Servisi

Servisi

- Komponenta je fokusirana na interaktivnu logiku sa korisnikom
- Uzimanje podataka sa servera ne treba da se radi u komponenti
- Servis je klasa koja komunicira sa serverom i obezbeđuje podatke za aplikaciju
- Servis je klasa sa uskom i jasno definisanom funkcionalnošću
- Komponenta koristi servis putem mehanizma koji se zove dependency injection (DI)
- Pomoću anotacije @Injectable() specificira se da je servis raspoloživ za dependency injection

Dependency Injection u Angularu

- Angular ima svoj DI framework
- Zavisnosti su servisi ili objekti koji su potrebni nekoj klasi da bi izvršavala svoju funkciju
- DI je kodni šablon gde klasa traži ubacivanje zavisnosti od ekstenog izvora umesto da sama instancira objekte klase od kojih zavisi
- Traženje zavisnosti od DI frameworka vrši se posredstvom konstruktora zavisne klase

Kreiranje klase

ng g class osoba

```
export class Osoba {
     constructor(public osobaId: number, public ime: string, public prezime: string, public starost: number ) {
     }
}
```

Fajl podaci.ts unutar app foldera

```
import { Osoba } from './osoba';

export const OSOBE: Osoba[] = [
    new Osoba(1, 'Marko', 'Markovic', 25),
    new Osoba(2, 'Petar', 'Petrovic', 34),
    new Osoba(3, 'Jovan', 'Jovanovic', 27),
    new Osoba(4, 'Milan', 'Mirkovic', 22)
  ];
```

Kreiranje servisa

ng g service osoba

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

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

  constructor() { }
}
```

```
import { Injectable } from '@angular/core';
import { OSOBE } from './podaci';
import { Osoba } from './osoba';

@Injectable({
  providedIn: 'root'
})
export class OsobaService {
  vratiOsobe():Osoba[]
  {
    return OSOBE;
  }

  constructor() { }
}
```

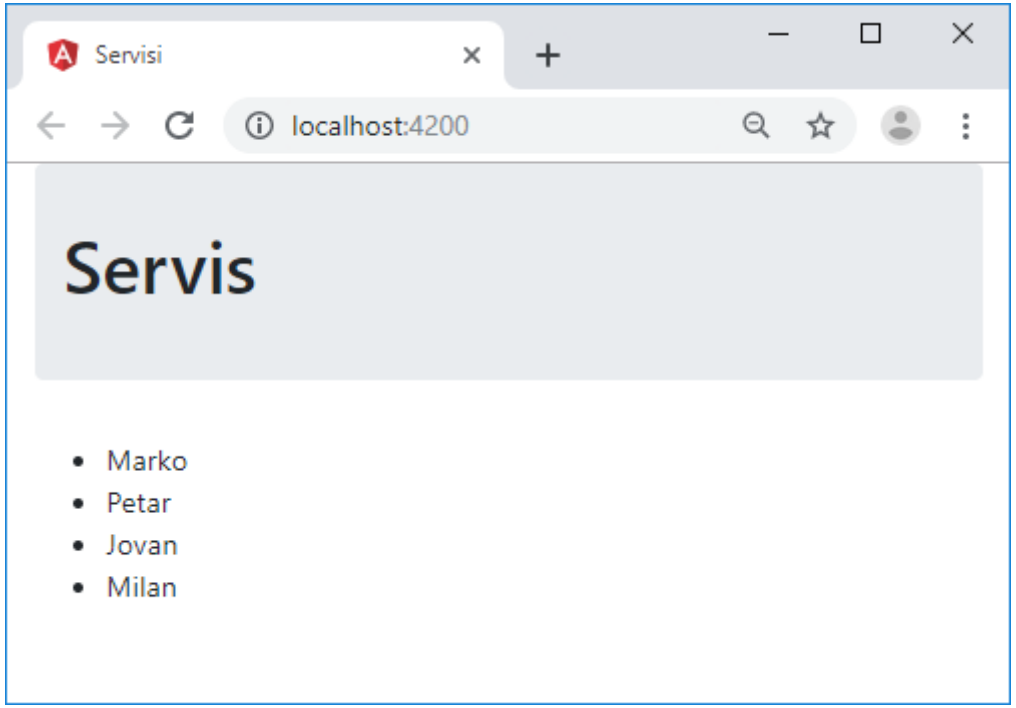
Singleton servis je servis za koga postoji jedna instance u aplikaciji.
providedIn: ‘root‘ – obezbeđuje singleton servis
Angular kreira jednu instancu servisa koju ubaciju u svaku komponentu koja zahteva podatke posredstvom servisa

Ubacivanje objekta servisa u konstruktor komponente

```
export class AppComponent implements OnInit {
  title = 'Servisi';
  osobe: Osoba[];
  constructor(private oServis: OsobaService) {

  }
  ngOnInit(): void {
    this.osobe = this.oServis.vratiOsobe();
  }

}
```

Šablon komponente

```
<div class="container">
  <div class="jumbotron">
    <h1>Servis</h1>
  </div>
  <div class="row">
    <div class="col-md-6">

      <ul>
        <li *ngFor="let osoba of osobe">
          {{osoba.ime}}
        </li>
      </ul>
    </div>
  </div>
</div>
```



Observables

- Omogućavaju razmenu poruka između izdavača (publisher) i pretplatnika (subscribers)
- Publisher definiše funkciju koja generiše vrednosti ali se ona neće izvršiti dok se korisnik ne pretplati na nju
- Pretplaćeni korisnik dobija notifikacije sve dok se ne završi izvršavanje publisherske funkcije ili se pretplatnik ne odjavi
- Publisher kreira instancu Observable koja ima **subscribe**() funkciju
- Da bi izvršili observable trebamo pozvati subscribe() funkciju
- Subscribe() funkciji se prosleđuje javascript objekat koji se naziva **observer**
- Observer sadrži tri svojstva **next**, **error** i **complete** od kojih je samo prvo svojstvo obavezno
- Svojstvu **next** se dodeljuje hendlerska funkcija za svaku primljenu vrednost
- Svojstvu **error** se dodeljuje hendlerska funkcija koja dobija obaveštenje o grešci
- Svojstvu **complete** se dodeljuje hendlerska funkcija koja dobija obaveštenje o završetku emitovanja vrednosti

Observables

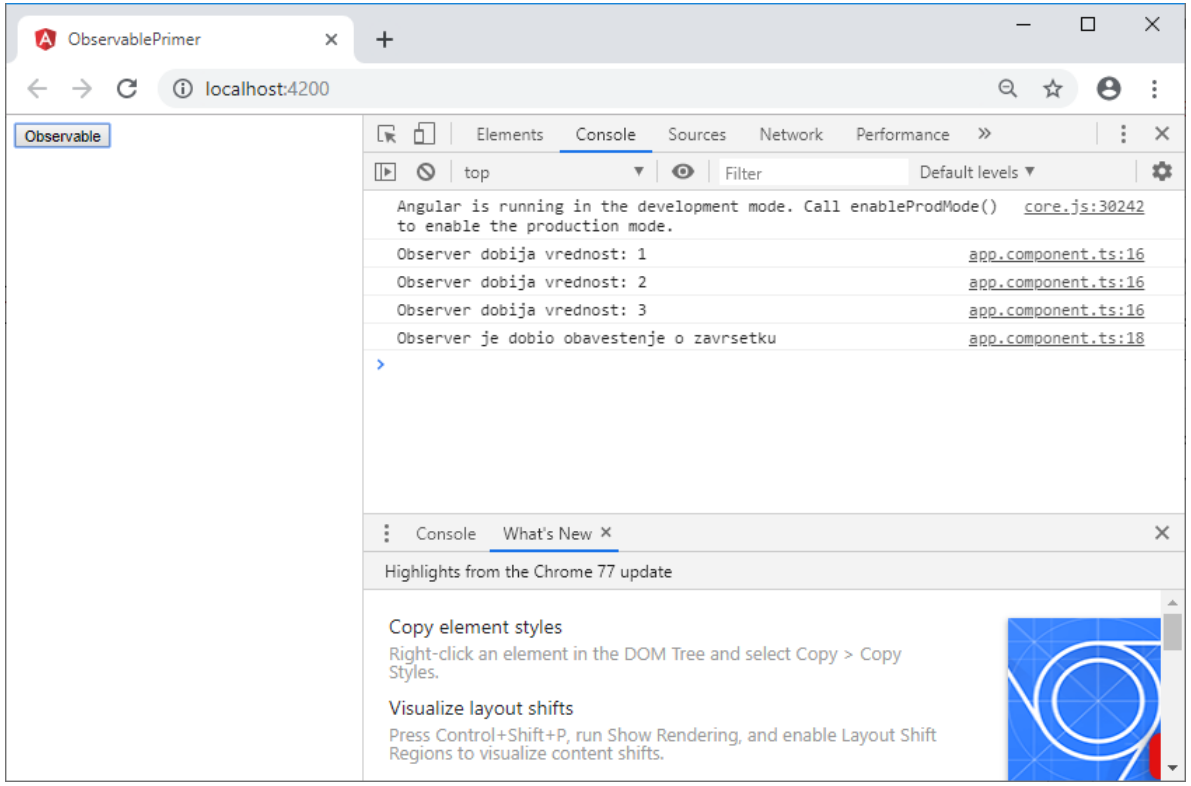
- Web server obično šalje podatke asinhrono i ti podaci se mogu modelovati kao Observable stream
- Observable je sekvenca stavki koje stižu asinhrono tokom vremena
- Observable može imati više pretplatnika i svi pretplatnici bivaju obavešteni kada se stanje observable menja
- Angular koristi biblioteku RxJs (Reactive Javascript) da bi implementirao observable
- HTTP modul koristi observables pri obradi AJAX zahteva i odgovora

Primer Observable i pretplata

```
export class AppComponent {
  title = 'ObservablePrimer';

  // Jednostavni observable koji emituje tri vrednosti
  mojObservable = of(1, 2, 3);

  observer = {
    next: (x: number) => console.log('Observer dobija vrednost: ' + x),
    error:  (err: string) => console.error('Observer je dobio  gresku: ' + err),
    complete: () => console.log('Observer je dobio obavestenje o zavrsetku')
   };

   pretplata() {
    this.mojObservable.subscribe(this.observer);
   }
}
```

Šablon komponente

```
<button (click)="pretplata()">Observable</button>
```



Modifikacija servisa

```
import { Injectable } from '@angular/core';
import { Osoba } from './osoba';
import { OSOBE } from './podaci';
import { of, Observable } from 'rxjs';

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

  constructor() { }

  vratiOsobe(): Observable<Osoba[]> {
    return of(OSOBE);
  }
}
```

Pretplata na observable – prosleđivanje metode

```
export class AppComponent implements OnInit {
  title = 'Servisi';
  osobe: Osoba[];

  constructor(private oServis: OsobaService) {
  }

  ngOnInit(): void {
    this.oServis.vratiOsobe()
      .subscribe(os => this.osobe = os);
  }

}
```

Komunikacija sa udaljenim serverom

Angular HttpClient API

- Klasa HttpClient omogućava asinhronu komunikaciju sa udaljenim serverom
- Nalazi se u paketu @angular/common/http
- **HttpClientModule** se importuje u koreni modul aplikacije
- Ovaj modul se ubacuje u imports niz glavnog modula aplikacije

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

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

Klasa HttpClient

- Klasa HttpClient je tzv. injectable klasa i ubacuje se u klasu servisa posredstvom njegovog konstruktora
- Klasa HttpClient ima metode kojima se generišu http zahtevi ka serveru
- Podatke za aplikaciju najčešće obezbeđuje web api
- Metoda get() generiše GET zahtev (čitanje podataka)
- Metoda post() generiše POST zahtev (kreiranje novih podataka)
- Metoda put() generiše PUT zahtev (promena postojećih podataka)
- Metoda delete() generiše DELETE zahtev (brisanje postojećih podataka)

Angular In-Memory Web API

- Ovaj modul simulira REST (REpresentational State Transfer) API back-end

```
npm i angular-in-memory-web-api
```

Klasa Osoba

ng g class osoba

```
export class Osoba {
    constructor(public id: number, public ime: string, public prezime: string,
     public starost: number ) {
    }
}
```

Kreiranje baze podataka u memoriji

ng g class baza

```
import { InMemoryDbService } from 'angular-in-memory-web-api';
import { Osoba } from './osoba';

export class Baza implements InMemoryDbService {
    createDb() {
      const osobe: Osoba[] = [
                new Osoba(1, 'Marko', 'Markovic', 25),
                new Osoba(2, 'Petar', 'Petrovic', 34),
                new Osoba(3, 'Jovan', 'Jovanovic', 27),
                new Osoba(4, 'Milan', 'Mirkovic', 22)
              ];

      return {osobe};
    }
}
```

'api/osobe' adresa web api -ja

Konfigurisanje glavnog modula aplikacije

```
import { Baza } from './baza';
import { InMemoryWebApiModule } from 'angular-in-memory-web-api';

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

Kreiranje servisa za komunikaciju sa api -jem

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpErrorResponse} from '@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { Osoba } from './osoba';
import { catchError } from 'rxjs/operators';

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

  constructor(private http: HttpClient) { }

  private apiUrl = '/api/osobe';

  private errorHandler(error: HttpErrorResponse) {
    return throwError(error.message || 'Server eror');
  }
```

GET metode servisa za komunikaciju sa web api aplikacijom

```
vratiOsobe(): Observable<Osoba[]>{
      return this.http.get<Osoba[]>(this.apiUrl)
      .pipe(catchError(this.errorHandler));
 }

vratiOsobu(id: number): Observable<Osoba> {
      const url = `${this.apiUrl}/${id}`;
      return this.http.get<Osoba>(url).pipe(
        catchError(this.errorHandler)
      );
}
```

pipe() funkcija kao argumente uzima funkcije koje treba da kombinuje
catchError () funkacija hvata greške

Obrada grešaka pri http komunikaciji

```
import { HttpClient, HttpErrorResponse } from '@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { catchError } from 'rxjs/operators';
```

```
private errorHandler(error: HttpErrorResponse) {
    return throwError(() => error);
  }
```

Glavna komponenta – poziv get metode

```
export class AppComponent {
  title = 'komunikacija';

  osobe: Osoba[] = Array<Osoba>();
  odabranaOsoba: Osoba = new Osoba(0,'','',0);
  idOsobe = 4;

  constructor(private oServis: OsobaService) {
  }
   prikaziOsobe(): void {
      this.oServis.vratiOsobe()
        .subscribe({
         next: os => this.osobe = os,
         error: grr => console.log('Greska: ' + grr)
        });
    }

  ngOnInit(): void {
      this.prikaziOsobe();
    }
}
```

Glavna komponenta – metoda prikaziOsobu

```
prikaziOsobu(): void {
        this.oServis.vratiOsobu(this.idOsobe)
        .subscribe({
           next: os => this.odabranaOsoba = os,
           error: grr => {
            console.log('Greska: ' + grr);
            this.odabranaOsoba= new Osoba(0,'','',0);
            }
          });
 }
```

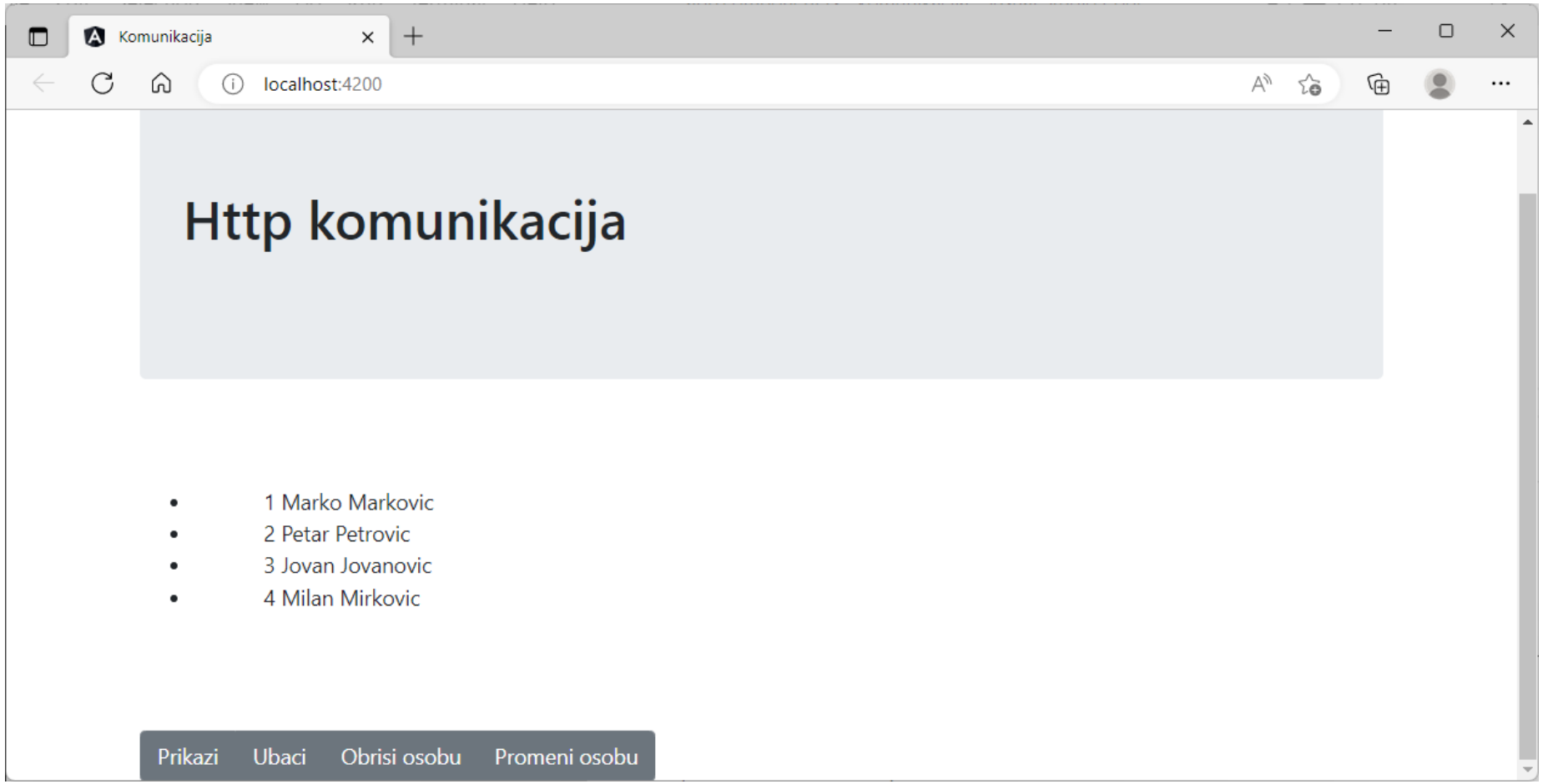
Šablon glavne komponente –1

```
<div class="container">
      <div class="jumbotron">
        <h1>Http komunikacija</h1>
      </div>

  <div class="row">
            <div class="col-6">
                <ul>
                  <li *ngFor="let osoba of osobe">
                    {{osoba.id}} {{osoba.ime}} {{osoba.prezime}}
                  </li>
                </ul>
             </div>

        <div class="col-6" *ngIf="odabranaOsoba.id != 0">
         <p>
              {{odabranaOsoba.ime}} {{odabranaOsoba.prezime}}
        </p>
       </div>
</div>
```

Šablon glavne komponente –2

```
<div class="row">
    <div class="col-6">
      <div class="btn-group" role="group" >
        <button class="btn btn-secondary" (click)="prikaziOsobu();">Prikazi</button> <br>
        <button class="btn btn-secondary" (click)="ubaciOsobu()">Ubaci</button> <br>
        <button class="btn btn-secondary" (click)="obrisiOsobu()">Obrisi osobu</button>
        <button class="btn btn-secondary" (click)="promeniOsobu()">Promeni osobu</button>
      </div>
    </div>
  </div>

</div>
```

Korisnički interfejs



POST metoda servisa za komunikaciju sa web api aplikacijom

```
ubaciOsobu(os: Osoba): Observable<Osoba> {
        return this.http.post<Osoba>(this.apiUrl, os).pipe(
          catchError(this.errorHandler)
        );
 }
```

Glavna komponenta – unos podataka

```
ubaciOsobu(): void {
      var ime:string = this.kreirajIme(5);
      var prezime:string = this.kreirajIme(5);
      var starost: number = Math.floor(Math.random() * 35);
      this.idOsobe++;
      const os1 = new Osoba(this.idOsobe, ime, prezime, starost);
      this.oServis.ubaciOsobu(os1).subscribe({
         next:  podaci => {
            console.log(podaci);
            this.prikaziOsobe();
          },
        error:  gr => console.log(gr)
         });
  }
```

kreirajIme je javascript funkcija koja generiše slučajni string od 5 karaktera

DELETE metoda servisa za komunikaciju sa web api aplikacijom

```
obrisiOsobu(id: number): Observable<{}> {
        const url = `${this.apiUrl}/${id}`;
        return this.http.delete<Osoba>(url).pipe(
            map(() => id),
            catchError(this.errorHandler)
          );
 }
```

Glavna komponenta – brisanje podataka

```
obrisiOsobu()  {
        this.oServis.obrisiOsobu(this.idOsobe).subscribe({
         next: os => {
            console.log(os);
            this.prikaziOsobe();
            this.idOsobe--;
          },
          error: grr => console.log('Greska: ' + grr)
        })
   }
```

Uvek se briše osoba koja ima najveći id

PUT metoda servisa za komunikaciju sa web api aplikacijom

```
promeniOsobu(os: Osoba): Observable<Osoba> {
      const url = `${this.apiUrl}`;
      return this.http.put<Osoba>(url, os)
      .pipe(
        map(() => os),
        catchError(this.errorHandler)
      );
    }
```

Glavna komponenta – promena podataka

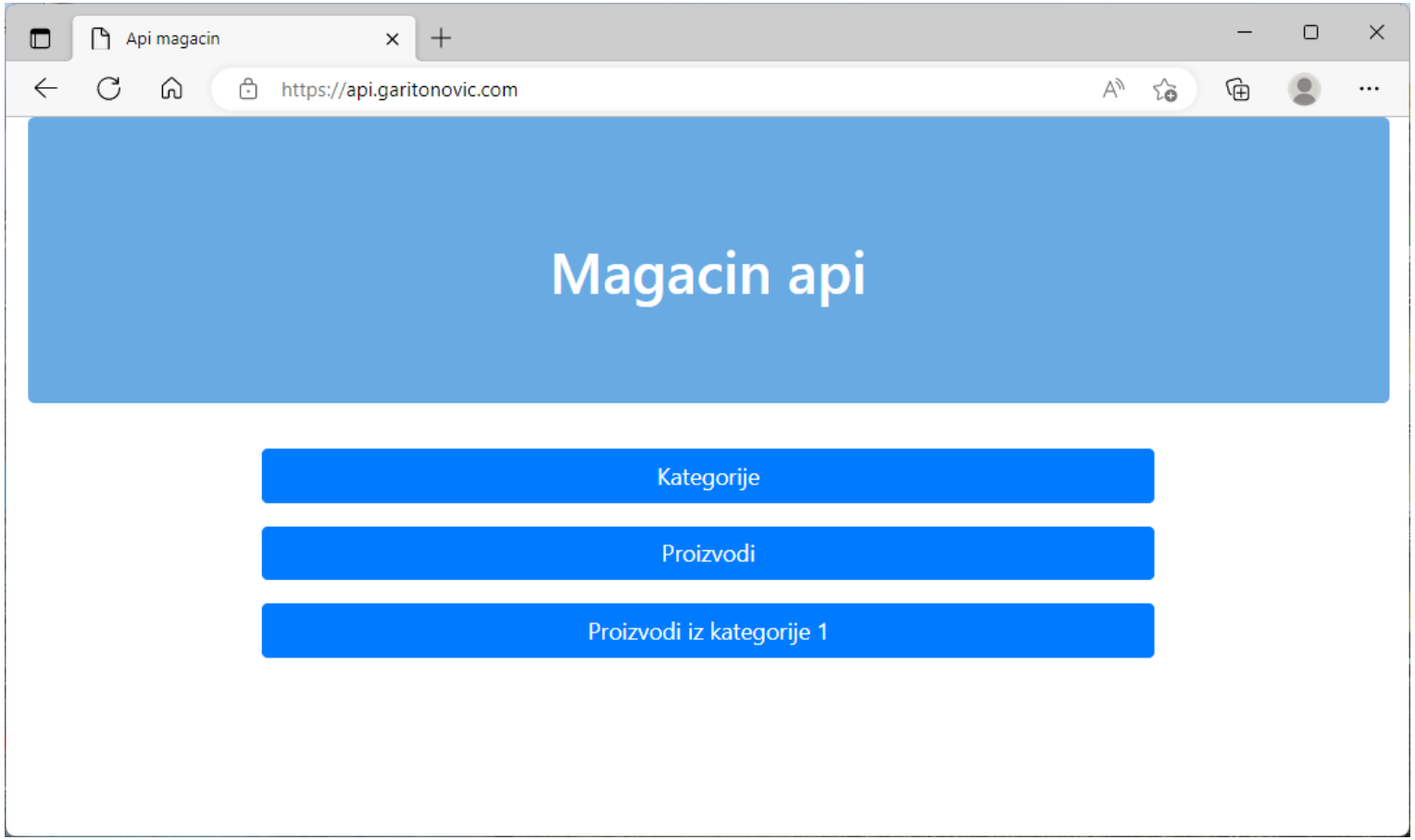
```
promeni() {
    const os1 = this.osobe.find(x => x.id == 1);
    os1.ime = 'test';
    this.oServis.promeniOsobu(os1).subscribe(
      os => console.log(os),
      grr => console.log('Greska: ' + grr)
    );
  }
```

Promena podataka

```
promeniOsobu() {
      const os1 = this.osobe.find(x => x.id == this.idOsobe);
      if (os1 != undefined) {
        os1.ime = this.kreirajIme(5);
        os1.prezime = this.kreirajIme(5);
        os1.starost = Math.floor(Math.random() * 30);
        this.oServis.promeniOsobu(os1).subscribe({
        next:  os => console.log(os),
        error:  grr => console.log('Greska: ' + grr)
      });
      }

    }
```

Primer web api -aplikacije

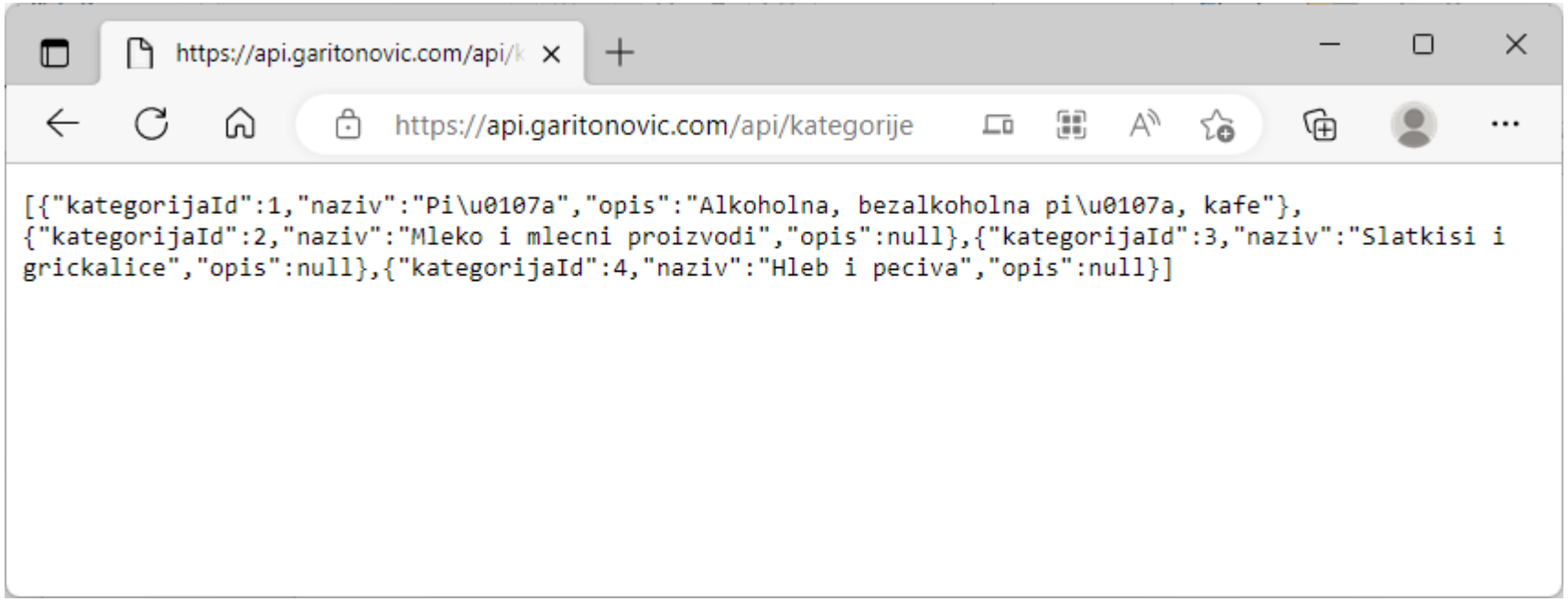
https://api.garitonovic.com/



web api dozvoljava CORS zahteve

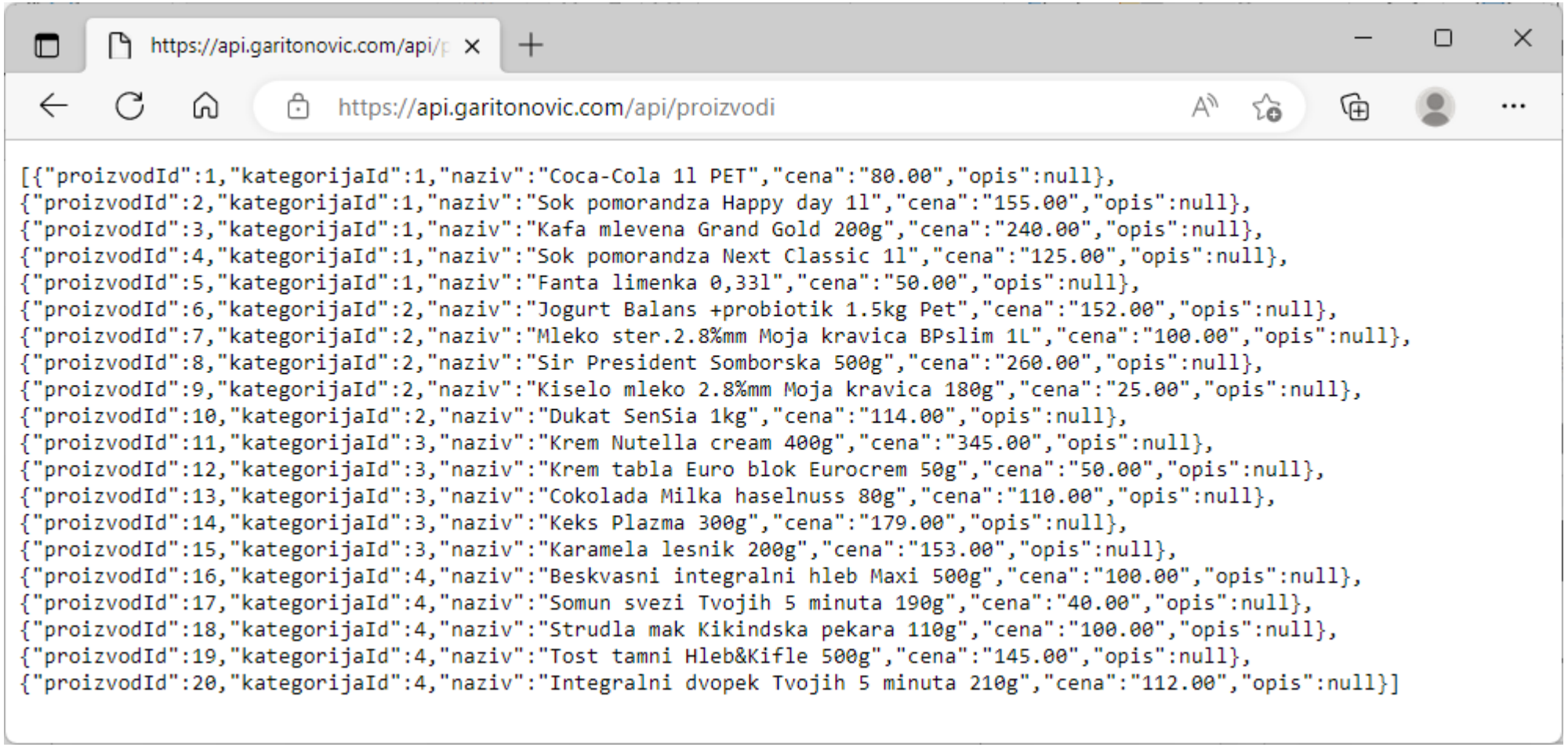
Prikaz kategorija

https://api.garitonovic.com/api/kategorije



Prikaz proizvoda

https://api.garitonovic.com/api/proizvodi



Klasa Proizvod

```
export class Proizvod {
    constructor(public proizvodId: number, public kategorijaId: number,
 public naziv: string, public cena: number, public opis: string ) {
}

}
```

Klasa MagacinService

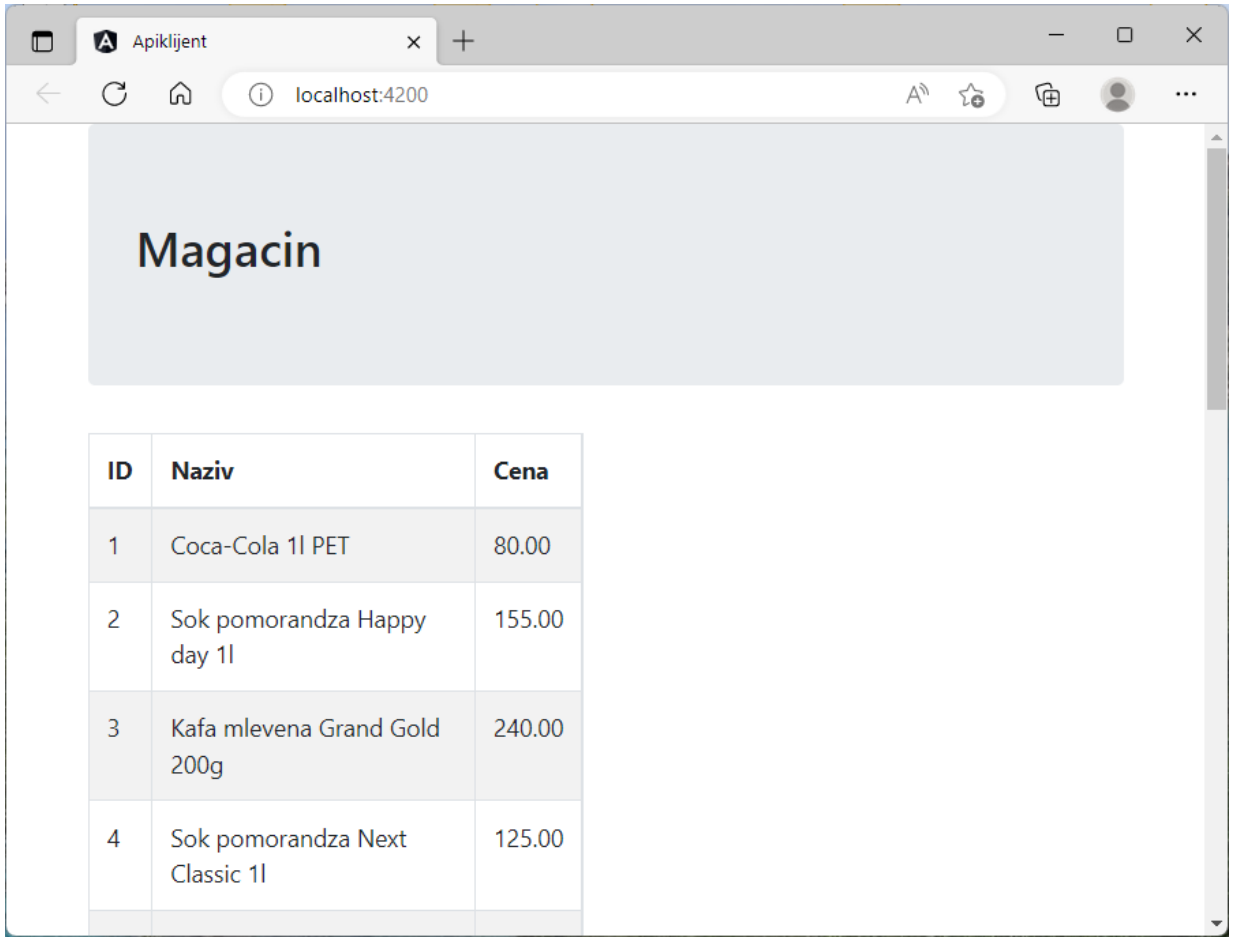
```
export class MagacinService {
  apiUrl = 'https://api.garitonovic.com/';

  constructor(private http: HttpClient) { }


 vratiProizvode(): Observable<Proizvod[]> {
    return this.http.get<Proizvod[]>(this.apiUrl + 'api/proizvodi' )
     .pipe(catchError(this.errorHandler));
  }

private errorHandler(error: HttpErrorResponse) {
      return throwError(() => error);
    }

}
```

Glavna komponenta

```
export class AppComponent {
  naziv= 'Magacin';
  proizvodi:Proizvod[] = Array<Proizvod>();


    constructor(private mServis:MagacinService) {
    }
    prikaziProizvode(): void {
      this.mServis.vratiProizvode()
        .subscribe({
         next: p=> this.proizvodi=p,
         error: grr => console.log('Greska: ' + grr)
         });
    }
    ngOnInit(): void {
      this.prikaziProizvode();
    }

}
```

```
<div class="container">

  <div class="jumbotron">
      <h2>{{naziv}}</h2>
  </div>

  <div class="row">
      <div class="col-6">
          <table class="table table-bordered table-striped">
            <thead>
                <tr>
                <th>ID</th>
                <th>Naziv</th>
                <th>Cena</th>
                </tr>
            </thead>

            <tbody>
                <tr *ngFor="let proizvod of proizvodi">
                  <td>{{proizvod.proizvodId}}</td>
                  <td>{{proizvod.naziv}}</td>
                  <td>{{proizvod.cena}}</td>
                </tr>
            </tbody>
          </table>
      </div>
  </div>
</div>
```

Prikaz podataka



Pitanje 1

Angular klasa koja komunicira sa serverom i obezbeđuje podatke za aplikaciju naziva se:

a. servis
b. pipe
c. komponenta

Odgovor: a

Pitanje 2

Angular servis opisuje se dekoratorom:

a. @Service
b. @HttpService
c. @Injectable

Odgovor: c

Pitanje 3

Ubacivanje objekta servisa u klasu komponente vrši se :

a. u metodi ngOnInit komponente
b. u konstruktoru komponente
c. u metodi ngDoCheck komponente

Odgovor: b

Pitanje 4

Klasa koja omogućava komunikaciju sa udaljenim api-jem naziva se:

a. HttpClient
b. HttpModule
c. HttpConnect

Odgovor: a

Pitanje 5

Promena podataka na serveru vrši se korišćenjem sledeće metode HttpClient objekta:

a. get
b. put
c. update

Odgovor: b

Pitanje 6

Da bi se koristila klasa HttpClient za komunikaciju sa udaljenim serverom u glavni modul aplikacije potrebno je importovati sledeći modul:

a. RemoteModule
b. ClientModule
c. HttpClientModule

Odgovor: c