Čitanje, obrada i pohrana JSON-a u web aplikacijama

Tajz, Sven

Undergraduate thesis / Završni rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka / Sveučilište u Rijeci**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:195:982458

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-07-25**

Sveučilište u Rijeci
Fakultet informatike
i digitalnih tehnologija

Repository / Repozitorij:

Repository of the University of Rijeka, Faculty of Informatics and Digital Technologies - INFORI Repository

zir.nsk.hr

DIGITALNA KNJIŽNICA

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

Sveučilište u Rijeci – Fakultet informatike i digitalnih tehnologija

Sveučilišni preddiplomski studij Informatika

Sven Tajz

Čitanje, obrada i pohrana JSON-a u web aplikacijama

Završni rad

Mentor: dr. sc. Vedran Miletić

Rijeka, kolovoz 2022.

Sažetak

Ovaj rad bavi se objašnjavanjem JavaScript Object Notationa (JSON-a), njegove povijesti, načinom na koji funkcionira, to jest kako JSON čita, sprema, obrađuje i šalje podatke unutar web aplikacija. Također će se opisati gdje se sve u web aplikacijama koristi JSON, razne baze podatka koje ga upotrebljavaju, njihove prednosti, mane i razlike. U ovom istraživanju JSON je iskorišten kako bismo podatke spremili i formatirali u željeni format, gdje baza podataka to može pročitati i pohraniti u željene tablice.

Ključne riječi: JSON, XML, podatak, formatiranje, baza podataka

Sadržaj

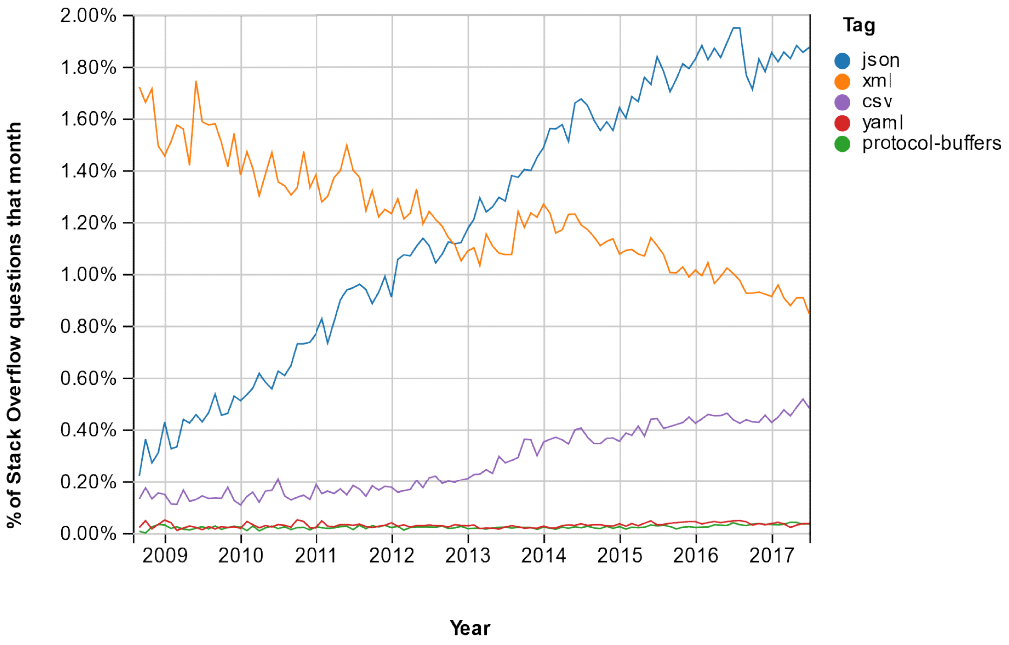
1. Uvod 1
2. Povijest 2
3. Svojstva JSON-a 5
3.1. Način rada JSON-a 5
4. Razlike JSON-a i XML-a 7
5. Upotreba JSON-a u NoSQL bazama (MongoDB) 8
6. Upotreba JSON-a u relacijskim bazama na primjeru MySQL-a 13
7. Zaključak 21
8. Literatura 22
9. Popis Slika 23
10. Popis tablica 24
11. Popis kratica 25

1. Uvod

U današnjem krajoliku razvoja modernih web aplikacija zahtijeva se manipuliranje velikih količina podataka. Podatci mogu biti korisnički podatci kao npr. korisnička imena, lozinke, datumi i puno raznih o korisniku, mogu biti podatci o nekakvom poslovanju itd. Kako bi se to omogućilo na jednostavan i efikasan način koristi se JSON, punim nazivom JavaScript Object Notation. Danas JSON se koristi svugdje, većinom popularnih i uspješnih web aplikacija koristi JSON za čitanje, brisanje, pohranu, ažuriranje i raznih drugih operacija podataka. Najveća prednost JSON-a je njegova jednostavnost, čitljivost i fleksibilnost u smislu gdje svaki moderni programski jezik može koristiti JSON, kao npr. Python, JavaScript itd. Upravo zbog toga je JSON trenutni titan u svijetu razvoja modernih web aplikacija.

2. Povijest

JSON je web API („*Application Programming Interface*“) koji je relativno nedavno postao popularan. Teško je pronaći web aplikaciju u današnjici koja ne koristi JSON tehnologiju za komunikaciju između raznih web aplikacija i razmjene online podataka. Tehnologija postoji već dva desetljeća, ali je tek u 2014. godini postala ECMA standard. Najpopularniji web API prije JSON-a je bio XML koji i u današnjici se koristi, ali niti blizu kao JSON. Prema Stack Overflowu koja je „*question and answer*“ web stranica za programere, više pitanja je postavljeno o JSON-u nego o XML, ili bilo kojem drugom formatu za izmjenu podataka.



Slika 1 Tablica popularnosti formata za izmjenu podataka [4]

Douglas Crockford američki informatičar je osoba koja je odgovorna za razvoj JavaScript jezika i JSON formata i Chip Morningstar, njegov su-osnivač tvrtke State Software su travnja 2001. godine u maloj garaži u okrugu San Fransica poslali prvu JSON poruku ikada. Morningstar i Crockford su pokušavali izgraditi AJAX („*Asynchronous JavaScript and XML*“) aplikaciju, no nisu bili zadovoljni podrškom internetskih preglednika koji su koristili. Prva JSON poruka izgledala je ovako:

```
<html><head><script>

    document.domain = 'fudco';

    parent.session.receive(

        { to: "session", do: "test",

          text: "Hello world" }

    )

</script></head></html>
```

Jedini dio te poruke koji sliči na današnji JSON je funkcija receive(). Nevjerojatna stvar ove prve JSON poruke jest da to očito uopće nije bila prva upotreba nove vrste formata podataka. To je bio čisti JavaScript. Ideja o korištenju JavaScripta na ovaj način je bila toliko jednostavna.Crockford je priznao da on nije prva osoba koja je to učinila. On tvrdi da je programer u *„Netscapeu“* koristio literale niza unutar JavaScripta za prenošenje informacija još 1996. godine. Pošto je poruka čisti JavaScript, nije zahtijevala nikakvo posebno raščlanjivanje, nego je interpreter mogao sve sam raščlaniti.

JavaScript rezervira ogroman broj riječi. Trenutno u JavaScriptu ima 64 rezerviranih riječi. Crockford i Morningstar su nesvjesno upotrijebili jednu u svojoj poruci. Koristili su „*do*“ kao ključ, ali riječ „*do*“ je rezervirana. Budući da JavaScript ima toliko rezerviranih riječi, Crockford kao rješenje za taj problem se odlučio da će i dalje koristiti sve rezervirane riječi, ali će ih pisati unutar navodnika. Ključ s navodnicima bi JavaScript interpreter tretirao kao niz, što znači da se rezervirane riječi mogu sigurno koristiti bez ikakvih komplikacija. Zbog toga se JSON ključevi pišu unutar navodnika i dan danas.

Crockford i Morningstar shvatili su da imaju nešto što se može koristiti u svim vrstama aplikacija. Htjeli su nazvati svoj format "JSML", za *JavaScript Markup Language*, ali su otkrili da se akronim već koristi za nešto što se zove *Java Speech Markup Language*. Stoga su

odlučili koristiti "JavaScript Object Notation" ili JSON. Počeli su je predstavljati klijentima, brzo shvatili da klijenti i investitori nisu htjeli riskirati s novom tehnologijom koja nema niti službenu specifikaciju. Tako je Crockford odlučio da će sam napisati službenu specifikaciju.

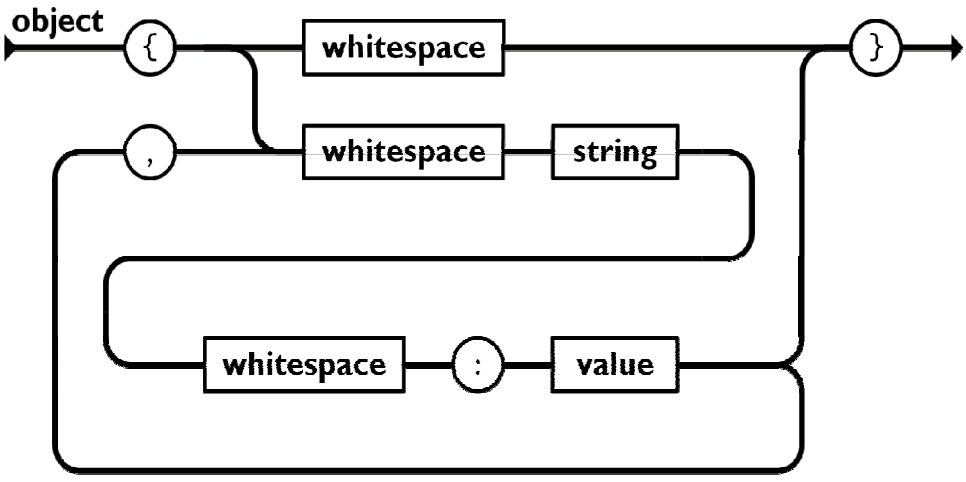
Godine 2002. Crockford je kupio domenu JSON.org i postavio JSON sintaksu i primjer implementacije parsera. Web mjesto je još uvijek u funkciji, iako sada uključuje istaknutu poveznicu na JSON ECMA standard ratificiran 2013. Nakon objavljivanja web stranice, Crockford je učinio malo više za promicanje JSON-a, ali je ubrzo otkrio da puno ljudi podnosi implementacije JSON parsera u svim vrste različitih programskih jezika. Podrijetlo JSON-a jasno ga je povezivalo s JavaScriptom, ali postalo je očito da je JSON dobro prilagođen za razmjenu podataka između proizvoljnih parova jezika.

3. Svojstva JSON-a

JSON ili JavaScript Object Notation je „lightweight“ format za izmjenu podataka. Ono što čini JSON-om toliko popularnim u današnjem razvoju web aplikacija je njegova lakoća čitanja i pisanja za ljudsko oko i lagan je za raščlaniti i generirati računalima. JSON se temelji na podskupu standarda JavaScript programskog jezika ECMA-262 trećeg izdanja. Čak iako je JSON temeljen na JavaScriptu, on je tekst format koji je jezično neovisan, tj. Može se koristiti uz druge jezike kao što su Python, C obitelj jezika, Java, JavaScript itd. Lakoća uporabe i svestranost čine JSON idealnim jezikom za izmjenu podataka.

3.1. Način rada JSON-a

JSON je izgrađen na dvije strukture. Prva struktura je skup parova imena i vrijednosti. U raznim jezicima to se uzima kao objekt, zapis, struktura, rječnik, hash tablica, popis ključeva ili asocijativni niz. Druga struktura je uređena lista vrijednosti. U većini jezika to se realizira kao niz, vektor, lista ili sekvenca.

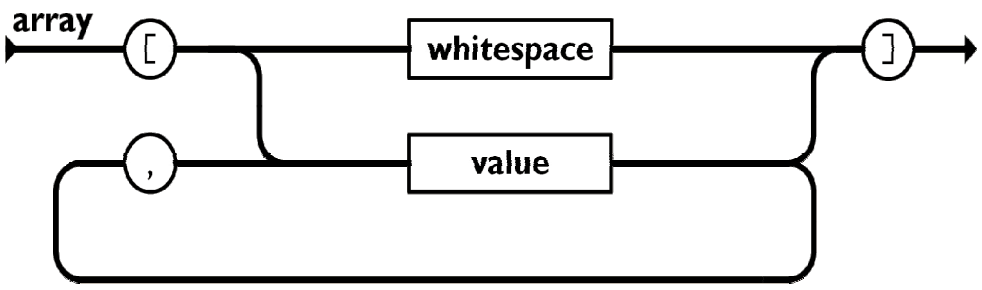


Slika 2 Diagram objekta u JSON formatu

U slici iznad možemo vidjeti strukturu objekta u JSON formatu. Svaki objekt počinje i završava s vitičastim zagradama. Unutar objekta nalazi se par imena i vrijednosti između kojih se nalazi dvotočka kako bi računalo moglo razlikovati između imena i vrijednosti,

također unutar objekta može biti više parova, ali moraju biti odvojeni sa zarezom. Primjer kako bi objekt izgledao u JavaScript kodu:

```
let student = {
      ime: „Sven“,
      prezime: „Tajz“
};
```



Slika 3 Diagram niza u JSON formatu

Struktura niza u JSON formatu prikazan je u slici iznad. Svaki niz započinje s uglatim zagradama, između svake vrijednosti nalazi se zagrada. Vrijednosti unutar niza mogu biti brojevi, objekti, *string* tj. tekst unutar zagrada , drugi nizovi, *true*, *false* ili *null*.

Primjer JavaScript koda koji unutar objekta koristi niz:

```
let garaža = {
      automobili: [“Hyundai“,“Porche“,“Ferrari“]
};
```

4. Razlike JSON-a i XML-a

U prošlom poglavlju objašnjeno je kako funkcionira JSON, ovdje će se objasniti u kratko Extensible Markup Language (XML) i njihove razlike. XML dizajniran je za prijenos podataka, a ne za prikaz podataka. XML je preporuka W3C World Wide Web Consortium je organizacija koja se bavi standardizacijom tehnologija korištenih na webu. Extensible Markup Language (XML) je označni jezik koji definira skup pravila za kodiranje dokumenata u formatu koji je čitljiv i za ljude i za stroj. Ciljevi dizajna XML-a usmjereni su na jednostavnost, općenitost i upotrebljivost na Internetu. To je format tekstualnih podataka sa snažnom podrškom putem Unicodea za različite ljudske jezike. Iako je dizajn XML-a usredotočen na dokumente, jezik se široko koristi za predstavljanje proizvoljnih struktura podataka poput onih koje se koriste u web uslugama.

Tablica 1 Razlike formata JSON i XML izmjene podataka

JSON	XML
JavaScript Object Notation	Extensible markup language
Baziran na jeziku JavaScript	Deriviran od formata SGML
Način reprezentiranja jezika	To je označni jezik i koristi strukturu oznaka za predstavljanje podatkovnih stavki.
Ne pruža nikakvu podršku za prostorna imena	Podržava prostorna imena
Podržava nizove	Ne podržava nizove
Njegove datoteke su vrlo lake za čitanje u usporedbi s XML-om.	Njegove je dokumente razmjerno teško čitati i tumačiti.
Ne koristi završnu oznaku	Koristi početnu i završnu oznaku
Nije toliko siguran	Sigurniji je od JSON-a
Ne podržava komentare	Podržava komentare
Podržava samo UTF-8 enkodiranje	Podržava razna enkodiranja

5. Upotreba JSON-a u NoSQL bazama (MongoDB)

MongoDB je baza podataka koja je odabrala JSON za predstavljanje njihove strukture podataka radi JSONove sveprisutnosti u trenutnom svijetu razboja web stranica i raznih aplikacija za pohranu podataka. Postoje par mana u JSONu koje ga čine manje idealnim za upotrebu unutar baze podataka. Ima manjak podrške datume i binarne podatke. Ovdje dolazi BSON („Binary JSON“). Pomoću BSONa MongoDB može podržavati kompliciranije strukture podataka. Korisnici šalju JSON zahtijeve bazi koja onda sprema podatke u BSON i vraća ih nazad u obliku JSONa opet. JSON mogu čitati ljudi i strojevi dok BSON samo strojevi.

Za primjer upotrebe JSON-a u NoSQL bazi tj. realizacija „create, read, update i delete“ (CRUD) operacija koristit će se baza podataka MongoDB, platforma NodeJS i paket ExpressJS. U ovom specifičnom primjeru prikazuje se jednostavni popis automobila unutar garaže. Za urednu i čitku realizaciju zadatka kod se podijelio u 3 međusobno povezane datoteke „app.js“, „index.js“ i „car.js“.

Kod „app.js“ datoteke priložen je u slici ispod:

Slika 4 Kod datoteke "app.js"

Datoteku app.js koristimo kako bi pokrenuli web API i spojili ga s mongoDB serverom. Od linije 1 do linije 4 deklariraju se konstante s paketima koji će se koristiti za realizaciju CRUD operacija. Unutar linije 6 zahtijeva se datoteka dotenv u kojoj se nalazi adresa i lozinka mongodb servera. U liniji 9 navodi se kako će aplikacija koristiti JSON format. Lokaciju svih ruta koje se nalaze unutar datoteke „garage.js“ povezujemo s datotekom „app.js“ pomoću konstante „garageRoutes“ u liniji 11. Od linije 14 do 20 pomoću funkcije spajamo se na bazu podataka gdje se procesira dotenv podatci radi autentifikacije i prilikom spajanja dobiva se poruka o uspješnom spajanju. Na kraju datoteke navodimo adresu localhosta na kojem želimo testirati web API. U ovom slučaju je to localhost:3000. Kod „garage.js“ datoteke priložen je u slici ispod.

garage.js

```
const express = require('express');
const router = express.Router();
const Car = require('../models/car');


//gets back all cars
router.get('/',async(req,res) => {
    try{
        const cars = await Car.find();
        res.json(cars);

    }catch(err){
        res.status(400).json({message: err.message})
    }
});
// submits a single car
router.post('/',async(req,res) => {
    const car = new Car({
        carName: req.body.carName,
        color:req.body.color
    });
    try{
        const newCar = await car.save()
        res.status(201).json(newCar)
    }catch(err){
        res.status(400).json({ message: err.message})
    }

});

// gets back specific post

router.get('/:carID',async(req,res) =>{
    try {
        const car = await Car.findById(req.params.carID);
        res.json(car)
    }catch{
        res.status(400).json({ message: err.message})
    }
});

//delete a specific post

router.delete('/:carID',async(req,res) =>{
    try {
        const removedCar = await Car.remove({_id: req.params.carID});
        res.json(removedCar)
    }catch{
        res.status(400).json({ message: err.message})
    }
});


// update a post

router.patch('/:carID', async(req,res)=>{
    try{
        const updatedCar = await Car.updateOne(
            {_id:req.params.carID},
            {$set: {carName:req.body.carName}}
        );
        res.json(updatedCar);
    }catch(err){
        res.status(400).json({ message: err.message});
    }

})


module.exports = router;
```

Slika 5 Kod datoteke "index.js"

U liniji 1 deklarira se konstanta express pomoću koje se može koristiti biblioteka expressa, nakon toga u liniji dva definira se konstanta router koja se koristi kako bi se definirale rute za svaku CRUD operaciju. Varijabla Car povezuje garage.js s datotekom car.js u kojoj se nalazi shema podataka svakog automobila, programski kod te datoteke biti će obrazložen poslije.

U linije 7 definirana je *get* metoda ili *read* operacija od CRUD. Cilj ove metode je vračanje svih podataka unutar baze podataka, ili u ovom slučaju automobila u trenutnoj bazi. Uz pomoć prijašnje definirane varijable „router“ može se definirati putanja, ili ruta za „*get*“ metodu koja je u ovom slučaju „/“, koristi se asinkrona tehnika programiranja koja omogućava pokretanje potencijalno dugotrajnog procesa s parametrima req i res tj. „*request*“, „*response*“ za trenutnu *arrow* metodu. Unutar „try“ izjave definira se konstanta „cars“ kojoj je vrijednost niz svih automobila u „inventuri“. Rezultat try izjave definira se unutar linije 10 pod res.json(cars) koji definira da je odgovor u JSON formatu. Catch izjava zaustavi metodu prilikom errora i obavijesti o vrsti errora.

U liniji 17 definira se metoda „*post*“ ili „*create*“ u CRUD. Cilj ove metode je slanje podataka u bazu podataka.Početak metode isti je kao i prijašnjoj get metodi, ali konstanta „car“ ima drugačiju svrhu nego u prijašnjoj metodi. Definira se „novi“ CAR koji sadrži varijable carName i color. Te varijable sadrže parametar req, što označava da se vrijednosti tih varijabla moraju upisati u JSON formatu kako bi post „objavio“ te podatke. Blok programskog koda unutar try izjave proslijedi podatke koje smo upisali i pomoću „await car.save()“ metode spremi u bazu podataka i obavijesti o uspjehu slanja podataka. Kao i u prošloj metodi, catch izjava obavještava u slučaju errora i o kojem se točno erroru radi.

U liniji 33 opet se definira metoda get, ali ova metoda prikazuje specifičan automobil unutar baze. To se realizira tako da je routing adresa „/:carID“, što znači da prilikom slanja „get“ zahtjeva mora se upisati ID automobila za prikaz. Unutar try zahtijeva arrow funkcije deklarira se konstanta „car“ čija je vrijednost automobil koji ima isti ID koji se upisao tijekom get zahtjeva, to se realizira pomoću findbyID metode i require.params.carID. Na kraju šalje se odgovor varijable car u JSON formatu. Catch kao i prije šalje error kod i poruku.

U liniji 44 definirana je delete metoda koja također zahtijeva carID za slanje zahtijeva. Deklarira se konstanta removedCar koja pronađe odgovarajući automobil uz pomoć req.params.carID i uz pomoć Car.remove izbriše taj automobil iz baze. Na kraju šalje se response koji prikazuje koji podatci su se izbrisali.

U liniji 56 definira se patch metoda ili update iz CRUD. Cilj ove metode je ažurirati jedan automobil unutar baze podataka. Ruta metode je također „/carID“ što znači da zahtijeva carID tijekom slanja zahtijeva. Koristi se opet asinkrona arrow funkcija s parametrima request i response. Unutar funkcije definira se konstanta updatedCar koja uz pomoć Car.updateOne metode s parametrom id:req.params.carID koja zahtijeva ID automobila i parametrom $set: carName:req.body.carName pomoću koje možemo ažurirati ime automobila. Na kraju šalje se response koji prikazuje koji podatci su ažurirani.

6. Upotreba JSON-a u relacijskim bazama na primjeru MySQL-a

MySQL podržava spremanje podatak u JSON formatu od verzije 5.7.8. Isto kao u MongoDB, MySQL sprema JSON u binarnom formatu radi podrške za više podatkovnih struktura, ali u MySQl postoji dva glavna nedostatka. Ako JSON ima više polja s istim ključem, samo jedan i to zadnji će biti zadržan. Druga mana je to što MySQL ne podržava indeksiranje JSON stupaca, što znači da prilikom pretraživanja JSON dokumenta može se skenirati cijela tablica umjesto traženih podataka.

Za primjer upotrebe JSON-a u SQL bazi tj. realizacija CRUD operacija koristit će se sustav za upravljanje bazom podataka MySQL, platforma NodeJS te paketi Axios i ExpressJS. U ovom specifičnom primjeru prikazuje se jednostavni popis filmova i kritika za te filmove uz jednostavno grafičko sučelje koje je realizirano pomoću okvira ReactJS. Za urednu i čitku realizaciju zadatka kod se podijelio u dvije međusobno povezane datoteke „app.js“ i „index.js“.

App.js

```
import React,{useState,useEffect} from "react";
import './App.css';
import  Axios  from "axios";
import axios from "axios";

function App() {

  const [movieName,setMovieName] = useState('')
  const [review,setReview] = useState('')
  const [movieReviewList,setMovieList] = useState([])
  const [newReview, setnewReview] = useState("")

  useEffect(()=>{
    Axios.get('http://localhost:3001/api/get').then((response)=>{
      setMovieList(response.data)
    }
    )
  })

  const submitReview = () => {
      Axios.post('http://localhost:3001/api/insert',{

        movieName: movieName,
        movieReview: review,
      }).then(()=>{
        setMovieList([...movieReviewList,
           {movieName: movieName, movieReview:review}])
      })
  }


  const deleteReview = (movie) => {
    Axios.delete(`http://localhost:3001/api/delete/${movie}`);
  }

  const updateReview = (movie) => {
    Axios.put(`http://localhost:3001/api/update`,{
      movieName: movie,
      movieReview: newReview,
    });
    setnewReview("");
  }
```

Slika 6. Datoteka "app.js" MySQL baza 1.dio

```
44
45    return <div className="App">
46      <h1>Crud application</h1>
47
48        <div className='form'>
49          <label>Movie name</label>
50          <input type="text"
51          name="movieName"
52          onChange={(e)=>{
53            setMovieName(e.target.value);
54          }}
55          />
56
57
58          <label>Review</label>
59          <input type="text"
60          name="review"
61          onChange={(e)=>{
62            setReview(e.target.value);
63          }}
64          />
65          <button onClick={submitReview}>Submit</button>
66
67          {movieReviewList.map((val)=>{
68            return <div>
69             <h1>{val.movieName}</h1>
70             <p>{val.movieReview}</p>
71             <button onClick={() => {deleteReview(val.movieName)}}>Delete</button>
72             <input
73              type = "text"
74              id = "updateInput"
75              onChange={(e) =>{
76              setnewReview(e.target.value);
77             }}/>
78             <button onClick={() =>{updateReview(val.movieName)}}>Update</button>
79              </div>
80          })}
81        </div>
82
83      </div>
84  }
85
86  export default App;
87
```

Slika 7 Datoteka "App.js" MySQL baza 2.dio

Cilj datoteke app.js je realiziranje CRUD operacije s jednostavnim grafičkim sučeljem. Programski kod sadržava html elemente, ali objasnit će se samo programski kod koji je relevantan za realizaciju CRUD operacija.

Unutar linije 1 „uvozi“ se ReactJS framework, useState i useEffect udice od React frameworka, u sljedećim linijama uvozi se datoteka „app.css“ koja sadrži sve stilske elemente

za izgled web sjedišta i Axios framework koji omogućuje slanje i primanje HTTP zahtjeva unutar web sjedišta.

Unutar funkcije App nalazi se ostatak programskog koda, to je kod koji sadrži sve CRUD operacije, njihove rute i HTML kod za strukturu web sjedišta. Prvo se definiraju konstante koje će koristiti „useState“ udice od ReactJS frameworka. Pomoću useState udice možemo definirati stanje svake varijable. Inače bi sve varijable bile u obliku objekta, ali ovako može se definirati kakav oblik će imati, u ovom slučaju varijable movieName, review i newReview su u obliku stringa, a varijabla movieReviewList je u obliku niza. Kao što je u prijašnjem programskom kodu korišten express.router() za slanje get, update, delete i post zahtijeva u ovom programskom kodu koristi se Axios. Unutar linije 11 definira se useEffect arrow funkcija koja koristi Axios.get i .then() metodu za slanje get zahtijeva. Programski kod unutar metode .then() izvršava se samo ako je ispunjeno „obećanje“ , u ovom slučaju je obećanje valjana ruta za get zahtjev. Unutar .then() metode lista svih podataka spremaju se u niz varijable seMovieList koja je bila definirana u liniji 8.

U liniji 18 definira se varijabla submitReview čija je vrijednost arrow funkcija unutar koje se realizira programski kod za slanje post zahtijeva. Axios.post sa odgovarajučom rutom sadrži „obećanje“ gdje se moraju upisati JSON objekti movieName i movieReview, ako je to zadovoljeno pokreće se programski kod unutar .then() gdje se u varijablu setMovieList nadodaje u niz novo upisani podatci.

U liniji 30 definira se funkcija deleteReview unutar koje se koristi Axios.delete za slanje delete zahtijeva koji unutar rute sadrži ID recenzije koje se treba izbrisati, ruta izgleda ovako : „http://localhost:3001/api/delete/${movie}“.

Varijabla updateReview definira se u liniji 36. Koristi se Axios.put za slanje update zahtijeva, Axios.put mora sadržavati ID recenzije unutar rute za ažuriranje stavke, u ovom slučaju ruta izgleda ovako:“ http://localhost:3001/api/update“. Također mora sadržavati objekte movieName: movie , i movieReview: newReview. Varijabla setnewReview zamijenjuje trenutnu recenziju s novo upisanom recenzijom.

index.js

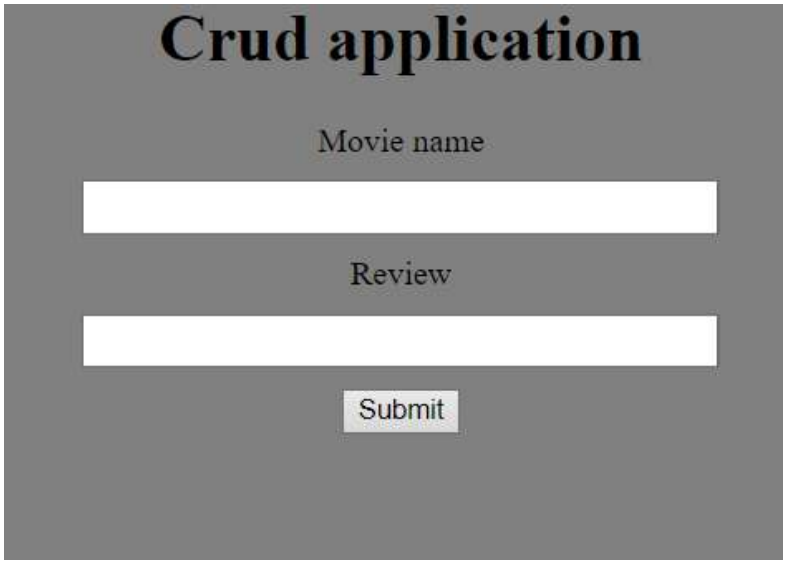
```
const express = require('express');
const bodyParser = require('body-parser');
const app = express ();
const mysql = require('mysql')
const cors = require ('cors');

const db = mysql.createConnection({
    host: "localhost",
    user: "root",
    password: "password",
    database: "cruddatabase"
})

app.use(bodyParser.urlencoded({extended:true}))
app.use(express.json());
app.use(cors());

app.get("/api/get",(req,res) => {
    const sqlSelect = "SELECT * FROM movie_review;"
    db.query(sqlSelect, (err,result) => {
            res.send(result);
        })
})


app.post("/api/insert",(req,res) => {

    const movieName = req.body.movieName
    const movieReview = req.body.movieReview

    const sqlInsert =
    "INSERT INTO movie_review (movieName,movieReview) VALUES (?,?);"
    db.query(sqlInsert, [movieName,movieReview],
        (err,result) => {
            console.log(result);
        })
})

app.delete("/api/delete/:movieName", (req,res) => {
    const name = req.params.movieName;
    const sqlDelete =
    "DELETE FROM movie_review WHERE movieName = ?";
    db.query(sqlDelete,name, (err,result) =>{
       if (err) console.log(err)
    })

})

app.put("/api/update",(req,res) => {
    const name = req.body.movieName;
    const review = req.body.movieReview
    sqlUpdate =
    "UPDATE movie_review SET  movieReview = ? WHERE movieName = ?";
    db.query(sqlUpdate,[review,name], (err,result) => {
        if (err) console.log(err);
    } )
})


app.listen (3001,() => {
    console.log("running on port 3001");
})
```

Slika 8 Datoteka "index.js" MySQL

Unutar datoteke index.js nalaze se sve prijašnje rečene rute za CRUD operacije i varijable koje šalju MySql bazi naredbe u SQL formatu. Efektivno ovo je back-end web sjedišta. Na početku programskog koda definiraju se konstante čije su vrijednosti svi frameworkovi koji će se koristiti za realiziranje zadatka. Express, body-parser, MySql itd. U liniji 7 definira se konstata „db“ tj. baza podataka koja koristi mysql.createConnection kako bi se datoteka spojila na bazu podataka. Metoda mysql.createConnection sadrži parametre host, user password i database. U liniji 15 definira se app.use(express.json()); koji omogućuje da aplikacija parsira samo JSON.

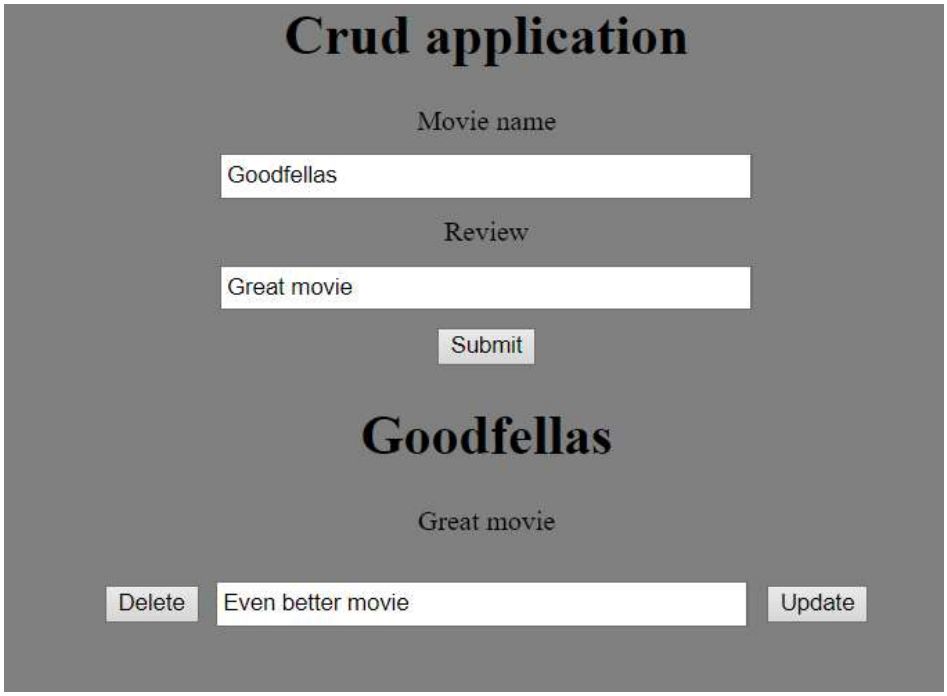
U liniji 18 definira se app.get metoda koja omogućuje slanje get zahtijeva bazi. Sve CRUD operacije su jako slične kao i u prethodnom prikazanom primjeru sa NoSQL bazom, ali ima dvije bitne razlike. Unutar svake CRUD operacije nalazi se konstanta sqlCRUD čije su vrijednosti SQL komande u obliku stringa, u ovom slučaju je to: const sqlSelect = "SELECT * FROM movie_review;" i korištenje metode db.query koja omogućuje pristup i manipulaciju podataka unutar MySQL baze podataka. Za app.get funkciju db,query sadrži konstantu sqlSelect i sve ostale CRUD operacije sadrže odgovarajuću konstantu. App.post sadrži sqlInsert, app.delete sadrži sqlDelete i app.put sadrži sqlUpdate.

Web sjedište se pokrene s komandama „node index,js“ i „npm start“ unutar terminala. Prilikom kompajlirana programskog koda bez ikakvih grešaka unutar web preglednika otvara se web sjedište. Web sjedište koristi zadani dizajn, što bi se lako moglo promijeniti uključivanjem nekog CSS okvira kao Bootstrap ili TailwindCSS.



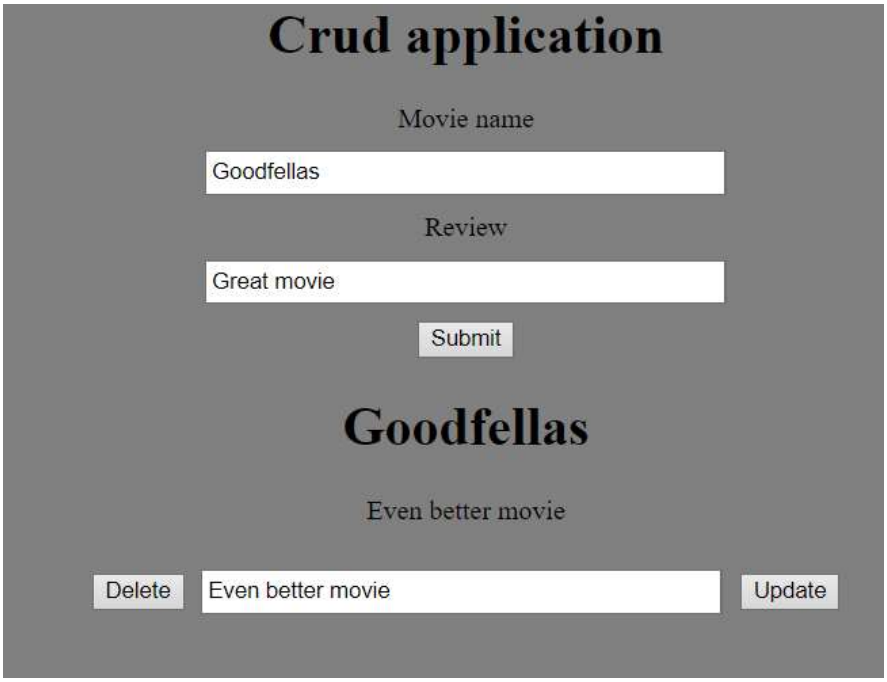
Slika 9 Izgled web sjedišta

Kada korisnik unese odgovarajuće podatke u polja i pritisne “Submit” gumb pokrene se kod linije 20 od “app.js” datoteke i na web sjedištu se prikaže ta stavka.



Slika 10 Unesena stavka na web sjedište

Gumb Delete briše stavku, a gumb Update spremi tekst koji je unesen u polje, spremi ga u JSON format i šalje update request kako bi se ažurirala stavka.



Slika 11 Ažurirana stavka na web sjedištu

7. Zaključak

Ovaj završni rad bavi se opisom standarda JSON-a, funkcionalnosti koju nudi, usporedbom s drugim tehnologijama iste namjene i daje primjere korištenja u modernom razvoju web aplikacija. Kroz dva primjera korištenja, s nerelacijskom bazom MongoDB i relacijskom bazom MySQL, pokazala se fleksibilnost formata JSON, lakoća razumijevanja od strane ljudi i podršku za rad s njim u modernim bazama podataka. Kao i svaka druga tehnologija, JSON ima svoje mane, ali zbog njegove jednostavnosti i fleksibilnosti s razlogom je najpopularniji format za strukturu podataka i uvelike je zamijenio XML u primjeni.

8. Literatura

[1] https://www.mongodb.com/docs/, kolovoz 2022

[2] https://dev.mysql.com/doc/refman/8.0/en/json.html, kolovoz 2022

[3] https://www.json.org/json-en.html, rujan 2022

[4] „The Rise and Rise of JSON“, 2017. , https://www.json.org/json-en.html, rujan 2022

[5] Pedamkar, Priya, n.d. „JSON vs XML“" https://www.educba.com/json-vs-xml/, rujan 2022

[6] „Difference between JSON and XML“ 2019. , https://www.geeksforgeeks.org/difference-between-json-and-xml/, rujan 2022

[7] „JSON Array Literals“, n.d. , https://www.w3schools.com/js/js_json_arrays.asp, rujan 2022

[8] „JSON methods, toJSON“, 2022. , https://javascript.info/json, rujan 2022

9. Popis Slika

Slika 1 Tablica popularnosti formata za izmjenu podataka2

Slika 2 Diagram objekta u JSON formatu5

Slika 3 Diagram niza u JSON formatu6

Slika 4 Kod datoteke "app.js"10

Slika 5 Kod datoteke "index.js"11

Slika 6. Datoteka "app.js" MySQL baza 1.dio15

Slika 7 Datoteka "App.js" MySQL baza 2.dio16

Slika 8 Datoteka "index.js" MySQL18

Slika 9 Izgled web sjedišta19

Slika 10 Unesena stavka na web sjedište20

Slika 11 Ažurirana stavka na web sjedištu21

10. Popis tablica

Tablica 1 Razlike JSON i XML formata izmjene podataka7

11. Popis kratica

JSON – Javascript Object Notation

JSML – Javascript Markup Language

XML – Extensible markup language

SGML – Standard Generalized Markup Language

NoSQL – Non Structured Query Language

SQL - Structured Query Language

AJAX - Asynchornous JavaScript and XML

CRUD – Create, read, update i delete

API – Application Programming Interface

ECMA – European Computer Manufacturers Association

W3C – World Wide Web Consortium

UTF – Unicode Transformation Format

HTTP – Hypertext Transfer Protocol

HTML – Hypertext Markup Language