Web aplikacija za informiranje klijenata o ponuđenim sadržajima u hotelskom smještaju

Pivk, Luka

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

WEB APLIKACIJA ZA INFORMIRANJE KLIJENATA O PONUĐENIM SADRŽAJIMA U HOTELSKOM SMJEŠTAJU

Završni rad

Luka Pivk

Osijek, 2020.

SADRŽAJ

1. UVOD

Internet, kao najveći masovni medij i tehnološki fenomen modernog doba, dolaskom u svakodnevni život postao je najefikasniji alat koji podupire marketinške aktivnosti u svakom obliku ljudske djelatnosti pa tako i u turizmu. Korisnici Internetu u 2020. godini mogu pristupiti gotovo u bilo koje vrijeme i na bilo kojem mjestu. [1] Uvijek otvoreni i trenutačni pristup informacijama preko Interneta omogućio je turizmu nešto što mu nijedan drugi medij nije mogao omogućiti – dvosmjernu komunikaciju s klijentima. [2] Time je Internet postao moćan marketinški i komunikacijski medij kojim se učinkovito povezuju subjekti koji generiraju ponudu i potražnju u turizmu. Implementacijom raznih strategija za Internet marketing moguće je osigurati rast prepoznatljivosti turističke destinacije, što će eventualno dovesti do povećanja potražnje za uslugama koje destinacija nudi, a time i do povećanja konkurentnosti destinacije te veće zarade. Kao osnova za uspješnu strategiju Internet marketinga u turizmu, uz društvene mreže, svakako se ističe postojanje web stranica za odgovarajuću turističku destinaciju. Web stranice osiguravaju veću informiranost potencijalnih klijenata o ponudi turističke destinacije u kojoj razmišljaju provesti svoje vrijeme i time mogu povećati zanimanje klijenta za posjetom te turističke destinacije. Ovim radom bit će analizirana koja su sve znanja i alati potrebni za izradu web aplikacije putem tehnologije *React* za hotelski smještaj neke turističke destinacije. Na primjeru web aplikacije za informiranje klijenata o ponuđenim sadržajima u hotelskom smještaju bit će prikazana implementacija potrebnih značajki *React* tehnologije, kao i dodatnih servisa potrebnih da bi web aplikacija funkcionirala kako je zamišljeno te će biti opisano kako koristiti takvu web aplikaciju iz dvije prespektive: kao klijent i kao administrator.

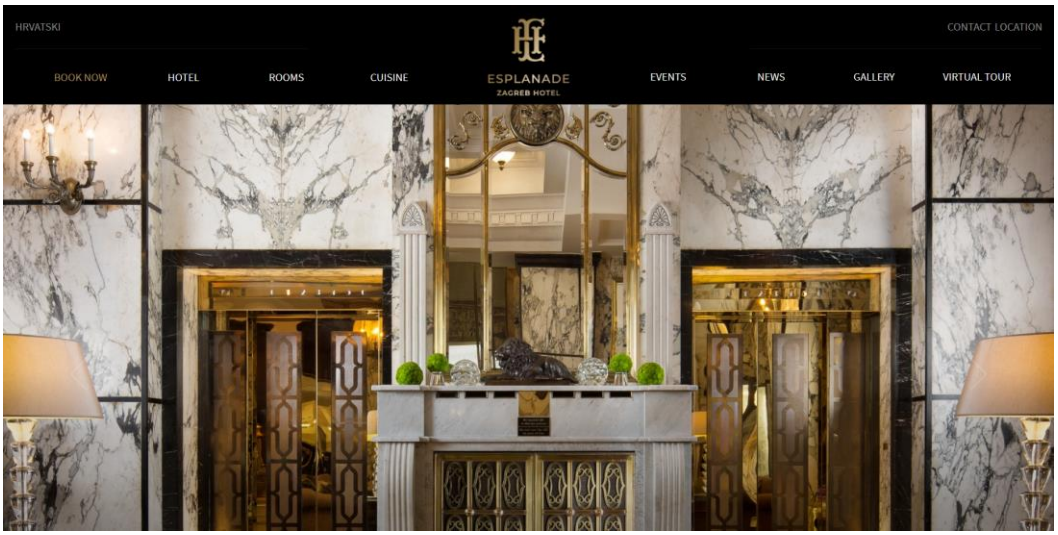
1.1. Zadatak završnog rada

Zadatak rada je putem tehnologije *React JS* izraditi web aplikaciju za informiranje klijenata o ponuđenim sadržajima u hotelskom smještaju, koja bi klijentu po registraciji u hotel omogućavala pristup web sadržaju putem kojeg bi se nudio pregled ponuđenih sadržaja tijekom njegova boravka. Informacije o sadržajima i događanjima koji će se u određenom periodu odvijati kao ponuda hotela unosio bi ovlašteni zaposlenik hotela u web sučelje koje bi bilo povezano s aplikacijom. Sustav bi se sastojao od sljedećih dijelova: baza podataka za pohranu informacija o ponuđenim sadržajima, web sučelje za uređivanje podataka u bazi za ovlaštene zaposlenike, web sučelje za pregled ponuđenih sadržaja.

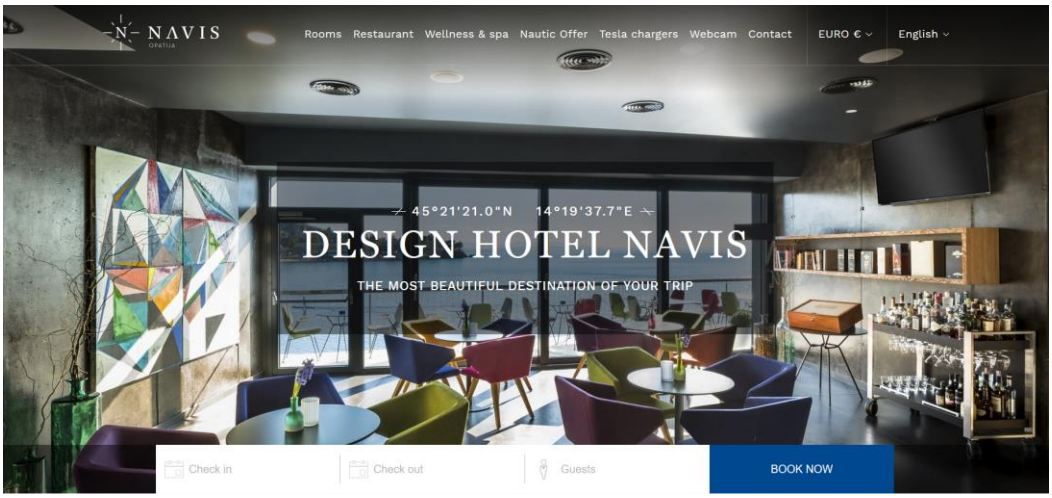
2. TEHNOLOGIJE I ALATI POTREBNI ZA IZRADU WEB APLIKACIJE

2.1. Postojeća rješenja

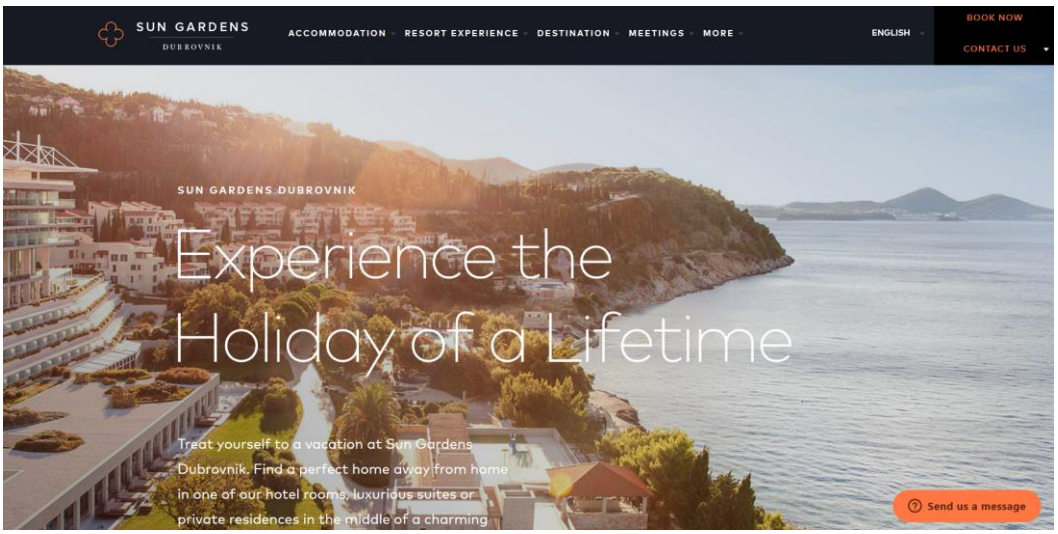
Iako je u 2020. godini važnost Interneta nikad veća, trend njegova korištenja je i dalje neupitno rastući. Prema [3], Internet polako, ali sigurno, istiskuje druge medije poput novina, a time mu marketinški potencijal raste, dok u isto vrijeme suparničkim medijima sve više opada. Kako je turizam grana ljudske djelatnosti koja je vrlo usko vezana za marketing te je ovisna o jednostavnom, razumljivom i agilnom prijenosu informacija, dolaskom Interneta u svakodnevni život bilo je za očekivati da će se turistički sadržaji početi reklamirati i putem Interneta. Danas je gotovo nemoguće zamisliti planiranje putovanja, bilo na blisko odredište ili u stranu državu, bez dostupnih informacija i turističkih ponuda na Internetu. Potencijalni posjetitelji žele brzo i jednostavno doći do informacija o smještaju i mogućim aktivnostima na određenoj destinaciji, a pri tome glavnu ulogu osim društenih mreža, imaju web stranice. Stoga, turističke web stranice moraju biti jednostavne, jasne, no privlačne i optimizirane za različite uređaje. Ukoliko nude specifične sadržaje poput noćenja, restorana ili sličnog, neophodan je i sustav za online rezervacije u realnom vremenu. Veliki broj osoba je u 2020. godini ovisan o spomenutom i nije mu niti potreban drugačiji način planiranja putovanja od onoga koji mogu pronaći online. Tijekom i nakon boravka na destinaciji, posjetitelji rado objavljuju slike i opise na društvenim mrežama te ocjenjuju sadržaje s kojima su se susreli kako bi budući posjetitelji stekli bolji dojam o destinaciji. Ukoliko su posjetitelji zadovoljni, ocjene bivaju pozitivne što privlači nove posjetitelje, a time i generira veću zaradu za vlasnika destinacije. Stoga, od početka masovnog korištenja Interneta do danas, vjerojatno svaki imalo uspješan hotel ima vlastitu web stranicu na koju su tvrtke za *web development* potrošile mnogo znanja i vremena. Slike 2.1., 2.2. i 2.3. prikazuju stranice nekih od najpoznatijih hrvatskih hotela koje dokazuju sve prethodno napisano.



Sl. 2.1. Web stranica hotela Esplanade Zagreb. [4]



Sl. 2.2. Web stranica hotela Navis Opatija. [5]



Sl. 2.3. Web stranica hotela Sun Gardens Dubrovnik .[6]

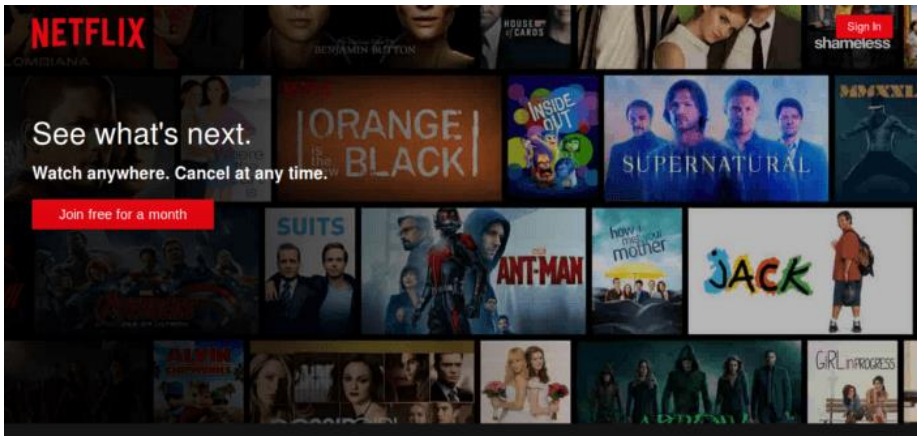
Svaka od navedenih stranica izrađena je pomoću različitih tehnologija za web programiranje. Neke od najpoznatijih tehnologija su *Ruby on Rails*, *jQuery*, *Angular*, *Vue*, *Laravel*, *Node*, no najpopularnija je *React*.

Koliko je tehnologija React zapravo važna govori činjenica kakva su sve web rješenja izrađenja pomoću nje – to su neke od globalno najposjećenijih stranica.

Sl. 2.4. Društvena mreža Facebook. [7]

Tim zadužen za izradu i održavanje najpoznatije društvene mreže na svijetu – *Facebook-a* (slika 2.4.) – stvorio je i tehnologiju *React*. Iako je *React* stvoren kako bi pomogao pri implementaciji novih verzija *Facebook-a*, *Facebook* ga danas koristi samo djelomično jer mu je potrebna nekolicina različitih tehnologija kako bi sadržavao sve potrebne funkcionalnosti. *Facebook* i slične velike aplikacije (tj. društvene mreže) imaju veliki broj aktivnih korisnika (prema [8] *Facebook* ih trenutno ima 2.8 milijardi) kojima je potrebno omogućiti registraciju, prijavu i odjavu, odnosno autentikaciju i autorizaciju. Svaki korisnik želi učitavati vlastiti sadržaj na stranicu s ciljem da ga podjeli s drugima, stoga je potrebno rješenje i za kontrolu velikih baza podataka. Stranicama s velikim količinama sadržaja kao što su društvene mreže potrebni su i alati za optimiziranje učitavanja tog sadržaja. Prema [14] *React* je samo *JavaScript* biblioteka orijentiranja na stvaranje klijentskih sučelja, odnosno *frontend*[1] dijelova aplikacija, stoga ne pruža mogućnost direktnog upravljanja bazom podataka ili autentikaciju. Zbog spomenutog, *React* se obično koristi u kombinaciji s drugim tehnologijama kao što su *Firebase*, *Node.js*, *Express.js* i druge.

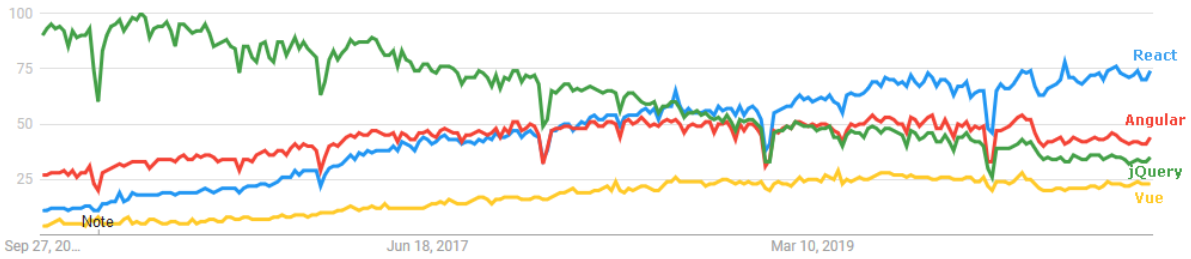
[1] Frontend – grafičko sučelje koje se prikazuje korisniku i s kojim korisnik može interaktirati, obično nastalo korištenjem HTML-a, CSS-a i JavaScript-a



Sl. 2.5. Streaming servis Netflix. [9]

Osim *Facebooka,* treba spomenuti i najpoznatiji *streaming* servis – *Netflix* (slika 2.5.), koji je u protekih nekoliko godina revolucionizirao gledanje filmova i TV-serija, a i za njegovu implementaciju zaslužna je tehnologija *React*.

Prema navedenome lako se može primjetiti da je *React* jedan od najpopularnijih alata za izradu svih vrsta web aplikacija te su *React developeri* iznimno traženi na tržištu programera. Ovu tezu potvrđuju podaci *Google-a* [8] o pretraživanju pojma *React* u odnosu na slične tehnologije kroz posljednjih 5 godina – prikazani na slici 2.6. Zapravo, jedina prava konkurencija *React-u* na tržištu trenutno je *Angular*. Također, zajednica *React developera* na Internetu je prilično razvijena, što olakšava njegovo učenje i svakodnevno korištenje.



Sl. 2.6. Količina pretraživanja pojma React i njegovih konkurenata u Google pretraživaču. [10]

2.2. Potrebna znanja za korištenje React-a

React je tehnologija koja ujedinjuje nekoliko postojećih samostalnih tehnologija u cjelinu i time omogućava stvaranje web aplikacija s mnoštvom funkcionalnosti (prikazivanje sadržaja, interakcija sa stranicom i drugim). Spomenute tehnologije su *Hyper Text Markup Language* (*HTML*) i *JavaScript*. U ovom poglavlju objašnjene su osnovne značajke *HTML-a* i *JavaScript-a*, kao i značajke *Cascading Style Sheets-a* (*CSS-a*) te *Document Object Model-a*. Poznavanje ovih tehnologija uvelike olakšava razumijevanje glavnih koncepata *React-a*.

2.2.1. Hyper Text Markup Language – HTML

Prema [11, 12, 13], *HTML* (engl. *Hyper Text Markup Language*) je standardni opisni jezik za kreiranje *HTML* dokumenata, tj. web stranica. Pomoću njega se opisuje struktura web stranice korištenjem oznaka (engl. *tags*). Oznake tvore elemente – gradivne jedinice *HTML* dokumenata. *HTML* kod govori korisničkom pregledniku kako prikazati sadržaj korisniku, no ne prikazuje oznake, već ih koristi za prikaz sadržaja unutar njih.

Linija ***Kod***

```
1:      <html>
2:        <head>
3:          <title>Page Title</title>
4:        </head>
5:        <body></body>
6:      </html>
```

Sl. 2.8. Osnovna struktura HTML koda.

Na slici 2.8. prikazana je osnovna struktura *HTML* koda koja se sastoji od sljedećih elemenata:

- **<html></html>** - element okružuje sav sadržaj stranice
- **<head></head>** - element u kojeg stavljamo informacije o web stranici koje se neće prikazati na monitoru korisnika kao sadržaj stranice
- **<title></title>** - element postavlja ime dokumenta u kartici preglednika
- **<body></body>** - element sadrži sve što se želi prikazati korisniku koji posjeti web stranicu – tekst, slike, videozapisi i slično.

2.2.2. Cascading Style Sheets – CSS

Prema [11, 12, 13], *CSS* (*Cascading Style Sheets*), odnosno kaskadni stilovi, jednostavan su mehanizam za dodavanje stilova (fontova, boja, razmaka između elemenata i sličnog) *HTML* elementima. Korištenjem *CSS-a* moguće je odvojiti dizajn od same strukture podataka na stranici.

HTML kod postaje pregledniji i manji što znači da ga je puno lakše kontrolirati i ponovo koristiti, a pritom je moguće jednostavno mijenjati dizajn stranice.

Struktura i sintaksa *CSS-a*:

- selektor: određuje element na koji se stilsko pravilo odnosi
- deklaracija: određuje kako izgleda sadržaj opisan *CSS-om*

Za definiranje pravila koristi se set posebnih znakova. Sintaksa za stilska pravila ima sljedeći uzorak:

selektor { deklaracija; }

Deklaracija se dijeli na dva dijela:

- svojstvo (engl. *property*) – ono što se želi promijeniti na određenom elementu
- vrijednost (engl. *value)* – na koji način se želi promijeniti određeno svojstvo

selektor { svojstvo: vrijednost; }

Na slici 2.9. prikazani su primjeri kako koristiti CSS na elementima <*body*> i <*h1*>.

Linija ***Kod***

```
1:      body { background-color: teal; }
2:      h1 { font-size: 35px; }
```

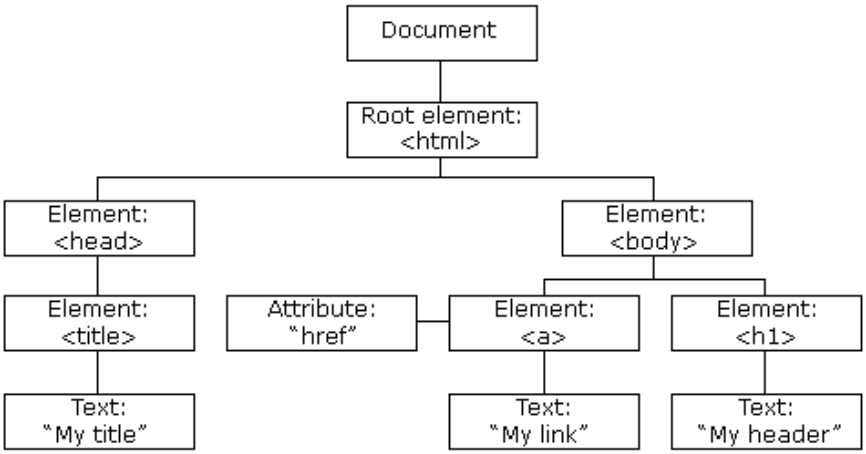
Sl. 2.9. Jednostavna primjena CSS-a.

2.2.3. Document Object Model – DOM

Prema [13], *DOM* (engl. *Document Object Model*) predstavlja dokument u memoriji računala koji kombinira sadržaj (*HTML*) sa stilom (*CSS-om*). Kada preglednik prikazuje dokument, obrađuje ga u dva koraka:

1. preglednik povezuje *HTML* i *CSS* te ih pretvara u *DOM*
2. preglednik prikazuje sadržaj *DOM-a.*

DOM ima strukturu stabla (slika 2.10.), gdje svaki element, atribut ili dio teksta postaje *DOM* čvor. Čvorovi su definirani po svojim odnosima s drugim *DOM* čvorovima. Neki elementi su roditeljski čvorovi, a čvorovi na razini niže su djeca koja također mogu imati djecu. Razumijevanje *DOM-a* pomaže prilikom dizajniranja i održavanja *CSS-a* jer je *DOM* mjesto gdje se susreću *CSS* i *HTML*.



Sl. 2.10. Vizualizacija strukture DOM-a za jednostavan primjer web stranice.

2.2.4. JavaScript

Prema [12, 14], *JavaScript* (ili samo *JS*) je skriptni programski jezik i uz *HTML* i *CSS* temeljna je tehnologija *World Wide Web-a*. *JavaScript* je esencijalan dio web aplikacija jer omogućava interakciju između korisnika i web aplikacije. Velika većina (oko 95%) stranica koristi *JavaScript* za omogućavanje interakcija korisniku, a svi poznati web preglednici imaju poseban *JavaScript engine*[2].

Kao programski jezik koji ima nekoliko paradigmi, *JavaScript* podržava funkcionalno programiranje, programiranje pogonjeno događajima i imperativno programiranje. *JavaScript-om* se pišu aplikacijska programska sučelja[3] (engl. *API*) koji podržavaju rad s tekstom, slikama, standardnim strukturama podataka i slično. Ipak, *JavaScript* ne sadrži usluge ulaza i izlaza ili usluge pohrane – nešto što inače nude web preglednici.

Osim korištenja u web preglednicima (klijentska strana), *JavaScript* je ugrađen i koristi se na serverskoj strani. Najčešće putem tehnologije *Node.js*.

Primjeri korištenja programskoj jezika *JavaScript*:

- učitavanje novog sadržaja bez osvježavanja stranice
- animacije sadržaja
- interaktivan sadržaj (igre, slike, videozapisi)

[2] *JavaScript engine* – program koji pokreće *JavaScript* kod
[3] Aplikacijsko programsko sučelje (engl. *API - application programming interface*) – sučelje koje definira interakciju između dva *software-a*

- provjera i potvrda vrijednosti pri ispunjavanju web upitnika ili obrazaca (npr. obrasci za prijavu)
- slanje informacija o korisnikovom ponašanju (analitika, personalizacija, reklame)

Na slici 2.11. prikazan je jednostavan primjer kako koristiti *JavaScript* za stvaranje interaktivnog sadržaja na web stranici. Sadržaj stranice je tekst koji se mijenja klikom na gumb „*Try it*“. Gumb u pozadini poziva *JavaScript* funkciju koja mijenja tekst prema tome kako je unutar nje zadano, bez da osvježavamo stranicu.

Linija ***Kod***

```
1:      <html>
2:        <head>
3:          <script>
4:            function myFunction(){
5:              document.getElementById("demo").innerHTML = "Transform"!;
6:            }
7:          </script>
8:        </head>
9:        <body>
10:         <h2>JavaScript in Head</h2>
11:         <p id="demo">A Paragraph.</p>
12:         <button type="button" onclick="myFunction()">Try it</button>
13:       </body>
14:     </html>
```

Sl. 2.11. Primjer jednostavnog JavaScript koda.

Kada se govori o *JavaScript-u* treba spomenuti i standard, odnosno sintaksu pisanja *JavaScript* koda, koja se svakih nekoliko godina ažurira i prilagođava novim uvjetima. *ES6* standard je trenutno najkorišteniji i osnovna je polazišna točka kada se uči *JavaScript* kodiranje. *ES6* je donio mnogo novih značajki i učinio je dotadašnju sintaksu *JavaScript-a* puno lakšom za čitanje.

Osim *ES6* standarda koji definira *JavaScript* sintaksu, postoje i *JavaScript* biblioteke i okviri (engl. *frameworks*) koji sadrže modificirani *JavaScript* kod prilagođen za različita korištenja. Upravo je *React*, jedna od biblioteka *JavaScript-a*, a ostale poznate biblioteke i okviri su *jQuery*, *Node.js*, *Angular*, *Vue.js* i druge.

2.3. React

Prema [15], *React* (*React.js* ili *React JS*) je *JavaScript* biblioteka otvorenog koda za stvaranje klijentskih sučelja. Stvorio ju je i održava ju *Facebook* zajedno sa zajednicom individualnih

developera i kompanija. *React* se koristi kao baza u razvoju web i mobilnih aplikacija. Ipak, *React-u* je jedina zadaća prevesti kod u *Document Object Model*, pa se pri stvaranju aplikacija *React-om* često moraju koristiti srodne biblioteke i tehnologije za npr. kontroliranje stanja aplikacije ili navigaciju - *Redux* i *React Router* su najčešći primjeri.

2.3.1. Glavni koncepti React-a

Glavna ideja *React-a* je da se pri prevođenju koda u *DOM* kombinira nekoliko funkcionalnosti: prikazivanje sadržaja stranice, reagiranje na događaje (interakcija sa stranicom), održavanje stanja stranice i slično. Umjesto razdvajanja koda u posebne dokumente, *React* je omogućio pisanje istog u komponente koje mogu sadržavati više funkcionalnosti. U nastavku, objašnjeni su glavni koncepti koji ostvaruju glavnu ideju *React-a.*

- **JSX**

JSX sintaksa nije niti *string*[4], niti *HTML* kod. Ona je kombinacija *JavaScript-a* i *HTML-a* kojom se opisuje izgled korisničkog sučelja (slika 2.13.). Umjesto razdvajanja pisanja logike programa i vizualnog dizajna u različite dokumente, *React* podržava pisanje koda koji ih kombinira. *JSX-om* se kreiraju *React* elementi koji se prevode u *ReactDOM*.

Linija ***Kod***

```
1:      const name = 'Josh Perez';
2:      const element = <h1>Hello, {name}!</h1>;
3:      ReactDOM.render{
4:        element,
5:        document.getElementById('root')
6:      );
```

Sl. 2.13. Primjer kombinacije JavaScript-a i HTML-a

- **Prikazivanje elemenata (Rendering Elements)**

Linija ***Kod***

```
1:      const element = <h1>Hello world!</h1>;
```

Sl. 2.14. Primjer osnovnog React elementa

[4] String – tip podatka u programiranju, niz znakova

Na slici 2.14. prikazana je najmanja gradivna jedinica *React* aplikacije - element. Elementom je opisano ono što želimo vidjeti na ekranu monitora. Jednom kada je element stvoren, njegovu djecu i atribute nemoguće je promijeniti. Element je kao jedan okvir (engl. *frame*) u filmu – predstavlja korisničko sučelje u određenoj točki vremena, prema tome, glavni izazov je ažurirati ga.

Slika 2.15. prikazuje aplikaciju koja predstavlja sat. Svake sekunde poziva se *tick()* funkcija koja poziva *ReactDOM.render()* funkciju koja će ažurirati stranicu. No, da bi uštedio memoriju, *React* ažurira samo ono što se promijenilo u odnosu na zadnji poziv *ReactDOM.render()* funkcije. Prema tome, iako na svaki poziv *tick()* funkcije se stvara novo sučelje, *React* ažurira samo sadržaj unutar *<h2>* elementa.

Linija ***Kod***

```
1:      function tick() {
2:        const element = (
3:          <div>
4:            <h1>Hello, world!</h1>
5:            <h2>It is {new Date().toLocaleTimeString()}.</h2>
6:          </div>
7:        );
8:      ReactDOM.render(element,
9:        document.getElementById('root'));
10:     }
11:     setInterval(tick,1000);
```

Sl. 2.15. Primjer jednostavne aplikacije sata

- **Komponente i svojstva (Components and Props)**

Komponente dozvoljavaju razdvajanje klijentskog sučelja u samostalne cjeline spremne za višekratnu i raznoliku upotrebu. Konceptualno, one su *JavaScript* funkcije ili klase. Funkcijske komponente (slika 2.16.) primaju ulazne podatke – svojstva (engl. *props*) i vraćaju *React* elemente koji opisuju sadržaj koji će se prikazati na ekranu monitora. Klasne komponente (slika 2.17.) imaju *render()* (engl. *render* – prikazati) funkciju u koje se stavlja sadržaj koji se želi prikazati, a mogu imati i vlastito stanje. O stanju kao jednom od koncepata *React-a* više u nastavku ovog poglavlja.

Linija ***Kod***

```
1:      function Welcome(props){
2:        return <h1>Hello, {props.name}</h1>;
3:      }
```

Sl. 2.16. Primjer funkcijske komponente.

Linija ***Kod***

```
1:      class Welcome extends React.Component {
2:        render(){
3:          return <h1>Hello, {this.props.name}</h1>;
4:        }
5:      }
```

Sl. 2.17. Primjer klasne komponente.

Osim elemenata koji sadržavaju *HTML* oznake, u *React-u* moguće je koristi i korisnički definirane komponente kao oznake za element (slika 2.18.).

Linija ***Kod***

```
1:      function Welcome(props) {
2:        return <h1>Hello, {props.name}</h1>;
3:      }
4:      const element = <Welcome name="Sara" />;
5:      ReactDOM.render(
6:        element,
7:        document.getElementById('root')
8:      );
```

Sl. 2.18. Primjer korištenja korisnički definirane komponente.

- **Stanje i Lifecycle metode (State and Lifecycle Methods)**

Ako se pobliže pogleda aplikacija sata sa slike 2.15., mogu se primjetiti njeni nedostatci. Činjenica je da takav sat treba nekakvu vanjsku funkciju koja otkucava sekunde i time ažurira sat. Idealno, komponenta bi trebala biti samoodživa te bi trebala biti jednostavna za ponovno korištenje. Te dvije funkcionalnosti se postižu dodavanjem stanja u klasnu komponentu. Stanje je slično svojstvu, no ono je privatno i potpuno ga kontrolira komponenta u kojem se nalazi. Prema tome, na slici 2.19. prikazana je implementacija nove komponente Clock. Funkcija *toLocaleTimeString()* svake sekunde vraća podatak o lokalnom vremenu i postavlja ga u stanje klase Clock. Takva komponenta spremna je za ponovno korištenje bez ikakvih dodatnih vanjskih funkcija.

Linija ***Kod***

```
1:      class Clock extends React.Component {
2:        constructor(props) {
3:          super(props);
4:          this.state = {date: new Date()};
5:        }
6:        render() {
7:          return(
8:            <div>
9:              <h1>Hello, world!</h1>
10:             <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
11:           </div>
12:         );
13:       }
14:     }
15:     ReactDOM.render(
16:       <Clock />,
17:       document.getElementById('root')
18:     );
```

Sl. 2.19. Implementacija komponente Clock.

U aplikacijama s mnoštvom komponenata, jako je važno osloboditi memoriju koja je do tada bila na raspolaganju komponentama koje više nisu potrebne. Za to služe dvije funkcije zvane „metode životnog vijeka“ (engl. *lifecycle methods*): *componentDidMount()* metoda se pokreće kad je komponenta prevedena u *DOM*, a *componentWillUnmount()* metoda se pokreće kada komponenta više nije potrebna. U *lifecycle* metode korisnik može dodati i proizvoljne funkcionalnosti, no primarna zadaća ovih metoda bi trebala biti kontrola memorije.

- **Rukovanje događajima (Handling Events)**

Rukovanje događajima kod *React* elemenata je vrlo slično kao i kod *HTML* elemenata. Postoje samo neke sintaksne razlike: *React* koristi *camelCase* umjesto *lowercase* imena, a umjesto da ime funkcije koja se poziva kao reakcija na događaj pišemo kao *string* – unutar navodnika, pišemo ga unutar vitičastih zagrada kao *JSX* element. Na slici 2.20. prikazane su razlike između *HTML* rukovatelja događajima i *React* rukovatelja događajima. U ovom slučaju na pritisak gumba (engl. *button*) u oba slučaja pozivamo funkciju *activateLasers()*.

Linija ***Kod***

```
1:      <button onclick="activateLasers()">
2:        Activate Lasers
3:      </button>
…
1:      <button onClick={activateLasers}>
2:        Activate Lasers
3:      </button>
```

Sl. 2.20. HTML i React rukovatelji događajima.

- **Uvjetno prikazivanje (Conditional Rendering)**

U *React-u* je moguće stvoriti posebne komponente koje se ponašaju isključivo po korisnikovim uputstvima. Preciznije, ovisno o svojstvima kojima im korisnik preda, komponente će prikazivati samo dio svog sadržaja. Na slici 2.21. primjer je uvjetnog prikazivanja. Ovisno o tome što je predano komponenti *Greeting* kao svojstvo, ona će prikazati *UserGreeting* ili *GuestGreeting* komponentu.

Linija ***Kod***

```
1:      function UserGreeting(props){
2:        return <h1>Welcome back!</h1>;
3:      }
4:
5:      function GuestGreeting(props){
6:        return <h1>Please sign up!</h1>;
7:      }
8:
9:      function Greeting(props){
10:       const isLoggedIn = props.isLoggedIn;
11:       if(isLoggedIn){
12:         return <UserGreeting />;
13:       }
14:       return <GuestGreeting />;
15:     }
16:
17:     ReactDOM.render(
18:       <Greeting isLoggedIn={false} />,    //or true
19:       document.getElementById('root')
20:     );
```

Sl. 2.21. Uvjetno prikazivanje.

- **React Hooks**

React Hooks (engl. *hook* – kuka) su relativno nov, ali jedan od trenutno najpopularnijih *React* koncepata koji omogućava korištenje stanja i *lifecycle* metoda unutar funkcijskih komponenti. Do pojave *Hooks-a,* funkcijske komponente nisu mogle imati vlastito stanje i *lifecycle* metode. Time je proces refaktoriranja funkcijskih komponenata u klasne komponente pao u drugi plan – moguće je napisati cijelu aplikaciju bez korištenja klasnih komponenti što pojednostavljuje *React-ov* kompliciran način korištenja apstraktnih klasa.

Osim spomenutih koncepata, u *Reactu* postoji još mnoštvo drugih, a kako je *React* relativno nova platforma, redovito se širi i unaprjeđuje. Neki napredniji koncepti bit će obrađeni u programskom rješenju problema ovog rada.

2.4. Firebase

Prema [16], *Firebase* je platforma koja pomaže mobilnim i web *developerima* razvijati aplikacije koje koriste baze podataka. Ono je *Backend-as-a-Service* (*BaaS*) – pruža *API* usluge za web i mobilne aplikacije, no jednostavnije rečeno, *Firebase* dozvoljava *developerima* da se fokusiraju na kreiranje klijentskog sučelja. *Developer* se pri tome ne mora baviti serverima, pisati vlastite *API-je* i sl. *Firebase* je u tom slučaju sve od navedenog: server, *API*, baza podataka – gotovo cijela pozadina aplikacije.

2.4.1. Firebase servisi

- **Realtime Database (Baza podataka u realnom vremenu)**

Ovaj servis pruža *developerima API* koji sinkronizira i sprema podatke korištene u aplikaciji na *Firebase Cloud*. Pri tome, također, pruža sve potrebne biblioteke za njezinu integraciju u *Android, iOS, JavaScript, Java,* C/C++/C#, *Swift* i *Node.js* aplikacije. *Realtime Database* je također povezana s nekoliko *JavaScript* biblioteka kao što su *React*, *Angular*, *Ember* i druge.

- **Authentication (Autentikacija)**

Authentication servis pruža mogućnost autentikacije korisnika koristeći samo kod na klijentskoj strani. Podržava autentikaciju putem društvenih mreža kao što su *Facebook*, *Twitter*, *Google* i *Github*. Također, sadrži i sustav upravljanja korisnicima koji *developeri* ne moraju ručno implementirati.

- **Storage (Pohrana)**

Storage servis pruža siguran sustav učitavanja i preuzimanja datoteka na *Firebase* (*Google*) *Cloud*. Putem njega aplikacije mogu slati i preuzimati slike, audio, video i slične korisnički kreirane datoteke.

- **Ostale usluge**

Cloud Messaging, Cloud Firestore, Google Analytics, Cloud Functions, Hosting, Machine Learning Kit, Crashlytics

3. PROGRAMSKO RJEŠENJE WEB APLIKACIJE

Aplikacija dana ovim radom dobar je primjer realizacije svih koncepata koji su pojašnjeni u prethodnim poglavljima. Putem aplikacije bit će prikazivani hotelski sadržaji. Aplikacija će imati funkcionalnost prijave i odjave za ovlaštene korisnike. Ovlašteni korisnici će moći dodavati novi sadržaje pritom spremajući ih u bazu podataka. Na takvu aplikaciju bez problema će se moći nadograditi brojne druge funkcionalnosti poput sustava za rezervacije.

3.1. Stvaranje polazne React aplikacije i modularna organizacija

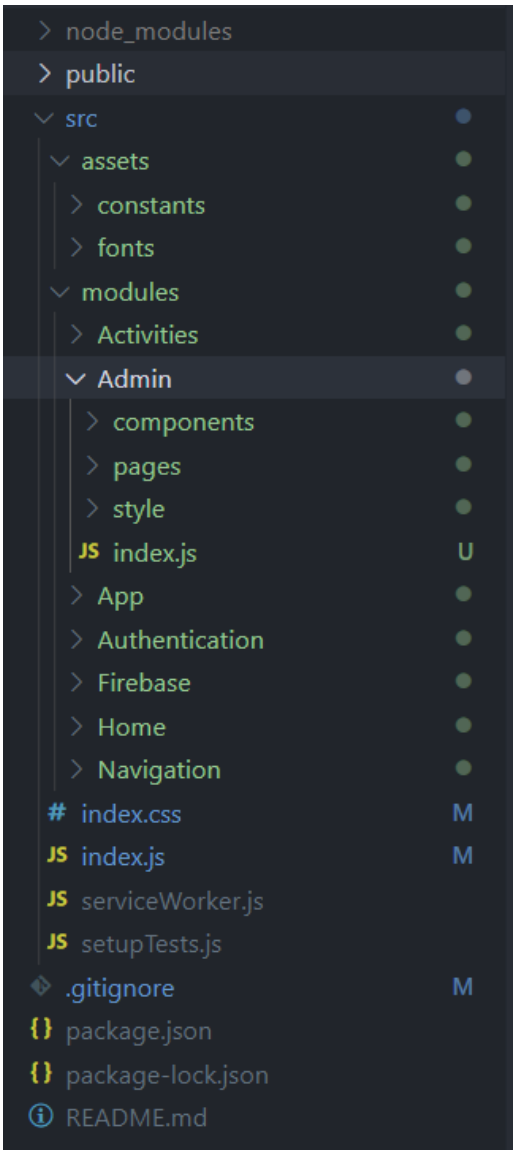
Prema [15], startna aplikacija stvorena je pomoću službenog *Facebook-ovog* predloška projekta koji se zove *create-react-app*. Za to je potrebno preuzeti i instalirati besplatan servis zvan *Node.js* na čijoj službenoj stranici imaju i instrukcije za preuzimanje i instaliranje. Sav dio pisanja koda odrađen je u službenom *Microsoft-ovom* programu za uređivanje koji se zove *Visual Studio Code* koji se besplatno može preuzeti na *Microsoft-ovim* službenim stranicama.

Nakon što su instalirani svi potrebne alati, potrebno je u *command* liniju *Visual Studio Code-a* upisati naredbu za stvaraje polazne React aplikacije (slika 3.1.).

```
C:\Users\Korisnik\Desktop>npm create-react-app react-hotel-website
```

Sl. 3.1. Naredba za stvaranje polazne React aplikacije.

Nakon što se aplikacija instalira, s lijeve strane *Visual Studio Code-a* otvorit će se i njezina organizacija koju je potrebno izmijeniti prema slici 3.2. Na ovakav tip organizacije koda *developeri* se odlučuju zbog lakšeg održavanja koda i njegove nadogradnje te brže i efikasnije suradnje unutar tima. Na ovako organiziranom projektu, više članova tima može raditi istovremeno bez da smetaju jedni drugima.



Sl. 3.2. Modularna organizacija React aplikacije.

Direktorij *assets* sadrži resurse (fontove, slike, konstante) koji se koriste na razini aplikacije. Direktorij *modules* sadrži module (komponente okupljene u jednu cjelinu). Svaki modul sadrži direktorij *components* u koji su spremljene manje gradivne jedinice - komponente, direktorij *pages* koji sadrži cijelu jednu stranicu koja se prikazuje na ekran monitora i direktorij *style* u kojem se nalazi *CSS* kod za stranicu koja se prikazuje. U dokumentu *index.js,* ključnim riječima *import* („uvoz“) i *export* („izvoz“) omogućeno je komponentama da se ponovo koriste u nekom drugom modulu. Osim korjenskog (engl. *root) index.js* dokumenta koji povezuje sve ostale dokumente u jednu cjelinu, svaka *components* mapa sadrži vlastiti *index.js* dokument.

3.2. React Router i navigacija kroz stranice

Kako je riječ o izradi velike aplikacije dobro je imati nekoliko stranica (npr. *Home* stranicu, *Admin* stranicu, stranicu za prijavu ovlaštenih korisnika) i tako podijeliti aplikaciju na kroz više putanja (engl. *URL*) (npr. */home, /admin*). Kroz te putanje će korisnici navigirati stranicama (slika 3.3.).

Linija ***Kod***

```
1:      export const HOME = '/';
2:      export const ACTIVITIES = '/activities';
3:      export const SINGLE_ACTIVITY = '/activities/:url';
4:      export const SIGN_IN = '/signin';
5:      export const ADMIN = '/admin';
```

Sl. 3.3. String konstante koje će biti korištene kao putanje za navigaciju kroz stranice.

Kao što se može vidjeti, aplikacija će imati početnu *Home* stranicu, *Activities* stranicu na kojoj će biti moguće pregledavati sadržaje koje hotel nudi, svaki *Activity* će imati svoju stranicu, *Sign In* stranicu preko koje će se ovlašteni zaposlenici hotela prijavljivati i *Admin* stranicu gdje će zaposlenici uređivati sadržaje hotela.

Zatim je potrebno instalirati posebnu *React* biblioteku koja se zove *React Router* upisujući naredbu "*npm install react-router-dom*" u *command* liniju *Visual Studio Code-a*. Kao što joj ime i govori, pomoću nje je moguće navigirati putanjama (engl. *routes*) navedenima maloprije.

Implementacijom potrebnog koda za *React Router* (slika 3.4.), kada se klikne na poveznicu u navigacijskoj traci, prikazati će se stranica koja odgovara njezinoj putanji.

Linija ***Kod***

```
1:      const AppBase() => (
2:        <Router>
3:          <Layout>
4:            <Switch>
5:             <Route exact path={ROUTES.HOME} component={HomePage} />
6:             <Route path={ROUTES.ACTIVITIES} component={ActivitiesPage} />
7:             <Route path={ROUTES.SIGN_IN} component={SignInPage} />
8:             <Route path={ROUTES.ADMIN} component={AdminPage} />
9:             <Route path={'/:url'} component={SingleActivityPage} />
10:           <Switch>
11:         <Layout>
12:       <Router>
13:     );
```

Sl. 3.4. Implementacija glavnog dijela koda React Router-a.

3.3. Implementacija Firebase-a

Prema [16], za početak potrebno je obaviti registraciju na službenoj *Firebase* web stranici. Nakon registracije potrebno je kreirati i inicijalizirati novi projekt. Zatim pronaći informacije o konfiguraciji projekta koje će se koristiti za njegovu implementaciju u *React* (slika 3.5.).

```
<script src="https://www.gstatic.com/firebasejs/4.7.0/firebase.js"></script>
<script>
  // Initialize Firebase
  var config = {
    apiKey: my-api-key,
    authDomain: "my-app-name.firebaseapp.com",
    databaseURL: "https://my-app-name.firebaseio.com",
    projectId: "my-app-name",
    storageBucket: "my-app-name.appspot.com",
    messagingSenderId: "999999999999"
  };
  firebase.initializeApp(config);
</script>
```

COPY

Sl. 3.5. Primjer informacija o projektu.

Unutar *Visual Studio Codea* potrebno je instalirati *Firebase* koriteći naredbu *"npm install firebase"*, a zatim implementirati Firebase u posebnom dokumentu (modulu) koji će izgledati kao modul na slici 3.6. Unutar *config.json* dokumenta se nalaze informacije o projektu sa slike 3.5.

Linija ***Kod***

```
1:      import app from 'firebase/app';
2:      import config from './cofig.json';
3:      class Firebase {
4:        constructor(){
5:          app.initializeApp(config);
6:        }
7:      }
8:      export default Firebase;
```

Sl. 3.6. Implementacija Firebase klase.

Kako je nužno da unutar cijele aplikacije bude korištena samo jedna instanca *Firebase* klase, tj. kako *Firebase* klasa ne bi bila iznova inicijalizirana pri svakom njenom korištenju, dobro je koristiti jedan od naprednih *React* koncepata koji se zove *Context*. *Context* pruža jednu globalnu instancu neke klase na najvišoj razini aplikacije. Ostale komponente u cijelom stablu aplikacije mogu koristiti podatke te klase, npr. dizajn, jezik aplikacije, podatke o trenutno prijavljenom korisniku i slično.

Linija ***Kod***

```
1:      import React from 'react';
2:      const FirebaseContext = React.createContext(null);
3:      export default Firebase Context;
…
1:      ReactDOM.render(
2:        <FirebaseContext.Provider value={new Firebase()}>
3:          <App />
4:        </FirebaseContext.Provider>,
5:        document.getElementById('root'),
6:      };
```

Sl. 3.7. Implementacija Firebase Context-a.

Firebase Context se implementira (slika 3.7.) na način da se cijela aplikacija "uokviri" *FirebaseContext.Provider-om* – komponentom koja aplikaciji daje mogućnost da se kroz aplikaciju koristi jedna instanca potrebnih *Firebase* usluga (engl. *provider* – davatelj usluga).

Linija ***Kod***

```
1:      import React from 'react';
2:      import { FirebaseContext } from '../Firebase';
3:      const SomeComponent = () => (
4:        <FirebaseContext.Consumer>
5:          {firebase => {
6:            return <div>I have acces to Firebase</div>;
7:          }}
8:        </FirebaseContext.Consumer>
9:      );
10:     export default SomeComponent;
```

Sl. 3.8. Primjer upotrebe Context-a u drugoj komponenti.

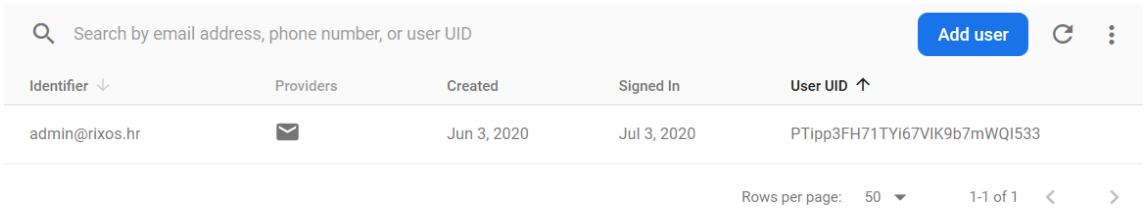
Na slici 3.8. prikazano je kako bilo koja komponenta unutar aplikacije može iskoristiti *Context.* Komponenta (u ovom slučaju *SomeComponent*) treba svoj sadržaj uokviriti *FirebaseContext.Consumer* komponentom (engl. *consumer* – potrošač) čime se označava da će komponenta koristiti podatke *Firebase* usluga.

3.4. Autentikacija, autorizacija

Pojmovi autentikacija i autorizacija često se koriste pri kontroli pristupa web stranicama. Autentikacija je provjera postoji li registrirani korisnik s danom kombinacijom korisničkog imena i lozinke, a autorizacija je davanje pristupa dijelu aplikacije samo određenim korisnicima. U

slučaju ove aplikacije, samo ovlaštenim zaposlenicima hotela koji će imati vlastito korisničko ime i lozinku bit će omogućeno pristupiti stranici na kojoj će se uređivati sadržaji o ponudi hotela.

Firebase pruža mogućnost implementacije autentikacije i autorizacije. Moguće je birati kojim načinom će se ovlašteni korisnici prijavljivati u sustav (kako je ranije spomenuto *Firebase* nudi prijave i putem društvenih mreža), no odabrana je prijava elektroničkom poštom i lozinkom. Ovlašteni korisnici se neće moći sami registrirati, nego će vlasnik aplikacije unositi njihovu elektroničku poštu i lozinku putem *Firebase* konzole kojom se pristupa na *Firebase-ovim* službenim web stranicama (slika 3.9.). Na strani *React* aplikacije, potrebno je implementirati metode koje dozvoljavaju korištenje *Firebase Authenthication* usluge, metode koje omogućuju prijavu i odjavu, te potrebno je napraviti obrazac kojeg će ovlašteni korisnik ispuniti za prijavu u sustav.



Sl. 3.9. Sučelje za dodavanje i brisanje ovlaštenih korisnika u Firebase konzoli.

Iako je implementacijom autentikacije omogućeno ovlaštenim zaposlenicima prijavu u sustav, oni neovlašteni, neprijavljeni korisnici i dalje mogu vidjeti sve stranice koje postoje u sustavu aplikacije. Kako bi to bilo onemogućeno, koristi se drugi dio *Firebase Authentication* usluge, a to je autorizacija. Stranica u kojoj će ovlašteni zaposlenici moći uređivati sadržaje hotela će biti sakrivena za neprijavljene korisnike. Osim toga, implementacijom autentikacije i autorizacije, navigacijske komponente će se mijenjati sukladno tome je li korisnik prijavljen u sustav ili ne, te će nakon prijave ili odjave, korisnik biti ispravno preusmjeren na odgovarajuće stranice.

3.5. Baza podataka i administratorsko sučelje

Nakon što samo ovlašteni korisnici imaju pristup stranici za unos ponude vezane za hotelski sadržaj i aktivnosti (*Admin* stranici), potrebno je implementirati način na koji će oni unositi te aktivnosti i kako će se one zapisivati u bazu podataka koju pruža *Firebase*. Prvo, treba definirati što zaposlenik želi unositi kao sadržaje koje hotel nudi. To će biti nekakva aktivnost (npr. obilazak grada, obližnjih znamenitosti, aktivnosti za zabavu djece, sportske aktivnosti) ili nekakva ponuda (npr. hotelski restoran i bar). Svaka aktivnost imati će naslov, svoj *URL*, kratak i dug opis, glavnu

i pozadinsku sliku, galeriju slika, dodatke, te će zaposlenik moći odabrati na koji način će se ona moći prikazati u listi aktivnost na kojoj će ih korisnik stranice pregledavati. Informacije će biti unošene preko jednostavnih obrazaca za unos teksta i učitavanje slika, a svaki obrazac imat će metodu koja reagira kada smo završili sa unosom. Ona će zapisati informaciju unutar stanja komponente u kojoj se obrasci nalaze. Kada je gotov unos svih informacija i pritisnut gumb za učitavanje, aktivira se metoda koja uzima podatke iz stanja glavne komponente prikazana na slici 3.10. Ona upisuje podatke u *Firebase Realtime Database*, a zatim postavlja stanje komponente na početno da bi ponovo bio omogućen unos nove aktivnosti. Također, potrebno je implementirati i metodu koja će na pritisak gumba obrisati jednu od prije unesenih aktivnosti ako je ona postala nepotrebna.

Linija ***Kod***

```
1:      onCreateActivity = (event) => {
2:        event.preventDefault();
3:        this.props.firebase.activities().push({
4:          title: this.state.title,
5:          url: this.state.url,
6:          description: this.state.description,
7:          …
8:        });
9:
10:       this.setState({
11:         title:'',
12:         url:'',
13:         description:'',
14:         …
15:       });
16:     };
```

Sl. 3.10. Metoda za učitavanje stanja u Firebase.

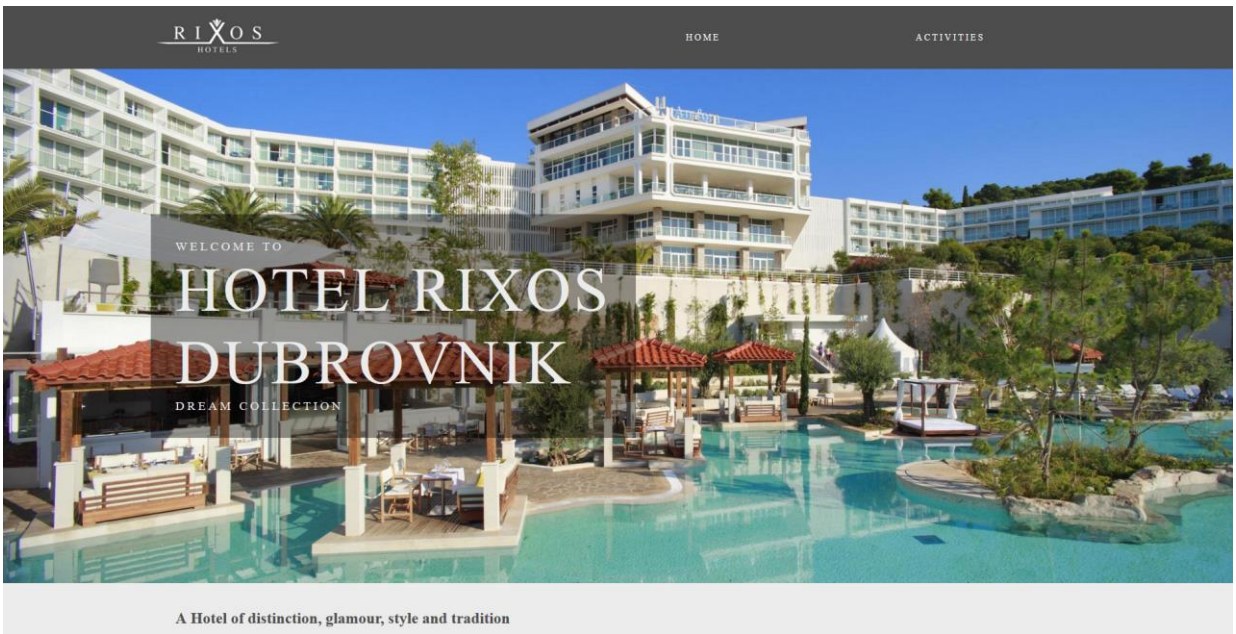
Nakon dodavanja svih potrebnih funkcionalnosti aplikacije (navigacija kroz stranice, prijava, autentikacija i autorizacija ovlaštenih korisnika te dodavanje novih aktivnosti u bazu podataka i njihovo prikazivanje na odgovarajućim stranicama), ostalo je samo pomoću jednostavnih *React* komponenata i *CSS-a* urediti ostatak korisničkog sučelja poput (*Home* stranice) kako bi korisnik imao bolje iskustvo kada posjećuje web stranicu.

4. KORISNIČKO SUČELJE I PRIMJENA

Osim pozadinske funkcionalnosti, vrlo bitna stavka pri izradi web stranice je i njen vizualni dizajn. Web stranica svojim izgledom mora vizualno privlačiti posjetitelje te oni ne smiju imati problema ponovo ju posjećivati u budućnosti. Kada je riječ o web stranicama koje su vezane za turizam, dizajn stranice je još značajniji. Turističke stranice moraju „na prvu“ privući posjetitelja te bi trebale na što jednostavniji način omogućiti posjetitelju da konzumira sadržaj koji turistička destinacija nudi. Stoga, dizajn i funkcionalnost web stranice se uzajamno nadopunjuju. U ovom poglavlju prikazan je dizajn stranice te načini kako ju posjetitelji i zaposlenici hotela mogu koristiti.

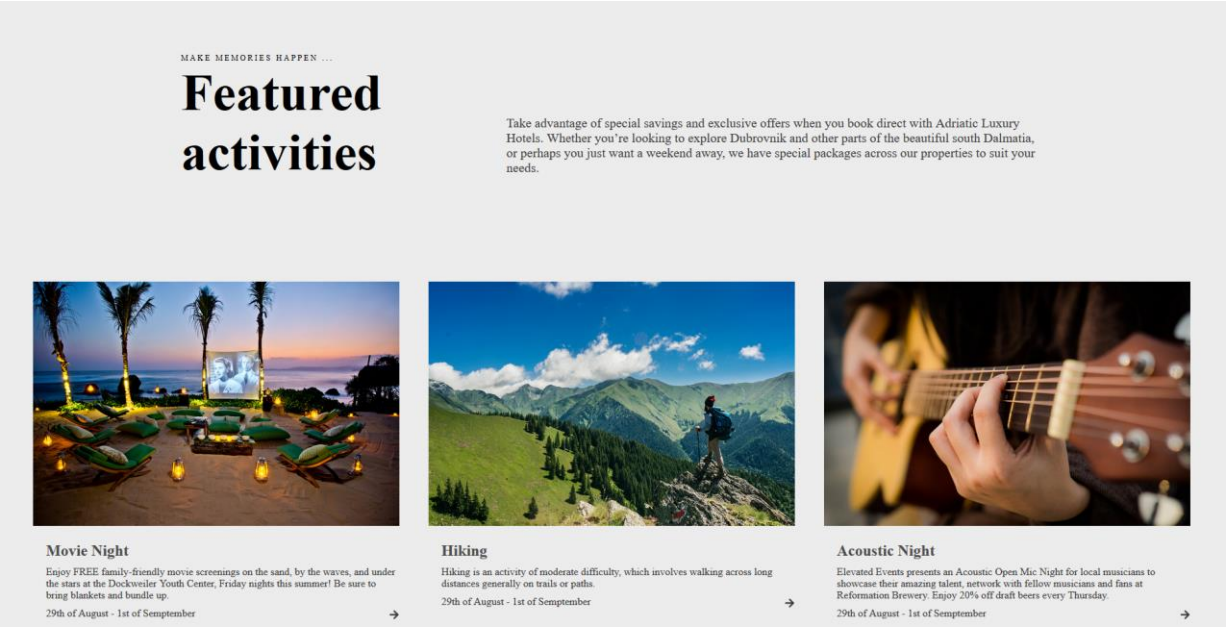
4.1. Klijentsko sučelje

Nakon učitavanja aplikacije, prvo što klijent može vidjeti je početna (engl. *Home)* stranica (slika 4.1.) koja se sastoji od navigacijske trake, pozadine na kojoj je fotografija hotela i njegovo ime, kratkog opisa hotela, nekoliko fotografija aktivnosti kojima se moguće baviti u hotelu te podnožja (engl. *footera*).

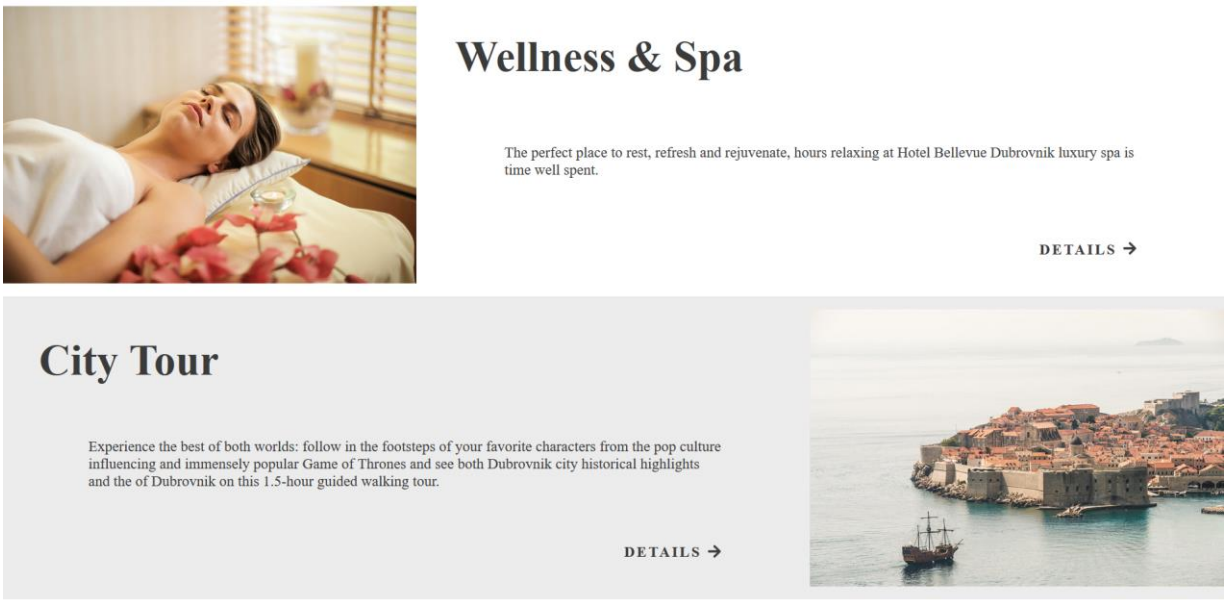


Sl. 4.1. Početna (Home) stranica.

Ako klijent želi pogledati koji se konkretan sadržaj nudi pri boravku u hotelu treba na navigacijskoj traci kliknuti poveznicu *Activities* koja će ga odvesti na stranicu za prikaz svih aktivnosti. Na njoj sadržaj je podijeljen u dvije cjeline: aktivnosti koje imaju ograničeno trajanje (*Featured Activities*) i aktivnosti koje su uvijek dostupne. Na slikama 4.2. i 4.3. prikazan je sadržaj stranice *Activities.*

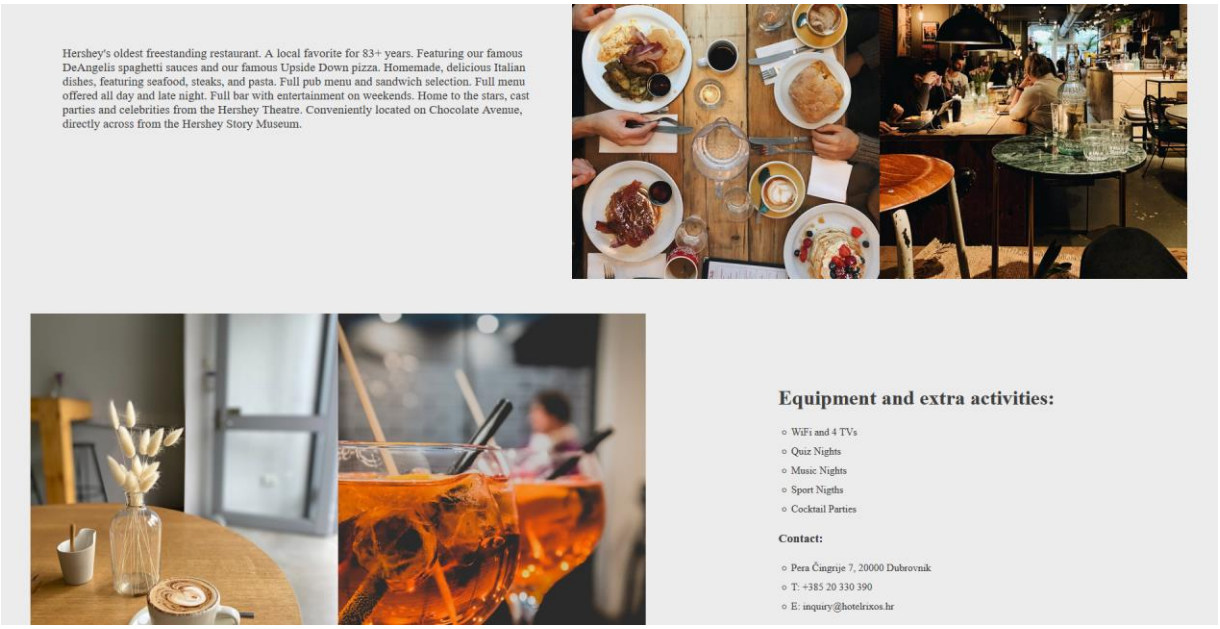


Sl. 4.2. Aktivnosti koje imaju ograničeno vrijeme trajanja.



Sl. 4.3. Aktivnosti koje su uvijek na raspolaganju.

Svaka aktivnost prikazana na ovoj stranici ima zasebnu stranicu na kojoj je moguće vidjeti njene detalje te pogledati galeriju fotografija, a pristupa joj se klikom na poveznicu *Details*. Na slici 4.4. prikazan je primjer stranice za hotelski restorana i bar.



Sl. 4.4. Restaurant & Bar stranica.

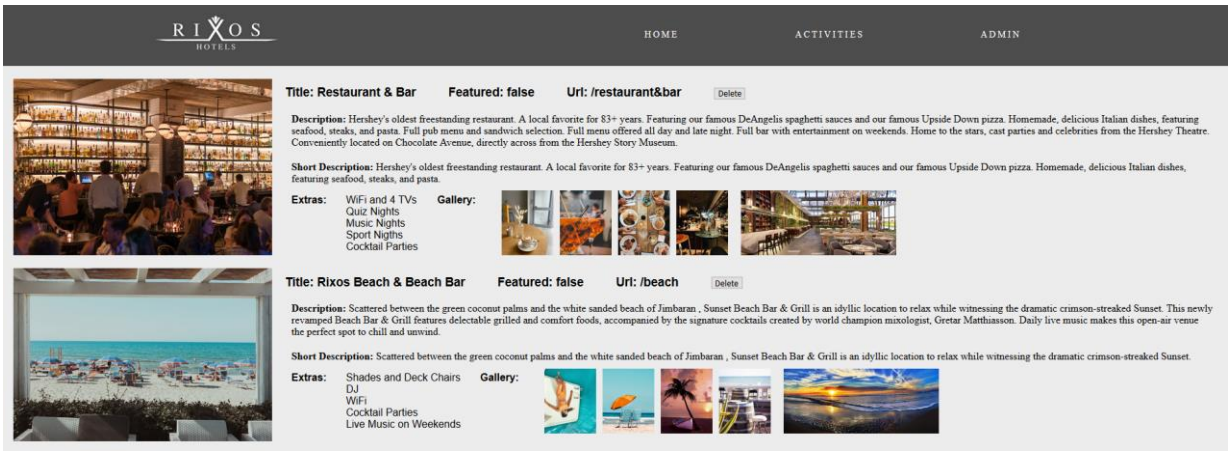
4.2. Sučelje za ovlaštene korisnike

Ukoliko se ovlašteni korisnik želi prijaviti u sustav i uređivati aktivnosti, u podnožju stranice (slika 4.5.) nalazi se poveznica *SIGN IN* koja ga preusmjerava na obrazac za prijavu (slika 4.6.).

Sl. 4.5. Podnožje stranice koje služi za navigaciju.

Sl. 4.6. Obrazac za prijavu.

Klikom na *Sign In* gumb na obrazcu za prijavu, ukoliko su podaci točni, zaposleniku se prikazuje odgovarajući sadržaj za autorizirane korisnike. Na slici 4.7. prikazano je sučelje *Admin* stranice i koje sve informacije ovlašteni korisnik može vidjeti o pojedinoj aktivnosti. To su: naslov aktivnosti, naslovna fotografija i njen položaj, položaj aktivnosti na *Activities* stranici, URL aktivnosti, opis, detalji i galerija slika. Ovlašteni korisnik može dodati novu aktivnost ili obrisati postojeće.



Sl. 4.7. Sučelje Admin stranice.

5. ZAKLJUČAK

Cilj ovoga završnog rada bio je kreirati web aplikaciju za informiranje klijenata o ponuđenim sadržajima u hotelskom smještaju. Za izradu web aplikacije korištene su tehnologije; *HTML*, *CSS*, *Firebase* i *JavaScript* biblioteka (*framework*) - *React*. Pri rješavanju praktičnog problema naglasak je bio na korištenju tehnologije *React* jer zapravo ona obuhvaća sve ostale tehnologije i povezuje ih u jednu cjelinu. Kako bi web stranica izrađena kroz ovaj rad imala funkcionalnosti koje su zadane u zadatku rada, bilo je potrebno koristiti pozadinsku platformu *Firebase*. Ona je zajedno s tehnologijom *React* omogućila stvaranje sučelja u kojem klijenti mogu pregledavati sadržaje hotela koji su spremljeni u bazu podataka, dok ovlašteni zaposlenici hotela mogu i uređivati te sadržaje. Na izrađenu web aplikaciju moguće je nadograditi i druge funkcionalnosti poput rezervacije hotelskih soba, tako da se ova začetna verzija web stranice može razviti u proizvod koji bi svojim preformansama bio konkurentniji na tržištu.

LITERATURA

[1] M. Vukman, K. Drpić, „Utjecaj Internet marketinga na razvoj brenda turističke destinacije“, Visoka škola za sportski menadžment Aspira, Split, Hrvatska, 2014.

[2] B. Andrlić, „Čimbenici marketinškog okruženja u turizmu, Veleučilište u Požegi, Požega, Hrvatska, 2011.

[3] L. Filistrucchi, „The Impact of Internet on the Market for Daily Newspapers in Italy“, European University Institute, Department of Economics, Italija, 2005.

[4] Web stranica hotela Esplanade Zagreb, dostupno na: https://www.esplanade.hr/, lipanj 2020.

[5] Web stranica hotela Navis Opatija, dostupno na: https://hotel-navis.hr/en/, lipanj 2020.

[6] Web stranica hotela Sun Gardens Dubrovnik, dostupno na https://www.dubrovniksungardens.com/, lipanj 2020.

[7] Web stranica društvene mreže *Facebook*, dostupno na: https://www.facebook.com/, lipanj 2020.

[8] *Statista*, vodeći provider podataka o ekonomskom tržištu i potrošačima, dostupno na: https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/, lipanj 2020.

[9] Web stranica streaming servisa *Netflix*, dostupno na: https://www.netflix.com/, lipanj 2020.

[10] *Google Trends*, podaci o pretraživanju pojmova, dostupno na: https://trends.google.com/trends/explore?date=today%205-y&q=%2Fm%2F012l1vxv,%2Fg%2F11c6w0ddw9,%2Fg%2F11c0vmgx5d,%2Fm%2F0268gyp, rujan 2020.

[11] J. Duckett, „*HTML & CSS, Desing and Build Websites*“, John Wiley & Sons, Indianapolis, Sjedinjene Američke Države, 2011.

[12] M. McGrath, „*HTML, CSS & JavaScript in Easy Steps*“, Ujedinjeno Kraljevstvo, 2020.

[13] Službena stranica World Wide Web Konzorcija, međunarodne organizacije za standarde za World Wide Web, dostupno na: https://www.w3.org, lipanj 2020.

[14] M. Haverbeke, „*Eloquent JavaScript, A Modern Introduction to Programming*“, William Pollock, San Francisco, Sjedinjene Američke Države, 2011.

[15] Službena web stranica *React* biblioteke, dostupno na: https://reactjs.org/, lipanj 2020.

[16] Službena web stranica *Firebase* platforme dostupno na: https://firebase.google.com/, lipanj 2020.

SAŽETAK

U ovom radu cilj je bila izrada web aplikacije koja će korisnicima omogućiti pregledavanje aktivnosti i sadržaja koje hotel pruža. Kao uvod u izradu aplikacije objašnjeno je koliki je značaj Internet marketinga u turizmu, pokazani su primjeri turističkih web stranica te je ukazano koja su sva znanja, tehnologije i alati potrebni prije same izrade aplikacije. Zatim su objašnjeni glavni koncepti i usluge koje pružaju dvije tehnologije u kojima će biti izrađena web aplikaciju, a to su *React* i *Firebase*. U glavnom dijelu, kroz primjenu ranije spomenutih koncepata i kroz dijelove napisanog koda, objašnjena je implementacija svake značajke koja je bila potrebna za ispravno funkcioniranje web aplikacije. U posljednjem dijelu ovog rada, prikazan je izgled korisničkog sučelja te način na koji ovlašteni korisnici mogu uređivati sadržaje hotela.

Ključne riječi: Firebase, hotel, React, web aplikacija

ABSTRACT

Web application for informing clients about hotel accommodation content

The main aim of this work was making of a web application which will enable users to browse through the activities and content that a hotel provides. As an introduction into application development, it was explained how crucial Internet marketing is for tourism, examples of tourism web sites were shown, and it was also provided what kind of knowledge, technologies and tools are needed before the beginning of building the application itself. Afterwards, the main concepts and services which React and Firebase - two technologies used for developing this app – provide, were explained. In the main part, through the application of the previously mentioned concepts and through code snippets, the implementation of every feature required for proper functioning of the web application was discussed. In the final part of this paper, user interface layout and the way in which authorized users can edit hotel content, were shown.

Keywords: Firebase, hotel, React, web application

ŽIVOTOPIS

Luka Pivk rođen je 10.10.1998. godine u Osijeku. Nakon završene Osnovne škole Josipa Antuna Ćolnića u Đakovu, upisuje Gimnaziju Antuna Gustava Matoša Đakovo gdje maturira 2017. godine. Iste godine upisuje sveučilišni preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Govori i piše engleski te njemački. Poznaje rad na računalu u raznim programskim i skriptnim jezicima poput C-a, C++-a, C#-a. HTML-a, CSS-a, JavaScript-a te tehnologijama kao što su React i Firebase.

Potpis:

_____