

Akademija tehničko-vaspitačkih strukovnih studija  
Niš

Predmet: Klijent server sistemi

**Uputstvo za izradu projekta u Vue.js radnom okviru**

Niš, oktobar 2024.

# Sadržaj

Cilj izrade projekta .....	2
Vue.js radni okvir .....	2
1. Inicijalizacija, konfiguracija projekta i instalacija biblioteka .....	3
1.1 Koraci za konfiguraciju projekta .....	3
2. Princip rada Vue.js-a.....	10
2.1 Glavne komponente i objašnjenje strukture u Vue.js projektu .....	10
3. Klijent server arhitektura .....	14
3.1 Pojam API-ja .....	14
3.2 Pojam Autentifikacije .....	17
3.2.1 Autentifikacija kolačićima (engl. Cookie authentication) .....	17
3.2.2 JWT Autentifikacija (engl. JSON Web Token authentication) .....	18
4. Izrada aplikacije .....	21
4.1 Izgled aplikacije, stranice i rute .....	21
4.2 Razdvajanje u komponente .....	24
5. Kreiranje komponenti i stranice za login .....	25
5.1 views/Login.vue .....	25
5.2 components/Navbar.vue .....	31
5.3 components/TableContainer.vue .....	37
5.4 components/FormContainer.vue.....	41
5.5 components/DeleteModal.vue .....	41
6. Kreiranje stranica za manipulaciju korisnicima .....	50
6.1 views/KorisnikNovi.vue .....	50
6.2 views/KorisnikIzmena.vue .....	55
6.3 views/Korisnici.vue .....	61
<b>LITERATURA</b> .....	<b>67</b>

# Klijent server sistemi

## Uputstvo za izradu projekta u Vue.js radnom okviru

### Cilj izrade projekta

- Upoznavanje studenata sa osnovama komunikacije između klijenta i servera, kroz implementaciju *REST API*-ja (engl. *Representational State Transfer Application Programming Interface*)
- Razumevanje kako se koristi Vue.js za kreiranje dinamičnih korisničkih interfejsa i upravljanje stanjem aplikacije.
- Savladavanje rada sa *HTTP* (engl. *Hypertext Transfer Protocol*) metodama putem biblioteke *Axios*, u kontekstu Vue.js aplikacija.
- Učenje kako pravilno organizovati reusable komponente u Vue.js-u, s ciljem smanjenja ponavljanja koda i poboljšanja modularnosti aplikacije.
- Savladavanje *Bootstrap*-a za kreiranje responzivnih korisničkih interfejsa, uključujući rad sa modalima i formama.
- Primena konceptata upravljanja stanjem u Vue.js-u kroz "*ref*" i "*props*" sisteme, radi efikasnog prenosa podataka između komponenti.

### Vue.js radni okvir

Vue.js je progresivni JavaScript radni okvir za izradu korisničkih interfejsa i jednostraničnih aplikacija *SPA* (engl. *Single Page Application*). Kreiran je 2014. godine od strane Evana You-a, kao lakša i fleksibilnija alternativa za druge popularne radne okvire poput *Angular*-a i *React*-a. Vue.js omogućava modularnu izgradnju aplikacija putem komponentnog pristupa, gde se logika i prezentacija lako razdvajaju. Posebno je cenjen zbog jednostavne integracije u postojeće projekte i mogućnosti postepenog usvajanja, što ga čini pogodnim za projekte svih veličina.



Sl. Vue.js logotip

## 1. Inicijalizacija, konfiguracija projekta i instalacija biblioteka

Zadatak je izrada *frontend* dela veb aplikacije u kojoj profesori vode evidenciju o studentima, predmetima i ocenama. Od funkcionalnosti je potrebno da ima login, logout, kreiranje, iščitavanje, izmenu i brisanje studenata, predmeta, ocena i profesora (u daljem tekstu korisnici). Za izradu, neophodna je instalacija određenih alata i biblioteka, kao i urađen projekat u Flask radnom okviru (predmet Veb programiranje), modifikovan u *backend API*.

- Link do Evidencija studenata Flask *API*-ja: <https://github.com/nevence/Klijent-Server-Sistemi-Evidencija-studenata/tree/main/flask-api>

### 1.1 Koraci za konfiguraciju projekta

#### 1. Instalacija Node.js-a

- Node.js je okruženje koje omogućava pokretanje JavaScript koda van pretraživača. Za rad sa Vue.js, Node.js je neophodan jer omogućava instalaciju paketa i pokretanje lokalnog servera. Moguće ga je instalirati na zvaničnom Node.js sajtu: <https://nodejs.org/en/download/prebuilt-installer>
- Na slici 1.1 prikazana je zvanična stranica za preuzimanje Node.js-a sa mogućnostima odabira željene verzije, kao i operativnog sistema. Nakon skidanja željene instalacije, potrebno ju je pokrenuti, i ispratiti korake kroz instalacioni čarobnjak. Na slici 2 prikazan je početak instalacije.



Sl. 1.1 Zvanični sajt za preuzimanje Node.js-a



Sl. 1.2 Pokretanje instalacije Node.js-a

## 2. Kreiranje novog Vue.js projekta pomoću Vite-a

- Potrebno je otvoriti *Command prompt*, navigirati do *Desktop-a* i pokrenuti komandu:

```
npm create vue@latest evidencija_studenata
```

- Za instaliranje dodatnih zavisnosti, odabrati samo Vue Router, koji će služiti za rutiranje i navigaciju ka stranicama.
- Pokrenuti komande iz konzole za instalaciju:

```
cd evidencija_studenata  
npm install
```

- Na slici 1.3 prikazana je konzola sa pokrenutim potrebnim komandama radi kreiranja projekta.

```
Microsoft Windows [Version 10.0.22631.4169]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\neven>cd Desktop  
  
C:\Users\neven\Desktop>npm create vue@latest evidencija_studenata  
  
> npx  
> create-vue evidencija_studenata  
  
Vue.js - The Progressive JavaScript Framework  
  
/ Add TypeScript? ... No / Yes  
/ Add JSX Support? ... No / Yes  
/ Add Vue Router for Single Page Application development? ... No / Yes  
/ Add Pinia for state management? ... No / Yes  
/ Add Vitest for Unit Testing? ... No / Yes  
/ Add an End-to-End Testing Solution? ... No  
/ Add ESLint for code quality? ... No / Yes  
/ Add Vue DevTools 7 extension for debugging? (experimental) ... No / Yes  
  
Scaffolding project in C:\Users\neven\Desktop\evidencija_studenata...  
  
Done. Now run:  
  
  cd evidencija_studenata  
  npm install  
  npm run dev
```

Sl. 1.3 Kreiranje Vue.js projekta

### 3. Instalacija potrebnih biblioteka:

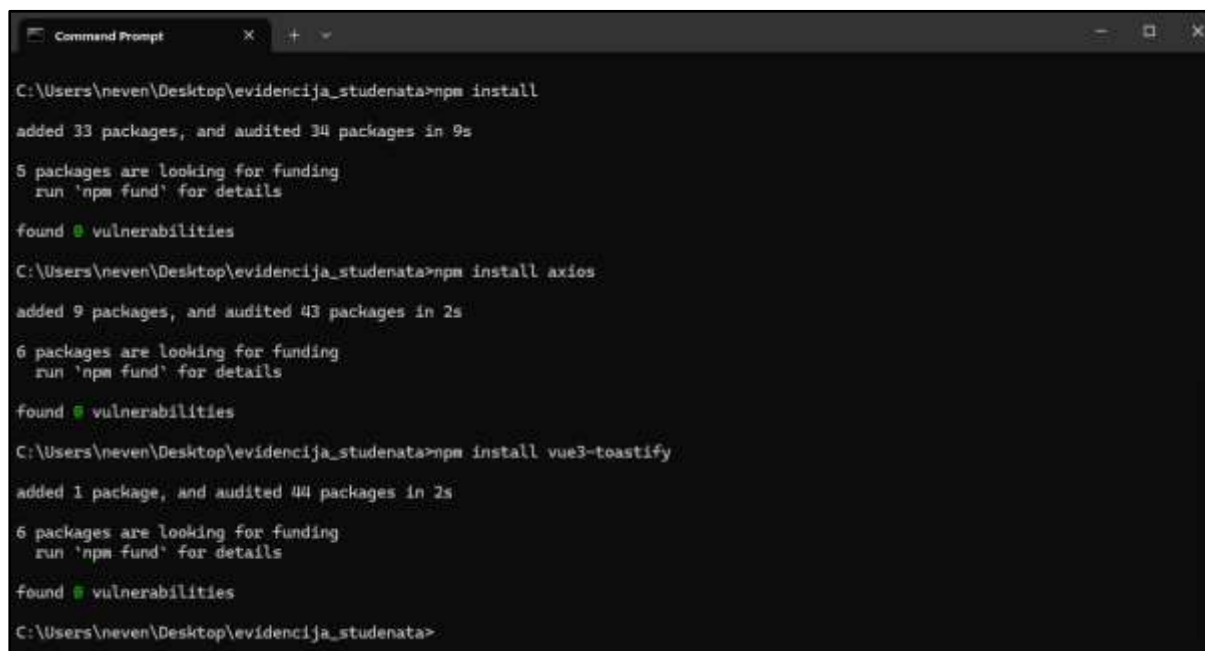
- Nakon instalacije projekta, potrebno je instalirati Axios biblioteku za slanje HTTP zahteva pokretanjem komande u istoj konzoli:

```
npm install axios
```

- Za prikazivanje notifikacija u aplikaciji, potrebno je instalirati Vue 3 Toastify biblioteku pokretanjem komande:

```
npm install vue3-toastify
```

- Na slici 1.4 prikazan je izgled konzole sa pokrenutim svim potrebnim komandama radi intaliranja projekta I svih potrebnih biblioteka.



```
Command Prompt
C:\Users\neven\Desktop\evidencija_studenata>npm install
added 33 packages, and audited 34 packages in 9s
5 packages are looking for funding
  run 'npm fund' for details
found 0 vulnerabilities

C:\Users\neven\Desktop\evidencija_studenata>npm install axios
added 9 packages, and audited 43 packages in 2s
6 packages are looking for funding
  run 'npm fund' for details
found 0 vulnerabilities

C:\Users\neven\Desktop\evidencija_studenata>npm install vue3-toastify
added 1 package, and audited 44 packages in 2s
6 packages are looking for funding
  run 'npm fund' for details
found 0 vulnerabilities

C:\Users\neven\Desktop\evidencija_studenata>
```

Sl. 1.4 Instalacija projekta, Axios i Vue 3 Toastify biblioteka

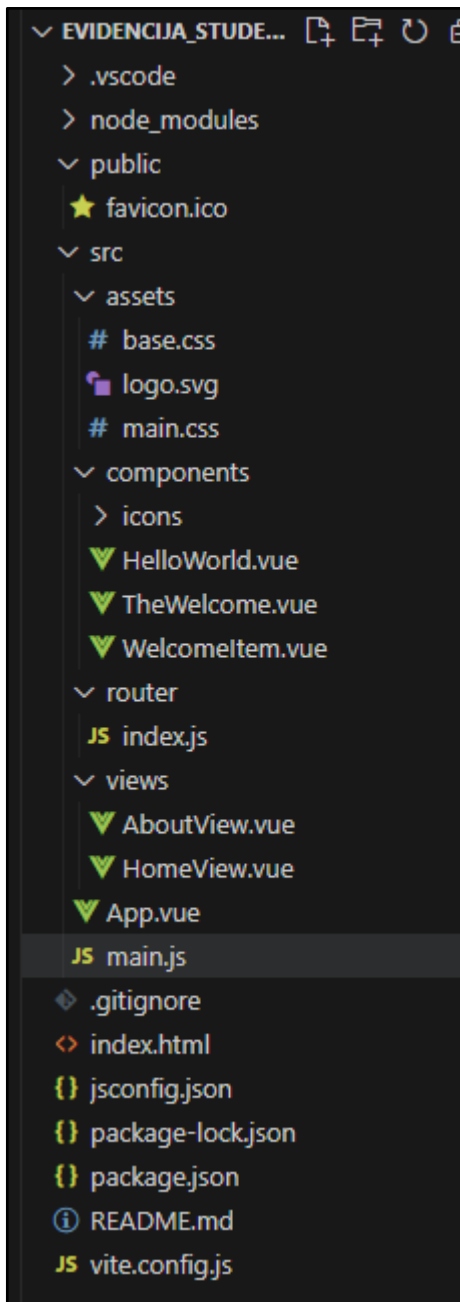
### 4. Brisanje nepotrebnih fajlova

- U istoj konzoli pokrenuti komandu za pokretanje aplikacije.:

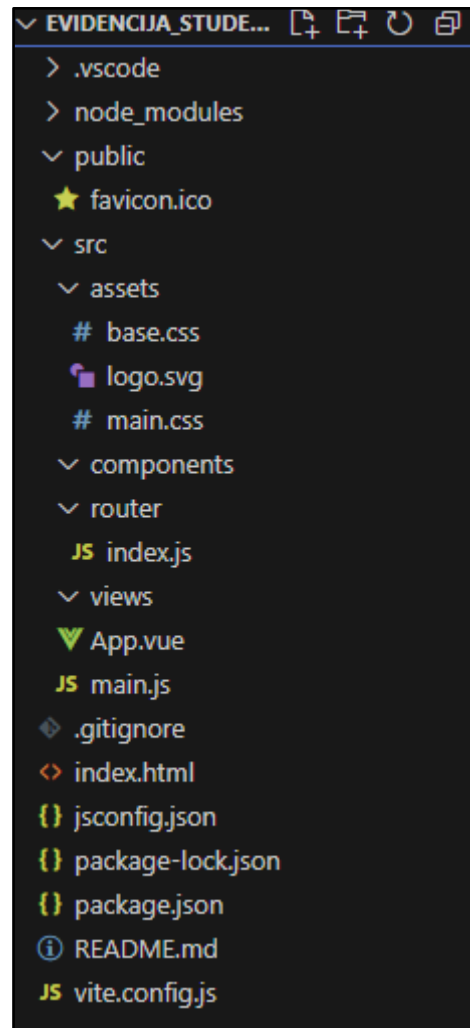
```
npm run dev
```

- Zatim je potrebno otvoriti projekat u VS Code-u. Vue projekat dolazi sa podrazumevanim komponentama (engl. *components*) i pogledima (engl. *views*), koje je potrebno obrisati i izmeniti.
- Na slici 1.5 prikazana je podrazumevana struktura novokreiranog Vue projekta, dok je na slici 1.6 prikazana krajnja struktura koju je potrebno postići nakon brisanja svih fajlova u components I views folderima.

Objašnjenje same strukture i princip rada Vue.js-a biće u zasebnom poglavlju.



Sl. 1.5 Inicijalni novokreirani Vue projekat



Sl. 1.6 Struktura projekta nakon brisanja nepotrebnih fajlova

- Zatim je potrebno izmeniti router/index.js i App.vue fajlove tako da se obrišu nepotrebne linije koda. Slika 1.7 prikazuje krajni router/index.js fajl, dok slika 1.8 prikazuje krajni App.vue fajl nakon brisanja.

```
src > router > JS index.js > ...
1  import { createRouter, createWebHistory } from "vue-router";
2
3  const router = createRouter({
4    history: createWebHistory(import.meta.env.BASE_URL),
5    routes: [],
6  });
7
8  export default router;
9
```

Sl. 1.7 Izmenjeni router/index.js fajl

```
src > App.vue > {} template
1  <script setup>
2    import { RouterLink, RouterView } from 'vue-router'
3  </script>
4
5  <template>
6    <RouterView />
7  </template>
```

Sl. 1.8 Izmenjeni App.vue fajl

## 5. Instalacija Bootstrap 5.3.3 i Font Awesome biblioteka za stilizaciju

- Za implementaciju ovih biblioteka, potrebno je izmeniti index.html tako da budu uključeni potrebni CDN-ovi:
  - <https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css>
  - <https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js>
  - <https://kit.fontawesome.com/0660ce38c6.js>
- Na slici 1.9 nalazi se index.html sa uključenim gorenavedenim *CDN-ovima* (engl. *Content Delivery Network*).



```
index.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8" />
5      <link rel="icon" href="/favicon.ico" />
6      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7      <title>Vite App</title>
8      <link
9        href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
10       rel="stylesheet"
11     />
12   </head>
13   <body>
14     <div id="app"></div>
15     <script type="module" src="/src/main.js"></script>
16     <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
17     <script
18       src="https://kit.fontawesome.com/0660ce38c6.js"
19       crossorigin="anonymous"
20     ></script>
21   </body>
22 </html>
```

Sl. 1.9 Uključivanje Bootstrap-a i Font Awesome-a u index.html

## 6. Dodatne konfiguracije

- Kako bi Vue 3 Toastify biblioteka bila uključena u aplikaciju, potrebno je izmeniti main.js i dodati odgovarajući kod koji je prikazan na slici 1.10.
- S obzirom da je potrebno da ova aplikacija konzumira Flask API koji se pokreće na <http://localhost:5000> URL-u (engl. *Uniform Resource Locator*), potrebno je izmeniti vite.config.js tako da se za slanje HTTP zahteva omogući korišćenje *"/api" alias-a* umesto <http://localhost:5000>. Izgled ove ismene prikazan je na slici 1.11.

```
2
3  import { createApp } from "vue";
4  import App from "./App.vue";
5  import Vue3Toastify from "vue3-toastify";
6  import "vue3-toastify/dist/index.css";
7  import router from "./router";
8
9  const app = createApp(App);
10
11  app.use(Vue3Toastify, {
12    autoClose: 3000,
13  });
14
15  app.use(router);
16
17  app.mount("#app");
```

Sl. 1.10 Izmena main.js fajla radi uključivanja Vue 3 Toastify biblioteke

```
JS vite.config.js > ...
1  import { fileURLToPath, URL } from "node:url";
2
3  import { defineConfig } from "vite";
4  import vue from "@vitejs/plugin-vue";
5
6  // https://vitejs.dev/config/
7  export default defineConfig({
8    plugins: [vue()],
9    server: {
10     proxy: {
11       "/api": {
12         target: "http://localhost:5000",
13         changeOrigin: true,
14         rewrite: (path) => path.replace(/^\/api/, ""),
15       },
16     },
17   },
18   resolve: {
19     alias: {
20       "@": fileURLToPath(new URL("./src", import.meta.url)),
21     },
22   },
23 });
```

Sl. 1.11 Izmena vite.config.js fajla radi korišćenja „/api” alias-a

## 2. Princip rada Vue.js-a

Vue.js je progresivni JavaScript framework za izradu korisničkih interfejsa *UI* (engl. *User Interface*). Radi tako što omogućava razdvajanje logike aplikacije, njenog prikaza i menadžmenta podataka u pojedinačne komponente. Vue koristi *Virtual DOM* (engl. *Virtual Document Object Model*), što znači da ažurira samo one delove korisničkog interfejsa koji se menjaju, čineći aplikaciju bržom i efikasnijom.

Vue.js aplikacija se sastoji od više komponenti. Svaka komponenta predstavlja jedan deo interfejsa i sadrži *HTML* (engl. *HyperText Markup Language*) za prikaz, JavaScript za logiku, i *CSS* (engl. *Cascading Style Sheets*) za stilizaciju. Komponente se međusobno povezuju, deleći podatke kroz "*props*" (za prosleđivanje podataka roditelj-deca) i komunicirajući kroz događaje. Vue.js koristi *data-binding* koji povezuje model podataka sa prikazom u realnom vremenu. To znači da kad se podaci promene, Vue automatski ažurira prikaz korisniku bez potrebe za ručnim manipulacijama DOM-om.

### 2.1 Glavne komponente i objašnjenje strukture u Vue.js projektu

1. `node_modules/`
  - Ovaj folder sadrži sve instalirane Node.js pakete i biblioteke koje aplikacija koristi. Ove biblioteke su navedene u `package.json` datoteci.
2. `public/`
  - Ovaj folder sadrži statičke datoteke koje će biti direktno dostupne na vebu (npr. slike, fontovi, `favicon.ico`). Vue ne vrši obradu ovih datoteka.
3. `src/` (Source folder)
  - Glavni folder za ceo kod. Ovde se nalazi aplikacija, njene komponente, rute, globalni stilovi itd.
4. `assets/`
  - Sadrži statičke resurse kao što su slike, CSS, ili fontovi koji će se obrađivati i optimizovati kroz build alat (Vite ili Vue *CLI* (engl. *Command Line Interface*)). Na primer, u ovom folderu se mogu čuvati slike logotipa, pozadinske slike ili fajlovi sa stilovima (`.css` ili `.scss`).
5. `components/`

Sadrži Vue komponente. Komponente su osnovne građevne jedinice aplikacije, a svaka komponenta može predstavljati deo korisničkog interfejsa (npr. dugme, tabela, kartica). Primer, moguće je imati fajlove `Navbar.vue`, `Sidebar.vue`, `Footer.vue`, koji čine različite delove korisničkog interfejsa.
6. `router/`
  - Sadrži fajl `index.js` povezan sa Vue Router-om. Vue Router je deo aplikacije koji omogućava upravljanje navigacijom između različitih delova aplikacije (komponenti). On omogućava kreiranje SPA, gde navigacija ne osvežava celu stranicu, već se dinamički menja sadržaj prikazan korisniku. U `router/index.js`, rute se definišu kao na slici 2.1:
    - `path`: URL putanja koja korisnika vodi do određene komponente.
    - `component`: Komponenta koja će biti prikazana kada se korisnik nađe na određenoj putanji.

```
import { createRouter, createWebHistory } from 'vue-router';
import Home from '../views/Home.vue';
import About from '../views/About.vue';

const routes = [
  {
    path: '/',
    name: 'Home',
    component: Home
  },
  {
    path: '/about',
    name: 'About',
    component: About
  }
];

const router = createRouter({
  history: createWebHistory(process.env.BASE_URL),
  routes
});

export default router;
```

Sl. 2.1 router/index.js

#### 7. store/

- Ovaj folder sadrži fajlove povezane sa Vuex-om, ako se koristi Vuex za globalno stanje aplikacije. Vuex je biblioteka za centralizovano upravljanje podacima (engl. *state management*) u Vue aplikacijama.

#### 8. views/

- Sadrži komponente koje predstavljaju različite "stranice" aplikacije (poznate kao pogleda). Svaka stranica može biti povezana sa specifičnom rutom. Primer: Home.vue, About.vue, Profile.vue. Ove komponente se obično prikazuju pomoću <RouterView> u glavnoj aplikaciji (kao što je App.vue).

#### 9. App.vue

- Ovo je koren (engl. *root*) komponenta Vue.js aplikacije. App.vue je obično kontejner za druge komponente i koristi se kao glavna komponenta u koju se učitavaju sve ostale komponente. Sadrži template (HTML), script (JavaScript), i style (CSS).
  - <template>: Sadrži HTML kod koji definiše kako će aplikacija biti prikazana.
  - <script>: Sadrži logiku aplikacije – može da upravlja stanjem, podacima, događajima itd.
  - <style>: Ovde se stilizuju elementi unutar komponente.
- U App.vue, sve što se stavi u <template> i <style> primenjuje se globalno na celu aplikaciju, primer je Navbar komponenta sa slike 2.2, koja će biti vidljiva na svim stranicama osim na login strani. RouterView služi za dinamičko prikazivanje

komponenta u zavisnosti od aktivne rute, omogućavajući da se samo sadržaj unutar RouterView menja dok ostatak aplikacije ostaje isti.

```
▼ App.vue > ...
<script setup>
import { RouterLink, RouterView } from "vue-router";
import Navbar from "../components/Navbar.vue";
</script>

<template>
  <div>
    <Navbar v-if="$route.name !== 'Login'"></Navbar>
    <RouterView></RouterView>
  </div>
</template>

<style>
  /* Stilovi za aplikaciju */
</style>
```

Sl. 2.2 App.vue

10. main.js

- main.js je ulazna tačka (engl. *entry point*) aplikacije. On inicijalizuje Vue instancu i povezuje aplikaciju sa HTML dokumentom (obično sa elementom koji ima id="app"). Takođe može da učitava globalne stilove, rute i druge osnovne konfiguracije. Na slici 2.3 prikazan je primer main.js fajla.

```
import { createApp } from 'vue';
import App from './App.vue';
import router from './router';

createApp(App).use(router).mount('#app');
```

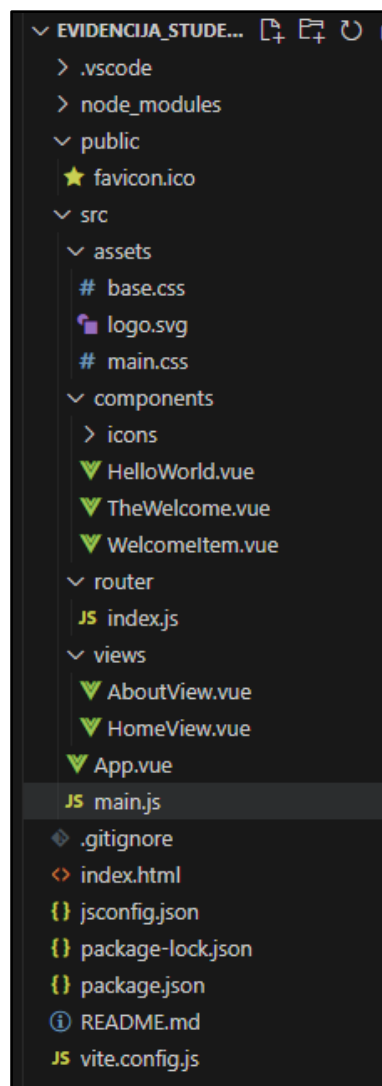
Sl. 2.3 main.js

11. index.html

- Ovo je statički HTML fajl koji služi kao polazna tačka aplikacije. On sadrži <div> element koji služi kao „sidro“ za Vue aplikaciju. Vue uzima ovaj statički sadržaj i dinamički ga menja koristeći JavaScript, prikazujući komponente aplikacije.
- Kao na slici 2.4, Vue aplikacija je vezana za element sa id="app". Sve komponente i prikaz će biti renderovani unutar ovog div elementa.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Vue App</title>
  </head>
  <body>
    <div id="app"></div>
    <script src="/main.js"></script>
  </body>
</html>
```

Sl. 2.4 index.html



Sl. 2.5 Prikaz strukture Vue.js projekta

### 3. Klijent server arhitektura

Klijent-server arhitektura je model komunikacije između dva entiteta: klijenta i servera. U ovom modelu, klijent (koji je obično veb pretraživač ili mobilna aplikacija) šalje zahteve serveru, a server obrađuje te zahteve i vraća odgovore.

U kontekstu REST API-ja, ovaj model se koristi za razmenu podataka između klijenta i servera. Klijent može da bude frontend aplikacija (npr. napisano u Vue.js ili React), dok server (backend) može biti implementiran u nekom jeziku kao što je .NET Core (C#), Flask (Python), Laravel (PHP), itd...

- Frontend: Deo aplikacije koji je vidljiv korisnicima i koji oni koriste direktno. Frontend se obično razvija koristeći HTML, CSS, i JavaScript okvire kao što su Vue.js ili React. Frontend šalje zahteve ka API-ju kako bi dobio ili poslao podatke sa servera.
- Backend: Deo aplikacije koji se izvršava na serveru, i koji upravlja obradom podataka, autentifikacijom, validacijom i komunikacijom sa bazom podataka. Backend je zadužen za primanje zahteva od klijenta (preko API-ja), obradu tih zahteva i slanje odgovarajućih podataka nazad na frontend.

#### 3.1 Pojam API-ja

API je interfejs koji omogućava različitim aplikacijama da komuniciraju međusobno. REST API je specifičan stil arhitekture za izgradnju API-ja, gde se komunicira putem HTTP metoda (GET, POST, PUT, DELETE). API omogućava klijentu da vrši različite akcije kao što su dobijanje podataka sa servera, slanje novih podataka, ažuriranje ili brisanje postojećih podataka.

Proces dobijanja podataka sa servera:

1. Zahtev od klijenta: Kada korisnik pokrene neku akciju na frontend aplikaciji (npr. klikne na dugme da vidi listu proizvoda), frontend šalje HTTP zahtev ka REST API-ju na serveru.
2. Obrada zahteva na serveru: Server prima zahtev i na osnovu rute i metode (GET, POST, itd.) obrađuje taj zahtev. Na primer, ako klijent zahteva listu proizvoda, server će poslati upit ka bazi podataka, dobiti tražene podatke i pripremiti ih u JSON formatu.
3. Odgovor sa servera: Nakon obrade zahteva, server vraća HTTP odgovor klijentu koji sadrži tražene podatke u formatu koji frontend može da interpretira (najčešće JSON).
4. Prikaz na frontendu: Frontend zatim koristi dobijene podatke da ih prikaže korisniku (npr. renderuje listu proizvoda na stranici).

Primer:

1. Klijent: Korisnik klikne na dugme da vidi sve proizvode u aplikaciji.
2. HTTP zahtev: Frontend (npr. Vue.js aplikacija) šalje HTTP GET zahtev ka ruti API-ja, npr. /products.
3. Server: Backend (npr. Flask API) prima zahtev na ruti /products, preuzima podatke iz baze i vraća ih u JSON (engl. *JavaScript Object Notation*) formatu.
4. HTTP odgovor: Server vraća odgovor koji sadrži JSON listu svih proizvoda.
5. Prikaz na frontendu: Frontend prima odgovor i prikazuje listu proizvoda korisniku.

```
# Jednostavna Flask ruta koja vraća listu proizvoda
from flask import Flask, jsonify

app = Flask(__name__)

# Simulirani podaci
products = [
    {"id": 1, "name": "Proizvod 1", "price": 100},
    {"id": 2, "name": "Proizvod 2", "price": 200},
]

# Ruta za vraćanje liste proizvoda
@app.route('/products', methods=['GET'])
def get_products():
    # jsonify funkcija koja konvertuje izlaz u JSON response objekat
    return jsonify(products)

if __name__ == '__main__':
    app.run(debug=True)
```

**app.py**

```
/* Frontend kod u Vue.js-u koji koristi Axios za slanje GET zahteva Flask
API-ju i prikazivanje podataka */

<script setup>
import { ref, onMounted } from 'vue';
import axios from 'axios';
import { toast } from 'vue3-toastify';

const products = ref([]); /* Ref (reaktivna promenljiva) za čuvanje liste
proizvoda */

const fetchProducts = async () => {
  try {
    // Poziv Flask API-ja
    const response = await axios.get('http://localhost:5000/products');
    products.value = response.data; // Čuvanje podataka u products ref
  } catch (error) {
    toast.error('Greška pri preuzimanju proizvoda: ' + error.message);
  }
};

onMounted(() => {
  /* Poziv API-ja kada se komponenta mount-uje
  (Dodavanje HTML elemenata koji odgovaraju ovoj komponenti u DOM) */
  fetchProducts();
});
</script>
```



```
<template>
  <div>
    <h1>Lista proizvoda</h1>
    <ul>
      <li v-for="product in products" :key="product.id">
        {{ product.name }} - Cena: {{ product.price }} RSD
      </li>
    </ul>
  </div>
</template>

<style>
/* Stilovi po želji */
</style>
```

#### ProductsPrimer.vue

```
/* Registrovanje /products rute kako bi se na njoj u veb pregledaču učitala
ProductsPrimer.vue komponenta */
import { createRouter, createWebHistory } from "vue-router";
import ProductsPrimer from "@views/ProductsPrimer.vue";

const routes = [
  {
    path: "/products",
    name: "Products",
    component: ProductsPrimer,
  },
];

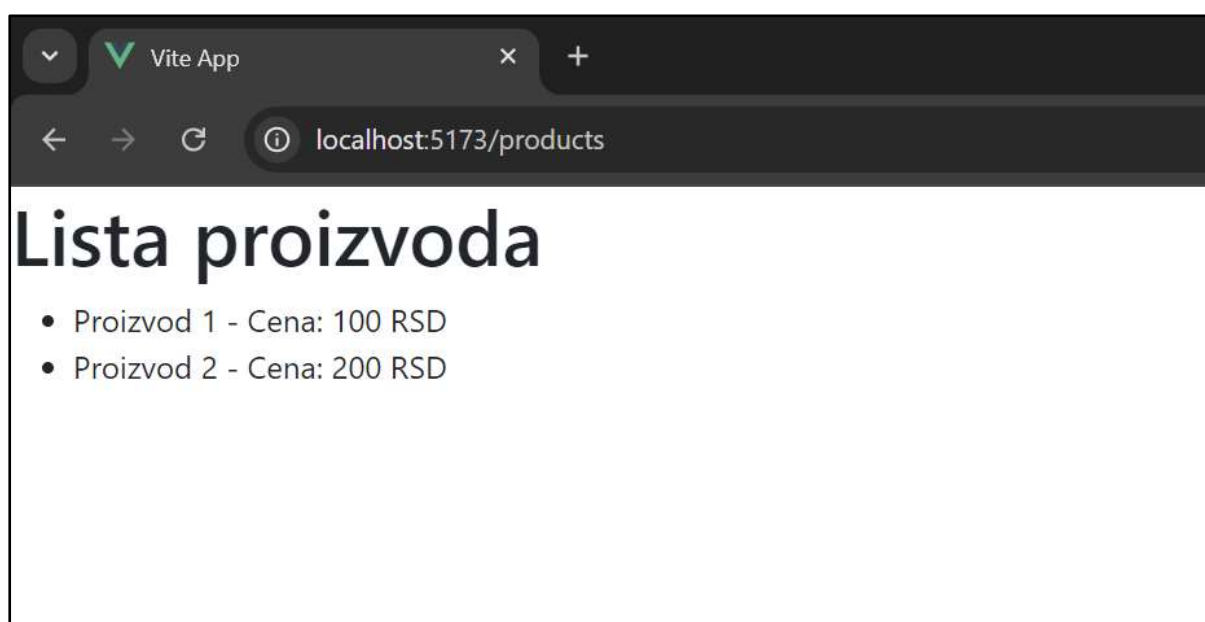
const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes,
});

export default router;
```

#### router/index.js

Pojašnjenje:

- Axios GET zahtev: Kada se komponenta mount-uje (tj. kada se učitava), Vue.js šalje GET zahtev ka ruti /products Flask API-ja koristeći `axios.get`. U ovom slučaju, URL je "http://localhost:5000/products", jer server radi lokalno.
- Odgovor API-ja: Ako je zahtev uspešan, odgovor (`response.data`) sadrži listu proizvoda u JSON formatu, koji se zatim čuva u `products` reaktivnoj promenljivoj u Vue.js komponenti.
- Renderovanje podataka: U template delu, Vue koristi `v-for` direktivu da iterira kroz listu `products` i prikaže svaki proizvod u `<li>` elementu sa informacijama o imenu i ceni.
- Na slici 3.1 prikazana je stranica u veb pregledaču nakon što se pristupi ruti /products, odnosno "http://localhost:5173/products"



Sl. 3.1 Pristupanje /products ruti koja render-uje `ProductsPrimer.vue`

### 3.2 Pojam Autentifikacije

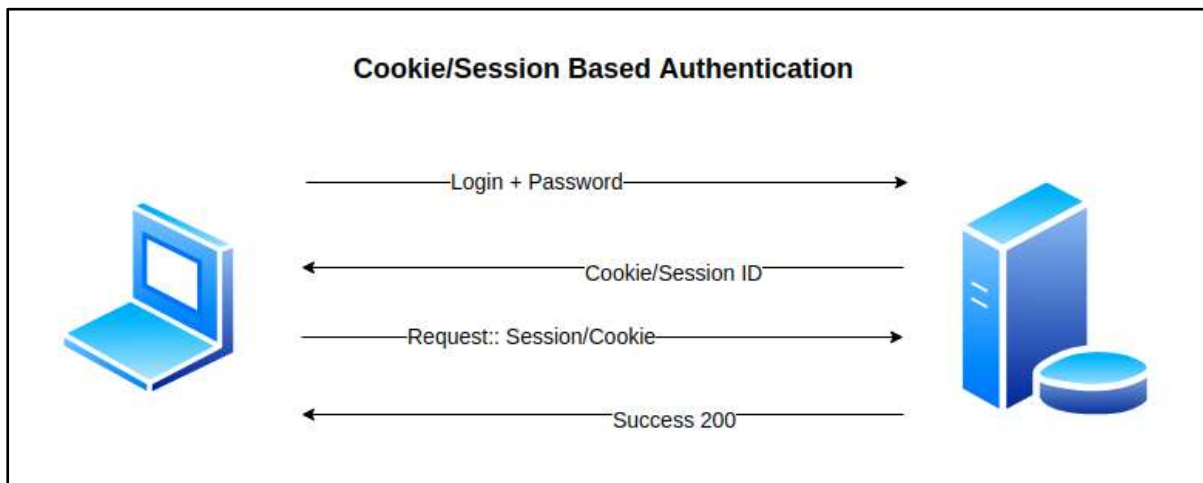
Autentifikacija je proces kojim se proverava identitet korisnika ili sistema koji pokušava da pristupi određenom resursu. U suštini, autentifikacija odgovara na pitanje: "Ko si ti?" Na primer, kada korisnik unosi korisničko ime i lozinku na login formi, API ili server proverava da li su te informacije tačne i na osnovu toga dozvoljava ili odbija pristup.

#### 3.2.1 Autentifikacija kolačićima (engl. Cookie authentication)

Cookie autentifikacija je tradicionalna metoda autentifikacije gde, nakon uspešne prijave korisnika, server kreira sesiju (engl. *session*) i povezuje je sa korisnikom. Server zatim šalje cookie nazad klijentu (obično veb pregledaču), koji čuva cookie i koristi ga u svim narednim zahtevima prema serveru.

Na slici 3.2 prikazano je funkcionisanje cookie autentifikacije:

1. Prijava korisnika: Korisnik unosi svoje kredencijale (korisničko ime i lozinku).
2. Validacija na serveru: Server proverava da li su kredencijali ispravni.
3. Kreiranje sesije: Ako su kredencijali ispravni, server kreira sesiju na serveru i povezuje je sa korisnikom.
4. Slanje cookie-a: Server šalje klijentu cookie koji sadrži ID sesije.
5. Svaki naredni zahtev: Klijent automatski šalje cookie sa ID-jem sesije u zaglavlju svakog narednog zahteva. Server preko ID-ja pronalazi sesiju i identifikuje korisnika.



Sl. 3.2 Funkcionisanje cookie autentifikacije

Problem sa cookie autentifikacijom:

1. Pamćenje stanja (engl. *stateful*): Server mora da pamti sve aktivne sesije, što znači da mora da skladišti informacije o prijavljenim korisnicima. Ovo može postati problematično u distribuiranim sistemima ili kada postoji više servera.
2. Pitanja skalabilnosti: Kako bi sistem bio skalabilan, sesije se često moraju skladištiti u zajedničkoj bazi podataka ili memori, što povećava složenost.

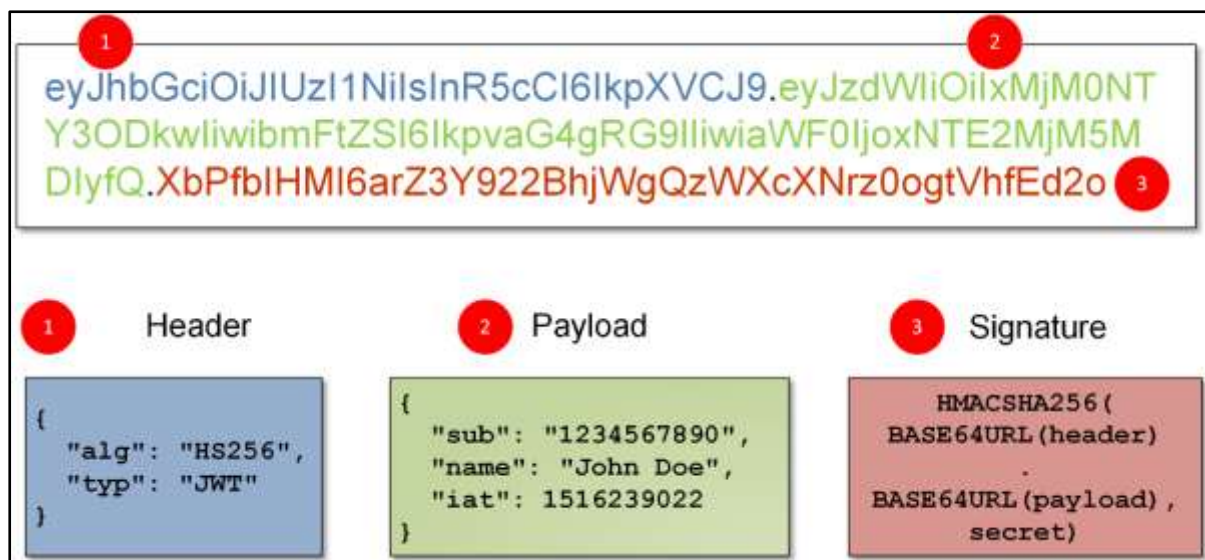
### 3.2.2 JWT Autentifikacija (engl. JSON Web Token authentication)

JWT autentifikacija koristi metodu autentifikacije bez pamćenja stanja (engl. *stateless*). Umesto da se server oslanja na sesije, JWT koristi token koji se generiše na serveru i šalje klijentu nakon prijave. Klijent zatim šalje ovaj token u svakom narednom zahtevu, a server koristi token za verifikaciju korisnika.

Token je mali digitalni token (u ovom slučaju JWT) koji sadrži korisničke podatke i koristi se za autentifikaciju. Token je obično šifrovan i potpisan, što znači da je siguran od izmena.

Slika 3.3 prikazuje komponente JWT tokena:

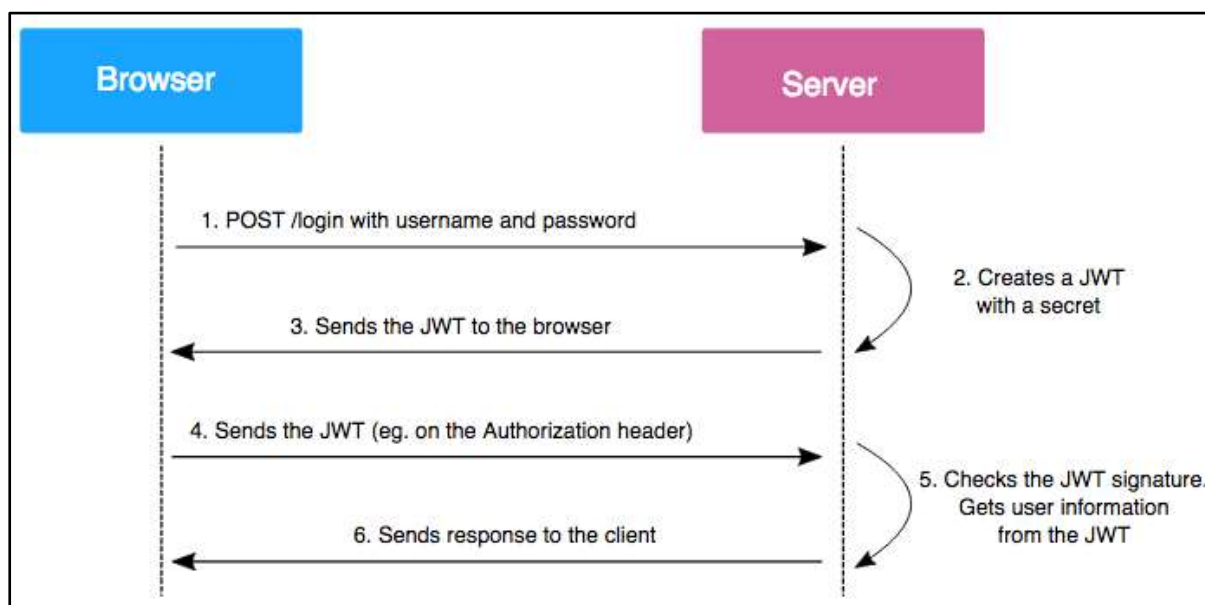
1. Header (zaglavlje): Sadrži tip tokena i algoritam za potpisivanje (npr. HS256).
2. Payload (telo): Sadrži korisničke podatke, kao što su korisnički ID, uloga, i druge informacije koje server može koristiti.
3. Signature (potpis): Obezbeđuje integritet tokena i koristi se da se osigura da token nije izmenjen nakon što je izdat.



Sl. 3.3 Komponente JWT tokena

Slika 3.4 prikazuje funkcionisanje JWT autentifikacije:

1. Prijava korisnika: Korisnik unosi svoje kredencijale.
2. Generisanje tokena: Server proverava kredencijale, i ako su tačni, kreira JWT token koji sadrži informacije o korisniku (kao što su korisnički ID, uloga, i sl.). Ovaj token je šifrovan i potpisan.
3. Slanje tokena: JWT token se šalje klijentu.
4. Svaki naredni zahtev: Klijent u svakom narednom zahtevu šalje JWT token u Authorization zaglavlju (obično sa "Bearer" prefiksom).
5. Verifikacija na serveru: Server dešifrje token, proverava njegov integritet (da li je validan i neizmenjen), i koristi podatke iz tokena da identifikuje korisnika i proveriti njegova prava pristupa.



Sl. 3.4 Funkcionisanje JWT autentifikacije

Prednosti korišćenja tokena:

1. Stateless (bez potrebe za skladištenjem sesija): Server ne mora da čuva informacije o prijavljenim korisnicima jer se svi potrebni podaci nalaze unutar tokena. Ovo olakšava skalabilnost i smanjuje opterećenje servera.
2. Lakša skalabilnost: Sa JWT, serveri ne moraju da dele informacije o sesijama između sebe, što omogućava lakše skaliranje aplikacije (npr. distribuciju opterećenja na više servera).
3. Bezbednost: JWT tokeni su šifrovani i potpisani, što znači da su podaci unutar njih bezbedni. Takođe, server može lako da proveri da li je token izmenjen.
4. Jednostavna autentifikacija u mikroservisima: JWT tokeni omogućavaju jednostavnu autentifikaciju u sistemima sa više mikroservisa jer svaki servis može da proveri token bez potrebe za pristupom centralnoj sesiji.
5. Fleksibilnost: Tokeni mogu sadržati dodatne informacije o korisniku (npr. prava pristupa), čime se omogućava lakša autorizacija bez potrebe za dodatnim upitima ka bazi podataka.

## 4. Izrada aplikacije

### 4.1 Izgled aplikacije, stranice i rute

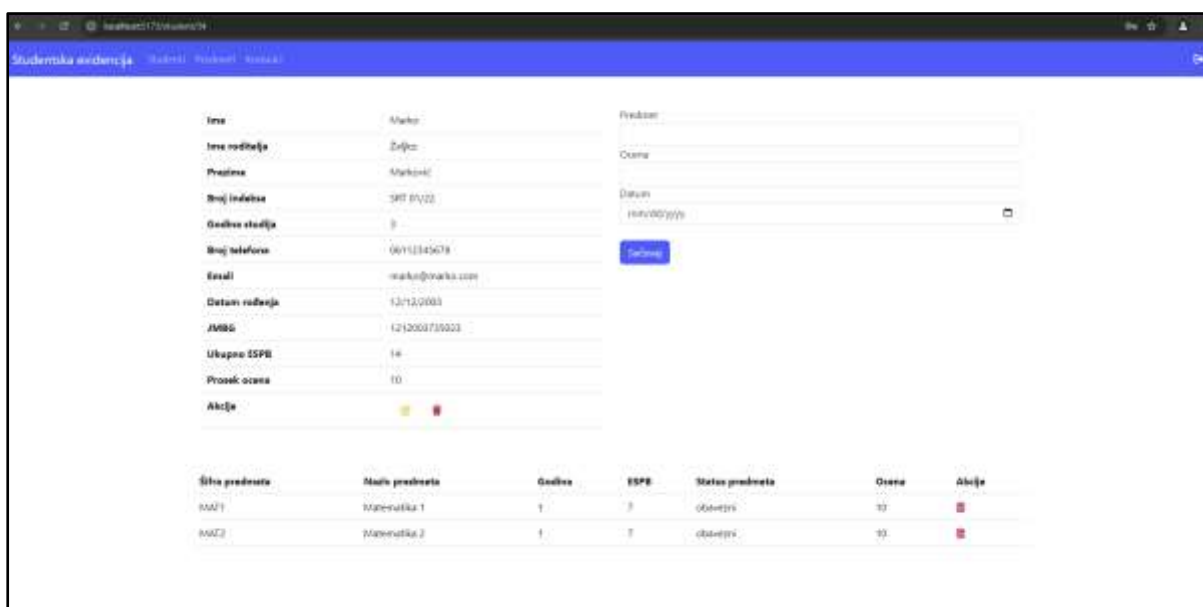
Na slikama 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 4.10 i 4.11 nalaze se stranice koje čine kompletnu aplikaciju.



Sl. 4.1 Login



Sl. 4.2 Studenti



Sl. 4.3 Student

Sl. 4.4 Izmena studenta

Sl. 4.5 Dodavanje novog studenta

Šifra	Naziv	Godina studija	EAPB	Obavezni / Izborni	Ocijena
SMAT2	Matematika 2	1	3	obavezni	2
SMAT1	Matematika 1	1	3	obavezni	2
KSI1	Klijent server sistemi	3	0	obavezni	2
VEB1	Web programiranje	3	0	obavezni	2
NET1	.NET tehnologije	3	0	obavezni	2

Sl. 4.6 Predmeti

Izmena predmeta

Slike

Naziv

Godina studija

EIPB

Olasnost / izdani

Sačuvaj

Sl. 4.7 Izmena predmeta

Novi predmet

Slike

Naziv

Godina studija




EIPB

Olasnost / izdani

Sačuvaj

Sl. 4.8 Dodavanje novog predmeta

Dodaj korisnika





Ime	Prezime	Email	Rola	Akcija
Petar	Petrović	test@test.com	profesor	 
Administrator	Administrator	test2@test.com	administrator	 

Sl. 4.9 Korisnici

Potvrda brisanja

Da li ste sigurni da želite da obrisete korisnika?

Otvori Otvori

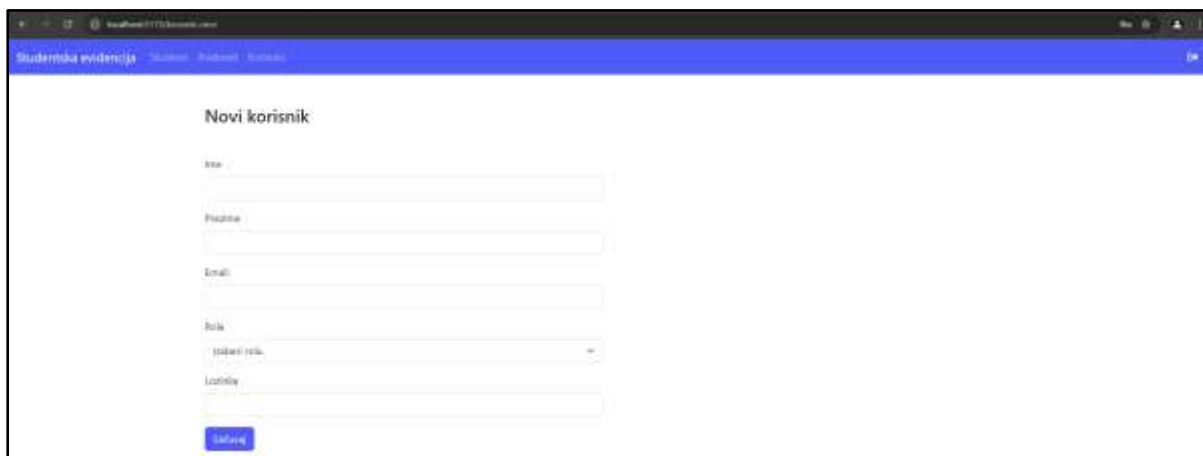
Ime	Prezime	Email	Rola	Akcija
Petar	Petrović	test@test.com	profesor	 
Administrator	Administrator	test2@test.com	administrator	 

Sl. 4.10 Modal za brisanje entiteta





Sl. 4.11 Izmena korisnika



Sl. 4.12 Dodavanje novog korisnika

## 4.2 Razdvajanje u komponente

Na gorenavedenim slikama moguće je primetiti neka ponavljanja i sličnosti na svim stranicama. Konkretno, navigacioni meni se javlja na svakoj stranici, osim na login-u. Takođe, forme za izmenu i dodavanje novih entiteta (studenata, predmeta i korisnika) imaju isti stil i izgled, a isto važi i za prikazivanje entiteta (studenata, predmeta i korisnika) u tabele. Što se tiče brisanja entiteta, za svaki iskače isti modal za potvrdu brisanja prikazan na slici 4.10.

1. Imajući ovo u vidu, poželjno je kreirati *reusable* komponente (components), koje će se koristiti na stranicama, tj. u pogledima (views) po potrebi, a to bi bile:
2. Navbar.vue – navigacioni meni
3. FormContainer.vue - <div> element, kontejner sa određenim stilovima za forme
4. TableContainer.vue - <div> element, kontejner sa određenim stilovima za tabele
5. DeleteModal.vue – potvrdni modal koji će iskakati kada se pokrene akcija brisanja

## 5. Kreiranje komponenti i stranice za login

### 5.1 views/Login.vue

Potrebno je u folder views kreirati novi fajl i nazvati ga Login.vue. Predstavljaje login stranicu.

```
<script setup>
import { ref } from "vue";
import axios from "axios";
import { useRouter } from "vue-router";
import { toast } from "vue3-toastify";

const email = ref("");
const lozinka = ref("");
const router = useRouter();

const handleSubmit = async (event) => {
  event.preventDefault();
  try {
    const response = await axios.post("/api/login", {
      email: email.value,
      lozinka: lozinka.value,
    });

    localStorage.setItem("access_token", response.data.access_token);
    localStorage.setItem("rola", response.data.rola);

    router.push("/studenti").then(() =>
toast.success(response.data.message));
  } catch (error) {
    toast.error(error.response.data.message || error.message);
  }
};
</script>

<template>
<div class="container">
  <div class="row justify-content-center my-5">
    
  </div>
  <div class="row justify-content-center">
    <div class="col-lg-6 col-xs-12">
      <form @submit="handleSubmit">
        <div class="form-group mb-3">
          <label>Email adresa</label>
          <input type="email" v-model="email" class="form-control" required
/>
```

```
</div>
<div class="form-group mb-3">
  <label>Lozinka</label>
  <input
    type="password"
    v-model="lozinka"
    class="form-control"
    required
  />
</div>
<input
  type="submit"
  class="btn btn-primary"
  role="button"
  value="Prijava se"
/>
</form>
</div>
</div>
</div>
</template>

<style scoped>
.logo {
  width: 20rem;
  height: auto;
}
</style>
```

Login.vue izvorni kod

## 1. Script setup

```
<script setup>
```

Ovo je specijalan način za pisanje JavaScript-a u Vue 3. Sve što se napiše unutar <script setup> direktno je dostupno u Vue.js komponenti. Ovo pojednostavljuje kod i omogućava korišćenje promenljiva i funkcija bez dodatnih deklaracija.

## 2. Importovanje

```
import { ref } from "vue";
import axios from "axios";
import { useRouter } from "vue-router";
```

```
import { toast } from "vue3-toastify";
```

Vue omogućava modularnost, pa je moguće uvoziti delove koda iz drugih fajlova. U ovom slučaju:

- **ref** je funkcija iz Vue.js koja omogućava kreiranje reaktivnih promenljiva. Reaktivne promenljive u Vue.js automatski ažuriraju prikaz kada se promene
- **useRouter** omogućava navigaciju između stranica unutar aplikacije.
- **toast** dolazi iz **vue3-toastify** i koristi se za prikazivanje malih obaveštenja (npr. poruke o uspešnoj odjavi).
- **Axios** je komponenta koja se koristi za slanje HTTP zahteva (u ovom slučaju potrebno je poslati POST zahtev za login)

### 3. Promenljive

```
const email = ref("");  
const lozinka = ref("");  
const router = useRouter();
```

- **ref**: funkcija iz Vue.js koja omogućava kreiranje reaktivnih promenljiva. Reaktivne promenljive (u ovom slučaju **email** i **lozinka**) u Vue.js automatski ažuriraju prikaz kada se promene. Primer, promenljiva **email** je definisana pomoću **ref("")**, što znači da je inicijalno postavljena na prazan string, a kasnije može biti ažurirana. Kada se vrednost **email** promeni, sve komponente koje zavise od ove vrednosti će automatski biti osvežene.
- **router**: instanca **router**-a koja se koristi za navigaciju.

### 4. **handleSubmit** funkcija

```
const handleSubmit = async (event) => {  
  event.preventDefault();  
  try {  
    const response = await axios.post("/api/login", {  
      email: email.value,  
      lozinka: lozinka.value,  
    });  
  
    localStorage.setItem("access_token", response.data.access_token);  
    localStorage.setItem("rola", response.data.rola);  
  
    router.push("/studenti").then(() =>  
      toast.success(response.data.message));  
  } catch (error) {  
    toast.error(error.response.data.message || error.message);  
  }  
}
```

```
}  
};
```

- `handleSubmit` je funkcija koja se poziva kada se forma podnese:

```
<form @submit="handleSubmit">
```

- `event.preventDefault()` sprečava podrazumevano ponašanje forme (npr. osvežavanje stranice).
- `axios.post` šalje POST zahtev na `/api/login` sa email i lozinka vrednostima.
- Ako je zahtev uspešan:
  - `localStorage.setItem` čuva `access_token` i rola u `localStorage`.
  - `router.push` preusmerava korisnika na `/studenti` rutu i prikazuje uspešnu notifikaciju.
- Ako dođe do greške, prikazuje grešku koristeći toast.

## 5. Slika logotipa

```

```

Potrebno je u `src/assets/img` folder ubaciti sliku pod nazivom `logo.png`, a ovaj isečak koda omogućava prikazivanje slike na vrhu forme. Slika je stilizovana klasom `.logo` u `<style>` komponenti:

```
<style scoped>  
.logo {  
  width: 20rem;  
  height: auto;  
}  
</style>
```

## 6. Input polja

```
<input type="email" v-model="email" class="form-control" required />
```

```
<input type="password" v-model="lozinka" class="form-control" required/>
```

`v-model` direktiva omogućava tzv. dvostruko vezivanje (engl. *Two-way binding*). **Dvostruko vezivanje** znači da promena vrednosti u input polju automatski ažurira povezanu promenljivu, i obrnuto, promena vrednosti promenljive automatski ažurira vrednost u input polju.

Način rada na konkretnom primeru:

- Deklaracija reaktivnih promenljivih:

- U JavaScript delu komponente, odnosno `<script setup>` deklarisanе su dve reaktivne promenljive (email i lozinka) koristeći ref:

```
const email = ref("");
const lozinka = ref("");
```

- email: Čuva vrednost unetu u polje za email.
- lozinka: Čuva vrednost unetu u polje za lozinku.
- Vezivanje input polja za promenljive
  - U HTML delu komponente, koristi se `v-model` direktiva radi vezivanja input polja za ove promenljive. `V-model="email"` vezuje vrednost input polja za promenljivu email, dok `v-model="lozinka"` vezuje vrednost input polja za promenljivu lozinka.
- Primer: ako korisnik unese "user@example.com" u input polje za email:

```
<input type="email" v-model="email" class="form-control" required />
```

- Promenljiva `email.value` postaje "user@example.com"
- Ako se kasnije u kodu promeni `email.value` na "newuser@example.com", input polje će automatski prikazati "newuser@example.com".
- Ovo olakšava rad sa formama i osigurava da su podaci uvek sinhronizovani između korisničkog interfejsa i logike aplikacije.

## 7. Registrovanje /login rute

Potrebno je u fajlu `router/index.js` registrovati rutu za login. Kada se pristupi URL-u: <http://localhost:5173/login>, učitaje se gorenavedena `Login.vue` komponenta.

```
import { createRouter, createWebHistory } from "vue-router";
import Login from "@views/Login.vue";

const routes = [
  {
    path: "/login",
    name: "Login",
    component: Login,
  },
];

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes,
});

export default router;
```

## router/index.js izvorni kod

### 8. Zahtevanje prijave (login) u aplikaciju na svakih 30 minuta

Potrebno je u fajlu `main.js` dodati funkciju `setInterval` koja će brisati ceo sadržaj, odnosno podatke o roli i tokenu iz `localStorage` na svakih 30 minuta. Ova funkcija, radi sigurnosti i bezbenosti, obezbeđuje da korisnik mora da se prijavi (login) na svakih pola sata.

```
import { createApp } from "vue";
import App from "./App.vue";
import Vue3Toastify from "vue3-toastify";
import "vue3-toastify/dist/index.css";
import router from "./router";
import axios from "axios";

setInterval(() => {
  localStorage.clear();
}, 30 * 60 * 1000);

const app = createApp(App);

app.use(Vue3Toastify, {
  autoClose: 3000,
});

app.use(router);

app.mount("#app");
```

## main.js izvorni kod

### 9. Slanje JWT tokena uz svaki HTTP zahtev

Potrebno je u fajlu `main.js` dodati kod za konfiguraciju Axios-a. Dodata funkcija koristi Axios interceptore za dodavanje Authorization zaglavlja sa JWT tokenom u svaki HTTP zahtev koji se šalje sa klijenta.

```
import { createApp } from "vue";
import App from "./App.vue";
import Vue3Toastify from "vue3-toastify";
import "vue3-toastify/dist/index.css";
import router from "./router";
import axios from "axios";
```

```
setInterval(() => {
  localStorage.clear();
}, 30 * 60 * 1000);

axios.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem("access_token");
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  (error) => {
    return Promise.reject(error);
  }
);

const app = createApp(App);

app.use(Vue3Toastify, {
  autoClose: 3000,
});

app.use(router);

app.mount("#app");
```

main.js izvorni kod

## 5.2 components/Navbar.vue

Potrebno je u folder components kreirati novi fajl i nazvati ga Navbar.vue.

```
<script setup>
import { RouterLink, useRoute } from "vue-router";
import { useRouter } from "vue-router";
import { useUserRole } from "@/composables/useUserRole.js";
import { toast } from "vue3-toastify";
import axios from "axios";
import { onMounted } from "vue";

const router = useRouter();
const { userRole, checkUserRole } = useUserRole();

const isActiveLink = (routePath) => {
  const route = useRoute();
```



```

    return route.path === routePath;
  };

  onMounted(() => {
    checkUserRole();
  });

  const handleLogout = async () => {
    try {
      localStorage.removeItem("access_token");
      router.push("/login").then(() => toast.success("Uspešno ste se odjavili."));
    } catch (error) {
      toast.error(error.response.data.message || error.message);
    }
  };
</script>

<template>
  <nav class="navbar navbar-expand-lg navbar-dark bg-primary">
    <div class="container-fluid">
      <RouterLink class="navbar-brand" to="/studenti">
        Studentska evidencija</RouterLink>
      <button
        class="navbar-toggler"
        type="button"
        data-bs-toggle="collapse"
        data-bs-target="#navbarSupportedContent"
        aria-controls="navbarSupportedContent"
        aria-expanded="false"
        aria-label="Toggle navigation"
      >
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav me-auto mb-2 mb-lg-0">
          <li :class="{ 'nav-item': true, active: isActiveLink('/studenti')
            <RouterLink class="nav-link"
to="/studenti">Studenti</RouterLink>
            </li>
            <li
              v-if="userRole === 'administrator'"
              :class="{ 'nav-item': true, active: isActiveLink('/predmeti')
            </li>
          </ul>
        </div>
      </div>
    </nav>
  </template>

```

```

<RouterLink class="nav-link"
to="/predmeti">Predmeti</RouterLink>
  </li>
  <li
    v-if="userRole === 'administrator'"
    :class="{ 'nav-item': true, active: isActiveLink('/korisnici')
}"
  >
    <RouterLink class="nav-link"
to="/korisnici">Korisnici</RouterLink>
    </li>
  </ul>
  <form class="d-flex" @submit.prevent="handleLogout">
    <button type="submit" class="btn btn-primary">
      <i class="fas fa-sign-out-alt"></i>
    </button>
  </form>
</div>
</div>
</nav>
</template>

```

Navbar.vue izvorni kod

## 1. Importovanje

```

import { RouterLink, useRoute } from "vue-router";
import { useRouter } from "vue-router";
import { useUserRole } from "@/composables/useUserRole.js";
import { toast } from "vue3-toastify";
import { onMounted } from "vue";

```

Vue omogućava modularnost, pa je moguće uvoziti delove koda iz drugih fajlova. U ovom slučaju:

- RouterLink i useRoute dolaze iz vue-router, a to je alat koji pomaže pri kreiranju ruta (putanja) unutar aplikacije.
- useRouter omogućava navigaciju između stranica unutar aplikacije.
- useUserRole je funkcija napisana u posebnom fajlu i koristi se da proveriti koja je uloga (role) korisnika.
- toast dolazi iz vue3-toastify i koristi se za prikazivanje malih obaveštenja (npr. poruke o uspešnoj odjavi).
- onMounted je specijalna funkcija iz Vue.js-a koja omogućava da izvršimo kod čim se komponenta prikaže na ekranu.

## 2. useUserRole composable funkcija

Potrebno je kreirati folder composables unutar src foldera, a zatim u njemu kreirati fajl useUserRole.js:

```
import { ref } from "vue";

export function useUserRole() {
  const userRole = ref(null);

  const checkUserRole = () => {
    const role = localStorage.getItem("rola");
    userRole.value = role;
  };

  return {
    userRole,
    checkUserRole,
  };
}
```

Ovaj kod je jednostavna Vue.js kompozibilna funkcija koja služi za rad sa korisničkom rolom, koristeći ref i funkciju za proveru role sačuvane u localStorage:

- **ref:** funkcija iz Vue.js koja omogućava kreiranje reaktivnih promenljiva. Reaktivne promenljive u Vue.js automatski ažuriraju prikaz kada se promene. U ovom slučaju, promenljiva userRole je definisana pomoću ref(null), što znači da je inicijalno postavljena na null, a kasnije može biti ažurirana. Kada se vrednost userRole promeni, sve komponente koje zavise od ove vrednosti će automatski biti osvežene.
- **checkUserRole:** funkcija koja služi za proveru korisničke role. Unutar nje se koristi localStorage.getItem("rola") kako bi se dobila vrednost sačuvana pod ključem "rola" u lokalnoj memoriji pregledača. Kada se ova vrednost preuzme, dodeljuje se reaktivnoj promenljivoj userRole putem userRole.value = role. Na taj način se reaktivna vrednost ažurira.
- **export function useUserRole():** omogućavanje korišćenje ove funkcije u drugim delovima aplikacije. Kada se funkcija useUserRole izveze, ona može biti uvezena i korišćena u različitim komponentama, konkretno ovde u Navbar.vue
- **return:** u Vue.js kompozibilnim funkcijama, return vraća objekte ili funkcije koje su dostupne komponentama koje koriste ovu kompozibilnu funkciju. U ovom slučaju, vraćaju se userRole i checkUserRole, omogućavajući komponentama pristup reaktivnoj promenljivoj userRole i funkciji checkUserRole.

### 3. onMounted funkcija

```
onMounted(() => {  
  checkUserRole();  
});
```

onMounted funkcija se koristi da izvrši određeni kod kada se komponenta učita. U ovom slučaju, kad se stranica učita, poziva se funkcija checkUserRole koja proverava koju ulogu ima korisnik (administrator ili profesor).

### 4. Logika za proveru aktivnog linka

```
const isActiveLink = (routePath) => {  
  const route = useRoute();  
  return route.path === routePath;  
};
```

Funkcija isActiveLink proverava da li je trenutni link na kojem se korisnik nalazi aktivan. Na primer, ako je korisnik na stranici "Studenti", ova funkcija će vratiti true za tu putanju, a false za sve ostale. Na ovaj način može se stilizovati aktivni link da se ističe u meniju:

```
<li :class="{ 'nav-item': true, active: isActiveLink('/studenti') }">  
  <RouterLink class="nav-link" to="/studenti">Studenti</RouterLink>  
</li>
```

Direktiva :class u Vue.js omogućava dinamičko dodavanje klasa HTML elementima. U ovom primeru, koristi se za dodavanje klasa nav-item i active elementu <li> na osnovu određenih uslova.

- 'nav-item': true: Ovo znači da će klasa nav-item uvek biti dodata elementu <li>.
- active: isActiveLink('/studenti'): Ovo znači da će klasa active biti dodata elementu <li> samo ako funkcija isActiveLink('/studenti') vrati true.

### 5. Odjava korisnika

```
const handleLogout = async () => {  
  try {  
    localStorage.removeItem("access_token");  
    router.push("/login").then(() => toast.success("Uspešno ste se  
odjavili."));  
  } catch (error) {  
    toast.error(error.response.data.message || error.message);  
  }  
}
```

```
};
```

Ova funkcija se koristi za odjavu korisnika. Kada korisnik klikne na dugme za odjavu, funkcija briše njegov token za pristup iz lokalne memorije i preusmerava ga na stranicu za prijavu. Takođe, prikazuje se obaveštenje da je korisnik uspešno odjavljen. Ovako izgleda dugme za odjavu:

```
<form class="d-flex" @submit.prevent="handleLogout">
  <button type="submit" class="btn btn-primary">
    <i class="fas fa-sign-out-alt"></i>
  </button>
</form>
```

@submit.prevent je direktiva koja sprečava podrazumevano ponašanje forme (npr. osvežavanje stranice) i izvršava navedenu funkciju.

## 6. Rutiranje (navigacija između stranica)

```
<RouterLink class="nav-link" to="/studenti">Studenti</RouterLink>
```

RouterLink je specijalan element iz vue-router paketa koji omogućava kreiranje linkova između različitih stranica u aplikaciji. U ovom primeru, klikom na "Studenti", korisnik se prebacuje na stranicu sa spiskom studenata.

## 7. v-if direktiva

```
<li v-if="userRole === 'administrator'">
```

v-if je Vue direktiva koja omogućava prikazivanje ili sakrivanje određenih elemenata na osnovu uslova. U ovom slučaju, proverava se da li je korisnik administrator. Ako jeste, prikazuju se određeni linkovi kao što su "Predmeti" i "Korisnici". Ako nije, ti linkovi neće biti prikazani.

## 8. Prikazivanje Navbar.vue na svim stranicama osim na Login

Potrebno je izmeniti App.vue tako da se primeni logika za prikazivanje ili sakrivanje navigacionog menija Navbar.vue na osnovu trenutne rute.

```
<script setup>
import { RouterLink, RouterView } from "vue-router";
import Navbar from "../components/Navbar.vue";
```

```
</script>

<template>
  <div>
    <Navbar v-if="$route.name !== 'Login'"></Navbar>
    <RouterView></RouterView>
  </div>
</template>

<style>
/* Stilovi za aplikaciju */
</style>
```

#### App.vue izvorni kod

- `<Navbar v-if="$route.name !== 'Login'">`: Ova linija koristi Vue direktivu `v-if` da uslovno prikaže komponentu `Navbar.vue` samo ako trenutna ruta nije `Login`. `$route.name` je svojstvo koje vraća ime trenutne rute.
- `RouterView` je komponenta koja prikazuje odgovarajuću komponentu za trenutnu rutu.

### 5.3 components/TableContainer.vue

Potrebno je u folder `components` kreirati novi fajl i nazvati ga `TableContainer.vue`. Ova komponenta služi da se isti stil primeni za svako prikazivanje entiteta (studenata, predmeta i korisnika) u tabele. Osim primene istog stila, ovim se izbegava ponavljanje pisanja dugmeta koje vodi do stranice kreiranja entiteta.

```
<script setup>
import { defineProps } from "vue";
import { RouterLink } from "vue-router";

const props = defineProps({
  to: {
    type: String,
    required: true,
  },
  buttonText: {
    type: String,
    required: true,
  },
});
</script>

<template>
  <div class="container">
```

```
<div class="row">
  <div class="my-5 justify-content-end">
    <RouterLink :to="to" role="button" class="btn btn-primary">
      {{ buttonText }}
    </RouterLink>
  </div>
  <div class="row">
    <table class="table">
      <thead class="thead-dark">
        <slot name="table-header"></slot>
      </thead>
      <tbody>
        <slot name="table-body"></slot>
      </tbody>
    </table>
  </div>
</div>
</div>
</template>
```

TableContainer.vue izvorni kod

## 1. Importovanje

```
import { defineProps } from "vue";
import { RouterLink } from "vue-router";
```

- **defineProps** je funkcija iz Vue.js koja omogućava definisanje svojstava (props) koje komponenta može primiti. Ovo omogućava komponenti da bude fleksibilna i ponovo upotrebljiva.
- **RouterLink** je komponenta iz vue-router paketa koja omogućava navigaciju između različitih ruta unutar aplikacije. Koristi se za kreiranje linkova koji vode do drugih stranica.

## 2. Vue.js defineProps

```
const props = defineProps({
  to: {
    type: String,
    required: true,
  },
  buttonText: {
    type: String,
    required: true,
  },
});
```

```
  },
});
```

- **props:** Objekat koji definiše svojstva koja komponenta može primiti. U ovom slučaju, komponenta očekuje dva svojstva:
  - **to:** String koji predstavlja putanju do koje RouterLink vodi. Ovo svojstvo je obavezno.
  - **buttonText:** String koji predstavlja tekst koji će biti prikazan na dugmetu. Ovo svojstvo je takođe obavezno.

```
<RouterLink :to="to" role="button" class="btn btn-primary">
  {{ buttonText }}
</RouterLink>
```

U gorenavedenom kodu nalazi se primena props svojstava:

- **<RouterLink>**: Ovo je komponenta iz vue-router paketa koja omogućava navigaciju između različitih ruta unutar Vue.js aplikacije. Koristi se za kreiranje linkova koji vode do drugih stranica.
- **:to="to"**: Direktiva :to (bind) koristi vrednost to iz props objekta. Ovo svojstvo definiše putanju do koje će link voditi. Na primer, ako je to="/studenti", klikom na link korisnik će biti preusmeren na stranicu sa putanjom /studenti.
- **role="button"**: HTML atribut role postavlja ulogu elementa. U ovom slučaju, role="button" označava da se ovaj link ponaša kao dugme, što može biti korisno za pristupačnost (accessibility).
- **class="btn btn-primary"**: Atribut class dodaje CSS klase elementu. U ovom slučaju, koristi se Bootstrap klasa btn btn-primary koja stilizuje link kao primarno dugme.
- **{{ buttonText }}**: Unutar dvostrukih vitičastih zagrada nalazi se buttonText, što je vrednost iz props objekta. Ovo svojstvo definiše tekst koji će biti prikazan na dugmetu. Na primer, ako je buttonText="Dodaj studenta", tekst na dugmetu će biti "Dodaj studenta".

### 3. Slotovi

```
<table class="table">
  <thead class="thead-dark">
    <slot name="table-header"></slot>
  </thead>
  <tbody>
    <slot name="table-body"></slot>
  </tbody>
</table>
```

- **<thead class="thead-dark">**: Ovo je deo tabele koji predstavlja zaglavlje tabele. Klasa thead-dark koristi Bootstrap stilove za tamno zaglavlje.



- **<slot name="table-header"></slot>**: Slot sa imenom table-header. Slotovi su mesta u komponenti gde se može umetnuti sadržaj iz roditeljske komponente. U ovom slučaju, sadržaj koji se umetne u slot sa imenom table-header će biti prikazan unutar <thead> elementa.
- **<tbody>**: Ovo je deo tabele koji predstavlja telo tabele.
- **<slot name="table-body"></slot>**: Slot sa imenom table-body. Sadržaj umetnut u slot sa imenom table-body će biti prikazan unutar <tbody> elementa.

**Slotovi:** Slotovi omogućavaju umetanje dinamičkog sadržaja u komponentu. U ovom slučaju, slotovi table-header i table-body omogućavaju umetanje prilagođenog zaglavlja i tela tabele iz roditeljske komponente:

- **<slot name="table-header"></slot>**: Ovaj slot omogućava umetanje prilagođenog sadržaja u zaglavlje tabele. Na primer, roditeljska komponenta može umetnuti kolone zaglavlja tabele.
- **<slot name="table-body"></slot>**: Ovaj slot omogućava umetanje prilagođenog sadržaja u telo tabele. Na primer, roditeljska komponenta može umetnuti redove sa podacima.

Primer umetanja sadržaja pomoću slotova:

```
<template>
  <TableContainer to="/korisnik-novi" buttonText="Dodaj korisnika">
    <template #table-header> //umetanje u slot
      <tr>
        <th scope="col">Ime</th>
        <th scope="col">Prezime</th>
        <th scope="col">Email</th>
        <th scope="col">Rola</th>
      </tr>
    </template>
    <template #table-body> //umetanje u slot
      <tr v-for="korisnik in korisnici" :key="korisnik.id">
        <td>{{ korisnik.ime }}</td>
        <td>{{ korisnik.prezime }}</td>
        <td>{{ korisnik.email }}</td>
        <td>{{ korisnik.rola }}</td>
      </tr>
    </template>
  </TableContainer>
</template>
```

## 5.4 components/FormContainer.vue

Potrebno je u folder components kreirati novi fajl i nazvati ga FormContainer.vue. Ova komponenta služi da se isti stil primeni za svaku formu prilikom kreiranja i izmene entiteta (studenata, predmeta i korisnika).

```
<script setup>
import { defineProps } from "vue";

const props = defineProps({
  title: {
    type: String,
    required: true,
  },
});
</script>

<template>
  <div class="container">
    <div class="row">
      <div class="col-lg-6 col-12">
        <div class="my-5">
          <h3>{{ title }}</h3>
        </div>
        <slot></slot>
      </div>
    </div>
  </div>
</template>
```

TableContainer.vue izvorni kod

## 5.5 components/DeleteModal.vue

Potrebno je u folder components kreirati novi fajl i nazvati ga DeleteModal.vue. Ovaj modal će se prikazati svaki put kada korisnik želi da izvrši akciju brisanja entiteta.

```
<script setup>
<script setup>
import { ref, defineExpose, onMounted, onBeforeUnmount } from "vue";

const props = defineProps({
  title: {
    type: String,
    default: "Potvrda brisanja",
  },
});
```

```
message: {
  type: String,
  default: "Da li ste sigurni da želite da obrišete stavku?",
},
onConfirm: {
  type: Function,
  required: true,
},
});

const modalElement = ref(null);
let modalInstance = null;

const showModal = () => {
  if (modalInstance) {
    modalInstance.show();
  }
};

const hideModal = () => {
  if (modalInstance) {
    modalInstance.hide();
  }
};

const handleConfirm = () => {
  props.onConfirm();
  hideModal();
};

onMounted(() => {
  modalInstance = new bootstrap.Modal(modalElement.value);
});

onBeforeUnmount(() => {
  if (modalInstance) {
    modalInstance.dispose();
  }
});

defineExpose({ showModal, hideModal });
</script>

<template>
  <div
    class="modal fade"
    ref="modalElement">
```

```
tabindex="-1"
aria-labelledby="deleteModalLabel"
aria-hidden="true"
>
<div class="modal-dialog">
  <div class="modal-content">
    <div class="modal-header">
      <h5 class="modal-title" id="deleteModalLabel">{{ title }}</h5>
      <button
        type="button"
        class="btn-close"
        data-bs-dismiss="modal"
        aria-label="Close"
      ></button>
    </div>
    <div class="modal-body">
      {{ message }}
    </div>
    <div class="modal-footer">
      <button
        type="button"
        class="btn btn-secondary"
        data-bs-dismiss="modal"
      >
        Otkazi
      </button>
      <button type="button" class="btn btn-danger"
@click="handleConfirm">
        Obriši
      </button>
    </div>
  </div>
</div>
</div>
</template>
```

DeleteModal.vue izvorni kod

## 1. Importovanje

```
import { ref, defineExpose, onMounted, onBeforeUnmount } from "vue";
```

- **defineExpose** dolazi iz Vue.js-a i koristi se da se eksplicitno izlože određene metode ili promenljive iz jedne komponente, kako bi ih roditeljska komponenta mogla koristiti:

```
defineExpose({ showModal, hideModal });
```

Konkretno u ovom primeru, izložene su funkcije **showModal** i **hideModal**, tako da neka druga komponenta može kontrolisati modal, odnosno da ga prikaže ili sakrije.

- **onMounted:**

```
onMounted(() => {  
  modalInstance = new bootstrap.Modal(modalElement.value);  
});
```

Konkretno u ovom primeru:

- **modalInstance = new bootstrap.Modal(modalElement.value)** kreira novu instancu Bootstrap-ovog modalnog dijaloga.
  - **modalElement.value** je referenca na HTML element koji predstavlja modal (to je div sa **ref="modalElement"** u **template-u**):

```
<template>  
  <div  
    class="modal fade"  
    ref="modalElement"  
    tabindex="-1"  
    aria-labelledby="deleteModalLabel"
```

- **new bootstrap.Modal(...)** koristi Bootstrap-ovu biblioteku da kreira instancu modala koja omogućava interakciju s modalom (npr. otvaranje, zatvaranje, i upravljanje modalnim dijalogom).

- **onBeforeUnmount** dolazi iz Vue.js-a i koristi se za izvršavanje nekog koda pre nego što komponenta bude uklonjena sa ekrana. To može biti korisno za čišćenje resursa, na primer uklanjanje *event listener*-a ili oslobađanje memorije:

```
onBeforeUnmount(() => {  
  if (modalInstance) {  
    modalInstance.dispose();  
  }  
});
```

Konkretno u ovom primeru:

- Proverava se da li postoji instanca modala (**modalInstance**), koja je ranije kreirana pomoću Bootstrapove metode.
- Ako postoji, poziva se metoda **dispose()**, koja dolazi iz Bootstrapovog API-ja. Ova metoda uklanja sve event listenere i druge resurse povezane sa modalom kako bi se izbeglo curenje memorije.
- Drugim rečima, pozivom **dispose()** Bootstrap modal se "deaktivira" i uklanjaju se svi resursi koji su bili potrebni za njegov rad.

## 2. Props

```
const props = defineProps({
  title: {
    type: String,
    default: "Potvrda brisanja",
  },
  message: {
    type: String,
    default: "Da li ste sigurni da želite da obrišete stavku?",
  },
  onConfirm: {
    type: Function,
    required: true,
  },
});
```

- U ovom slučaju, props omogućava da roditeljska komponenta prosledi **naslov** (title), **poruku** (message), i **funkciju** za potvrdu brisanja (onConfirm) ovoj modalnoj komponenti.
  - **title** i **message** imaju unapred definisane vrednosti koje će se koristiti ako ih roditeljska komponenta ne prosledi.
  - **onConfirm** je funkcija koja je **obavezna** i koja se mora proslediti kako bi komponenta radila pravilno. Ova funkcija se poziva kada korisnik potvrdi brisanje stavke.

## 3. Promenljive i funkcije

```
const modalElement = ref(null);
let modalInstance = null;

const showModal = () => {
  if (modalInstance) {
    modalInstance.show();
  }
};

const hideModal = () => {
  if (modalInstance) {
    modalInstance.hide();
  }
};

const handleConfirm = () => {
  props.onConfirm();
};
```

```
hideModal();  
};
```

- **modalElement:** Ovo je **ref** koji dolazi iz Vue.js-a i koristi se za referenciranje elementa u DOM-u.
- **modalElement** se ovde koristi za čuvanje referenci na HTML element modala (konkretno `<div>` element koji sadrži Bootstrap modal).
  - **ref(null):** Postavlja inicijalnu vrednost na `null`, što znači da referenca još nije postavljena, dok se modal ne prikaže u DOM-u.
  - Kasnije, kada je komponenta montirana (`onMounted`), **modalElement** će sadržati referencu na stvarni HTML element modala.
- **modalInstance:** Ova promenljiva čuva instancu Bootstrap modala, koja se dobija pri kreiranju modala sa `new bootstrap.Modal()`.
  - U početku, vrednost joj je `null`, jer modal još nije kreiran.
  - Kada se komponenta montira (`onMounted`), **modalInstance** se kreira i dobija vrednost stvarne Bootstrap modal instance, što omogućava kontrolu nad modalom (kao što su metode `show()`, `hide()`, `dispose()`).
- **showModal:** Ova funkcija se koristi za prikazivanje modala.
  - Prvo proverava da li postoji instanca modala (**modalInstance**).
  - Ako postoji, koristi Bootstrap metodu **show()** koja prikazuje modal.
  - Ova funkcija se eksplicitno izlaže kroz `defineExpose` kako bi roditeljska komponenta mogla pozvati `showModal` i tako prikazati modal u određenom trenutku.
- **hideModal:** Ova funkcija sakriva modal.
  - Slično kao kod `showModal`, proverava da li postoji instanca modala.
  - Ako postoji, poziva se Bootstrap metoda **hide()**, koja zatvara modal.
- **handleConfirm:** Ova funkcija se poziva kada korisnik klikne na dugme za potvrdu brisanja u modalnom prozoru.
  - Prvo poziva funkciju **props.onConfirm()**, koja je prosleđena kao prop iz roditeljske komponente. Ova funkcija definiše šta će se desiti kada korisnik potvrdi brisanje (npr. brisanje stavke iz baze).
  - Nakon toga, poziva funkciju **hideModal()** kako bi sakrila modal nakon što korisnik potvrdi akciju.

#### 4. Primer korišćenja DeleteModal komponente

```
<script setup>  
.....  
.....  
  
import DeleteModal from "@/components/DeleteModal.vue";
```

```
.....
.....

const predmetToDelete = ref(null);
const confirmModalRef = ref(null);

const confirmDelete = (predmet) => {
  predmetToDelete.value = predmet;
  confirmModalRef.value.showModal();
};

const deletePredmet = async () => {
  try {
    const response = await axios.delete(
      `/api/predmet-brisanje/${predmetToDelete.value.id}`
    );
    predmeti.value = predmeti.value.filter(
      (predmet) => predmet.id !== predmetToDelete.value.id
    );
    predmetToDelete.value = null;
    toast.success(response.data.message);
  } catch (error) {
    toast.error(error.response.data.message || error.message);
  }
};
</script>

<template>
.....
.....
<button
  @click="confirmDelete(predmet)"
  class="text-danger mx-1 btn btn-link"
>
.....
.....
<DeleteModal
  ref="confirmModalRef"
  :title="'Potvrda brisanja'"
  :message="'Da li ste sigurni da želite da obrišete predmet?'"
  :onConfirm="deletePredmet"
/>
</template>
```

Predmeti.vue izvorni kod - view za prikaz predmeta



- **Povezanost između confirmModalRef i modalElement:**

- **confirmModalRef:** Ova referenca se koristi u roditeljskoj komponenti za direktan pristup DeleteModal komponenti. Pomoću confirmModalRef mogu se pozivati metode unutar DeleteModal komponente, poput showModal() i hideModal().
- **modalElement:** Ovo je referenca unutar DeleteModal komponente, gde se modal (HTML element) povezuje sa Bootstrap modal instancom. Kada se DeleteModal prikaže, modalElement se koristi da se manipuliše Bootstrap modal prozorom.
- Kada se u roditeljskoj komponenti pozove confirmModalRef.value.showModal(), unutar DeleteModal komponente, metoda showModal() koristi modalElement da prikaže Bootstrap modal prozor.

- **Funkcija confirmDelete:**

- Kada korisnik klikne na dugme za brisanje nekog predmeta, funkcija confirmDelete postavlja predmet koji treba da se obriše u promenljivu predmetToDelete.
- Zatim, poziva se metoda showModal() na confirmModalRef, što otvara modalni prozor za potvrdu brisanja.

- **Funkcija deletePredmet**

- Kada korisnik potvrdi brisanje, funkcija deletePredmet se izvršava.
- Prvo se šalje DELETE zahtev ka API-ju koristeći axios, gde se predmet koji treba obrisati identifikuje pomoću predmetToDelete.value.id.
- Nakon uspešnog brisanja, predmet se uklanja iz lokalne liste (predmeti.value) pomoću filter metode.
- Ako je brisanje uspešno, prikazuje se poruka o uspehu koristeći toast.success(). Ako dođe do greške, prikazuje se odgovarajuća poruka o grešci.

- **DeleteModal props:**

```
<DeleteModal
  ref="confirmModalRef"
  :title="'Potvrda brisanja'"
  :message="'Da li ste sigurni da želite da obrišete predmet?'"
  :onConfirm="deletePredmet"
/>
```

- **title:** Ovaj prop se koristi za prikazivanje naslova u modalnom prozoru. U ovom slučaju, naslov je "Potvrda brisanja".
- **message:** Ovaj prop definiše poruku unutar modalnog prozora. U ovom slučaju, poruka je "Da li ste sigurni da želite da obrišete predmet?".

- **onConfirm:** Ovaj prop je funkcija koja se izvršava kada korisnik potvrdi akciju u modalnom prozoru. U ovom slučaju, funkcija `deletePredmet` se izvršava kako bi obrisala izabrani predmet

## 6. Kreiranje stranica za manipulaciju korisnicima

### 6.1 views/KorisnikNovi.vue

Potrebno je u folder views kreirati novi fajl i nazvati ga KorisnikNovi.vue. Predstavljaće stranicu za kreiranje novih korisnika.

```
<script setup>
import { ref } from "vue";
import axios from "axios";
import { toast } from "vue3-toastify";
import { useRouter } from "vue-router";
import FormContainer from "@/components/FormContainer.vue";

const router = useRouter();

const ime = ref("");
const prezime = ref("");
const email = ref("");
const rola = ref("");
const lozinka = ref("");

const handleSubmit = async (event) => {
  event.preventDefault();
  try {
    const response = await axios.post("/api/korisnik-novi", {
      ime: ime.value,
      prezime: prezime.value,
      email: email.value,
      rola: rola.value,
      lozinka: lozinka.value,
    });
    router.push("/korisnici").then(() =>
toast.success(response.data.message));
  } catch (error) {
    toast.error(error.response.data.message || error.message);
  }
};
</script>

<template>
<FormContainer title="Novi korisnik">
  <form @submit="handleSubmit">
    <div class="mb-3">
      <label for="ime" class="form-label">Ime</label>
      <input
        type="text"
        class="form-control"

```

```
        id="ime"
        v-model="ime"
        required
    />
</div>

<div class="mb-3">
    <label for="prezime" class="form-label">Prezime</label>
    <input
        type="text"
        class="form-control"
        id="prezime"
        v-model="prezime"
        required
    />
</div>

<div class="mb-3">
    <label for="email" class="form-label">Email</label>
    <input
        type="email"
        class="form-control"
        id="email"
        v-model="email"
        required
    />
</div>

<div class="mb-3">
    <label for="rola" class="form-label">Rola</label>
    <select id="rola" class="form-select" v-model="rola" required>
        <option selected disabled value="">Izaberi rolu</option>
        <option value="administrator">Administrator</option>
        <option value="profesor">Profesor</option>
    </select>
</div>

<div class="mb-3">
    <label for="lozinka" class="form-label">Lozinka</label>
    <input
        type="password"
        class="form-control"
        id="lozinka"
        v-model="lozinka"
        required
    />
</div>
```

```
        <button type="submit" class="btn btn-primary">Sačuvaj</button>
      </form>
    </FormContainer>
  </template>
```

KorisnikNovi.vue izvorni kod

## 10. Importovanje

```
import FormContainer from "@/components/FormContainer.vue";
```

- Iz foldera components potrebno je da se uveze FormContainer komponenta koja je obrađena u prethodnom poglavlju. Ona „omotava“ formu što omogućava standardizovan izgled i strukturu.

## 11. Promenljive

```
const router = useRouter();

const ime = ref("");
const prezime = ref("");
const email = ref("");
const rola = ref("");
const lozinka = ref("");
```

- ime, prezime, email, rola, lozinka:** Reaktivne promenljive koje čuvaju unos korisnika u odgovarajuća polja. Koristeći v-model direktivu, vrednosti ovih polja se automatski sinhronizuju sa podacima u formi:

```
<input
  type="text"
  class="form-control"
  id="ime"
  v-model="ime"
  required
/>
```

- router:** instanca router-a koja se koristi za navigaciju.

## 12. Funkcija `handleSubmit`

```
const handleSubmit = async (event) => {
  event.preventDefault();
  try {
    const response = await axios.post("/api/korisnik-novi", {
      ime: ime.value,
      prezime: prezime.value,
      email: email.value,
      rola: rola.value,
      lozinka: lozinka.value,
    });
    router.push("/korisnici").then(() =>
      toast.success(response.data.message));
  } catch (error) {
    toast.error(error.response.data.message || error.message);
  }
};
```

- **handleSubmit**: Funkcija koja se poziva pri slanju forme. Ova funkcija:
  - Sprečava podrazumevano ponašanje forme (osvežavanje stranice) korišćenjem `event.preventDefault()`.
  - Šalje POST zahtev na `/api/korisnik-novi`, prosleđujući podatke unesene u formu (**ime**, **prezime**, **email**, **rola**, **lozinka**).
  - U slučaju uspešnog odgovora, korisnik se preusmerava na stranicu sa spiskom korisnika, a notifikacija o uspehu se prikazuje pomoću **toast.success**.
  - Ako dođe do greške, prikazuje se odgovarajuća notifikacija pomoću **toast.error**.

## 13. Props u `FormContainer`-u

```
<FormContainer title="Novi korisnik">
  .....
  .....
</FormContainer>
```

- **title**: Prikazuje naslov forme (u ovom slučaju "Novi korisnik").
- Ovaj prop olakšava ponovnu upotrebu komponente **FormContainer** za različite forme sa različitim naslovima.

## 14. Registrovanje `/korisnik-novi` rute

Potrebno je u fajlu `router/index.js` registrovati rutu za kreiranje novog korisnika. Kada se pristupi URL-u: <http://localhost:5173/korisnik-novi>, učit će se gorenavedena `KorisnikNovi.vue` komponenta. Pored toga, potrebno je implementirati funkciju za rutiranje koja će proveravati da li korisnik ima odgovarajuću ulogu (rolu) za pristup određenim rutama. Korisnicima i predmetima može upravljati samo administrator, dok studentima mogu upravljati i administrator i profesor. Ako korisnik nema odgovarajuću ulogu, biće preusmeren na login stranicu ili na drugu odgovarajuću rutu.

```
import { createRouter, createWebHistory } from "vue-router";
import KorisnikNovi from "@views/KorisnikNovi.vue";
import Login from "@views/Login.vue";

const routes = [
  {
    path: "/korisnik-novi",
    name: "KorisnikNovi",
    component: KorisnikNovi,
    meta: { requiresRole: ["administrator"] },
  },
  {
    path: "/login",
    name: "Login",
    component: Login,
  },
];

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes,
});

router.beforeEach((to, from, next) => {
  const role = localStorage.getItem("rola");

  if (role) {
    if (to.meta.requiresRole && !to.meta.requiresRole.includes(role)) {
      return next("/login");
    }
  } else if (to.meta.requiresRole) {
    return next("/login");
  }

  next();
});

export default router;
```

**router/index.js izvorni kod**

- **Funkcija router.beforeEach:**

- Funkcija router.beforeEach se koristi za presretanje svake navigacije između ruta. Ona omogućava postavljanje provera i logike pre nego što korisnik dobije pristup odabranoj ruti. U ovom slučaju, koristi se za proveru da li korisnik poseduje odgovarajuću rolu. Ako korisnik nema odgovarajuću rolu, preusmerava se na login stranicu ili na rutu kojoj ima pristup.

- **meta: { requiresRole: ["administrator"] }:**

- Ovaj meta podatak definiše koja rola ima pristup određenoj ruti. U primeru je navedeno da samo korisnici sa ulogom "administrator" mogu pristupiti ruti. Meta podaci se ne izvršavaju direktno, već omogućavaju korišćenje dodatnih provera unutar router.beforeEach funkcije, gde se provera uloge vrši na osnovu zahteva određenih u meta.requiresRole.

## 6.2 views/KorisnikIzmena.vue

Potrebno je u folder views kreirati novi fajl i nazvati ga KorisnikIzmena.vue. Predstavljaće stranicu za izmenu korisnika.

```
<script setup>
import { ref, onMounted } from "vue";
import axios from "axios";
import { useRoute } from "vue-router";
import { useRouter } from "vue-router";
import FormContainer from "@/components/FormContainer.vue";
import { toast } from "vue3-toastify";

const router = useRouter();
const route = useRoute();
const id = route.params.id;

const ime = ref("");
const prezime = ref("");
const email = ref("");
const rola = ref("");
const lozinka = ref("");

const fetchKorisnik = async () => {
  try {
    const response = await axios.get(`/api/korisnik/${id}`);
    ime.value = response.data.ime;
    prezime.value = response.data.prezime;
    email.value = response.data.email;
    rola.value = response.data.rola;
  } catch (error) {
```



```

    if (error.response.status == 401) {
      router
        .push("/login")
        .then(() => toast.error(error.response.data.message));
    } else {
      toast.error(error.response.data.message || error.message);
    }
  }
};

const handleSubmit = async (event) => {
  event.preventDefault();
  try {
    const response = await axios.put(`/api/korisnik-izmena/${id}`, {
      ime: ime.value,
      prezime: prezime.value,
      email: email.value,
      rola: rola.value,
      lozinka: lozinka.value,
    });
    router.push("/korisnici").then(() =>
toast.success(response.data.message));
  } catch (error) {
    toast.error(error.response.data.message || error.message);
  }
};

onMounted(() => {
  fetchKorisnik();
});
</script>

<template>
  <FormContainer title="Izmena korisnika">
    <form @submit="handleSubmit">
      <div class="mb-3">
        <label for="ime" class="form-label">Ime</label>
        <input
          type="text"
          class="form-control"
          id="ime"
          v-model="ime"
          required
        />
      </div>

      <div class="mb-3">

```

```
<label for="prezime" class="form-label">Prezime</label>
<input
  type="text"
  class="form-control"
  id="prezime"
  v-model="prezime"
  required
/>
</div>

<div class="mb-3">
  <label for="email" class="form-label">Email</label>
  <input
    type="email"
    class="form-control"
    id="email"
    v-model="email"
    required
  />
</div>

<div class="mb-3">
  <label for="rola" class="form-label">Rola</label>
  <select id="rola" class="form-select" v-model="rola" required>
    <option selected disabled value="">Izaberi rolu</option>
    <option value="administrator">Administrator</option>
    <option value="profesor">Profesor</option>
  </select>
</div>

<div class="mb-3">
  <label for="lozinka" class="form-label">Lozinka</label>
  <input
    type="password"
    class="form-control"
    id="lozinka"
    v-model="lozinka"
    required
  />
</div>

  <button type="submit" class="btn btn-primary">Sačuvaj</button>
</form>
</FormContainer>
</template>
```

KorisnikIzmena.vue izvorni kod

## 1. Importovanje

```
import { useRoute } from "vue-router";
```

- **useRoute** je funkcija iz Vue Router biblioteke koja omogućava pristup trenutnoj ruti unutar komponente. Ova funkcija vraća objekat koji sadrži informacije o ruti, kao što su:
  - **params**: parametri rute, npr. id u ruti /korisnici/:id.
  - **query**: *query* parametri, npr. u ruti /korisnici?ime=Marko.
  - **name**: naziv rute ako je definisan u routes konfiguraciji.
  - **path**: puna putanja trenutne rute.

## 2. Promenljive

```
const router = useRouter();
const route = useRoute();
const id = route.params.id;

const ime = ref("");
const prezime = ref("");
const email = ref("");
const rola = ref("");
const lozinka = ref("");
```

- **ime, prezime, email, rola, lozinka**: Reaktivne promenljive koje se koriste za popunjavanje input polja sa podacima dobijenim sa servera. Koristeći `v-model` direktivu, ove vrednosti su povezane sa podacima u formi.
- **router** i **route**: Omogućavaju navigaciju između stranica i pristup parametrima u URL-u (u ovom slučaju id korisnika).

## 3. Funkcija fetchKorisnik

```
const fetchKorisnik = async () => {
  try {
    const response = await axios.get(`/api/korisnik/${id}`);
    ime.value = response.data.ime;
    prezime.value = response.data.prezime;
    email.value = response.data.email;
    rola.value = response.data.rola;
  } catch (error) {
    toast.error(error.response.data.message || error.message);
  }
}
```

```
};
```

- Ova funkcija šalje GET zahtev na `/api/korisnik/${id}` kako bi dohvatila podatke korisnika sa servera. Nakon uspešnog odgovora, polja u formi (ime, prezime, email, rola) se popunjavaju vrednostima koje su dobijene sa servera pomoću `ref` promenljivih.

#### 4. Funkcija `handleSubmit`

```
const handleSubmit = async (event) => {
  event.preventDefault();
  try {
    const response = await axios.put(`/api/korisnik-izmena/${id}`, {
      ime: ime.value,
      prezime: prezime.value,
      email: email.value,
      rola: rola.value,
      lozinka: lozinka.value,
    });
    router.push("/korisnici").then(() =>
toast.success(response.data.message));
  } catch (error) {
    toast.error(error.response.data.message || error.message);
  }
};
```

- Funkcija koja šalje PUT zahtev za izmenu podataka korisnika na osnovu unosa u formi. U slučaju uspešnog zahteva, korisnik se preusmerava na stranicu sa spiskom korisnika i prikazuje se notifikacija o uspehu.

#### 5. Props u `FormContainer-u`

```
<FormContainer title="Izmena korisnika">
  .....
  .....
</FormContainer>
```

- **title:** Prikazuje naslov forme (u ovom slučaju "Izmena korisnika").

#### 6. Registrovanje `/korisnik-izmena` rute

Potrebno je u fajlu `router/index.js` registrovati rutu za izmenu korisnika. Kada se pristupi URL-u: <http://localhost:5173/korisnik-izmena/:id>, učitane se gorenavedena `KorisnikIzmena.vue` komponenta. Pored toga, potrebno je da ova ruta bude zaštićena, i da može da joj pristupi samo korisnik sa ulogom administratora.

```
import { createRouter, createWebHistory } from "vue-router";
import KorisnikNovi from "@/views/KorisnikNovi.vue";
import KorisnikIzmena from "@/views/KorisnikIzmena.vue";
import Login from "@/views/Login.vue";

const routes = [
  {
    path: "/korisnik-novi",
    name: "KorisnikNovi",
    component: KorisnikNovi,
    meta: { requiresRole: ["administrator"] },
  },
  {
    path: "/korisnik-izmena/:id",
    name: "KorisnikIzmena",
    component: KorisnikIzmena,
    props: true,
    meta: { requiresRole: ["administrator"] },
  },
  {
    path: "/login",
    name: "Login",
    component: Login,
  },
];

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes,
});

router.beforeEach((to, from, next) => {
  const role = localStorage.getItem("rola");

  if (role) {
    if (to.meta.requiresRole && !to.meta.requiresRole.includes(role)) {
      return next("/login");
    }
  } else if (to.meta.requiresRole) {
    return next("/login");
  }
});
```

```
    next();
  });

export default router;
```

#### router/index.js izvorni kod

- **meta: { requiresRole: ["administrator"] }:**
  - Ovaj meta podatak se dodaje I za rutu /korisnik-izmena/:id, kao I kod stranice za kreiranje novih korisnika.

### 6.3 views/Korisnici.vue

Potrebno je u folder views kreirati novi fajl I nazvati ga Korisnici.vue. Predstavljace stranicu za prikazivanje svih korisnika.

```
<script setup>
import { ref, onMounted } from "vue";
import axios from "axios";
import { toast } from "vue3-toastify";
import { useRouter, RouterLink } from "vue-router";
import TableContainer from "@/components/TableContainer.vue";
import DeleteModal from "@/components/DeleteModal.vue";

const korisnici = ref([]);
const korisnikToDelete = ref(null);
const confirmModalRef = ref(null);
const router = useRouter();

const fetchKorisnici = async () => {
  try {
    const response = await axios.get("/api/korisnici");
    korisnici.value = response.data;
  } catch (error) {
    toast.error(error.response.data.message || error.message);
  }
};

const confirmDelete = (korisnik) => {
  korisnikToDelete.value = korisnik;
  confirmModalRef.value.showModal();
};

const deleteKorisnik = async () => {
  try {
    const response = await axios.post(
```

```

    `/api/korisnik-brisanje/${korisnikToDelete.value.id}`
  );
  korisnici.value = korisnici.value.filter(
    (korisnik) => korisnik.id !== korisnikToDelete.value.id
  );
  korisnikToDelete.value = null;
  confirmModalRef.value.hideModal();
  toast.success(response.data.message);
} catch (error) {
  toast.error(error.response.data.message || error.message);
}
};

onMounted(() => {
  fetchKorisnici();
});
</script>

<template>
  <TableContainer to="/korisnik-novi" buttonText="Dodaj korisnika">
    <template #table-header>
      <tr>
        <th scope="col">Ime</th>
        <th scope="col">Prezime</th>
        <th scope="col">Email</th>
        <th scope="col">Rola</th>
        <th scope="col">Akcija</th>
      </tr>
    </template>
    <template #table-body>
      <tr v-for="korisnik in korisnici" :key="korisnik.id">
        <td>{{ korisnik.ime }}</td>
        <td>{{ korisnik.prezime }}</td>
        <td>{{ korisnik.email }}</td>
        <td>{{ korisnik.rola }}</td>
        <td>
          <button
            @click="router.push(`/korisnik-izmena/${korisnik.id}`)"
            role="button"
            class="text-warning mx-1 btn btn-link"
          >
            <i class="fas fa-edit"></i>
          </button>
          <button
            @click="confirmDelete(korisnik)"
            class="text-danger mx-1 btn btn-link"
          >

```

```
        <i class="fas fa-trash"></i>
      </button>
    </td>
  </tr>
</template>
</TableContainer>

<DeleteModal
  ref="confirmModalRef"
  :title="'Potvrda brisanja'"
  :message="'Da li ste sigurni da želite da obrišete korisnika?'"
  :onConfirm="deleteKorisnik"
/>
</template>
```

KorisnikIzmena.vue izvorni kod

## 1. Importovanje

```
import TableContainer from "@/components/TableContainer.vue";
import DeleteModal from "@/components/DeleteModal.vue";
```

- **TableContainer** i **DeleteModal** su prethodno kreirane komponente koje se koriste za prikaz tabele korisnika i modala za potvrdu brisanja.

## 2. Promenljive

```
const korisnici = ref([]);
const korisnikToDelete = ref(null);
const confirmModalRef = ref(null);
const router = useRouter();
```

- **korisnici**: reaktivna promenljiva koja čuva listu svih korisnika.
- **korisnikToDelete**: koristi se za privremeno skladištenje podataka o korisniku koji će biti obrisani.
- **confirmModalRef**: referenca na modal za brisanje korisnika, omogućava prikaz i skrivanje modala.

## 3. Funkcija fetchKorisnici

```
const fetchKorisnici = async () => {
  try {
```



```
const response = await axios.get("/api/korisnici");
korisnici.value = response.data;
} catch (error) {
  toast.error(error.response.data.message || error.message);
}
};
```

- Ova funkcija šalje GET zahtev na /api/korisnici kako bi preuzela sve korisnike iz baze podataka. Podaci o korisnicima se zatim smeštaju u reaktivnu promenljivu **korisnici**.

#### 4. Funkcija confirmDelete

```
const confirmDelete = (korisnik) => {
  korisnikToDelete.value = korisnik;
  confirmModalRef.value.showModal();
};
```

- Ova funkcija se poziva kada korisnik klikne na dugme za brisanje. Postavlja korisnika koji će biti obrisani u promenljivu **korisnikToDelete** i prikazuje modal za potvrdu brisanja.

#### 5. Funkcija deleteKorisnik

```
const deleteKorisnik = async () => {
  try {
    const response = await axios.post(
      `/api/korisnik-brisanje/${korisnikToDelete.value.id}`
    );
    korisnici.value = korisnici.value.filter(
      (korisnik) => korisnik.id !== korisnikToDelete.value.id
    );
    korisnikToDelete.value = null;
    confirmModalRef.value.hideModal();
    toast.success(response.data.message);
  } catch (error) {
    toast.error(error.response.data.message || error.message);
  }
};
```

- Ova funkcija šalje **POST** zahtev na /api/korisnik-brisanje/\${korisnikToDelete.value.id} kako bi obrisala korisnika iz baze. Nakon uspešnog brisanja, korisnik se uklanja iz liste **korisnici**, modal se zatvara, a korisnik se informiše o uspešnom brisanju putem toast notifikacije.

## 6. Registrovanje /korisnici rute

Potrebno je u fajlu router/index.js registrovati rutu za prikazivanje. Kada se pristupi URL-u: <http://localhost:5173/korisnici>, učitace se gorenavedena Korisnici.vue komponenta. Pored toga, potrebno je da ova ruta bude zaštićena, i da može da joj pristupi samo korisnik sa ulogom administratora.

```
import { createRouter, createWebHistory } from "vue-router";
import Korisnici from "@/views/Korisnici.vue";
import KorisnikNovi from "@/views/KorisnikNovi.vue";
import KorisnikIzmena from "@/views/KorisnikIzmena.vue";
import Login from "@/views/Login.vue";

const routes = [
  {
    path: "/korisnici",
    name: "Korisnici",
    component: Korisnici,
    meta: { requiresRole: ["administrator"] },
  },
  {
    path: "/korisnik-novi",
    name: "KorisnikNovi",
    component: KorisnikNovi,
    meta: { requiresRole: ["administrator"] },
  },
  {
    path: "/korisnik-izmena/:id",
    name: "KorisnikIzmena",
    component: KorisnikIzmena,
    props: true,
    meta: { requiresRole: ["administrator"] },
  },
  {
    path: "/login",
    name: "Login",
    component: Login,
  },
];

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes,
});

router.beforeEach((to, from, next) => {
  const role = localStorage.getItem("rola");
```

```
if (role) {
  if (to.meta.requiresRole && !to.meta.requiresRole.includes(role)) {
    return next("/login");
  }
} else if (to.meta.requiresRole) {
  return next("/login");
}

next();
});

export default router;
```

**router/index.js izvorni kod**

- **meta: { requiresRole: ["administrator"] }:**
  - Ovaj meta podatak se dodaje I za rutu /korisnici

## LITERATURA

- [1] <https://vuejs.org/guide/introduction.html>
- [2] <https://www.w3schools.com/vue/index.php>
- [3] <https://github.com/bradtraversy/vue-crash-2024>
- [4] <https://getbootstrap.com/docs/5.3/getting-started/introduction/>
- [5] <https://fontawesome.com/>