Pregled korištenja modernih alata za razvoj web aplikacija na primjeru jednostavne web aplikacije primjenjive u vatrogasnoj postaji

Lepen, David

Sveučilište Sjever

Završni rad br. 547/EL/2024

Pregled korištenja modernih alata za razvoj web aplikacija na primjeru jednostavne web aplikacije primjenjive u vatrogasnoj postaji

David Lepen, 0336042891

Varaždin, rujan 2024. godine

Sveučilište
Sjever

Odjel za elektrotehniku

Završni rad br. 547/EL/2024

Pregled korištenja modernih alata za razvoj web aplikacija na primjeru jednostavne web aplikacije primjenjive u vatrogasnoj postaji

Student

David Lepen, 0336042891

Mentor

Mr.sc. Matija Mikac, dipl.ing.

Varaždin, rujan 2024. godine

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR-42000 Varaždin

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL: Elektrotehnika

STUDIJ: Stručni prijediplomski studij elektrotehnike

PRISTUPNIK: David Lepen | MATIČNI BROJ: 0336042891

DATUM: 12.09.2024. | KOLEGIJ: Osnove web programiranja

NASLOV RADA: Pregled korištenja modernih alata za razvoj web aplikacija na primjeru jednostavne web aplikacije primjenjive u vatrogasnoj postaji

NASLOV RADA NA ENGL. JEZIKU: Overview of the Use of Modern Web Development Tools in the Example of a Simple Web Application Applicable in a Fire Station

MENTOR: mr.sc. Matija Mikac, dipl.ing.el. | ZVANJE: viši predavač

ČLANOVI POVJERENSTVA:

1. mr.sc. Ivan Šumiga, dipl.ing.el., viši predavač - predsjednik povjerenstva
2. Miroslav Horvatić, dipl.ing.el., viši predavač - član
3. mr.sc. Matija Mikac, dipl.ing.el., viši predavač - član / mentor
4. dr. sc. Emil Dumić, dipl.ing. el., izvanredni profesor - zamjenski član
5.

Zadatak završnog rada

BROJ: 547/EL/2024

OPIS:

Iako temeljni pristup razvoju web aplikacija (HTML5, CSS3, JavaScript, PHP), obrađivan na kolegijima tijekom studija, omogućava izradu kompleksnih programskih rješenja, u praksi poslodavci često koriste moderne alate i razvojne predloške. Jedan od pristupa je korištenje React razvojnog okvira, zajedno sa drugim vezanim razvojnim podsustavima.

Završnim radom je potrebno ukazati na mogućnosti korištenja React razvojnog okvira, te ih primijeniti u izvedbi web aplikacije za evidenciju intervencija vatrogasne postaje. Rezultat rada mora biti funkcionalno programsko rješenje koja će uključivati implementaciju klijentskog i serverskog dijela.

Pismeni dio rada mora uključiti: * opis osnovnih tehnologija za razvoj web aplikacija, * idejni opis funkcionalnosti web aplikacije koja se razvija kao praktični dio rada, * pregled osnova React razvojnog okvira i pratećih podsustava odabranih za izvedbu web aplikacije, * prikaz implementacije nekoliko ključnih dijelova web aplikacije (isječci izvornog koda, objašnjenja i potrebni prilozi), * opis funkcionalnosti i korisničkog sučelja razvijane web aplikacije, * primjer izvedbe bar jedne od funkcionalnosti korištenjem osnovnih tehnologija (čisti HTML, JavaScript, PHP...) i usporedbu s izvedbom iste funkcionalnosti korištenjem React i odabranih podsustava, * osvrt i zaključak vezan uz iskustvo razvoja aplikacije.

ZADATAK URUČEN: 16.09.2024. | POTPIS MENTORA:

SVEUČILIŠTE SJEVER

Predgovor

Prije svega, posebne zahvale upućujem mentoru koji mi je pomogao s mnogo korisnih savjeta i stručnih mišljenja.

Također, zahvaljujem se svim profesorima, asistentima i ostalim djelatnicima koji su mi prenijeli mnogo znanja iz područja elektrotehnike.

Na kraju, želio bih se zahvaliti roditeljima na potpori i osloncu tijekom studija, a posebne zahvale upućujem svojoj djevojci.

Sažetak

Cilj ovog završnog rada je izrada web aplikacije koja prikazuje vatrogasnu postaju i načine na koje se ista može modernizirati. Na primjer, za lakše praćenje izlaženja na intervencije kroz godinu, može se napraviti središnja baza podataka koja sadrži razne skupove podataka dovoljne za praćenje i analizu evidencija, što u konačnici može unaprijediti funkcioniranje postaje. Također, to omogućava lakše održavanje postaje jer svi podaci mogu biti na istom mjestu i lako im se može pristupiti. Modernije i naprednije vatrogasne postaje u pravilu već koriste različite informacijske sustave koji olakšavaju upravljanje, no manje i dobrovoljno vatrogasna društva često nemaju nikakva pomoćna rješenja. Praktični dio rada je namijenjen upravo takvim starijim postajama. Druga funkcionalnost koja nije usko vezana uz rad stanice, u smislu modernizacije, je da gost (posjetitelj web aplikacije) ima mogućnost da umjesto prijave intervencije mobitelom, vrši prijavu i preko web aplikacije. Web aplikacija, razvijana u sklopu završnog rada, omogućava unos tih intervencija, te daljnje administriranje i obradu od strane osoblja postaje. Tehnološki, praktično rješenje koje je sastavni dio rada izvedeno je korištenjem trenutno najpopularnijih razvojnih okvira i na klijentskoj i na poslužiteljskoj strani (*React*, *Next* JS, *PostgreSQL*).

Ključne riječi: Web aplikacija, *JavaScript*, *React*, *Next* JS, *Visual Studio Code*, vatrogasna postaja, *Prisma*

Popis korištenih kratica

HTML *HyperText Markup Language*
Sintaksa za obilježavanje hipertekstualnih dokumenata.

WWW *World Wide Web*
Globalni informacijski prostor koji korisnicima omogućava pregled, povezivanje i međusobnu razmjenu informacija.

CSS *Cascading Style Sheets*
Jezik za opisivanje stila HTML dokumenata.

URL *Uniform Resource Locator*
Jedinstvena adresa koja identificira resurs na webu, bilo da je to web stranica, slika, video ili bilo koja druga vrsta sadržaja.

HTTP *Hypertext Transfer Protocol*
protokol koji omogućava prijenos podataka između web preglednika i web poslužitelja, te je osnovni protokol za komunikaciju na webu.

AJAX *Asynchronous JavaScript and* XML
Tehnika za stvaranje bržih i dinamičnijih web aplikacija tako da učitava podatke u pozadini i prikazuje ih bez potrebe za ponovnim učitavanjem cijele stranice

PWA *Progressive web App*
Vrsta aplikacije koja koristi modernu web tehnologiju kako bi korisnicima pružila iskustvo slično korištenju nativnih mobilnih aplikacija

API *Application Programming Interface*
Skup definiranih pravila i protokola koji omogućavaju različitim softverskim aplikacijama da komuniciraju međusobno

JSX *Javscript* XML
Sintaksna ekstenzija za *JavaScript* koja se koristi u *React* programiranju za opisivanje kako korisničko sučelje treba izgledati

DOM *Document Object Model*
Predstavlja strukturirani model dokumenta koje izgleda kao stablo, gdje svaki čvor predstavlja dio dokumenta

SSR *Server Side Rendering*
Tehnika gdje se sadržaj web stranice generira na poslužitelju, a ne u pregledniku korisnika

SSG *Static Site Generation*
Za razliku od SSR-a gdje se dinamično generiraju stranice na poslužitelju, SSG unaprijed generira HTML stranice i pohranjuje ih kao statične datoteke koje se onda poslužuju korisnicima

PHP *Hypertext Preprocessor*
Skriptni jezik koji je prvobitno stvoren kao alat za dinamičko generiranje web stranica, a često se koristi i za izradu složenih web aplikacija, za upravljanjem sadržaja, forumom i drugih vrsta web stranica.

SEO *Search Page Optimization*
Proces optimizacije web stranica i sadržaja kako kako bi se poboljšalo rangiranje stranica u rezultatima pretraživanja na tražilicama.

Sadržaj

1. Uvod

U današnje doba, većina tvrtki i ustanova ima svoju web stranicu, pri čemu dobar dio onih orijentiranih krajnjim korisnicima nude i određene oblike web aplikacija. S obzirom da sam imao uvid u funkcioniranje lokalne vatrogasne postaje koja nema web stranicu, dobio sam ideju da stečeno znanje isprobam izradom web aplikacije za praćenje evidencije intervencija neke imaginarne vatrogasne postaje. Mogao sam koristiti standardne alate i tehnologije o kojima sam naučio tijekom obrazovanja, no uz konzultacije s potencijalnim poslodavcem odlučio sam za razvoj odabrati neke od alata razvojnih okvira koji se trenutno najviše koriste u web programiranju – *React, Next* JS i s njima kompatibilne razvojne sustave. Izradom web aplikacije i osnovnog web sjedišta (naslovne, *landing* stranice) stekao sam dodatnu praksu u radu s rješenjima naprednijim od onih naučenih tijekom studija, a to iskustvo bi mi moglo koristiti u budućnosti. Razvijana aplikacija nije, naravno, nužna za rad postaje, ali u određenim segmentima može pomoći i ukazati na mogućnosti vođenja raznih vrsta evidencija, pri čemu se dio informacija može i javno objaviti na web stranicama sjedišta postaje, što je također implementirano u web aplikaciji. Završnim radom je opisan razvoj i klijentskog (engl. *frontend*) i poslužiteljskog (engl. *backend*) dijela, pri čemu sam se funkcionalno ograničio na evidenciju intervencija. Sastavni dio aplikacije je i naslovna stranica imaginarne postaje koju postaja prilagođava prema potrebama - konkretna izvedba uključuje vizuale stvarne postaje, osnovne kontakt informacije i kratku povijest postaje. Koncept evidencije intervencija predviđa višekorisnički rad, mogućnost rada anonimnih i prijavljenih korisnika, kao i posebne ovlasti za administratore.. Završni rad je organiziran kako slijedi – nakon uvodnog poglavlja slijedi poglavlje koje opisuje samu pripremu i planiranje za izradu web aplikacije, odabir tehnologija i alata, s posebnim naglaskom i detaljnijim opisom alata korištenih u izradi praktičnog dijela. Nakon toga, treće poglavlje opisuje motivaciju za izradu i razvoj web aplikacije, zajedno s popisom funkcionalnosti i opisom procesa razvoja web aplikacije, dok završava s idejama mogućih nadogradnji aplikacije. Nakon toga u poglavlju s rezultatima detaljno je prikazano korisničko sučelje i opisano uobičajeno korištenje aplikacije.

1.1. Motivacija

Želeći naučeno o web programiranju nekako primijeniti u praksi, razmišljao sam o temi praktičnog projekta (razvoju web aplikacije) koja nije toliko uobičajena, a može imati praktičnu vrijednost. Obzirom na neka prethodna iskustva, došao sam na ideju vezanu uz vatrogastvo, konkretnije evidenciju vatrogasnih intervencija. Jasno je da sve postaje moraju voditi i vode takvu

evidenciju, a mene je zanimalo bi li se dio tih informacija mogao i ponuditi javnosti preko web stranica postaje.

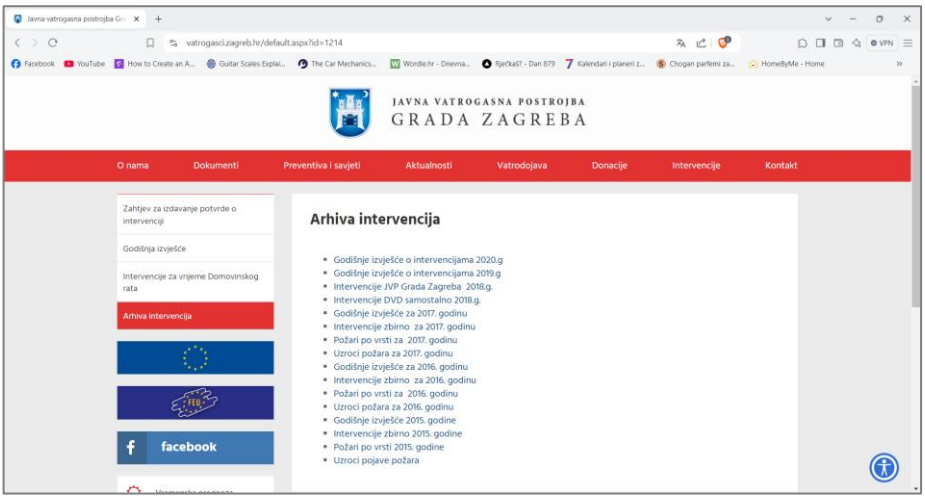
Ideja za web aplikacijom vatrogasne postaje je proizašla iz toga što većina manjih vatrogasnih postaja i dobrovoljno vatrogasnih društva nemaju web stranicu. Proučio sam nekoliko lokalnih dobrovoljno vatrogasna društva i javnih vatrogasnih postaja (tablica 1.) i došao do zaključka da većina lokalnih dobrovoljno vatrogasnih društva nemaju aktivnu web stranicu. Prva, ujedno i veća vatrogasna postaja koja ima web stranicu jest JVP Čakovec. Zato sam odlučio da ću izraditi takvu, za početak i potrebe rada jednostavnu, web aplikaciju - za unos i pregled evidencije intervencija koje neka imaginarna postaja odrađuje. Pretpostavljam da već i ovako jednostavna izvedba može biti korisna i primjenjiva u manjim postajama, dok za složenije sustave ipak nije primjerena. Vodio sam se web stranicom veće vatrogasne postaje - javne vatrogasne postaje Zagreb. Javna vatrogasna postaja Zagreb ima mogućnost pregleda popisa intervencija kroz godinu te druga izvješća (uzroci požara, požari po vrsti i sl.). Ti primjeri izvješća mogu se vidjeti niže na slici 1. Dok je na slici 2. dan prikaz arhive intervencija dobrovoljnog vatrogasnog društva Mala Subotica.

Tablica 1: Pregled lokalnih vatrogasnih postaja i njihova aktivnost na webu.

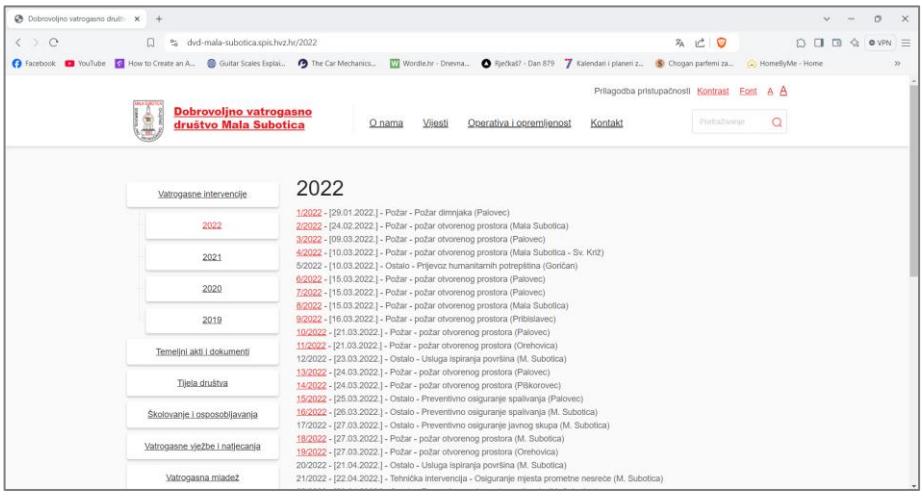
Mjesto	Tip	Aktivna Web stranica	Arhiva intervencija
Donji Kraljevec	DVD	Ne	Ne
Hodošan	DVD	Ne	Ne
Donji Hrašćan	DVD	Ne	Ne
Palinovec	DVD	Ne	Ne
Sv. Juraj u Trnju	DVD	Ne	Ne
Mala Subotica	DVD	Da	Da
Prelog	JVP	Ne	Ne
Čakovec	JVP	Da	Ne
Zagreb	JVP	Da	Da

Drugi korak je bio izbor alata i tehnologija kojima bi realizirao takav projekt. Iako bi i osnovnim pristupom (HTML, CSS, *JavaScript*, PHP) obrađivanim tijekom studija mogao realizirati željeno, kroz neke kontakte s potencijalnim budućim poslodavcima došao sam do zaključka da bi bilo korisnije primijeniti neki od pristupa i tehnika koje bih koristio u daljnjem zaposlenju.

Ponukan time, odlučio sam naučiti i pokušati primijeniti često korištene razvojne okvire i dodatke, konkretno *React* i *Next* JS te više njima prilagođenih razvojnih podsustava.



Slika 1. Arhiva intervencije javne vatrogasne postrojbe Zagreb [1]



Slika 2. Arhiva intervencija dobrovoljno vatrogasnog društva Mala Subotica [2]

2. Osnove web programiranja

Tijekom školovanja tema programiranja i mreže bila je obrađivana na više kolegija. Na samom početku odslušani su kolegiji vezani uz računalne mreže. Konkretno, upoznavanje s računalnim mrežama, standardima komunikacijskih protokola te razumijevanje metoda pristupa računala i mobilnih uređaja Internetu. Sljedeći kolegij je bio vezan uz same algoritme i programske jezike (konkretno, programiranje u C/C++ jeziku). Tim kolegijem se studentu omogućava doticaj s osnovnim principima programiranja, primjena naučenih algoritama i razradu i osmišljavanje jednostavnijih algoritama. Prvi kolegij koji se doticao web programiranja jest 'Baze podataka i SQL'. Svaka moderna web aplikacija na strani poslužitelja koristi neku bazu podataka kako bi se unesene informacije pohranile i obrađivale. Često su to relacijske baze i specijalizirani jezik, SQL, a to je upravo ono što smo naučili. Prvi pravi doticaj s web aplikacijom biva na izbornom kolegiju 'Osnove web programiranja'. Sam rad i učenje na kolegiju imali su utjecaj na odabir teme završnog rada. Teorijski se obrađuju koncepti web aplikacije te stječu znanja za samostalnu izradu jednostavne web aplikacije. Obrađuju su elementi poslužitelja gdje se koristi programski jezik PHP te elementi klijenta na kojem se koristi programski jezik *JavaScript.* Zaključak nakon odslušanih predavanja i rada na kolegijima jest da se programiranje može smatrati kao jednom vrstom alata s kojim možemo realizirati određene funkcije ili riješiti nastale probleme.

2.1. Web stranice i web aplikacije

U posebnim slučajevima web stranice i aplikacije mogu biti izvedene samo na strani klijenta. Ali, uobičajeno o webu govorimo kao o sadržajima (i funkcionalnostima, ako se radi o aplikaciji) pohranjenim i realiziranim na web poslužitelju (HTTP poslužitelj - protokol koji služi za dohvat sadržaja je HTTP, web preglednici kao softver koji korisnici koriste se smatraju HTTP klijentima).

Web stranica je statički internetski sadržaj koji najčešće služi samo za pregledavanje sadržaja i pružanje informacija korisnicima. Interaktivnosti nema i korisnik jedino može čitati, odnosno prolaziti web stranicom. Primjeri mogu biti osobni portfolio, dokumentacija proizvoda, stranica proizvoda, blog [3].

Za razliku od web stranica, kod web aplikacija glavno obilježje je interaktivnost (s korisnikom) i/ili realizacija određenih funkcionalnosti (najčešće vezano uz interakciju s korisnikom, ali ne nužno). Korisnici mogu vršiti interakciju i obavljati specifične zadatke. Web sadržaji (i stranice i aplikacije) u pravilu su zapisani pomoću HTML-a. HTML se smatra jezikom za definiranje sadržaja i strukture dokumenta [3]. Izgled (manje bitno za nas programere, ali vrlo bitno za korisničko iskustvo) se definira stilovima i jezikom CSS, a interakcija i programiranje na

klijentskoj strani izvodi programskim jezikom *JavaScript*. Kad je potrebno (u aplikacijama gotovo sigurno) realizirati nešto na poslužitelju, za to se često koristi programski jezik PHP. No, za sve te osnovne jezike postoje različite "nadogradnje", biblioteke i razvojni okviri koji mogu ubrzati razvoj (upravo o tome se i radi u ovom završnom radu - odabran je *React* i *Next* JS s pratećim podsustavima). Primjeri web aplikacija su online trgovine, društvene mreže i sl.

Za krajnjeg korisnika, web stranice i web aplikacije mogu naizgled biti iste, štoviše jednake jer korisnike se isto usmjerava klikom na bilo web stranicu ili web aplikaciju. Naime, odatle se korisničko iskustvo razlikuje ovisno radi li se o stranici ili aplikaciji. Web stranica je većinom sastavljena od statičkog sadržaja, što ju čini idealnom za pisanje blogova i članaka te su one informativne prirode. Najbolji primjer su web stranice restorana jer se tamo nalazi hrpa informacija, od kojih neke mogu biti adresa restorana, jelovnik ili kontakt. Web stranica može imati dijelove web aplikacije koji su interaktivni. Najbolji primjer za to su veze na druge stranice ili mjesta na webu. Ako web platforma ne sadrži elemente koji se mijenjaju ovisno o radnji korisnika, a koji bi rezultirali odgovorom, odnosno promjenom na web stranici, to nije web aplikacija nego web stranica. Danas je često teško razlučiti radi li se o stranici ili aplikaciji. Tehnički, čim se na strani poslužitelja koristi PHP ili neki drugi programski jezik, koji dinamički generira dio sadržaja govorimo o aplikaciji (jer je netko morao to isprogramirati). Slično i na klijentu - ako postoji *JavaScript* koji nešto smisleno odradi, već možemo govoriti o klijentskoj web aplikaciji. S korisničke strane, moguće je da rijetko tko uopće razmišlja o tome - sve je to "web".

Dobar primjer web aplikacije, iako možda to doživljavamo toliko normalno i uobičajeno, su društvene mreže (krajnji korisnici danas pojam aplikacija obično vežu uz mobilne uređaje). Web shop je također dobar primjer kompleksne web aplikacije. Ukoliko se spusti na najnižu razinu, u konačnici to je web stranica koja prikazuje informacije, ali dodatak tome je interakcija korisnika. Ako korisnik može potaknuti interakciju, a natrag dobiti odgovor – to je web aplikacija. Tako, u gore navedenom primjeru korisnik može naručiti artikle, sortirati ili filtrirati sadržaj čime on aktivno i dinamično vrši interakciju s web aplikacijom. Također, razlika je i u samom kodu i izradi jednog i drugog. Naime, da bi web aplikacije mogle vršiti interakciju s korisnikom, to trebaju imati i podržano u svom kodu. Za običnu web stranicu je potrebno imati „samo“ HTML i CSS, ali za web aplikaciju je potreban i dodatan jezik kao što je *JavaScript*. S tim jezikom mogu se implementirati dodatne mogućnosti poput trake za pretraživanje, interakcija pritiska na gumb ili popunjavanje obrazaca.

U današnje vrijeme, crta između statičke i dinamičke stranice ili web stranice i web aplikacije može postati nerazlučiva iz razloga što mnoge web platforme sadrže oboje. Primjer gdje se web stranica i web aplikacija prepliću je web stranica poslovnog objekta koja bi pružala informacije o

tom objektu, ali također bi se i vezala na internetsku trgovinu tog objekta. Konkretan primjer je frizerski salon koji ima svoju web stranicu gdje prikazuje informacije o svojim uslugama ili kontaktu, no ima i mogućnost online naručivanja. Mnogi obrti ili mala poduzeća započinju kreiranjem jednostavnih web stranica koje zadovoljavaju njihove osnovne potrebe. No, kako se šire i rastu, potrebna im je veća funkcionalnost kako bi ispunili potrebe svojih klijenata [3].

2.2. Standardi i pojmovi

2.2.1. HTML i HTTP

U 1991. godini, Berners-Lee je predstavio HTML (akronim od engl. *Hypertext Markup Language*), temeljni jezik web-a. HTML je omogućavao strukturiranje dokumenata s hiperlinkovima koji su formirali bazu web stranice. Prva ikad web stranica je bila objavljena 1993. Osim hiperlinkova i HTML-a, tri su ključne karakteristike moderne web aplikacije: interaktivnost, responzivnost i sadržaj. Web aplikacije su osnovni gradivni blok interneta i koriste se u različite svrhe, uključujući informiranje, zabavu, komunikaciju, e-trgovinu, obrazovanje itd. Web aplikacije mogu sadržavati interaktivne elemente kao što su animacije, forme ili skripte koje omogućavaju dinamičko ponašanje. Također, moderne web aplikacije obično su dizajnirane da budu responzivne, odnosno da se prilagođavaju različitim veličinama zaslona i uređajima kako bi pružile optimalno korisničko iskustvo. Od samih početaka HTML je bio nadograđivan, verzija koja je bila aktivna dugi niz godina je HTML5. Međutim, bilo je odlučeno da se prestane s numeracijom i koristi tzv. HTML *Living Standard*, o čemu više u poglavlju 2.2.3. [4].

HTTP (akronim od engl. *HyperText Transfer Protocol*) je možda i najkorišteniji aplikacijski protokol (aplikacijski protokol - jedan od četiri slojeva internet modela) koji je temelj svake razmjene podataka na webu. Princip rada je takav da klijent šalje zahtjev (engl. HTTP *Request*), a poslužitelj odgovara (HTTP *response*). S obzirom da se radi o web aplikaciji i izmjenama podataka na webu, HTTP protokol je temelj i od njega sve polazi [5].

2.2.2. CSS i *JavaScript*

CSS je W3C (akronim od engl. *World Wide Web Consortium*) u 1996. godini odabrao kao preporučeni alat za stiliziranje na webu. CSS dopušta programerima da odvoje sadržaj od prezentacije, čineći web fleksibilnijim i standardiziranim. Pomoću CSS-a se može promijeniti

izgled web stranice te definirati izgled na različitim zaslonima uređaja. CSS nije programski jezik kao C++ ili *JavaScript*, no to ne znači da ga je lakše razumjeti [6].

JavaScript je programski jezik koji dodaje interakciju za korisnika koji posjećuje web stranicu. Preko *JavaScripta* može se isprogramirati što će se dogoditi prilikom pritiska na gumb, nakon popunjavanja obrasca i drugih radnji. Autor *JavaScripta* je Brendan Eich. Prvi naziv *JavaScript*-a bio je „*Mocha*“, kasnije je promijenjen u „*LiveScript*“ i na kraju „*JavaScript*“. Interpretirani jezik *JavaScript* postao je preporučeni 1997. godine. Ima široku primjenu, no glavna je davanje interaktivnosti web stranici. JavaScript aktiviraju i izvode web preglednici, no uz to moderni preglednici posjeduju i napredne mogućnosti prevođenja (JIT) što je poboljšalo performanse izvođenja. Relativno je kompaktan, ali i fleksibilan. Programeri su tijekom godina nadodali mnoštvo alata koji se vežu na njega, s čime su otvorili pregršt novih funkcionalnosti, a neke od tih su:

- API (akronim od engl. *Application Programming Interface*) - skup određenih pravila koji programeri slijede kako bi se mogli koristiti resursima ili uslugama operacijskog sustava ili web preglednika
- API trećih strana - omogućavaju programerima pristup funkcionalnostima drugih aplikacija, kao što je npr. *Facebook*
- razvojni okviri (engl. *framework*) i biblioteke (engl. *libraries*) koje ubrzavaju izradu web aplikacija [7]

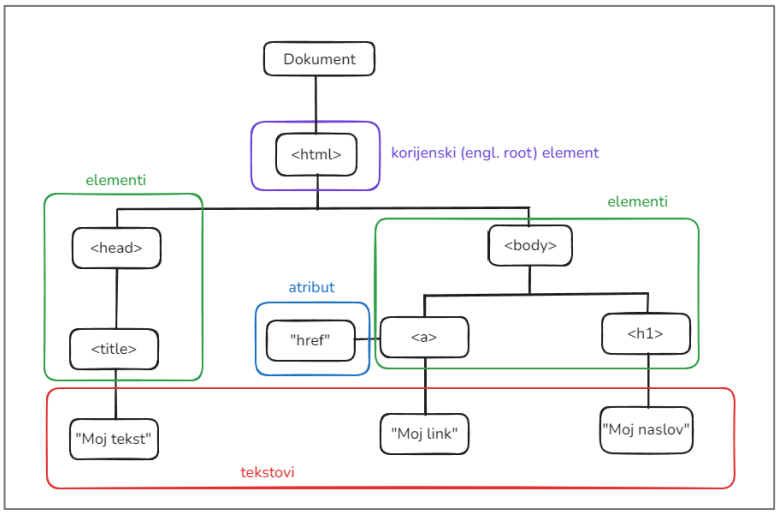
Nekoliko konkretnih primjera koji se svakodnevno koriste [8]:

- *Geolocation* API - omogućava web aplikacijama pristup geografskim podacima uređaja
- *Web Storage* API - omogućava spremanje *key-value* parova u samom pregledniku
- *WebSocket* API – omogućava dvosmjernu komunikaciju između preglednika i poslužitelja (aplikacije za razmjenjivanje poruka – *chat*, notifikacije uživo) u stvarnom vremenu
- *Camera* API – omogućava web aplikacijama da zatraže pristup kameri i mikrofonu korisnika

Najpopularniji je programski jezik u web programiranju ključan za izradu dinamičkih web aplikacija. 95% od više od 1.6 milijardi web aplikacija na Internetu koristi *JavaScript*. Primjeri korištenja *JavaScript*-a su web aplikacije, serverske aplikacije, igre, mobilne aplikacije i dr. Standard koji se koriti uz *JavaScript* zove se *ECMAScript* i osigurava interoperabilnost web

aplikacija u različitim web preglednicima. Standardizirao ga je *Ecma International* u dokumentu „ECMA-262“ [7].

Pomoću DOM-a (akronim od engl. *Document Object Model*) preglednici interno zapisuju dokument. Koriste strukturu stabla kako bi modelirali HTML i CSS. DOM prikazuje dokument pomoću čvorova (engl. *nodes*) i objekata. Kao objektno orijentirani prikaz (engl. *object-oriented representation*) web stranice, može se modificirati skriptnim jezikom - *JavaScriptom*. Te modifikacije uključuju dinamičko mijenjanje sadržaja, strukture ili stila web stranica [9]. Primjer DOM-a je prikazan na slici 3.



Slika 3. Primjer DOM-a imaginarne web stranice

HTML dokument na slici 3. je hijerarhijska struktura stabla. Na samom vrhu je korijenska (engl. *root*) grana koju predstavlja „`html`“ element (obavezan za cjelokupnu strukturu i interpretaciju web stranice). Ispod korijenskog elementa nalaze se njegove tzv. *child* grane koje prikazuju „`head`“ (sadržava *metadata* i podatke o dokumentu koji nisu prikazani direktno na web stranici) i „`body`“ (sadržava cjelokupan sadržaj koji je vidljiv korisniku na web stranici) elemente, između ostalih. Postoje više vrsta grana, odnosno čvorova, a najuobičajeniji su [10]:

- ***Element Nodes*** (predstavljaju elemente kao što su *„div“, „p“, „span“*)
- ***Text Nodes*** (predstavljaju tekstualni sadržaj koji se nalazi unutar elemenata)
- ***Attribute Nodes*** (predstavljaju atribute elemenata)
- ***Document Nodes*** (predstavljaju cijeli dokument)
- ***Comment Nodes*** (predstavljaju komentare unutar HTML dokumenata)
- ***Document Type Nodes*** (predstavljaju deklaraciju tipa dokumenata – „!DOCTYPE html“)

2.2.3. Moderno web programiranje

Dva moderna, ali različita aspekta HTML-a, koja su značajno pomogla održati HTML u toku sa web tehnologijama su HTML *Living Standard* i HTML5. Čim je HTML5 bio finaliziran fokus se prebacio na HTML *Living Standard*. Tablica 2 sažima ključne razlike između HTML5 i HTML *Living Standarda*, ističući njihove razvojne modele, održavanja, ažuriranja i kompatibilnosti sa starijim verzijama. Bitna razlika koju je potrebno naglasiti jest da se HTML *Living Standard*, kojeg održava WHATWG (akronim od engl. *Web Hypertext Application Technology Working Group*), konstantno razvija i nema numeriranja verzija (HTML5 za svako novo ažuriranje zahtjeva novu verziju).

Tablica 2: Razlike između HTML5 i HTML Living Standard.

Aspekti	**HTML5**	**HTML *Living Standard***
Razvojni model	verzionirana specifikacija	stalna ažuriranja
Održavanje	održava W3C	održava WHATWG
Adaptacija i korištenje	ažuriranje kod značajne promjene	moderna osnova za web razvoj
Kompatibilnost s prethodnim verzijama	stabilna i kompatibilna s prethodnim verzijama	razvija se, ali održava kompatibilnost s prethodnim verzijama
Ažuriranja značajki	nema novih značajki nakon verzije HTML5 (osim sitnih verzija)	redovito se dodavaju nove značajke i poboljšanja
Verzija	specifična verzija (HTML5)	bez brojeva verzija, uvijek aktualna

CSS3 je trenutna verzija CSS-a te predstavlja moderni standard za stiliziranje web sadržaja, pružajući snažne alate za stvaranje responzivnih, vizualno bogatih i interaktivnih web stranica. Modularna priroda CSS3 omogućuje stalna poboljšanja, ali također znači da programeri moraju biti u tijeku s najnovijim specifikacijama i podrškom za preglednik. Iako su prednosti značajne, posebno u pogledu izgleda i mogućnosti dizajna, programeri se i dalje moraju snalaziti u izazovima povezanim s kompatibilnošću i performansama preglednika [11].

Web programiranje se širi i još će se širiti. Također, unaprjeđivati će se i korisničko iskustvo. Sve više će se uključivati umjetna inteligencija i strojno učenje za personalizirano korisničko iskustvo. Virtualna realnost (engl. *Virtual Reality*) i AR (akronim od engl. *Augmented Reality*) već sada utječu na web programiranje i pružaju nove zanimljive mogućnosti. Programeri će uvijek tražiti i istraživati nove funkcionalnosti. Uz svu raznovrsnost web aplikacija ipak postoje segmenti koji su zajednički ili vrlo slični u svim situacijama - npr. autentifikacija korisnika, potreba za

pretraživanjem izvora podataka i kvalitetnim prikazom. Za realizaciju tih funkcionalnosti, kako na strani klijenta tako i na poslužiteljskoj strani, programeri moraju uložiti poprilično vremena na razvoj i testiranje, pa se s vremenom, obzirom da se radi o često potrebnim funkcionalnostima, pojavio niz biblioteka, razvojnih okvira (engl. *framework*), komercijalnih i besplatnih, koji olakšavaju i ubrzavaju posao programerima. Zbog naprednijeg i modernijeg programiranja sami programski jezici nisu bili „dovoljni“ za potrebe klijenata. Naravno, programiranje od nule (HTML, CSS, JS, PHP) omogućava potpunu kontrolu nad svim segmentima web aplikacije, ali iziskuje dosta vremena pa često nije prvi izbor i nije pogodno za brzi razvoj klasičnih aplikacija namijenjenih široj publici. „Čistim“ *JavaScriptom* se može raditi i danas, no sada je više pitanje vremena, odnosno koliko je potrebno programeru da kreira nešto s bazičnim jezikom u odnosu na rad s alatom koji se nadovezuje na *JavaScript* te smanjuje vrijeme programiranja. Samim skraćivanjem vremena aplikacija može prije stići do korisnika što uvećava korisničko iskustvo i zadovoljstvo.

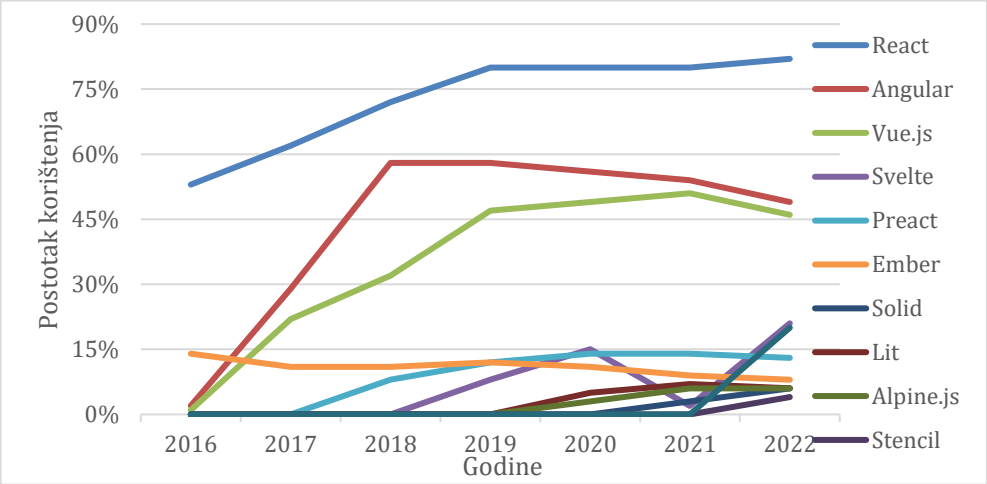
Vezano uz moderno web programiranje, jasno je da se posao dijeli na više dijelova. Aplikacija ima *frontend* (strana klijenta) te može uključivati *backend* (strana poslužitelja) i upravljanje bazama i testiranje istih. *Full-stack developer* je programer koji ima znanja i iskustva za upravljanje cijelim procesom razvoja, od stvaranja korisničkog sučelja do upravljanja bazama podataka i logikom na strani poslužitelja. Programer ne treba biti *full-stack* kako bi odrađivao poslove web programiranja, već može raditi jedan od dijelova web razvoja zasebno. Povijesno, razvijanjem sve više tvrtki koje su se bavile web programiranjem, primijećena je potražnja za *frontend* programerima jer su tvrtke željele ostati ispred konkurencije razvijanjem zanimljivih, raznovrsnih web stranica sa što naprednijim korisničkim iskustvom (engl. *user experience*, UX) kako bi na prvi pogled privukle više posjetitelja i dugoročnih korisnika. Danas potreba za *backend* programerima raste eksponencijalno zahvaljujući značaju koji oni pridonose izradi funkcionalnih web aplikacija. Prema istraživanju US Bureau of Labor and Statistics, potreba za back-end programerima će strahovito rasti do 2028. godine zbog potražnje u industriji e-trgovine [12] .

Praktični dio završnog rada izveden je *full-stack*, ali uz korištenje biblioteka i razvojnih okvira koje je prvo bilo potrebno upoznati i savladati, više o tome biti će napisano u nastavku rada.

2.3. Alati i tehnologije korištene u radu

Jednostavno sučelje i aplikacija mogli su se izraditi u jednostavnoj *JavaScript* + PHP kombinaciji, no razlog odabira niže spomenutih tehnologija jest zbog činjenice da su najkorištenije trenutno u svijetu programiranja i zbog prednosti koje nose sa sobom. Primjer korištenosti *React*-a kao biblioteke *JavaScripta* može se iščitati iz slike 4. Jasno se vidi da se tijekom godina

povećavao udio korištenja *React*-a te se kroz zadnji par godina konstantno zadržao na vrhu. Također, od 2022. godine značajan je porast novih tehnologija, no u skorije vrijeme neće se pojaviti tehnologija koja bi mogla značajno smanjiti korištenje ili potpuno zamijeniti *React* [13]. Usto, prilikom budućeg zaposlenja koristio bi upravo tehnologije spomenute niže, što je bio dovoljan razlog za upotrebu.



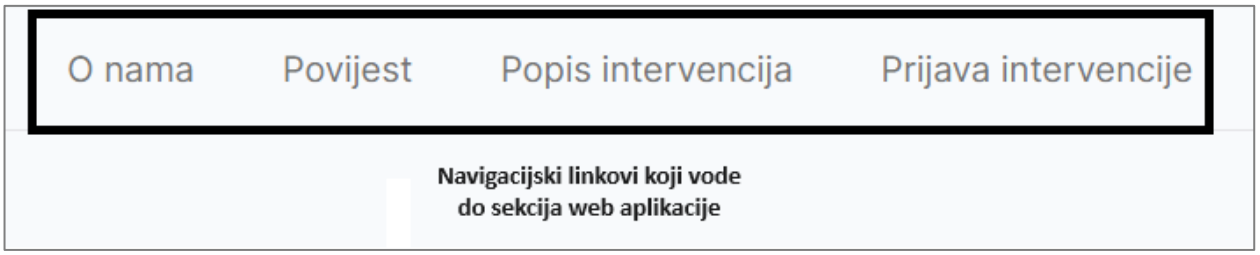
Slika 4. Zastupljenost/udio različitih JavaScript biblioteka po godinama [13]

Najveći fokus stavio sam na učenje i savladavanje *Next JS*-a i *React*-a. *Next* JS je stekao veliku popularnost u *React* ekosustavu zahvaljujući svojim značajkama prilagođenim programerima, optimizacijom performansi i fleksibilnosti pri izradi modernih web aplikacija. Za aplikaciju koja zahtjeva korištenje poslužitelja, *Next.JS* je najprikladnija opcija zbog svojih optimizacija i renderiranja sadržaja na strani poslužitelja. Zahvaljujući renderiranju na poslužitelju, *React* omogućava korisniku da prilikom dohvata stranice dobije kompletnu učitanu web stranicu što uvelike povećava korisničko iskustvo. Također, to pridonosi i korisnicima sa slabijom internetskom vezom jer se smanjuje renderiranje stranica na klijentu i nije potrebno preuzimanje i izvršavanje *JavaScript*-a. Ako web aplikacija vrši interakciju s bazom podataka, taj se proces također događa na poslužitelju (*Node* JS) i već prilikom slanja inicijalne stranice klijentu podaci će biti učitani u sam HTML kod. Na projektu se koristila *PostgreSQL* baza podataka, dok za manipulacijom baze se koristio alat *Prisma*. Više o tome biti će napisano i objašnjeno u odlomcima koji slijede.

2.3.1. *React JS*

Najpopularnija biblioteka današnjice je *React*. Osnovao ga je Jordan Warlke, programer koji je radio u tvrtki Facebook, a kasnije je *React* preuzeo *Instagram* i ugradio ga u svoje aplikacije.

Služi za izgradnju interaktivnih korisničkih sučelja (engl. *User intefaces*) [14]. Korisničko sučelje su elementi koje korisnik vidi i s kojima može vršiti interakciju, a primjer je prikazan na slici 5.



Slika 5. Primjer korisničkog sučelja projekta

Virtualni DOM (detalji u [15]), jedan od najvažnijih značajki *React* programiranja, može se opisati kao jednostavnija ili laganija verzija realnog DOM-a. Virtualni DOM nalazi se u memoriji *JavaScripta*, točnije dijelu memorije kojeg preglednici alociraju za izvršavanje JavaScript koda. Virtualni DOM poboljšava performanse tako što smanjuje broj direktnih manipulacija realnog DOM-a. Umjesto ažuriranja cijelo DOM-a kod svake promjene, kod virtualnog se ažurira samo onaj objekt kod kojeg je došlo do promjene stanja. Kada se stanje komponente promijeni, *React* kreira novi virtualni DOM na temelju promijenjene komponente i uspoređuje ga sa „starim" realnim DOM-om kako bi ustanovio gdje je došlo do promjene. Zatim, samo promjene se primjenjuju u realni DOM.

Uz virtualni DOM, bitan pojam je i JSX (detalji u [16]). JSX je sintaktička ekstenzija koja omogućuje programeru pisanje HTML koda u *JavaScript* datoteci. Isprve, poprilično neobičan način pisanja koda jer se dugi niz godina sadržaj pisao u HTML, stilovi i dizajn u CSS, a logika u *JavaScript* datoteci. Web preglednik „zna" samo HTML, *JavaScript* i CSS. Kako bi JSX bio prilagođen pregledniku React ga prevodi u JavaScript. Primjer prevođenja JSX-a je prikazan u kodu niže.

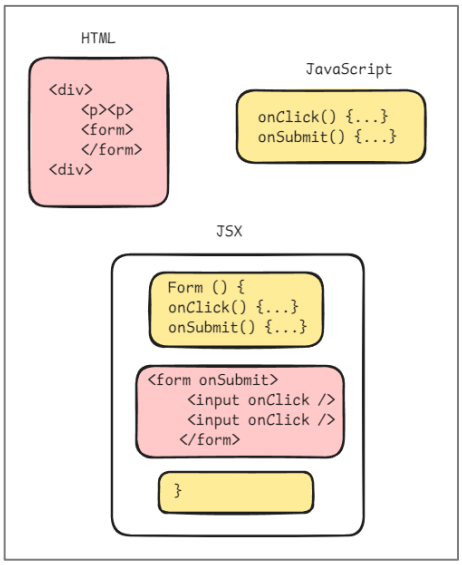
```
Function MyComponent() {
    Return <h1>naslov razine 1</h1>
}
```

Način pisanja JSX-a je sličan HTML-u, no razlika je što je pisan *JavaScript* kodom (*.js* ekstenzija). *React*, kako bi se prilagodio pregledniku, transformira JSX u „čisti" *JavaScript* i to pomoću funkcije „`React.createElement`". Gornji JSX kod *React* prevodi u slijedeći:

```
function MyComponent() {
  return React.createElement('h1', null, 'naslov razine 1');
}
```

Postoje posebni alati (npr. *Babel*) koji služe za transformiranje JSX-a u *JavaScript* kako bi web preglednik razumio napisano. Kod *Next* JS aplikacije ti alati automatski transformiranju kod tijekom razvoja ili tijekom građenja za produkciju. Kada se transformacija dovrši, preglednik

izvršava *JavaScript* kako bi renderirao pripadajuće HTML elemente u DOM. Sljedeće, primjer JSX-a gdje se vežu i HTML i *JavaScript* prikazan je na slici 6.



Slika 6. Prikaz JSX koda iz kombinacije HTML-a i JavaScripta

Kod web programiranja uz pomoć *React*-a, JSX često koristimo za definiranje nekog izraza s vitičastim zagradama „{}“ direktno u varijabli. To nam omogućava dinamičko renderiranje sadržaja ovisno o varijablama, što uvelike pomaže u programiranju gdje se želi izbjeći „zakopavanje“ koda. Također, takvo renderiranje se često koristi jer se u određenim slučajevima (ne)želi prikazati neki element na temelju nekog drugog elementa ili varijable, a to pomoću JSX-a se postiže u jednom retku. Ispod je prikazan primjer iz praktičnog dijela rada gdje se prikazuje korisniku jedna vrsta opcija u navigaciji ako je prijavljeni u aplikaciju, ako nije, prikazuje mu se druga opcija.

```
<nav>
   {useAuth().isSignedIn ? (
      <MaxWidthWrapper>
        <Link>Popis intervencija</Link>
        <Link>Statistika</Link>
      </MaxWidthWrapper>
  ) : (
      <MaxWidthWrapper>
        <SignInButton />
      </MaxWidthWrapper>
)}
</nav>
```

Radi se provjera na temelju statusa prijave korisnika. Ako je korisnik prijavljeni u aplikaciju, ima mogućnost odabira između opcija navigacije „Popis intervencija“ ili „Statistika“, dok u slučaju kada nije prijavljeni može kliknuti samo na gumb za prijavu.

2.3.2. Next JS

Next JS je jedan od najpopularnijih okvira *(engl, framework) React*-a koji može nadodati gradivne blokove za kreiranje dinamičkih i modernih web aplikacija. *Next* JS upravlja alatima i konfiguracijom potrebnom za *React* te pruža dodatnu strukturu, značajke i optimizaciju za aplikaciju. *Next* JS je poseban okvir jer omogućava programerima izradu web aplikacija koje mogu renderirati sadržaj na poslužitelju (prije slanja klijentu) ili na samom klijentu. Preduvjet za rad *Next* JS-a je *Node* JS (više o *Node* JS-u u 2.3.5.) posebno kod razvojnog okruženja gdje se pomoću *Node* JS-a pokreće lokalni razvojni poslužitelj, ali i u samoj produkciji gdje *Node* JS poslužitelj pokreće *Next* JS aplikaciju.

Ključne funkcionalnosti koje *Next* JS nudi su: *rendering, caching, automatic code splitting* i *file-based routing*, Međutim, *Next* JS je za projekt bio bitan zbog API ruta (krajnje točke API koje služe za definiranje logike poslužitelja) koje on nudi. Omogućava definiranje ruta unutar projekta, što olakšava kreaciju i upravljanje funkcionalnostima *backenda*. Konkretno, primjer korištenja API točaka na projektu je kod slanja podataka prema bazi nakon uspješnog unosa obrasca. Kad korisnik podnese obrazac, podaci se šalju prema API ruti koja u sebi sadrži logiku definiranu od strane programera (npr. slanje podataka prema bazi). Kod iz aplikacije je prikazan niže. Kod klasičnog pristupa („čisti“ *JavaScript*) za definiranje API krajnjih točaka potrebno je posložiti zasebnu *backend* strukturu (npr. *Express* JS) koju je potrebno i posebno upravljati. [17].

```
export default function createIntervention(
  req: NextApiRequest,
  res: NextApiResponse) {
      if(req.method === „POST“) {
         const data = req.body

           const result = await prisma.intervencija.create({
             data: result
         })

         res.status(200).json(result)
           } catch (error) {
             console.error(error);
             res.status(500).json({ error: „Failed to save intervention“ });
       }
```

Nakon podnošenja (engl. *submit*) obrasca, radi se „POST“ zahtjev prema API krajnjoj točki čiji je kod prikazan iznad. Podaci iz obrasca (koji dolaze s klijentske strane) se nalaze u tijelu zahtjeva (engl. *request body*). Kako bi mogli poslati te podatke prema bazi podataka, treba ih izvaditi van u posebnu varijablu (gore u kodu je to varijabla „`data“)` te je onda pomoću *Prisme* (alat koji nam služi za manipuliranje bazom podataka) moguće poslati te podatke u bazu.

Druga bitna značajka *Next* JS-a je renderiranje (engl. *rendering*). Renderiranje se u web programiranju referira na proces generiranja i prikazivanja vizualnog sadržaja web stranice u

pregledniku. Konkretno, pretvaranje koda (HTML, CSS, *JavaScript*) u sadržaj s kojim korisnici mogu vršiti interakciju. Klijent (web preglednik) šalje zahtjeve poslužitelju za kod aplikacije, dok odgovor poslužitelja pretvara u korisničko sučelje. Ono što nam *Next* JS omogućava, za razliku od „normalnog" klijentskog renderiranja jest kreiranje hibridnih web aplikacija gdje dio koda može biti renderiran na klijentu a dio na poslužitelju. Uz renderiranje na klijentu (engl. *Client-Side Rendering* – CSR) i renderiranje na poslužitelju (engl. *Server-Side Rendering* – SSR), *Next* JS još nudi generiranje statične stranice (engl. *Static Site Generation* – SSG), inkrementalno statičko regeneraciju (engl. *Incremental Static Regeneration* – ISR). Svako od navedenih renderiranja ima svoje prednosti i nedostatke, utječe na performanse i korisničko iskustvo. Kod projekta web aplikacije vatrogasne postaje, nije bilo potrebno koristiti neku od posebnih renderiranja, no za neke veće projekte ta mogućnost *Next* JS-a je jako popularna i korištena [18].

Još jedna funkcionalnost zbog koje je *Next* JS poseban jest *caching*. Proces spremanja podataka kojima se često pristupa i to u privremenu lokaciju koja se zove *cache*. Time se poboljšavaju performanse i smanjuje se upotreba resursa. Ako se nekoj web stranici pristupa često, velika je vjerojatnost da se tu koristi *cache* kako bi se ubrzalo učitavanje te stranice. *Next* JS ubrzava i poboljšava performanse aplikacije i smanjuje uporabu resursa preko *caching* mehanizama. Primjer *caching*-a može biti na već prije spomenutim API rutama. Moguće je optimizirati krajnje točke API-a tako da se primjerice smanji potreba za konstantnim upitima prema bazi. Nakon upita, spremi se rezultat API zahtjeva na određeno vrijeme i ako se u tom vremenu ponovi upit - poslužitelj vraća spremljeni odgovor [19].

Zadnja, ali ne i manje bitna značajka koju je potrebno spomenuti je usmjeravanje pomoću datoteka (engl. *File based routing*). *Next* JS u sebi ima integriran ruter (engl. *router*) koncipiran na mapi „`pages`". Ako se ubaci datoteka u mapu, automatski je dostupna kao API ruta. Primjer je gore napisani kod za dodavanje nove intervencije u bazu podataka. Kod se nalazi u datoteci s imenom „`createIntervention.tsx`" koja šalje upit prema bazi. Tu datoteku *Next* JS automatski tretira kao API rutu. Posebno je što je takvim pristupom moguće i definirati dinamičke rute. Primjerice, ako se datoteka nalazi u „`pages/api`" mapi i naziva je „`interventions/[id].tsx`", ona može dohvatiti „`id`" parametar iz URL-a (akronim od engl. *Uniform Resource Locator*) i na temelju toga može odraditi logiku za taj određeni identifikator. Dinamičke rute su popularne i često se koriste kad su u pitanju upiti prema bazi [20].

2.3.3. Prisma

Prisma je tzv. ORM (akronim od engl. *Object Relation mapping*), odnosno alat otvorenog koda koji služi za usklađivanje programskog koda s bazom podataka. Sastoji se od:

- *Prisma client*: automatski generiran alat koji služi za kreiranje upita prema bazi
- *Prisma migrate*: sistem koji služi za migraciju baze (migracija – proces prijenosa podataka između dva sustava)
- *Prisma studio*: grafičko korisničko sučelje (engl, *Graphical User Interface*, GUI) preko kojeg je moguće vidjeti i uređivati podacima u bazi

Na samom početku, *Prisma* je bila korištena kako bi se definirao model „`intervencija`“. Model reprezentira tablicu u relacijskoj bazi (*PostgreSQL*). Nakon definiranja modela, generira se *Prisma Client* na kojem se temelje, između ostalih i CRUD upiti za taj model [21].

2.3.4. PostgreSQL

PostgreSQL je relacijska baza podataka otvorenog koda. Osim što je besplatna i otvorenog koda, PostgreSQL je vrlo proširiv. Na primjer, mogu se definirati vlastiti tipovi podataka, implementirati programeru prilagođene funkcije te pisati kod iz različitih programskih jezika bez ponovnog kompiliranja baze podataka. Tablice su povezane definiranim relacijama što omogućuje kombiniranje podataka iz više tablica. Zadovoljava sve zahtjeve ACID-a (engl. *Atomicity, Consistency, Isolation and Durability properties*) što je nužno za integritet podataka. Podržava kompleksne i zahtjevnije upite, indeksiranje te dozvoljava složenije tipove podataka (npr. JSON) što ga čini svestranim. Također, učinkovito rukuje velikim količinama podataka i radnim opterećenjima [22].

Za potrebe razvoja jednostavne web aplikacije korištene su tri osnovne tablice - „Intervencija“, „Korisnik“ i „PrijavljeniKorisnik“, ali jasno je da dodavanje dodatnih mogućnosti utječe i na kompleksnost baze podataka. U tablicu „Intervencija“ spremaju se podaci intervencije koju je gost stranice unio. Tablica „Korisnik“ sprema podatke o korisniku koji je unio intervenciju. To može poslužiti kako bi se mogla povezati unesena intervencija sa osobom. S time se sprječava anonimno unošenje tablice, kao što se može i u slučaju nevaljanih podataka kontaktirati korisnika (preko adrese e-pošte ili mobitel) i ispitati točnost istih. Dok se preko tablice „PrijavljeniKorisnik“ vodi evidencija o korisnicima koji su se prijavili na web aplikaciju. Primjerice, iz tih podataka se mogu vaditi van statistike o prijavljivanju i dr.

2.3.5. Razvojno okruženje

Razvojno okruženje projekta sastoji se od više komponenata koje u sinergiji pružaju mogućnost izrade *full-stack* (klijentski i serverski dio) aplikacija. Obzirom da se radi o web

aplikaciji sve leži na HTTP poslužitelju koji je ukomponirani u *Node* JS. Kako je *Node* JS *JavaScript runtime* okruženje koje pruža izvršavanje *JavaScript* koda izvan preglednika [23], također nudi i objedinjeni jezik (*JavaScript*) za razvoj na strani klijenta i na strani poslužitelja, što može dovesti do povećane produktivnosti i mogućnosti ponovne upotrebe koda. U *Node* JS je uključeni HTTP modul koji omogućava programerima kreaciju HTTP poslužitelja zahvaljujući kojemu se mogu obrađivati nadolazeći HTTP zahtjevi. Dok se *Next* JS aplikacija oslanja na rad *Node* JS-a koji izvršava kod na strani poslužitelja uz pomoć već spomenutog HTTP poslužitelja. Konkretan primjer sinergije *Next* JS-a i *Node* JS-a je kod renderiranja stranice na poslužitelju (engl. *server side rendering*). *Next* JS omogućava programeru definiranje stranice kao klijenta ili kao poslužitelja. Ako je stranica odabrana kao poslužiteljska i sastoji se od više *React* komponenata, prilikom slanja zahtjeva prema toj stranici *Node* JS izvršava te komponente na poslužitelju kako bi generirao HTML sadržaj. Tek nakon generiranja sadržaj se šalje prema klijentu, odnosno njegovom pregledniku. Druga mogućnost Node JS-a i Next JS-a, koja je bila korištena na projektu za slanje upita prema bazi podataka su API rute. Svaka API ruta je samostalni modul koji se pokreće na poslužitelju *Node* JS-a kada se pozove. U suštini, API rute su funkcije koje *Node* JS procesuira, izvršava kod na poslužitelju (npr. upiti prema bazi podataka) te na kraju vraća JSON tip podataka prema klijentu. *Next* JS ima formiranu takvu strukturu datoteka i mapa da svaka datoteka unutar mape „`pages/api`“ se mapira i tretira kao kranja API točka umjesto „normalne“ stranice. API ruta koja je korištena kako bi se prikazale sve intervencije koje se nalaze u *PostgreSQL* bazi podatka naziva se „`getInterventions`“ te je prikazana ispod.

```
export default async function handler(
  req: NextApiRequest,
  res: NextApiResponse
) {
  if (req.method === "GET") {
    try {
      const interventions = await prisma.intervencija.findMany();
      res.status(200).json(interventions);
    } catch (error) {
      res.status(500).json({ error: "Error fetching interventions" });
    }
  } else {
    res.setHeader("Allow", ["GET"]);
    res.status(405).end(`Method ${req.method} Not Allowed`);
  }
}
```

Funkcija iznad zorno prikazuje povezanost HTTP poslužitelja i Next JS-a. Točnije, “`Req`” je primjer „`http.IncomingMessage“` objekta kojeg je kreirao HTTP.server te ga prosljeđuje kao prvi argument događajima (engl. *event*) „`request`“ i „`response`“. Može se koristiti za pristup statusu odgovora, zaglavlju ili podacima. Na gornjem primjeru, API ruta vraća JSON odgovor (sve intervencije u bazi podataka) s kodom statusa „`200“`. Dok je „`Res`“ primjer

„`http.ServerResponse`“ objekta. HTTP poslužitelj ovaj objekt kreira interno te se prosljeđuje kao drugi parametar događaju „`request`“.

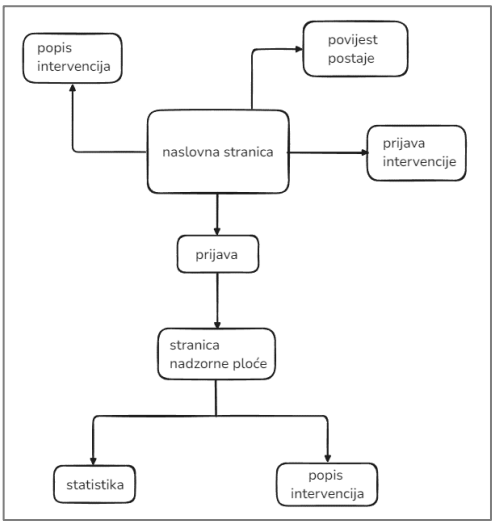
Bitno je napomenuti da se kod korištenja *Node* JS-a nije potrebno koristiti klasičnim web serverom (npr. *Apache*). Iako se tijekom školovanja koristio *Apache* kao web poslužitelj, na projektu je odabrana verzija s *Node* JS-om. Web aplikacija je dinamička, koristi dosta komuniciranja s bazom, pa uz to i prikazivanje dinamičkog sadržaja te je zbog svojih prednosti bio izabran *Node* JS. Dok u produkciji, *Node* JS i *Apache* mogu raditi zajedno. Točnije, *Node* JS može funkcionirati kao samostalni web poslužitelj – slušati nadolazeće HTTP zahtjeve i procesuirati ih preko zadanog usmjeravanja aplikacije (API ruta). No, zbog sigurnosti, skalabilnosti i još većih performansi *Node* JS poslužitelj se postavlja iza tzv. *reverse proxy* (poslužitelj, aplikacija ili servis u „oblaku“ koji se nalazi ispred web poslužitelja kako bi presreo i provjerio dolazne zahtjeve klijenta prije nego ih proslijedi web poslužitelju i naknadno vrati poslužiteljev odgovor klijentu), a to može biti upravo *Apache*. No, *Node* JS aplikacija se tipično postavlja (engl. *deploy*) na javni poslužitelj

2.3.6. CSS okvir

Sam CSS kao takav je dovoljan za izgradnju elementarnih web aplikacija, no postoje načini i alati kako si programer može olakšati posao i više staviti u fokus logičku stranu izgradnje, dok ovu „dizajnersku“ stranu može nadoknaditi preko tih alata. Za složenije animacije i izgleda aplikacije vrlo je mukotrpno i teško koristiti bazični CSS. Iz tog razloga postoje CSS okviri, a neki od njih su *Bootstrap, Tailwind* CSS, *Semantic* UI, *Materialize, Bulma* i dr. Tijekom pohađanja kolegija „Osnove web programiranja“ koristili smo *Bootstrap* CSS okvir. Prema stranici https://2023.stateofcss.com/en-US/css-frameworks/ *Bootstrap* je zapravo i najkorišteniji okvir u zadnjih pet godina, dok je drugi najkorišteniji okvir *Tailwind. Tailwind* CSS je izabran zato jer se kod potencijalnog poslodavca koristi pa sam želio dopuniti znanja o istome. Također, *Tailwind* je izabran i iz razloga što ima dobro pokriven responzivni dizajn pa je zahvaljujući njemu web aplikaciju vatrogasne postaje moguće koristiti na svim uređajima; mobitelima, tabletima i korisničkim računalima te će se elementi na aplikaciji poslagivati ovisno na kojem uređaju se koristi. Funkcionira na način da postoji jedna globalna CSS datoteka u kojoj su napisane klase (engl. *classes*). Loša stvar *Tailwind*-a, zbog kojeg ga neki programeri izbjegavaju, jest što kod postaje zatrpan ako se na svaki element dodaje hrpa klasa – kod postaje manje čitljiv [21].

3. Planiranje i razvoj web aplikacije

Ideja je bila izraditi jednostavnu, ali korisnu aplikaciju jedne fiktivne vatrogasne postaje. Aplikacija se bazira na temelju primjera javnih vatrogasnih postrojba Hrvatske, a k tome imao sam uvid i u rad lokalne vatrogasne postaje te sam zaključio da bi svako dobrovoljno vatrogasno društvo ili javna vatrogasna postaja (obavezno) trebala imati web stranicu na kojem bi objavljivala intervencije i na taj način informirala građane. Vođen time, cilj web aplikacije je praćenje intervencija, ali i dodatna mogućnost u vidu obrasca preko kojega gost stranice može unijeti podatke o intervenciji i poslati prema bazi. Naravno, ovakva aplikacija nije nužna za rad vatrogasne postaje, no daje još jednu mogućnost obavještavanja građana. Web aplikacija je jednostavnijeg karaktera, sadrži jednostavan raspored elemenata kako bi gostu aplikacije bilo intuitivno snalaziti i kretati se njome. Također, bitno je dati gostu jasnu poruku što web aplikacija predstavlja –dobiveno preko opcija navigacije i jasnog i kratkog naslova i podnaslova. Motivacija za raspored elemenata dobivena je pregledom više web stranica vatrogasnih postaja. Svaka veća vatrogasna postaja ima svoju web stranicu (aplikaciju) na kojoj se nalaze ili informacije o radu ili dinamički elementi kao što su obrasci ili mogućnosti prijave. Shodno tome, projekt je planiran da ima i statički dio (informacije o vatrogasnoj stanici ili popis intervencija) i dinamički dio (obrazac za popunjavanje, prijava korisnika). Niže je prikazana struktura (model) web aplikacije. Povijest postaje, popis intervencija i prijava intervencije su dijelovi naslovne stranice, dok se za prijavu u aplikaciju korisniku otvara prozor koji ga nakon uspješne prijave vodi na stranicu nadzorne ploče. Iz stranice nadzorne ploče korisnik može otići na jednu od dvije stranice; statistika ili popis intervencija. Konkretno, naslovna stranica je standardna stranica, dok su segmenti vezani uz funkcionalnost aplikacije prijava korisnika i rad s evidencijom intervencija. Programiranje stranica i dijelova aplikacije je moguće izvesti na više načina. Standardno, kao aplikacija sa više stranica (engl. *Multi-Page Applications* – MPA) ili načinom koji je postao popularan zadnjih godina aplikacija s jednom stranicom (engl. *Single-Page Applications* – SPA). *React* je popularna biblioteka za izradu aplikacija s jednom stranicom zahvaljujući internom renderiranju i arhitekturi koja je koncipirana na komponentama, no u slučaju web aplikacije vatrogasne postaje korišten je klasični način s više stranica (MPA). SPA pristup se koristi za aplikacije kojima je bitna skalabilnost, brža navigacija i najviše korisničko iskustvo jer kod aplikacija s jednom stranicom nema učitavanja stranica pa su brže. Plan je i isprobati SPA pristup na istoj aplikaciji kako bi stekao i to iskustvo programiranja za potrebe poslodavca



Slika 7. Ideja navigacije unutar web aplikacije

Plan – sučelje i funkcionalnosti

Osim funkcionalne web aplikacije ideja je bila ponuditi korisnicima i *landing* stranicu s osnovnim informacijama o vatrogasnoj postaji (slika 7.). Dizajn je kreiran po uzoru na realne primjere te od vlastitog nahođenja. Odabran je jednostavan ali intuitivan izgled: izbornik na vrhu, veliki logo vatrogasne stanice i par značajnih naslova i podnaslova. Prvenstveno zbog toga što stranica ne treba biti kompleksna i nije cilj da se korisniku pokažu sve mogućnosti weba, već bi to trebala biti korisna web aplikacija na koju bi korisnik sletio s razlogom. Zbog jednostavnosti web aplikacije sama izrada dizajna nije bila kompliciranа te ga je moguće napraviti i bez znanja dizajniranja web aplikacija. Cilj *landing* stranice projekta je da korisniku pobliže prikaže vatrogasnu postaju, kratku povijest stanice, pregled intervencija kroz godinu te na kraju da mogućnost korisniku prijavu intervencije preko obrasca.

Landing stranica je bitna jer je to prvo što korisnik vidi nakon klika linka koji otvara web aplikaciju. Na njoj bi se na početku trebale nalaziti osnovne informacije o samoj stranici i izbornik pomoću kojeg se korisnici mogu brže kretati po web aplikaciji. Stil bi trebao biti primamljiv, uredan i čitljiv jer je i to jedan od načina zadržavanja korisnika na aplikaciji. Idealno, trebala bi sadržavati [26]:

- **Jasan naslov** - koji treba sadržavati jednu od ključnih riječi aplikacije.
- **Podnaslov** - potpora glavnom naslovu te dodatno pojašnjava bit aplikacije
- **Vizuali** - fotografije ili videozapisi, s obaveznom potporom tekstu jer slika vrijedi tisuću riječi

- **Cilj stranice – CTA (akronim od engl. *Call To Action*),** web aplikacija treba sadržavati poziv na akciju korisnika
- **Web obrazac** - idealno za web aplikaciju jer preko njega se dobivaju nove informacije i puni se baza podataka interakcijom korisnika s aplikacijom

3.1. Razvoj web aplikacije

Uz već opisano razvojno okruženje u odjeljku 2.3.5., također je potreban i uređivač koda. Uređivač koda korišten na projektu web aplikacije vatrogasne postaje je *Visual Studio Code*. Ima mogućnost dodavanja ekstenzija koje pomažu pri pisanju koda (npr. ekstenzija za *Prismu* omogućava isticanje sintakse i dopunjavanje koda). Upravo zbog tih ekstenzija i ostalih mogućnosti u usporedbi sa nekim bazičnim uređivačem je toliko popularan za pisanje koda. Za inicijalizaciju i izradu *Next* JS projekta postoji jasno definiran hodogram (više u [24]) kojega je potrebno slijediti. Također, za autentifikaciju korisnika korišteni su elementi platforme *Clerk* koja pruža gotova, besplatna rješenja za prijavu i registraciju korisnika.

3.1.1. Inicijalizacija projekta

Struktura generičkog projekta sa svim datotekama u njemu prikazan je na slici 8.

```
web-vatrogasna-postaja
|-- node_modules
|-- public/
|   |-- images
|   |-- svg
|-- src/
|   |-- app/
|   |   |-- layout.tsx
|   |   |-- page.tsx
|   |   |-- globals.css
|   |-- components
|   |-- pages/
|   |   |-- api
|-- .env
|-- .env.local
|-- next.config.js
|-- package.json
|-- tsconfig.json
```

Slika 8. Struktura generičkog Next JS projekta nakon inicijalizacije

Objašnjenja svih datoteka koje se nalaze u projektu nakon inicijalizacije prikazana su u tablici 4.

Tablica 4: opis datoteka koje su na početku svakog Next JS projekta.

Naziv datoteke	Objašnjenje funkcionalnosti
node_modules	sprema vanjske biblioteke i osnovne pakete (engl. dependencies) koji pridonose dodatne alate za projekt
public	Statički elementi aplikacije (npr. slike ili svg)
src	izborna (programer sam odlučuje želi li ju dodati ili ne prilikom inicijalizacije projekta) izvorna mapa projekta
src/app	tzv. *App Router* – služi za organizaciju ruta, stranica i cjelokupnog rasporeda datoteka
src/app/page.tsx	korijenska React komponenta – renderira glavni sadržaj aplikacije (kada korisnik želi otvoriti web aplikaciju vatrogasne postaje, ova stranica mu se otvara)
src/components	Mapa koja sadržava sve proizvoljne komponente programera (programiranje s *React*-om se temelji na programiranju s komponentama koje zajedno grade jedan blok)
src/pages/api	Mapa koja služi za definiranje API ruta čija je namjena komunikacija s bazom podataka
.env	Varijable okoline (engl. *environment variables*) koriste se za pohranu i upravljanje konfiguracijskim vrijednostima koje su specifične za različita okruženja (razvoj, proizvodnja ili testiranje)
.env.local	Lokane varijable okoline (engl. *local environment variables*) – pomažu u sigurnom upravljanju osjetljivim informacijama kao što su ključevi API-a, niza znamenki (engl. connection strings) potrebnih za komunikaciju s bazom podataka i druge postavke koje ne bi trebale biti tvrdo kodirane (engl. *hard-coded*) u kodu
next-config.js	Konfiguracijska datoteka za *Next* JS
package.json	Skripte projekta
tsconfig.json	Konfiguracijska datoteka za *TypeScript*

Najbitnije bazične datoteke potrebne za razvoj web aplikacije su datoteka s komponentama, „`page.tsx`“ datoteka i mapa za komunikaciju s *backendom*, odnosno bazom - „`pages/api`“.

3.1.2. Izrada naslovne stranice

Kako je *React* baziran na komponentama koje se mogu uključiti u jednu datoteku i tvoriti cjelinu, tako sam i ja krenuo implementirati komponente koje ću kasnije uvesti u korijensku datoteku (*page.tsx*). Prva komponenta koju sam krenuo raditi bila je „`MaxWidthWrapper`“ (proizvoljnog naziva) koja je korištena preko cijele aplikacije. Za proizvoljne komponente koje programer razvija je uvijek potrebno dati kvalitetno ime koje što više opisuje rad iste. Komponente

koje programer izradi mora se nalaziti u „`Components`“ mapi. Svaka komponenta u React-u je u suštini funkcija koja vraća *React* element u obliku JSX-a. Primjer jednostavne komponente:

```
const HeadingTwo = () => {
  return (
    <h2>Naslov razine 2</h2>
  )
}

export default HeadingTwo
```

Funkcija (komponenta) može sadržavati svojstva (engl. *props*) koja služe za slanje podataka od elementa više razine hijerarhije prema nižemu (tzv. *parent – child* veza). Komponenta koja prima svojstva ne može ih mijenjati. Primjer gdje se tekst naslova dva dobiva od komponente veće razine hijerarhije:

```
const HeadingTwo = ({ headingText }) => {
  return (
    <h2>{headingText}</h2>
  )
}

export default HeadingTwo
```

Lista komponenata koje sam izradio i bile su potrebne na projektu su:

- `AdminDialog.tsx`
- `AdminUserDialog.tsx`
- `BackToTopButton.tsx`
- `Columns.tsx`
- `DataTable.tsx`
- `DataTablePagination.tsx`
- `DataTableRowActions.tsx`
- `EmailItem.tsx`
- `Footer.tsx`
- `Hero.tsx`
- `Icons.tsx`
- `InterventionForm.tsx`
- `MaxWidthWrapper.tsx`
- `Navbar.tsx`
- `NavbarLinks.tsx`
- `Pictures.tsx`
- `ResponsiveDialog.tsx`
- `ToastButton.tsx`

Osnovna funkcionalnost komponente „`MaxWidthWrapper`“ je da svakom sadržaju koji se nalazi unutar nje daje određene stilove – konkretno, razmak od rubova na različitim veličinama

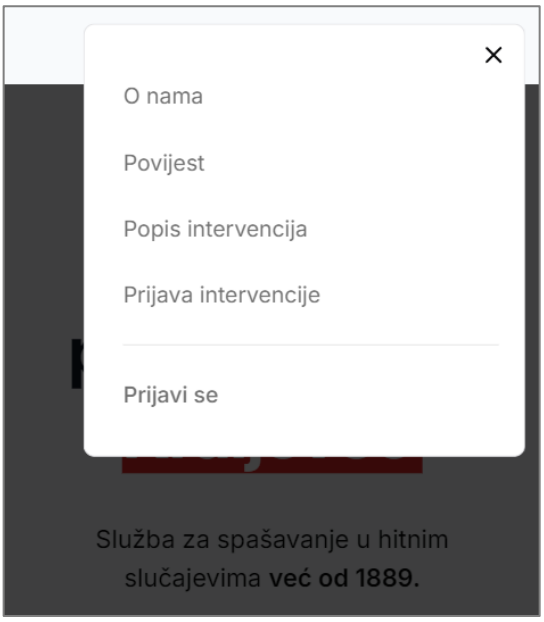
ekrana kao i visinu i širinu. Time se eliminira bespotrebno ponavljanje istog koda za svaku stranicu, komponentu ili element (ušteda vremena i resursa – cilj *React*-a). Izrada same naslovne stranice teče tako da se izrade više komponenata (ako je to potrebno) koje predstavljaju jednu cjelinu (npr. navigaciju). Cilj toga je da se izbjegne gomilanje koda u jednu datoteku i lakše pronalaženje grešaka. Prva implementacija na samoj naslovnici je navigacija. Za navigaciju je kreirana posebna komponenta (proizvoljnim imenom „`Navbar`“) koja u sebi sadrži određene *Tailwind* CSS klase (npr. elementi poredani horizontalno jednog do drugog s međusobnim razmakom). Proces uključivanja komponente na stranicu je sljedeći; U datoteku gdje se nalazi naslovna stranica („`page.tsx`“) uključimo (engl. *import*) komponentu tako da joj damo naziv, a zatim njezino mjesto u mapi gdje je izrađena. Prikaz koda koji uključuje „`Navbar`“ komponentu na stranicu je prikazan ispod.

```
import Navbar from "../components/Navbar";
```

Nakon import direktive, „`Navbar`“ se referencira na komponentu koja se nalazi u „`Components`“ mapi. Nakon toga, korištenje te komponente je jednostavno kao što je i korištenje osnovnih elemenata.

```
<Navbar>
     *sadržaj unutar te komponente*
</Navbar>
```

Sve druge komponente slijede ovaj proces uključivanja u datoteku i korištenja. Za mobilnu navigaciju je korištena gotova komponenta dijalog (engl. *dialog*) koji u sebi sadrži modal, odnosno skočni prozor (engl. *pop-up window*) koji preuzima fokus na sebe tako što blokira interakciju korisnika s drugim elementima stranice. Najčešće se koristi kako bi se prikazala navigacija na mobilnom uređaju, a prikazan je ispod na slici 9.



Slika 9. Navigacija na mobilnom uređaju

Navigacija sadrži četiri linka (veze) koji, nakon odabira korisnika, vode do određenih sekcija naslovne stranice. Uz to, navigacija sadrži i gumb s kojim se korisnik može prijaviti u aplikaciju. Nadalje, kako je navedeno i prije, naslovna stranica je podijeljena na sekcije pri čemu je svaka sekcija obgrljena „`MaxWidthWrapper`“ komponentom radi stiliziranja. Prilikom izrade naslovne stranice, uobičajeno je na samom vrhu imati veliki naslov i podnaslov, pa tako i u projektu stoji naslov „Vatrogasna postaja Donji Kraljevec“ uz logotip vatrogasne postaje kako bi se korisniku jasno dalo do znanja tematika web aplikacije. Za taj vrh naslovne stranice korištena je zasebna komponenta „Hero“ (također proizvoljni naziv), a kod je prikazan ispod.

```
<MaxWidthWrapper className="pb-24 pt-10 sm:pb-32 lg:gap-x-0 xl:gap-x-8 lg:pt-24
xl:pt-32 lg:pb-52 h-screen">
      <div
        className="px-6 lg:px-0 lg:pt-4"
      >
        <div className="mx-auto my-16 text-center lg:text-left flex flex-col items-
center justify-center lg:flex lg:flex-row-reverse">
          <div className="w-60 hidden lg:block">
            <Image
              src="/images/logo-transparent.png"
              alt="logo vatrogasne postaje Donji Kraljevec"
              width={200}
              height={200}
            />
          </div>
          <div>
           <h1 className="tracking-tight text-balance font-bold !leading-tight text-
gray-900 text-5xl md:text-6xl lg:text-7xl">
              Vatrogasna postaja{" "}
              <span className="bg-red-600 px-2 text-white">
                Donji Kraljevec
              </span>
            </h1>
           <p className="mt-8 text-lg lg:pr-10 max-w-prose text-center lg:text-left
text-balance md:text-wrap">
```

```
            Služba za spašavanje u hitnim slučajevima{" "}
            <span className="font-semibold">već od 1889.</span>
          </p>
        </div>
      </div>
    </div>
  </MaxWidthWrapper>
```

Primjer HTML koda u pregledniku kada se gleda naslovna stranica, odnosno gore prikazani kod je prikazan ispod na slici 10. Dok je sami izgled u web pregledniku prikazan na slici 11.

Slika 10. HTML kod u pregledniku za gore navedenu sekciju – „Hero“

Slika 11. Izgled sekcije „Hero“ u web pregledniku

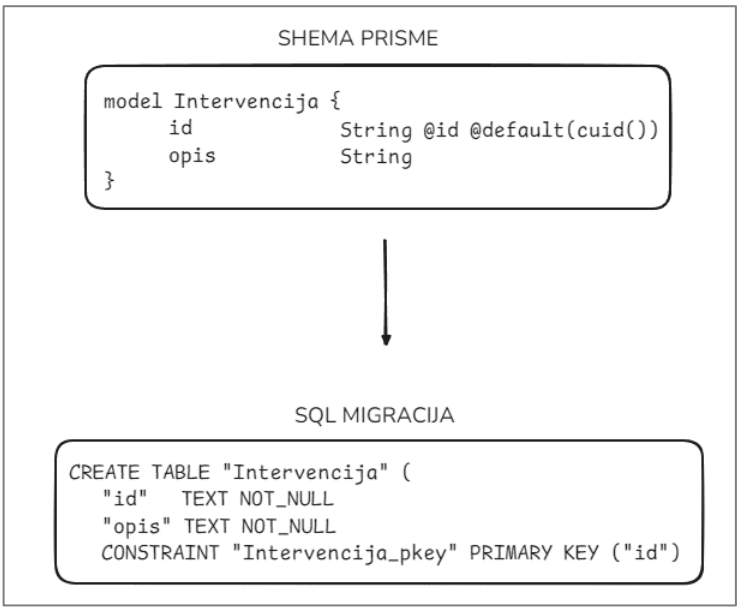
Za ostale sekcije je proces isti – izrađene kao komponente koje se uključuju na naslovnu stranicu. Cilj takvog programiranje je radi lakše organizacije koda, lakšeg održavanja i skalabilnosti. Također, idealna struktura komponenata mora biti takva da svaka odrađuje jedan dio posla i uključuje svoje stilove. Sve to kako bi se lakše snalazili u kodu i otklonili greške (engl. *debugging*).

3.1.3. Baza podataka

Baza podataka korištena na aplikaciji je *PostgreSQL*. Tijekom studiranja, na kolegijima su bile obrađene i *Oracle*, *MySQL* te *MariaDB* baze. Zbog jednostavnosti projekta, odnosno baze, svaki od spomenutih primjera bio bi prikladan za korištenje. Također, integracija s alatom *Prisma* je moguća sa svakom od spomenutih baza. Prema stranici https://db-engines.com/en/ Oracle je trenutno najpopularnija baza, slijedi ga *MySQL*, dok je *PostgreSQL* „tek“ na četvrtom mjestu. Razlog odabira *PostgreSQL* za projekt vatrogasne postaje bilo je samoinicijativno. No, *Prisma* i *PostgreSQL* imaju snažan ekosustav te se često preferira ta kombinacija za nove projekte zbog kvalitetnih značajki i veliku podršku zajednice. Također, migracijski alat *Prisme* (*Prisma migrate*) radi besprijekorno sa svim spomenutim bazama, međutim ponašanje varira od baze do baze. Poznato je da s *PostgreSQL* bazom ima najmanje komplikacija uz najsnažnije iskustvo migracije. Također, za veće projekte *PostgreSQL* nudi napredne tipove podataka (npr. „`ARRAY`“ ili „`JSONB`“) i mogućnost za kompleksnije upite. U konačnici, pri odabiru baze podataka odluka bi se trebala temeljiti na specifičnim zahtjevima projekta, značajke koje su potrebne te poznavanje baza podataka programera ili tima programera.

Bitno je napomenuti kako za ovakvu bazu s tri jednostavne tablice nije potrebno puno zahtjevnih alata ili naredba. Ovo je više primjer povezanosti s drugim tehnologijama te rada cijelog „ekosustava“ što itekako dolazi do značaja na većim, zahtjevnijim projektima.

Za definiranje sheme, upravljanjem bazom, migracijama koristi se *Prisma*. Besplatan alat koji radi na principu objektno relacijskog mapiranja (akronim od engl. ORM), odnosno to je metoda pretvaranja podataka u oblik s kojim može raditi aplikacija (sustav). *Prisma* dobiveni kod prevodi (mapira) u SQL naredbu. Slika 12. ispod prikazuje generiranje SQL naredbe za kreiranje tablice „`Intervencija`“ na temelju modela pod istim imenom. Skica je napravljen po uzoru na rad *Prisme* konkretno na projektu.



Slika 12. Primjer sheme Prisme – buduće tablice u bazi podataka

Kod korištenja *Prisme* povezivanje s bazom vrši se pomoću *Prisminog* klijenta (engl. *Prisma client*). *Prisma* generira klijent, u suštini skup metoda temeljenih na shemi (engl. *Prisma schema*). U projektu se shema nalazi u mapi „`prisma`“ s nazivom „`schema.prisma`“, gdje se i nalazi klijent *Prisme*. Shema je centralna konfiguracijska datoteka korištena za definiranje modela te opisom konekcije s bazom podataka. Izgled koda za definiranje modela i konekcije s bazom u datoteci sheme nalazi se ispod.

```
 datasource db {
  provider  = "postgresql"
  url       = env("POSTGRES_PRISMA_URL") // uses connection pooling
  directUrl = env("POSTGRES_URL_NON_POOLING") // uses a direct connection
}
```

Dio bloka koji određuje kako se *Prisma* povezuje na odabranu bazu podataka. Također, specificira se baza (npr. *PostgreSQL*, *MySQL*, *SQLite*) i niz znakova za povezivanje (engl. *connection string*) kojeg se obično definira pomoću varijable okruženja (engl. *environment variable*) u „`.env`“ datoteci.

```
model Intervencija {
  id                String   @id @default(cuid())
  opis              String
  datum             DateTime
  vrsta             String
  mjesto            String
  datumObjavljivanja DateTime @default(now())
}
```

Shema *Prisme* omogućava definiranje tablica baze podataka kao modela. Jedan model odgovara jednoj tablici u bazi i svako polje unutar modela odgovara stupcu u toj tablici.

Specificiraju se tipovi podataka za svako od polja, ograničenja (npr. '`@id`' za određivanje primarnog ključa), zadane vrijednosti i drugi atributi.

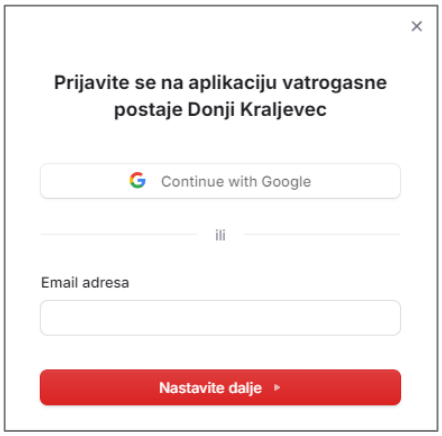
Za rukovanje s „`select`“, „`insert`“, „`delete`“ i drugim operacijama također se koristi klijent *Prisme*.

```
const interventions = await prisma.interventions.findMany();
```

Kod ispisan gore prikazuje upit „`findmany()`“ koji vraća sve podatke koji se nalaze u tablici „`interventions`“ zapisane u bazi. Klijent *Prisme* vrši upit na način da vrši interakciju s tzv. *Prisma Query Engine* komponentom. Komponenta koja je pisana jezikom *Rust* i izvršava SQL upit prema bazi podataka. Točnije, komponenta prevodi odabranu metodu (npr. „`select`“) koja dolazi od klijenta *Prisme* u SQL upit, šalje taj upit prema bazi, dobiva natrag rezultat kojega šalje prema aplikaciji u strukturiranom formatu.

3.1.4. Autentifikacija korisnika

Zadnja od opcija navigacije nudi korisniku prijavu na web aplikaciju čime se otvaraju nove opcije i mogućnosti. Prikaz sučelja za prijavu gosta prikazano je na slici 13.



Slika 13. Prikaz sučelja za prijavu korisnika

Za logiku prijave korištena je platforma *Clerk*. Popularna platforma jer nudi gotova rješenja u obliku komponenata za registraciju i prijavu. Prije korištenja *Clerk* komponenata u radu, potrebna je bila registracija na službenoj stranici te je potrebno konfigurirati novi projekt na upravljačkoj ploči *Clerk*-a. Upravljačka ploča *Clerk*-a prikazana je na slici 14. Ima mogućnosti kao što su dodavanje organizacije (dijeli aplikaciju na uloge unutar te organizacije), konfiguraciju prijave i registracije (odabira načina prijave, dozvole i ograničenja pri prijavi), daje uvid o nedavnim prijavama ili registracijama, aktivnim korisnicima, broj prijava, broj registracija i ukupan broj korisnika kao što se može vidjeti na slici 14. *Clerk* nudi gotove komponente koje je moguće

implementirati u projekt samo je potrebno dodatno stilizirati po potrebama programera. Kako bi mogli uključiti komponente, u projekt je potrebno instalirati *Clerk*-ov paket za razvoj programa – SDK (akronim od engl. *software development kit*). Komandom kojom se instalira taj paket:

```
npm install @clerk/nextjs
```

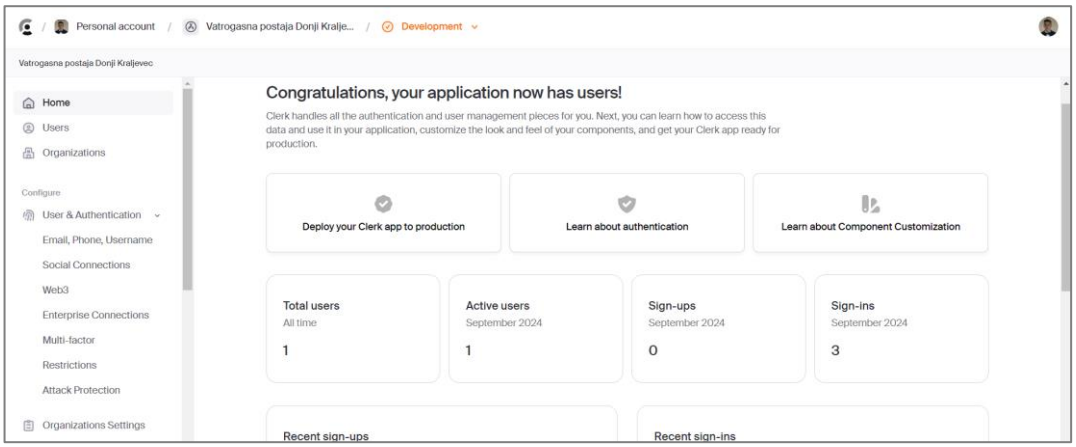
Zatim, svaku komponentu je potrebno dodati pojedinačno (po potrebi programera) s komandom koja se piše na samom vrhu dokumenta. U radu je korištena gotova *Clerk* komponenta „`SignInButton`“ za prijavu korisnika s određenom konfiguracijom specifično za projekt. Komanda s kojom uključujemo tu komponentu u datoteku:

```
import { SignInButton } from "@clerk/nextjs";
```

“`SignInButton`” komponenta dolazi s već generičkim stilovima. Kako bi se prilagodili aplikaciji, unutar te komponente doda se “`Button`” komponenta (inicijalno dostupna) i klase sa stilovima po želji programera.

```
<SignInButton mode="modal" fallbackRedirectUrl="/dashboard">
   <Button className="p-0 text-base font-base text-neutral-500 duration-200
dark:text-neutral  -400 dark:hover:text-white hover:cursor-pointer hover:text-gray-
900">
     Prijavi se
   </Button>
</SignInButton>
```

Za vatrogasnu stanicu predviđena je prijava samo određenim korisnicima, odnosno vatrogascima te vatrogasne postaje. Vanjske prijave nisu moguće, kao ni registracija korisnika. To je moguće jer *Clerk* nudi opciju „popis dopuštenih“ (engl. *allowlist*) koju administrator može uključiti i konfigurirati. Opcija radi na principu identifikatora – administrator može unijeti adresu (broj mobitela ili korisničko ime, ovisnosti o tome koja metoda je odabrana pri prijavi) npr. ime_prezime@gmail.com u popis dopuštenih. Pri prijavi korisnika, *Clerk* prolazi tom listom i ako nema unesene e-mail adrese, gostu se javlja greška na temelju nepoznate e-mail adrese. U suprotnom, pri uspješnoj prijavi, korisnika se preusmjerava na kontrolnu ploču web aplikacije vatrogasne postaje.



Slika 14. Upravljačka ploča Clerk platforme

Administrator upisuje e-mail adrese korisnika kojima odobrava prijavu u aplikaciju. Zamišljeno je da administrator koji upravlja web aplikacijom vatrogasne postaje odobri prijavu (upiše njihove e-mail adrese) svim osobama koji su dio te vatrogasne postaje. Zbog toga je registracija na web aplikaciju isključena kako se ne bi mogli prijaviti gosti koji nisu dio vatrogasne postaje. Uz to, *Clerk* nudi mogućnost prijave gostima preko raznih društvenih veza (npr. *Google*, *Facebook*, *Apple*, *Github* i dr.) naravno, preko adresa e-pošte koje se nalaze na listi. Na web aplikaciji je prikazana prijava preko *Google*-a uz pomoć SSO-a (akronim od engl. *Single Sign-On*). U suštini, to je metoda identifikacije koja omogućava korisnicima prijavu na više različitih web aplikacija i servisa uz pomoć jednog seta akreditacije. *Clerk* platforma i njezine komponente su besplatne za korištenje prilikom razvoja web aplikacije, no kod javne web aplikacije gore spomenuti „popis dopuštenih“ (uz još nekolicinu komponenata) je potrebno zasebno kupiti ako se želi koristiti. Nedostatak korištenja *Clerk*-a je sigurnost podataka. Podaci (lista prijavljenih korisnika i podaci o svakom korisniku) se spremaju na server *Clerk*-a. Ako je na web aplikaciji prioritet kontrola podataka, prilagođavanje i minimiziranje ovisnosti o trećim stranama bolje rješenje je napraviti svoju bazu podataka. No, iako to predstavlja negativnu stranu, *Clerk* u sebi ima dovoljno snažne sigurnosne mjere usklađene s propisima kao što su GDPR, HIPAA ili CCPA [25] što može biti složeno i može zahtijevati puno resursa za vlastitu implementaciju i održavanje (posebno za manje projekte).

3.1.5. Unos intervencija

Na samom kraju naslovne stranice, gost stranice ima mogućnost unosa intervencije. Za unos intervencije je izrađena posebna komponenta „InterventionForm“. Obrazac i validacija su kreirane pomoću kombinacije alata *React Hook Form* i *Zoda*. Validacija je rađena pomoću *Zoda* zbog jednostavne implementacije. Radi na principu sheme, konkretno, definira se objekt koji u sebi sadrži imena polja obrasca kojima su pridružene opcije validacije. Kod sheme iz projekta prikazan je ispod.

```
const formSchema = z.object({
  opis: z.string().min(1, "Opis intervencije je potrebno obavezno unijeti!"),
  datum: z.date({
    required_error: "Datum intervencije je potrebno obavezno unijeti!",
  }),
  vrsta: z.string().min(1, "Vrstu intervencije je potrebno obavezno unijeti!"),
  mjesto: z
    .string()
    .min(1, "Mjesto intervencije je potrebno obavezno unijeti!"),
  ime: z.string().min(1, "Ime korisnika je potrebno obavezno unijeti!"),
  prezime: z.string().min(1, "Prezime korisnika je potrebno obavezno unijeti!"),
  email: z
    .string()
```

```
    .min(1, "Email korisnika je potrebno obavezno unijeti!")
    .email({ message: "Potrebno je unijeti pravilan e-mail račun!" }),
  mobitel: z
    .string()
    .min(1, "Broj mobitela korisnika je potrebno obavezno unijeti!"),
});
```

„z“ je objekt koji uz sebe veže parametre koje možemo postavljati kao uvjete validacije. Konkretno, za obrazac na aplikaciji postavljen je uvjet da ni jedno polje ne smije ostati prazno prilikom podnošenja obrasca, a to se postiže tako što *zod* provjera broj znakova u polju – ako je manji od jedan (znači da je prazno) javlja grešku korisniku koju programer sam definirana poslije minimalnog broja znakova (1). Dok su za izradu obrasca i potrebnu funkcionalnost dovoljne osnovne komponente obrasca („`form`“, „`input`“, „`label`“), one često zahtijevaju puno ponavljajućeg koda za upravljanje stanjem, validacijom i logikom podnošenja (engl. *submit*). Bibiloteka *React Hook Form* nudi učinkovitiji, skalabilniji i održiviji način za izradu obrasca u *Next* JS aplikacijama. Bitna stavka kod *Next* JS-a je renderiranje koje je opisano u odjeljku 2.3.2. Ako aplikacija ima puno nepotrebnih renderiranja, usporava se rad te aplikacije i smanjuje se dobro korisničko iskustvo. *React Hook Form* biblioteka efikasno renderira obrazac – pokreće ponovno renderiranje samo za polja koja se mijenjaju. Kod za generiranje obrasca je prikazan ispod.

```
  const form = useForm<FormData>({
    resolver: zodResolver(formSchema),
    defaultValues: {
      opis: "",
      datum: new Date(),
      vrsta: "",
      mjesto: "",
      ime: "",
      prezime: "",
      email: "",
      mobitel: "",
    },
  });
```

Prvo se definira varijabla „`form`“ koja predstavlja objekt u kojem su definirana sva imena polja i njihov tip. Uz to, navede se i shema („`formSchema`“) po kojoj će *zod* pratiti pravilne unose u polja obrasca. Nakon toga, može se kreirati obrazac. Ispod je prikazan obrazac s jednim tekstualnim poljem.

```
<Form {...form}>
        <form
          className="…"
          onSubmit={form.handleSubmit(handleSubmit)}
        >
          <FormField
            control={form.control}
            name="opis"
            render={({ field }) => (
              <FormItem>
                <FormLabel>Kratki opis</FormLabel>
```

```
                <FormControl className="mt-2">
                  <Input className="…" {...field} />
                </FormControl>
                <FormMessage className="…" />
              </FormItem>
            )}
          />
          </form>
 </Form>
```

Hijerarhija komponenata je: „Form“, „form“, „FormField“, „FormItem“, „FormLabel“, „FormControl“, „FormMessage“ i „Input“. Nabrojane komponente rade zajedno kako bi formirale polja obrasca i cjelokupan obrazac. Strukturiraju obrazac, upravljaju njime i povezuju ga s komponentama biblioteke *React Hook Form* koje upravljaju stanjem obrasca. „FormField“ povezuje svako polje obrasca s logikom i validacijom koju pruža navedena biblioteka, dok „control“, „name“ i „field“ su tzv. *props* koji osiguravaju pravilnim vezanjem i rukovanjem validacije. „FormItem“, „FormLabel“, „FormControl“ i „FormMessage“ se koriste kako bi sva polja bila vizualno konzistentna i pružala povratne informacije korisnicima (npr. „FormMessage“ komponenta služi za ispisivanje poruke greške ukoliko polje za unos nije pravilno ispunjeno).

Nakon ispunjenog obrasca, „form.handleSubmit“ funkcija automatski dohvaća vrijednosti svakog polja i grupira ih u jedan objekt („data“) gdje ključevi (engl. *keys*) odgovaraju „name“ *propu* koji je definiran u svakoj „FormField“ komponenti. Nakon toga, funkcija (pomoću *zod* sheme) provjera pravilnost unosa polja. Ako polje nije pravilno ispunjeno, povratno se javlja poruka korisniku ispod tog polja. U suprotnom, gornja „handleSubmit“ funkcija poziva moju, prilagođenu „handleSubmit“ funkciju i kao argument stavlja „data“ objekt (kod funkcije prikazan je ispod).

```
const handleSubmit = async (data: FormData) => {
    const { opis, datum, vrsta, mjesto } = data;

    try {
      const response = await fetch("/api/createIntervention", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({ opis, datum, vrsta, mjesto }),
      });

      if (!response.ok) {
        throw new Error("Network response was not ok");
      }

      console.log(response);

      const result = await response.json();
      console.log("Server response:", result);
    } catch (error) {
      console.error("Error", error);
    }
```

```
  };
```

Zahvaljujući *React Hook Form* biblioteci i njezinom jednostavnom načinu za dohvaćanje podataka iz obrasca, na početku kao argument funkcije je „`data`“ objekt u kojem se nalaze upravo vrijednosti polja za unos. Nakon toga se destrukturira (engl. destructure) objekt i izvade samo ona polja koja se šalju prema API krajnjoj točci.

3.1.6. Pregled intervencija

Na samoj naslovnoj strani omogućeno je gostu pregledavanje prošlih intervencija vatrogasne postaje. Tablica koja prikazuje najosnovnije informacije o intervenciji (mjesto, vrijeme, datum i kratki opis) te uz to i datum objavljivanja pojedine intervencije. Pojedina polja (datum intervencije i datum objavljivanja) moguće je sortirati. Također, na tablici je implementirana i paginacija s kojom se korisnik može koristiti. Sve to je lakše za implementirati (za razliku od čistog JavaScripta) zahvaljujući *TanStack Table* biblioteci. Popularna je zato što pruža logiku, status i procesuiranje za elemente te interakciju između njih. Potrebno je pratiti jasno definirane korake za izradu dinamičke tablice. Za pregled intervencija kreirana je zasebna komponenta „`dataTable`“ koja se sastoji od još manjih, funkcionalno bitnih komponenata. Primjerice, za paginaciju tablice je implementirana posebna komponenta kako bi se lakše upravljalo s njome te ju je potrebno samo uključiti u datoteku gdje se nalazi tablica („`dataTable`“ datoteka proizvoljnog naziva). Svu logiku odrađuje spomenuta biblioteka, dok je programer potreban definirati stupce tablice. To odrađuje u komponenti „`Columns`“ (proizvoljnog naziva), čiji kod za jedan stupac je prikazan ispod.

```
export type Intervencija = {
  id: string;
  opis: string;
  datum: string;
  vrsta: string;
  mjesto: string;
  datumObjavljivanja: string;
};

export const Columns: ColumnDef<Intervencija>[] = [
  {
    accessorKey: "opis",
    header: () => {
      return (
        <p className="font-bold text-gray-900">Kratki opis intervencije</p>
      );
    },
  },
]
```

„`Columns`“ komponenta izvozi (engl. *export*) niz (engl, *array*) ispunjen definicijama stupaca. Svaki objekt u nizu definira pojedini stupac u tablici (u kodu je prikazan prvi stupac tablice –

„kratki opis intervencije“). Bitno je napomenuti da je za rad „`Columns`“ komponente potrebno u njoj na početku definirati tip („Intervencija“) po kojem će se tablica voditi, više o tome i značenje bitnih dijelova niže.

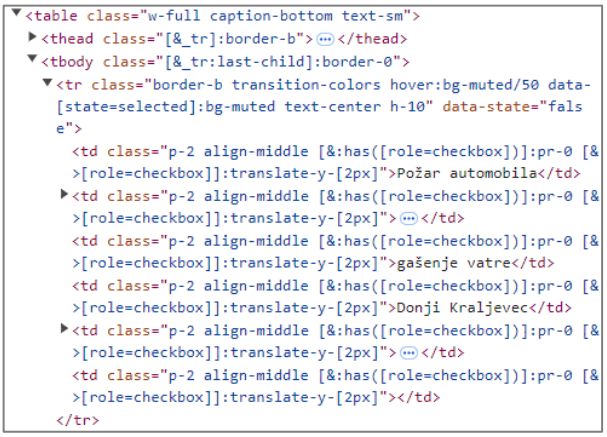
1. `ColumnDef<Intevencija>[]`
    - Tablica očekuje podatke koji se nalaze u objektu „Intervencija“

2. `accessorKey`
    - Ključ koji definira koje će se polje od tipa „Intervencija“ prikazati.
    - U gornjem primjeru „`accessorKey`“ je podešen na „`opis`“, što znači da će tablica tražiti svojstvo „`opis`“ u svakom objektu „Intervencija“ kako bi popunila ćelije ovog stupca.

3. `Header: () => {…}`
    - Svojstvo „`Header`“ definira sadržaj koji će biti renderiran u zaglavlju ovog stupca.
    - „`Header`“ je funkcija koja vraća JSX element. To omogućava programeru prilagođavanje stilova za zaglavlje.

Ovakva struktura je ključna za definiranje načina na koji će se podaci prikazivati i vršiti interakciju u tablici, omogućavajući fleksibilne konfiguracije tablica.

Sam HTML kod preglednika za jedan redak u tablici izgleda kao na slici 15.



Slika 15. HTML kod tablice popisa intervencije

Tablica nije dinamička samo zbog toga jer ima mogućnost sortiranja, već i zbog toga što se ovisno o određenom uvjetu prikazuje dodatan stupac. Ako je korisnik ulogiran i administrator, na stranici „popis intervencija“ dodaje se na kraju stupac „`actions`“. Zahvaljujući tome stupcu,

administrator ima mogućnost uređivanja ili brisanja intervencija. Implementacija stupca je također u „Columns“ komponenti, a kod zaslužan za to je sljedeći:

```
  {
    id: "actions",
    cell: ({ table, row, column }) => (
      <DataTableRowActions table={table} row={row} column={column} />
    ),
  },
```

1. „id“
    - Svojstvo koje daje stupcu jedinstvenu identifikaciju. Razlika između „accessorKey“ koji povezuje stupac s određenim poljem u objektu (primjer naveden gore – „Interventions“) kojeg definiramo je ta da se „id“ koristi kada želimo definirati novi stupac, ali ne nužno i povezati s određenim poljem u objektu

2. `cell: ({ table, row, column }) => (…)`
    - funkcija koje prima tri argumenta: „table“, „row“ i „column“. Ti argumenti pružaju kontekst o trenutnoj tablici, retku s podacima koji se prikazuju i samom stupcu. *TanStack* biblioteka na temelju tih argumenata omogućava kreiranje proizvoljnih stupaca, redaka ili ćelija.

3. `<DataTableRowActions table={table} row={row} column={column} />`
    - „cell“ funkcija vraća JSX element, odnosno komponentu „DataTableRowActions“ koja se kreira u „Components“ mapi.
    - „DataTableRowActions“ komponenta je zaslužna za renderiranje „uredi intervenciju“ i „obriši intervenciju“ gumbova, a njezin kod je prikazan i objašnjen niže.

```
Async function handleDelete() { … }


return isSignedIn ? (
    <div>
      <DropdownMenu>
        <DropdownMenuTrigger asChild>
          <Button variant="ghost" className="…">
            <span className="…">Open menu</span>
            <MoreHorizontal className="…" />
          </Button>
        </DropdownMenuTrigger>
        <DropdownMenuContent
          className={…}
        >
          <DropdownMenuItem asChild>
            <DrawerDialogButton
```

```
              row={row.original}
              setIsDialogOpen={setIsDialogOpen}
            />
          </DropdownMenuItem>
          <DropdownMenuItem
            asChild
            className={…}
}
          >
            <Button
              onClick={() => {
                handleDelete(row.original);
              }}
            >
              <span className="…">
                Izbriši intervenciju
              </span>
              <Trash2 className="…" />
            </Button>
          </DropdownMenuItem>
        </DropdownMenuContent>
      </DropdownMenu>
    </div>
  ) : null;
```

Komponenta „`DataTableRowActions`“ u sebi sadrži asinkronu funkciju „`handleDelete`“ kojom se šalje zahtjev prema API krajnjoj točki zaslуženoj za brisanje intervencije u bazi podataka. Funkcionira tako da se šalje parametar „`id`“, odnosno identifikator odabrane intervencije uz „`delete`“ HTTP metodu. Funkcija, odnosno komponenta „`DataTableRowActions`“ vraća JSX element jedino ako je administrator prijavljen (želi se prikazati dodatan stupac samo administratoru). *Clerk* nudi posebne funkcije koje aplikacija može koristiti u kodu, a koje su vezane uz autentifikaciju.

```
import { useAuth } from "@clerk/nextjs";
```

Gore napisani kod prikazuje uključivanje *useAuth()* funkcije koja vraća objekt u kojima se nalaze informacije o trenutnom stanju autentifikacije, uz pomoćne metode koje mogu upravljati s trenutno aktivnom sesijom (npr. „`signOut`“, „`sessionId`“).

```
const { isSignedIn } = useAuth();
```

Za prikazivanje dodatnog stupca za administratora, potrebna je informacija da li je prijavljeni ili ne. Ta se informacija dobije iz „`isSignedIn`“ varijable (tipa *boolean*) koja je destrukturirana iz objekta kojeg vraća *useAuth()* funkcija.

```
return isSignedIn ? (
    <div>
      …
    </div>
)
```

Na temelju varijable „`isSignedIn`“ se prikazuje komponenta, odnosno stupac. Ako je „`isSignedIn`“ istina (engl. *true*) prikaže se cijeli novi stupac, u suprotnom (*false*) se ne događa ništa.

3.2. Izvedba ključnih funkcionalnosti

3.2.1. Redoslijed operacija nakon zahtjeva korisnika

Prikazani su koraci pri zahtjevu korisnika za unos nove intervencije (popunjeni obrazac).

1. **Mrežni zahtjev**

Web preglednik šalje HTTP zahtjev prema *Node* JS serveru na kojem leži *Next* JS aplikacija. Nakon toga je taj zahtjev preusmjeren prema odgovarajućem *API* rukovatelju (engl. *handler*). Preusmjeravanje se događa u „handleSubmit“ funkciji koja se okida pritiskom gumba obrasca nakon uspješne validacije unesenih podataka.

```
  const handleSubmit = async (data: FormData) => {
    try {
      const response = await fetch("/api/createIntervention", {
          method: "POST",
          headers: {
            "Content-Type": "Application/json",
          },
          body: JSON.stringify({
            opis,
            datum,
            vrsta,
            mjesto,
            korisnikId: userId,
          }),
        });

        if (!response.ok) {
          throw new Error("Failed to create intervention");
        }

        const result = await response.json();
        console.log("Server response:", result);
      } catch (error) {
        console.error("Error", error);
      }
  }
```

2. **Rukovanje na strani servera**

```
export default async function createIntervention(
   req: NextApiRequest,
   res: NextApiResponse
) {
```

```
if (req.method === “POST“) {
  try {
     const data = req.body;

     const result = await prisma.intervencija.create({
        data,
     });

    res.status(200).json(result);
 } catch (error) {
    console.error(error);
    res.status(500).json({ error:“Failed to save intervention“ });
}
```

„createIntervention“ funkcija (prikazana iznad) radi na serveru i ovisno o zahtijevanoj metodi („GET“, „POST“ ili dr.) izvršava određenu logiku. U slučaju projekta funkcija se nalazi u mapi „pages/api/createIntervention“. Na početku funkcije radi se provjera kako bi se ta funkcija okinula samo kad je u pitanju dohvat podataka, odnosno „POST“ metoda. Konkretno na mojem primjeru, „data“ su podaci koje je klijent poslao u tijelu zahtjeva („opis“, „datum“, „vrsta“ i „mjesto“). Funkcija je također asinkrona jer je potrebno kreirati novu intervenciju, odnosno popuniti bazu podataka s novim podacima, pa će se to odraditi u pozadini i neće stopirati rad aplikacije.

3. **Konstrukcija odgovora**

Kada su podaci zaprimljeni i procesuirani, poslužitelj izrađuje HTTP odgovor. Odgovor se izrađuje upravo u API ruti „createIntervention“ i općenito uključuje statusne kodove (npr. „200“ za uspješno izvršavanje) te podatke u obliku JSON-a. Primjer odgovora prikazan je ispod na slici 16.

Slika 16. Primjer HTTP odgovora servera

4. **Rukovanje odgovorom na strani klijenta**

Na kraju, klijent dobije HTTP odgovor od poslužitelja. Ispod je prikazan kod u kojemu se obrađuje odgovor od poslužitelja. Odgovor se nalazi u varijabli „response“. Kako bi mogli obrađivati podatke, odnosno tijelo HTTP odgovora, treba se poslužiti JSON metodom. To se radi jer je HTTP odgovor tipično tekstualan format, zbog toga koristimo „response.json()“ metodu

koja pretvara tekst u *JavaScript* objekt. Na kraju u varijabli „`result`“ dobijemo objekt koji predstavlja intervenciju koju je korisnik unio u obrazac.

```
const handleSubmit = async (data: FormData) => {
    const { opis, datum, vrsta, mjesto } = data;

    try {
      const response = await fetch("/api/createIntervention", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({ opis, datum, vrsta, mjesto }),
      });

      if (!response.ok) {
        throw new Error("Network response was not ok");
      }

      console.log(response);

      const result = await response.json();
      console.log("Server response:", result);
    } catch (error) {
      console.error("Error", error);
    } finally {
      setIsLoading(false);
      setFormSubmitted(true);
      reset();

      setTimeout(() => window.location.reload(), 3500);
    }
  };
```

5. **Rukovanje greškama (engl. *error handling*)**

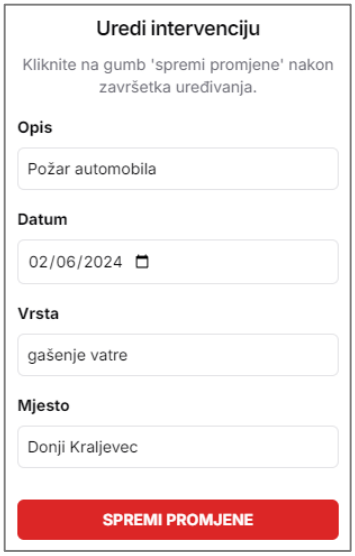
Ako tijekom izvođenja funkcije dođe do greške, ona će se rukovati u „`catch`“ bloku te se definira što će se prikazati ako ima određenih grešaka. Primjerice, ako bi statusni kod rezultata bio „`500`“, javila bi se poruka „*Failed to save intervention*“.

Za ostale metode princip je sličan, samo je pitanje da li se u tijelu zahtjeva šalju podaci (ako radimo unos podataka ili izmjenu postojećih – „`POST`“ i „`PUT`“ metode) ili ne (primjer kod dohvata podataka iz baze – „`GET`“ metoda).

3.2.2. Izvedba sučelja za uređivanje i brisanje intervencije

Na temelju uloga se pojavljuju i dodatne mogućnosti tablice. Uloge su administrator i gost. Razlika između tih uloga jest što administrator ima mogućnost uređivanja ili brisanja intervencija. Ako je neka od informacija krivo unesena, prilikom klika na opciju „uredi intervenciju“

administratoru se otvara prozor koji preuzima fokus stranice. Sučelje za uređivanje intervencije prikazano je na slici 17.



Slika 17. Prikaz sučelja za uređivanje intervencije

Moguće je urediti svaku od opcija te spremiti promjene. Nakon završetka uređivanja i klika na gumb „spremi promjene“ kao povratna informacija za uspješno uređivanje javlja se tzv. „toast“ obavijest. To je kratka poruka koja se obično pojavljuje na dnu ekrana i koristi se kako bi se korisnicima pružile kratke informacije o onome što se trenutno događa u aplikaciji ili kao na primjeru gore - odgovor na temelju određene korisničke radnje. Kod brisanja intervencije korisniku se, nakon klika gumba „Izbriši intervenciju“, nakon određenog kratkog vremena osvježi stranica.

Tijekom tog osvježenja stranice događa se više radnji. Pokreće se asinkrona funkcija „`handleDelete`“ čiji kod je prikazan niže.

```
async function handleDelete(intervencija: any) {
   const id = intervencija.id;
   console.log(id);

   const response = await fetch(`/api/deleteIntervention?id=${id}`, {
     method: "DELETE",
   });

   if (response.ok) {
     console.log("Row deleted successfully");
     window.location.reload();
   }
 }
```

Problem kod brisanja intervencije je dohvaćanje identifikacije („`id`“) intervencije koja je odabrana od strane administratora za brisanje. „`id`“ predstavlja identifikaciju za svaki redak tablice, odnosno intervenciju. Pri brisanju intervencije na temelju te identifikacije briše se redak u

bazi koji predstavlja tu intervenciju. Na aplikaciji nema izravnog komuniciranja s bazom, već za to služi alat *Prisma* . Kao što je već spomenuto korištena je *TanStack Table* biblioteka. Za gore navedeni problem daje objekt „*Row*“ (redak) koji u sebi sadržava podatke o retku i razna programska sučelja (engl. API) koja omogućavaju interakciju s tablicom, retkom ili stupcem. Svaki objekt retka u sebi sadržava i identifikaciju, odnosno „`id`“. Za dobivanje identifikacije kliknutoga retka upotrjebljeno je svojstvo gumba „`onClick`“. Naredba za dobivanje objekta odabranog retka preko svojstva „`onClick`“ je:

```
onClick={() => { handleDelete(row.original); }}
```

Nakon toga u funkciji „`handleDelete`“ (prikazanoj gore) se na početku dohvaća taj objekt te se iz njega uzima identifikacija odabranog retka kako bi se poslao zahtjev prema bazi za brisanje. Time rješavamo prvi dio problema – dohvaćanje „`id`“ kliknute/odabrane intervencije, dok je drugi dio problema brisanje intervencije na serverskoj strani. Rješenje tog problema pronalazi se u asinkronoj funkciji „`deleteIntervention`“ koja predstavlja API krajnju točku. U projektu se nalazi u mapi „pages/api“ uz ostale API točke, a prikazana je niže.

```
export default async function deleteIntervention(
  req: NextApiRequest,
  res: NextApiResponse
) {
  const { id } = req.query;

  console.log(id);

  if (req.method === "DELETE") {
    try {
      const deletedIntervention = await prisma.intervencija.delete({
        where: {
          id: String(id),
        },
      });

      res
        .status(200)
        .json({
          message: " Intervention successfully deleted ",
          deletedIntervention,
        });
    } catch (error) {
      res.status(500).json({ error: "Failed to delete the intervention" });
    }
  } else {
    res.setHeader("Allow", ["DELETE"]);
    res.status(405).end(`Method ${req.method} Not Allowed`);
  }
}
```

Na samom početku je potrebno izvaditi identifikaciju iz parametara upita. Radi se provjera je li zahtjev “`DELETE`“. Ako je, briše se intervencija iz baze pomoću *Prisme* i njezine “`delete`”

metode. Nakon uspješnog brisanja šalje se povratno JSON odgovor s obrisanom intervencijom i porukom „Intervencija je uspješno obrisana“.

Redoslijed izvršenja za brisanje intervencije:

1. **Strana klijenta:** „`handleDelete`“ funkcija se okine, šalje HTTP **„`DELETE`“** zahtjev prema serveru.
2. **Strana servera:** „`deleteIntervention`“ programsko sučelje (API) dohvaća zahtjev, procesuira ga - obriše intervenciju i šalje nazad odgovor.
3. **Strana klijenta:** „`handleDelete`“ funkcija prima odgovor servera, provjerava da li se uspješno obrisala intervencija, ispisuje poruku na konzolu te na kraju ponovo učitava stranicu.

3.2.3. Izvedba dohvaćanja i prikazivanja prijavljenih korisnika administratoru

Pri prijavi administratora u web aplikaciju, između pregleda intervencije i prikaza statistike, pojavljuju se i dvije dodatne opcije. Jedna od opcija navigacije je naziva „Prijavljeni korisnici“ te prilikom klika na nju, administratoru se prikazuje popis korisnika koji su se prijavili na web aplikaciju i njihova uloga. Kada administrator zatraži pregled korisnika (klikom na „Prijavljeni korisnici“) otvara mu se dijalog/prozorčić koji prikazuje adrese e-pošte prijavljenih korisnika te njihove uloge (engl. *role*) na web aplikaciji. Prilikom otvaranja klijentska strana komunicira s API krajnjom točkom „`getDBUsers`“. U tome API dokumentu se događa nekoliko vrsta slanja i primanja podataka – slanje zahtjeva prema *backend* API sučelju *Clerk*-a (kako bi se dohvatili prijavljeni korisnici koje *Clerk* sprema), nakon toga se ti podaci spremaju u *PostgreSQL* bazu podataka (pomoću *Prisme*) i na kraju se šalje odgovor prema klijentskoj strani koji sadrži podatke iz baze. U slučaju da se podaci s *Clerk* API sučelja ne mogu dohvatiti, administratoru će se podaci svejedno prikazati jer ih dobiva iz lokalne baze podataka – izbjegavanje ovisnosti o vanjskom izvoru. Kada se veza s *Clerk*-om uspostavi, baza se ažurira i dohvate se najnoviji podaci.

```
export default async function handler(
  req: NextApiRequest,
  res: NextApiResponse
) {
  const apiKey = "sk_test_BWH0t5qUcVjyI8geZfBZq60Haho0reJITtJQrsRv0f";

  try {
    const response = await fetch("https://api.clerk.com/v1/users", {
      method: "GET",
      headers: {
        Authorization: `Bearer ${apiKey}`,
```

```
          "Content-Type": "application/json",
        },
      });

      if (!response.ok) {
        throw new Error(`Failed to retrieve user: `${response.status}`);
      }

      const data = await response.json();

      const users = data.map((user: any) => ({
        id: user.id,
        email_address: user.email_addresses.map(
          (emailObj: any) => emailObj.email_address
        ),
      }));

      for (const user of users) {
        const emailAddress = user.email_address[0];

        await prisma.prijavljeniKorisnik.upsert({
          where: {
            korisnikId: user.id,
          },
          update: {
            email: emailAddress,
          },
          create: {
            korisnikId: user.id,
            email: emailAddress,
          },
        });
      }
    } catch (error) {
      console.log(error);
    }

    try {
      const users = await prisma.prijavljeniKorisnik.findMany();
      res.status(200).json(users);
    } catch (error) {
      res.status(500).json({ error: "“Error retrieving user" });
    }
  }
```

Kod prikazan gore služi za dohvaćanje podataka *Clerk* API točke, spremanje u tablicu baze podataka i slanje podataka prema klijentskoj stranici dohvaćene iz baze. Kod je kompliciraniji zato jer je potrebno podesiti podatke jer ne dolaze u istom obliku kakvoga tablica u bazi podataka očekuje. Tablica „`PrijavljeniKorisnik`“ sadrži 5 polja – „`id`“, „`korisnikId`“, „`email`“, `uloga`“, „`vrijemePrijave`“. Kako API odgovor sadrži objekt u kojemu se nalaze više polja, kreirana je pomoćna varijabla „`users`“ koja prolazi/mapira „`data`“ odgovorom i sprema u objekt samo identifikaciju korisnika i adresu e-pošte – jedina polja koja su potrebna za prikaz administratoru prijavljene korisnike. Dovoljno je bilo dohvatiti samo adresu e-pošte, no sada s identifikacijom korisnika se mogu raditi druge radnje ako je to potrebno (primjerice, s tim

identifikatorom može se brisati korisnik, zabraniti prijavu određenom korisniku i dr.). Polje „`uloga`“ se automatski dodjeljuje svakom novom korisniku (engl. *user role*) jer je tako određeno arhitekturom aplikacije, odnosno tablice u bazi podataka. Administrator se definira u službenoj upravljačkoj ploči *Clerk*-a. Automatsko dodjeljivanje je implementirano u *Prisma* shemi (`@default(KORISNIK)`) koja je prikaza niže.

```
model PrijavljeniKorisnik {
  id             String    @id @default(cuid())
  korisnikId     String    @unique
  email          String
  uloga          Uloga     @default(KORISNIK)
  vrijemePrijave DateTime?
}
```

Također, na samom kraju dodano je opcionalno polje „`vrijemePrijave`“ koje programer može iskoristiti za implementiranje dodatnih funkcionalnosti uz prijavljenog korisnika. Primjerice, može se povezati stranica sa statistikom i u njoj se prikazivati uobičajena vremena prijave ili dr. U suštini, preko ove tablice se programer može odlučiti na dodavanje raznih funkcionalnosti koje mogu unaprijediti rad web aplikacije i mogućnosti samog administratora.

3.2.4. Izvedba dohvaćanja podataka „klasičnim“ pristupom (JS + PHP)

Za potrebe završnog rada na stranici „popis intervencija“ implementirana je tablica koja dohvaća podatke klasičnim pristupom – JavaScript (klijentski dio) i PHP + *Apache* (poslužiteljski dio). Kako se na kolegiju „Osnove Web programiranja“ obrađivao PHP uz *Apache* poslužitelj, imao sam predznanja da napravim dodatnu tablicu. Kako bi dohvatili podatke sa servera potrebno je bilo implementirati asinkronu funkciju. Proizvoljnim imenom, izradio sam funkciju „`fetchData`“ koja dohvaća podatke sa servera koristeći XHR (akronim od engl. *XMLHttpRequest*) nakon pritiska na gumb „Dohvati podatke“. Uz pomoć XHR-a moguće je dohvatiti podatke sa servera bez potrebe osvježavanja cijele stranice. Funkcija je prikazana ispod.

```
    const fetchData = () => {
      const xhr = new XMLHttpRequest();

      xhr.open("GET", "http://localhost:8080/fetch_data.php", true);

      xhr.onreadystatechange = function () {
        if (xhr.readyState === 4) {
          if (xhr.status === 200) {
            try {
              const responseData: XHRIntervention[] = JSON.parse(
                xhr.responseText
              );
              console.log("Response Data: ", responseData);

              setXHRData(responseData);
            } catch (error) {
              setError("Failed to parse JSON");
```

```
            console.error("Error parsing JSON", error);
          }
        } else {
          setError(`Failed to fetch data: HTTP status ${xhr.status}`);
          console.error("Failed to fetch data:", xhr.status);
        }
      }
    };
    xhr.send();
  };
```

Na početku je potrebno inicijalizirati novi XHR objekt koji omogućava slanje zahtjeva i primanje odgovora poslužitelja. Kako bi se dohvatili podaci, „otvori“ se novi „`GET`“ zahtjev i u njemu je potrebno definirati URL adresu (kamo se šalje zahtjev) i dodatan parametar (*true*) koji označava da bi zahtjev trebao biti asinkroni (što znači da ne sprječava ostale operacije dok se čeka odgovor). Odgovorom se upravlja u dodatnoj funkciji (`xhr.onreadystatechange = function () {}`) koja se okida kada se „`readyState`“ svojstvo u XHR objektu promijeni. Time je omogućeno upravljanje odgovorom tek kada se zahtjev procesuirao. „`readyState`“ svojstvo ukazuje na trenutno stanje zahtjeva (vrijednost „`4`“ označava da je zahtjev dovršen i odgovor je spreman) s korisničke strane. Dok „`xhr.status`“ označava stanje odgovora (vrijednost „`200`“ - uspješno dohvaćanje) s poslužiteljske strane. Na kraju se sprema odgovor u varijablu „`responseData`“ i to u *JavaScript* objekt - „`JSON.parse(xhr.responseText)`“. Kako bi se mogao odgovor prikazati i izvan ove funkcije koristi se varijabla stanja (`setData(responseData)`). Korisniku se prikazuju podaci nakon pritiska na gumb „Dohvati podatke“ i to u tablici, mapiranjem kroz „`data`“ varijablu stanja (u slučaju više objekata – više redaka tablice).

```
      <table className="mt-10 lg:mx-auto">
            <thead className="border-2">
              <tr>
                <th>opis</th>
                <th>datum</th>
                <th>vrsta</th>
                <th>mjesto</th>
                <th>datumObjavljivanja</th>
              </tr>
            </thead>
            <tbody className="border-2">
              {data.length > 0 ? (
                data.map((row) => (
                  <tr key={row.id}>
                    <td className="px-10 border-r-2">{row.opis}</td>
                    <td className="px-10 border-r-2">{row.datum}</td>
                    <td className="px-10 border-r-2">{row.vrsta}</td>
                    <td className="px-10 border-r-2">{row.mjesto}</td>
                    <td className="px-10 border-r-2">
                      {row.datumObjavljivanja}
                    </td>
                  </tr>
                ))
              ) : (
                <tr>
```

```
                        <td className="px-10 border-r-2">Podaci nisu dostupni</td>
                        <td className="px-10 border-r-2">Podaci nisu dostupni</td>
                        <td className="px-10 border-r-2">Podaci nisu dostupni</td>
                        <td className="px-10 border-r-2">Podaci nisu dostupni</td>
                        <td className="px-10 border-r-2">Podaci nisu dostupni</td>
                      </tr>
                    )}
                  </tbody>
                </table>
```

Kod PHP skripte na kojoj se dohvaćaju podaci s *Apache* poslužitelja koji je konfiguriran za slušanje na portu „`8080`“ („http://localhost:8080/fetch_data.php“) je sljedeći:

```
   <?php
   $host = 'ep-cool-mode-a2nstwla-pooler.eu-central-1.aws.neon.tech';
   $dbname = 'verceldb';
   $user = 'default';
   $password = 'uQJ0ewSas2ZY';
   $port = '5432';

   // povezivanje na PostgreSQL bazu
   $conn = pg_connect("host=$host dbname=$dbname user=$user password=$password
port=$port sslmode=require connect_timeout=15");

   if (!$conn) {
       die("Connection failed: " . pg_last_error());
   }

   // upit
   $result = pg_query($conn, "SELECT * FROM \"Intervencija\"");

   if (!$result) {
       echo json_encode(['error' => 'Failed to execute query']);
       exit;
   }

   // dohvaćanje podataka i definiranje podataka u JSON obliku
   $data = pg_fetch_all($result);

   echo json_encode($data);

   pg_close($conn);

   ?>
```

Znakovi “`<?php`” i “`?>`” označavaju početak i kraj PHP skripte. Na samom početku potrebno je dati parametre baze podataka (naziv baze, port na koji se spaja, lozinka, ime korisnika) pomoću kojih će se dohvatiti podaci. Veza prema *PostgreSQL* bazi podataka se uspostavlja pomoću PHP funkcije „`pg_connect()`“ u kojoj se nalaze definirani parametri baze spomenuti prije. Upit prema bazi se kreira pomoću funkcije „`pg_query`“. U varijablu „`result`“ spremaju se podaci koji se dohvate pomoću SQL upita (`SELECT * FROM \"Intervencija\")` koji dohvaća sve (*) podatke iz tablice „`Intervencija`“. Dok se u varijablu „`data`“ transformiraju podaci u JSON oblik („`echo json_encode($data)`“)

3.3. Responzivni dizajn

Tijekom 2020.godine procijenjeno je da se oko 25% ukupne medijske potrošnje dogodilo na mobilnim oglasima. Iz toga se dade zaključiti da su marketinški stručnjaci prilično uvjereni da mobilna industrija nudi veći prodajni potencijal od računala. Prema stranici *ComScore*, potrošači troše gotovo 90% vremena na mobilnim uređajima. U web programiranju postoji više pristupa dizajnu i izradi web aplikacija – tradicionalna web aplikacija, mobilni pristup te responzivni dizajn. Tradicionalna web stranica ne mijenja izgled ovisno o uređaju te se u današnje vrijeme više ni ne pomišlja na izradu takve. Mobilni pristup (engl. *mobile-first approach*) gdje se prvo dizajnira sučelje prilagođeno mobilnim uređajima, a tek onda „klasičnim“ računalima. Istim redoslijedom i programer izrađuje web aplikaciju. Odabirom mobilnog pristupa jamči se korisniku laka navigacija i nezatrpanost slikama ili dugim tekstovima. Prilikom odlučivanja hoće li se krenuti mobilnim pristupom postavlja se pitanje kako dizajnirati stranicu proizvoda koja je dovoljno blistava za korisnike računala, a opet dovoljno „prijateljska“ za mobilne korisnike? Responzivni dizajn nadopuna je mobilnom pristupu. Programer koristi tzv. *media queries* kako bi modificirao sadržaj prema vrsti uređaja ili veličini zaslona. U suštini, dopušta većim ekranima dodavanje slojeva medija koji unaprjeđuju sadržaj, ali ne oduzimaju ništa od temeljne poruke proizvoda.

Na projektu vatrogasne postaje korišten je pristup responzivnog dizajna. Zahvaljujući alatu *Tailwind* CSS responzivni dizajn je jednostavno implementirati. Slika 18. prikazuje izgled naslovne strane za mobilne uređaje uz pomoć takvog dizajna.

Slika 18. Izgled naslovne stranice na mobilnom uređaju

To je sve moguće zahvaljujući tzv. *media queries*. *Tailwind* CSS u sebi ima implementiranih pet prijelomnih točaka koji su inspirirani uobičajenim rezolucijama uređaja. Slika 19. prikazuje tih pet točaka.

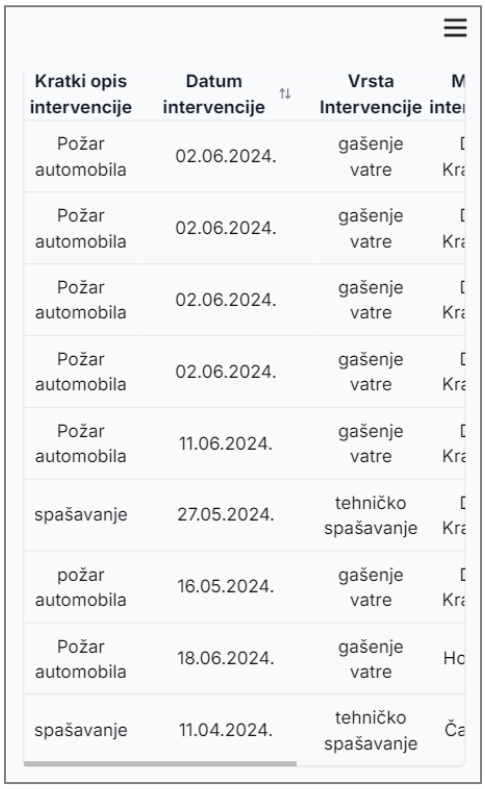
Breakpoint prefix	Minimum width	CSS
`sm`	640px	`@media (min-width: 640px) { ... }`
`md`	768px	`@media (min-width: 768px) { ... }`
`lg`	1024px	`@media (min-width: 1024px) { ... }`
`xl`	1280px	`@media (min-width: 1280px) { ... }`
`2xl`	1536px	`@media (min-width: 1536px) { ... }`

Slika 19. Prikaz pet prijelomnih točaka

Ako želimo da nam se neki od elemenata ne prikaže na određenim rezolucijama zaslona, to možemo jednostavno napraviti prema kodu niže.

```
<div class="hidden md:block">Vidljivo samo na zaslonima većim od 'md' postavke</div>
```

Div element neće biti vidljiv na rezolucijama ekrana manjim od 'md' postavke. U projektu se puno koristilo tih pet točaka kako bi sve slike i tablice bile prikazane u cijelosti i u pravim postavkama na svim rezolucijama. Tablice se na mobilnim uređajima prikazuju djelomično, no korisnik ima mogućnost pomicanja u desno kako bi prošao cijelom širinom tablice, što se može vidjeti na slici 20.



Slika 20. Izgled tablice na mobilnom uređaju

3.4. Mogućnosti proširenja

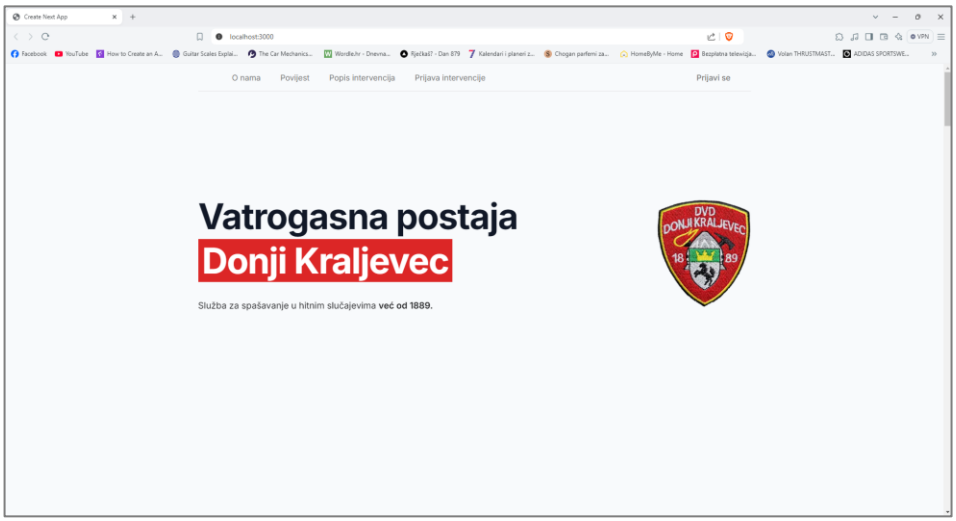
Web aplikaciju je moguće dodatno unaprijediti ili personalizirati po potrebi. Može otići u više smjerova. Web aplikaciji se mogu nadodati i *Google* karte s kojima se mogu implementirati razne funkcionalnosti. Primjer mogućnosti jest da karte prikažu lokaciju i smjer putovanja navalnih vozila na prijašnjim intervencijama preko koje bi se mogli vaditi statistike vremena putovanja i mogućnosti unaprjeđenja vremena putovanja. Također, preko *Google* karata bi se mogla i ucrtati mjesta hidranata što bi pomagalo vatrogascima na intervenciji. Drugi primjer unaprjeđenja aplikacije jest dodavanje mogućnosti komuniciranja vatrogasaca. Tako bi zapovjednik mogao komunicirati i obavještavati kolege bitnim informacijama ili sastancima. Dodavanjem kalendara olakšao bi se proces planiranja vremena, radnih sati ili bitnih događaja za koje su potrebne pripreme i pamćenje datuma.

4. Web aplikacija – sučelje i korištenje

U sljedećem odlomku prikazana su sučelja web aplikacije vatrogasne postaje. Konkretno, izgled te funkcionalnost i namjena sučelja.

4.1. Naslovna stranica

Naslovna, odnosno *landing* stranica jest prvo što korisnik može vidjeti kad pomoću web adrese otvori stranicu. Najbitnije što se korisniku na samom vrhu prikaže je navigacija. Pomoću navigacije gost stranice prolazi web aplikacijom te jednostavno i brzo prelazi na dijelove stranice koje ga zanimaju. Naslovna stranica treba biti upečatljiva, čista te mora korisniku privući pozornost. Također, uz pomoću naslova i kratkih tekstova treba dati najbitnije informacije o web mjestu. Slika 21. prikazuje vrh naslovne stranice vatrogasne postaje.



Slika 21. Prikaz naslovne stranica Web aplikacije

4.2. Stranice i sučelja web aplikacije

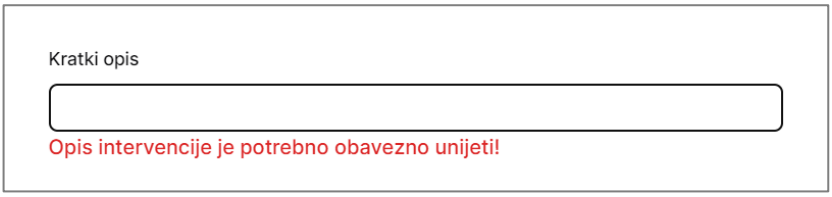
Naslovna stranica je podijeljena na sekcije kako bi se korisnik lakše kretao njome. Također, prilikom klika na jednu od opcija navigacije, korisnika se usmjeri na taj dio web aplikacije. Sekcije su imenovane jednako kao i opcije u navigaciji. Za prve tri sekcije („o nama“, „povijest stanice“ i „popis intervencija“) nije moguća interakcija korisnika, već su na njima samo prikazani tekst i slike vezane uz vatrogasnu stanicu i njezino djelovanje.

Kod dijela „popis intervencija“ je zanimljivo što povezuje *frontend* i *backend*. Naime, podaci koji popunjavaju tablicu dolaze iz baze podataka koje je administrator unio. Sve prošle intervencije se nalaze u jednoj bazi kojoj je moguće pristupiti kako bi se prikazali podaci ili izradila statistika. Prilikom prvog učitavanja stranice, povlače se podaci iz te baze kako bi se tablica popunila i bila spremna za pregled. Na slici 22. uneseni su testni podaci kako bi prikazali izgled popunjene tablice. Također, tablica ima i mogućnost sortiranja podataka kako bi se olakšalo snalaženje i pregled.

Kratki opis intervencije	Datum intervencije ↑↓	Vrsta Intervencije	Mjesto intervencije	Datum objavljivanja ↑↓
Požar automobila	02.06.2024.	gašenje vatre	Donji Kraljevec	17.06.2024.
spašavanje	17.06.2024.	tehničko spašavanje	Prelog	18.06.2024.
Požar automobila	31.05.2024.	tehničko spašavanje	Donji Kraljevec	18.06.2024.
spašavanje	17.06.2024.	tehničko spašavanje	Čakovec	18.06.2024.

Slika 22. Uneseni podaci u tablicu „popis intervencija“

Za sekciju „prijava intervencije“ dana je mogućnost popunjavanja obrasca. Obrazac bi bio dodatan način prijave intervencije, uz standardan - broj na koji je moguće nazvati. Za popunjavanje obrasca bitni su detalji intervencije i detalji pozivatelja, odnosno podnositelja zahtjeva. Ako korisnik ostavi polje prazno i pokuša podnesti (engl. *submit*) obrazac, aplikacija javlja grešku. Greška se pojavljuje ispod polja za unos te uz to obrazac neće moći biti podnesen tako dugo dok se ne isprave sve greške. Slučaj u kojem je korisnik krivo unio (ostavio prazno) opis intervencije prikazan je na slici 23.



Slika 23. Prikaz greške kod nepravilnog unosa polja za opis intervencije

Nakon pravilnog unosa svih polja, obrazac je moguće podnesti. Prilikom pritiska korisnika na gumb „`UNESI`“ aktivira se funkcija „`handleSubmit`“. To je asinkrona funkcija, što znači da se obrađuje drugačije od ostalih funkcija koje su sinkrone. Izgled i što sve sadrži ova funkcija slijedi ispod.

```
const handleSubmit = async (dana: FormData) => {
    const { opis, datum, vrsta, mjesto } = data;

    try {
       const response = await fetch(“/api/createIntervention“, {
           methos: „POST“,
           headers: {
```

```
                „Content-Type“: „application/json“,
            },
            body: JSON.stringify({ opis, datum, vrsta, mjesto }),
          });

          const result = await response.json();
          Console.log(result);

       } catch (error) {
          console.error(„Error, error);
        }
};
```

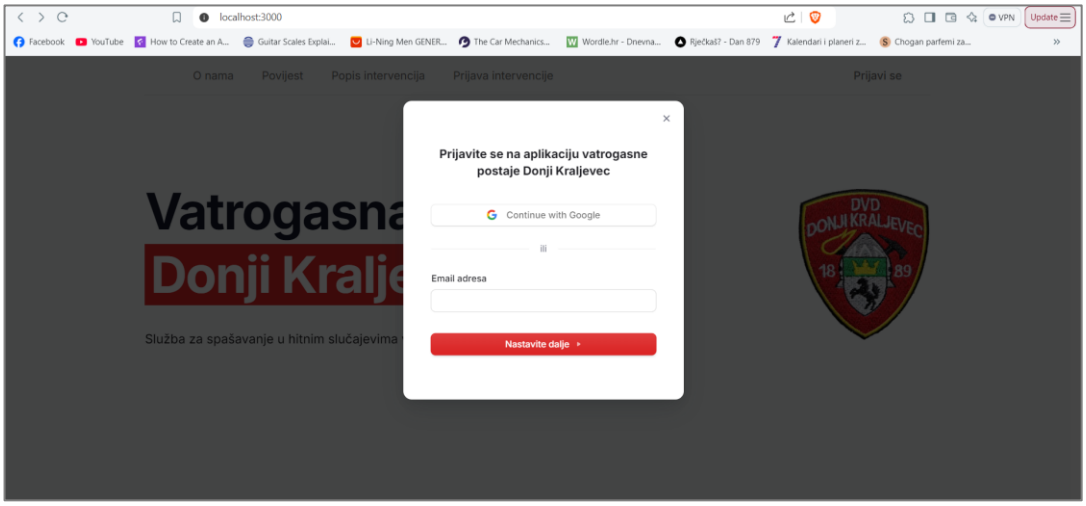
U funkciji se nalazi „`try-catch`“ blok uz još dodatna pomoćna stanja (engl. *states*). U „`try`“ bloku koristi se *fetch* metoda koja radi HTTP zahtjev. Konkretno „`POST`“ zahtjev uz koji možemo slati podatke prema *backendu,* odnosno bazi podataka. „`Fetch`*“* metoda vraća „`promise`*“* koji se sprema u „`response`“ konstantu. To je u suštini odgovor na zahtjev u obliku objekta. Predstavlja cijeli HTTP zahtjev, od zaglavlja, statusnih kodova i tijela. Čitanje podataka iz tijela odgovora, rješava se uključivanjem metode JSON (akronim od engl. *Javascript Object Notation*). Tekstualnog je formata koji je lako čitljiv za ljude a lagan za obradu i generiranje koje rade mašine. Bitno svojstvo jest neovisnost o programskom jeziku programera i koristi konvencije koje su poznate programerima C obitelji *(C, C++, C#, Java, JavaScript, Python*). Kada se doda ova metoda na odgovor poslužitelja, dobije se objekt koji se sastoji od ključ(engl. *key)*/vrijednost*(*engl. *value)* parova. Svaki ključ je `string`, dok „`value`*“* može biti „`string“`, „`number“`, „`object“`, „`array“`, „`true“`, „`false“`, `ili` „`null“`.

Bitna stvar prilikom slanja podataka prema *backendu* jest da treba te podatke staviti u tijelo zahtjeva. Identično, ako želimo dobiti podatke iz servera, njih ćemo pronaći upravo u tijelu zahtjeva. Za slanje je potrebno konvertirati iz proizvoljnog tipa podataka, kao što je objekt, u tekstualno format kojeg podržava HTTP zahtjev. Još jedan od razloga je taj što se baš u zaglavlju definira serveru da će tijelo zahtjeva biti JSON podatak („*Content-Type“: „application/json*“). „`Catch`“ blok služi za definiranje koda u slučaju nastanka grešaka koje nisu dopustile zahtjevu uspješno izvršavanje. Jednostavno rješenje je da se prikaže u konzoli koja od greški se dogodila. Ako greška počinje sa znamenkom 4xx to označava grešku na klijentu, dok greška na serveru počinje s znamenkom 5xx.

Nakon uspješnog zahtjeva, obrazac je podnesen i javlja se „`toast`*“* poruka kao znak korisniku da je uspješno podnio obrazac.

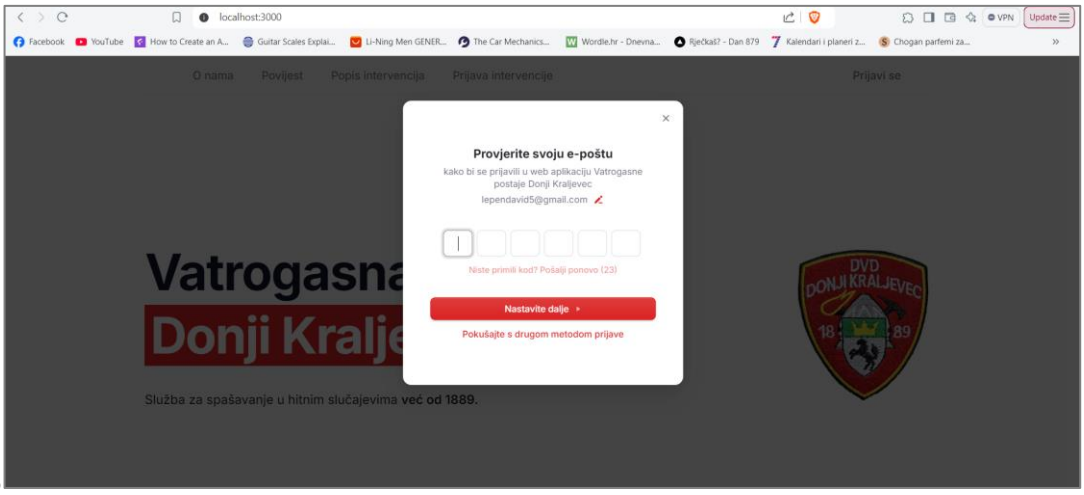
4.2.1. Prijava korisnika

Nakon klika gumba za prijavu, pojavljuje se obrazac kao što je na slici 24.



Slika 24. „Pop-up“ obrazac za prijavu gosta

Gost stranice ima mogućnost prijave preko *Google*-a, odnosno preko adrese s kojim je prijavljen na njegove alate. Drugi način prijave je konfiguriran tako da korisnik unese adresu, ako se ona nalazi u administratorovom popisu, šalje se kod na tu adresu kojeg gost mora unijeti kako bi se uspješno prijavio. Implementacija je toga zbog dodatne provjere identiteta i sigurnosti. Kako izgleda sučelje kada korisnik čeka potvrdu na e-adresi pošte prikazano je na slici 25. Nakon uspješnog unosa koda, aplikacija preusmjerava korisnika na nadzornu ploču web aplikacije.



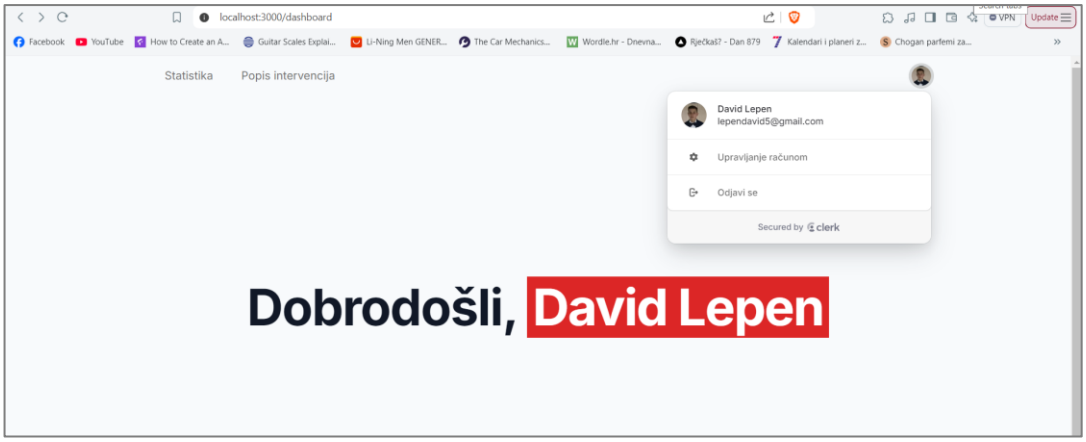
3

Slika 25. Prijava uz pomoć email adrese i koda za provjeru identiteta

4.2.2. Stranica kontrolne ploče korisnika

Korisnika se, nakon uspješne prijave, usmjerava na stranicu kontrolne ploče. Novi link nakon usmjeravanja je „`/dashboard`“. Navigacija se mijenja i sada su korisniku ponuđene tri opcije –

„statistika“ link koji otvara stranicu statistike, link koji otvara stranicu popisa prijašnjih intervencija i zadnji je korisnički gumb (engl. *user button*) koji je cijela zasebna komponenta implementirana od strane *Clerk*-a. Sve tri opcije mogu se vidjeti na slici 26.

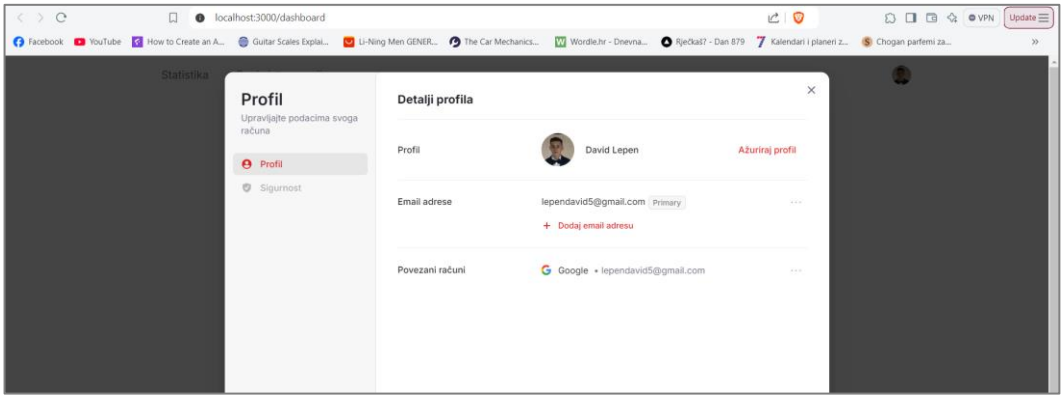


Slika 26. Stranica kontrolne ploče s opcijama navigacije

Gumb koji otvara opcije vezane uz upravljanjem korisničkoga računa („upravljanje računom“, „odjavi se“) je gotova *Clerk* komponenta. Potrebno ju je uvesti u komponentu na kojoj želimo da se prikaže

```
import { UserButton } from "@clerk/nextjs";
```

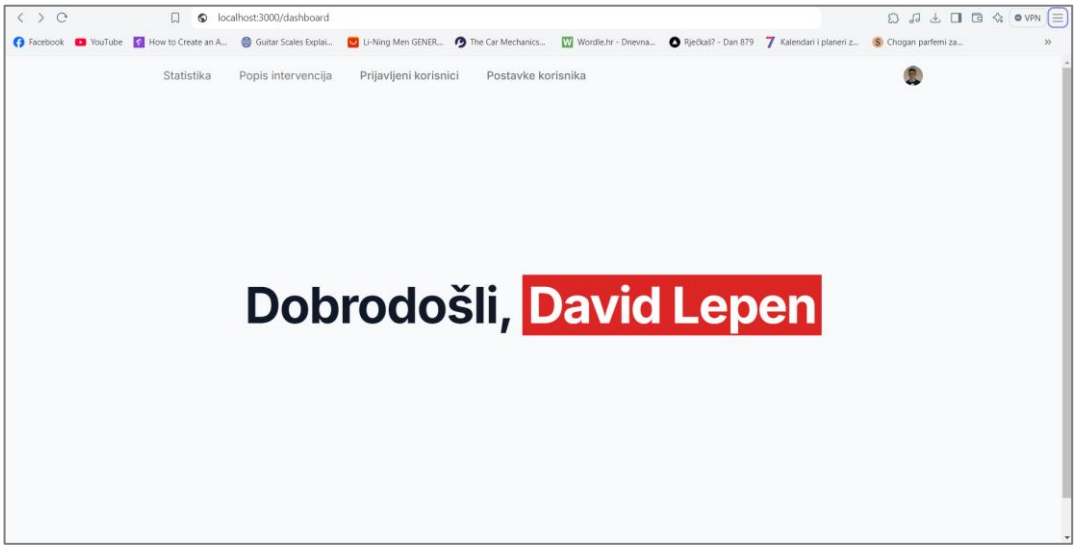
Korisniku se pri odabiru “upravljanje računom” otvara sučelje koje mu daje mogućnost uređivanja profilne slike i brisanja profila. Sučelje je prikazano na slici 27.



Slika 27. Sučelje za uređivanje profila prijavljenog korisnika

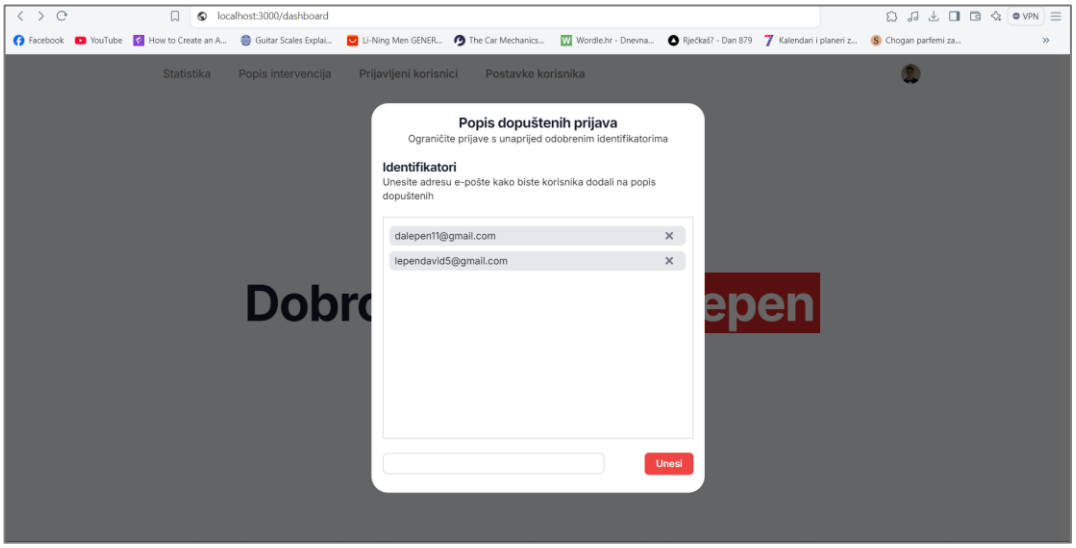
4.2.3. Stranica kontrolne ploče administratora

Kao što je već spomenuto u odjeljku 3.3.3. administratoru se na kontrolnoj ploči pojavljuju dvije dodatne mogućnosti – prikaz prijavljenih korisnika i postavke korisnika, što je i prikazano na slici 28.



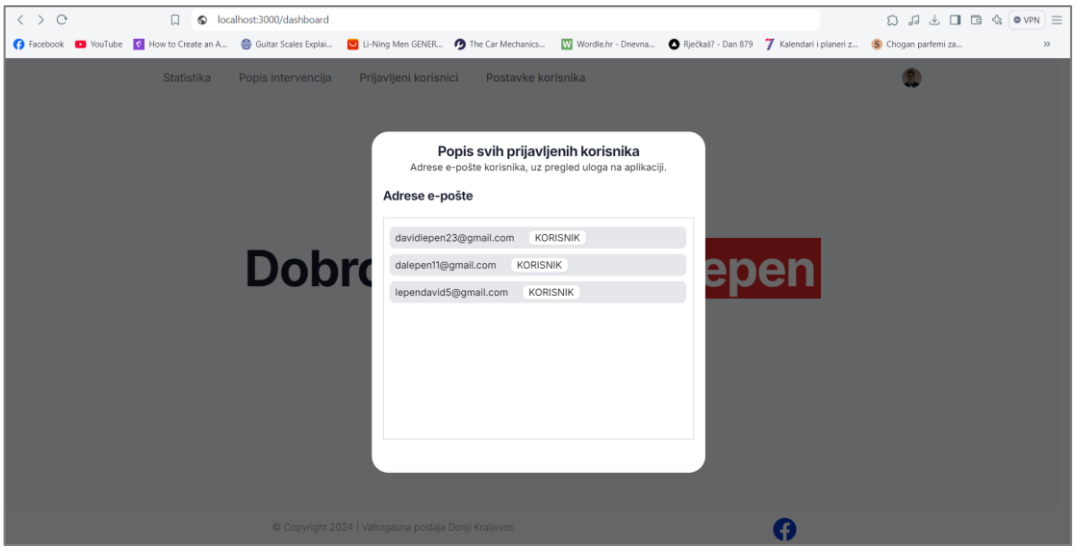
Slika 28. Stranica kontrolne ploče administratora

Ako administrator želi dodati gostu mogućnost prijave, odabire opciju „Postavke korisnika“ i otvara mu se sučelje koje je prikazano na slici 29. U tom sučelju se nalazi jedan polje za unos u kojem administrator unosi adresu e-pošte korisnika kojemu dopušta prijavu u web aplikaciju.



Slika 29. Sučelje za dodavanje mogućnosti prijave korisnika

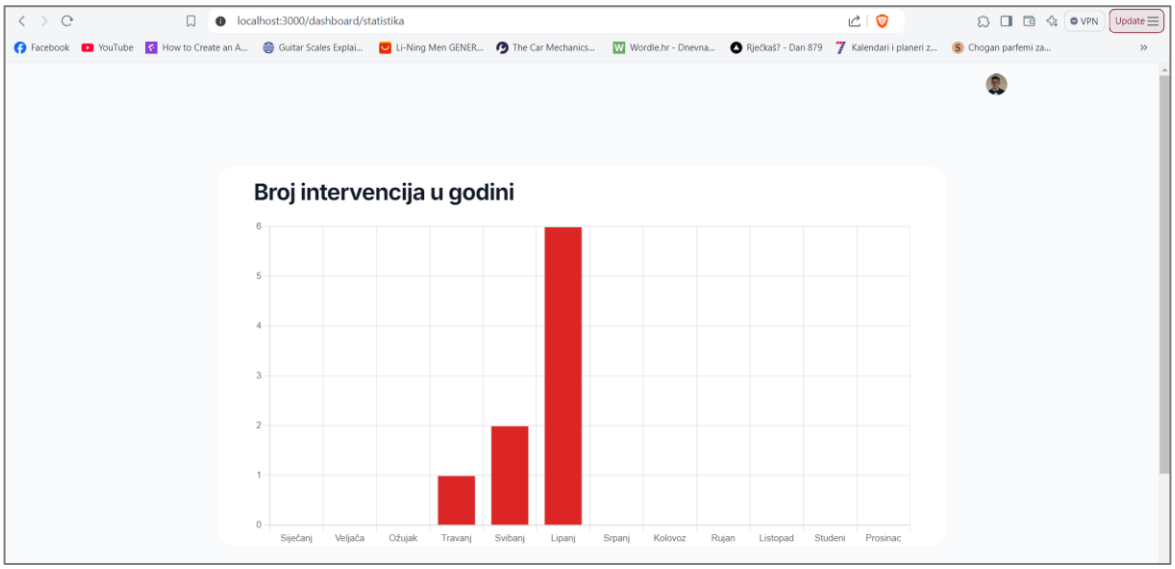
S drugom opcijom administrator može vidjeti tko se sve prijavio na web aplikaciju i njihovu ulogu na web aplikaciji i to je prikazano na slici 30.



Slika 30. Sučelje koje prikazuje popis adresa e-pošta prijavljenih korisnika

4.2.4. Stranica statistike

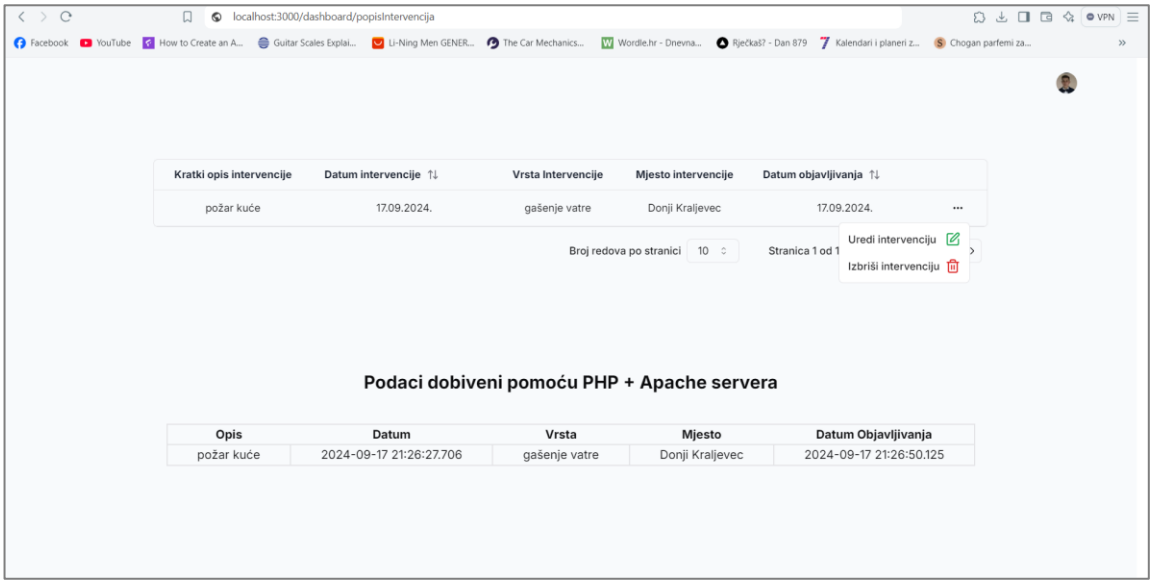
Nakon prijave, korisnika se preusmjerava na novu stranicu „`dashboard`“. Stranicu možemo predstaviti kao kontrolnu ploču koja daje prijavljenim korisnicima uvid u statistiku ili tablicu s popisom intervencija. Te dvije mogućnosti prikazane su na samom vrhu na navigaciji te klikom na jednu od opcija otvara se zasebna stranica. Prilikom klika korisnika na opciju „statistika“ otvara se nova stranica. Stranica je osmišljena kako bi dala korisnicima uvid kroz razne aspekte statistike vatrogasne stanice. Trenutno je prikazan primjer statistike koja prikazuje rad vatrogasne stanice kroz cijelu godinu, odnosno broj intervencija kroz tekuću godinu (prikazano na slici 31.). Moguća proširenja stranice su raznolika. Ako bi vatrogasna stanica upisivala vremena odlaska na intervenciju te vremena dolaska sa intervencije moguće je prikazati statistiku na temelju vremena potrošenog na određenoj intervenciji. Ako bi administrator upisivao vrijeme primitka intervencije, to bi se moglo usporediti s vremenom odlaska na intervenciju te bi se iz tih razlika i podataka mogla napraviti statistika. Cilj stranice sa statistikama bio bi samo za napredak rada vatrogasne stanice. Iz tih primjera mogla bi se vaditi rješenja ili poboljšanja koja bi utjecala na efikasnost rada stanice te bolju pripremljenost za intervencije.



Slika 31. Stranica koja prikazuje statistiku vatrogasne postaje

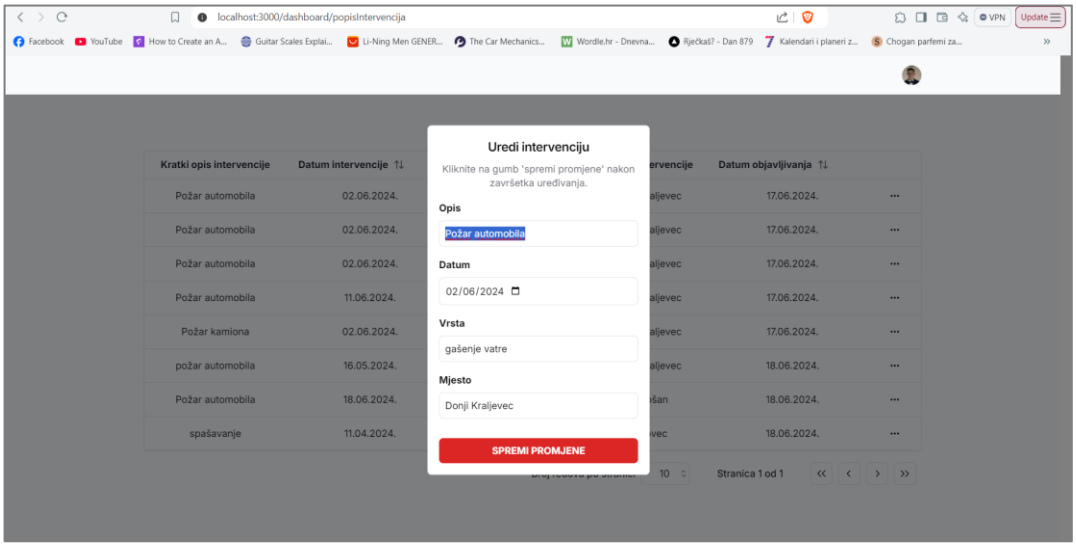
4.2.5. Stranica popisa intervencija

Pored opcije za pregled statistike vatrogasne postaje, nalazi se i opcija koja otvara novu stranicu s popisom intervencija. Stranica (koju je moguće vidjeti na slici 32.) sadrži tablicu koja prikazuje osnovne informacije o intervencijama (opis, datum, mjesto, vrsta). Uz to, implementirano je i prikaz tablice koja je napravljena s kombinacijom *JavaScripta* i PHP-a te *Apache* serverom. Zaključak nakon implementacije jest da je potrebno puno više pisanja i razrađivanja koda za nekakvu osnovnu funkcionalnost.

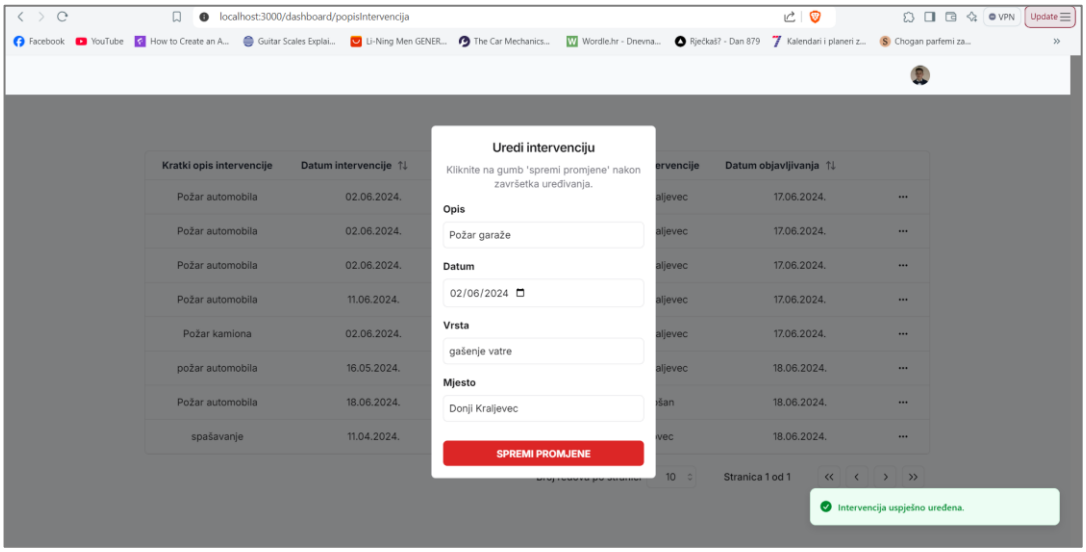


Slika 32. Stranica „popis Intervencija“ s tablicom svih intervencija

Ako je neka od intervencija krivo unesena (provjerava administrator), u zadnjem stupcu tablice ima mogućnost uređivanja ili kompletnog brisanja retka, odnosno intervencije. Opcije uređivanja i brisanja moguće je vidjeti gore na slici 33. Ako korisnik želi urediti intervenciju nakon odabira te opcije otvara se sučelje za uređivanje (prikazano na slici 33.). Administrator ima mogućnost mijenjanja svaku od opcija te nakon završetka uređivanja i spremanja promjena, javlja se „`toast`“ poruka koje ukazuje na uspješno uređivanje. Prikaz uspješnog uređivanja je na slici 34. Ako administrator smatra da intervencija nije valjana ima mogućnost brisanja. Kod brisanja se stranica ponovno učita i prikaže se ažurirana tablica.



Slika 33. Sučelje za uređivanje intervencije



Slika 34. „toast“ poruka nakon uspješnog uređivanja intervencije

5. Zaključak

Iako je još uvijek popularan temeljni način izrade web aplikacija preko „osnovnih“ alata (HTML, CSS, JavaScript), korištenje alata i tehnologija koje se vežu na temeljne alate je od velikog značaja. Internet je pun dinamičkih web aplikacija pa je i izrada dinamičnijih i kompleksnijih programskih rješenja, u minimalno kratkom roku, danas prioritet svakom poslodavcu. Svakim dodatnom tehnologijom programer pokušava smanjiti vrijeme izrade određenih funkcionalnosti.

React, uz ostale alate korištene u radu, zajedno stvara eko-sustav u kojem programer može biti efikasniji i može kreirati moderna programska rješenja. Na primjeru jednostavnije web aplikacije ta mogućnost dolazi do izražaja, kao i sama skalabilnost sustava. Web aplikaciju projekta je moguće nadograđivati na razne načine jer upravo taj eko-sustav sadrži odličnu podlogu za to.

Na samom radu prikazano je i dohvaćanje podataka upravo pomoću temeljnih alata (HTML, CSS, *JavaScript,* PHP). Zaključak koji se povlači jest da se može napraviti ista funkcionalnost kao i kod korištenja dodatnih tehnologija, no ostaje pitanje vremena koje potrebno za implementaciju.

Web aplikacija jest funkcionalnog karaktera i u njoj je prikazana većina elemenata moderne *full-stack* web aplikacije. Mogućnosti kao što su prijava na web aplikaciju, prijava intervencije, brisanje i uređivanje intervencije i upravljanje bazom podataka. Nekim dijelom aplikacija ovisi o vanjskim sustavima, no cilj je bio napraviti što više lokalnu aplikaciju kako bi se mogla koristiti u svim uvjetima i kako ne bi ovisila o drugim sustavima. Glavni cilj je bio prikazati upravljanje bazom podataka (dohvaćanje, slanje, uređivanje ili brisanje podataka iz tablica baze podataka) za evidenciju intervencija, uz implementaciju raznih pomagala administratora.

Zaključak rada jest da u svakoj kombinaciji alata, uz dobru podlogu i pripremu, može se postići velika skalabilnost web aplikacije. Jednostavna aplikacija kao na primjeru završnog rada može se proširiti sa novim mogućnostima koje ispunjavaju suvremene korisničke zahtjeve.

6. Literatura

[1] https://vatrogasci.zagreb.hr/default.aspx?id=1214, dostupno 02.08.2024.
[2] https://dvd-mala-subotica.spis.hvz.hr/2022, dostupno 02.08.2024.
[3] https://netbit.hr/web-aplikacija-ili-web-stranica-2/, dostupno 02.08.2024.
[4] Mikac, M. (2022). *P01 – Uvod WWW, HTML, CSS* [PDF].
[5] Mikac, M. 2023) * P4 - Protokoli aplikacijskog sloja - HTTP, FTP, e-mail...* [PDF]
[6] https://medium.com/@danamulder/a-brief-history-of-css-d27986eebc86, dostupno 04.08.2024.
[7] https://medium.com/@shahamisha012/what-is-javascript-5a0a74f553bc, dostupno 04.08.2024.
[8] https://developer.mozilla.org/en-US/docs/Web/API, dostupno 04.08.2024.
[9] https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model, dostupno 04.08.2024.
[10] https://dev.to/sidramaqbool/understanding-dom-nodes-a-comprehensive-guide-with-example-22m5, dostupno 05.08.2024.
[11] https://www.geeksforgeeks.org/difference-between-css-and-css3/, dostupno 05.08.2024.
[12] https://medium.com/@Infowind/the-history-of-full-stack-development-8f3b1844da9f, dostupno 05.08.2024.
[13] https://medium.com/quick-code/javascript-usage-statistics-2023-thatll-blow-your-mind-2af8bce5a4f7, dostupno 06.08.2024.
[14] https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs, dostupno 14.08.2024.
[15] https://legacy.reactjs.org/docs/faq-internals.html, dostupno 14.08.2024.
[16] https://react.dev/learn/writing-markup-with-jsx, dostupno 14.08.2024.
[17] https://nextjs.org/docs, dostupno 14.08.2024.
[18] https://nextjs.org/docs/app/building-your-application/rendering, dostupno 17.08.2024.
[19] https://nextjs.org/docs/app/building-your-application/caching, dostupno 17.08.2024.
[20] https://nextjs.org/docs/pages/building-your-application/routing, dostupno 17.08.2024.
[21] https://www.prisma.io/docs, dostupno 19.08.2024.
[22] https://group.miletic.net/hr/nastava/materijali/postgresql-sustav-za-upravljanje-bazom-podataka/, dostupno 19.08.2024.
[23] https://www.freecodecamp.org/news/what-is-node-js/, dostupno 20.08.2024.
[24] https://nextjs.org/docs/getting-started/installation, dostupno 20.08.2024.
[25] https://clerk.com/legal/dpa, dostupno 24.08.2024.
[26] https://blog.hubspot.com/blog/tabid/6307/bid/26866/9-must-haves-for-the-perfect-landing-page.aspx, dostupno 24.08.2024.

Sveučilište
Sjever

IZJAVA O AUTORSTVU
I
SUGLASNOST ZA JAVNU OBJAVU

Završni/diplomski rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, DAVID LEPEN *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor završnog rada pod naslovom PREGLED KORIŠTENJA MODERNIH ALATA ZA RAZVOJ WEB APLIKACIJA NA PRIMJERU JEDNOSTAVNE WEB APLIKACIJE PRIMJENJIVE U VARAŽDINSKOJ POŠTAJI *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student:
(upisati ime i prezime)

(vlastoručni potpis)

Sukladno Zakonu o znanstvenoj djelatnost i visokom obrazovanju završne/diplomske radove sveučilišta su dužna trajno objaviti na javnoj internetskoj bazi sveučilišne knjižnice u sastavu sveučilišta te kopirati u javnu internetsku bazu završnih/diplomskih radova Nacionalne i sveučilišne knjižnice. Završni radovi istovrsnih umjetničkih studija koji se realiziraju kroz umjetnička ostvarenja objavljuju se na odgovarajući način.

Ja, DAVID LEPEN *(ime i prezime)* neopozivo izjavljujem da sam suglasan s javnom objavom završnog (*obrisati nepotrebno*) rada pod naslovom PREGLED KORIŠTENJA MODERNIH ALATA ZA RAZVOJ WEB APLIKACIJA NA PRIMJERU JEDNOSTAVNE WEB APLIKACIJE PRIMJENJIVE U VARAŽDINSKOJ POŠTAJI *(upisati naslov)* čiji sam autor.

Student:
(upisati ime i prezime)

(vlastoručni potpis)