koraku, u dekoratoru `@NgModule`, neophodno je dodati `HttpClientModule` u listu importovanih modula. Ovim se postiže efekat mogućnosti umetanja `Http` (i nekoliko drugih modula) unutar komponenta Angular aplikacije.

```
@NgModule({
  declarations: [
    AppComponent,
    SimpleHttpComponent,
    MoreHttpRequestsComponent,
    YouTubeSearchComponent,
    SearchResultComponent,
    SearchBoxComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule // <-- right here
  ],
  providers: [YouTubeSearchInjectables],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Sada je moguće obaviti umetanje `HttpClient` servisa u komponente ili bilo gde drugo u programu gde se koristi umetanje zavisnosti, na primer kao na sledećoj slici:

```
class MojaKomponenta {
  constructor(public http: HttpClient) {

  }

  makeRequest(): void {
    // uradi nešto sa this.http ...
  }
}
```

Slika 1.1 Umetanje `HttpClient` putem konstruktora klase komponente

## UKOVANJE OSNOVNIM ZAHTEVOM

### *Pokazivanje rukovanja osnovnim GET HTTP zahtevom*

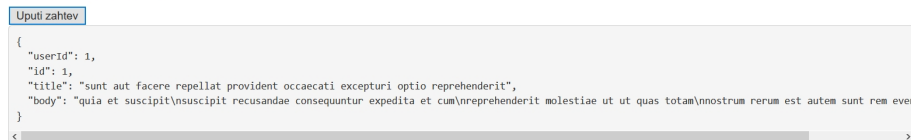
U nastavku je cilj pokazivanje rukovanja osnovnim `GET` HTTP zahtevom. Zbog svega navedenog iskristalisala jedna ideja - prosleđivanje jednostavnog `GET` zahteva ka [jsonplaceholder API](https://jsonplaceholder.typicode.com/) (<https://jsonplaceholder.typicode.com/>). Neophodno je obaviti sledeće:

1. obezbediti dugme za poziv konkretne metode `makeRequest()`;
2. metoda `makeRequest()` će pozvati `http` biblioteku za izvođenje `GET` zahteva nad izabranim `API`;
3. kada se vrati odgovor na prosleđeni zahtev, biće ažurirana vrednost `this.data` rezultatima u formi podataka koji će se koristiti za kreiranje pogleda veb aplikacije.

Sledećom slikom je prikazan izlaz u formi kreiranog pogleda na osnovu vraćenog odgovora na prosleđeni [GET](#) zahtev ka [API](#).

### Angular HTTP primer

Osnovni zahtev



Slika 1.2 Izvršavanje osnovnog GET zahteva

## KREIRANJE DEFINICIJE KOMPONENTE SIMPLEHTTPCOMPONENT

*Importovanje odgovarajućih modula i specifikacija selektora unutar dekoratora komponente.*

Prethodnom slikom je demonstrirana funkcionalnost komponente aplikacije koja bi u narednom izlaganju trebalo da bude definisana i analizirana. Prvo što je potrebno učiniti jeste importovanje odgovarajućih modula i specifikacija selektora unutar dekoratora komponente `@Component` koja se u projektu čuva na lokaciji `src/app/simple-http/simple-http.component.ts`.

```
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Component({
  selector: 'app-simple-http',
  templateUrl: './simple-http.component.html'
})
export class SimpleHttpComponent implements OnInit {
  data: Object;
  loading: boolean;

  constructor(private http: HttpClient) {}
```

Priloženim listingom nije završena definicija klase komponente. O ostatku listinga, između ostalog i pomenutoj metodi `makeRequest()`, biće detaljno govora u nastavku.

## KREIRANJE ŠABLONA KOMPONENTE SIMPLEHTTPCOMPONENT

*Za nastavak analize od presudnog značaja je kreiranje šablona komponente.*

U prethodnom izlaganju data je delimična definicija klase kreirane komponente *SimpleHttpComponent*. Za nastavak analize od presudnog značaja je kreiranje šablona komponente *SimpleHttpComponent*.

Sledećim listingom dat je *HTML* kod šablona koji se čuva u datoteci *src/app/simple-http/simple-http.component.html*:

```
<h2>Osnovni zahtev</h2>
<button type="button" (click)="makeRequest()">Uputi zahtev</button>
<div *ngIf="loading">loading...</div>
<pre>{{data | json}}</pre>
```

Kada se pogleda priloženi listing, moguće je primetiti da šablon sadrži tri zanimljiva detalja:

- dugme "Uputi zahtev";
- indikator učitavanja zahetva (loading);
- podatke.

Kontrola dugme je povezana akcijom (click) sa pozivom metode *makeRequest()* kontrolera (klase) komponente. Ova metoda će biti definisana uskoro,

Indikator pokazuje korisniku da se prosleđeni zahtev učitava, prezentacijom stringa "loading...", ukoliko objektna promenljiva *loading* poseduje vrednost *true* u direktivi grananja *ngIf*.

Promenljiva *data* predstavlja objekat. Odličan način rukovanja objektima jeste njihovo prikazivanje u *JSON* (*JavaScript Object Notation*) formatu. Na ovaj način dobija se veoma čitljiv i lako upotrebljiv format prikaza objekata.

## KREIRANJE SIMPLEHTTPCOMPONENT KONTROLERA

*Ono što klasi nedostaje su konstruktor i metoda makeRequest() koja šalje GET zahtev ka serveru.*

Ono što trenutno postoji, a odnosi se na primer koji služi kao podrška trenutnom izlaganju, jeste osnovna definicija klase *src/app/simple-http/simple-http.component.ts*:

```
export class SimpleHttpComponent implements OnInit {
  data: Object;
  loading: boolean;
```

Klasa poseduje dve objektno promenljive: *data* i *loading* koje će biti upotrebljene da sačuvaju API povratnu vrednost i indikator učitavanja zahteva, respektivno.

Ono što klasi nedostaje, da bi njena definicija bila zaokružena, su:

- konstruktor;
- metoda *makeRequest()* koja šalje *GET* zahtev ka serveru;

Dakle, u nastavku je prvo neophodno dodati u kod konstruktor, kao na sledeći način:

```
constructor(private http: HttpClient) {}
```

Telo konstruktora je prazno ali ipak, konstruktor ovde obavlja jednu važnu ulogu - vrši umetanje jednog od ključnih modula *HttpClient*.

Sada je sve spremno za kodiranje metode kojom će biti omogućeno kreiranje prvog HTTP zahteva. Ova metoda kontrolera se naziva *makeRequest()* i njen kod je priložen u sledećem listingu:

```
makeRequest(): void {  
  this.loading = true;  
  this.http  
    .get('https://jsonplaceholder.typicode.com/posts/1')  
    .subscribe(data => {  
      this.data = data;  
      this.loading = false;  
    });  
}
```

## ANALIZA KODA KONTROLERA

*this.http.get()* prihvata URL prema kojem se upućuje zahtev.

Kada se pozove funkcija *makeRequest()*, prva stvar koja se dešava je podešavanje vrednosti objektna promenljive: *this.loading = true*. Ovom akcijom će biti uključen, i prikazan u pogledu, indikator učitavanja zahteva (slika 3).

Prosleđivanje HTTP zahteva je veoma jednostavno: poziva se *get()* metoda na sledeći način: *this.http.get()* kojoj se prosleđuje, kao argument, URL prema kojem se upućuje zahtev. Metoda kao rezultat vraća *Observable* objekat koji osmatra promene u podacima nakon prosleđenog zahteva.

Dalje, *Observable* objekat, da bi obavio svoju dužnost, prijavljuje se da osmatra promene pozivom funkcije *subscribe()* - slično kao *Promise* u nativnom *JavaScript*-u.

Kada se obrađeni zahtev *http.request* vrati od servera, ulazni tok će emitovati odgovor (*response*) kao instancu tipa *Object*. Primenom *JSON*-a vrši se raspakivanje tela objekta čija se vrednost pridružuje objektnoj promenljivoj *this.data*. Budući da je odgovor dobijen, indikator slanja zahteva više nije potreban i podešava se *this.loading=false*. Dobijen odgovor se u pogledu prikazuje izvršavanjem izraza `{{data | json}}` (slika 4).

### Angular HTTP primer

Osnovni zahtev

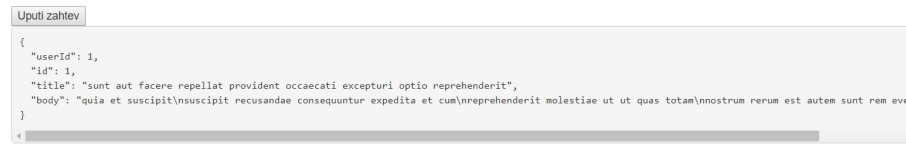
Uputi zahtev  
loading...

Slika 1.3 Indikator slanja zahteva



## Angular HTTP primer

### Osnovni zahtev



Slika 1.4 Dobijeni odgovor od servera

## ZAOKRUŽENA DEFINICIJA KOMPONENTE SIMPLEHTTPCOMPONENT

*Komponenta je prikazivana iz delova, a sada je objedinjena u celinu.*

U prethodnom izlaganju je kreirana komponenta SimpleHttpComponent, a pogotovo njena kontrolerska klasa, prikazivana iz delova koji su posebno obrađivani i diskutovani.

Za sticanje šire slike je, ipak, potrebno delove sklopiti i prikazati u formi funkcionalne celine.

Sledi listing objedinjene klase komponente (datoteka: [src/app/simple-http/simple-http.component.ts](#)):

```
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Component({
  selector: 'app-simple-http',
  templateUrl: './simple-http.component.html'
})
export class SimpleHttpComponent implements OnInit {
  data: Object;
  loading: boolean;

  constructor(private http: HttpClient) {}

  ngOnInit() {}

  makeRequest(): void {
    this.loading = true;
    this.http
      .get('https://jsonplaceholder.typicode.com/posts/1')
      .subscribe(data => {
        this.data = data;
        this.loading = false;
      });
  }
}
```

## ▼ Poglavlje 2

# Kreiranje YouTubeSearchComponent komponente

## POKUŠAJ SLOŽENIJEG API POZIVA

*Biće kreirana komponenta koja sadrži funkcionalnost pretrage YouTube- a kao tipa podataka.*

U prethodnom primeru je pokazano kako Angular veb aplikacija upućuje jednostavan **GET** zahtev ka serveru i kako se prikazuje dobijeni odgovor u šablonu komponente.

U nastavku lekcije, cilj je da se uključe naprednije funkcionalnosti u aplikaciju, a jedna od njih može da bude mehanizam pretrage - biće kreirana komponenta koja sadrži funkcionalnost pretrage YouTube - a kao tipa podataka. Kada je pretraga završena, kao rezultat će biti prikazana lista sa odgovarajućim video klipovima (slika 1) zajedno sa opisom i odgovarajućim linkom za svaki od pronađenih klipova,

Sledećom slikom je prikazan rezultat pretrage, u aplikaciji koja predstavlja primer za podršku izlaganja lekcije, za upit pretrage "*državni posao epizode*".



Slika 2.1 Rezultat pretrage komponente aplikacije

U svetlu realizovanja postavljenih ciljeva, nove komponente aplikacije, biće kreirano nekoliko stvari:

1. Objekat **SearchResult** će čuvati podatke koji se dobijaju iz svakog rezultata pretrage;
2. Servis **YouTubeSearchService** će upravljati API zahtevima ka serveru **YouTube** i obavljati konverziju rezultata u niz **SearchResult[]**;
3. Komponenta **SearchBoxComponent** će imati zadatak poziva YouTube servisa;
4. Komponenta **SearchResultComponent** će kreirati specifičan **SearchResult**;

5. Komponenta *YouTubeSearchComponent* će enkapsulirati celokupnu aplikaciju za pretragu *YouTube* -a i kreirati i prikazati listu rezultata pretrage.

## KLASA SEARCHRESULT

*Klasa je model za komponentu i biće pogodan način čuvanja informacija - rezultata pretrage.*

Razvoj komponente, dobiće zvaničan naziv *you-tube-search*, započinje kreiranjem osnovne klase *SearchResult*. Ova klasa će praktično predstavljati model za komponentu i biće pogodan način čuvanja specifičnih polja koja odgovaraju rezultatima pretrage. Kod klase se čuva u datoteci */src/app/you-tube-search/search-result.model.ts* i priložen je sledećim listingom:

```
/**
 * SearchResult je struktura podataka koja čuva pojedinačne
 * rezultate YouTube video pretrage
 */
export class SearchResult {
  id: string;
  title: string;
  description: string;
  thumbnailUrl: string;
  videoUrl: string;

  constructor(obj?: any) {
    this.id = obj && obj.id || null;
    this.title = obj && obj.title || null;
    this.description = obj && obj.description || null;
    this.thumbnailUrl = obj && obj.thumbnailUrl || null;
    this.videoUrl = obj && obj.videoUrl ||
      `https://www.youtube.com/watch?v=${this.id}`;
  }
}
```

Posebnu pažnju bi trebalo obratiti na deo koda koji je predstavljen konstruktorom klase. Uočava se prvi put šablon u formi *obj?: any*, tako da je ovde prioritet da se razotkrije njegova uloga. Navedeni šablon, zapravo, dozvoljava simuliranje argumenata ključnih reči pretrage. Ideja je da je moguće kreirati nov *SearchResult* objekat i obaviti prosleđivanje objekta koji sadrži specificirane ključne reči.

Jedino što ovde treba istaći je da se konstruiše *videoUrl* koristeći teško kodirani (*hard coded*) URL format. Možda bi ovo bi bilo dobro promeniti u funkciju koja uzima više argumenata ili bi trebalo upotrebiti *id* videa, direktno u HTML šablonu, za kreiranje traženog URL, ako se za to ukaže potreba.

## KREIRANJE SERVISA YOUTUBESEARCHSERVICE - API

*Za kreiranje navedenog servisa biće neophodno koristiti YouTube v3 search API.*

U nastavku sledi kreiranje servisa koji će komponenti omogućiti izvođenje pretrage nad *YouTube* platformom. Za kreiranje navedenog servisa biće neophodno koristiti *YouTube v3 search API* (<https://developers.google.com/youtube/v3/docs/search/list>). Da bi navedeni API mogao da bude upotrebljen neophodno je nabaviti API ključ. U primeru, koji je priložen uz ovu lekciju, ugrađen je API ključ kojeg je moguće koristiti. Međutim, kada student dođe u fazu čitanja i izučavanja ove lekcije, upotreba navedenog ključa je možda premašila dozvoljeni broj korišćenja. Ukoliko se to desi, neophodno je da studenti nabave vlastiti API ključ.

Da bi student nabavio vlastiti API ključ neophodno je da konsultuje sledeću dokumentaciju: [https://developers.google.com/youtube/registering\\_an\\_application#Create\\_API\\_Keys](https://developers.google.com/youtube/registering_an_application#Create_API_Keys). Zbog jednostavnosti rada na ovom projektu, registrovan je serverski ključ ali bi verovatno trebalo koristiti i ključ za veb pregledač ukoliko je cilj publikovanje JavaScript koda na Internetu (*online*).

Za početak je neophodno podesiti dve konstante preko kojih se *YouTubeSearchService* mapira sa API ključem i API URL:

```
let YOUTUBE_API_KEY: string = "XXX_YOUR_KEY_HERE_XXX";  
let YOUTUBE_API_URL: string = "https://www.googleapis.com/youtube/v3/search";
```

## KREIRANJE SERVISA YOUTUBESEARCHSERVICE - UMETANJE ZAVISNOSTI

*Konstante API ključa moraju da imaju sposobnost primene u umetanju zavisnosti.*

U izvesnim situacijama javiće se potreba za testiranjem aktuelne aplikacije. Jedna od stvari koja bi mogla da se javi prilikom testiranja da nije nužno da se aplikacija testira samo u fazi produkcije, već se testiranje obavlja kroz sve faze životnog ciklusa razvoja aplikacije i za korišćeni razvojni API.

Da bi podešavanje ovakvog okruženja bilo uspešno, navedene konstante moraju da imaju sposobnost primene u umetanju zavisnosti (*injectable*).

Postavlja se pitanje: **zašto se koriste navedene konstante preko umetanja zavisnosti, a ne na uobičajen način?** Odgovor leži u činjenici da ako su one dostupne za umetanje u aplikaciji, tada:

1. moguće je imati kod koji injektuje pravu konstantu za dato okruženje i vreme izvršavanja;
2. moguće je jednostavno menjanje umetnute vrednosti tokom vremena testiranja.

Umetanjem navedenih vrednosti, dobija se mnogo veća fleksibilnost u radu u pogledu njihovih vrednosti. Da bi bilo omogućeno korišćenje navedenih vrednosti za umetanje u ostalim delovima aplikacije, koristi se sintaksa `{ provide: ... , useValue: ... }` na sledeći način (datoteka: `/src/app/you-tube-search/you-tube-search.injectables.ts`):

```
import {
  YouTubeSearchService,
  YOUTUBE_API_KEY,
  YOUTUBE_API_URL
} from './you-tube-search.service';

export const youTubeSearchInjectables: Array<any> = [
  {provide: YouTubeSearchService, useClass: YouTubeSearchService},
  {provide: YOUTUBE_API_KEY, useValue: YOUTUBE_API_KEY},
  {provide: YOUTUBE_API_URL, useValue: YOUTUBE_API_URL}
];
```

Primenom prikazanog koda specificirano je da se povezivanje `YOUTUBE_API_KEY` sa vrednošću nabavljenog API ključa (`YOUTUBE_API_KEY`) obavlja preko umetanja zavisnosti. Isto važi i za `YOUTUBE_API_URL`.

## KOREKCIJE KODA GLAVNE KOMPONENTE APLIKACIJE

### *Za injekrovanje servis mora da*

U ovom trenutku je važno prisetiti se da, ukoliko je nešto dostupno za umetanje u različitim modulima aplikacije, neophodno je da bude registrovano pod ključem `providers` u klasi obeleženoj dekoratorom `@NgModule`. Budući da se vrši eksportovanje `youTubeServiceInjectables`, u datoteci glavne komponente `app.module.ts` neophodno je dodati linije koda koje su istaknute na sledećoj slici:

```
// http/app.ts
import { HttpClientModule } from '@angular/common/http';
import { youTubeServiceInjectables } from "components/YouTubeSearchComponent";

// ...
// još koda dole
// ...

@NgModule({
  declarations: [
    HttpApp,
    // ostalo ....
  ],
  imports: [ BrowserModule, HttpClientModule ],
  bootstrap: [ HttpApp ],
  providers: [
    youTubeServiceInjectables // <--- upravo ovde
  ]
})
class AppModule {}
```

Slika 2.2 Registrovanje youtubeServiceInjectables u AppModule

Sada je u potpunosti omogućeno umetanje `YOUTUBE_API_KEY` (iz `youtubeServiceInjectables`) umesto direktne upotrebe promenljive.

## KREIRANJE KONSTRUKTORA ZA YOUTUBESEARCHSERVICE

*Kao što je već dobro poznato, iz prethodnog izlaganja, servis se kreira pravljenjem servisne klase.*

Kao što je već dobro poznato, iz prethodnog izlaganja, servis `YouTubeSearchService` se kreira pravljenjem servisne klase. Sledećim listingom je dat kod vezan za inicijalni razvoj navedene klase:

```
/**
 * YouTubeService se povezuje na YouTube API
 * Pogledajte: * https://developers.google.com/youtube/v3/docs/search/list
 */
@Injectable()
export class YouTubeSearchService {
  constructor(
    private http: HttpClient,
    @Inject(YOUTUBE_API_KEY) private apiKey: string,
    @Inject(YOUTUBE_API_URL) private apiUrl: string
  ) {}
}
```

Primena anotacije `@Injectable` dozvoljava da se konkretne vrednosti umetnu u konstruktor klase.

U konstruktor su, putem umetanja zavisnosti, dodate tri stvari:

1. `HttpClient`;
2. `YOUTUBE_API_KEY`;
3. `YOUTUBE_API_URL`.

Važno je primetiti da se iz sva tri argumenta kreiraju objektne promenljive, a to znači da im je moguće pristupiti preko: `this.http`, `this.apiKey`, i `this.apiUrl`, respektivno.

Takođe, umetanje zavisnosti je obavljeno na eksplicitan način preko sintakse `@Inject(YOUTUBE_API_KEY)`.

## DODAVANJE FUNKCIJE PRETRAGE U SERVISNU KLASU

*Da bi obavljanje pretrage bilo moguće neophodno je u servis dodati odgovarajuću funkciju.*

Aktuelni primer još ne poseduje traženu funkcionalnost pretrage video klipova preko YouTube platforme. Da bi navedeno bilo moguće, neophodno je u servisnu klasu, nakon konstruktora, dodati odgovarajuću funkciju pod nazivom `search()`. Funkcija će kao argument da uzme string koji odgovara upitu pretrage, a vratiće objekat `Observable` koji će emitovati tok `SearchResult[]`. To znači, u praksi, da svaka emitovana stavka predstavlja niz tipa `SearchResults`.

Sledi prvi deo listinga navedene funkcije:

```
search(query: string): Observable<SearchResult[]> {
  const params: string = [
    `q=${query}`,
    `key=${this.apiKey}`,
    `part=snippet`,
    `type=video`,
    `maxResults=10`
  ].join('&');
  const queryUrl = `${this.apiUrl}?${params}`;
```

Iz priloženog listinga se vidi da je kreiranje `queryUrl` urađeno na manuelan način. Jednostavno, procedura je započela dodavanjem parametra `query` funkcije `search()` u njenu konstantu `params`. Značenje svake od navedenih vrednosti možete potražiti u API dokumentaciji na linku <https://developers.google.com/youtube/v3/docs/search/list>.

U nastavku, nastavlja se izgradnja `queryUrl` spajanjem vrednosti za `apiUrl` i `params`.

Sada kada je `queryUrl` dostupan, moguće je kreirati zahtev. U konkretnom slučaju, zahtev će biti realizovan pozivom `http.get`, iako `HttpClient` može da obavi bilo koji tip `HTTP` zahteva (`POST`, `DELETE`, `GET` i tako dalje). Navedeno je implementirano u kompletiranom listingu metode `search()`:

```
search(query: string): Observable<SearchResult[]> {
  const params: string = [
    `q=${query}`,
    `key=${this.apiKey}`,
    `part=snippet`,
    `type=video`,
    `maxResults=10`
  ].join('&');
  const queryUrl = `${this.apiUrl}?${params}`;
  return this.http.get(queryUrl).map(response => {
    return <any>response['items'].map(item => {
      // console.log("raw item", item); // uklonite komentar za debug
      return new SearchResult({
        id: item.id.videoId,
        title: item.snippet.title,
        description: item.snippet.description,
        thumbnailUrl: item.snippet.thumbnails.high.url
      });
    });
  });
}
```

Gledano od linije koda 13, koriste se povratna vrednost [http.get](#) i funkcija ([map](#)) za dobijanje odgovora ([response](#)) iz prosleđenog zahteva. Iz odgovora (response) se izvlači telo kao objekat primenom [.json\(\)](#), obavlja se iteracija preko svake stavke ([item](#)) koja se konvertuje u [SearchResult](#).

## YOUTUBESEARCHSERVICE - KOMPLETAN LISTING

*Za širu sliku kreirani listinzi klase se sklapaju u celinu.*

Za širu sliku kreirani listinzi klase se sklapaju u celinu.

```
/**
 * YouTubeService se povezuje na YouTube API
 * Pogledajte: * https://developers.google.com/youtube/v3/docs/search/list
 */
@Injectable()
export class YouTubeSearchService {
  constructor(
    private http: HttpClient,
    @Inject(YOUTUBE_API_KEY) private apiKey: string,
    @Inject(YOUTUBE_API_URL) private apiUrl: string
  ) {}

  search(query: string): Observable<SearchResult[]> {
    const params: string = [
      `q=${query}`,
      `key=${this.apiKey}`,
      `part=snippet`,
      `type=video`,
      `maxResults=10`
    ].join('&');
    const queryUrl = `${this.apiUrl}?${params}`;
    return this.http.get(queryUrl).map(response => {
      return <any>response['items'].map(item => {
        // console.log("raw item", item); // uklonite komentar za debug
        return new SearchResult({
          id: item.id.videoId,
          title: item.snippet.title,
          description: item.snippet.description,
          thumbnailUrl: item.snippet.thumbnails.high.url
        });
      });
    });
  }
}
```

Da bi primer bio u potpunosti funkcionalan, u daljem radu se pristupa kreiranju preostalih delova komponente.



## KLASA SEARCHBOXCOMPONENT

*SearchBoxComponent je posrednik između korisničkog interfejsa i YouTubeSearchService servisa.*

Klasa `SearchBoxComponent` ima ključnu ulogu u aplikaciji. Ona je posrednik između korisničkog interfejsa i `YouTubeSearchService` servisa.

Klasa `SearchBoxComponent` će obavljati sledeće zadatke:

1. vodi računa o unosu sa tastature i slanju unetog stringa ka servisu `YouTubeSearchService`;
2. emituje događaj učitavanja;
3. emituje događaj rezultata kada postoje novi rezultati.

Neka je data definicija klase `SearchBoxComponent` na sledeći način (datoteka: <http://src/app/youtube-search/search-box.component.ts>):

```
@Component({
  selector: 'app-search-box',
  template: `
    <input type="text" class="form-control" placeholder="Search" autofocus>
  `
})
export class SearchBoxComponent implements OnInit {
  @Output() loading: EventEmitter<boolean> = new EventEmitter<boolean>();
  @Output() results: EventEmitter<SearchResult[]> = new
  EventEmitter<SearchResult[]>();

  constructor(private youtube: YouTubeSearchService,
    private el: ElementRef) {
  }
}
```

Selektor je analiziran veliki broj puta, ovde omogućava kreiranje taga `<app-search-box>`. Dve osobine obeležene dekoratorom `@Outputs` ukazuju da će događaji biti emitovani iz ove klase. To znači da je u pogledu moguće koristiti sintaksu `(output)="callback()"` za osluškivanje događaja ove komponente. Na primer, na sledeći način je moguće upotrebiti tag `<app-search-box>` u pogledu:

```
<app-search-box
  (loading)="loading = $event"
  (results)="updateResults($event)">
</app-search-box>
```

U konkretnom primeru, kada `SearchBoxComponent` klasa emituje događaj učitavanja, podešava se promenljiva `loading` u kontekstu roditeljske komponente. Na isti način, kada `SearchBoxComponent` emituje događaje prikupljanja rezultata, poziva se funkcija `updateResults()`, sa vrednostima, u kontekstu roditeljske komponente (`YouTubeSearchComponent`).

U klasi, obeleženoj dekoratorom `@Component` (`YouTubeSearchComponent`) specificiraju se osobine događaja pod nazivima: `loading` i `results`. U konkretnom primeru, svaki događaj će imati odgovarajući `EventEmitter` kao objektne promenljive kontrolerske klase. Navedeno će biti implementirano u najkraćem roku.

Za sada, dovoljno je rezonovati o `@Component` kao javnom API za komponentu pa se ovde samo specificiraju nazivi događaja.

## SEARCHBOXCOMPONENT - DEFINICIJA ŠABLONA

*Jednostavan šablon, sa jednim input tagom.*

Šablon `SearchBoxComponent` je poprilično jednostavan i sadrži jedan `input` tag integrisan pod ključem `template` dekoratora `@Component` kase `SearchBoxComponent` (datoteka: `src/app/you-tube-search/search-box.component.ts`).

```
@Component({
  selector: 'app-search-box',
  template: `
    <input type="text" class="form-control" placeholder="Search" autofocus>
  `
})
```

## SEARCHBOXCOMPONENT - DEFINICIJA KONTROLERA

*Klasa implementira metodu `OnInit` iz razloga što je neophodno koristiti povratni poziv (callback) `ngOnInit` životnog ciklusa*

Klasa kontroler `SearchBoxComponent` je nova klasa koja se u projektu čuva u datoteci `/src/app/you-tube-search/search-box.component.ts`. Početna definicija klase je data sledećim listingom:

```
export class SearchBoxComponent implements OnInit {
  @Output() loading: EventEmitter<boolean> = new EventEmitter<boolean>();
  @Output() results: EventEmitter<SearchResult[]> = new
  EventEmitter<SearchResult[]>();
}
```

Veoma je bitno da ova klasa implementira metodu `OnInit` iz razloga što je neophodno koristiti povratni poziv (callback) `ngOnInit` životnog ciklusa. Ukoliko klasa implementira metodu `OnInit`, metoda `ngOnInit` će biti pozvana čim se prva promena detektuje. Ova metoda je odlično mesto za zadatke inicijalizacije (za razliku od konstruktora) iz razloga što skup ulaza u komponentu nije dostupan u konstrukturu.

Za obe osobine klase: `loading` i `results`, kreira se `EventEmitters` koji će vratiti podatak tipa `boolean` kada se pretraga učitava i niz tipa `SearchResults` kada je pretraga završena.

## SEARCHBOXCOMPONENT - KONSTRUKTOR KONTROLERA

*Definicija klase `SearchBoxComponent` se proširuje kreiranjem konstruktora.*

Definicija klase `SearchBoxComponent` se proširuje kreiranjem konstruktora.

```
constructor(private youtube: YouTubeSearchService,  
             private el: ElementRef) {  
}
```

U konstruktor je umetnuto sledeće:

1. servis `YouTubeSearchService`;
2. element `el` na koji ova komponenta povezana. Ovaj element je objekat tipa `ElementRef` koji predstavlja Angular omotač oko nativnih elemenata.

Oba parametra konstruktora su preko umetanja zavisnosti podešeni kao objektne promenljive.

## SEARCHBOXCOMPONENT - FUNKCIJA ONINIT KONTROLERA

*Odlično rešenje predstavlja uključivanje događaja unosa sa tastature u osmatrani (`Observable`) tok.*

Ako se obrati pažnja na šablon komponente `SearchBoxComponent` moguće je primetiti da on sadrži polje za unos teksta preko kojeg se unose ključne reči za pretragu `YouTube` video klipova.

U ovom svetlu je moguće uraditi tri stvari sa ciljem unapređenja korisničkog iskustva:

1. filtriraju se (i odbacuju) svi prazni i prekratki upiti;
2. automatski se pokreće pretraga u zavisnosti od vremena proteklog od unosa poslednjeg karaktera (na primer najkraće od 250 ms);
3. odbacuju se sve stare pretrage, ukoliko je korisnik inicirao novu pretragu.

U kodu je moguće manuelno povezati (bind) unos sa tastature i poziv funkcije za svaki događaj unosa i, potom, primeniti filtriranje i vremensko pokretanje pretrage, po automatizmu. Međutim bolje rešenje predstavlja uključivanje događaja unosa sa tastature u osmatrani (`Observable`) tok.

Reaktivna biblioteka `RxJS` (*Reactive Extensions Library for JavaScript* - <https://rxjs-dev.firebaseapp.com/>) obezbeđuje mehanizme osluškivanje događaja konkretnog elementa na osnovu poziva funkcije `Rx.Observable.fromEvent()`. Navedeno je moguće obaviti na sledeći način:

```
ngOnInit(): void {  
    // konvertuje unos sa tastature u observable tok  
    Observable.fromEvent(this.el.nativeElement, 'keyup')
```

Iz priloženog koda metode `fromEvent()` moguće je primetiti sledeće:

- prvi element je `this.el.nativeElement` (nativni DOM element na koji je komponenta povezana);
- drugi argument predstavlja string `'keyup'`, koji predstavlja naziv događaja kojeg je neophodno konvertovati u tok.

## RXJS FUNKCIONALNOSTI U ONINIT METODI

### *Demonstracija benefita koji se ostvaruju primenom RxJS funkcionalnosti.*

Definicija metode `onInit()` klase komponente još uvek nije gotova. U ovom delu izlaganja je cilj **demonstracija benefita koji se ostvaruju primenom RxJS funkcionalnosti**. Izlaganje će ići korak po korak sa ciljem prezentovanja i implementacije navedenih funkcionalnosti.

Navedeni tok događaja tastature (`keyup`) moguće je povezati u lanac sa većim brojem metoda. Upravo, u sledećem izlaganju, nekoliko funkcija će biti povezano u lanac zajedno sa tokom pri čemu će biti obavljena transformacija toka u niz `SearchResult[]`. Na samo kraju svi listinzi koji su opisivali detalje primera biće povezani u jednu celinu zbog jasnijeg sagledavanja rešenja.

U prvom koraku je neophodno izolovati vrednost dobijenu preko `<input>` taga (uneto sa tastature):

```
.map((e: any) => e.target.value) // izvlači vrednost inputa
```

Priloženi listing kaže da je neophodno obaviti mapiranje svakog događaja tastature (`keyup`), a zatim pronaći ciljni element događaja (`e.target`), a to je u ovom slučaju input element. Kada je to gotovo neophodno je izvući vrednost tog elementa. To praktično znači da je posmatrani tok sada konvertovan u tok stringova.

U sledećem koraku u lanac se dodaje poziv sledeće funkcije:

```
.filter((text: string) => text.length > 1) // odbacuje ulaz ako je prazan
```

Ovaj filter obezbeđuje da za string dužine manje od 1 neće biti vršena pretraga. Programer, ukoliko misli da je zgodno, ovaj broj može povećati za drugačiju definiciju ignorisanja pretrage po kratkim stringovima.

U nastavku, u lanac metoda je moguće dodati i sledeći poziv:

```
.debounceTime(250) // na unos stringa pokreće pretragu za vreme nakon 250ms
```

Metoda `debounceTime()` obezbeđuje da zahtevi koji stignu u intervalu manjem od 250ms neće biti uzimani u obzir. To znači, neće biti obavljana pretraga za svaki pritisak na taster tastature, već kada je napravljena mala pauza nakon poslednjeg pritiska ta taster.

Sledeći poziv u lancu metoda može biti:

```
.do(() => this.loading.emit(true)) // omogućava učitavanje
```

Primenom metode `do` na tok je način izvođenja funkcije među-toka za svaki događaj, ali još uvek ne menja ništa u toku. Ideja je sledeća: postoji pretraga, postoji dovoljan broj karaktera ključne reči pretrage, `our search`, it has enough characters, prošao je dovoljan vremenski interval i pretraga može biti započeta, odnosno uključuje se učitavanje. Vrednost `this.loading` je tipa `EventEmitter`. Učitavanje se uključuje emitovanjem vrednosti `true` kao sledećeg događaja. Za `EventEmitter` vrši se emitovanje događaja pozivom `next`: Sintaksa `this.loading.emit(true)` govori da se emituje događaj `"true"` na učitavanju objekta `EventEmitter` - vrednost `$event` je `true`.

## DODATNE FUNCKIJE U LANCU TOKA DOGAĐAJA

*Prethodni lanac je moguće nastaviti pozivima novih korisnih metoda.*

Prethodni lanac je moguće nastaviti pozivima novih korisnih metoda. Na primer, moguće je dodati sledeće:

```
// pretraga, odbacuju se stari događaji ukoliko se javi nov input
.map((query: string) => this.youtube.search(query))
.switch()
```

Poziv `.map()` se koristi za pozivanje izvođenja pretrage po svakom postavljenom upitu koji je emitovan. Pozivom `switch()` ignorišu se svi rezultati pretrage osim poslednjeg. To praktično znači, ukoliko je dostupan rezultat najnovije pretrage on se prikazuje, a stariji se odbacuju.

Konačno, za svaki emitovani upit, vrši se pretraga preko kreiranog servisa `YouTubeSearchService`. Objedinjavanjem lanca poziva metoda u celinu, dobija se sledeći listing:

```
ngOnInit(): void {
  // konvertuje unos sa tastature u observable tok
  Observable.fromEvent(this.el.nativeElement, 'keyup')
    .map((e: any) => e.target.value) // izvlači vrednost inputa
    .filter((text: string) => text.length > 1) // odbacuje ulaz ako je prazan
    .debounceTime(250) // na unos stringa pokreće
    pretragu za vreme do 250ms
    .do(() => this.loading.emit(true)) // omogućava učitavanje
    // pretraga, odbacuju se stari događaji ukoliko se javi nov input
    .map((query: string) => this.youtube.search(query))
    .switch()
    // postupa po vraćanju rezultata
    .subscribe(
      (results: SearchResult[]) => { // uspešno (onSuccess)
```

```
        this.loading.emit(false);
        this.results.emit(results);
    },
    (err: any) => { // greška (onError)
        console.log(err);
        this.loading.emit(false);
    },
    () => { // kompletirano (onComplete)
        this.loading.emit(false);
    }
  );
}
```

U konačnu definiciju lanca metoda dodat je i poziv `subscribe()`. Budući da se vrši obraćanje servisu `YouTubeSearchService`, posmatrani tok je sada tok izgrađen od objekata tipa `SearchResult[]`. Pozivom metode `subscribe()` vrši se prijavljivanje na ovaj tok i omogućeno je izvođenje odgovarajućih operacija. Metoda uzima tri argumenta:

- `onSuccess`,
- `onError`,
- `onCompletion`.

Prvi argument ukazuje šta je neophodno uraditi ukoliko tok emituje regularan događaj. Događaj se emituje za oba `EventEmitter` - a:

1. pozivom `this.loading.emit(false)` se ukazuje da je učitavanje zaustavljeno;
2. pozivom `this.results.emit(results)` se ukazuje na emitovanje događaja koji sadrži listu sa rezultatima.

Drugi argument ukazuje šta se dešava ukoliko tok poseduje događaj sa greškom. Zaustavlja se učitavanje i u logu pregledača se dobija informacija o grešci.

Treći argument ukazuje šta se dešava kada se stigne do kraja toka. Takođe, vrši se emitovanje da je učitavanje završeno.

## SEARCHBOXCOMPONENT - KOMPLETAN LISTING

*Za sagledavanje šire slike, prethodni segmenti koda se sklapaju u funkcionalnu celinu.*

Za sagledavanje šire slike, prethodni segmenti koda se sklapaju u funkcionalnu celinu.

Klasa `SearchBoxComponent` se čuva u datoteci sa sledeće lokacije: `/src/app/you-tube-search/search-box.component.ts` i priložena je sledećim listingom:

```
import {
  Component,
  OnInit,
  Output,
  EventEmitter,
```

```

    ElementRef
  } from '@angular/core';

import { Observable } from 'rxjs/Rx';
import 'rxjs/add/observable/fromEvent';
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/filter';
import 'rxjs/add/operator/debounceTime';
import 'rxjs/add/operator/do';
import 'rxjs/add/operator/switch';

import { YouTubeSearchService } from '../you-tube-search.service';
import { SearchResult } from '../search-result.model';

@Component({
  selector: 'app-search-box',
  template: `
    <input type="text" class="form-control" placeholder="Search" autofocus>
  `
})
export class SearchBoxComponent implements OnInit {
  @Output() loading: EventEmitter<boolean> = new EventEmitter<boolean>();
  @Output() results: EventEmitter<SearchResult[]> = new
  EventEmitter<SearchResult[]>();

  constructor(private youtube: YouTubeSearchService,
    private el: ElementRef) {
  }

  ngOnInit(): void {
    // konvertuje unos sa tastature u observable tok
    Observable.fromEvent(this.el.nativeElement, 'keyup')
      .map((e: any) => e.target.value) // izvlači vrednost inputa
      .filter((text: string) => text.length > 1) // odbacuje ulaz ako je prazan
      .debounceTime(250) // na unos stringa pokreće
      pretragu za vreme do 250ms
      .do(() => this.loading.emit(true)) // omogućava učitavanje
      // pretraga, odbacuju se stari događaji ukoliko se javi nov input
      .map((query: string) => this.youtube.search(query))
      .switch()
      // postupa po vraćanju rezultata
      .subscribe(
        (results: SearchResult[]) => { // uspešno (onSuccess)
          this.loading.emit(false);
          this.results.emit(results);
        },
        (err: any) => { // greška (onError)
          console.log(err);
          this.loading.emit(false);
        },
        () => { // kompletirano (onComplete)
          this.loading.emit(false);
        }
      );
  }
}

```

```
    }  
    );  
  }  
}
```

## KREIRANJE KOMPONENTE SEARCHRESULTCOMPONENT

*Zadatak ove komponente jeste kreiranje i prikazivanje pojedinačnih SearchResult objekata.*

U prethodnom izlaganju je bilo diskusije i rada na veoma složenoj komponenti, za ova nivo znanja, pod nazivom *SearchBoxComponent*. Sada je moguće posvetiti pažnju jednostavnijoj komponenti *SearchResultComponent*. Zadatak ove komponente jeste kreiranje i prikazivanje pojedinačnih rezultata pretrage, tj. jednog *SearchResult* objekta.

Kod klase ove komponente predstavlja "već viđeno" i priložen je sledećim listingom (datoteka: *src/app/you-tube-search/search-result.component.ts*):

```
import {  
  Component,  
  OnInit,  
  Input  
} from '@angular/core';  
import { SearchResult } from './search-result.model';  
  
@Component({  
  selector: 'app-search-result',  
  templateUrl: './search-result.component.html'  
})  
export class SearchResultComponent implements OnInit {  
  @Input() result: SearchResult;  
  
  constructor() { }  
  
  ngOnInit() {  
  }  
}
```

Komponenta poseduje jednu (ulaznu) objektnu promenljivu *result*, koja predstavlja instancu tipa *SearchResult*.

Šablon komponente će sadržati: naziv, opis i sliku vezanu za pronađeni klip, a zatim i link ka videu kojem se pristupa putem dugmeta (slika 3). Sledi listing šablona (datoteka: *src/app/you-tube-search/search-result.component.html*):



```
<div class="col-sm-6 col-md-3">
  <div class="thumbnail">
    
    <div class="caption">
      <h3>{{result.title}}</h3>
      <p>{{result.description}}</p>
      <p><a href="{{result.videoUrl}}"
        class="btn btn-default" role="button">
          Gledaj</a></p>
    </div>
  </div>
</div>
```



Slika 2.3 Jedan SearchResult objekat

## KREIRANJE GLAVNE KOMPONENTE - YOUTUBESEARCHCOMPONENT

*Kreiranje komponente koja objedinjuje sve ostale komponente u celinu.*

Ovaj deo projekta se okončava kreiranjem glavne komponente - *YouTubeSearchComponent* čiji je zadatak da poveže sve prikazane komponente i njihove funkcionalnosti u jednu celinu.

Definicija komponente započinje kreiranjem dekoratora *@Component* i osobina klase na sledeći način (datoteka: */src/app/you-tube-search/you-tube-search.component.ts*):

```
@Component({
  selector: 'app-you-tube-search',
  templateUrl: './you-tube-search.component.html'
})
export class YouTubeSearchComponent implements OnInit {
  results: SearchResult[];
  loading: boolean;
```

Klasa definiše identifikovanje komponente putem taga *<app-you-tube-search>* i primenjuje dve objektne promenljive: *results* i *loading*.

Da bi kreirana klasa mogla da se smatra kontrolerom, ona mora da implementira neke pogodne metode. Na prethodni listing dodaje se kod koji odgovara pomenutim metodama.

```
import { Component, OnInit } from '@angular/core';
import { SearchResult } from './search-result.model';

@Component({
  selector: 'app-you-tube-search',
  templateUrl: './you-tube-search.component.html'
})
export class YouTubeSearchComponent implements OnInit {
  results: SearchResult[];
  loading: boolean;

  constructor() { }
  ngOnInit() { }

  updateResults(results: SearchResult[]): void {
    this.results = results;
    // console.log("results:", this.results); // uklonite komentar da vidite
    rezultat
  }
}
```

Dodate metode su konstruktor i `updateResults()`. Funkcija `updateResults()` se ponaša veoma jednostavno - uzima bilo koji novi rezultat pretrage (`SearchResult[]`) i podešava `this.results` na novu vrednost.

## YOUTUBESEARCHCOMPONENT - KREIRANJE ŠABLONA

*Kreiranjem šablona glavne komponente završena je definicija primera.*

Kreiranjem šablona glavne komponente završena je definicija primera. Ovaj pogled bi trebalo da obavi sledeće tri stvari:

1. prikazuje indikator učitavanja, ako je zahtev u toku;
2. osluškuje događaje za kontrolu unosa ključnih reči pretrage;
3. prikazuje rezultate pretrage.

Za početak, data je osnovna struktura šablona sa GIF animacijom koja ukazuje da je učitavanje zahteva u toku (slika 4).

```
<div class='container'>
  <div class="page-header">
    <h1>YouTube pretraga
      <img
        style="float: right;"
        *ngIf="loading"
        src='assets/images/loading.gif' />
    </h1>
  </div>
</div>
```

YouTube pretraga



dit

Slika 2.4 Indikator učitavanja zahteva

Navedena slika se prikazuje samo ako je vrednost promenljive `loading true`. Za ovo je upotrebljena `ngIf` direktiva.

Sada je neophodno uključiti u pogled i kontrolu za pretragu:

```
<div class="row">
  <div class="input-group input-group-lg col-md-12">
    <app-search-box
      (loading)="loading = $event"
      (results)="updateResults($event)"
    ></app-search-box>
  </div>
</div>
```

Interesantan detalj predstavlja povezivanje: `loading` i `results` izlaza. Primećuje se korišćenje sintakse, opšteg oblika, `(output)="action()"`.

Za učitavanje izlaza, pokreće se izraz `loading = $event`. `$event` biće zamenjena vrednošću koja je emitovana od strane `EventEmitter`. To znači, kada se u komponenti `SearchBoxComponent` pozove `this.loading.emit(true)`, `$event` se podešava na vrednost `true`.

Slično, za izlaz `results` poziva se funkcija `updateResults()` uvek prilikom emitovanja novog skupa rezultata pretrage. Ovo za efekat ima ažuriranje objektna promenljive `results` komponente.

Konačno, neophodno je u pogledu prikazati rezultate pretrage.

```
<div class="row">
  <app-search-result
    *ngFor="let result of results"
    [result]="result">
  </app-search-result>
</div>
</div>
```

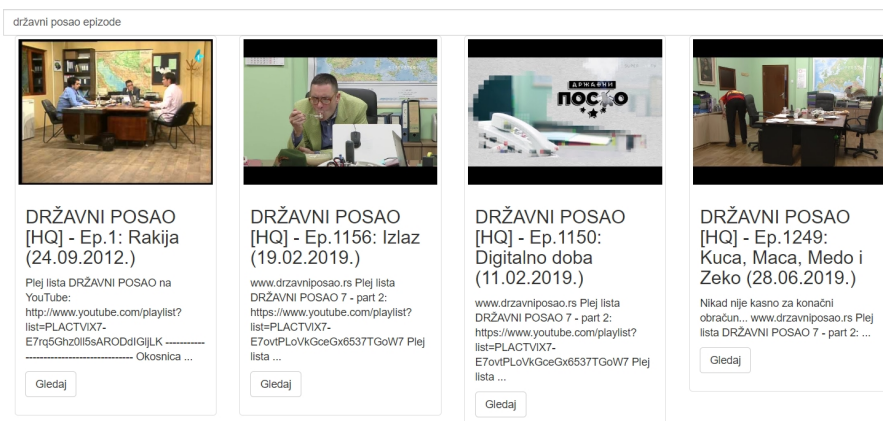
## YOUTUBESEARCHCOMPONENT - DEMO

*Sledi demonstracija urađenog primera.*

Sledi demonstracija urađenog primera. Prvo se unose ključne reči pretrage i ako zadovoljavaju uslov, pokreće se pretraga (ovo je prikazano slikom 4).

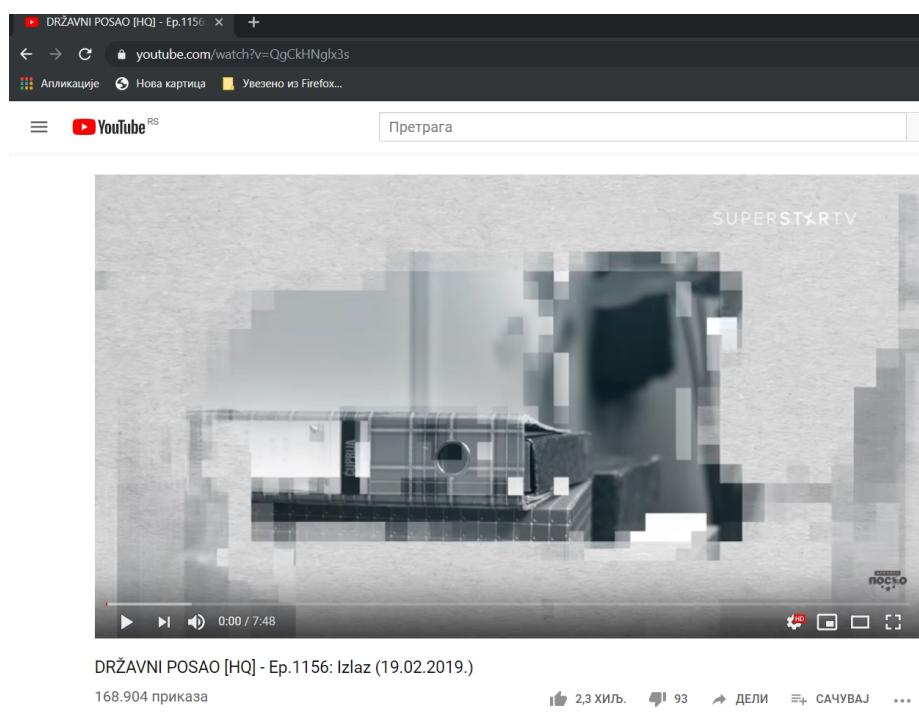
Uskoro prikazuju se rezultati pretrage - ograničeno na 10 rezultata (slika 5).

## YouTube pretraga



Slika 2.5 Rezultati pretrage

Iz rezultata pretrage bira se jedan i klikom na dugme "Gledaj" pokreće se u YouTube aplikaciji izabrani video klip (slika 6).



Slika 2.6 Izbor iz rezultata pretrage

## ▼ Poglavlje 3

# angular/common/http API - ostali HTTP zahtevi

## IZVOĐENJE POST ZAHTEVA

*API [jsonplaceholder](http://jsonplaceholder.typicode.com/) obezbeđuje pogodan URL za testiranje POST zahteva.*

U prethodnom izlaganju akcenat je bio na primeni osnovnog HTTP zahteva GET. Naravno, važno je da se shvati da je na veoma jednostavan način u [Angular](#) aplikacijama moguće napraviti i ostale [HTTP](#) zahteve.

Prvi zahtev koji dolazi na red za diskusiju je [POST](#) zahtev. Kreiranje [POST](#) zahteva primenom paketa [@angular/common/http](#) veoma podseća na kreiranje [GET](#) zahteva uključujući jedan dodatni parametar: *body*.

[API \[jsonplaceholder\]\(http://jsonplaceholder.typicode.com/\)](#) (<http://jsonplaceholder.typicode.com/>), takođe, obezbeđuje pogodan [URL](#) za testiranje [POST](#) zahteva. U narednom izlaganju će upravo navedeno biti i pokazano.

Za početak je kreirana nova komponenta [more-http-requests.component](#). Sledećom listingom je prikazana metoda [makePost\(\)](#) kontrolerske klase komponente čiji se kod čuva u datoteci [src/app/more-http-requests/more-http-requests.component.ts](#).

```
makePost(): void {
  this.loading = true;
  this.http
    .post(
      'https://jsonplaceholder.typicode.com/posts',
      JSON.stringify({
        body: 'bar',
        title: 'foo',
        userId: 1
      })
    )
    .subscribe(data => {
      this.data = data;
      this.loading = false;
    });
}
```

U drugom argumentu, kao što je moguće primetiti iz listinga, uzima se [Object](#) koji se prevodi u [JSON](#) string pozivom [JSON.stringify\(\)](#).

## PUT / PATCH / DELETE / HEAD

*Postoje i ostali HTTP zahtevi koji se pozivaju na veoma sličan način kao prikazani GET i POST.*

Postoje i ostali, prilično opšti HTTP zahtevi koji se pozivaju na veoma sličan način kao prikazani [GET](#) i [POST](#) zahtev:

- [http.put](#) i [http.patch](#) mapiraju se u [PUT](#) i [PATCH](#), respektivno, i oba koriste [URL](#) i [body](#);
- [http.delete](#) i [http.head](#) mapiraju se u [DELETE](#) i [HEAD](#), respektivno i koriste samo [URL](#).

Sledi primer izvođenja [DELETE](#) zahteva (datoteka: [src/app/more-http-requests/more-http-requests.component.ts](#)):

```
makeDelete(): void {
  this.loading = true;
  this.http
    .delete('https://jsonplaceholder.typicode.com/posts/1')
    .subscribe(data => {
      this.data = data;
      this.loading = false;
    });
}
```

## PRILAGOĐENA HTTP ZAGLAVLJA

*Kreiranje GET zahteva koji koristi specijalno X-API-TOKEN zaglavlje.*

Konačno, recimo da je cilj kreiranje GET zahteva koji koristi specijalno [X-API-TOKEN](#) zaglavlje. Moguće je kreirati zahtev sa navedenim zaglavljem na sledeći način:

```
makeHeaders(): void {
  const headers: HttpHeaders = new HttpHeaders({
    'X-API-TOKEN': 'IT-255'
  });

  const req = new HttpRequest(
    'GET',
    'https://jsonplaceholder.typicode.com/posts/1',
    {
      headers: headers
    }
  );

  this.http.request(req).subscribe(data => {
    this.data = data['body'];
  });
}
```

## OBJEDINJENI LISTINZI KONTROLERA I ŠABLONA KOMPONENTE

*Za širu sliku kreirani listinzi kontrolera i šablona komponentese sklapaju u celinu.*

Sledi celokupan listing klase komponente:

```
import { Component, OnInit } from '@angular/core';
import {
  HttpClient,
  HttpRequest,
  HttpHeaders
} from '@angular/common/http';

@Component({
  selector: 'app-more-http-requests',
  templateUrl: './more-http-requests.component.html'
})
export class MoreHttpRequestsComponent implements OnInit {
  data: Object;
  loading: boolean;

  constructor(private http: HttpClient) {}

  ngOnInit() {}

  makePost(): void {
    this.loading = true;
    this.http
      .post(
        'https://jsonplaceholder.typicode.com/posts',
        JSON.stringify({
          body: 'bar',
          title: 'foo',
          userId: 1
        })
      )
      .subscribe(data => {
        this.data = data;
        this.loading = false;
      });
  }

  makeDelete(): void {
    this.loading = true;
    this.http
      .delete('https://jsonplaceholder.typicode.com/posts/1')
      .subscribe(data => {
        this.data = data;
        this.loading = false;
      });
  }
}
```

```
});  
}  
  
makeHeaders(): void {  
  const headers: HttpHeaders = new HttpHeaders({  
    'X-API-TOKEN': 'IT-255'  
  });  
  
  const req = new HttpRequest(  
    'GET',  
    'https://jsonplaceholder.typicode.com/posts/1',  
    {  
      headers: headers  
    }  
  );  
  
  this.http.request(req).subscribe(data => {  
    this.data = data['body'];  
  });  
}  
}
```

Sledi celokupan listing šablona komponente:

```
<h2>Još zahteva</h2>  
<button type="button" (click)="makePost()">Kreiraj Post</button>  
<button type="button" (click)="makeDelete()">Kreiraj Delete</button>  
<button type="button" (click)="makeHeaders()">Kreiraj Headers</button>  
<div *ngIf="loading">loading...</div>  
<pre>{{data | json}}</pre>
```

## PRIMENA KOMPONENTE MORE-HTTP-REQUESTS.COMPONENT - DEMO

*Izlaganje završava demonstracijom izvršavanja poslednjeg primera.*

Izlaganje završava demonstracijom izvršavanja poslednjeg primera. Sledećom slikom prikazan je rezultat dobijen klikom na dugme "*Kreiraj POST*".

### Još zahteva

|              |                |                 |
|--------------|----------------|-----------------|
| Kreiraj Post | Kreiraj Delete | Kreiraj Headers |
|--------------|----------------|-----------------|

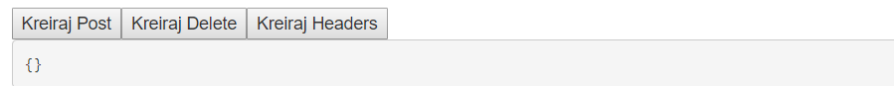
```
{  
  "id": 101  
}
```

Slika 3.1 Kreiranje i izvršavanje POST zahteva

Sledećom slikom prikazan je rezultat dobijen klikom na dugme "*Kreiraj DELETE*".



## Još zahteva



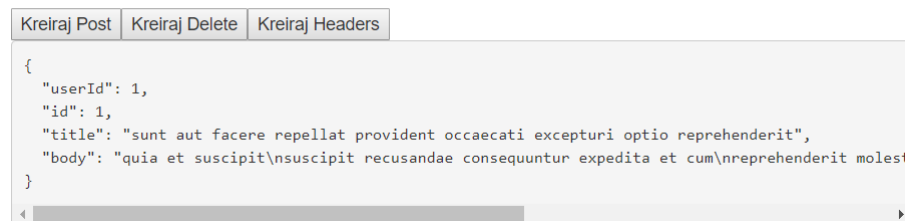
Kreiraj Post Kreiraj Delete Kreiraj Headers

```
{}
```

Slika 3.2 Slika-1 Kreiranje i izvršavanje DELETE zahteva

Sledećom slikom prikazan je rezultat dobijen klikom na dugme " *Kreiraj HEADERS*".

## Još zahteva



Kreiraj Post Kreiraj Delete Kreiraj Headers

```
{  "userId": 1,  "id": 1,  "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",  "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molest
```

Slika 3.3 Kreiranje i izvršavanje HEADER zahteva

**Odmah, nakon ovog objekta učenja, možete preuzeti, analizirati i testirati kompletno urađen pokazni primer koji je koristio kao podrška u izlaganju ove lekcije.**

## ANGULAR I HTTP - DODATNI POGLED

*Kada se pravi Angular aplikacija tako da koristi REST API, najbitniji servis je svakako [HTTPClient](#)*

Kada se pravi Angular aplikacija tako da koristi REST API, najbitniji servis je svakako [HTTPClient](#), jer se pomoću njega mogu dalje implementirati specifični servisi koji komuniciraju sa spoljnom aplikacijom i koji prikupljaju ili šalju podatke. U tom slučaju, ovakvi servisi služe kao veza između klijentskog i serverskog dela aplikacije. Servis [HttpClient](#) ima obezbeđene metode za svaki tip zahteva ( [GET](#), [POST](#), [PUT](#), [DELETE](#) ...) i svaka od ovih metoda vraća vrednost na koju se korisnik servisa kasnije može pretplatiti.

## ANGULAR I HTTP - VIDEO MATRIJALI

*Izlaganje lekcije biće zaokruženo odgovarajućim video primerima.*

Angular HTTP Client Quick Start Tutorial - Trajanje 9:55

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

Angular 8 Tutorial - 20 - HTTP and Observables - Trajanje 7:40

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

Angular 6 Tutorial 12: HTTP Requests - Trajanje 13:33.

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

## ▼ Poglavlje 4

### Vežba 9

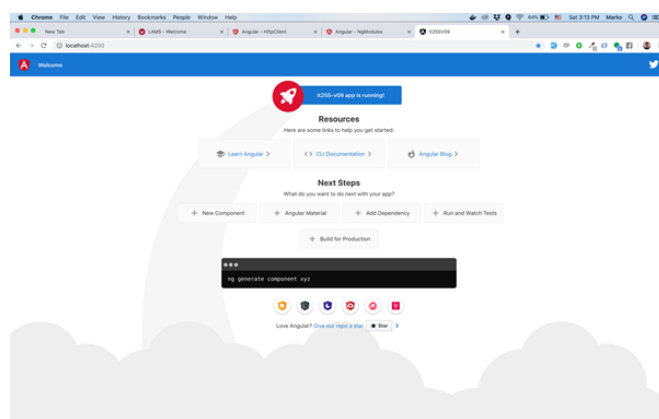
## HTTP I ANGULAR - POKAZNA VEŽBA (POČETNA PODEŠAVANJA)

*Prednost Angular - a, u odnosu na neke druge frameworke je što dolazi sa već gotovim bibliotekama za kreiranje HTTP zahteva*

Prednost Angular - a, u odnosu na neke druge radne okvire (framework) je što dolazi sa već gotovim bibliotekama za kreiranje i obradu HTTP zahteva. Konkretno, u našem slučaju koristimo HttpClient. Prethodno navedena biblioteka dolazi kao sastavni deo modula HttpClientModule, koji je neophodno da uključimo unutar imports sekcije kroz datoteku app.module.ts.

Za potrebe ove vežbe, biće kreiran novi projekat kao što je prikazano u prethodnim vežbama koristeći komandu: ng new it255-v09.

Po kreiranju projekta pokrećemo isti i imamo rezultat kao na slici:



Slika 4.1 Inicijalni izgled kreiranog projekta

Kako bismo malo stilizovali našu aplikaciju, učit ćemo bootstrap - ov CSS fajl unutar naše aplikacije, tako što ćemo dodati sledeće parče koda u head sekciju index.html fajla:

```
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" crossorigin="anonymous">
```

Nakon toga, kako bismo mogli da obavljamo HTTP pozive, neophodno je da uključimo HttpClientModule, unutar imports sekcije na nivou čitave aplikacije kroz app.module.ts. Potrebno da je prvo uvezemo biblioteku iz Angular okvira na sledeći način:

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

A potom da istu biblioteku uključimo unutar *Imports* dela na sledeći način:

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { HttpClientModule } from '@angular/common/http';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Više informacija o tome se može naći na sledećem linku: <https://angular.io/guide/http>

## PRIMENA GOTOVIH SERVISA I KREIRANJE MODELA

*JSONPlaceholder omogućava da obavimo testiranje HTTP poziva kroz Angular okvir.*

Za potrebe ove vežbe, koristićemo već dostupni REST servis pod nazivom *JSONPlaceholder*, koji će nam omogućiti da obavimo testiranje *HTTP* poziva kroz *Angular* okvir. Primenjujući znanje stečeno na časovima predavanja, vežbi i radeći servise, odmah pristupamo kreiranju servisa pod nazivom *PostService* unutar *services* foldera. Za to ćemo iskoristiti komandu: *ng g service services/post*.

Unutar tog servisa, kroz konstruktor neophodno je da inicijalizujemo *HttpClient* - a na sledeći način:

```
constructor(private _httpClient: HttpClient) { }
```

A pre toga je potrebno da uvezemo biblioteku na sledeći način:

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

Imajući u vidu sve navedeno, moguće je prikazati kod servisa *PostService* u celini, sledećim listingom:

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

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

  constructor(private _httpClient: HttpClient) { }
}
```

Unutar servisa ćemo definisati metode pod imenima: *getPost*, *deletePost*, *createPost*. Postojeće i par pomoćnih metoda, ali o njima će biti reči malo kasnije.

Kako bismo ispoštovali standarde prethodno definisane u vežbama, kreiraćemo model pod naziv *Post* sledećom komandom:

- *ng g class models/post*

Sam model *Post* sadržaće atribute koji su prikazani na slici ispod:

```
- {
  userId: 1,
  id: 1,
  title: "sunt aut facere",
  body: "quia et suscipit",
},
,
```

Slika 4.2 Atributi modela

## KOD KLASSE MODELA I DOPUNA DEFINICIJE SERVISA

*Na osnovu istaknutih osobina modela, kodira se i odgovarajuća klasa.*

Na osnovu istaknutih osobina modela, kodira se i odgovarajuća klasa. Klasa se naziva *Post*, čuva se u TypeScript datoteci *post.ts* i priložena je sledećim listingom:

```
import { Optional } from '@angular/core';

export class Post {
  id: number;
  userId: number;
  title: string;
  body: string;
}
```

```

        constructor(userId: number, title: string, body: string, @Optional() id:
number) {
            this.id = id;
            this.userId = userId;
            this.title = title;
            this.body = body;
        }
    }
}

```

Kao što je istaknuto u izlaganju sa prethodne sekcije, unutar servisa *PostService* ćemo definisati metode pod imenima: *getPost*, *deletePost*, *createPost*, kao i par pomoćnih metoda. Ove metode su date primerom koda ispod:

```

import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Post } from '../models/post';
import { map } from 'rxjs/operators';

@Injectable({
    providedIn: 'root'
})
export class PostService {
    private baseUrl = 'https://jsonplaceholder.typicode.com/posts';

    constructor(private _httpClient: HttpClient) { }

    public getPosts() : Observable<Post[]> {
        return this._httpClient.get(this.baseUrl).pipe(
            map((data: any[]) => data.map((item: any) =>
this._createPostFromObject(item))),
        );
    }

    public getPost(id: number) : Observable<Post> {
        return this._httpClient.get(this.baseUrl + '/' + id).pipe(
            map((data: any) => this._createPostFromObject(data)),
        );
    }

    public deletePost(id: number) : Observable<Post> {
        return this._httpClient.delete(this.baseUrl + '/' + id).pipe(
            map((data: any) => this._createPostFromObject(data)),
        );
    }

    public createPost(post: Post) : Observable<Post> {
        return this._httpClient.post(this.baseUrl, post).pipe(
            map((data: any) => this._createPostFromObject(data)),
        );
    }
}

```

```
}

private _createPostFromObject(item:any) {
  return new Post(item.userId, item.title, item.body, item.id);
}

}
```

Dodatne metode koje možemo videti unutar koda su:

- `pipe()` koja služi za vezivanje više metoda u cilju manipulacije izlaznog toka podataka.
- `map()` - Izvršava više funkcija nad izlaznim tokovima podataka.
- `_createPostFromObject()` - Pomoćna metoda koju koristimo za kreiranje modela podatka na osnovu JSON reprezentacija istog.

## GLAVNA KOMPONENTA APLIKACIJE

*Neophodno je da kreiramo i formu, koja će sadržati polja za kreiranje novog Post - a.*

Unutar `app.component.ts` datoteke, u kojoj se čuva glavna klasa glavne komponente aplikacije `AppComponent`, koju ćemo koristiti za reprezentativni primer manipulacije podataka, imaćemo više podataka koje ćemo pozivati u raznim životnim ciklusima aplikacije, kao i prilikom određenih akcija, poput `getPost` i `deletePost`. Takođe, neophodno je da kreiramo i *formu*, koja će sadržati polja za kreiranje novog `Post` - a. Kod navedene forme je priložen sledećim listinzima:

### Inicijalizacija forme:

```
public initForm() {
  this.postForm = new FormGroup({
    title: new FormControl('', [
      Validators.required
    ]),
    userId: new FormControl(1, [
      Validators.required
    ]),
    body: new FormControl('', [
      Validators.required
    ])
  });
}
```

### HTML kod forme (implementirano u šablonu komponente):

```
<form [formGroup]="postForm" class="container mt-5 pt-5" (ngSubmit)="submitForm()">
  <div class="form-group">
    <label for="title">Title</label>
    <input class="form-control" formControlName="title" type="text" name="title">
```

```

</div>

<div class="form-group">
  <label for="body">Body</label>
  <input class="form-control" formControlName="body" type="text" name="body">
</div>

<div class="form-group">
  <label for="userId">userId</label>
  <input class="form-control" formControlName="userId" type="number"
name="userId">
</div>
<div class="col button-holder d-flex justify-content-center align-items-center
mt-4">
  <button type="submit" name="submit" class="btn btn-success">Dodaj</button>
</div>
</form>

```

Prilikom inicijalizacije same komponente, vrši se učitavanje svih postova koristeći metodu koja je navedena ispod:

```

constructor(private _postService: PostService) {
  this._postService.getPosts().subscribe((data) => {
    this.posts = data;
  })
  this.initForm();
}

```

## GLAVNA KOMPONENTA APLIKACIJE - CELOVITI PRIKAZ

*Neophodno je sve prikazane segmente komponente povezati u funkcionalnu celinu.*

Finalni zadatak razvoja glavne komponente aktuelne Angular aplikacije ukazuje da je neophodno je sve prikazane segmente komponente povezati u funkcionalnu celinu.

Sledi listing klase komponente [AppComponent](#):

```

import { Component } from '@angular/core';
import { PostService } from '../services/post.service';
import { Post } from '../models/post';
import { FormGroup, FormControl, Validators } from '@angular/forms';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})

```



```
export class AppComponent {
  public postForm: FormGroup;
  public posts: Post[] = [];

  constructor(private _postService: PostService) {
    this._postService.getPosts().subscribe((data) => {
      this.posts = data;
    })
    this.initForm();
  }

  public initForm() {
    this.postForm = new FormGroup({
      title: new FormControl('', [
        Validators.required
      ]),
      userId: new FormControl(1, [
        Validators.required
      ]),
      body: new FormControl('', [
        Validators.required
      ])
    });
  }

  public submitForm() {
    let title = this.postForm.get('title').value;
    let userId = this.postForm.get('userId').value;
    let body = this.postForm.get('body').value;
    let post = new Post(userId, title, body, 0);
    this.createPost(post)
  }

  public getPost(id: number) {
    this._postService.getPost(id).subscribe((data) => {
      alert(JSON.stringify(data));
    })
  }

  public createPost(post: Post) {
    this._postService.createPost(post).subscribe((data) => {
      this.posts.unshift(data);
    })
  }

  public deletePost(id: number) {
    this._postService.deletePost(id).subscribe((data) => {
      this._removePostFromList(id);
      alert("Post je obrisan sa servera");
    })
  }

  private _removePostFromList(id: number) {
```

```

    let ind = this.posts.findIndex(post => post.id == id);
    this.posts.splice(ind, 1);
  }
}

```

Sledi listing šablona komponente iz datoteke [app.component.html](#):

```

<form [formGroup]="postForm" class="container mt-5 pt-5" (ngSubmit)="submitForm()">
  <div class="form-group">
    <label for="title">Title</label>
    <input class="form-control" formControlName="title" type="text" name="title">
  </div>

  <div class="form-group">
    <label for="body">Body</label>
    <input class="form-control" formControlName="body" type="text" name="body">
  </div>

  <div class="form-group">
    <label for="userId">userId</label>
    <input class="form-control" formControlName="userId" type="number"
name="userId">
  </div>
  <div class="col button-holder d-flex justify-content-center align-items-center
mt-4">
    <button type="submit" name="submit" class="btn btn-success">Dodaj</button>
  </div>
</form>

<div class="container mt-5 pt-5">
  <table class="table">
    <thead>
      <tr>
        <th scope="col">#</th>
        <th scope="col">User id</th>
        <th scope="col">Title</th>
        <th scope="col">Description</th>
        <th scope="col">Delete</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let post of posts; index as i;">
        <th scope="row">{{post.id}}</th>
        <th>{{post.userId}}</th>
        <td>{{post.title}}</td>
        <td>{{post.body}}</td>
        <td>
          <button (click)="getPost(post.id)" type="button" class="btn
btn-primary">Get post</button>
        </td>
        <td>
          <button (click)="deletePost(post.id)" type="button" class="btn

```

```
btn-danger">Delete post</button>
    </td>
  </tr>
</tbody>
</table>
</div>
```

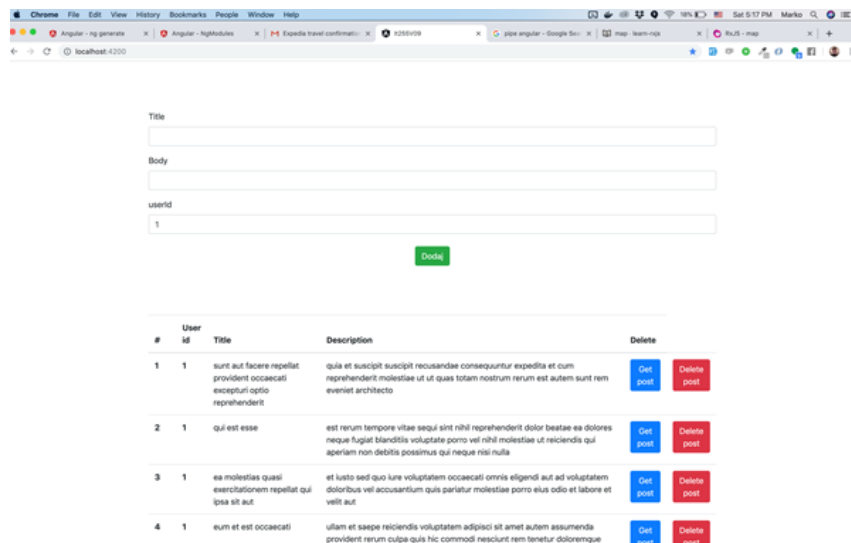
## DEMO APLIKACIJE

*Sledi demonstracija urađenog pokaznog primera.*

Sledi demonstracija urađenog pokaznog primera. Pre same demonstracije sledi bitna napomena.

**Kako je u pitanju mockup REST server, postovi neće biti kreirani niti obrisani na serveru, već samo lokalno, dodali smo da se prikazuju samo unutar aplikacije rezultati tih akcija.**

Aplikacija se snima, prevodi i angažuje pozivom: [ng serve](#). Nakon toga se pokreće u veb pregledaču na [portu 4200](#). Izgled finalne aplikacije dat je na slici ispod:



Slika 4.3 Demo aplikacije

Kod vežbe je dostupan na sledećem linku:

<https://github.com/markorajevic/it255-v09>

## INDIVIDUALNA VEŽBA

*Samostalno vežbanje implementacije naučenon na predavanjima i pokaznoj vežbi.*

Nastavlja se rad na primeru koji je delimično urađen na pokaznim vežbama. Ukoliko imate poteškoća u vezi sa funkcionisanjem vašeg primera, kompletno funkcionalan primer možete da preuzmete sa sledećeg linka: <https://github.com/markorajevic/it255-v09>.

Uradite modifikacije nad primerom sa pokaznih vežbi tako što ćete u aplikaciju uvesti sledeće dopune:

- **dodati Pipe koji će imati funkcionalnost filtiranja postova**
- **filtriranje postova se obavlja ili po titlu ili po sadržaju posta.**

Nakon urađene vežbe pozovite predmetnog asistenta i demonstrirajte funkcionisanje uvedenih dopunskih funkcionalnosti u aplikaciju.

## ▼ Poglavlje 5

### Domaći zadatak 9

#### DOMAĆI ZADATAK

*Samostalna izrada domaćeg zadatka.*

Uradite domaći zadatak prema sledećim zahtevima:

1. Osnova domaćeg zadatka je zadatak koji ste obradili u prethodnim vežbama sa [YouTube](#) snimcima,
2. Prilikom kreiranja videa-a, pozvati [Youtube REST API](#) za dovlačenje [Title](#) i [Description](#) podataka
3. Na navedeni način ispuniti polja za [description](#) i [title](#).

Domaći zadatak dodati na [Github](#) pod " [commit](#) - om" [IT255-DZ09](#) i poslati obaveštenje predmetnom asistentu o postavljenom domaćem zadatku.

## ▼ Poglavlje 6

# Zaključak

## ZAKLJUČAK

*Lekcija se bavila izučavanjem mehanizama sinhronog rukovanja HTTP zahtevima.*

Na samom početku lekcije je istaknuto kako radni okvir *Angular* distribuira vlastitu *HTTP biblioteku* koja može biti veoma pogodna za obavljanje poziva ka eksternim API - jima.

Posebna tema bavljenja lekcije je bila problematika poziva ka eksternim serverima. U tom slučaju, neophodno je obezbediti da korisnik poseduje kontinuiranu interakciju sa stranicom veb aplikacije. To praktično znači, da je neophodno sprečiti da se stranica "zamrzne" dok ne stigne HTTP odgovor sa servera. Da bi navedeno bilo moguće realizovati, **HTTP zahtevi moraju biti asinhroni** i to je bio ključni zaključak uvodnog razmatranja lekcije.

U tom svetlu, lekcija ističe da rad sa asinhronim kodom, a to ima i istorijsko uporište u programiranju, može biti složeniji nego rukovanje sinhronim kodom.

U JavaScript jeziku, kao osnovi Angular TypeScript jezika, postoje tri pristupa za pomoć u rukovanju asinhronim kodom:

1. povratni pozivi (*callback*);
2. obećanja (*promises*);
3. osmatranja (*observable*).

U Angular okviru preferirani metod manipulisanja asinhronim kodom predstavlja treći slučaj - osmatranje, na čemu će biti i bazirano izlaganje u ovoj lekciji.

Imajući u vidu sve navedeno, opredeljeno je da lekcija pokrije tri ključne stvake:

1. analiza i diskusija osnovnog *HttpClient* primera;
2. kreiranje *YouTube* komponente za pretragu;
3. diskusija o API detaljima vezanim za *HttpClient* biblioteku.

Savladavanjem ove lekcije student je razumeo i osposobljen je da rukuje asinhronim *HTTP* zahtevima u *Angular* aplikacijama.

## LITERATURA

*Za pripremu lekcije korišćena je aktuelna pisana i elektronska literatura.*

**Pisana literatura:**

1. Nate Murray, Felipe Coury, Ari Lerner, Carlos Taborda, ng-Book – The Complete Book on Angular 6, Fullstack.io, 2018
2. Cody Lindley, Frontend Handbook – 2017, Frontend masters, 2017
3. Sandeep Panda, AngularJS – From Novice to Ninja, SitePoint Pty. Ltd, 2014

**Elektronska literatura:**

4. <https://angular.io/>
5. <https://angular.io/tutorial>
6. <https://www.w3schools.com/angular/>
7. <https://www.tutorialspoint.com/angular4/>
8. <https://nodejs.org/en/>
9. <https://code.visualstudio.com/>
10. [https://www.w3schools.com/whatis/whatis\\_html5dom.asp](https://www.w3schools.com/whatis/whatis_html5dom.asp)