Izrada web aplikacije za upravljanje kalendarom djelatnika fakulteta

Glavaš, Tihana

Master's thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:332068

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-16**

Repository / Repozitorij:

Digital Repository Juraj Dobrila University of Pula

Sveučilište Jurja Dobrile u Puli
Fakultet informatike

TIHANA GLAVAŠ

IZRADA WEB APLIKACIJE ZA UPRAVLJANJE KALENDAROM DJELATNIKA FAKULTETA

Diplomski rad

Pula, rujan 2023. godine

Sveučilište Jurja Dobrile u Puli
Fakultet informatike

TIHANA GLAVAŠ

IZRADA WEB APLIKACIJE ZA UPRAVLJANJE KALENDAROM DJELATNIKA FAKULTETA

Diplomski rad

JMBAG: 0303010993
Studijski smjer: Sveučilišni diplomski studij Informatika
Predmet: Izrada informatičkih projekata
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija
Mentor: doc. dr. sc. Nikola Tanković

Pula, rujan 2023. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisana **Tihana Glavaš**, kandidatkinja za magistra **informatike** ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, 20. rujna 2023.

IZJAVA O KORIŠTENJU AUTORSKOG DJELA

Ja, **Tihana Glavaš** dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom **„Izrada web aplikacije za upravljanje kalendarom djelatnika fakulteta“** koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

Student

_____

U Puli, 20. rujna 2023.

Sadržaj

1. UVOD 1
2. MOTIVACIJA 2
2.1. SWOT analiza 3
2.2. Prednosti „ureda bez papira“ 4
3. ANALIZA KORISNIČKIH ZAHTJEVA 5
3.1. Use Case dijagram 6
3.2. Dijagram aktivnosti 7
4. KORIŠTENE TEHNOLOGIJE I ALATI 8
4.1. Arhitektura MERN 8
4.2. Glavne tehnologije 10
4.3. Razvojni alati 10
4.4. Pomoćni paketi 12
5. IMPLEMENTACIJA 13
5.1. Arhitekturni stil REST 13
5.1.1. *Ograničenja* 14
5.1.2. *Anatomija zahtjeva* 15
5.1.3. *Komunikacija s REST-om* 16
5.2. Implementacija klijentske strane 18
5.2.1. *Komponenta „App.jsx“* 19
5.2.2. *Komponenta „Form“* 21
5.2.3. *Komponenta „Calendar“* 24
5.3. Implementacija poslužiteljske strane 26
5.3.1. *Agregacijski pipeline* 29
6. KORISNIČKE UPUTE 33
6.1. Korisničke upute – profesor 36
6.2. Korisničke upute – tajnica 51
6.3. Korisničke role i prava pristupa 59
7. BUDUĆI RAZVOJ 60
8. ZAKLJUČAK 61
LITERATURA 62
POPIS SLIKA 63
PRILOZI 64
SAŽETAK 65
SUMMARY 66

1. UVOD

Uvođenje aplikacija u svakodnevne poslovne procese postalo je neizbježno u suvremenom poslovnom okruženju. Jedno od područja koje zahtijeva preciznu evidenciju i praćenje su i radni dani zaposlenika, a ciljano tržište ove aplikacije su sveučilišta i obrazovne institucije koje traže svestrano rješenje za evidenciju i praćenje radnih dana svojih zaposlenika čime bi se unaprijedio i sam proces.

U radu se predstavlja web aplikacija namijenjena zaposlenicima sveučilišta, odnosno tajnici i profesorima. Ova aplikacija predstavlja nastavak projekta započetog na kolegiju Izrada informatičkih projekata. Trenutno, aplikacija omogućava samo prijavu korisnika putem korisničkog imena i lozinke, međutim još nisu implementirane korisničke role i pravo pristupa poput mogućnosti tajnice da vidi izvještaj svih djelatnika fakulteta, dok bi profesor mogao pristupiti samo vlastitom mjesečnom izvještaju. To znači da tajnica i profesori imaju jednake ovlasti. Također, početna stranica aplikacije je koncipirana kao kalendar izrađen u Reactu, koji omogućava profesorima bilježenje njihovih radnih dana. Važno je napomenuti da se trenutno ti podaci, odnosno „eventi“ u kalendaru, ne pohranjuju u bazu podataka.

Uvođenjem ove aplikacije ostvaruju se brojne prednosti. Prvo, aplikacija omogućava brzu i jednostavnu evidenciju radnih dana zaposlenika, odnosno dana kada su radili na sveučilištu, radili od kuće, bili na službenom putu, bolovanju ili godišnjem odmoru. Zatim, svi ti podaci su dostupni u realnom vremenu, čime se poboljšava komunikacija i omogućava bolje planiranje vremena. Na kraju, aplikacija pruža izvještaje i analize o radnim danima zaposlenika.

2. MOTIVACIJA

Ova aplikacija za evidenciju radnih dana zaposlenika ima za cilj unaprijediti proces praćenja i evidentiranja radnog vremena djelatnika na sveučilištu. S obzirom na kompleksnost i broj zaposlenika na sveučilištima, dosadašnji procesi vođenja evidencije mogu biti neprecizni i podložni pogreškama. Uvođenjem ove aplikacije postiže se preciznija evidencija radnih dana, povećava se transparentnost te omogućava brže generiranje izvještaja.

Ciljano tržište aplikacije za evidenciju radnih dana zaposlenika su sveučilišta i obrazovne institucije. Sveučilišta su kompleksne organizacije s velikim brojem zaposlenika koji imaju različite radne obveze i odgovornosti. Dosadašnji proces evidencije radnih dana zaposlenika na sveučilištima često se odvijao ručno ili putem tradicionalnih metoda poput Excel tablica ili papirnatih obrazaca. Ovi procesi su neefikasni, podložni pogreškama i zahtijevaju puno vremena za administrativno održavanje. Na tržištu postoje i konkurentska rješenja koja nude aplikacije za evidenciju radnog vremena. Međutim, većina tih rješenja nije prilagođena specifičnim potrebama sveučilišnih organizacija. Stoga postoji potreba za prilagođenom aplikacijom koja će zadovoljiti sveučilišne potrebe u pogledu evidencije radnih dana zaposlenika.

2.1. SWOT analiza

SWOT analiza daje uvid u snage, slabosti, prilike i prijetnje vezane uz uvođenje aplikacije za evidenciju radnih dana zaposlenika sveučilišta:

Snage:

- Poboljšana točnost i preciznost evidencije radnog vremena
- Povećana transparentnost i komunikacija između profesora i uprave
- Brže i efikasnije generiranje izvještaja o prisutnosti i odsutnosti
- Automatizacija administrativnih procesa i smanjenje mogućnosti ljudske pogreške

Slabosti:

- Potreba za ulaganjem u razvoj i implementaciju aplikacije
- Potrebna je obuka zaposlenika za korištenje nove aplikacije

Prilike:

- Rast potražnje za efikasnijim upravljanjem radnim vremenom u sveučilišnim organizacijama
- Nedostatak prilagođenih rješenja na tržištu koja zadovoljavaju sveučilišne potrebe

Prijetnje:

- Konkurencije od postojećih rješenja koja se mogu prilagoditi sveučilišnim potrebama (Google Docs)
- Otpor zaposlenika prema promjenama u sustavu evidencije radnog vremena

Ukratko, uvođenje aplikacije za evidenciju radnih dana zaposlenika pruža organizacijsku korist i olakšava administraciju za upravu sveučilišta, ali i za profesore.

2.2. Prednosti „ureda bez papira“

„Ured bez papira“ (engl. „paperless office“) je koncept koji se sve više istražuje u poslovnom svijetu. Ovaj koncept teži eliminaciji upotrebe papira u svakodnevnim poslovnim operacijama, zamjenjujući ih elektroničkim dokumentima i digitalnim procesima. Implementacija „ureda bez papira“ donosi brojne prednosti, uključujući smanjenje troškova, brži pristup informacijama, bolju organizaciju i povećanu ekološku svijest.

Međutim, prelazak na „ured bez papira“ može biti izazovan za organizacije koje već dugi niz godina koriste tradicionalni papirnati sustav. No, kako se tehnologija razvija, olakšava prelazak na digitalne poslovne procese, što dovodi do povećane efikasnosti i smanjenog utjecaja na okoliš.

3. ANALIZA KORISNIČKIH ZAHTJEVA

Prilagodba za mobilni prikaz: Potrebno je poboljšati web aplikaciju kako bi se osigurao optimalan prikaz na mobilnim uređajima.

Korisničke role: Implementirati sustav korisničkih rola kako bi se omogućile različite razine pristupa i funkcionalnosti. Na primjer, tajnica će imati privilegije za pregled izvještaja svih djelatnika fakulteta, dok će profesor moći pristupiti samo vlastitim mjesečnim izvještajima.

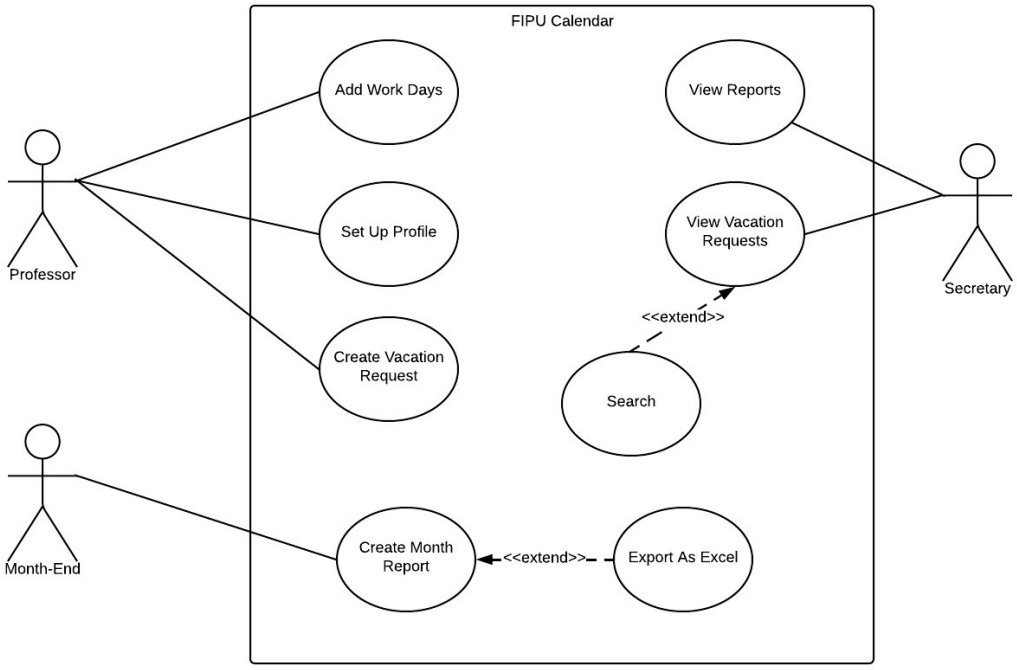
Profesorima je već omogućen unos različitih vrsta događaja (radni dan, rad od kuće, službeni put, bolovanje, godišnji odmor). Međutim, potrebno je dodatno implementirati funkcionalnost za ažuriranje (update) i brisanje (delete) tih događaja. Trenutno se događaji uneseni u kalendar ne spremaju u bazu podataka.

Mjesečni izvještaj o radnim danima: Implementirati funkcionalnost koja omogućuje generiranje mjesečnog izvještaja o radnim danima za sve profesore kao i za svakog profesora pojedinačno.

Zahtjev za godišnjim odmorom: Uključiti u aplikaciju mogućnost predaje zahtjeva za godišnji odmor kojeg će tajnica potom odobriti ili odbiti. Također, omogućiti tražilicu koja će olakšati pronalaženje specifičnog profesora.

3.1. Use Case dijagram

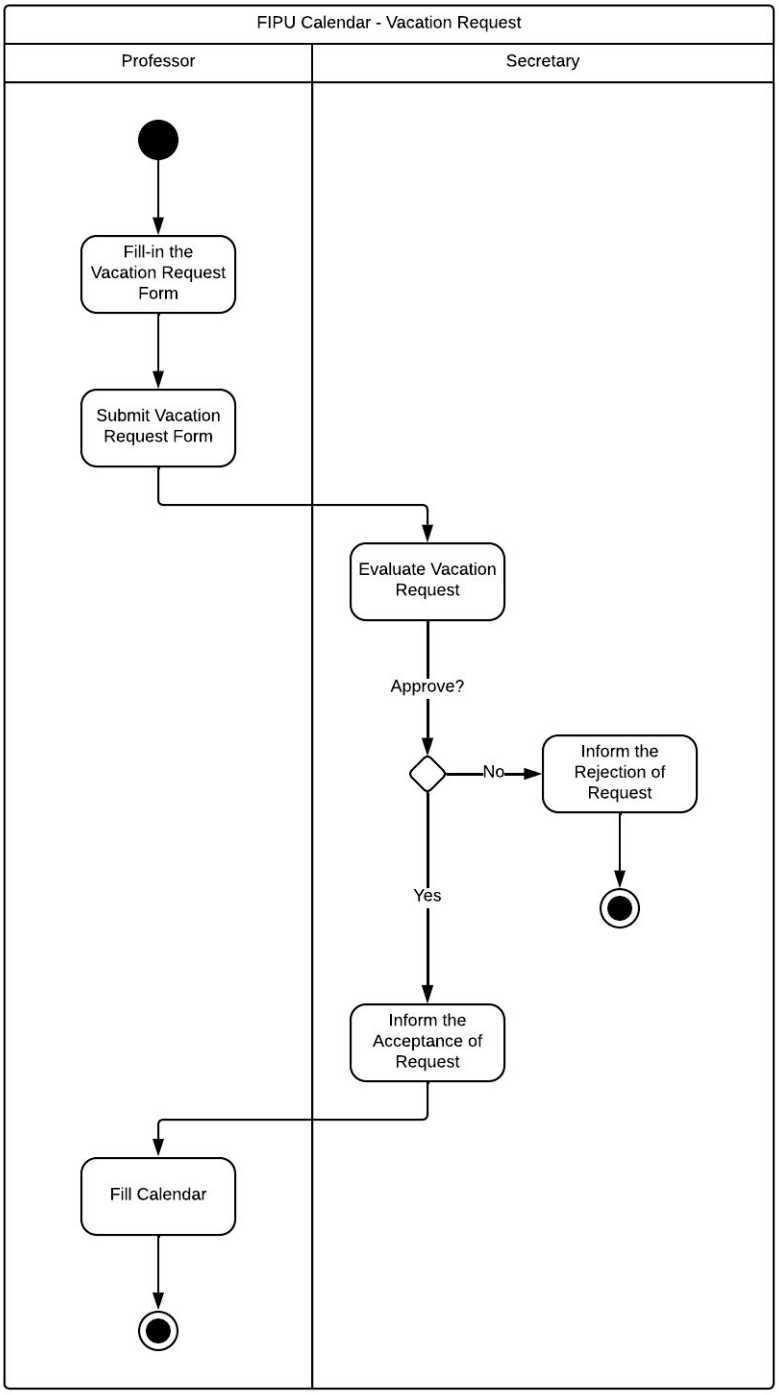
Use case dijagram koji se nalazi na slici 1 pokazuje na koje se načine aplikacija može koristiti. Dijagram se sastoji od tri aktora, profesora, tajnice i aktora koji označava kraj mjeseca. Glavne funkcionalnosti aplikacije su dodavanje radnih dana u kalendar, kreiranje zahtjeva za godišnji odmor te generiranje mjesečnog izvještaja kojeg tada tajnica može pregledavati. Dodavanje radnih dana (rad, rad od kuće, bolovanje, službeni put i godišnji) je ručno, a mjesečni izvještaj bi se trebao izvršavati automatski. Također tajnici treba omogućiti uvid u predane zahtjeve za godišnji odmor, koje onda tada po potrebi treba ažurirati.



Slika 1. Use Case dijagram sustava (izvor: autor)

3.2. Dijagram aktivnosti

U dijagramu aktivnosti kojeg možemo vidjeti na slici 2 je detaljnije prikazana predaja zahtjeva za godišnji odmor.



Slika 2. Dijagram aktivnosti za predaju zahtjeva za godišnji odmor (izvor: autor)

4. KORIŠTENE TEHNOLOGIJE I ALATI

U ovom poglavlju detaljno ćemo pregledati tehnologije i alate koji su korišteni za razvoj aplikacije. Svrha ovog pregleda je bolje razumijevanje okoline i infrastrukture samog projekta.

4.1. Arhitektura MERN

Arhitektura cjelokupnog rješenja za aplikaciju evidencije radnih dana zaposlenika na sveučilištu temelji se na tzv. MERN (MongoDB, Express.js, React.js, Node.js) Stack razvojnom okviru. Ova arhitektura odabrana je zbog svoje skalabilnosti, fleksibilnosti i podrške za razvoj modernih web aplikacija.

MERN stog (engl. stack) je skup tehnologija koji se često koristi za izgradnju modernih web aplikacija. Evo što svaka od tih tehnologija predstavlja:

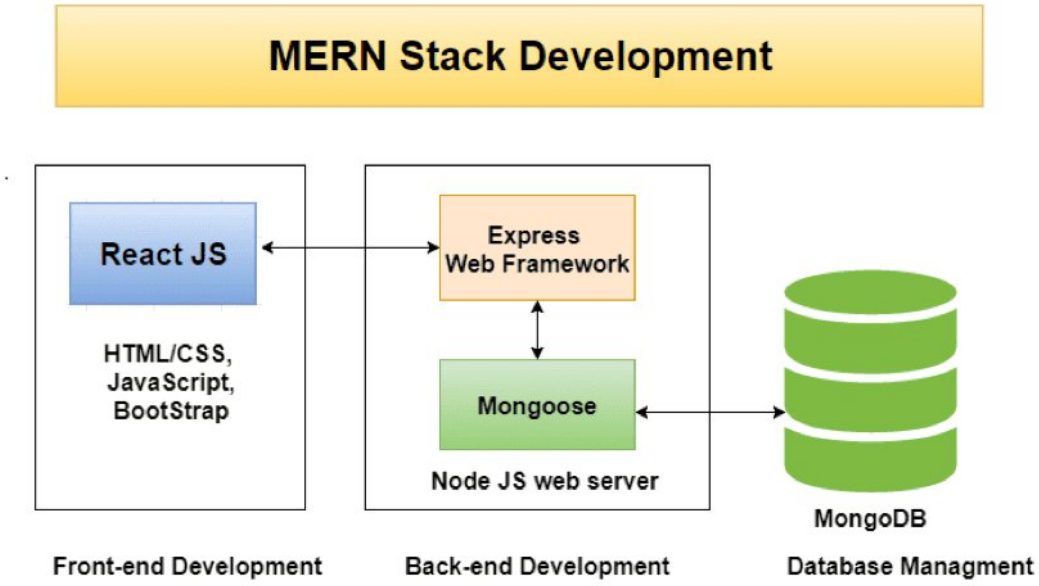
MongoDB je vrsta baze podataka koja koristi dokumente u JSON formatu za pohranu podataka što znači da se podaci pohranjuju u obliku struktura koje podsjećaju na JavaScript objekte.

Express.js je popularni web okvir za Node.js koji olakšava izgradnju i upravljanje web aplikacijama na strani poslužitelja. Express.js pruža razne alate i funkcionalnosti za upravljanje rutama, obradu HTTP zahtjeva i komunikaciju s bazom podataka.

React.js je JavaScript biblioteka te se koristi za izgradnju korisničkih sučelja i omogućuje dinamičko ažuriranje stranice bez potrebe za osvježavanjem cijele stranice. Koristi se za izgradnju komponenata koje čine osnovnu strukturu i interakciju aplikacije stoga je posebno dobar za složene aplikacije s puno interakcije.

Node.js je platforma koja omogućuje izvršavanje JavaScript koda na poslužiteljskoj strani, što je ključno za izgradnju web aplikacija.

U nastavku na slici broj 3 je prikazan ilustrativni dijagram arhitekture na kojoj možemo vidjeti kako funkcionira MERN stog.



Slika 3. Ilustrativni dijagram MERN Stack razvojnog okvira (izvor: [6])

React.js se koristi za izgradnju korisničkog sučelja i prikaz podataka na klijentskoj strani. On komunicira s poslužiteljskom stranom kako bi dohvatio ili poslao podatke.

Express.js i Node.js čine poslužiteljsku stranu aplikacije. Express.js omogućuje obradu zahtjeva, dok je Node.js platforma za izvođenje aplikacije. Zajedno komuniciraju s bazom podataka MongoDB u kojoj se pohranjuju podaci.

Korištenjem MERN stoga, razvojni programeri mogu izgraditi cijelu web aplikaciju koristeći samo JavaScript i JSON što olakšava razvoj modernih i dinamičnih web aplikacija.

Odabir MERN Stack razvojnog okvira donosi niz prednosti:

Skalabilnost: Arhitektura temeljena na JavaScriptu i Node.js-u omogućuje skalabilnost i mogućnost rada s velikim brojem korisnika i podataka.

Brzina razvoja: Upotreba React.js-a i Node.js-a olakšava razvoj korisničkog sučelja i poslovne logike, uz ponovno korištenje komponenata i biblioteka.

Fleksibilnost: Budući da su u MongoDB bazi podataka podaci organizirani u dokumente to pruža fleksibilnost u pohrani podataka, što je važno za različite vrste informacija o radnim danima zaposlenika.

4.2. Glavne tehnologije

Osim **React.js**-a na klijentskoj strani i **Express.js**-a na poslužiteljskoj strani koji su opisani u prethodnom potpoglavlju o arhitekturi MERN i koji su temeljni dio ove aplikacije, korištene su još i sljedeće tehnologije:

React Router: Ovaj paket omogućuje upravljanje rutama i navigacijom unutar React aplikacije na klijentskoj strani što omogućuje jednostavnu navigaciju između različitih dijelova aplikacije.

Axios: Axios je HTTP klijent koji koristimo za komunikaciju s našim backend serverom putem HTTP zahtjeva. Omogućava nam slanje i primanje podataka između frontend i backend dijela aplikacije.

4.3. Razvojni alati

U razvojnom procesu korišteni su sljedeći alati za olakšavanje razvoja i testiranja:

Vite[1] (dolazi od francuske riječi što znači „brzo“, a izgovara se /vit/) je brzi razvojni alat koji je odabran za razvoj ove aplikacije. Tijekom razvoja, Vite izbjegava kompleksnu konfiguraciju i brzo osvježava aplikaciju pri svakoj promjeni, što rezultira gotovo trenutnim prikazom promjena u pregledniku. Ova brzina čini razvojni proces učinkovitijim. Osim React.js-a, Vite podržava i druge okvire poput Vue.js-a i Svelte-a.

Vite je alat za razvoj web aplikacija koji rješava probleme koji su nastali s evolucijom web razvoja. Razvojem složenijih aplikacija broj JavaScript modula dramatično je porastao što je stvorilo izazove razvojnim alatima temeljenim na JavaScriptu. Vite koristi nove tehnologije, poput ES modula (standardizirani način za organiziranje i uvoz/izvoz JavaScript koda iz jedne datoteke u drugu) i alata napisanih u jezicima koji

[1] https://vitejs.dev/

se kompajliraju u native kod, odnosno izvršni kod kako bi pružio brže iskustvo razvoja u usporedbi s postojećim alatima [11].

Nodemon[2] se koristi kao razvojni alat za automatsko ponovno pokretanje Node.js aplikacije tijekom razvoja. Umjesto ručnog pokretanja servera svaki put kada se napravi promjena u kodu, nodemon automatski detektira te promjene i automatski ponovno pokreće server. Iako je prvotno razvijen za Node.js, može se koristiti za praćenje promjena u drugim jezicima kao što su Python, Ruby i drugi [9].

Concurrently[3] je alat koji omogućuje pokretanje više skripti istovremeno. Koristi se za pokretanje servera i klijentske strane paraleno. Primjerice u MERN aplikaciji imamo zasebne naredbe za pokretanje klijentske i poslužiteljske strane `"npm run server"` i `"npm run client"`, a naredba `"concurrently \"npm run server\" \"npm run client\""` istovremeno pokreće te dvije skripte. Ovo može značajno ubrzati razvojni proces jer omogućuje brzu provjeru rada aplikacije na obje strane (klijentskoj i poslužiteljskoj).

Visual Studio Code[4] je integrirano razvojno okruženje koje olakšava uređivanje koda i jedan od najpopularnijih alata među programerima zbog svoje jednostavnosti, fleksibilnosti i bogatog sustava proširenja. VS Code ima ugrađenu integraciju za Git koji omogućuje praćenje verzija koda, pretraživanje povijesti i druge operacije vezane uz verzioniranje.

Thunder Client[5] je proširenje u Visual Studio Code-u za testiranje API-ja (engl. Application Programming Interface) i ruta koje komuniciraju s backendom čime se ubrzava razvojni proces i omogućava brže otklanjanje problema. Ukratko, Thunder Client olakšava testiranje API-ja i praćenje njihovih odgovora.

Chrome Developer Tools[6], (poznat i kao DevTools) je skup alata koji su ugrađeni izravno u Google Chrome web preglednik. Namijenjeni su programerima i omogućuju im analizu i debugiranje web aplikacije tijekom razvoja. Omogućuje inspekciju elemenata, odnosno pregled HTML i CSS koda web stranice, analizu mrežnog prometa, odnosno praćenje svih HTTP zahtjeva koje web stranica šalje prema

[2] https://www.npmjs.com/package/nodemon
[3] https://www.npmjs.com/package/concurrently
[4] https://code.visualstudio.com/
[5] https://www.thunderclient.com/
[6] https://developer.chrome.com/docs/devtools/

poslužitelju, omogućujući programerima u praćenju performansi i provjeri resursa koji se prenose.

4.4. Pomoćni paketi

Styled-components omogućuju stilizaciju React komponenata koristeći tzv. „CSS-in-JS" pristup, odnosno mogućnost definiranja stilova direktno u JavaScript datotekama, umjesto da se koriste vanjske CSS datoteke.

moment, odnosno Moment.js je biblioteka za manipulaciju datumima i vremenima.

polished je biblioteka s pomoćnim funkcijama za rad s bojama i stilovima.

react-icons je biblioteka za uključivanje ikona u komponente.

react-toastify je biblioteka za prikazivanje povratnih obavijesti u aplikaciji.

Mongoose je biblioteka koja se koristi za rad s MongoDB bazom podataka na serverskoj strani, definiranje shema i komunikaciju s bazom podataka.

bcryptjs se koristi za sigurno hashiranje lozinki na serverskoj strani kako bi se osiguralo sigurno pohranjivanje lozinki u bazu podataka.

jsonwebtoken se koristi za generiranje i verifikaciju JSON web tokena (JWT) na serverskoj strani za autentifikaciju i autorizaciju korisnika.

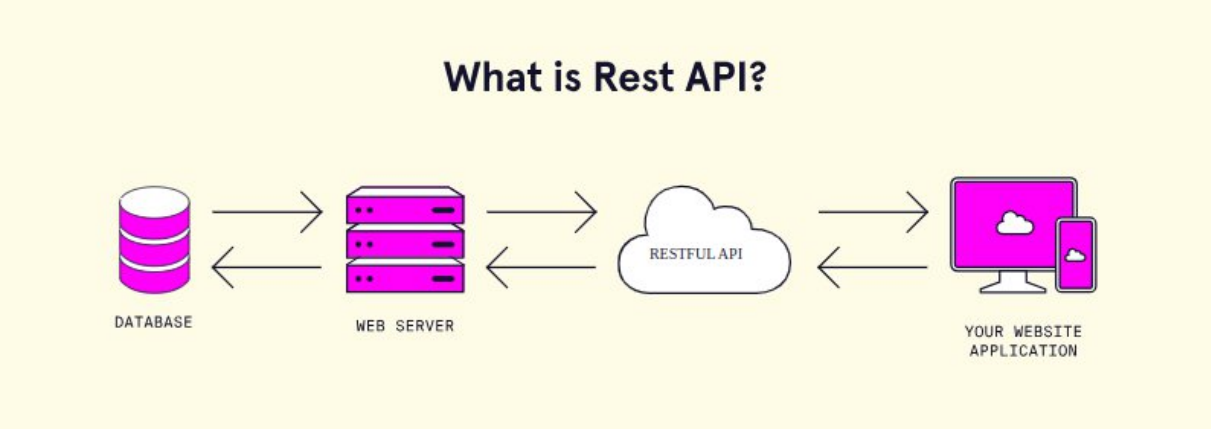
5. IMPLEMENTACIJA

Sve ove tehnologije i alati igraju ključnu ulogu u izradi stabilne i sigurne web aplikacije. Poslužiteljska strana (engl. backend) i klijentska strana (engl. frontend) zajedno čine cjelokupnu aplikaciju, a komuniciraju putem HTTP protokola. Backend obrađuje zahtjeve, pristupa bazi podataka, obrađuje poslovnu logiku i šalje podatke frontendu. Ovo je tipična arhitektura modernih web aplikacija, odnosno klijent-server arhitektura.

5.1. Arhitekturni stil REST

Živimo u svijetu u kojem se svake sekunde dijeli ogromna količina podataka putem preglednika, poslužitelja, softvera i aplikacija. Kako bi svi ti sustavi međusobno komunicirali u tome ključnu ulogu igraju API-ji. API je skraćenica od engl. „Application Programming Interface“ što znači da predstavlja sučelje programskog dijela aplikacije. Podaci koji se čuvaju u bazi podataka smješteni su na serveru. Kako bi pristupili tim podacima moramo na neki način zatražiti ono što želimo, a to radi API. API prima zahtjeve i gotovo trenutačno odgovara na njih. Postoje različiti načini kako se API-ji mogu oblikovati i organizirati. Međutim, trenutno je najvažniji i najrašireniji arhitekturni stil REST (skraćenica koja označava „Representational State Transfer“) kojeg je predstavio Roy Fielding u svojoj doktorskoj disertaciji 2000. godine.

Arhitekturni stil daje skup ograničenja, a API koji se temelji na tim ograničenjima može se nazvati RESTful. Na sljedećoj slici broj 4 je prikazan RESTful API:



Slika 4. Prikaz arhitekturnog stila „RESTful“ (izvor: [12])

5.1.1. Ograničenja

Postoji 5 važnih ograničenja, a to su [2]:

Klijent – poslužitelj: Klijent i poslužitelj su odvojeni entiteti koji obavljaju različite uloge. Klijent je onaj koji šalje zahtjev za određenom informacijom ili uslugom, dok je poslužitelj odgovoran za obradu tog zahtjeva i vraćanje odgovora s traženim informacijama. Važno je napomenuti da klijent i poslužitelj rade neovisno jedan o drugome. Klijent ne mora znati detalje o tome kako poslužitelj obradi zahtjev, niti poslužitelj treba znati kako je klijent generirao taj zahtjev. Ovo razdvajanje omogućava veću fleksibilnost u razvoju i održavanju sustava, jer se mogu mijenjati ili nadograđivati neovisno jedan o drugome.

Server ne smije pamtiti interakciju (engl. stateless server): Ovaj princip naglašava da sve informacije koje su potrebne za obavljanje zahtjeva moraju biti poslane od strane klijenta. To znači da klijent treba pružiti sve relevantne podatke koji su potrebni poslužitelju kako bi ispravno obradio zahtjev. Drugi važan aspekt je da poslužitelj ne bi trebao pohranjivati podatke o klijentu između pojedinih zahtjeva. Svaki zahtjev se tretira kao zaseban događaj i poslužitelj ne čuva nikakvo „stanje” između zahtjeva. To znači da svaki zahtjev mora sadržavati sve informacije koje su mu potrebne za obradu, jer poslužitelj nema prethodno znanje o klijentu izvan samog zahtjeva koji je primio.

Privremeno skladištenje sadržaja (engl. cache): Svaki put kad korisnik (klijent) pristupa određenim podacima ili resursima na webu, ti podaci se obično moraju dohvatiti s poslužitelja. Međutim, ako se ti podaci često koriste, bilo bi neučinkovito uvijek ih iznova dohvaćati s udaljenog poslužitelja, pa se radi toga koristi predmemorija (engl. cache) koja pamti podatke koji se često koriste. Kada korisnik ponovno zatraži iste podatke, umjesto da idu na udaljeni poslužitelj, podaci se preuzimaju iz predmemorije, što je puno brže. To znači da klijent može dobiti podatke brže i učinkovitije jer ne mora čekati da se podaci preuzmu s udaljenog poslužitelja. Također, to može smanjiti opterećenje na poslužitelju jer se često korišteni podaci čuvaju lokalno. Poslužitelj mora označiti podatke koji se mogu pohraniti u predmemoriju i koliko dugo se oni smiju čuvati, kako bi se osigurala ažurnost informacija.

Podrška za slojevitu arhitekturu: Slojevita arhitektura označava sposobnost da između klijenta i poslužitelja postoje dodatne komponente i podsustavi koji mogu posredovati u komunikaciji. Kao što je već navedeno, u arhitekturi „RESTful” klijent ne

mora znati sve pojedinosti o poslužitelju s kojim komunicira. Umjesto toga, klijent šalje zahtjev, a taj zahtjev može putovati kroz različite komponente prije nego što dođe do krajnjeg poslužitelja.

Uniformno sučelje označava na koji način će klijent i poslužitelj dijeliti informacije definiranjem sučelja koje se mora slijediti u svakom zahtjevu. REST se temelji na resursima, a resursi predstavljaju informacije koje mogu imati ime (npr. web stranica, podaci, slike). Kada klijent želi pristupiti određenom resursu na poslužitelju, mora specificirati koji resurs želi putem URL-a (engl. Uniform Resource Locator). Primjerice, URL http://example/api/v1/events označava resurs za popis događaja. S druge strane, klijent mora osigurati da svaki zahtjev poslan poslužitelju sadrži dovoljno informacija za manipulaciju odabranim resursom. Ovi resursi mogu biti prikazani u različitim formatima poput JSON-a, XML-a ili HTML-a. Na kraju, poruke koje klijent šalje poslužitelju i odgovori koje poslužitelj šalje klijentu trebaju sadržavati dovoljno informacija da ih obje strane mogu razumjeti samo na temelju samih poruka te na taj način osigurati jasno i jednostavno komuniciranje između klijenta i poslužitelja.

5.1.2. Anatomija zahtjeva

Svaki zahtjev predstavlja ključnu interakciju između klijenta i poslužitelja te uključuje specifične elemente koji omogućuju precizno identificiranje resursa i prijenos potrebnih podataka za obradu zahtjeva.

URL (engl. Uniform Resource Locator) je adresa koja se koristi za identifikaciju i pristupanje određenom resursu na internetu. To uključuje ne samo lokaciju resursa, već i specifikaciju kako pristupiti tom resursu. U API-ju, URL često predstavlja temeljnu adresu koja se koristi u svakom zahtjevu, na primjer: http://api.example.com.

URI (engl. Uniform Resource Identifier) je opći pojam koji obuhvaća sve vrste identifikatora resursa što uključuje i URL-ove. URI se koristi u URL-u kako bi se preciziralo kojem resursu klijent želi pristupiti u zahtjevu. U primjeru http://api.example.com/events, URI je „/events“.

Tijelo zahtjeva (engl. Body) je dio zahtjeva koji sadrži sve podatke potrebne poslužitelju kako bi uspješno obradio zahtjev. To se obično koristi u zahtjevima koji moraju poslati podatke, kao što su zahtjevi za stvaranje novih podataka ili ažuriranje postojećih.

Primjer tijela zahtjeva u JSON formatu:

```
{
    "eventType": "godisnji",
    "eventDate": "2023-08-07"
}
```

Parametri rute su parametri koji se umeću u URL s informacijom za identifikaciju određenog resursa kako bi se poduzela određena radnja, kao što su: dohvaćanje, uređivanje, ažuriranje ili brisanje. U primjeru http://api.example.com/events/123 parametar rute s vrijednošću 123 identificira resurs koji će biti manipuliran u zahtjevu.

Upiti (engl. Query Params) su parametri koji omogućuju dodavanje dodatnih informacija u URL, s ključnom razlikom da se koriste za filtriranje, pretraživanje, paginaciju ili sortiranje rezultata vezanih uz resurs.

U primjeru http://api.example.com/events/reports?month=07&year=2023, parametri „mjesec=07“ i „godina=2023“ specificiraju tražene podatke, u ovom slučaju izvještaje o događajima za srpanj 2023. godine.

Na kraju, zaglavlja (engl. Headers) omogućuju slanje dodatnih informacija u zahtjevu, poput autentifikacijskih tokena.

5.1.3. Komunikacija s REST-om

Komunikacija s REST-om uključuje korištenje HTTP metoda i interpretaciju HTTP statusnih kodova. Postoji pet metoda koje predstavljaju način na koji klijent može komunicirati s API-jem kako bi manipulirao resursima. Evo njihovih osnovnih značenja:

- GET: koristi se za dohvaćanje podataka s poslužitelja. Klijent specificira resurs koji želi dohvatiti u URL-u. Na primjer, ako želimo dohvatiti popis proizvoda s API-ja, koristimo GET metodu s URL-om http://api.example.com/events.
- POST: koristi se za stvaranje novog resursa na poslužitelju. Klijent specificira resurs u URL-u, a podatke o resursu šalje u tijelu zahtjeva. Primjerice, ako želimo dodati novi proizvod, koristimo POST metodu s URL-om http://api.example.com/events.

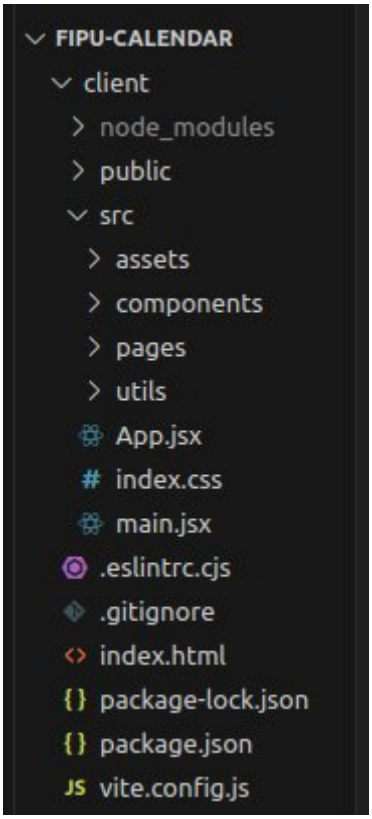
- PUT: koristi se za ažuriranje podataka resursa na poslužitelju. Klijent identificira resurs u URL-u i šalje nove informacije koje će zamijeniti postojeće podatke u tijelu zahtjeva. Na primjer, PUT zahtjev „PUT http://api.example.com/events/123“ znači da će se podaci o proizvodu s ID-jem 123 ažurirati s novim informacijama koje klijent šalje.
- PATCH: se također koristi za ažuriranje podataka resursa na poslužitelju, ali s razlikom da se ažurira samo određena informacija. Klijent opet identificira resurs u URL-u, ali u tijelu zahtjeva šalje samo one podatke koji se trebaju promijeniti. Primjerice, „PATCH http://api.example.com/events/123“ znači da će se samo svojstvo „eventType“ proizvoda s ID-om 123 ažurirati, a ostali podaci će ostati nepromijenjeni.
- DELETE: koristi se za brisanje resursa na poslužitelju. Klijent identificira resurs u URL-u i šalje zahtjev za brisanje. U primjeru „DELETE http://api.example.com/events/123“ to znači da će se proizvod s ID-jem 123 izbrisati.

Svaki put kad klijent šalje zahtjev poslužitelju, poslužitelj mu vraća odgovor koji uključuje HTTP statusni kod. Ti statusni kodovi pomažu klijentu u razumijevanju odgovora. Postoji pet glavnih skupina statusnih kodova:

- Grupa 2 – uspješan zahtjev: Ovi statusni kodovi označavaju da je zahtjev klijenta uspješno primljen i da s obradom nije bilo problema. Na primjer, statusni kod 200 znači „OK“ i označava da je zahtjev uspješno izvršen.
- Grupa 3 – preusmjeravanje: Ova grupa označava da je poslužitelj obavijestio klijenta o potrebi za preusmjeravanjem na drugu lokaciju ili URL. To može biti korisno u situacijama gdje je prvobitna adresa promijenjena.
- Grupa 4 – greške na strani klijenta: ovi statusni kodovi označavaju da je klijent poslao zahtjev koji nije ispravno sastavljen što može uključivati sintaksne pogreške ili pokušaje pristupa resursima na koje klijent nema dopuštenje.
- Grupa 5 – greške na strani poslužitelja: ova grupa označava da je zahtjev poslan ispravno od strane klijenta, ali se dogodila greška tijekom obrade na strani poslužitelja što može biti uzrokovano tehničkim problemima na strani poslužitelja.

5.2. Implementacija klijentske strane

Na slici broj 4 je prikazan *client* direktorij koji predstavlja glavno razvojno okruženje za klijentski dio aplikacije. U njemu se nalaze sve datoteke i direktoriji koji su potrebni za izradu React aplikacije. Sadrži komponente, konfiguracijske datoteke te sve što je potrebno za razvoj i izgradnju korisničkog sučelja.



Slika 5. Struktura klijentske strane (izvor. autor)

Osim *node_modules* direktorija u kojem su pohranjeni svi vanjski paketi i biblioteke te *public* direktorija u kojem su smještene statičke datoteke poput slika, *favicon* ikone i HTML datoteka koje se izravno dostavljaju pregledniku tu je i tzv. *src* direktorij, odnosno glavni direktorij u kojem se nalazi izvorni kod React aplikacije. Podjeljen je u nekoliko poddirektorija:

- *assets* u kojem su smještene slike i *styled-components* pojedinih komponenti.
- *components* u kojem se nalaze React komponente koje se koriste u različitim dijelovima aplikacije.

– *pages*: Ovaj direktorij sadrži više složenih komponenti koje tada predstavljaju cijele stranice ili rute unutar aplikacije.

– *utils* u kojem se nalaze pomoćne funkcije, konstante i ostali kod koji se koristi u različitim dijelovima aplikacije

5.2.1. Komponenta „App.jsx“

App.jsx je glavna React komponenta koja predstavlja korijen aplikacije. U njoj se nalazi *router* i osnovne komponente koje čine sam izgled aplikacije. Verzija 6.4 react-router-dom modula uvodi niz novih API-ja za upravljanje podacima, kao što su *loader*, *action* i drugi slični koncepti. Ovi API-ji olakšavaju dohvaćanje i manipulaciju podacima unutar aplikacije te omogućuju različite CRUD (*Create*, *Read*, *Update*, *Delete*) operacije kao što su dohvaćanje podataka s poslužitelja, izvođenje akcija kao što su dodavanje ili brisanje podataka i druge slične zadatke.

Primjerice *loaders* bi bile funkcije ili komponente koje se koriste za dohvaćanje podataka s poslužitelja i pripremu tih podataka za prikaz u aplikaciji, na primjer prikaz događaja u kalendaru. *Actions* bi bile funkcije koje se izvode kada korisnik izvrši određenu radnju, poput dodavanja događaja u kalendar ili editiranja korisničkog profila.

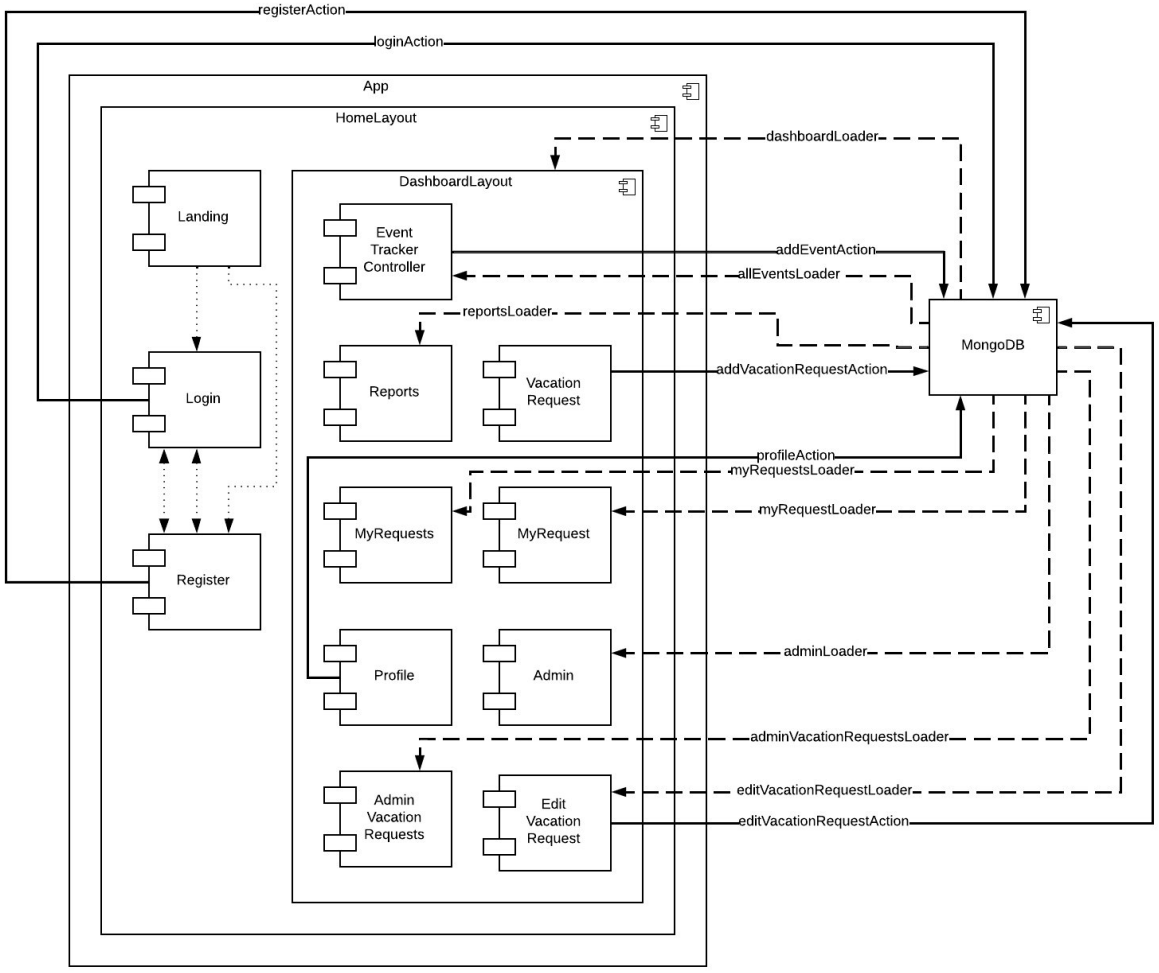
U suštini, ovi novi API-ji omogućuju efikasnije upravljanje podacima i olakšavaju implementaciju različitih funkcionalnosti u aplikaciji.

Na sljedećoj slici broj 5 možemo vidjeti App.jsx komponentu ove web aplikacije. `<RouteProvider>` prima objekt `router` koji prima konfiguraciju rute.

```
1  const App = () => {
2    return <RouterProvider router={router} />;
3  };
4  export default App;
```

Slika 6. Prikaz glavne komponente aplikacije „App“ (izvor: autor)

Taj objekt, konfiguriran preko `createBrowserRouter` funkcije, sadrži informacije o tome kako se rute u aplikaciji trebaju ponašati. Na sljedećoj slici broj 7 možemo vidjeti strukturu routera ove web aplikacije prikazanog komponentnim dijagramom.



Slika 7. Komponentni dijagram strukture App.jsx routera (izvor: autor)

Ruta je način na koji se upravlja prikazom različitih dijelova web aplikacije ovisno o URL-u koji korisnik unese u pregledniku. Dakle, ruta može uključivati putanju (engl. Path), parametre i druge opcionalne dijelove koji pomažu u preciznom određivanju što će se prikazati na ekranu kada korisnik unese određeni URL:

`path`: Ovo je svojstvo koje definira URL putanju koja će se koristiti za usmjeravanje na određenu komponentu. Na primjer, `path: "/"` znači da će se ruta podudarati s početnom URL putanjom.

`element`: Ovo svojstvo definira React komponentu koja će se prikazati kada se ruta podudara s URL putanjom. Na primjer, element: `<HomeLayout />` znači da će se `<HomeLayout />` komponenta prikazati kada se ruta podudarila s početnom URL putanjom.

`errorElement`: Ovo je opcionalno svojstvo koje definira komponentu koja će se prikazati ako se dogodi greška pri učitavanju rute. Na primjer, `errorElement: <Error />` znači da će se `<Error />` komponenta prikazati ako putanja rute ne postoji.

`children`: Ovdje se definiraju potomci (children) rute. To su rute koje su „unutar" glavne rute. Na primjer, unutar rute s početnom URL putanjom `"/"` definirane su druge rute kao `"register"`, `"login"` i `"dashboard"`.

`index: true`: Ovo svojstvo označava da je ruta primarna ili zadana ruta za roditeljsku rutu. Na primjer, ako korisnik posjeti `"/dashboard"`, tada će se `<EventTrackerController />` prikazati kao zadana komponenta za `"dashboard"` rutu.

`loader`: Ovo svojstvo definira funkciju koja se poziva kako bi se dohvatili podaci potrebni za prikaz određene rute.

`action`: Ovo svojstvo definira akciju koja će se izvršiti prilikom submita na određenoj ruti.

Ne moraju sve rute imati komponente, kao što je to u slučaju brisanja događaja iz kalendara. Ta ruta ima samo definiranu akciju budući da je proces brisanja ograničen na jednostavno klikanje na gumb, bez prikaza zasebne komponente.

5.2.2. Komponenta „Form"

Novost u react-router-dom v6 modulu je i `<Form>` komponenta, koja za razliku od običnog HTML `<form>` elementa omogućava bolju integraciju s rutiranjem na klijentskoj strani. Ova komponenta djeluje kao omotač oko standardnog HTML obrasca.

```jsx
<Form method="post" className="form">
          ...
        </Form>
```

Slika 8. Komponenta „Form“ (izvor: autor)

„action“, odnosno akcija koja je prikazana na slici broj 9 najprije dohvaća podatke iz forme koristeći `{ request }` objekt te stvara novi objekt iz `formData` podataka te ih putem API-ja sprema u bazu podataka.

```jsx
export const action = async ({ request }) => {
  const formData = await request.formData();
  const data = Object.fromEntries(formData);
  try {
    const response = await customFetch.post("/vacationRequest", data);
    return redirect("/dashboard");
  } catch (error) {
    toast.error(error?.response?.data?.msg);
    return redirect("/dashboard");
  }
};
```

Slika 9. Prikaz funkcije „action“ za dohvat podataka iz komponente Form (izvor: autor)

Ako se pojavi greška tijekom izvršavanja zahtjeva (na primjer ako API vrati grešku, što je definirano u backend dijelu aplikacije) tada se prikazuje obavijest pomoću react-toastify paketa te se korisnika preusmjerava na ("`/dashboard`") stranicu. U slučaju da greške nema, podatke šaljemo na API i također se korisnika preusmjerava na početnu ("`/dashboard`") stranicu.

Na sljedećoj slici broj 10 možemo vidjeti prikaz funkcije „loader“ koji za razliku od funkcije „action“ prima parametar „id“ dohvaća podatke sa poslužitelja, u ovom slučaju određeni zahtjev za godišnji odmor.

```
1  export const loader = async ({ params }) => {
2    try {
3      const { data } = await customFetch.get(
4        `/users/admin/edit-vacation-request/${params.id}`
5      );
6      return data;
7    } catch (error) {
8      toast.error(error?.response?.data?.msg);
9      return redirect("/dashboard/vacation-requests");
10   }
11 };
12
13 const EditVacationRequest = () => {
14   const { vacationRequest } = useLoaderData();
15   console.log(vacationRequest);
16   ...
17 };
18 export default EditVacationRequest;
```

Slika 10. Prikaz funkcije „loader“ za dohvat podataka sa poslužitelja (izvor: autor)

5.2.3. *Komponenta „Calendar“*

Komponente u Reactu su temeljni građevni blokovi svake aplikacije. One omogućuju programerima da izgrade aplikaciju sastavljajući manje, autonomne komponente. Ovaj modularni pristup čini komponente ponovno upotrebljivim i pruža niz prednosti koje pomažu u olakšavanju razvojnog procesa. One mogu biti jednostavne poput komponenti za input polja ili buttone, no međutim mogu biti i složenije. Jedna od takvih složenijih komponenti u ovoj aplikaciji je i `<Calendar>` prikazana na slici broj 11. Naime, ova komponenta se koristi u dva dijela u aplikaciji, jednom na početnoj stranici gdje profesor evidentira svoje radne dane te drugi puta na stranici gdje označava dane u mjesecu kako bi mogao poslati zahtjev za godišnji odmor. Sve funkcionalnosti kalendara su iste, jedino je razlika u dizajnu.

```
<Calendar
  getCellProps={(dayMoment) => {
    const eventsForDay = events.filter((event) => {
      return moment(event.eventDate).isSame(dayMoment, "day");
    });
    return {
      isToday: dayMoment.isSame(today, "day"),
      events: eventsForDay,
    };
  }}
  onCellClicked={(date, month, year) => {
    const clickedMoment = moment(
      `${date}${month}${year}`,
      "DDMMYYYY"
    ).toDate();
    const isEventDate = events.some((event) =>
      moment(event.eventDate, "YYYY-MM-DD").isSame(clickedMoment, "day")
    );
    if (!isEventDate) {
      displayModal(date, month, year);
    }
  }}
  month={currentMonthMoment.format("MM")}
  year={currentMonthMoment.format("YYYY")}
  onPrev={decrementMonth}
  onNext={incrementMonth}
  setToday={setToday}
  cellComponent={EventCell}
/>
```

Slika 11. Komponenta „Calendar“ (izvor: autor)

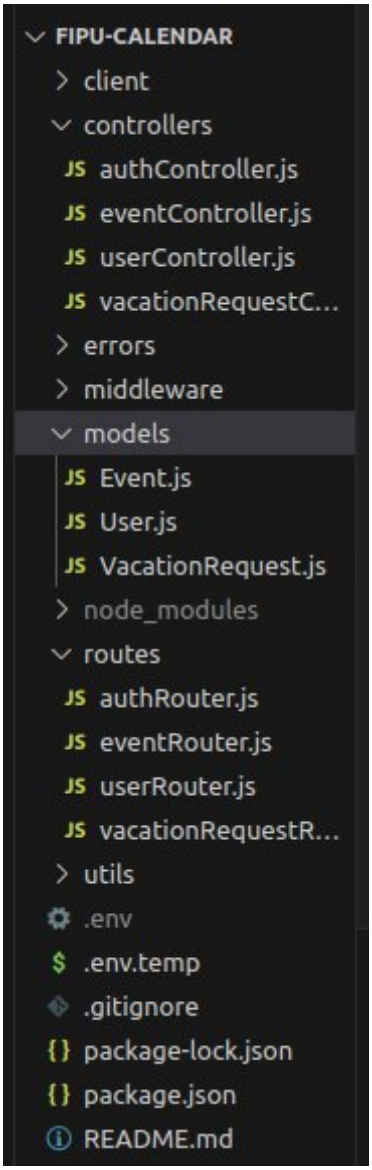
Ovaj kod omogućuje interaktivno prikazivanje kalendara s mogućnošću prikazivanja događaja (events) i odabira datuma. Komponente mogu imati svoja svojstva, tzv. „propse” (props, kratica za properties). To je način na koji komponente komuniciraju jedna s drugom. Neizmjenjivi su što znači da komponenta koja prima propse ih ne može mijenjati. Oni mogu biti String, boolean, funkcije pa čak i drugi .jsx element.

`getCellProps` je funkcija pomoću koje svaka ćelija kalendara pravilno prikazuje događaj za određeni dan te označava trenutni dan dok `onCellClicked` kontrolira što će se dogoditi kada korisnik klikne na određenu ćeliju u kalendaru. Primjerice, props naziva `cellComponent` je .jsx komponenta `<EventCell>` koja predstavlja prikaz evenata u kalendaru na početnoj stranici, ali neće biti prikazana prilikom prikaza kalendara za godišnji odmor.

Ponovna upotreba komponenata smanjuje ponavljanje koda i olakšava održavanje aplikacije. Ako je potrebno promijeniti ponašanje ili izgled dijela aplikacije, to se može učiniti na jednom mjestu (u komponenti) umjesto da se isti kod mijenja na mnogim mjestima. To znači da React komponente čine razvoj aplikacija efikasnijim i održivijim. Mogu se koristiti za izgradnju različitih dijelova aplikacije, čineći ih prilagodljivima i ponovno upotrebljivima. Ova fleksibilnost i modularnost čine React popularnim alatom za izradu modernih web aplikacija koje zadovoljavaju visoke standarde brzine i efikasnosti.

5.3. Implementacija poslužiteljske strane

Na slici broj 12 možemo vidjeti strukturu poslužiteljske strane, odnosno backenda.



Slika 12. Struktura poslužiteljske strane (izvor: autor)

Iz ove strukture može se jasno uočiti organizacija aplikacije koja se temelji na tri ključna modela, Event.js, User.js i VacationRequest.js, gdje svaki od njih predstavlja zasebnu kolekciju u MongoDB bazi podataka.

Osim toga, može se primijetiti kako su različiti dijelovi aplikacije organizirani u odvojenim direktorijima, čime se olakšava razvoj i održavanje aplikacije.

Na primjer direktorij „controllers“ sadrži datoteke koje sadrže kontrolere za pojedine modele u aplikaciji.

„authController“ sadrži funkcije za autentifikaciju korisnika, dok „eventController“ i „vacationRequestController“ sadrže funkcije koje omogućuje aplikaciji izvođenje različitih CRUD operacija kao što su kreiranje, čitanje, ažuriranje i brisanje, kao i generiranje izvještaja u odabranom vremenskom okviru.

„userController“ obuhvaća različite funkcionalnosti koje se odnose na administrativni dio aplikacije, odnosno na dio aplikacije kojeg koristi tajnica. To je skup funkcija koje omogućuju dohvat podataka o korisnicima, odnosno profesorima i njihovim aktivnostima, kao što su evidencija radnih dana u određenom mjesecu te pregled zahtjeva za godišnji odmor.

Još jedan direktorij kojeg možemo vidjeti u strukturi backenda je i „routes“. Naime, sve rute su strukturirane kako bi se osigurala čitljivost i organizacija koda. Ovaj pristup omogućava jasno identificirati odgovarajuće rute za različite funkcionalnosti, što olakšava održavanje i proširenje aplikacije. Svaka ruta ima definirane HTTP metode (npr. GET, POST, PATCH, DELETE) koje određuju kako se aplikacija ponaša kada primi određeni zahtjev na toj ruti. Ove rute su sastavni dio organizacije backenda aplikacije i omogućuju interakciju s bazom podataka i manipulaciju podacima. Na slici broj 13 možemo vidjeti definirane rute za registraciju, prijavu i odjavu korisnika.

```
1  router.post('/register', validateRegisterInput, register)
2  router.post('/login', validateLoginInput, login)
3  router.get('/logout', logout)
```

Slika 13. Prikaz ruta za autentifikaciju korisnika (izvor: autor)

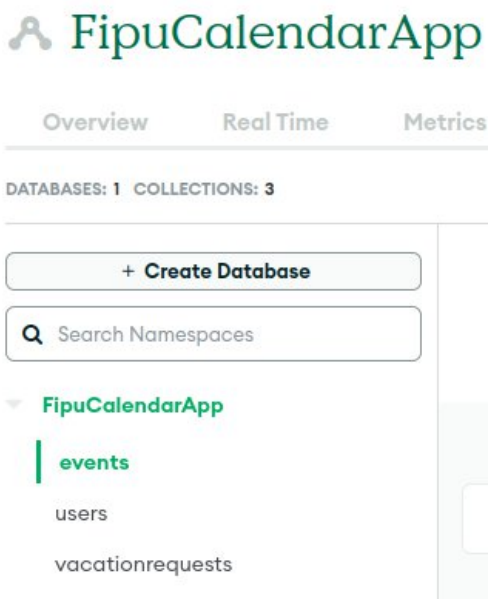
Nadalje, u strukturi imamo i „middleware“, odnosno funkcije koje omogućavaju provjeru pristupa i validaciju podataka prije nego što se zahtjev obradi, čime se osigurava sigurnost i konzistentnost podataka.

„Middleware“ može obavljati različite zadatke kao što su autentifikacija, obrada podataka, rukovanje pogreškama i sl. Također, može obavljati mnoge druge funkcije u okviru zahtjeva i odgovora.

Primjerice, middleware za autentifikaciju može provjeriti je li korisnik prijavljen prije nego što nastavi s obradom zahtjeva. Ako korisnik nije prijavljen, middleware može preusmjeriti korisnika na stranicu za prijavu ili vratiti odgovor sa statusom 401 (neprijavljen korisnik). S druge strane, middleware za obradu pogrešaka se koristi za hvatanje i rukovanje pogreškama koje se mogu dogoditi tijekom obrade zahtjeva. On može oblikovati odgovor koji se šalje klijentu u slučaju pogreške, omogućujući programeru veću kontrolu nad načinom na koji se rukuje s greškama u aplikaciji.

5.3.1. Agregacijski pipeline

U razvoju svake web ili mobilne aplikacije, jedna od ključnih odluka je kako organizirati podatke u bazi podataka. Ova odluka ima značajan utjecaj na performanse, skalabilnost i fleksibilnost aplikacije. NoSQL baze podataka poput MongoDB-a omogućuju jednostavno spremanje različitih entiteta u odvojenim kolekcijama bez potrebe za strogim definiranjem sheme ili tablica. U ovoj aplikaciji korištene su zasebne kolekcije Users, Events i VacationRequests što je i prikazano na slici broj 14. Korisnici imaju osobne informacije poput imena, adrese i e-mail adrese, dok događaji (eventi) sadrže informacije kao što su datum i vrsta događaja.



Slika 14. Kolekcije u MongoDB bazi podataka (izvor: autor)

Zasebne kolekcije pružaju brži pristup podacima kad je neovisno potrebno čitati informacije o korisnicima ili događajima. Kako nema potrebe za složenim JOIN operacijama koje su često prisutne u relacijskim bazama podataka, brzina čitanja podataka iz zasebnih kolekcija značajno se povećava. To je od posebne važnosti u ovoj aplikaciji gdje se često dohvaćaju informacije o događajima i korisnicima. Međutim, postoje i nedostaci ovog pristupa. Kada je potrebno dohvatiti podatke iz više kolekcija, to može dovesti do složenih upita ili više mrežnih zahtjeva, što može utjecati na performanse u određenim slučajevima. Korištenje posebnih kolekcija može biti

prikladno za aplikacije koje imaju jasno definirane entitete (korisnici, događaji) koji su neovisni i ne zahtijevaju često složene JOIN operacije. U suprotnom, relacijski model s povezanim tablicama (one-to-many) može biti bolji izbor za organizaciju podataka.

Primjerice, jedan takav problem u ovoj aplikaciji je bio prikazati radne dane određenog korisnika za pojedini mjesec, a u tome nam uveliko pomaže tzv. „aggregation pipeline“.

Agregacijski pipeline u MongoDB bazi podataka predstavlja moćan mehanizam za obradu podataka i analizu unutar baze podataka. Omogućava nam da manipuliramo, filtriramo, grupiramo i transformiramo podatke prije nego što ih izvučemo ili prikažemo korisnicima. Ovaj koncept se često koristi za izradu složenih upita i generiranje raznih izvještaja iz ogromnih skupova podataka.

Na slici broj 15 je prikazan agregacijski pipeline kako obrađuje podatke o događajima (events) i generira izvještaje na temelju parametara koji se šalju putem upita.

```
const userReports = await User.aggregate([
  {
    $lookup: {
      from: "events",
      let: { userId: "$_id" },
      pipeline: [
        {
          $match: {
            $expr: {
              $and: [
                { $eq: ["$createdBy", "$$userId"] },
                {
                  $gte: [
                    "$eventDate",
                    moment({ year: selectedYear, month: selectedMonth })
                      .startOf("month")
                      .toDate(),
                  ],
                },
                {
                  $lt: [
                    "$eventDate",
                    moment({ year: selectedYear, month: selectedMonth })
                      .endOf("month")
                      .toDate(),
                  ],
                },
              ],
            },
          },
        },
      ],
      as: "userEvents",
    },
  },
  {
    $project: {
      name: 1,
      lastName: 1,
      email: 1,
      role: 1,
      userEvents: {
        $map: {
          input: "$userEvents",
          as: "event",
          in: {
            eventType: "$$event.eventType",
            eventDate: "$$event.eventDate",
            count: 1,
          },
        },
      },
    },
  },
  {
    $match: {
      role: { $ne: "admin" } // Filter out documents where the role is not "admin"
    }
  },
  {
    $sort: {
      lastName: 1,  // Sort by lastName in ascending order
      name: 1       // Sort by name in ascending order
    }
  }
]);
```

Slika 15. Primjer koda za agregacijski pipeline (izvor: autor)

Evo pregleda ključnih koraka u tom procesu:

`$lookup`: služi za spajanje dvije kolekcije (User i Event) na temelju zajedničkog polja (`_id` i `createdBy`) kako bi se dobili podaci o događajima koji su povezani s korisnicima.

`$match`: filtrira dokumente koji zadovoljavaju određeni uvjet. U ovom slučaju, filtrira se događaje koji se nalaze unutar odabranog mjeseca i godine.

`$project`: odabire koje atribute želimo zadržati u rezultirajućem dokumentu. Ovdje se zadržavaju `name`, `lastName`, `email`, `role` i transformira se polje `userEvents`.

`$map`: se koristi za mapiranje i transformaciju polja. U ovom slučaju mapira se polje `userEvents` kako bi se izvukle informacije o vrsti događaja i datumu.

`$match`: dodatna filtracija koja filtrira korisnike koji nisu admnistratori kako se nebi prikazivali u izvješću.

`$sort`: sortira rezultate prema prezimenima i imenima.

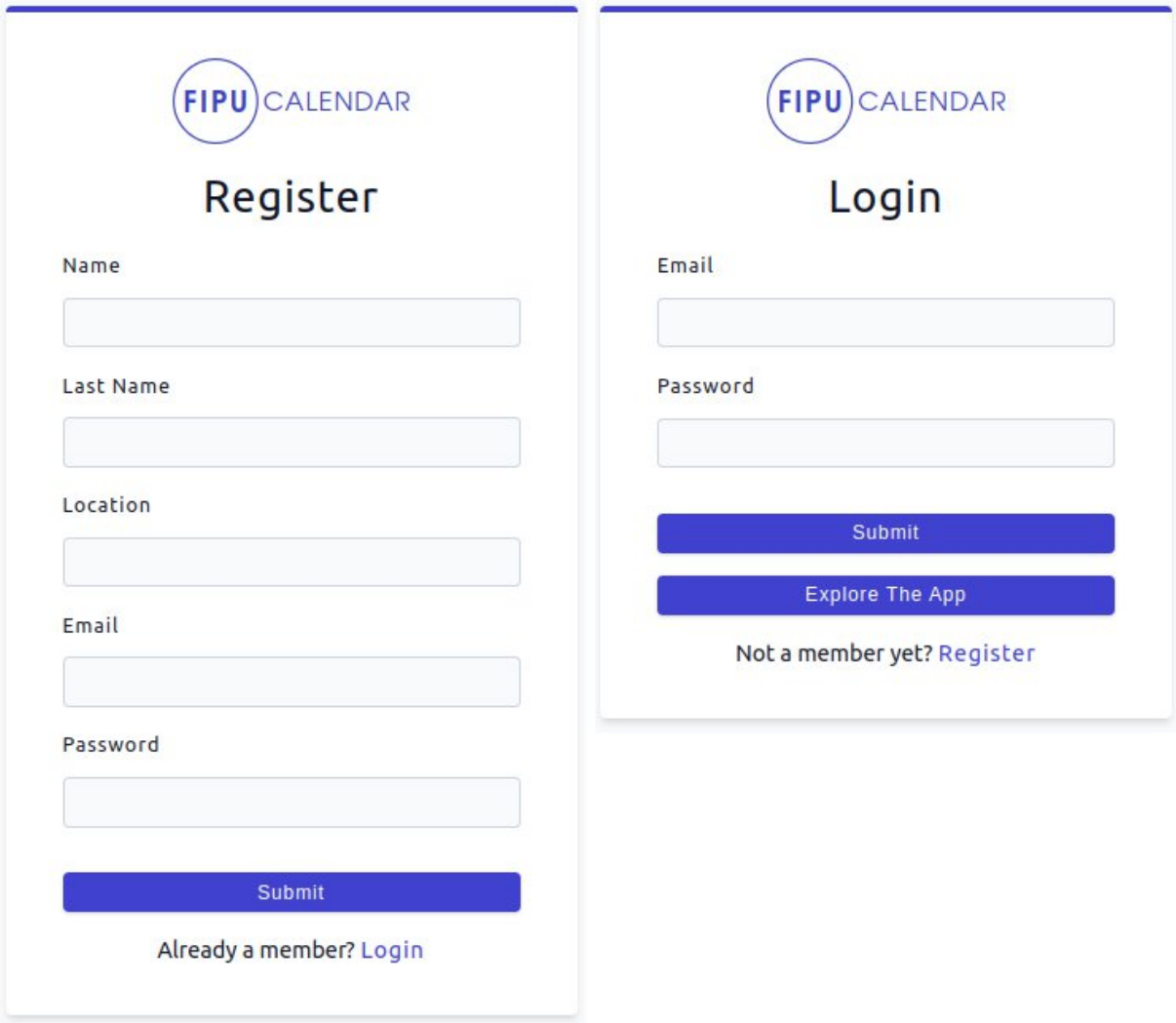
Na kraju, ovaj MongoDB pristup često olakšava rad s nestrukturiranim podacima poput dokumenata u NoSQL bazama.

6. KORISNIČKE UPUTE

Kod otvaranja aplikacije, prvo što je vidljivo je početna stranica s istaknutim logom (prikazana na slici broj 16). Na ovom zaslonu, korisnik, u našem slučaju profesor ili tajnica, imaju opciju izbora između prijave i registracije. Registracija je obavezna kako bi se omogućilo korištenje aplikacije, jer se informacije o profesoru i njegovim radnim danima pohranjuju u bazu podataka. Ova funkcionalnost omogućava korisniku da očuva svoje podatke i prilagodi ih pri prijavi s drugog uređaja. Važno je naglasiti da je ova aplikacija razvijena s posebnom pažnjom prema mobilnom korisničkom iskustvu, koristeći tzv. „mobile first" pristup odnosno pristup koji prioritet daje mobilnom sučelju, kako bi korisnici imali preglednost na svim uređajima.

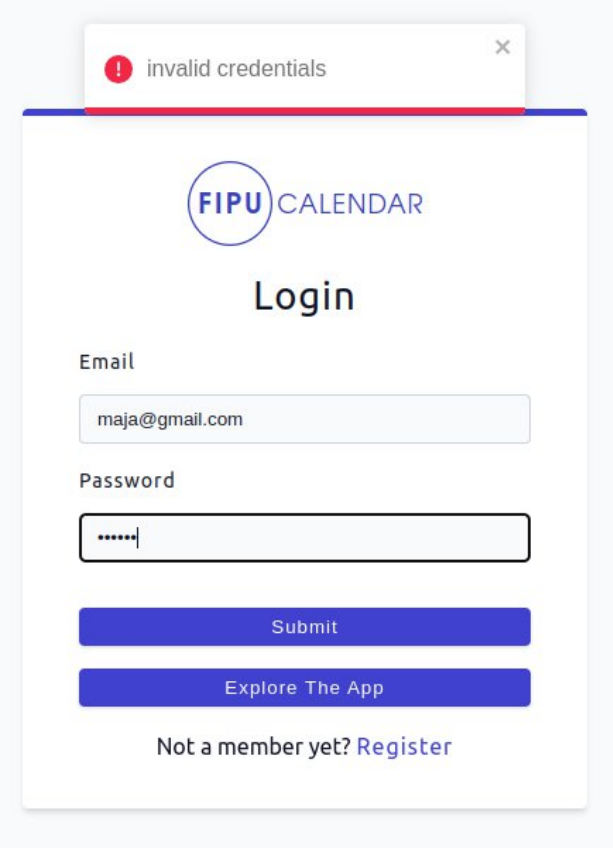
Slika 16. Početna stranica aplikacije (izvor: autor)

Neregistrirani korisnik najprije mora izvršiti registraciju kako bi mogao pristupiti formi za prijavu. U toj „login” formi korisnik mora unijeti svoju e-mail adresu i lozinku kako bi se prijavio. Nakon što je prijava uspješna, korisnik će dobiti puni pristup aplikaciji.



Slika 17. Prikaz stranica za registraciju ili prijavu korisnika (izvor: autor)

U slučaju krivog unosa dobijemo povratnu poruku o netočnom unosu podataka.

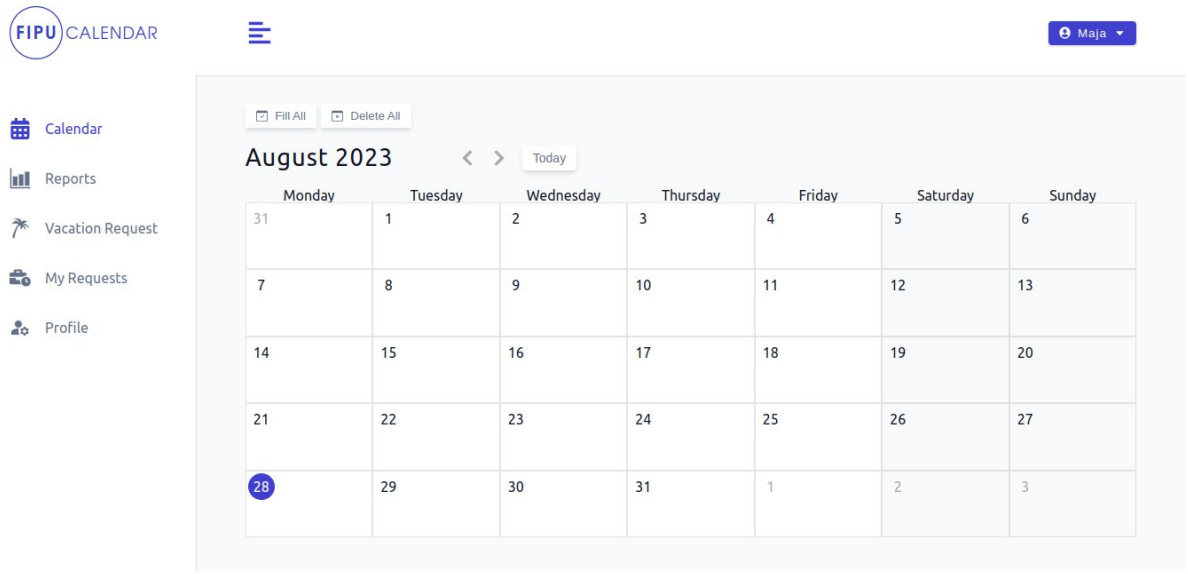


Slika 18. Neuspješna prijava (izvor: autor)

6.1. Korisničke upute – profesor

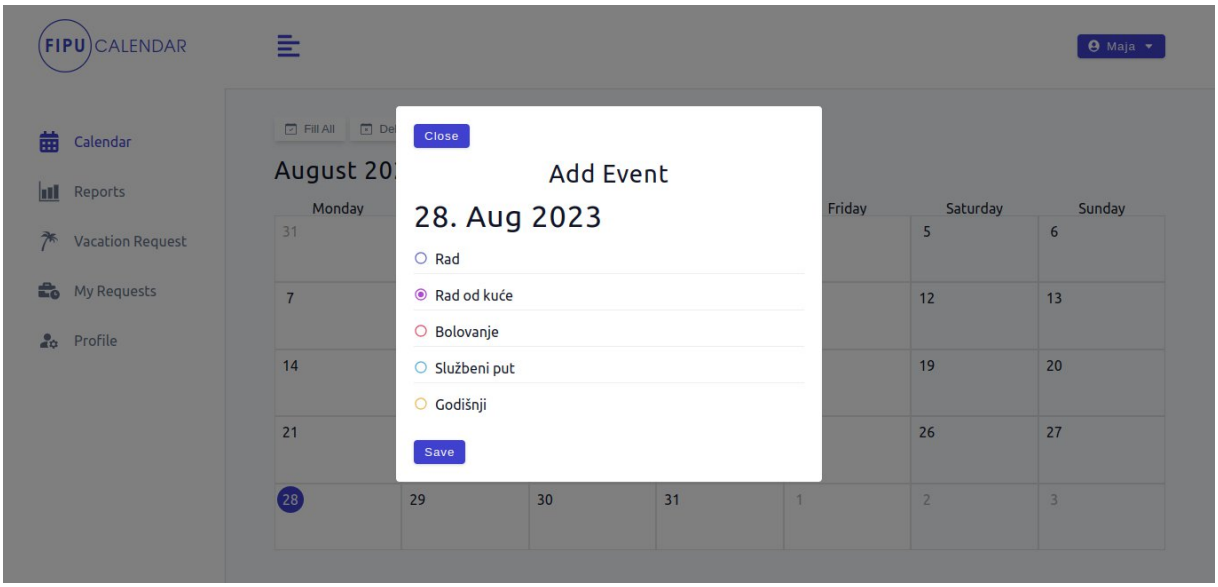
Nakon uspješne registracije i prijave, aplikacija će nas preusmjeriti na početnu stranicu profesora, odnosno kalendar gdje profesor može evidentirati svoje radne dane u tekućem mjesecu što je vidljivo na slici broj 19.

S lijeve strane se nalazi navigacijski izbornik koji pruža pristup različitim stranicama unutar aplikacije: „Reports“, „VacationRequest“, „My Requests“ i „Profile“. Kalendar, s druge strane, prikazuje trenutni mjesec i označeni današnji datum unutar mjeseca. Tu se također nalazi navigacijska traka koja omogućava prelazak na prethodni ili sljedeći mjesec, te gumb „Today” koji nas vraća na trenutni mjesec. Važno je napomenuti da se na dane u prethodnom mjesecu ne možemo kliknuti, kao niti na subote i nedjelje budući da su ti dani označeni kao neradni.



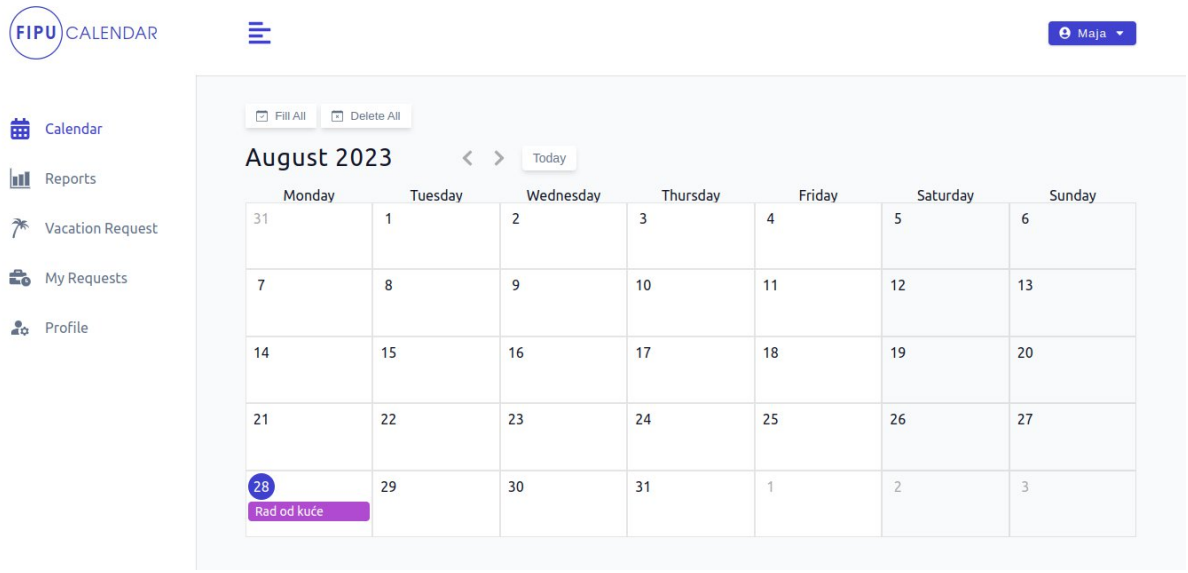
Slika 19. Početna stranica profesora (izvor: autor)

Kada kliknemo na bilo koji datum u mjesecu, otvara nam se tzv. Modal odnosno prozor u kojem možemo odabrati jedan od evenata za odabrani datum, a to su, rad, rad od kuće, bolovanje, službeni put ili godišnji. Svaki od tih evenata, je označen drugom bojom koja se tada provlači kroz cijelu aplikaciju. Na slici broj 20 možemo vidjeti kako je odabran event „rad od kuće“ te nakon šta kliknemo „Save“ taj je događaj kreiran.



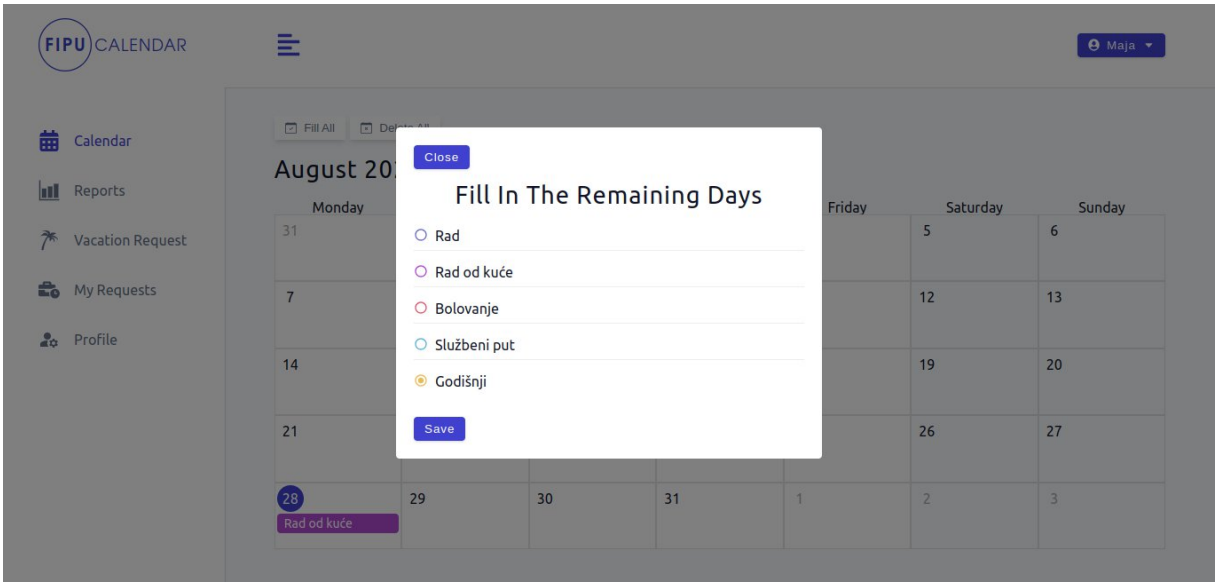
Slika 20. Kreiranje događaja „Rad od kuće“ u kalendar (izvor: autor)

Na sljedećoj slici broj 21 možemo vidjeti kreirani događaj označen bojom koja predstavlja odabrani tip događaja.



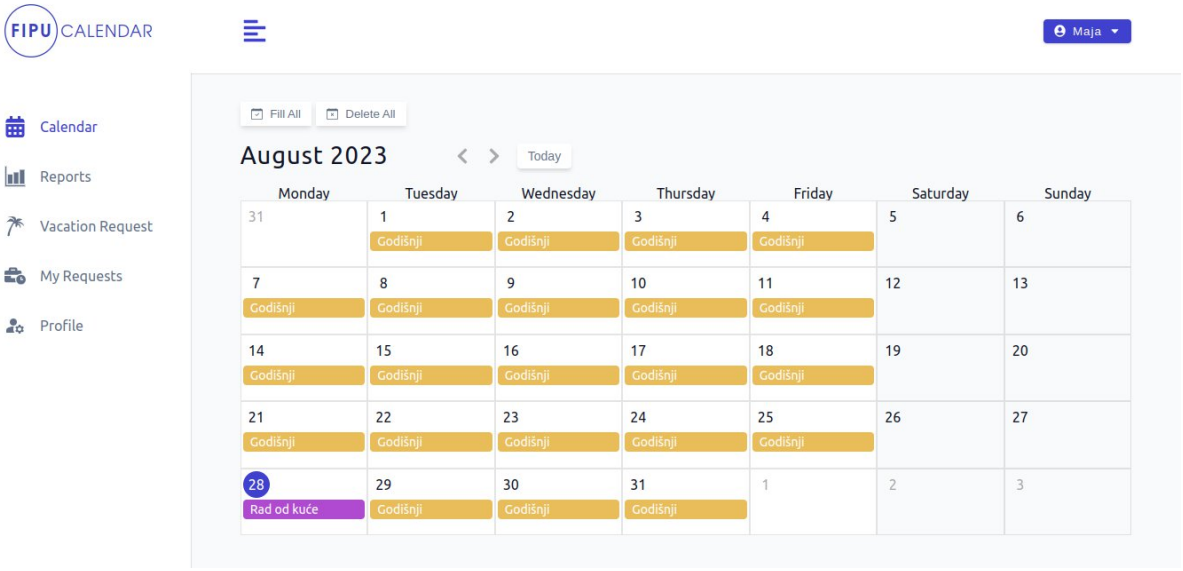
Slika 21. Prikaz kreiranog događaja „Rad od kuće“ u kalendaru (izvor: autor)

Još jedna funkcionalnost kalendara je kada kliknemo na gumb „Fill All“ također nam se otvara prozor gdje možemo odabrati događaj te popuniti sve preostale dane u mjesecu, osim subote i nedjelje koje su označene kao neradni dani. Na sljedećoj slici broj 22 možemo vidjeti tu funkcionalnost.



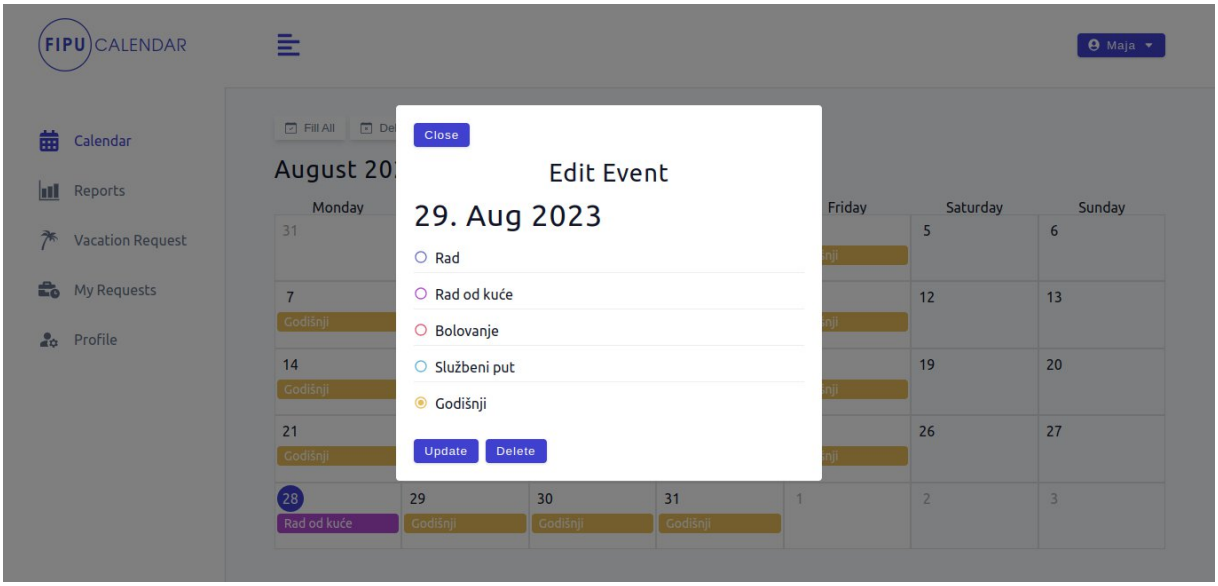
Slika 22. Popunjavanje preostalih dana u mjesecu (izvor: autor)

Nakon šta smo odabrali „Godišnji“ kao odabrani događaj za cijeli mjesec kliknemo na „Save“, prozor se zatvori te se popune preostali dani u mjesecu. Na slici broj 23 možemo vidjeti kako su subote i nedjelje ostale prazne te kako je prethodno kreirani događaj 28.08.2023. ostao nepromijenjen.



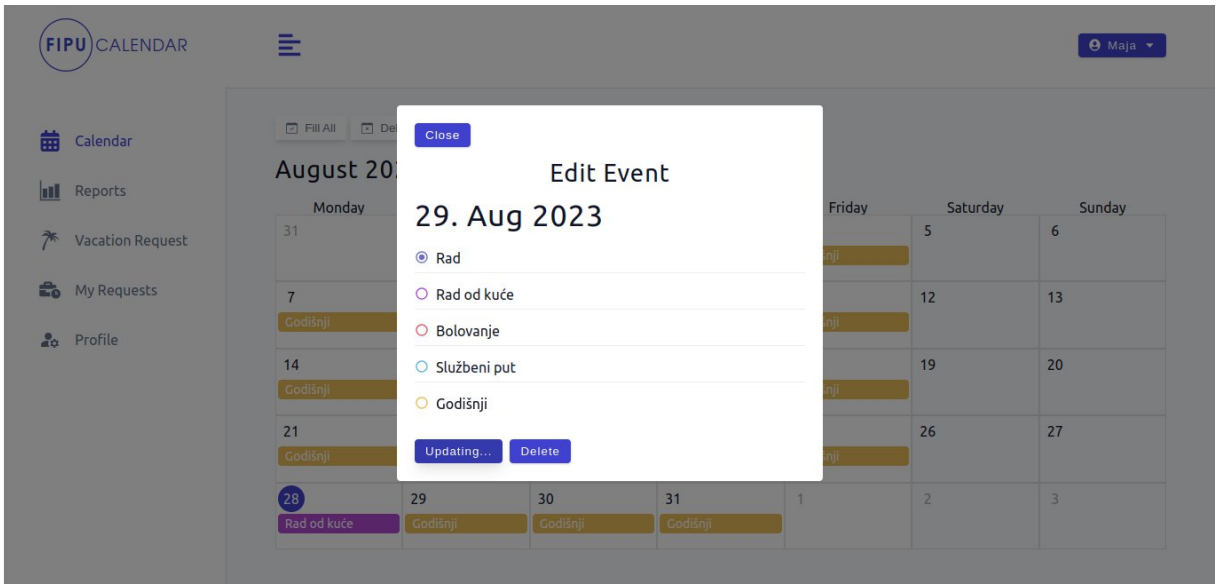
Slika 23. Prikaz kreiranih događaja za sve dane u mjesecu (izvor: autor)

Također, na one datume u mjesecu za koje postoji odabrani događaj nije više moguće kliknuti, odnosno moguće je kliknuti samo na određeni događaj kojeg želimo mijenjati, što je prikazano na slici broj 24. Ovdje je vidljivo kako smo kliknuli na event 29.08.2023. koji je trenutno označen kao godišnji.



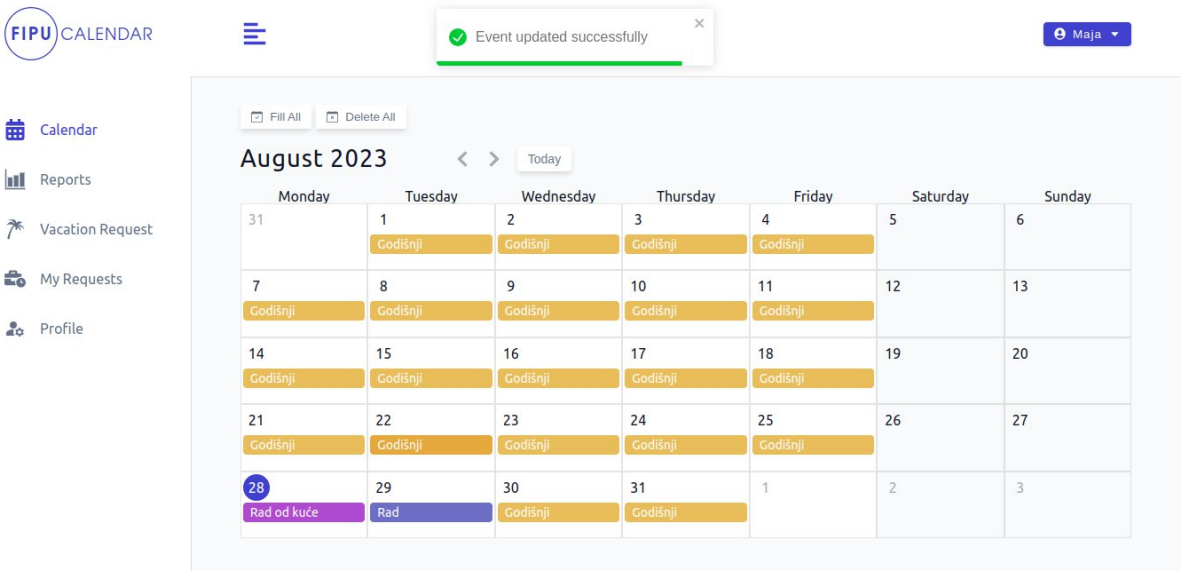
Slika 24. Ažuriranje događaja (izvor: autor)

Na slici broj 25 možemo vidjeti kako smo umjesto eventa „Godišnji” odabrali event „Rad”. Također, treba primijetiti kako je na gumbu prikazano „Updating...”, što znači da je gumb onemogućen tj. nije moguće kliknuti na njega. Svrha ovog pristupa je spriječiti korisnike da kliknu „Update” više puta dok je proces još u tijeku. To pomaže osigurati da se proces slanja ne prekida ili duplicira.



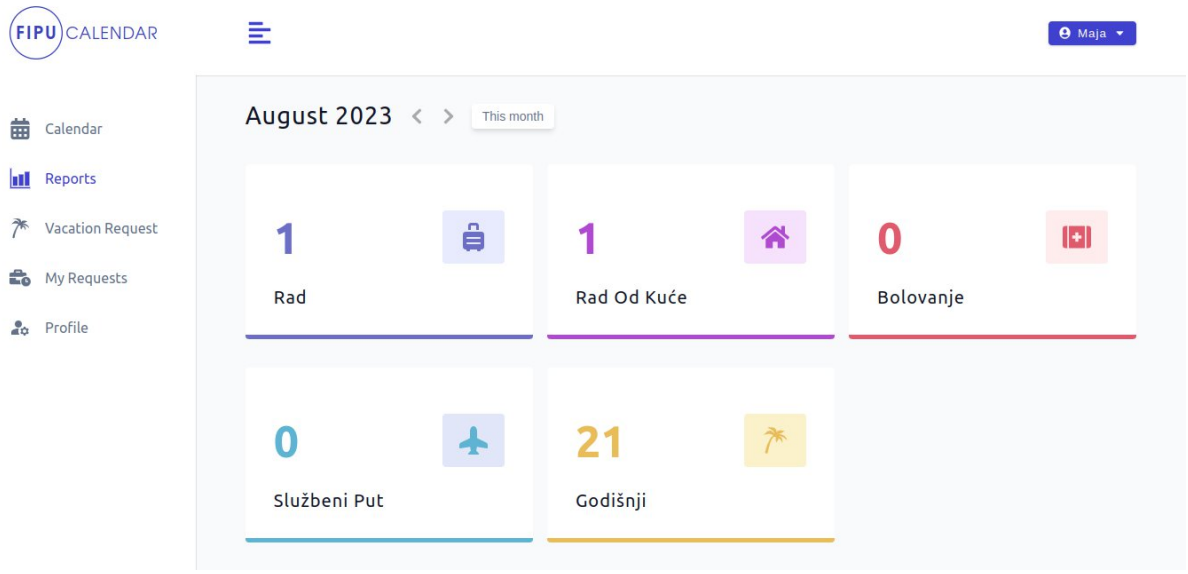
Slika 25. Ažuriranje događaja u tijeku (izvor: autor)

Nakon toga dobivamo povratnu poruku kako smo uspješno ažurirali odabrani event i možemo vidjeti kako je sada 29.8.2023. označen kao „Rad”.



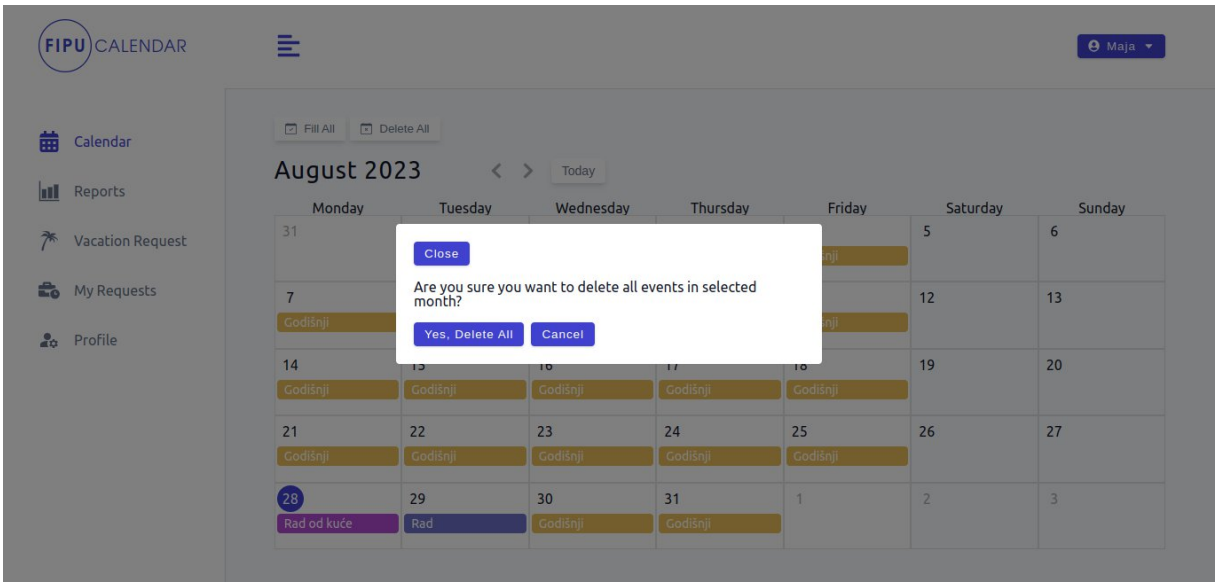
Slika 26. Prikaz uspješnog ažuriranja događaja (izvor: autor)

U navigacijskoj traci možemo odabrati „Reports”, odnosno izvještaj za pojedini mjesec koji korisniku prikazuje ukupan broj radnih sati po tipu događaja. Ovdje, na slici broj 27 je vidljivo kako je u osmom mjesecu jedan dan radio na fakultetu, jedan dan od kuće, a 21 dan je bio na godišnjem odmoru.



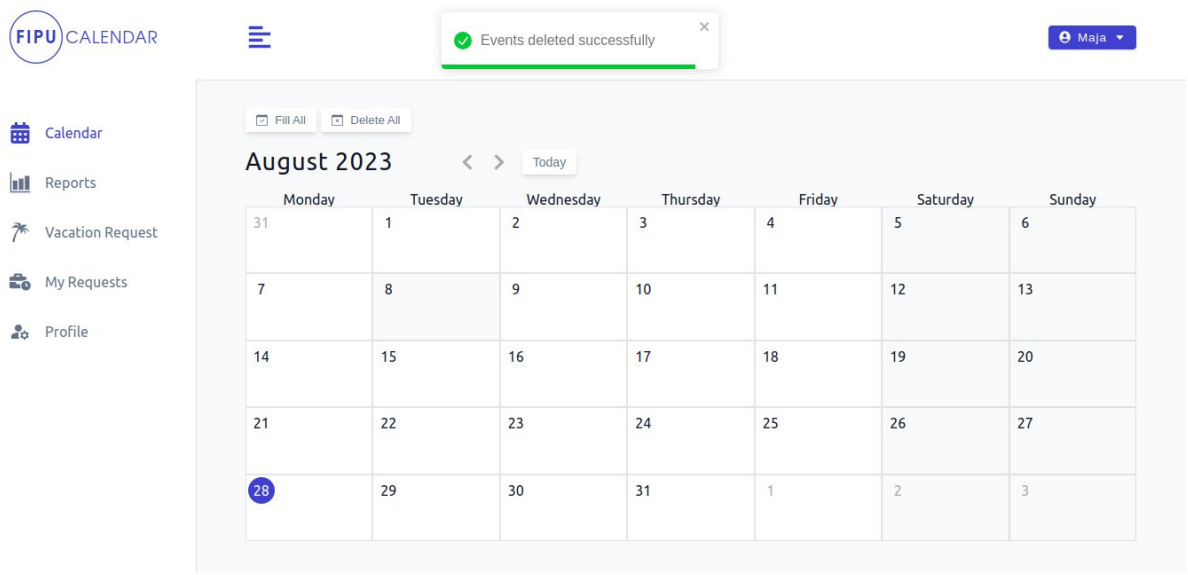
Slika 27. Mjesečni izvještaj profesora o radnim danima (izvor: autor)

Također je moguće izbrisati sve evente u odabranom i prikazanom mjesecu klikom na gumb „Delete All”, no prije toga nam se prikaže prozor s porukom „Da li ste sigurni da želite izbrisati sve dane u mjesecu?”. U slučaju da korisnik odabere „Yes, Delete All” svi eventi za taj mjesec će biti izbrisani.



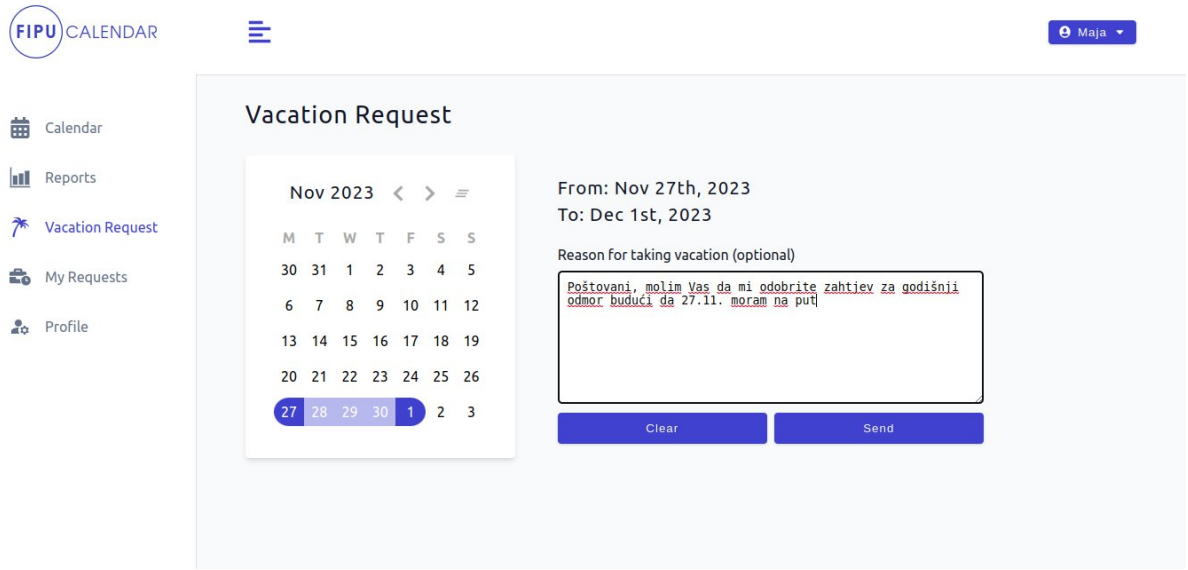
Slika 28. Brisanje svih događaja u mjesecu (izvor: autor)

Na sljedećoj slici broj 29 možemo vidjeti povratnu poruku kako su eventi uspješno izbrisani te prazne ćelije kalendara na koje je opet moguće kliknuti budući da nema događaja.



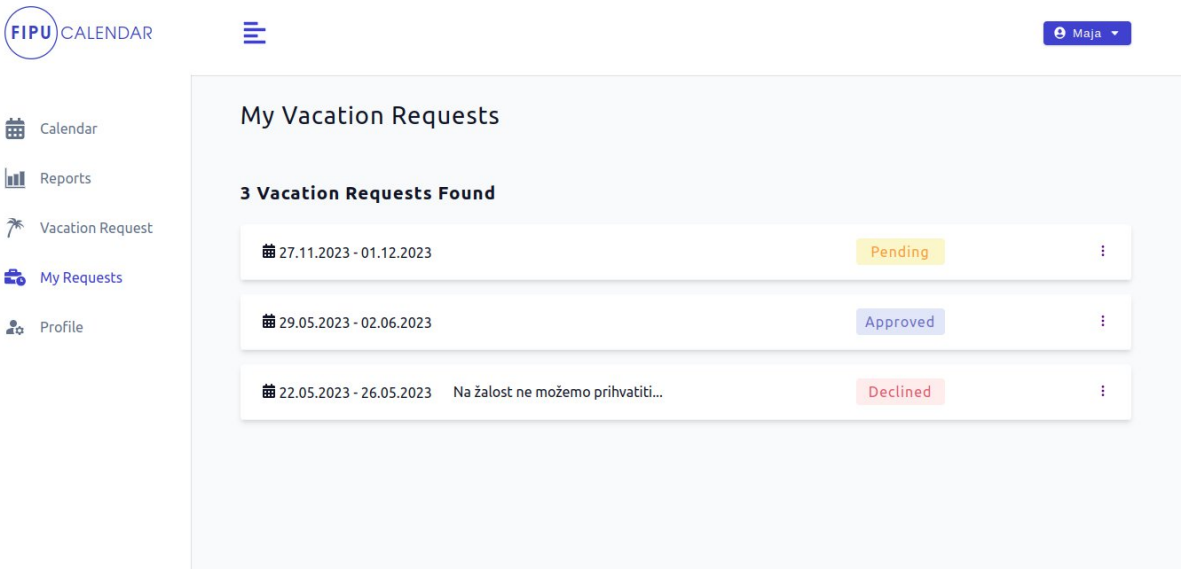
Slika 29. Prikaz uspješno izbrisanih događaja (izvor: autor)

Osim funkcionalnosti za dodavanje događaja u kalendar još jedna funkcionalnost aplikacije je i predavanje zahtjeva za godišnji odmor što je vidljivo na slici broj 30. Kada korisnik u navigacijskom izborniku odabere „Vacation Request” tada u kalendaru može odabrati početni i zadnji dan godišnjeg te (opcionalno) razlog zbog kojeg mu treba godišnji odmor. Nakon što klikne na gumb „Send” taj zahtjev je poslan tajnici, a aplikacija nas preusmjerava na stranicu sa svim zahtjevima. Jednom poslani zahtjev više nije moguće brisati.



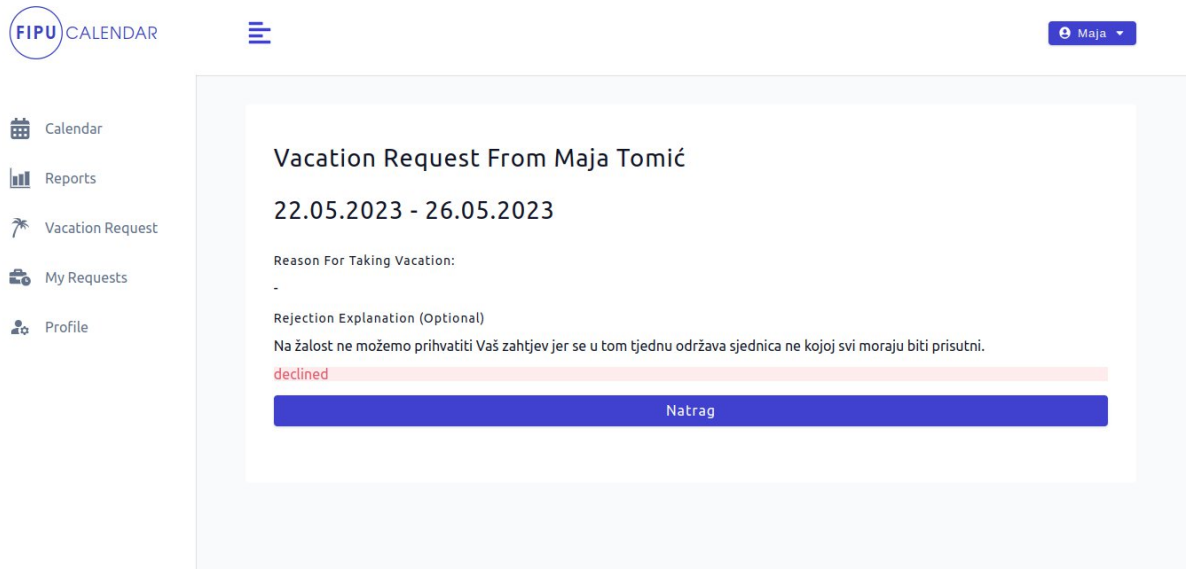
Slika 30. Kreiranje zahtjeva za godišnji odmor (izvor: autor)

Na slici broj 31 možemo vidjeti stranicu sa svim predanim zahtjevima, od kojih neki imaju status na čekanju, dok su ostali odobreni ili odbijeni.



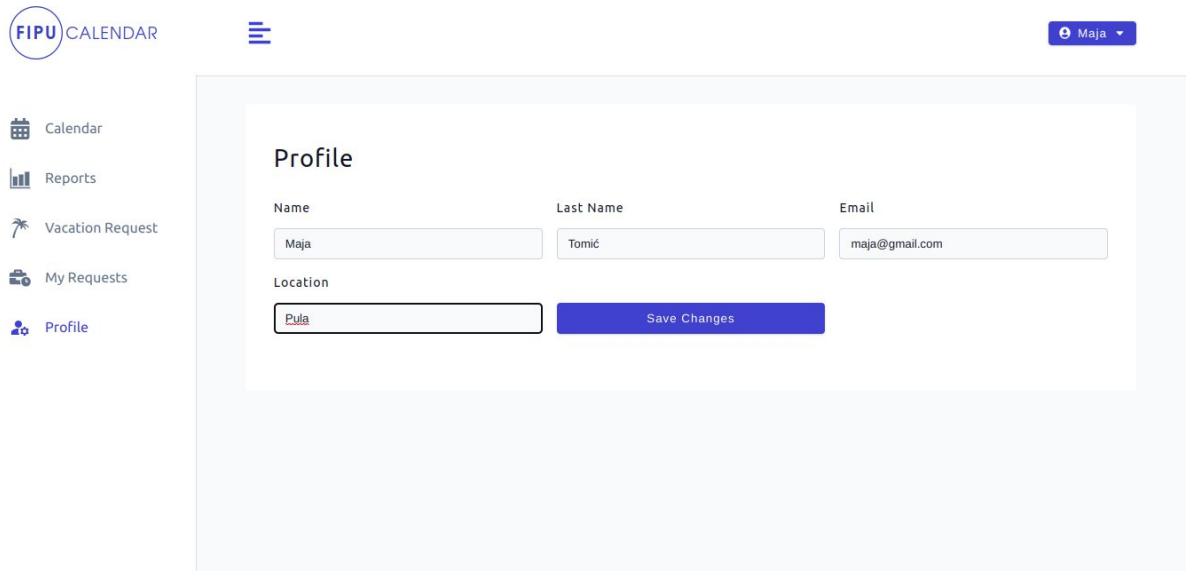
Slika 31. Popis predanih zahtjeva za godišnji odmor (izvor: autor)

Profesor ne može ažurirati zahtjev, to može samo tajnica. Međutim može vidjeti više informacija o predanom zahtjevu klikom na ikonu koja označava „više detalja” što možemo vidjeti na slici broj 32.



Slika 32. Prikaz predanog zahtjeva za godišnji odmor (izvor: autor)

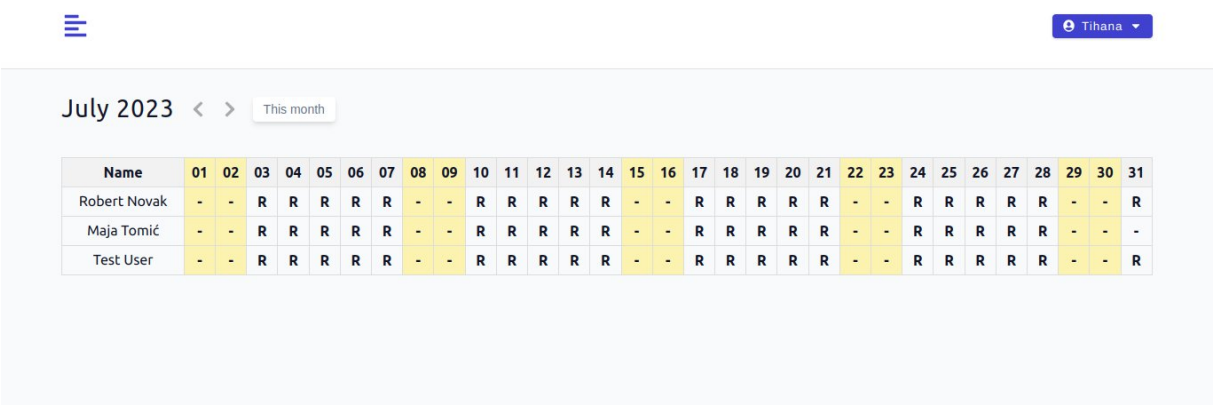
Zadnja u navigacijskom izborniku je i stranica „Profile” na kojoj korisnici mogu ažurirati svoje podatke. Trenutno je i profesorima i tajnici omogućen pristup ovoj stranici jer dijele iste funkcionalnosti. Međutim, u budućnosti bi se to moglo promijeniti, ako će biti potrebno dodati specifične informacije o profesorima koje će tajnica moći ispunjavati umjesto njih i slično.



Slika 33. Editiranje profila (izvor: autor)

6.2. Korisničke upute – tajnica

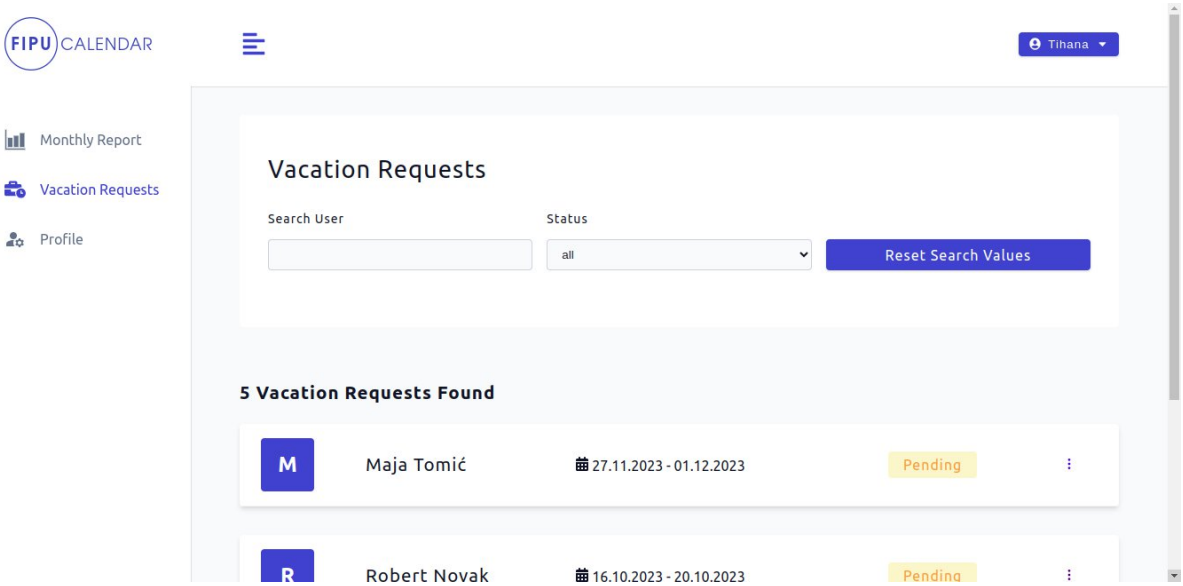
Nakon uspješne prijave, tajnica se automatski preusmjerava na početnu stranicu gdje odmah može vidjeti mjesečni izvještaj svih profesora. Na ovoj slici broj 34 možemo vidjeti mjesečni izvještaj za lipanj 2023. godine, koji uključuje imena i prezimena svih profesora te prikaz svih dana u mjesecu. Pomoću strelica možemo navigirati u prethodni, odnosno sljedeći mjeseci ili se pak pritiskom na gumb „This Month" vratiti na trenutni mjesec. U budućnosti se može razmotriti dodavanje tražilice ili slične funkcionalnosti radi dodatne preglednosti. Subote i nedjelje su posebno označene radi lakšeg raspoznavanja.



Name	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Robert Novak	-	-	R	R	R	R	R	-	-	R	R	R	R	R	-	-	R	R	R	R	R	-	-	R	R	R	R	R	-	-	R
Maja Tomić	-	-	R	R	R	R	R	-	-	R	R	R	R	R	-	-	R	R	R	R	R	-	-	R	R	R	R	R	-	-	-
Test User	-	-	R	R	R	R	R	-	-	R	R	R	R	R	-	-	R	R	R	R	R	-	-	R	R	R	R	R	-	-	R

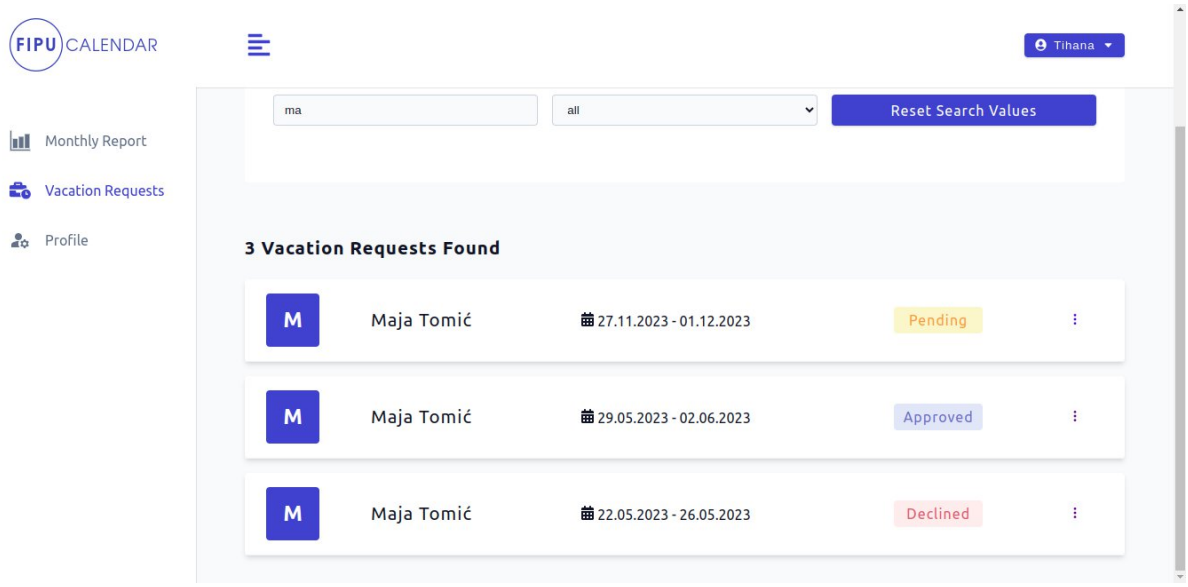
Slika 34. Mjesečni izvještaj o radnim danima svih profesora (izvor: autor)

Za razliku od mjesečnog izvještaja stranica „Vacation Requests” ima implementiranu tražilicu. Ona sadrži zahtjeve za godišnji svih profesora koji mogu biti redom „pending”, „approved” i „declined”. Na slici broj 35 možemo vidjeti kako imamo ukupno 5 zahtjeva za godišnji. Trenutno je zadano da se odmah prikažu svi zahtjevi no to je po potrebi lako izmijeniti u slučaju da želimo da nam se prikazuju zahtjevi sa statusom „pending”.



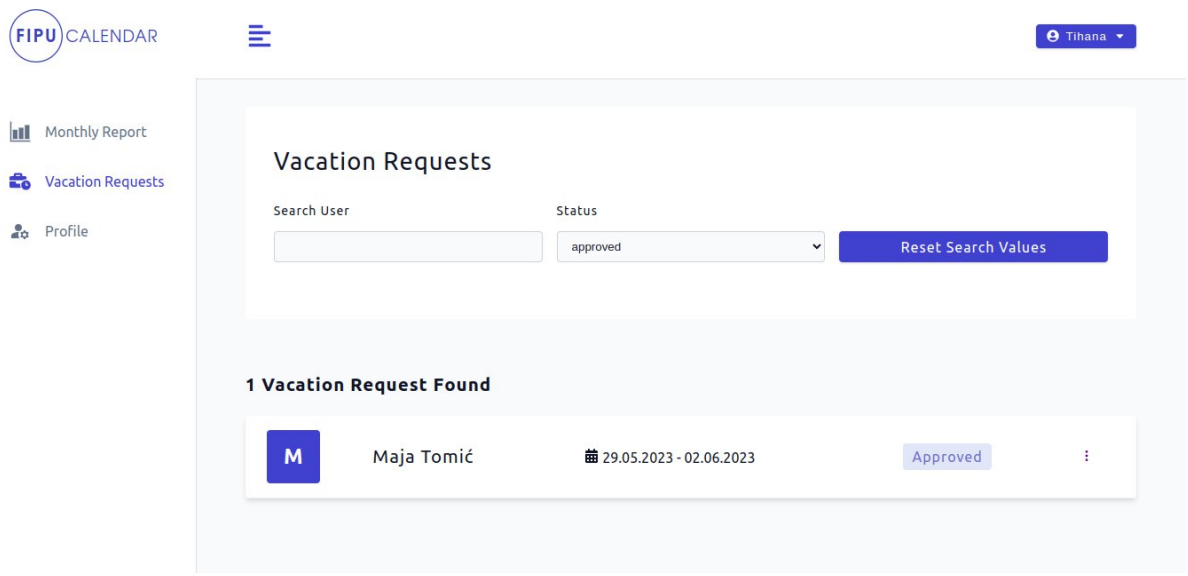
Slika 35. Prikaz svih zahtjeva za godišnji odmor (izvor: autor)

U primjeru na slici broj 36 možemo vidjeti pretraživanje koje pokaže samo one profesore koji u svojem imenu ili prezimenu imaju „ma”, u ovom slučaju je to Maja. Vidimo kako Maja ima tri zahtjeva od kojih je jedan na čekanju dok je drugi odobren, a treći odbijen.



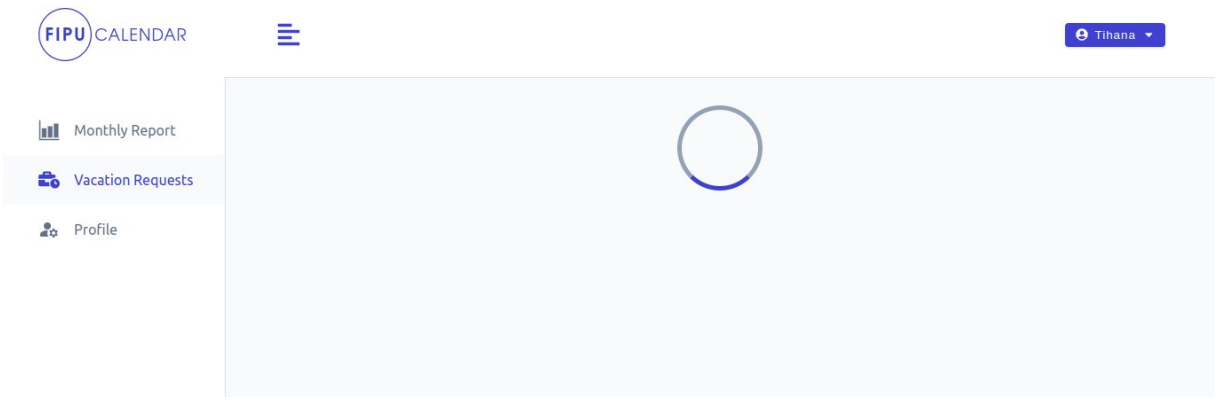
Slika 36. Pretraživanje po imenu i prezimenu (izvor: autor)

Osim pretraživanja po imenu i prezimenu, moguće je pretraživati, odnosno filtrirati status zahtjeva za godišnji odmor. Prema zadanim postavkama odabran je status „all”, no možete odabrati između „pending”, „approved” i „declined” koristeći padajući izbornik. U ovom primjeru pretražujemo sve korisnike koji imaju status ”approved”.



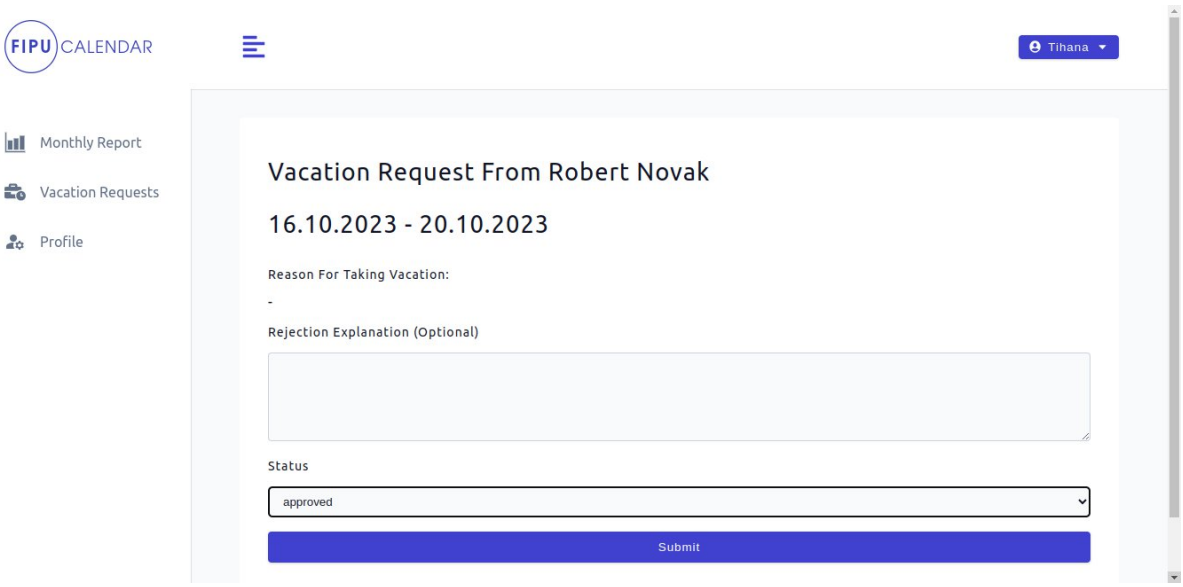
Slika 37. Pretraživanje po statusu (izvor: autor)

Kada kliknemo na gumb „Reset Search Values” aplikacija nas vraća na zadanu stranicu te briše sve prethodno unesene upite. U tom trenutku, možemo primijetiti tzv. „loader” koji signalizira da se stranica učitava.



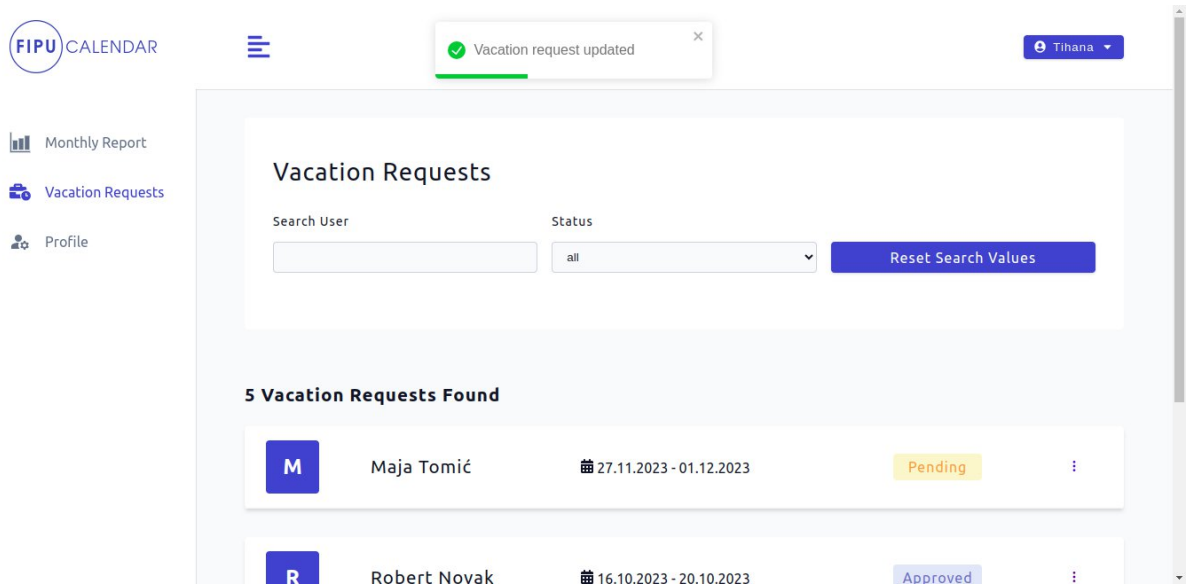
Slika 38. Resetiranje tražilice (izvor: autor)

Klikom na odabrani zahtjev možemo odgovoriti na taj zahtjev, odnosno odobriti ga ili odbiti. To je vidljivo na slici broj 39. Zahtjev za godišnji označavamo kao „approved”, odnosno odobren.



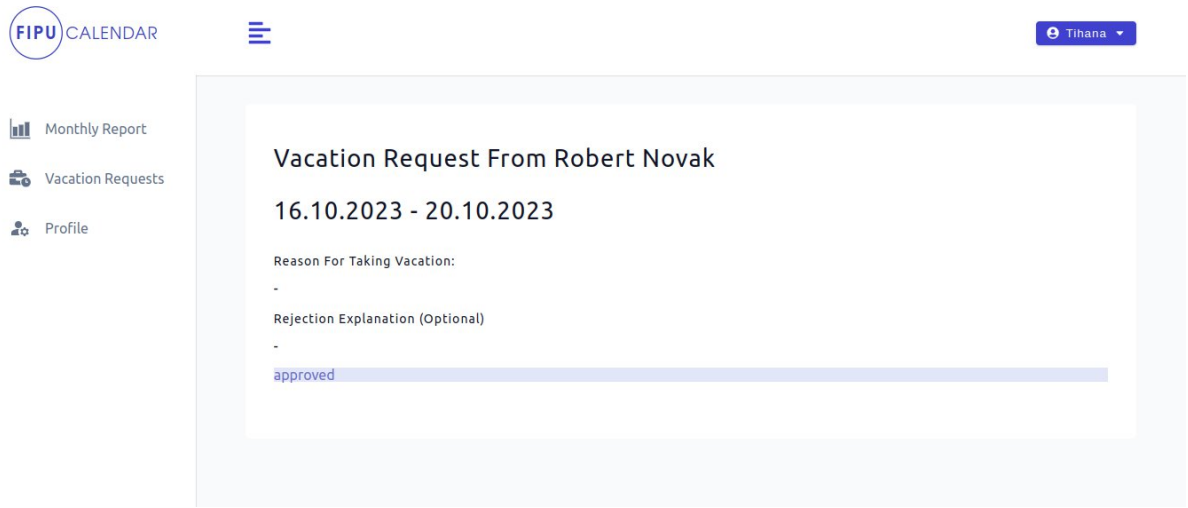
Slika 39. Ažuriranje zahtjeva za godišnji odmor (izvor: autor)

Na sljedećoj slici broj 40 jasno je prikazano kako je zahtjev uspješno ažuriran. Nakon ažuriranja, aplikacija nas automatski preusmjerava na stranicu s popisom svih zahtjeva, gdje sada možemo vidjeti da je taj zahtjev označen kao „approved”.



Slika 40. Prikaz uspješno ažuriranog zahtjeva za godišnji odmor (izvor: autor)

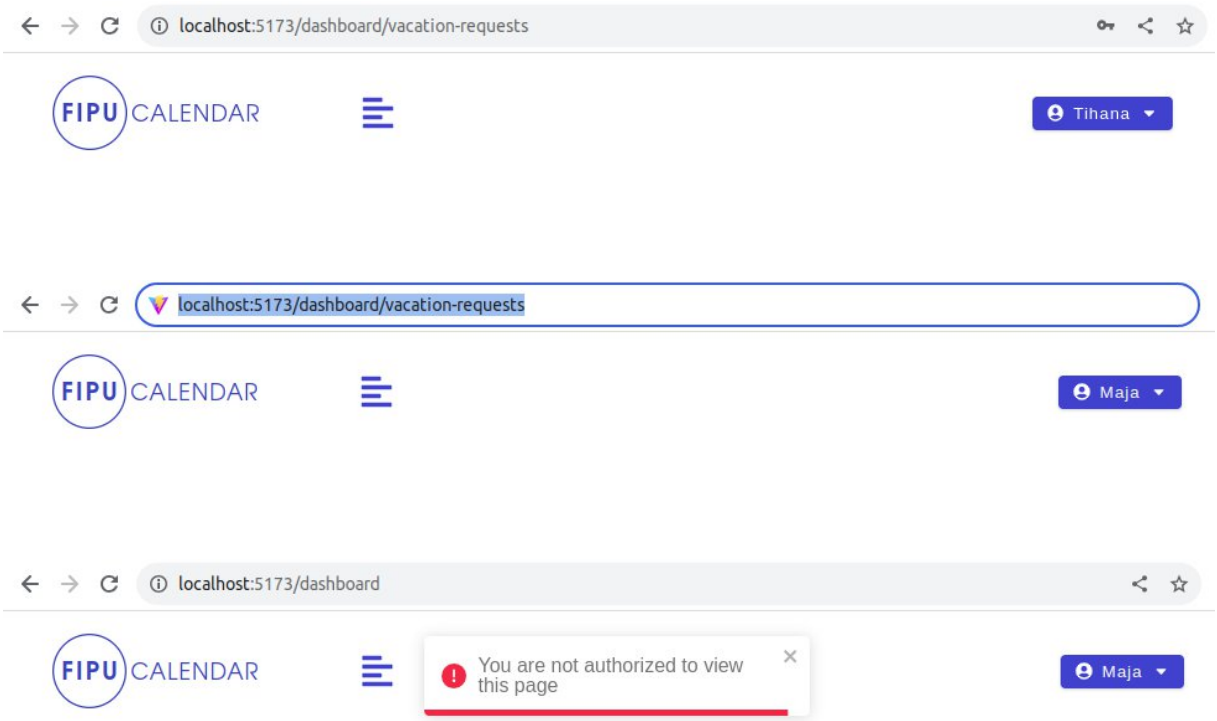
Jednom ažurirani zahtjev nije više moguće promijeniti, moguće je samo ga pregledati, a to je prikazano na sljedećoj slici broj 41.



Slika 41. Ažurirani zahtjev za godišnji odmor (izvor: autor)

6.3. Korisničke role i prava pristupa

Sljedeća slika broj 42 prikazuje URL „/dashboard/vacation-request” stranicu kojoj tajnica ima pristup. Međutim, ako je profesor prijavljen i pokuša pristupiti toj stranicu dobiva povratnu informaciju „Nemate dopuštenje za pregled ove stranice” i automatski će biti preusmjeren na „/dashboard”.



Slika 42. Korisničke role i pravo pristupa (izvor: autor)

7. BUDUĆI RAZVOJ

Kao i svaka aplikacija i ova ima potencijala za daljnje unapređenje, a ovo je tek početak u usporedbi s mogućnostima koje ovakva jedna aplikacija ima za ponuditi.

Jedna od ključnih nadogradnji bi bila implementacija „read-only“ pregleda kalendara, za prethodne mjesece. Treba još razmotriti na koji način to implementirati: Može li tajnica označiti mjesečni izvještaj kao pregledan ili bi se automatski odredio dan u mjesecu nakon kojeg više nije moguće izmijeniti događaje u kalendaru.

Također, nakon što tajnica odobri zahtjev za godišnji odmor, bilo bi dobro da ti dani budu automatski evidentirani u kalendaru korisnika te da uređivanje tih događaja više neće biti moguće.

Za sada aplikacija ima samo dvije korisničke role, profesor i tajnica, međutim uvođenje različitih rola bi omogućila bolje upravljanje pristupom i privilegijama unutar aplikacije.

Dodavanje neradnih dana, uključujući državne praznike i blagdane, koji bi odmah u aplikaciji bili označeni ako neradni, kao što su to trenutno subote i nedjelje.

Mogućnost praćenja kolektivnog godišnjeg odmora, kao i prikaz preostalog godišnjeg odmora za svakog korisnika, bili bi dodatni elementi koji bi obogatili funkcionalnosti aplikacije.

Mjesečni izvještaj o prijevozu bi omogućio jasno sagledavanje prava profesora na novčanu naknadu za prijevoz, što bi uvelike olakšalo administrativne poslove.

U cjelini, ova aplikacija predstavlja početak puta prema izgradnji aplikacije koja će unaprijediti način na koji se prate radni dani zaposlenika i olakšati evidenciju.

8. ZAKLJUČAK

Uvođenje ovakve jedne aplikacije u svakodnevne poslovne procese donosi mnoge prednosti, posebno u suvremenom poslovnom okruženju gdje se digitalne tehnologije koriste za efikasnije upravljanje poslovanjem. Ovaj projekt donosi praktično rješenje za evidenciju radnih dana zaposlenika na sveučilištu. Također, aplikacija pruža izvještaje i analize o radnim danima zaposlenika, što olakšava planiranje i komunikaciju među članovima sveučilišne zajednice. Osim toga, postoji mogućnost podnošenja zahtjeva za godišnji odmor, omogućujući tajnici pregled zahtjeva svih zaposlenika.

Uvođenjem ovakvog sustava, sveučilišta i obrazovne institucije mogu značajno unaprijediti procese praćenja i evidencije radnih dana svojih zaposlenika. Iako postoji konkurencija na tržištu, u obliku raznih kalendara, ova aplikacija se izdvaja prilagodbom specifičnih potreba sveučilišnih organizacija. Dodatno, uvođenje „ureda bez papira“ donosi brojne prednosti, uključujući smanjenje troškova, brži pristup informacijama, bolju organizaciju, ali i povećanu ekološku svijest.

Implementacija ove aplikacije predstavlja značajan korak prema olakšanoj komunikaciji između tajnice i profesora na sveučilištu, uz potencijal za širu primjenu u različitim organizacijama koje cijene efikasnu razmjenu informacija i koordinaciju među članovima svoje zajednice.

LITERATURA

1. Ashiq KS, Aggregation in MongoDB, 2019, Dostupno na: https://medium.com/@ashiqgiga07/aggregation-in-mongodb-209515e5b0ba (20. 8. 2023)
2. Cappellari, C. S. Fundamentals of REST API, 2021, Dostupno na: https://dev.to/cassiocappellari/fundamentals-of-rest-api-2nag (12. 9. 2023)
3. Couto, J. R. The Beginner's Guide to MongoDB Aggregation (With Exercise), 2023, Dostupno na: https://studio3t.com/knowledge-base/articles/mongodb-aggregation-framework/ (21. 8. 2023)
4. Karlsson, J. MongoDB Aggregation Pipeline Queries vs SQL Queries, 2022, Dostupno na: https://www.mongodb.com/developer/products/mongodb/sql-to-aggregation-pipeline/ (20. 8. 2023)
5. Lemonaki, D. What is a REST API? API Endpoint Request Example, Dostupno na: https://www.freecodecamp.org/news/what-is-a-rest-api/ (12.09.2023)
6. Tanjona, How does the MERN stack work?, 2020, Dostupno na: https://www.bocasay.com/how-does-the-mern-stack-work/ (6. 7. 2023)
7. MongoDB, Aggregation Pipeline, Dostupno na: https://www.mongodb.com/docs/manual/core/aggregation-pipeline/ (25. 8. 2023)
8. MongoDB, Model One-to-One Relationships with Embedded Documents, Dostupno na: https://www.mongodb.com/docs/manual/tutorial/model-embedded-one-to-one-relationships-between-documents/ (25. 8. 2023)
9. Nodemon, Dostupno na: https://nodemon.io/ (12.09.2023)
10. React Router, Dostupno na: https://reactrouter.com/en/main (25. 8. 2023)
11. Vite, Dostupno na: https://vitejs.dev/guide/why.html (12.09.2023)
12. What is REST?, Dostupno na: https://www.codecademy.com/article/what-is-rest (12. 9. 2023)

POPIS SLIKA

Slika 1. Use Case dijagram sustava (izvor: autor) 6
Slika 2. Dijagram aktivnosti za predaju zahtjeva za godišnji odmor (izvor: autor) 7
Slika 3. Ilustrativni dijagram MERN Stack razvojnog okvira (izvor: [6]) 9
Slika 4. Prikaz arhitekturnog stila „RESTful“ (izvor: [12]) 13
Slika 5. Struktura klijentske strane (izvor. autor) 18
Slika 6. Prikaz glavne komponente aplikacije „App“ (izvor: autor) 19
Slika 7. Komponentni dijagram strukture App.jsx routera (izvor: autor) 20
Slika 8. Komponenta „Form“ (izvor: autor) 22
Slika 9. Prikaz funkcije „action“ za dohvat podataka iz komponente Form (izvor: autor) 22
Slika 10. Prikaz funkcije „loader“ za dohvat podataka sa poslužitelja (izvor: autor) 23
Slika 11. Komponenta „Calendar“ (izvor: autor) 24
Slika 12. Struktura poslužiteljske strane (izvor: autor) 26
Slika 13. Prikaz ruta za autentifikaciju korisnika (izvor: autor) 27
Slika 14. Kolekcije u MongoDB bazi podataka (izvor: autor) 29
Slika 15. Primjer koda za agregacijski pipeline (izvor: autor) 31
Slika 16. Početna stranica aplikacije (izvor: autor) 33
Slika 17. Prikaz stranica za registraciju ili prijavu korisnika (izvor: autor) 34
Slika 18. Neuspješna prijava (izvor: autor) 35
Slika 19. Početna stranica profesora (izvor: autor) 36
Slika 20. Kreiranje događaja „Rad od kuće“ u kalendar (izvor: autor) 37
Slika 21. Prikaz kreiranog događaja „Rad od kuće“ u kalendaru (izvor: autor) 38
Slika 22. Popunjavanje preostalih dana u mjesecu (izvor: autor) 39
Slika 23. Prikaz kreiranih događaja za sve dane u mjesecu (izvor: autor) 40
Slika 24. Ažuriranje događaja (izvor: autor) 41
Slika 25. Ažuriranje događaja u tijeku (izvor: autor) 42
Slika 26. Prikaz uspješnog ažuriranja događaja (izvor: autor) 43
Slika 27. Mjesečni izvještaj profesora o radnim danima (izvor: autor) 44
Slika 28. Brisanje svih događaja u mjesecu (izvor: autor) 45
Slika 29. Prikaz uspješno izbrisanih događaja (izvor: autor) 46
Slika 30. Kreiranje zahtjeva za godišnji odmor (izvor: autor) 47
Slika 31. Popis predanih zahtjeva za godišnji odmor (izvor: autor) 48
Slika 32. Prikaz predanog zahtjeva za godišnji odmor (izvor: autor) 49
Slika 33. Editiranje profila (izvor: autor) 50
Slika 34. Mjesečni izvještaj o radnim danima svih profesora (izvor: autor) 51
Slika 35. Prikaz svih zahtjeva za godišnji odmor (izvor: autor) 52
Slika 36. Pretraživanje po imenu i prezimenu (izvor: autor) 53
Slika 37. Pretraživanje po statusu (izvor: autor) 54
Slika 38. Resetiranje tražilice (izvor: autor) 55
Slika 39. Ažuriranje zahtjeva za godišnji odmor (izvor: autor) 56
Slika 40. Prikaz uspješno ažuriranog zahtjeva za godišnji odmor (izvor: autor) 57
Slika 41. Ažurirani zahtjev za godišnji odmor (izvor: autor) 58
Slika 42. Korisničke role i pravo pristupa (izvor: autor) 59

PRILOZI

FIPU Calendar App

https://github.com/TihanaG/fipu-calendar-app

Aplikacija dostupna na:

https://fipu-calendar-app.onrender.com/

SAŽETAK

Ovaj projekt predstavlja tzv. MERN web aplikaciju koja je namijenjena olakšavanju evidencije radnih dana zaposlenika na sveučilištu, fokusirajući se prvenstveno na tajnicu i profesore. Aplikacija omogućava korisnicima da bilježe svoje radne dane putem interaktivnog kalendara, pružajući opcije za označavanje prisutnosti na sveučilištu, rada od kuće, službenih putovanja, bolovanja ili godišnjeg odmora. Kroz implementaciju „ureda bez papira“, aplikacija donosi brojne prednosti u smanjenju troškova, trenutnom pristupu informacijama, boljoj organizaciji i jačanju ekološke svijesti.

Za komunikaciju između klijenta i poslužitelja implementiran je RESTful API, odnosno arhitekturni stil koji omogućava jednostavnu i učinkovitu komunikaciju između različitih dijelova sustava putem HTTP zahtjeva. Ovaj pristup olakšava manipulaciju podacima, omogućavajući operacije kao što su stvaranje, čitanje, ažuriranje i brisanje resursa na serveru. Kombinacija React.js-a na klijentskoj strani i Node.js-a s Express.js-om na serverskoj strani omogućava implementaciju RESTful API-ja, osiguravajući visoku razinu učinkovitosti i pouzdanosti u rukovanju zahtjevima korisnika.

Podaci su pohranjeni u MongoDB bazu podataka, koja je visoko skalabilna i fleksibilna NoSQL baza podataka. Ova tehnologija omogućava učinkovito upravljanje podacima i prilagodbu specifičnim potrebama aplikacije.

Na kraju, ova aplikacija ima potencijal za širu primjenu u različitim organizacijama.

Ključne riječi: arhitektura MERN, REST, React.js, Node.js, Express.js, MongoDB, NoSQL, web aplikacija, evidencija radnih dana, interaktivni kalendar, „ured bez papira“

SUMMARY

This project represents a so-called MERN web application designed to facilitate the tracking of employees' workdays at the university, with a primary focus on secretary and professors. The application enables users to record their workdays through an interactive calendar, providing options to mark attendance at the university, work from home, official trips, sick leaves, or annual leaves. Through the implementation of a „paperless office", the application brings numerous advantages in cost reduction, instant access to information, improved organization, and enhanced environmental awareness.

To faciliate communication between the client and server, a RESTful API has been implemented. This architectural style enables simple and efficient communication between different parts of the system through HTTP requests. This approach simplifies data manipulation, allowing operations such as creation, reading, updating, and deletion of resources on the server. The combination of React.js on the client side and Node.js with Express.js on the server side enables the implementation of a RESTful API, ensuring a high level of efficiency and reliability in handling user requests.

Data is stored in a MongoDB database, which is a highly scalable and flexible NoSQL database. This technology enables efficient data management and customization to the specific needs of the application.

Keywords: MERN architecture, REST, React.js, Node.js, Express.js, MongoDB, NoSQL, web applications, workday tracking, interactive calendar, „paperless office"