Usporedba Next.js i React.js radnih okvira za razvoj web aplikacija

Krišković, Petra

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

Usporedba Next.js i React.js radnih okvira za razvoj web aplikacija

Rijeka, rujan 2023.

Petra Krišković

0069089035

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

Usporedba Next.js i React.js radnih okvira za razvoj web aplikacija

Mentor: doc. dr. sc. Marko Gulić

Rijeka, rujan 2023.

Petra Krišković

0069089035

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 8. ožujka 2023.

Zavod: **Zavod za računarstvo**
Predmet: **Razvoj web aplikacija**
Grana: **2.09.06 programsko inženjerstvo**

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Petra Krišković (0069089035)**
Studij: Sveučilišni prijediplomski studij računarstva

Zadatak: **Usporedba Next.js i React.js radnih okvira za razvoj web aplikacija / The comparison of Next.js and React.js frameworks for web application development**

Opis zadatka:

Treba napraviti detaljnu usporedbu React.js i Next.js radnih okvira klijentske strane koji se koriste za razvoj web aplikacija. Izvršit će se usporedba ta dva radna okvira pomoću dvije identične RESTful jednostranične aplikacije (Single Page Application, SPA) koje će biti izrađene pomoću zadanih radnih okvira. Aplikacije trebaju sadržavati osnovne CRUD (create, read, update, delete) operacije i autentifikaciju. Treba analizirati mogućnosti, arhitekturu, potrošnju resursa i načine razvoja web aplikacija pomoću ova dva radna okvira. Za razvoj poslužiteljskog dijela web aplikacije treba koristiti Laravel radni okvir uz proizvoljno odabran sustav za upravljanje bazama podataka. Također, za realizaciju autentifikacije s klijentskom stranom, može se koristiti proizvoljan Laravel paket za autentifikaciju.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 26. ožujka 2023.

Mentor:

Doc. dr. sc. Marko Gulić

Predsjednik povjerenstva za završni ispit:

Prof. dr. sc. Miroslav Joler

IZJAVA

Izjavljujem da sam samostalno izradila završni rad pod nazivom „Usporedba Next.js i React.js radnih okvira za razvoj web aplikacija“ iz kolegija Razvoj web aplikacija uz mentorstvo doc.dr.sc. Marka Gulića.

_____

Petra Krišković

ZAHVALA

Zahvaljujem se profesoru Marku Guliću na mentorstvu i strpljenju te svim profesorima tijekom prijediplomskog studija računarstva koji su me uveli u svijet računarstva. Također, zahvaljujem cijeloj obitelji i svim prijateljima na strpljenju i podršci tijekom studiranja.

SADRŽAJ

POPIS SLIKA

1 UVOD

Postoji mnogo različitih radnih okvira koji pružaju programerima i organizacijama fleksibilnost i mogućnost prilagodbe razvoja softvera prema svojim jedinstvenim potrebama i preferencijama. Ovaj rad će se baviti detaljnom usporedbom React.js i Next.js radnih okvira, istražujući njihove mogućnosti, arhitekturu, potrošnju resursa i načine razvoja web aplikacija.

U svrhu usporedbe napravljene su dvije identične To-do aplikacije temeljene na REST arhitekturi. RESTful aplikacije koriste HTTP zahtjeve za dohvaćanje i obrađivanje podataka. Koriste standardne HTTP metode kao što su GET za dohvaćanje podataka, POST za stvaranje novih podataka, PUT za ažuriranje postojećih podataka te DELETE za uklanjanje podataka. [1] Za upravljanje HTTP zahtjevima te komunikaciju između klijentske i poslužiteljske strane korištena je Axios JavaScript biblioteka. Aplikacije su jednostranične (eng. Single Page Application, SPA), odnosno sastoje se od samo jedne web stranice koja se dinamički ažurira umjesto da se pri svakoj interakciji s korisnikom učitava nova stranica. [2] Aplikacije sadrže osnovne CRUD operacije (*create, read, update, delete*), odnosno imaju funkcionalnosti stvaranja, čitanja, ažuriranja te brisanja. Za razmjenu podataka između servera i web klijenta koristi se JSON (*JavaScript Object Notation*) format koji se temelji na dvije glavne strukture podataka: objekti i nizovi. [3]

Za razvoj klijentske strane korišteni su React.js i Next.js JavaScript radni okviri. React.js je open-source JavaScript biblioteka koja se koristi za izgradnju korisničkih sučelja (eng. user interface, skraćeno UI), dok je Next.js radni okvir za izradu web aplikacija temeljen na Reactu, ali pruža dodatnu strukturu, značajke i optimizacije.

Za razvoj poslužiteljskog dijela web aplikacije korišten je Laravel radni okvir koji služi kao aplikacijsko programsko sučelje (eng. application programming interface, skraćeno API). Za realizaciju autentifikacije s klijentskom stranom korišten je Laravel Breeze paket za autentifikaciju. Laravel Breeze je minimalistička i jednostavna implementacija svih autentifikacijskih značajki u Laravelu, uključujući prijavu, registraciju, obnavljanje lozinke, verifikaciju e-pošte te potvrdu lozinke. [4] Pogledi (eng. views) Laravel Breezea stilizirani su uz pomoć Tailwind CSS-a, dok je ostatak aplikacije stiliziran običnim CSS-om. Korištena je MySQL baza podataka te TablePlus alat za upravljanje bazama podataka.

2 KORIŠTENE TEHNOLOGIJE

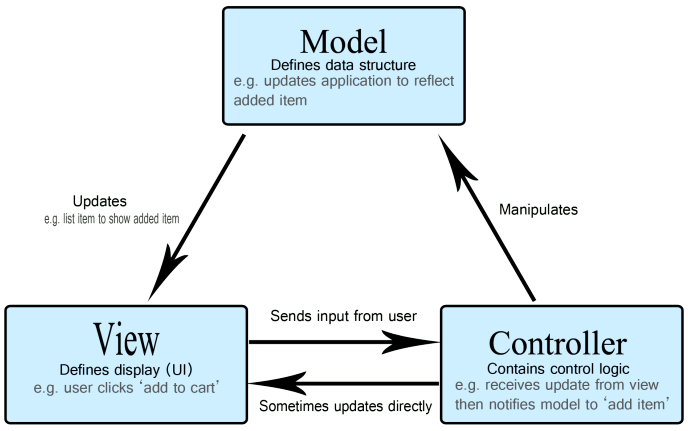
2.1 Laravel

Laravel je jedan od najpopularanijih PHP radnih okvira uz Symfony, Zend i CodeIgniter. Temelji se na Symfony programskom okviru, odnosno koristi Symfony komponente za rješavanje HTTP zahtjeva, upravljanje sesijama, validaciju unosa podataka i druge osnovne funkcije. Razvijen je od strane Taylor Otwell-a i prvi put objavljen 2011. godine, dok je zadnja inačica (10), koja je i korištena pri izradi ovoga rada, izdana u veljači 2023. godine. U ovome radu Laravel je korišten kao aplikacijsko programsko sučelje (API). [4]

2.1.1 Karakteristike Laravela

Neke od glavnih karakteristika Laravel radnog okvira jesu: [4]

- Temelji se na Model-Pogled-Kontroler (eng. Model-View-Controller, MVC) arhitekturi. Modeli se koriste za interakciju s bazom podataka, pogledi za prikazivanje podataka korisnicima, a kontroleri za upravljanje poslovnom logikom. Na *Slici 2.1* prikazan je način komunikacije između modela, pogleda i kontrolera. [5]



Slika 2.1 MVC arhitektura [19]

- Koristi *middleware* mehanizam za filtriranje zahtjeva klijenta prema serveru. U Laravel okviru postoji nekoliko *middleware*-a kao na primjer za autentifikaciju i CSRF zaštitu. *Middleware VerifyCsrfToken* u Laravelu služi za provjeru *Cross-Site Request Forgery* (CSRF) tokena na dolaznim HTTP zahtjevima prema aplikaciji. CSRF je sigurnosna ranjivost koja se javlja kada zlonamjerna web stranica prevari korisnikov preglednik da izvrši neželjeni zahtjev prema drugoj stranici, često uključujući radnje koje mijenjaju podatke ili izvršavaju radnje u ime korisnika bez njihovog pristanka. Svi *middleware*-i nalaze se u direktoriju *app/Http/Middleware*.
- Dolazi s ugrađenim zaštitama od najčešćih ranjivosti kao što su SQL *injection*, XSS (*Cross-Site Scripting*).
- Dolazi s moćnim ORM (*Object-Relational Mapping*) alatom po imenu *Eloquent*. Ovaj alat omogućava programerima da rade s bazama podataka koristeći objekte i modeliranje podataka umjesto SQL upita. To čini pristup i manipulaciju podatcima puno jednostavnijima i čitljivijima.
- Laravelovo Artisan sučelje naredbenog retka (eng. Command Line Interface, CLI) pomaže programerima da brže i efikasnije razvijaju aplikacije. Omogućava generiranje koda, izvršavanje migracije baze podataka, upravljanje rutama, testiranje aplikacije i mnoge druge zadatke koristeći Artisan.
- Blade sustav za predloške je uključen u sam Laravel te omogućava stvaranje novih, prilagođenih, enkapsuliranih PHP i HTML komponenti.

2.1.2 Instalacija Laravela i postavljanje projekta

Prije instalacije Laravela potrebno je instalirati PHP i *Composer* (alat za upravljanje PHP paketima i Laravelovim zavisnostima). Nakon instalacije *Composera* potrebno je otvoriti terminal te provesti naredbu: *composer global require laravel/installer* . Tom naredbom na računalu se instalira *Laravel Installer*. Sljedeći korak je stvaranje novog Laravel projekta u određenom direktoriju naredbom: *laravel new naziv_projekta*. Projekt je tada stvoren te se naredbom *php artisan serve* pokreće server, najčešće na adresi http://localhost:8000. [4]

2.1.3 Konfiguriranje Laravela

Konfiguracija Laravel aplikacije odvija se u *.env* datoteci u korijenskom (eng. root) direktoriju. [4] Unutar te datoteke također se konfigurira baza podataka te se navodi URL aplikacije (u ovom slučaju URL poslužiteljske strane http://localhost:8000) te URL klijentske strane http://localhost:3000. Za konfiguraciju baze potrebno je unijeti sljedeće podatke o bazi podataka:

- *DB_CONNECTION* - određuje koju vrstu baze podataka koristi aplikacija, u ovom slučaju to je *MySQL*
- *DB_HOST* - postavljanje adrese baze podataka
- *DB_PORT* - određuje *port* na kojemu se baza podataka osluškuje, za *MySQL* to je *port* 3306
- *DB_DATABASE* – ime baze podataka koja se koristi
- *DB_USERNAME* i *DB_PASSWORD* - korisničko ime i lozinka koji se koriste za autentifikaciju na bazi podataka.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=root
DB_PASSWORD=
```

Slika 2.2 Konfiguracija baze podataka u Laravelu

2.1.4 Migracije baze podataka

U direktoriju *database/migrations* definiraju se migracije baze podataka. Svaka migracija ima odgovarajuću klasu koja nasljeđuje *Migration* klasu. U klasi se definiraju promjene koje se rade u bazi podataka. U svakoj migraciji definiraju se dvije metode: *up()* i *down()*. Metoda *up()* sadrži SQL naredbe za dodavanje ili mijenjanje strukture baze podataka, dok naredba *down()* sadrži obratne

naredbe za vraćanje baze podataka u prethodno stanje. Migracije se ostvaruju putem Artisan naredbi napisanih u Laravelovom sučelju naredbenog retka (CLI). Neke od Artisan naredbi za migracije su:

- *php artisan migrate* – Koristi se kako bi se migracije primijenile na bazu podataka. Izvršava *up()* metode svih migracija koje nisu već izvršene.
- *php artisan migrate:rollback* – Koristi se kada je potrebno vratiti promjene. Izvršava *down()* metode svih migracija u obrnutom redoslijedu.
- *php artisan migrate:fresh* – Nakon izvršavanja ove naredbe svi podatci u bazi podataka se brišu te se zatim iznova izvršavaju sve migracije i stvaraju sve tablice koje su definirane u migracijama. [4]

Na *Slici 2.3* prikazan je primjer klase za migraciju baze podataka.

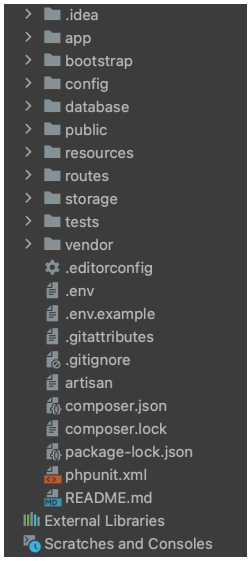
```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('flights', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('airline');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::drop('flights');
    }
};
```

Slika 2.3 Klasa za migraciju u Laravelu [4]

2.1.5 Osnovna struktura Laravel aplikacije

Osnovna struktura aplikacije sadrži direktorije: *app*, *bootstrap*, *config*, *database*, *public*, *resources*, *routes*, *storage*, *tests* te *vendor*. Prikazana je na *Slici 2.4*. [4]

1. *app* direktorij - Direktorij u kojemu se nalazi osnovni kod razvijene aplikacije. Sadrži direktorije *Console* (sadrži sve prilagođene Artisan naredbe), *Exceptions*, *Models* (modeli predstavljaju strukturu i logiku za pristup i manipulaciju podatcima u bazi podataka) te *Providers* (pružatelji usluga). Također, sadrži *Http* direktorij u kojemu se nalazi najveći dio logike aplikacije. Direktorij *Http* sadrži kontroler, *middleware* i *form request-ove (*mehanizme validacije i verifikacije zahtjeva).
2. *bootstrap* direktorij - Sadrži datoteku *app.php* koja pokreće okvir i *cache* direktorij koji sadrži datoteke generirane od strane okvira za optimizaciju performansi.
3. *config* direktorij - Sadrži sve konfiguracijske datoteke, npr. konfiguracija baze podataka, sesije, autentifikacije.
4. *database* direktorij - Ovdje se nalaze datoteke za upravljanje bazom podataka: *database/migrations* sadrži skripte za migraciju baze podataka, *database/seeds* sadrži datoteke koje pomažu popuniti bazu testnim podatcima i *database/factories* za generiranje testnih objekata.
5. *public* direktorij - Direktorij koji je dostupan izravno putem web preglednika. Sadrži javne resurse kao što su CSS datoteke, JavaScript datoteke i slike. *public/index.php* je glavna ulazna točka za sve web zahtjeve.
6. *resources* direktorij - Sadrži sve poglede (okvirne i parcijalne) te ne kompajlirane resurse kao što su CSS ili Javascript.
7. *routes* direktorij - Sve rute (URL-ovi) za aplikaciju definiraju se u ovom direktoriju. *routes/web.php* obično sadrži rute za web aplikaciju, dok *routes/api.php* sadrži rute za API.
8. *storage* direktorij - Sadrži kompajlirane Blade predloške, predmemorirane datoteke i datoteke sesije te ostale datoteke generirane od strane Laravela. Sastoji se od *app* direktorija (pohrana datoteka geniriranih od strane aplikacije), *framework* direktorija (pohrana datoteka generiranih od strane Laravela) i *logs* direktorija.
9. *tests* direktorij – Sadrži testne skripte. Laravel dolazi s ugrađenom podrškom za PHPUnit za pisanje testova.
10. *vendor* direktorij – Sadrži *Composer* zavisnosti.



Slika 2.4 Struktura Laravel aplikacije

2.1.6 Laravel Breeze

Laravel Breeze je Laravelov paket za implementaciju osnovne autentifikacije i registracije u Laravel projektu. Komponente paketa sadrže obrasce za prijavu, registraciju i ponovo postavljanje lozinke, a također dolazi i s funkcionalnostima poput potvrde e-pošte, brisanje računa te zaštite od CSRF napada i drugih sigurnosnih mjera. Koristi Blade predloške za prikaz stranica te JavaScript za interakciju s klijentskom stranom, no također podržava i tehnologije poput React.js-a, Next.js-a i Vue.js-a. Postoji i naprednija verzija *Laravel Breeza* pod nazivom *Laravel Jetstream*. Za razvijanje ovih aplikaciji korišten je *Laravel Breeze* API te je JavaScript za prikazivanje stranica napisan u Next.js i React.js radnim okvirima. [4]

2.2 React.js

React.js je JavaScript biblioteka za izradu korisničkih sučelja. Razvio ga je Jordan Walke, Facebook-ov softverski inženjer. React.js je postao temelj za izradu dinamičkih web aplikacija i korisničkih sučelja. U ovom poglavlju navedene su neke od bitnih značajki React.js-a, dok će se neke značajke dodatno objasniti u poglavlju usporedbe radnih okvira React.js i Next.js. [6]

2.2.1 Postavljanje React.js projekta

Za razvijanje React.js aplikacije potrebni su Node.js i njegov paket *npm (Node Package Manager)* koji se mogu preuzeti s Node.js web stranice. Nakon instalacije, u terminalu se izvršava naredba: *npx create-react-app naziv_projekta* za stvaranje novog React.js projekta. Projekt se pokreće naredbom *npm run dev* u terminalu. Umjesto *npm-a* može se koristiti i *Yarn. Yarn* je upravitelj za pakete za JavaScript projekte te se može instalirati putem *Yarn* web stranice. Ukoliko se koristi *Yarn* umjesto *npm-a* projekt se pokreće naredbom *yarn start.* [6] Projekt je dostupan na adresi http://localhost:8000.

2.2.2 Komponente

React.js se temelji na konceptu komponenata koje su ključne za organizaciju i strukturu aplikacije. Dakle, komponente su osnovne građevne jedinice React aplikacije. Svaka komponenta predstavlja određeni dio korisničkog sučelja ili funkcionalnosti, a to mogu biti jednostavni dijelovi poput tipke ili složenije cjeline poput obrasca ili navigacijske trake. Svaka komponenta ima svoju unutarnju logiku i uređenje. Komponente se mogu koristiti na više mjesta u aplikaciji bez potrebe za pisanjem ponovnog koda. Mogu primati podatke ili parametre putem svojih svojstava *props* što omogućava dinamičko generiranje komponenata te komunikaciju između „roditeljskih“ i „dječjih“ komponenata. Prilikom izrade ove aplikacije koncept komponenta je iskorišten za definiranje različitih *layouta* što se odnosi na organizaciju, raspored i dizajn elemenata na ekranu kako bi se stvorilo određeno korisničko sučelje. Također, koristio se za izradu komponenata poput tipke, padajućeg izbornika i tako dalje. [7]

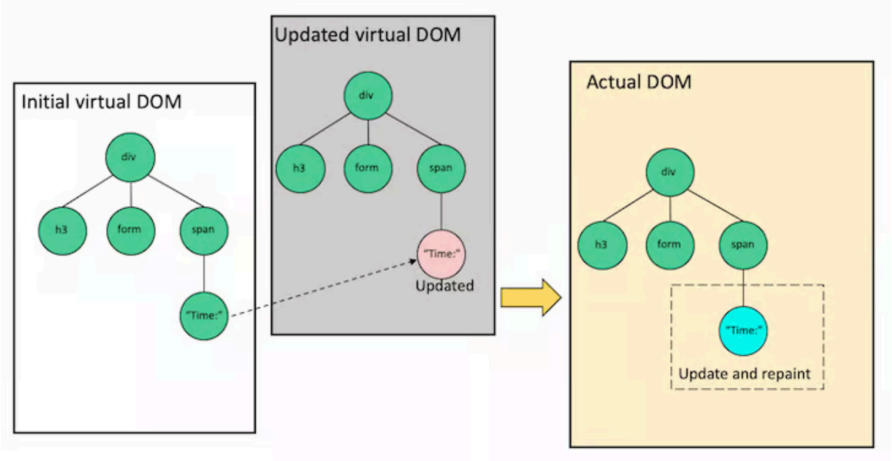
Na *Slici 2.5* prikazana je struktura komponente *Layout.* U liniji *const Layout =({children})* definiramo komponentu *Layout* koja prima jedan *prop* nazvan *children* koji sadržava sadržaj koji se prikazuje unutar *Layout* komponente. *Prop children* predstavlja sve ono što je smješteno unutar otvarajuće i zatvarajuće oznake *Layout* kada se koristi u kodu. Unutar kontejnera * * nalaze se predefinirane (gotove) komponente koje će sadržavati *Layout,* kao što su alatna traka (eng. toolbar), bočna traka (eng. sides) i pozadina (eng. backdrop). *<main>{children}</main>* predstavlja glavni sadržaj stranice, odnosno unutar *<main>* elementa prikazano je sve što je smješteno unutar oznaka *Layout.* Na kraju se izvozi komponenta *Layout* kako bi bila dostupna za korištenje u drugim dijelovima aplikacije.

```
import React from 'react';

const Layout =({children}) =>{
    return(
        <>
        <div>
            <ToolBar/>
            <Sides/>
            <Backdrop/>
        </div>
        <main>{children}</main>
        </>
    )
}

export default Layout;
```

Slika 2.5 Komponenta Layout [7]

2.2.3 Virtualni objektni model dokumenta

Virtualni objektni model dokumenta (eng. Virtual Document Object Model, VDOM) je tehnologija u Reactu koja pomaže u optimizaciji i efikasnosti ažuriranja korisničkog sučelja. Svaka web stranica ima „pravu“ DOM strukturu koja predstavlja trenutno stanje korisničkog sučelja. Pravi DOM je hijerarhija elemenata (HTML oznaka) koja se može manipulirati pomoću JavaScripta. React uvodi koncept Virtualnog DOM-a koji je apstrakcija iznad prave DOM strukture te se održava u memoriji i nije izravno povezana s prikazom na ekranu. VDOM je virtualna kopija prave DOM strukture koja se očekuje da bude brža i efikasnija za manipulaciju. Kada se dogodi promjena u stanju komponente u Reactu, React stvara novi Virtualni DOM koji reflektira te promjene. React uspoređuje novi Virtualni DOM s prethodnim Virtualnim DOM-om koji je generiran prije promjene stanja. Otkriva razlike između novog i prethodnog Virtualnog DOM-a, odnosno koje su se točno oznake ili elementi promijenili. Umjesto da mijenja cijelu pravu DOM strukturu, React primjenjuje minimalne promjene (eng. *patch*) samo na promijenjene dijelove. Ovo znači da se ne mijenjaju svi elementi na stranici, već samo oni koji su se promijenili. React potom primjenjuje te minimalne promjene na pravu DOM strukturu čime se ažurira korisničko sučelje. Prednosti VDOM-a su brže ažuriranje korisničkog sučelja, bolja izvedba i smanjeno opterećenje prave DOM manipulacije. Na *Slici 2.6* prikazano je kako VDOM funkcionira te kako utječe na pravi DOM. [8]

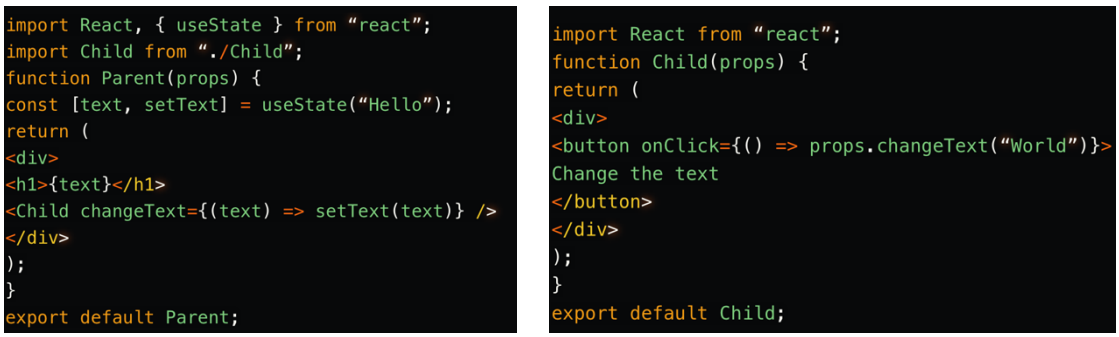


Slika 2.6 Funkcioniranje VDOM-a [8]

2.2.4 Jednosmjerno povezivanje podataka

React koristi jednosmjerno povezivanje podataka (eng. one-way binding) što se odnosi na način na koji se podatci prenose iz „roditeljske“ komponente u „dječju“ komponente. Koncept se naziva jednosmjernim jer podatci putuju iz „roditeljske“ komponente prema „djetetu“, ali ne putuju u obrnutom smjeru. „Roditeljska“ komponenta može proslijediti podatke „djeci“ kao svojstva (eng. props). „Djeca“ komponente mogu koristiti te *props-e* za prikazivanje podataka ili izvođenje operacija, ali ih ne mogu izravno mijenjati jer su samo za čitanje (eng. read-only). Ukoliko je potrebna promjena podataka, to se odvija u „roditeljskoj“ komponenti i ponovno se prosljeđuju „djetetu“. Jednosmjerno povezivanje podataka olakšava praćenje toka podataka u aplikaciji. Budući da se podatci kreću samo u jednom smjeru, lakše je pratiti kako su podatci prošli kroz različite komponente. [9]

Na *Slici 2.7* prikazan je primjer jednosmjernog povezivanja podataka između „roditeljske“ i „dječje“ komponente. U "roditeljskoj" komponenti, sadržaj unutar oznake za naslov trebao bi se promijeniti. Inicijalno je postavljen na *Hello*. Pritiskom na tipku u „dječjoj“ komponenti, pokreće se slušatelj događaja (eng. event listener) *onClick* koji poziva povratnu funkciju prenesenu iz "roditeljske" u "dječju" komponentu što mijenja tekst u *World* i ponovno pokreće komponentu.



```
import React, { useState } from "react";
import Child from "./Child";
function Parent(props) {
const [text, setText] = useState("Hello");
return (
<div>
<h1>{text}</h1>
<Child changeText={(text) => setText(text)} />
</div>
);
}
export default Parent;
```

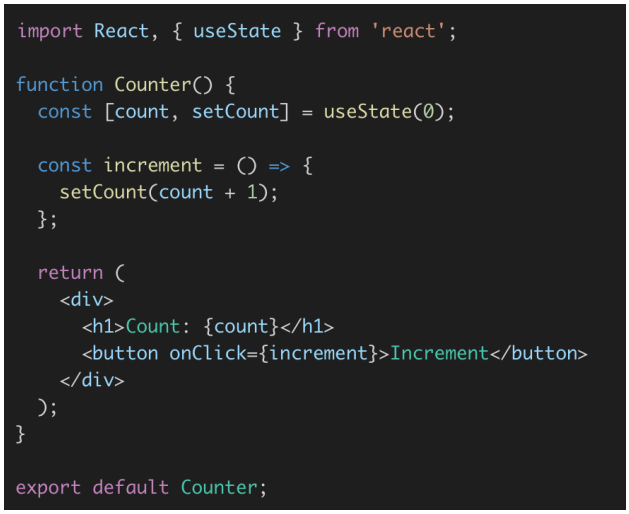
```
import React from "react";
function Child(props) {
return (
<div>
<button onClick={() => props.changeText("World")}>
Change the text
</button>
</div>
);
}
export default Child;
```

Slika 2.7 Primjer korištenja jednosmjernog povezivanja podataka [9]

2.2.5 Kuke *useState* i *useEffect*

Dvije najčešće korištene React kuke (eng. hooks) su *useState* i *useEffect* koje olakšavaju rad sa stanjima i „efektima" u funkcijskim komponentama.

Kuka *useState* omogućava komponenti da koristi lokalno stanje. Pomoću *useState* definira se varijabla koja će čuvati trenutno stanje komponente, a također pruža i funkciju za ažuriranje tog stanja. [10] Na *Slici 2.8* prikazan je jednostavni primjer korištenja *useState* kuke za funkciju brojača (*Counter()*). Na vrhu datoteke potrebno je uvesti *useState* iz paketa *react. useState* se koristi za definiranje stanja varijable *count* s početnom vrijednosti 0. *useState* funkcija vraća niz s dvije vrijednosti: vrijednost trenutnog stanja *count* i funkciju za ažuriranje stanja varijable *count* naziva *setCount.* Funkcija *increment* je jednostavna funkcija koja koristi *setCount* za ažuriranje vrijednosti *count* tako da joj nadoda 1. Funkcija *increment* aktivira se pritiskom na tipku uz pomoć slušatelja događanja *onClick*. [10]



```
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  const increment = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <h1>Count: {count}</h1>
      <button onClick={increment}>Increment</button>
    </div>
  );
}

export default Counter;
```

Slika 2.8 Primjer korištenja kuke useState [10]

Kuka *useEffect* se koristi za izvršavanje koda koji nije direktno povezan s funkcijom komponente. Koristi se za dohvaćanje podataka iz vanjskog izvora, postavljanje slušatelja događaja ili ažuriranje objektnog modela dokumenta. Pomoću njega definiramo funkciju koja se poziva nakon svakog učitavanja komponente (prvi put kada se pozove i svaki idući put kada se učita ili obnovi). Također, ima opciju konfiguriranja kada će se ta funkcija izvršiti, na primjer, samo pri prvom učitavanju ili kada se određeno stanje promjeni. *useEffect* odvaja sporedne procese od osnovne logike komponente. [10]

Na *Slici 2.9* prikazan je primjer istovremenog korištenja kuka *useState* i *useEffect* u React funkcijskoj komponenti *UsersList* za dohvaćanje i prikazivanje popisa korisnika s udaljenog API-ja. Prvotno se uvode sve potrebne biblioteke: *React*, *useState*, i *useEffect* iz *React* paketa. Funkcijska komponenta *UsersList* definira stanje *users* pomoću *useState* kuke. Inicijalno je postavljeno na prazno polje. Zatim se koristi *useEffect* kako bi se obavio sporedni proces, u ovom slučaju dohvaćanje podataka o korisnicima s udaljenog API-ja. Unutar *useEffect* funkcije, koristi se *fetch* za slanje GET zahtjeva prema URL-u https://jsonplaceholder.typicode.com/users. Kada se dobije odgovor, koristi se *.json()* metoda kako bi se odgovor pretvorio u JSON format. Potom se koristi *setUsers* za ažuriranje stanja *users* s dobivenim podatcima. Prazno polje [] je proslijeđeno kao drugi argument za *useEffect* što znači da će se *useEffect* izvršiti samo nakon prvog pokretanja komponente. [10]

```jsx
import React, { useState, useEffect } from 'react';

function UsersList() {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/users')
      .then((response) => response.json())
      .then((data) => setUsers(data));
  }, []);

  return (
    <ul>
      {users.map((user) => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
}

export default UsersList;
```

Slika 2.9 Primjer korištenja kuka useState i useEffect u komponenti [10]

2.3 Next.js

Next.js je radni okvir za web razvoj otvorenog koda koji je stvorila privatna tvrtka Vercel. Temeljen je na Reactu, ali ima dodatne značajke. Omogućuje razvoj modernih web aplikacija koje su brze, skalabilne i jednostavne za održavanje. Omogućuje *rendering* na strani poslužitelja (eng. server-side rendering, skraćeno SSR) i generiranje statičkih stranica (eng. static site generation, skraćeno SSG), poboljšavajući time performanse i optimizaciju web stranice (eng. search engine optimization, skraćeno SEO). Također, Next.js omogućuje automatsko razdvajanje koda (eng. code splitting) kako bi se ubrzalo inicijalno učitavanje stranica. Rute aplikacije mogu se jednostavno definirati i upravljati uz pomoć *next/router* modula. Next.js također podržava TypeScript što omogućuje programerima pisanje tipiziranog koda za bolju sigurnost i održavanje. [11] Sve značajke Next.js-a biti će detaljnije objašnjene u poglavlju usporedbe radnih okvira React.js i Next.js.

Instalacija i postavljanje novog Next.js-a projekta provodi se na isti način kao i za React.js osim razlike u naredbi za stvaranje novog projekta koja glasi: *npx create-next-app naziv_projekta*.

2.4 Axios

Axios je HTTP klijent za node.js i preglednike (eng. browser) temeljen na obećanjima (eng. promise-based). On je izomorfan što znači da se može izvoditi i u preglednicima i na node.js platformi koristeći isti kod. Na strani poslužitelja (eng. server-side) Axios koristi node.js *http* modul, dok na strani klijenta (na pregledniku) koristi *XMLHttpRequests*. Podržava Promise API što omogućava upotrebu asinkronih zahtjeva. Omogućava presretanje zahtjeva i odgovora što znači da omogućava da se HTTP zahtjevi (eng. request) koji se šalju prema poslužitelju i HTTP odgovori (eng. response) koji dolaze s poslužitelja dohvate i manipuliraju prije nego što dođu do krajnjeg korisnika ili aplikacije. Automatski pretvara podatke u zahtjevima i odgovorima u JSON, FormData, ili URL enkodirane (eng. URL encoded form) oblike. Omogućava otkazivanje zahtjeva i postavljanje vremenskih ograničenja (eng. timeouts). Serijski pretvara upitne parametre (eng. query parameters) u odgovarajući format i podržava ugniježđene unose. Pruža zaštitu od XSRF napada na strani klijenta. Axios se preuzima naredbom: *npm install axios* u terminalu. [12]

Primjer sa *Slike 2.10* koristi Axios za izvršavanje HTTP POST zahtjeva prema URL-u */user* s određenim podatcima. Analiza koda:

- *axios.post(/user, { firstName: Fred, lastName: Flintstone })*: Ovdje se koristi Axios za izvršavanje POST zahtjeva prema */user* URL-u. U tijelu zahtjeva šalju se podatci koji sadrže dvije stavke: *firstName* s vrijednošću *Fred* i *lastName* s vrijednošću *Flintstone*. Ovo su podatci koji će biti poslani na poslužitelj.
- *.then(function (response) { console.log(response); })*: Ova linija koda dodaje rukovatelja (eng. handler) za uspješan odgovor od poslužitelja. Kada poslužitelj uspješno obradi zahtjev i vrati odgovor, ovaj rukovatelj će biti pozvan, a odgovor poslužitelja bit će dostupan kao argument *response* te se ispisuje u konzolu.
- *.catch(function (error) { console.log(error); })*: Ova linija koda dodaje rukovatelja za slučaj da dođe do greške tijekom izvršavanja zahtjeva. Ako dođe do greške, ovaj rukovatelj će biti pozvan, a greška će biti dostupna kao argument *error* te će se greška ispisati u konzolu.

```
axios.post('/user', {
    firstName: 'Fred',
    lastName: 'Flintstone'
  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });
```

Slika 2.10 Primjer korištenja Axiosa [12]

2.5 Tailwind CSS

Tailwind CSS je brza i fleksibilna CSS biblioteka koja omogućava razvoj modernih web aplikacija. Umjesto da se ručno pišu CSS dizajni, Tailwind CSS pruža gotove klase koje mogu biti dodane HTML elementima za primjenu stilova. Ovo pojednostavljuje proces izrade dosljednih i atraktivnih korisničkih sučelja. Tailwind CSS također podržava prilagodbu i proširivost kroz konfiguracijske datoteke, omogućavajući prilagodbu dizajna aplikacije prema potrebama. Koristi utilitarni pristup, omogućujući brzo stvaranje kompleksnih dizajna kombiniranjem različitih klasa. Također, pruža optimiziranu izlaznu CSS datoteku koja smanjuje veličinu i poboljšava performanse aplikacije te obično sadrži samo stilove koji se stvarno koriste u aplikaciji uklanjajući nepotrebne ili neiskorištene stilove. [13] Na *Slici 2.11* prikazana je primjena Tailwinda u programskom kodu.

```
<div class="p-6 max-w-sm mx-auto bg-white rounded-xl shadow-lg flex items-center sp
  <div class="shrink-0">
    <img class="h-12 w-12" src="/img/logo.svg" alt="ChitChat Logo">
  </div>
  <div>
    <div class="text-xl font-medium text-black">ChitChat</div>
    <p class="text-slate-500">You have a new message!</p>
  </div>
</div>
```

Slika 2.11 Primjena Tailwinda [13]

Neke od najkorištenijih klasa Tailwinda su:

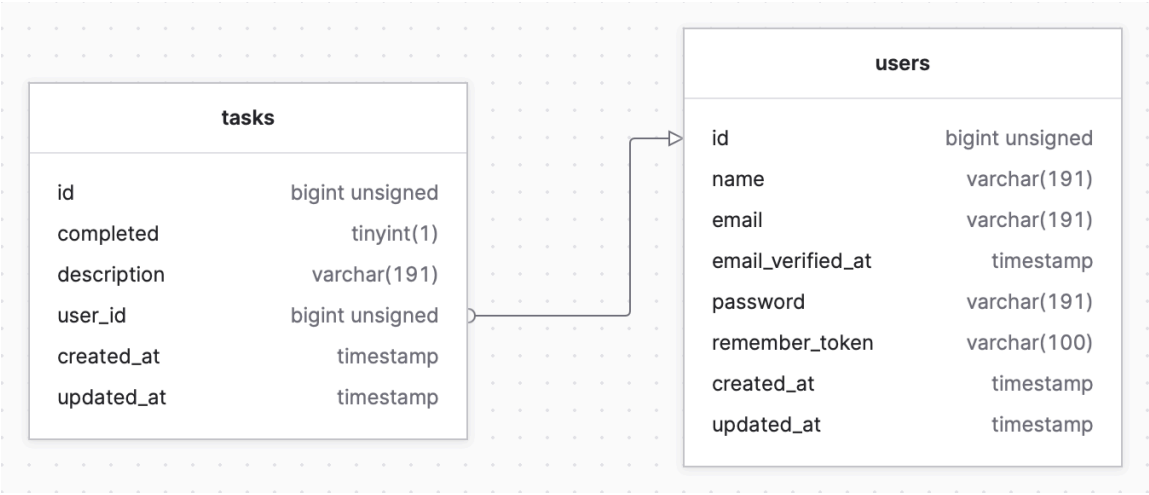
- *bg-{color}* – postavlja pozadinsku boju elementa, npr. *bg-white* postavlja pozadinsku boju u bijelu
- *text-{size}* – postavlja veličinu fonta
- *font-bold* – postavlja font na podebljani stil
- *justify-{position}* - koristi se za horizontalno poravnanje elemenata, npr. *justify-center* će horizontalno poravnati elemente na sredinu
- *text-{color}* – postavlja boju teksta, npr. *text-black* postavlja tekst crne boje…

Mijenjanje stilova koristeći Tailwind je vrlo jednostavno jer se klase stilova definiraju unutar samih komponenti te se promjenom jedne riječi mijenja stil, npr. promjena iz *bg-red* u *bg-blue* mijenja pozadinsku boju iz crvene u plavu.

2.6 MySQL

MySQL je sustav za upravljanje relacijskim bazama podataka otvorenog koda (eng. open-source). Koristi se za pohranu i organizaciju strukturiranih podataka u tablicama, a pristup podatcima odvija se putem SQL (*Structured Query Language*) upita. Putem SQL upita omogućava stvaranje, čitanje, ažuriranje i brisanje podataka u bazi podataka. [14]

Kao alat za upravljanje bazom podataka korišten je TablePlus. Unutar TablePlus alata postoji priključak (eng. plugin) *Diagram Generator* koji se koristi za stvaranje dijagrama postojećih baza podataka. Na *Slici 2.12* nalazi se ER dijagram baze podataka korištene u ovom radu koji je stvoren uz pomoć *Diagram Generator* priključka. Entiteti su *tasks* koji označava zadatke unutar aplikacije i *users* koji označava korisnike aplikacije. *tasks* sadrži atribute *id, completed, description, created_at, updated_at* te strani ključ *user_id*. *users* sadrži atribute *id, name, email, password* i tako dalje.



Slika 2.12 ER dijagram baze podataka

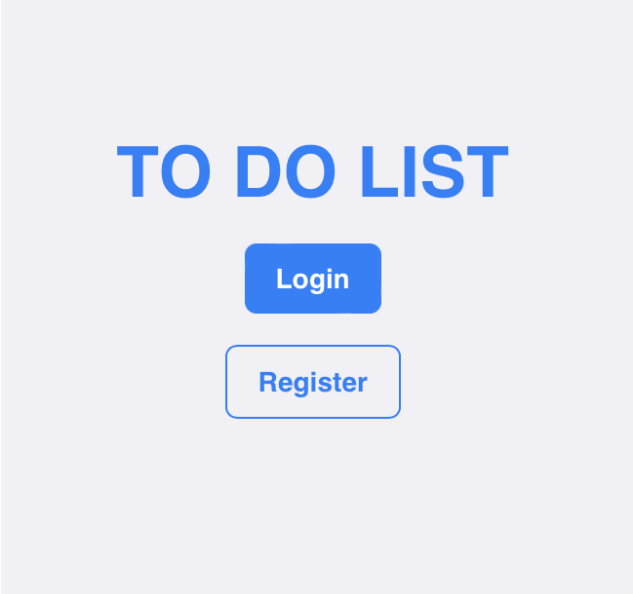
3 OPIS APLIKACIJE

Razvijena je *To do list* aplikacija koja služi za upravljanje zadacima s naprednim značajkama autentifikacije. Omogućava dodavanje, označavanje i brisanje zadataka, čineći organizaciju obaveza brzom i jednostavnom, dok autentifikacija osigurava privatnost i siguran pristup korisnikovim zadatcima. U nastavku su detaljno istražene sve funkcionalnosti aplikacije.

Pošto su aplikacije napravljene u svrhu testiranja performansi radnih okvira React.js-a i Next.js-a, dizajn obje aplikacije je identičan te će se u prikazu pogleda aplikacije sve funkcionalnosti prikazivati samo jedanput.

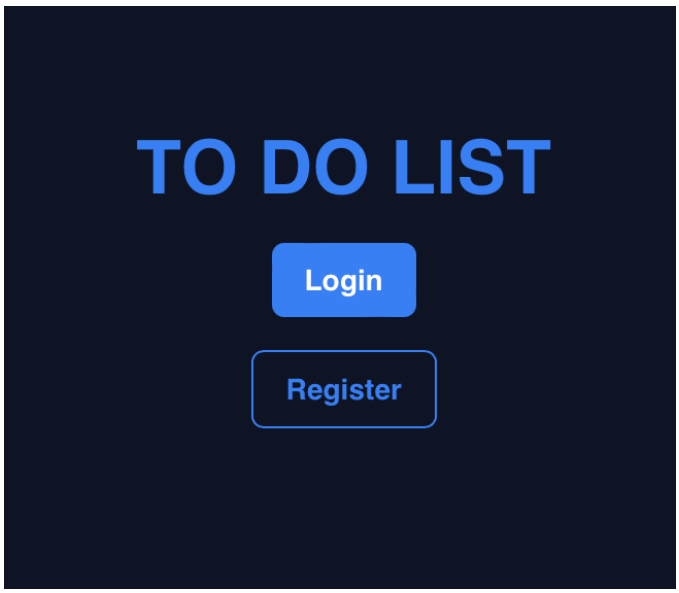
3.1 Početni pogled

Na početnom pogledu, kao što je prikazano na *Slici 3.1*, pojavljuje se naslov aplikacije *TO DO LIST* te ispod naslova dvije tipke: *Login* koja vodi na pogled za prijavu te *Register* koja vodi na pogled za registraciju.



Slika 3.1 Početni pogled

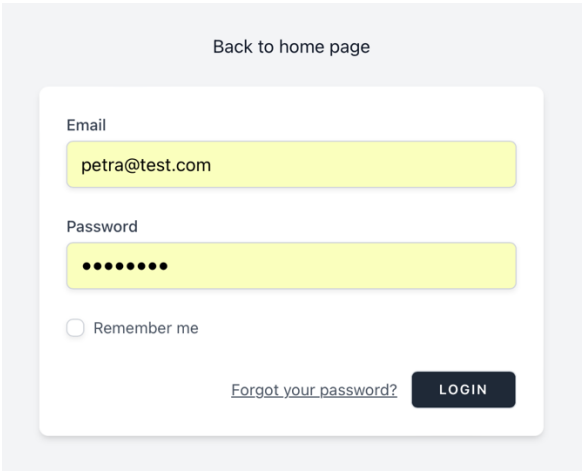
Postoji i tamni način (eng. dark mode) prikaza početnog pogleda kao što je prikazano na *Slici 3.2*.



Slika 3.2 Tamni prikaz početnog pogleda

3.2 Pogledi autentifikacije

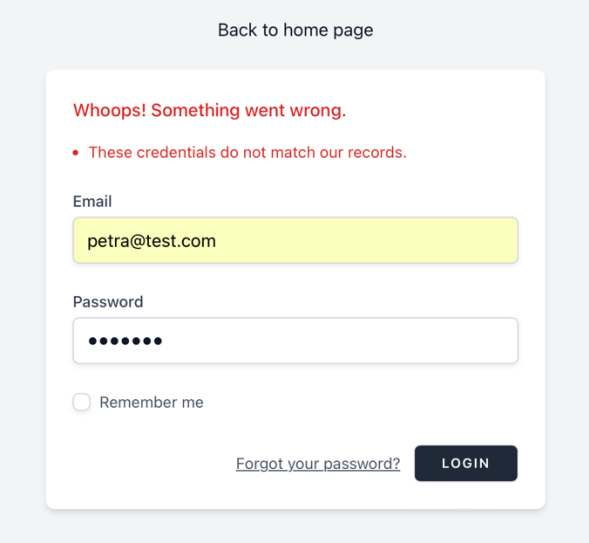
Pritiskom tipke *Login* otvara se originalni pogled za prijavu koji dolazi u Laravel Breeze paketu za autentifikaciju s prilagodbom tipke *Back to home page* koja, ukoliko ju korisnik pritisne, vraća korisnika na početni pogled. Pogled prijave prikazan je na *Slici 3.3*.



Slika 3.3 Pogled za prijavu

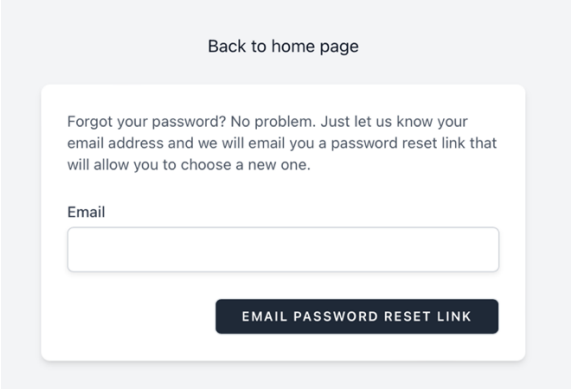
Ukoliko je korisnik prethodno registriran, nakon unošenja svoje e-pošte (*Email*) i lozinke (*Password*), pritiskom tipke *LOGIN* on se prijavljuje te se otvara njegova To do lista. Funkcionalnost *Remember me*, ukoliko je označena kvačicom na potvrdnom okviru (eng. checkbox), omogućava nam da preglednik zapamti prijavu te prilikom iduće prijave nije potrebno unositi podatke e-poštu i lozinku.

Ukoliko korisnik ne unese neki od potrebnih podataka ili unese nevažeću e-poštu ili lozinku pojavljuje mu se poruka kao što je prikazano na *Slici 3.4* te zahtjeva ispravak određenih podataka.



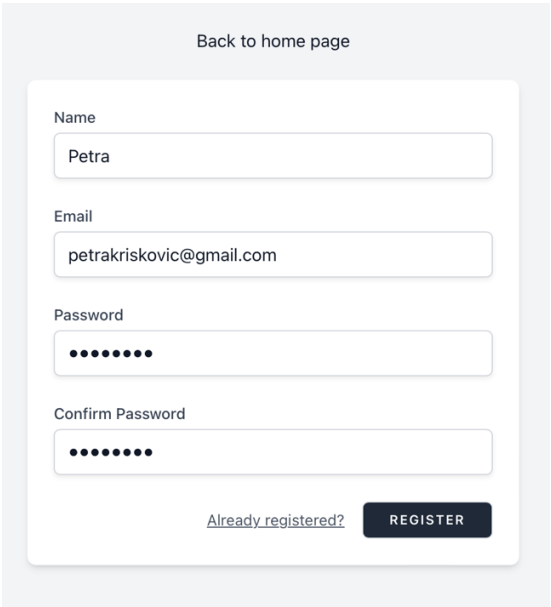
Slika 3.4 Poruka prilikom unošenja nevažećih podataka

Pritiskom na *Forgot your password*? otvara se pogled kao što je prikazano na *Slici 3.5*. Svrha te funkcionalnosti jest opcija promjene lozinke u slučaju da ju korisnik zaboravi, odnosno da se link za ponovno postavljanje lozinke pošalje na korisnikovu e-poštu. Međutim, pošto za svrhu zadatka nije bilo potrebno koristiti važeće, odnosno korištene e-pošte, ta funkcionalnost nije aktivirana. Na vrhu pogleda nalazi se tipka *Back to home page* koja, ukoliko ju korisnik pritisne, vraća korisnika na početni pogled.



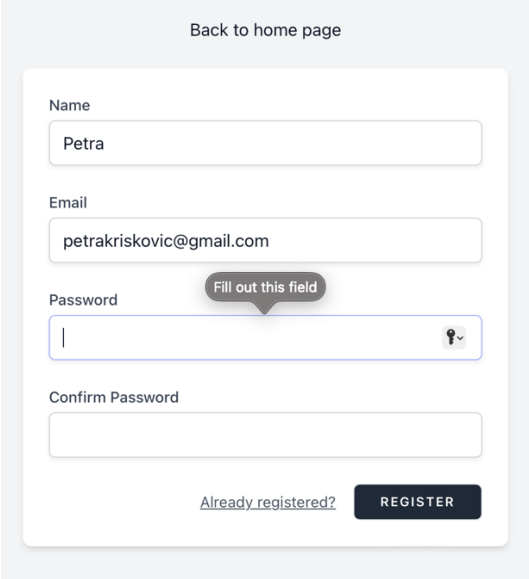
Slika 3.5 Pogled Forgot my password?

Pritiskom tipke *Register* na početnom pogledu otvara se originalni pogled za registraciju koji dolazi u Laravel Breeze paketu za autentifikaciju s prilagodbom tipke *Back to home page* koja, ukoliko ju korisnik pritisne, vraća korisnika na početni pogled. Pogled registracije prikazan je na *Slici 3.6*. Da bi se korisnik uspješno registrirao mora upisati podatke: ime (*Name*), e-poštu (*Email*), proizvoljnu lozinku (*Password*) te mora ponoviti lozinku (*Confirm Password*) radi osiguravanja ispravnosti i sigurnosti. Ukoliko su svi podatci ispravno uneseni, pritiskom tipke *REGISTER* otvara se njegov novi neispunjeni To do list. Na pogledu za registraciju također se nalazi tipka *Already registered*? koja služi ukoliko je korisnik već registriran te ako je pritisnuta odvodi korisnika na pogled za prijavu.



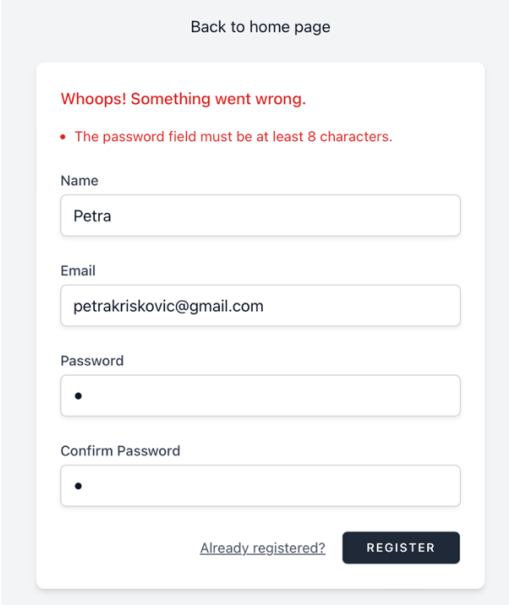
Slika 3.6 Pogled za registraciju

Na *Slici 3.7* prikazana je poruka *Fill out this field* koja se pojavljuje ukoliko korisnik nije unio neki od traženih podataka prilikom registracije te ga upozorava da ispuni označeno polje.



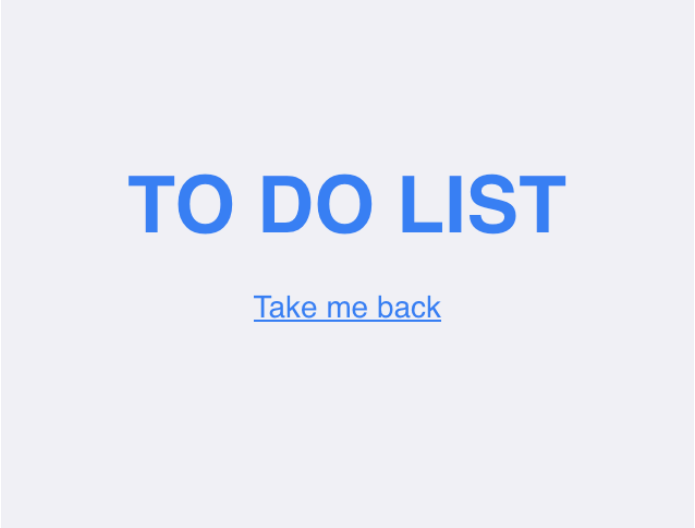
Slika 3.7 Poruka Fill out this field

Na *Slici 3.8* prikazano je upozorenje *The password field must be at least 8 characters* koje korisnika upozorava da mu je lozinka nepravilno unesena te da lozinka mora sadržavati barem osam znakova.



Slika 3.8 Poruka upozorenja o nepravilno unesenoj lozinki

Prilikom odjave korisnika korisnik se odvodi na početni pogled. Ukoliko korisnik ostane prijavljen nakon gašenja servera, prilikom ponovnog pokretanja servera otvara se pogled prikazan na *Slici 3.9* na kojemu se nalazi naslov aplikacije *TO DO LIST* te tipka *Take me back* koja, ukoliko je pritisnuta, odvodi korisnika na pogled njegove liste zadataka. Dakle, početni pogled se otvara samo ukoliko je prethodni korisnik odjavljen. Za pogled nakon neodjavljenog korisnika, kao i za početni pogled, postoji prilagođavanje za tamni način prikaza prema istom principu kao i za početni pogled.



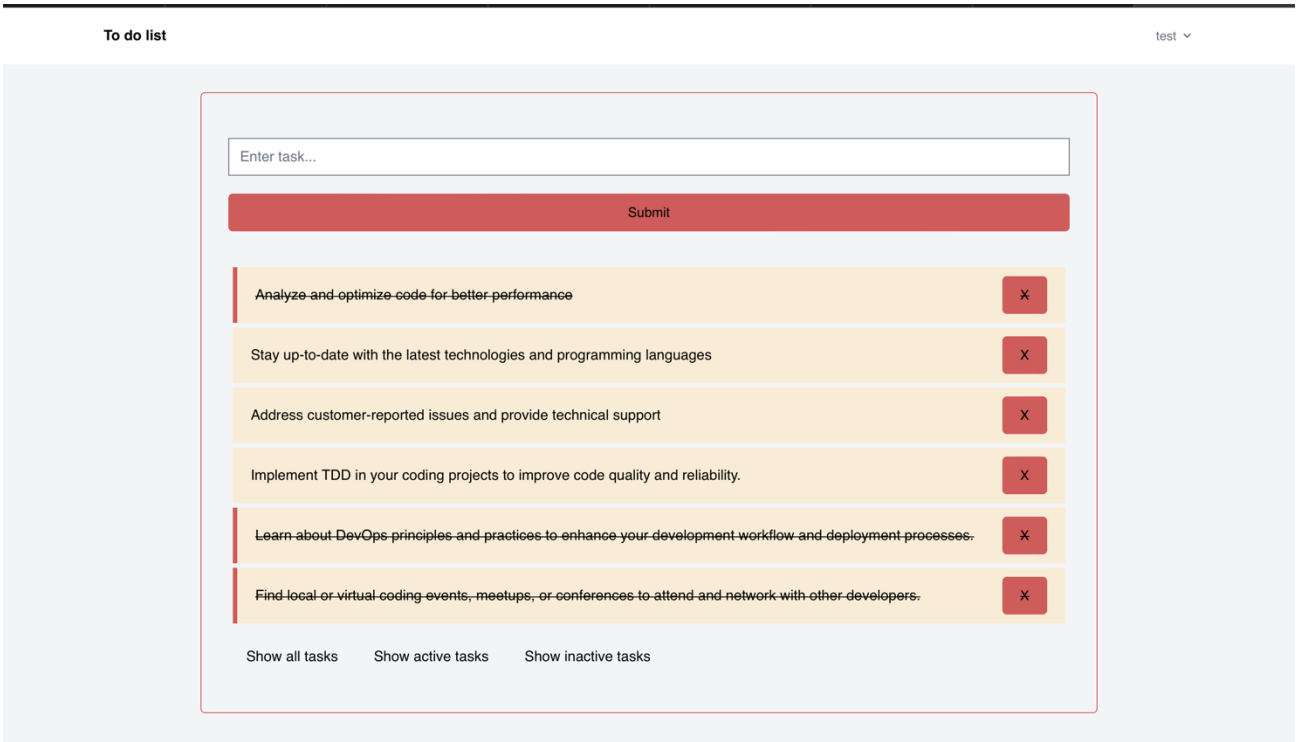
Slika 3.9 Pogled nakon neodjavljenog korisnika

3.3 Pogledi koji sadrže listu zadataka

3.3.1 Pogled liste zadataka korisnika

Na *Slici 3.9* prikazan je pogled liste zadataka korisnika. Na samom vrhu pogleda u lijevom kutu nalazi se naziv aplikacije *To do list*, a u desnom kutu ime trenutno prijavljenog korisnika, koje u ovom slučaju navodi da je trenutni prijavljeni korisnik *test*.

Ukoliko se pritisne na ime korisnika otvara se padajući izbornik s opcijom *Logout* te u slučaju pritiska te opcije korisnik se odjavi te se vraća na početni pogled.



Slika 3.10 Pogled liste zadataka korisnika

Zatim, ispod naslova i imena korisnika nalazi se okvir unutar kojega se nalazi okvir u koji se unose zadatci, tipka za dodavanje novih zadataka te lista korisnikovih zadataka.

Unutar okvira za unošenje novih zadataka nalazi se *placeholder Enter task…* te pritiskom na taj okvir korisnik može unijeti novi zadatak za svoju To do listu. Novi zadatak se dodaje i sprema pritiskom na tipku *Submit* ili pritiskom tipke *enter* na tipkovnici. Time je ispunjen kriterij stvaranja (eng. create) za CRUD aplikaciju.

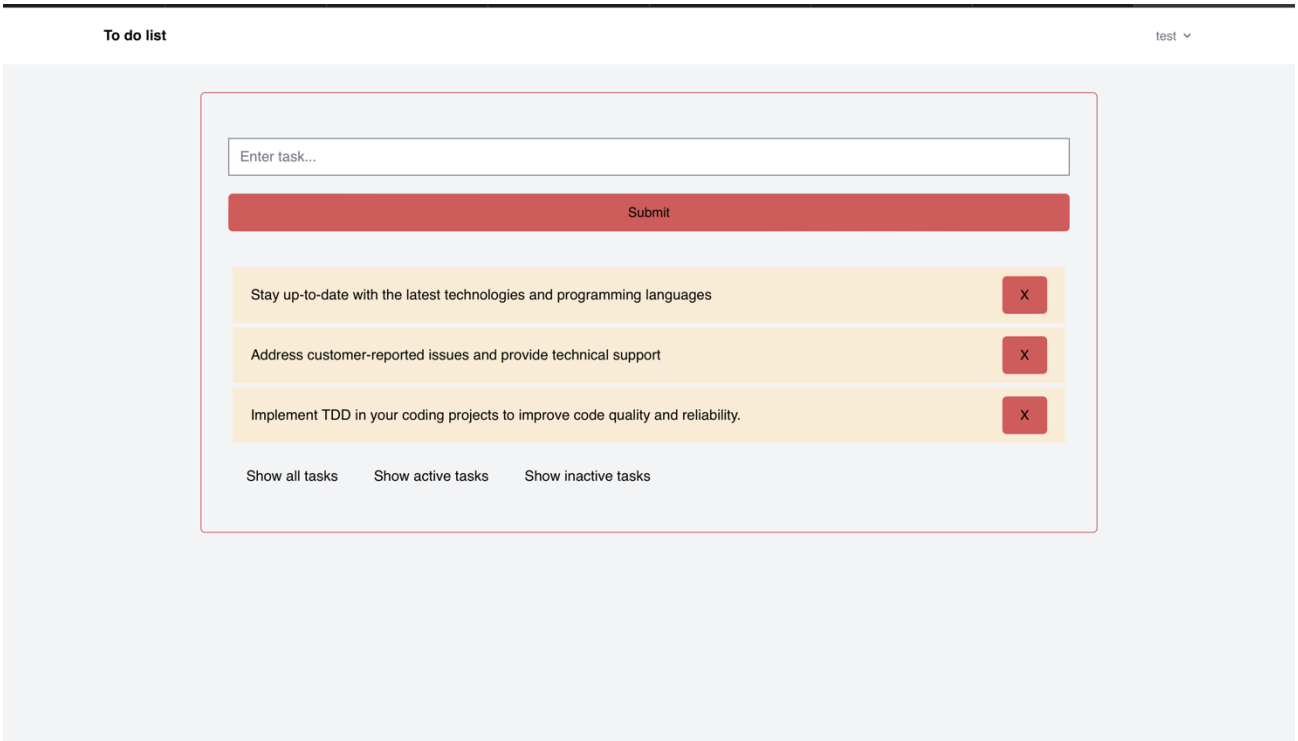
Ispod tipke *Submit* nalazi se lista svih unesenih zadataka prijavljenog korisnika, npr. *Identify and fix bugs in the codebase*. Time je ispunjen kriterij čitanja (eng. read) za CRUD aplikaciju.

Duplim pritiskom na okvir unutar kojega se nalazi naziv zadatka zadatak se označava odrađenim te se naziv zadatka precrtava punom duljinom te se na lijevom kraju okvira zadatka pojavljuje crvena linija. Ponovnim duplim pritiskom na okvir zadatak se vraća nazad u stanje neodrađenog zadatka. Time je ispunjen kriterij ažuriranja (eng. update) za CRUD aplikaciju.

Na desnoj strani okvira zadatka nalazi se gumb *X* koji ukoliko je pritisnut briše određeni zadatak s liste zadataka prijavljenog korisnika. Time je ispunjen kriterij brisanja (eng. delete) za CRUD aplikaciju.

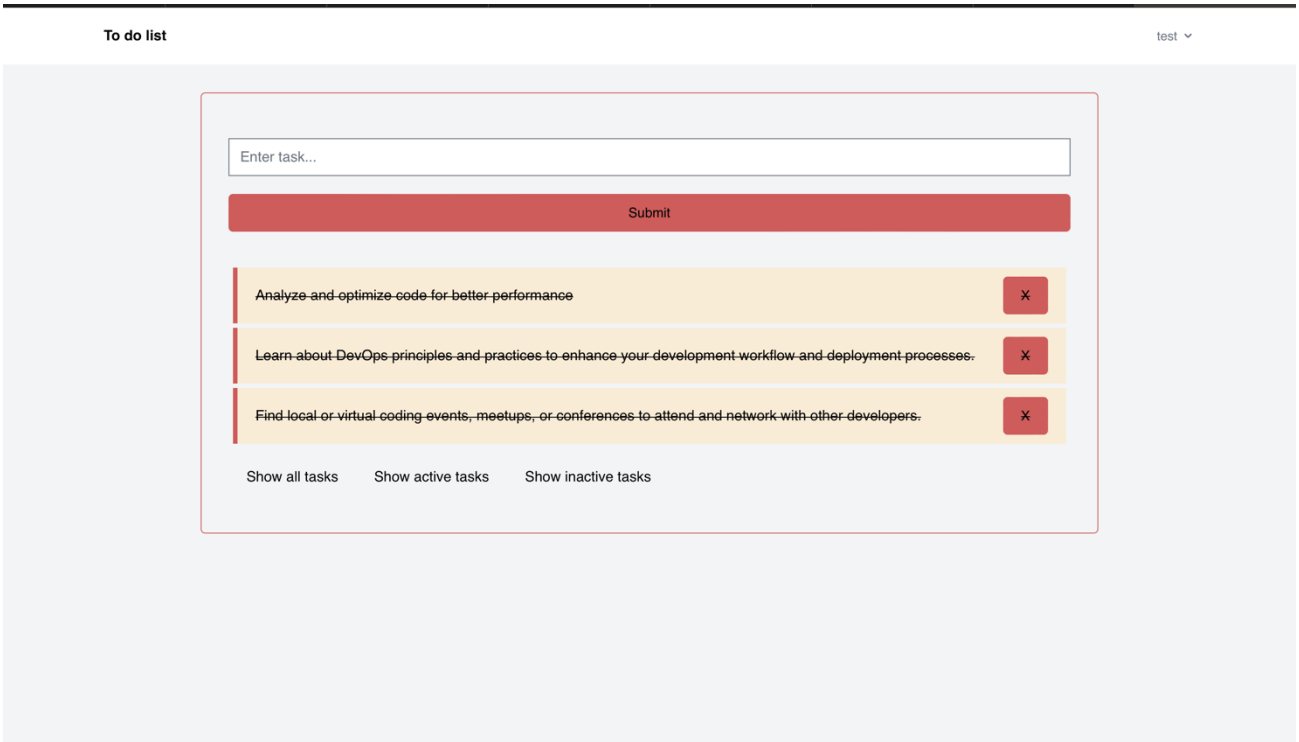
Ispod liste svih zadataka nalaze se još tri dodatne tipke: *Show all tasks, Show active tasks* i *Show inactive tasks*. Pritiskom na tipku *Show all tasks* prikazuje se lista svih korisnikovih zadataka.

Pritiskom na tipku *Show active tasks* prikazuje se korisnikova lista zadataka koji su aktivni, odnosno nisu izvršeni kao što je prikazano na *Slici 3.11*.



Slika 3.11 Pogled liste aktivnih zadataka korisnika

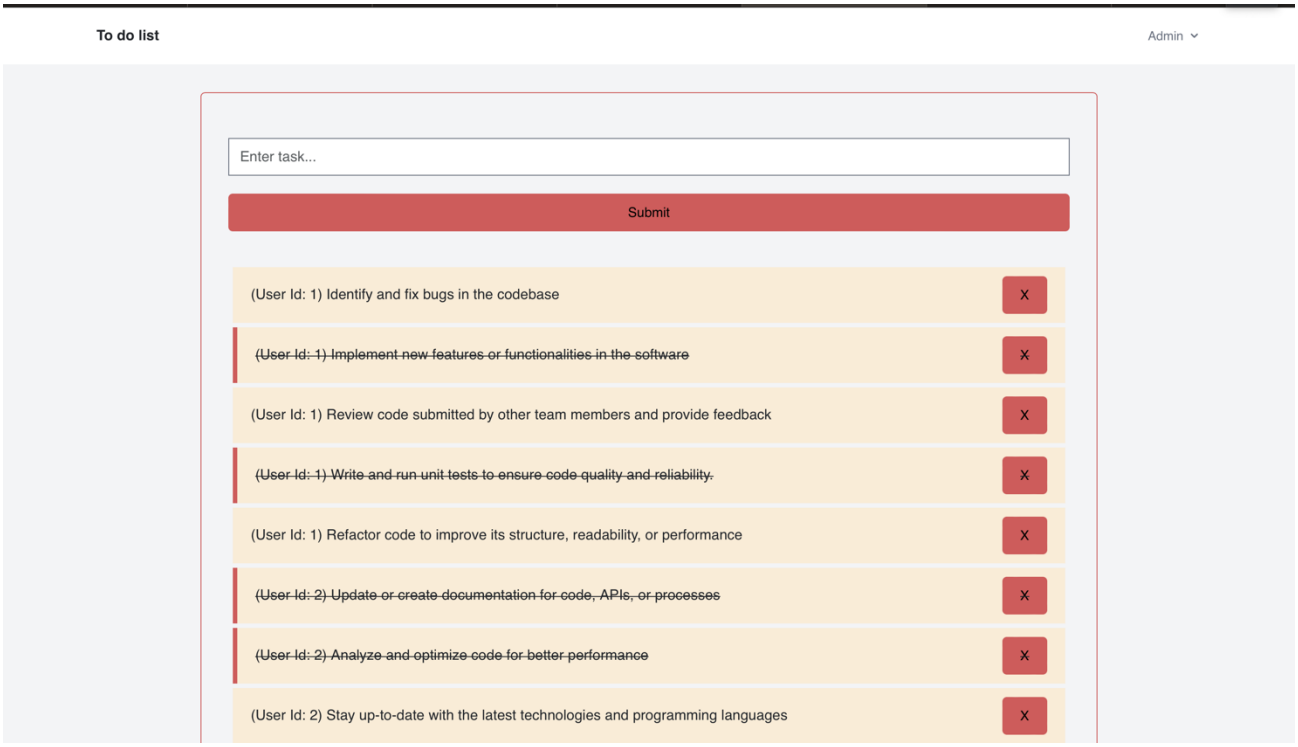
Pritiskom na tipku *Show inactive tasks* prikazuje se korisnikova lista zadataka koji nisu aktivni, odnosno koji su označeni kao izvršeni kao što je prikazano na *Slici 3.12*.



Slika 3.12 Pogled liste neaktivnih zadataka korisnika

3.3.2 Pogled liste zadataka admina

Pogled liste zadataka admina sadrži iste komponente kao i pogled liste zadataka korisnika. Razlikuje se po tome što ima privilegiju vidjeti sve zadatke svih korisnika. Na *Slici 3.10* prikazan je pogled liste zadataka admina.



Slika 3.13 Pogled liste zadataka admina

Admin vidi sve zadatke svih korisnika u jednoj listi zadataka te vidi koji od zadataka su označeni kao odrađeni, odnosno neodrađeni. Pored svakog zadatka vidljiv mu je identifikacijski broj korisnika koji je napravio taj zadatak, npr. (User Id: 1). On ima opciju pobrisati bilo koji zadatak, označit ga kao odrađenog ili ga označit kao neodrađenog.

Admin također na dnu ekrana ima tipke *Show all tasks, Show active tasks* i *Show inactive tasks* te tako može pregledavati koji zadatci su izvršeni, a koji nisu.

3.4 Poslužiteljska strana

Poslužiteljska strana aplikacije razvijena je u Laravelu. Sve komponente vezane uz *Laravel Breeze* paket za autentifikaciju automatski su implementirane preuzimanjem *Laravel Breeze* API-a: *php artisan breeze:install api.*

Model u Laravelu predstavlja ključni dio ORM (*Object-Relational Mapping*) sustava. Osnovna svrha modela je omogućiti interakciju s tablicama u relacijskim bazama podataka na način koji je orijentiran prema objektima što pojednostavljuje rad s podatcima u web aplikacijama. Na *Slici 3.14* prikazan je model zadatka *to do* liste pod imenom *Task.*

- *public $timestamps = false*: Ovo svojstvo „govori“ modelu *Task* da ne koristi automatsko upravljanje vremenima (eng. timestamps) za stvaranje i ažuriranje zapisa. To znači da neće automatski popunjavati *created_at* i *updated_at* stupce u tablici.
- *protected $attributes = [completed => 0]*: Ovdje se zadaju zadane vrijednosti za atribute modela. Konkretno, ovaj redak postavlja zadanu vrijednost *completed* atributa na 0 (što označava da zadatak nije završen) ako se stvara novi zadatak.
- *protected $fillable = [description, user_id]*: Ovo svojstvo definira koje atribute modela *Task* se mogu popuniti masovno (eng. mass assignable) putem metode *addTask()*. Atributi navedeni ovdje mogu se postaviti masovno, dok će svi ostali biti automatski zaštićeni od masovnog dodavanja. U ovom slučaju, dopušteno je masovno dodavati *description* i *user_id* (id prijavljenog korisnika).

```php
class Task extends Model
{
    use HasFactory;

    public $timestamps = false;
    protected $attributes = ['completed' => 0];
    protected $fillable = ['description', 'user_id'];
}
```

Slika 3.14 Model zadatka

Kontroler (eng. controller) na poslužiteljskoj strani ima ključnu ulogu u upravljanju i obradi zahtjeva koji dolaze od klijentske strane (web preglednika) prema web aplikaciji. Njegova glavna svrha je obrađivati zahtjeve, izvršavati poslovnu logiku, komunicirati s bazom podataka i generirati odgovor koji se šalje natrag klijentu. U dokumentu *TaskController.php* definirane su sve potrebne funkcije za uspješno obrađivanje CRUD aplikacije.

Jedna od tih funkcija jest *addTask()* čija implementacija je prikazana na *Slici 3.15*. Detaljan opis svake linije funkcije:

- *public function addTask(Request $request*): Ovo je deklaracija PHP funkcije naziva *addTask* koja će biti dostupna preko određene HTTP rute. Parametar *Request $request* označava da će funkcija primiti HTTP zahtjev kao argument i taj zahtjev će biti dostupan kao objekt *$request*. Klasa *Request* je dio Laravel radnog okvira te omogućuje obradu i pristup podatcima iz HTTP zahtjeva.
- *$task = Task::create($request->all())*: Ovdje se stvara nova instanca modela *Task* koji predstavlja zadatke u aplikaciji. Metoda *create* je dio *Eloquent ORM-a* koji omogućava jednostavno stvaranje novih redaka u bazi podataka. Pozivanjem *$request->all()* dobivaju se svi podatci iz HTTP zahtjeva, a ti podatci se koriste za stvaranje novog zadatka u bazi podataka.
- *return response($task, 201)*: Ova linija koda šalje odgovor natrag klijentu. Varijabla *$task* sadrži novi stvoreni zadatak koji je prethodno dodan u bazu podataka. Metoda *response()* se koristi za generiranje HTTP odgovora. U ovom slučaju, šalje se odgovor sa statusnim kodom 201 što znači *Created*. To označava da je zahtjev uspješno stvorio novi resurs (u ovom slučaju, novi zadatak). Nakon uspješnog dodavanja, klijent će dobiti odgovor s informacijama o novom zadatku, uključujući i njegovu jedinstvenu identifikaciju (ID) koja se automatski generira od strane baze podataka.

```php
public function addTask(Request $request)
{
    $task = Task::create($request->all());
    return response($task, 201);
}
```

Slika 3.15 Funkcija addTask()

U datoteci *Api.php* definirane su sve API rute koje aplikacija podržava. To uključuje definiranje ruta za dohvaćanje, stvaranje, ažuriranje i brisanje resursa te druge akcije koje su dostupne putem API-ja. U njoj su rute povezane s odgovarajućim kontrolerima. Unutar definiranja ruta definiraju se i HTTP

metode koje se koriste u web razvoju kako bi se definirale različite vrste zahtjeva prema resursima na web serveru.

- Metoda GET (dohvati) koristi se za prikazivanje stranica ili dohvaćanje podataka iz baze podataka, npr. *getActiveTasks* dohvaća sve aktivne, neodrađene zadatke po korisnikovom id-u kada korisnik zatraži stranicu na adresi *getActiveTasks/{id}*.
- Metoda POST (pošalji) koristi se za dodavanje novih resursa u bazu podataka, npr. *addTask* služi za dodavanje/slanje novih zadataka kada se pošalje obrazac na adresu *addTask.*
- Metoda PUT (ažuriraj) koristi se za ažuriranje postojećih zapisa u bazi podataka, npr. *toggleCompleted* mijenja stanje statusa *completed* određenog zadatka kada se pošalje PUT zahtjev na adresu *toggleCompleted/{id}*.
- Metoda DELETE (obriši) koristi za brisanje zapisa iz baze podataka, npr. *deleteTask* briše određeni zadatak kada se izvrši DELETE zahtjev na adresu *delete/{id}*.

Na *Slici 3.16* prikazan je isječak definiranje ruta te njihova povezanost s kontrolerima.

```php
// Add task
Route::post('addTask', [TaskController::class, 'addTask']);

// Delete task
Route::delete('delete/{id}', [TaskController::class, 'deleteTask']);

// Get active tasks
Route::get('getActiveTasks/{id}', [TaskController::class, 'getActiveTasks']);

// Get inactive tasks
Route::get('getInactiveTasks/{id}', [TaskController::class, 'getInactiveTasks']);

// Toggle completed
Route::put('toggleCompleted/{id}', [TaskController::class, 'toggleCompleted']);
```

Slika 3.16 Definiranje ruta u Api.php

```php
class TaskController extends Controller
{
    public function getAllTasks($id)
    {
        return response()->json(Task::where('user_id' , '=', $id)->get(), 200);
    }

    public function getTask($id)
    {
        $task = Task::find($id);
        if (is_null($task)) {
            return response()->json(['message' => 'Task not found'], 404);
        }
        return response($task, 200);
    }

    public function addTask(Request $request)
    {
        $task = Task::create($request->all());
        return response($task, 201);
    }

    public function updateTask(Request $request, $id)
    {
        $task = Task::find($id);
        if (is_null($task)) {
            return response()->json(['message' => 'Task not found'], 404);
        }
        $task->update($request->all());
        return response($task, 200);
    }

    public function deleteTask($id)
    {
        $task = Task::find($id);
        if (is_null($task)) {
            return response()->json(['message' => 'Task not found'], 404);
        }
        $task->delete();
        return response()->json(null, 204);
    }
    public function toggleCompleted(Request $request, $id)
        {
            $task = Task::find($id);
            if (is_null($task)) {
                return response()->json(['message' => 'Task not found'], 404);
            }
            $task->completed = !$task->completed;
            $task->update($request->all());
            return response($task, 200);
        }

    public function getActiveTasks($id)
    {
        return response()->json(Task::where('user_id' , '=', $id)
                        ->where('completed', '=', 0)
                        ->get(), 200);
    }

    public function getInactiveTasks($id)
        {
            return response()->json(Task::where('user_id' , '=', $id)
                            ->where('completed', '=', 1)
                            ->get(), 200);
        }

    public function toggleCompleted(Request $request, $id)
    {
        $task = Task::find($id);
        if (is_null($task)) {
            return response()->json(['message' => 'Task not found'], 404);
        }
        $task->completed = !$task->completed;
        $task->update($request->all());
        return response($task, 200);
    }
```

Slika 3.17 TaskController

Klijentska strana

Klijentska strana aplikacije odrađena je u dva različita JavaScript radna okvira: Next.js i React.js. Pošto je Next.js samo nadogradnja na React.js kodovi se pretežito ne razlikuju. Stoga su u ovom poglavlju sve funkcije prikazane u samo jednom radnom okviru.

Na *Slici 3.17* prikazana je definicija funkcije *handleToggle* koja se koristi za označavanje određenog zadataka izvršenim ili neizvršenim.

- *const customConfig = {...}*: Prvo se stvara objekt *customConfig* koji sadrži konfiguraciju za HTTP zahtjev. Ovdje se postavlja zaglavlje (eng. header) *Content-Type* na *application/json* kako bi se naznačilo da će se slati JSON podatci u zahtjevu.
- *return axios.put(...)*: Zatim se šalje PUT zahtjev prema određenom URL-u koji je oblikovan kako bi se promijenilo stanje određenog zadatka. PUT zahtjev se koristi za ažuriranje postojećih podataka.
- *http://localhost:8000/api/toggleCompleted/${todo.id}*: Navodi se URL za PUT zahtjev koji uključuje identifikator (ID) zadatka (*todo.id*) kako bi se jasno naznačilo koji se zadatak želi označiti kao izvršen ili neizvršen.

```
const handleToggle = todo => {
    const customConfig : {headers: {...}}  = {
        headers: {
            'Content-Type': 'application/json',
        },
    }
    return axios.put(
        url: `http://localhost:8000/api/toggleCompleted/${todo.id}`,
        customConfig,
    )
}
```

Slika 3.18 Funkcija handleToggle

Na *Slici 3.18* prikazana je funkcija *handleClick* unutar koje se poziva prethodno definirana funkcija *handleToggle*. Funkcija *handleClick* poziva se prilikom duplog pritiska na određeni zadatak uz pomoć slušatelja evenata *onDoubleClick={handleClick}*.

- *e.preventDefault()*: Prvo, funkcija poziva *e.preventDefault()* što sprječava pretragu zadatka koja se inače može dogoditi prilikom pritiska na HTML element unutar kojega se nalazi opis zadatka čime se sprječava da se stranica ponovno učita prilikom pritiska
- *handleToggle(todo):* Poziva se funkcija *handleToggle(todo)* koja mijenja stanje određenog zadatka (*todo*).
- *setCompleted(!completed)*: Koristi za promjenu lokalnog stanja komponente. Varijabla *completed* koja označava je li zadatak izvršen ili neizvršen. Ova linija koda preokreće trenutno stanje što znači da će, ako je zadatak bio označen kao izvršen, sada biti postavljen na neizvršen i obrnuto.

```
const [completed, setCompleted] = useState(todo.completed)

const handleClick = e => {
    e.preventDefault()
    handleToggle(todo)
    setCompleted(!completed)
}
```

Slika 3.19 Funkcija handleClick

4 USPOREDBA RADNIH OKVIRA NEXT.JS I REACT.JS

Za početak, vrijedno je napomenuti da je React.js biblioteka za izradu korisničkih sučelja. To znači da se koristi za izradu komponenata sučelja, upravljanje stanjem korisničkog sučelja i interakcijom s korisnicima. S druge strane, Next.js je radni okvir (eng. framework) za izradu web aplikacija koji uključuje React.js kao svoju biblioteku za korisničko sučelje, ali dodaje mnoge dodatne značajke koje mu uvelike mijenjaju performanse. Dakle, Next.js je samo nadogradnja na React.js. Iako je React.js JavaScript biblioteka, u nastavku usporedbe između React.js-a i Next.js-a, oba će se tretirati kao radni okviri. Na *Slici 4.1* ukratko su skicirane razlike između React.js-a i Next.js-a koje su detaljno objašnjene u nastavku ovog poglavlja.

Slika 4.1 Razlike između Next.js-a i React.js-a [15]

4.1 Razlike u performansama

Po pitanju performansi, Next.js zauzima prednost nad samim React.js-om iz razloga što on pruža dodatne optimizacije koje unaprjeđuju performanse web aplikacija.

Next.js ima ugrađenu podršku za *rendering* na strani poslužitelja (SSR) što znači da se HTML dokument u cijelosti generira na poslužitelju prilikom zahtjeva korisnika te se potom šalje na

preglednik korisnika. Time poboljšava vrijeme prvotnog učitavanja stranice te može pružiti bolje performanse na web stranicama koje imaju bogatiji sadržaj. Brže vrijeme prvog učitavanja posebno je korisno za optimizaciju web stranice (SEO) jer tražilice brže indeksiraju sadržaj koji je prisutan u HTML-u. Međutim, SSR može rezultirati slabijim performansama na strani klijenta iz razloga što nakon što se HTML pošalje klijentu, JavaScript se koristi za dinamičko ažuriranje sadržaja i interaktivnost što može dovesti do sporijih prijelaza između stranica jer se JavaScript mora iznova izvoditi. Također, Next.js podržava i CSR. [16, 17]

S druge strane, React.js se pretežito usredotočuje na CSR. U tom slučaju početni HTML je minimalan, a većinski dio posla vezan uz prikazivanje odvija se na pregledniku korisnika. Vrijeme prvog učitavanja može biti sporije iz razloga što preglednik prvotno treba preuzeti JavaScript datoteke te ih izvršiti kako bi generirao i prikazao sadržaj stranice. Nakon preuzimanja JavaScript datoteka, prijelazi između stranica su brži jer se dinamički ažurira samo određeni dio stranice bez potrebe za ponovnim učitavanjem cijelog dokumenta. SSR se često koristi za aplikacije s bogatim sadržajem gdje su brzina indeksiranja stranica i SEO od velike važnosti, dok se CSR pretežito koristi za jednostranične aplikacije jer su interaktivnost i brzi prijelazi između stranica za takav pristup ključni. [17] Pošto su u ovom radu za usporedbu radnih okvira korištene dvije SPA aplikacije, odabir radnih okvira Next.js i React.js nije uzrokovao značajnu razliku iz razloga što Next.js kombinira oba pristupa (SSR i CSR) kako bi pružio najbolje performanse za sve potrebe.

Next.js automatski vrši razdvajanje koda (eng. code splitting) što znači da samostalno razdvaja JavaScript kod na manje dijelove koji se potom učitavaju po potrebi. On analizira rute aplikacije te automatski generira odvojene JavaScript segmente za svaku rutu. React.js ne pruža tu značajku. To je vrijedna tehnika za razvoj SPA aplikacije jer pomaže optimizirati performanse, smanjuje vrijeme prvog učitavanja te poboljšava ukupno korisničko iskustvo. [16]

Još jedna od značajnih razlika između radnih okvira Next.js i React.js jest da Next.js ima ugrađenu podršku za statičko generiranje stranica (eng. static site generation, skraćeno SSG). HTML stranice generiraju se tijekom izrade, a to rezultira brzim učitavanjem stranica jer se unaprijed generirani HTML može izravno posluživati s mreže za dostavljanje podataka (eng. content delivery network, skraćeno CDN) bez izvršavanja JavaScripta. Pošto React.js nema podršku za SSG, ukoliko se koristi, programeri mogu koristiti alate treće strane, poput *Gatsbyja*. Jednostranične aplikacije ovise o dinamičkom sadržaju koji se dohvaća putem API-a te se prikazuje na klijentskoj strani. SSG sadržaj

generira tijekom izgradnje što ga čini manje prikladnim za jednostranične aplikacije koje zahtijevaju često mijenjanje podataka. [16]

4.2 Razlike u usmjerivačima

Next.js i React.js usmjerivače (eng. routers) koriste za upravljanje rutama unutar aplikacije, no imaju različite svrhe i posebne značajke.

Next.js sadrži ugrađeno rješenje za upravljanje rutama te nema potrebe za korištenjem dodatnih biblioteka poput *React Routera.* On nudi mogućnost usmjeravanja na klijentskoj strani i mogućnost usmjeravanja na poslužiteljskoj strani. Ovaj radni okvir također podržava dinamičko usmjeravanje koje omogućava stvaranje dinamičkih URL-ova s parametrima. Značajke usmjeravanja Next.js-a integrirane su u sami radni okvir što uvelike olakšava upravljanje rutama u kontekstu aplikacije. U Next.js aplikacijama usmjeravanje se integrira pomoću ugrađenog modula *next/router*. [18]
Za korištenje usmjeravanja u Next.js aplikaciji prvotno se treba uvesti modul *next/router*: *import { useRouter } from next/router;*. Next.js pruža *useRouter* kuku (eng. hook) koja omogućava pristup *router* objektu. Kuka se koristi kako bi dobili pristup metodama i svojstvima za usmjeravanje: *const router = useRouter();*. Neke od uobičajenih metoda uključuju:

- *router.push(url, as, options)* - Dodaje novi URL na stog i usmjerava do njega. Mogu se specificirati dodatne opcije kao što je *as* za promjenu prikazanog URL-a bez promjene stvarnog URL-a. Na *Slici 4.2* prikazan je primjer korištenja *router.push* metode. Ako je status odgovora 409, koristi se *router.push(/verify-email)* kako bi se korisnika preusmjerilo na stranicu za potvrdu e-pošte.

```
const { data: user, error, mutate } = useSWR('/api/user', () =>
    axios
        .get('/api/user')
        .then(res => res.data)
        .catch(error => {
            if (error.response.status !== 409) throw error

            router.push('/verify-email')
        }),
)
```

Slika 4.2 Primjer metode router.push

- *router.replace(url, as, options)* - Zamjenjuje trenutni URL na stogu s novim URL-om. Često se koristi za ponovno učitavanje stranice ili slanje obrazaca.
- *router.back()* - Usmjerava unatrag na prethodnu stranicu na stogu, slično kao tipka za povratak u pregledniku.
- *router.pathname* - Svojstvo koje se koristi u Next.js za dobivanje trenutne putanje (URL-a) učitane stranice. Na *Slici 4.3* prikazan je primjer korištenja *router.pathname* metode. U slučaju da je URL trenutne putanje jednak */dashboard* atribut *active* se postavlja u stanje *True*.

```
active={router.pathname === '/dashboard'}
```

Slika 4.3 Primjer metode router.pathname

- *router.query* - Omogućava pristup i manipulaciju upitnim parametrima (eng. query parameters) iz URL-a. Upitni parametri su obično dijelovi URL-a koji slijede nakon znaka "?", a koriste se za prenošenje dodatnih informacija na server ili za definiranje dinamičkih parametara u URL-ima. Na *Slici 4.4* prikazan je primjer korištenja *router.query* metode. Kod na slici se koristi za pristupanje i praćenje promjena u upitnom parametru *email* u URL-u. Kada se upitni parametar *email* promijeni u URL-u, *useEffect* će se izvršiti ponovno, ažurirati lokalno stanje *email* s novom vrijednošću i reagirati na promjene u *email* parametru.

```
useEffect(() => {
    setEmail(router.query.email || '')
}, [router.query.email])
```

Slika 4.4 Primjer metode router.query

React Router jest samostalna biblioteka koja je posebno dizajnirana za upravljanje rutama za React.js aplikacije. Glavna svrha *React Routera* je omogućiti dinamičko upravljanje prikazima i komponentama na temelju URL-a u React aplikacijama što je ključno za izradu interaktivnih web aplikacija. Omogućava deklarativno definiranje ruta i njihovih komponenti što znači da se jasno mogu odrediti koje komponente će se prikazivati za određene URL-ove. Također, omogućava programski pristup navigaciji putem *history* objekta ili putem kuka (eng. hooks) poput *useHistory*. Daje opciju ugnježđivanja ruta unutar drugih ruta kako bi se stvorile složenije aplikacije s više razina navigacije.

Omogućuje i privremeno blokiranje ili preusmjeravanje usmjeravanja što je korisno za implementaciju zaštite ruta. [18]

Na *Slici 4.5* prikazan je primjer implementacije usmjeravanja u aplikaciji napisanoj u React.js radnom okviru. Slika prikazuje cijeli *App.js* dokument. Na vrhu su uključene sve potrebne komponente:

- *App.css* koji sadrži CSS koji se odnosi na kompletnu aplikaciju.
- *Routes* i *Route* su komponente koje dolaze iz biblioteke *react-router-dom* i koriste se za upravljanje usmjeravanjem u React aplikacijama. Biblioteka *react-router-dom* omogućuje stvaranje dinamičkih i interaktivnih navigacijskih sustava za aplikaciju.
- Komponente koje navode koje stranice će se otvarati putem ruta (*Dashboard, Login, Register, Home, ForgotPassword, PasswordReset, NotFoundPage*).

Potom se stvara funkcija *App()* unutar koje se definiraju sve rute te predstavlja strukturu aplikacije.

- Unutar komponente <*Routes*> definira se konfiguracija usmjeravanja za aplikaciju.
- Svaki <*Route*> element predstavlja određenu rutu i komponentu koja će se prikazati kada se ta ruta aktivira. Komponenta je definirana unutar svojstva *element.* Na primjer, kada je aktivirana ruta /*login* prikazat će se komponenta *Login.*
- Ruta *path=“*“* je općenita ruta koja se pokreće ukoliko je unesena bilo koja ruta koja nije prethodno definirana te prikazuje komponentu *NotFoundPage*

Na kraju, komponenta *App* se izvozi kao zadani izvoz ovoga modula čime postaje dostupna za upotrebu u drugim dijelovima aplikacije.

```
import './App.css';
import { Routes, Route } from 'react-router-dom';
import Dashboard from 'pages/dashboard';
import Login from 'pages/login';
import Register from 'pages/register';
import Home from 'pages/home';
import ForgotPassword from 'pages/forgot-password';
import PasswordReset from 'pages/password-reset';
import NotFoundPage from 'pages/404';

function App() {
  return (
    <div className="antialiased">
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/login" element={<Login />} />
        <Route path="/register" element={<Register />} />
        <Route path="/forgot-password" element={<ForgotPassword />} />
        <Route path="/password-reset/:token" element={<PasswordReset />} />
        <Route path="/dashboard" element={<Dashboard />} />
        <Route path="*" element={<NotFoundPage/>}
        />
      </Routes>
    </div>
  );
}

export default App;
```

Slika 4.5 Primjer usmjeravanja u React aplikaciji

4.3 Razlike u navigaciji

Next/link je komponenta koja se koristi u Next.js aplikacijama za implementaciju navigacije između različitih ruta aplikacije. Za jednostranične aplikacije pruža optimiziranu navigaciju. Upotreba *next/link* komponente je vrlo jednostavna. Oko teksta ili tipke koje se želi pretvoriti u navigacijski element omota se *<Link>* komponenta te se unutar *href* parametra postavi željeni URL na koji je potrebno da link vodi. Da bi se *next/link* koristio u datoteci potrebno ga je uvesti na vrhu datoteke: *import Link from next/link*. [11] Na *Slici 4.6* prikazan je primjer korištenja *next/link* komponente. Primjer sa slike pokazuje isječak JavaScript datoteke za registraciju novih korisnika. Na dnu ekrana nalazi se link *Already registered*? koji kada je pritisnut odvodi korisnika na stranicu za prijavu (*/login)*.

```
<Link
    href="/login"
    className="underline text-sm text-gray-600 hover:text-gray-900">
    Already registered?
</Link>
```

Slika 4.6 Primjer korištenja next/link komponente

U React.js postoje *Link* i *NavLink* komponente unutar *react-router-dom* biblioteke. Obje komponente služe za navigaciju po stranicama, no postoje neke razlike među njima. [6]

Link komponenta je osnovna komponenta za navigaciju u *React Routeru*. Ona se koristi za stvaranje jednostavnih linkova koji vode na određene rute. [6] Na *Slici 4.7* prikazan je primjer uporabe *Link* komponente. Pritiskom na tekst *Back to home page* stranica vodi na početnu stranicu, odnosno /.

```
<Link to="/">
  Back to home page
</Link>
```

Slika 4.7 Primjer korištenja Link komponente

NavLink komponenta unaprijeđena je verzija *Link* komponente. Omogućuje dodavanje dodatnih stilova ili klasa aktivnom linku. [6] Na *Slici 4.8* vidljiv je primjer korištenja komponente *NavLink.* Pod *classname* su dodatno definirane promjene u stilu linka *Login* kao na primjer zatamnjenje okvira teksta ukoliko se mišem prelazi preko okvira teksta.

```
<NavLink
    to="/login"
    className="mb-4 px-4 py-2 text-sm font-semibold text-white bg-blue-500 rounded-md hover:bg-blue-600">
    Login
</NavLink>
```

Slika 4.8 Primjer korištenja NavLink komponente

4.4 Razlike u podrški za TypeScript

I Next.js i React.js imaju podršku za TypeScript. Razlike u podrški za TypeScript između React.js i Next.js su prvenstveno u načinu konfiguracije i ugrađenih opcija.

Next.js ima ugrađenu podršku za TypeScript. Prilikom stvaranja Next.js aplikacije korištenjem *create-next-app* postoji mogućnost odabira TypeScripta te se automatski konfigurira TypeScript okolina. Osnovne konfiguracije su već postavljene. Next.js također nudi opciju da API rute budu napisane u TypeScriptu što olakšava upravljanje podatcima između klijentske i poslužiteljske strane. Mogu se stvoriti i tipizirane stranice što olakšava manipulaciju podatcima između stranica. [16]

React.js zahtjeva ručnu konfiguraciju TypeScripta dodavanjem TypeScript paketa ili prilagodbom *tsconfig.jso*n datoteke. Moraju se i instalirati i definirati „tipovi" za React i DOM elemente pomoću *@types/react* i *@types/react-dom* paketa. [16]

4.5 Razlike u korištenju

Next.js se koristi za izradu marketinških stranica, stranica e-trgovine te *landing pages* (odredišne stranice u online marketingu). [15]

React.js se koristi za izradu platformi društvenih mreža (Facebook, Instagram, Pinterest i X), ekonomičnih platformi (Airbnb, Lyft, Uber), medijskih platformi (Yahoo!), platformi za *online video streaming* (Netflix) te SaaS (Software as a Service) alate kao što su SendGrid, Asana, InVisionApp i Zapier. [15]

4.6 Razlike u dokumentaciji

U dokumentaciji se najviše istražuju razlike između React.js-a i Next.js-a. Dobra dokumentacija pomaže u razumijevanju alata, odabiru biblioteka i drugim aspektima razvoja projekta. Internet je bogat resursima za Next.js i React. Next.js nudi praktične vodiče za izradu komponenata i razvoj aplikacija, dok React također nudi sličan pristup, ali s nekoliko uvodnih aktivnosti koje objašnjavaju osnove. Dokumentacija za React.js usmjerena je na osnovne koncepte, pristupe za izradu komponenti

i upravljanje stanjem korisničkog sučelja, dok se dokumentacija za Next.js fokusira na napredne teme vezane uz *server-side rendering* (SSR), *static site generation* (SSG) i rute unutar web aplikacija. Naredba *Create React App* štedi vrijeme i trud pri postavljanju razvojnog okruženja, omogućujući fokusiranje na razvoj aplikacije umjesto na konfiguraciju alata. Njihove dokumentacije zajedno čine obilje resursa za razvoj web aplikacija, pružajući programerima fleksibilnost u izboru alata i pristupa za izradu aplikacija na temelju React ekosustava. [15]

4.7 Provođenje testiranja

Testiranje je provođeno s Web inspector alatom u sklopu internetskog preglednika Safari. Podatci o brzini web aplikacija iščitavali su se iz korisničkog sučelja koje pruža pregled raznih statistika. Korisničko sučelje prikazano je na *Slici 4.9.*

Slika 4.9 Korisničko sučelje Web inspectora

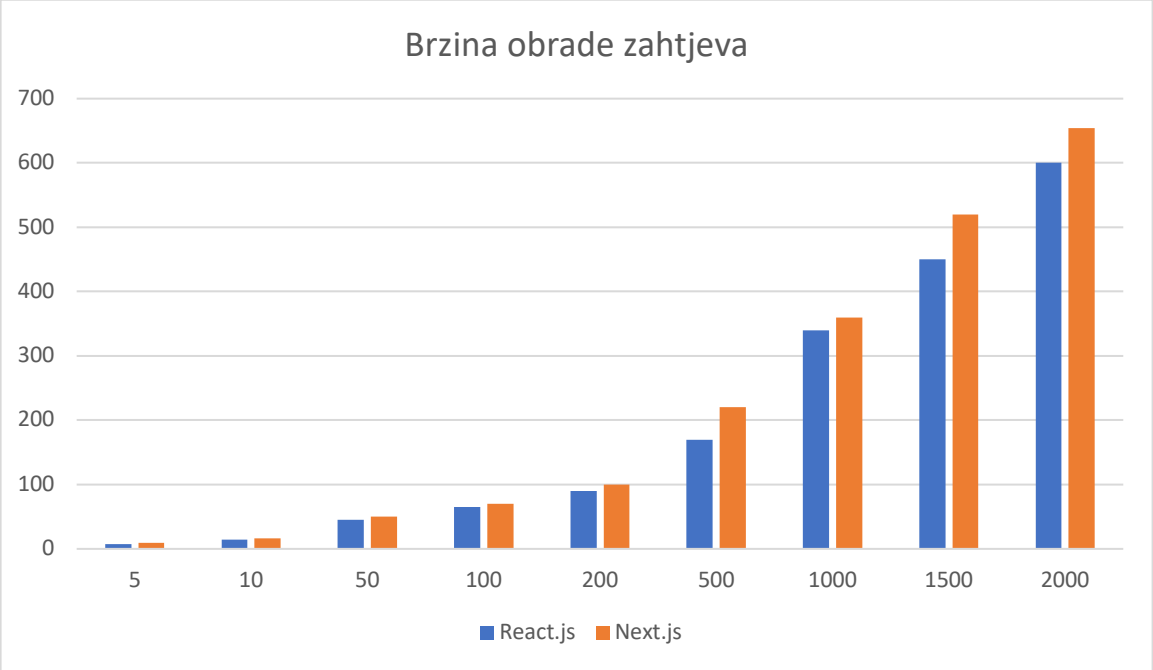
Pomoću tog korisničkog sučelja iščitali su se sljedeći podatci:

1. Next.js
    - Vrijeme prvog učitavanja stranice: cca 370 ms
    - Vrijeme za osvježavanje stranice: cca 440 ms
2. React.js
    - Vrijeme prvog učitavanja stranice: cca 120 ms
    - Vrijeme za osvježavanje stranice: cca 90 ms

Kao što se može primijetiti iz sakupljenih podataka, vrijeme učitavanja stranice u oba slučaja je više od tri puta brže za aplikaciju odrađenu s React.js klijentskom stranom.

Također, provedeno je testiranje brzine obrade HTTP zahtjeva koji se šalju prema poslužiteljskoj strani. Napravljena je funkcija kojom se mjeri brzina izvođenja asinkronih HTTP zahtjeva (POST zahtjeva) s pomoću Axios biblioteke. Koristeći *axios.post('http://localhost:8000/api/addTask', testingTask)* slali su se POST zahtjevi na poslužitelja, funkcijom *console.time* pokrenulo se mjerenje vremena te se nakon izvršenja slanja zahtjeva funkcijom *console.timeEnd* završilo mjerenje vremena te se u konzolu ispisalo proteklo vrijeme.

Na *Slici 4.10* prikazan je grafikon usporedbe brzine obrade zahtjeva između Next.js-a i React.js-a. Vidljivo je da su razlike u brzini neznačajno male, ali vodstvo preuzima React.js. Na horizontalnoj liniji navedeni su brojevi poslanih zahtjeva (5, 10, 50, 100, 200, 500, 1000, 1500, 2000), dok se na vertikalnoj liniji nalaze rezultati testiranja odraženi u milisekundama. Korišteni vremenski iznosi rezultat su srednje vrijednosti pet uzastopnih mjerenja vrijednosti.



Slika 4.10 Grafikon usporedbe brzine obrade zahtjeva

5 ZAKLJUČAK

U okviru ovog rada detaljno su analizirani i uspoređeni radni okviri za razvoj web aplikacija, Next.js i React.js. Cilj je bio istražiti njihove prednosti, nedostatke i specifičnosti te ih usporediti po njihovim performansama i karakteristikama. Oba okvira odličan su izbor za izradu web aplikacije poput To do aplikacije koja je napravljena u sklopu ovoga rada. Iako je Next.js temeljen na React.js-u postoje neke bitne razlike između ova dva radna okvira te svaki od njih ima svoje prednosti i mane za izradu različitih web stranica i aplikacija.

React.js je moćna i popularna JavaScript biblioteka za razvoj web aplikacija koja se temelji na komponentnoj arhitekturi što olakšava razvoj i održavanje. Njegova jednostavnost i velika zajednica programera čine ga privlačnim izborom za brz i jednostavan razvoj aplikacija. React.js se ističe brzinom izvođenja zahvaljujući virtualnom DOM-u što rezultira brzim učitavanjem korisničkog sučelja. Uz pomoć njegovih *useState* i *useEffect* kuka programeri održavaju bolju kontrolu nad stanjem varijabli i njegovim ažuriranjem što čini kod preglednijim i jednostavnijim za testiranje. Također, njegova sposobnost zaštite od XSRF-a čini ga sigurnim alatom za razvoj. Unatoč nedostatku podrške za *rendering* na strani poslužitelja, React.js ostaje snažan izbor za izgradnju modernih web aplikacija.

Radni okvir Next.js napredniji je utoliko što ima dodatne podrške i funkcionalnosti. Ističe se podrškom za *rendering* na strani poslužitelja što znači da se HTML stranice generiraju na poslužiteljskoj strani što poboljšava brzinu učitavanje i optimizaciju web stranice (SEO). Također, podržava i generiranje statičkih stranica što je korisno za web stranice s velikim brojem posjetitelja jer smanjuje opterećenje poslužitelja. Provodi dinamičko učitavanje komponenata što pomaže u smanjenju inicijalnog vremena učitavanja aplikacije. Za razliku od React.js-a ima ugrađen sustav za usmjeravanje te mogućnost definiranja API ruta unutar projekta čime se olakšava komunikacija između klijenta i poslužitelja. Automatski razdvaja JavaScript kod na strani poslužitelja i na strani klijenta, omogućavajući slanje samo ključnog koda potrebnog za inicijalno učitavanje. Međutim, upotreba Next.js-a zahtijeva dodatnu praksu u odnosu na osnovno razumijevanje React.js-a što može produžiti vrijeme učenja za neiskusne programere.

Odabir ova dva radna okvira nije igrao pretjeranu ulogu u izradi To do aplikacije pošto se radi o vrlo jednostavnoj SPA aplikaciji s RESTful arhitekturom te osnovnim CRUD operacijama. Sami kod, u oba okvira, poprilično je slično napisan te se jedine razlike u performansama primjećuju radi razlika u funkcionalnostima samih okvira. React.js pokazao se kao bolji odabir što se tiče brzine generiranja i prikazivanja web aplikacije na klijentskoj strani.

LITERATURA

[1] A. S. Gillis, "techtarget" [Online]. Dostupno: https://www.techtarget.com/searchapparchitecture/definition/RESTful-API. [Pristupljeno 4. rujna 2023.].

[2] K. Lawson, "bloomreach" [Online]. Dostupno: https://www.bloomreach.com/en/blog/2018/what-is-a-single-page-application. [Pristupljeno 4. rujna 2023.].

[3] [Online]. Dostupno: https://www.json.org/json-en.html. [Pristupljeno 4. rujna 2023.].

[4] "Laravel dokumentacija" [Online]. Dostupno: https://laravel.com/docs/10.x/starter-kits. [Pristupljeno 4. rujna 2023.].

[5] "geeksforgeeks" [Online]. Dostupno: https://www.geeksforgeeks.org/introduction-to-laravel-and-mvc-framework/. [Pristupljeno 4. rujna 2023.].

[6] "React dokumentacija" [Online]. Dostupno: https://react.dev/learn. [Pristupljeno 4. rujna 2023.].

[7] [Online]. Dostupno: https://dev.to/olenadrugalya/layout-component-and-why-we-use-it-in-react-d8b . [Pristupljeno 4. rujna 2023.].

[8] I. Mojeed. [Online]. Dostupno: https://blog.logrocket.com/virtual-dom-react/ . [Pristupljeno 4. rujna 2023.].

[9] Parnika-Gupta, "DEV" [Online]. Dostupno: https://dev.to/parnikagupta/one-way-data-binding-in-react-30ea. [Pristupljeno 4. rujna 2023.].

[10] M. Mohan, "codedamn" [Online]. Dostupno: https://codedamn.com/news/reactjs/usestate-and-useeffect-hooks. [Pristupljeno 4. rujna 2023.].

[11] "Next.js dokumentacija" [Online]. Dostupno: https://nextjs.org. [Pristupljeno 4. rujna 2023.].

[12] "Axios dokumentacija" [Online]. Dostupno: https://axios-http.com/docs/intro. [Pristupljeno 4. rujna 2023.].

[13] "TailwindCSS dokumentacija" [Online]. Dostupno: https://tailwindcss.com/docs/installation. [Pristupljeno 4. rujna 2023.].

[14] "Oracle" [Online]. Dostupno: https://www.oracle.com/mysql/what-is-mysql/ . [Pristupljeno 4. rujna 2023.].

[15] J. Patadiya, "Radix” [Online]. Dostupno: https://radixweb.com/blog/nextjs-vs-react. [Pristupljeno 6. rujna 2023.].

[16] G. Molina, "DistantJob” [Online]. Dostupno: https://distantjob.com/blog/nextjs-vs-react-pros-and-cons/. [Pristupljeno 6. rujna 2023.].

[17] "Search Engine Journal” [Online]. Dostupno: https://www.searchenginejournal.com/client-side-vs-server-side/482574/#close . [Pristupljeno 7. rujna 2023.].

[18] "geeksforgeeks” [Online]. Dostupno: https://www.geeksforgeeks.org/reactjs-router/ . [Pristupljeno 7. rujna 2023.].

[19] "MDN” [Online]. Dostupno: https://developer.mozilla.org/en-US/docs/Glossary/MVC. [Pristupljeno 7. rujna 2023.].

POJMOVNIK

HTTP – HyperText Transfer Protocol

HTML – HyperText Markup Language

SQL – Structured Query Language

JSON – JavaScript Object Notation

API – Application Programming Interface

URL – Uniform Resource Locator

DOM – Document Object Model

VDOM – Virtual Document Object Model

CSS – Cascading Style Sheets

ORM – Object-Relation Mapping

SSR – Server Side Rendering

CSR – Client Side Rendering

SEO – Search Engine Optimization

SAŽETAK

Predmet istraživanja ovoga rada bila je usporedba jednih od najpoznatijih radnih okvira za izradu web aplikacija kada je u pitanju klijentska strana: React.js i Next.js. U svrhu usporedbe izrađena je jednostavna jednostranična To do aplikacija s RESTful arhitekturom te osnovnim CRUD operacijama. Poslužiteljska strana izrađena je u programskom okviru Laravel koji je korišten kao aplikacijsko programsko sučelje. U radu su definirane i objašnjene sve karakteristike navedenih radnih okvira te su uspoređene po kategorijama: performanse, načini upravljanja rutama, navigacija među rutama, podrška za TypeScript, dokumentacija te u koju svrhu se koriste. Obrađen je i detaljan opis svih funkcionalnosti aplikacije. Također, provedeno je testiranje brzine učitavanja stranice i brzine izvođenja asinkronih HTTP POST zahtjeva poslanih na poslužitelja te su analizirani rezultati testiranja.

Ključne riječi: React.js, Next.js, Laravel, jednostranična aplikacija, usporedba, CRUD

ABSTRACT

The subject of this research was the comparison of two of the most well-known frontend frameworks for web application development: React.js and Next.js. In purpose to the comparison, a simple single-page To-Do application with RESTful architecture and basic CRUD operations was created. The server-side was developed using the Laravel framework serving as the application programming interface. The research defined and explained all the features of the mentioned frameworks and they were compared across various categories: performance, routing management methods, route navigation, TypeScript support, documentation and use cases. A detailed description of all application functionalities was provided. Additionally, application rendering speed and the execution time of asynchronous HTTP POST requests sent to the server were tested and the results were analyzed.

Keywords: React.js, Next.js, Laravel, single page aplication, comparison, CRUD