Usporedba React i HTMX radnih okvira za razvoj klijentske strane web-aplikacija

Colliva, Edi

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

Usporedba React i HTMX radnih okvira za razvoj klijentske strane web-aplikacija

Rijeka, rujan 2024.

Edi Colliva

0069090956

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

Usporedba React i HTMX radnih okvira za razvoj klijentske strane web-aplikacija

Mentor: Izv. prof. dr. sc. Goran Mauša

Rijeka, rujan 2024.

Edi Colliva

0069090956

IZJAVA O SAMOSTALNOJ IZRADI RADA

Izjavljujem da sam samostalno izradio završni rad.

Rijeka, rujan 2024.

_____

Edi Colliva

SADRŽAJ

Popis slika

Popis tablica

1. UVOD

Neizostavni dio razvoja klijentske strane web-aplikacija su radni okviri (eng. frameworks). Oni pružaju vlastite alate, funkcionalnosti i gotove module koji korisnicima olakšavaju razvoj aplikacija i smanjuju potrebu za pisanjem koda od nule. Prate provjerene predloške i prakse koje im omogućuju standardiziran način razvoja, poboljšavaju upravljanje kodom, kvalitetu koda i posjeduju vlastite sigurnosne mehanizme kojima se povećava sigurnost aplikacija.

Zadatak završnog rada bio je izrada dvije identične jednostranične RESTful aplikacije, detaljna analiza i usporedba dvaju radnih okvira, a radni okviri u kojima su razvijene aplikacije su React i HTMX. React je jedna od najpopularnijih knjižnica otvorenog koda koja služi za izradu korisničkih sučelja [1]. Poznata je po tome što razdvaja aplikaciju na komponente, te se onda one mogu mijenjati bez ponovnog učitavanja cijele stranice. HTMX je, s druge strane, relativno nova knjižnica otvorenog koda, prvi puta objavljena krajem 2020. godine, također za razvoj korisničkih sučelja i njezina glavna prednost je jednostavnost i intuitivnost pri učenju i implementiranju [2]. Poslužiteljska strana aplikacije razvijena je u programskom jeziku Java pomoću njezinog radnog okvira Spring Boot-a.

Aplikacija nudi autentifikaciju korisnika i osnovne CRUD (create, read, update, delete) operacije za bilješke. To znači da korisnik ima mogućnost kreiranja bilješki, pregleda svih bilješki, ažuriranja i brisanja istih. Aplikacija koristi HTTP zahtjeve za dohvaćanje podataka s poslužiteljske strane, „get“ metoda koristi se za dohvaćanje bilješki, „post“ metoda za stvaranje novih bilješki, „put“ metoda za ažuriranje postojećih bilješki, dok se „delete“ metoda koristi za brisanje bilješki. U React aplikaciji za upravljanje HTTP zahtjevima koristi se „fetch“ funkcija ugrađena u TypeScript. Podatci se šalju i primaju u formatu JSON, što omogućava jednostavnu razmjenu strukturiranih podataka između klijenta i poslužitelja. U HTMX aplikaciji za slanje HTTP zahtjeva koriste se atributi HTMX-a i URL-Encoded Form Data, što je standardni način razmjene podataka iz HTML forme.

Usporedba radnih okvira prezentirana je kroz brzinu učitavanja i rada stranica, količinu korištenih resursa, usporedbu veličine paketa, SEO-a (search engine optimization) te organizaciju koda razvijenih web-aplikacija. Analizirano je i upravljanje kodom u

aplikacijama, slanje zahtjeva, renderiranje, načini pozivanja funkcija te logika dodavanja elemenata u DOM (document object model). Također, uspoređeni su i jednostavnost, odnosno složenost razvoja, krivulja učenja, dostupnost informacija, te veličina zajednice za svaki okvir.

.

2. TEHNOLOGIJE

U ovom poglavlju bit će opisane sve tehnologije korištene za cjelokupni razvoj dviju web-aplikacija, od tehnologija za razvoj klijentskog i poslužiteljskog dijela, tehnologija za testiranje, do svih ostalih tehnologija i alata potrebnih za razvoj i funkcioniranje aplikacija.

2. 1. React

React je knjižnica programskog jezika JavaScript za izradu korisničkih sučelja web-aplikacija, razvijena od strane Facebook-a [3]. React je deklarativna knjižnica temeljena na komponentama koja korisnicima omogućuje izradu komponenti korisničkog sučelja za višestruku upotrebu i slijedi pristup virtualnog DOM-a, koji optimizira proces renderiranja smanjivanjem broja ažuriranja stvarnog DOM-a .

2.1.1. Integracija React-a s poslužiteljskom stranom

React koristi Node.js i npm (Node Package Manager), Node.js je okruženje za izvođenje JavaScript koda, a preuzima se sa službene stranice Node.js-a [4]. Npm je alat koji dolazi s Node.js ekosustavom, a to je alat za izvršavanje Node.js paketa bez potrebe za njihovom instalacijom. Omogućuje jednostavno pokretanje binarnih datoteka iz lokalnih ili udaljenih paketa. Integracija klijentske strane izvršena je naredbom „npx create-nx-workspace@latest frontend“, zatim je odabran okvir React, paket Webpack, te uređivač CSS [5].

2.1.2. Komponentna arhitektura

Osnovna ideja React aplikacije je podjela na komponente, one su osnovne građevne jedinice aplikacije koje imaju određenu funkcionalnost, a svaka od njih predstavlja jedan dio korisničkog sučelja, međusobno su neovisne i olakšavaju održavanje aplikacije [6]. Komponente posjeduju stanja (eng. state) i svojstva (eng. props), stanja se mogu mijenjati, a svojstva ne mogu jer se nasljeđuju od roditeljskih komponenti. Za korištenje komponenti potrebno ih je uvesti (eng.

import), ponovno su iskoristive, što znači da se pri ponovnom korištenju ne trebaju ponovno implementirati. U ovoj aplikaciji, svaka od CRUD operacija ima svoju komponentu, one primaju podatke od roditeljske im komponente i ugniježđene su unutar nje.

2.1.3. React kuke (hooks)

Kuke (eng. hooks) omogućuju korištenje React značajki u komponentama. Moguće je koristiti ugrađene kuke ili ih kombinirati za izradu vlastitih. Dvije osnovne React kuke su „useState“ i „useEffect“. Kuka „useState“ služi za upravljanje stanjima u komponentama, a pruža i funkciju za promjenu stanja.

Na slici 2.1 prikazana je „useState“ kuka koja služi za upravljanje stanjem, odnosno bilješkama u komponenti. Naziv stanja je „notes“, a „setNotes“ je funkcija za ažuriranje stanja. Prilikom poziva te funkcije, ažurira se polje, te se React komponenta ponovno renderira. „<Note[]>“ označava da je stanje polje objekata tipa „Note“, a početna vrijednost stanja je prazno polje „[]“.

```
const [notes : Note[] , setNotes : React.Dispatch<React.SetStateA... ] = useState<Note[]>( initialState: []);
```

Slika 2.1 „useState“ kuka

Kuka „useEffect“ služi za izvršavanje nuspojava unutar komponente. Na slici 2.2 nalazi se primjer korištenja „useEffect“-a, koristi se za dohvaćanje podataka s poslužiteljske strane kada se komponenta prvi put renderira. „useEffect“ na slici prihvaća dva argumenta, funkciju i niz ovisnosti. Funkcija se izvršava odmah nakon renderiranja komponente, a prazan niz ovisnosti „[ ]“ osigurava da se izvršava samo jednom. Više detalja o izvršavanju funkcija bit će opisano u poglavlju 4.

```
useEffect( effect: () : void => {
    fetch( input: "http://localhost:8080/notes", init: {
        method: "GET"
    }).then(response : Response =>{
        if(response.status == 200){
            return response.json();
        }
        return null;
    }).then(data =>{
        if(data !== null){
            setNotes(data);
        }
    })
}, deps: []);
```

Slika 2.2 "useEffect" kuka

2.1.4. TypeScript

TypeScript je sintaktički nadskup JavaScript-a koji dodaje statičko tipiziranje. To u osnovi znači da TypeScript dodaje sintaksu povrh JavaScript-a, omogućavajući programerima da definiraju tipove [7]. TypeScript omogućava i definiranje strukture objekata i provjeru tipova za složenije strukture podataka. Na slici 2.3 prikazano je sučelje (eng. interface) „Note“. Sučelje je apstraktna vrsta koja opisuje oblik objekta, definira strukturu podataka, ali ne implementira funkcionalnost. „Export“ se koristi za izvoz sučelja, čineći ga dostupnim za uvoz u druge datoteke. Unutar vitičastih zagrada definirani su atributi tog modela, a svaki od tih atributa ima vlastiti tip.

```
export interface Note { Show usages  new *
    id: number;
    title: string;
    content: string;
}
```

Slika 2.3 Interface "Note"

2.2. HTMX

HTMX je jednostavna JavaScript knjižnica otvorenog koda koja se koristi za razvoj klijentske strane dinamičkih web-aplikacija. Njezina glavna prednost je što omogućuje razvoj responzivnog i interaktivnog klijentskog sučelja bez potrebe za složenim radnim okvirima. HTMX koristi deklarativni pristup za definiranje dinamičkih akcija, te omogućuje kontrolu nad sučeljem preko HTML-a.

2.2.1. Integracija HTMX-a u poslužiteljsku aplikaciju

Integracija HTMX-a u postojeću aplikaciju izvodi se vrlo lako i brzo, a značajno poboljšava korisničko iskustvo dodavanjem dinamičnih elemenata bez potrebe za potpunim ponovnim učitavanjem stranice. Prvi korak je kreiranje HTML datoteke u projektu, čije stvaranje se preporučuje u posebnoj datoteci „templates“ unutar direktorija „resources“. U zaglavlje HTML datoteke se zatim dodaje CDN link, što je prikazano na slici 2.4, pri čemu oznaka „2.0.0“ označava najnoviju verziju HTMX-a [8]. Nakon toga, neophodna je konfiguracija poslužiteljske strane, kako bi poslužitelj na zahtjeve klijentske strane odgovarao HTML fragmentima.

```
<script src="https://unpkg.com/htmx.org@2.0.0"></script>
```

Slika 2.4 CDN link HTMX knjižnice

2.2.2. Opis HTMX-a

Struktura HTMX aplikacije vrlo je slična strukturi HTML stranice, uz dodatak HTMX atributa. Atributi čine HTMX vrlo jednostavnim i praktičnim alatom. Građevne jedinice aplikacije nazivaju se HTML fragmenti, a zapravo su označeni HTML elementi. HTTP zahtjevi šalju se na poslužiteljsku stranu putem HTMX atributa kojima se uklanja potreba pisanja JavaScript koda [8]. Na primjer, slanje HTTP „get“ zahtjeva odvija se atributom „hx-get“ u koji se dodaje putanja, a nakon vraćanja odgovora ažurira se ciljani dio stranice (HTML fragment), određen "hx-target" atributom. HTMX također omogućuje upravljanje poviješću preglednika putem atributa „hx-push-url“, na način da se navigacija naprijed natrag koristi bez potrebe za

osvježavanjem cijele stranice što je vrlo korisno za aplikacije s mnogo interakcija i gdje je važno korisničko iskustvo. Omogućava jednostavnu integraciju CSRF zaštite, koja je važna za sigurnost web-aplikacija, a pruža i vrlo koristan „hx-trigger“ atribut, kojim je moguće definirati različite događaje koji pokreću zahtjeve, kao što je klik.

Na slici 2.5 prikazan je kontejner „div“ unutar kojeg se nalazi HTML forma koja služi za dodavanje novih bilješki, odnosno slanje unesenih podataka na poslužiteljsku stranu kada korisnik klikne gumb „Add note“. Atribut „hx-post="/notes"“ definira da će forma koristiti metodu „post“ za slanje podataka na putanju „/notes“ kada se forma podnese. Atributom „hx-target="#operations"“ određuje se da će, kada se vrati odgovor sa poslužiteljske strane, biti umetnut u element koji posjeduje ID „operations“. Atribut „hx-swap="outerHTML“ definira način na koji će se novi sadržaj umetnuti, u ovom slučaju, „outerHTML“ znači da će cijeli element s ID-jem „operations“ biti zamijenjen novim sadržajem sa poslužitelja.

```
<div id="create">
    <h2>Create</h2>
    <form hx-post="/notes" hx-target="#operations" hx-swap="outerHTML" class="content-box">

            <input type="text" name="title" placeholder="Title" required>
            <input type="text" name="content" placeholder="Content" required>

        <button type="submit">Add Note</button>
    </form>
</div>
```

Slika 2.5 Kreiranje bilješke

2.3. Java i Spring Boot

2.3.1. Java

Java je napredni, objektno orijentirani programski jezik koji se temelji na klasama. Njegova višestruka namjena omogućuje pisanje koda jednom, a izvršavanje bilo gdje, na bilo kojem operacijskom sustavu [9]. Java nudi razne alate za automatizaciju, uklanjanje pogrešaka, testiranje i implementaciju, a Java kod obično se kompilira u „bytecode“ koji može biti izvršen na bilo kojoj Java virtualnoj mašini, bez obzira na računalnu arhitekturu. Java ima mnogo

aktivnih korisnika i zajednicu koja može pomoći korisnicima kada naiđu na probleme prilikom razvoja, a softver Java platforme također se redovito održava i ažurira.

Java Spring Boot je radni okvir izgrađen na temelju Spring-a, koji je okvir otvorenog koda za stvaranje samostalnih, proizvodnih aplikacija koje se izvršavaju na Java virtualnoj mašini [10]. Spring Boot čini razvoj web-aplikacija i mikroservisa bržim i jednostavnijim uz osnovne mogućnosti autokonfiguracije i sposobnost stvaranja samostalnih aplikacija. Ove značajke zajedno pružaju alat za postavljanje aplikacije temeljene na Spring-u s minimalnom konfiguracijom i podešavanjem.

2.3.2. Kreiranje Spring boot aplikacije

Kreiranje Spring Boot aplikacije odvija se brzo i jednostavno zahvaljujući alatu Spring Initializr [11]. Na web-stranici „start.spring.io“ moguće je konfigurirati projekt odabirom zavisnosti kao što je „Spring Web“ koji služi za razvoj web-aplikacija. Zatim se projekt generira i spreman je za preuzimanje u obliku ZIP datoteke. Kada se projekt preuzme, raspakira se i otvara u razvojnom okruženju.

2.3.3. Autokonfiguracija

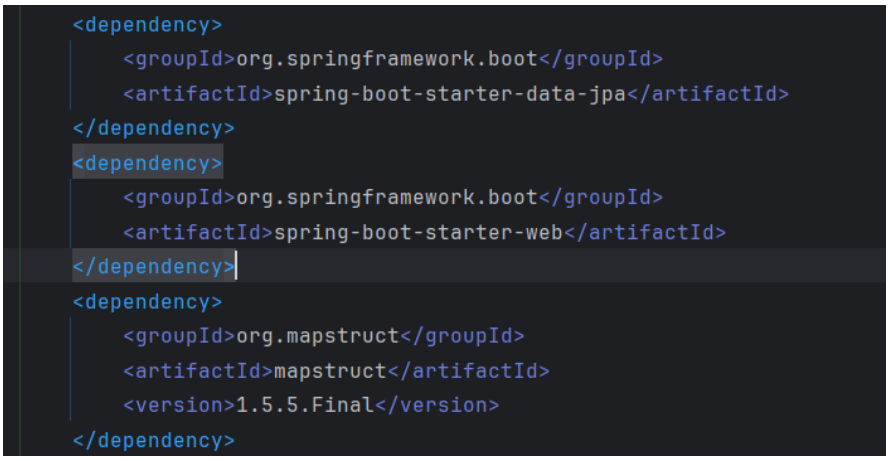
Jedna od glavnih značajki Spring Boot-a je autokonfiguracija, koja značajno pojednostavljuje razvoj aplikacija, automatski konfigurira Spring aplikaciju na temelju zavisnosti koje su definirane u „pom.xml“ datoteci i raznih postavki koje su navedene u konfiguracijskim datotekama [12]. Cilj autokonfiguracije je smanjiti potrebu za ručnom konfiguracijom, omogućujući brzo postavljanje funkcionalne aplikacije bez puno početnog rada.. Na slici 2.6 prikazan je isječak iz „pom.xml“ datoteke, gdje je prikazano nekoliko zavisnosti.

Prva zavisnost dodaje sve što je potrebno za rad s JPA koristeći Spring Data JPA. Omogućuje jednostavno mapiranje Java objekata na relacijske baze podataka i obratno.

Druga zavisnost dodaje sve što je potrebno za razvoj web-aplikacija koristeći Spring MVC (Model-View-Controller). Uključuje ugrađeni Tomcat poslužitelj, koji omogućava pokretanje

aplikacije kao samostalne web-aplikacije bez potrebe za vanjskim aplikacijskim poslužiteljem. Sadrži podršku za RESTful web-servise i Spring MVC, te omogućava rukovanje HTTP zahtjevima i odgovorima.

Treća zavisnost je anotacijski procesor za mapiranje Java objekata (DTO-ova i entiteta). Omogućava automatsko generiranje mapera, što su klase koje mapiraju podatke između različitih objekata na temelju anotacija, čime se eliminira potreba za ručnim pisanjem koda za mapiranje.



Slika 2.6 Zavisnosti aplikacije u "pom.xml"

Spring Boot skenira putanju klase i automatski konfigurira različite Spring bean-ove na temelju prepoznatih zavisnosti. Na primjer, ako je „spring-boot-starter-web“ prisutan na putanji klase, Spring Boot će automatski konfigurirati ugrađeni Tomcat poslužitelj i postaviti osnovne postavke za web-aplikaciju.

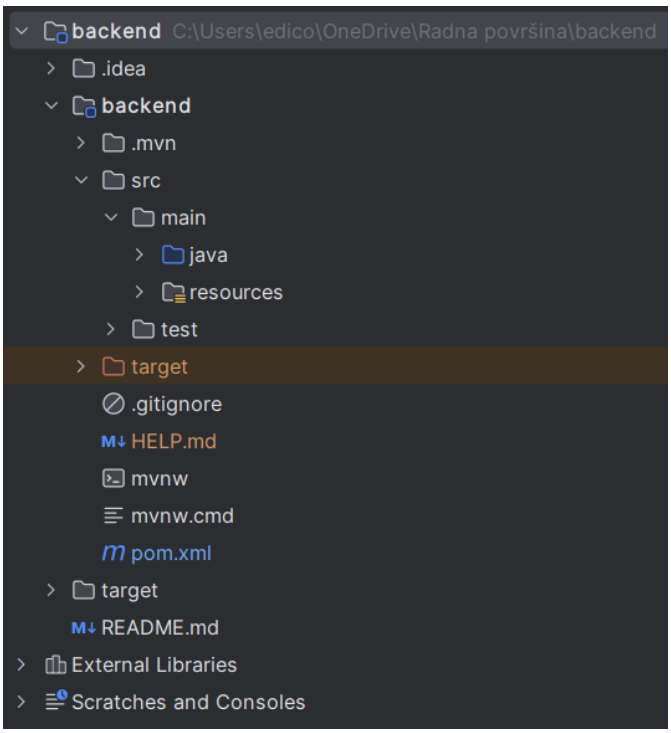
2.3.4. Sposobnost stvaranja samostalnih aplikacija

Spring Boot omogućuje stvaranje aplikacija koje rade samostalno, ne oslanjajući se na vanjski web-poslužitelj i jednostavno se pokreću [12]. To omogućava ugradnja poslužitelja u aplikaciju tijekom procesa inicijalizacije, u ovom slučaju to je Tomcat poslužitelj. Kao rezultat toga, omogućeno je pokretanje aplikacije na bilo kojoj platformi izvršavanjem naredbe za pokretanje.

2.3.5. Struktura Spring Boot aplikacije

Osnovna struktura poslužiteljske Spring Boot aplikacije po direktorijima prikazana je na slici 2.7, a sastoji se od sljedećih elemenata:

1. idea – karakterističan direktorij za IntelliJ IDEA i sadrži konfiguracijske datoteke projekta. Obično nije dio izvornog koda aplikacije, već služi za postavke razvojnog okruženja.
2. backend – glavni, odnosno korijenski direktorij aplikacije.
3. mvn - direktorij koji sadrži Maven Wrapper datoteke. Maven Wrapper omogućava pokretanje Maven nardebi bez potrebe za instalacijom Maven-a na sustav.
4. src – direktorij koji sadrži sav izvorni kod aplikacije, glavni poddirektorij za kod aplikacije mu je „main“.
5. java – direktorij koji sadrži glavne Java kodove aplikacije u kojem se nalaze sve klase i paketi.
6. resources - Ovaj direktorij sadrži resurse potrebne aplikaciji, „application.yml“, statičke resurse (HTML, CSS, JavaScript) i predloške.
7. test - direktorij koji sadrži testove koji provjeravaju funkcionalnost aplikacije.
8. target - direktorij generiran tijekom „build“ procesa koji sadrži kompilirane klase, JAR datoteke i druge datoteke koje Maven stvara tijekom kompiliranja i pakiranja aplikacije.
9. gitignore - datoteka u kojoj se specificira koje datoteke trebaju biti ignorirane od strane. Git-a, a uobičajeno sadrži konfiguracije specifične za razvojno okruženje, „build“ direktorije te zavisnosti.
10. HELP.md - datoteka koja sadrži osnovne informacije o projektu i upute za korištenje ili razvoj aplikacije.
11. mvnw i mvnw.cmd - datoteke su Maven Wrapper skripte za Unix i Windows sustave.
12. pom.xml – datoteka koja definira zavisnosti projekta, „build“ konfiguraciju, informacije o projektu i druge konfiguracije potrebne za Maven „build“ proces.
13. README.md - datoteka koja sadrži informacije o projektu, upute za instalaciju i korištenje. Koristi se za dokumentaciju i obično je prvi kontakt s informacijama o projektu.



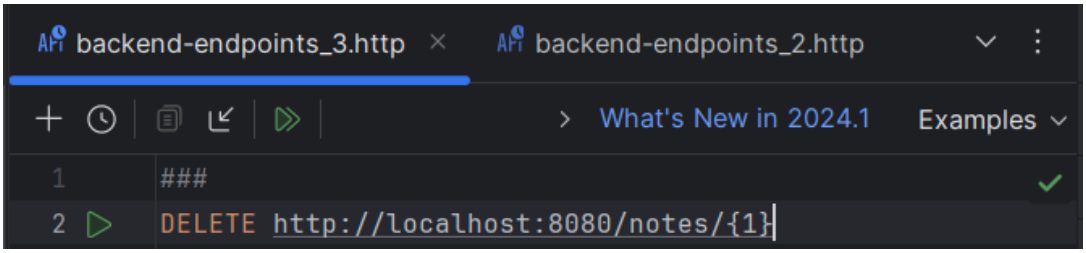
Slika 2.7 Struktura aplikacije

2.4. IntelliJ IDEA

IntelliJ IDEA je napredno integrirano razvojno okruženje, razvijeno od strane JetBrains-a koje je posebno dizajnirano za razvoj aplikacija u programskom jeziku Java [13]. Dizajnirano je za maksimalno povećanje produktivnosti programera. Ono obavlja rutinske i repetitivne zadatke, pružajući pametno dovršavanje koda i statičku analizu koda. Pametno dovršavanje koda znači da pruža mogućnost dovršavanja koda, nudeći prijedloge koji su optimalni za trenutno mjesto u kodu i vrstu podataka s kojima se radi. Statička analiza predstavlja kontinuiranu analizu koda u stvarnom vremenu, otkrivajući potencijalne probleme i poboljšanja čak i prije nego što se pokrene kompilacija. IntelliJ IDEA pruža snažne alate za restrukturiranje koji omogućuju sigurno preimenovanje, premještanje i mnoge druge operacije koje pomažu u održavanju čistog i preglednog koda. Nudi i napredne alate za otklanjanje pogrešaka i integrirana okruženja za testiranje, omogućujući jednostavno pokretanje i provjeru vašeg koda. Osim Jave, IntelliJ IDEA podržava mnoge druge jezike, razvojne okvire i alate, kao što su u ovom slučaju Spring, Spring Boot, JavaScript, TypeScript, React, HTML, HTMX i mnogi drugi. IntelliJ IDEA također krasi brza i vrlo intuitivna navigacija kodom, to uključuje skakanje do definicija, pronalaženje korištenja i implementacija, brzo pretraživanje datoteka i simbola, te druge značajke koje

olakšavaju rad s velikim količinama koda. Pruža podršku za Maven, alat za upravljanje projektima, automatizaciju gradnje, testiranje, pakiranje, i implementaciju Java aplikacija.

Vrlo bitan alat za razvoj aplikacije bio je ugrađeni HTTP klijent koji omogućava jednostavno testiranje i izvršavanje HTTP zahtjeva izravno u razvojnom okruženju. Za ovu funkcionalnost inače je potrebno instalirati potpuno drugi program koji je namijenjen samo tome, te je odvojen od okruženja. Ova funkcionalnost vrlo je korisna za razvoj i testiranje web-aplikacija i API-ja jer omogućuje provjeru na koji način poslužiteljska strana odgovara na HTTP zahtjeve. Na slici 2.8 prikazana je HTTP datoteka za zahtjeve, koja sadrži jednostavan primjer slanja „DELETE“ zahtjeva na poslužiteljsku stranu. Ovaj zahtjev traži od poslužitelja da obriše bilješku čiji je ID jedan. Pritiskom na zelenu strelicu pokraj zahtjeva, on se izvršava, te se odmah vraća povratna informacija o uspješnosti zahtjeva.

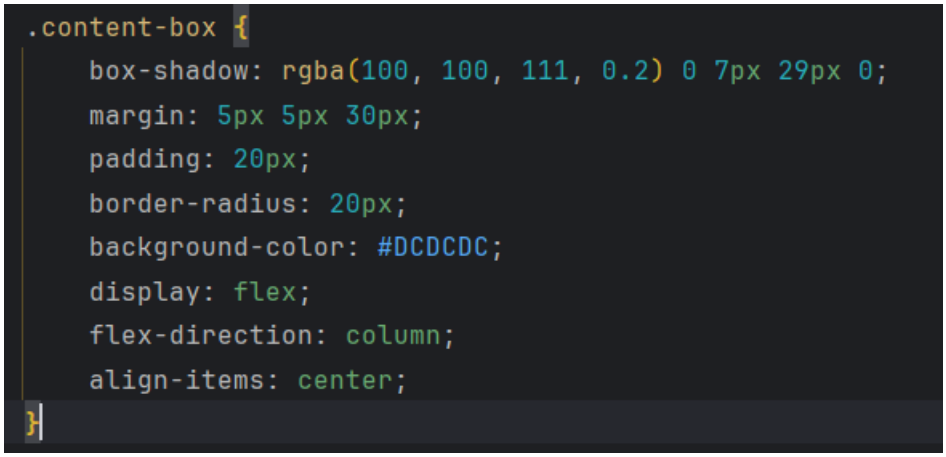


Slika 2.8 HTTP klijent funkcionalnost

2.5. CSS

CSS je stilski jezik za oblikovanje sadržaja u HTML ili XML datotekama. CSS definira izgled i oblikovanje teksta, tablica i drugih elemenata odvojeno od samog sadržaja [14]. CSS atributi se mogu nalaziti unutar HTML datoteka ili u zasebnoj datoteci na koju upućuje više web-stranica. Omogućava stvaranje željenog izgleda cijele web-stranice. Umjesto oblikovanja izgleda svakog elementa i bloka teksta u HTML kodu web-stranice, stil se definira jednom u CSS listi stilova. Uobičajene HTML oznake za oblikovanje, poput <h1>, i <p> imaju prilagođeno oblikovanje definirano u CSS datoteci, prilagođeni stilovi također se mogu primijeniti na tekst, slike i tablice. Jednom kada je stil definiran, može ga koristiti bilo koja stranica koja je povezana na CSS datoteku.

Na slici 2.9 prikazana je CSS deklaracija za klasu „content-box“. Atribut „box-shadow“ dodaje sjenu oko elementa, „rgba“ definira boju, a parametri nakon boje definiraju pomak i zamućenje sjene. „margin“ definira vanjsku marginu oko elementa, „padding“ definira unutarnji razmak unutar elementa, „border-radius“ zaokružuje rubove elementa, „background-color“ postavlja pozadinsku boju, „display: flex“ postavlja element kao fleksibilni kontejner, „flex-direction“ definira smjer fleksibilnog rasporeda unutar kontejnera, te „align-items: center“ služi za poravnavanje elemenata unutar fleksibilnog kontejnera po sredini vodoravno.



Slika 2.9 Primjer CSS-a za klasu

2.6. PostgreSQL

PostgreSQL je objektno-relacijski sustav baza podataka, otvorenog koda, koji koristi i proširuje SQL jezik kombiniran s mnogim značajkama koje sigurno pohranjuju i skaliraju najsloženija podatkovna radna opterećenja. Popularan je zbog svoje provjerene arhitekture, pouzdanosti, integriteta podataka, bogatog skupa značajki, proširivosti i predanosti zajednice otvorenog koda koji stoje iza softvera da dosljedno isporučuje učinkovita i inovativna rješenja. PostgreSQL radi na svim glavnim operativnim sustavima, dolazi s mnogim značajkama koje pomažu pri izradi aplikacija, administraciji u zaštiti integriteta podataka i izgradnji okruženja otpornog na greške, te u upravljanju podatcima bez obzira na veličinu skupa podataka. Osim što je besplatan i otvorenog koda, PostgreSQL je vrlo proširiv, na primjer, možete definirati vlastite tipove podataka, izgraditi prilagođene funkcije, pa čak i pisati kod u različitim programskim jezicima bez ponovnog kompiliranja baze podataka. PostgreSQL nastoji biti u skladu sa SQL standardom tamo gdje takvo usklađivanje ne proturječi tradicionalnim značajkama ili ne dovodi do loših

arhitektonskih odluka. Mnoge značajke koje zahtijeva SQL standard su podržane, iako ponekad s malo drugačijom sintaksom ili funkcijom. Daljnji pomaci prema usklađenosti se mogu očekivati tijekom vremena. PostgreSQL je u skladu s najmanje 170 od 179 obveznih značajki za SQL:2023 Core usklađenost [15].

U aplikaciji, korištenje Spring Data JPA-a zajedno s NotesRepository (repozitorij bilješki) omogućava jednostavan rad s bazom podataka bez potrebe za pisanjem puno ponavljajućeg koda za CRUD operacije. Na slici 2.10 prikazana je konfiguracija za povezivanje aplikacije s PostgreSQL bazom podataka. Konfiguracija je smještena u „application.yml“ datoteci. Najprije se u „datasource“ dijelu definira JDBC driver koji se koristi za povezivanje s PostgreSQL bazom podataka, zatim se specificira JDBC URL koji uključuje domaćina, priključak (eng. port) baze i naziv baze. U „jpa“ dijelu postavlja se Hibernate (JPA implementacija) za rad s PostgreSQL bazom podataka. Hibernate je moćan okvir otvorenog koda za objektno-relacijsko mapiranje koji olakšava rad s relacijskim bazama podataka u Java aplikacijama. Ovdje se koristi za generiranje SQL upita specifičnih za PostgreSQL bazu podataka, te se postavlja postavka „create-drop“, što znači da će Hibernate kreirati shemu baze podataka prilikom pokretanja aplikacije i obrisati je prilikom gašenja aplikacije.

```yaml
spring:
  datasource:
    driver-class-name: org.postgresql.Driver
    url: jdbc:postgresql://localhost:5434/backenddb
    username: backend
    password: backend
  jpa:
    database-platform: org.hibernate.dialect.PostgreSQLDialect
    hibernate:
      ddl-auto: create-drop
```

Slika 2.10 Konfiguracija za povezivanje s bazom podataka

2.7. Docker

Docker je softverska platforma koja omogućuje pakiranje i pokretanje aplikacija u izoliranom okruženju. Docker pakira aplikacije u izolirana okruženja koja se nazivaju kontejneri, oni imaju

sve što je softveru potrebno za pokretanje, uključujući knjižnice, sistemske alate, kod i vrijeme izvođenja. Izolacija i sigurnost omogućuju pokretanje više spremnika istovremeno na određenom poslužitelju [16]. Docker slike su osnovne građevne jedinice za kontejnere, nepromjenjive su i samo za čitanje, sastoje se od niza slojeva, a svaki sloj predstavlja promjenu ili nadogradnju na prethodni sloj. Pohranjuju se u Docker registrima i povlače se prema potrebi. Dockerfile je tekstualni dokument koji definira upute za izgradnju Docker slika, a svaka nova naredba u Dockerfile-u kreira novi sloj u slici. Koristeći Docker, omogućena je brza implementacija i skaliranje aplikacija u bilo koje okruženje s uvjerenjem da će se kod pokrenuti. U aplikaciji, Docker se koristi tako da se prije pokretanja aplikacije pokreće novi kontejner koristeći specificiranu Docker sliku. Konfiguracija slike omogućava jednostavno pokretanje PostgreSQL baze podataka unutar Docker kontejnera, koja je dostupna na specificiranom priključku i sa specificiranim korisničkim podatcima.

2.8. Lighthouse

Lighthouse je automatizirani alat otvorenog koda koji služi za mjerenje kvalitete web-aplikacija razvijen od strane Google-a. Dolazi u obliku Google ekstenzije i vrlo ga je lako ugraditi. Lighthouse se lako pokreće, neovisno o tome je li aplikacija javnog tipa ili je potrebna autentifikacija [17]. Ima mogućnosti mjerenja performansi, pristupačnosti i SEO-a, te pruža mogućnost mjerenja progresivnih web-aplikacija na usklađenost s najboljom praksom. Lighthouse se pokreće u proširenju preglednika Google Chrome ili pomoću terminala, a može mjeriti web-aplikaciju u mobilnoj ili stolnoj (eng. desktop) verziji.

2.9. Apache JMeter

Apache JMeter je aplikacija otvorenog koda napisana u programskom jeziku Java, koristi se u obliku stolne aplikacije. Radi na razini protokola, izgleda kao preglednik, no ne izvršava sve radnje kao preglednik. Namijenjena je za testiranje web-aplikacija, preciznije testiranje performansi, statičkih i dinamičkih resursa, te funkcionalnog ponašanja aplikacije. Omogućuje rad s JSON, HTML i tekstualnim podatcima što odgovara za testiranje ove aplikacije. Omogućuje i simuliranje velikog opterećenja na poslužitelju ili mreži zbog testiranja rada

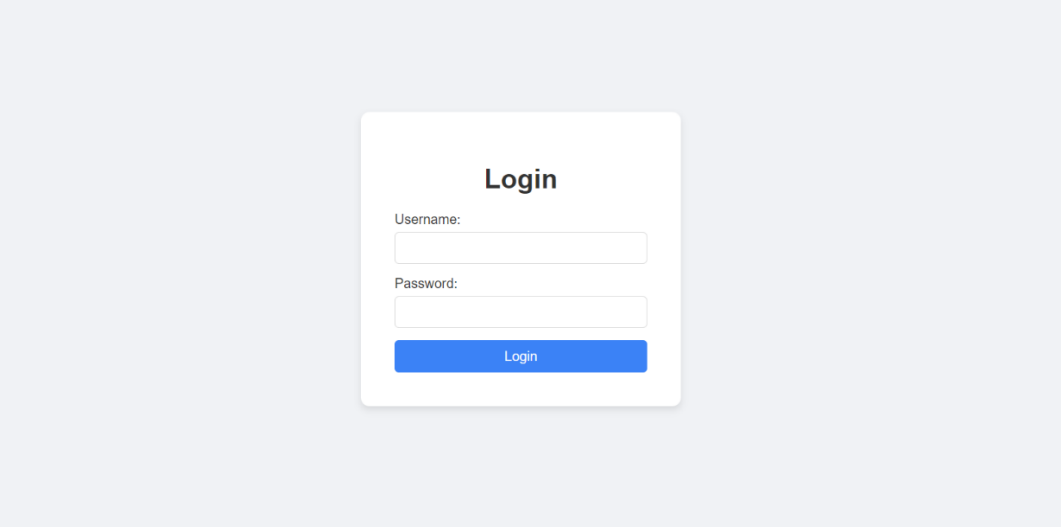
cjelokupne aplikacije pod različitim opterećenjima. Prilikom provođenja testiranja omogućava snimanje cijelog procesa, prikazuje rezultate testiranja, pruža dodatke za analizu podataka i vizualni prikaz rezultata [18].

3. OPIS APLIKACIJE

Aplikacije za upravljanje bilješkama, razvijene u svrhu ovog rada, identičnog su izgleda i pružaju korisnicima mogućnost prijave, omogućavaju učinkovit i intuitivan način za pregled bilješki, dodavanje novih, uređivanje postojećih i brisanje neželjenih bilješki. Aplikacija, osim što je alat za upravljanje bilješkama, prvenstveno služi kao pokazni primjer za demonstraciju mogućnosti modernih web-tehnologija i njihove upotrebe u stvaranju klijentske strane web-aplikacija. U nastavku će biti detaljno objašnjene funkcionalnosti i prikazan dizajn aplikacije, a kasnije će biti provedena i usporedba.

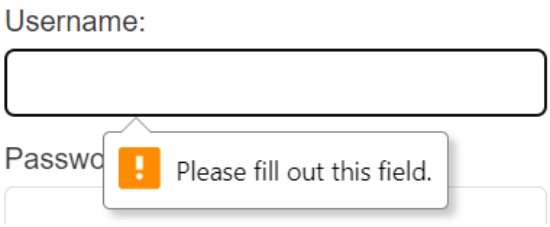
3. 1. Početna stranica

Prilikom pokretanja aplikacije, prikazuje se prozor koji nudi formu za prijavu korisniku. Na slici 3.1 prikazan je početni prozor i forma, upisivanjem odgovarajućeg korisničkog imena (eng. username), lozinke (eng. password) i pritiskom na gumb „Login“ korisnik ima mogućnost autentifikacije, odnosno prijave u aplikaciju. Ukoliko se korisnik uspješno prijavi, otvara se stranica za upravljanje bilješkama. Za potrebe usporedbe radnih okvira nije bilo potrebe za implementacijom složenijeg modela autentifikacije, ovdje se radi o modelu jednostavne autentifikacije gdje su vrijednosti korisničkog imena i lozinke izravno upisane u izvorni kod aplikacije.



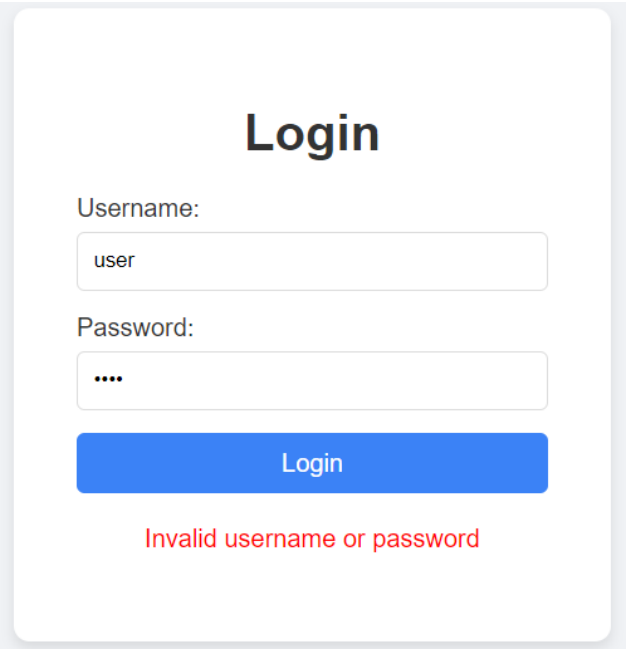
Slika 3.1 Početna stranica

Prilikom unosa podataka za prijavu, ukoliko korisnik ne ispuni obavezna polja, prilikom pritiska gumba za prijavu pojavit će se malena obavijest „Please fill out this field“ ispod unosa koji je prazan. To znači da je to polje obavezno ispuniti prilikom prijave, ta obavijest prikazana je na slici 3.2.



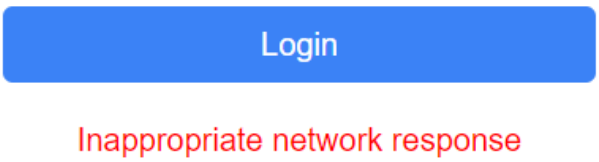
Slika 3.2 „Please fill out this field“ obavijest

Ukoliko korisnik prilikom pokušaja prijave unese pogrešno korisničko ime ili lozinku, odmah ispod forme za prijavu prikazat će se crvena poruka „Invalid username or password“. To znači da jedno od ta dva uvjeta za prijavu nije ispravno, te klijentska strana aplikacije prilikom pokušaja prijave dobiva odgovor „401“, „Unauthorized“. To znači da klijentski zahtjev nije dovršen jer nema valjane vjerodajnice za provjeru autentičnosti za traženi resurs. Poruka je prikazana na slici 3.3.



Slika 3.3 „Invalid username or password“ poruka

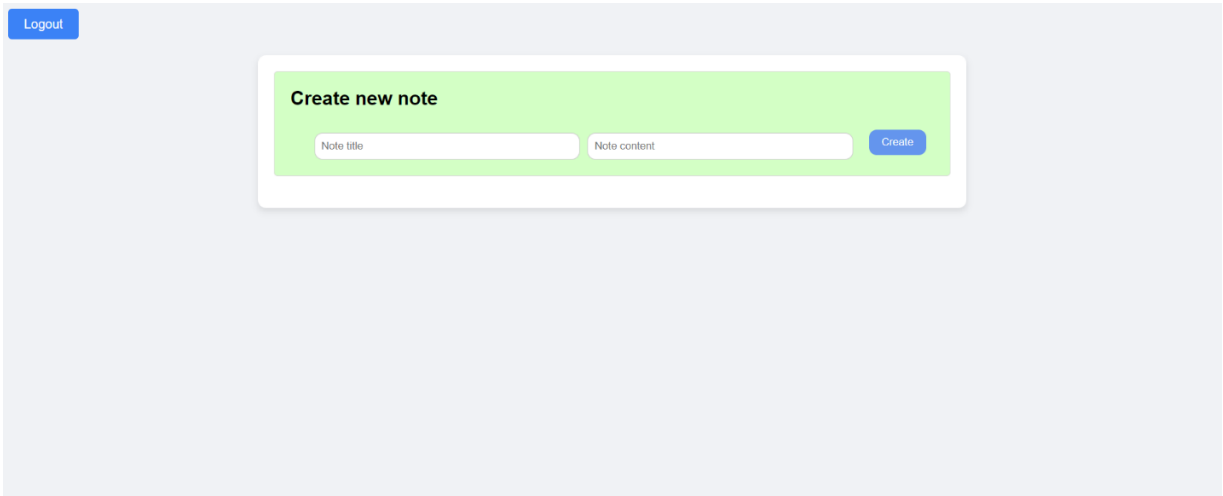
Ukoliko kod pokušaja prijave postoji bilo koji problem druge vrste i klijentska strana kao odgovor dobiva bilo koji drugi statusni kod koji pripada skupini „Client error responses“ ili „Server error responses“, ispod forme za prijavu prikazat će se poruka „Inappropriate network response“. To znači da je klijentska strana dobila odgovor koji nije valjan i poruka je prikazana na slici 3.4.



Slika 3.4 „Inappropriate network response“ poruka

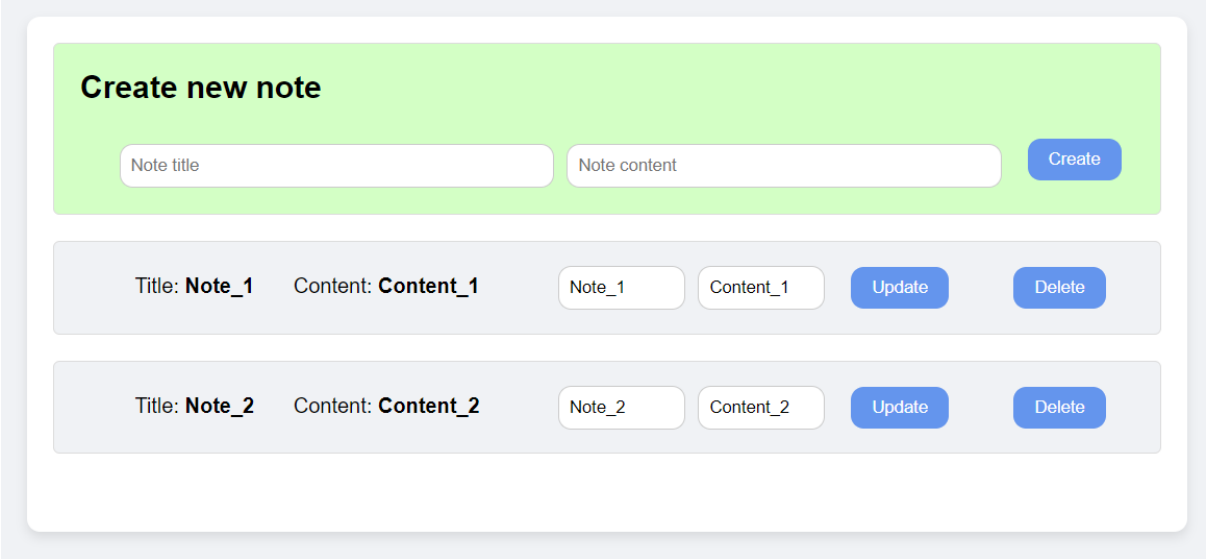
3. 2. Stranica za prikaz bilješki

Prilikom uspješne prijave korisnika učitava se stranica namijenjena autentificiranim korisnicima, prikazana na slici 19. U gornjem lijevom kutu nalazi se gumb „Logout“, pritiskom na njega korisnik se odjavljuje i aplikacija ga vraća na prijašnju stranicu za prijavu. Unutar bijelog kontejnera nalaze se sve opcije za operacije s bilješkama. Na slici 3.5 prikazana je samo opcija „Create new note“ za kreiranje nove bilješke jer trenutno u bazi podataka nema bilješki za prikaz.



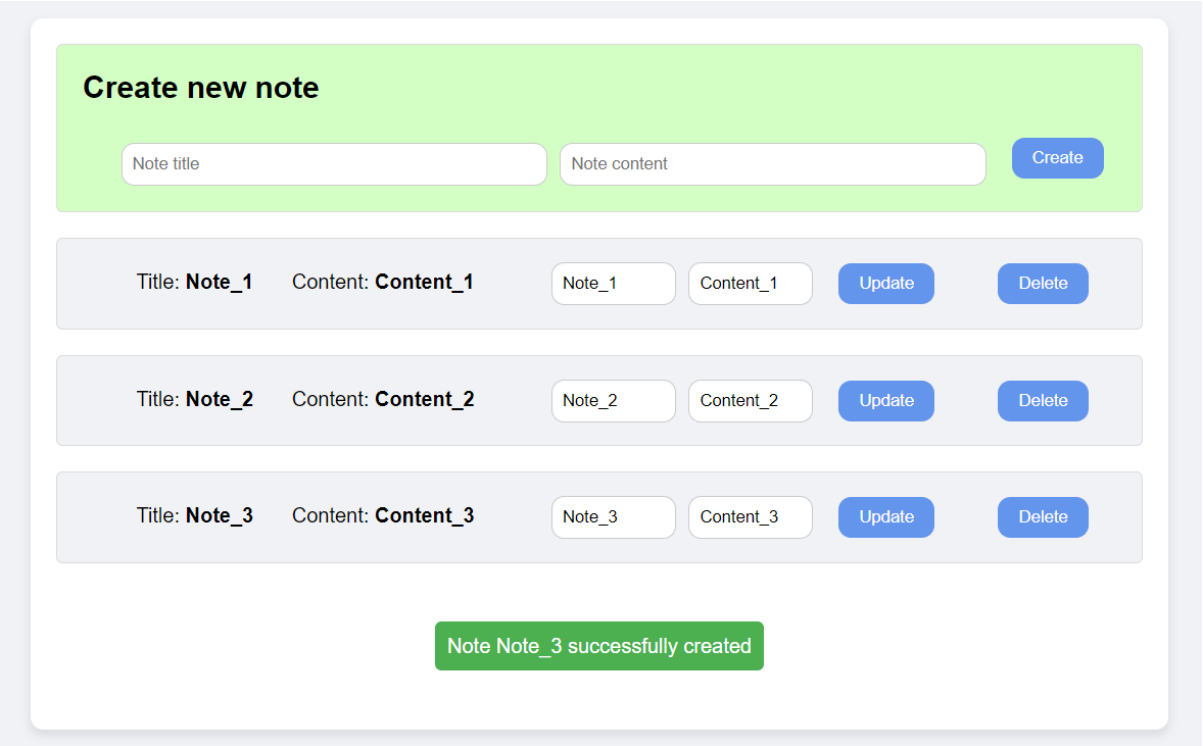
Slika 3.5 Stranica za bilješke

Prikaz stranice, kada u bazi podataka postoje dvije bilješke za prikaz prikazan je na slici 3.6. U sivim sekcijama, koje se nalaze ispod zelene sekcije za kreiranje bilješki nalaze se kreirane bilješke prikazane u redoslijedu od vrha prema dnu stranice. Svaka sekcija predstavlja pojedinu bilješku, a na početku sekcije prikazuje se naslov i sadržaj bilješke.



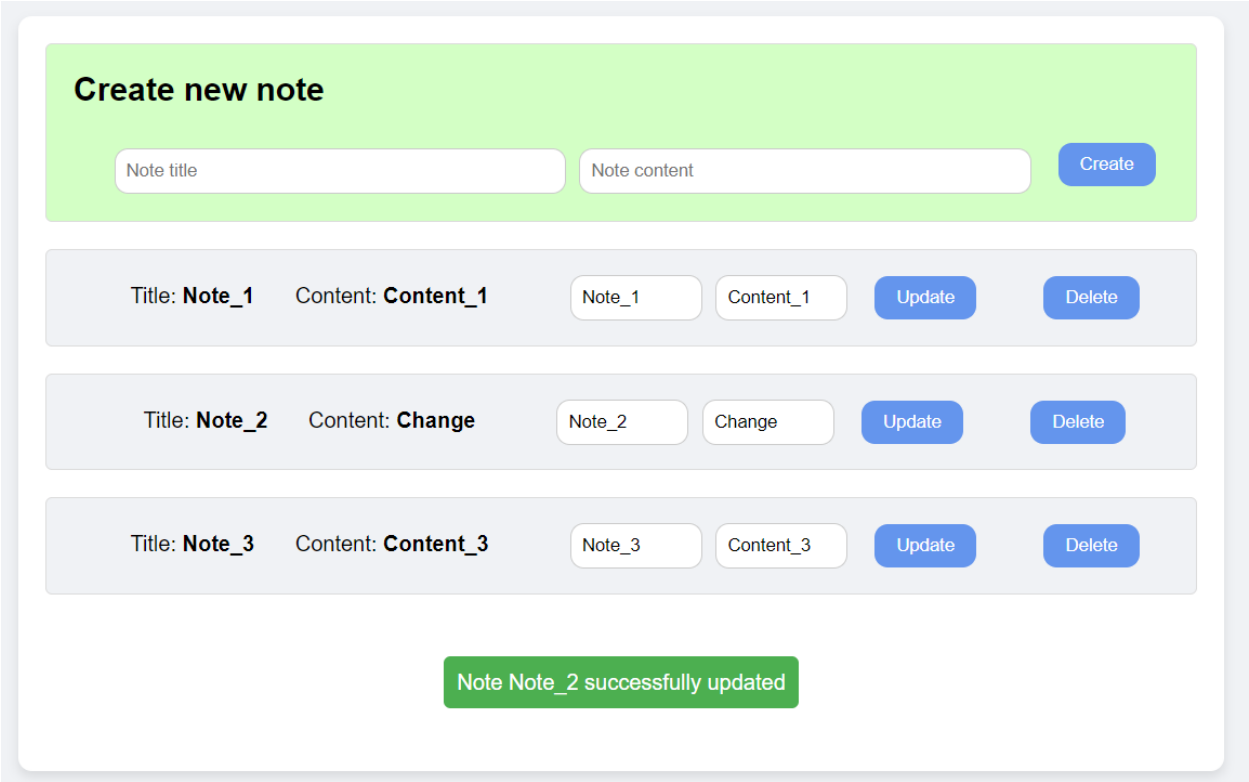
Slika 3.6 Stranica s kreiranim bilješkama

Kreiranje nove bilješke izvršava se ispunjavanjem polja za unos „Note title“ za naslov bilješke, „Note content“ za sadržaj bilješke i potvrđivanjem kreiranja bilješke pritiskom na gumb „Create“. Primjer kreiranja nove bilješke prikazuje se na slici 3.7, nakon uspješnog kreiranja na dnu bijelog kontejnera prikazuje se poruka u zelenom kvadratu koja potvrđuje uspješnost operacije. U ovom slučaju to je „Note Note_3 successfully created“, međutim ako operacija ne prođe uspješno poruka obavještava o neuspješnosti operacije kreiranja.



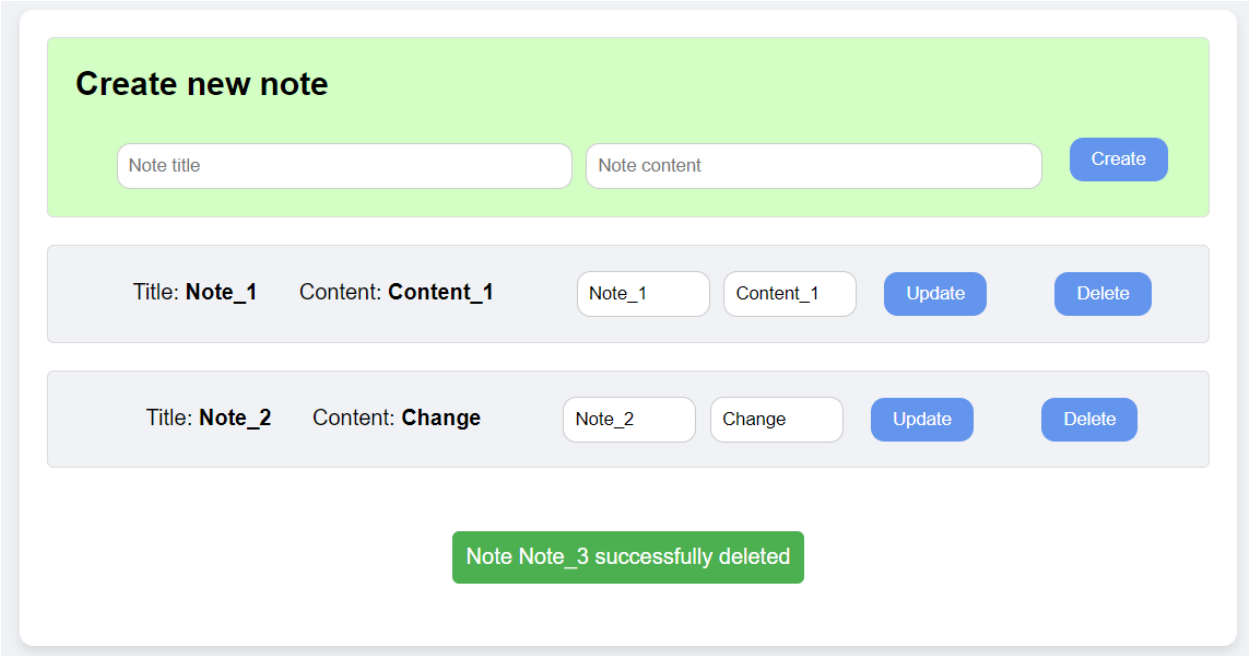
Slika 3.7 Prikaz kreiranja bilješke

U sredini sive sekcije za pojedinu bilješku nalazi se dio sekcije koji pruža operaciju ažuriranja sadržaja. U tom dijelu sekcije nalaze se dva polja za unos novih podataka te gumb „Update“, u tim dvjema poljima već se nalaze stari, odnosno aktualni podatci. Korisnik promjenom tih unosa i pritiskom na gumb „Update“ ima mogućnost ažurirati podatke u bazi podataka. Odmah nakon pritiska gumba ažuriraju se svi podatci o toj bilješci na stranici za prikaz, te se ispisuje poruka u zelenom kvadratu ispod svih bilješki koja potvrđuje uspješnost operacije. U ovom slučaju to je „Note Note_3 successfully updated“, a ukoliko operacija ne prođe uspješno poruka obavještava o neuspješnosti operacije ažuriranja. Primjer operacije ažuriranja sadržaja prikazan je na slici 3.8.



Slika 3.8 Prikaz ažuriranja bilješke

Na desnoj strani sive sekcije za pojedinu bilješku nalazi se dio sekcije koji pruža operaciju brisanja sadržaja. U tom krajnjem dijelu sekcije nalazi se gumb „Delete“ koji služi za potpuno brisanje pojedine bilješke. Pritiskom gumba, osim što se bilješka obriše iz baze podataka, ažurira se i popis bilješki na stranici za prikaz i ispisuje se poruka o uspješnosti operacije u zelenom pravokutniku ispod svih bilješki. U ovom slučaju to je „Note Note_3 successfully deleted“, a ukoliko operacija ne prođe uspješno, poruka obavještava o neuspješnosti operacije. Primjer operacije brisanja sadržaja prikazan je na slici 3.9.



Slika 3.9 Prikaz brisanja bilješke

4. OPIS ODREĐENE FUNKCIONALNOSI NA RAZINI PROGRAMSKOG KODA

U narednom poglavlju bit će opisana funkcionalnost brisanja bilješki putem predstavljanja programskog koda. Biti će prikazane jednake funkcionalnosti putem React i HTMX koda na klijentskoj strani, te kontroleri koji primaju i obrađuju njihove zahtjeve na poslužiteljskoj strani putem Java (Spring Boot) koda.

4. 1. Poslužiteljski dio

Prema Spring MVC arhitekturi, kontroleri se koriste za upravljanje i obradu HTTP zahtjeva, te vraćanje HTTP odgovora klijentskoj strani. Kontroler je u aplikaciji izrađen kao klasa unutar koje se dodaje anotacija „@Contoler“ ili „@RestControler“ koja služi da aplikacija automatski otkrije namjenu te klase pri skeniranju. Razlika u anotacijama kod kontrolera je da se u React aplikaciji koristi „@RestController“ koji služi za vraćanje odgovora u formatu JSON, a „@Controller“ u HTMX aplikaciji koristi se za vraćanje odgovora u formatu HTML-a.

4.1.1. Poslužiteljski dio za React aplikaciju

Na slici 4.1 prikazana je metoda za brisanje bilješki unutar „NoteController“-a (za React aplikaciju) koja se poziva prilikom slanja HTTP „delete“ zahtjeva na putanju „/notes/{id}“ gdje „id“ predstavlja identifikator pojedine bilješke. To joj omogućuje anotacija „@DeleteMapping“. Metoda „ResponseEntity<NoteDto> deleteNote(@PathVariable Integer id)“ prima varijablu „id“, te vraća statusni kod zajedno s tijelom entiteta, a „noteService.deleteNote(id)“ poziva metodu iz klase „noteService“ za brisanje određene bilješke.

```
@DeleteMapping("/notes/{id}")
public ResponseEntity<NoteDto> deleteNote(@PathVariable Integer id) {
    return ResponseEntity.ok(noteService.deleteNote(id));
}
```

Slika 4.1 Funkcija za brisanje u NoteController-u (React aplikacija)

4.1.2. Poslužiteljski dio za HTMX aplikaciju

Na slici 4.2 prikazana je ista metoda za brisanje bilješki unutar „NoteController“-a, ali za HTMX aplikaciju. Metoda „deleteNote(@PathVariable Integer id, Model model)“ prima „id“ i „model“. „noteService.deleteNote(id)“ poziva metodu iz klase „noteService“ za brisanje određene bilješke, „model.addAttribute("notes", noteService.getAllNotes())“ dodaje atribut „notes“ u model i vraća listu svih bilješki. Na kraju metoda vraća pogled (fragment) koji će biti prikazan na klijentskoj strani.

```
@DeleteMapping("/notes/{id}")  Edi *
public String deleteNote(@PathVariable Integer id, Model model) {
    noteService.deleteNote(id);
    model.addAttribute( attributeName: "notes", noteService.getAllNotes());
    return "fragments/operations";
}
```

Slika 4.2 Funkcija za brisanje u NoteController-u (HTMX aplikacija)

4.2. Klijentski dio

Klijentski dio aplikacije sadrži funkcije za slanje HTTP zahtjeva na poslužiteljski dio. U slijedeće dvije sekcije bit će opisane funkcije za slanje HTTP zahtjeva „delete“ u obje aplikacije.

4.2.1 Klijentski dio React aplikacije

Na slici 4.3 prikazana je funkcija koja se poziva prilikom pritiska gumba „Delete“ za brisanje bilješke. Osim za brisanje bilješke, funkcija služi i za ažuriranje stanja u aplikaciji. Funkcija je

tipa „void“, znači da ne vraća ništa, prima „id“ bilješke kao broj, a „const“ u ovom slučaju služi za definiranje konstantne reference na funkciju. Funkcija najprije stvara objekt „headers“ prema klasi „Headers“ koji služi za slanje dodatnih informacija prilikom slanja HTTP zahtjeva, zatim postavlja zaglavlje naziva „Authorization“ u „'Basic ' + btoa(username + ":" + password)“. To znači da se koristi osnovna autentifikacija, a korisničko ime i lozinka kodiraju se u „Base64“ formatu.

Funkcija „fetch“ služi za slanje HTTP zahtjeva na adresu „http://localhost:8080/notes/${id}“, koristeći metodu „delete“ i objekt „headers“ skupa s njim. Nakon što se odgovor na zahtjev vrati, u prvom „then“ bloku provjerava se je li statusni kod odgovora „ok“ (200), ako je to točno funkcija vraća odgovor u JSON formatu, a ako nije funkcija vraća „null“. Odgovor se vraća u idući (drugi) „then“ blok, zatim se provjerava je li odgovor različit od „null“, ako je različit tada se ažurira stanje u listi bilješki te se postavlja poruka o uspješnom brisanju bilješke. Stanje se ažurira korištenjem „useState“ React kuke koja stvara novu listu bilješki u koju uvrštava sve bilješke čiji „id“ je različit od „id“-ja obrisane bilješke. Ako je odgovor „null“ postavlja se poruka o neuspješnom brisanju poruke.

```
const handleDeleteSubmit = (id: number) : void  => {  Show usages  new *
    const headers : Headers  = new Headers();
    headers.set('Authorization', 'Basic ' + btoa( data: username + ":" + password));

    fetch( input: `http://localhost:8080/notes/${id}`,  init: {
        method: "DELETE",
        headers: headers
    }).then(response : Response  => {
        if (response.status == 200) {
            return response.json()
        }
        return null;
    }).then(data => {
        if (data !== null) {
            setNotes(notes.filter(note : Note  => note.id !== data.id));
            setMessage ( value: `Note ${data.title} successfully deleted`);
        }else {
            setMessage( value: `Failed to delete the note`);
        }
    });
};
```

Slika 4.3 Funkcija za brisanje bilješke (React)

4.2.2. Klijentski dio HTMX aplikacije

Koristeći HTMX programski okvir, nema potrebe za pozivanjem dodatne funkcije kod funkcionalnosti brisanja bilješke. HTMX se integrira izravno u HTML kod te na taj način pojednostavljuje izradu klijentske strane aplikacije. Koristeći HTMX funkcionalnosti, klijentski HTTP zahtjev „delete“ šalje se na poslužiteljski dio izravno iz HTML forme, odnosno u ovom slučaju pritiskom gumba „Delete“. Unutar HTML elementa „button“ definirani su svi atributi potrebni za pravilno i uspješno slanje zahtjeva. Funkcionalnost brisanja prikazana je na slici 4.4.

„hx-trigger="click"“ je HTMX atribut koji služi za definiranje da će se operacija pokrenuti prilikom pritiska na gumb.

„hx-target="#operations"“ je atribut koji služi za određivanje elementa koji će se ažurirati nakon što se odgovor na zahtjev vrati, to će ovdje biti element s „id“-jem „operations“.

„hx-swap="outerHTML"“ je atribut koji određuje da će se prilikom ažuriranja promijeniti cijeli odabrani element. „innerHTML“ bi značilo da se mijenja samo unutrašnjost elementa.

„th:attr="hx-delete=@{'/notes/' + ${note.id}}"“ je atribut šablonskog jezika „Thymeleaf„ koji služi za definiranje putanje na koju se šalje „delete“ zahtjev koristeći „id“ bilješke. Ista putanja kao i u React aplikaciji.

„hx-headers='{"Authorization": "Basic dXNlcm5hbWU6cGFzc3dvcmQ="}'“ je atribut koji definira zaglavlje (eng. header) koje se šalje zajedno s zahtjevom. Naziv zaglavlja je „Authorization“, a tijelo mu znači da koristi osnovnu autentifikaciju s korisničkim imenom i lozinkom kodiranim u „Base64“ formatu.

Posljednji atribut „hx-on:htmx:afterRequest="…““ služi za definiranje operacija (JavaScript kod) koji će se sprovesti nakon što se vrati odgovor sa poslužiteljske strane. Ako se zahtjev uspješno izvrši, u elementu s „id“-jem „message-container“ ispisat će se poruka „Note ${note.title} successfully deleted“, a ako bude neuspješan ispisati će se „Failed to delete the note“.

```
<div  class="content-box">
    <button
        hx-trigger="click"
        hx-target="#operations"
        hx-swap="outerHTML"
        th:attr="hx-delete=@{'/notes/' + ${note.id}}"
        hx-headers='{"Authorization": "Basic dXNlcm5hbWU6cGFzc3dvcmQ="}'
        hx-on:htmx:afterRequest="
            if (event.detail.successful) {
                document.querySelector('#message-container').innerHTML = 'Note ${note.title} successfully deleted';
            } else {
                document.querySelector('#message-container').innerHTML = 'Failed to delete the note';
        }">
    >
        Delete
    </button>
</div>
```

Slika 4.4 Funkcionalnost brisanja (HTMX)

5. USPOREDBA REACT I HTMX RADNIH OKVIRA

S obzirom da se HTMX našao na drugom mjestu 2023. godine u kategoriji JavaScript Rising Stars „Front-end Frameworks" (odmah iza React-a) [1], našao mjesto u GitHub Acceleratoru i dobio preko 20 tisuća zvjezdica na GitHub-u, nastala je ideja za usporedbom s najpoznatijim radnim okvirom React-om.

Cilj HTMX-a je pružiti modernu interaktivnost preglednika izravno unutar HTML-a, bez potrebe za JavaScript kodom, u odnosu na React koji se kompletno zasniva na JavaScript kodu (u slučaju ove aplikacije TypeScript – sekcija 2.1.5). HTMX proširuje mogućnosti HTML-a koristeći prilagođene atribute pomoću kojih šalje AJAX zahtjeve na poslužiteljsku stranu, za razliku od React-a koji svoje zahtjeve šalje na standardan način „fetch" funkcijom. Osnovna ideja HTMX-a, za razliku od React-a nije da klijentska strana razmjenjuje podatke izravno s poslužiteljskom stranom, nego umjesto toga poslužiteljska strana priprema HTML koji će biti prikazan na stranici. Taj HTML može biti pojedini dio (fragment), kao u slučaju ove aplikacije ili cijeli sadržaj stranice.

U ovoj usporedbi HTMX-a i React-a, bit će prikazana analiza njihove arhitekture, mogućnosti, performanse, potrošnje resursa, način razvoja, zajednice i krivulje učenja. U tablici 5.1 prikazana je usporedba HTMX-a i React-a po sekcijama.

Segment	HTMX	React
Arhitektura	HTML orijentirana s minimalnim JavaScript-om	Temeljena na komponentama
Renderiranje	Renderiranje na poslužiteljskoj strani	Različiti načini renderiranja
Krivulja učenja	Niska, fokus na jednostavnosti	Visoka, fokus na skalabilnosti
Veličina paketa	Manja	Veća (10 do 20 puta)
Performanse	Bolje za jednostavne aplikacije	Lošiji za poslovne aplikacije, ali može biti optimiziran
Upravljanje stanjima	Jednostavno	Robusno
Renderiranje na poslužiteljskoj strani	Savršena integracija	Snažna podrška
Zajednica i ekosustav	Rastući	Izgrađen
Upotreba	Inkrementalno usvajanje	Složena sučelja i jednostranične aplikacije

Tablica 5.1 Usporedba HTMX-a i React-a po segmentima [19]

5.1. Arhitektura

5.1.1. Arhitektura React-a

Arhitektura React-a, kao alata za razvijanje interaktivnih i dinamičkih web-aplikacija, temelji se na konceptima komponenata i virtualnom DOM-u. Arhitektura obuhvaća cjelokupni raspored i dizajn komponenti, te definira načine na koje različite komponente samostalno i međusobno djeluju. Takva arhitektura pruža aplikacijama fleksibilnost i bolje uvjete za promjene ili nadogradnje kako bi se aplikacija lakše i jednostavnije održavala ili mijenjala [20]. Komponente, koje su osnovne građevne jedinice aplikacije, enkapsulirane su, što znači da jedna komponenta ugrađuje drugu komponentu njezinim pozivanjem. Ta je komponenta samostalna te enkapuslira svoj izgled, stanje i funkcije. Komponente samostalno upravljaju svojim stanjem, a jedna od njihovih glavnih prednosti je i da se one mogu koristiti bilo gdje u kodu i koliko god puta. Omogućuju aplikaciji modularnost i razdvajanje zahtjeva aplikacije te na taj način pojednostavljivanje problema i zadataka. Razlikujemo dvije vrste komponenti, komponente stanja i komponente bez stanja. Komponente stanja imaju ulogu održavanja stanja u kojima se spremaju informacije o komponentama, one su obično roditeljske komponente i prosljeđuju podatke putem svojstva. Komponente bez stanja nemaju mogućnost održavanja i upravljanja stanjima. One služe za korisnička sučelja i primaju podatke putem svojstva i prikazuju ih u DOM-u.

Temeljni koncept React arhitekture je i virtualni DOM, pružajući prikaz stvarnog DOM-a u memoriji. Kada dođe do promjene u stanju komponente, React stvara novi virtualni DOM i uspoređuje ga s prijašnjim. Na taj način ima mogućnost ažuriranja samo dijelova stvarnog DOM-a koji su promijenjeni u virtualnom DOM-u, umjesto ažuriranja cijelog DOM-a, čime se smanjuje broj stvarnih manipulacija DOM-om i poboljšavaju performanse.

Na slici 5.1 prikazan je dio datoteke „App.tsx“ u kojem se prikazuje uvoz komponenti „ReadContentBox“, „CreateContentBox“, „DeleteContentBox“ i „UpdateContentBox“. Te se komponente u komponenti „App“ koriste kao komponente djece, odnosno komponenta „App“ je njihov roditelj i ugrađuje ih.

```
import ReadContentBox from "../content-box/ReadContentBox";
import CreateContentBox from "../content-box/CreateContentBox";
import DeleteContentBox from "../content-box/DeleteContentBox";
import UpdateContentBox from "../content-box/UpdateContentBox";
```

Slika 5.1 Uvoz React komponenti

Na slici 5.2 prikazan je također dio datoteke „App.tsx“ u kojemu se prikazuje način ugradnje uvezenih komponenti „ReadContentBox“, „CreateContentBox“, „DeleteContentBox“ i „UpdateContentBox“ za prikaz unutar „App“ komponente. Prilikom poziva komponenata „CreateContentBox“, „DeleteContentBox“ i „UpdateContentBox“ definiraju se i funkcije za svaku od njih koje će automatski biti pozvane prilikom potvrde, a za „DeleteContentBox“ i „UpdateContentBox“ postavljaju se i podatci o bilješkama „content“ koji će biti poslani u komponente i služiti im za njihove potrebe.

```
<div className="create-section">
    <h2>Create new note</h2>
    <CreateContentBox onSubmit={handleCreateSubmit}/>
</div>
{notes.map(note : Note  => (
    <div key={note.id} className="note-section">

        <ReadContentBox content={note}/>
        <UpdateContentBox onSubmit={handleUpdateSubmit} content={note}/>
        <DeleteContentBox onSubmit={handleDeleteSubmit} content={note}/>
    </div>
))}
```

Slika 5.2 Korištenje komponenti

Na slici 5.3 prikazan je izvoz komponente kako bi se ona dalje mogla koristiti.

```
export default App;  Show usages  Edi
```

Slika 5.3 Izvoz komponente

5.1.2. Arhitektura HTMX-a

Arhitektura HTMX-a omogućava kreiranje interaktivnih i dinamičkih web-aplikacija sa minimalnim pisanjem JavaScript koda, koji mu omogućuje HDA (Hypermedia Driven Application). HDA arhitektura proširuje postojeću HTML infrastrukturu kako bi omogućila razvoj naprednijih interakcija pogonjenih hipermedijom (HTML-om). Umjesto komunikacije s poslužiteljskom stranom korištenjem JSON formata kao kod React-a i drugih JavaScript radih okvira, HDA komunicira s poslužiteljskim dijelom pomoću hipermedije [21]. Za postizanje interaktivnosti na klijentskoj strani, HDA koristi deklarativnu sintaksu ugrađenu u HTML, umjesto imperativne skripte. Pisanje skripte koristi se jedino u slučaju poboljšanja iskustva na klijentskoj strani aplikacije. Na slici 5.4 prikazan je primjer HDA fragmenta koji ima funkciju stvaranja novih bilješki. Bitne karakteristike HDA prikazane na primjeru su da je prednji dio značajke u potpunosti naveden u deklarativnim HTMX atributima unutar HTML-a, te da se interakcija s poslužiteljskim dijelom odvija putem HTML-a.

```
<h2>Create new note</h2>
<form hx-post="/notes" hx-target="#operations" hx-swap="outerHTML" class="content-box">

 <input type="text" name="title" placeholder="Title" required>
 <input type="text" name="content" placeholder="Content" required>

<button type="submit">Add Note</button>
</form>
```

Slika 5.4 Primjer HDA fragmenta

5.2. API zahtjevi

API zahtjev predstavlja glavni način komunikacije između korisnika i API-ja (aplikacijskog programskog sučelja) koji omogućuje dohvaćanje podataka i izvršavanje određenih zadataka. API-ji su mehanizmi koji omogućuju dvjema softverskim komponentama međusobnu komunikaciju pomoću skupa definicija i protokola.

Za razliku od standardnih interakcija web-stranica, API zahtjevi nude poboljšanu fleksibilnost, omogućujući fluidnu integraciju s različitim izvorima podataka, softverske interakcije, automatizaciju zadataka i inovativne metode pristupa podatcima.

API zahtjev sastoji od URL-a (standardiziranog načina identificiranja izvora) koji navodi lokaciju željenog resursa i HTTP metodu (npr. „get“) koja predstavlja željenu radnju. Dodatni parametri zahtjeva mogu se uključiti kako bi se pružili specifični detalji, proširujući funkcionalnost zahtjeva. API zahtjevi putuju od klijentske strane do API krajnje točke.

5.2.1. API zahtjevi React

Iako React radni okvir podržava više načina upravljanja API zahtjevima, ova aplikacija koristi „fetch, then“. Funkcija „fetch“ ugrađena je u JavaScript i omogućava aplikaciji asinkrono dohvaćanje podataka, što znači da aplikacija i dalje neprestano radi jer se podatci dohvaćaju u pozadinskom procesu. Kao argument, „fetch“ funkcija prima URL s dodatnim parametrima i specifikacijom HTTP zahtjeva, a vraća objekt „promise“ koji se može koristiti za dohvaćanje podataka. Kada se razriješi, on vraća „response“ objekt koji sadrži dohvaćene podatke koji su u formatu JSON. Blok „then“ izvršava se kada se podatci dohvate. Blok „catch“ izvršava se ukoliko pri dohvaćanju dođe do pogreške. Primjer slanja API zahtjeva prikazan je na slici 5.5, a detaljan opis funkcionalnosti prateći kod opisan je u sekciji 4.2.1. (Klijentski dio React aplikacije).

```
fetch( input: `http://localhost:8080/notes/${noteToUpdate.id}`, init: {
    method: "PUT",
    headers: headers,
    body: JSON.stringify( value: { title: noteToUpdate.title, content: noteToUpdate.content })
}) Promise<Response>
    .then(response : Response  => {
        if (response.status === 200) {
            return response.json();
        } else if (response.status === 401) {
            throw new Error('Unauthorized');
        } else {
            throw new Error('Network response was not ok');
        }
    }) Promise<any>
    .then(data => {
        setNotes(notes.map(note : Note  => (note.id === data.id ? { ...note, title: data.title, content: data.content } : note)));
        setMessage ( value: `Note ${data.title} successfully updated`);

    }) Promise<void>
    .catch(error => {
        setError(error.message);
        setMessage( value: 'Failed to update the note');
    });
```

Slika 5.5 Slanje PUT API zahtjeva

5.2.2. API zahtjevi HTMX

HTMX radni okvir ne podržava klasične metode upravljanja API zahtjevima „fetch“ funkcijama. HTMX pojednostavljuje rad s AJAX-om i koristi svoje ugrađene atribute poput „hx-get“, „hx-post“, „hx-put“ ili „hx-delete“ za slanje API zahtjeva, a ostale dodatne atribute za njihovo dodatno definiranje i upravljanje odgovorima. HTMX također omogućava slanje i dohvaćanje zahtjeva u pozadinskom procesu.

Kod za slanje API zahtjeva („put“) za ažuriranje bilješke prikazan je na slici 5.6. Prilikom slanja zahtjeva u aplikaciji koristi se Thymeleaf atribut „th:attr“. Taj atribut nam omogućuje dinamičko postavljanje putanje za zahtjev, to jest dinamičko postavljanje vrijednosti „id“ -a bilješke (${note.id}). Kada bi se „hx-put“ koristio bez atributa „th:attr“, putanja ne bi mogla biti dinamička, te bi se „id“ bilješke trebao ugraditi u kod („hardcode“-irati), što nije prikladno za izradu responzivne aplikacije. Za upravljanje dohvaćenim podatcima i eventualnim greškama koristi se HTMX atribut „hx-on“ sa dodatkom „htmx:afterRequest“. To omogućava izvršavanje JavaScript koda koji je napisan i prilagođen ovisno o tome što se vraća na zahtjev. Detaljan opis koda pri slanju API zahtjeva prikazan je u sekciji 4.2.2. (Klijentski dio HTMX aplikacije).

```
<form th:attr="hx-put=@{'/notes/' + ${note.id}}" hx-target="#operations"
      hx-include="[name='title'], [name='content']" hx-swap="outerHTML"
      class="content-box">
```

Slika 5.6 Isječak koda za slanje PUT zahtjeva u HTMX-u

5.3. Renderiranje

U ovoj sekciji uspoređuje se renderiranje komponenti u React aplikaciji, odnosno fragmenata u HTMX aplikaciji.

5.3.1. Renderiranje u React aplikaciji

Komponente React aplikacije renderiraju se, odnosno mijenjaju i ažuriraju na klijentskoj strani aplikacije. Svaki put kada se stanje u komponentama promijeni, React automatski renderira

(ažurira) te komponente, te se mijenja stvarni DOM. Na slici 5.7 prikazan je isječak iz koda u kojemu su prikazane funkcija za kreiranje bilješki i funkcija za postavljanje poruke o uspješnosti kreiranja bilješke. Te funkcije mijenjaju stanja u komponenti, čime se potiče ponovno renderiranje komponente.

```
setNotes( value: [...notes, data]);

setMessage ( value: `Note ${data.title} successfully created`);
```

Slika 5.7 Funkcije za promjenu stanja

Što se tiče poslužiteljske strane aplikacije, prilikom slanja HTTP zahtjeva, ona vraća statusni kod odgovora i objekt novokreirane bilješke, što je prikazano na slici 5.8. U ovom slučaju poslužiteljska strana nema izravnu vezu s renderiranjem komponenti u aplikaciji.

```
return ResponseEntity.created(URI.create("/notes/" + createdNote.getId()))
        .body(createdNote);
```

Slika 5.8 „return" funkcija prilikom kreiranja bilješke na poslužiteljskoj strani

5.3.2. Renderiranje u HTMX aplikaciji

Fragmenti HTMX aplikacije renderiraju se na poslužiteljskoj strani aplikacije. Nakon što se HTTP zahtjev pošalje s klijentske strane, na poslužiteljskoj strani obavljaju se operacije nad listom bilješki ovisno o operaciji (npr. kreiranje), te se onda u „model" postavljaju sve bilješke iz liste. „model" je objekt koji služi kao posrednik između poslužiteljske i klijentske strane pomoću Spring MVC-a i omogućuje klijentskoj strani prikaz fragmenata. Pomoću „model"-a dinamički se slaže fragment (HTML element), koji se zatim renderira, a nakon toga kontroler vraća naziv fragmenta „fragments/operations" za prikaz na klijentskoj strani. Ta funkcionalnost prikazana na slici 5.9.

```
@PostMapping("/notes")  Edi *
public String createNote(@ModelAttribute NoteDto noteDto, Model model) {
    noteService.createNote(noteDto);
    model.addAttribute( attributeName: "notes", noteService.getAllNotes());
    return "fragments/operations";
}
```

Slika 5.9 Funkcionalnost kontrolera za kreiranje bilješki

5. 4. Performanse i potrošnja resursa

5.4.1. Veličina paketa

S obzirom da je React jedan od najrazvijenijih klijentskih radnih okvira, posjeduje veliki ekosustav i prirodu bogatu značajkama koje rezultiraju velikom veličinom paketa (eng. bundle size). Čak i jednostavne jednostranične aplikacije razvijene u React-u sadrže mnogo TypeScript (ili JavaScript) koda, što može povećati početnu veličinu paketa. To u početku rezultira većom potrošnjom resursa i duljim vremenom renderiranja na klijentskoj strani. Međutim, zbog korištenja učinkovitih algoritama, ponovne iskoristivosti komponenti i virtualnog DOM-a, React pruža brzo ažuriranje sadržaja stranice. Navedena karakteristike čine React prikladnim za razvoj velikih aplikacija koje zahtijevaju često ažuriranje sadržaja.

S druge strane, HTMX je dizajniran s laganom prirodom bez ovisnosti i prikladniji je za razvoj aplikacija s minimalnim interakcijama, što rezultira malom veličinom paketa. Za korištenje HTMX-a nije potrebno instalirati i postavljati dodatne pakete ili knjižnice, nego se uključuje izravno u HTML putem „<script>“ značajke. To omogućava HTMX aplikacijama brzo početno učitavanje stranice i smanjenu obradu podataka na klijentskoj strani. HTMX podržava predmemoriranje odgovora, što može poboljšati performanse za ponovljene zahtjeve prema istom resursu smanjujući opterećenje poslužiteljske strane i latenciju.

5.4.2. Performanse

S obzirom na to da React koristi virtualni DOM, koji je opisan u sekciji 5.1.1. (Arhitektura React-a), uporaba resursa (memorije) povećava se zbog postojanja te značajke, ali se značajno

smanjuje potreba za stalnim učitavanjem i ažuriranjem, te se time poboljšavaju performanse. Kod HTMX-a nema uporabe resursa koji se koriste za implementiranje dodatnih značajki, već HTMX omogućava izravno upravljanje DOM-om preko HTMX atributa. Ovo je brži način učitavanja stranice s korištenjem manje resursa. HTMX pokazuje svoju prednost u performansama kada su bitni jednostavnost, upravljivost i interaktivnost.

5.4.3. Resursi

Oba radna okvira smanjuju potrošnju resursa na način da učitavaju i ažuriraju sadržaj samo kada je to potrebno. React koristi tehniku „lazy-loading“, što znači da učitava komponente, odnosno podatke (npr. slike) samo ako je to neophodno. Ova tehnika pomaže u brzom učitavanju stranice i prikazivanju samo određenog sadržaja koji je potreban za interakciju. Na primjer, slika se učitava tek kada postane vidljiva korisniku (npr. skrolanjem). Također, koristi i tehniku „code-splitting“, koja omogućava podjelu koda aplikacije i stvaranje paketa koji dinamički učitavaju po potrebi tijekom izvođenja. Tehnika pridonosi tome da kod bude učinkovit jer paket sadrži sve potrebne uvoze i datoteke. Ove tehnike pomažu smanjenju vremena učitavanja stranice i poboljšava optimizaciju resursa.

Htmx također koristi tehniku „lazy-loading“ i već spomenutu tehniku predmemoriranja, koje smanjuju vrijeme učitavanja, izvršavanja i optimiziraju resurse. Prednosti HTMX su u scenarijima kada postoje ograničeni resursi ili je važna brzina početnog učitavanja.

5.4.4. SEO

SEO odnosi se na optimizaciju web-stranica (aplikacija) u web preglednicima, osnovni zadatak optimizacije je povećanje količine i kvalitete posjećenosti web-stranice (aplikacije), koristeći neplaćena sredstva. Web-stranice (aplikacije) trebaju biti prepoznatljive pomoću njihovih imena i ključnih riječi koje su vezane za svrhu aplikacije. Postoje dvije vrste SEO optimizacije web-aplikacija: optimizacija na web-aplikaciji i optimizacija na drugim web-aplikacijama ili servisima. U kontekstu ove aplikacije važnija je ona na web-aplikaciji jer se ona odnosi na optimizaciju sadržaja na stranici i izvornog HTML koda.

React aplikacije zahtijevaju detaljnu analizu kako bi se osigurao dobar SEO i pristupačnost. Dodavanje renderiranja na poslužitelju pomoću radnog okvira kao što je Next.js može poboljšati SEO, ali to povećava složenost aplikacije.

S obzirom na to da HTMX renderira svoj sadržaj na poslužiteljskoj strani, web preglednici lakše detektiraju takav sadržaj pri pretraživanju, a to osigurava aplikaciji dobar SEO i pristupačnost.

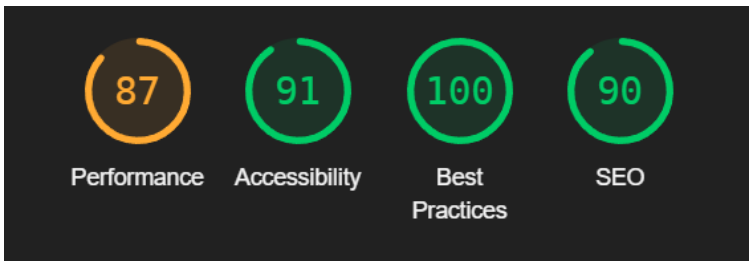
5. 5. Testiranje

Testiranje obje aplikacije pojedinačno je provedeno alatima Lighthouse i Apache JMeter. U ovoj sekciji bit će opisan način testiranja i predstavljeni rezultati.

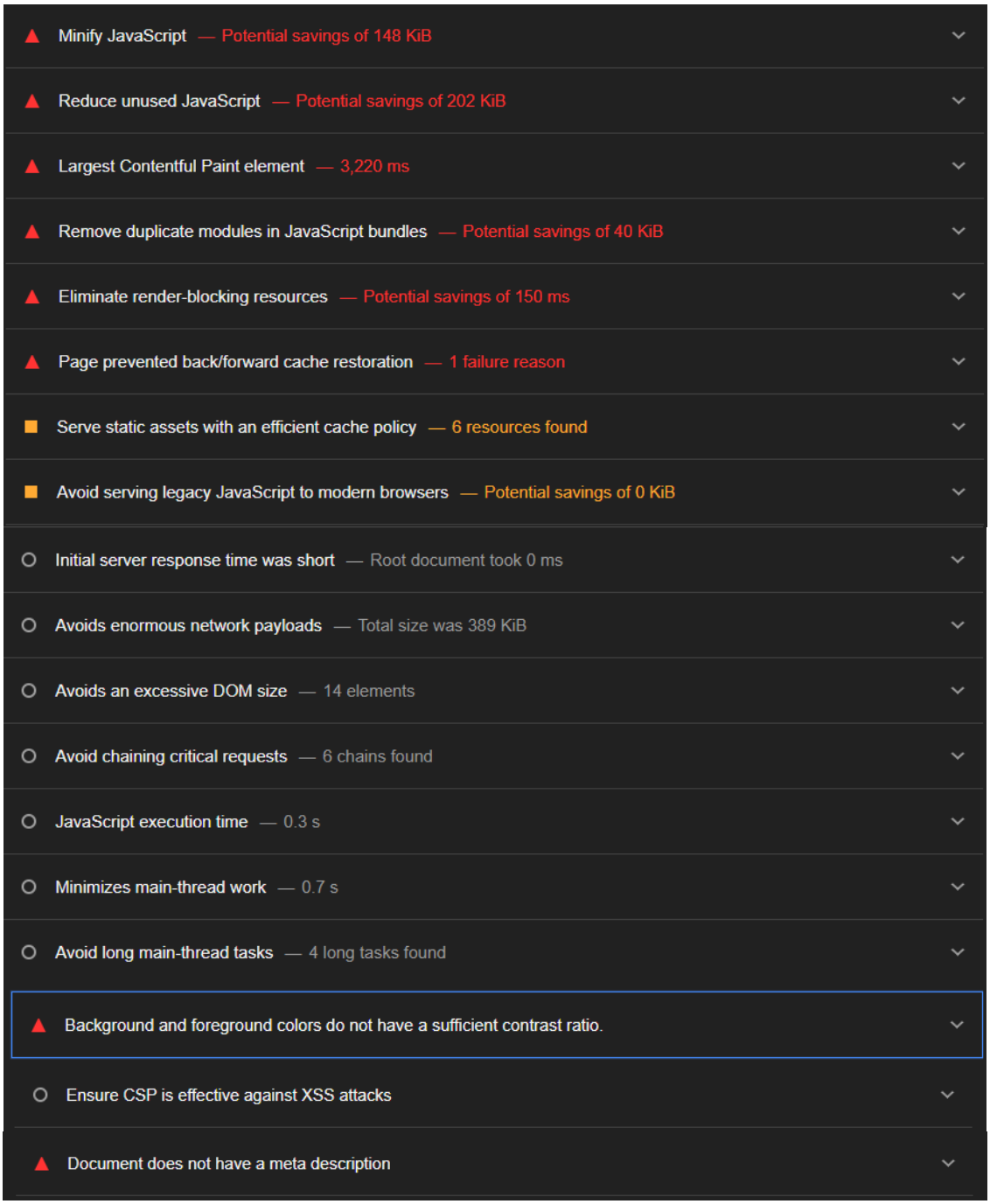
5.5.1. Testiranje alatom Lighthouse

Alatom Lighthouse testirane su performanse, pristupačnost, najbolje prakse, SEO i progresivne web-aplikacije. Mjerenje performansi uključuje vrijeme učitavanja stranice, prikazivanje teksta, vrijeme učitavanja resursa, itd. Testiranje pristupačnosti uključuje strukturu HTML elementa, opise slika, kontrast boja. Najbolje prakse uključuju sigurnosne provjere, prikazivanje resursa sigurnih izvora i upotrebu HTTPS-a. SEO-om se testira struktura URL-ova, valjanost meta tagova, ispravnost korištenja atributa za linkove i optimizaciju za mobilne uređaje. Progresivne web-aplikacije provjeravaju prisutnost HTTPS-a i provjeru PWA (Progressive Web App) funkcionalnosti.

Klasičnim načinom testiranja u Lighthouse-u testirano je svih pet značajki, a nakon izvršenog testiranja, Lighthouse objašnjava dobivene rezultate i preporučuje načine poboljšanja značajki. Na slici 5.10 prikazani su rezultati klasičnog testiranja za React aplikaciju, dok su na slici 5.11 prikazani detalji o rezultatima za značajke. Lošiji rezultati u kategoriji performanse dobiveni su zbog robusnije veličine paketa React aplikacije, te gotovo svi detalji o rezultatima ukazuju na probleme oko toga.

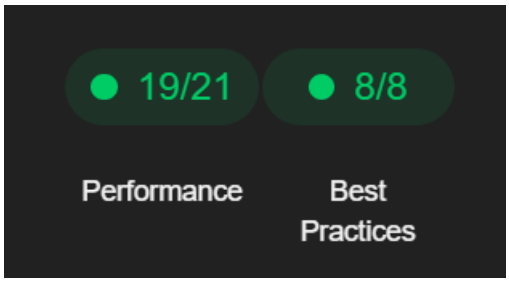


Slika 5.10 Rezultati React aplikacije klasičnim testiranjem (Lighthouse)



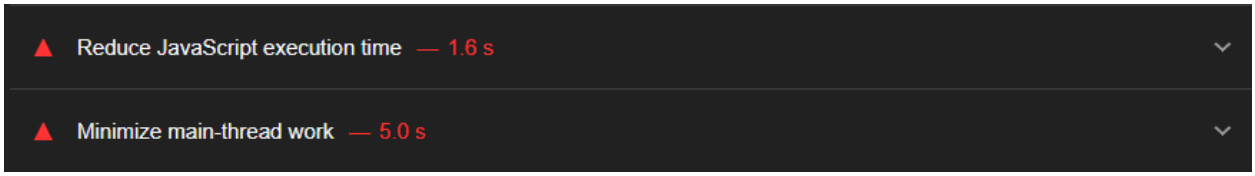
Slika 5.11 Detalji o dobivenim rezultatima klasičnog testiranja React aplikacije (Lighthouse)

Timespan (vremenski raspon) načinom testiranja mjere se samo performanse i najbolje prakse. Mjerenje se provodi dok korisnik vrši radnje u aplikaciji. Radnje koje su poduzete tijekom ovog testiranja su dodavanje tri bilješke, ažuriranje svih triju bilješki i brisanje svih triju bilješki. Rezultati mjerenja prikazani su na slici 5.12.



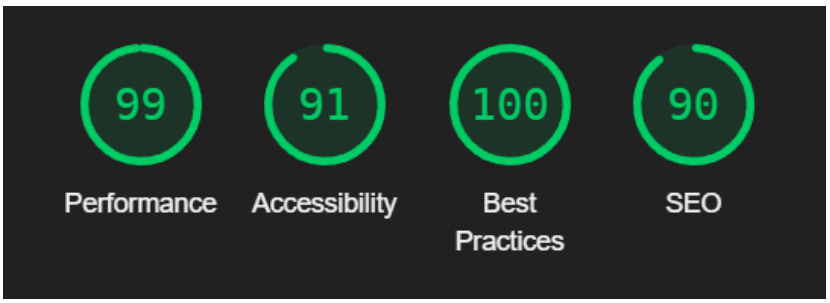
Slika 5.12 Rezultati React aplikacije timespan testiranjem (Lighthouse)

Dijagnoza nakon testiranja prikazana je na slici 5.13, ovim testiranjem dobiveni su vrlo slični rezultati kao pri klasičnom testiranju.

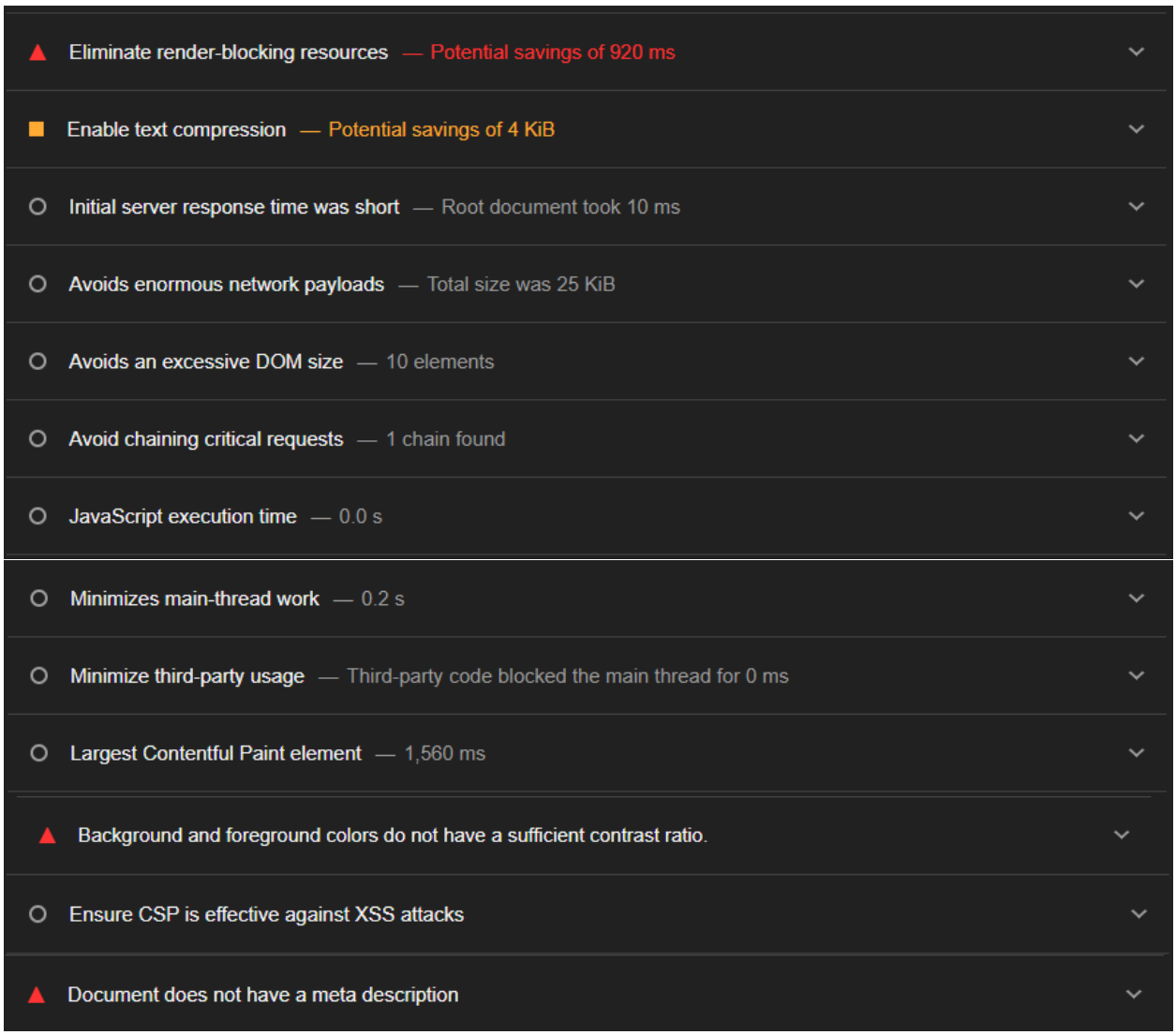


Slika 5.13 Detalji o dobivenim rezultatima timespan testiranja (Lighthouse) za React aplikaciju

Na slici 5.14 prikazani su rezultati klasičnog testiranja za HTMX aplikaciju, dok su na slici 5.15 prikazana objašnjenja dobivenih rezultata i preporuke za poboljšanje rada aplikacije. Dobiveni rezultati u značajci performanse značajno su bolji u odnosu na React aplikaciju zbog minimalne veličine paketa HTMX aplikacije. Značajka pristupačnost približno je jednaka kod obje aplikacije zbog toga što su korišteni isti HTML elementi, iste boje i oznake. Značajka najbolje prakse i SEO također su slične zbog toga što su korišteni iste putanje, odgovarajući API-i, isti naslovi i elementi.

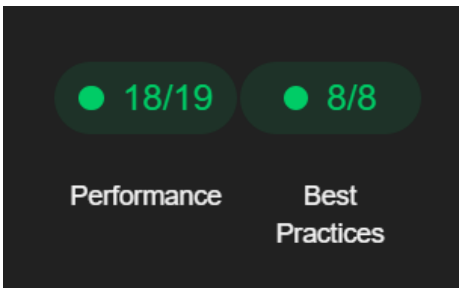


Slika 5.14 Rezultati HTMX aplikacije klasičnim testiranjem (Lighthouse)



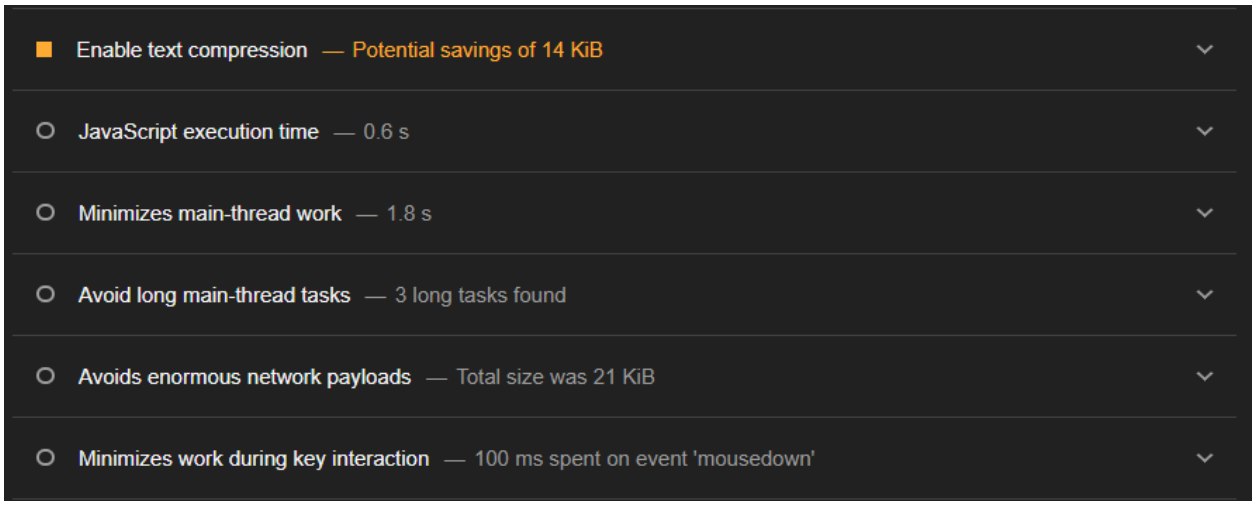
Slika 5.15 Detalji o dobivenim rezultatima klasičnog testiranja HTMX aplikacije (Lighthouse)

Pri testiranju HTMX aplikacije timespan načinom testiranja poduzete su iste radnje kao i kod testiranja React aplikacije, a rezultati tog testiranja prikazani su na slici 5.16. Također, rezultati ovog testiranja prilično su slični rezultatima klasičnog načina testiranja.



Slika 5.16 Rezultati HTMX aplikacije timespan testiranjem (Lighthouse)

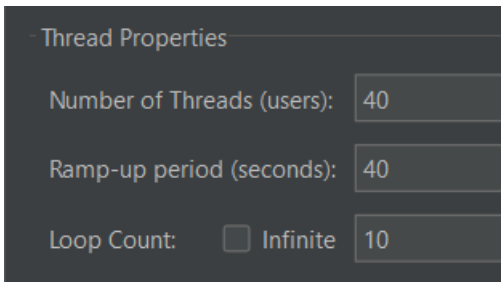
Dijagnoza timespan testiranja prikazana je na slici 5.17.



Slika 5.17 Detalji o dobivenim rezultatima timespan testiranja (Lighthouse) za HTMX aplikaciju

5.5.2. Testiranje alatom Apache JMeter

Alatom Apache JMeter testirane su aplikacije slanjem HTTP „post“ zahtjeva za kreiranje novih bilješki. U aplikaciji JMeter definira se protokol, ime poslužitelja, broj priključka, vrsta HTTP zahtjeva, putanja, kodiranje sadržaja, parametri i podatci u tijelu kako bi se zahtjevi pravilno slali. Također definira se nit (eng. thread) u kojoj se određuje broj korisnika koji šalju zahtjeve (eng. Number of Threads), vrijeme koje je potrebno da svi korisnici počnu slati zahtjeve u sekundama (eng. Ramp-Up Period) i koliko puta svaki korisnik ponavlja slanje zahtjeva (eng. Loop count). Svojstva niti za testiranje prikazana su na slici 5.18, 40 korisnika šalje zahtjeve, potrebno je 40 sekundi da se svi korisnici uključe u slanje zahtjeva, a korisnici ponavljaju slanje 10 puta.



Slika 5.18 Svojstva niti za testiranje

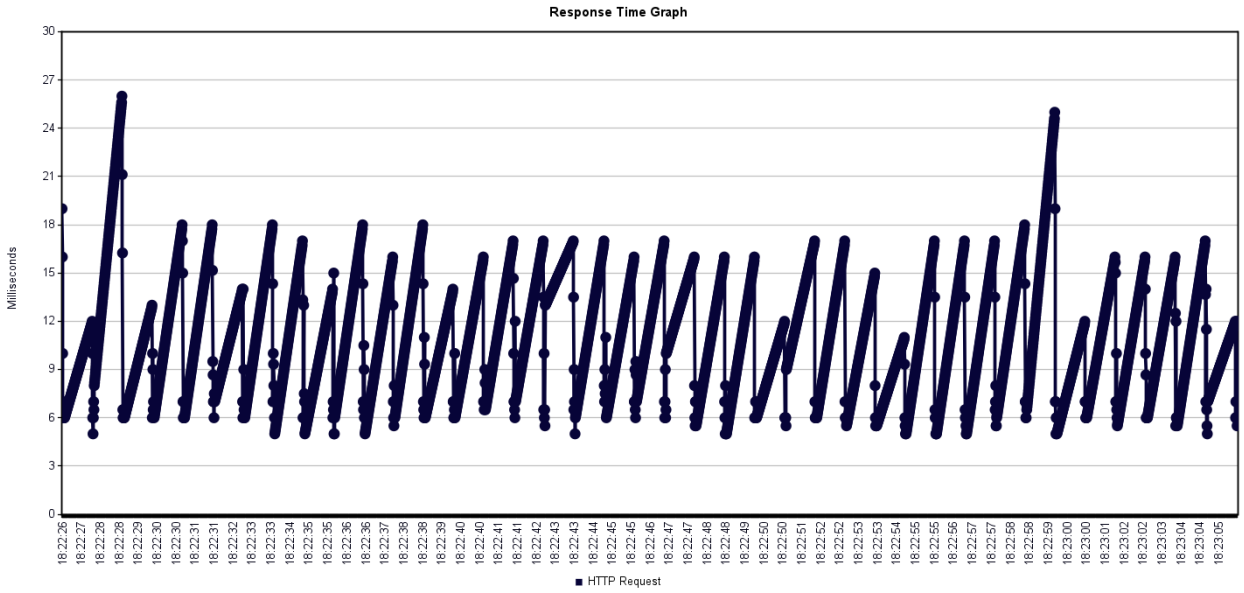
Nakon provedenog testiranja dobiveni su rezultati koji prikazuju: broj primjeraka zahtjeva (Samples), prosječno vrijeme izvršavanja jednog zahtjeva (Average), minimalno vrijeme izvršavanja jednog zahtjeva (Min), maksimalno vrijeme izvršavanja jednog zahtjeva (Max), standardna devijacija (Std.Dev.), postotak zahtjeva koji su rezultirali greškom (Error %), broj zahtjeva obrađenih u sekundi (Throughput), količina primljenih podataka u sekundi (Received KB/sec), količina poslanih podataka u sekundi (Sent KB/sec) i prosječna veličina odgovora (Avg.Bytes).

Prilikom testiranja poslano je 400 zahtjeva, a u tablici 5.2 prikazani su rezultati testiranja alatom Apache JMeter za React aplikaciju.

Label	Samples	Average	Min	Max	Std.Dev.	Error	Throughput	Received KB/sec	Sent KB/sec	Avg.Bytes
HTTP Request	400	7	5	26	3.51	0.00%	10.2/sec	3.31	2.25	330.6
TOTAL	400	7	5	26	3.51	0.00%	10.2/sec	3.31	2.25	330.6

Tablica 5.2 Rezultati testiranja React aplikacije (Apache JMeter)

Na slici 5.19 prikazan je graf vremena odaziva na zahtjeve za testiranje React aplikacije. Na osi ordinata prikazuje se vrijeme u milisekundama koje je potrebno za izvršavanje pojedinog zahtjeva, a na osi apscisa prikazano je vrijeme tijekom kojeg se izvršavaju zahtjevi. Graf prikazuje rezultate u obliku točaka koje su u intervalima od 10 milisekundi. Vrijeme odaziva je konstantno zbog toga što React aplikacija s poslužiteljske strane prima samo statusni kod odgovora i promjenu koja se dešava u bazi podataka, a ne sve podatke za prikaz.



Slika 5.19 Graf vremena odaziva za React aplikaciju

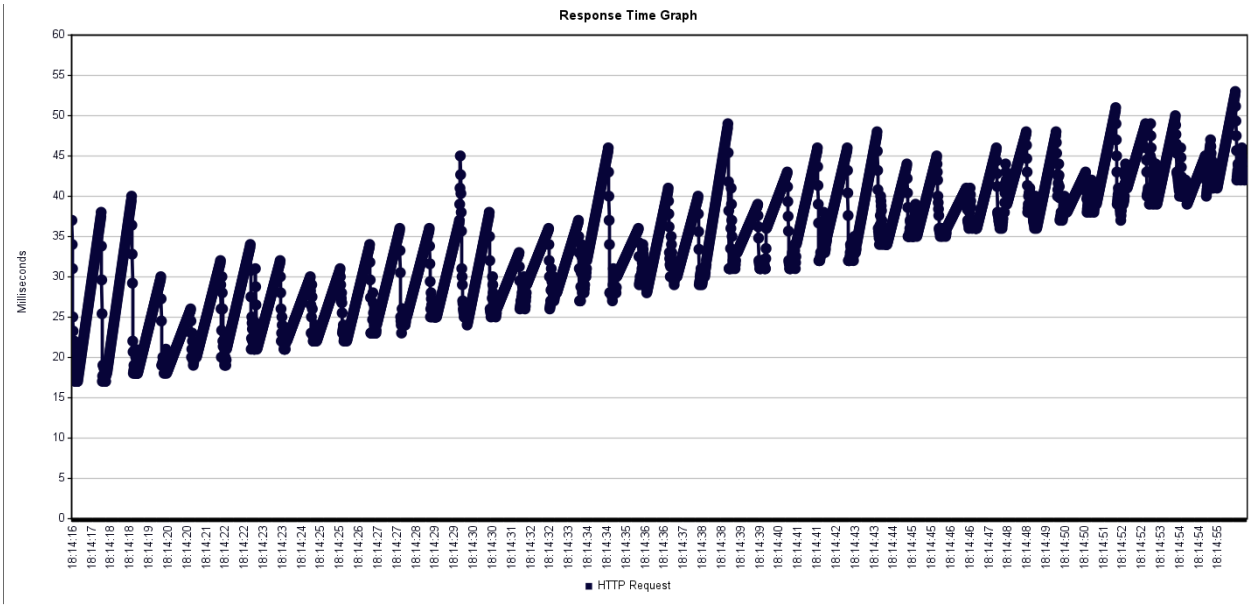
U tablici 5.3 prikazani su rezultati testiranja alatom Apache JMeter za HTMX aplikaciju.

Label	Samples	Average	Min	Max	Std.Dev.	Error	Throughput	Received KB/sec	Sent KB/sec	Avg.Bytes
HTTP Request	400	31	17	53	8.12	0.00%	10.1/sec	2696.93	2.31	272340.6
TOTAL	400	31	17	53	8.12	0.00%	10.1/sec	2696.93	2.31	272340.6

Tablica 5.3 Rezultati testiranja HTMX aplikacije (Apache JMeter)

Nakon testiranja obiju aplikacija kod HTMX aplikacije uočava se dulje vrijeme izvršavanja pojedinog zahtjeva za sve tri kategorije zbog toga što HTMX aplikaciju gubi više vremena za renderiranje podataka koje se odvija na poslužiteljskoj strani i uz to ide veća standardna devijacija. Količina grešaka koje se pojavljuju, propusnost i poslani podatci prilikom slanja zahtjeva su jednak za obije aplikacije. Količina primljenih podataka u odgovoru i prosječna veličina odgovora su mnogo veći kod HTMX aplikacije zbog toga što poslužiteljski dio HTMX aplikacije priprema fragmente koje šalje na klijentski dio, dok se kod React aplikacije vraća puno manje podataka.

Na slici 5.20 prikazan je graf vremena odaziva na zahtjeve za testiranje HTMX aplikacije.



Slika 5.20 Graf vremena odaziva za HTMX aplikaciju

Kod testiranja HTMX aplikacije primjećuje se konstantan rast vremena koje je potrebno za izvršavanje jednog zahtjeva, u odnosu na React aplikaciju kod koje je vrijeme izvršavanja zahtjeva cijelo vrijeme približno jednako. Ta pojava se događa zbog toga što HTMX aplikacija s poslužiteljske strane postepeno vraća sve veće i veće fragmente koda kako se broj bilješki u bazi podataka povećava, dok React aplikacija vraća samo nužne informacije s poslužitelja i obrađuje ih na klijentskoj strani.

5. 6. Krivulja učenja

Prije početka učenja React-a potrebno je razumjeti HTML i JavaScript (u ovom slučaju TypeScript) skriptne programske jezike. Za samu njegovu inicijalizaciju i korištenje potrebno je instalirati Node.js i npm, a zatim je potrebno stvoriti aplikaciju putem terminala. Učenje React-a donosi stepenastu sporo rastuću krivulju učenja. To uzrokuje činjenica da je React složeniji radni okvir koji zahtijeva učenje apstraktnih pojmova i koncepata poput komponenti, prijenosa podataka među komponentama, stanja, kuki, API zahtjeva i JSX-a (u ovom slučaju TSX), sintakse koja je temelj za konstrukciju React aplikacije.

S druge strane, za početak učenja HTMX-a potrebno je razumjeti samo HTML skriptni programski jezik, a poznavanje JavaScripta nije neophodno. Na njegovu inicijalizaciju nije

potrebno instalirati nikakve dodatne knjižnice ni alate, nego samo ugraditi skriptu, što znatno pojednostavljuje početno postavljanje aplikacije. S obzirom da za izradu HTMX aplikacije nije potrebno poznavanje apstraktnih pojmova niti koncepata, nego samo osnovnog koncepta izrade web-aplikacija, HTMX donosi stepenasto brzo rastuću krivulju učenja. Korisniku koji već ima iskustva u izradi web-aplikacija, učenje HTMX-a neće biti previše složeno. Učenje HTMX-a, za programere web-aplikacija svodi se na shvaćanje principa rada HTMX atributa.

5. 7. Zajednica

React, kao jedan od najpopularnijih radnih okvira, godinama je stekao veliku i bogatu zajednicu. U prilog tome ide i to što je React masovno korišten i od većine programerskih kompanija. Tijekom procesa učenja React-a, zajednica je veoma bitna zbog svih problema i poteškoća na koje se u tom procesu nailazi. Ekosustav React-a također je jedan od najvećih i najraznovrsnijih ekosustava, izuzetno koristan za pomoć pri razvoju web-aplikacija. Dobri resursi za učenje React-a osim njegove službene dokumentacije su i platforme za učenje (npr. Udemy), GitHub repozitoriji, razne web-stranice i razni tutorijali.

Kao noviji radni okvir, star tek nekoliko godina, HTMX još uvijek ima manju i nedovoljno razvijenu zajednicu. Međutim, zbog jednostavnosti učenja, korištenja i lagane integracije s radnim okvirima na poslužiteljskoj strani, manja zajednica ne predstavlja veliki problem u razvoju jednostavnih aplikacija. Najbolji resursi za učenje HTMX-a su službena dokumentacija na službenoj stranici HTMX-a i razni tutorijali. HTMX ne posjeduje veliki ekosustav, što je u skladu s njegovom orijentacijom na jednostvanost.

6. ZAKLJUČAK

Cilj ovog rada bila je temeljna analiza i usporedba React i HTMX radnih okvira u kontekstu izrade klijentskog dijela web-aplikacija. Kroz izradu ovih aplikacija, bilo je neophodno savladavanje i ostalih tehnologija koje su neophodne za razvoj web-aplikacija. Iako su razvijene dvije identične aplikacije, njihova izrada nije bila slična, osim u pogledu dizajna i funkcionalnosti. React razdvaja klijentski dio aplikacije na svoje komponente, dok HTMX razdvaja na HTML fragmente i za gotovo sve svoje funkcionalnosti koristi određene atribute.

Osnovne razlike, osim arhitekture su i to što React koristi virtualni DOM za manipulaciju stvarnim DOM-om, a HTMX manipulira DOM-om svojim HTMX atributima na način da njima definira kada će se i koji element će se mijenjati. Prilikom slanja API zahtjeva React koristi svoje ugrađene funkcije, a HTMX to odrađuje svojim atributima. Razlika u postavljanju radnog okvira za razvoj je također velika, kod React-a postavljanje okvira je nešto složenije, dok je kod HTMX-a to pojednostavljeno. Kod oba radna okvira nisu se pojavljivale poteškoće prilikom integracije s Javom i Spring Boot-om, kao ni kod povezivanja s CSS-om.

Učenje i implementacija znatno se razlikuje između dva radna okvira. Savladavanje React-a puno je složenije zbog usvajanja apstraktnijih pojmova i koncepata poput komponentne arhitekture, no uvelike ga olakšava velika zajednica te mnoštvo dostupnih materijala na internetu. S druge strane, HTMX je relativno nov i zbog toga ne objedinjuje veliku zajednicu, te ne pruža mnogo materijala na internetu, ali je zato znatno jednostavniji i intuitivniji za učenje i implementaciju što ga čini alternativom koja bi u budućnosti mogla postati veoma iznimno korišteni alat za izradu jednostavnih web-aplikacija.

Sveobuhvatnim testiranjem obje aplikacije rezultati su podjednaki, osim u pogledu performansi, gdje se na primjer testira prvi prikaz sadržaja na ekranu, i gdje prednost dobiva HTMX. To mu omogućuje njegova jednostavna arhitektura i korištenje atributa, dok je React sam po sebi puno složeniji radni okvir kojemu je potrebno puno više koda za pokretanje. Testiranjem koje se temelji na slanju velikog broja zahtjeva, veliku prednost dobiva React kojem je za obradu tih zahtjeva potrebno manje vremena i koristi znatno manje resursa.

Za razvoj ovakvog tipa aplikacije, odnosno jednostavne jednostranične aplikacije, bolji izbor je HTMX radni okvir, zbog znatno brže krivulje učenja, jednostavnije i intuitivnije implementacije, te dovoljnog skupa funkcionalnosti koje on pruža. Korištenje React-a za ovakav tip aplikacije nema previše smisla zbog njegove složenosti. Međutim, za razvoj složenijih aplikacija, koje

uključuju velik broj korisnika i korisničkih zahtjeva, bolji izbor je React. React ima pouzdanu arhitekturu, renderira podatke na klijentskoj strani, troši manje resursa i brže obrađuje velik broj podataka. Nasuprot tome, HTMX sporije obrađuje velik broj podataka i troši više resursa. Razlog tome je renderiranje podataka na poslužiteljskoj strani i sve veća količina podataka koja se povećava izvršavanjem zahtjeva, time HTMX fragmenti postaju sve veći i zahtjevniji za prijenos.

LITERATURA

[1] „Most popular front-end frameworks“, s Interneta, https://risingstars.js.org/2023/en#section-framework, 16. svibnja 2024.

[2] „htmx 1.0.0 has been released!“, s Interneta, https://htmx.org/posts/2020-11-24-htmx-1-0-0-is-released/, 17. svibnja 2024.

[3] Gackenheimer, C., „Introduction to React“, s Interneta, https://pepa.holla.cz/wp-content/uploads/2016/12/Introduction-to-React.pdf, 17. svibnja 2024.

[4] „Run JavaScript Everywhere“, s Interneta, https://nodejs.org/en, 17. svibnja 2024.

[5] „React Getting Started“, s Interneta, https://create-react-app.dev/docs/getting-started, 17. svibnja 2024.

[6] „React dokumentacija“, s Interneta, https://react.dev/learn, 17. svibnja 2024.

[7] „TypeScript for the New Programmer“, s Interneta, https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html, 17. svibnja 2024.

[8] „HTMX dokumentacija“, s Interneta, https://htmx.org/docs, 28. svibnja 2024.

[9] „Java Introduction“, s Interneta, https://www.w3schools.com/java/java_intro.asp, 21. ožujka 2024.

[10] „Spring boot konfiguracija“, s Interneta, https://www.geeksforgeeks.org/springboot-configuration/, 22. ožujka 2024.

[11] „Spring Quickstart Guide“, s Interneta, https://spring.io/quickstart, 22. ožujka 2024.

[12] „Spring Boot dokumentacija“, s Interneta, https://docs.spring.io/spring-boot/documentation.html, 22. ožujka 2024.

[13] „IntelliJ IDEA dokumentacija“, s Interneta, https://www.jetbrains.com/help/idea/discover-intellij-idea.html, 24. ožujka 2024.

[14] „CSS: Cascading Style Sheets“, s Interneta, https://developer.mozilla.org/en-US/docs/Web/CSS, 19. svibnja 2024.

[15] „PostgreSQL dokumentacija“, s Interneta, https://www.postgresql.org/about/, 2. travnja 2024.

[16] „Docker dokumentacija“, s Interneta, https://docs.docker.com/get-started/docker-overview/, 2. travnja 2024.

[17] „Lighthouse dokumentacija“, s Interneta, https://developer.chrome.com/docs/lighthouse/overview, 27. lipnja 2024.

[18] „ApacheJMeter dokumentacija“, s Interneta, https://jmeter.apache.org/index.html, 29. lipnja 2024.

[19] Barot, S., „CTO’s Perspective on HTMX vs React“, s Interneta, https://www.linkedin.com/pulse/ctos-perspective-htmx-vs-react-replace-saurabh-barot-1bmwf/, 4. lipnja 2024.

[20] „React architecture“, s Interneta, https://www.cronj.com/blog/react-js-tutorial/topics/react-architecture/, 19. svibnja 2024.

[21] „Hypermedia-Driven Applications“, s Interneta, https://htmx.org/essays/hypermedia-driven-applications/, 29. svibnja 2024.

[22] Zanini, A., Ackerson, D., „HTMX vs React: A Complete Comparison“, s Interneta, https://semaphoreci.com/blog/htmx-react, 4. lipnja 2024.

POPIS OZNAKA I KRATICA

HTMX - HTML Extended

CRUD - Create, Read, Update, Delete

HTTP - HyperText Transfer Protocol

HTML - HyperText Markup Language

URL - Uniform Resource Locator

CSS - Cascading Style Sheets

DOM - Document Object Model

CDN - Content Delivery Network

CSRF - Cross-Site Request Forgery

ID - Identifier

JPA - Java Persistence API

MVC - Model-View-Controller

DTO - Data Transfer Object

JAR - Java ARchive

API - Application Programming Interface

SQL - Structured Query Language

ACID - Atomicity, Consistency, Isolation, Durability

JDBC - Java Database Connectivity

SEO - Search Engine Optimization

JSON - JavaScript Object Notation

AJAX - Asynchronous JavaScript and XML

HDA - Hypermedia Driven Application

PWA - Progressive Web App

NPM - Node Package Manager

JSX - JavaScript XML

TSX - TypeScript XML

SAŽETAK

Cilj rada bio je usporedba React-a kao jednog od najkorištenijih i najpopularnijih radnih okvira i HTMX-a kao novog i inovativnog radnog okvira za razvoj klijentske strane web-aplikacija. Pomoću navedenih radnih okvira i ostalih tehnologija za razvoj web-aplikacija, razvijene su dvije identične aplikacije za upravljanje bilješkama. Nakon temeljne analize aplikacija, provedena je usporedba na temelju arhitekture, slanja API zahtjeva, performansi, potrošnje resursa, renderiranja, zajednice i krivulje učenja. Rezultati usporedbe pokazali su da je HTMX jednostavniji i intuitivniji za učenje, lakši za postavljanje i implementaciju, i pogodniji za razvoj jednostavnijih aplikacija. S druge strane, pri upravljanju velikim brojem zahtjeva, opadaju mu performanse i troši više resursa. React je složeniji za učenje i implementaciju, te je pogodniji za razvoj većih aplikacija zbog toga što mu se performanse ne snižavaju i potrošnja resursa ne povećava, povećanim protokom zahtjeva.

Ključne riječi: React, HTMX, Spring Boot, radni okvir, usporedba, analiza, testiranje, performanse, web-aplikacija.

ABSTRACT

The aim of this work was to compare React, one of the most widely used and popular frameworks, with HTMX, a new and innovative framework for client-side web-application development. Using these frameworks along with other web development technologies, two identical note management applications were developed. After a thorough analysis of the applications, a comparison was made based on architecture, API request handling, performance, resource consumption, rendering, community support, and learning curve. The comparison results showed that HTMX is simpler and more intuitive to learn, easier to set up and implement, and more suitable for developing simpler applications. On the other hand, its performance and resource consumption degrade when handling a large number of requests. React is more complex to learn and implement but is better suited for developing larger applications as its performance does not degrade and resource consumption does not increase with higher request volumes.

Keywords: React, HTMX, Spring Boot, framework, comparison, analysis, testing, performance, web-application.