Web aplikacija za evidenciju volontera Crvenog križa

Vrsalović, Ivan

Sveučilište u Rijeci – Fakultet informatike i digitalnih tehnologija

Preddiplomski jednopredmetni studij informatike

Ivan Vrsalović

Web aplikacija za evidenciju volontera Crvenog križa

Završni rad

Mentor: Izv. Prof. Dr. sc. Marina Ivašić Kos
Komentor: Andrej Arbanas mag. ing. comp. , Juice d.o.o.

Rijeka, 15.8.2022.

FAKULTET INFORMATIKE I DIGITALNIH TEHNOLOGIJA
SVEUČILIŠTE U RIJECI

Rijeka, 15.04.2022.

Zadatak za završni rad

Pristupnik: **Ivan Vrsalović**

Naziv završnog rada: **Web aplikacija za evidenciju volontera Crvenog križa**

Naziv završnog rada na eng. jeziku: **Web application for the registration of Red Cross volunteers**

Sadržaj zadatka: Proučiti odgovarajuće Java skript softverske alate, biblioteke i baze koji se danas uobičajeno koriste za izradu web aplikacije kao što su Angular i Node.js, te serverske baze kao što je ne relacijska baza podataka MongoDB ili relacijska baza MariaDB. Odabrati domenu primjene i namjenu aplikacije te definirati zahtjeve korisnika koji će se realizirati.

Opisati osnovnu arhitekturu klijent-server sustava te osnovne karakteristike i funkcionalnost web aplikacija koju će se razviti. Predložiti i opisati arhitekturu sustava te alate koji će se koristiti za razvoj frontend i backend dijela.

Napraviti prototip web aplikacije koja omogućava spremanje novih volontera, vježbi i edukacija Crvenog križa, pregled i uređivanje te brisanje postojećih. Prikazati sučelje aplikacije (front-end) i opisati kako korisnici koriste aplikaciju. Unutar aplikacije integrirati funkcije za pretraživanje podataka te za lakše korištenje aplikacije. Opisati ključne korake implementacije sa dijelovima koda i elementima jezika (klase/entiteti, varijable, metode i sl.) koji su korišteni u razvoju.

Mentor

Izv. prof. dr. sc. Marina Ivašić-Kos

Voditelj za završne radove

Doc. dr. sc. Miran Pobar

Zadatak preuzet: datum

(potpis pristupnika)

Sadržaj

Sažetak

U ovom završnom radu opisan je razvoj Angular web aplikacije za evidenciju volontera, vježbi i edukacija Crvenog Križa. Aplikacija je namijenjena za Crveni Križ Primorsko-goranske Županije, i omogućava upis podataka o novim volonterima, vježbama i edukaciji, kao i pregled, uređivanje i brisanje podataka. Unutar aplikacije integrirane su funkcije za pretraživanje podataka te za lakše korištenje aplikacijom.

Ključne riječi: HTML, CSS, TypeScript, Angular, Node.js, server, MongoDb, baza, WebStorm,

Uvod

Prije razvoja softvera i digitalnih metoda pohrana informacija, svaka evidencija se temeljila na čuvanju papira na kojima su bili zapisani podaci. S vremenom papiri su postali nepregledni i spori, te je počela digitalizacija. Digitalizacijom omogućujemo brzi i laki pregled podataka, a i ujedno olakšano prenošenje ili daljinski pristup tim podacima.

Web aplikacija je program koji se izvodi u preglednicima, na razliku od desktop aplikacije koja se može samostalno izvoditi na računalu. U ovom radu opisan je razvoj web aplikacije za evidenciju volontera jer web aplikacija omogućuje jednostavan daljinski pristup evidenciji, što olakšava vođenje evidencije Crvenom križu koji često putuje po vježbama i intervencijama. Svaka web aplikacija funkcionira na principu klijent – server. Klijentu omogućuje korisničko sučelje koje je definirano na serveru, pomoću kojeg korisnik može slati zahtjeve na serversku stranu. Server obrađuje zahtjeve i vraća korisniku željeni rezultat. Razvoj web aplikacije temelji se na korištenju okruženja napravljenih za lakše razvijanje web aplikacija. React.js je najpopularnije okruženje za razvijanje web aplikacija u 2022. sa čak godine 42.62% korištenosti, zatim Angular sa 20.39%, Vue.js sa 18.82% i drugi [1]. Web aplikacija razvijena u ovom radu je razvijena koristeći Angular okruženje.

Angular omogućuje razvoj dinamičkih web aplikacija, sa mogućnošću ponovnog korištenja istih komponenti što u razvoju aplikacije za evidenciju uvelike olakšava proces jer se zahtjevi ponavljaju sa svaku vrstu entiteta, skalabilnost aplikacije koja omogućuje razvijanje web aplikacije neovisno o broju entiteta (volontera) spremljenih u njoj.

Angular aplikacija u ovom radu nastala kako bi digitalizirali evidenciju volontera Crvenog Križ Primorsko-goranske županije, koja je do sada još uvijek bila potpuno u analognom obliku.

Klijent – server aplikacije

Razumijevanje načina rada aplikacije najprije je potrebno shvatiti arhitekturu klijent – server aplikacije. Moderne web aplikacije se grade u dva dijela: klijentski dio (*eng. front-end*) i serverski dio (*eng. back-end*). Klijent – server je model softverske arhitekture razvijen kako bi omogućio komunikacijsku između više računalnih korisnika.

Klijenti su uređaji kojima se koristi korisnik web aplikacije, računala, tableti, mobiteli na kojima se nalazi internetski preglednik pomoću kojeg se oni spajaju na web aplikaciju. Korisnik komunicira sa serverom pomoću HTTP/HTTPS[1] protokola te uvijek započinje zahtjev (Slika 1.), čeka i prima odgovore od servera te njegovo korištenje aplikacije ovisi o serveru. Softver na strani klijenta zovemo klijentskim dijelom aplikacije te se pokreće u internetskom pregledniku. Klijentski dio obuhvaća sve s čime korisnici imaju interakciju. Najpopularniji jezici razvoja klijentske strane su HTML, CSS i JavaScript, zajedno te tehnologije čine prezentacijsku logiku i izgled aplikacije.

Server je uređaj koji je povezan na internet i koriste se za pohranjivanje web stranica, lokacija, aplikacija i datoteka. Kada korisnik otvori stranicu ona se preuzima sa određenog servera. Serveri uglavnom koriste TCP/IP[2], ali i druge kao što su FTP [3]za prijenos podataka, SMTP [4]za slanje pošte. Server uvijek čeka korisnički zahtjev, te klijentu odgovara sa statusnim kodom koji predstavlja uspjeh ili grešku, ovisno o zahtjevu klijenta server može izvršiti razne operacije poput prijenose datoteke, spremanje ili brisanje podataka, slanje pošte (Slika 2.). Softver na strani servera zovemo serverska strana aplikacije, te se pokreće na lokalnom serveru (računalo) ili na internetskom servisu koji omogućuje usluge poslužitelja servera. Najpopularniji jezici serverske strane su JavaScript, Python, PHP te Ruby.

---

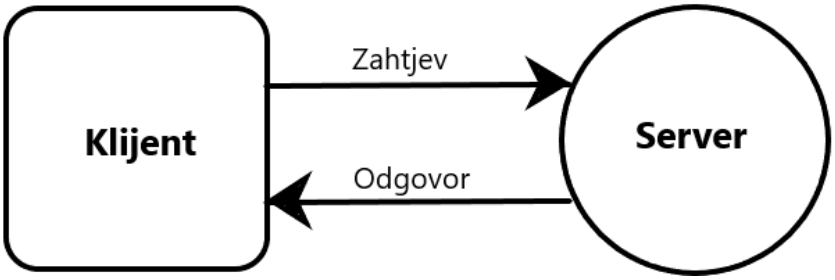
[1] HTTP/HTTPS (eng. *Hypertext Transfer Protocol Secure)* – skup pravila po kojim se web aplikacija prenosi od servera do korisnika

[2] TCP/IP (*eng. Ttransmission Control Protocol / Internet Protocol*) – protokol za komunikaciju između raznih međusobno povezanih mreža

[3] FTP (*eng. File Transfer Protocol*) -protokol za prijenos podataka

[4] SMTP (*eng. Simple Mail Transfer Protocol*) – protokol za prijenos elektroničke pošte

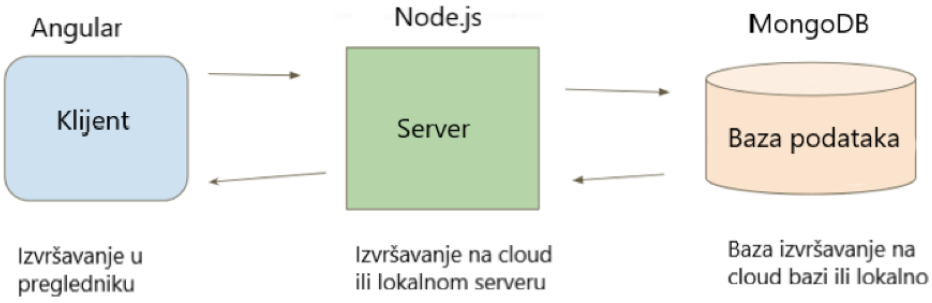
Razlika između klijenta i servera



Slika 1: Klijent - server komunikacija

Tablica 1: Razlika klijenta i servera

Klijent	Server
Uređaj koji zahtjeva uslugu na internetu	Uređaj koji je spojen na Internet i odgovara na zahtjeve klijenta, nudeći im određene usluge
Šalje zahtjev serveru	Čeka korisnički zahtjev te vraća odgovor
Primjer: Osobna računala, pametni telefoni, tableti…	Primjer: Apache server, Amazon Web Services, Google Cloud Platfrom



Slika 2: Arhitektura aplikacije

Korištene tehnologije i alati

WebStorm je text-editor u kojem je razvijen projekt. Omogućava razvijanje JavaScript i TypeScript aplikacija za web. Napravljen od tvrtke JetBrains za Linux, macOS te Windows. Omogućava olakšano otkrivanje grešaka, ispravljanje sintaktičkih pogrešaka, automatsko dovršavanje koda i importa te olakšano povezivanje s Git5-om i GitHubom6.

HTML (*eng. HyperText Markup Language*) je osnovni jezik za izradu web stranica koji preglednicima daje podatke o sadržaju i strukturi učitane web stranice [2]. Elementi HTML-a čine osnovnu za izgradnju web stranice, svakom elementu možemo pristupiti uz njegovu oznaku poput <p> - paragraf ili <img> - slika. Prvu HTML verziju napisao je Tim Berners-Lee 1993. godine, od tad HTML se razvijao kroz mnoge verzije te trenutna verzija je HTML5, koja omogućava pojavu novih oznaka poput <video> i <audio> za videozapise i zvuk, <nav> - za navigacijske izbornike, vektorsku grafiku te mnoge druge [3].

CSS (*eng. Cascading Style Sheets*) – je jezik namijenjen uređivanju stila HTML elemenata, poput pozadina, boja, fontova, animacija, itd. CSS je razvijen kako bi odvojili podatke od dizajna zbog lakše uređivanja i održavanja web aplikacija [4].

TypeScript je besplatan objektno orijentirani programski jezik razvijen od Microsoft-a [5]. Razvijen kao nad skup JavaScript-a kako bi olakšao razvijanje web aplikacija. TypeScript se pri kompilaciji prevodi u JavaScript, te sadrži sve mogućnosti JavaScripta uz dodatne mogućnosti kao što su korištenje klasa, nasljeđivanja, sučelja te provjeru statičkog tipa, što znatno pomaže u sprječavanju problema i čitljivosti koda.

Node.js ili Node je besplatni softver otvorenog koda baziran na JavaScript-u, koji omogućava pokretanje JavaScript koda izvan preglednika, tj. na serverima [6]. Node programeri koriste

[5] Git – sustav namjenjen za distribuirane kontrole verzija softvera
[6] GitHub – web servis namjenjen za distribuirane kontrole verzija softvera

JavaScript kako bi pisali skripte za servere pomoću kojih se stvaraju dinamičke web aplikacije. 2020 godine Node.js je postao najkorišteniji programski okvir, zbog paradigme JavaScript svugdje, koja omogućuje programerima pisanje jednog programerskog jezika za svaki aspekt razvoja web aplikacije [7].

Angular je besplatno okruženje razvijeno od Google-a s namjerom za olakšano razvijanje klijentske strane aplikacije. Temeljen na TypeScript-u, služi kako bi lakše kreirali dinamičke web aplikacije. [8]

MongoDB je NoSQL besplatna baza podataka, razvijena za jednostavnu skalabilnost, brzinu i dostupnost. Kolekcija i dokument su izrazi kojima opisujemo podatke u MongoDB, kolekciju bi mogli smatrati kao jednom tablicom u SQL tipu baze dok dokumente redovima u tablici, ali u MongoDB redovi nisu poput zapisa u SQL-u već su to podaci zapisani u JSON[7] obliku. Baza korištena u ovom projektu obavljena na web servisu MongoDB Atlas koji omogućuje besplatni objavljivanje baze [9].

Instalacija osnovnih alata serverske strane

Node.js

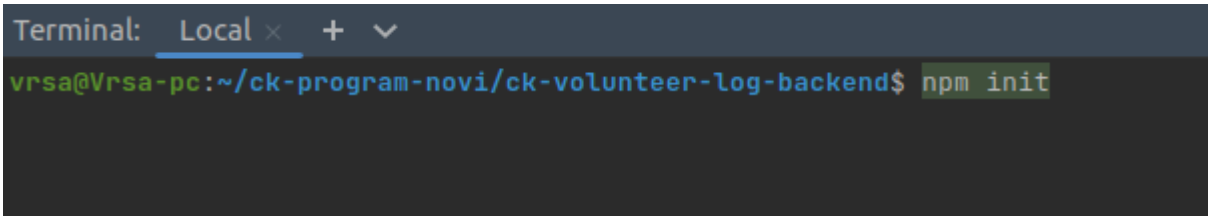
Razvijanje serverskog dijela aplikacije radit ćemo preko Node.js alata pisanjem TypeScript koda koji će se pri izvršenju prevoditi u JavaScript. Prije korištenje Node.js potrebno je preuzeti Node.js server s njihove web stranice, te nakon instalacije on je spreman za korištenje.

Node package manager ili NPM je upravitelj paketa za JavaScript pomoću kojeg se omogućuje instalacija i uređivanje različitih paketa i okruženja za razvoj aplikacije, uz to omogućuje povezivanje sa Git-om i GitHub-om koji olakšavaju on-line razmjenu koda. Mogućnost korištenja NPM ne ovisno o Node.js, paketa tj. iako NPM instaliramo instalirajući

[7] JSON – (*eng. JavaScirpt Object Notation*) – format za razmjenu i pohranu podataka

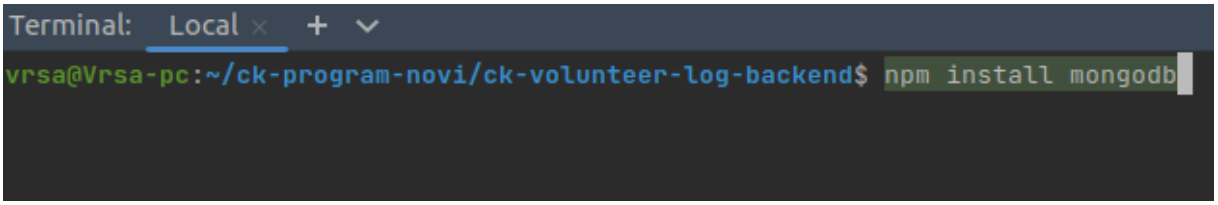
Node.js, NPM nije ovisan o Node.js te se može koristiti i za druge serverske alate, iz toga razloga je najpopularniji upravitelj paketa.

Stvaranje Node.js projekta počinje naredbom npm init, koja kreira datoteku *package.json*, iako ako u WebStorm-u unaprijed odredimo da želimo razvijati novi projekt tipa Node.js, text-editor sam pokreće naredbu prilikom kreiranja novog projekta.



Slika 3 Inicijalizacija projekta

Dodatne pakete ili dodatke za lakše razvijanje aplikacije instaliramo pomoću naredbe npm install ili skraćeno npm i .



Slika 4: Instalacija paketa MongoDb

Moguće je instalirati više od jednog dodatka pomoću naredbe npm install tako da imena paketa odvojimo razmakom, npm i instalira pakete u direktorij node_modules, te ako pozovemo naredbu npm i bez imena paketa, instalirat će se svi paketi čija su imena navedena u package.json datoteci.

Pošto je aplikacija rađena uz pomoć vanjskog mentora iz tvrtke Juice Commerce dobio sam mogućnosti korištenja nekih od njihovih paketa poput @juice/entites koji je znatno olakšali izgradnju modela volontera, omogućavajući već postojeći model na koji su dodani potrebni atributi kako bi model za volontera bio potpun i prilagođen za ovu aplikaciju. Datoteka package.json (Slika 4.) sadrži osnovne podatke aplikacije, pakete i dodatke koji omogućuju lakši razvoj aplikacije. Package.json se mijenja kako instaliramo nove pakete te ih sprema u listu kako bi se izvela lakša instalacija ukoliko se razvoj promijeni na okruženje koje ih nema

instalirano. Definirano je ime projekta, autor te link za povezivanje na GitHub koji omogućuje lako preuzimanje projekta na drugo računalo ili preuzimanje projekta od strane drugog programera.

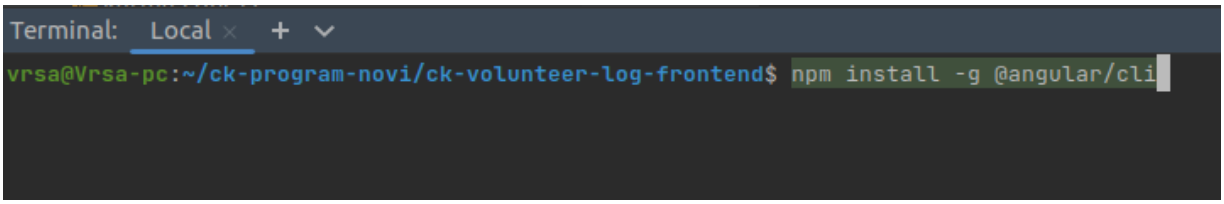

```
{
 "name": "ck-volunteer-log-backend",
 "version": "1.0.0",
 "description": "",
 "main": "bin/serve.js",
 "scripts": {
  "test": "mocha"
 },
 "repository": {
  "type": "git",
  "url": "git+https://github.com/Cpt-Shime/ck-
volunteer-log-backend.git"
 },
 "author": "",
 "license": "ISC",
 "bugs": {
  "url": "https://github.com/Cpt-Shime/ck-
volunteer-log-backend/issues"
 },
 "homepage": "https://github.com/Cpt-Shime/ck-
volunteer-log-backend#readme",
 "devDependencies": {
  "@types/cors": "^2.8.12",
  "@types/express": "^4.17.13",
  "@types/node": "^17.0.16"
 },
 "dependencies": {
  "@juice/entities": "^3.1.4",
  "@juice/juice": "^1.4.18",
  "@juice/networking": "^2.3.3",
  "@juice/users": "^3.5.0",
  "@types/mongodb": "^3.6.20"
 }
}
```


Slika 5: package.json

Instalacija osnovnih alata klijentske strane

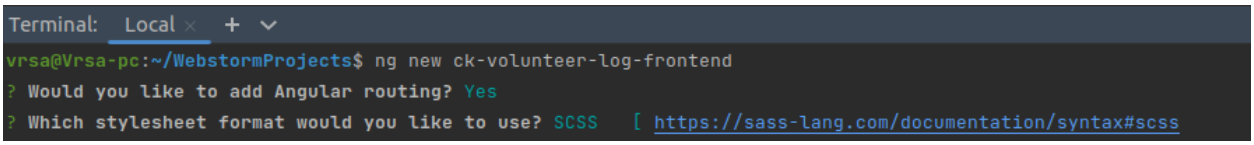
Angular okruženje možemo koristiti tek kada imamo instaliranu verziju Node.js-a tj. NPM upravitelj paketa kako bi mogli instalirati i upravljati s dodatnim funkcijama i mogućnosti Angular-a.

AngularCLI ili Angular command-line interface je alat koji koristimo kako bi stvorili, razvili i održavali Angular aplikaciju pomoću naredbenih redaka. Instalaciju AngularCLI započinjemo pomoću naredbe npm install -g @angular/cli (Slika 5).



Slika 6: Instalacija AngularCLI

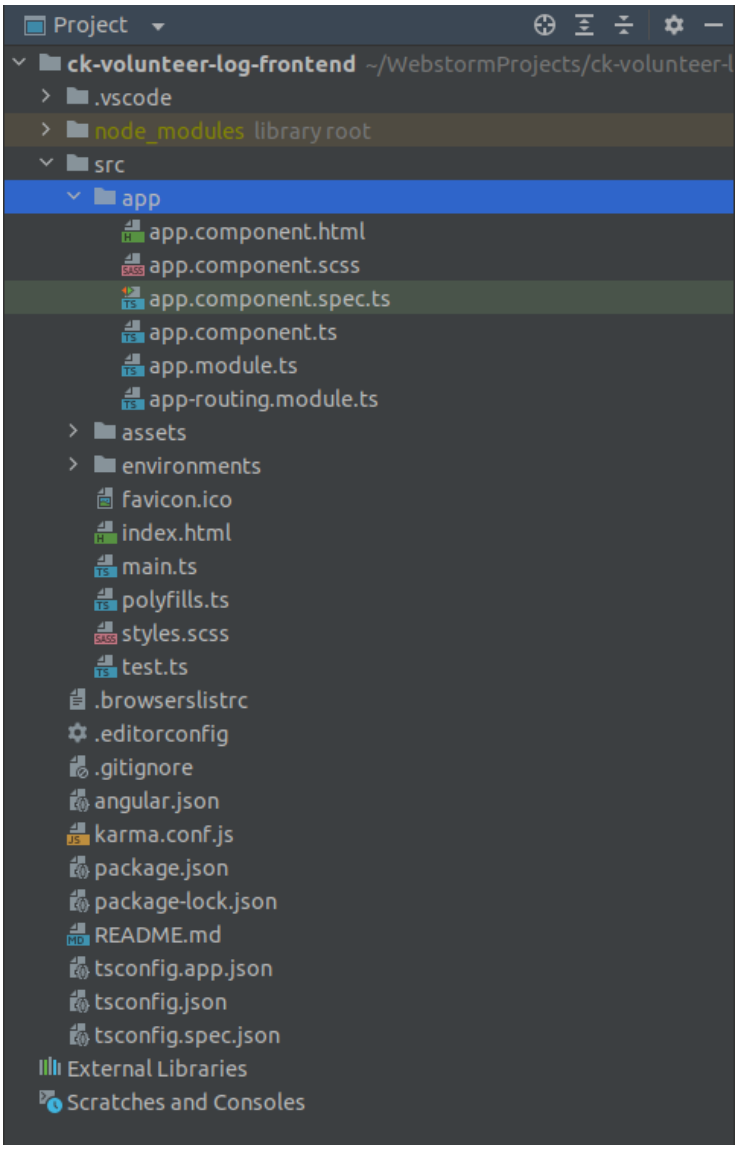
Nakon instalacije AngularCLI dobivamo pristup nizu različitih naredbi poput ng new koja omogućava stvaranje novog projekta (Slika 6).



Slika 7: Stvaranje novog projekta

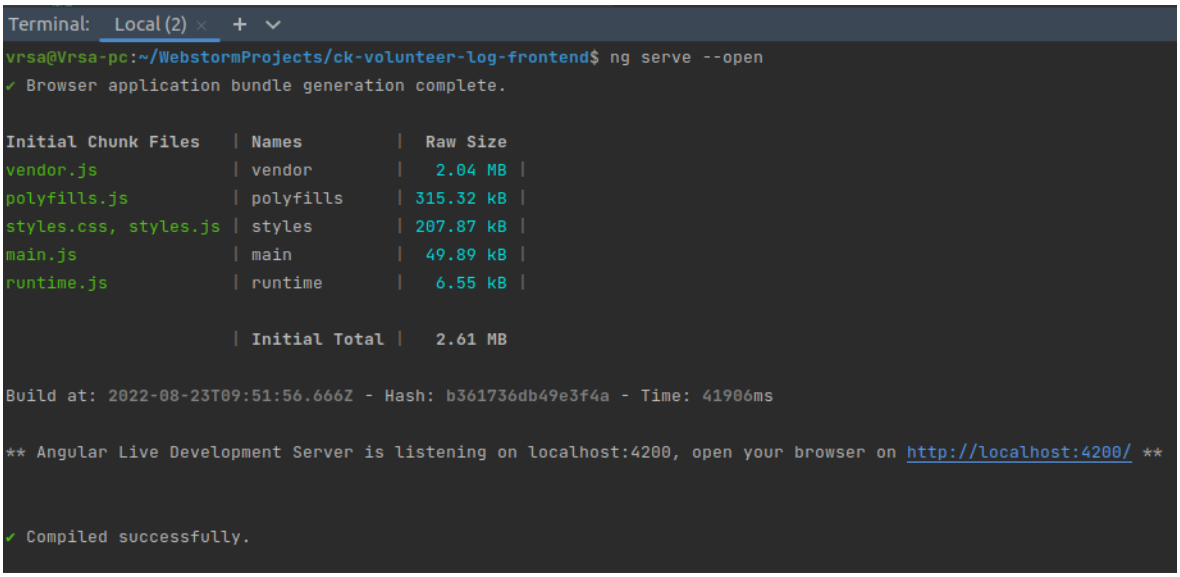
Za vrijeme izvršenja naredbe ponudit će se opcije za dodavanje Angular routing koje omogućavaju lakšu navigaciju unutar aplikacije, te ćemo izabrati koji format stila želimo koristiti.

Nakon izvršenja naredbe ng new, stvara se direktorij (Slika 7.) s imenom koji smo proslijedili naredbi kao argument. U tom direktoriju nalaze se već sve potrebne datoteke za pokretanje Angular aplikacije. Unutra projekta nalaze se više direktorija, node_modules u kojem se spremljene sve biblioteke projekta, src u kojem se nalaze datoteke s uputama za aplikaciju, unutar src direktorija se nalazi direktorij app u kojem ćemo dodavati, izmjenjivati i uređivati komponente aplikacije.



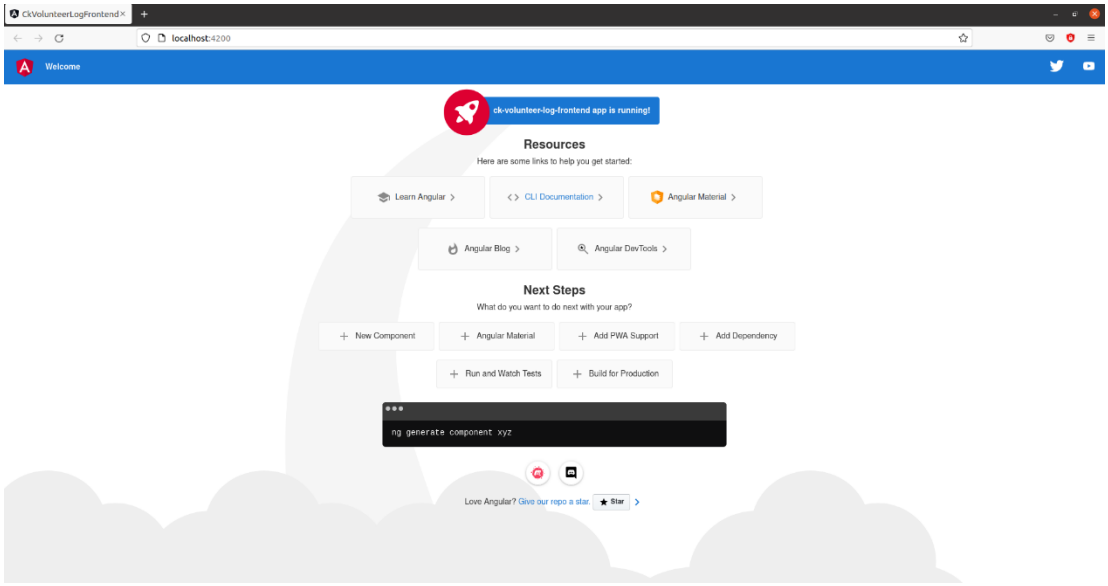
Slika 8: Struktura projekta

Pokretanje aplikacije izvršavamo pomoću naredbe ng serve (Slika 8.) kada se nalazimo u direktoriju projekta ili ng serve --open koja automatski otvara web preglednik s aplikacijom.



Slika 9: Pokretanje projekta

Poslije uspješne kompilacije otvara se zadani web preglednik na lokalnoj adresi localhost:4200 s početnom stranom Angular-a (Slika 9.).



Slika 10: Početna stranica projekta

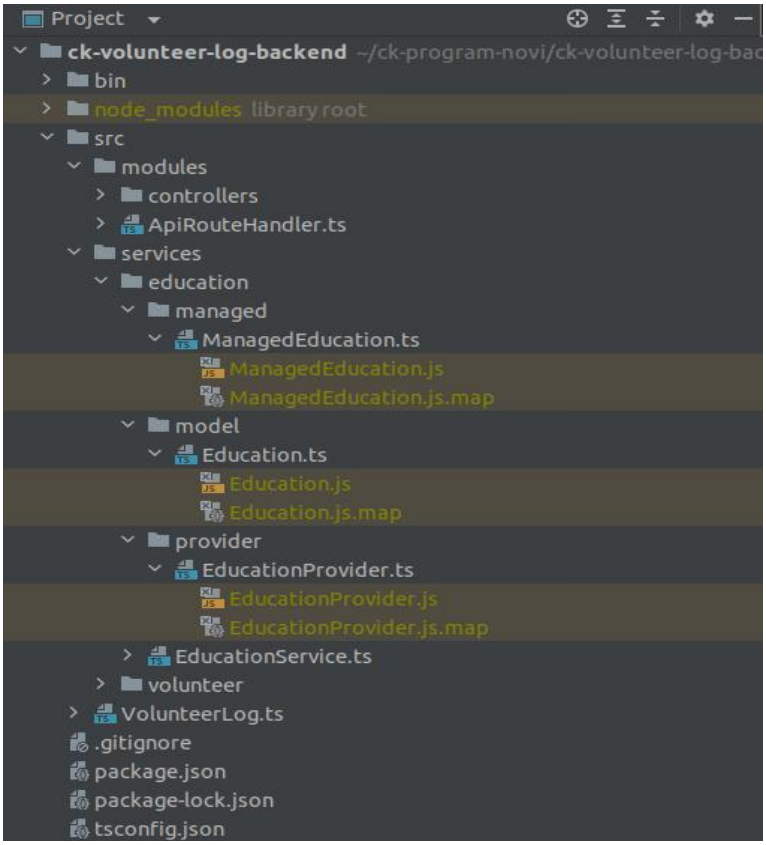
Kreiranje CRUD[8] metoda na serverskoj strani

U ovom radu opisano je kreiranje CRUD metoda za entitet edukacija. Entitet predstavlja nešto o čemu želimo spremati podatke, što se može identificirati ali je u stanju postojati ili ne postojati, može biti stvar događaj ili pojava (edukacija, volonter, vježba) [10]. Edukacija je entitet koji se sastoji od atributa koji je detaljnije opisuju poput: ime edukacije, datum početka, datum završetka, maksimalno sudionika, lokacija. Svaki entitet sadrži svoje atribute te atribute saznajmo iz komunikacije sa korisnikom, proučavanjem dokumenta i prošlih načina evidencije. Svaki entiteti u aplikaciji će proći isti proces, stvaranje po zahtjevu korisnika i spremanje u bazu, prikaz po potrebi korisnika te ukoliko je potrebno ažuriranje i brisanje. Svi dodatni entiteti (volonteri, vježbe) koji su potrebni se razvijaju po istom principu poput entiteta edukacija, uz promjenu ili dodavanje određenih atributa.

Struktura direktorija serverske strane

Direktorije na serverskoj strani ćemo organizirati po direktorijima bin, koji će sadržavati serve.ts skriptu za pokretanje servera, te src direktorija u kojem ćemo napraviti VolunteerLog.ts datoteku te dva direktorija modules i services (Slika 10.) Modules direktorij će sadržavati ApiRouteHandler.ts i direktorij controllers u kojem se nalaze kontroleri za svaku vrstu entiteta. Services direktorij će sadržavati direktoriji education, koji će sadržavati direktorije managed, model i provider te EducationService.ts datoteku. Services direktorij sadrži sve potrebne servise i funkcije koji će biti potrebni kako bi kreirali, ažurirali, obrisali i li pročitali entitet.

[8] CRUD – *(eng. Create read update delete*) stvori, prikaži, ažuriraj, obriši



Slika 11: Struktura direktorija serverske strane

Razvoj CRUD-a

Početak razvijanja serverske strane aplikacije počine sa skriptom koja pokreće aplikaciju. Unutar bin direktorija serverske strane kreiramo TypeScirpt datoteku serve.ts (Slika 11.) u kojoj inicijaliziramo serversku stranu, povežemo bazu te pokrenemo skriptu koja učitava servise.

Iz slike vidimo da nakon povezivanja s bazom pokreće se VolunteerLog.

```
import { Juice } from "@juice/juice/Juice";
import { VolunteerLog } from "../src/VolunteerLog";

process.on('unhandledRejection', (reason, p) => {
   console.error(reason);
   console.error(p);
});

process.on('uncaughtException', (e) => {
   console.error(e);

});
//EmrcGTjmvpXsj6C  sifra za cluster
const db = { connectionString: "mongodb+srv://Cpt_Shime:EmrcGTjmvpXsj6C@ck-
cluster.jk18p.mongodb.net/?retryWrites=true&w=majority" };
Juice.init().connect(db).start(VolunteerLog).then(res => {

});
```

Slika 12: serve.ts

VolunteerLog (Slika 12.) je TypeScript datoteka u kojoj smo opisali konfiguraciju projekta, instaliramo i registriramo potrebne module, vanjske i lokalne servise. Servisi nam omogućavaju lakše razvijanje softvera. ColoredConsoleLogger omogućava lakšu preglednost i čitljivost u konzoli, do Networking servisa koji omogućava osnovne internetske funkcije i pozive.

```
@ApplicationConfiguration({
  key: "ck-volunteer",
  options: {
    logger: "logger:colored-console"
  }
})
export class VolunteerLog implements IJuiceApplication {
  options?: any;

  async configure(): Promise<boolean> {
    return true;
  }
  async prepare() {
    Juice.install(LanguageModule);
    Juice.register(ColoredConsoleLogger);
    //Juice services
    Juice.install(Networking);
    //Local services
    Juice.install(VolunteerService);
    Juice.install(EducationService);
    return true;
  }
  async ready(): Promise<any> {
    const networking = Juice.service<Networking>("networking");
    await networking.deployRoute("/api", new ApiRouteHandler());
    const log = Juice.service<Logging>("juice:logging");
    log.info("CK VOLUNTEER", "App ready");
```

Slika 13: VolunteerLog

Završetkom instalacije servisa i modula, kreira se novi ApiRouteHandler koji upravlja s API[9] pozivima aplikacije.

[9] API – (*eng. Aplication programming interface*) aplikacijsko programsko sučelje

ApiRouteHandler (Slika 13.) je TypeScript datoteka u kojoj dodajemo kontrolere ovisno za koji tip entiteta želimo obraditi. „/educations" je dio URL[10] na kojem će se obavljati API pozivi vezani za entitet edukacije, u nastavak tog URL-a pozivamo EducationController();

```
export class ApiRouteHandler implements IRouteHandler {

  router: ExpressRouter;
  options: any;
  constructor() {
    this.router = new ExpressRouter();
    this.router.use(cors());
  }
  async init(): Promise<boolean> {
    this.router.addController("/volunteer", new VolunteerController());
    this.router.addController("/education", new EducationController());
    return true;
  }

  getRouter(): ExpressRouter {
    return this.router;
  }
  setOptions(options) {
    this.options = options;
  }
}
```

Slika 14: ApiRouteHandler

EducationController (Slika 14.) je TypeScript datoteka koja se bavi kontroliranjem zahtjeva korisnika koji se poslani putem klijentske strane aplikacije.

Zahtjevi poput stvori, dosegni, obriši, objavi i stavi (*eng. create, fetch, delete, post, put*) koje kontroler mora provjeriti i preusmjeriti na EducationService koji prima zahtjeve od EducationControllera i odrađuje zahtjeve te vraća rezultate EducationControlleru. Sa slike vidimo da svaka od metoda zahtjeva zahtjev od korisnika koji prima kao argument u pozivu funkcije, te obećava odgovor koji vidimo kao povratni rezultat nakon izvršenja funkcije. Ako određeni uvjeti nisu zadovoljeni, nedostatak podataka, ne točan ID objekta, odgovor će biti neuspjeh, a ako su uvjeti zadovoljeni onda se vraća rezultat. Podaci u rezultatu ovise koja se funkcija pozvala, ali svaki rezultat dobiva svoju vrijednost iz jedne od funkcija u EducationService.ts datoteci.

---

[10] URL – (*eng. Unifrom Resource Locator*) jednoznačna adresa neke internet stranice

```
export class EducationController extends ExpressController {

  @Inject("ck-volunteer:educations")
  private educations: EducationService;

  @Get("/:id")
  async fetchById(req: express.Request, res: express.Response): Promise<express.Response> {
    if (!ObjectId.isValid(req.params.id))
      return res.send({ success: false });

    const result = await this.educations.fetchById(new ObjectId(req.params.id));
    return res.send(result);
  }

  @Post("/")
  async create(req: express.Request, res: express.Response): Promise<express.Response> {
    if (!req.body)
      return res.send({ success: false });

    //Validate mandatory data
    const result = await this.educations.create(req.body);
    return res.send(result);
  }


  @Delete("/:id")
  async deleteEducation(req: express.Request, res: express.Response): Promise<express.Response> {
    if (!ObjectId.isValid(req.params.id))
      return res.send({ success: false, error: "INVALID_DATA" });

    const result = await this.educations.delete(new ObjectId(req.params.id));
    return res.send(result);
  }
```

Slika 15. Education Controller

EducationService.ts sadrži sve metode koje su korisnike potrebne za CRUD metode, stvori, pročitaj, ažuriraj i pobriši. Kako bi se te funkcije mogle izvršavati potrebno je napraviti par dodatnih datoteka koje definiraju entitet edukacija, poput modela edukacije, kontrolirane (*eng. managed*) edukacije te davatelj usluga *(eng. provider*) za edukaciju.

Education.ts (Slika 15.) definira model edukacija, određuje koje podatke i koje vrste će biti podaci spremljeni u edukaciju. Svaki od ovih atributa možemo zamisliti kao atribut u tablici te će se jednako tako spremati u bazu podataka, osim podataka prikazanih na slici svakom entitetu

u MongoDB automatski se dodjeljuje jedinstveni ID koji djeluje kao primarni ključ tog entiteta. ID se ne kreira kao atribut nad modelom već pri stvaranju bilo kakvog entiteta MongoDB automatski dodjeljuje jedinstveni ID.

```
@Collection("educations")
export class Education extends Model {

   name: string;
   date_from: Date;
   date_to: Date;
   location: string;
   start_time?: string; //09:00
   type?: string;
   attributes?: any;
   maximum_participants?: number;
   description?: string;
}
```

Slika 16:Model edukacije

ManagedEducation.ts (Slika 16.) omogućuje spremanje edukacija u obliku kontrolirane (*eng. managed)* Edukacije, što nam omogućuje spremanje edukacije kao jedan objekt u memoriji što znatno olakšava ažuriranje podataka pomoću zadanih funkcija.

```
export class ManagedEducation extends Managed<Education> {

   constructor(education?: Education) {
      super(Education);

      this.model = education || new Education();
   }

   async updateEducation(): Promise<Result<Education>> {

      return new Result<Education>();
   }
}
```

Slika 17: Managed edukacija

EducationProvider.ts (Slika 17.) je datoteka koja nam omogućuje da vršimo dodatne radnje nad entitetom. Dosegni po ID (*eng. fetch by ID*) nam omogućava da dosegnemo jedan zapis edukacije koristeći njezin jedinstveni ID u bazi, a funkcija nam vraća podatke spremljene u toj kontroliranoj *(eng. managed*) edukaciji sa zadanim ID-om.

```
export class EducationProvider extends AProvider<ManagedEducation, Education> {


  async fetchById(id, options?: any): Promise<ManagedEducation> {
    const fields: any = {};
    if (options?.fields) {
      options.fields.forEach(f => {
        fields[f] = true;
      });
    }

    let education: any = await Education.findOne<Education>({ _id: id }, { projection: fields });
    if (!education)
      return null;

    education = new ManagedEducation(education);
    return education;
  }
}
```

Slika 18: Provider

FetchById je jedna od najkorištenijih funkcija u ovom radu. Korištena je prilikom ažuriranje podataka i brisanje entiteta, pomoću nje dobivamo podatke o pojedinačnom entitetu kojem unaprijed znamo ID te nam omogućava lakše izvršavanje CRUD metoda nad entitetom.

EducationService.ts sada ima potrebne funkcije koje mu omogućuju da funkcije za stvaranje, ažuriranje, čitanje i brisanje edukacije se izvršavaju uspješno

Funkcija stvori *(eng. create)* (Slika 18.) pri pozovi stvara novu kontroliranu (*eng. managed*) edukaciju te u već napravljeni model sprema podatke dobivene od korisnika. Funkcija čeka da se edukacija spremi i onda vraća ID edukacije kako bi bili sigurni da se edukacija spremila u bazu.

```
async create(data: any): Promise<Result<ObjectId>> {
  const mEducation = new ManagedEducation();

  mEducation.model.name = data.name;
  mEducation.model.date_from = data.date_from;
  mEducation.model.date_to = data.date_to;
  mEducation.model.location = data.location;
  mEducation.model.start_time = data.start_time;
  mEducation.model.maximum_participants = data.maximum_participants;
  mEducation.model.description = data.description;

  await mEducation.save();

  return new Result<ObjectId>(mEducation._id);
}
```

Slika 19: Skripta za stvaranje

Funkcija ažuriraj *(eng. update)* (Slika 19.) pri pozivu dobiva ID edukacije koje želimo ažurirati i nove podatke s kojima želimo zamijeniti stare, te pomoću funkcije FetchbyId dohvaća podatke edukacije s tim ID-om. Provjerava se da li postoji edukacija s tim ID u bazi, ako ne postoji funkcija vraća grešku, ako postoji poziva se update funkcija koja ažurira stare podatke novima.

```
async update(_id: ObjectId, data: any): Promise<Result<Education>> {
  const Education = await this.provider.fetchById(_id);
  if (!Education || !Education._id)
    return new Result(false, "MISSING_ENTITY");

  const result = await Education.update(data);
  if (!result.success)
    return new Result(false);


  return new Result();
}
```

Slika 20: Skripta za ažuriranje

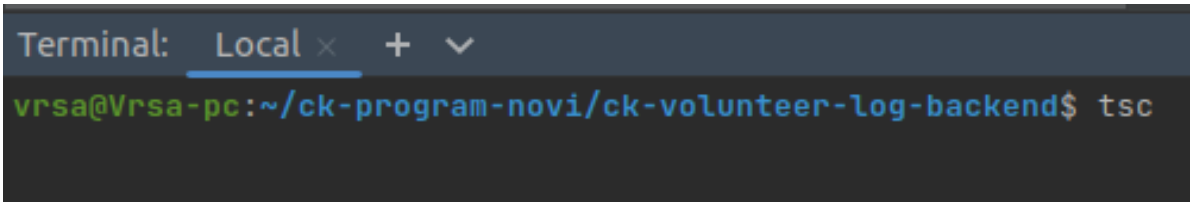
Funkcija obriši (*eng. delete*) (Slika 20.) pri pozivu dobiva ID edukacije koje želimo obrisati, zatim pomoću funkcije fetchByID dohvaća podatke te edukacije. Provjerava da li ta edukacija postoji, ako ne postoji vraća grešku, a ako postoji poziva se funkcija za brisanje zadane edukacije.

```
async delete(_id: ObjectId): Promise<Result> {
  const Education = await this.provider.fetchById(_id);
  if (!Education || !Education._id)
    return new Result(false, "MISSING_ENTITY");

  const result = await Education.delete();
  if (!result)
    return new Result(false);

  return new Result();
}
```

Slika 21. Skripta za brisanje

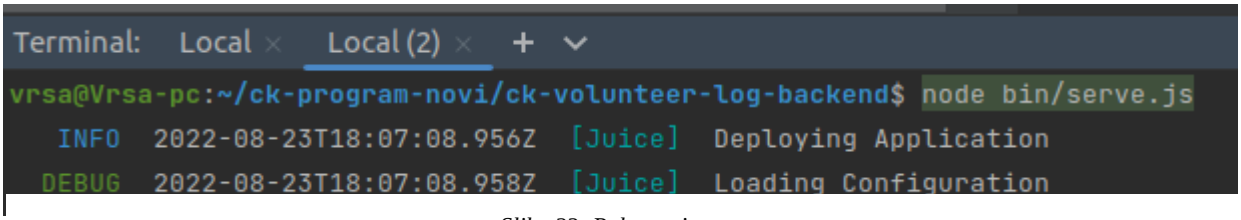
Pokretanje servera

Prije pokretanja servera potrebno je pokrenuti naredbu tsc (TypeScript compiler) (Slika 21.) koji prevodi TypeScript datoteke u JavaScript.



Slika 22: Prevođenje TypeScript u JavaScript

Zatim u drugom terminalu pokrećemo naredbu node bin/serve.js (Slika 22.) kako bi pokrenuli server.



Slika 23: Pokretanje servera

Kreiranje klijentske strane

Kako bi razvili klijentsku stranu aplikacije pomoću Angular-a, moramo prvo razumjeti kako izgleda struktura Angular aplikacija. Angular okruženje prati princip arhitekture temeljenje na komponentama i načelo SPA (*eng. single page application*), aplikacija na jednoj stranici. SPA aplikacije stvaraju osjećaj da korisnik je stalno na jednoj web stranici iako se zapravo aplikacija mijenja i otvaraju nove stranice. Takav osjećaj se ostvaraje s dinamičkim promjenama sadržaja na aplikacije tako da se jedan sadržaj zatvori, a drugi prikaže, stvarajući iluziju promjene stranice. Iako se URL mijenja, sadržaj se mijenja ali nema ponovnog učitavanja stranice. Takva aplikacija zahtjeva kompleksnu klijentsku stranu koja komunicira sa API-jem serverske strane [11].

Arhitektura temeljena na komponentama omogućuje rastavljanje velike aplikacije na mnogo manjih funkcionalnih i logičkih cjelina koje omogućavaju lakše razvijanje i održavanje te „recikliranje" komponenti kroz aplikaciju. Komponente prate logiku roditelj – dijete (*eng. parent-child*) odnos pomoću kojih „dijete" komponenta nasljeđuje atribute i funkcije od „roditelja" komponente.

Struktura Angular aplikacije

Osnovne gradivne jedinice Angular-a su: Moduli (*eng. modules*), komponente (*eng. components*), usluge (*eng. services*) i predlošci (*eng. templates*) (Slika 23.).

Moduli su grupa gradivnih jedinica kao što su komponente, usluge i predlošci koji zajedno omogućuju izgradnju jedne funkcionalne jedinice. Grupa modula zajedno gradi cijelu Angular Aplikaciju.

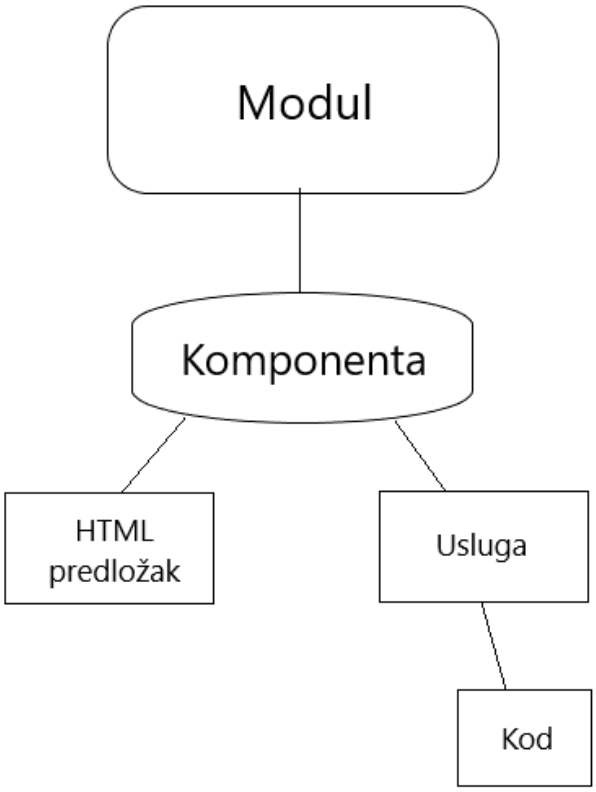
Komponente su osnovne jedinice UI-a [11], lako se „recikliraju" kroz aplikaciju te se sastoje od HTML datoteke, CSS datoteke za uređivanje stila komponente te TypeScript komponente koja omogućuje ugradnju složenih funkcija, i dinamički prikaz podataka.

---

[11] UI – *(eng. User Interface)* korisničko sučelje

Usluge su klase koje omogućuje neko svojstvo nad komponentom (kod). Odvojene su od komponenti kako bi se mogle ponovo koristiti za druge komponente gdje nam je potrebna ta usluga.

Predlošci su HTML datoteke koje uz pomoć klase komponente stvaraju korisnički pogled na aplikaciju.



Slika 24: Gradivne jedinice Angular aplikacije

Kreiranje navigacije bočne trake

Navigacija skroz aplikaciju omogućena je sa glavnom navigacijskom bočnom trakom. Kako bi aplikacija bila pregledna i za korisnike s na mobilnim uređajima navigacijsku traku moramo isprogramirati kako bi se zatvorila ukoliko je širina ekrana manja od zadovoljavajuće.

Kako bi to omogućili potrebna je pozvati promatrača (*eng. observer*[12]) na atribut maksimalna širina (*eng. max-width*) kako bi kontrolirali širinu zaslona, te ukoliko je širina manja od 900 piksela, bočnu navigacijsku traku ćemo zatvoriti i staviti u način rada 'preko' (*eng. over*), te ako je širina se povećala i postala veća od 900 piksela napraviti ćemo obratno i otvoriti bočnu navigacijsku traku i staviti je u način rada 'bočno' (*eng. side*) (Slika 24.).

```
@ViewChild(MatSidenav) sidenav!: MatSidenav;
constructor(private observer: BreakpointObserver) {
}
ngAfterViewInit() {
 this.observer.observe(['(max-width: 900px)']).subscribe((res)=> {
  if(res.matches){
    this.sidenav.mode = 'over';
    this.sidenav.close();
  }
  else {
    this.sidenav.mode = 'side';
    this.sidenav.open();
  }
 });
}
```

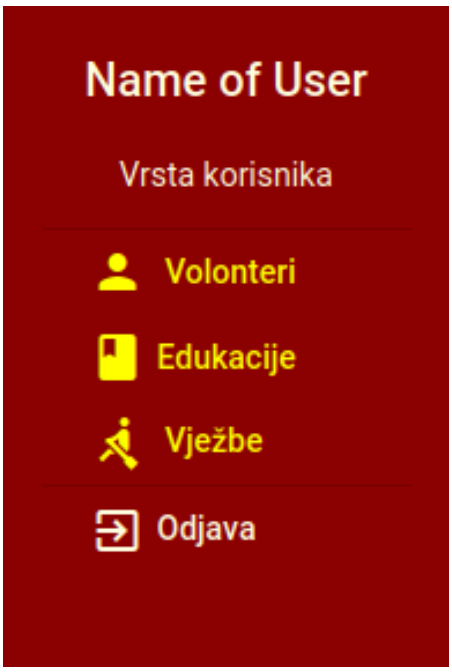
Slika 25: Logika navigacijske trake

Nakon stvaranja TypeScript skripte za navigacijsku traku moramo skriptu povezati sa HTML datotekom (Slika 25.) u kojoj smo kreirali navigacijsku traku i njezine gumbove (Slika 26).

[12] Observer – (*eng. Design pattern*) uzorak dizajna koji omogućuje praćenje nekog objekta i njegove promjene i događaje

```
<mat-toolbar>
  <button mat-icon-button *ngIf = "sidenav.mode === 'over'" (click) ="sidenav.toggle()"
>
    <mat-icon *ngIf = "!sidenav.opened">
      menu
    </mat-icon>
    <mat-icon *ngIf = "sidenav.opened"> </mat-icon>
  </button>
  <a routerLink="volunteers" style = "color:white;">  Evidencija volontera Crvenog
Križa Primorsko-goranske županije</a>
</mat-toolbar>
<mat-sidenav-container>
  <mat-sidenav #sidenav="matSidenav">

    <h2> Name of User</h2>
    <p> Vrsta korisnika</p>

    <mat-divider> </mat-divider>
    <a routerLink="volunteers" >
    <button mat-button class="menu-button" >
      <mat-icon>person</mat-icon>
      <span> Volonteri </span>
    </button>
    </a>
```

Slika 26: HTML navigacijske trake



Slika 27. Izgled navigacijske trake

Kreiranje modula edukacija

Slika struktura modula edukacije nam prikazuje različite komponente koje ćemo koristiti kako bi kreirali CRUD metode na klijentskoj strani. Svaka od metoda će se izvršavati preko vlastite komponente, a stvaranje, ažuriranje i brisanje će biti izvršeno preko modala. Modal je prozor koji se otvara preko glavne aplikacije i nudi korisniku upis, izbor ili potvrdu neke radnje.

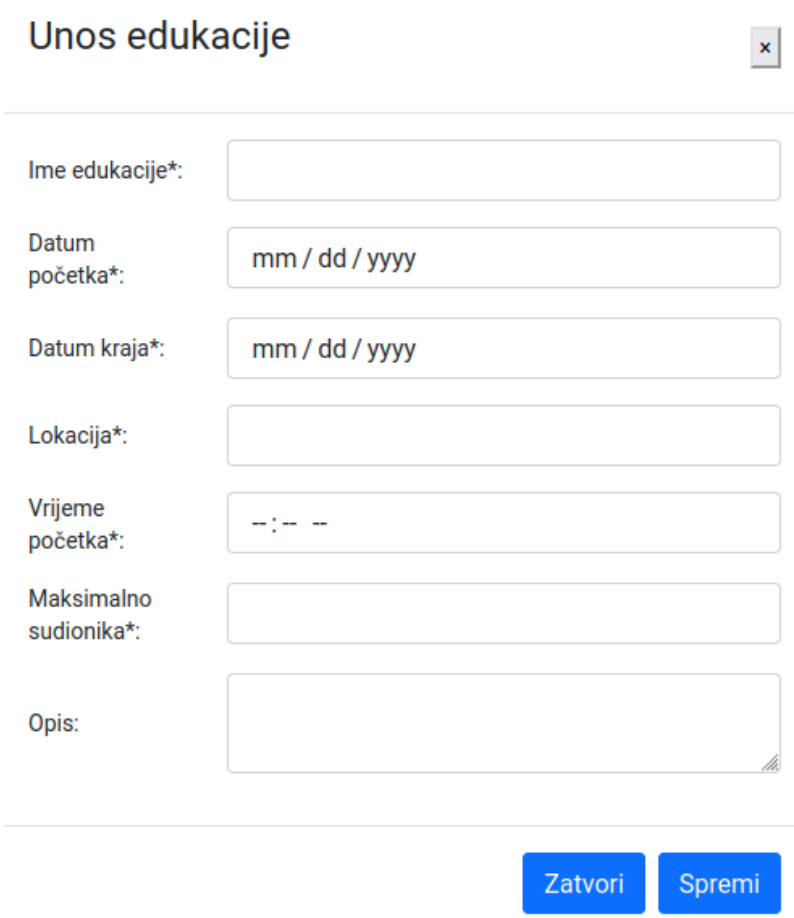
Stvaranje entiteta edukacije

Za stvaranje entiteta edukacije od korisnika moramo dobiti podatke o pojedinoj edukacije. Tip podataka koje korisnik upisuje je već definiran na serverskoj strani, zato na klijentskog strani korisniku moramo omogući upis tih podataka. Unos podataka se izvodi koristeći form group *(eng. grupa obrazaca*), Angular metodu koja je osnovni način prikupljanja podataka od korisnika sa mogućnosću kontrole unose. U TypeScript datoteci (Slika 27.) kreirat ćemo form group sa varijablama koje će korisnik upisivati vrijednosti, te kako bi korisnik mogao upisivati te vrijednost u HTML datoteci (Slika 28.) moramo omogući ulaz podataka pomoću ulazne (*eng. input*) oznake. Svaku input oznaku ćemo povezati sa njoj pripadajućom varijablom u form groupu. Nakon upisivanja svih željenih podataka korisnik pritiskom na gumb spremi, poziva funkciju spremi (*eng. save*), koja dobivene podatke sprema u jednu varijablu koja sadrži sve unesene podatke te šalje ih preko create metode na serversku stranu, koja obrađuje podatke i sprema ih u bazu. Ukoliko je spremanje bilo uspješno modal se zatvara i korisniku se javlja toastr[13] poruka s kojom potvrđuje uspješno spremanje.

[13] Toastr – biblioteka u Angular-u koje omogućuje prikazivanje jednostavnih poruka

```
export class EducationModalComponent implements OnInit {

  createForm: UntypedFormGroup = new UntypedFormGroup({
    name: new UntypedFormControl("",Validators.nullValidator),
    date_from: new UntypedFormControl("",Validators.nullValidator ),
    date_to: new UntypedFormControl("",Validators.nullValidator ),
    location: new UntypedFormControl("",Validators.nullValidator ),
    start_time: new UntypedFormControl("",Validators.nullValidator),
    maximum_participants: new UntypedFormControl("",Validators.nullValidator),
    description: new UntypedFormControl("",Validators.nullValidator),
  });

  promiseBtn: any;
  constructor(private aModal: NgbActiveModal,
          private educationsService: EducationsService,
          private toastr: ToastrService) {
  }
  ngOnInit(): void {
  }
  close() {
    this.aModal.close({ success: false });
  }
  save() {
    this.promiseBtn = (async () => {
      const data = this.createForm.value;
      const result = await this.educationsService.create(data);
      if (!result.success) {
        this.failedToastr();
        return;
      }
      this.savedToastr();
      this.aModal.close({ success: true });
    })()
  }
  savedToastr() {
    this.toastr.success("Edukacija spremljena ", 'Uspjeh!');
  }

  failedToastr() {
    this.toastr.error(" Neuspješno spremanje", 'Greška!');
  }



}
```

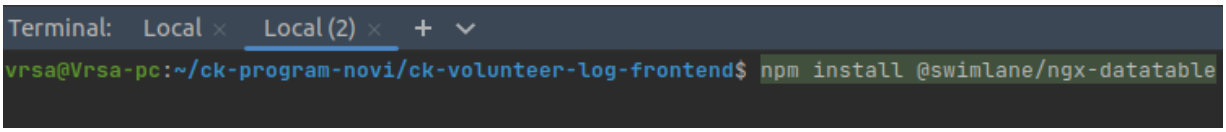
Slika 28: Skripta stvaranje edukacije



Slika 29: Izgled obrasca za stvaranje edukacije

Prikaz entiteta edukacije

Prikazivanje entiteta će biti prikazano u tabličnom obliku. Ngx-datatable je Angular komponenta koja omogućava prezentiranje velike količine podataka u obliku tablice. Sadrži sve mogućnosti normalne tablice te se lako modificira po potrebama ovisno o podacima. [12]. Kako bi instalirali ngx-datatable ponovno nam je potrebna naredba NPM, te pomoću nje instaliramo paket (Slika 29).



Slika 30: Instalacija ngx-datatable-a

Korištenje ngx-datatable je jednostavno, sada možemo kreirati tablicu pomoću HTML oznake <ngx-datatable> u kojoj ćemo definirati što sve želimo prikazati i kakav stil tablice želimo (Slika 30.).

```
<ngx-datatable
      class="material striped"
      [rowHeight]="'auto'"
      [headerHeight]="50"
      (activate)='onSelect($event)'
      [rows]="rows"
>
```

Slika 31: Postavke tablice

Oznaka <ngx-datatable-column> unutar oznake <ngx-datatable> predstavlja jedan stupac u tablici te kroz tu oznaku definiramo koje će biti naslov stupca tablice i koji atribut entiteta želimo prikazivati (Slika 31.).

```
<ngx-datatable-column name="Ime edukacije">
  <ng-template ngx-datatable-cell-template let-value="value" let-row="row">
    {{ row.name }}
  </ng-template>
`
```

Slika 32: Prikaz stupca tablice

Osim atributa edukacije u tablici ćemo prikazati i dva gumba, gumb za ažuriranje edukacije koji će otvoriti modal za ažuriranje te gumb za brisanje edukacije koji će otvoriti modal za brisanje edukacije (Slika 32.).

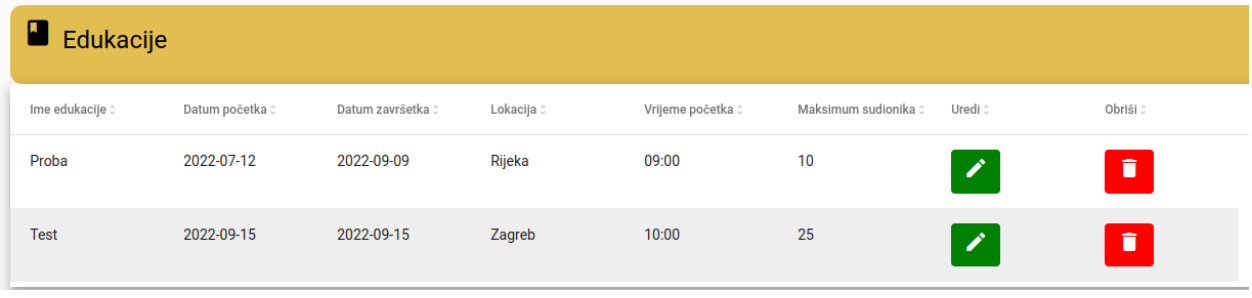
```
<ngx-datatable-column name="Uredi" prop="name">
  <ng-template ngx-datatable-cell-template let-rowIndex="rowIndex"
        let-row="row">

    <button type="button" class="btn btn-success" style="background-color:green; border: green;"
        (click)="openEducationEdit(row);$event.stopPropagation()">
      <mat-icon>edit</mat-icon>
    </button>

  </ng-template>
</ngx-datatable-column>
```

Slika 33: Gumb za ažuriranje edukacije

Kako bi ngx-datatable imao podatke za prikazivati u TypeScript datoteci moramo dosegnuti (*eng. fetch*) podatke o entitetima edukacije koji se nalaze u bazi. Pri pokretanju stranice na kojoj želimo prikazati već postojeće edukacije, pokrećemo i skriptu (Slika 33.) koja dobiva podatke sa serverske strane i sprema ih u rows tipa edukacije, te pomoću varijable rows pristupamo atributima edukacije te ih tako prikazujem po stupcima pomoću <ngx-datatable-column> (Slika 34.)

```
rows: Education[] = [];


constructor(private educations: EducationsService,
        private modal: NgbModal,
        private router: Router) {
}

async ngOnInit(): Promise<void> {
  await this.fetchEducations();
}

 async fetchEducations() {
  const result = await this.educations.fetch(this.page, this.pageSize,
this.options);
  if (!result.success)
    return;

  this.rows = result.payload.items;
}
```

Slika 34: Skripta za dohavaćanje podataka edukacija



Slika 35: Izgleda tablice edukacija

Kreiranje ažuriranja edukacije

Ažuriranje edukacije nam omogućava da već unesene podatke u bazu, zamijenimo novim podacima. Kako bi to omogućili prvo moramo učitati već postojeće podatke te zatim korisniku omogućiti izmjenu tih podataka. Modal ažuriranja podataka otvaramo pomoću gumba u tablici koji je već stvoren, pri otvaranju modala, prosljeđuju se podaci o toj edukaciji. Ažuriranje radimo slično kao i stvaranje edukacije osim što sa proslijeđenim podacima već unaprijed ispunjavamo form group (Slika 35.), te korisniku preko HTML datoteke nudimo izmjenu podataka koji već učitani

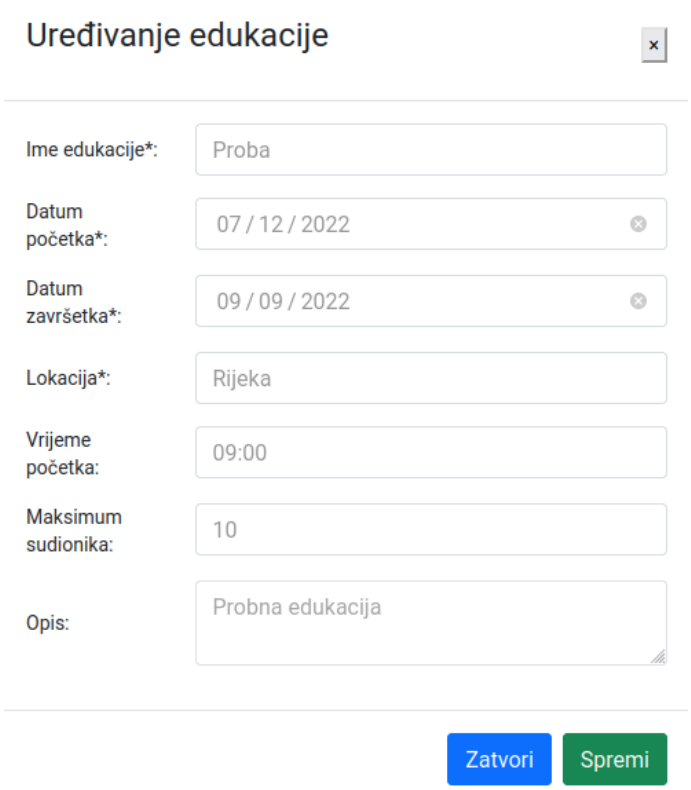
```
ngOnInit(): void {
  this.editForm.controls['name'].setValue(this.row.name);
  this.editForm.controls['date_from'].setValue(this.row.date_from);
  this.editForm.controls['date_to'].setValue(this.row.date_to);
  this.editForm.controls['location'].setValue(this.row.location);
  this.editForm.controls['start_time'].setValue(this.row.start_time);
  this.editForm.controls['maximum_participants'].setValue(this.row.maximum_participants);
  this.editForm.controls['description'].setValue(this.row.description);
}
```

Slika 36: Unaprijed postavljanje vrijednosti na već postojeće

U HTML datoteci kreiramo input oznake poput i za stvaranje edukacije (Slika 36.), kako bi korisniku omogućili unos podataka (Slika 37.).

```
<form [formGroup]="editForm">

  <div class="form-group row">
    <div class="col-3 d-flex align-items-center">
      <label>Ime edukacije*:</label>
    </div>
    <div class="col-9">
      <input class="form-control" type="text"  formControlName="name"   >
    </div>
    <div>

    </div>
  </div>
```

Slika 37: Obrazac za ažuriranje edukacije



Slika 38: Izgled obrasca za ažuriranje

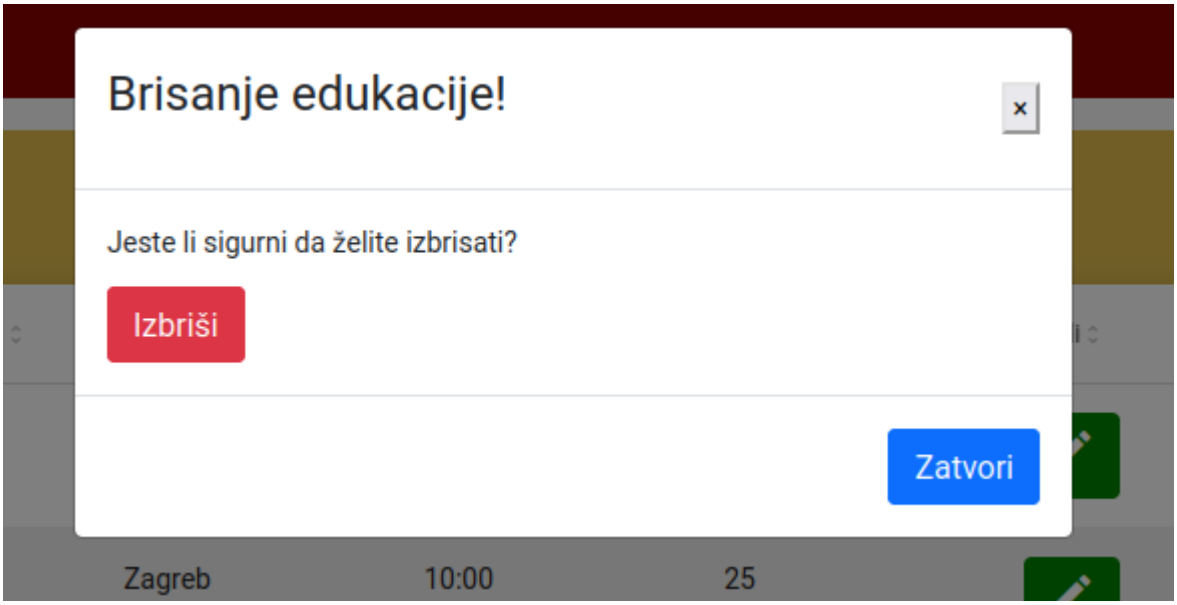
Prilikom klika na gumb spremi pozivamo funkciju ažuriraj *(eng. update*) (Slika 38.), koja dobivene podatke kroz form group sprema u jednu varijablu te ih šalje serverskog strani zajedno sa ID-om edukacije. Moramo poslati ID jer jedinstveno označuje entitet i pomoću njega govorimo serverskog strani koju edukaciju ažurirati.

```
update() {
  const data = this.editForm.value;
  this.promiseBtn = (async () => {
    const result = await this.educationService.update(this.row._id,data );
    if (!result.success) {
      this.failedToastr()
      return;
    }
    this.savedToastr();
    this.aModal.close({ success: true });

  })()
}
```

Slika 39: Skripta za ažuriranje

Brisanje edukacije

Brisanje edukacije iz baze se izvršava preko modala za brisanje (Slika 39.) koji nudi korisniku izbornik da li je siguran da želi pobrisati edukaciju. Modal komponenta za brisanje otvaramo pomoću gumba za brisanje koji se nalazi u tablici, te klikom na gumb ujedno komponenti za brisanje prosljeđuje ID te edukacije kako bi mogli jedinstveno identificirati taj entitet u bazi.



Slika 40: Izgled modala za brisanje

Klikom na gumb izbriši poziva se funkcija obriši (*eng. delete*) (Slika 40.) koja šalje naredbu brisanja zajedno sa ID te edukacija na serversku stranu kako bi se pobrisala iz baze.

```
delete(){
  this.promiseBtn = (async () => {
    const result = await this.educationService.delete(this.education_id);
    if (!result.success) {
      this.failedToastr()
      return;
    }
    this.aModal.close({ success: true });
    this.savedToastr()
  })()
}
```

Slika 41: Skripta za brisanje

Zaključak

U ovom radu obrađen je razvoj CRUD metoda nad entitetom edukacije, koristeći Node.js kao serversku stranu i Angular okruženje za razvijanje klijentske strane. Ovim procesom razvijeni su i entiteti volontera i vježbi.

Entiteti i njihovi atributi su razvijeni u dogovoru sa Crvenim križom Primorsko-goranske županije, koristeći njihove prijavne obrasce i dokumente te razgovor sa zaposlenicima tj. budućim korisnicima.

Aplikacija je testirana na pregledniku Mozzila Firefox unutar operativnog sustava Ubuntu.

Projekt dokumentiran u ovom radu nije konačan te će se nastavljati razvijati kako se budu mijenjale potrebe Crvenog Križa. Sljedeći korak u ovom projektu je stvaranje jednostavnog sustava prijave koji će sadržavati samo administrator korisnika jer pristup aplikaciji će biti dozvoljen samo upravi Crvenog Križa Primorsko-goranske županije.

Popis slika

Literatura

1. Statistički podaci o popularnosti klijentski okruženja (https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190) (pristupljeno 6. Rujna 2022).
2. Definicija HTML-a (https://html.com/#What_is_HTML)) (pristupljeno 6. Rujan 2022).
3. HTML5 oznake (https://www.geeksforgeeks.org/difference-between-html-and-html5/) (pristupljeno 6. Rujan 2022).
4. Definicija CSS-a (https://developer.mozilla.org/en-US/docs/Web/CSS) (pristupljeno 6. Rujna 2022).
5. TypeScript (https://www.typescriptlang.org/) (pristupljeno 6. Rujna 2022).
6. Node.js (https://www.w3schools.com/nodejs/nodejs_intro.asp) (pristupljeno 6. Rujna 2022).
7. Načelo JavaScript svugdje (https://sainttobs.medium.com/javascript-everywhere-web-mobile-and-desktop-68131878d22d) (pristupljeno 6.Rujna 2022).
8. Angular (https://angular.io) (pristupljeno 20. Kolovoz 2022).

9. MongoDB (https://www.mongodb.com/docs/manual/) pristupljeno (6. Rujna 2022).
10. Definicija entiteta (https://www.srce.unizg.hr/files/srce/docs/edu/osnovni-tecajevi/d310_polaznik.pdf) (6. Rujna 2022).
11. Načelo aplikacije na jednoj stranici (https://www.bloomreach.com/en/blog/2018/what-is-a-single-page-application) (6. Rujna. 2022)
12. Ngx-datatable tablični prikaz podataka (https://swimlane.github.io/ngx-datatable) (6. Rujna 2022.)