Web trgovina zasnovana na tehnologijama .NET i Angular

Nemet, Ivan

VELEUČILIŠTE U BJELOVARU
PREDDIPLOMSKI STRUČNI STUDIJ RAČUNARSTVO

WEB TRGOVINA ZASNOVANA NA TEHNOLOGIJAMA .NET I ANGULAR

Završni rad br. 07/RAČ/2022

Ivan Nemet

Bjelovar, listopad 2022.

obrazac ZR - 001

Veleučilište u Bjelovaru

Trg E. Kvaternika 4, Bjelovar

1. DEFINIRANJE TEME ZAVRŠNOG RADA I POVJERENSTVA

Student: **Ivan Nemet** JMBAG: **0314020701**

Naslov rada (tema): **Web trgovina zasnovana na tehnologijama .NET i Angular**

Područje: **Tehničke znanosti** Polje: **Računarstvo**

Grana: **Primjenjeno računarstvo**

Mentor: **Krunoslav Husak, dipl. ing. rač.** zvanje: **predavač**

Članovi Povjerenstva za ocjenjivanje i obranu završnog rada:

1. **Tomislav Adamović, mag. ing. el., predsjednik**
2. **Krunoslav Husak, dipl. ing. rač., mentor**
3. **Ivan Sekovanić, mag.ing.inf.et comm.techn., član**

2. ZADATAK ZAVRŠNOG RADA BROJ: 07/RAČ/2022

U sklopu završnog rada potrebno je:

1. Analizirati i opisati Angular razvojni okvir na strani klijenta (engl. frontend web framework) i usporediti s drugim razvojnim okvirima.
2. Analizirati i opisati .NET razvojni okvir na strani poslužitelja (engl. backend web framework) i usporediti s drugim razvojnim okvirima.
3. Predložiti i opisati relacijski model baze podataka koji će omogućiti pohranu svih podataka web trgovine.
4. Izraditi i opisati rješenje web trgovine na strani klijenta koristeći Angular razvojni okvir.
5. Izraditi i opisati rješenje web trgovine na strani poslužitelja koristeći .NET razvojni okvir.

Datum: 19.07.2022. godine

Mentor: **Krunoslav Husak, dipl. ing. rač.**

Zahvaljujem svim djelatnicima Veleučilišta u Bjelovaru i roditeljima na pruženoj pomoći, i podršci prilikom izrade završnog rada.

Sadržaj

1. Uvod ... **1**

2. Uvod u Angular ... **2**

2.1 Angular ... *2*

2.2 Povijest ... *2*

2.3 Prednosti ... *2*

2.3.1 MVC ... 3

2.3.2 TypeScript ... 3

2.3.3 Dvosmjerno vezanje podataka ... 3

2.3.4 Dependency injection ... 4

2.3.5 Razvijen od strane Googlea ... 5

2.4 Mane ... *5*

2.4.1 Krivulja učenja ... 5

2.4.2 Popularnost ... 5

3. Elementi Angulara ... **6**

3.1 Komponente ... *6*

3.2 Angular sintaksa u predlošcima ... *7*

3.3 Moduli ... *8*

3.4 Direktive ... *9*

4. Korisne Angular biblioteke ... **10**

4.1 Usmjeravanje ... *10*

4.2 Obrasci ... *11*

4.3 Komuniciranje s backend servisima pomoću HTTP-a ... *11*

4.4 Testiranje ... *13*

4.5 Internacionalizacija ... *13*

4.6 Animacije ... *14*

4.7 Service Workers i PWA ... *14*

5. Uvod u .NET ... **15**

5.1 .NET ... *15*

5.2 Povijest ... *15*

5.3 Prednosti ... *15*

5.3.1 Objektno orijentirano okruženje ... 15

5.3.2 Cross-platform design ... 15

5.3.3 Sustav predmemoriranja ... 16

5.4 Mane ... *16*

5.4.1 Cijena licenciranja ... 16

6. Elementi .NET-a ... **17**

6.1 Asinkrono programiranje ... *17*

6.2 Delegati i lambda funkcije ... *17*

6.3 LINQ ... *18*

6.4 Iznimke ... *18*

6.5 Automatsko upravljanje memorijom ... *19*
6.6 Generičke zbirke ... *19*

7. Stvaranje jednostavne Angular aplikacije ... **20**
7.1 Instalacija Node.js ... *20*
7.2 Instalacija Angular CLI-a ... *20*
7.3 Stvaranje i pokretanje aplikacije ... *20*
7.4 Stvaranje komponente ... *22*

8. Stvaranje jednostavne .NET API aplikacije ... **23**
8.1 Instalacija Visual Studioa ... *23*
8.2 Stvaranje jednostavne aplikacije ... *23*

9. Izrada programskog rješenja ... **24**
9.1 Relacijski model baze podataka ... *24*
9.2 Programsko rješenje na strani klijenta ... *25*
9.2.1 Komponente ... 26
9.2.2 Servisi ... 27
9.3 Programsko rješenje na strani poslužitelja ... *29*
9.3.1 Modeli ... 29
9.3.2 Servisi ... 30
9.3.3 Kontroleri ... 32

10. ZAKLJUČAK ... **34**

11. LITERATURA ... **35**

12. OZNAKE I KRATICE ... **36**

13. SAŽETAK ... **37**

14. ABSTRACT ... **38**

1. Uvod

Svjedoci smo naglog razvoja web aplikacija. Nove tehnologije i poboljšanja događaju se svakodnevno. No, uz nove tehnologije, dolaze veći i kompleksniji zahtjevi korisnika. Nekada su se web aplikacije radile samo za stolna računala jer se za razvoj koristio *html, css* i običan *JavaScript*. Danas korisnici zahtijevaju da se aplikacija može pokrenuti na računalu, mobitelu, tabletu ili nekom drugom uređaju. Također, većina današnjih aplikacija su aplikacije na jednoj stranici (*engl. Single Page Application*). Aplikacije koje prikazuju logiku na jednoj stranici kao *Google, Facebook* ili *Google Maps*, ne osvježavaju stranicu svaki put kad se dogodi promjena, nego osvježavaju komponentu koja je zadužena za prikaz logike. Između ostalog, web aplikacije za dohvaćanje podataka sa servera koriste REST API te je navedene podatke potrebno obraditi na klijentskoj strani aplikacije. To drastično utječe na brzinu kojom aplikacija prikazuje podatke na klijentskoj strani. Responzivnost aplikacija odnosi se na sposobnost aplikacije da se automatski prilagodi veličini zaslona uređaja na kojem se pokreće. Za razvoj takvih aplikacija potrebno je koristiti razvojne okvire koji nude napredne stvari, a jedno od takvih okvira je Angular.

Cilj je ovog završnog rada opisati .NET i Angular tehnologije, istaknuti njihove prednosti i mane te upoznati obje tehnologije kroz razvoj dinamične web aplikacije u vidu internetske trgovine za proizvode Veleučilišta u Bjelovaru. Internetska trgovina koristi API za dohvaćanje podataka te ih prikazuje i obrađuje na klijentskoj strani. Korisnik može filtrirati podatke prema određenom filteru i može dodati artikl u košaricu. Prilikom dodavanja artikla u košaricu, košarica ispisuje ime, količinu i cijenu te računa ukupnu cijenu košarice. Svakom korisniku dodijeljena je njegova košarica što omogućuje aplikaciji rad s više korisnika.

2. Uvod u Angular

2.1 Angular

Angular je, prema [3], *TypeScript* razvojni okvir (*engl. framework*) otvorenog koda (*engl. open source*), razvijen od strane Googlea i koristi se primarno za razvoj dinamičnih web aplikacija gdje se sve promjene događaju na jednoj stranici. Kao platforma Angular uključuje:

- komponente koje služe za stvaranje web aplikacija
- biblioteke (*engl. libraries*) koje uključuju usmjeravanje (*engl. routing*), obrasce (*engl. forms*), komunikaciju server-klijent (*engl. client-server communication*) itd.
- paket alata koji pomaže pri razvijanju, izgradnji, testiranju i ažuriranju koda.

Neki od većih brendova koji danas koriste Angular za web aplikacije su: *BMW*, *Microsoft*, *Xbox*, *Google* i drugi.

2.2 Povijest

Prva verzija Angulara, prema [4], nastala je 2010. godine i bila je poznata kao AngularJS. AngularJS bio je sporedni projekt Googleovog zaposlenika po imenu Miško Hevery. Taj sporedni projekt trebao je pomoći prilikom izrade web aplikacija. *GitHub Community Forum*, *PayPal*, *MasterCard* i *AT&T* samo su neki od većih brendova koji su za razvoj web stranica koristili AngularJS. Kako se stvari u IT-u brzo mijenjaju, tako se mijenjao i razvoj web aplikacija, zahtjevi korisnika bili su sve teži, a razvojni tim AngularJSa približio se granicama mogućnosti razvojnog okvira. Sve više ljudi počelo je koristiti ReactJS za razvoj aplikacija te se tako razvojni tim odlučio za stvaranje novog razvojnog okvira naziva Angular 2.0, ali nisu željeli biti vezani za prethodni dizajn AngularJSa. Tako je 2016. godine prvi put izašla prva službena verzija Angulara koja za razliku od AngularJSa koristi *TypeScript* kao primarni jezik.

2.3 Prednosti

Od prve do trenutne verzije Angular podliježe raznim promjenama, a svakom verzijom uvodi nove značajke i prednosti nad ostalim razvojnim okvirima. Ovo su neke prednosti nad običnim JavaScriptom i ostalim razvojnim okvirima kao što su *ReactJS* i *VueJS*.

2.3.1 *MVC*

***M**odel **V**iew **C**ontroler* način je dizajniranja aplikacije kod kojega se logika korištena u aplikaciji izolira od korisničkog sloja. MVC sastoji se od tri dijela:

- Model (*engl. Model*) – upravlja podacima aplikacije i odgovara na zahtjeve pogleda i kontrolera.
- Pogled (*engl. View*) – prezentira podatke korisnicima u određenom formatu na zahtjev kontrolera.
- Kontroler (*engl. Controller*) - prima sve zahtjeve aplikacije te zatim komunicira s modelom kako bi pripremio sve potrebne podatke i pogledom kako bi prezentirao podatke korisnicima.

2.3.2 *TypeScript*

Primarni programski jezik Anuglara, prema [1], je *TypeScript* koji dodaje puno sintaksnog šećera *(engl. syntax sugar*) u odnosu na običan JavaScript. U uređivačima koda omogućuje se pristup pomoći za vrijeme pisanja koda što podrazumijeva lakše uočavanje pogrešaka i značajki te čišći kod.

2.3.3 *Dvosmjerno vezanje podataka*

Dvosmjerno vezanje podataka, prema [3], način je kojim komponente u aplikaciji dijele podatke. Koristi se za slušanje događaja (engl. events) i istovremeno ažuriranje podataka između roditeljske (*engl. parent*) i dječje (engl. child) komponente.
Programskim kodom 2.1. i 2.2. prikazano je dvosmjerno vezanje podataka. U *TypeScript* datoteci komponente, dodan je podatak *title* i dodijeljena mu je vrijednost *Hello World*. Predložak komponente sadrži polje koje prima model kao vrijednost polja te se ispod njega ispisuje vrijednost varijable *title*. Upisom teksta u polje, u realnom vremenu, mijenja se vrijednost polja te sam podatak *title*.

Programski kod 2.1.: Dvosmjerno vezanje podataka, predložak

```
<input type="text" [(ngModel)]="title">
<h5>{{title}}</h5>
```

Programski kod 2.2: Dvosmjerno vezanje podataka, TypeScript

```
export class TwoWayBindingComponent implements OnInit {

  title: string = 'Hello world!'

  constructor() { }

  ngOnInit(): void {
  }

}
```

2.3.4 ***Dependency injection***

Dependencies su prema [3], servisi ili objekti koje klasa treba za obavljanje funkcija. *Dependency injection* način je dizajniranja aplikacije u kojoj klasa zahtijeva servise iz vanjskih izvora umjesto da ih klasa stvori. Prilikom instanciranja klase Angular pruža korištenje *dependenciesa* kako bi povećao fleksibilnost i modularnost u aplikacijama. Programskim kodom 2.3. i 2.4. prikazan je primjer korištenja *dependency injectiona* . Servis dohvaća artikle, zatim komponenta u konstruktoru instancira servis i ima pristup svim metodama i varijablama unutar servisa.

Programski kod 2.3.: Dependency injection, servis

```
export class ArticlesService {

  constructor(private httpClient: HttpClient) { }

  getArticles(): Observable<Article[]> {
    return this.httpClient.get<Article[]>(API);
  }

}
```

Programski kod 2.4: Dependency injection, komponenta

```
export class ArticleDisplayComponent implements OnInit {
  constructor(private articleService: ArticlesService) { }

  ngOnInit(): void {
    this.articleService.getArticles().subscribe((items) => {
      this.articles = items;
    })
  }
}
```

2.3.5 *Razvijen od strane Googlea*

Razvijen od strane Googlea znači da iza Angulara stoji tvrtka od povjerenja i tvrtka kojoj zajednica vjeruje. Upravo to daje programerima povjerenje da će se razvojni okvir održavati, a svi problemi rješavati uz pomoć zajednice.

2.4 Mane

2.4.1 *Krivulja učenja*

Angular je razvojni okvir što znači da ima određeni skup pravila koja se moraju naučiti i slijediti. Iz tog razloga moguće su poteškoće kod početnika koji nisu upoznati s *JavaScriptom*.

2.4.2 *Popularnost*

Za većinu projekata programeri su se dvoumili između Angulara i Reacta sve dok se prije nekoliko godina nije pojavio Vue.js koji je lakši za učenje, ima bolju dokumentaciju, bolje performanse i fleksibilniji je.

3. Elementi Angulara

3.1 Komponente

Svaka Angular aplikacija, prema [3], građena je od komponenata, a svaka komponenta sastoji se od:

- *HTML* predloška, tj. kostura web stranice
- *TypeScript* klase koja određuje ponašanje komponente
- *CSS* predloška koji određuje izgled komponente.

Instanca komponente ima životni ciklus koji počinje kada Angular instancira klasu komponente i renderira prikaz komponente zajedno s određenim pogledima. Životni ciklus nastavlja se i prilikom ažuriranja ili mijenjanja svojstava komponente. Životni ciklus završava kada Angular uništi instancu komponente i ukloni njezin renderirani predložak iz DOM-a.

Programskim kodom 3.1., 3.2. i 3.3. prikazan je sadržaj komponente koja ispisuje tekst *Hello World* i ima polje za unos. U TypeScript datoteci komponente nalazi se dekorator komponente i sama klasa komponente. Dekorator komponenti sadrži selektor komponente te putanju do predloška i dizajna komponente. Klasa implementira određeni životni ciklus i u klasi nalazi se sva programska logika koju komponenta obavlja. Predložak sadrži *HTML* elemente koji se prikazuju prilikom pokretanja aplikacije, a stil određuje izgled i prikaz komponente.

Programski kod 3.1..: Sadržaj komponente, typescript

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

@Component({
  selector: 'app-two-way-binding',
  templateUrl: './two-way-binding.component.html',
  styleUrls: ['./two-way-binding.component.css']
})
export class TwoWayBindingComponent implements OnInit {

  title: string = 'Hello world!'

  constructor() { }

  ngOnInit(): void {
  }
}
```

Programski kod 3.2.: Sadržaj komponente, predložak

```
<input type="text" [(ngModel)]="title">
<h5>{{title}}</h5>
```

Programski kod 3.3.: Sadržaj komponente, stil

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

@Component({
  selector: 'app-two-way-binding',
  templateUrl: './two-way-binding.component.html',
  styleUrls: ['./two-way-binding.component.css']
})
export class TwoWayBindingComponent implements OnInit {

  title: string = 'Hello world!'

  constructor() { }

  ngOnInit(): void {
  }
}
```

3.2 Angular sintaksa u predlošcima

Predložak prema [1], određuje kako se elementi komponenata prikazuju na klijentskoj strani. Izgleda kao uobičajeni HTML, osim što sadrži Angular sintaksu. Angular sintaksa mijenja HTML na temelju aplikacija, stanja aplikacije itd. Programskim kodom 3.4. prikazana je Angular sintaksa unutar html elemenata. Pomoću *for* petlje, kod će izvršiti šest iteracija. *If* definira uvjete izvršavanja koda, u ovom slučaju naslov će se ispisati samo ako je varijabla cjelobrojno djeljiva s brojem dva.

Programski kod 3.4.: Predložak komponente

```
<div *ngFor="let i of [1,2,3,4,5,6]">
    <h5 *ngIf="i % 2 === 0">{{title}}</h5>
</div>
```

3.3 Moduli

Moduli, prema [3], izvrstan su način organiziranja komponenata aplikacije i dodatnih biblioteka dodanih u aplikaciju. Dakle, to su spremnici koji mogu sadržavati komponente, servise i drugo. U Angularu se to naziva NgModules i svaka Angular aplikacija ima barem jednu NgModule klasu koja je ujedno i korijenski modul (*engl. root module).* Klasa je označena dekoratorom NgModule koji sadrži polja:

- deklaracije (*engl. decalarations*) - govori Angularu koje komponente pripadaju korijenskom modulu. Komponente se dodaju automatski u polje deklaracija prilikom stvaranja komponente iz Angular CLI-a
- uvozi (*engl. imports*) – module koji ne dolaze prilikom stvaranja aplikacije kao što su: *FormsModule, HttpClient* ili *RoutingModule*, potrebno je dodati u polje uvoza kako bi se moduli mogli koristiti u aplikaciji
- usluge – ovdje su unose servisi koji se koriste u aplikaciji, prilikom deklariranja servisa u ovo polje, servis postaje dostupan u cijeloj aplikaciji
- bootstrap – u ovo polje se unose komponente koje su baza vlastitog stabla, unosom komponente u *bootstrap* polje stvaraju se komponente koje ispunjavaju to stablo.

Programski kod 3.5. prikazuje izgled *AppModule.ts* datoteke prilikom stvaranja nove aplikacije.

Programski kod 3.5.: Prikaz komponente AppModule.ts

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

import { AppComponent } from './app.component';

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

3.4 Direktive

Direktive, prema [1], klase su koje dodaju dodatno ponašanje u Angular aplikacijama. Već ugrađene Angular direktive koriste se za obrasce, liste, stilove itd. Programskim kodom 3.6. i 3.7. opisan je postupak korištenja vlastite direktive koja mijenja boju teksta u aplikaciji. Unosom naredbe *ng generate directive* i zatim naziv direktive, u Angular CLI, stvara se nova direktiva koja nema implementiranu logiku. Direktiva uvozi *ElementRef* modul, koji dohvaća referencu na HTML element nad kojim se poziva direktiva. U konstruktoru je potrebno instancirati objekt kako bi svojstvo unutar modula, *nativeElement*, dobilo referencu na HTML element. Zatim, selektor direktive potrebno je dodati u predložak na oznaku *HTML* elementa.

Programski kod 3.6.: Direktiva

```
import { Directive, ElementRef } from
'@angular/core';

@Directive({
  selector: '[appChangestyle]'
})
export class ChangestyleDirective {

  constructor(private er: ElementRef) {
    er.nativeElement.style = "color: red";
  }
}
```

Programski kod 3.7.: Predložak

```
<input type="text" class="inputText" appChangestyle
[(ngModel)]="title">
```

4. Korisne Angular biblioteke

4.1 Usmjeravanje

U Angular aplikacijama koje koriste samo jednu stranicu ili SPA, prema [1], umjesto odlaska na server kako bismo učitali novu stranicu, možemo koristiti usmjeravanje i tako prikazati ili sakriti dijelove zaslona koji odgovaraju određenim komponentama. Unosom naredbe *ng generate module app-routing -- flat --module=app*, u Anuglar CLI. U aplikaciju uvozi se modul za usmjeravanje te pomoću dodanih parametara, modul se dodaje u korijenski modul aplikacije. Programskim kodom 4.1. i 4.2. opisan je postupak stvaranja putanje do komponente i njihova implementacija. U modulu za usmjeravanje potrebno je uvesti *RouterModule* i *Routes* te stvoriti varijablu koja sprema rute do komponenata. U ovom slučaju ruta *shop* usmjerava korisnika do *MainLayoutComponent* komponente, a ruta *login* usmjerava korisnika do *LoginComponent* komponente. Uvozom modula *Router* u određenu komponentu i instaciranjem objekta modula unutar konstruktora komponente, ostvaruje se pristup metodi *navigate*. Metoda prima putanju koja je prethodno određenaunutar samog modula.

Programski kod 4.1.: Modul za usmjeravanje

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { MainLayoutComponent } from 'src/app/components/main-
layout/main-layout.component';
import { LoginComponent } from './components/login/login.component';

const routes: Routes = [
  { path: 'shop', component: MainLayoutComponent},
  { path: 'login', component: LoginComponent}
];

@NgModule({
  declarations: [],
  imports: [
    RouterModule.forRoot(routes)
  ],
  exports: [RouterModule]
})
export class AppRoutingModule {}
```

Programski kod 4.2.: TypeScript komponenta

```
constructor( private router: Router) { }

NavigateUserToLoginPage(){
   this.router.navigate(['/login']);
}
```

4.2 Obrasci

Obrasci se, prema [1], koriste kako bi korisnicima omogućili registraciju, prijavu i ažuriranje profila. Postoje dvije vrste obrazaca: reaktivne i vođene predloškom. Reaktivni obrasci koriste *FormsModule*, temelje se na direktivama i uklanjaju logiku validacije iz predloška te tako čine kod čišćim. Reaktivne forme bolji su izbor prilikom stvaranja aplikacije jer su moćniji i jednostavniji za korištenje.

4.3 Komuniciranje s *backend* servisima pomoću HTTP-a

Većina web aplikacija, prema [1], za dohvaćanje i učitavanje podataka mora komunicirati s poslužiteljem preko HTTP protokola. Angular pruža klijentski HTTP API za aplikacije koji nudi sljedeće značajke:

- mogućnost traženja po tipu objekta
- pojednostavljeno upravljanje pogreškama
- mogućnost testiranja
- presretanje zahtjeva i odgovora.

HttpClient modul za sve pozive prema serveru, koristi *Observable* tip podatka. *Observable* je značajka unutar Angulara koja pruža podršku za prijenos poruka između dijelova aplikacije. Često se koriste za rukovanje događajima i višestrukim vrijednostima. Instanca *observable* počinje prikazivati vrijednosti tek kad se netko na nju pretplati. Pretplata se vrši pozivom metode *Subsribe*.

Za dohvaćanje podataka sa servera, *HttpClient* pruža metodu *Get*. Metoda *Get* asinkrono šalje *HTTP* zahtjev prema serveru i vraća *Observable* podatak. *Get* metoda prima dva podatka. Prvi je tip objekta kojeg dohvaćamo, a drugi je poveznica do servera. Programski

kod 4.3., 4.4, 4.5. i 4.6. prikazuju primjere korištenja metode za komunikaciju s poslužiteljem .

Programski kod 4.3.: Primjer Get metode

```
configUrl = 'assets/config.json';

getObject() {
  return this.http.get<Object>(this.configUrl);
}
```

Slanje podataka na poslužitelj odvija se preko *POST* zahtjeva, *HtppCLient* pruža *Post* metodu koja nasljeđuje *Observable* te prima tip podatka kojeg šaljemo i poveznicu do poslužitelja.

Programski kod 4.4.: Primjer Post metode

```
configUrl = 'assets/config.json';

addObject(object: Object): Observable<Object> {
  return this.http.post<Object>(this. configUrl, object);
}
```

Za ažuriranje zapisa, potrebno je koristiti *PUT* zahtjev, odnosno *Put* metodu unutar *HttpClient* modula. Kao i *Post* metoda, *Put* metoda također prima tip podatka i poveznicu do servera.

Programski kod 4.5.: Primjer Put metode

```
configUrl = 'assets/config.json';

updateObject(object: Object): Observable<Object> {
  return this.http.put<Object>(this.configUrl, object);
}
```

Brisanje zapisa odvija se pomoću metode *Delete* iz *HtppClient* modula. Metoda šalje *DELETE* zahtjev prema serveru te prima identifikacijski broj objekta i vraća *Observable*.

Programski kod 4.6.: Primjer Delete metode

```
configUrl = 'assets/config.json';

deleteObject(id: number): Observable<any> {
  return this.http.delete(this.configUrl + id);
}
```

4.4 Testiranje

Testiranje aplikacija, prema [3], koristi se za utvrđivanje radi li aplikacija kako je prethodno zamišljeno. Svaka aplikacija stvorena pomoću Angular CLI-a već je spremna za testiranje. Testiranje se obavlja pokretanjem naredbe *ng test* u Angular CLI-u. Pokretanjem naredbe pokreće se Angular testno okruženje pod nazivom *Karma*. U konzoli bismo trebali dobiti informacije o prolaznosti testova koji je prikazan programskim kodom 4.7.

Programski kod 4.7.: Primjer prolaznosti testova

```
10% building modules 1/1 modules 0 active
…INFO [karma]: Karma v1.7.1 server started at http://0.0.0.0:9876/
…INFO [launcher]: Launching browser Chrome …
…INFO [launcher]: Starting browser Chrome
…INFO [Chrome …]: Connected on socket …
Chrome …: Executed 3 of 3 SUCCESS (0.135 secs / 0.205 secs)
```

4.5 Internacionalizacija

Internacionalizacija ili kraće i18n, prema [3], proces je dizajniranja i pripreme aplikacije za korištenje na različitim lokacijama diljem svijeta. Lokalizacija je proces izgradnje aplikacije za različite lokalne postavke. Lokalizacija identificira regiju u kojoj ljudi govore određenim jezikom. Proces lokalizacije uključuje:

- izdvajanje teksta za prijevod na različite jezike i
- formatiranje podataka za određeni jezik.

Također, lokalizacija određuje oblik i analizu sljedećih detalja:

- mjerne jedinice, uključujući datum i vrijeme, brojeve i valute
- imena, uključujući vremenske zone, jezike i države.

4.6 Animacije

Animacije, prema [3], pružaju iluziju kretanja elemenata na stranici i mogu aplikaciju učiniti zabavnijom i ugodnijom za korištenje. Prijelazi web stranica bez animacija mogu djelovati naglo i tako pogoršati korisničko iskustvo. Dobre animacije daju korisnicima priliku otkriti kako aplikacija djeluje na njihove radnje i intuitivno skreću pozornost korisnika gdje je to potrebno. Animacije se mogu dobiti transformacijom i promjenom stila HTML elemenata tijekom nekog perioda.

4.7 Service Workers i PWA

Angularov servisni radnik, prema [3], osmišljen je za optimiziranje korisničkog iskustva prilikom korištenja aplikacije preko spore ili nepouzdane mrežne veze, istovremeno minimizirajući rizike posluživanja zastarjelog sadržaja.

5. Uvod u .NET

5.1 .NET

Razvojna platforma, prema [5], otvorenog koda za programere, razvijena od strane Microsofta za izgradnju različitih aplikacija. Za izradu web, mobilnih, desktop i IoT aplikacija dopušta korištenje više jezika, uređivača i biblioteka. .NET aplikacije mogu biti napisane u C#, F# i Visual Basic jezicima. No, bez obzira na jezik u kojem se piše kod, aplikacije će raditi na bilo kojem operativnom sustavu. Neki od većih brendova koji za rješavanje poslovnih problema koriste .NET su: *Stack Overflow*, *UPS*, *Microsoft* i *Siemens*.

5.2 Povijest

Prva verzija .NET razvojnog okvira, prema [5], objavljena je 2002. godine pod nazivom *.NET 1.0*. Glavne značajke bile su *Common Language Runtime* ili CLR, i objektno orijentirani razvoj aplikacija. Kako se .NET svake godine sve više razvijao te uz promjenu značajki, promijenilo se ime iz .NET u .NET Core. Trenutno je zadnja službena verzija .NET 6.0 u kojoj se pojednostavio razvoj aplikacija, poboljšane su performanse i produktivnost te je promijenjeno ime iz .NET Core u .NET. U trenutku pisanja rada u pretpregledu (*engl. preview*) verzija je .NET 7.0.

5.3 Prednosti

5.3.1 *Objektno orijentirano okruženje*

.NET, prema [2], temelji se na konceptu objektno orijentiranog programiranja. Aplikacija se dijeli na manje dijelove što programerima omogućuje lakši rad. Nakon što je jedan dio gotov, programeri mogu prijeći na drugi dio te kasnije kad su svi manji dijelovi gotovi, dijelovi se mogu kombinirati.

5.3.2 *Cross-platform design*

Od prve do trenutne verzije, prema [5], primijenilo se dosta stvari i uvelo puno značajki. Jedna od važnijih značajki mogućnost je dizajniranja aplikacija za više operativnih sustava. Kao što je u uvodu rada spomenuto, zahtjevi klijenta su da aplikacije budu skalabilne i

responzivne. Pojam skalabilnosti i responzivnosti aplikacija, odnosi se na mogućnost pokretanja aplikacije na različitim uređajima i na samu brzinu kojom aplikacija prikazuje podatke korisniku.

5.3.3 *Sustav predmemoriranja*

Sustav predmemoriranja, prema [5], omogućuje pohranu podataka u memoriju radi bržeg pristupa podacima. Kad se podacima ponovno pristupi, aplikacije mogu dobiti podatke iz predmemorije umjesto da ih dohvaćaju iz izvornog izvora. To može poboljšati performanse i skalabilnost. Osim toga, predmemoriranje čini podatke dostupnima kada je izvor podataka privremeno nedostupan.

5.4 Mane

5.4.1 *Cijena licenciranja*

Mnogi aspekti .NET razvojnog okvira traže licenciranje. Troškovi licenciranja, prema [6], nisu jeftini i mogu se nakupiti. Što je projekt zahtjevniji, može biti skuplji. Na primjer, Visual Studio može stajati 540 američkih dolara godišnje.

6. Elementi .NET-a

6.1 Asinkrono programiranje

Pristup web resursima ponekad je spor ili onemogućava aktivnost. U sinkronom procesu ta aktivnost onemogućava aplikaciju i korisnik mora čekati, dok kod asinkronog procesa aplikacija može nastaviti s drugim poslom koji ne ovisi o web resursu. Međutim, pisanje asinkronih aplikacija može biti kompliciranо čineći aplikacije težim za otklanjanje pogrešaka i za održavanje. Od verzije 4.5 pa nadalje, prema [5], sve verzije imaju podršku za korištenje asinkronog programiranja u aplikacijama čija struktura koda izgleda kao sinkrono programiranje. Na primjeru programskog koda 6.1. moguće je vidjeti primjer funkcije asinkronog programiranja.

Programski kod 6.1..: Primjer asinkrone metode

```
public async Task<ActionResult<IEnumerable<Article>>> GetArticle()
{
    return Ok(await articleService.All());
}
```

6.2 Delegati i lambda funkcije

Delegat je, prema [5], tip koji predstavlja referencu na metodu s određenim popisom parametara i vrstom povratnog tipa. Instanciranjem delegata moguće je pridružiti njegovu instancu bilo kojoj metodi koji ima kompatibilan potpis i povratni tip. Delegati se koriste za prosljeđivanje metoda kao argumente drugim metodama. Lambda izrazi koriste se za stvaranje anonimnih funkcija. Lambda izraz može biti u dva oblika: lambda izraz i lambda iskaz. U oba slučaja, na lijevoj strani lambda operatora specificiraju se ulazni parametri, a na desnoj strani blok izraza. Programski kod 6.2. opisuje postupak korištenja delegata.

Programski kod 6.2.: Primjer korištenja delegata

```
// Stvaranje delagata
public static void DelegateMethod(string message)
{
    Console.WriteLine(message);
}
// Instanciranje delegata
Del handler = DelegateMethod;
// Pozivanje delegata
handler("Hello World");
```

6.3 LINQ

LINQ je, prema [5], jedinstvena sintaksa upita za dohvaćanje podataka iz različitih izvora i formata. Integriran je u *Csharp* (C#) i *Visual Basic* čime se eliminira nesukladnost između programskih jezika i baza podataka te pruža jedinstveno sučelje za dohvaćanje podataka iz različitih izvora podataka. Korištenjem sintakse upita mogu se izvoditi operacije filtriranja, sortiranja i grupiranja nad podacima s minimalnom količinom koda. *LINQ* upiti se mogu pisati za *SQL* baze podataka, *XML* dokumente, *ADO.NET* skupove podataka i sve ostale kolekcije objekata koja podržava *IEnumerable* sučelje. Primjer korištenja *LINQ* upita opisan je programskim kodom 6.3.

Programski kod 6.3.: Primjer korištenja LINQ upita

```
public async Task<IEnumerable> GetArticlesFilteredByPrice(int min,
int max)
{
   return await DbContext.Article.Where( x => x.Price >= min &&
x.Price <= max).ToListAsync();
}
```

6.4 Iznimke

Iznimka je, prema [5], svako stanje pogreške ili neočekivano ponašanje aplikacija prilikom nailaska na problem. Iznimka se može pojaviti zbog grešaka u kodu, nedostupnih resursa, neočekivanih uvjeta itd. Prilikom pojavljivanja iznimke, od nekih iznimaka aplikacija se neće srušiti, ali od nekih hoće. Iznimke se prikazuju u području koda gdje se potencijalni

problem može pojaviti i aplikacije moraju biti u stanju rukovati pogreškama koje se javljaju tijekom izvođenja na dosljedan način. *.NET* pruža model za obavještavanje aplikacija o potencijalnim pogreškama. Programski kod 6.4. opisuje rukovanje iznimkama.

Programski kod 6.4.: Primjer rukovanja iznimkama

```
public async Task<IActionResult> PutArticleInCart(int id, Cart cart)
{
       if (id != cart.Id){
           return BadRequest();
       }
       if (await cartService.Update(cart)){
           return NoContent();
       }
     return StatusCode(StatusCodes.Status500InternalServerError);
}
```

6.5 Automatsko upravljanje memorijom

Automatsko upravljanje memorijom je, prema [5], jedna od usluga *Common Language Runtimea* koja se pruža prilikom izvođenja aplikacije. *Garbage collector* upravlja dodjelom i uzimanjem memorije aplikaciji koja se izvodi. Za razliku od programskih jezika kao što su *C* i *C++* gdje se mora voditi računa o objektima i njihovom zauzeću memorije da ne bi došlo do curenja memorije (*engl. memory leak*), u .NETu to CLR koji obavlja automatski.

6.6 Generičke zbirke

Prvi put predstavljene u .NETu 2.0, generičke zbirke su, prema [5], odredile strukturu podataka za pohranu podataka bez obvezivanja na stvarni tip podatka. Prije dodavanja generičkih zbirki, prilikom dodavanja elemenata u kolekciju ili čitanja elemenata iz kolekcije, svaki element se u pozadini trebao pretvoriti u tip Objekt što je utjecalo na performanse. Na primjer, List<T> generička je zbirka koja se može deklarirati i koristiti s bilo kojim tipom podatka kao što je List<String> ili List<Int>.

7. Stvaranje jednostavne Angular aplikacije

U ovom poglavlju bit će objašnjeno kako instalirati Angular razvojni okvir te kako stvoriti jednostavnu Angular aplikaciju koja sadrži i pokriva sve glavne elemente i značajke Angulara.

7.1 Instalacija Node.js

Prije instaliranja Angulara, potrebno je instalirati Node.js i NPM. NPM je alat za instaliranje biblioteka trećih strana koje su bitne za pokretanje aplikacija. Node.js služi za uključivanje svih potrebnih elemenata u projekt koji su bitni za izvršavanje koda.

7.2 Instalacija Angular CLI-a

Nakon instalacije Node.jsa i NPM-a potrebno je instalirati Angular CLI. Angular CLI je sučelje naredbenog retka koje se koristi za razvoj, inicijalizaciju i održavanje aplikacije izravno iz naredbenog retka. Unosom naredbe *npm install -g @angular/cli* u prethodno instalirani NPM, instalirat će se Angular naredbeni redak. U naredbi, parametar g označava globalnu instalaciju što znači da kad je jednom instaliran naredbe redak, za svaku sljedeću aplikaciju stvorenu na računalu, Angular naredbeni redak nije potrebno ponovno instalirati.

7.3 Stvaranje i pokretanje aplikacije

Unosom naredbe prikazanoj na slici 7.1, stvara se prva aplikacija u naredbenom retku. Prilikom stvaranja aplikacije, Angular nudi opciju dodavanja usmjeravanja ili navigacije u projekt. Odabirom opcije *N* za ne, odbija se dodavanje usmjeravanja u projekt. Za potrebe ovog rada, usmjeravanje će se dodati naknadno u projekt.

```
D:\Angular>ng new my-first-app
? Would you like to add Angular routing? (y/N)
```

Slika 7.1. Primjer unosa naredbe za stvaranje aplikacije

Zatim se prikazuje odabir ponuđenih stilskih jezika koji se mogu koristiti u aplikaciji. Neki od ponuđenih stilskih jezika su :

- CSS - standardni stilski jezik koji opisuje kako se prikazuju elementi web aplikacija
- SCSS - naprednija verzija standardnog stilskog jezika. Prednosti SCSS-a su čišći kod jer koristi manje linija koda, raznovrsnost i kompatibilan je s CSS-om tako da su mogu koristiti iste biblioteke
- Sass - također stilski jezik koji dodaje neke napredne funkcionalnosti u odnosu na standardni CSS. Jedna od prednosti nad CSS-om je sintaksni šećer (*engl. syntax sugar*) koji pomaže pri pisanju koda i čini sam proces pisanja koda jednostavnijim.
- Less - proširenje za CSS koje izgleda isto kao CSS samo jednostavnije za korištenje. Koristi JavaScript alat naziva *Less.js* za pretvaranje Less stilova u CSS stilove.

Primjer odabira stilskih jezika moguće je vidjeti na slici 7.2.

```
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS
SCSS   [ https://sass-lang.com/documentation/syntax#scss               ]
Sass   [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
Less   [ http://lesscss.org                                            ]
```

Slika 7.2. Primjer odabira stilskih jezika

Nakon odabira stilskog jezika za stilove, instaliraju se potrebni paketi i nakon nekoliko sekundi, stvara se aplikacija. Ulaskom u mapu projekta pomoću naredbe *cd* i naziva projekta te unosom naredbe *dir*, mogu se vidjeti sve stvorene datoteke u projektu.
Unosom naredbe *ng serve* u naredbeni redak lokalno se pokreće aplikacija na *portu* 4200.
Kopiranjem linka *http://localhost:4200/* u web preglednik otvara se web stranica na kojoj se nalaze koraci i poveznice na elemente koji su potrebni za daljnju nadogradnju aplikacije. Unosom kombinacije *ctrl + c* zaustavlja se izvođenje aplikacije. Također, unosom naredbe *code .*, datoteke projekta otvaraju se u uređivaču koda *Visual Studio Code.*

7.4 Stvaranje komponente

Komponenta se može stvoriti na dva načina. Jedan način je preko naredbenog retka, a drugi način je ručno dodavanjem datoteka u projekt. Stvaranje komponente iz naredbenog retka obavlja se unoseći *ng generate component* naredbu te naziv komponente u Angular CLI, ili skraćeno *ng g c* pa naziv komponente. Stvaranje komponenata iz naredbenog retka je jednostavnije i brže zbog toga što se potrebne datoteke za komponentu generiraju automatski i odmah se komponenta dodaje u glavni modul, odnosno *app.module.ts*. datoteku. Zatim, stvorenu komponentu potrebno je dodati u glavnu komponentu projekta naziva *app.component.html*. Komponenta se dodaje unosom *selektora* nove komponente u glavnu komponentu. Svaka komponenta u predlošku ima paragraf te prilikom ponovnog pokretanja aplikacije, ispisuje se sadržaj paragrafa što upućuje na to da je komponenta uspješno dodana u projekt. Primjer stvaranja komponente moguće je vidjeti na slici 7.3.

```
D:\Angular\my-first-app>ng generate component text
CREATE src/app/text/text.component.html (19 bytes)
CREATE src/app/text/text.component.spec.ts (585 bytes)
CREATE src/app/text/text.component.ts (267 bytes)
CREATE src/app/text/text.component.css (0 bytes)
UPDATE src/app/app.module.ts (388 bytes)
```

Slika 7.3. Primjer stvaranja komponente

8. Stvaranje jednostavne .NET API aplikacije

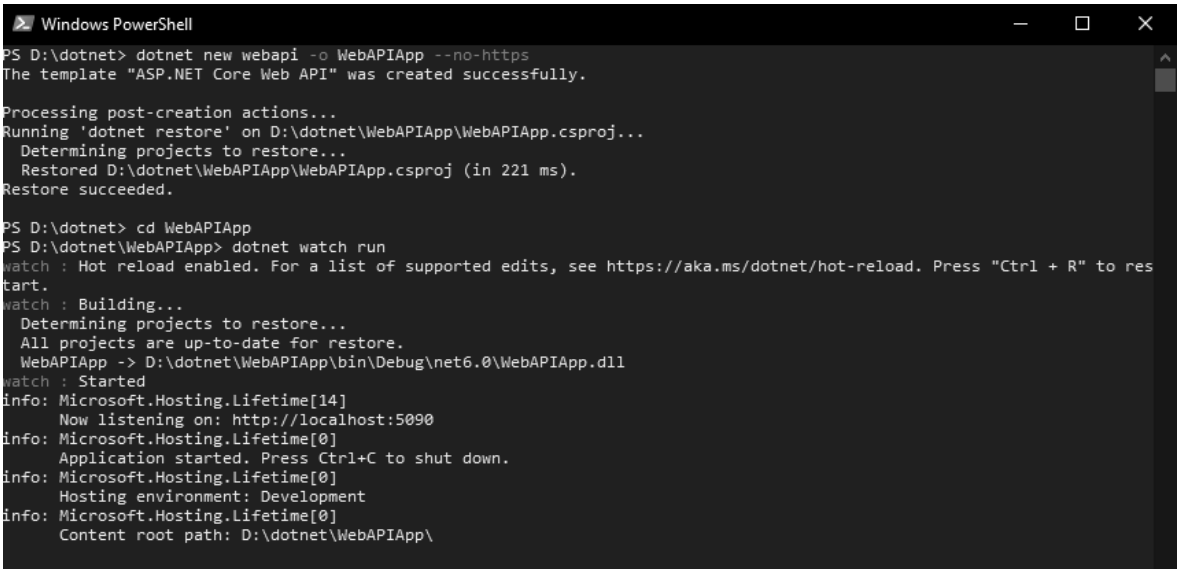
U ovom poglavlju objašnjava se što je sve potrebno i kako stvoriti jednostavnu .NET API aplikaciju.

8.1 Instalacija Visual Studioa

Ako Visual Studio nije instaliran, potrebno ga je instalirati. Ako je instaliran, onda je potrebno provjeriti je li instalirana podrška za *ASP.NET and web development*.

8.2 Stvaranje jednostavne aplikacije

Aplikacija se može stvoriti na dva načina. Prvi način je iz Visual Studioa, a drugi i jednostavniji je unosom naredbe u Powershell. Pozicioniranjem u željeni direktorij i unosom naredbe *dotnet new webapi -o nazivaplikacije -no-https,* stvara se nova aplikacija. Zatim, ulaskom u stvorenu aplikaciju i unosom naredbe dotnet watch run ili dotnet run, pokreće se aplikacija i u web pregledniku otvara se Swagger UI. Aplikacija se sastoji od klase Weather koji ima javne podatke datum, temperaturu i sažetak te kontroler koji upravlja zahtjevima prema API-u. Slika 8.1. prikazuje proces i pokretanje prve aplikacije.



Slika 8.1. Stvaranja i pokretanje prve aplikacije

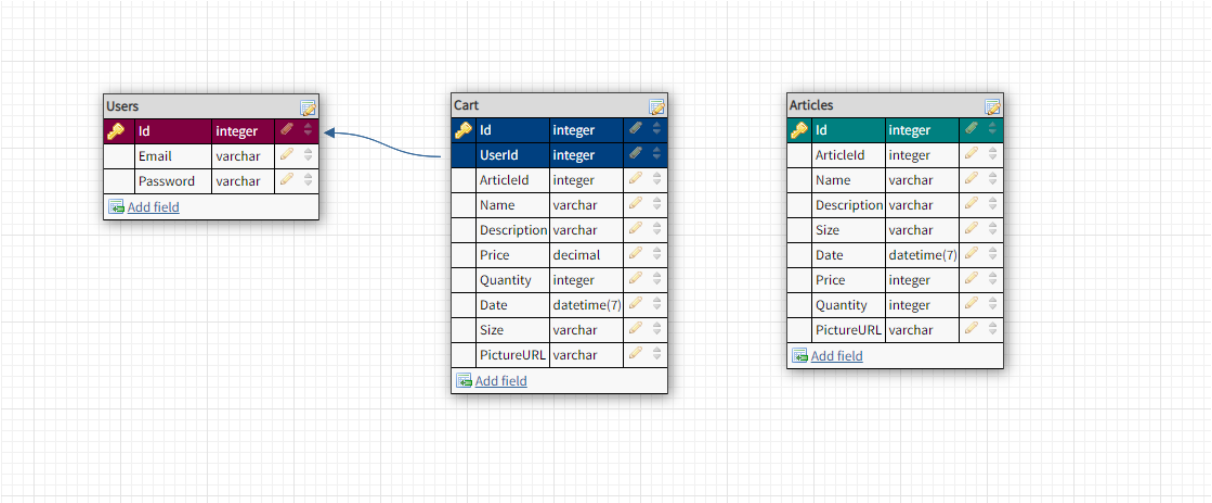
9. Izrada programskog rješenja

U ovom poglavlju opisana je izrada programskog rješenja aplikacije za potrebe ovog završnog rada. Aplikacija se sastoji od dijela za klijente, za koji je zadužen Angular, i serverskog dijela za koji je zadužen .NET API. .NET API služi za komunikaciju s bazom podataka pomoću razvojnog okvira entiteta , a Angular prikazuje podatke klijentima. Svrha aplikacije je web trgovina za proizvode Veleučilišta u Bjelovaru.

9.1 Relacijski model baze podataka

Baza podataka organizirani je sustav za spremanje, čitanje i obradu prikupljenih podataka. Svaka baza podataka sastoji se od minimalno jedne tablice. Tablice se sastoje od redaka i stupaca gdje stupci predstavljaju različite vrste podataka, a redci predstavljaju zapise u bazi. Relacijski model baza podataka odnosi se na povezivanje nekoliko tablica. Tablice se povezuju vezanjem stranog ključa jedne tablice na primarni ključ druge tablice. Na slici 9.1. moguće je vidjeti primjer relacijskog modela baze podataka. Tablica *Cart* je preko stranog ključa povezana na primarni ključ tablice *Users* jer korisnici sadrže košaricu u kojoj može biti više artikala. Tablica *Articles* ima ulogu dodavanja i spremanja dostupnih artikala u web trgovini, a tablica *Cart* sprema artikle pojedinih korisnika u košaricu. Spremanje artikala u tablicu *Cart* bitno je jer se prilikom osvježavanja aplikacije, artikli iz košarice brišu. Pomoću identifikacijskog broja, prilikom osvježavanja aplikacije, dohvaća se sadržaj košarice korisnika.

Na slikama 9.2., 9.3. i 9.4. moguće je vidjeti sadržaj tablica *Users, Cart* i *Articles.*



Slika 9.1. Prikaz relacijskog modela baze podataka

dbo.User

RESULTS

	Id	Email	Password
1	1	admin@gmail.c...	12214141
2	2	user@gmail.com	test

Slika 9.2. Sadržaj tablice *Users*

dbo.Cart

RESULTS CTRL+ALT+R

	Id	Name	Price	Quantity	ArticleId	DateAdded	Description	PictureURL	Size	UserId
1	43	Majica	20	1	1	2022-09-01 15:...	Kratka pamučn...	https://i.postim...	XL	1
2	49	Majica	20	1	1	2022-08-23 08:...	Pamučna kratk...	https://i.postim...	XL	1
3	50	Šalica	15	1	2	2022-08-25 18:...	Kvalitetna šalica	https://i.postim...	S	1
4	51	Šalica	15	1	2	2022-08-25 18:...	Kvalitetna šalica	https://i.postim...	S	1
5	52	Šalica	15	1	2	2022-08-25 18:...	Kvalitetna šalica	https://i.postim...	S	1
6	53	Šalica	15	1	2	2022-08-25 18:...	Kvalitetna šalica	https://i.postim...	S	1
7	54	Majica	20	1	1	2022-08-23 08:...	Pamučna kratk...	https://i.postim...	XL	1
8	63	Šalica	15	1	2	2022-08-25 18:...	Kvalitetna šalica	https://i.postim...	S	2
9	64	Šalica	15	1	2	2022-08-25 18:...	Kvalitetna šalica	https://i.postim...	S	2
10	65	Šalica	15	1	2	2022-08-25 18:...	Kvalitetna šalica	https://i.postim...	S	2
11	66	Šalica	15	1	2	2022-08-25 18:...	Kvalitetna šalica	https://i.postim...	S	2
12	69	Majica	20	1	1	2022-08-23 08:...	Pamučna kratk...	https://i.postim...	XL	2
13	71	Majica	20	1	1	2022-08-23 08:...	Pamučna kratk...	https://i.postim...	XL	2
14	72	Majica	20	1	1	2022-08-23 08:...	Pamučna kratk...	https://i.postim...	XL	2
15	75	Majica	20	1	1	2022-08-23 08:...	Pamučna kratk...	https://i.postim...	XL	1
16	87	Majica	20	1	1	2022-08-23 08:...	Pamučna kratk...	https://i.postim...	XL	2

Slika 9.3. Sadržaj tablice *Cart*

dbo.Article

RESULTS CTRL+ALT+R

	Id	Name	Description	Size	DateAdded	PictureURL	Price	ArticleId	Quantity
1	1	Majica	Pamučna kratk...	XL	2022-08-23 08:...	https://i.postim...	20	1	1
2	2	Šalica	Kvalitetna šalica	S	2022-08-25 18:...	https://i.postim...	15	2	1

Slika 9.4. Sadržaj tablice Articles

9.2 Programsko rješenje na strani klijenta

Klijentski dio aplikacije bavi se slanjem zahtjeva prema serverskoj strani i prikazivanjem odgovara tih zahtjeva klijentima. Sastoji se od nekoliko modela, servisa, komponenata i modula. Komponente služe za poslovnu logiku i prikazivanje te iste logike korisnicima. Servisi služe za komunikaciju sa serverskom stranom aplikacije. Prilikom razvoja aplikacije dodano je mekoliko modula kao što su bootstrap, modul za putanje itd.

9.2.1 ***Komponente***

Glavne komponente u aplikaciji su: *footer*, *header*, *login*, *main layout* i *navigation bar*. Zatim, prema principu „roditelj-dijete“, komponenta *main-layout* dijeli se na: *article display*, *article filter* i *shopping cart*. *Footer* je komponenta koja se nalazi i prikazuje na dnu stranice te sadrži elemente kao što su datum izrade. *Navigation bar* je komponenta koja se prikazuje na vrhu stranice i sadrži poveznice na često korištene stranice te tipke za prijave i registraciju korisnika. *Article display* bavi se prikazom artikala prema bootstrap album predlošku (*engl. template*). *Article filter* je komponenta koja sadrži opcije filtriranja artikala. *Shopping cart* je komponenta koja prikazuje i ažurira košaricu. Programski kod 9.1. prikazuje izgled prethodno spomenute komponente za ažuriranje i prikazivanje artikala u košarici.

Programski kod 9.1.: Primjer izgleda komponente iz aplikacije

```
import { Component, OnInit, Injectable} from '@angular/core';
import { Cart } from 'src/app/models/cart';
import { CartService } from 'src/app/services/cart.service';
import { SharedService } from 'src/app/services/shared.service';

@Injectable({
  providedIn: 'root'
})

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

export class ShoppingCartComponent implements OnInit {

  cartArticles: Cart[] = [];
  Total: number = 0;
  Id: number = 0;
  message: number;

  constructor(private cartService: CartService, private shared:
SharedService) {
```

```
  }

  ngOnInit(): void {
    this.shared.currentMessage.subscribe(message => this.message =
message);
    this.Id = this.message;
    if(this.Id !== 0){
      this.cartService.getArticles(this.Id).subscribe((items:
Cart[]) => {
        this.cartArticles = items;
        this.cartArticles.forEach(article => {
          this.Total += article.price * article.quantity;
        })
      });
    }
    return;
  }
}
```

9.2.2 ***Servisi***

Aplikacija koristi tri servisa za komunikaciju s *backend* servisima pomoću *HttpClient* i *Observable* modula. Korišteni servisi u aplikaciju su:

- *Article* – servis koji pomoću zahtjeva za dohvaćanje (engl. *GET request*) dohvaća sve artikle koji se nalaze u tablici Artikli
- *Cart* – servis koji preko identifikacijskog broja dohvaća, šalje ili briše podatke iz tablice Košarica

- *User* – servis koji iz tablice Korisnici dohvaća sve zapise korisnika.

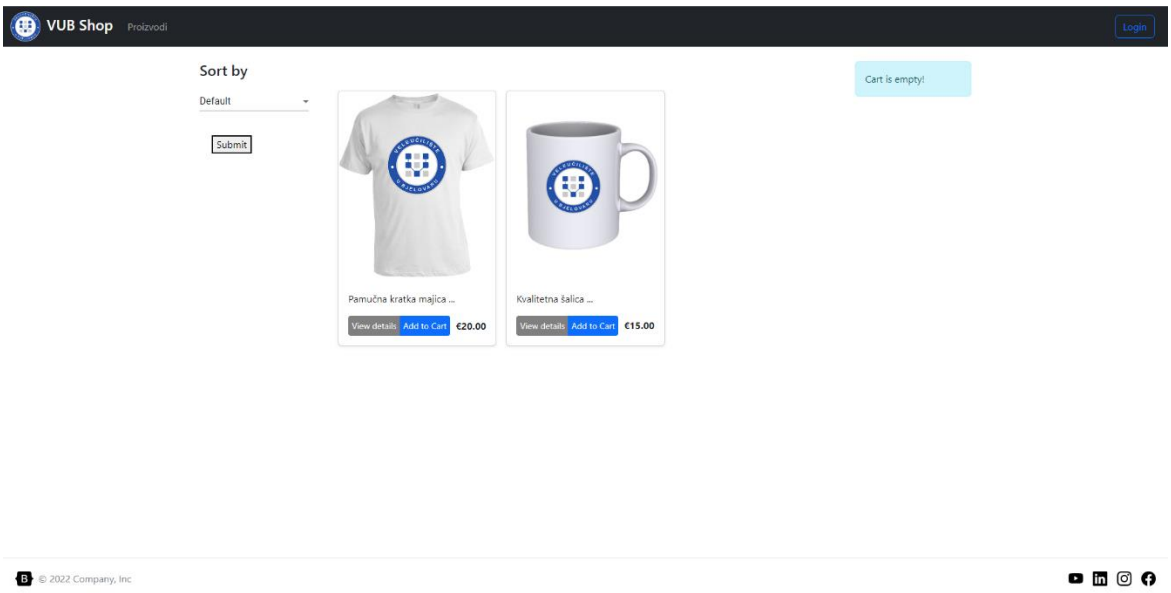
Programski kod 9.2. prikazuje izgled servisa koji se koristi u aplikaciji za dohvaćanje artikala preko *HttpClient* modula i *observable* objekta .

Programski kod 9.2.: Primjer izgleda servisa u aplikaciji

```
import { Injectable } from '@angular/core';
import { Article } from '../models/article';
```

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

const API = 'http://localhost:5211/api/Articles';

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

  constructor(private httpClient: HttpClient) { }

  getArticles(): Observable<Article[]> {
    return this.httpClient.get<Article[]>(API);
  }
}
```

Primjer izgleda korisničkog dijela aplikacije prikazano je slikom 9.5. Klijentski dio aplikacije prikazuje artikle u obliku *Bootstrap Albuma* te ima opcije razvrstavanja prema cijeni i veličini. Na dnu se nalaze poveznice za društvene mreže, a na vrhu tipka za prijavu korisnika.



Slika 9.5. Prikaz klijentskog dijela aplikacije

9.3 Programsko rješenje na strani poslužitelja

Serverski dio aplikacije sastoji se od tri modela, servisa i kontrolera. Modeli sadrže članove koji predstavljaju jedan stupac u bazi podataka za taj model. Svaki model posebna je tablica u bazi podataka. Nakon određivanja modela, unosom naredbe za generiranje koda koji omogućuje pristup bazi podataka stvara se :

- nova datoteka i model koji omogućuju vezu s bazom podataka
- nova varijabla u datoteci *appsettings.json* u kojoj se postavlja adresa i ime baze podataka te korisničko ime i lozinka za prijavu na prethodno spomenutu bazu
- jednostavan kontroler koji upravlja zahtjevima za stvoreni model.

Stvaranjem servisa za prethodno spomenuti model i implementacijom metoda u servisu, omogućuje se stvaranje, čitanje, ažuriranje i brisanje objekata stvorenog modela. Za potrebe ovog rada, servis ima dodatne metode koje služe za filtriranje podataka. Jedna od tih dodatnih metoda omogućuje filtriranje i sortiranje artikala. Dodavanjem servisa u *program.cs* datoteku, omogućuje se korištenje servisa u cijeloj aplikaciji. Također, u spomenutu datoteku *program.cs,* potrebno je dodati *CORS* opciju bez koje pozivi s klijentske aplikacije neće prolaziti.

9.3.1 ***Modeli***

Aplikacija se sastoji od tri modela: artikl, košarica i korisnik. Model artikl predstavlja jedan artikl koji se sastoji od identifikacijskog broja, koji je primarni ključ i mora postojati u svakom modelu, zatim identifikacijskog broja za pojedini artikl, imena, opisa, veličine, cijene, datuma i slike. Model košarica predstavlja artikl koji je dodan u košaricu određenog korisnika i sličan je modelu za artikle, osim što ima strani ključ *UserId* koji se veže na model korisnika. Model korisnik model je koji predstavlja jednog korisnika u aplikaciji, sastoji se od identifikacijskog broja, e-pošte, lozinke i kolekcije artikala dodanih u košaricu. Kolekcija je potrebna jer jedan korisnik može dodati više artikala u košaricu. Primjer modela za artikle opisan je programski kodom 9.3.

Programski kod 9.3.: Primjer modela za artikle

```
public class Article
{
    public int Id { get; set; }
    public int ArticleId { get; set; }
```

```
    public string? Name { get; set; }
    public string? Description { get; set; }
    public string? Size { get; set; }
    public float Price { get; set; }
    public int Quantity { get; set; }
    public DateTime DateAdded { get; set; }
    public string? PictureURL { get; set; }
}
```

9.3.2 ***Servisi***

U aplikaciji se koriste tri servisa, za svaki model jedan servis. Korištenjem servisa omogućeno je stvaranje, dohvaćanje, ažuriranje i brisanje instance modela artikla, košarice i korisnika. Servisi imaju metode:

- *All()* – metoda koja dohvaća sve instance određenog modela
- *Get(id)* – metoda koja prima identifikacijski broj i dohvaća instancu određenog modela s tim identifikacijskim brojem
- *Insert(Model model)* – metoda koja prima Model i instancu određenog modela te stvara objekt tog modela
- *Update(Model model)* – metoda koja prima Model i instancu određenog modela, nakon čega provjerava postoji li taj isti model. Ako postoji, ažurirat će dani model, a u suprotnom će prikazati iznimku.
- *Delete(id)* – metoda koja prima identifikacijski broj određenog modela, poziva metodu *Get(id)* koja će dohvatiti instancu s tim identifikacijskim brojem te zatim obrisati tu instancu.

U servise je moguće dodati i ostale metode, a tako servis za artikle uz ove metode ima još četiri metode koje su potrebne za klijentsku stranu aplikacije. Metode su:

- *GetArticlesAscending()* – metoda koja dohvaća sve artikle razvrstane prema cijeni, od najniže do najviše cijene
- *GetArticlesDescending()* – metoda koja dohvaća sve artikle razvrstane prema cijeni, od najviše do najniže cijene
- *GetArticlesFilteredByPrice(min, max)* – metoda prima dva broja i dohvaća sve artikle u rasponu između ta dva broja
- *GetArticlesFilteredBySize(size)* – metoda koja prima veličinu artikla i dohvaća sve artikle s tom veličinom.

Primjer metoda za dohvaćanje svih artikala iz servisa i izgled servisa, opisano je programskim kodom 9.4.

Programski kod 9.4.: Primjer izgleda servisa

```
public class ArticleService : IArticleService
{
    public ApplicationDbContext DbContext { get; set; }

    public ArticleService(ApplicationDbContext context){
        DbContext = context;
    }
    public async Task<IEnumerable> All(){
        return await DbContext.Article.ToListAsync();
    }

    public async Task<Article?> Get(int? id){
        return await DbContext.Article.FirstOrDefaultAsync(c => c.Id == id);
    }
}
```

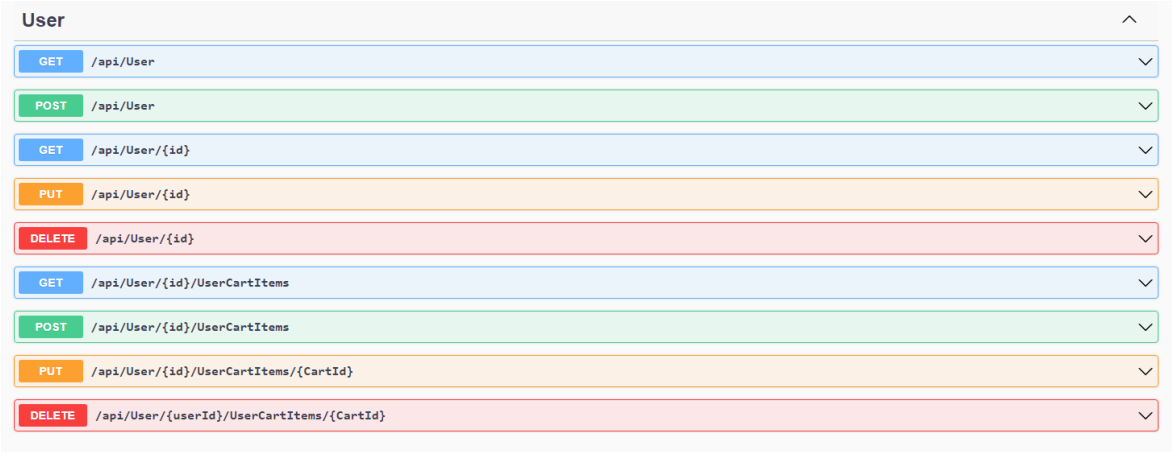
9.3.3 *Kontroleri*

Prethodno objašnjene metode potrebno je implementirati u kontroleru. Prilikom unosa naredbe kojom se omogućuje pristup bazi podataka, stvoren je jednostavan kontroler kojeg treba podesiti. Za sva tri modela moguće je koristiti jedan kontroler no u ovoj aplikaciji, svaki model ima svoj kontroler zbog jednostavnosti i čitljivosti koda. Kontroleri se razlikuju u nazivu ruta kojima se pristupa. Na primjeru kontrolera za artikle, moguće je vidjeti kako funkcionira pozivanje zahtjeva prema kontroleru. U uglatim zagradama potrebno je navesti tip zahtjeva te ako putanja postoji. Putanju je potrebno navesti unutar običnih zagrada i pod znacima navoda. Također, na primjeru kontrolera za korisnike, moguće je vidjeti pozivanje zahtjeva za dohvaćanje košarice nekog korisnika. Programski kod 9.5. prikazuje izgled kontrolera koji koristiti i implementira metode definirane u servisu.

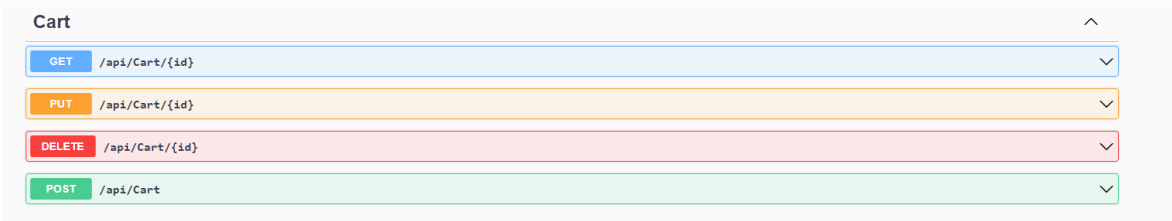
Programski kod 9.5.: Primjer izgleda kontrolera

```
public class ArticleService : IArticleService
{
    public ApplicationDbContext DbContext { get; set; }

    public ArticleService(ApplicationDbContext context){
        DbContext = context;
    }
    public async Task<IEnumerable> All(){
        return await DbContext.Article.ToListAsync();
    }

    public async Task<Article?> Get(int? id){
        return  await  DbContext.Article.FirstOrDefaultAsync(c  =>
c.Id == id);
    }
}
```

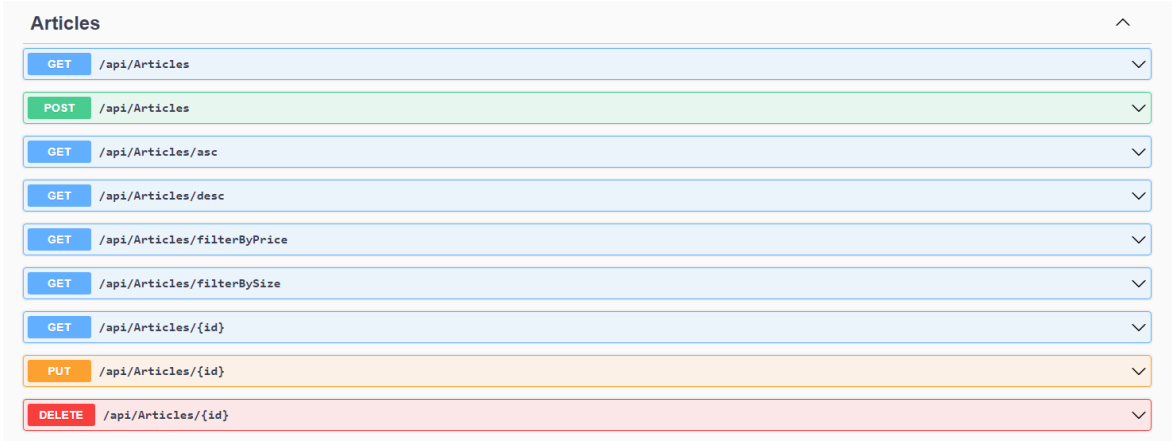
Pomoću *Swagger UI* alata, koji se automatski generira i prikazuje prilikom pokretanja API-a, moguće je vizualizirati sve tablice i pripadne servise. Na slikama 9.6., 9.7. i 9.8. moguće je vidjeti vizualnu dokumentaciju API-a serverske strane aplikacije.



Slika 9.6. Dokumentacija *User* entiteta



Slika 9.7. Dokumentacija *Cart* entiteta



Slika 9.8. Dokumentacija *Article* entiteta

10. ZAKLJUČAK

U ovom završnom radu objašnjene su prednosti, mane i značajke Angular i .NET tehnologija kroz primjere te kroz razvoj web aplikacije. Iščitavanjem literature i prikazom praktičnih primjera dolazi se do zaključka da su obje tehnologije veoma korisne i sadrže velik broj značajki koje aplikaciju čine skalabilnom, responzivnom i stabilnom. Responzivnost i skalabilnost odnose se na mogućnost i dizajn aplikacije da se prilagodi veličini zaslona ekrana uređaja na kojem se pokreće.

Angular je jako korisno i elegantno rješenje za razvoj dinamičnih web aplikacija na jednoj stranici. Razvoj Angular aplikacija temelji se na komponentama koje pojednostavljuju sam izgled korisničkog sučelja i proces razvoja aplikacija te čine kod lakim za održavanje. Komponente su također neovisne jedna o drugoj što donosi mogućnost zasebnog testiranja komponenata. Angular koristi jednostavnu sintaksu i jednostavan je za učenje, ali ima svoja pravila koja se moraju poštovati. Također, u aplikaciju se može uvesti puno korisnih modula koji doprinose funkcionalnosti aplikacija.

.NET razvojni okvir donosi mnogobrojne prednosti programerima i klijentima. Programeri s lakoćom i jednostavnošću razvijaju aplikacije bogate značajkama s intuitivnim korisničkim iskustvom, a klijenti dobivaju potpuno funkcionalne, fleksibilne i dinamične aplikacije. .NET razvojni okvir dolazi s mnoštvom značajki koje su poboljšale razvoj aplikacija. Također, razvojni okvir prilikom razvoja aplikacija koristi moćne alate za rješavanje problema u bilo kojoj vrsti aplikacije, a pritom su jednostavni za korištenje.

11. LITERATURA

[1] Murray N, Coury F, Lerner A, Taborda C. ng-book: The complete Book of Angular: San Francisco, California; 2016. dostupno na: https://www.academia.edu/49151733/Murray_N_ng_book_The_Complete_Book_on_Angular

[2] Price J.M. C# 10 and .NET 6 – Modern Cross -Platform Development: Ujedinjeno Kraljevstvo, 2021.

[3] Web stranice razvojnog okvira Angular, https://angular.io/

[4] Web stranice razvojnog okvira AngularJS, https://angularjs.org/

[5] Web stranice razvojnog okvira .NET, https://docs.microsoft.com/en-us/dotnet/

[6] Web stranica troškova licenciranja uređivača koda Visual Studio, https://visualstudio.microsoft.com/vs/pricing/?tab=business

12. OZNAKE I KRATICE

MVC – Model View Controller
HTML – HyperText Markup Language
CSS – Cascading Stye Sheets
DOM – Document Object Model
SPA – Single Page Application
CLR – Common Language Runtime
LINQ – Language Integrated Query
SQL – Structured Query Language
XML – Extensible Markup Language
NPM – Node Package Manager
CLI – Command Line Interface
CD – Change Directory
DIR – Directory
CORS – Cross Origin Resource Sharing

13. SAŽETAK

Naslov: Web trgovina zasnovana na tehnologijama .NET i Angular

U ovom završnom radu predstavljeni su elementi i arhitektura Angular i .NET tehnologija. Upoznali smo se s prednostima, manama i naprednim elementima te usporedili spomenute tehnologije s ostalim popularnim tehnologijama. Prošli smo kroz stvaranje i pokretanje prve aplikacije, ali i kroz razvoj složenije aplikacije. Složenija aplikacija koristi .NET API za komunikaciju s bazom podataka i Angular za prikazivanje podataka klijentu. Prilikom razvoja aplikacije uočene su glavne prednosti korištenja Angular i .NET tehnologija. .NET API na jednostavan i elegantan način zamjenjuje stotine linija SQL upita pomoću razvojnog okvira entiteta i *LINQ* upita. Angular, također, na jednostavan i elegantan način, obrađuje dohvaćene podatke s poslužitelja u minimalno linija koda. Obje tehnologije posjeduju velik broj biblioteka i modula koje dodavanjem u projekt proširuju funkcionalnost i poboljšavaju korisničko iskustvo kako krajnjim korisnicima, tako i programerima.

Ključne riječi: angular, .NET, web trgovina.

14. ABSTRACT

Title: Web store based on .NET and Angular technologies

In this final paper, the elements and architecture of Angular and .NET technologies are presented. We got familiar with advantages, disadvantages and advanced elements of both technologies and compared already mentioned technologies with other popular technologies. We went through the creation and launch of the first application, but also through the development of a more complex application. A more complex application uses the .NET API to communicate with the database and Angular to display data to the client. When developing the application, the main advantages of using Angular and .NET technologies were observed. The .NET API replaces hundreds of lines of SQL queries using *Entity Framework* and *LINQ* queries in a simple and elegant way. Angular, also in a simple and elegant way, processes the data retrieved from the server in a minimum of lines of code. Both technologies have a lot of libraries and modules that, when added to the project, extend the functionality and improve the user experience, both for end users and developers.

Keywords: angular, .net, web store

IZJAVA O AUTORSTVU ZAVRŠNOG RADA

Pod punom odgovornošću izjavljujem da sam ovaj rad izradio/la samostalno, poštujući načela akademske čestitosti, pravila struke te pravila i norme standardnog hrvatskog jezika. Rad je moje autorsko djelo i svi su preuzeti citati i parafraze u njemu primjereno označeni.

Mjesto i datum	Ime i prezime studenta/ice	Potpis studenta/ice
U Bjelovaru, 8.09.2022.	Ivan Nemet	Nemet

Prema Odluci Veleučilišta u Bjelovaru, a u skladu sa Zakonom o znanstvenoj djelatnosti i visokom obrazovanju, elektroničke inačice završnih radova studenata Veleučilišta u Bjelovaru bit će pohranjene i javno dostupne u internetskoj bazi Nacionalne i sveučilišne knjižnice u Zagrebu. Ukoliko ste suglasni da tekst Vašeg završnog rada u cijelosti bude javno objavljen, molimo Vas da to potvrdite potpisom.

Suglasnost za objavljivanje elektroničke inačice završnog rada u javno dostupnom nacionalnom repozitoriju

Ivan Nemet

ime i prezime studenta/ice

Dajem suglasnost da se radi promicanja otvorenog i slobodnog pristupa znanju i informacijama cjeloviti tekst mojeg završnog rada pohrani u repozitorij Nacionalne i sveučilišne knjižnice u Zagrebu i time učini javno dostupnim.

Svojim potpisom potvrđujem istovjetnost tiskane i elektroničke inačice završnog rada.

U Bjelovaru, 8.09.2022.

Nemet

potpis studenta/ice