Web portal za vijesti

Čuković, Josip

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:182372

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-27**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

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

Sveučilišni studij

WEB PORTAL ZA VIJESTI

Diplomski rad

Josip Čuković

Osijek, 2022.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 05.09.2022.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Josip Čuković
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1110R, 13.10.2020.
OIB studenta:	56153278641
Mentor:	Prof. dr. sc. Krešimir Nenadić
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Ivica Lukić
Član Povjerenstva 1:	Prof. dr. sc. Krešimir Nenadić
Član Povjerenstva 2:	Robert Šojo, mag. ing. comp.
Naslov diplomskog rada:	Web portal za vijesti
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	Potrebno je opisati način funkcioniranja web portala za objavu vijesti. Navesti funkcionalnosti koje će podržavati web portal koji će biti izrađen. Modelirati i izraditi bazu podataka koja će podržavati funkcionalnosti web portala. Predvidjeti barem tri različite korisničke uloge: gost, urednik i administrator. Omogućiti različitim ulogama odgovarajuće funkcionalnosti web portala. Potrebno je opisati postupak izrade web portala kao i njegove funkcionalnosti. Tema rezervirana: Josip Čuković Sumentor iz tvrtke: Damir Ljubičić, Cobe
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	05.09.2022.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 30.09.2022.

Ime i prezime studenta:	Josip Čuković
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1110R, 13.10.2020.
Turnitin podudaranje [%]:	10

Ovom izjavom izjavljujem da je rad pod nazivom**: Web portal za vijesti**

izrađen pod vodstvom mentora Prof. dr. sc. Krešimir Nenadić

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD **1**

2. PREGLED POSTOJEĆIH WEB PORTALA ZA VIJESTI **2**

2.1. Dnevnik.hr **2**

2.2. Index.hr **2**

2.3. BBC News **3**

2.4. The New York Times **4**

2.5. Slobodna Dalmacija **4**

3. TEHNOLOGIJE KORIŠTENE PRILIKOM IZRADE APLIKACIJE **5**

3.1. Node.js **5**

3.2. Express.js **6**

3.3. MongoDB **7**

3.4. TypeScript **7**

3.5. Postman **8**

3.6. React **9**

3.7. CSS **10**

4. PROGRAMSKO RJEŠENJE **11**

4.1. Struktura aplikacije **11**

4.1.1. Backend 11

4.1.2. Frontend 13

4.2. Registracija i prijava korisnika **14**

4.3. Dohvaćanje, kreiranje, ažuriranje i brisanje članaka **18**

4.4. Komentiranje članaka **32**

4.5. Pretraživanje članaka **36**

5. VIZUALNA REPREZENTACIJA **38**

6. ZAKLJUČAK **47**

LITERATURA **48**

SAŽETAK **49**

ABSTRACT **50**

ŽIVOTOPIS **51**

PRILOZI **52**

1. UVOD

Čitanje o događajima u svijetu putem interneta postalo je neizostavan dio ljudske svakodnevnice. Sve užurbaniji život zahtjeva mogućnost informiranja putem interneta kroz odgovarajući web portal.

Web portal za vijesti, čija je izrada objašnjena i opisana u nastavku rada, pruža sve potrebne funkcionalnosti koje ispunjavaju svrhu aplikacije, a to je čitanje aktualnih vijesti.

Web portal za vijesti ima mogućnosti pregledavanja vijesti, pretraživanje vijesti prema kategoriji, prikaz pojedinačne vijesti, komentiranje, podržavanje više uloga korisnika gdje određene uloge imaju mogućnost kreiranja nove i brisanje vijesti. Prijavljeni korisnik ima mogućnost komentiranja vijesti i prikazan mu je podatak o broju pregleda svake vijesti, dok neprijavljeni korisnik ima samo mogućnost pregledavanja, odnosno, čitanja vijesti. Korisnik s ulogom „editor“ ima mogućnost kreiranja novih i ažuriranja postojećih vijesti, dok korisnik s ulogom „admin“ ima dodatnu mogućnost brisanja vijesti.

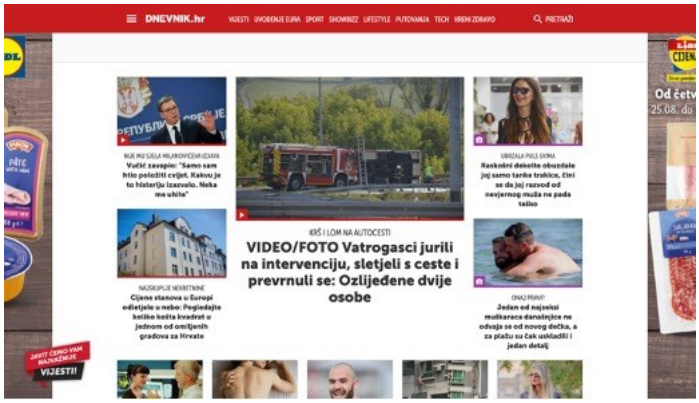
Nakon uvoda, u drugom poglavlju su opisani primjeri postojećih rješenja kako bi se istaknule razlike te neke od prednosti izrađenog web portala u odnosu na navedena postojeća rješenja. U trećem poglavlju opisane su tehnologije korištene prilikom izrade portala. Za svaku od navedenih tehnologija objašnjene su najvažnije karakteristike. Četvrto poglavlje opisuje programsko rješenje u kojem su opisani svi pojedini dijelovi aplikacije te kako su spojeni u jednu cjelinu koja čini aplikaciju, počevši od strukture klijentske i poslužiteljske aplikacije, registracije i prijave, dohvaćanje, kreiranje, ažuriranje, brisanje te komentiranje vijesti. U petom poglavlju prikazan je izgled aplikacije prilikom određenih stanja, primjerice kada je korisnik prijavljen ili kada nije.

2. PREGLED POSTOJEĆIH WEB PORTALA ZA VIJESTI

U ovom poglavlju navedeni su online web portali za vijesti. Za svaki portal prikazana je njegova početna stranica i navedene su njihove funkcionalnosti.

2.1. Dnevnik.hr

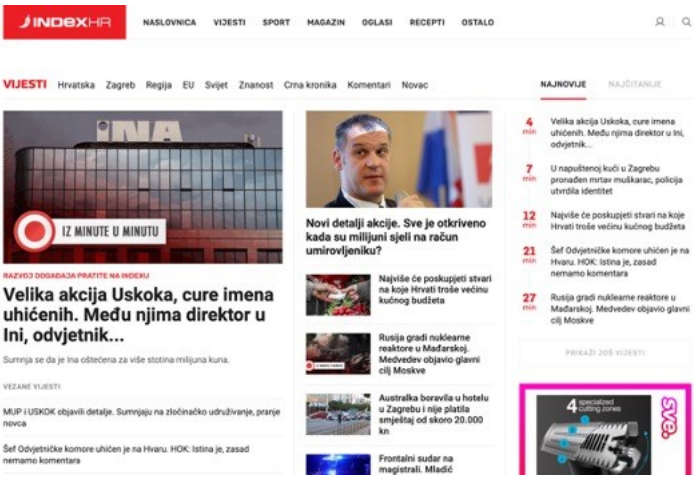
Dnevnik.hr jedan je od najpoznatijih portala u hrvatskoj. Portal nudi pregled članaka prema kategorijama, a neki od njih su: sport, showbizz, lifestyle, putovanja, tech. Također, prikazano je po nekoliko najnovijih vijesti iz različitih kategorija. Portal nudi i pretragu članaka prema ključnoj riječi (slika 2.1.) [1] .



Slika 2.1. *Prikaz početne stranice portala Dnevnik.hr*

2.2. Index.hr

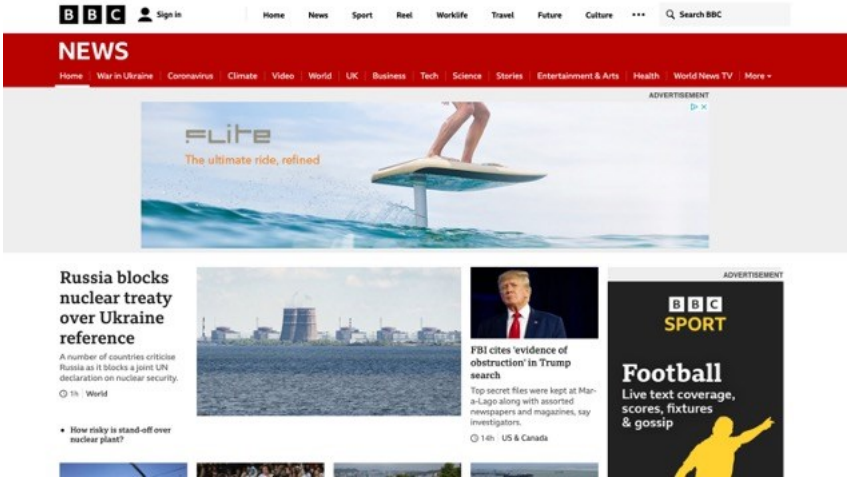
Index.hr također je jedan od poznatijih portala. Funkcionalnosti koje nudi su slijedeće: pregled članaka prema kategorijama kao što su: sport, magazin, znanost. Nadalje, portal na početnoj stranici prikazuje po nekoliko vijesti iz različitih kategorija gdje je na jednoj vijesti od svake kategorija prikazan broj komentara. Vijesti je moguće pretraživati prema ključnoj riječi. Također, nudi se mogućnost registracije gdje samo registrirani korisnici mogu komentirati članke (slika 2.2.) [2].



Slika 2.2. *Portal za vijesti Index.hr*

2.3. BBC News

BBC (engl. *British Broadcasting Corporation*) News je poslovni odjel odgovoran za prikupljanje i prikazivanje vijesti. BBC News nudi kategorije vijesti kao što su: vijesti iz svijeta (engl. *World*), sporta (engl. *Sport*), znanosti (engl. *Science*), tehnologije (engl. *Tech*) i slično [3]. Korisnici imaju mogućnost registracije čime stječu pravo komentiranja članaka. Važno je napomenuti kako se samo određeni članci mogu komentirati. BBC News kao i ostali portali nudi mogućnost pretrage vijesti prema ključnoj riječi. (slika 2.3.) [4].



Slika 2.3. *BBC News*

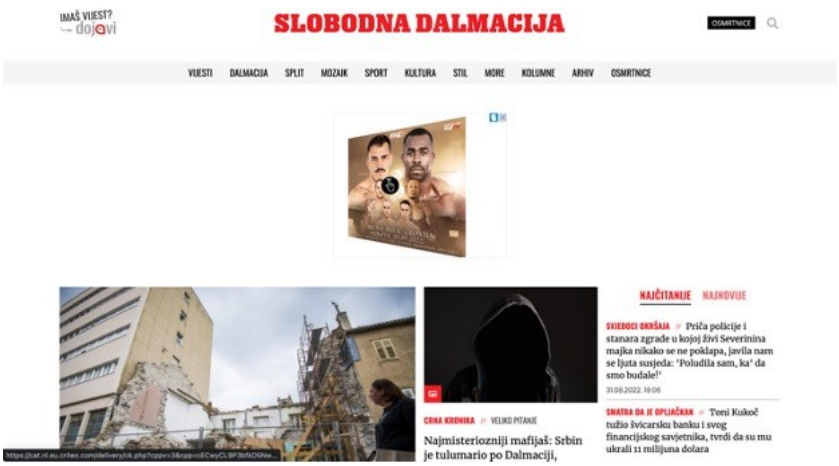
2.4. The New York Times

New York Times su američke dnevne novine sa sjedištem u New Yorku, a prepoznate su diljem svijeta. Portal nudi mogućnost pregleda različitih kategorija kao što su: „Politics“, „World“, „Science“, „Tech“ i slično (slika 2.4.) [5] . Korisnici imaju mogućnost pretraživanja vijesti, registracije, no potrebno je pretplatiti se kako bi dobili pristup člancima. Na početnoj stranici prikazano je po nekoliko vijesti iz različitih kategorija.

Slika 2.4. *The New York Times*

2.5. Slobodna Dalmacija

Slobodna Dalmacija jedan je od najpoznatijih dalmatinskih portala. Portal nudi mogućnost pregleda kategorija kao što su: sport, kultura, svijet, biznis i politika. Početna stranica portala prikazuje nekoliko vijesti iz različitih kategorija. Također, moguće je i pretraživati članke prema ključnoj riječi (slika 2.5.) [6].



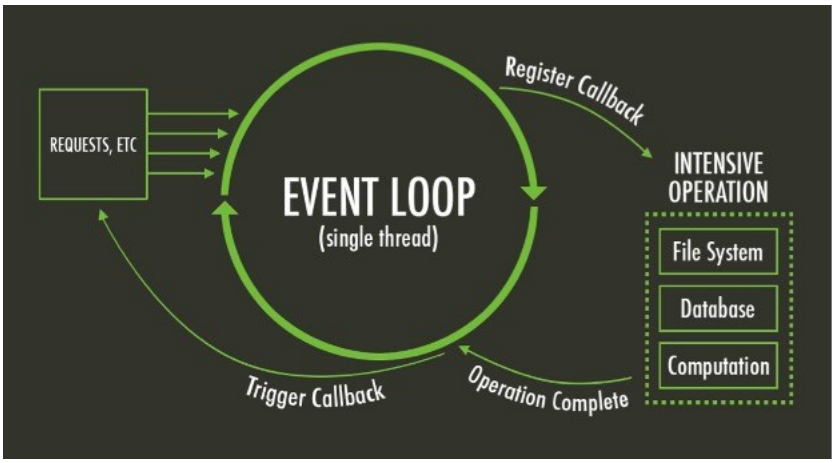
Slika 2.5. *Portal Slobodna Dalmacija*

3. TEHNOLOGIJE KORIŠTENE PRILIKOM IZRADE APLIKACIJE

Postoji nekoliko popularnih development stackova kao što su MEAN, MERN ili MEVN u kojima se koristi Express.js zajedno s MongoDB bazom i određenim JavaScript *frontend* okruženjem ili bibliotekom. U ovoj aplikaciji koriste se MERN stack tehnologije, odnosno MongoDB, Express, React i Node.js te su u nastavku ukratko objašnjene.

3.1. Node.js

Node.js je *open-source* okruženje kojeg pokreće Google-ov V8 *engine*, te se može koristiti na više platformi i izvodi se na jednoj niti, odnosno on je *single-threaded*. Nema potrebe za stvaranjem nove niti za svaki zahtjev. Iako je većina Node.js aplikacija pisana u JavaScript programskom jeziku, aplikacije se mogu uz dodatnu konfiguraciju pisati i u TypeScript-u koji je korišten prilikom izrade aplikacije te je objašnjen u nastavku. Node.js je *event-driven,* odnosno temelji se na događajima koji su pokrenuti asinkrono i zahtjevi se obavljaju slijedno što znači da jedna aktivnost ne blokira drugu [7]. Važno je napomenuti da glavna petlja događaja (engl. *event loop*) delegira poslove operacijskom sustavu kao što su čitanje/pisanje datoteka i slično. Prava prednost se postiže kada postoji veliki broj zahtjeva i podataka. Brzo izvršavanje koda, asinkronost i jednostavnost postavljanja poslužitelja su glavne prednosti Node.js-a koje su pružene prilikom izrade aplikacije.



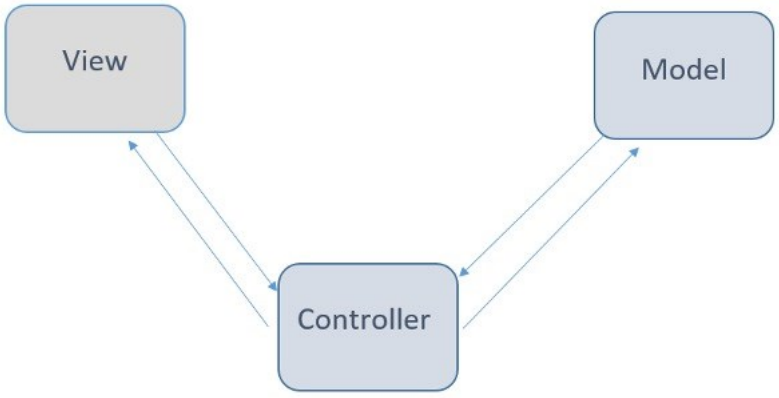
Slika 3.1. *Prikaz načina rada Node.js (Izvor: https://redberry.international/what-is-event-loop-things-to-know/)*

Slika 3.1. prikazuje način rada Node.js-a. Kao što je već rečeno, glavna značajka načina rada ovog okruženja je postojanje petlje događaja koja ih osluškuje te obrađuje.

Uz instalaciju okvira instalira se i alat nazvan NPM (engl. *Node Package Manager*). To je alat pomoću kojeg se pristupa različitim paketima i pomoću kojeg se oni instaliraju, a predstavlja standard za definiranje ovisnosti o paketima. Paketi su dostupni na https://www.npmjs.com/ službenoj stranici.

3.2. Express.js

Express.js ili kraće Express je *backend* okvir za Node.js i njegova svrha je olakšati proces kreiranja web aplikacija i API-a (engl. *Application Programming Interface*) koji omogućuju komunikaciju između dvije aplikacije. Glavna karakteristika je minimalizam, odnosno nije kompliciran i pruža samo ono najpotrebnije. Express omogućuje postavljanje posredničkog softvera (engl. *middleware*) [8] što je dio koda koji se izvršava prilikom svakog ili određenih zahtjeva kako bi se ulazni podaci dodatno obradili. Kada jedan *middleware* završi svoj 'posao' prelazi se na idući i tako sve dok se ne pošalje nekakav odgovor (engl. *response*) ili dok ne dođe do nekakve pogreške (engl. *bug*). Express omogućuje definiranje ruta (engl. *routes*) koristeći HTTP (engl. *HyperText Transfer Protocol*) zahtjeve kao što su GET, PUT, POST i DELETE koje koristimo ovisno o tome želi li se dohvaćati, ažurirati, slati ili brisati podatke odnosno sadržaj te omogućuje dinamičko prikazivanje HTML (engl. *HyperText Markup Language*) stranica na temelju proslijeđenih argumenata u predlošku. Struktura aplikacije nije strogo definirana, no često se koristi MVC (engl. *Model-View-Controller*) obrazac dizajna softvera (engl. *software design pattern*).



Slika 3.2. *Prikaz rada MVC obrasca (Izvor: https://towardsdatascience.com/everything-you-need-to-know-about-mvc-architecture-3c827930b4c1)*

Na slici 3.2. prikazan je MVC obrazac. Unutar MVC obrasca korisnik komunicira s pogledom, a u slučaju ove aplikacije to je *frontend*, odnosno React aplikacija. Nakon korisnikove interakcije zahtjev se šalje prema kontroleru koji obrađuje zahtjev te ukoliko je potrebno šalje odgovarajući

zahtjev prema modelu, podaci se obrađuju i šalje se odgovarajući odgovor. Model je odgovoran za komunikaciju s bazom, a sva ostala logika nalazi se unutar kontrolera.

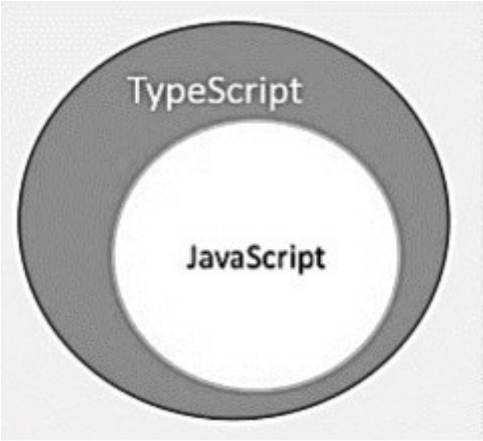
3.3. MongoDB

MongoDB je dokumentno orijentirana NoSQL baza podataka. Podaci unutar baze pohranjuju se u BSON formatu (engl. *Binary JSON*), koji je vrlo sličan JSON formatu [9]. Svi podaci koji su relevantni za određeni dokument su spremljeni zajedno što uvelike ubrzava čitanje podataka jer nema potrebe za korištenjem dodatnih naredbi kao što je 'JOIN' u SQL-u. Skupina dokumenata čine kolekciju (engl. *collection*). Kolekcija sadržava skupinu dokumenata koji pripadaju određenoj skupini, npr. korisnici (engl. *users*). U odnosu na relacijske baze podataka gdje svaki zapis ima jednaku strukturu, u MongoDB to nije slučaj. Svaki dokument u kolekciji može imati različitu strukturu, ali se zato koristi NPM paket *Mongoose* koji između ostaloga omogućuje definiranje strukture odnosno sheme za svaki dokument unutar pojedine kolekcije. Na slici 3.3. prikazan je jednostavan primjer dokumenta korisnika unutar baze podataka.

Slika 3.3. *Prikaz dokumenta korisnika*

3.4. TypeScript

TypeScript je programski jezik kojeg je razvio i kojeg održava Microsoft. Predstavlja strogi sintatički nadskup JavaScripta što je prikazano slikom 3.4. i dodaje mogućnost statičkog pisanja.



Slika 3.4. *Prikaz odnosa JavaScripta i TypeScripta (Izvor: https://www.quora.com/What-is-TypeScript)*

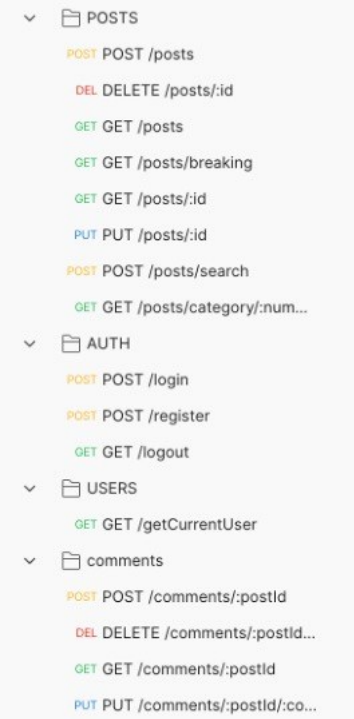
Drugim riječima, TypeScript je zapravo JavaScript s dodatnim mogućnostima kao što je definiranje tipova podataka prilikom pisanja koda. To u JavaScriptu nije moguće zato što je JavaScript dinamički pisani (engl. *dynamically typed*) jezik. TypeScript uvodi pojmove kao što su 'Interface', kreiranje prilagođenih tipova podataka pomoću *type* ključne riječi i slično. Statički pisani (engl. *statically typed*) jezici provode provjeru tipova podataka prilikom kompajliranja (engl. *compile time*), dok dinamički pisani jezici to rade u trenutku izvođenja. Također, statički pisani jezici, za razliku od dinamičkih, zahtijevaju eksplicitno navođenje tipa podatka. Glavna prednost korištenja TypeScript-a je što smanjuje mogućnost pojavljivanja pogrešaka u kodu. Primjerice, slika 3.5. prikazuje funkciju u kojoj su parametri definirani te je vrlo jasno kakve parametre, odnosno, kojeg tipa parametri koje funkcija prima moraju biti. U JavaScript-u to nije slučaj i zbog toga može doći do neželjenog ponašanja aplikacije.

```
function sum(a: number, b: number){
  return a + b;
}
```

Slika 3.5. *Prikaz funkcije u TypeScriptu*

3.5. Postman

Postman je platforma koja nudi jednostavno testiranje kreiranog API-a putem grafičkog sučelja za slanje i pregled HTTP zahtjeva i odgovora. U radu se koristi za dodavanje, brisanje, ažuriranje i dohvaćanje podataka iz baze te provjeru putanja koje su definirane u aplikaciji kako bi se mogle nesmetano koristiti. Testiranje se vrši na način da se šalju različiti zahtjevi te se promatra ponašanje poslužitelja, primjerice kako će reagirati u slučaju neispravnog unosa očekivanih podataka i slično.



Slika 3.6. *Prikaz Postman kolekcije testiranih ruta*

Slika 3.6. prikazuje sve rute koje su omogućene i testirane unutar aplikacije.

3.6. React

React je jedna od najpopularnijih *frontend* JavaScript biblioteka koja se koristi za izradu korisničkih sučelja baziranih na komponentama [10]. Tvrtka Meta, prijašnji Facebook, kreirao je i održava ovu biblioteku. React se koristi za izradu aplikacija koje sadrže samo jednu stranicu (engl. *single page application*) te se sadržaj naknadno 'ubrizgava' po potrebi. Omogućava kreiranje modularnog i prilagodljivog dizajna što ju čini prikladnom tehnologijom za ovu aplikaciju. Sučelje se može uređivati na tradicionalne načine kao i kada se ne koristi dodatno okruženje ili biblioteka pomoću CSS-a što je učinjeno i u ovome radu.

React u odnosu na okvire kao što su Angular ili Vue ima bolju učinkovitost, fleksibilnost i troši manje resursa [11]. Angular dolazi s puno predefiniranih funkcionalnosti, dok je React fleksibilniji i omogućava korištenje biblioteke trećih strana (engl. *third party libraries*). React koristi komponente koje služe kao gradivni elementi i moguće ih je, uz pravilnu implementaciju, ponovno iskoristiti i na taj način smanjiti dupliciranje koda unutar aplikacije. Koristi JSX (engl. *JavaScript XML*) sintaksu koja naizgled može podsjećati na neke od predložaka kao što su EJS ili Blade. JSX

omogućuje pisanje HTML elemenata te direktno upisivanje određenih vrijednosti u njih [12]. Slikom 3.7. prikazan je primjer komponente kreirane pomoću JSX sintakse.

```
return (
  <div className="login">
    <h2>Login</h2>
    <form onSubmit={handleSubmit}>
      <label>Email:</label>
      <input type="email" required value={email} onChange={(e) => setEmail(e.target.value)} />
      <label>Password:</label>
      <input type="password" required value={password} onChange={(e) => setPassword(e.target.value)} />
      {!isPending && <button>Login</button>}
    </form>
    {isPending && <p>Loading...</p>}
    {error && <p className="errorMessage">{error}</p>}
    <p>
      Don't have account? <Link to="/register">Register</Link>
    </p>
  </div>
);
```

Slika 3.7. *Prikaz jednostavnog React predloška koristeći JSX sintaksu*

React koristi virtualni objektni model dokumenta, odnosno DOM (engl. *Document Object Model*), koji predstavlja apstrakciju osnovnog modela. DOM je sučelje (engl. *interface*) koje predstavlja web stranicu te omogućuje da programi utječu na strukturu dokumenta. Direktna promjena ovog modela zahtijeva više resursa i vremena za ponovno učitavanje, dok virtualni DOM uzima vrijednost koja se promijenila nakon čega prolazi cijelu strukturu, uspoređujući prethodno stanje objekta s novim. Tada ažurira samo objekt koji je izmijenjen i prikazuje promjenu na stranici umjesto ažuriranja i ponovnog prikazivanja cijele stranice [13].

3.7. CSS

CSS (engl. *Cascading Style Sheets*) je jezik kojim je opisan stil HTML dokumenta, odnosno, prezentacija web stranica [14]. Opći oblik CSS deklaracija ima sljedeći oblik: selektor:{svojstvo: vrijednost} gdje selektor određuje element na koje se stilsko sredstvo odnosi. Neki najčešće korišteni selektori su: jednostavni, klasni i id selektor.

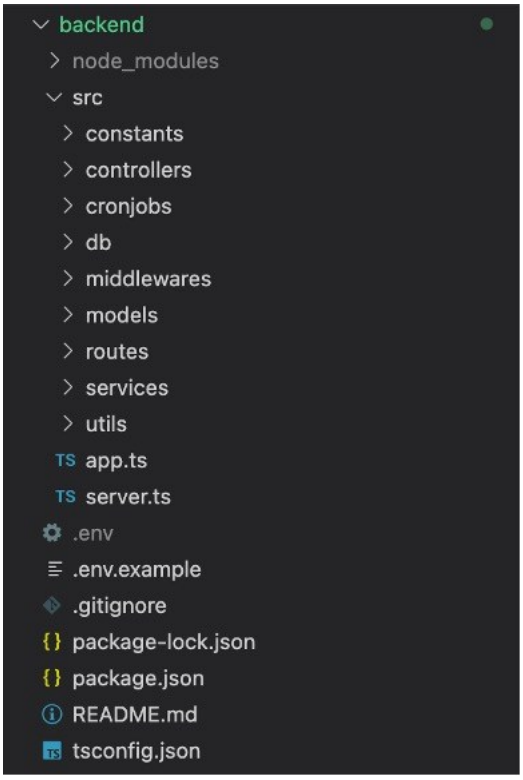
4. PROGRAMSKO RJEŠENJE

Aplikacija je podijeljena na dva dijela - *frontend* i *backend*, a u nastavku prikazana i objašnjena struktura za svaki dio. Nakon toga detaljno su objašnjene najbitnije funkcionalnosti aplikacije.

4.1. Struktura aplikacije

4.1.1. Backend

Prvo je potrebno postaviti Node.js okruženje. Kreirana je mapa *backend* u kojoj se pomoću naredbe *npm init* postavlja projekt te se kreira datoteka package.json koja sadrži sve ovisnosti aplikacije koje su potrebne za ispravan rad. Zatim, navedene su i korištene verzije svake ovisnosti kako bi se, ukoliko se projekt preuzme, svi potrebni moduli ispravno instalirali putem naredbe *npm install*. Nakon toga, kako bi se mogao koristiti TypeScript potrebno je instalirati TypeScript pomoću naredbe *npm install -D typescript* te kreirati tsconfig.json datoteku pomoću naredbe *tsc --init*. Potrebno je imati TypeScript instaliran globalno, a u suprotnom je potrebno koristiti *npx tsc -- init*.



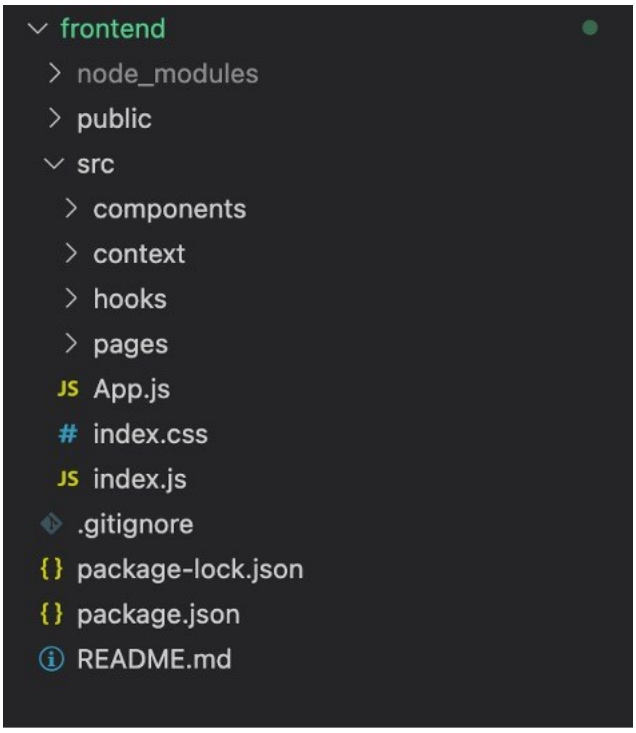
Slika 4.1. *Prikaz strukture backend dijela aplikacije*

- node_modules – mapa (engl. *folder*) koja sadrži sve pakete koji su preuzeti za svaku projektnu ovisnost

- .env – datoteka koja sadrži varijable koje ne smiju biti vidljive, npr. podaci za povezivanje na bazu i slično. Bitno je napomenuti da se mora koristiti *dotenv* ili sličan paket kako bi sve varijable bile dostupne u kodu.
- .env.example – datoteka koja sadrži sve iste varijable kao i .env datoteka, no varijablama nije dodijeljena nikakva vrijednost. Razlog postojanja ove datoteke je taj da kada netko drugi preuzme kod zna što .env datoteka treba sadržavati.
- tsconfig.json – datoteka koja je spomenuta ranije u radu, no važno je napomenuti da ova datoteka sadrži sve postavke ponašanja TypeScript-a.
- src – mapa koja sadrži cijeli *backend* kod za aplikaciju
    - constants – mapa koja sadrži sve konstante korištene u aplikaciji (npr. poruke o pogrešci i slično).
    - controllers – mapa koja sadrži sve kontrolere, odnosno, svu poslovnu logiku te čini jedan od tri dijela MVC obrasca kako je ranije i objašnjeno.
    - cronjobs – mapa koja sadrži sve aktivnosti koje se periodno ponavljaju.
    - db – mapa koja sadrži funkcije za komuniciranje i spajanje s bazom.
    - middlewares – mapa koja sadrži sve posredničke softvere, odnosno *middleware*. Sadržava funkcije kao što su provjera je li korisnik prijavljen, ima li korisnik određena prava i slično.
    - models – mapa koja sadrži modele koji definiraju strukturu podataka koji će biti spremljeni unutar baze. Model je sastavni dio MVC obrasca te je odgovoran za direktno komuniciranje s bazom.
    - routes – mapa koja sadrži sve rute koje su definirane pomoću HTTP zahtjeva kao što su GET, PUT, POST, DELETE i putanje.
    - services – mapa koja sadrži sve poslove koje koriste API treće strane.
    - utils – mapa koja sadrži sve pomoćne funkcije ili logiku koja se koristi kroz aplikaciju na više mjesta.
    - app.ts – datoteka koja sadrži sve vezano uz Express konfiguraciju.
    - server.ts – datoteka u kojoj se nalazi logika poslužitelja. U ovoj datoteci se server kreira pomoću metode „*http.createServer*“ te joj se predaje Express objekt koji je prije toga konfiguriran, a Express je odgovoran za sve nadolazeće zahtjeve.

4.1.2. Frontend

Nakon postavljanja poslužitelja, potrebno je implementirati klijentsku stranu koja obuhvaća razvoj aplikacije u React-u i povezivanje s poslužiteljem. Projekt je kreiran pomoću naredbe *npx create-react-app <ime projekta>*. Navedena naredba kreira React projekt s osnovnom strukturom i potrebnim ovisnostima za lakši početak razvoja aplikacije. Slika 4.2. prikazuje strukturu *frontend*, odnosno, klijentske aplikacije.



Slika 4.2. *Prikaz strukture frontend dijela aplikacije*

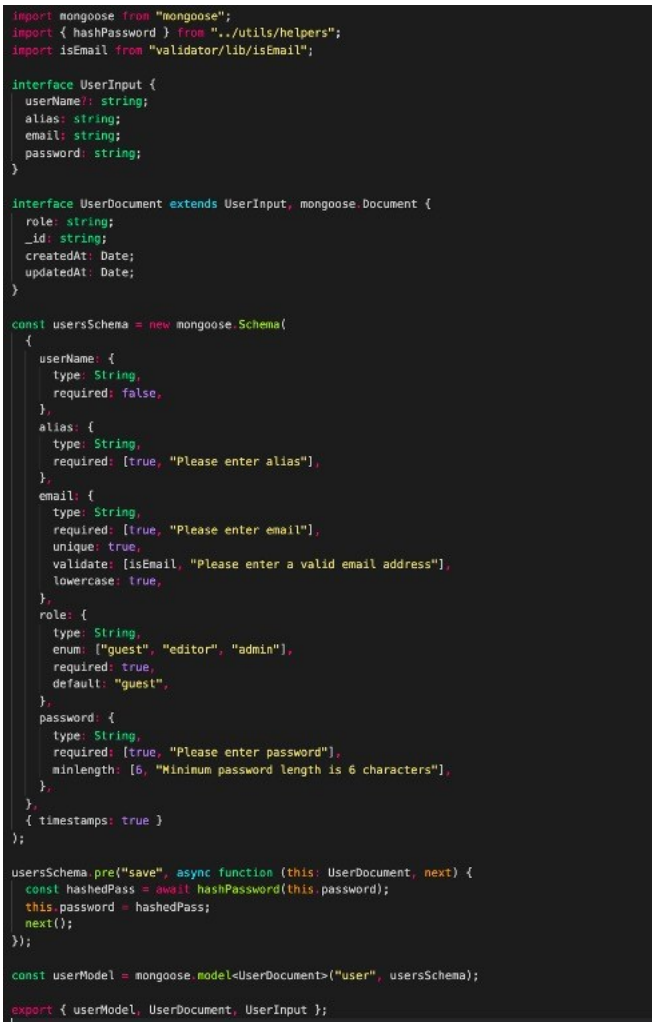
- public – mapa koja sadrži pregledniku dostupne datoteke kao što su slike ili index.html stranica u kojoj se kroz korištenje aplikacije ubrizgavaju sadržaji prema potrebi.
- src – mapa koja sadrži svu logiku i stilove koju aplikacija koristi.
    - components – mapa koja sadrži komponente odnosno gradivne elemente svake React aplikacije
    - context – mapa koja sadrži kontekst koji se koristi za provjeru je li korisnik prijavljen, koja mu je uloga i slično.
    - hooks – mapa koja sadrži oblik „helper“ funkcija.
    - pages – mapa koja sadrži sve stranice ili poglede u aplikaciji.
    - App.js – datoteka u kojoj se nalazi aplikacijski ruter koji je zadužen za prikazivanje ispravnih stranica/pogleda ovisno o putanji kojoj se pristupa.

- index.css – datoteka koja sadrži stilove aplikacije.
- index.js – glavna JavaScript datoteka koja ubrizgava sadržaj unutar index.html datoteke.

Ostale mape i datoteke su slične kao i u *backend* aplikaciji.

4.2. Registracija i prijava korisnika

Aplikaciji mogu pristupiti četiri vrste korisnika, a to su: anonimni i prijavljeni koji se dijele na „guest“, „editor“ i „admin“ uloge. Anonimni korisnici mogu pregledavati objavljene vijesti pojedinačno u uvećanom prikazu gdje je vidljiv cijeli članak, vijesti neke kategorije ili vijesti na početnoj stranici. Korisnik koji je prijavljen i ima ulogu „guest“ može komentirati pojedinačni članak, a korisnik s ulogom „editor“ može objavljivati nove članke te ažurirati postojeće. Korisnik s ulogom „admin“ može objavljivati, uređivati i brisati postojeće članke. Kako bi se osiguralo da svaki korisnik unutar baze podataka ima jednaku strukturu podataka koristi se paket *Mongoose*. Između ostalog, ovaj paket omogućuje i stvaranje sheme modela za određeni tip dokumenta unutar baze podataka, u ovom slučaju korisnika. *Users* kolekcija predstavlja jedan od modela unutar ovoga projekta. Modeli se definiraju pomoću ranije spomenute sheme koja omogućuje definiranje tipova podataka i određenih restrikcija/validacija polja unutar svakog dokumenta. Slikom 4.3. prikazana je shema i model korisnika.



```
import mongoose from "mongoose";
import { hashPassword } from "../utils/helpers";
import isEmail from "validator/lib/isEmail";

interface UserInput {
  userName?: string;
  alias: string;
  email: string;
  password: string;
}

interface UserDocument extends UserInput, mongoose.Document {
  role: string;
  _id: string;
  createdAt: Date;
  updatedAt: Date;
}

const usersSchema = new mongoose.Schema(
  {
    userName: {
      type: String,
      required: false,
    },
    alias: {
      type: String,
      required: [true, "Please enter alias"],
    },
    email: {
      type: String,
      required: [true, "Please enter email"],
      unique: true,
      validate: [isEmail, "Please enter a valid email address"],
      lowercase: true,
    },
    role: {
      type: String,
      enum: ["guest", "editor", "admin"],
      required: true,
      default: "guest",
    },
    password: {
      type: String,
      required: [true, "Please enter password"],
      minlength: [6, "Minimum password length is 6 characters"],
    },
  },
  { timestamps: true }
);

usersSchema.pre("save", async function (this: UserDocument, next) {
  const hashedPass = await hashPassword(this.password);
  this.password = hashedPass;
  next();
});

const userModel = mongoose.model<UserDocument>("user", usersSchema);

export { userModel, UserDocument, UserInput };
```

Slika 4.3. *Prikaz modela korisnika za kolekciju users.*

Predložak za registraciju korisnika prikazan je slikom 4.4. na kojoj je vidljivo da je za registraciju potrebno unijeti valjani email, alias kako bi čitatelji znali tko je napisao članak i zaporku (engl. *password*) dok je unos korisničkog imena neobavezan.

```jsx
return (
  <div className="create">
    <h2>Register</h2>
    <form onSubmit={handleSubmit}>
      <label>User name:</label>
      <input
        type="text"
        value={userName}
        onChange={(e) => setUserName(e.target.value)}
        placeholder="this field is optional"
      />
      <label>Alias:</label>
      <input type="text" minLength={2} required value={alias} onChange={(e) => setAlias(e.target.value)} />
      <label>Email:</label>
      <input type="email" required value={email} onChange={(e) => setEmail(e.target.value)} />
      <label>Password:</label>
      <input type="password" minLength={6} required value={password} onChange={(e) => setPassword(e.target.value)} />
      {!isPending && <button>Register</button>}
    </form>
    {isPending && <p>Loading...</p>}
    {error && <p className="errorMessage">{error}</p>}
  </div>
);
```

Slika 4.4. *Prikaz predloška za registraciju*

Nakon uspješnog popunjavanja forme i pritiska na gumb *Register* šalje se HTTP POST zahtjev prema poslužitelju s unesenim podacima.

```jsx
registerRouter.post("/register", createAccountLimiter, registerUser);
```

Slika 4.5. *Prikaz rute za registraciju*

Na slici 4.5. prikazana je ruta, odnosno, pristupna točka za registraciju. *Middleware „createAccountLimiter“* je odgovoran za ograničavanje korisniku koliko korisničkih računa se može napraviti unutar određenog vremena. Kako bi se postigla opisana funkcionalnost korišten je paket *express-rate-limit*. Nakon toga zahtjev je proslijeđen kontroler funkciji *„registerUser“* koja je prikazana u nastavku.

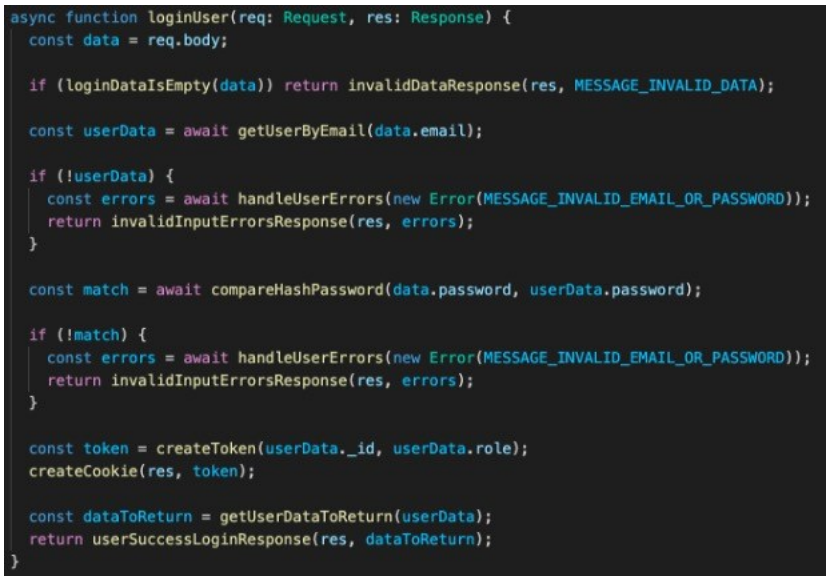
```jsx
async function registerUser(req: Request, res: Response) {
  const data = req.body;

  if (registerDataIsEmpty(data)) return invalidDataResponse(res, MESSAGE_FILL_ALL_FIELDS);

  try {
    const user = await addUserToDatabase(data);
    const token = createToken(user._id, user.role);
    createCookie(res, token);

    const dataToReturn = getUserDataToReturn(user);

    return userSuccessRegisterResponse(res, dataToReturn);
  } catch (error) {
    const errors = await handleUserErrors(error);

    return invalidInputErrorsResponse(res, errors);
  }
}
```

Slika 4.6. *Prikaz kontroler funkcije odgovornu za registraciju korisnika*

Slika 4.6. prikazuje kontroler funkciju koja je odgovorna za logiku registracije korisnika. Najprije se dohvaćaju podaci iz zahtjeva, odnosno podaci koji su poslani kroz klijentsku aplikaciju. Zatim, provjerava se jesu li svi podaci ispunjeni. Ukoliko neki podatak nedostaje moglo bi doći do neispravnog rada aplikacije. Ako je sve ispravno, korisnički podaci se dodaju u bazu te se kreira *token* koji je potreban kako bi aplikacija znala da je korisnik prijavljen te mu na osnovu toga pružila sve funkcionalnosti koje bi korisnik trebao imati. Kreira se kolačić (engl. *cookie*) pod nazivom *cobeCookie* unutar kojeg se sprema stvoreni *token* te se nakon toga, ukoliko nije bilo nikakvih pogrešaka, podaci šalju na klijentsku stranu zajedno s tokenom i cookie-em koji su definirani i podešeni. Ukoliko je sve prošlo bez pogrešaka, korisnik je u mogućnosti koristiti svoje podatke prilikom iduće prijave u aplikaciju. Važno je naglasiti kako se korisnička zaporka provlači kroz *hash* algoritam pomoću poznatog paketa *bcrypt* te se na taj način štiti od moguće zlouporabe. Slikom 4.7. prikazan je primjer MongoDB dokumenta za kolekciju *users*.

Slika 4.7. *Prikaz dokumenta koji predstavlja korisnika unutar kolekcije users*

Nakon uspješne registracije korisnik dobije zadanu vrijednost za ulogu a to je „guest“ te se može prijaviti unutar aplikacije i komentirati postavljene članke. Prilikom prijave provjerava se ispravnost unesenih podataka te ukoliko korisnik s unesenim podacima postoji unutar baze podataka, korisniku se dodjeljuju prava unutar aplikacije ovisno o ulozi. Slikom 4.8. prikazana je kontroler funkcija odgovorna za prijavu korisnika.



Slika 4.8. *Prikaz kontroler funkcije odgovorne za prijavu korisnika*

Ukoliko uneseni podaci nisu ispravni ispisati će se poruka o pogrešci. Prilikom prijave također se radi o POST HTTP zahtjevu kao i u slučaju registracije.

4.3. Dohvaćanje, kreiranje, ažuriranje i brisanje članaka

Prilikom pristupa stranici prikazano je nekoliko članaka iz svake kategorije. Ukoliko korisnik nije prijavljen unutar aplikacije, broj pogleda svakog pojedinog članka neće biti prikazan. Dodatno, neprijavljeni korisnik nije u mogućnosti komentirati članak. Slikama 4.9., 4.10., 4.11. i 4.12. prikazana je komponenta početne stranice kao i ostale korištene komponente kako bi se željena reprezentacija aplikacije ostvarila.

Slika 4.9. *Prikaz komponente početne stranice*

```jsx
const BlogList = ({ blogs, breaking }) => {
  return (
    <div className="news">
      {breaking && <BreakingNews breaking={breaking} />}
      {blogs.length &&
        blogs.map((categoryBlogs, index) => {
          if (categoryBlogs.posts.length !== 0) {
            return (
              <div key={index} className="categoryGroup">
                {categoryBlogs.posts.length !== 0 && (
                  <Link to={`/${categoryBlogs.category.toLowerCase()}`}>
                    <p id="categoryDisplay" className={`${categoryBlogs.category.toLowerCase()}HomePage`}>
                      {categoryBlogs.category.charAt(0).toUpperCase() + categoryBlogs.category.slice(1)}
                    </p>
                  </Link>
                )}
                <div className="categoryNews">
                  <BlogCategory blogs={categoryBlogs.posts} />
                </div>
              </div>
            );
          } else {
            return <p className="notDisplayed" key={index}></p>;
          }
        })}
    </div>
  );
};
```

Slika 4.10. *Prikaz komponente zaslužne za prikazivanje svih članaka na početnoj stranici*

```jsx
const BreakingNews = ({ breaking }) => {
  return (
    <div className="breaking">
      <Link to={`/single/${breaking._id}`}>
        <div className="wholeBreaking grid">
          <div className="breakingPictureAndHeadline">
            <h2>BREAKING NEWS</h2>
            <img src={breaking.image} alt="newsPicture" id="breakingPicture" loading="lazy" />
          </div>
          <div className="breakingDesc">
            <div className="breakingEssentialInfo">
              <p className="timeCreated">{formatDistance(new Date(breaking.createdAt), new Date(), { addSuffix: true })}</p>
              <h2>{breaking.headline}</h2>
              <p className="shortDesc">{breaking.shortDescription}</p>
            </div>
            {"views" in breaking && (
              <div className="additionalInfo viewNumbers">
                <ViewCommentNumbers news={breaking} />
              </div>
            )}
          </div>
        </div>
      </Link>
    </div>
  );
```

Slika 4.11. *Prikaz komponente zadužene za prikazivanje prijelomne vijesti (engl. breaking news)*

```jsx
const BlogCategory = ({ blogs }) => {
  return blogs.map((blog) => {
    return (
      <Link to={`/single/${blog._id}`} key={blog._id}>
        <div className="blog-preview">
          <img src={blog.image} alt="newsPicture" id="newsPicture" loading="lazy" />
          <div>
            <div className={`blog-previewDetails ${blog.category.toLowerCase()}Preview`}>
              <h2>{blog.headline}</h2>
            </div>
            <div className="viewNumbers">
              <ViewCommentNumbers news={blog} />
            </div>
          </div>
        </div>
      </Link>
    );
  });
};
```

Slika 4.12. *Prikaz komponente za prikazivanje niza članaka*

Važno je primijetiti da svi predlošci primaju određene podatke kako bi obavili svoj posao, a takvi proslijeđeni podaci nazivaju se rekviziti (engl. *props*). Kada se prosljeđivanje podataka ne bi koristilo, bilo bi potrebno dohvaćati podatke ponovno unutar svake komponente, a takav pristup je iz očitih razloga vrlo loš. Glavna prednost korištenja rekvizita je ta što na taj način komponenta postaje ponovno upotrebljiva, odnosno, ne ovisi o podacima koje prikazuje pa ju je moguće koristiti unutar cijele aplikacije gdje je potrebno. U primjeru ove aplikacije podaci se dohvaćaju samo unutar *„Home“* komponente te su zatim proslijeđene kao rekviziti ostalim komponentama kako bi se ostvario željeni izgled početne stranice.

```jsx
postsRouter.post("/", requireEditorOrAdmin, httpSavePost);
postsRouter.post("/search", searchPosts);
postsRouter.post("/searchTitle", searchPostByTitle);

postsRouter.put("/:id", requireEditorOrAdmin, httpUpdatePost);

postsRouter.get("/", httpGetAllPosts);
postsRouter.get("/breaking", httpGetBreakingNews);
postsRouter.get("/:id", httpGetPost);
postsRouter.get("/category/:numberOfPosts", getNewestPosts);

postsRouter.delete("/:id", requireAdmin, httpDeletePost);
```

Slika 4.13. *Prikaz mogućih HTTP zahtjeva i ruta za članke*

Slika 3.13. prikazuje sve moguće upite poslužitelju vezano uz članke. Važno je primijetiti kako neke od ruta koriste *middleware* koji provjerava je li korisnik „editor“ ili „admin“.

```
async function httpGetBreakingNews(req: Request, res: Response) {
  const isLogged = isLoggedIn(req);
  const breakingNews = await getBreakingNews(isLogged);

  if (!breakingNews) return notFoundResponse(res, MESSAGE_NO_BREAKING_NEWS);

  return successBreakingNewsResponse(res, breakingNews);
}
```

Slika 4.14. *Prikaz kontroler funcije odgovorne za dohvaćanje prijelomne vijesti*

Slikom 4.14. prikazana je kontroler funkcija koja je odgovorna za dohvaćanje prijelomne vijesti. Prvo je potrebno ustanoviti o kakvom se korisniku radi, odnosno, ustanoviti je li korisnik prijavljen ili ne kako bi se dohvatili ispravni podaci. Ukoliko prijelomna vijest nije pronađena unutar baze kao odgovor se šalje poruka o pogrešci. Slika 4.15. prikazuje funkciju koja je zadužena za dohvaćanje prijelomne vijesti iz baze koristeći se modelom za vijesti.

```
async function getBreakingNews(loggedIn: boolean): Promise<PostDocument[] | null> {
  convertOldBreakingNews();
  const options = loggedIn ? { __v: 0 } : { __v: 0, views: 0 };
  const post = await postModel.aggregate([
    {
      $match: {
        createdAt: { $gte: new Date(Date.now() - appConstants.twoDaysInMs) },
        breakingNews: true,
      },
    },
    { $sort: { createdAt: -1 } },
    { $limit: 1 },
    { $project: options },
  ]);

  if (!post[0]) return null;
  const comments = await getPostComments(post[0]._id);
  post[0].commentsId = comments;
  return post[0]!;
}
```

Slika 4.15. *Prikaz funkcije zadužene za dohvaćanje prijelomne vijesti iz baze podataka (MongoDB)*

Najprije se vrši poziv funkcije za provjeru je li neka prijelomna vijest starija od dva dana, a ukoliko je, takva vijest više nije prijelomna te se na adekvatan način ažurira. Tada je potrebno provjeriti je li korisnik prijavljen te na osnovu tog podatka dohvatiti prijelomnu vijest s ili bez podataka o broju pogleda jer korisnici koji nisu prijavljeni nemaju pristup tom podatku.

```
async function getNewestPosts(req: Request, res: Response) {
  const { numberOfPosts } = req.params;
  const isLogged = isLoggedIn(req);
  const posts = await getPostsOfEachCategory(Number(numberOfPosts), isLogged);
  return successPostResponse(res, posts);
}
```

Slika 4.16. *Prikaz kontroler funkcije za dohvaćanje određeni broj najnovijih vijesti za svaku kategoriju*

Na slici 4.16. prikazana je kontroler funkcija koja je zadužena za dohvaćanje određenog broja najnovijih vijesti od svake kategorije gdje se broj vijesti definira zahtjevom na poslužitelja. Kao i kod prijelomnih vijesti, potrebno je provjeriti je li korisnik prijavljen te nakon provjere dohvatiti željene članke.

Korisnik osim pregleda članaka na početnoj stranici ima mogućnost klikom na pojedini članak otvoriti isti u novom prikazu. Prikaz članka daje uvid u kategoriju kojoj pripada, naslovu, autoru članka, kojeg dana i u koliko sati je članak napisan što su korisne informacije jer daju korisniku do znanja radi li se o staroj ili novijoj vijesti. Nadalje, prikazana je slika, glavna priča (odnosno tekst članka) te podatak tko je zadnji ažurirao članak. Ukoliko je korisnik urednik tada se prikazuje i mogućnost ažuriranja članka, a korisnik s ulogom administratora može i obrisati članak. Slikom 4.17. prikazan je predložak komponente zaduženog za prikaz pojedinog članka.

Slika 4.17. *Prikaz predloška za prikaz pojedinačnog članka*

```
async function httpGetPost(req: Request, res: Response) {
  const { id } = req.params;

  try {
    await incrementPostViews(id);

    const post = await getPost(id, await isGuest(req));

    if (post) return successPostResponse(res, post);
    return invalidDataResponse(res, MESSAGE_INVALID_POST_ID);
  } catch (err) {
    return invalidDataResponse(res, MESSAGE_INVALID_POST_ID);
  }
}
```

Slika 4.18. *Prikaz kontroler funkcije za dohvaćanje pojedine vijesti*

Slikom 4.18. prikazana je kontroler funkcija koja je odgovorna za dohvaćanje pojedine vijesti. Dohvaćanje pojedine vijesti vrši se pomoću proslijeđenog ID-a. Najprije je potrebno povećati broj pogleda za jedan (ako se želi dohvatiti vijest to znači da će biti pogledana), a zatim je potrebno kao i u prošloj kontroler funkciji provjeriti radi li se o prijavljenom ili neprijavljenom korisniku. Ukoliko je sve bilo ispravno i vijest je dohvaćena, ona se šalje kao odgovor na zahtjev te se prikazuje na prikladan način.

Korisnik također ima mogućnost pregleda vijesti prema kategoriji. Kategorije koje postoje su sljedeće: Sport, Posao, Tehnologija, Znanost i Općenito (engl. *Sport*, *Business*, *Technology*, *Science* i *General*). Za svaku od navedenih kategorija koristi se ista komponenta, odnosno, isti predložak.

```
return (
  <div className="news">
    {(blogs.length !== 0 || (newestBlog && Object.keys(newestBlog).length !== 0)) && (
      <div className="lastNews">
        <div id="categoryDisplay" className={category.toLowerCase()}>
          <p>{category}</p>
        </div>
        <div className="breaking">
          <Link to={`/single/${newestBlog._id}`}>
            <div className="wholeBreaking lastNews">
              <div className="breakingPictureAndHeadline">
                <img src={newestBlog.image} alt="newsImage" id="breakingPicture" loading="lazy" />
              </div>
              <div>
                <div className="breakingEssentialInfo">
                  <h2>{newestBlog.headline}</h2>
                  <p className="shortDesc">{newestBlog.shortDescription}</p>
                </div>
                <div className="viewNumbers newestBlogNumbers">
                  <ViewCommentNumbers news={newestBlog} />
                </div>
              </div>
            </div>
          </Link>
        </div>
        <div className="categoryNews">
          <div className="categoryNews categoryPage">
            <BlogCategory blogs={blogs} />
          </div>
        </div>
        {pageCount && (
          <div className="pagination-container">
            <PaginationList pageCount={pageCount} currentPage={currentPage} setCurrentPage={setCurrentPage} />
          </div>
        )}
      </div>
    )}

    {isPending && <p>Loading...</p>}
    {error && <p className="errorMessage">{error}</p>}
  </div>
);
```

Slika 4.19. *Prikaz predloška za pojedinu kategoriju*

Slika 4.19. prikazuje predložak koji se koristi za prikaz vijesti iz pojedine kategorije. Može se uočiti kako se koristi komponenta prikazana slikom 4.12. te upravo u ovome primjeru dolazi do izražaja prednost korištenja Reacta, odnosno ponovna iskoristivost komponenti na različitim mjestima. Kod prikazivanja vijesti iz kategorije, ona vijest koja je najnovija prikazuje se uvećano, a ostale vijesti su prikazane nakon najnovije vijesti. Također, kao i na početnoj stranici, klikom se prikazuje vijest i svi podaci vezani za nju, no kod prikaza vijesti iz kategorije ne prikazuju se samo vijesti koje su objavljene na aplikaciji već se prikazuju i vijesti s drugih portala koji su dohvaćani pomoću javno dostupnog API-a. Cijeli postupak je objašnjen u nastavku.

```
const searchPosts = async (req: Request, res: Response, next: NextFunction) => {
  const { category } = req.body;
  const { pageSize, skip } = getPaginationData(req);

  try {
    const { posts, postCount } = await getPostsByCategory(category, pageSize, skip);

    return successPostResponse(res, { posts, paginationData: getPaginationDetails(pageSize, postCount) });
  } catch (error) {
    return invalidDataResponse(res, MESSAGE_INVALID_CATEGORY);
  }
};
```

Slika 4.20. *Prikaz kontroler funkcije za dohvaćanje vijesti iz određene kategorije*

Slikom 4.20. prikazana je kontroler funkcija koja je zadužena za dohvaćanje vijesti iz odabrane kategorije. Osim željene kategorije, funkcija kroz zahtjev prima i podatke o broju vijesti iz kategorije kako se ne bi dohvaćale sve vijesti ukoliko to nije potrebno nego npr. dvadeset najnovijih.

Kako bi se dohvatile vijesti preko ranije spomenutog javno dostupnog API-a koristi se npm paket nazvan *node-cron*. Paket omogućuje odrađivanje neke definirane aktivnosti periodično. U slučaju ove aplikacije svakih sat vremena dohvaćaju se vijesti s https://newsapi.org/. Navedena stranica dohvaća vijesti s različitih portala te ih nudi kroz definirani API gdje je potreban ključ kako bi mu se moglo pristupiti (engl. *API Key*).

```
function getNews() {
  cron.schedule("0 * * * *", () => {
    fetchNews();
  });
}
```

Slika 4.21. *Prikaz način korištenja node-cron paketa*

Slikom 4.21. prikazuje se korištenje *node-cron* paketa. Funkcija „*fetchNews*“ izvesti će se svaki sat.

```
async function fetchNews() {
  if (!newsApi) newsApi = new NewsAPI(process.env.NEWS_API_KEY!);

  fetchBreakingNews();

  for (let j = 0; j < appConstants.allCategories.length - 1; j++) {
    newsApi.v2
      .topHeadlines({
        category: appConstants.allCategories[j],
        language: "en",
        country: "us",
        pageSize: 4,
      })
      .then((response: any) => {
        mapPosts(response, appConstants.allCategories[j], false);
      });
  }
}
```

Slika 4.22. *Prikaz funkcije za dohvaćanje vijesti s drugih portala*

Slika 4.22. prikazuje funkciju koja dohvaća vijesti pomoću javno dostupnog API-a. Kako bi pristupili podacima koje nudi NewsAPI stranica potrebno je koristiti vlastiti API ključ koji je pohranjen unutar *.env* datoteke kako bi ostao skriven. Najprije se dohvaća jedna prijelomna vijest pozivom funkcije „*fetchBreakingNews*“, a nakon toga se za svaku definiranu kategoriju dohvaća po četiri članka i podaci se prilagođavaju, odnosno „mapiraju“ kako bi odgovarali definiranoj shemi koja je prikazana u nastavku.

```
function mapPosts<T extends NewsApiNews>(response: T, category: string, breakingNews: boolean) {
  if (breakingNews) return mapBreaking(response, category);

  response.articles.forEach((article) => {
    let index = 0;
    let fullDescription = "";
    if (article.content) {
      index = article.content.indexOf("[");
      fullDescription = article.content.substring(0, index - 1);
    }
    if (!article.description || !fullDescription || !article.urlToImage || !isURL(article.urlToImage)) return;

    const post: PostInput = {
      headline: article.title,
      shortDescription: article.description,
      fullDescription: fullDescription,
      image: article.urlToImage,
      category: category,
      author: article.source.name,
      lastUpdatedBy: article.source.name,
      url: article.url,
    };
    savePost(post);
  });
}
```

Slika 4.23. *Prikaz pomoćne funkcije za prilagođavanje podataka*

Slikom 4.23. prikazana je funkcija kojom se dohvaćeni podaci prilagođavaju strukturi definiranoj shemom. Nakon kreiranja novog objekta koji odgovara definiranoj strukturi podaci se spremaju unutar baze podataka.

```
interface PostInput {
  author: string;
  lastUpdatedBy: string;
  headline: string;
  shortDescription: string;
  fullDescription: string;
  image: string;
  category: string;
  breakingNews?: boolean;
  url?: string;
}

interface PostDocument extends PostInput, mongoose.Document {
  views: number;
  _id: string;
  createdAt: Date;
  updatedAt: Date;
}
```

Slika 4.24. *Prikaz sučelja koja definiraju strukturu ulaznih i izlaznih podataka*

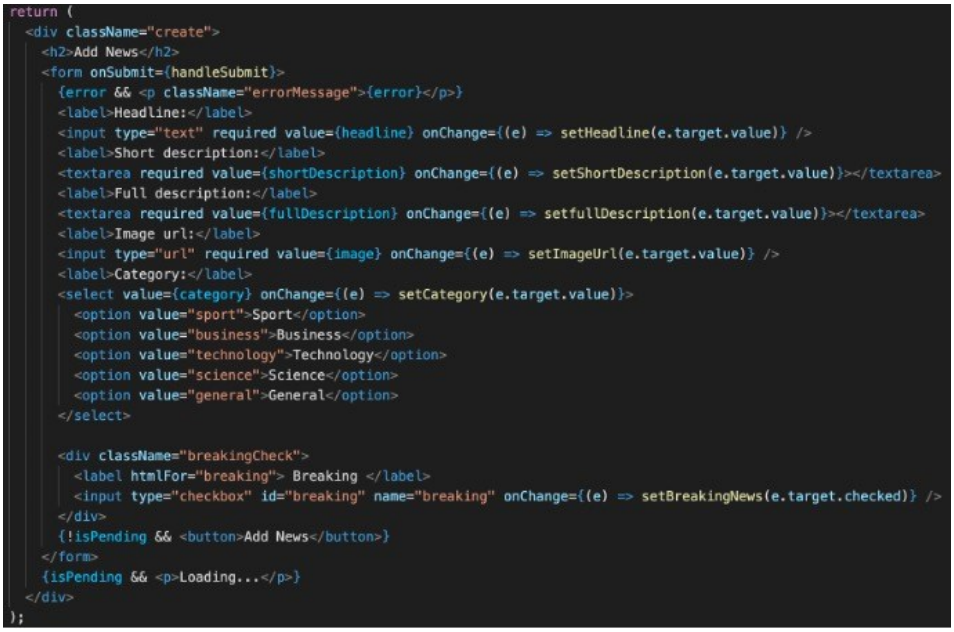
```
const postsSchema = new mongoose.Schema(
  {
    author: {
      type: String,
      required: [true, "Please enter an author"],
    },
    lastUpdatedBy: {
      type: String,
      required: true,
    },
    headline: {
      type: String,
      required: [true, "Please enter headline"],
    },
    shortDescription: {
      type: String,
      required: [true, "Please enter short description"],
    },
    fullDescription: {
      type: String,
      required: [true, "Please enter full description"],
    },
    image: {
      type: String,
      required: [true, "Please enter image url"],
      validate: [isURL, "Please, enter valid url"],
    },
    category: {
      type: String,
      required: [true, "Please enter category"],
      lowercase: true,
    },
    breakingNews: {
      type: Boolean,
      required: false,
      default: false,
    },
    views: {
      type: Number,
      required: true,
      default: 0,
    },
    url: {
      type: String,
      required: false,
    },
  },
  { timestamps: true }
);
```

Slika 4.25. *Prikaz sheme za pojedinu vijest*

Slikom 4.24. prikazana su sučelja koja definiraju strukturu podataka koji trebaju biti primljeni od strane klijentske aplikacije (PostInput), te strukturu koju će imati podaci kada budu dohvaćeni iz baze podataka (PostDocument). Podatak *„url“* i *„breakingNews“* su neobavezni jer imaju svoju zadanu vrijednost pa ih nije potrebno prosljeđivati. Nadalje, slika 4.25. prikazuje sve podatke, njihove tipove, zadanu vrijednost i provjeru ispravnosti dolazećih podataka. Važno je napomenuti

da članci koji su kreirani u aplikaciji neće imati podatak *„url"* koji definira poveznicu na vijest, već se ovaj podatak koristi za vijesti s drugih portala. Pomoću ovog podatka se uvjetno prikazuje poveznica na članak koji se nalazi na nekom drugom portalu. Podaci s drugih portala ne mogu se uređivati niti brisati već se mogu pročitati ili posjetiti poveznicu na portal autora članka.

Dodavanje novih vijesti ostvaruje se pomoću forme u kojoj je potrebno popuniti određene podatke. Samo „editor" i „admin" korisnici mogu kreirati novi članak. Slikom 4.26. prikazan je predložak za kreiranje novih vijesti. Za uspješno kreiranje novog članka potrebno je popuniti podatke poput naslova, kratkog opisa, dugog opisa odnosno glavnog teksta članka, link slike te izabrati kategoriju i označiti radi li se o prijelomnoj vijesti ili ne. Kako je prikazano i shemom, link na sliku mora biti ispravnog oblika, a u suprotnom cijeli članak neće biti moguće pohraniti unutar baze podataka. Prilikom kreiranja novog članka koristi se POST zahtjev prema poslužitelju.



```
return (
  <div className="create">
    <h2>Add News</h2>
    <form onSubmit={handleSubmit}>
      {error && <p className="errorMessage">{error}</p>}
      <label>Headline:</label>
      <input type="text" required value={headline} onChange={(e) => setHeadline(e.target.value)} />
      <label>Short description:</label>
      <textarea required value={shortDescription} onChange={(e) => setShortDescription(e.target.value)}></textarea>
      <label>Full description:</label>
      <textarea required value={fullDescription} onChange={(e) => setfullDescription(e.target.value)}></textarea>
      <label>Image url:</label>
      <input type="url" required value={image} onChange={(e) => setImageUrl(e.target.value)} />
      <label>Category:</label>
      <select value={category} onChange={(e) => setCategory(e.target.value)}>
        <option value="sport">Sport</option>
        <option value="business">Business</option>
        <option value="technology">Technology</option>
        <option value="science">Science</option>
        <option value="general">General</option>
      </select>

      <div className="breakingCheck">
        <label htmlFor="breaking"> Breaking </label>
        <input type="checkbox" id="breaking" name="breaking" onChange={(e) => setBreakingNews(e.target.checked)} />
      </div>
      {!isPending && <button>Add News</button>}
    </form>
    {isPending && <p>Loading...</p>}
  </div>
);
```

Slika 4.26. *Prikaz predloška za kreiranje novih vijesti*

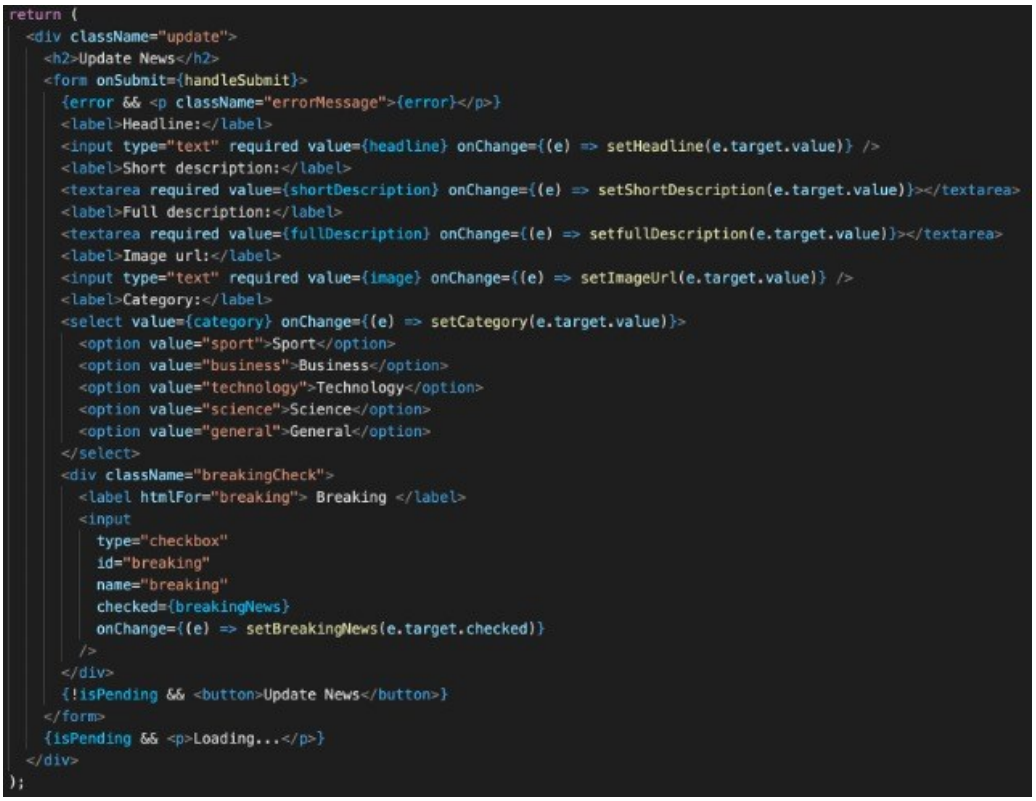
```
async function httpSavePost(req: Request, res: Response) {
  const author = await getUserName(req);

  if (!author) return getDataErrorResponse(res, MESSAGE_GET_USER_ERROR);

  const post = Object.assign(req.body, { author, lastUpdatedBy: author });

  try {
    const savedPost = await savePost(post);

    if (savedPost) return successPostResponse(res, savedPost);
  } catch (err) {
    const errors = handleErrors(err, "post");
    invalidInputErrorsResponse(res, errors);
  }
}
```

Slika 4.27. *Prikaz kontroler funkcije odgovorne za spremanje novih vijesti*

Slikom 4.27. prikazana je kontroler funkcija koja sadrži svu logiku potrebnu za spremanje nove vijesti. Potrebno je dohvatiti ime korisničko ime trenutno prijavljenog korisnika kako bi se taj podatak mogao spremiti uz ostale podatke članka. Slikom 4.28. prikazan je primjer spremljenog članka unutar MongoDB baze podataka.


```
_id: ObjectId("62ae2f21d60e030c2c48fdab")
author: "someAlias"
lastUpdatedBy: "someAlias"
headline: "This is groundbreaking! I just made this headline a little longer so i..."
shortDescription: "Ipsum has been the industry's standard dummy text ever since the 1500s..."
fullDescription: "Ipsum has been the industry's standard dummy text ever since the 1500s..."
image: "https://cdn.pixabay.com/photo/2015/04/23/22/00/tree-736885__480.jpg"
category: "sport"
breakingNews: false
views: 96
createdAt: 2022-06-21T20:01:37.314+00:00
updatedAt: 2022-06-23T20:28:38.622+00:00
__v: 0
```


Slika 4.28. *Prikaz dokumenta članka*



Slika 4.29. *Prikaz predloška za ažuriranje članka*

Slikom 4.29. prikazan je predložak forme za ažuriranje članka. Naizgled je isti kao i predložak za dodavanje, no u ovom slučaju su polja popunjena s podacima članka. Za ažuriranje podataka koristi se PUT zahtjev prema poslužitelju.

```
async function httpUpdatePost(req: Request, res: Response) {
  const { id: postId } = req.params;
  const post = await getUpdateData(req);

  try {
    const updatedPost = await updatePost(postId, post);
    return successPostResponse(res, updatedPost!);
  } catch (err: any) {
    if (err._message == "Validation failed") {
      const errors = handleErrors(err, "");
      return invalidInputErrorsResponse(res, errors);
    }
    return updatePostErrorResponse(res, MESSAGE_UPDATE_POST_ERROR);
  }
}
```

Slika 4.30. *Prikaz kontroler funkcije za ažuriranje vijesti*

Slikom 4.30. prikazana je kontroler funkcija koja je zadužena za ažuriranje podataka određenog članka. Podatke mogu ažurirati samo urednici ili korisnici s ulogom administratora. Najprije je potrebno dohvatiti sve podatke koji su poslani zahtjevom, a tada se ti isti podaci šalju prema bazi

podataka gdje se, ako je sve u redu, podaci ažuriraju. U suprotnom se prema klijentskoj aplikacije šalje poruka o pogrešci.

Osim dohvaćanja, kreiranja i ažuriranja, korisnici s ulogom administratora također mogu i obrisati članke. Jednostavnim klikom na gumb *Delete*, koji se nalazi na predlošku prikazan slikom 4.17., šalje se DELETE zahtjev prema poslužitelju i ukoliko članak s predanim identifikacijskim nizom znakova (skraćeno ID) postoji unutar baze podataka, briše se. Važno je napomenuti da gumb za brisanje neće biti prikazan svim korisnicima već samo onima koji zadovoljavaju uvjete, odnosno samo korisnicima koji imaju ulogu „admin“. Također, osim provjere i ograničenja na klijentskoj strani, provjera se vrši i na poslužiteljskoj strani. Slika 4.31. prikazuje kontroler funkciju zaduženu za upravljanje zahtjevom za brisanje. Prilikom brisanja članka brišu se i svi pripadajući komentari.

```
async function httpDeletePost(req: Request, res: Response) {
  const { id } = req.params;

  try {
    const response = await deletePost(id);
    if (response) {
      deletePostComments(id);
      return postDeletedResponse(response, res, MESSAGE_POST_DELETED_SUCCESS);
    }

    invalidDataResponse(res, MESSAGE_INVALID_POST_ID);
  } catch (err) {
    return invalidDataResponse(res, MESSAGE_INVALID_POST_ID);
  }
}
```

Slika 4.31. *Prikaz kontroler funckije za brisanje vijesti*

4.4. Komentiranje članaka

Svaki korisnik koji je prijavljen u aplikaciji ima mogućnost komentirati objavljene članke, dok neprijavljeni korisnici nemaju mogućnost komentiranja već mogu samo pregledavati članak. Komentiranje je moguće u prikazu pojedinog članka na dnu stranice. Za komentare je kreirana nova kolekcija pod nazivom *comments*. Svaki dokument unutar kolekcije sadrži polje komentara, gdje je svaki komentar zaseban objekt, odnosno dokument, što ovakvu strukturu podataka čini nešto kompliciranijom nego li je to bilo u prijašnjim kolekcijama. Također, sadrži i ID članka kojem pripadaju te se na taj način dohvaćaju komentari za odgovarajući članak. Slikom 4.32. prikazana je shema jednog komentara i shema dokumenta koja sadržava polje komentara od kojih je svaki prikazan kao zaseban dokument.

```
const commentSchema = new mongoose.Schema(
  {
    comment: {
      type: String,
      required: [true, "Please enter comment"],
      trim: true,
    },
    userName: {
      type: String,
      required: [true, "Please enter user name"],
      default: " ",
    },
    userId: {
      type: Schema.Types.ObjectId,
      required: true,
    },
  },
  { timestamps: true }
);

const commentsSchema = new mongoose.Schema(
  {
    comments: [commentSchema],
    postId: {
      type: String,
      required: [true, "Please enter postId"],
    },
  },
  { timestamps: true }
);
```

Slika 4.32. *Prikaz shema za komentare*

Kako bi se svaki komentar uspješno spremio u bazu, potrebno je predati komentar, ime korisnika koji je napisao komentar, njegov pripadajući ID i ID članka kojeg se komentira. Slikom 4.33. prikazani su mogući upiti koji su vezani uz komentare.

```
commentsRouter.post("/:postId", requireAuthentication, httpAddComment);
commentsRouter.get("/:postId", httpGetComments);
commentsRouter.delete("/:postId/:commentId", requireAdmin, httpDeleteComment);
commentsRouter.put("/:postId/:commentId", requireAuthentication, httpUpdateComment);
```

Slika 4.33. *Prikaz mogućih HTTP zahtjeva i ruta za komentare*

```
async function httpAddComment(req: Request, res: Response) {
  const { postId } = req.params;
  const commentData: CommentsInput = await getCommentData(req, postId);

  if (!(await postExists(postId))) return invalidDataResponse(res, MESSAGE_INVALID_POST_ID);

  try {
    const postComments = await getPostComments(postId);

    if (!postComments) {
      const post = await addComment(commentData);

      const fetched = await getPost(postId);
      fetched.commentsId = post._id as any;
      fetched.save();

      return successAddCommentResponse(res, post.comments);
    }
    postComments.comments.push(req.body);

    let savedPost = await saveComment(postComments);
    return successAddCommentResponse(res, savedPost.comments.reverse());
  } catch (err) {
    const errors = handleErrors(err, "comment");
    return invalidInputErrorsResponse(res, errors);
  }
}
```

Slika 4.34. *Prikaz kontroler funkcije za dodavanje komentara*

Slika 4.34. prikazuje kontroler funkciju odgovornu za dodavanje komentara. Najprije je potrebno provjeriti postoji li članak koji se pokušava komentirati jer postoji mogućnost da se radi o pogrešci gdje korisnik pokušava komentirati članak koji je nedavno obrisan. Ukoliko članak postoji, dohvaća se cijeli dokument komentara preko ID-a članka. Tada se unutar polja komentara „ubacuje“ novi komentar i takav ažurirani dokument se sprema u bazu podataka.

```
_id: ObjectId("61ab51e84d306ae5b69ebda5")
comments: Array
  0: Object
      comment: "komentar2"
      userName: "someAlias"
      userId: ObjectId("618edc620592ad61ceb5255b")
      _id: ObjectId("61ab51e84d306ae5b69ebda6")
      createdAt: 2021-12-04T11:32:56.025+00:00
      updatedAt: 2021-12-04T11:32:56.025+00:00
postId: "61950345384188e72e71f57a"
createdAt: 2021-12-04T11:32:56.026+00:00
updatedAt: 2021-12-04T11:32:56.026+00:00
__v: 0
```

Slika 4.35. *Primjer dokumenta komentara*

Slika 4.35. prikazuje dokument komentara gdje je vidljivo kako se radi o dokumentu koji sadrži više dokumenata unutar polja koji predstavljaju komentare.

Slika 4.36. *Kontroler funkcija za dohvaćanje komentara članka*

Kako bi se dohvatili komentari članka koristi se GET zahtjev prema poslužitelju. Slika 4.36. prikazuje kontroler funkciju koja je odgovorna za obrađivanje navedenog zahtjeva. Prilikom slanja zahtjeva kao parametar potrebno je poslati ID članka kako bi se odgovarajući komentari mogli pronaći u bazi podataka. Kao i kod dodavanja novih komentara, u ovoj funkciji se prvo provjerava postoji li članak čiji se komentari pokušavaju dohvatiti, a nakon toga, ukoliko je sve prošlo bez pogrešaka, komentari se šalju kao odgovor na poslani zahtjev.

Nadalje, kada se komentar želi ažurirati, šalje se PUT zahtjev prema poslužitelju. Slika 4.37. prikazuje funkciju koja obrađuje zahtjev za ažuriranjem komentara.

Slika 4.37. *Kontroler funkcija za ažuriranje komentara*

Da bi se komentar uspješno ažurirao potrebno je kroz zahtjev kao parametre poslati ID komentara, ID članka na koji se komentar odnosi i novi komentar na koju će vrijednost biti ažuriran. Za ažuriranje je zaslužna funkcija „*updateComment"* koja dohvaća komentare iz kolekcije *comments* prema ID-u članka te postavlja vrijednost željenog komentara na novu.

Komentare je moguće i obrisati te su s tim upotpunjene i omogućene sve CRUD (engl. *Create, Read, Update, Delete)* operacije nad kolekcijom komentara.

```
async function httpDeleteComment(req: Request, res: Response) {
  const { postId, commentId } = req.params;
  try {
    const remainingComments = await deleteComment(commentId, postId);

    if (!remainingComments) return invalidDataResponse(res, MESSAGE_INVALID_COMMENT_ID);

    return successDeletedCommentResponse(res, remainingComments);
  } catch (err: any) {
    if (err.message == MESSAGE_COMMENTS_NOT_FOUND) return invalidDataResponse(res, MESSAGE_COMMENTS_NOT_FOUND);
    invalidDataResponse(res, MESSAGE_INVALID_COMMENT_ID);
  }
}
```

Slika 4.38. *Kontroler funkcija za brisanje komentara*

Slikom 4.38. prikazana je kontroler funkcija za brisanje komentara gdje je potrebno kroz parametre predati ID članka i ID komentara koji se želi obrisati. Ovdje je važno napomenuti kako komentare može brisati samo korisnik s ulogom „admin“. Ova funkcionalnost je omogućena kako bi se mogli obrisati neki od komentara koji krše pravila portala.

4.5. Pretraživanje članaka

Korisnicima je omogućeno pretraživanje članaka prema naslovu. Primjerice, ukoliko korisnik želi pronaći sve članke koje sadržavaju riječ „*ball*“ u naslovu tada je potrebno kliknuti na „*search*“ ikonu u izborniku te upisati riječ u za to predviđeno polje.

```
return (
  <>
    <form onSubmit={handleSearch} className="search-form">
      <input type="text" ref={searchBox} placeholder="Search" />
      <img src="search.png" alt="searchIcon" width="20" height="20" onClick={handleSearch} />
    </form>
    {isPending && <p>Searching...</p>}

    {newsCount !== null && <div className="resultsFoundNumber">{<p>{newsCount} results found</p>}</div>}
    <div className="searchNews">
      {news.map((blog) => {
        return (
          <Link to={`/single/${blog._id}`} key={blog._id}>
            <div className="blog-preview">
              <img src={blog.image} alt="newsPicture" id="newsPicture" loading="lazy" />
              <div>
                <div className={`blog-previewDetails ${blog.category.toLowerCase()}Preview`}>
                  <h2>{blog.headline}</h2>
                  <p>{blog.shortDescription}</p>
                </div>
                <p className="datePreview"> {new Date(blog.createdAt).toLocaleDateString()}</p>

                <p id="categoryDisplay" className={`${blog.category.toLowerCase()} search`}>
                  {blog.category.charAt(0).toUpperCase() + blog.category.slice(1)}
                </p>
              </div>
            </div>
          </Link>
        );
      })}
      {pageCount !== 0 && (
        <div className="pagination-container">
          <PaginationList pageCount={pageCount} currentPage={currentPage} setCurrentPage={setCurrentPage} />
        </div>
      )}
    </div>
  </>
);
```

Slika 4.39. Prikaz predloška za pretraživanje

Slikom 4.39. prikazan je predložak za pretraživanje članaka prema njihovom naslovu. Predložak se sastoji od forme za unos ključne riječi, dijela u kojem se prikazuju svi pronađeni članci te kako bi se ograničio broj prikazanih članaka po stranici koristi se paginacija gdje je na svakoj stranici prikazano po 20 članaka.

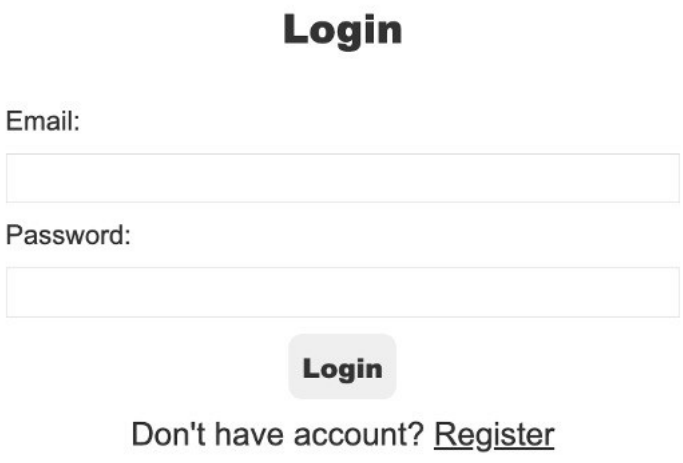
```
const searchPostByTitle = async (req: Request, res: Response) => {
  const { title } = req.body;
  const { pageSize, skip } = getPaginationData(req);

  try {
    const { news, newsCount } = await getNewsByTitle(title, pageSize, skip);
    return successPostResponse(res, { news, paginationData: getPaginationDetails(pageSize, newsCount) });
  } catch (error) {
    return getDataErrorResponse(res, MESSAGE_INVALID_DATA);
  }
```

Slika 4.40. *Prikaz kontroler funkcije za pretragu članaka*

Slikom 4.40. prikazana je kontroler funkcija koja je odgovorna za dohvaćanje željenih vijesti ovisno o unesenom naslovu. Funkcija ovisno o upitu vraća određeni broj vijesti, gdje se broj vijesti i broj stranice definira pomoću parametara upita (engl. *Query parameters*). Nakon pronađenih članaka, vraća se odgovor, koji se sastoji od članaka i dodatnih podataka o paginaciji, odnosno broj članaka koji odgovaraju kriteriju te koja je zadnja stranica na paginaciji. Primjerice, ukoliko se želi prikazati 5 članaka po stranici, no u bazi postoji 19 koji odgovaraju kriteriju tada će 4. stranica biti zadnja.

5. VIZUALNA REPREZENTACIJA

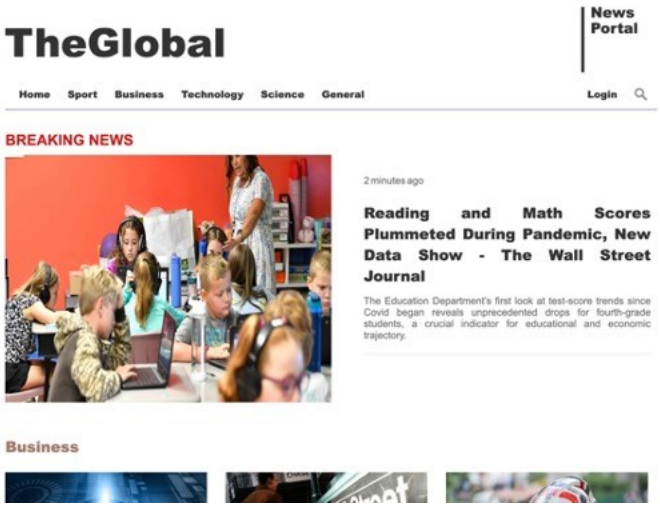
U nastavku je prikazan i prokomentiran izgled aplikacije koja sadržava sve funkcionalnosti koje su ranije u radu navedene i ukratko objašnjene. Na slikama koje slijede prikazana je forma za registraciju i prijavu, forma za kreiranje i ažuriranje vijesti, početna stranica i prikaz vijesti iz pojedine kategoriju u dva stanja – kada je korisnik prijavljen i kada nije kako bi se uvidjela razlika u prikazu za te dvije vrste korisnika.



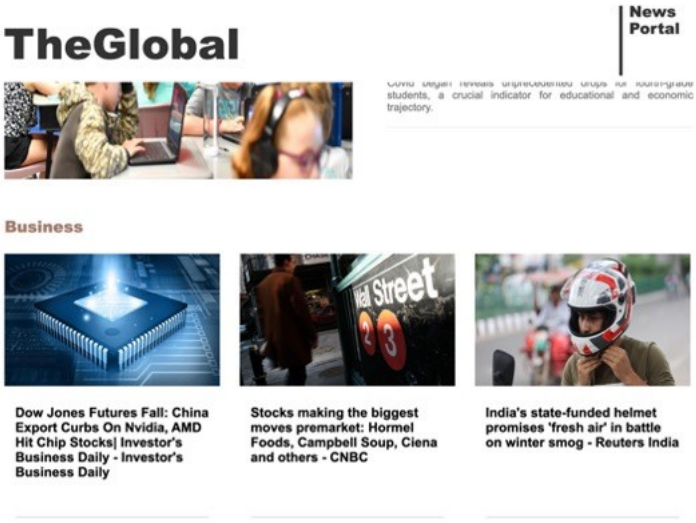
Slika 5.1. *Prikaz forme za prijavu*

Slika 5.2. *Prikaz forme za registraciju*

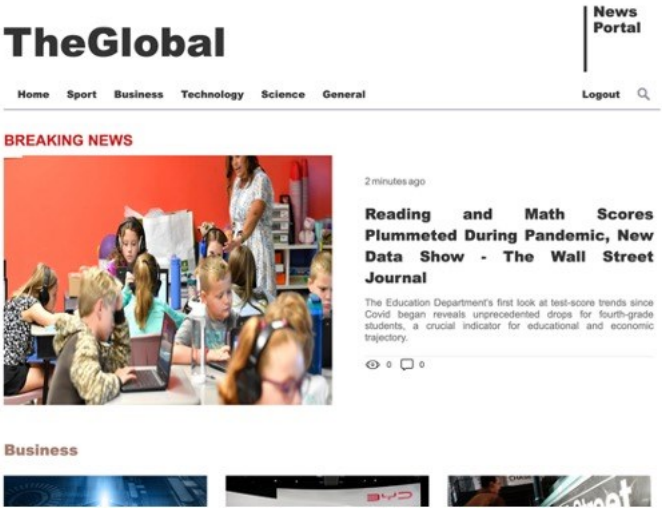
Slika 5.1. prikazuje formu za prijavu u aplikaciju. Sastoji se od dva obavezna polja: email i zaporka. Ukoliko korisnik nije registriran, klikom na poveznicu *Register* prikazuje se forma za registraciju koja je prikazana slikom 5.2.. Forma se sastoji od nekoliko polja kao što su korisničko ime, alias, email i zaporka.



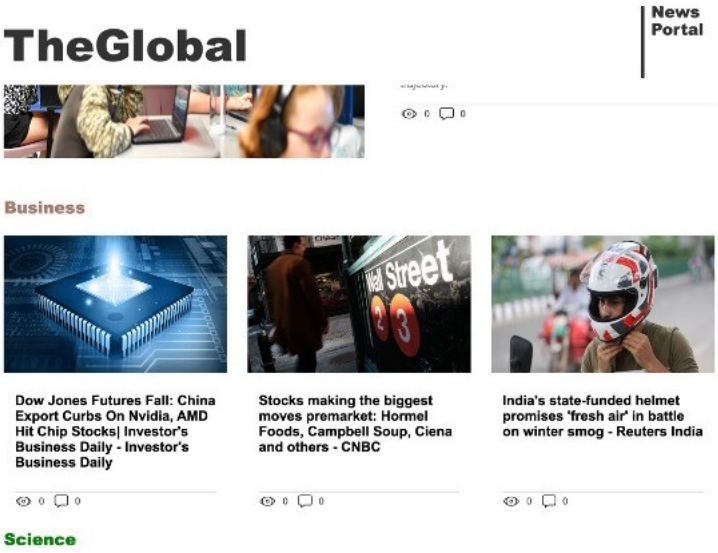
Slika 5.3. *Prikaz prijelomne vijesti na početnoj stranici kada korisnik nije prijavljen*



Slika 5.4. *Prikaz kategorija na početnoj stranici kada korisnik nije prijavljen*

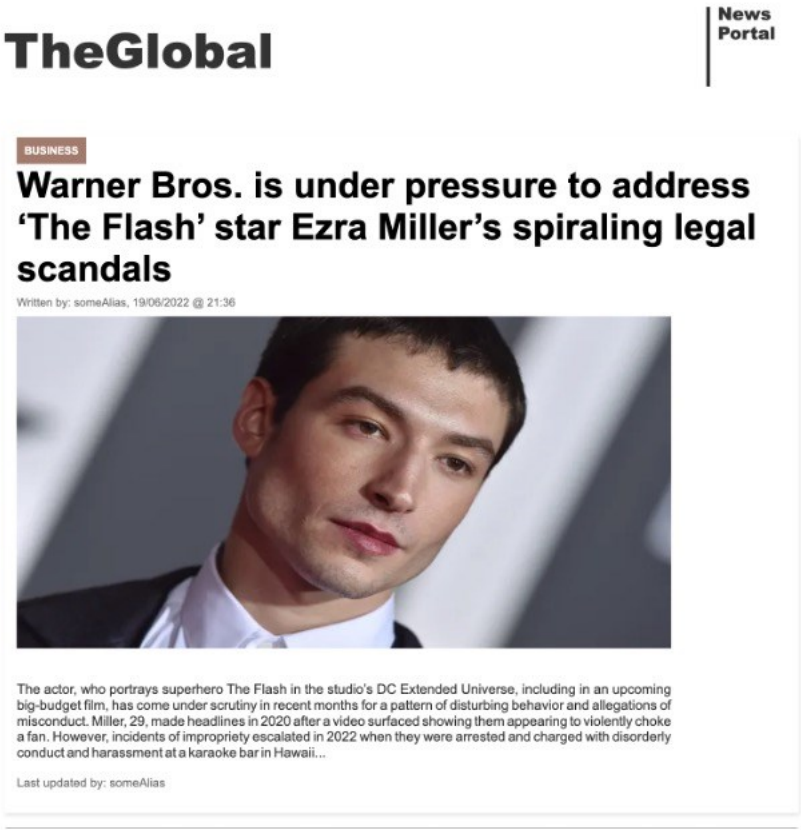


Slika 5.5. *Prikaz prijelomne vijesti na početnoj stranici kada je korisnik prijavljen*



Slika 5.6. *Prikaz kategorija na početnoj stranici kada je korisnik prijavljen*

Na slikama 5.3. i 5.5. prikazana je prijelomna vijest kada je korisnik prijavljen i kada nije. Za svaku prijelomnu vijest prikazano je proteklo vrijeme od objave, naslov, kratak opis, broj pogleda i komentara koji se prikazuju ovisno o korisnikovoj ulozi. Razlika u izgledu aplikacije je ta što neprijavljenim korisnicima nije prikazan podatak o broju pogleda i komentara prijelomne vijesti. Slikama 5.4. i 5.6. prikazane su kategorije na početnoj stranici. Važno je napomenuti kako za svaku kategoriju postoje po tri vijesti na početnoj stranici, no za potrebe objašnjavanja izgleda aplikacije dovoljna je jedna kategorija. Svaka vijest ima svoju sliku, naslov, broj pogleda i komentara koji se uvjetno prikazuju. Razlika kao i kod prijelomnih vijesti kod prijavljenih i neprijavljenih korisnika je u tome što neprijavljenim korisnicima nije prikazan podatak o broju pogleda i komentara članaka.



Slika 5.7. *Prikaz pojedinog članka*

Slika 5.7. prikazuje kako izgleda članak kada se na njega klikne. Prikaz se sastoji od klikabilne kategorije kojoj pripada te vodi na stranicu gdje su prikazane vijesti iz te kategorije. Svaka kategorija je prikazana određenom bojom kako bi se razlikovali. Također, prikaz sadrži naslov članka, kratke informacije tko je napisao članak kojeg dana i u koliko sati. Prikazana je i uvećana slika članka nakon koje slijedi glavna priča, a na kraju se nalazi informacija o tome tko je zadnji ažurirao članak. Kada se radi o korisniku koji je urednik na kraju je prikazan i izbornik koji pruža opciju ažuriranja članka čija je funkcionalnost u nastavku detaljnije opisana. Ako je korisnik „admin“ uz opciju za ažuriranje se prikazuje i opcija za brisanje.

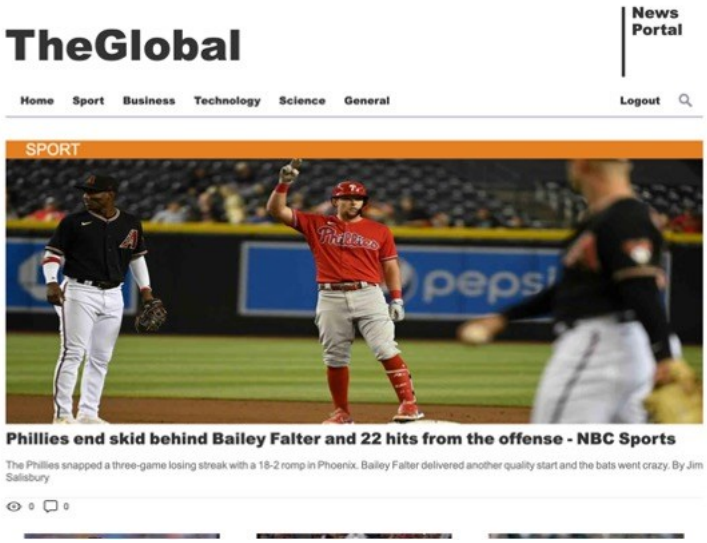
Slika 5.8. *Prikaz članka dohvaćenog s NewsAPI*

Slikom 5.8. prikazan je članak koji je dohvaćen s API-a kojeg omogućuje NewsAPI stranica. Izgled je vrlo sličan kao i kod ostalih vijesti, no bitna razlika je što ovdje postoji link na portal koji je autor ovoga članka. Cijela priča ne može biti objavljena u aplikaciji jer NewsAPI omogućuje tek 201 znak od cijele priče besplatno.

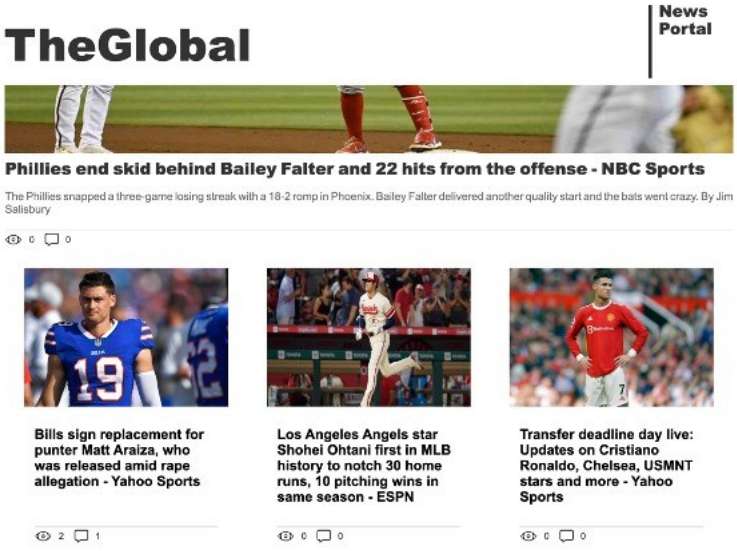
Slika 5.9. *Prikaz komentara članka*

Slikom 5.9. prikazan je komentar korisnika na članak, proteklo vrijeme od njegove objave te polje u koje se upisuje komentar. Nakon što korisnik upiše komentar unutar predviđenog polja, potrebno je pritisnuti tipku „*publish*“ kako bi komentar bio javno vidljiv. Na svakom članku vidljiv je broj

komentara. Također, klikom na dodatne opcije korisnik koji je vlasnik komentara može ažurirati i obrisati komentar.

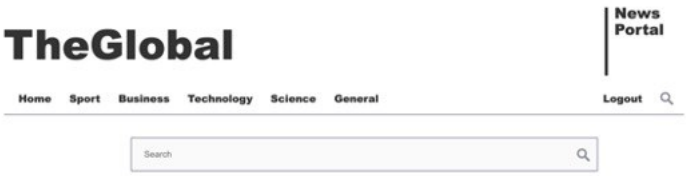


Slika 5.10. *Prikaz vijesti iz kategorije Sport*

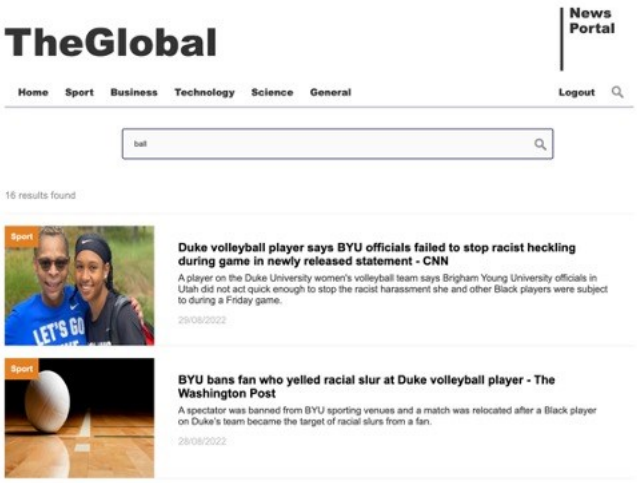


Slika 5.11. *Prikaz ostalih vijesti iz kategorije Sport*

Slikom 5.10. prikazuju se vijesti iz kategorije *Sport*. Najnovija vijest prikazana je uvećano, a sastoji se od slike, istaknutog naslova, kratkog opisa i prikaza brojeva pogleda i komentara. Najnovija vijest popraćena je ostalim vijestima koje su prikazane slikom 5.11. i nalaze se odmah ispod nje istim prikazom kao što je to bilo i na početnoj stranici. Dakle, prikazuje se slika, naslov, broj pogleda i komentara koji je vidljiv samo prijavljenim korisnicima. U poglavlju 4.3. je objašnjen postupak dohvaćanja i „mapiranja“ podataka. Za svaku ostalu kategoriju primjenjuje se isti prikaz.

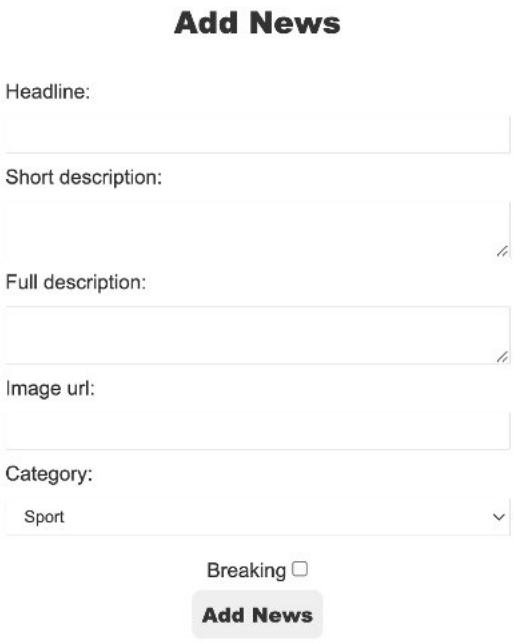


Slika 5.12. *Prikaz tražilice*

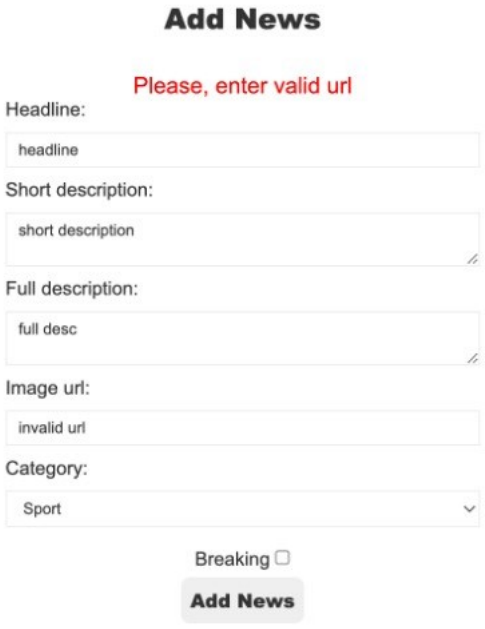


Slika 5.13. *Prikaz traženja članaka prema ključnoj riječi*

Slikom 5.12. prikazana je tražilica s kojom je moguće nakon unosa ključne riječi pretražiti članke po njihovom naslovu. Drugim riječima, ukoliko se unesena riječ nalazi unutar naslova članka tada će se taj članak prikazati kao rezultat pretrage. Slika 5.13. prikazuje rezultat pretrage prema ključnoj riječi *"ball"*. Prikazuje se broj pronađenih članaka te nakon toga slijedi lista članaka, ukoliko ih ima više od 20 tada je moguće koristiti paginaciju kako bi se pristupilo i starijim člancima koji odgovaraju unesenoj ključnoj riječi.



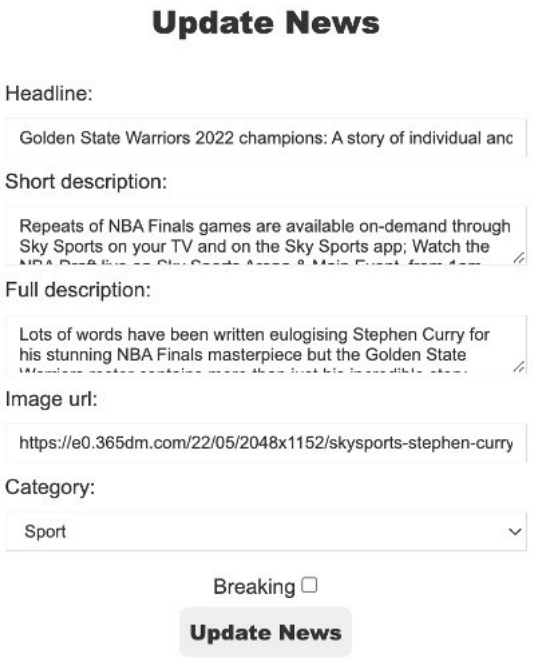
Slika 5.14. *Forma za dodavanje vijesti*



Slika 5.15. *Prikaz poruke o pogrešci*

Slikom 5.14. prikazana je forma za dodavanje vijesti. Vijesti mogu dodavati korisnici s ulogom „editor“ ili „admin“. Prilikom dodavanja nove vijesti, odnosno, novog članka je potrebno popuniti polje za naslov, kratki opis, dugi opis, link na sliku, odabrati kategoriju i označiti radi li se o prijelomnoj vijesti ili ne. Slikom 5.15. prikazana je poruka o pogrešci koja se ispisuje ovisno o

kojem se polju radi. U ovom slučaju je poslužiteljska strana prepoznala kako se radi o nizu znakova koji nisu valjani link na sliku pa je stoga poslana poruka o pogrešci kao odgovor na zahtjev za spremanje novog članka u bazu podataka.



Slika 5.16. *Forma za ažuriranje članka*

Slikom 5.16. prikazana je forma za ažuriranje članka. Forma ima ista polja kao i prilikom stvaranja, no ovaj put su polja popunjena s vrijednostima članka koji se želi ažurirati. Sve što je potrebno je izmijeniti tekst ili nešto drugo te pritisnuti gumb *Update News* čime se šalje zahtjev za ažuriranjem s novim podacima.

6. ZAKLJUČAK

Čitanje o svakodnevnim događajima iz svijeta putem interneta postala je ljudska potreba, a pojavom iste raste i potreba za razvojem web portala za vijesti koje to i omogućavaju. U ovom diplomskom radu je prikazan i objašnjen postupak izrade web portala za vijesti koji sadrži sve potrebne funkcionalnosti kako bi ispunio svoja očekivanja.

Aplikacija je rađena u dva odvojena dijela, a to su poslužitelj koji je ostvaren koristeći Node.js okruženje te klijentski dio aplikacije koji je ostvaren pomoću React biblioteke. Za ovu aplikaciju implementirane su funkcionalnosti kao što su prijava i registracija korisnika, dohvaćanje, kreiranje, ažuriranje i brisanje članaka, njihovo komentiranje i pretraga članaka prema njihovom naslovu. Također, aplikacija omogućuje prijavljenim korisnicima uvid u broj pregleda i komentara pojedinog članka što može poboljšati korisničko iskustvo. Implementiranje navedenih funkcionalnosti bio je i cilj stoga se može zaključiti kako je uspješno ispunjen zadatak diplomskog rada.

Ovaj projekt služi kao primjer kako uz neke od najpopularnijih tehnologija ostvariti potpuno funkcionalan web portal za vijesti. Velika prednost izrade ovakve aplikacije je mogućnost učenja i savladavanje tehnologija koje su uvelike zastupljene na tržištu te samim time i povećavanje konkurentnosti programera.

LITERATURA

[1] Dnevnik.hr, dostupno na: https://dnevnik.hr/ (pristupljeno 20.8.2022.)

[2] Index.hr, dostupno na: https://www.index.hr/ (pristupljeno 20.8.2022.)

[3] Wikipedia, BBC News, dostupno na: https://en.wikipedia.org/wiki/BBC_News#cite_note-90 (pristupljeno 20.8.2022.)

[4] BBC News, dostupno na: https://www.bbc.com/news (pristupljeno 20.8.2022.)

[5] The New York Times, dostupno na: https://www.nytimes.com/ (pristupljeno 20.8.2022.)

[6] Slobodna Dalmacija, dostupno na: https://slobodnadalmacija.hr/(pristupljeno 1.9.2022.)

[7] Tutorials point, Node.js - Introduction, dostupno na: https://www.tutorialspoint.com/nodejs/nodejs_quick_guide.htm (pristupljeno: 20.5.2022.)

[8] Tutorials point, Node.js - Express Framework, dosutpno na: https://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm (pristupljeno: 20.5.2022.)

[9] MongoDB, JSON and BSON, dostupno na: https://www.mongodb.com/json-and-bson (pristupljeno: 21.5.2022.)

[10] React, A JavaScript library for building user interfaces, dostupno na: https://reactjs.org/ (pristupljeno: 15.6.2022.)

[11] Guru99, React vs Angular: 10 Most important Differences You Must Know!, dostupno na: https://www.guru99.com/react-vs-angular-key-difference.html (pristupljeno 15.6.2022.)

[12] React, Introducing JSX, dostupno na: https://reactjs.org/docs/introducing-jsx.html (pristupljeno 17.6.2022.)

[13] GeeksforGeeks, ReactJS | Virtual DOM, dostupno na: https://www.geeksforgeeks.org/reactjs-virtual-dom/?ref=lbp (pristupljeno 21.6.2022.)

[14] javatpoint, What is CSS, dostupno na: https://www.javatpoint.com/what-is-css (pristupljeno: 21.6.2022.)

SAŽETAK

Temeljni zadatak diplomskog rada je izraditi web portal za vijesti koji omogućava prijavu i registraciju korisnika, dohvaćanje, kreiranje, ažuriranje i brisanje članaka, njihovo komentiranje i pretragu članaka prema njihovom naslovu. Ovaj rad prikazuje način izrade web portala za vijesti primjenom tehnologija kao što su Node.js, Express.js, MongoDB i React. Node.js, Express.js i MongoDB korišteni su kako bi se izradila poslužiteljska strana, dok je React korišten za izradu klijentske strane.

Ključne riječi: članak, Express.js, MongoDB, Node.js, portal, React,.

ABSTRACT

Web portal for news

Fundamental task of this paper was to create web portal for news which provides login and registration for users, reading, creating, updating and deleting articles, commenting on them and searching articles by their headline. This paper shows a way to create web portal for news using technologies like Node.js, Express.js, MongoDB and React. Node.js, Express.js and MongoDB are used to implement server side while React was used to implement client side.

Key words: article, Express.js, MongoDB, Node.js, portal, React,

ŽIVOTOPIS

Josip Čuković rođen je 3. srpnja 1998. godine u Đakovu, Hrvatska. Osnovnoškolsko te srednjoškolsko obrazovanje završava u Đakovu. Srednju školu „Srednja strukovna škola Antuna Horvata, Đakovo“ smjer „Računalni tehničar za strojarstvo“ završava 2017. godine. Nakon završetka srednjoškolskog obrazovanja i uspješnog polaganja državne mature, upisuje se na preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku kojeg uspješno završava 2020. godine. Nakon završetka preddiplomskog studija obrazovanje nastavlja na istom fakultetu upisujući diplomski studij računarstvo, modul programsko inženjerstvo. U tvrtki COBE d.o.o. je od veljače 2022. godine zaposlen kao student na poziciji *backend* developera.

_____

Potpis autora

PRILOZI

CD:

1. Diplomski rad „Web portal za vijesti.docx“

2. Diplomski rad „Web portal za vijesti.pdf“

3. Izvorni kod web portala