Usporedba Angular-a i React-a pri razvoju web aplikacija

Damjanović, Dino

Sveučilište u Zadru

Stručni preddiplomski studij Informacijske tehnologije

Dino Damjanović

Usporedba Angular-a i React-a pri razvoju web aplikacija

Završni rad

Zadar, 2023.

Sveučilište u Zadru

Stručni preddiplomski studij Informacijske tehnologije

Usporedba Angular-a i React-a pri razvoju web aplikacija

Završni rad

Student:

Dino Damjanović

Mentor:

Niko Vrdoljak, mag. ing. el.

Zadar, 2023.

Izjava o akademskoj čestitosti

Ja, **Dino Damjanović,** ovime izjavljujem da je moj **završni** rad pod naslovom **Usporedba Angular-a i React-a pri razvoju web aplikacija** rezultat mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na izvore i radove navedene u bilješkama i popisu literature. Ni jedan dio mojega rada nije napisan na nedopušten način, odnosno nije prepisan iz necitiranih radova i ne krši bilo čija autorska prava.

Izjavljujem da ni jedan dio ovoga rada nije iskorišten u kojem drugom radu pri bilo kojoj drugoj visokoškolskoj, znanstvenoj, obrazovnoj ili inoj ustanovi.

Sadržaj mojega rada u potpunosti odgovara sadržaju obranjenoga i nakon obrane uređenoga rada.

Zadar, 29. srpnja 2023.

Sažetak

U ovom radu provedena je detaljna usporedba Angulara i Reacta, dva okvira, odnosno knjižnica koje se ubrajaju među najpopularnije za razvoj web aplikacija. Rad započinje opisom osnovnih principa rada web aplikacija te objašnjava glavnu razliku između okvira (eng. frameworks) i knjižnica (eng. libraries) za razvoj web aplikacija.

Nadalje, teoretski su opisane funkcionalnosti oba alata, uključujući komponente, vezivanje podataka, preusmjeravanje i upravljanje stanjem. Osim teorijskog pregleda, izrađena je i vizualno i funkcionalno identična web aplikacija u oba alata, kako bi se omogućila praktična usporedba na vlastitim primjerima.

Na kraju rada, analizirano je stanje na tržištu u vrijeme izrade ovog rada, pružajući uvid u popularnost, krivulju učenja, fleksibilnost i namjenu Angulara i Reacta. Kroz ovu analizu, dobiven je pregled trenutne pozicije oba alata na tržištu te njihovih prednosti i nedostataka u kontekstu modernog razvoja web aplikacija.

Ključne riječi: web aplikacija, okvir, Angular, React

Sadržaj

Uvod

Popularizacija interneta uvelike je podigla zainteresiranost gotovo svih djelatnosti da sebe i svoje usluge približe svojim trenutnim i potencijalnim klijentima putem interneta. To je značilo ogroman porast u potražnji za izradom web stranica za iznimno brzo rastući broj klijenata, što je značilo da su programeri dobivali sve više sadržaja kojeg treba održavati, a uz to se i kompleksnost samih web stranica konstantno povećavala.

Uz povećanje kompleksnosti web stranica njihov razvoj i održavanje predstavljalo je sve veći izazov programerima. Odgovor na ovaj rastući problem stigao je u obliku razvoja frontend okvira (eng. frameworks) i knjižnica (eng. libraries) koje su znatno ubrzale razvoj web sadržaja te su programerima pružile dodatne mogućnosti kod razvoja istih. U daljnjem tekstu će okvir (framework) i knjižnica (library) biti oslovljavani kao okvir (framework) radi jednostavnosti, osim u slučaju kada je bitno naglasiti razliku.

Cilj ovog rada je opisati osnovne funkcionalnosti frontend okvira te napraviti usporedbu navedenih funkcionalnosti i tržišnih faktora između Angular-a i React-a, trenutno među najpopularnijim okvirima za razvoj web aplikacija (striktno gledano Angular je okvir, a React je knjižnica). Na kraju je razvijena aplikacija kako bi se u praktičnom dijelu demonstrirala usporedba.

1. Web aplikacije

Web aplikacija (eng. web application) je program pohranjen na udaljenom serveru. Klijent pristupa web aplikaciji koristeći web preglednik. Za vrijeme pristupa web aplikaciji korisnik mora ima aktivnu internetsku vezu [1]. Web aplikacije danas su postale trend zbog brojnih prednosti koje pružaju, od kojih su one najbitnije:

- Pružaju korisniku interakciju za razliku od statičnih web stranica: Korisničko iskustvo je gotovo identičnom onom koje pružaju desktop aplikacije. Web aplikacije korisniku pružaju brojne interaktivne mogućnosti koje statične stranice ne podržavaju, kao što je npr. rezervacija raznih objekata i usluga ili kupnja artikala.
- Fleksibilnost i agilnost korištenja: Web aplikaciju nije potrebno instalirati na uređaj na kojem se koristi iz razloga što se aplikacija pokreće u web pregledniku. Korisnik može pristupiti web aplikaciji bilo gdje uz uvjet da ima aktivnu internetsku vezu i na bilo kakvom uređaju koji na sebi ima web preglednik. Web aplikacije imaju jako niske zahtjeve glede računalne snage što omogućava njihovo pokretanje na uređajima kao što su pametni telefoni i tableti.
- Web aplikacije se ažuriraju centralno na serveru na kojem su naseljene što znači da će korisnici pri svakom pristupu web aplikaciji uvijek imati najnoviju verziju aplikacije. Uz to što ovaj pristup znatno olakšava održavanje, također se znatno olakšava dodavanje novih funkcionalnosti u već postojeću web aplikaciju. Sigurnosne postavke se također postavljaju centralno na serveru za sve korisnike.

Kao što je prethodno spomenuto, web aplikacije se pokreću u web pregledniku. Kako bi to bilo moguće, potrebno je da se web aplikacija u potpunosti učita u web preglednik pri inicijalnom pristupu korisnika URL-u (eng. Uniform Resource Locator) na kojem se nalazi sama web aplikacija. Jednom kad je aplikacija učitana u web preglednik sadržaj se dinamički učitava ovisno o tome na što korisnik klikne u aplikaciji. Ovaj princip implementacije web aplikacije naziva se „aplikacija s jednom stranicom“ (eng. Single-Page Application – skraćeno SPA). Ovime se postiže ugodno korisničko iskustvo identično onome u desktop aplikaciji. Naime, kad korisnik prelazi na novu stranicu, u aplikaciji nije potrebno slati novi zahtjev na server kako bi se učitao novi izgled stranice pošto je cijela aplikacija već učitana lokalno u web preglednik. Ovaj proces se naziva preusmjeravanje, koje će biti detaljnije opisano u nastavku ovog rada.

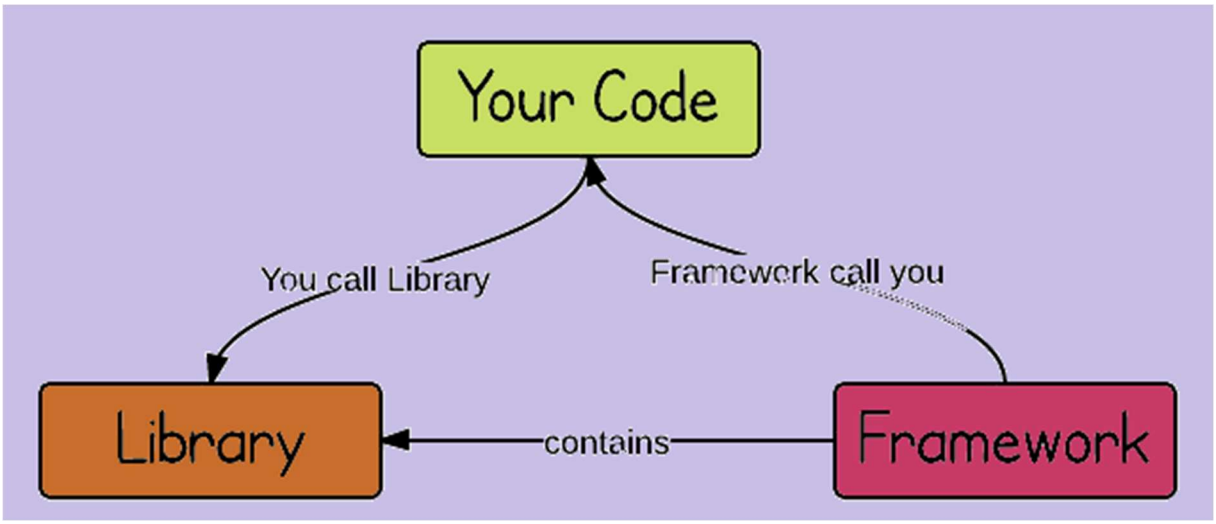
Web aplikacije se izrađuju koristeći komponente. To su „blokovi“ sadržaja koje korisnik vidi u aplikaciji kao što su gumbovi, izbornici, kartice itd. Komponente su iznimno praktične jer se mogu višestruko koristiti tamo gdje je to potrebno. Još jedna bitna značajka kod korištenja komponenti za izradu sučelja je to što se pri učitavanju stranica u web aplikaciji učitavaju samo oni dijelovi stranice, tj. one komponente koje su se promijenile u odnosu na prijašnju stranicu, dok se kod statičnih web stranica svaki put pri navigaciji na novu stranici u cijelosti učitava cjelokupna stranica. Konkretan primjer komponenti koje ostaju nepromijenje pri promijeni stranice su izbornici i logo tvrtke koji su obično na vrhu stranice. Pošto su ove komponente prisutne gotovo na svakoj stranici u web aplikaciji, nije potrebno da ih se učitava iznova pri svakoj promjeni stranice u aplikaciji već se učitavaju samo one komponente koje se dinamički mijenjaju čime se postiže brza i fluidna promjena sadržaja na ekranu. Komponente će također biti detaljnije opisane u nastavku ovog rada.

2. Frontend okviri i knjižnice

U ovom poglavlju bit će opisana glavna razlika između frontend okvira (eng.framework) i knjižnice (eng. library).

Kako se popularizacijom interneta broj web stranica kroz godine uvelike povećavao, programeri su tijekom razvoja istih tokom vremena uočili kako se određeni kôd često ponavlja. Kako bi ubrzali i olakšali implementaciju ponavljajućeg kôda razvijene su frontend knjižnice. Knjižnice su u osnovi skup napisanog kôda (funkcije, metode, klase, objekti) koje programeri mogu jednostavno koristiti u svom projektu. Važno je naglasiti da kod knjižnice programer ima potpunu kontrolu nad time kako će koristiti kôd iz knjižnice. Naime, programer uvozi knjižnicu u svoj projekt i nakon toga ima potpunu kontrolu nad time što iz knjižnice želi koristiti, te gdje i kada to želi koristiti.

Kod okvira je princip drugačiji – okvir funkcionira na temelju nečega što se naziva „inverzija kontrole“ (eng. inversion of control). Ovdje okvir preuzima kontrolu nad tokom izvođenja kôda. Okvir pruža programeru „mjesta“ gdje on može ubaciti svoj kôd koji će biti pozvan od strane okvira kada je to potrebno. Programeru je pružen „kostur“ ili prazan „okvir“ koji on mora popuniti svojim sadržajem uz to da poštuje strukturu i pravila okvira [2]. Prednost ovakvog pristupa je lakše održavanje, nadograđivanje i proširivanje aplikacije.



Slika 1 *Kontrola toka izvođenja kôda*

(Izvor: https://stackoverflow.com/questions/148747/what-is-the-difference-between-a-framework-and-a-library)

3. Funkcionalnosti okvira

Statičke web stranice se sastoje od tri glavna sloja: strukturni sloj koji čini osnovni sadržaj stranice (HTML), prezentacijski sloj kojim se postiže stilizacija (CSS), te sloj odgovoran za korisničku interakciju, gdje je ulogu preuzeo skriptni jezik JavaScript. Kako su poslovni zahtjevi za web stranice postajali sve kompleksniji, implementacija interaktivnih funkcionalnosti za korisnike web stranica postajala je sama po sebi sve kompleksnija. Jedan od najčešćih zahtjeva je postao taj da se korisnicima omoguće CRUD operacije nad podacima (CRUD je akronim za CREATE, READ, UPDATE, DELETE). To znači da se korisnicima omogućuje da čitaju podatke, izvršavaju unošenje novih, te izmjenjuju i brišu postojeće podatke. Implementacija ovih zahtjeva povlačila je za sobom brojne komplikacije od kojih su najveće bile izrazito zahtjevna implementacija spomenutih zahtjeva koristeći čisti JavaScript, te osvježavanje web stranice pri ažuriranju podataka koji se prikazuju korisniku na samoj stranici. Bilo je nužno da se korisniku dinamički mijenja sadržaj web stranice kako se mijenjaju podaci koje treba prikazivati, što je bilo iznimno problematično za implementirati koristeći čisti HTML, CSS i JavaScript. Frontend okviri su programerima uvelike pojednostavnili i ubrzali implementaciju ovakvih kompleksnih zahtjeva.

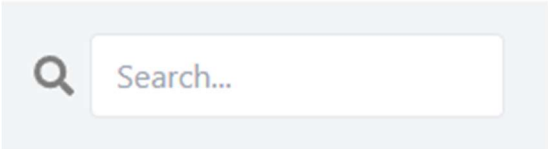
Iako se okviri i knjižnice razlikuju u vidu kontrole toka izvođenja kôda, oboje se temelje na nekim glavnim funkcionalnostima koje će biti opisane u nastavku, te uspoređene koristeći Angular i React.

3.1. Komponente

Većina frontend razvojnih okvira zasniva se na konceptu kreiranja web aplikacija kroz korištenje komponenata (eng. components) . Takav pristup omogućava strukturiranje aplikacija tako da se pojedinačni elementi kao što su zaglavlje, podnožje, glavni meni, filteri, kartice i ostali samostalni elementi logički enkapsuliraju, organiziraju i povezuju, čime zajedno formiraju jedinstvenu cjelinu. Komponente su manji dijelovi korisničkog sučelja koji uključuju HTML (sadržaj i strukturu), CSS (stilizaciju) i JavaScript (logiku ponašanja). Kao što je već spomenuto, iste komponente mogu se ponovno koristiti i referencirati unutar jedne aplikacije.

Sav kod povezan s komponentom obično se smješta u jednu datoteku ili direktorij, što olakšava izmjene i ispravke.

Slijedi primjer kako se jedna konkretna komponenta implementira u Angular-u i React-u. Radi se naime o vizualno i funkcionalno potpuno identičnoj komponenti implementiranoj u oba okvira koristeći standardne konvencije. Komponenta u pitanju naziva se `Search` komponenta – namijenjena je brzom pretraživanju sadržaja u web aplikaciji.



Slika 2 *Vizualni prikaz komponente* `Search`

Angular implementacija

Bitno je naglasiti da se u Angular-u strogo potiče korištenje TypeScript-a (TS) koji je superset JavaScript-a. Glavna značajka TS-a je što omogućava dodavanje statičkih tipova u JavaScript. To znači da se mogu deklarirati tipovi varijabli, funkcija, objekata itd., što pomaže u otkrivanju grešaka prije izvođenja kôda, a ne samo za vrijeme izvođenja. U Angular-u, sav kod povezan uz komponentu nalazi se u istom direktoriju koji se po konvenciji imenuje po nazivu komponente (malim slovima bez sufiksa „component“) [3]. U ovom slučaju direktorij se naziva „search“. Svaka Angular komponenta definirana je s četiri datoteke:

- Datoteka koja sadrži logiku ponašanja, `<ime-komponente>.component.ts`
- Datoteka koja sadrži sadržaj i strukturu, `<ime-komponente>.component.html`
- Datoteka koja sadrži stilove, `<ime-komponente>.component.css`
- Datoteka koja sadrži testnu specifikaciju, `<ime-komponente>.component.spec.ts`

Prve dvije datoteke su obavezne pošto svaka komponenta ima svoju logiku ponašanja te strukturu i sadržaj, dok su zadnje dvije opcionalne.

Slijedi Angular implementacija komponente u kodu. Dekorator `@Component` u Angularu je specijalna vrsta deklaracije koja dodaje metapodatke klasi kako bi se definiralo kako ta klasa treba funkcionirati kao komponenta Angular aplikacije. Dekorator `@Component` pretvara klase TS-a u komponente Angular-a koje se mogu koristiti unutar HTML kôda aplikacije.

Unutar `@Component` dekoratora se obično nalazi:

- **`selector`**: String koji predstavlja CSS selektor. Koristi se za identifikaciju komponente unutar HTML kôda. Na primjer, ako je selector postavljen na 'app-search', komponenta se u HTML-u koristi kao `<app-search></app-search>`.

- **`templateUrl`**: Putanja do vanjske datoteke koja sadrži HTML kod za komponentu. Alternativno, može se koristiti inline HTML. U tom slučaju se sav HTML kod komponente definira unutar templateUrl polja u samoj TS datoteci.

- **`styleUrls`**: Niz putanja do CSS datoteka koje sadrže stilove specifične za komponentu. Također se može koristiti i styles za inline CSS za definiciju stilova u samoj TS datoteci.

U prikazanom primjeru, kao što je vidljivo, `styleUrls` nije korišten pošto nije obavezan, već su stilovi implementirani uz pomoć vanjske biblioteke.

`search.component.ts` datoteka:

```
import { Component } from '@angular/core';
import { DestinationsDataService } from "../../services/destinations-data.service";
import { faSearch } from '@fortawesome/free-solid-svg-icons';

@Component({
  selector: 'app-search',
  templateUrl: './search.component.html'
})
export class SearchComponent {
  public searchIcon = faSearch;
  public searchTerm: string = '';
  public searchPlaceholder: string = 'Search...';
```

```
  constructor(private destinationsDataService: DestinationsDataService) {
  }

  public onSearchTermChange() {
    if (this.searchTerm.length > 1) {
      this.destinationsDataService.toggleSearch(this.searchTerm);
    }
  }

}
```

`search.component.html` datoteka:

```
<div class="flex items-center">
  <fa-icon [icon]="searchIcon" class="mr-2" [style]="{ 'color': 'gray', 'font-size': '20px' }"/>
  <input
    class="text-gray-500 border rounded px-3 py-2 transition ease-in-out duration-150 focus:outline-none
focus:border-blue-500"
    [(ngModel)]="searchTerm"
    (ngModelChange)="onSearchTermChange()"
    placeholder="{{searchPlaceholder}}"
  />
</div>
```

React implementacija

U React-u se komponenta implementira na znatno drugačiji način. Naime, u React-u se kod za logiku (konvencionalno napisan u TS-u) i HTML kod nalaze u istoj datoteci, dok se stilovi uvoze iz vanjske datoteke pomoću `import` ključne riječi na početku datoteke. HTML se piše unutar React JSX-a (JavaScript XML), što omogućuje pisanje HTML-a unutar JavaScript-a, tj. TypeScript-a. Ako se koristi TS, sav kod vezan uz komponentu nalazi se u datoteci koja se konvencionalno imenuje po nazivu komponente koristeći PascalCase gdje svaka riječ u imenu počinje velikim slovom te se na kraju imena dodaje sufiks „Component“ [4]. U slučaju spomenute Search komponente, naziv komponente bi bio `SearchComponent`, a sama komponenta bi se u JSX kodu koristila kao `<SearchComponent/>`. Tip datoteke je `.tsx` - TSX je skraćenica za "TypeScript XML". TSX datoteke omogućuju pisanje HTML kôda unutar TypeScript kôda. Glavna prednost korištenja TSX datoteka u odnosu na standardne JSX datoteke je ta što TSX podržava statičku provjeru tipova koju nudi TS. To znači da programeri

mogu koristiti prednosti stroge tipizacije u svom kôdu, što može poboljšati čitljivost, održivost i sigurnost kôda, smanjujući mogućnost grešaka.

Slijedi React implementacija komponente u kodu. U React-u, komponente se mogu implementirati kao klasne komponente ili kao funkcionalne komponente. U posljednje vrijeme, funkcionalne komponente su postale popularnije te se klasne komponente gotovo više i ne koriste. Slijedi implementacija `Search` komponente kao funkcionalne komponente u React-u. Kao što je vidljivo, JSX se nalazi unutar `return` naredbe funkcionalne komponente. Stilovi niti u ovom primjeru nisu uvezeni iz vanjske datoteke, već su implementirani uz pomoć vanjske biblioteke.

`SearchComponent.tsx` datoteka:

```
import { useContext, useState } from "react";
import { FaSearch } from "react-icons/fa";
import { DestinationsContext } from "../DestinationsProvider";

export default function SearchComponent() {
  const searchPlaceholder: string = 'Search...';
  const [searchTerm, setSearchTerm] = useState('');
  const { toggleSearch } = useContext(DestinationsContext);

  const onSearch = (searchTerm: string) => {
    setSearchTerm(searchTerm);

    if (searchTerm.length !== 1) {
      toggleSearch(searchTerm);
    }
  }

  return (
    <div className="flex items-center">
      <FaSearch size={20} color="gray" className="mr-2" />
      <input
        className="text-gray-500 border rounded px-3 py-2 transition ease-in-out duration-150 focus:outline-
none focus:border-blue-500"
        value={searchTerm}
        placeholder={searchPlaceholder}
        onChange={(e) => onSearch(e.target.value)}
      />
    </div>
  );
};
```

3.2.Vezivanje podataka i vezivanje događaja

Vezivanje podataka (eng. data binding) je automatska sinkronizacija podataka između modela koji drži podatke i pogleda koji prikazuje podatke. Postoje dvije vrste vezivanja podataka – jednosmjerno i dvosmjerno vezivanje.

1. **Jednosmjerno vezivanje podataka:**
    - Podaci se prenose samo u jednom smjeru, obično iz modela u pogled. Promjene u modelu automatski se odražavaju u pogledu, ali promjene u pogledu ne utječu na model.
2. **Dvosmjerno vezivanje podataka:**
    - Podaci se prenose u oba smjera. Promjene u modelu se odražavaju u pogledu, a promjene u pogledu se automatski odražavaju u modelu.

Vezivanje događaja (eng. event binding) je mehanizam pomoću kojeg aplikacija reagira na razne korisničke akcije ili događaje (npr. klik mišem, pritisak tipke, itd.). Povezuje korisničke akcije sa odgovarajućim funkcijama ili metodama koje se izvršavaju kada se ti događaji dogode.

Angular implementacija

U Angular-u se **podaci jednosmjerno vezuju**, obično iz modela u pogled, na sljedeći način. Uzmimo za primjer prethodno spomenutu `Search` komponentu.

`search.component.ts` datoteka sadrži polje `searchPlaceholder` – ovo je string koji želimo prikazati u pogledu:

```
...

@Component({
  selector: 'app-search',
  templateUrl: './search.component.html'
})
export class SearchComponent {
  ...
  public searchPlaceholder: string = 'Search...';
  ...
}
```

`search.component.html` datoteka sadrži `input` element kojem atribut `placeholder` postavljamo na vrijednost polja `searchPlaceholder` koje se nalazi u modelu. Podaci se na ovaj način prenose samo u jednom smjeru, iz modela u pogled, što znači da bi se promjena stanja u modelu odrazila u pogled, ali obrnuto ne vrijedi.

```
<div class="flex items-center">
 ...
 <input
  ...
  placeholder="{{searchPlaceholder}}"
 />
</div>
```

Angular nudi izvornu podršku za **dvosmjerno vezivanje podataka** što se postiže korištenjem direktive `[(ngModel)]` koja omogućava automatsku sinkronizaciju između modela i pogleda. Kada se podaci promijene u modelu, pogled se automatski ažurira i obrnuto. Dvosmjerno vezivanje podataka je u prethodno spomenutoj `Search` komponenti postignuto na sljedeći način:

`search.component.ts` datoteka sadrži polje `searchTerm` u koje se pohranjuje string koji predstavlja vrijednost koju korisnik unosi u polje za pretragu koje se nalazi u `Search` komponenti (inicijalno postavljeno na `''`, tj. prazan string).

```
...

@Component({
 selector: 'app-search',
 templateUrl: './search.component.html'
})
export class SearchComponent {
 ...
 public searchTerm: string = '';
 ...
}
```

`search.component.html` datoteka sadrži `input` element u kojem koristeći `[(ngModel)]` direktivu vežemo pogled sa `searchTerm` poljem. Na ovaj način, sadržaj koji se prikazuje u `input` elementu (pogled), dvosmjerno se veže za polje `searchTerm` u modelu što znači da se promjene u modelu odražavaju u pogledu, a promjene u pogledu se automatski odražavaju u modelu.

```
<div class="flex items-center">
 ...
 <input
  ...
  [(ngModel)]="searchTerm"
  ...
 />
</div>
```

Angular omogućava **vezivanje događaja** korištenjem okruglih zagrada `(event)`. Primjerice, `(click)="doSomething()"` bi povezao klik miša s metodom `doSomething()` u komponenti. Vezivanje događaja je u prethodno spomenutoj `Search` komponenti postignuto na sljedeći način:

`search.component.ts` datoteka sadrži metodu `onSearchTermChange()`:

```
...

@Component({
 selector: 'app-search',
 templateUrl: './search.component.html'
})
export class SearchComponent {
 ...
 public onSearchTermChange() {
  if (this.searchTerm.length > 1) {
   this.destinationsDataService.toggleSearch(this.searchTerm);
  }
 }

}
```

`search.component.html` datoteka sadrži `input` element u kojem koristeći `[ngModelChange]` direktivu vežemo događaj `ngModelChange`, tj. promjenu sadržaja u `input` elementu, s metodom `onSearchTermChange` u modelu. Na ovaj način svaki događaj promjene sadržaja u `input` elementu poziva metodu `onSearchTermChange` u modelu.

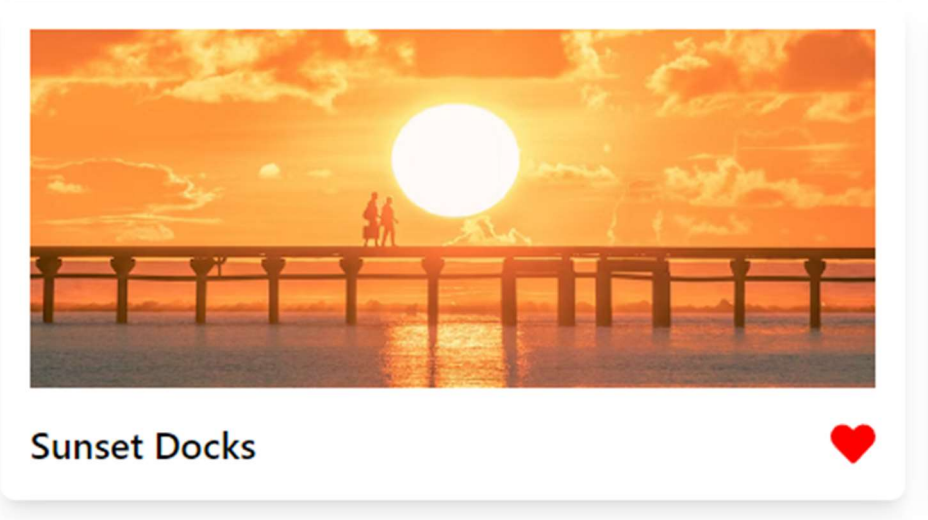
```
<div class="flex items-center">
 ...
 <input
  ...
  (ngModelChange)="onSearchTermChange()"
```

```
  />
</div>
```

React implementacija

U Reactu, podaci teku jednosmjerno – od roditeljskih komponenti prema dječjim komponentama. Ovo se postiže korištenjem rekvizita (props). Rekviziti su način na koji roditeljska komponenta prenosi podatke dječjoj komponenti. Oni su slični parametrima funkcija u JavaScriptu. Kada se podaci u roditeljskoj komponenti promijene, ti ažurirani podaci automatski se prosljeđuju dječjim komponentama kroz rekvizite. Ako roditeljska komponenta ažurira svoje stanje (state), sve dječje komponente koje primaju taj dio stanja kao rekvizit automatski će se ponovno renderirati s novim podacima.

Uzmimo za primjer `DestinationCard` komponentu koja prikazuje specifičnu destinaciju u aplikaciji.



Slika 3 *Vizualni prikaz* `DestinationCard` *komponente*

`DestinationCard.tsx` datoteka predstavlja `DestinationCard` komponentu:

```
export default function DestinationCard({id, name, imageUrl, isFavorite}: DestinationCardProps) {
  const {toggleFavorite} = useContext(DestinationsContext);

  return (
    <div className="relative p-4 bg-white shadow-lg rounded-lg overflow-hidden">
      <img className="w-full h-48 object-cover" src={imageUrl} alt="Destination"/>
      <div className="flex justify-between items-center mt-4">
        <div>
          <h3 className="text-xl font-semibold">{name}</h3>
        </div>
        <button onClick={() => toggleFavorite(id)} className="focus:outline-none">
          <FaHeart size={24} color={isFavorite ? 'red' : 'gray'}/>
        </button>
      </div>
    </div>
  );
};
```

Komponenta prima rekvizite `id`, `name`, `imageUrl` i `isFavorite`. Navedeni rekviziti se šalju u komponentu iz roditelja na sljedeći način – `DestinationList.tsx` je roditeljska komponenta u kojoj se kreiraju objekti `DestionationCard` komponente:

```
export default function DestinationList({onlyFavorites}: { onlyFavorites: boolean }) {
  const {destinationData} = useContext(DestinationsContext);

  return (
    <div className="grid gap-4 pb-4 px-4 grid-cols-1 md:grid-cols-2 lg:grid-cols-3">
      {destinationData
        .filter(destination => !onlyFavorites || destination.isFavorite)
        .map(destinationData => (
          <DestinationCard
            key={destinationData.id}
            id={destinationData.id}
            name={destinationData.name}
            imageUrl={destinationData.imageUrl}
            isFavorite={destinationData.isFavorite}
          />
        ))}
    </div>
  );
};
```

Svakom objektu `DestionationCard` komponente pojedinačno se šalju rekviziti koje komponenta koristi u prikazu (eng. view).

Za razliku od Angulara, **React ne nudi izvornu podršku za dvosmjerno vezivanje**. Međutim, može se implementirati ručno. Dvosmjerno vezivanje u Reactu postiže se kombinacijom stanja (state) komponente i događaja (event handlers). Stanje se koristi za praćenje vrijednosti, a događaji se koriste za ažuriranje tog stanja temeljem korisničkih interakcija. Uzmimo za primjer `Search` komponentu koja u sebi sadrži `input` element. `input` element ima svojstvo `value` vezano za stanje komponente (varijabla `searchTerm`). `onChange` događaj ažurira stanje pomoću metode `onSearch` kada korisnik mijenja tekst čime se postiže dvosmjerno vezivanje.

```
export default function SearchComponent() {
  const searchPlaceholder: string = 'Search...';
  const [searchTerm, setSearchTerm] = useState('');
  const { toggleSearch } = useContext(DestinationsContext);

  const onSearch = (searchTerm: string) => {
    setSearchTerm(searchTerm);

    if (searchTerm.length !== 1) {
      toggleSearch(searchTerm);
    }
  }

  return (
    <div className="flex items-center">
      <FaSearch size={20} color="gray" className="mr-2" />
      <input
        className="text-gray-500 border rounded px-3 py-2 transition ease-in-out duration-150 focus:outline-
none focus:border-blue-500"
        value={searchTerm}
        placeholder={searchPlaceholder}
        onChange={(e) => onSearch(e.target.value)}
      />
    </div>
  );
};
```

U Reactu, **vezivanje događaja** se obično radi unutar TSX kôda. To se postiže definiranjem handler-a za događaje kao atributa elemenata, kao što su `onClick`, `onChange`. Tako se u `Search` komponenti metoda `onSearch` poziva svaki put kada korisnik unese tekst u polje.

3.3.Preusmjeravanje

Kao što je prethodno spomenuto, u SPA vrsti web aplikacije, kakva se implementira koristeći Angular i React okvire, veći dio sadržaja se učitava inicijalno kada korisnik otvori web aplikaciju u pregledniku. U klasičnoj web aplikaciji, navigacija između stranica uključuje zahtjev prema poslužitelju, koji potom šalje cijelu novu stranicu korisnikovom pregledniku. Nasuprot tome, SPA dinamički preuređuje trenutnu web stranicu s novim podacima s poslužitelja. To smanjuje vrijeme potrebno za učitavanje stranica i poboljšava korisničko iskustvo.

Preusmjeravanje (eng. routing) u SPA-ma koristeći Angular i React okvire omogućuje navigaciju između različitih "pogleda" ili "komponenata" bez učitavanja nove stranice u pregledniku.

Angular implementacija

Konfiguracija za preusmjeravanje u priloženoj web aplikaciji je definirana u `app-routing.module.ts` datoteci:

```
import {NgModule} from '@angular/core';
import {RouterModule, Routes} from '@angular/router';

import {AboutComponent} from "./components/about/about.component";
import {HomeComponent} from "./components/home/home.component";

const routes: Routes = [
  {path: '', redirectTo: '/home', pathMatch: 'full'},
  {path: 'home', component: HomeComponent},
  {path: 'about', component: AboutComponent}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {
}
```

Angular koristi `RouterModule` za upravljanje rutiranjem. Prvo je potrebno definirati rute koje će se koristiti u aplikaciji te komponente koje se s njima povezuju. Rute koje su definirane

u priloženoj web aplikaciji su `home` (povezana s komponentom `HomeComponent`) i `about` (povezana s `AboutComponent`).

Zatim je potrebno uvesti `RouterModule` u glavni modul aplikacije koristeći `RouterModule.forRoot()` metodu te joj predati definirane rute kao argument.

U glavnom HTML predlošku (po konvenciji se naziva `app.component.html`), dodaje se `<router-outlet></router-outlet>` na mjesto gdje želimo da se prikažu komponente za određenu rutu.

`app.component.html` datoteka sadrži samo jednu liniju kôda:

```
<router-outlet></router-outlet>
```

Ono što se događa pri učitavanju aplikacije jest to da će se unutar `<router-outlet></router-outlet>` tagova prikazati komponenta koja je povezana s početnom rutom. Početna ruta je prazna ruta, a pošto je u `app-routing.module.ts` datoteci postavljeno da ukoliko se korisnik nađe na praznoj ruti, bit će presumjeren na `/home` rutu, korisnik će biti presumjeren na `/home` te će mu se prikazati `HomeComponent` koja je povezana sa `/home` rutom.

Za navigaciju unutar HTML-a koristiti se `routerLink` direktiva za kreiranje veza. Jedan takav primjer nalazi se u `Footer` komponenti, gdje se putem linka u footer-u korisnika preusmjerava na `/about` rutu:

```
<div class="bg-gray-100 text-gray-500 py-8 px-6 flex justify-between items-center">
  <p>© 2023 Dino Damjanović</p>

  <div class="flex space-x-10">
    <a [routerLink]="['/about']" class="hover:text-blue-500">
      About
    </a>
  </div>
</div>
```

Slika 4 *Vizualni prikaz* `Footer` *komponente*

Za programsku navigaciju unutar TypeScript kôda koristi se `Router` servis iz `@angular/router` paketa. Programska navigacija tada bi se izvela pomoću `navigate` metode iz Router servisa. Ovako bi izgledala implementacija u TS datoteci:

```
@Component({
  selector: 'app-footer',
  templateUrl: './footer.component.html'
})
export class FooterComponent {
  constructor(private router: Router) {}

  goToAboutPage() {
    // Programski navigira na '/about' putanju
    this.router.navigate(['/about']);
  }
}
```

React implementacija

U React-u se preusmjeravanje postiže pomoću `<BrowserRouter>` komponente koja se uvozi iz `react-router-dom` biblioteke. Ovo omogućuje korištenje HTML5 History API-a za navigaciju. Prvi korak je omatanje glavne aplikacije `<App/>` unutar `<BrowserRouter>` komponente. To se obično radi unutar korijenske datoteke aplikacije koja se konvencionalno naziva `index.tsx`:

```
import { createRoot } from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import { BrowserRouter } from "react-router-dom";

let root = createRoot(document.getElementById('root') as HTMLElement);

root.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
);
```

Zatim se definiraju rute i povezuju se s komponentama. U priloženoj aplikaciji rute i njihove pripadajuće komponente definirane se unutar `RoutesComponent`:

```
import {Navigate, Route, Routes} from "react-router-dom";
import Home from "../pages/Home";
import About from "../pages/About";

export default function RoutesComponent() {
  return (
    <Routes>
      <Route path="/" element={<Navigate to="/home"/>}/>
      <Route path="/home" element={<Home/>}/>
      <Route path="/about" element={<About/>}/>
    </Routes>
  );
}
```

Zatim se za stvaranje navigacijskih veza unutar web aplikacije koristiti `<Link>` komponenta iz `react-router-dom` biblioteke. Identičan primjer prikazan u Angular implementaciji u React-u bi izgledao ovako:

```
import { Link } from "react-router-dom";

export default function Footer() {
 return (
  <div className="bg-gray-100 text-gray-500 py-8 px-6 flex justify-between items-center">
   <p>© 2023 Dino Damjanović</p>

   <div className="flex space-x-10 hover:text-blue-500">
    <Link to={`/about`}>
     About
    </Link>
   </div>
  </div>
 );
};
```

3.4. Upravljanje stanjem

Upravljanje stanjem (eng. state management) je jedan od ključnih elemenata web aplikacije. Moderne web aplikacije nerijetko upravljaju velikom količinom podataka koje je potrebno spremiti, mijenjati i dohvaćati po potrebi. Stanje se dijeli na komponentno i globalno stanje.

Komponentno stanje

Svaka komponenta može imati svoje lokalno stanje. To su podaci koji su dostupni samo toj komponenti i njenoj djeci. Ukoliko se komponenta uništi, nestaju i svi njeni podaci.

U Angular-u se s komponentnim stanjem obično upravlja kroz svojstva klase komponente. Ta svojstva se mogu inicijalizirati izravno pri deklaraciji ili unutar konstruktora komponente. Slijedi primjer `Search` komponente koja ima tri stanja inicijalizirana pri deklaraciji i jedno stanje unutar konstruktora:

```
import { DestinationsDataService } from "../../services/destinations-data.service";
import { faSearch } from '@fortawesome/free-solid-svg-icons';

@Component({
  selector: 'app-search',
  templateUrl: './search.component.html'
})
export class SearchComponent {
  public searchIcon = faSearch;
  public searchTerm: string = '';
  public searchPlaceholder: string = 'Search...';

  constructor(private destinationsDataService: DestinationsDataService) {}
}
```

Stanje komponente je povezano s predloškom komponente kroz binding, omogućujući dinamično ažuriranje pogleda kada se stanje promijeni pomoću Angular-ovog mehanizma detekcije promjena.

U React-u, otkad se s klasnih komponenti prešlo na funkcionalne komponente, s komponentnim stanjem upravlja se pomoću React Hook-ova, najčešće koristeći `useState`. Stanje se dakle inicijalizira kroz `useState`, a ažurira se pomoću setter-a kojeg vraća `useState` hook. Na istom primjeru `Search` komponente slijedi primjer React implementacije:

```
import { useContext, useState } from "react";
import { FaSearch } from "react-icons/fa";
import { DestinationsContext } from "../DestinationsProvider";

export default function SearchComponent() {
 const searchPlaceholder: string = 'Search...';
 const [searchTerm, setSearchTerm] = useState('');

 const onSearch = (searchTerm: string) => {
  setSearchTerm(searchTerm);

  if (searchTerm.length !== 1) {
   toggleSearch(searchTerm);
  }
 }

 return (
  <div className="flex items-center">
   <FaSearch size={20} color="gray" className="mr-2" />
   <input
            ...
    value={searchTerm}
    placeholder={searchPlaceholder}
    onChange={(e) => onSearch(e.target.value)}
   />
  </div>
 );
};
```

`searchPlaceholder` nije potrebno naknadno ažurirati te je definiran kao varijabla funkcionalne komponente, dok je s druge strane `searchTerm` potrebno ažurirati pa se u tom slučaju koristi `useState` za inicijalizaciju i `setSearchTerm` za ažuriranje stanja. Kad se stanje komponente definirano kroz `useState` hook promijeni, React automatski ponovno renderira komponentu i njenu djecu ako je potrebno.

Globalno stanje

Globalno upravljanje stanjem omogućuje aplikacijama da održavaju podatke i logiku koja se dijeli kroz više komponenata ili čak kroz cijelu aplikaciju. Ovo je posebno važno u većim aplikacijama gdje je potrebno centralizirano upravljanje i ažuriranje podataka.

Angular koristi servise i injekciju ovisnosti za dijeljenje podataka između komponenata. Servisi su singleton objekti koji se mogu injektirati u bilo koju komponentu, omogućujući tako dijeljenje stanja. Singleton objekti su instancirani u skladu s dizajnerskim obrascem (eng.

design pattern) Singleton koji osigurava da klasa ima samo jednu instancu tijekom životnog ciklusa aplikacije. Na ovaj način sve komponente koje koriste singleton objekt će imati pristup istoj instanci klase pa samim time i istom stanju – onom koje se nalazi u toj instanci.

Slijedi primjer `DestinationsDataService` servisa koji drži podatke o destinacijama i aktivnom otvorenom tabu u aplikaciji. Zbog sažetosti, u servisu je prikazana samo jedna destinacija:

```
import { Injectable } from '@angular/core';
import { MenuOption } from "../../constants";
import { BehaviorSubject, Observable } from "rxjs";


export interface DestinationData {
  id: number;
  name: string;
  imageUrl: string;
  isFavorite: boolean;
}


@Injectable({
  providedIn: 'root'
})
export class DestinationsDataService {
  private readonly MountainLake: string = "../../assets/img/mountain_lake.png";
  ...

  public activeTab: MenuOption = MenuOption.DESTINATIONS;
  private isSearchActive: boolean = false;

  private allDestinationData: DestinationData[] = [
    {
      id: 1,
      name: 'Mountain Lake',
      imageUrl: this.MountainLake,
      isFavorite: false
    },
    ...
  ];

  private _activeDestinationDataSubject = new  BehaviorSubject<DestinationData[]>(this.allDestinationData);

  get activeDestinationDataSubject(): BehaviorSubject<DestinationData[]> {
    return this._activeDestinationDataSubject;
  }

  get activeDestinationData$(): Observable<DestinationData[]> {
    return this._activeDestinationDataSubject.asObservable();
```

```
  }

  toggleFavorite(id: number): void {
    const updatedDestinations = this.allDestinationData.map(destination => {
      if (destination.id === id) {
        return { ...destination, isFavorite: !destination.isFavorite };
      }
      return destination;
    });

    this.allDestinationData = updatedDestinations;

    if (this.isSearchActive) {
      const idSet = new Set(this._activeDestinationDataSubject.getValue()
                  .map(destination => destination.id));
      this._activeDestinationDataSubject.next(updatedDestinations
        .filter(destination => idSet.has(destination.id)));
    } else {
      this._activeDestinationDataSubject.next(updatedDestinations);
    }
  }

  toggleSearch(searchTerm: string): void {
    if (searchTerm.length === 0) {
      this.isSearchActive = false;
      this._activeDestinationDataSubject.next(this.allDestinationData);
      return;
    }

    const lowerCasedSearchTerm = searchTerm.toLowerCase();

    const updatedDestinations = this.allDestinationData.filter(destination => {
      return destination.name.toLowerCase().includes(lowerCasedSearchTerm);
    });

    this.isSearchActive = true;
    this._activeDestinationDataSubject.next(updatedDestinations);
  }

}
```

`@Injectable` anotacija označava ovu klasu kao injektabilnu u druge klase putem konstruktora. `activeTab` i `allDestinationData` varijable služe kao jedno centralno mjesto u aplikaciji gdje se nalaze podaci o trenutno aktivnom tabu te podaci o destinacijama u aplikaciji. Bitno je naglasiti da bi se podaci o destinacijama inače povlačili s poslužitelja u Angular aplikaciju te bi se zatim spremali u navedeni servis, međutim ovdje su zbog jednostavnosti ti podaci direktno zapisani u servis.

Komponente koje koriste ovaj servis, nakon što se injektira u komponentu, pomoću metode `get activeDestinationData$` pretplaćuju se (eng. subscribe) podacima i promjenama u podacima čime se te promjene u istom trenutku reflektiraju u samoj komponenti. Metoda `get activeDestinationDataSubject` daje pristup podacima u servisu komponentama koje koriste sam servis, metoda `toggleFavorite` omogućuje komponentama da upravljaju određenim stanjem u servisu, dok `toggleSearch` omogućuje pretragu podataka u servisu. Slijedi primjer kako se `DestinationsDataService` koristi u komponenti:

```
import { Component, Input, OnDestroy, OnInit } from '@angular/core';
import { DestinationData, DestinationsDataService } from "../../services/destinations-data.service";
import { Subject, takeUntil } from "rxjs";

@Component({
 selector: 'app-destination-list',
 templateUrl: './destination-list.component.html',
})
export class DestinationListComponent implements OnInit, OnDestroy {
 @Input() onlyFavorites!: boolean;
 public destinations!: DestinationData[];
 private destroy$ = new Subject<void>();

 constructor(private destinationsDataService: DestinationsDataService) {
 }

 ngOnInit() {
  this.destinationsDataService.activeDestinationData$
   .pipe(takeUntil(this.destroy$))
   .subscribe(data => {
    this.destinations = data;
   });
 }

 ngOnDestroy() {
  this.destroy$.next();
  this.destroy$.complete();
 }

}
```

Servis se injektira u komponentu putem konstruktora te se zatim u komponenti pomoću `get activeDestinationData$` metode pretplaćuje podacima i promjenama u podacima.

Za kompleksnije Angular aplikacije koristi se NgRx biblioteka. Osnovni koncept NgRx-a je upravljanje stanje koristeći Store (pohrana), Actions (akcije), Reducers (reduceri), Selectors (selektori) i Effects (efekti).

Pohrana je jedini izvor istine za stanje aplikacije. To je nepromjenjivo stablo objekata koje predstavlja stanje cijele aplikacije. Pohrana je kombinacija više reducera, od kojih svaki upravlja dijelom stanja aplikacije. Akcije su nositelji informacija koji šalju podatke iz aplikacije u pohranu. One su jedini izvor informacija za pohranu i opisuju što se dogodilo u aplikaciji. Reduceri određuju kako se stanje aplikacije mijenja kao odgovor na akcije. Oni su čiste funkcije koje uzimaju trenutno stanje i akciju kao argumente i vraćaju novo stanje. Selektori su čiste funkcije koje se koriste za odabir, izvođenje i sastavljanje dijelova stanja. Oni se mogu koristiti za upite prema pohrani za specifične dijelove podataka. Efekti se koriste za rukovanje asinkronim operacijama u Angular aplikaciji. Oni slušaju akcije poslane iz pohrane, obavljaju neki posao (npr. dohvaćanje podataka s API-ja) i šalju nove akcije kao rezultat.

U **React-u** se za upravljanje globalnim stanjem koristi React Context API-a. Context API omogućuje dijeljenje stanja između komponenti bez potrebe za prop drillingom (prosljeđivanjem propova kroz više razina komponenti) pošto sve komponente imaju pristup stanju na jednom centralnom mjestu koji je jedini izvor istine za stanje aplikacije. Slijedi primjer `DestinationsProvider` komponente koja implementira Context API. Svi podaci o destinacijama u aplikaciji nalaze se u ovoj komponenti. Zbog sažetosti, u servisu je prikazana samo jedna destinacija:

```
import React, { createContext, useState } from 'react';
import MountainLake from "../src/assets/img/mountain_lake.png";
...

interface DestinationData {
  id: number;
  name: string;
  imageUrl: string;
  isFavorite: boolean;
}

export const DestinationsContext = createContext({
  destinationData: [] as DestinationData[],
  toggleFavorite: (id: number) => {
  },
  toggleSearch: (searchTerm: string) => {
  }
});

export default function DestinationsProvider({ children }: React.PropsWithChildren<{}>) {
  const [isSearchActive, setIsSearchActive] = useState(false);
  const [allDestinationData, setAllDestinationData] = useState([
    {
```

```
    id: 1,
    name: 'Mountain Lake',
    imageUrl: MountainLake,
    isFavorite: false
  }, ...
 ]);
 const [activeDestinationData, setActiveDestinationData] = useState<DestinationData[]>(allDestinationData);

 const toggleFavorite = (id: number) => {
  const updatedDestinations = allDestinationData.map(destination => {
   if (destination.id === id) {
    return { ...destination, isFavorite: !destination.isFavorite };
   }
   return destination;
  });

  setAllDestinationData(updatedDestinations);

  if (isSearchActive) {
   const idSet = new Set(activeDestinationData.map(destination => destination.id));
   setActiveDestinationData(updatedDestinations.filter(destination => idSet.has(destination.id)));
  } else {
   setActiveDestinationData(updatedDestinations);
  }
 };

 const toggleSearch = (searchTerm: string) => {
  if (searchTerm.length === 0) {
   setIsSearchActive(false);
   setActiveDestinationData(allDestinationData);
   return;
  }

  const lowerCasedSearchTerm = searchTerm.toLowerCase();
  const updatedDestinations = allDestinationData.filter(destination => {
   return destination.name.toLowerCase().includes(lowerCasedSearchTerm);
  });

  setIsSearchActive(true);
  setActiveDestinationData(updatedDestinations);
 };

 const value = {
  destinationData: activeDestinationData,
  toggleFavorite,
  toggleSearch
 };

 return (
  <DestinationsContext.Provider value={value}>
   {children}
  </DestinationsContext.Provider>
 );
}
```

Ponovno, bitno je naglasiti da bi se podaci o destinacijama inače povlačili s poslužitelja u React aplikaciju, međutim ovdje su zbog jednostavnosti ti podaci direktno zapisani u kontekst. `DestinationsContext` je kreiran pomoću `createContext`, što omogućuje dijeljenje stanja i funkcija unutar cijelog React stabla komponenti koje koriste ovaj kontekst. `DestinationsProvider` komponenta sadrži stanje i funkcije za ažuriranje stanja. Ova komponenta omotava dijelove aplikacije kojima je potrebno pristupiti globalnom stanju. `useState` hookovi se koriste za definiranje i upravljanje lokalnim stanjem unutar `DestinationsProvider` komponente. `allDestinationData` sadrži sve podatke o destinacijama. `activeDestinationData` sadrži trenutno aktivne podatke o destinacijama, koji mogu biti filtrirani rezultati pretrage. Kao i u Angular implementaciji, metoda `toggleFavorite` omogućuje komponentama da upravljaju određenim stanjem u kontekstu, dok `toggleSearch` omogućuje pretragu podataka u kontekstu. `value` objekt sadrži trenutno stanje (`activeDestinationData`) i funkcije za njegovo ažuriranje (`toggleFavorite`, `toggleSearch`).

`DestinationsContext.Provider` omotava `children` – to su komponente koje su omotane ovim pružateljem konteksta i omogućuje svim potrošačima unutar njega pristup `value` objektu, čime se omogućuje globalno upravljanje stanjem. Slijedi primjer kako se `DestinationsProvider` koristi u aplikaciji za upravljanje globalnim stanjem. Prvo se sve komponente kojima želimo omogućiti pristup konteksu omotaju u `DestinationsProvider` – u ovom slučaju, sama korijenska komponenta `RouterComponent` je omotana, čime se svim komponentama u aplikaciji omogućuje pristup kontekstu:

```
import RoutesComponent from "./components/RoutesComponent";
import DestinationsProvider from "./DestinationsProvider";

export default function App() {
  return (
    <DestinationsProvider>
      <RoutesComponent />
    </DestinationsProvider>
  );
};
```

Zatim se `DestinationList` komponenti omogućuje pristupiti globalnom stanju pomoću `useContext` hook-a za dohvaćanje konteksta:

```
import {useContext} from "react";
import {DestinationsContext} from "../DestinationsProvider";
import DestinationCard from "./DestinationCard";

export default function DestinationList({onlyFavorites}: { onlyFavorites: boolean }) {
  const {destinationData} = useContext(DestinationsContext);

  return (
    <div className="grid gap-4 pb-4 px-4 grid-cols-1 md:grid-cols-2 lg:grid-cols-3">
      {destinationData
        .filter(destination => !onlyFavorites || destination.isFavorite)
        .map(destinationData => (
          <DestinationCard
            key={destinationData.id}
            id={destinationData.id}
            name={destinationData.name}
            imageUrl={destinationData.imageUrl}
            isFavorite={destinationData.isFavorite}
          />
        ))}
    </div>
  );
};
```

Na identičan način se može pristupiti i funkcijama unutar konteksta. Npr. za pristup `toggleFavorite()` funkciji u nekoj komponenti, implementacija bi izgledala ovako:

```
const {toggleFavorite} = useContext(DestinationsContext);
```

Na ovaj način, upravljanje globalnim stanjem postaje jednostavno i centralizirano.

Za kompleksnije React aplikacije, najpopularnija je Redux biblioteka za upravljanje stanjem. Inspiriran je arhitektonskim uzorkom Flux razvijenim od Facebook tima. Prethodno spomenuta NgRx biblioteka koja se koristi za upravljanje globalnim stanjem u Angular aplikaciji je također inspirirana Flux-om, zbog čega su Redux i NgRx biblioteke u principe jako slične.

Ključne komponente Redux-a su Store (pohrana), Actions (akcije), Reducers (reduceri), Selectors (selektori) i Middleware (efekti – Effects u NgRx-U). Kao što je vidljivo, komponente Redux-a su istog naziva kao one već spomenute u NgRx-u (izuzev Middleware-a). Namjena im je također identična.

Bitne značajke kod usporedbe Redux-a i NgRx-a su:

- NgRx je specifično dizajniran za Angular i koristi Angular-ove značajke poput Dependency Injection-a i modulskih sustava dok je Redux neovisan o okviru i može se koristiti s bilo kojim JavaScript okvirom ili knjižnicom, uključujući React, Angular, Vue.js itd.

- NgRx koristi `@ngrx/effects` za rukovanje asinkronim operacijama, dok Redux koristi middleware poput redux-thunk ili redux-saga (dva najpopularnija middleware-a za Redux).

- NgRx je napisan u TypeScriptu, što pruža jaku tipizaciju i sigurnost pri razvoju, dok Redux nema ugrađenu podršku za TypeScript, ali se može koristiti s TypeScript-om uz dodatnu konfiguraciju.

4. Tržišni faktori

4.1. Popularnost na tržištu

React i Angular su danas među najpopularnijim okvirima za razvoj web aplikacija s time da React zauzima visoko drugo mjesto, dok je Angular na petom mjestu po popularnosti [5]. Ulazeći malo dublje u analizu popularnosti među iskusnim programerima te onima koji su tek početnici, ono što se brzo uočava i što je bitno istaknuti jest da je React mnogo popularniji među početnicima u usporedbi s Angularom [6]. Razlog ovome će biti očit nakon usporedbe koja slijedi u nastavku.

4.2. Prednosti i nedostatci – usporedba

Angular

Prednosti:

- Angular je kompletan okvir, što znači da obuhvaća mnoge ugrađene funkcionalnosti poput dvostrukog vezivanja podataka, injekcije zavisnosti, preusmjeravanja, validaciju

obrazaca, upravljanje HTTP zahtjevima, kompletan alat za razvoj (Angular CLI), integrirane alate za testiranje i još mnogo toga.

- Angular ima vrlo dobre performanse što se ponajviše postiže korištenjem AoT (eng. Ahead-of-Time) kompajlera koji omogućava brzo učitavanje i izvršavanje kôda, čime se poboljšavaju performanse aplikacije [7].
- Angular je izrazito skalabilan što znači da je idealan za razvoj velikih, enterprise aplikacija zahvaljujući svojoj modularnoj arhitekturi i podršci za lijeno učitavanje (eng. lazy loading). Lijeno učitavanje omogućuje da se dijelovi Angular aplikacije ne učitavaju sve do trenutka kada budu potrebni [8]. Omogućava razvijanje aplikacija koje mogu lako rasti i skalirati se sa povećanjem broja korisnika i funkcionalnosti.
- Angular koristi MVC (Model-View-Controller) arhitekturu koja pruža jasnu strukturu i organizaciju kôda. Ovo olakšava održavanje i razvoj velikih aplikacija (naročito developerima koji tek počinju raditi na već velikoj Angular aplikaciji), jer razdvaja poslovnu logiku od prezentacijskog sloja.
- Angular ima opsežnu službenu dokumentaciju i aktivnu zajednicu developera, što pruža obilje resursa za učenje i podršku. Mnogi online kursevi, vodiči, blogovi i forumi su dostupni za pomoć developerima na svim nivoima.

Nedostatci:

- Glavni nedostatak Angular-a je strma krivulja učenja. Da bi se potpuno ovladao i razumio razvoj u Angular-u, developer mora imati dobro znanje o TypeScript-u jer je Angular zasnovan na TypeScript-u, koji je nadogradnja na JavaScript kao što je spomenuto u Angular sekciji, što znači da developer prvo mora savladati TypeScript prije nego počne raditi u Angular-u. Uz to, developer mora imati dobro razumijevanje Model-View-Controller (MVC) arhitekture za efikasan rad sa Angular-om.
- Drugi nedostatak se nadovezuje na prvi, a to je Angular-ova opsežna i detaljna dokumentacija, koja iako korisna, može biti preplavljujuća za novije developere. Ovo može otežati brzo savladavanje osnovnih koncepata i smanjiti produktivnost tokom početnih faza učenja, što je izuzetno nepovoljno ukoliko developer uči raditi u Angular-u na projektu u sklopu posla.
- Sljedeći nedostatak se također nadovezuje na prijašnje, a to je Angular-ova opsežnost. Angular može biti složen za upravljanje, posebno za manje projekte. Njegova struktura

zahtijeva mnogo konfiguracije i postavki prije nego što se može započeti sa razvojem, što može produžiti vrijeme razvoja i povećati složenost projekta.

- Sljedeći nedostatak je Angular-ova veličina. Naime, Angular je veliki okvir sa mnogo ugrađenih funkcionalnosti, što može dovesti do velikih fajlova i dužih vremena učitavanja. Ovo može biti problem za aplikacije koje zahtijevaju brze performanse i kratko vrijeme odziva, posebno na mobilnim uređajima sa ograničenim resursima.
- Angular u određenim situacijama ima sporije performanse u usporedbi s lakšim bibliotekama kao što je React. Njegova dvostrana vezanost podataka i složeni sustav za DOM manipulaciju mogu dovesti do nižih performansi u aplikacijama s velikim brojem interaktivnih elemenata.
- Zadnji bitan nedostatak je fleksibilnost. Za razliku od React-a, Angular je više "opinioni" okvir, što znači da ima striktna pravila i smjernice za razvoj aplikacija. Ovo može ograničiti kreativnost i fleksibilnost developera, jer se moraju pridržavati određenih obrazaca i praksi koje Angular nameće.

React

Prednosti:

- Jedna od najbitnijih prednosti React-a je to što je jednostavan za učenje, posebno za developere koji su već upoznati s JavaScript-om. React koristi JSX (JavaScript XML), sintaksu koja omogućava pisanje HTML-a unutar JavaScript koda, što čini kod preglednijim i lakšim za razumijevanje. React-ova jednostavna i intuitivna priroda olakšava brzo učenje i implementaciju.
- Sljedeća prednost se nadovezuje na prvu, a to je React-ova visoka fleksibilnost i modularnost u razvoju aplikacija. Zahvaljujući svojoj prirodi kao biblioteke, a ne kompletnog okvira, React omogućava developerima da izgrade svoje aplikacije koristeći komponente koje se mogu ponovo koristiti. Ovo olakšava održavanje i testiranje kôda jer se svaka komponenta može razvijati i testirati zasebno.
- Sljedeća prednost se također veže na prijašnje, a to činjenica da se React lako integrira s mnogim drugim bibliotekama i alatima za upravljanje stanjem (kao što su Redux i MobX) i preusmjeravanje (kao što je React Router). Ova modularnost omogućava developerima da biraju najbolje alate za specifične potrebe projekta, umjesto da budu

ograničeni unaprijed ugrađenim implementacijama kao što je slučaj kod Angular-a. Ovo čini React iznimno fleksibilnim naspram Angular-a.

- Nadalje, React ima veliku i aktivnu zajednicu koja pruža obilje resursa za učenje, uključujući vodiče, forume, blogove i online kurseve koji su vrlo korisni za rješavanje problema i učenja najboljih praksi. Pored toga, postoji veliki broj biblioteka i alata razvijenih od strane zajednice koji proširuju funkcionalnosti React-a.
- React dolazi s moćnim alatima kao što su Create React App za brzo postavljanje novih projekata i React Developer Tools za otklanjanje greški i inspekciju komponenti. Ovi alati pomažu developerima da brzo započnu razvoj i efikasno upravljaju svojim aplikacijama.
- React ima izuzetno visoke performanse zahvaljujući korištenju Virtual DOM-a koji omogućava efikasnije ažuriranje i prikaz korisničkog sučelja. DOM je stablasta struktura objekata koja predstavlja web stranicu, a web preglednici su zaduženi za njegovu obradu i prikaz elemenata korisnicima. U početku, kada je DOM bio osmišljen i implementiran, nije bio optimiziran za dinamička korisnička sučelja, budući da su prve web stranice bile statične. Iz tog razloga, izmjene u DOM-u su resursno jako skupe iz razloga što se prilikom mijenjanja jednog objekta ponovno obrađuje cijelo stablo i sve se ponovno ažurira i prikazuje na sučelju. Moderne web stranice imaju mnoštvo promijenjivih i interaktivnih elemenata – upravljanje promjenama bi nad njima bilo izuzetno zahtjevno glede resursa ukoliko se koristi zastarjeli način upravljanja DOM-om. Umjesto da direktno mijenja stvarni DOM, React prvo ažurira virtualni DOM, a zatim primjenjuje samo nužne promjene na stvarnom DOM-u. Ovaj pristup značajno poboljšava performanse aplikacija, osobito onih koje sadrže veliki broj interaktivnih elemenata.
- React Native je okvir zasnovan na React-u koji omogućava razvoj mobilnih aplikacija za iOS i Android koristeći isti kod. Ovo omogućava developerima da ponovo koriste značajan dio kôda između web i mobilnih aplikacija, čime se smanjuje vrijeme razvoja, povećava efikasnost i poboljšava produktivnost.

Nedostatci:

- React sadrži osnovne alate za izradu korisničkog sučelja, no za mnoge druge funkcionalnosti koje su potrebne za izradu kompletne web aplikacije, potrebno je koristiti dodatne biblioteke (npr. React Router, Redux, Axios itd.).
- Pošto React ne nameće svoju strukturu za izradu web aplikacije, potrebno je pažljivo planirati i organizirati projektni kod kako razvoj i održavanje aplikacije ne bi postali komplicirani. Ovo može postati veliki problem kod velikih web aplikacija, naročito ako u projektnom timu ima neiskusnih programera koji nisu iskusni u organizaciji kôda.
- Iako React koristi navedeni virtualni DOM za poboljšanje performansi, pogrešne prakse kodiranja mogu dovesti do problema s performansama. Npr. nepotrebna ažuriranja stanja mogu negativno utjecati na performanse aplikacije gdje se resursno zahtjevne komponente osvježavaju bez potrebe. Ovo naročito može postati problem u velikim aplikacijama.
- React često dobiva ažuriranja i promjene. Iako su ažuriranja općenito pozitivna stvar, mogu zahtjevati dodatni trud od developera da ostanu u toku s promjenama, što može biti izazovno u velikim projektima. Iz ovog razloga mnogi projekti, naročito oni veliki, budu u „zaostatku“ te koriste stare verzije React-a i odgovarajućih biblioteka, što može dovesti do poteškoća u razvoju i održavanju aplikacije.

Iz priložene usporedbe, može se zaključiti da je React pogodan za početnike jer je jednostavan za učenje, posebno za programere koji već znaju JavaScript. Također, modularni pristup omogućava postepeno učenje – nakon što savlada osnove React knjižnice, programer postepeno prelazi na React biblioteke. Ovo velikim dijelom objašnjava zašto je React popularniji kod početnika. Angular je bolje rješenje za velike, enterprise aplikacije koje zahtijevaju strukturu, dosljednost i sveobuhvatne ugrađene funkcionalnosti. Pogodan je za timove koji preferiraju strože smjernice i najbolje prakse razvoja. S druge strane, React pruža visoku fleksibilnost i modularnost. Omogućava programerima da biraju dodatne alate i biblioteke prema potrebama projekta, što daje slobodu u organizaciji i strukturi kôda.

4.3. Usporedba – tablica

Značajka	Angular	React
Tip	Okvir	Knjižnica
Razvijen od strane	Google	Facebook
Prvo izdanje	2010 (kao AngularJS), 2016 (kao Angular)	2013
Jezik	TypeScript	JavaScript (JSX)
Arhitektura	MVC (Model-View-Controller)	Komponentna
Podatkovno vezivanje	Dvostrano podatkovno vezivanje	Jednostrano podatkovno vezivanje
DOM	Pravi DOM	Virtualni DOM
Performanse	Visoke, ali može biti složeno zbog pravog DOM-a; koristi AoT (Ahead-of-Time) kompajler za brže učitavanje i izvršavanje	Općenito brže zbog virtualnog DOM-a; Virtualni DOM omogućava učinkovita ažuriranja, poboljšavajući performanse, posebno s velikim brojem interaktivnih elemenata
Veličina	Veća	Manja, zahtjeva dodatne knjižnice
Krivulja učenja	Strma; zahtjeva dobro razumijevanje TypeScript-a i MVC arhitekture	Umjerena; lakša za developere upoznate s JavaScript-om
Fleksibilnost	Manje fleksibilan, više opinioni okvir; stroga pravila i smjernice za razvoj	Vrlo fleksibilan, više slobode; modularan i omogućava izbor alata i knjižnica
Podrška zajednice	Velika, snažna korporativna podrška; opsežna službena dokumentacija i aktivna zajednica developera	Velika, snažna podrška zajednice; obilje resursa za učenje uključujući vodiče, forume, blogove i online tečajeve
Upravljanje stanjem	Ugrađene funkcionalnosti (NgRx, itd.)	Zahtjeva dodatne knjižnice (Redux, itd.)
Testiranje	Sveobuhvatni alati za testiranje (Jasmine, Karma)	Vanjski alati (Jest, Mocha, itd.)
Razvoj mobilnih aplikacija	NativeScript, Ionic	React Native; omogućava ponovnu upotrebu koda između web i mobilnih aplikacija

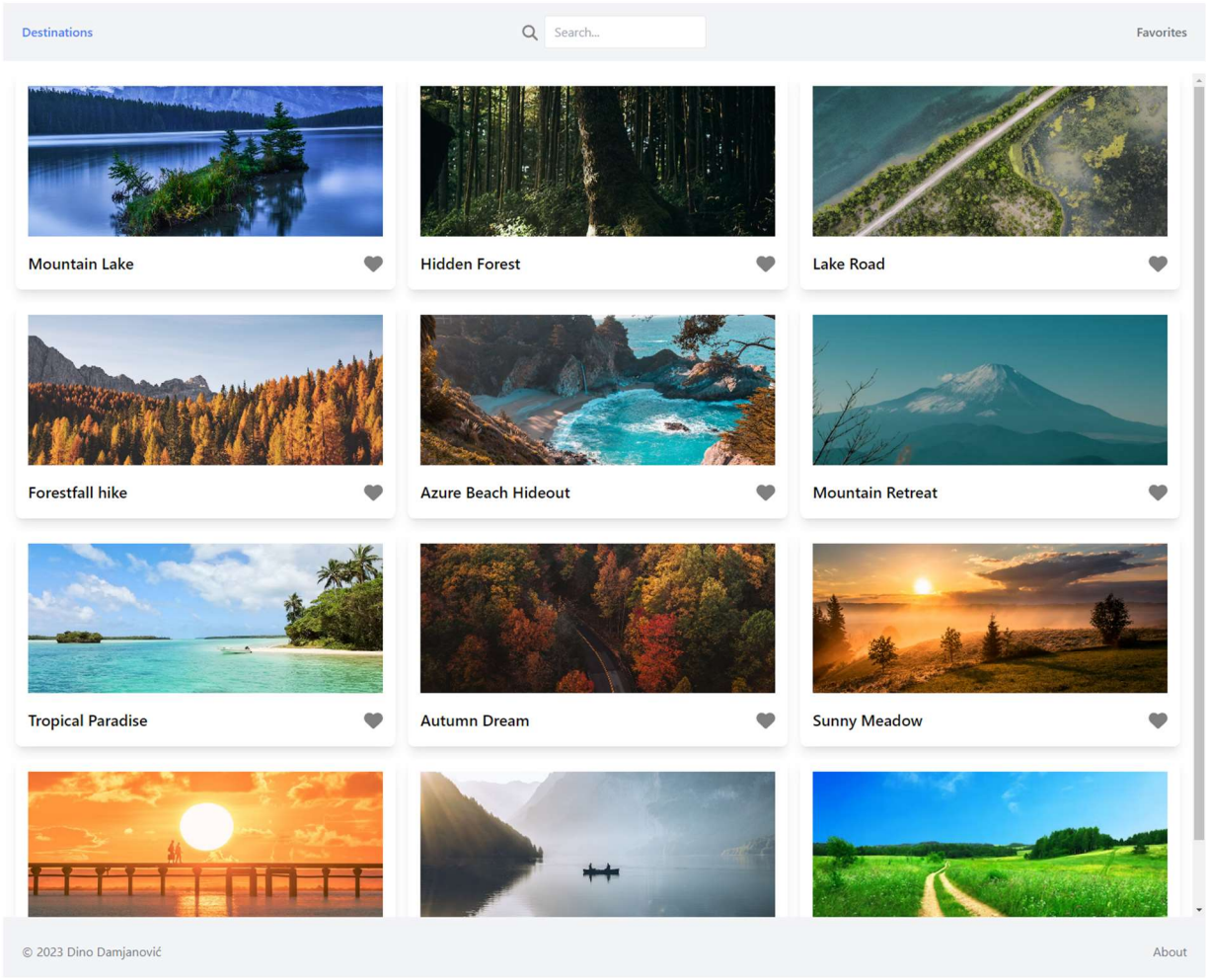
Predlošci	Predlošci temeljeni na HTML-u	JSX (JavaScript XML)
Routiranje	Ugrađeno (Angular Router)	Zahtijeva vanjsku knjižnicu (React Router)
Ažuriranja	Redovita ažuriranja s promjenama koje mogu uzrokovati nekompatibilnost	Postepena ažuriranja
Dokumentacija	Opsežna i detaljna; može biti preplavljujuća za nove developere	Dobra, ali često se oslanja na alate trećih strana
Upotreba	Velike enterprise aplikacije; visoko skalabilan s modularnom arhitekturom i lijenim učitavanjem	Jednostrane aplikacije, interaktivna korisnička sučelja; pogodna za male do velike aplikacije s mnogim interaktivnim elementima
Popularnost	Popularan, peto mjesto po tržišnoj popularnosti u vrijeme izrade rada	Vrlo popularan, drugo mjesto po tržišnoj popularnosti u vrijeme izrade rada; mnogo popularniji među početnicima zbog lakše krivulje učenja
Prednosti	Kompletan okvir s mnogim ugrađenim funkcionalnostima (podatkovno vezivanje, injekcija zavisnosti, preusmjeravanje, validacija obrazaca, upravljanje HTTP zahtjevima, Angular CLI, integrirani alati za testiranje); visoke performanse s AoT kompajlerom; visoko skalabilan	Jednostavan za učenje za JavaScript developere; vrlo fleksibilan i modularan; lako se integrira s drugim knjižnicama i alatima; snažna podrška zajednice; visoke performanse s Virtual DOM-om; moćni alati poput Create React App i React DevTools
Nedostaci	Strma krivulja učenja, posebno za nove developere; složena dokumentacija; opsežno postavljanje i konfiguracija; veća veličina i potencijalno sporije performanse za interaktivne aplikacije; manje fleksibilan zbog opinioniranosti	Zahtijeva dodatne knjižnice za mnoge funkcionalnosti; pažljivo planiranje i organizacija koda potrebni da bi se izbjegla složenost u velikim aplikacijama; česta ažuriranja mogu zahtijevati dodatni trud da bi se ostalo u tijeku

5. Web aplikacija – projekt

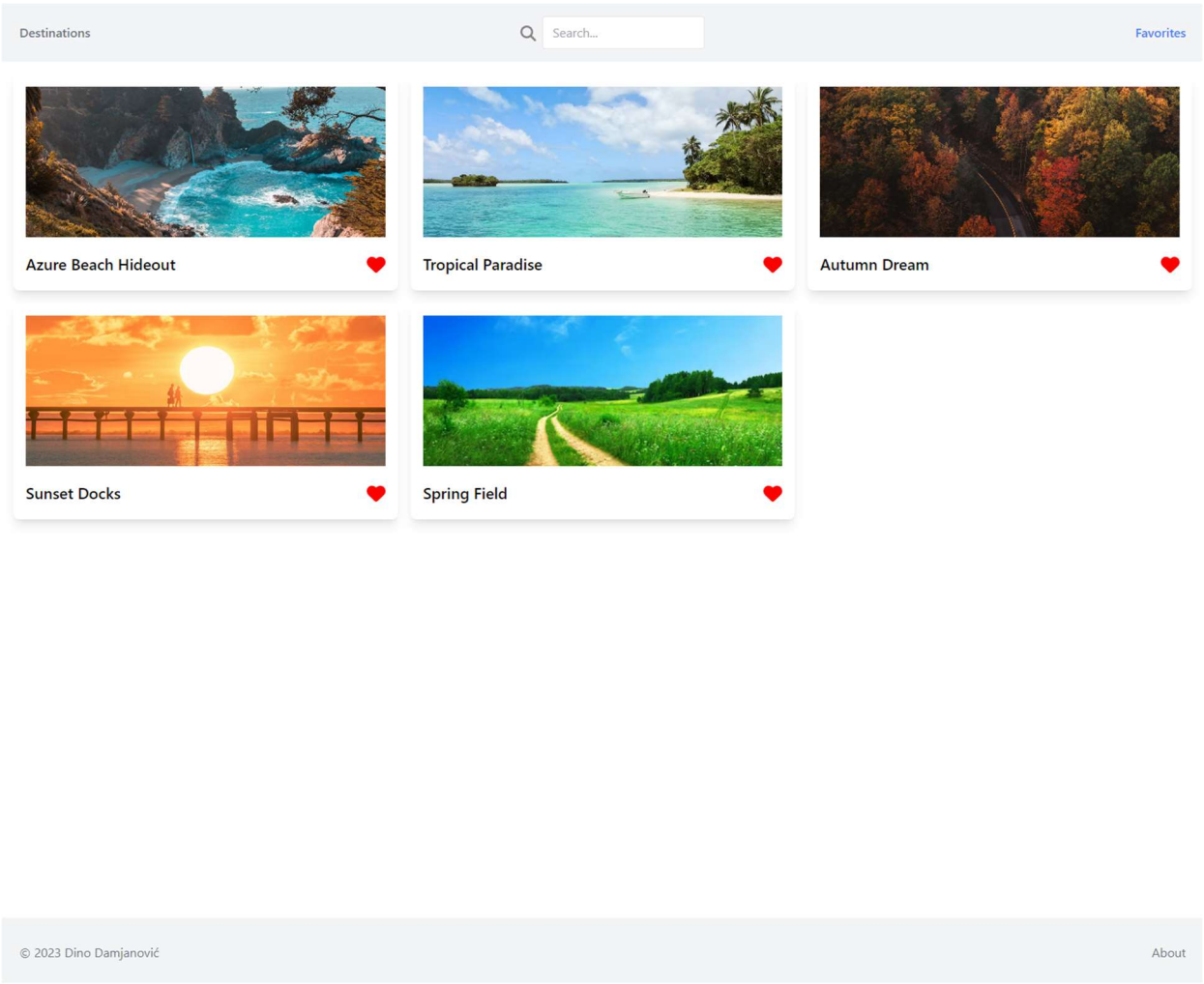
5.1. Opis aplikacije

U svrhu praktičnog dijela završnog rada izrađena je web aplikacija za pregledavanje prirodnih destinacija koja omogućuje korisnicima da otkrivaju i spremaju destinacije koje im se sviđaju pod omiljene (eng. favorites), označavajući ih srcem. Korisnici također mogu pretraživati sve destinacije unosom pojma za pretragu te pregledavati i također pretraživati svoje omiljene destinacije. Korisnici mogu lako navigirati kroz različite dijelove aplikacije uz dinamičku promjenu sadržaja, što je jedna od ključnih značajki SPA-a. Aplikacija pruža intuitivno i glatko korisničko iskustvo, čime se olakšava otkrivanje i organizacija prirodnih destinacija za posjetiti. Aplikacija je izrađena u React-u i Angular-u te identično izgleda u oba frameworka.

5.2. Izgled aplikacije



Slika 5 *Početna stranica "Destinations"*



Slika 6 *Stranica za pregled korisnikovih omiljenih destinacija "Favorites"*

5.3. Izrada projekta u Angular-u i React-u – osobni osvrt

Slijedi moj osobni osvrt na izradu projekta – svojevrsni osobni dojam te usporedba izrade projekta u React-u i Angular-u.

Izrada projekta u React-u bila je vremenski kraća – radi se o marginalno manjoj količini vremena uloženog u izradu.

Snalaženje u projektnom kôdu tijekom izrade projekta bilo je lakše u React-u. Razlog tome je uvelike Angular-ova nametnuta struktura projekta koja je namijenjena za veće projekte. React-ova sloboda samoorganizacije projektnog kôda ovdje je išla meni u korist. Pošto se radi o malenom projektu, imao sam slobodu organizirati komponente onako kako mi je bilo najintuitivnije.

Izrada komponenti za mene osobno bila je lakša u Angular-u. Jedan razlog tome je što preferiram da je logika ponašanja komponente (smještena u .ts datoteku) odvojena od sadržaja i strukture komponente (smješteni u .html datoteku). U React-u je sve zajedno smješteno u jednu .tsx datoteku. Drugi razlog je Angular-ova ugrađena podrška za dvosmjerno vezivanje podataka. Zahvaljujući već ugrađenoj podršci, implementacija dvosmjernoga toga podataka u Angular komponente je jednostavnija i inicijalno lakša za razumijeti. Treći razlog je što mi je Angular-ova mehanika za upravljanje lokalnim te naročito globalnim stanjem intuitivnija od one u React-u. Pošto preferiram strukturu i organizaciju kôda sličnu onoj kod razvoja aplikacija na backend-u, injektiranje servisa u klasu kako bi se upravljalo globalnim stanjem mi je intuitivnije i sam tok izvođenja kôda mi je lakše pratiti u Angular-u.

Zaključno, osobno sam preferirao razvoj projekta u Angular-u unatoč inicijalno težem snalaženju pri organizaciji projektnog kôda te kompleksnijem postavljanju potrebne početne konfiguracije kako bi se omogućilo pokretanje aplikacije (prvenstveno mislim na postavljanje konfiguracijske datoteke za module). Angular-ov način za implementaciju komponenti putem klasa te način za upravljanje stanjem koji logikom uvelike podsjeća na način pisanja kôda na backend-u, meni je kao prvenstveno backend developer-u bio intuitivniji i lakši za shvatiti od onog u React-u.

6. Zaključak

Iz usporedbi Angular-a i React-a napravljenih u ovom radu, može se zaključiti da oba okvira imaju svoje prednosti i nedostatke, te su prilagođeni različitim vrstama projekata i korisnika. Angular je robustan, sveobuhvatan okvir koji pruža mnoge ugrađene funkcionalnosti, što ga čini idealnim za velike, enterprise aplikacije koje zahtijevaju strukturu, dosljednost i skalabilnost. Međutim, strma krivulja učenja i složenost mogu biti prepreke za početnike i manje projekte.

S druge strane, React je lakši za učenje, posebno za programere koji su već upoznati s JavaScript-om. Njegova fleksibilnost i modularnost omogućavaju brzo i učinkovito razvijanje aplikacija, posebno onih s velikim brojem interaktivnih elemenata. React je popularniji među početnicima zbog jednostavnije krivulje učenja i veće slobode u izboru dodatnih alata i knjižnica.

Angular je bolje rješenje za timove koji preferiraju strože smjernice i najbolje prakse razvoja, dok React pruža više slobode i kreativnosti u razvoju aplikacija. Konačan izbor između Angular-a i React-a ovisi o specifičnim potrebama projekta, razini iskustva developera i preferencijama tima. Oba okvira imaju snažnu podršku zajednice i kontinuirano se razvijaju, što osigurava njihovu relevantnost i održivost u budućnosti.

Popis literature

[1] Riverbed (bez dat.), How Does a Web Application Work? [Na internetu]. Dostupno: https://www.riverbed.com/faq/how-does-web-application-work/ [pristupano 13.08.2023.]

[2] Ariel Camus, Microverse (2022), JavaScript Library vs JavaScript Frameworks - The Differences [Na internetu]. Dostupno: https://www.microverse.org/blog/javascript-library-vs-javascript-frameworks-the-differences [pristupano 14.08.2023.]

[3] Angular (2023), Angular Components Overview [Na internetu]. Dostupno: https://angular.io/guide/component-overview#angular-components-overview [pristupano 16.08.2023.]

[4] Smail Oubaalla, Medium (2023), How to name your React Component: Conventions [Na internetu]. Dostupno: https://medium.com/@smail.oubaalla/how-to-name-your-react-component-conventions-b8daf3abc574 [pristupano 16.08.2023.]

[5] Statista (2023), Most used web frameworks among developers worldwide, as of 2023 [Na internetu]. Dostupno: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/ [pristupano 17.06.2024.]

[6] Stack Overflow (2023), 2023 Developer Survey [Na internetu]. Dostupno: https://survey.stackoverflow.co/2023/#section-most-popular-technologies-web-frameworks-and-technologies [pristupano 17.06.2024.]

[7] Angular (2023), Ahead-of-time (AOT) compilation [Na internetu]. Dostupno: https://angular.dev/tools/cli/aot-compiler [pristupano 18.06.2024.]

[8] Angular (2023), Lazy-loading feature modules [Na internetu]. Dostupno: https://angular.dev/guide/ngmodules/lazy-loading [pristupano 18.06.2024.]

Comparison of Angular and React in web development

SUMMARY

From the comparisons made between Angular and React in this paper, it can be concluded that both frameworks have their own advantages and disadvantages, and are suited to different types of projects and users. Angular is a robust, comprehensive framework that provides many built-in functionalities, making it ideal for large, enterprise-level applications that require structure, consistency, and scalability. However, its steep learning curve and complexity can pose challenges for beginners and smaller projects.

On the other hand, React is easier to learn, especially for developers already familiar with JavaScript. Its flexibility and modularity allow for quick and efficient development, particularly for applications with a large number of interactive elements. React is more popular among beginners due to its simpler learning curve and greater freedom in selecting additional tools and libraries.

Angular is a better solution for teams that prefer stricter development guidelines and best practices, while React offers more freedom and creativity in application development. The final choice between Angular and React depends on the specific needs of the project, the experience level of the developers, and the preferences of the team. Both frameworks have strong community support and are continuously evolving, ensuring their relevance and sustainability in the future.

Keywords: web aplikacija, okvir, Angular, React

Popis slika