APLIKACIJA ZA PRIKUPLJANJE I ANALIZU PODATAKA KVALITETE ZRAKA

Krstulović, Dino

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Specijalistički diplomski stručni studij Elektrotehnike

DINO KRSTULOVIĆ

ZAVRŠNI RAD

APLIKACIJA ZA PRIKUPLJANJE I ANALIZU PODATAKA KVALITETE ZRAKA

Split, prosinac 2022.

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Specijalistički diplomski stručni studij Elektrotehnike

Predmet: Senzorske mreže

ZAVRŠNI RAD

Kandidat: Dino Krstulović

Naslov rada: Aplikacija za prikupljanje i analizu podataka kvalitete zraka

Mentor: dr. sc. Tonko Kovačević

Split, prosinac 2022.

Sadržaj

Sažetak

Aplikacija za prikupljanje i analizu podataka kvalitete zraka

U ovom radu biti će opisan razvoj web aplikacije za kontrolu kvalitete zraka. Određeno je 5 mjernih točaka u gradu Splitu prema kojima će se podaci prikazati tablično i grafički. Vrijednosti koje se mjere su temperatura, vlaga te postotoak vlage u zraku. Aplikacija "Air Quality Control" izrađena je u JavaScript programskom jeziku, HTML prezentacijskom jeziku te CSS stilskom jeziku. Za izradu sučelja aplikacije korištena je kombinacija nekoliko JavaScript biblioteka od kojih treba posebno izdvojiti React te MaterialUI biblioteke. Pozadinski servis temeljen je na Express aplikacijskom okruženju i Node.js okruženju za vrijeme izvođenja. Za pohranu podataka korišten je MongoDB program baze podataka. Tehnologije MongoDB, Express, React te Node.js zajedno čine takozvani MERN stog. Mjerenja koja su se koristila u izradi ovog rada svojom strukturom temelje se na mjerenjima kolege Ante Botice u njegovom završnom radu "Bežična senzorska mreža za mjerenje kvalitete zraka".

Ključne riječi: Javascript, HTML, CSS, React, Redux, MongoDB, web, aplikacija, podaci

Summary

Application for collection and analysis of air quality data

This paper will describe the development of a web application for air quality control. 5 measuring points in the city of Split have been determined, according to which the data will be presented tabularly and graphically. The values that are measured are temperature, humidity and the percentage of moisture in the air. "Air Quality Control" application is created in JavaScript programming language, HTML presentation language and CSS style language. A combination of several JavaScript libraries was used to create the application interface, of which React and MaterialUI libraries should be singled out. The backend service is based on the Express application environment and the Node.js runtime environment. The MongoDB database program was used for data storage. Technologies MongoDB, Express, React and Node.js together form the so-called MERN stack. The measurements that were used in the creation of this work are based on the measurements of my colleague Ante Botica in his final work "Wireless sensor network for measuring air quality".

Keywords: Javascript, HTML, CSS, React, Redux, MongoDB, web, application, data

1. Uvod

Razvijanje za web je započelo 1989. godine kada je Tim Berners-Lee izumio internet, tj WWW (engl. *World Wide Web*). Prve web stranice su bile jako jednostavne i nisu imale mnoge funkcionalnosti koje imaju moderne web stranice. Tadašnje web stranice uglavnom su se sastojale od nekakvog teksta te možda neke jednostavne tablice i slike. Današnje web stranice su više aplikacije nego statične web stranice kakve su se poznavale u 90-im godnama prošlog stoljeća. Razvojem modernih tehnologija kao što su JavaScript, React, CSS i mnoge druge razvoj web stranica i aplikacija ograničen je samo maštom programera i zahtjevima klijenata.

Bibiloteke i okruženja kao što je React omogućuju razvoj naprednih i "pametnih" komponenti koje se mogu koristiti više puta unutar koda i aplikacije. Na taj način skraćuje se vrijeme razvijanja aplikacije, štede se sredstva, lakše se držati standarda za razvijanje web aplikacija te je programiranje stranica i aplikacija jako intuitivno. CSS je omogućio stiliziranje i animiranje apsolutno svakog elementa HTML koda, od jednostavnog klizećeg teksta do animacija koje su prije bile nezamislive za jednu web stranicu.

Na pozadinskom servisu aplikacije biblioteke poput Node.js-a i Express-a omogućuju jako jednostavno kreiranje servera te povezivanjem s istim. Također kreiranje podataka, manipuliranje istima te njihovo spremanje jako je olakšano s programima i paketima kao što je MongoDB.

Moderne tehnologije također omogućuju povezivanje web aplikacija sa razno raznim tehnologijama kao što su na primjer senzori za mjerenje kvalitete zraka.

Sve te teme i još mnogo toga biti će detaljno obrađeno u narednim poglavljima, od osnovnih programskih jezika sve do bežičnih senzorskim mreža i moderne React aplikacije.

2. Razvojne tehnologije pozadinskog servisa aplikacije

Pozadinski servis (engl. *backend*) je serverska strana web aplikacije. Koristi se za pohranu i raspoređivanje podataka, a također osigurava da sve na klijentskoj strani web aplikacije radi kako treba. To je dio web aplikacije koji se s klijentske strane ne može vidjeti i s kojim klijent ne ostvaruje direktnu interakciju. Pozadinski servis je dio softvera koji ne dolazi u izravan kontakt s korisnicima. Korisnici neizravno pristupaju dijelovima i karakteristikama koje su razvili programeri pozadinskog servisa putem sučelja aplikacije. Aktivnosti, poput pisanja API-ja, stvaranja biblioteka i rada s komponentama sustava bez korisničkih sučelja ili čak sustava znanstvenog programiranja, također su uključene u rad pozadinskog servisa [1].

Za pisanje koda web aplikacije koriste se programski jezici. Programski jezici sastoje se od sintakse i semantike. Sintaksa predstavlja osnovna pravila i strukturu programskog jezika, dok semantika sintaksi daje značenje. Jezici koji se najčešće koriste za pisanje pozadinskog servisa web aplikacije su: Java, JavaScript, PHP, C++, C# te Python. Pozadinski servis i sučelje web aplikacije napravljene u sklopu ovog rada pisani su u JavaScript programskom jeziku. Više informacija o JavaScript programskom jeziku biti će navedeno u potpoglavlju 3.1.

Uz programske jezike za razvoj pozadinskog servisa aplikacije koriste se aplikacijska okruženja. Aplikacijsko okruženje je osnovna struktura na kojoj se temelji sustav aplikacije. Ono pruža gotove komponente ili rješenja koja se prilagođavaju kako bi se ubrzao razvoj aplikacije. Aplikacijsko okruženje može uključivati biblioteku, ali je definiran načelom inverzije kontrole. Kod tradicionalnog programiranja pisani kod poziva biblioteku kako bi pristupio kodu koji se može ponovno koristiti, dok uz korištenje inverzije kontrole okruženje poziva pisani kod samo kada je to potrebno [2]. Express je aplikacijsko okruženje korišteno u ovom projektu. Više informacija o Express okruženju biti će navedeno u potpoglavlju 2.1.

Da bi pozadinski servis web aplikacije bio kompletan, uz JavaScript i Express koriste se Node.js te MongoDB. Node.js je okruženje za vrijeme izvođenja JavaScript koda izvan preglednika, a MongoDB je baza podataka koja dokumente pohranjuje u obliku jako sličnom JSON formatu.

Za instalaciju biblioteka i paketa koristi se registar softvera naziva npm koji sadrži preko 800,000 paketa i biblioteka.

2.1. Express

Express se instalira u aplikaciju komandom:

```
>> npm i express
```

te ne zahtjeva dodatno podešavanje za uporabu. Verzija Express-a korištena u projektu je 4.18.2.

Express je minimalno i fleksibilno okruženje Node.js web aplikacije koji pruža robustan skup značajki za web i mobilne aplikacije. Uz mnoštvo HTTP uslužnih metoda i međuprograma koji su na raspolaganju, stvaranje robusnog API-ja uz Express brzo je i jednostavno. Express je sloj iznad Node.js-a koji pomaže pri upravljanju poslužiteljima i rutama. Glavne značajke Express-a su brzi razvoj na strani poslužitelja, posluživanje međusoftvera koji djeluje između aplikacije i mreže, rutiranje, stvaranje predložaka koda za višekratnu upotrebu te za lakše detektiranje i popravljanje pogrešaka u kodu. Express se brzo i lako povezuje s bazama podataka kao što je MongoDB.

Rutiranje se odnosi na to kako krajnje točke aplikacije odgovaraju na zahtjeve klijenta. Usmjeravanje je definirano pomoću metoda Express aplikacijskog objekta koje odgovaraju HTTP metodama zahtjeva. Postoji mnogo HTTP metoda i njihovih uporaba u Express okruženju, a najčešće korištene su navedene u tablici 2.1.

Tablica 2.1 Vrste HTTP metoda, odgovarajuće Express funkcije te upotreba

HTTP	Express	Upotreba
GET	router.get()	Dohvaćanje podataka s rute.
POST	router.post()	Slanje podataka na server za spremanje podataka na rutu.
PUT	router.put()	Slanje podataka na server za modificiranje podataka na ruti.
DELETE	router.delete()	Brisanje podataka na ruti.
HEAD	router.head()	Dohvaćanje zaglavlja s rute.

Ove metode usmjeravanja određuju funkciju povratnog poziva (ponekad zvane "funkcije obrađivača") koja se poziva kada aplikacija primi zahtjev prema navedenoj ruti (krajnjoj točki) i HTTP metodi. Drugim riječima, aplikacija "osluškuje" zahtjeve koji odgovaraju

navedenoj ruti i metodi, a kada otkrije podudaranje, poziva navedenu funkciju povratnog poziva. Zapravo, metode usmjeravanja mogu imati više od jedne funkcije povratnog poziva kao argumente. S višestrukim funkcijama povratnog poziva, važno je dati "next" kao argument funkciji povratnog poziva, a zatim pozvati "next()" unutar tijela funkcije kako bi se predala kontrola sljedećem povratnom pozivu [3].

Parametri rute imenovani su segmenti URL-a koji se koriste za hvatanje vrijednosti navedenih na njihovoj poziciji u adresi. Dohvaćene vrijednosti nalaze se u objektu koji se naziva "parametri zahtjeva" (req.params), gdje su parametri rute definirani kao ključevi u strukturi puta ("path"). Put je definiran kao mapa gdje je neki resurs pohranjen unutar projekta. Primjer jednostavne upotrebe Express metode u projektu vidljiv je na ispisu 2.1.

Ispis 2.1 Primjer get() Express funckije

```
router.get("/", async (req, res) => {
    await newDataArray.forEach(dataSet => location.create(dataSet));
    const data = await location.find();
    const responseObject = [];
    data.forEach(obj => {
        responseObject.push({
            id: obj.id,
            name: obj.name,
            position: obj.position,
            description: obj.description,
            type: obj.type,
            url: obj.url,
            measurementData: obj.measurementData
        })
    })
    res.json(responseObject)
});
```

U ovom dijelu koda vidi se primjena Express get() funkcije koja dohvaća podatke na ruti putu "/". Ključna riječ "async" označava da je funkcija asinkrona. Asinkrona funkcija kao rezultat izvršavanja vraća "obećanje (engl. *promise*)". To obećanje vraća vrijednost nakon što se operacija unutar funkcije izvrši ili vraća grešku ako se operacija ne izvrši. Parametri funkcije "req" i "res" predstavljaju zahtjev (engl. *request*) te odgovor (engl. *response*). Ključna riječ "await" uz "async" čini potpunu asinkronu funkciju te govori funkciji da iščekuje obećanu vrijednost. U ostatku funkcije se od varijable "newDataArray" kreira objekt "responseObject" koji se šalje kao odgovor na poziv funkcije "res.json (responseObject)" u JSON formatu. Konačni objekt za svaku lokaciju ima identifikator, ime, koordinate na mapi, opis, tip, adresu te mjerene podatke.

Unutar priložene funkcije može se primjetiti uporaba varijable "router". Ta varijabla je zapravo klasa "express.Router()" koja se koristi za stvaranje i rukovanje modularnim rutama. Instanca napravljena putem te klase je potpuni posredni software (engl.

middleware) i sustav rutiranja te se zbog toga često naziva “mini-aplikacija”[3]. Posredni softveri su funkcije koje imaju pristup objektu zahtjeva (“req”), objektu odgovora (“res”) te sljedećoj funkciji u ciklusu zahtjev-odgovor aplikacije. Sljedeća funkcija je funkcija u Express usmjerivaču koja, kada se pozove, izvršava posredni softver koji nasljeđuje trenutni posredni softver.

U ovoj funkciji također se koristi i metoda odgovora(engl. *response method*) res.json(). Metode odgovora se koriste zajedno s Express metodama kako bi se zatvorio ciklus funkcije poziv-odgovor. U tablici 2.2 mogu se vidjeti metode odgovora te opis istih.

Tablica 2.2 Express metode odgovora

Express metoda	Upotreba
res.download()	Pokretanje preuzimanja datoteke.
res.end()	Završavanje procesa odgovora.
res.json()	Slanje JSON objekta kao odgovor.
res.jsonp()	Slanje JSON objekta kao odgovor s podrškom za JSONP.
res.redirect()	Preusmjeravanje zahtjeva.
res.render()	Prikazivanje predložaka odgovora.
res.send()	Slanje različitih tipova odgovora.
res.sendFile()	Slanje datoteke kao tok okteta.
res.sendStatus()	Postavljanje koda statusa odgovora i slanje njegove reprezentacije niza kao odgovora.

2.2. Node.js

Node se instalira u aplikaciju komandom:

```
>> npm i node
```

te ne zahtjeva dodatno podešavanje za uporabu. Verzija Node.js-a korištena u projektu je v14.17.5.

Node.js je okruženje za vrijeme izvođenja JavaScript koda izvan preglednika koje je softver otvorenog koda i višeplatformski softver. NodeJS nije okvir i nije programski jezik. Node.js se često koristi za izgradnju pozadinskih usluga poput API-ja kao što su mrežne i mobilne aplikacije. U prdukciji ga koriste neke velike tvrtke kao što su Paypal, Uber, Netflix, Walmart, itd [1].

Kao asinkrono okruženje za vrijeme izvođenja JavaScript koda, Node.js je dizajniran za izgradnju skalabilnih mrežnih aplikacija. Skalabilna aplikacija je aplikacija koja je prilagođena dugotrajnom unaprijeđivanju te rastu, dodavanju novog i modificiranju starog koda i funkcionalnosti te koja je prilagođena za obradu velikog broja korisnika i zahtjeva prema serveru.

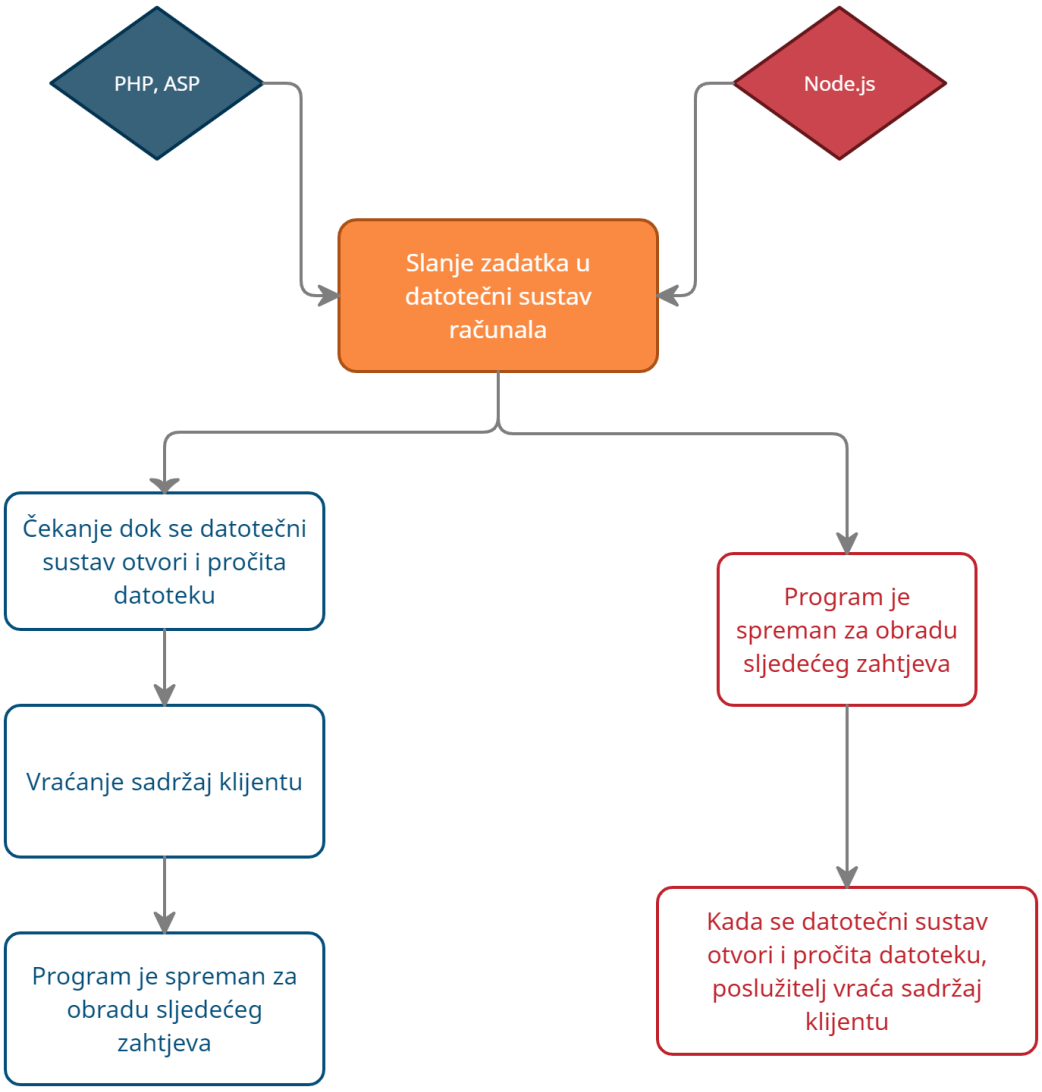
Node.js pokreće V8 JavaScript mehanizam, jezgru Google Chrome-a, izvan preglednika. To omogućuje Node.js-u da bude vrlo učinkovit. Node.js aplikacija radi u jednom procesu, bez stvaranja nove niti za svaki zahtjev. Node.js pruža skup asinkronih I/O (engl. *input/output*) primitiva u svojoj standardnoj biblioteci koje sprječavaju blokiranje JavaScript koda i općenito su biblioteke u Node.js-u napisane korištenjem neblokirajućih paradigmi, čineći ponašanje blokiranja iznimkom, a ne normom. Kada Node.js izvodi I/O operaciju, poput čitanja s mreže, pristupa bazi podataka ili datotečnom sustavu, umjesto blokiranja niti i trošenja procesorskog ciklusa na čekanje, Node.js će nastaviti s operacijama kada se odgovor vrati. To omogućuje Node.js-u da rukuje tisućama istodobnih veza s jednim poslužiteljem bez uvođenja tereta upravljanja istovremenošću niti, što bi moglo biti značajan izvor grešaka. Node.js ima jedinstvenu prednost jer pri izradi sučelja JavaScript aplikacije za preglednik može se pisati kod na strani poslužitelja uz kod na strani klijenta bez upotrebe potpuno drugačijeg jezika [4].

Ispis 2.2 Primjer “createServer” Node.js funckije

```
const http = require('http');
const app = express();

http.createServer(app).listen((3030), () => {
    console.log("Server is running on port 3030!")
});
```

U Ispisu 2.2 vidi se jednostavan primjer Node.js funkcije za kreiranje servera. Node.js slijedi sustav modula CommonJS, a ugrađena funkcija “require()” je najlakši način za uključivanje modula koji postoje u zasebnim datotekama. Metoda “http.createServer()” stvara poslužitelj koji poziva varijablu “app” kada god stigne zahtjev, dok “server.listen(3030)” poziva metodu slušanja (engl. *listen*) koja uzrokuje da poslužitelj čeka dolazne zahtjeve na navedenom portu 3030.

Uobičajen zadatak web poslužitelja može biti otvaranje datoteke na poslužitelju i vraćanje sadržaja klijentu. Prednost Node.js-a naspram PHP-a ili ASP-a je u tome što Node.js, kada dobije zahtjev, eliminira čekanje na odgovor sustava već nastavlja dalje sa sljedećim zahtjevom.



Slika 2.1 Usporedba obrade zahtjeva za pristup datoteci kod PHP-a ili ASP-a te Node.js-a

2.3. MongoDB

MongoDB se instalira u aplikaciju na malo drugačiji način nego Express te Node.js. MongoDB baza podataka može instalirati komandom:

```
>> npm i mongodb
```

te se tim načinom instalacije dobiju neke prednosti kao fleksibilne sheme, sadržavanje ogromnog broja podataka, spremanje različitih dokumenata u kolekcije, itd. Ovaj projekt u svojoj naravi ne zahtjeva kompleksnu strukturu baze podataka i koristi razne prednosti Mongoose-a kod definiranja sheme objekta te se zbog toga za rad s bazom podataka MongoDB instalira paket mongoose:

```
>> npm i mongoose
```

Verzija Mongoose paketa korištena u projektu je 6.6.5. MongoDB je sustav za upravljanje bazom podataka orijentiran na dokumente koji pohranjuje podatke u formatu BSON dokumenata. To je baza podataka tipa NoSQL (engl. *Not-only SQL*) koja korisnicima omogućuje pohranjivanje ogromnih količina podataka. Za razliku od SQL baza podataka gdje su podaci pohranjeni u obliku tablica, NoSQL baza podataka učinkovito pohranjuje podatke kao dokumente unutar zbirki.

Baza podataka nije ništa drugo nego organizirana zbirka strukturiranih podataka ili informacija koje su općenito pohranjene u računalu. Baza podataka obično komunicira sa sustavom za upravljanje bazom podataka DBMS (engl. *Database Management System*) kako bi korisnik mogao kontrolirati i upravljati podacima koji su u njoj pohranjeni. DBMS nije ništa drugo nego softver ili sučelje koji omogućuje potpunu kontrolu nad podacima kao što su stvaranje, čitanje, uređivanje, brisanje, itd. Također olakšava sustave kontrole pristupa i druge usluge kao što su sigurnosne kopije, izvješćivanje, pohrana, sigurnost i više.

Mongoose je zapravo maper dokumenata objekta ODC (engl. *Object Document Mapper*). Jednostavnim rječnikom, program za preslikavanje dokumenata objekata ODM (engl. *Object Document Mapper*) preslikava objekte s bazom podataka koja se temelji na dokumentu kao što je MongoDB. Maper dokumenata objekta omogućuje korisniku da definira shemu za dokumente unutar zbirki. Korisnicima omogućuje dobro strukturiranje dokumenata radi boljeg predstavljanja. ODM također omogućuje korisnicima jednostavno dodavanje novih svojstava i polja. Također se naziva i alat za modeliranje objekata. Izgrađen je na temelju MongoDB upravljačkog programa za MongoDB i Node.js. Omogućuje jednostavno modeliranje podataka, definiraje sheme za dokumente unutar zbirke i upravljajnje odnosima između podataka. Mongoose omogućuje korisnicima prikladnu izradu i upravljanje podacima u MongoDB-u. Iako je moguće upravljati podacima, definirati sheme, itd. pomoću MongoDB API-ja, prilično je teško to učiniti. Stoga je Mongoose mnogo praktičnija opcija [5].

Tablica 2.3 Značajke MongoDB-a i Mongoose-a

MongoDB	Mongoose
Fleksibilne sheme.	Ulančane funkcije čine kod fleksibilnim i čitljivim.
Sadrži ogromne količine podataka.	Uklanja potrebu za korištenjem imenovanih kolekcija.
Jednostavan za skaliranje i promjenu.	Obavlja skupne zadatke uključivanja zadanih vrijednosti za svojstva i provjere valjanosti podataka.
Nema shemu jer omogućuje pohranu različitih dokumenata u jednu zbirku.	Jednostavnije definiranje sheme.
Snažan, dinamičan i dubok upit.	Značajke poput pretvaranja tipa, provjere podataka, izrade upita i više.

Za pravilnu upotrebu Mongoose paketa korisnik prvo treba definirati Mongoose shemu za kolekciju. Mongoose shema definira strukturu dokumenta, početne vrijednosti, tipove svojstava, ključeve, validatore, itd. U ispisu 2.3 može se vidjeti Mongoose shema korištena u projektu. Mongoose shema lokacije sastoji se od 7 osnovnih ključeva: “id” (identifikator), “name” (ime lokacije), “position” (geografska duživa i širina), “description” (kratki opis lokacije), “type” (tip lokacije), “url” (web adresa lokacije) te “measurementData” (izmjereni podaci). Ključ “measurementData” unutar sebe ima 4 podključa: “date” (datum mjerenja), “temperature” (temperature zraka), “humidity” (vlažnost zraka), te “CO2” (količina ugljikovog dioksida u zraku). Svi podaci u shemi su ili tipa “String” (skup karaktera) ili “Number” (broj). Mongoose shema se izvodi iz datoteke kao Mongoose model. Mongoose model je omotač (engl, *wrapper*) za Mongoose shemu.

Ispis 2.3 Mongoose shema za lokaciju

```
const mongoose = require("mongoose");

const locationSchema = new mongoose.Schema({
    id: {
        type: String,
        required: true
    },
    name: {
        type: String,
        required: true
    },
    position: {
        type: [Number],
        required: true
    },
    description: {
        type: String,
        required: true
    },
    type: {
        type: String,
        required: true
    },
    url: {
        type: String,
        required: true
    },
    measurementData: {
        date: {
            type: [String],
            required: false
        },
        temperature: {
            type: [Number],
            required: false
        },
        humidity: {
            type: [Number],
            required: false
        },
        co2: {
            type: [Number],
            required: false
        }
    }
})

module.exports = mongoose.model("Location", locationSchema);
```

Nakon što se Mongoose shema izveze kao Mongoose model s njom treba nešto i napraviti kako bi se ona pohranila u bazu podataka. To se može vidjeti u ispisu 2.4.

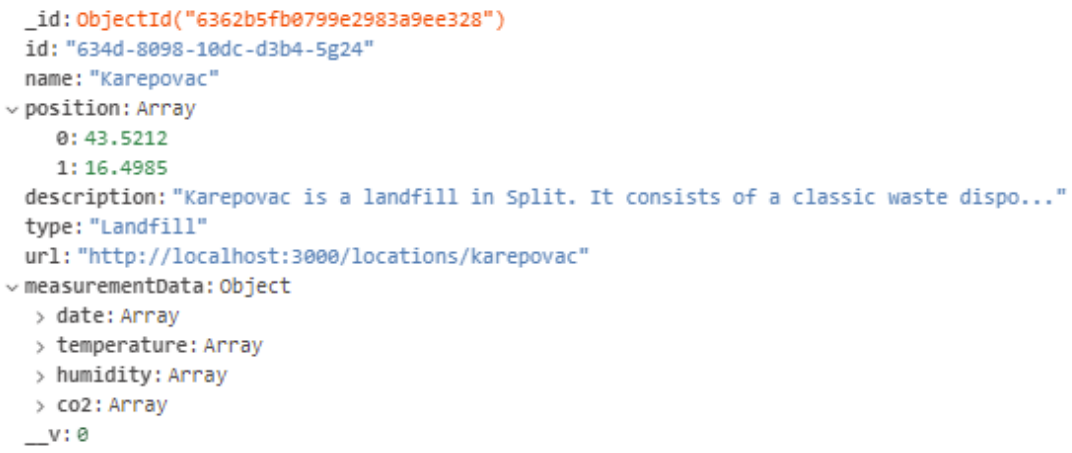
Ispis 2.4 Primjena Mongoose sheme i konekcija s MongoDB bazom podataka

```
const mongoose = require("mongoose");

mongoose.connect("mongodb://localhost:27017/aqc");
const db = mongoose.connection;
db.on("error", (error) => console.error(error))
db.once("open", () => console.log('Connected to database!'));
```

Mongoose paket ako već ne postoji pravi novu MongoDB bazu podataka na lokalnoj adresi "localhost:27017/aqc", ako baza podataka postoji onda se samo priključuje na istu. Za provjeru konekcije koristi se varijabla "db". Ako baza podataka kao odgovor na spajanje pošalje pogrešku konzola će ispisati tu pogrešku, ako se zahtjev uspješno izvrši tada će se u konzoli ispisati "Connected to database!".

Kada je konekcija uspješno uspostavljena tada Mongoose u napravljenu bazu podataka šalje setove podataka prema definiranoj shemi. Taj process se može vidjeti u ispisu 2.1. Varijabla "newDataArray" sadrži predefinirani niz podataka u kojima se nalaze sve potrebne vrijednosti za sve definirane ključeve u Mongoose shemi. Iz tog niza podataka se za svaki objekt unutar niza kreira set podataka koji se prosljeđuje u Mongoose shemu te se na taj način popunjava baza podataka.

Za jednostavan pregled baza podataka napravljlenih putom Mongoose-a i MongoDb-a koristi se program MongoCompass. On funkcionira tako da se spaja na adresu baze podataka koja je u ovom slučaju "localhost:27017/aqc" te ispisuje sve podatke koje je pronašao na toj adresi kao kolekcije (engl. *collection*). Prijmjer jedne kolekcije vidljiv je na slici 2.2.



Slika 2.2 Primjer MongoDB kolekcije

Svaka kolekcija sadrži 2208 mjerenja za svaki od ključeva iz “measurementData” objekta. Podaci su nisu stvarna mjerenja već su izmišljeni podaci napravljeni prema strukturi podataka mjerenih u sklopu rada pod mentorstvom dr.sc. Tonka Kovačevića te autora Ante Botice “Bežićna sentorska mreža za mjerenje kvalitete zraka” iz rujna 2019. godine. Mjerenja su smještena u vremenskom periodu od početka svibnja 2022. godine do kraja srpnja 2022. godine. Korišteno je pet različitih lokacija u gradu Splitu: Karepovac, Marjan, Poljud, Vranjic i Žnjan. Vremenski razmak između svakog mjerenja je 1 sat te su mjerene vrijednosti temperatura zraka prikazana u celzijusovim stupnjevima, vlažnost zraka prikazana u postotcima te količina ugljikovog dioksida u zraku prikazana u količini čestica ugljikovog dioksida u milijun čestica zraka ppm(engl. *parts per million*).

3. Razvojne tehnologije sučelja aplikacije

Sučelje aplikacije (engl. *frontend*) je sve što korisnik vidi i s čime komunicira kada klikne na poveznicu ili upiše web adresu. Web adresa je također poznata kao URL (engl. *Uniform Resource Locator*) i govori koja se web stranica treba učitati i pojaviti u pregledniku. To je dio web aplikacije na strani klijenta. Sučelje je sve vizualno s čime korisnik dolazi u kontakt, svi dijelovi aplikacije s kojima je u izravnoj interakciji, sav sadržaj i stilovi, gumbi i različiti efekti prije nego što korisnik klikne neki element, obrasci za kontakt s raznim poljima za unos, okvirima za pretraživanje i padajućim izbornicima, izgledi, tekst i boje elemenata, slike, videozapisi, itd. Također je važno koliko se brzo web stranica učitava, koliko je lako koristiti se njome i koliko je dostupna osobama s invaliditetom, koliko je aplikacija prilagođena za različite uređaje. Aplikacijsko sučelje su svi dijelovi web aplikacije koji stvaraju njezin izgled i dojam [6].

Sučelje aplikacije je prezentacijski sloj, to je dio aplikacije koji korisnik može vidjeti, kao na primjer grafičko korisničko sučelje GUI (engl. *Graphical User Interface*). Obilježja kvalitetnog sučelja aplikacije su:

- korisnik bi trebao odmah razumjeti cilj i upotrebu aplikacije,
- aplikacija mora biti visoko pozicionirana unutar tražilica SEO (engl. *Search Engine Optimization*) standard,
- aplikacija treba imati responzivni dizajn, tj da funkcionira na različitim uređajima,
- sadržaj, izgled i dizajn bi trebali biti atraktivni,
- korištenje ikona pomaže korisniku u snalaženju u aplikaciji,
- unikatnost, ali i profesionalnost i
- brzina učitavanja aplikacije.

U apliakciji “Air quality control” za izradu sučelja korišteno je nekoliko jezika, alata, paketa te biblioteka od kojih je potrebno istaknuti:

- JavaScript,
- HTML,
- CSS,
- React,
- Redux,
- MaterialUI te
- Axios.

U sljedećih nekoliko poglavlja sve navedene tehnologije biti će detaljno opisane

3.1. JavaScript

JavaScript ili skraćeno JS je lagani, interpretirani te pravodobno kompilirani programski jezik s klasnim funkcijama. Iako je najpoznatiji kao skriptni jezik za web stranice, koriste ga i mnoga okruženja bez preglednika, poput Node.js-a, Apache CouchDB-a i Adobe Acrobat-a. JavaScript je dinamički jezik temeljen na prototipu, s više paradigmi i jednom niti, koji podržava objektno orijentirane, imperativne i deklarativne (npr. funkcionalno programiranje) stilove. Standardi za JavaScript su specifikacija jezika ECMAScript (ECMA-262) i specifikacija API-ja za internacionalizaciju ECMAScript (ECMA-402).

JavaScript je višeplatformski, objektno orijentirani skriptni jezik koji se koristi da web stranice i web aplikacije budu interaktivne (npr. imaju složene animacije, gumbe na koje se može kliknuti, skočne izbornike, itd.). Postoje i naprednije verzije JavaScripta na strani poslužitelja, kao što je Node.js, koji omogućuju da se web stranici doda više funkcionalnosti od preuzimanja datoteka (kao što je komunikacija u stvarnom vremenu između više računala). Unutar okruženja poslužitelja (npr. web preglednika), JavaScript se može povezati s objektima svog okruženja kako bi omogućio programsku kontrolu nad njima.

JavaScript sadrži standardnu biblioteku objekata, kao što su ”Array”, “Date” i ”Math”, te osnovni skup jezičnih elemenata kao što su operatori, kontrolne strukture i izjave. Jezgra JavaScripta može se proširiti za različite svrhe nadopunjavanjem dodatnim objektima:

- JavaScript na strani klijenta proširuje temeljni jezik dobavljanjem objekata za kontrolu preglednika i njegovog Objektnog modela dokumenta (engl. *Document Object Model*). Na primjer, proširenja na strani klijenta omogućuju aplikaciji postavljanje elemenata na HTML obrazac i reagiranje na korisničke događaje kao što su klikovi mišem, unos obrasca i navigacija stranicom.
- JavaScript na strani poslužitelja proširuje temeljni jezik dobavljanjem objekata relevantnih za pokretanje JavaScript-a na poslužitelju. Na primjer, proširenja na strani poslužitelja omogućuju aplikaciji komunikaciju s bazom podataka, pružanje kontinuiteta informacija od jednog do drugog pozivanja aplikacije ili izvođenje manipulacija datotekama na poslužitelju.

To znači da u pregledniku JavaScript može promijeniti izgled web stranice (DOM). I, isto tako, Node.js JavaScript na poslužitelju može odgovoriti na prilagođene zahtjeve poslane kodom koji se izvršava u pregledniku [7].

Ponekad se JavaScript i Java-u zbrka kao istu stvar, ali iako su u mnogo čemu ti jezici slični također imaju nekoliko velikih razlika.

Tablica 3.1 Razlike između JavaScript-a i Java-e

JavaScript	Java
Objektno orijentirano jezik, što znači da nema razlike između vrsta objekata. Nasljeđivanje se vrši kroz mehanizam prototipa, a svojstva i metode mogu se dinamički dodati bilo kojem objektu.	Jezik baziran na klasama, što znači da su objekti podijeljeni u klase i instance sa svim nasljeđivanjem kroz hijerarhiju klasa. Klase i instance ne mogu imati svojstva ili metode dodane dinamički.
Tipovi varijabli podataka nisu deklarirani, već se koriste općenite deklkaracije (const, var, let)	Tipovi varijabli podataka moraju biti deklarirani (int, double, string, itd.)
Ne može automatski upisivati na disk za pohranu	Može automatski upisivati na disk za pohranu

JavaScript je standardiziran u Ecma International — europskom udruženju za standardizaciju informacijskih i komunikacijskih sustava (ECMA je prije bila kratica za "European Computer Manufacturers Association") za isporuku standardiziranog, međunarodnog programskog jezika temeljenog na JavaScript-u. Ova standardizirana verzija JavaScripta, nazvana ECMAScript, ponaša se na isti način u svim aplikacijama koje podržavaju standard. Tvrtke mogu koristiti otvoreni standardni jezik za razvoj svoje implementacije JavaScripta. ECMAScript standard dokumentiran je u ECMA-262 specifikaciji.

Standard ECMA-262 također je odobrio Međunarodna organizacija za standardizaciju ISO (engl. *International Standardization Organization*) kao ISO-16262. Specifikaciju se također može pronaći na službenoj web stranici Ecma International. ECMAScript specifikacija ne opisuje DOM, koji je standardizirao World Wide Web Consortium (W3C) i/ili WHATWG (engl. *Web Hypertext Application Technology Working Group*). DOM definira način na koji su objekti HTML dokumenta izloženi skripti [7].

U ispisu 3.1 može se vidjeti upotreba JavaScript koda za formatiranje datuma i vremena iz unix formata u standardni format.

Ispis 3.1 Formatiranje datuma i vremena iz unix u standardni format

```
let unixTime = 1651356000000;
let dates = [];
const numberOfMeasurements =
dataArray[0].measurementData.temperature.length;

for (let i = 0; i < numberOfMeasurements; i++) {
    const formattedTime = new Date(unixTime);
    let year = formattedTime.getFullYear();
    let month = formattedTime.getMonth() + 1;
    if (month < 10) month = '0' + month;
    let date = formattedTime.getDate();
    if (date < 10) date = '0' + date;
    let hour = formattedTime.getHours();
    if (hour < 10) hour = '0' + hour;
    let min = formattedTime.getMinutes();
    if (min < 10) min = '0' + min;
    let time = year + '-' + month + '-' + date + ' ' + hour + ':' + min;
    dates.push(time);
    unixTime = unixTime + 3600 * 1000;
}
```

Varijabla “unixTime” ima vrijednost datuma i vremena “01.05.2022. 00:00” u unix formatu predstavljenu milisekundama koje su protekle od standardiziranog datuma “01.01.1970.”. For petlja se izvodi onoliko puta koliko ima mjerenja neke vrijednosti što predstavlja varijabla “numberOfMeasurements”. Unutar for petlje se radi nova varijabla “formattedTime” koja iz unix formata vrijeme pretvara u format “Sun May 01 2022 00:00:00 GMT+0200 (srednjoeuropsko ljetno vrijeme)”. Iz tog formata se uzimaju vrijednosti za godinu, mjesec, dan u mjesecu, sat te minute putem JavaScript metoda “getFullYear()”, “getMonth()”, “getDate()”, “getHours()” i “getMinutes()”. Također kod mjeseca, dana u mjesecu, sati i minuta se za vrijednosti manje od 10 pridodaje “0” kako bi se dobio željeni format. Kada se dobiju vrijednosti za svaku oznaku vremena one se u varijabli “time” spremaju u željenom formatu te se metodom “dates.push(time)” dodaju u niz podataka koji sadržava sve datume mjerenja. Varijabla “unixTime” se inkrementira za jedan sat te se ponovno izvršava idući ciklus for petlje. Kada se za sva mjerenja izvrši for petlja tada su u varijabli “dates” pohranjeni svi datumi mjerenja.

3.2. HTML

HTML (engl. *HyperText Markup Language*) najosnovniji je građevni blok web stranica i aplikacija. Definira značenje i strukturu web sadržaja. Hipertekst se odnosi na veze koje povezuju web stranice jednu s drugom, bilo unutar jedne web stranice ili između web stranica. Veze su temeljni aspekt web stranica i aplikacija.

HTML koristi oznake (engl. *markup*) za označavanje teksta, slika i drugog sadržaja za prikaz u web pregledniku. U tablici 3.2 mogu se vidjeti neke od mnogih HTML oznaka za posebne elemente.

Tablica 3.2 HTML oznake za posebne elemente

Oznaka	Uloga
<head>	Spremnik za metapodatke.
<title>	Definira naslov dokumenta koji se prikazuje u naslovnoj traci preglednika ili na kartici stranice.
<body>	Definira tijelo dokumenta.
<header>	Predstavlja spremnik za uvodni sadržaj ili skup navigacijskih veza.
<footer>	Definira podnožje za dokument ili odjeljak.
<p>	Definira odlomak.
	Definira odjeljak u HTML dokumentu.
	Generički ugrađeni spremnik za fraziranje sadržaja.
< img>	Koristi se za umetanje slike u HTML stranicu.
	Služi za grupiranje zbirke predmeta koji nemaju numerički poredak, a njihov redoslijed na listi je besmislen.
	Definira stavku popisa.

HTML element je odijeljen od drugog teksta u dokumentu oznakama (engl. *tag*), koji se sastoje od naziva elementa okruženog s "<" i ">" zagradama. Naziv elementa unutar oznake ne razlikuje velika i mala slova. Odnosno, može se pisati velikim, malim slovima ili mješavinom. Na primjer, oznaka <title> može se napisati kao <Title>, <TITLE> ili na bilo koji drugi način. Međutim, konvencija i preporučena praksa je pisanje oznaka malim slovima [8].

U ispisu 3.2 se može vidjeti index.html datoteka koja se napravi kada se pokrene skripta

```
>> npx create-react-app "app name"
```

kojom se kreira nova React aplikacija.

Ispis 3.2 Datoteka index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1"
/>
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <link rel="stylesheet"
href="https://unpkg.com/leaflet@1.8.0/dist/leaflet.css"
    integrity="sha512-
hoalWLoI8r4UszCkZ5kL8vayOGVae1oxXe/2A4AO6J9+580uKHDO3JdHb7NzwwzK5xr/Fs0W4
0kiNHxM9vyTtQ=="
    crossorigin=""/>
    <script src="https://unpkg.com/leaflet@1.8.0/dist/leaflet.js"
    integrity="sha512-
BB3hKbKWOc9Ez/TAwyWxNXeoV9c1v6FIeYiBieIWkpLjauysF18NzgR1MBNBXf8/KABdlkX68
nAhlwcDFLGPCQ=="
    crossorigin=""></script>
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root">
      <div id="map"></div>
    </div>
  </body>
</html>
```

Ta datoteka je glavna HTML datoteka aplikacije koja uključuje React kod i pruža kontekst za renderiranje React-a. Točnije, uključuje oznaku unutar koje će se pojaviti React aplikacija. Ovo se također naziva točka montiranja za React aplikaciju. Navedena HTML oznaka je dio HTML koda unutar kojeg će se nalaziti cijela struktura i kod React aplikacije.

3.3. CSS

CSS (engl. *Cascading Style Sheets*) je stilski jezik koji se koristi za opisivanje prezentacije dokumenta napisanog u HTML-u ili XML-u (uključujući XML dijalekte kao što su SVG, MathML ili XHTML). CSS opisuje kako se elementi trebaju prikazati na ekranu, na papiru, u govoru ili na drugim medijima. On je jedan od temeljnih jezika otvorenog web-a i standardiziran je za sve web preglednike prema specifikacijama W3C. Prethodno se razvoj različitih dijelova CSS specifikacije odvijao sinkrono, što je omogućilo verziju najnovijih preporuka. Postoje stare verzije CSS1, CSS2.1 i CSS3, ali u modernom CSS-u postoji samo naziv CSS koji nema brojeve verzija.

Nakon CSS-a 2.1, opseg specifikacije značajno se povećao i napredak na različitim CSS modulima počeo se toliko razlikovati da je postalo učinkovitije razvijati i objavljivati preporuke odvojeno po modulu. Umjesto verzije CSS specifikacije, W3C sada povremeno radi snimku najnovijeg stabilnog stanja CSS specifikacije i napretka pojedinačnih modula. CSS moduli sada imaju brojeve verzija ili razine, kao što je CSS Color Module Level 5.

Ispis 3.3 CSS kod u datoteci App.css

```
@import url('https://fonts.googleapis.com/css2?family=Lato&display=swap')
* {
  box-sizing: border-box;
  margin: 0;
  padding: 0;
  font-family: 'Lato', sans-serif;
}
.home, .about {
  display: flex;
  height: 90vh;
  align-items: center;
  justify-content: center;
  font-size: 3rem;
}
main {
  background-color: #c2c2c2;
}
.leaflet-container {
  width: 100vw;
  height: 85.5vh;
}
.landing-page-container {
  height: 87vh;
  background-color: #c2c2c2;
}
.item1 {
  height: 50vh;
}
```

U datoteci App.css, koja je vidljiva u ispisu 3.3, definirani su stilovi nekoliko bitnih elemanta "Air Quality Control" aplikacije. Simbol "*" označava sve elemente u aplikaciji te im dodjeljuje 4 svojstva:

- "box-sizing: border-box" – ispuna ("padding") i obrub ("border") uključeni su u širinu i visinu,
- "margin: 0" – sve margine imaju vrijednost nula,
- "padding: 0" – sve ispune imaju vrijednost nula te
- "font-family: 'Lato', sans-serif" – obitelj fontova je "sans-serif", a tip fonta je "Lato".

U ostatku CSS koda postavljaju se stilovi i svojstva za ostale dijelove aplikacije, kao na primjer za elemente s klasama "home" i "about" za stranice aplikacije "Home" i "About", postavlja se prikaz, visina stranice, veličina slova te se centreiraju elementi koji se u HTML strukturi nalaze ispod elemenata s tim klasama te se ti elementi zovu djeca (engl. *children*).

Što se tiče strukture CSS koda ona se može podijeliti na dva dijela. Prvi dio je selektor, tj. način na koji se dohvaća neki HTML element. To se najčešće vrši putem klase ili identifikatora, ali može se dohvaćati i sve elemente koje imaju HTML oznaku, na primjer " " elemente koji imaju neke definirane atribute, itd.

U praksi je kod React aplikacija koje koriste neke dodatne komponentne biblioteke, preporučljivo da se koristi što manje "čistog" CSS-a, a da se više koriste originalne komponente biblioteke uz njihova svojstva i stilove.

3.4. React

React.js je biblioteka otvorenog koda (engl. *open source library*) za JavaScript i biblioteka koju je razvio Facebook. Koristi se za brzu i učinkovitu izgradnju interaktivnih korisničkih sučelja i web aplikacija sa znatno manje koda nego što bi se to postiglo koristeći samo JavaScript. U React-u aplikacije se razvijaju stvaranjem komponenti za višekratnu upotrebu koje se mogu zamisliti kao neovisni blokovi. Ove komponente su pojedinačni dijelovi konačnog sučelja, koji, kada se sastave, čine cjelokupno korisničko sučelje aplikacije. Primarna uloga Reacta u aplikaciji je rukovanje slojem pogleda (engl. "view") te aplikacije, što označava "V" u MVC (engl. *Model View Controller*), pružajući najbolje i najučinkovitije izvršenje renderiranja. Umjesto da se bavi cijelim korisničkim sučeljem kao jednom jedinicom, React potiče da se razdvoji ova složena sučelja u pojedinačne komponente za višekratnu upotrebu koje čine građevne blokove cijelog sučelja. Radeći to, React kombinira brzinu i učinkovitost JavaScript-a s učinkovitijom metodom manipuliranja DOM-om za brže renderiranje web stranica i stvaranje vrlo dinamičnih i responzivnih web aplikacija [10].

React čini stvaranje interaktivnih korisničkih sučelja bezbolnim. Za svako stanje u aplikaciji može se dizajnirati jednostavne prikaze, a React će učinkovito ažurirati i prikazati prave komponente kada se podaci promijene. Deklarativni pogledi čine kod predvidljivijim i lakšim za otklanjanje pogrešaka. React omogućuje gradnju enkapsuliranih komponenti koje upravljaju vlastitim stanjem, a zatim se od njih mogu sastaviti složena sučelja. Budući da je logika komponente napisana u JavaScript-u umjesto u predlošcima, može se jednostavno proslijediti bogate podatke kroz aplikaciju i zadržati stanje izvan DOM-a. Nove značajke u Reactu mogu se razvijati bez ponovnog pisanja postojećeg koda neovisno o tehnologijama koje se koriste. React također može renderirati na poslužitelju koristeći Node.js i pokretati mobilne aplikacije koristeći React Native [11].

Obično korisnik web stranicu traži upisivanjem njenog URL-a u web preglednik. Korisnikov preglednik tada šalje zahtjev za tu web stranicu, koju preglednik prikazuje. Ako korisnik klinke vezu na toj web-stranici da bi otišao na drugu stranicu na web-mjestu, poslužitelju se šalje novi zahtjev za dobivanje te nove stranice. Ovaj obrazac učitavanja naprijed-natrag između preglednika i poslužitelja nastavlja se za svaku novu stranicu ili resurs kojem korisnik pokuša pristupiti na web lokaciji. Ovaj tipični pristup učitavanju web lokacija dobro funkcionira, ali učitavanje cijele web stranice naprijed i natrag bilo bi suvišno i stvorilo bi loše korisničko iskustvo. Osim toga, kada se podaci mijenjaju u tradicionalnoj JavaScript aplikaciji, potrebno je ručno manipulirati DOM-om da bi se te promjene odrazile. Moraju se identificirati koji su se podaci promijenili te ažurirati DOM da odražava te promjene, što rezultira ponovnim učitavanjem cijele stranice. React ima drugačiji pristup dopuštajući korisniku da napravi ono što je poznato kao jednostranična aplikacija ili SPA (engl. *Single Page Application*). Aplikacija s jednom stranom učitava samo jedan HTML dokument na prvi zahtjev. Zatim ažurira određeni dio, sadržaj ili tijelo web stranice koje je potrebno ažurirati pomoću JavaScript-a. Taj je obrazac poznat kao usmjeravanje na strani klijenta jer klijent ne mora ponovno učitati cijelu web stranicu da bi dobio novi prikaz stranice svaki put kada korisnik podnese novi zahtjev. Umjesto toga, React presreće zahtjev i samo dohvaća i mijenja dijelove koje treba promijeniti bez potrebe za pokretanjem ponovnog učitavanja cijele stranice. Ovaj pristup rezultira boljom izvedbom i dinamičnijim korisničkim iskustvom. React se oslanja na virtualni DOM, koji je kopija stvarnog DOM-a. React-ov virtualni DOM odmah se ponovno učitava kako bi odražavao ovu novu promjenu kad god dođe do promjene u stanju podataka. Nakon toga React uspoređuje virtualni DOM sa stvarnim DOM-om kako bi shvatio što se točno promijenilo. React zatim pronalazi najbrži način da zakrpi stvarni DOM s tim ažuriranjem bez renderiranja stvarnog DOM-a. Kao rezultat toga, React-ove komponente i korisničko sučelje vrlo brzo odražavaju promjene budući da se ne mora ponovno učitati cijela stranica svaki put kada se nešto ažurira.

React se može uključiti u već postojeću web aplikaciju putem mreže za isporuku sadržaja ili CDN (engl. *Content Delivery Network*) kako bi na tu HTML stranicu korisnik dodao

interaktivnost. Čineći to, React dobiva kontrolu nad određenim dijelom te web stranice, poput bočne trake, widgeta ili nečeg sasvim drugog. Ovo su samo višekratno upotrebljive i interaktivne React komponente s malo React funkcionalnosti. Upravo to se može postići u tri jednostavna koraka:

1. Dodati dvije glavne CDN skripte u HTML indeksnu datoteku web aplikacije. Ove skripte su potrebne za učitavanje Reacta u aplikaciju putem CDN usluge.

```
<!-- The first link is that of the core React library API itself". -->
<script src="https://unpkg.com/react@17/umd/react.development.js"
crossorigin></script>

<!-- The second link is that of the React DOM needed to render to the
DOM". -->
<script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"
crossorigin></script>

<!-- The 3rd link is that of Babel, which is needed to compile our React
code so it is understood by all browsers". -->
<script src='https://unpkg.com/babel-
standalone@6.26.0/babel.js'></script>

<!-- Load the React component file. -->
<script type="text/babel" src="like_widget.js"></script>
```

2. Stvoriti negdje u index.html datoteci kako bi se označio dio u koji korisnik želi uključiti i prikazati React komponentu. Također mu se daje jedinstveni ID koji se kasnije koristi u JavaScript kodu za lociranje mjesta.

```
<div id="root">
      <div id="map"></div>
</div>
```

3. Stvoriti index,js JavaScript datoteku u koju se uz ostali kod dodaje i ReactDom.render [10].

```
ReactDOM.render(
  <React.StrictMode>
    <Provider store={store}>
      <App />
    </Provider>
  </React.StrictMode>,
  document.getElementById('root')
)
```

Iako se lako može dodati React u postojeću web aplikaciju kako bi se stvorilo male dijelove sučelja, praktičnije je koristiti React za izgradnju potpuno razvijene web aplikacije.

Međutim, React ima teške zahtjeve za konfiguraciju alata koji su obično zastrašujući i zamorni za postavljanje pri izradi novih React aplikacija. Srećom Facebook je stvorio alat naredbenog paketa node koji se zove “create-react-app” kako bi pomogao korisnicima u generiranju standardne verzije React aplikacije.

Kao što je već spomenuto u potpoglavlju 3.2, stvaranje novog React je jako jednostavno pokretanjem skripte u terminalu:

```
>> npx create-react-app “app name”
```

te se naredbom

```
>> cd “app name”
```

može putem terminala navigirati do mape u kojoj se pokreće skripta

```
>> npm start
```

koja uključuje sučelje aplikacije na predefiniranoj lokalnoj adresi koja je u slučaju ovog projekta “localhost:3000”.

Jako bitna značajka React koda su svojstva (engl. *props*). React komponente koriste svojstva za međusobnu komunikaciju. Svaka nadređena komponenta može proslijediti neke informacije svojim podređenim komponentama dajući im svojstva. Svojstva podsjećaju na HTML atribute, ali kroz njih se može proslijediti bilo koju JavaScript vrijednost, uključujući objekte, nizove i funkcije. Kroz aplikaciju “Air quality control” React je korišten pri izradi većine komponenti i stranica. Osnovna struktura React-a koja se povezuje s index.html datotekom nalazi se u već spomenutoj index.js datoteci koja se nalazi unutar mape src. Cijelovitu datoteku index.js može se vidjeti u ispisu 3.4.

Ispis 3.4 Index.js datoteka

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import { Provider } from 'react-redux';
import { store } from './redux/store';
import { fetchLocations } from './redux/locationsSlice';
import './index.css';

store.dispatch(fetchLocations());
ReactDOM.render(
  <React.StrictMode>
    <Provider store={store}>
      <App />
    </Provider>
  </React.StrictMode>,
  document.getElementById('root')
);
```

U njoj je sadržan ReactDOM.render koji renderira cijelu aplikaciju. Funkcija "store.dispatch(fetchLocations())" je funkcija za dodavanje lokacija u Redux trgovinu (engl. *store*) unutar komponente "Provider" kao svojstvo. O Redux-u i njegovom store-u više u potpoglavlju 3.5. Također u ovoj datoteci koristi se komponenta "App". To je komponenta koja je omotač (engl. *wrapper*) web aplikacije.

Ispis 3.5 App.js datoteka

```
import './App.css';
import Pages from './pages';

function App() {
  return (
    <div className='app'>
       <Pages/>
    </div>
  );
}

export default App;
```

React komponente mogu biti pisane u obliku klasnih i funkcijskih komponenti. Kao što se iz same deklaracije "function App()" može zaključiti komponenta App.js je funkcijska komponenta. Kroz cijeli kod ove aplikacije koristiti će se funkcijske komponente. Obje vrste React komponenti su jednako valjane, ali moderni način pisanja React koda je preko funkcijskih komponenti te programeri koji su zaduženi za razvijanje React okruženja i biblioteke idu u smjeru korištenja funkcijskih komponenti najviše zbog urednosti pisanog koda. Funkcija "App()" vraća komponentu "Pages". U toj komponenti deklarirane su sve stranice ove aplikacije te njihovo rutiranje na sučelju aplikacije. U ispisu 3.6 može se vidjeti komponenta Pages.js.

Ispis 3.6 Komponenta Pages.js

```
import React from 'react';
import { BrowserRouter as Router, Route, Routes } from "react-router-
dom";
import Home from './Home';
import Locations from './Locations';
import LocationDetails from './LocationDetails';
import Map from './Map'
import About from './About';
import Layout from '../components/Layout';

const Pages = () => {
    return (
        <Router>
            <Layout>
                <Routes>
                    <Route exact path="/" element={<Home />} />
                    <Route path="/locations" element={<Locations />} />
                    <Route path="/locations/karepovac"
element={<LocationDetails />} />
                    <Route path="/locations/marjan"
element={<LocationDetails />} />
                    <Route path="/locations/poljud"
element={<LocationDetails />} />
                    <Route path="/locations/vranjic"
element={<LocationDetails />} />
                    <Route path="/locations/znjan"
element={<LocationDetails />} />
                    <Route path="/map" element={<Map />} />
                    <Route path="/about" element={<About />} />
                </Routes>
            </Layout>
        </Router>
    );
};
export default Pages;
```

Za rutiranje aplikacijskog sučelja u React aplikaciji koristi se paket “react-router-dom”. Komponentom “Router” se omotava komponenta “Layout” u kojoj je definirana alatna traka i navigacija aplikacije te komponenta “Layout” omotava komponentu “Routes” u kojoj su navedene sve rute aplikacije. Postoji 5 osnovnih ruta te stranica aplikacije, a to su:

1. **“Home”** – početna stranica aplikacije,
2. **“Locations”** – stranica na kojoj su može vidjeti kartice svih lokacija,
3. **“LocationDetails”** – stranica na kojoj su vidljivi detalji svake lokacije zasebno, koja se renderira ovisno o tome koja svojstva dobije stranica (props),
4. **“Map”** – stranica na kojoj se vidi karta s lokacijama koje su prikazane “pribadačama” na karti te
5. **“About”** – informativna stranica na kojoj se nalaze podaci o aplikaciji, autoru te mentoru

Sve stranice i rute definirane su tako da se komponenti “Route” pridodjeli njen put (engl. *path*) koji se nastavlja na osnovnu adresu aplikacije, npr. za "path = ”map” adresa će biti “localhost:3000/map” te element koji je zapravo komponenta koja se renderira na toj adresi, npr. “element={<Map />}”.

U ispisu 3.7 može se vidjeti primjer pravilno napisane stranice aplikacije. Kod stranice, ako je moguće, trebao bi biti jednostavan i sadržavati samo nužne elemente kao što su naslov stranice (document.title) te kada se pozove ta funkcijska komponenta ono što ona vrati (return).

Ispis 3.7 Datoteka Map.js stranice Map

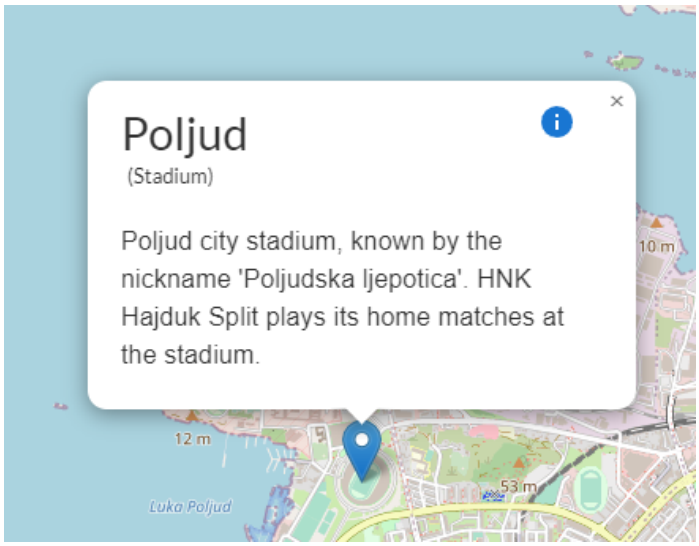
```
import React, { useEffect } from 'react';
import MapContent from '../components/mapContent/MapContent';

const Map = () => {
    useEffect(() => {
        document.title = 'Map';
    });
    return (
        <>
            <MapContent />
        </>
    );
};
export default Map;
```

Funkcija Map vraća komponentu “MapContent” koja sadrži kartu i sve podatke vidljive na karti. Što se tiče koda u navedenoj komponenti, ono što je vezano isključivo za React je funkcija “locationPopup()” te ono što funkcijska komponenta “MapContent()” vrati. U funkciji “locationPopup()” za svaku lokaciju radi se marker na koji ima svoj ključ (key) te poziciju na karti, tj koordinate (element.position). Oznaka lokacije će na karti biti prikazana kao pribadača. Unutar komponente “Marker” definira se komponenta “Popup”. Ta komponenta predstavlja stilizirane elemente i podatke koji će biti vidljivi kada korisnik klikne na pribadaču na mapi. Kod koji se nalazi unutar komponente “Popup” je vezan za komponentnu biblioteku MaterialUI o kojoj će biti više navedeno u potpoglavlju 3.6. Ono što funkcijska komponenta “MapContent()” vrati je spremnik (engl. *container*) karte, tj. komponentu “MapContainer” koja ima svojstva koja predstvaljaju središte mape, razinu zumiranja, omogućeni zum kotačićem miša te minimalni i maksimalni zum. Unutar te komponente nalazi se komponenta “TileLayer” koja definira da se za kanvas karte koristi OpenStreetMap koja je besplatna geografska baza podataka svijeta koja se može uređivati.

Ispis 3.8 Datoteka MapContent

```
import React from 'react';
import { MapContainer, TileLayer, Marker, Popup } from 'react-leaflet';
import { Box, Grid, Typography } from '@mui/material';
import InfoIcon from '@mui/icons-material/Info';
import { useSelector } from 'react-redux';
import { selectAllLocations } from '../../redux/locationsSlice';
import { useNavigate } from "react-router-dom";
import './mapContent.scss'
const MapContent = () => {
    const locations = useSelector(selectAllLocations);
    let navigate = useNavigate();
    const locationPopup = (data) => {
        return (
            data.map((element) =>
                <Marker key={element.name} position={element.position} >
                    <Popup className='leaflet-popup'>
                        <Box sx={{ flexGrow: 1 }}>
                            <Grid className='popup-header' container
spacing={2}>
                                <Grid item md={10}>
                                    <Box className='popup-
title'>{element.name}</Box>
                                    <Box className='popup-
subtitle'>({element.type})</Box>
                                </Grid>
                                <Grid className='popup-details-container'
item md={2}>
                                    <InfoIcon onClick={(e) =>
{e.stopPropagation(); navigate("/locations/" + element.name)}} sx={{
color: "#1976d2", cursor:'pointer' }} className='popup-details-icon' />
                                </Grid>
                            </Grid>
                            <Typography>
                                {element.description}
                            </Typography>
                        </Box>
                    </Popup>
                </Marker >
            )
        )
    };
    return (
        <MapContainer center={[43.515, 16.444]} zoom={14}
scrollWheelZoom={true} minZoom={13} maxZoom={16}>
            <TileLayer
                attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors'
                url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
            />
            {locationPopup(locations)}
        </MapContainer>)};
export default MapContent;
```

Na slici 3.1 može se vidjeti izgleda lokacija i njezina skočna komponenta (engl. *popup*) unutar aplikacije.



Slika 3.1 Prikaz lokacije i skočne komponente u aplikaciji

3.5. Redux

Redux je spremnik predvidljivih stanja za JavaScript aplikacije. Redux je obrazac i biblioteka za upravljanje i ažuriranje stanja aplikacije, koristeći događaje koji se nazivaju radnje (engl. *actions*). Služi kao centralizirano spremište za stanje koje treba koristiti u cijeloj aplikaciji, s pravilima koja osiguravaju da se stanje može ažurirati samo na predvidljiv način. Pomaže u pisanju aplikacija koje se dosljedno ponašaju, izvode u različitim okruženjima (klijent, poslužitelj i nativno) te koje je lako testirati. Povrh toga, pruža sjajno iskustvo za korsinike, kao što je uređivanje koda uživo u kombinaciji s programom za otklanjanje pogrešaka koji "putuje kroz vrijeme". Redux se može koristiti zajedno s Reactom ili s bilo kojom drugom bibliotekom prikaza. Memorija koju zauzima je mala (2kB, uključujući ovisnosti), ali ima velik dostupan ekosustav dodataka.

Redux Toolkit je službeni preporučeni pristup za pisanje Redux logike. Zaokružuje jezgru Redux-a te sadrži pakete i funkcije koji su bitni za izradu Redux aplikacije. Redux Toolkit ugrađuje predložene najbolje prakse, pojednostavljuje većinu Redux zadataka, sprječava uobičajene pogreške i olakšava pisanje Redux aplikacija. Redux Toolkit uključuje pomoćne programe koji pomažu pojednostaviti mnoge uobičajene slučajeve upotrebe, uključujući postavljanje pohrane, stvaranje reduktora i pisanje nepromjenjive logike ažuriranja, pa čak i stvaranje cijelih odsječaka (engl. *slice*) stanja odjednom [12]. Redux Toolkit dostupan je kao npm paket za korištenje sa skupljačem modula ili u Node.js aplikaciji:

```
npm install @reduxjs/toolkit
```

Redux je jako koristan kada:

- korisnik ima velike količine stanja aplikacije koje su potrebne na mnogim mjestima u aplikaciji,
- se stanje aplikacije često se ažurira tijekom vremena,
- logika ažuriranja tog stanja može biti složena te
- aplikacija ima bazu srednje ili velike veličine i na njoj bi moglo raditi mnogo ljudi

Tablica 3.3 Osnovna svojstva Redux-a

Svojstvo	Objašnjenje
Predvidljivost	Redux pomaže u pisanju aplikacija koje se ponašaju dosljedno, izvode se u različitim okruženjima (klijent, poslužitelj i nativno) i koje je lako testirati.
Centraliziranost	Centraliziranje stanja i logike aplikacije omogućuje moćne mogućnosti poput poništavanja/ponavljanja, postojanosti stanja i još mnogo toga.
Mogućnost otklanjanja pogrešaka	Redux DevTools olakšavaju praćenje kada, gdje, zašto i kako se stanje aplikacije promijenilo. Redux-ova arhitektura omogućuje bilježenje promjena, korištenje "debugginga kroz vrijeme", pa čak i slanje potpunih izvješća o pogreškama poslužitelju.
Fleksibilnost	Redux radi s bilo kojim slojem korisničkog sučelja i ima veliki ekosustav dodataka koji odgovaraju potrebama korisnika.

U aplikaciji “Air Quality Control” Redux je definiran pomoću dvije datoteke store.js i locationSlice.js unutar mape redux. Sadržaj datoteke locationSlice.js vidljiv je u ipisu 3.9, a datoteke store.js u ispisu 3.10.

Ispis 3.9 Datoteka locationSlice.js

```
import { createAsyncThunk, createSlice } from '@reduxjs/toolkit';
import API from './../api/index';

const initialState = {
    data: [],
    status: 'idle',
    error: null,
};

export const fetchLocations =
createAsyncThunk('locations/fetchlocations', async () => {
    const response = await API.locations.getAllLocations();
    return response.data;
})

export const locationsSlice = createSlice({
    name: 'locations',
    initialState,

    extraReducers: (builder) => {
        builder
            .addCase(fetchLocations.pending, (state) => {
                state.status = 'loading';
            })
            .addCase(fetchLocations.rejected, (state) => {
                state.status = 'failed';
                state.error = '';
            })
            .addCase(fetchLocations.fulfilled, (state, action) => {
                state.status = 'complete';
                state.data = action.payload;
            });
    }
});

export const selectAllLocations = (state) => state.locations.data;

export const selectLocation = (state, locationId) => {
    return state.locations.find(location => location.id === locationId)
}

export default locationsSlice.reducer;
```

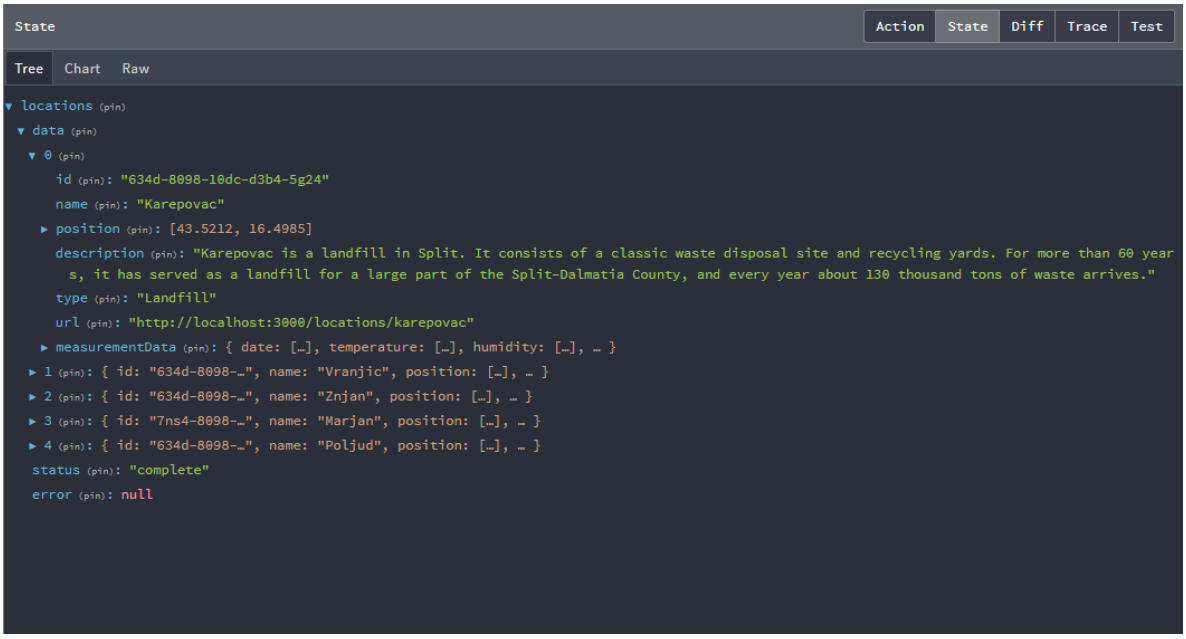
Funkcije “createAsyncThunk()” i “createSlice()” dolaze direktno iz paketa ‘@reduxjs/toolkit’. Funkcija “createAsyncThunk()” je funkcija koja prihvaća Redux akcije tipa “string” i funkciju povratnog poziva koja bi trebala vratiti obećanje (engl. *Promise*). Generira tipove radnji životnog ciklusa obećanja na temelju prefiksa tipa radnje koji se prosljeđuje i vraća kreator radnje pretvornika koji će pokrenuti povratni poziv obećanja i otpremiti radnje životnog ciklusa na temelju vraćenog obećanja. Ovo apstrahira standardni preporučeni pristup za rukovanje životnim ciklusima asinkronih zahtjeva. Ne generira

nikakve reduktorske funkcije jer ne zna koje se podatke dohvaća, kako treba pratiti stanje učitavanja ili kako treba obraditi podatke koji se vraćaju. Trebalo bi se napisati reduktorsku logiku koja upravlja ovim radnjama, s bilo kojim stanjem učitavanja i logikom obrade koja je prikladna za aplikaciju. Za to se koristi varijabla "locationsSlice" koja kreira reduktor koji sadrži ime, početno stanje te dodatne reduktore s obzirom na status API poziva. Varijabla "locationsSlice" je definirana funkcijom "createSlice()", a ona je funkcija koja prihvaća početno stanje, objekt reduktorskih funkcija i naziv odsječka, te automatski generira kreatore radnji i tipove radnji koji odgovaraju reduktorima i stanju. Funkcije "selectAllLocations()" i "selectLocation()" su selektori. Selektor je jednostavna funkcija koja prihvaća Redux stanje kao argument i vraća podatke koji su izvedeni iz tog stanja. Te funkcije će se koristiti u aplikaciji kada se iz Redux store-a treba dohvatiti stanje ili vrijednost nekih podataka iz lokacija koje su spremljene u store-u [12]. API pozivi biti će detaljnije definirani i objašnjeni u potpoglavlju 3.7.

Ispis 3.10 Redux store

```
import locationsReducer from './locationsSlice';
import { configureStore } from '@reduxjs/toolkit';

export const store = configureStore({
    reducer: {
        locations: locationsReducer
    }
})
```

Redux store je jednostavan za postaviti. Funkcija "configureStore()" iz paketa '@reduxjs/toolkit' je apstrakcija u odnosu na standardnu Redux "createStore()" funkciju koja dodaje dobre zadane postavke u postavku store-a za bolje razvojno iskustvo. Unutar navedene funkcije definira se reduktor za lokaciju koji se uvozi iz datoteke locationsSlice.js te je rezultat istog vidljiv unutar konzole pretraživača ukoliko je instaliran Redux dodatak za preglednik. Unutar svojstva reduktora moguće je definirati više različitih reduktora ako ima potrebe za tim. Na slici 3.2 vidi se kako Redux store izgleda na Google Chrome pretraživaču.



Slika 3.2 Redux store na Google Chrome pretraživaču

Dobar primjer upotrebe selektora u kodu aplikacije vidljiv je u datoteci Locations.js za stranicu “Locations” u ispisu 3.11.

Ispis 3.11 Selektor “selectAllLocations()” u datoteci Locations.js

```
import React, { useEffect } from 'react';
import { Grid } from '@mui/material';
import LocationCard from '../components/locationCard/LocationCard';
import { useSelector } from 'react-redux';
import { selectAllLocations } from './../redux/locationsSlice';

const Locations = () => {
    useEffect(() => {
        document.title = 'locations';
    });
    const locations = useSelector(selectAllLocations);

    return (
        <Grid container spacing={1} sx={{ p: 1 }}>
            <LocationCard locations={locations} />
        </Grid>
    );
};
export default Locations;
```

U varijablu “locations” spremaju se sve lokacije iz Redux store-a pomoću “useSelector()” funkcije iz ‘react-redux’ paketa i definiranog selektora “selectAllLocations()”. Varijabla “locations” prosljeđuje se kao svojstvo (engl. *prop*) u komponentu “LocationCard” za daljnju upotrebu.

3.6. Material UI

Material UI je biblioteka React komponenti otvorenog koda koja implementira Googleov Material design. Uključuje sveobuhvatnu kolekciju unaprijed izgrađenih komponenti koje su spremne za upotrebu u proizvodnji. Korisničko sučelje Material UI prekrasno je po dizajnu i ima niz opcija za prilagodbu koje olakšavaju implementaciju prilagođenog sustava dizajna povrh komponenti biblioteke [13].

Prednosti Material UI biblioteke su:

- gotove komponente daju korisniku uštedu na vremenu izrađivanja tih komponenti u standardnom JavaScript, HTML i CSS paketu,
- osiguravanje da svaka komponenta korisničkog sučelja biblioteke zadovoljava najviše standarde oblika i funkcije, ali i da po potrebi odstupaju od službenih specifikacija gdje je to potrebno kako bi korisnik imao više opcija,
- biblioteka uključuje opsežan skup intuitivnih značajki prilagodljivosti,
- kompleti za dizajn pojednostavljuju tijek rada i povećavaju dosljednost između dizajna i funkcionalnosti te
- Material UI ima najveću UI zajednicu u React okruženju

Za instalaciju Material UI biblioteke u terminalu se pokrene skripta:

```
>> npm install @mui/material
```

te se uglavnom pokreće i skripta:

```
>> npm install @mui/icons-material
```

za instalaciju Material UI standardiziranih ikona. Isto tako u ovom projektu korištena je još jedna pogodnost Material UI biblioteke, a to je komponenta “Date picker”. Taj paket se instalira zasebno od ostatka biblioteke skriptom:

```
>> npm install @mui/x-date-pickers
```

te se u njoj nalazi navedena komponenta koja služi za odabiranje datuma i vremena. U MUI X kolekciji se nalazi još i “Data Grid” komponenta. Te dvije komponente se nalaze u zasbnom paketu jer su to naprednije komponente za sučelje aplikacije te se neke usluge koje dolaze s njima naplaćuju.

Material UI biblioteka pruža veliki spektar gotovih komponenti koje su korištene u projektu “Air Quality Control”. U tablici 3.4 mogu se vidjeti korištene komponente te njihova uloga.

Tablica 3.4 Material UI komponente

Komponenta	Uloga
AppBar	Komponenta koja prikazuje informacije i radnje koje se odnose na trenutni zaslon.
Box	Služi kao komponenta omotača za većinu potreba CSS uslužnog programa.
Breadcrumbs	Komponenta koja se sastoji od popisa poveznica koje pomažu korisniku vizualizirati lokaciju stranice unutar hijerarhijske strukture web stranice i omogućuju navigaciju do bilo kojeg od njegovih predaka (engl *parents*).
Button	Komponenta koja dopušta korisnicima da poduzimaju radnje i donose odluke jednim dodirom.
ButtonGroup	Komponenta koja se može koristiti za grupiranje povezanih gumba.
Card	Komponenta u koju se dodaje sadržaj i radnje o jednoj temi.
CardActions	Komponenta u koju se dodaju akcije za komponentu “Card”.
CardContent	Komponenta u koju se dodaje sadržaj za komponentu “Card”.
Divider	Komponenta koja predstavlja tanku liniju koja grupira ili odvaja sadržaj.
Grid	Responzivna komponenta izgleda koja se prilagođava veličini i orijentaciji zaslona, osiguravajući dosljednost u svim prikazima.
Paper	Komponenta koja prevodi fizička svojstva papira na ekran.
Table	Tablica prikazuje skupove podataka koji su u potpunosti prilagodljivi.
TableBody	Komponenta koja sadržava sve retke tablice i njihove ćelije.

TableCell	Komponenta u koju se unosi dio ili cijeli sadržaj jednog retka tablice.
TableContainer	Služi kao omotač komponente “Table” kako bi se prilagođavala na sve zaslone i prikaze.
TableHead	Komponenta zaglavlja tablice.
TablePagination	Komponenta za paginaciju, tj obilježavanja stranica tablice.
TableRow	Komponenta koja predstavlja redak tablice.
TextField	Komponenta za unošenje i uređivanje teksta.
Toolbar	Komponenta koja služi kao spremnik, koji omogućuje korištenje svojstava fleksibilnih stavki za raspoređivanje elemanta.
Typography	Komponenta koja služi za prezentiranje teksta.

Jednostavan primjer upotrebe Material UI komponenti u kodu aplikacije “Air Quality Control” vidljiv je u ispisu 3.12.

Ispis 3.12 Upotreba MUI komponenti u AppBar.js komponenti

```
import * as React from 'react';
import { AppBar, Box, Toolbar, Typography } from '@mui/material';
import MenuIcon from '@mui/icons-material/Menu';
import TemporaryDrawer from '../sideMenu/SideMenu';
const BasicAppBar = () => {
    return ( <Box sx={{ flexGrow: 1 }}>
            <AppBar position="static" sx={{ bgcolor: "#1976d2" }}>
                <Toolbar>
                    <TemporaryDrawer size="large" edge="start"
color="inherit" aria-label="menu" sx={{ mr: 2 }} >
                        <MenuIcon />
                    </TemporaryDrawer>
                    <Typography variant="h6" component="div" sx={{
flexGrow: 1, padding: '1vh' }}>
                        Air quality
                    </Typography>
                </Toolbar>
            </AppBar>
        </Box>
    )}
export default BasicAppBar;
```

3.7. Axios

Axios je HTTP klijent temeljen na obećanjima za Node.js i preglednik. Izomorfan je (može se izvoditi u pregledniku i Node.js-u s istom bazom koda). Na strani poslužitelja koristi izvorni Node.js HTTP modul, dok na klijentu (pregledniku) koristi XMLHttp zahtjeve. Axios služi za povezivanje sučelja aplikacije s pozadinskim servisom [14].

Svojstva Axios paketa:

- XMLHttp zahtjevi iz preglednika,
- HTTP zahtjevi iz Node.js-a,
- podržava Promise API,
- presretanje zahtjeva i odgovora,
- transformiranje podataka zahtjeva i odgovora,
- otkazivanje zahtjeva,
- automatske transformacije za JSON podatke te
- podrška na strani klijenta za zaštitu od XSRF-a

Axios paket se instalira unošenjem skripte:

```
>> npm install axios
```

u terminal. U ispisu 3.13 može se vidjeti kako se radi nova instanca Axios s prilagođenom konfiguracijom.

Ispis 3.13 Axios instanca u datoteci axiosEnv.js

```
import axios from 'axios';

const apiUrl = "http://localhost:3030";

export default axios.create({
    baseURL: apiUrl,
    headers: {'Content-Type' : "application/json; charset=UTF-8",
"Access-Control-Allow-Origin": "http://localhost:3030/"}
})
```

Axios instanci se dodaje osnovna adresa API-ja koja je se u projektu “Air Quality Control” nalazi na “http://localhost/3030”. Također se dodaju zaglavlja za HTTP zahtjeve. Zaglavlja HTTP zahtjeva koriste se za pružanje dodatnih informacija o zahtjevu. U ovom primjeru se postavlja zaglavlje za vrstu resursa koji je u JSON formatu uz set karaktera UTF-8 te se drugim zaglavljem definira da se odgovor na zahtjev može dijeliti s kodom zahtjeva iz danog izvora. U tablici 3.5 prikazane su metode Axios instance s odgovarajućim HTTP metodama.

Tablica 3.5 Metode Axios instance

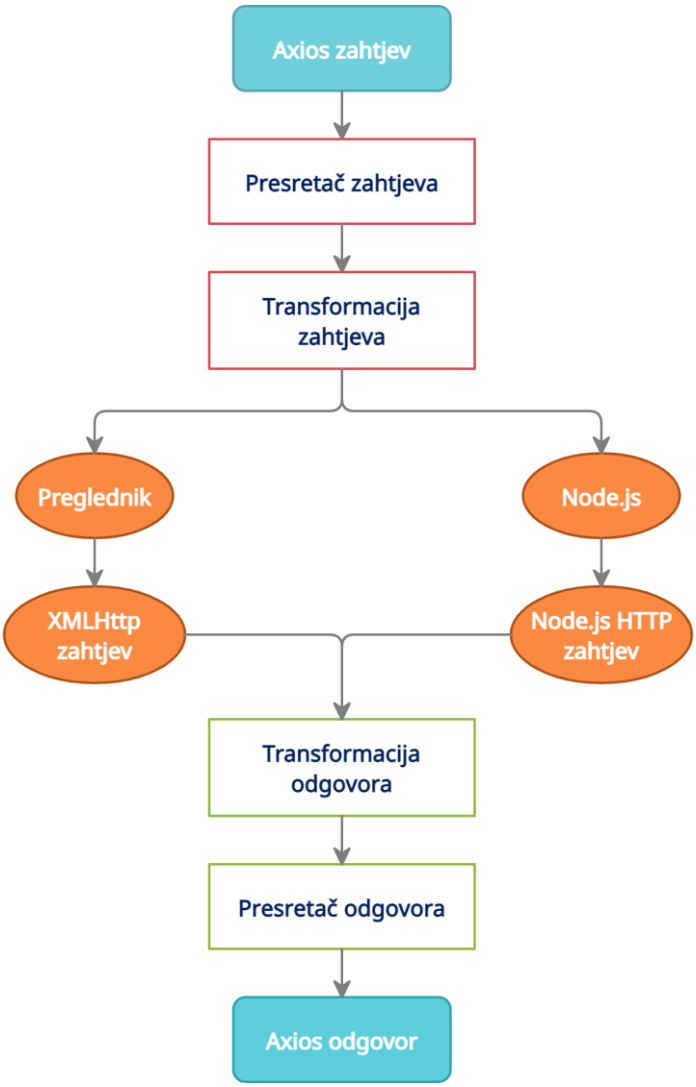
Metode Axios instance	Odgovarajuća HTTP metoda
axios.request(config)	/
axios.get(url[, config])	GET
axios.delete(url[, config])	DELETE
axios.head(url[, config])	HEAD
axios.options(url[, config])	OPTIONS
axios.post(url[, data[, config]])	POST
axios.put(url[, data[, config]])	PUT
axios.patch(url[, data[, config]])	PATCH

Axios instanca iz ispisa 3.13 koristi se u datoteci locationsAPI.js za kreiranje funkcija zahtjeva prema pozadinskom servisu. S obzirom na jednostavnost pozadinskog servisa aplikacije "Air Quality Control" za komunikaciju s istim na sučelju aplkacije dovoljno je definirati samo jedan zahtjev "instance.get()" funkcije "getAllLocations()" koja dohvaća sve lokacije na adresi "http://localhost:3030/locations" što je vidljivo u ispisu 3.14.

Ispis 3.14 Datoteka locationsAPI.js sa Axios zahtjevom

```
import instance from './axiosEnv';

const getAllLocations = () => {
    return instance.get('/locations');
}

const locationsAPI = {
    getAllLocations
}

export default locationsAPI;
```

Na slici 3.3 može se vidjeti diagram stanja koji prikazuje kako Axios komunicira s aplikacijom.



Slika 3.3 Dijagram stanja komunikacije Axios-a s aplikacijom

4. Bežična senzorska mreža

Bežična senzorska mreža ili WSN (engl. *Wireless Sensor Network*) je bežična mreža bez infrastrukture koja je postavljena u veliki broj bežičnih senzora na "ad-hoc" način koji se koristi za nadzor sustava, fizičkih ili okolišnih uvjeta. Senzorski čvorovi koriste se u bežičnoj senzorskoj mreži s ugrađenim procesorom koji upravlja i nadzire okolinu u određenom području. Spojeni su na baznu stanicu koja djeluje kao procesorska jedinica u sustavu bežične senzorske mreže. Bazna stanica u sustavu bežične senzorske mreže povezana je putem interneta radi dijeljenja podataka. Može se koristiti za obradu, analizu, pohranu i prikupljanje podataka.

Neke od primjena bežične senzorske mreže:

- Internet stvari ili IOT (engl. *Internet of Things*),
- nadzor za sigurnost ili otkrivanje prijetnji,
- temperatura okoline, vlažnost, tlak zraka, količina plinova, itd,
- razina buke u okolini,
- medicinske aplikacije poput praćenja pacijenata,
- poljoprivreda te
- detekcija klizišta

U sklopu aplikacije "Air quality control" primjenjena je za mjerenje temperature okoline, vlažnosti i količine ugljikovog dioksida u zraku.

Osnovne komponente od kojih se sastoji bežična senzorska mreža su:

1. Senzori - koriste se za hvatanje varijabli okoline,
2. Radio čvorovi - koriste se za primanje podataka koje proizvode senzori i šalju ih WLAN pristupnoj točki,
3. WLAN pristupna točka - prima podatke koje bežično šalju radio čvorovi, uglavnom putem interneta te
4. Softver za procjenu - podatke koje prima WLAN pristupna točka obrađuje softver koji se naziva softver za procjenu kako bi se korisnicima predstavilo izvješće

Bazna stanica izgleda kao sučelje između korisnika i mreže. Može pretvoriti potrebne informacije s mreže slanjem upita i prikupljanjem rezultata. Obično bežična senzorska mreža sadrži tisuće senzorskih čvorova. Čvorovi mogu međusobno komunicirati pomoću radio signala. Bežični senzorski čvorovi opremljeni su senzorskim i radio primopredajnicima, računalnim uređajima i komponentama napajanja. Senzorski čvor u bežičnoj senzorskoj mreži ograničen je resursima, također ima ograničenu brzinu obrade, kapacitet pohrane i komunikacijsku propusnost. Nakon što su senzorski čvorovi instalirani,

oni su odgovorni za samoorganiziranje odgovarajuće mrežne infrastrukture često s višestrukom komunikacijom s istima. Tada ugrađeni senzori počinju prikupljati informacije koje su potrebne. Zatim posebno dizajnirani uređaji bežičnih senzorskih mreža odgovaraju na te upite poslane s kontrolnog mjesta kako bi izvršili specifične upute ili dali uzorke senzora. Način rada senzorskih čvorova također može biti kontinuiran ili vođen događajima. Globalni sustav pozicioniranja ili GPS (engl. *Global Positioning System*) te lokalni algoritmi za pozicioniranje ili LPA (engl. *Local Positioning Algorithms*) mogu se koristiti za dobivanje informacija o lokaciji i pozicioniranju. Bežični senzorski uređaji često su opremljeni aktuatorima za djelovanje u određenim uvjetima. Te se mreže ponekad ili obično nazivaju bežična mreža senzora i mreža pokretača.

Klasifikacija bežičnih senzorskih mreža je sljedeća:

1. **Statička i mobilna bežična senzorska mreža** - svi senzorski čvorovi povezani su bez kretanja i to su statične mreže u mnogim aplikacijama. Primjer jedne statičke bežične mreže je u biološkim sustavima. Mobilna mreža je ona u kojoj senzorski čvorovi mijenjaju svoju lokaciju. Primjer mobilne mreže je praćenje životinja.
2. **Deterministička i nedeterministička bežična senzorska mreža** - u deterministčkim bežičnim senzorskim mrežama, položaj senzorskog čvora je izračunat i fiksan. Određivanje položaja senzorskih čvorova može biti nemoguć zbog nekoliko čimbenika kao što su oštra okruženja ili neprikladni radni uvjeti. Takve vrste mreža su nedeterminističke i trebaju složen sustav.
3. **Bežična senzorska mreža s jednom baznom stanicom i s više baznih stanica** - u mrežama s jednom baznom stanicom koristi se samo jedna bazna stanica koja se nalazi u blizini područja senzorskog čvora.
4. **Statička bazna stanica i mobilna bazna stanica** - slična je senzorskim čvorovima, čak su i bazne stanice mreže često ili statične ili mobilne. Statička bazna stanica ima fiksnu poziciju obično blizu područja osjeta. Mobilna bazna stanica kreće se oko senzorne regije jer je opterećenje senzorskih čvorova uravnoteženo.
5. **Single-hop i Multi-hop bežična senzorska mreža** - u Single-hop mrežama senzorski čvorovi su izravno povezani s baznom stanicom. U slučaju Multi-hop mreže, ravnopravni čvorovi i glave klastera koriste se za prijenos informacija kako bi se smanjila potrošnja energije.
6. **Samokonfigurabilne i nesamokonfigurabilne bežične senzorske mreže** - u nesamokonfigurabilnim mrežama, senzorske mreže se ne mogu organizirati u mrežu i smatrati kontrolnom jedinicom za prikupljanje podataka. U mnogim mrežama senzorski čvorovi mogu se organizirati, održavati vezu i surađivati s drugim senzorskim čvorovima kako bi izvršili zadatak te se one nazivaju samokonfigurabilne mreže.
7. **Homogene i heterogene bežične senzorske mreže** - u slučaju homogenih mreža, svi senzorski čvorovi imaju istu potrošnju energije, mogućnosti pohrane i računsku

snagu. U slučaju heterogenih mreža, neki senzorski čvorovi imaju veću računsku snagu i zahtjeve za energijom od drugih, a također su i zadaci obrade i komunikacije podijeljeni u skladu s tim [15].

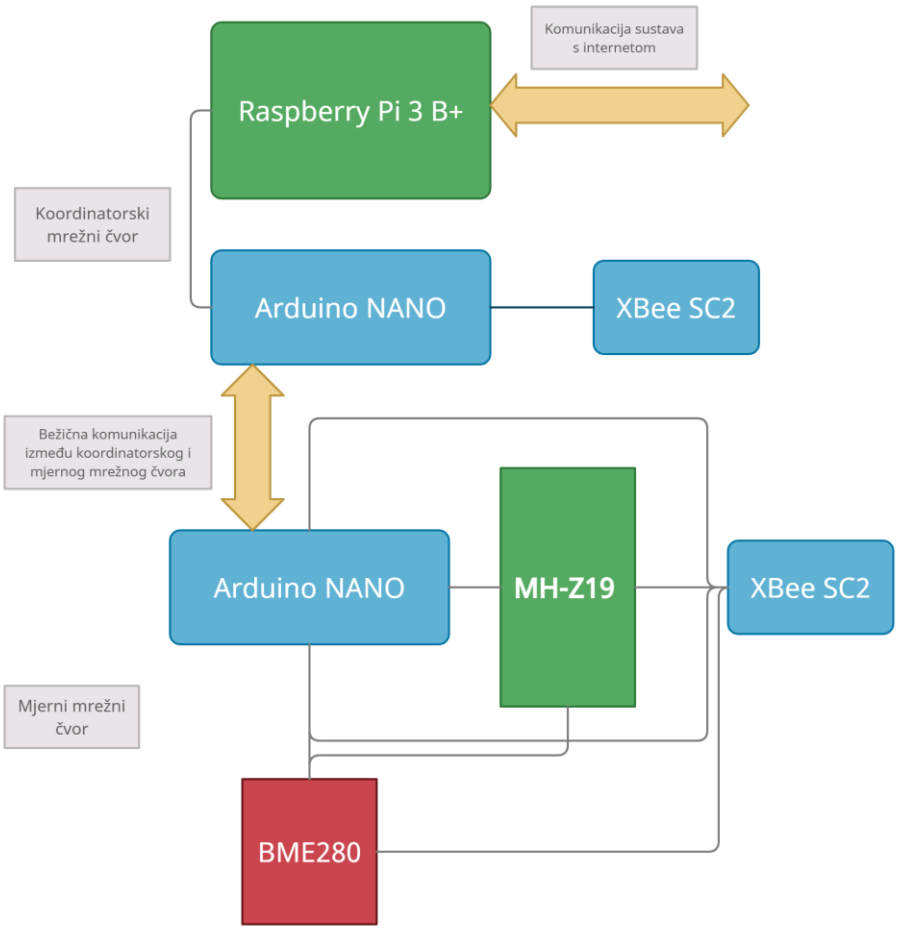
4.1. Komponente bežične senzorske mreže

U tablici 4.1 može se vidjeti komponente bežične senzorske mreže kojom su mjereni podaci prema kojima su se radila mjerenja za lokacije.

Tablica 4.1 Komponente bežične radio mreže [16]

Komponenta	Opis	Količina
Digi XBee S2c ZB RF Module	ZigBee podržani radio primopredajnik, RF modul.	5
Arduino NANO	Arduino NANO mikrokontroler.	5
MH-Z19 infrared gas sensor for detection of CO2	Senzor za detekciju razina koncentracije CO2 plina u zraku te posljedično i utvrđivanje same kvalitete zraka.	4
BME280 Digital Sensor Module for temperature, humidity and air pressure	Senzor temperature, vlage i tlaka zraka.	4
SainSmart XBee USB Explorer	Pomoćni modul za povezivanje terminala RF modula sa mikrokontrolerom te povezivanje istog sa računalnim alatima u svrhu konfiguriranja mrežnih parametara pojedinog čvora.	5
Raspberry Pi 3 B+	Minijaturno računalo bazirano na Linux operacijskom sustavu u službi gateway-a za realiziranu senzorsku mrežu	1

Na slici 4.1 može se vidjeti blok shemu Arduino sustava povezanog na bežičnu senzorsku mrežu korištenjem komponenti iz tablice 4.1.



Slika 4.1 Blok shema Arduino sustava povezanog na bežičnu senzorsku mrežu

1. **Digi XBee S2c ZB RF Module** - XBee/XBee-PRO Zigbee RF moduli omogućuju bežično povezivanje krajnjih uređaja u Zigbee mesh mreže. Korištenjem Zigbee PRO skupa značajki, ovi moduli su interoperabilni s ostalima Zigbee uređaji, uključujući uređaje drugih proizvođača. Uz XBee, korisnici mogu postaviti svoju Zigbee mrežu u nekoliko minuta bez konfiguracije ili dodatnog razvoja. XBee/XBee-PRO Zigbee RF moduli su kompatibilni s drugim uređajima koji koriste XBee Zigbee tehnologiju. To uključuje ConnectPortX pristupnike, XBee i XBee-PRO adaptere, zidne usmjerivače, XBee senzorei ostale proizvode pod nazivom Zigbee [17].

Slika 4.2 Digi XBee S2c

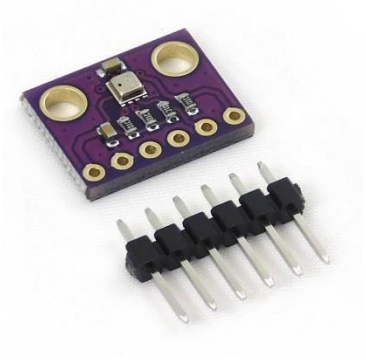
2. **Arduino NANO** - Nano je najstariji član Arduino Nano obitelji pločica. Sličan je Arduino Duemilanove, ali je napravljen za korištenje kao matična ploča i nema namjensku utičnicu za napajanje. Nasljednici klasičnog Nano-a su na primjer Nano 33 IoT s WiFi modulom ili Nano 33 BLE Sense s Bluetooth Low Energy i nekoliko senzora za okoliš. ATmega328 mikrokontroler radi na 16 MHz i ima 32 KB flash memorije (od čega 2 KB koristi bootloader). S duljinom od 45 mm i širinom od 18 mm, Nano je Arduino najmanja ploča i teži samo 7 grama. Nano je napravljen za korištenje matične ploče i ima zalemljene zaglavlja za sve pinove, što omogućuje jednostavno pričvršćivanje ploče na bilo koju matičnu ploču [18].

Slika 4.3 Arduino NANO

3. **MH-Z19 infrared gas sensor for detection of CO2** - MH-Z19B NDIR infracrveni plinski modul uobičajeni je senzor male veličine koji koristi nedisperzivni infracrveni ili NDIR (engl. *Non-Dispersive Infrared*) princip za otkrivanje postojanja ugljikovog dioksida u zraku, s dobrom selektivnošću. Ne ovisi o kisiku i ima dugi vijek trajanja. Ima ugrađenu temperaturnu kompenzaciju te ima i UART (engl. *Universal Asynchronous Receiver-Transmitter*) izlaz i PWM (engl. *Pulse-Width Modulation*) izlaz. Sastavljen je od izdržljivih i naprednih integracija tehnologije za detekciju plina koja apsorbira infracrveno zračenje, preciznog optičkog kruga i vrhunskog strujnog kruga [19].

Slika 4.4 MH-Z19 infrared gas sensor

4. **BME280 Digital Sensor Module for temperature, humidity and air pressure** - Senzorski modul BME280 očitava barometarski tlak, temperaturu i vlažnost. Budući da se tlak mijenja s visinom, može procijeniti i visinu. Postoji nekoliko verzija ovog senzorskog modula. Senzor BME280 koristi I2C ili SPI (engl. *Serial Peripheral Interface*) komunikacijski protokol za razmjenu podataka s mikrokontrolerom [20].



Slika 4.5 BME280 Digital Sensor Module

5. **SainSmart XBee USB Explorer** - Xbee USB adapter koristi se za konfiguracijske parametre XBee modula kako bi se olakšao korištenje bežičnog prijenosa podataka. Jednostavno se povezuje s računalom putem mini USB kabela. Ima softver za podršku XBee postavki X-CTU te se može koristiti i kao USB-TTL adapter. Napon uređaja je +5V (USB napajanje) [21].

Slika 4.6 SainSmart XBee USB Explorer

6. **Raspberry Pi 3 B+** - Raspberry Pi 3 Model B+ najnoviji je proizvod u liniji Raspberry Pi 3, ima 64-bitni četverojezgreni procesor koji radi na 1,4 GHz, dual-band 2,4 GHz i bežični LAN od 5 GHz, Bluetooth 4.2/BLE, brži Ethernet i PoE mogućnost putem zasebnog PoE HAT-a. Dvopojasni bežični LAN dolazi s certifikatom modularne sukladnosti, omogućujući da se ploča dizajnira u krajnje proizvode sa značajno smanjenim testiranjeem usklađenosti bežičnog LAN-a, poboljšavajući i troškove i vrijeme razvoja. Raspberry Pi 3 Model B+ zadržava isti mehanički trag kao oba Raspberry Pi 2 Model B i Raspberry Pi 3 Model B [22].

Slika 4.7 Raspberyy Pi 3 B+

4.2. Zigbee protokol

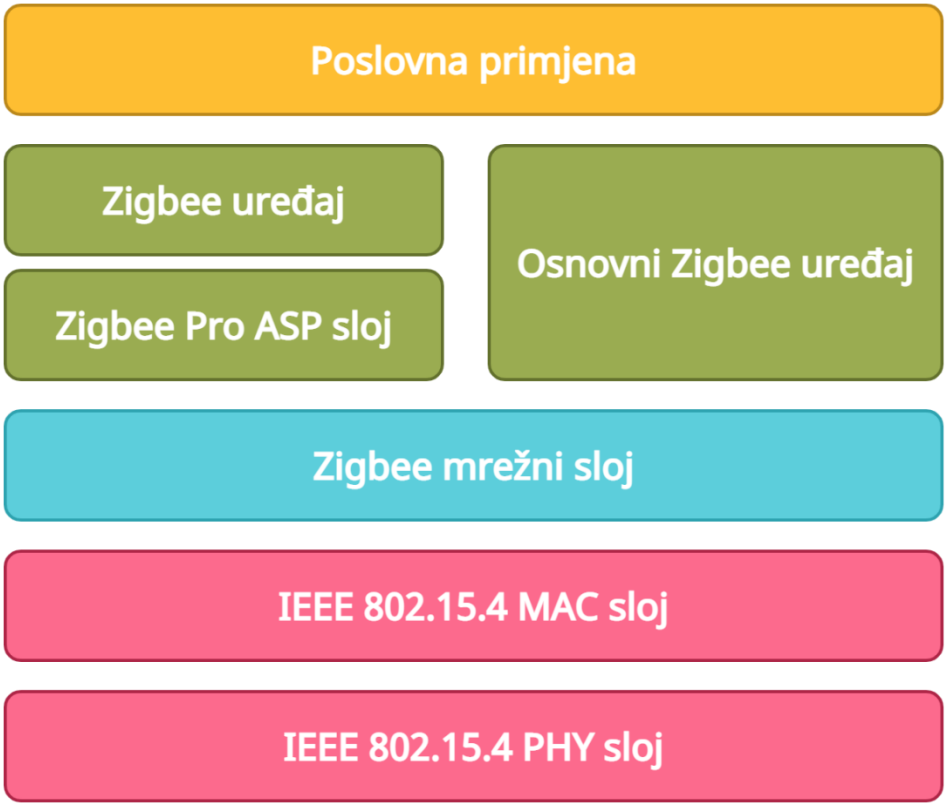
Zigbee je bežična tehnologija razvijena kao standard povezivanja otvorenog globalnog tržišta za rješavanje jedinstvenih potreba jeftinih bežičnih IoT podatkovnih mreža male snage. Zigbee standard povezivanja radi na IEEE 802.15.4 radijskoj specifikaciji fizičke ploče i radi u nelicenciranim radijskim opsezima uključujući 2,4 GHz, 900 MHz i 868 MHz. Bežična specifikacija 802.15.4 na kojoj radi Zigbee stog dobila je ratifikaciju Instituta inženjera elektrotehnike i elektronike ili IEEE (engl. *Institute of Electrical and Electronics Engineers*) 2003. godine. Specifikacija je paketni radijski protokol ploče namijenjen jeftinim uređajima i proizvodima koji rade na baterije. Protokol omogućuje uređajima prijenos podataka u različitim mrežnim topologijama i može imati trajanje baterije od nekoliko godina. Zigbee protokol kreirale su i ratificirale tvrtke članice Zigbee

Board Alliance-a. Preko 300 vodećih proizvođača poluvodiča na tržištu, tehnoloških tvrtki, OEM-ova i uslužnih tvrtki čine članski odbor Zigbee Alliance-a. Zigbee protokol je osmišljen kako bi pružio bežično podatkovno rješenje jednostavno za korištenje koje karakterizira sigurna, pouzdana bežična mrežna arhitektura. Protokol Zigbee 3.0 dizajniran je za prijenos podataka kroz šumna RF okruženja koja su uobičajena u komercijalnim i industrijskim tržišnim aplikacijama. Verzija 3.0 temelji se na postojećem standardu povezivanja Zigbee, ali objedinjuje profile aplikacija specifičnih za tržište kako bi se svim uređajima omogućilo bežično povezivanje u istoj mreži, bez obzira na njihovu tržišnu oznaku i funkciju. Nadalje, certifikacijska shema Zigbee 3.0 osigurava interoperabilnost proizvoda različitih proizvođača uređaja. Povezivanjem Zigbee 3.0 mreža na IP domenu otvara se bežični nadzor i kontrola s radijskih uređaja kao što su pametni telefoni i tableti na LAN-u ili WAN-u, uključujući Internet, i donosi stvarni Internet of Things [23].

Značajke Zigbee protokola su:

- podrška za više mrežnih topologija kao što su “point-to-point”,
- “point-to-multipoin”t i mesh mreže,
- nizak radni ciklus, tj. osigurava dug vijek trajanja baterije,
- niska latencija,
- prošireni spektar izravne sekvence ili DSSS (engl. *Direct Sequence Spread Spectrum*),
- do 65 000 čvorova po mreži,
- 128-bitna AES enkripcija za sigurne podatkovne veze te
- izbjegavanje kolizije, ponovni pokušaji i potvrde

Softverski skup Zigbee 3.0 uključuje osnovni uređaj koji pruža dosljedno ponašanje za puštanje čvorova i uređaja u mrežu. Omogućen je zajednički skup metoda puštanja u rad, uključujući Touchlink, metodu puštanja u rad iz blizine. Na slici 4.7 vidljiv je Zigbee 3.0 stog [23].



Slika 4.8 Zigbee 3.0 stog

Zigbee 3.0 pruža poboljšanu sigurnost mreže. Postoje dvije metode sigurnosti koje dovode do dvije vrste mreže:

- **Centralizirana sigurnost** - ova metoda ima koordinatora/centar povjerenja koji formira mrežu i upravlja dodjelom sigurnosnih ključeva mreže i veze čvorovima koji se pridružuju.
- **Distribuirana sigurnost** - ova metoda nema koordinator/centar povjerenja i formira je usmjerivač. Bilo koji čvor Zigbee usmjerivača može naknadno pružiti mrežni ključ čvorovima koji se pridružuju.

Čvorovi prihvaćaju bilo koju sigurnosnu metodu koju koristi mreža čvorišta kojoj se pridružuju. Zigbee 3.0 podržava rastući opseg i složenost bežičnih mreža i nosi se s velikim lokalnim mrežama s više od 250 čvorova. Zigbee također upravlja dinamičkim ponašanjem ovih mreža, s čvorovima koji se pojavljuju, nestaju i ponovno pojavljuju u mreži, te omogućuje čvorovima koji su ostali bez nadređenih čvorova, da se ponovno pridruže mreži putem drugog nadređenog čvora. Samoobnavljajuća priroda Zigbee Mesh mreža također omogućuje čvorovima da ispadnu iz mreže bez ikakvog prekida internog usmjeravanja. Kompatibilnost Zigbee 3.0 sa prethodnim verzijama znači da su aplikacije i pametni kućni uređaji koji su već razvijeni pod profilom Zigbee Light Link 1.0 ili Home Automation 1.2

spremni za Zigbee 3.0. Profil Zigbee Smart Energy također je kompatibilan sa Zigbee 3.0 na funkcionalnoj razini, ali Smart Energy ima dodatne sigurnosne zahtjeve koji se rješavaju samo unutar profila. Zigbee-ova značajka OTA (engl *Over-The-Air*) nadogradnje za ažuriranje softvera tijekom rada uređaja osigurava da se aplikacije na uređajima koji su već postavljeni na terenu ili tržištu mogu neprimjetno premjestiti na Zigbee 3.0. Nadogradnja je izborna funkcionalnost koju se proizvođači potiču da podrže u aplikacijskom sloju svojih Zigbee proizvoda. Ključna komponenta Zigbee protokola je mogućnost podržavanja kompleksnog umrežavanja (engl. *mesh*). U isprepletenoj mreži čvorovi su međusobno povezani s drugim čvorovima tako da svaki čvor povezuje na više putova. Veze između čvorova dinamički se ažuriraju i optimiziraju pomoću sofisticirane, ugrađene mrežaste tablice usmjeravanja. Mesh mreže su po prirodi decentralizirane što znači da je svaki čvor sposoban za samootkrivanje na mreži. Također, kako čvorovi napuštaju mrežu, mrežasta topologija omogućuje čvorovima rekonfiguraciju staza usmjeravanja na temelju nove strukture mreže. Karakteristike mrežne topologije i "ad-hoc" usmjeravanja pružaju veću stabilnost u promjenjivim uvjetima ili kvarovima na pojedinačnim čvorovima. Zigbee omogućuje široku implementaciju bežičnih mreža s jeftinim rješenjima niske potrošnje. Omogućuje rad godinama na jeftinim baterijama za mnoštvo nadzornih i kontrolnih aplikacija. Pametna energija ili pametna mreža, automatsko očitavanje brojila AMR (engl, *Automatic Meter Reading*), kontrole rasvjete, sustavi automatizacije zgrada, nadzor spremnika, HVAC (engl. *Heating, Ventilation and Air Conditioning*) kontrola, medicinski uređaji, bežične mreže senzora i aplikacije za vozni park samo su neki od mnogih prostora gdje Zigbee tehnologija čini značajan napredak[23].

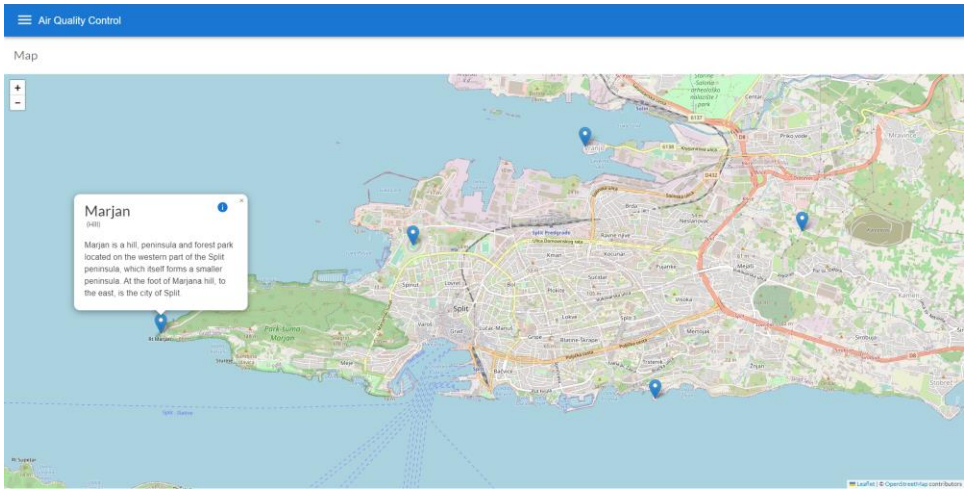
5. Air Quality Control aplikacija

U ovom poglavlju biti će prikazane stranice aplikacije “Air Quality Control” te njihove značajke i funkcionalnosti. U svakom prikazu vidljiva je alatna traka na vrhu aplikacije (AppBar), gumb za izbornik (Menu) u gornjem lijevom kutu te traka na kojoj je prikazana hijerarhijska struktura navigacije kroz aplikaciju (BreadCrumbs).

1. **Home -** Početna stranica aplikacije “Air Quality Control”. Jedna jako jednostavna stranica na kojoj se nalazi pozadinska slika te naslov i logo aplikacije.

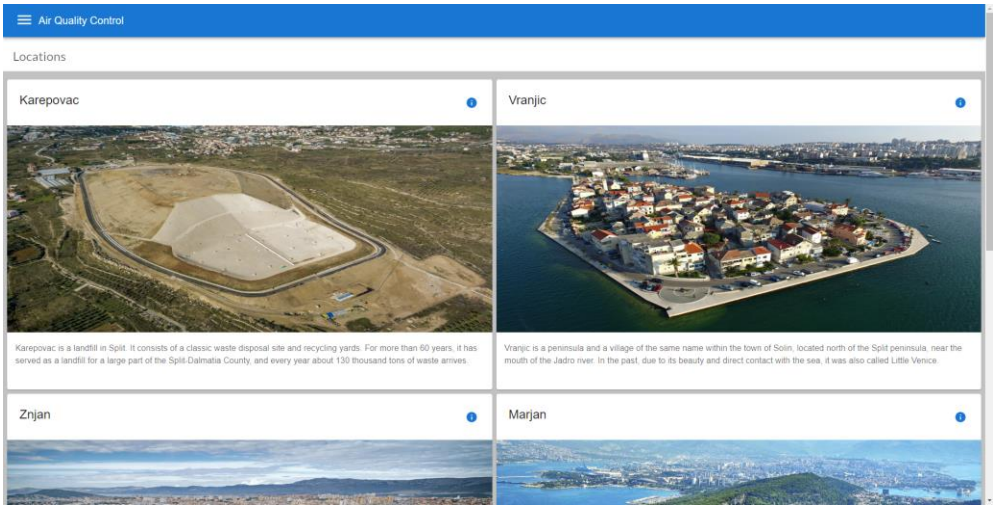
Slika 5.1 Stranica “Home”

2. **Map** – Na ovoj stranici nalazi se OpenStreetMap karta sa pribadačama za sve lokacije. Svaka od pribadača može se kliknuti lijevim klikom miša te se otvara njezina skočna komponenta u kojoj se nalaze osnovni podaci o toj lokaciji kao što su naziv lokacije, tip lokacije, kratki opis te informacijska ikona koja kada korisnik klikne na nju vodi na “LocationDetails” stranicu te lokacije. Također u gornjem desnom kutu skočne komponente nalazi se ikona u obliku križića koja zatvara skočnu komponentu.



Slika 5.2 Stranica “Map”

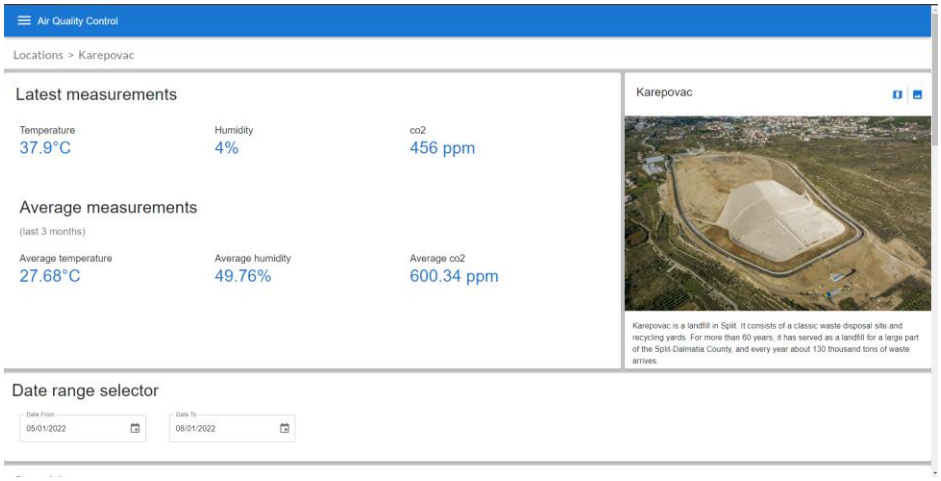
3. **Locations -** Stranica na kojoj su prikazane kartice za svaku lokaciju. Kartice su raspoređene u retke od kojih svaki ima dvije lokacije te su dimenzije istih povezane s veličinom prikaza tako da je njiihov raspored uvijek jednak neovisno o prikazu i uređaju. Svaka kartica u prikazu ima ispisan naziv aplikacije, sliku aplikacije te kratki opis. Također u gornjem desnom kutu kartice nalazi se informacijska ikona koja vodi na “LocationsDetails” stranicu lokacije.



Slika 5.3 Stranica “Locations”

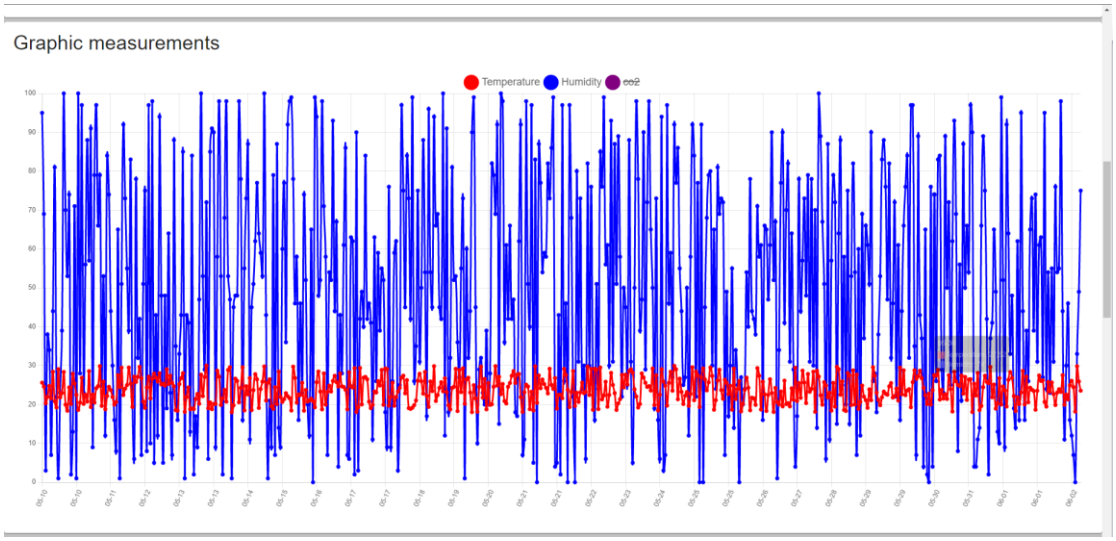
4. **Location Details -** Stranica s najviše sadržaja u aplikaciji. Na vrhu stranice prikazana su posljednja i prosječn mjerenja temperature zraka, vlažnosti zraka te postotka ugljikovog dioksida u zraku. Također na desnoj strani vidljiv je manji prozor u kojem se nalazi naziv lokacije, kratki opis lokacije te izbor između prikaza slike lokacije ili njene pozicije na karti. Mijenjanje prikaza iz slike u kartu i obratno postiže se tako da korisnik u desnom gornjem uglu može izabrati između dvije

ikone, jedna prikazuje kartu, a druga sliku. Nadalje ispod te dvije komponente nalazi se komponenta izbornika raspona datuma za koji će se prikazati mjerenja u grafikonu ili tablici, dok se ispod tog izbornika nalaze grafikon i tablica.



Slika 5.4 Stranica “Location Details” – mjerenja, detalji lokacije i izbornik raspona datuma

U grafikonu su prikazana sva mjerenja za odabrani raspon datuma. Kada korisnik prijeđe pokazivačem miša iznad neke točke otvara se skočna komponenta koja prikazuje sve rezultate mjerenja za datum u kojem ja ta točka prikazana na grafikonu. Na vrhu grafikona je vidljiva legenda u kojoj se mjerenja za određenu mjerenu vrijednost može sakriti s grafikona te on dinamički mijenja svoj prikaz za ostala mjerenja. Na dnu grafikona mogu se vidjeti datumi mjerenja koji se također dinamički mijenjaju s obzirom na to koliki vremenski period je određen za prikaz mjerenja.



Slika 5.5 Stranica “Location Details” – grafikon sa sakrivenim mjerenjima za ugljikov dioksid u vremenskom rasponu od 10.05.2022. do 03.06.2022.

Na dnu stranice “Location Details” nalazi se tablica sa svim mjerenjima za odabranu lokaciju. Tablica ima 5 stupaca u kojima se redom nalaze naziv lokacije, datum i vrijeme mjerenja, izmjerena temperature zraka, postotak vlage u zraku te količina ugljikovog dioksida u zraku. Tablica je zbog velike količine podataka podjeljena na stranice. Svaka stranica ima 24 podatka s obzirom da su mjerenja napravljena tako da se rezultati prikazuju svako sat vremena. Između stranica se navigira pomoću strelica paginacije koja se nalazi u podnožju aplikacije.

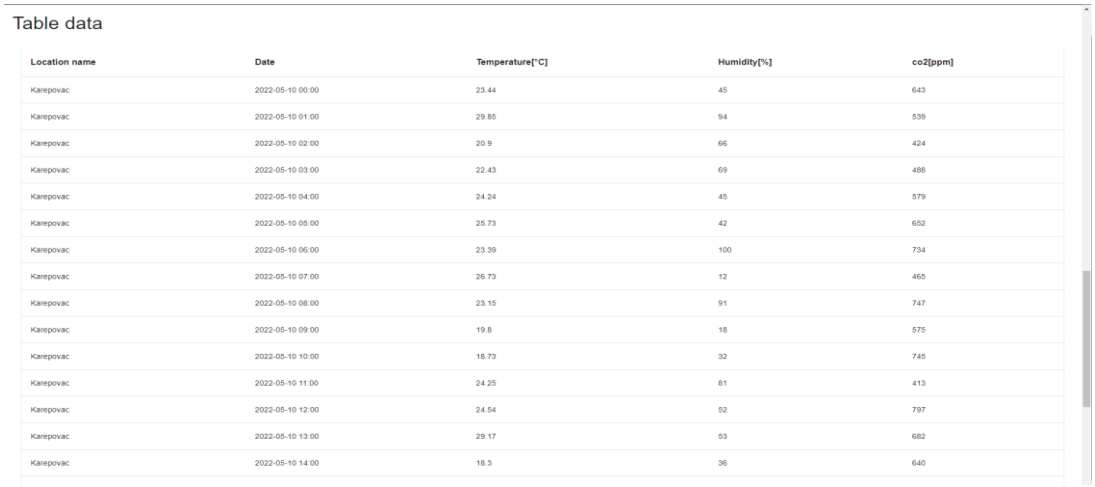


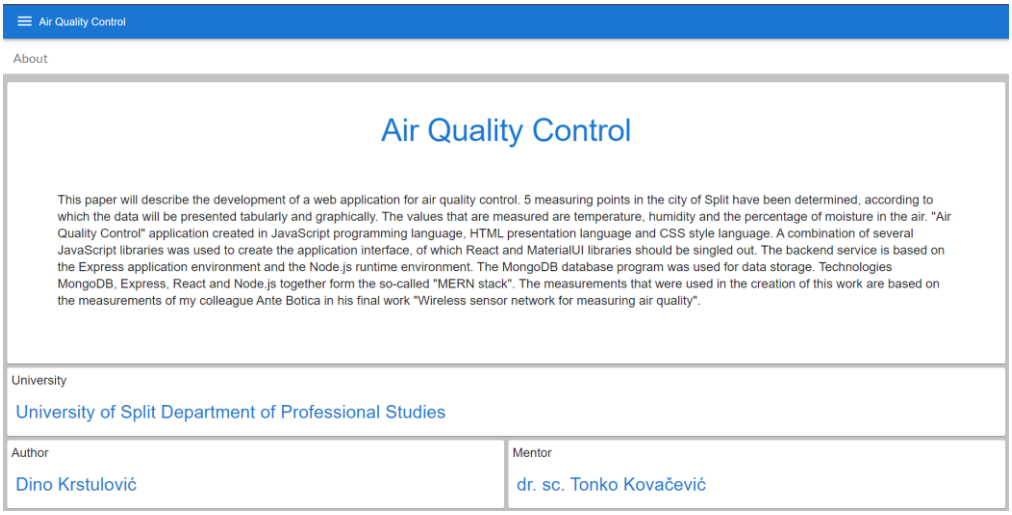
Table data

Location name	Date	Temperature[°C]	Humidity[%]	co2[ppm]
Karepovac	2022-05-10 00:00	23.44	45	643
Karepovac	2022-05-10 01:00	29.85	94	539
Karepovac	2022-05-10 02:00	20.9	66	424
Karepovac	2022-05-10 03:00	22.43	69	488
Karepovac	2022-05-10 04:00	24.24	45	579
Karepovac	2022-05-10 05:00	25.73	42	652
Karepovac	2022-05-10 06:00	23.39	100	734
Karepovac	2022-05-10 07:00	26.73	12	465
Karepovac	2022-05-10 08:00	23.15	91	747
Karepovac	2022-05-10 09:00	19.8	18	575
Karepovac	2022-05-10 10:00	18.73	32	745
Karepovac	2022-05-10 11:00	24.25	81	413
Karepovac	2022-05-10 12:00	24.54	52	797
Karepovac	2022-05-10 13:00	29.17	53	682
Karepovac	2022-05-10 14:00	18.3	36	640

Slika 5.6 Stranica “Location Details” – tablica sa rezultatima svih mjerenja u vremenskom rasponu od 10.05.2022. do 03.06.2022.

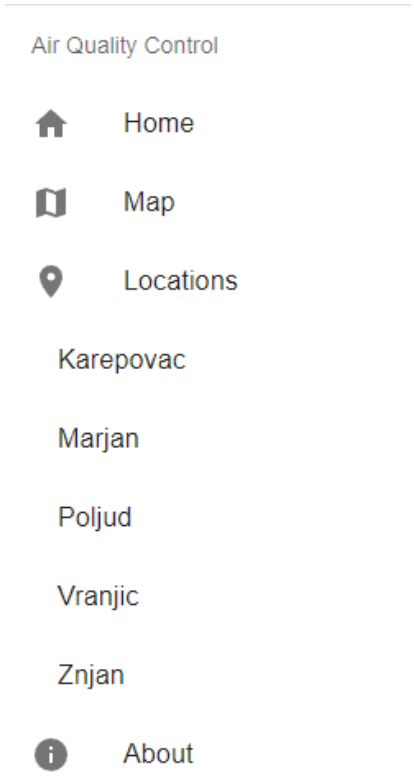
Slika 5.7 Stranica “Location Details” – paginacija tablice za vremenski raspon od 10.05.2022. do 03.06.2022.

5. **About** - Na ovoj stranici može se pronaći naziv aplikacije, tekst iz poglavlja “Sažetak” preveden na engleski jezik, naziv fakulteta i odjela u sklopu kojeg je izrađen ovaj project te imena autora te mentora.



Slika 5.8 Stranica “About”

6. **Menu** - Izbornik aplikacije “Air Quality Control” je osnovna vrsta navigacije među stranicama aplikacije. U zaglavlju izbornika vidljiv je naziv aplikacije. U izborniku se nalaze sve stranice i lokacije u aplikaciji te korisnik samo jednim klikom može otvoriti bilo koju stranicu aplikacije. Stranice i lokacije su raspoređene u retke. U svakom retku za stranicu nalazi se ikona stranice te njen naziv, a za lokacije vidljiv je samo naziv lokacije.



Slika 5.9 Izbornik aplikacije

6. Zaključak

U današnjem svijetu u kojem su ljudi okruženi tehnologijom na svakom koraku jako puno onoga što mogu vidjeti oko sebe temelji se na sustavima koji su obrađeni u ovom radu. Počevši od jednostavnog čitanja portala s dnevnim vijestima, korištenja društvenih mreža, pristupanje stranicama za analizu podataka pa sve do prometih sustava, meteoroloških stanica i razno raznih senzora, to su sve primjene tehnologija iz ovog rada u stvarnom svijetu.

Razvoj ove aplikacije pokazao je dio mogućnosti i moći modernih web aplikacija. Od razvoja web stranice preko modernih biblioteka, paketa i jezika sve do bežične senzorske mreže te obrade podataka mjerenih istom. Iako podaci u ovom projektu nisu rezultati stvarnih mjerenja, već su modelirani po pravim podacima to ne utječe na to koliko mogućnosti i vrijednosti jedna web aplikacija može donijeti "IoT" okruženju.

Mnogo stvari kojima su ljudi okruženi koje danas zahtjevaju manualni rad ili su temeljene na analognim tehnologijama. U budućnosti puno toga na što su ljudi danas navikli će biti modernizirano te automatizirano. Mogućnosti web aplikacija su skoro neograničene te su one u suradnji s drugim tehnologijama budućnost cijelog svijeta.

Literatura

[1] https://www.geeksforgeeks.org pristupljeno: listopad 2022

[2] https://www.netsolutions.com/insights/what-is-a-framework-in-programming pristupljeno: listopad 2022

[3] https://expressjs.com pristupljeno: listopad 2022

[4] https://nodejs.dev pristupljeno: listopad 2022

[5] https://codeforgeek.com/difference-between-mongodb-vs-mongoose pristupljeno: listopad 2022

[6] https://www.freecodecamp.org pristupljeno: studeni 2022

[7] https://developer.mozilla.org/en-US/docs/Web/JavaScript pristupljeno: studeni 2022

[8] https://developer.mozilla.org/en-US/docs/Web/HTML pristupljeno: studeni 2022

[9] https://developer.mozilla.org/en-US/docs/Web/CSS pristupljeno: studeni 2022

[10] https://blog.hubspot.com/website/react-js pristupljeno: studeni 2022

[11] https://reactjs.org pristupljeno: studeni 2022

[12] https://redux.js.org pristupljeno: studeni 2022

[13] https://mui.com pristupljeno: studeni 2022.

[14] https://axios-http.com pristupjeno: studeni 2022.

[15] https://www.tescaglobal.com/blog/what-is-wireless-sensor-network-and-types-of-wsn pristupjeno: studeni 2022.

[16] Botica A.: *Bežična senzorska mreža za mjerenje kvalitete zraka*, Sveučilište u Splitu, Sveučilišni odjel za stručne studije, Split, 2019 pristupljeno studeni 2022.

[17] https://www.digi.com/resources/documentation/digidocs/pdfs/90002002.pdf pristupljeno: studeni 2022.

[18] https://docs.arduino.cc/hardware/nano pristupljeno: studeni 2022.

[19] https://www.winsen-sensor.com/d/files/infrared-gas-sensor/mh-z19b-co2-ver1_0.pdf pristupljeno: studeni 2022

[20] https://randomnerdtutorials.com/bme280-sensor-arduino-pressure-temperature-humidity/#:~:text=The%20BME280%20sensor%20module%20reads,exchange%20data%20with%20a%20microcontroller pristupljeno: studeni 2022

[21] https://www.sainsmart.com/products/xbee-usb-adapter-for-zigbee pristupljeno: studeni 2022.

[22] https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf pristupljeno: studeni 2022.

[23] https://www.digi.com/solutions/by-technology/zigbee-wireless-standard pristupljeno: studeni 2022.

Popis slika

Popis tablica

Popis ispisa