



VEGOVA

ELEKTROTEHNIŠKO-RAČUNALNIŠKA
STROKOVNA ŠOLA IN GIMNAZIJA
LJUBLJANA

SOCIALNO OMREŽJE

Strokovno poročilo za 4. predmet poklicne mature

Mentor: Matic Podpadec, dipl. inž.

Avtor: Tilen Anzeljc, R 4. B

Ljubljana, april 2021

POVZETEK

V poročilu je predstavljena spletna aplikacija, ki je namenjena komunikaciji med mladimi in njihovimi vrstniki, predvsem v času korona krize. Na začetku so opisana orodja, programski jeziki ter metode, ki sem jih med izdelavo uporabljal. Nato je razloženo osnovno delovanje aplikacije in njena komunikacija s strežnikom. Sledi predstavitev spletne strani po glavnih komponentah in njihove funkcije. Predstavitev posameznega dela obsega njegovo delovanje, zgradbo ter specifične postopke, ki so bili uporabljeni. Izpostavil sem tudi glavne težave, na katere sem naletel med samim razvijanjem in njihove rešitve. Na koncu je predstavljena baza podatkov in njena shema ter sistem za odkrivanje napak skozi celotno aplikacijo. V zaključku so podane še moje ugotovitve in reference, ki sem jih od samega načrtovanja do implementacije pridobil.

Ključne besede

Socialno omrežje, uporabnik, komunikacija, podatki, MERN sklad, JavaScript, »fullstack« spletni razvoj

ABSTRACT

In this report, a web application used for communication between young people and their peers, especially now in corona crisis, is presented. At the beginning, the tools, programming languages and methods used during development are described. After that I briefly explained how application works and its communication with server. Next there is presentation of webpage by main parts and its functions. Presentation of each part consist of its operation, structure and specific procedures that were used. I also exposed issues, I encountered during development and their solutions. At the end database, its schema and system for error detection are described. In conclusion my findings and references which I gained from planning to implementation are given.

Keywords

Social network, user, communication, data, MERN stack, JavaScript, fullstack web development

KAZALO VSEBINE

1. UVOD	6
2. ORODJA IN PROGRAMI.....	6
2.1 React.....	6
2.2 Node.js	7
2.3 Express	7
2.4 MongoDB	8
2.5 CSS.....	8
2.6 JavaScript.....	8
3.RAZVIJANJE	9
4.SPLETNA APLIKACIJA.....	9
4.1 Komponente	10
4.2 Komunikacija komponenta – strežnik	11
4.3 Sistem za kreiranje, avtentikacijo in avtorizacijo uporabnika	12
4.3.1 Registracija	12
4.3.2 Prijava	12
4.3.3 Resetiranje gesla	13
4.3.4 Avtentikacija	13
4.4 Stran domov	14
4.4.1 Glava.....	14
4.4.2 Iskanje uporabnikov in pregled profila.....	14
4.4.3 Obvestila	15
4.4.4 Odjava.....	16
4.4.5 Objave.....	16
4.4.6 Posamezna objava.....	18
4.4.7 Komentari	19
4.4.8 Profil	21
4.4.9 Dogodki.....	22
4.5 Filtriranje.....	22
4.6 Klepet	24
4.6.1 Kontakti in skupine	24
4.6.2 Pogovori	25

5. ODKRIVANJE NAPAK.....	26
6. BAZA	27
7. ZAKLJUČEK.....	29
8. VIRI IN LITERATURA.....	31
8.1 SPLETNE STRANI.....	31
6.2 SLIKE	33

KAZALO SLIK

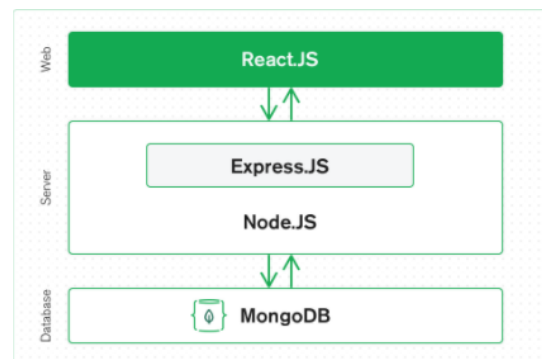
Slika 1: Shematski prikaz MERN sklada.....	6
Slika 2: Uporabniški vmesnik platforme Postman	7
Slika 3: Prototip komponente za nalaganje datotek.....	9
Slika 4: Drevesni prikaz DOM elementov	10
Slika 5: Hook-i komponente Feed.....	11
Slika 6: Primer storitve za dodajanje všečka	11
Slika 7: Odsek kode, ki zgenerira JWT in piškotek	12
Slika 8: Polje za resetiranje gesla	13
Slika 9: Funkcija za preverjanje resničnosti uporabnika	13
Slika 10: Stran domov.....	14
Slika 11: Profil uporabnika	15
Slika 12: Odsek kode, kjer Pusher spremlja spremembe v bazi	16
Slika 13: Nalaganje datotek.....	17
Slika 14: Objave	19
Slika 15: Seznam komentarjev	20
Slika 16: Urejevalnik profila	21
Slika 17: Stran za filtriranje	23
Slika 18: Javascript funkcija za filtriranje	23
Slika 19: Stran za pogovarjanje	24
Slika 20: Funkcija, ki poskrbi za prikaz napak	26
Slika 21: Koda za kreiranje sheme sporočila	27
Slika 22: Skica ER modela	28
Slika 23: Primer poizvedbe v bazo	29

1. UVOD

Za svoj maturitetni izdelek sem si izbral izdelavo enostranske (angl. singlepage) aplikacije - socialnega omrežja. Aplikacija vsem registriranim uporabnikom omogoča, da z uporabo preprostega uporabniškega vmesnika med seboj komunicirajo in preko objav delijo njihove informacije. Spletna stran ima na voljo ogromno funkcionalnosti, ki uporabniku poleg interakcije z drugimi vrstniki, omogočajo hitro iskanje željenih podatkov in organiziranost. Za izdelek sem se odločil, ker sem se želel bolj podrobno seznaniti s popolnoma drugačnim načinom programiranja, kot z jezikoma C++ in Php, ki sem ga spoznal v šoli,. Izbral sem zelo raznolik tip spletne aplikacije, ki mi je omogočil, da sem uporabil kar največ možnosti, ki so mi jih orodja ponujala. Ker so nekatera relativno nova, sem veliko truda in prostega časa vložil v samostojno učenje, pri čemer sem si pomagal z osnovami, ki sem jih pridobil v času 4-letnega šolanja.

2. ORODJA IN PROGRAMI

Za postavitev strežnika in aplikacije sem izbral MERN spletni sklad(ang. stack), ki je sestavljen iz podatkovne baze MongoDB, spletnih ogrodij Express.js in Node.js ter knjižnice React.js.



Slika 1: Shematski prikaz MERN sklada

2.1 React

React (imenovan tudi React.js) je odprtokodna »front-end« JavaScript knjižnica, ki se uporablja za izdelavo interaktivnih grafičnih uporabniških vmesnikov. Logika knjižnice temelji na komponentah, ki poskrbijo za optimizacijo kompleksno zgrajenih aplikacij. Pri programiranju se pogosto uporablja sintaksna razširitev JSX, ki spominja na HTML kodo, s sposobnostmi JavaScript-a.

React sem izbral, ker ponuja velik nabor knjižnic, med katerimi lahko uporabnik svobodno izbira. Koda v primerjavi s PHP omogoča lažje in bolj učinkovito povezovanje med dinamičnimi in statičnimi podatki spletne strani ter je veliko bolj pregledna.

2.2 Node.js

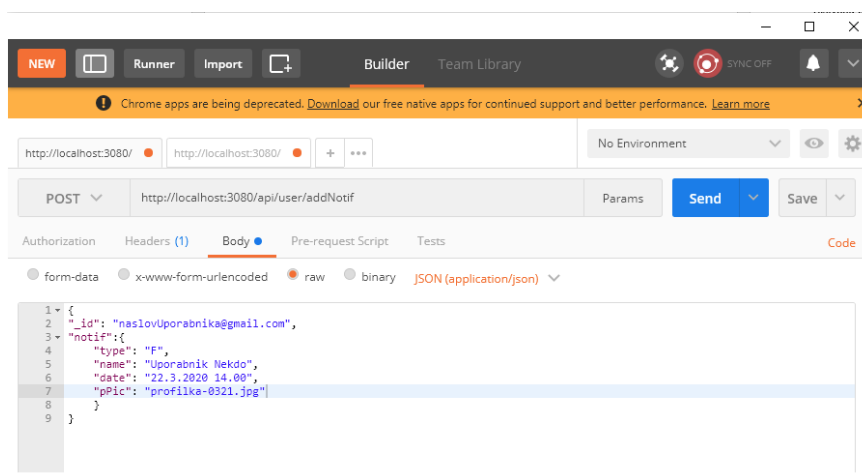
Node.js je odprtokodno, več platformno, JavaScript okolje za razvoj različnih orodij in aplikacij. Za delovanje uporablja Googlov V8 JavaScript pogon, ki omogoča hitro izvajanje kode in uporabo enega programskega jezika, tako na strani strežnika, kot odjemalca. Node.js operacije so eno nitne (angl. single-threaded), zato komunikacija temelji na asinhronih vhodno-izhodnih (angl. input-output) operacijah, ki so med seboj neodvisne.

Za Node.js sem se odločil, ker njegov napreden način delovanja omogoča razširljivost in hitro komunikacijo med ozadjem ter uporabniškim vmesnikom. Ker sem lahko uporabljal JavaScript tudi za strežnik, nisem potreboval preklapljati med jeziki.

2.3 Express

Express (tudi Express.js) je minimalistično in prilagodljivo ogrodje, ki se izvaja znotraj strežnika Node.js. Njegova glavna naloga je usmerjanje enoličnih krajevnikov vira (angl. URL-jev) in obdelava http (Hypertext Transfer Protocol) zahtev ter odgovorov med aplikacijo in njenim strežnikom.

Med povezovanjem sprednjega dela aplikacije (angl. frontend) in ozadja (angl. backend) mi je Express.js omogočil veliko lažje in bolj pregledno delo s HTTP zahtevami, kakor že vgrajeni, bolj okrnjeni Node.js pripomočki. V veliko pomoč mi je bila platforma Postman, s pomočjo katere sem hitreje odkril napake in testiral komunikacijo s strežnikom.



Slika 2: Uporabniški vmesnik platforme Postman

2.4 MongoDB

MongoDB je odprtokodna, ne relacijska oz. »no SQL« baza podatkov. V primerjavi z drugimi SUPB (sistem za upravljanje s podatkovnimi zbirkami podatkov) sistemi, kot sta MySql in Firebird, za shranjevanje ne uporablja tabel oz. vrstic, ampak JSON (JavaScript Object Notation) dinamične podatkovne tipe oz. BSON (binarno kodirani JSON) dokumente.

Takšen tip baze sem izbral, ker se JSON datoteke zelo dobro povezujejo z JavaScript-om in ostalimi tehnologijami v MERN skladu.

2.5 CSS

CSS (angl. Cascading Style Sheets) ali kaskadne slogovne podloge so podloge, ki omogočajo razvijalcem spletnih strani, da predpišejo obliko posameznim elementom. Bistvena prednost CSS-a je, da loči predstavitevni oz. oblikovni in logični del aplikacije, kar pripomore k večji preglednosti kode.

Poleg CSS sem uporabil React-ovo knjižnico za oblikovanje, Material-UI, kjer so nekatere že napisane komponente, z veliko uporabnimi funkcijami.

2.6 JavaScript

JavaScript (krajše JS) je skriptni programski jezik, ki temelji na standardu ES (ECMA script). Sprva se je jezik uporabljal samo v spletnih brskalnikih, danes pa je osrednji sestavni del tudi v sistemih na strani strežnika, kot je Node.js. Deluje lahko kot proceduralno ali objektno usmerjeni jezik, njegova sintaksa pa je zelo podobna C++ in Javi.

Pri programiranju sem uporabil standard ES6, razvit leta 2015, ki mi je s svojimi naprednimi funkcionalnostmi, kot so npr. puščične (angl. arrow) funkcije, precej olajšal delo z dinamičnimi podatki.

3.RAZVIJANJE

Preden sem pričel s pisanjem kode, sem se temeljito lotil načrtovanja. Na grobo sem si skiciral aplikacijo in razmislil, katere podatke in na kakšen način bo uporabnik dostopal do njih. Ko sem vedel, kaj potrebujem, sem se dokončno odločil za orodja in programe, ki jih bom uporabljal. Zaradi lažje predstave, sem si okvirno sestavil shemo podatkovne baze, po potrebi pa sem jo sproti spreminjal. Naredil sem tudi preprost prototip aplikacije. Delo sem si razdelil na manjše dele, urejene po težavnosti izvedbe. Ko sem okvirno zaključil z uporabniškim vmesnikom in oblikovanjem, sem se lotil programiranja strežnika in povezovanja med posameznimi odseki. Na koncu sem se posvetil detajlom, optimizaciji in odpravljanju napak, ki so se pojavile med testiranjem.



Slika 3: Prototip komponente za nalaganje datotek

4.SPLETNA APLIKACIJA

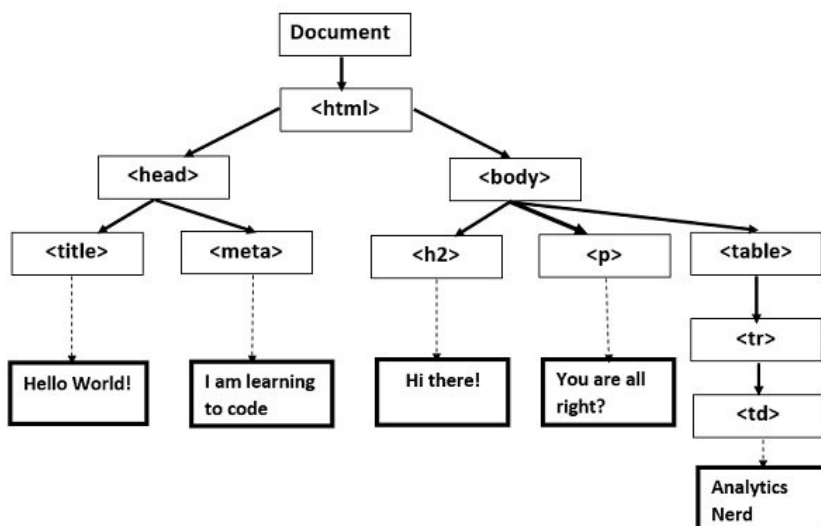
Spletna aplikacija je sestavljena iz 4 glavnih delov, ki so združeni v datoteki App.js:

- sistema za kreiranje, avtentikacijo in avtorizacijo uporabnika,
- strani domov,
- strani za filtriranje,
- klepetalnica

Vsak del je sestavljen iz več manjših komponent iz mape Components, ki so ustrezno oblikovane s pomočjo .css datotek in MaterialUI komponent.

4.1 Komponente

Komponenta je funkcija ali razred, ki sprejme vhodne parametre (angl. Props) in vrne React element. To so običajni objekti, ki so zelo podobni klasičnim elementom standarda DOM (Document Object Model). Pred React-om so razvijalci morali neposredno manipulirati s temi moduli, kar je pomenilo, da je moral brskalnik ob vsaki posodobitvi podatkov ponovno narisati celotno spletno stran.



Slika 4: Drevesni prikaz DOM elementov

Za osveževanje React-ovih elementov pa skrbi React DOM (tudi virtualni DOM), ki zagotavlja, da se posodobi le specifičen del, kjer je prišlo do spremembe, kar pripomore k precejšnji optimizaciji.

Spremembe so definirane s pomočjo stanj (angl. State) oz. spremenljivk in življenjskih ciklov (angl. LifeCycle) komponente. Stanja so spremenljivke, ki s spremembo svoje vrednosti povzročijo posodobitev celotne komponente. Ker so podatki shranjeni v stanjih, jih ob posodabljanju ni potrebno ponovno pridobiti iz baze. Življenjski cikli komponente pa so dogodki, ki se zgodijo tekom prikazovanja komponente. Ti dogodki se npr. prožijo ob prvem risanju komponente, po koncu vsakega posodabljanja, itd. Večinoma sem uporabljal funkcijske komponente, kjer so ti cikli poimenovani Hooks. Obe zgoraj omenjeni zadevi sta zelo pomembni, saj poskrbita, da se podatki iz baze shranijo v komponento in se nemoteno prikazujejo uporabniku.

```
function Feed() {
  const currUser = useContext(UserContext);
  const [posts, setPosts] = useState([]);
  const [number, setNumber] = useState(0);
  const [count, setCount] = useState({ from: 0, for: 0 });
  const [hasMore, setHasMore] = useState(true);
  useEffect(async () => { ...
  }, []);
}
```

Slika 5: Hook-i komponente Feed

4.2 Komunikacija komponenta – strežnik

Aplikacija deluje tako, da pošlje strežniku zahtevo, ki se sproži ob različnih dogodkih oz. uporabnikovi interakciji z vmesnikom. HTTP zahteva (angl. request) je poslana preko storitev oz. funkcij, ki se nahajajo v ločeni mapi Services. Tu se s pomočjo knjižnice Axios določi pot(ang. path) in tip ter podatki, ki so posredovani strežniku.

```
const apiEndpointPost = "http://localhost:3080/api/post"

export function addLike(_id, userId) {
  return http.post(`${apiEndpointPost}/addLike`, {
    _id: _id,
    userId: userId
  })
}
```

Slika 6: Primer storitve za dodajanje všečka

Ko strežnik sprejme zahtevo uporabnika se najprej kličejo vmesne funkcije(angl. middleware). To so operacije, ki se izvedejo tik pred obdelavo zahteve, npr. pretvorba poslanih podatkov v drug format, avtentikacija, različne varnostne nastavitve, itd. Nato strežnik na podlagi poti, ki je bila določena izvede operacijo. V primeru, da je ta uspešno izvedena, se v aplikacijo pošlje odgovor, ki nosi informacije o uspešno ali neuspešno izvedeni transakciji. Za posredovanje odgovora skrbi parameter res (angl. response), za pridobitev zahteve pa req (angl. request). Zahteve so asinhronne in so lahko namenjene branju, zapisovanju, brisanju in kreiranju.

4.3 Sistem za kreiranje, avtentikacijo in avtorizacijo uporabnika

Pri realizaciji sistema sem uporabil JWT (Json Web Token), odprtokodni standard (RFC 7519), ki omogoča kompaktno in varno posredovanje informacij med objekti tipa JSON. Podatke je mogoče preveriti in jim zaupati, saj so digitalno podpisani, z uporabo določenih algoritmov.

4.3.1 Registracija

Uporabnik mora za registracijo vpisati svoje ime, priimek, geslo, datum rojstva, status in izobraževalno ustanovo. Če se vsi podatki ujemajo z omejitvami v formi, npr. (oblika e-poštnega naslova), se pošljejo na strežnik, kjer se izvede preverjanje, ali uporabnik že obstaja. Če je preverjanje uspešno se zgenerira povezava, ki se s pomočjo React-ove knjižnice Nodemailer pošlje na uporabnikov E-poštni naslov. Ta vsebuje zgoraj omenjeni JWT s podatki, ki ima življenjsko dobo 10 minut. Ob aktivaciji je uporabnik shranjen v bazo in preusmerjen na prijavo.

4.3.2 Prijava

Poleg E-poštnega naslova je potrebno tudi preverjanje gesla. V primeru uspešne prijave se zgenerira JWT, ki se shrani v piškotek (ang. Cookie), preprosto tekstovno datoteko, ki vsebuje podatke. Ta se naprej uporablja za preverjanje resničnosti uporabnika. Z nastavljanjem dodatnih parametrov (httpOnly, maxAge, path, secure), sem prej precej ranljiv piškotek spremenil v veliko bolj varnega. Uporabniku, ki se je prijavil je potrebno nastaviti podatek isActive na veljavno (true), s pomočjo katerega sistem ve, kateri uporabniki so trenutno aktivni.

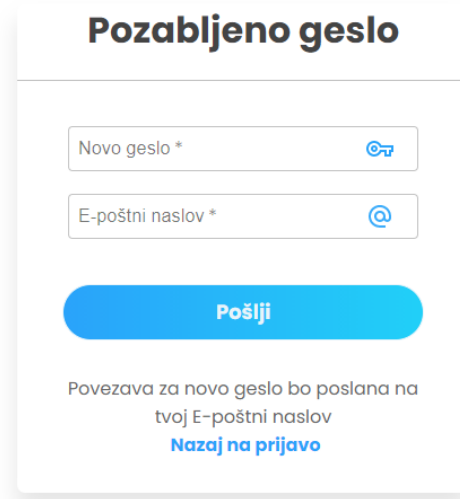
```
//zgenerira se JWT, v tem primeru ima neskončno življenjsko dobo
const token = jwt.sign(
  { _id: user._id, name: user.name, sname: user.sname, pImg: user.pImg }
  , process.env.TOKEN_SECRET)

//v front-end se pošlje uspešen odgovor s podatki za trenutnega uporabnika
//zgenerira se piškotek
res.status(200).cookie('userToken', token, {
  maxAge: 1000 * 60 * 10,
  httpOnly: true,
  secure: true, //velja le za produkcijsko okolje
  path: '/'
}).json({ token: token, data: { id: user._id, name: user.name, sname: user.sname, pImg: user.pImg } })
```

Slika 7: Odsek kode, ki zgenerira JWT in piškotek

4.3.3 Resetiranje gesla

Za resetiranje pozabljenega gesla uporabnik poda novo geslo in E-poštni naslov, na katerega mu je posredovana povezava za resetiranje, z veljavnostjo 10 minut. Po uspešni validaciji podatkov, se novo geslo shrani v bazo. Še prej pa ga je potrebno zakodirati, kar sem realiziral s pomočjo knjižnice bcrypt.js, ki uporablja Bcrypt (Blowfish block cipher cryptomatic) algoritem.



Slika 8: Polje za resetiranje gesla

4.3.4 Avtentikacija

Želel sem, da bi bilo preverjanje resničnosti uporabnika čim bolj učinkovito. To sem izvedel na 2 načina:

- preverjanje resničnosti ob vsaki zahtevi v strežnik (s pomočjo middleware)
- vsako časovno periodo (v tem primeru 10 minut)
-

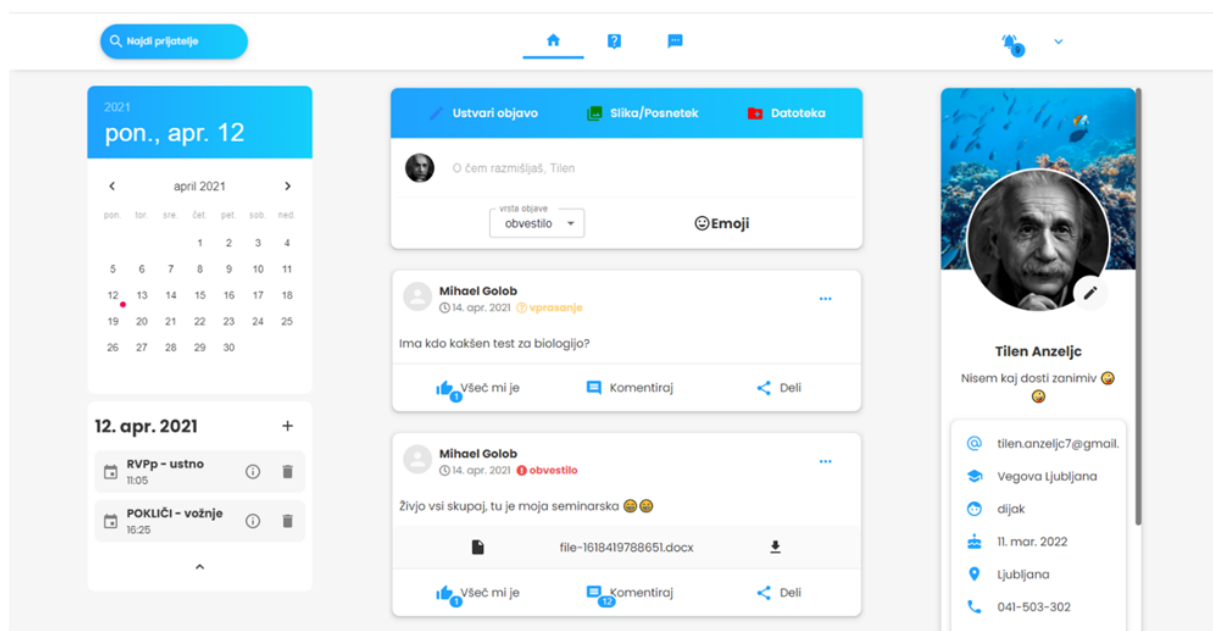
Preverjanje poteka v dveh korakih. Najprej se preverja veljavnost piškotka, nato pa sledi primerjanje s podatki trenutnega uporabnika. Ti so shranjeni v stanju (spremenljivki) glavne datoteke aplikacije App.js. V primeru neujemanja ali kakršnekoli druge napake, je uporabnik preusmerjen na začetno stran, isActive tega uporabnika pa se nastavi nazaj na neveljavno (false). Podatki o trenutnem uporabniku morajo biti na voljo vsem komponentam aplikacije, kar omogočata »hook« useContext in vhodni parametri komponent.

Pri tem sem naletel na težavo, saj so se podatki o trenutnem uporabniku ob vsakem osveževanju strani izbrisali, piškotek pa je ostal. To sem rešil s pomočjo življenjskega cikla useEffect, ki se izvede zgolj ob prvem upodabljanju React-ovega elementa.

```
//definiramo kot spremenljivko, saj je potreben
//izbris na login,... straneh
int = setInterval(async function () {
  try {
    await refreshUser(); // zahteva na strežnik
  } catch (err) {
    // če uporabnik ni avtenticiran
    if (err.response && err.response.status === 401) {
      await logoutUser();
      setCurrUser(null);
      clearInterval(int);
    }
  }
}, 3000); //10 minut v sekundah
```

Slika 9: Funkcija za preverjanje resničnosti uporabnika

4.4 Stran domov



Slika 10: Stran domov

4.4.1 Glava

Naslovna vrstica omogoča uporabniku hitro premikanje po aplikaciji in iskanje uporabnikov, pregled nad obvestili in gumb za odjavo. Premikanje po različnih delih je realizirano s komponento Router, ki uporabnika preusmeri na željeno destinacijo.

4.4.2 Iskanje uporabnikov in pregled profila

Podatki o uporabnikih se pridobivajo sproti, ob spremembi vrednost v vnosnem polju za iskanje. Teh pa je lahko ogromno, zato bi bil prenos vseh naenkrat zelo ne optimiziran. Problem sem rešil s pomočjo knjižnice React-infinite-scroll-component, ki iz baze, glede na našo pozicijo na seznamu pridobiva bloke podatkov, ki jih dodaja v tabelo že prej pridobljenih elementov. Ko se dolžina tabele ujema z dolžino vseh zapisov v bazi, komponenta ve, da so bili prebrani vsi zapisi in preneha s pošiljanjem zahtev na strežnik.

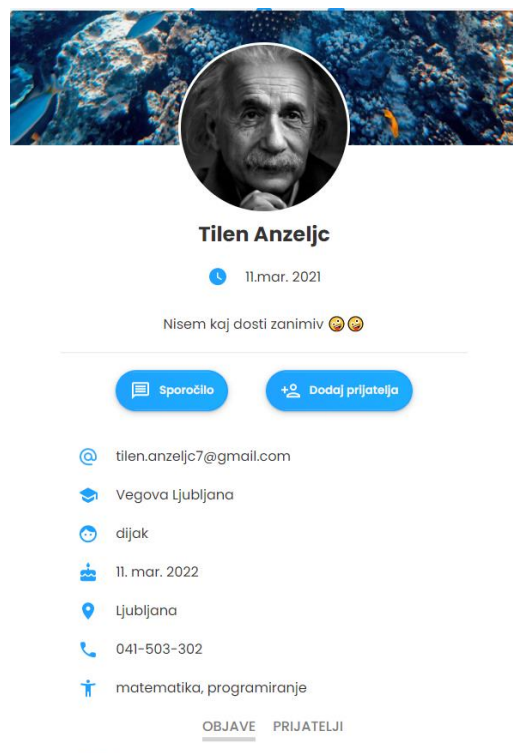
Vsaka komponenta seznama je sestavljena iz imena in profilne slike ter funkcije, ki ob vstopu kazalca miške prikaže še dodatne podatke, za hitrejšo predstavbo o osebi. Ob

pritisku nanjo se pokliče funkcija, ki dobi podatke določene osebe in nato odpre profil, ki prikazuje sliko ozadja in uporabnika ter njegove informacije.

Uporabnik lahko ob ogledu profila osebi pošlje prošnjo za prijateljstvo ali pa jo kontaktira preko direktnega sporočila. Gumba za dodajanje prijatelja spremeni svojo obliko, glede na to ali lastnik profila sodi med prijatelje osebe, ki si profil ogleduje. V primeru dodajanja prijatelja se šifra uporabnika posreduje lastniku, kjer v vrsti obvestil čaka na potrditev. Če uporabnik sprejme prošnjo, se ključ pošiljatelja shrani med njegove prijatelje, uporabnikova šifra pa se shrani med prijatelje pošiljatelja.

Ob pritisku gumba za sporočilo, pa nas sistem preusmeri na klepetalnik, kjer se odpre že obstoječ pogovor oz. zgenerira nov. Da klepetalnik ve, kateri pogovor mora odpreti, se šifra uporabnika shrani v začasno shrambo brskalnika Local Storage. Ko sistem konča s preverjanjem, podatki niso več potrebni, zato se izbrišejo.

Na dnu sledijo še seznam posameznikovih objav in prijateljev. Nad objavami so uporabniku omogočene vse spodaj opisane funkcije, na seznamu prijateljev pa lahko išče le te in si ogleda njihov profil.



Slika 11: Profil uporabnika

4.4.3 Obvestila

Da bi uporabnik imel kar najboljšo izkušnjo z uporabniškim vmesnikom, sem uporabil okolje Pusher, ki omogoča vzpostavitev realno-časovne komunikacije med bazo in aplikacijo. Sistem na podlagi sprememb v bazi zgenerira dogodke in jih preko strežnika pošilja v uporabnikov del aplikacije.

Seznam iz obvestil je sestavljen iz manjših komponent Notification, ki nosijo podatek o imenu in profilni sliki sporočevalca ter tipu notifikacije, na podlagi katerega se zgenerira ustrezno besedilo, ikona ob profilni sliki in dodatne funkcije, če so te

potrebne. Obvestila se delijo na dva tipa. Prvi prikaže podatke in ob pritisku uporabnika izvede določeno akcijo, npr. prikaz vsečkane objave. Drugi pa ponudi uporabniku izbiro in na podlagi le te, se izvede neka operacija, npr. potrditev prošnje za prijateljstvo.

Obvestilo se uporabniku pošlje ob določeni akciji, kot je npr. komentiranje ali deljenje objave. Potrebno je navesti enolično šifro prejemnika, da sistem ve katere in čigave podatke je potrebno posodobiti. Ob spremembi zapisov uporabnikov v bazi, Pusher posreduje novo dodano obvestilo v komponento z vsemi obvestili, kjer pride do posodobitve podatkov. Neprebrane notifikacije so shranjene v ločeni tabeli, na podlagi katere se določi koliko in katere je potrebno prikazati kot nove.

```
changeStreamUser.on('change', async (change) => {
  if (change) {
    if (change.updateDescription && change.operationType) {
      if (change.updateDescription.updatedFields["notifs"].length > 0 && change.operationType === 'update') {
        if (change.updateDescription.updatedFields["notifs"][0].type.charCodeAt(0) <= 97) {
          pusher.trigger(
            channelNotifs,
            'addedNotif',
            {
              id: change.documentKey._id,
              notifs: change.updateDescription.updatedFields["notifs"]
            }
          );
        }
      }
    }
  }
});
```

Slika 12: Odsek kode, kjer Pusher spremlja spremembe v bazi

4.4.4 Odjava

Odjava je zelo preprosta, saj se le izbriše piškotek in podatki o trenutnem uporabniku. Poleg tega se onemogoči funkcija, ki se kliče na 10 minut, saj se mora ta izvajati le, če je uporabnik prijavljen. Potrebna je tudi nastavitev parametra `isActive` na `neveljavno(false)`.

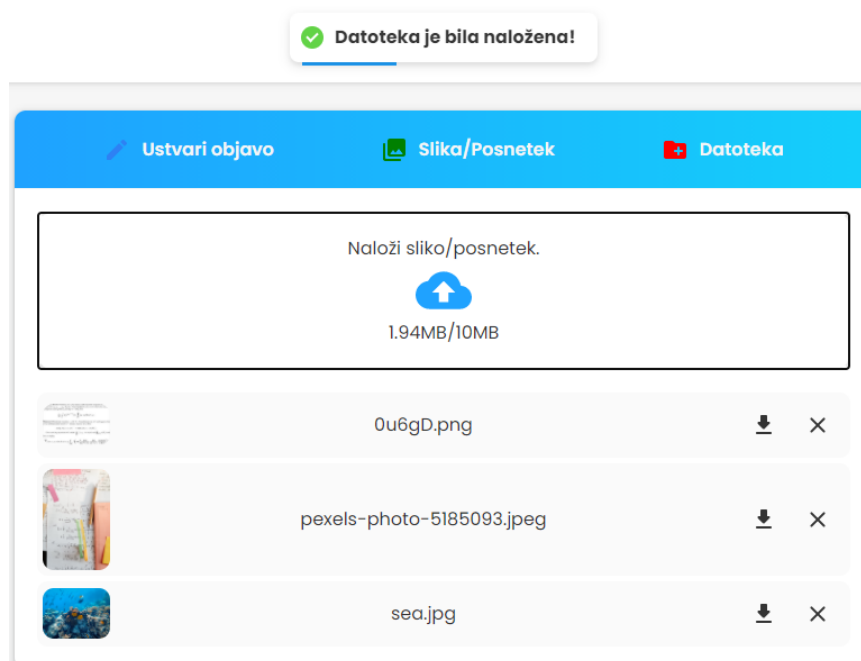
4.4.5 Objave

Največji in najbolj pomemben je prostor z objavami (ang. news-feed), ki je sestavljen iz 2 glavnih delov:

- Ustvarjanje objave,

- Seznam objav

Uporabnik pri kreiranju objave poda njeno vsebino in tip, opcijsko pa lahko naloži slike, posnetke ter ostale tipe datotek. Za nalaganje datotek skrbi knjižnica React-dropzone, ki omogoča, da datoteko preprosto z miško povlečemo v označeno polje. Med nalaganjem se preverja, ali skupna velikost naloženih datotek presega 10 MB. Uporabniku se posreduje sporočilo, ki je povezano z asinhrono operacijo, kar pomeni, da je vidno njeno celotno stanje in ne samo končni rezultat. Ko se vsebina uspešno naloži, se pošlje na strežnik, kjer se pretvori iz tipa BLOB v zahtevano binarno obliko in shrani v datotečni sistem GridFS, ki ga ponuja MongoDB. Trenutno naložene datoteke, so shranjene v komponenti za delo z datotekami in so prikazane pod poljem za nalaganje. Vsaka datoteka na seznamu ima opcijo, da se jo izbriše ali prenese. Pri prenosu sem dobljeni BSON format pretvoril v URL (Uniform Resource Locator) objekt, ki sem ga kot povezavo poklical ob pritisku na gumb.



Slika 13: Nalaganje datotek

Ko se uporabniku naloži stran se najprej s pomočjo react-infinite-scroll komponente iz baze pridobijo najnovejše objave, ki se shranijo v tabelo trenutnih objav. Tukaj se pokaže šolski primer, zakaj se uporabljajo stanja komponent v Reactu. Tabela objav predstavlja stanje komponente Feed, ki nosi ogromno število podatkov. Ob vsaki spremembi v bazi jo je potrebno spremeniti, kar pripomore k zmanjšanju nepotrebnih transakcij. Zelo enostaven je tudi prikaz elementov, če se nahajajo v tabeli, saj se z

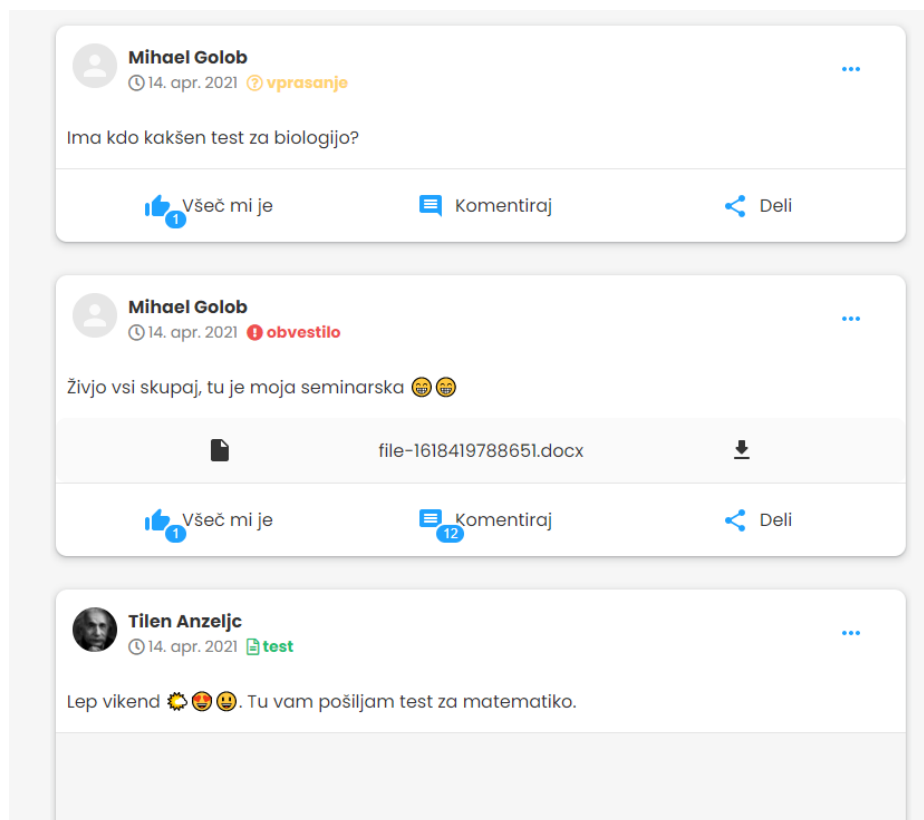
uporabo `map()` funkcije React sprehodi čez elemente tabele in za vsakega vrne svojo objavo, komponento `Post`.

4.4.6 Posamezna objava

Komponenti `Post` sem se posvetil najdlje, zato jo bom tudi bolj podrobno opisal. Sestavljena je iz glave, kjer se nahaja: slika profila, ime in priimek lastnika, tip ter datum objave. Na desni strani je meni, ki ponuja operacije, kot je npr. brisanje. Ena izmed funkcij je tudi prenos vseh datotek, ki vse podatke združi v skupno datoteko tipa `ZIP`. Na podlagi imen vseh datotek, ki so podani za vsako objavo se iz baze pridobijo datoteke, ki se pretvorijo v binarni zapis. Nato so s pomočjo logike `jszip` arhivirajo in prenesejo. Težava je, ker sistem zanesljivo deluje le do 50 MB podatkov, zato je bilo potrebno omejiti velikost naloženih datotek.

Drugi sestavni del objave je vsebina, pri kateri se je pojavila težava pri prikazovanju besedila. Če je uporabnik v vnosno polje pri kreiranju objave vpisal več vrstično besedilo, brez presledkov, se je to v objavi prikazalo v eni vrsti. Problem sem rešil s pomočjo `CSS` in `JavaScript` kode.

Vsebina pa lahko vsebuje tudi slike, posnetke ali datoteke. Seznam datotek je zelo podoben tistemu pri ustvarjanju objave. Prikaz slik in posnetkov pa je bolj kompleksen, saj je potrebno preverjati količino vseh datotek. Če gre zgolj za eno datoteko, naprej sledi pregled tipa, torej če je datoteka tipa `image` uporabimo osnovno `html` značko `img`, posnetek pa definiramo z oznako `video`. Ko je dolžina vseh večja od ena, pa so v prikaz vključeni še gumbi za premikanje po seznamu, število vseh in trenutna pozicija ter gumb za prenos vsake posamezne datoteke. Da brskalnik ve, katere podatke mora vključiti pri upodabljanju, je v parametru značk `img` in `video` navedena povezava za klic v strežnik, kjer se pridobi željena datoteka za prikaz.



Slika 14: Objave

Tretji del pa je vrstica za všečkanje, komentiranje in deljenje. Ko posameznik všečka objavo se izvede preverjanje ali se njegova šifra nahaja v polju likes, kjer so vse šifre uporabnikov, ki jim je objava všeč. Če se izkaže, da je uporabnik že vpisan na seznamu se šifra odstrani in število lajkov se zmanjša. V drugem primeru, pa se doda na listo, število všečkov se poveča in lastniku objave je posredovano obvestilo.

Deljenje deluje tako, da se kopirajo vsi podatki trenutne objave, razen komentarji in všečki. Lastnik objave se vpiše v polje `sharedOwner`, ki določa prvotnega lastnika, trenutni uporabnik pa se shrani pod normalnega lastnika. Če polje `sharedOwner` vsebuje podatke, sistem ve, da gre za deljeno objavo in v vrstici z imenom izpiše ustrezno besedilo ter imeni obeh lastnikov.

4.4.7 Komentarji

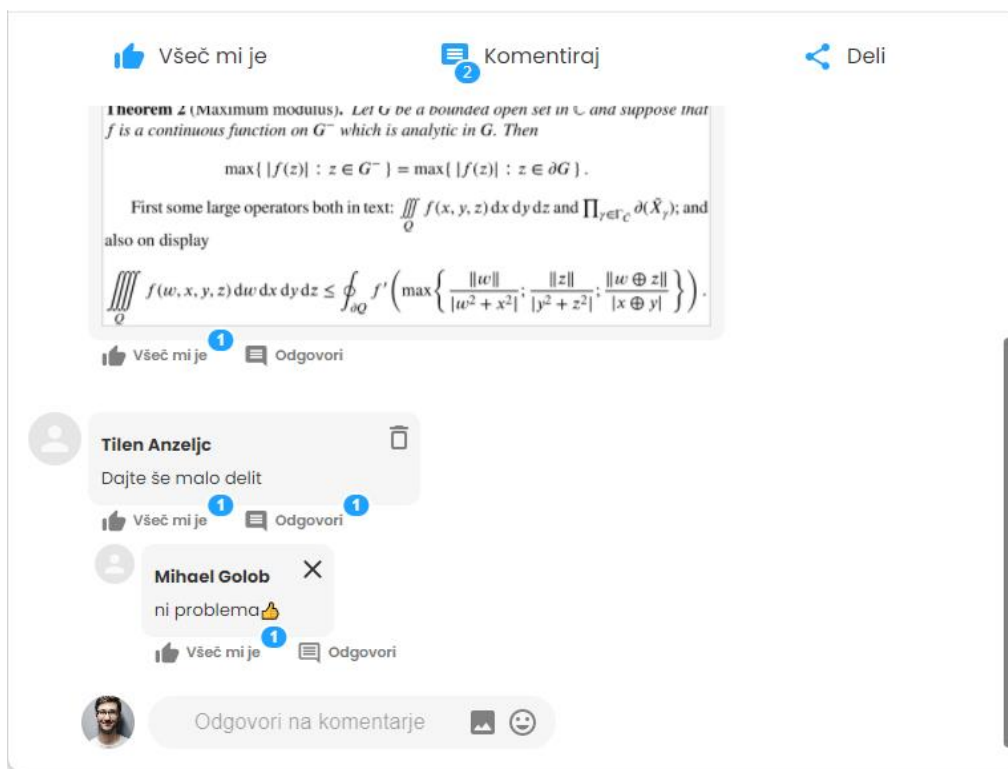
Nalaganje komentarjev se izvede ob pritisku na gumb komentiraj. Na vrhu polja s komentarji se nahaja vrstica za kreiranje, ki je sestavljena iz že prej uporabljenih komponent: vnosnega polja za besedilo, gumbov za nalaganje slik ter dodajane emodžijev. Komentarji delujejo zelo podobno kot objave, le da so malo bolj okrnjeni.

Nosijo podatke o lastniku, profilni sliki in opcijsko vsebujejo tudi sliko ali posnetek. Všečkanje in komentiranje komentarjev deluje na enak princip kakor pri objavah.

Komentarji komentarjev so zelo podobni komentarjem, razlikujejo se le po izgledu in nekaterih funkcionalnostih. Ko uporabnik odgovarja na komentar ali objavo lahko označi določeno osebo. Če se v vnosnem polju kot prvi znak pojavi @, se odpre enak seznam oseb, ki sem ga omenil že v glavi aplikacije, le da ta vsebuje zgolj uporabnikove prijatelje. Ob izbiri uporabnika na seznamu, se v besedilo polja vpiše ime uporabnika. V primeru, da se vpisana vrednost ob objavi komentarja ujema z obliko @Ime Priimek, sistem pošlje obvestilo osebi, da je bila označena pod objavo, del z imenom pa je prikazan krepko.

Ko sem razvijal del za komentarje se je izkazalo, da je v nekaterih primerih nujno, da je vsaka komponenta označena s svojim ključem. Prvotno na to nisem bil pozoren, zato se je odgovor na komentar dodal čisto drugemu komentarju na seznamu.

Za boljšo uporabniško izkušnjo sistem poskrbi, da vse profilne slike, ki so prikazane v tem sklopu odprejo posameznikov profil, kar je glavni razlog, zakaj komentar in objava vsebujeta ime o profilne slike uporabnika.



Slika 15: Seznam komentarjev

4.4.8 Profil

Izjemno pomembno je, da ima uporabnik svobodo in pregled nad podatki. Moja aplikacija mu to omogoča s komponento Profile, ki je sestavljena iz dela za spreminjanje in dela za pregled nad profilom. Pregled je sestavljen iz naslovne in profilne slike ter informacij: ime in priimek, opis, status, šola, e-poštni naslov, naslov bivanja, datum rojstva, telefonska številka ter najbolj priljubljeni predmeti. Uporabnik ima na voljo gumbe, ki omogočajo bližnjice za:

- Prikaz profila, kot ga vidijo uporabniki,
- Spreminjanje podatkov,
- Skrij ali pokaži vse podatke.

Spreminjanje profila vključuje obrazec, kjer uporabnik spreminja svoje podatke in dela za spreminjanje slik in opisa. Slike se prav tako nalagajo s pomočjo komponente react-dropzone, vendar sem to nekoliko prilagodil svojim potrebam. Uporabnik sliko prenese s pritiskom na gumb in je omejen samo na eno datoteko. Pri posodabljanju profilne slike, je zelo pomembno, da se ta posodobi tudi v piškotku in ostalih zapisih, kjer je zaradi hitrejšega dostopa naveden ta podatek. Obrazec s podatki ob odpiranju okna za spreminjanje iz baze naloži trenutne podatki. Ob pritisku na gumb so vse posodobljene vrednosti preverjene in v primeru uspešne validacije, se izvede posodabljanje.

Šola *
Vegova Ljubljana

Datum rojstva *
11-03-2022

Tvoj status
študent

Kje stanuješ? *
Ljubljana

Tvoj telefon
041-503-302

Naštej stvari, ki jih obvladaš

Shrani

Slika 16: Urejevalnik profila

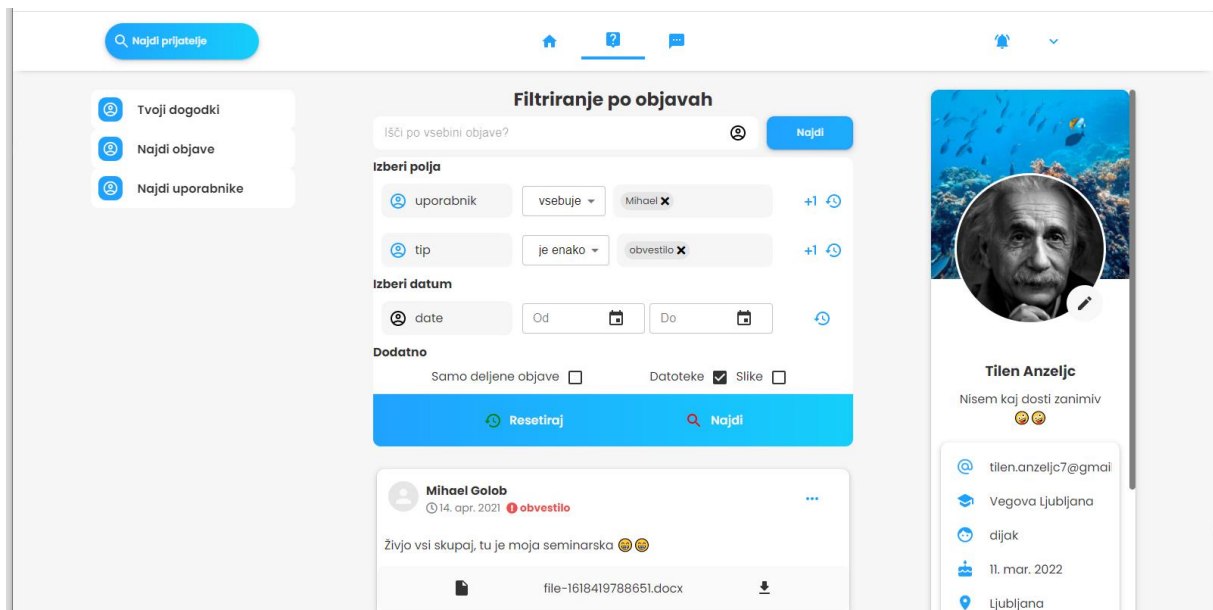
4.4.9 Dogodki

Sistem za dogodke je sestavljen iz koledarja, ki je del okolja MaterialUi in skrbnika za planiranje dogodkov in prikaz le teh. Koledar ob spremembi meseca naloži dogodke, ki jih prikaže ob pritisku na določen dan v spodnjem delu koledarja. Za upodabljanje dni za posamezen mesec, sem moral napisati svojo funkcijo, ki poleg številke dneva doda oznako, ki ponazarja, ali se bo na določen dan zgodil kateri izmed dogodkov. Ob pritisku uporabnika na posamezen dan se pod koledarjem prikažejo vsi dogodki, ki se bodo oz. so se že zgodili. Uporabniku so na voljo funkcije za brisanje, dodajanje in pregled dodatnih informacij, ki se prožijo ob pritisku na določen gumb.

Dodajanje dogodkov poteka zelo podobno, kot spreminjanje profila. Uporabnik v formo vpiše podatke in ob pritisku na gumb za določen dan shrani dogodek. Podatek za kraj in čas sem iz standardnega podatkovnega tipa timestamp pretvoril v obliko dd-mmm-yyyy hh:mm , s čimer sem se izognil nepotrebnim podatkom in pretvarjanju ob upodabljanju. Pozoren sem moral biti tudi na pretvorbo v slovenščino, predvsem pri mesecih, ki imajo drugačno kratico kakor v angleščini, npr. mesec maj in avgust. Vse kratice mesecev,, ki se ne ujema z angleškimi , zaradi morebitnih napak pretvoril.

4.5 Filtriranje

Tretji del aplikacije je stran za filtriranje, ki omogoča uporabniku bolj napredno iskanje po objavah, uporabnikih in kreiranih dogodkih. Za vse 3 tipe je uporabljen enak princip, razlike so le v parametrih iskanja in oblikovnih lastnostih. Na vrhu polja za filtriranje se nahaja vnosni element, ki skrbi za iskanje besedila, npr. opisa dogodka. Ta del deluje neodvisno in pri iskanju upošteva samo vneseno besedilo. Spodnji del filtra pa je nekoliko bolj kompleksen. Sestavljen je iz več polj, ki vsebujejo ime parametra, tip filtriranja, polje in gumb za dodajanje zapisov ter gumb za resetiranje posameznega polja. Datumska polja so nekoliko drugačna, saj nosijo zgolj ime ter začetno in končno mejo. Pri nekaterih iskanjih se na dnu nahajajo še izbirna okenska za ožji krog iskanja, npr. iskanje samo po prijateljih. Na koncu sledita še gumba za resetiranje in potrditev iskanja, ki skrbi za klic v ozadje aplikacije.



Slika 17: Stran za filtriranje

Ko se parametri pošljejo na strežnik se izmed njih izberejo bolj pomembni oz. tisti, ki omejijo iskane elemente na čim ožji krog. Preostali podatki iz baze pa se potem pregledajo s splošno JavaScript funkcijo, ki na podlagi vhodnih spremenljivk vrne drži ali ne drži. Vse funkcije se kličejo v enem pogojnem (angl. if) stavku in so združene z operatorjem logični in. V primeru, da parameter ni naveden funkcija avtomatsko vrne drži, saj nas ujemanje ne zanima. Pri implementaciji sem imel veliko težav, saj sem bil naučen na SQL poizvedbe, ki so narejene tudi za bolj kompleksne poizvedbe, medtem, ko baza, ki sem jo uporabil pri temu ni bila najboljša. Ravno zato, sem se potem odločil, da naredim preproste poizvedbe, s katerimi samo delno omejim število podatkov in jih nato naprej obdelam z JavaScript-om.

```
function check(type, value1, value2) {
  if (value1.length > 0) {
    if (type === 'vsebuje') {
      for (let value of value1) {
        value2 = value2.toLowerCase();
        value = value.toLowerCase();
        if (value2.includes(value))
          return true;
      }
      return false;
    } else if (type === 'je enako') {
      return value1.includes(value2) ? true : false;
    } else if (type === 'ni enako') {
      return value1.includes(value2) ? false : true;
    }
  } else {
    return true;
  }
}
```

Slika 18: JavaScript funkcija za filtriranje

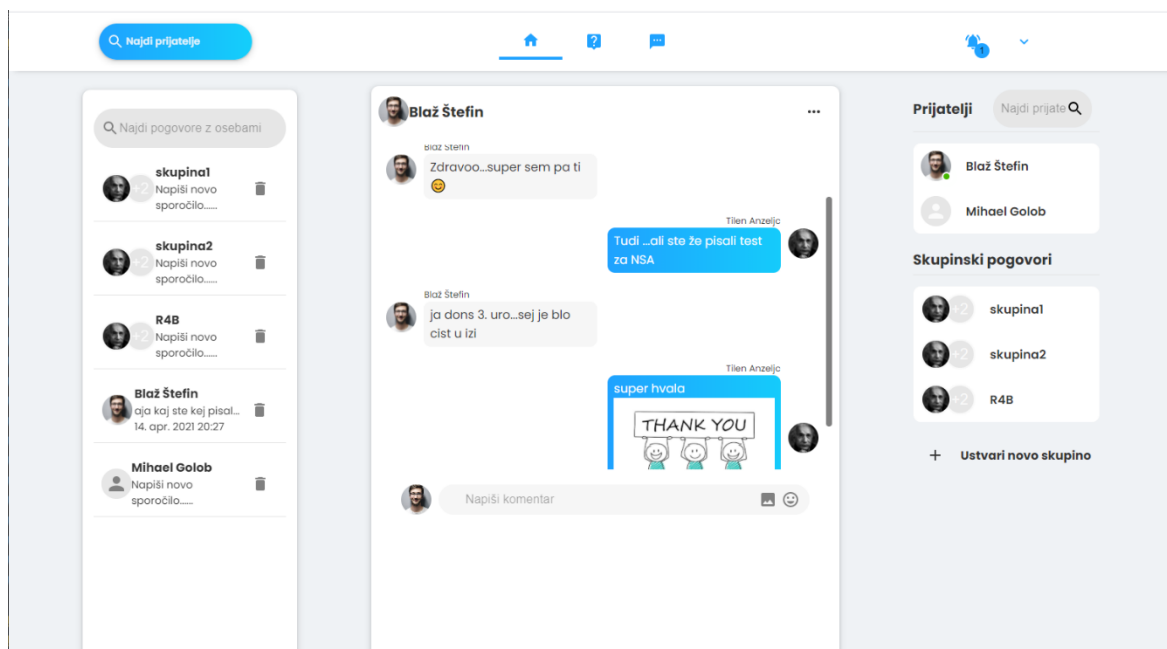
Po filtriranju se podatki prikažejo v komponentah, ki sem jih uporabljal že na zavihku domov, to so objave, dogodki in informacije o uporabniku.

4.6 Klepet

Stran za pogovarjanje predstavlja zadnji del aplikacije in je sestavljena iz 3 modulov. Na levi strani se nahaja seznam vseh pogovorov, na sredini trenutno odprt pogovor, na desni strani pa so uporabnikovi kontakti in skupine.

4.6.1 Kontakti in skupine

Polje s kontakti je seznam vseh podatkov trenutnega uporabnika. Vsak element na tem seznamu ima profilno sliko, ime in ponuja prikaz dodatnih informacij. Ob avatarju profila se pojavi tudi animacije zelene pike, če je uporabnik aktiven. Na vrhu je na voljo tudi polje za iskanje, ki uporablja enako logiko kakor vsi ostali vnosni elementi tega tipa. Ko se zazna pritisk miške na enega izmed elementov seznama sistem preveri, ali pogovor že obstaja. Na podlagi tega zgenerira novega, ali pa pridobi in prikaže podatke obstoječega. Zelo podobno delujejo tudi skupine, edina razlika je, da jih uporabnik lahko ustvari ročno. Za to poskrbi forma, ki sprejme podatek o imenu skupine in seznam uporabnikov. Za dodajanje sem porabil komponento iz sistema za filtriranje, kar je tudi glavni namen načina razvijanja s komponentami. Ko uporabnik ustvari skupino, se avtomatsko zgenerira nov pogovor.



Slika 19: Stran za pogovarjanje

4.6.2 Pogovori

Seznam pogovorov vsebuje skoraj identične komponente kakor kontakti. Odstranjena je zgolj opcija za prikaz dodatnih informacij, dodana pa sta zadnje napisano sporočilo, datum in ura kreiranja ter gumb za izbris.

Na seznamu je potrebno ločiti med prebranimi in neprebranimi pogovori. Število neprebranih je prikazano tudi poleg ikone v glavi za premikanje po aplikaciji. Pogovor se v neprebranega spremeni, ko se nahaja v polju unread. Neprebrani pogovori se nahajajo na vrhu seznama in so drugače oblikovani ter poleg vseh podatkov nosijo spremenljivko unread, na podlagi katere sistem spreminja oblikovne lastnosti.

Brisanje pogovorov poteka tako, da se vsem lastnikom razen brisalcu pošlje obvestilo, s katerim potrdijo, da se strinjajo z izbrisom celotnega pogovora. Če prošnjo za izbris zavrnejo, se pošiljatelju nazaj sporoči, da je bila zahteva preklicana.

Pogovor je glavni del klepetalnika. Sestavljen je iz glave, kjer se nahajajo informacije o osebi, s katero si dopisujemo. To je določeno tako, da se iz tabele lastnikov pogovora poišče vse tiste, ki se ne ujemajo s trenutnim prijavljenim uporabnikom.

Spodaj se nahaja polje s pogovorčki, ki nosijo podatke o imenu in profilni sliki lastnika, vsebini ter času kreiranja. Ko z miško preidemo čez posamezen pogovorček se prikažejo funkcije, ki so nam na voljo, prenos slike in izbris pogovorčka. Sporočilo se lahko nahaja na levi ali desni strani in je drugače oblikovano, kar se določi s primerjanjem lastnika pogovorčka in trenutno prijavljenega uporabnika. Pred prikazom celotnega seznama je potreben premik drsnika na dno, saj se pogovorčki gledajo iz dna proti vrhu.

Uporabnik sporočilo pošlje s pomočjo vnosnega polja, ki sem ga omenil že v komentarjih. Ob kreiranju je potrebno nastavljanje zadnjega napisanega sporočila. Nato sledi preverjanje ali gre za prebran oz. neprebran pogovor. Kot neprebranega ga obravnavamo takrat, ko se uporabnik ob novem sporočilu ne nahaja na tem pogovoru, kjer je bilo sporočilo generirano. Nato se pogovor doda v polje unread, kjer se izvede preverjanje, da ne pride do nepotrebnega podvajanja. Če velja, da se pogovor prvič dodaja, torej ni še neprebran in uporabnik ni aktiven, se pošlje tudi obvestilo o prejtem sporočilu. Pomembno je, da se preverja te stvari, saj bi drugače uporabnik ob vsakem ustvarjenem pogovorčku prejel ogromno obvestil, kar bi bilo precej moteče. Težave pri

komunikaciji v živo so se pojavile zaradi večkratne inicializacije povezave med Pusherjem in aplikacijo. Ker je v medpomnilniku obstajalo več povezav, se je pogovorček oz. celoten pogovor pri kreiranju dodal večkrat, čeprav je sistem od strežnika pridobil zgolj eno zahtevo. Preprosta rešitev je bila, da sem definiral novo spremenljivko z vrednostjo true in jo ob prvotni inicializaciji nastavil na vrednost false.

5. ODKRIVANJE NAPAK

Na strani strežnika se preverjajo splošne težave pri delovanju in se shranjujejo v .log datoteko. Sistem za preverjanje napak deluje s pomočjo Winston knjižnice in middleware funkcije Errors. Poleg splošnih napak strežnika pa se preverjajo tudi ostale nepravilnosti pri operacijah s podatki. V primeru težav, se iz ozadja pošlje odgovor, ki nosi podatke, kaj je šlo narobe in kakšnega tipa je ta napaka. Npr. tip napake za avtorizacijo je 401, medtem, ko je 400 oznaka za splošen neuspeh pri pridobitvi podatkov.

Na strani odjemalca pa preverjanje realizirano s pomočjo »try« in »catch« blokov. Ko sistem prestreže napako se kliče funkcija, ki preveri, za kakšen tip gre in na podlagi tega uporabniku pokaže določeno besedilo. Za prikazovanje skrbi React-toastify, ki naredi javljanje napak uporabniku prijazno in čim manj stresno, saj so sporočila lepo oblikovana in animirana.

```
function error(err) {  
  if (err.response && err.response.status === 401) {  
    toast.info("🔒 Za nadaljevanje se moraš prijaviti.", {  
      position: "top-center",  
      autoClose: 1200,  
    });  
    logoutUser();  
  } else {  
    toast.error("❌ Prišlo je do napake.", {  
      position: "top-center",  
      autoClose: 1000,  
    });  
  }  
}
```

Slika 20: Funkcija, ki poskrbi za prikaz napak

6. BAZA

Baza je sestavljena iz 7 dokumentov, ki predstavljajo podatke za uporabnike, objave, dogodke, komentarje, pod komentarje, pogovore in pogovorčke. Ob inicializaciji je najprej potrebno definirati shemo vseh elementov baze. S pomočjo te se določijo podatkovni tipi, omejitve, ki se ujemajo s tistimi v aplikaciji in relacije med podatki.

Ker je MongoDB ne relacijska baza podatkov, se uporabljajo nekoliko drugačni pristopi pri definiranju sheme. Struktura baze nima ene same rešitve za relacije med podatki, kot velja za ER model SQL baz, zato jo lahko razvijalec sproti prilagaja potrebam aplikacije. Shema se definira ob kreiranju dokumenta, ki ga program avtomatsko generira, v primeru, da določen dokument ne nahaja v podatkovni bazi.

Pri relacijah sta na voljo dva principa povezovanja elementov:

- povezovanje (angl. Link)
- vstavljanje (angl. Embedding)

Povezovanje je zelo podobno kakor pri navadnih bazah. Zapisi se povezujejo na podlagi enoličnih šifer z relacijami 1:1, 1:m, m:m. Edina razlika, ki predstavlja tudi veliko optimizacijo je, da za relacijo mnogo proti mnogo ni potrebno kreiranje dodatne tabele, ki nosi povezavo med posameznimi elementi. To enostavno rešimo s pomočjo podatkovnega tipa Array, v katerem navedemo vse šifre, ki so povezane z danim elementom. Odličen primer je dokument uporabnika, ki vsebuje tabelo šifer objav, prijateljev in ostalih podatkov. S pomočjo Array tipa lahko v en dokument vstavimo podatke, ki jih zelo pogosto potrebujemo in jih tako ni potrebno vedno pridobivati iz baze.

Pri vstavljanju, kot nam že ime pove vstavimo celotne podatke v dokument. Torej namesto šifer bodo v tabeli kar celotni podatki, ki imajo relacijo na ta dokument. V

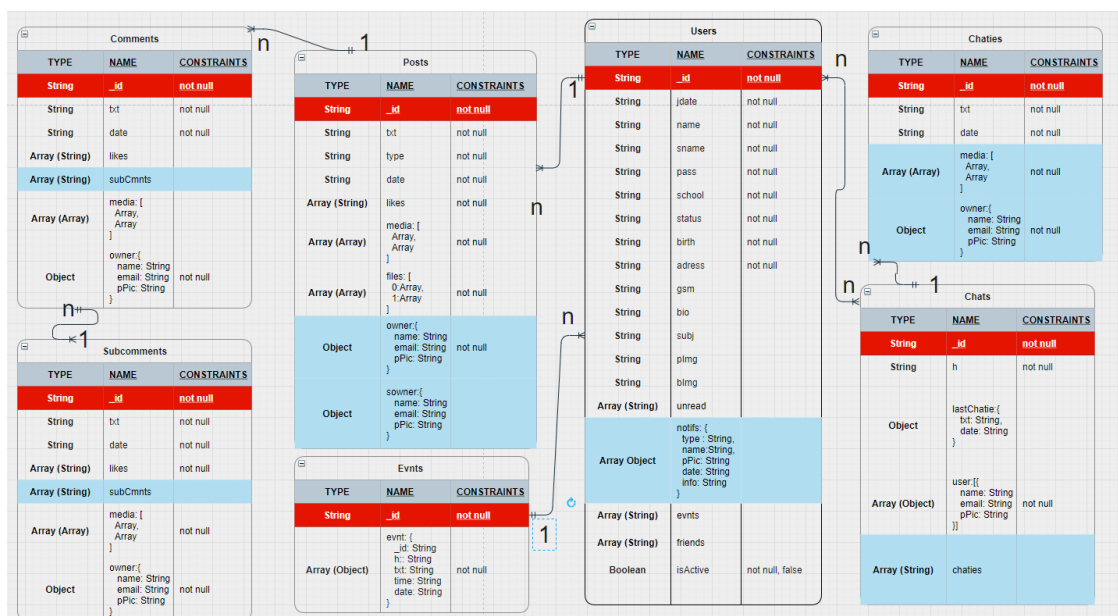
```
const mongoose = require("mongoose");

const chatieSchema = new mongoose.Schema({
  _id: {
    type: String,
    required: true,
  },
  owner: {
    name: {
      type: String,
      required: true
    },
    email: {
      type: String,
      required: true
    },
    pPic: {
      type: String,
    }
  },
  txt: {
    type: String,
  },
  date: {
    type: String,
    required: true
  },
  media: { type: Array },
  files: { type: Array },
});

module.exports = mongoose.model("Chatie", chatieSchema);
```

Slika 21: Koda za kreiranje sheme sporočila

določenih primerih je to zelo uporabna metoda, saj zmanjša število potrebnih transakcij in omogoča hitrejšo pridobitev željenih podatkov. Zelo pogosto pa se zgodi, da takšen način relacij ni najbolj primeren, saj pride do podvajanja. Sam sem ga uporabil za obvestila, ki so vgrajena v uporabniku, saj so po velikosti zelo majhna, vendar jih je potrebno velikokrat pridobiti iz baze.



Slika 22: Skica ER modela

Zaradi drugačnih pristopov pri relacijah, ne moremo ER (entitetno relacijskega) diagrama definirati enako, kakor pri SQL bazah. Zgornja slika je zgolj skica, kjer so označene glavne povezave. Rdeče označene vrstice so glavni ključ, modre pa predstavljajo večje povezave med elementi. Vsak dokument vsebuje tudi nekaj podatkov drugih dokumentov, zaradi razlogov omenjenih zgoraj, vendar takšnih povezav, zaradi boljše preglednosti nisem vključil v skico.

Baza ima definiranega enega samega uporabnika, ki ima pravice do spreminjanja in branja samo iz baze socialnega omrežja. Zaradi varnosti se pred povezovanjem na bazo izvede avtorizacija uporabnika. Da procesi v ozadju vejo, da je potrebno preverjanje prijave, se poleg ukaza za zagon doda ukaz -auth.

Ker je maksimalna dolžina dokumenta 16 MB, sem moral za shranjevanje večjih datotek uporabiti MongoDB-jev sistem GridFS. Deluje tako, da datoteko shrani v dva različna dokumenta. V enem se nahajajo osnovni podatki, kot so ime, velikost in tip. Drugi dokument pa je sestavljen iz 255 KB velikih delčkov, na katere se razbije

datoteka. Ko želi aplikacija pridobiti določeno datoteko se zberejo vsi delčki z določeno šifro in se sestavijo v človeku dosegljivo datoteko.

Zaradi realnočasovne komunikacije a Pusher-jem je bilo bazo potrebno definirati kot repliko (angl. Replica set). Replika je skupina procesov v ozadju, ki vzdržuje isti nabor podatkov. Ti procesi se uporabljajo v produkcijskem okolju, za razpoložljivost in redundantnost podatkov, v primeru, da pride do sesutja enega izmed strežnikov.

Poizvedbe v bazo so zelo podobne .JSON formatu, v kateremu podamo vse omejitve. Zaradi takšne oblike so nekoliko okrnjene, vendar zadostujejo za bolj preproste poizvedbe. Pri transakcijah v več dokumentov hkrati, pa sem uporabil knjižnico Fawn. Ko uporabnik izbriše komentar se more izvesti še kup ostalih operacij, kot je npr. brisanje iz objave. Namesto, da bi za vsako posebej pisali kodo, lahko to s Fawn zapišemo kot eno transakcijo po delih.

```
router.delete("/dropCmnt", verifyJWT, async (req, res) => {
  new Fawn.Task()
    .remove('comments', { _id: req.body._id })
    .remove('subcomments', { _id: { $in: req.body.subCmntIds } })
    .update('posts', { _id: req.body.postId }, {
      $pull: {
        cmnts: req.body._id //odstrani _id komentarja iz arraya
      }
    })
    .run();
  res.status(200).send("Uspešno izbrisan komentar");
});
```

Slika 23: Primer poizvedbe v bazo

7. ZAKLJUČEK

Kljub relativno velikemu izzivu, ki sem si ga zadal mi je uspelo razviti aplikacijo, s katero sem precej zadovoljen. Spoznal sem, da se vložen trud in odrekanje prostemu času za učenje na koncu vedno poplača, saj sem pridobil ogromno neprecenljivih izkušenj, ki predstavljajo dobro podlago za prihodnost. Naučil sem se veliko novih metod, ki se uporabljajo pri spletnem programiranju in poglobil znanje oblikovanja spletnih strani ter programskega jezika JavaScript. Izboljšal sem svoje sposobnosti reševanja problemov in iskanja zanesljivih virov na spletu.

V prihodnosti imam namen aplikaciji dodati še nekaj funkcionalnosti, ki je zaradi pomankanja časa in šole nisem mogel realizirati. Še naprej bom nadgrajeval svoje znanje in se ukvarjal s področjem spletnega programiranja, saj sem ne dolgo nazaj dobil ponudbo za sodelovanje pri razvoju večje aplikacije.

8. VIRI IN LITERATURA

8.1 SPLETNE STRANI

- <https://www.youtube.com/c/StefanMischook/videos> (obiskano dne 3. 8. 2020)
- <https://socialmediaweek.org/blog/2017/11/create-social-network-platform-5-steps/> (obiskano dne 8. 9. 2020)
- <https://fonts.google.com/> (obiskano dne 8. 9. 2020)
- <https://developer.mozilla.org/en-US/docs/Web/JavaScript> (obiskano dne 10. 9. 2020)
- <https://www.w3schools.com/> (obiskano dne 10. 9. 2020)
- <https://codewithmosh.com/courses/> (obiskano dne 15. 9. 2020)
- <https://reactjs.org> (obiskano dne 14. 10. 2020)
- <https://www.youtube.com/c/DevEd/videos> (obiskano dne 15. 10. 2020)
- <https://www.npmjs.com> (obiskano dne 15. 10. 2020)
- <https://stackoverflow.com/> (obiskano dne 16. 10. 2020)
- <https://www.youtube.com/c/TraversyMedia/videos> (obiskano dne 28. 10. 2020)
- <https://nodejs.org/en> (obiskano dne 3. 11. 2020)
- <https://expressjs.com> (obiskano dne 3. 11. 2020)
- <https://jwt.io/> (obiskano dne 4. 11. 2020)
- <https://nodemailer.com/about/> (obiskano dne 4. 11. 2020)
- <https://www.pinterest.com/> (obiskano dne 15. 11. 2020)
- <https://dribbble.com/> (obiskano dne 15. 11. 2020)
- <https://material-ui.com> (obiskano dne 22. 11. 2020)
- <https://developer.mozilla.org/en-US/docs/Web/CSS> (obiskano dne 22. 11. 2020)
- <https://www.youtube.com/c/AnthonySistilli/videos> (obiskano dne 22. 11. 2020)
- <https://www.youtube.com/c/CleverProgrammer/videos> (obiskano dne 20. 12. 2020)
- <https://www.mongodb.com> (obiskano dne 28. 12. 2020)
- <https://www.youtube.com/c/MongoDBofficial/videos> (obiskano dne 7. 1. 2021)
- <https://www.freecodecamp.org/news/grids-making-file-uploading-to-mongodb> (obiskano dne 22. 1. 2021)

- <https://dev.to/vaibhavkhulbe/7-security-tips-for-your-react-application-4e78> (obiskano dne 8. 2. 2021)
- <https://alligator.io/nodejs/express-cookies/> (obiskano dne 10. 2. 2021)
- <https://www.youtube.com/c/WebDevSimplified/videos> (obiskano dne 12. 2. 2021)
- <https://www.youtube.com/c/FredrikChristenson/videos> (obiskano dne 12. 2. 2021)
- <https://blog.logrocket.com/4-ways-to-render-large-lists-in-react/> (obiskano dne 2. 3. 2021)
- <https://pusher.com/> (obiskano dne 14. 3. 2021)
- <https://fkhadra.github.io/react-toastify/introduction/> (obiskano dne 22. 3. 2021)
- <https://viri.cjvt.si/sopomenke/slv/> (obiskano dne 27. 3. 2021)
- <https://fran.si/iskanje?FilteredDictionaryIds=130&View=1&Query=%2A> (obiskano dne 27. 3. 2021)
- https://medium.com/@blockchain_simplified/what-is-mern-stack-9c867dbad302 (obiskano dne 28. 3. 2021)
- <https://www.hostko.si/podpora/kb/kaj-je-node-js/> (obiskano dne 28. 3. 2021)
- <https://www.neoserv.si/blog/kako-pognati-react-aplikacijo> (obiskano dne 28. 3. 2021)
- <https://nsa-splet.si/js/zgledi/js-zgledi-22-dom.php> (obiskano dne 28. 3. 2021)
- <http://dis-slovarcek.ijs.si/> (obiskano dne 1. 4. 2021)

6.2 SLIKE

- Slika 1: <https://www.mongodb.com/mern-stack> (obiskano dne 3. 4. 2021)
- Slika 4: <https://www.optimizesmart.com/advanced-google-universal-analytics-tracking-beginners-guide/> (obiskano dne 9. 4. 2021)

IZJAVA O AVTORSTVU

Izjavljam, da je strokovno poročilo Socialno omrežje v celoti moje avtorsko delo, ki sem ga izdelal samostojno s pomočjo navedene literature in pod vodstvom mentorja.

Ljubljana, 14. 4. 2021

Tilen Anzeljc