Univerzitet u Beogradu  
Fakultet organizacionih nauka  
Katedra za elektronsko poslovanje

TEMA

Seminarski rad iz predmeta Internet tehnologije

Mentor: Studenti:

Beograd, 2023.

Sadržaj

[Korisnički zahtev 1](#_Toc158336297)

[Opis sistema 1](#_Toc158336298)

[Opis slučajeva korišćenja 2](#_Toc158336299)

[Prijava korisnika 2](#_Toc158336300)

[Kreiranje dokumenta 3](#_Toc158336301)

[Pretraga dokumenta 4](#_Toc158336302)

[Opis arihtekture aplikacije 4](#_Toc158336303)

[Opis procesa slučajeva korišćenja 5](#_Toc158336304)

[Dijagram sekvenci – prijava korisnika 5](#_Toc158336305)

[Dijagram sekvenci – kreiranje dokumenta 6](#_Toc158336306)

[Dijagram sekvenci – pretraga dokumenta 7](#_Toc158336307)

[Model podataka 8](#_Toc158336308)

[Specifikacija REST API-ja 9](#_Toc158336309)

[Opis tehnologija korišćenih u aplikaciji 17](#_Toc158336310)

[Korisničko uputstvo 18](#_Toc158336311)

[Prikaz delova koda 22](#_Toc158336312)

[Laravel 22](#_Toc158336313)

[React 40](#_Toc158336314)

[Link ka github repozitorijumu 57](#_Toc158336315)

[Reference 58](#_Toc158336316)

# Korisnički zahtev

Ova aplikacija je dizajnirana kao platforma za upravljanje dokumentima, pružajući funkcionalnosti za kreiranje, čuvanje, ažuriranje i brisanje dokumenata. Korisnici mogu pretraživati dokumente koristeći različite filtere poput naslova, sadržaja, autora, kategorija i tagova, a takođe imaju mogućnost preuzimanja dokumenata. Pored upravljanja dokumentima, aplikacija podržava kreiranje i upravljanje korisničkim nalogom, uključujući registraciju, prijavljivanje, odjavljivanje i resetovanje lozinke. Postoji i sistem kategorizacije i tagovanja dokumenata za bolju organizaciju i pretragu.

Korisnici koji nisu ulogovani imaju mogućnost pregledanja IT dokumenta sa spoljnog apija, dok administrator ima mogućnost pregleda statistike vezane za dokumenta.

# Opis sistema

## Opis slučajeva korišćenja

### Prijava korisnika

Slučaj korišćenja: Prijava korisnika na sistem

Akteri:

* Korisnik
* Sistem (program)Preduslov:
* Sistem je uključen, a korisnik ima kreiran i verifikovan nalog za pristup.

Osnovni scenario:

1. Korisnik bira opciju za prijavu na sistem.
2. Sistem prikazuje formu za prijavu koja traži korisničko ime i lozinku.
3. Korisnik unosi svoje korisničko ime i lozinku u predviđena polja.
4. Korisnik potvrđuje unos i šalje podatke sistemu klikom na dugme "Prijavi se".
5. Sistem proverava da li su uneti podaci validni i da li se podudaraju sa podacima u bazi.
6. Ukoliko su podaci ispravni, sistem autorizuje korisnika i daje mu pristup odgovarajućim funkcionalnostima na osnovu njegovog nivoa pristupa.
7. Sistem prikazuje poruku "Uspešno ste prijavljeni na sistem" i preusmerava korisnika na početnu stranu.

### Kreiranje dokumenta

Akteri:

Korisnik

Sistem (program)

Preduslov:

Sistem je uključen, a korisnik ima pristup funkcionalnostima za kreiranje novih dokumenata.

Osnovni scenario:

1. Korisnik bira opciju za kreiranje novog dokumenta unutar sistema.
2. Sistem prikazuje formu za unos podataka o dokumentu, uključujući naziv dokumenta, sadržaj, autora, kategoriju, tagove, fajl dokumenata, broj preuzimanja i druge relevantne informacije.
3. Korisnik unosi tražene informacije o dokumentu u formu.
4. Korisnik pregleda unete podatke i proverava da li su sve informacije ispravne i potpune.
5. Korisnik potvrđuje unos klikom na dugme "Kreiraj dokument" ili slično.
6. Sistem proverava unete podatke na validnost i potpunost.
7. Ukoliko su svi podaci ispravni, sistem kreira novi dokument i registruje ga u bazi.
8. Sistem obaveštava korisnika porukom: "Novi dokument je uspešno kreiran."

### Pretraga dokumenta

Akteri:

Korisnik

Sistem (program)

Preduslovi:

Sistem je uključen i korisnik ima pristup funkcionalnostima za pretraživanje dokumenata.

Osnovni scenario:

1. Korisnik unosi kriterijume pretrage u odgovarajuća polja ili filtere unutar sistema.
2. Sistem prikazuje listu dokumenata koji odgovaraju unetim kriterijumima pretrage.
3. Korisnik pregleda listu rezultata pretrage.
4. Korisnik bira dokument koji želi detaljnije pregledati ili preuzeti.
5. Sistem prikazuje odabrani dokument u detaljnom prikazu.
6. Korisnik ima opciju za preuzimanje odabranog dokumenta.
7. Korisnik može izvršiti dodatnu akciju na dokumentu, kao što su izmena ili brisanje, ako je autorizovan za to.

## Opis arihtekture aplikacije

Aplikacija koja je pred vama je izgrađena koristeći skup alata koji uključuje Laravel, React i Axios, što nam omogućava brz razvoj sofisticiranih web rešenja. Laravel, koji koristimo kao osnovu za backend, poznat je po svojim performansama, bezbednosnim karakteristikama i jednostavnosti upotrebe. On formira temelj naše aplikacije, upravlja bazu podataka, autentikacijom korisnika, poslovnom logikom i API endpointima. MVC (Model-View-Controller) arhitektura Laravela obezbeđuje da logika aplikacije bude odvojena od korisničkog interfejsa, što olakšava razvoj i ažuriranje koda. API-ji razvijeni u Laravelu omogućavaju komunikaciju sa frontendom preko JSON-a, što efikasno omogućava razmenu informacija.

Za kreiranje interfejsa koristimo React, biblioteku za JavaScript koja nam pomaže da konstruišemo dinamične i interaktivne web stranice. React nam omogućava da dizajniramo komponente koje definišu kako će naša aplikacija izgledati i funkcionišati, čime se pojednostavljuje izrada jednostraničnih aplikacija (SPA). Uz React, naša aplikacija može ažurirati korisnički interfejs u realnom vremenu bez potrebe za ponovnim učitavanjem cele stranice, što pruža glatko korisničko iskustvo.

Komunikacija između React frontend-a i Laravel backend-a ostvaruje se preko Axiosa, koji je široko prihvaćen za slanje HTTP zahteva. Axios olakšava upravljanje zahtevima i odgovorima, slanje tokena za autentikaciju i obradu povratnih informacija sa servera. Zahvaljujući ovoj biblioteci, naša aplikacija može da preuzima i šalje podatke serveru te izvodi CRUD operacije na efikasan i siguran način.

## Opis procesa slučajeva korišćenja

### Dijagram sekvenci – prijava korisnika

Slika 1 ds - prijava korisnika

### Dijagram sekvenci – kreiranje dokumenta

Slika na kojoj se nalazi tekst, snimak ekrana, Font, broj

Opis je automatski generisan

Slika 2 ds -kreiranje dokumenta

### Dijagram sekvenci – pretraga dokumenta

Slika na kojoj se nalazi tekst, snimak ekrana, Font, broj

Opis je automatski generisan

Slika 3 ds -pretraga dokumenta

## Model podataka

U modelu Category.php, kategorija sadrži polje name koje čuva ime kategorije, i ima vezu mnogo-na-mnogo sa modelom Document, što znači da svaka kategorija može biti povezana sa više dokumenata, a svaki dokument može pripadati više kategorija.

Model Comment.php uključuje polja content, document\_id i user\_id, gde svaki komentar sadrži tekstualni sadržaj, referencu na dokument na koji se odnosi i korisnika koji ga je kreirao. Ovaj model ima veze mnogo-na-jedan sa modelima Document i User, što ukazuje da je svaki komentar povezan sa jednim dokumentom i jednim korisnikom.

U Document.php, dokument ima polja kao što su title, content, file\_path, author\_id, downloads, is\_public, category\_id i tags. Ovaj model ima mnogo-na-jedan vezu sa User (preko author\_id), mnogo-na-jedan vezu sa Category i mnogo-na-mnogo vezu sa Tag, što omogućava dokumentima da imaju više tagova i da pripadaju više kategorija.

Model Tag.php sadrži polja name, color i slug, i ima mnogo-na-mnogo vezu sa Document, omogućavajući svakom tagu da bude povezan sa više dokumenata.

Na kraju, model User.php obuhvata polja kao što su name, email, password, date\_of\_birth i bio, i ima veze jedan-na-mnogo sa Document i Comment, što znači da svaki korisnik može kreirati više dokumenata i komentara.

Slika na kojoj se nalazi tekst, dijagram, linija, Plan

Opis je automatski generisan

Slika 4 ERD DIJAGRAM

## Specifikacija REST API-ja

|  |  |
| --- | --- |
| Opis funkcije | Prikaz svih dokumenata |
| HTTP metoda | GET |
| URL | /api/ documents |
| URL parametri | nema |
| HTTP body parametri | nema |
| Format HTTP body parametara | JSON |
| Izlazni parametri | [      {          "id": 4,          "title": "dasda",          "content": "dsada",          "file\_path": "documents/zJxbzLWt4le7K34az25zryHyoGwAw3657r7FsYdD.png",          "author\_id": 1,          "category\_id": 1,          "downloads": 5,          "is\_public": 0,          "tags": [              "1"          ],          "created\_at": "2024-01-26T21:56:30.000000Z",          "updated\_at": "2024-01-26T21:56:30.000000Z",          "deleted\_at": **null**      },      {          "id": 6,          "title": "das",          "content": "da",          "file\_path": "documents/IabI7QfnAkOO6SL4z4UcQxnuvM4olVTB4mVeFcBB.png",          "author\_id": 5,          "category\_id": 1,          "downloads": 0,          "is\_public": 0,          "tags": [              "1",              "2"          ],          "created\_at": "2024-01-29T20:42:43.000000Z",          "updated\_at": "2024-01-29T20:42:43.000000Z",          "deleted\_at": **null**      },  …  **}** |
| Format izlaznih parametara | Application/json |
|  |  |
| Opis funkcije | Kreiranje dokumenta |
| HTTP metoda | POST |
| URL | Api/documents |
| URL parametri | ?title=naslov&content=sadrzaj&author\_id=1&category\_id=1&tags[]=[1]&downloads=0 |
| HTTP body parametri | Slika na kojoj se nalazi tekst, snimak ekrana, Font, linija  Opis je automatski generisan |
| Format HTTP body parametara | nema |
| Izlazni parametri | {      "title": "naslov",      "content": "sadrzaj",      "author\_id": "1",      "category\_id": "1",      "downloads": "0",      "tags": [          "1"      ],      "file\_path": "documents/dBHGUnp58sDP0Ve2TSOWG7xq13yyC8PcrHlF4qMb.png",      "is\_public": 0,      "updated\_at": "2024-01-30T13:55:12.000000Z",      "created\_at": "2024-01-30T13:55:12.000000Z",      "id": 12  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | Ažuriranje dokumenta |
| HTTP metoda | PUT |
| URL | /api/documents |
| URL parametri | 12?title=naslov&content=sadrzaj&author\_id=1&category\_id=1&downloads=0 |
| HTTP body parametri | (nema) |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | {      "id": 12,      "title": "naslov",      "content": "sadrzaj",      "file\_path": "documents/dBHGUnp58sDP0Ve2TSOWG7xq13yyC8PcrHlF4qMb.png",      "author\_id": "1",      "category\_id": "1",      "downloads": "0",      "is\_public": 0,      "tags": [          "1"      ],      "created\_at": "2024-01-30T13:55:12.000000Z",      "updated\_at": "2024-01-30T13:55:12.000000Z",      "deleted\_at": **null**  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | Brisanje dokumenta |
| HTTP metoda | DELETE |
| URL | /api/documents |
| URL parametri | 12 |
| HTTP body parametri | (nema) |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | -No Content- |
| Format izlaznih parametara | application/json |
| Opis funkcije | Pretraga dokumenata |
| HTTP metoda | GET |
| URL | /api/documents |
| URL parametri | ?author\_id=1 |
| HTTP body parametri | nema |
| Format HTTP body parametara | JSON |
| Izlazni parametri | [      {          "id": 4,          "title": "dasda",          "content": "dsada",          "file\_path": "documents/zJxbzLWt4le7K34az25zryHyoGwAw3657r7FsYdD.png",          "author\_id": 1,          "category\_id": 1,          "downloads": 5,          "is\_public": 0,          "tags": [              "1"          ],          "created\_at": "2024-01-26T21:56:30.000000Z",          "updated\_at": "2024-01-26T21:56:30.000000Z",          "deleted\_at": **null**      }  ] |
| Format izlaznih parametara | Application/json |
|  |  |
| Opis funkcije | Kreiranje komenatara |
| HTTP metoda | POST |
| URL | /api/comments |
| URL parametri | content=sadrzaj&user\_id=1&document\_id=11 |
| HTTP body parametri | (nema) |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | {      "content": "sadrzaj",      "document\_id": "11",      "user\_id": "1",      "updated\_at": "2024-01-30T14:03:48.000000Z",      "created\_at": "2024-01-30T14:03:48.000000Z",      "id": 4  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | Ažuriranje komentara |
| HTTP metoda | PUT |
| URL | /api/ comments |
| URL parametri | 4?content=sadrzaj&user\_id=1&document\_id=11 |
| HTTP body parametri | (nema) |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | {      "id": 4,      "content": "sadrzaj",      "document\_id": "11",      "user\_id": "1",      "created\_at": "2024-01-30T14:03:48.000000Z",      "updated\_at": "2024-01-30T14:03:48.000000Z"  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | Brisanje komenatara |
| HTTP metoda | DELETE |
| URL | /api/ comments |
| URL parametri | 4 |
| HTTP body parametri | (nema) |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | --No Content-- |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | Registracija korisnika |
| HTTP metoda | POST |
| URL | /api/register |
| URL parametri | ?name=korisnik&email=korisnik@gmail.com&password=password&date\_of\_birth=2024-01-01&bio=biografija&password\_confirmation=password |
| HTTP body parametri | (nema) |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | {      "message": "User successfully registered",      "user": {          "name": "korisnik",          "email": "korisnik@gmail.com",          "date\_of\_birth": "2024-01-01",          "bio": "biografija",          "updated\_at": "2024-01-30T14:08:06.000000Z",          "created\_at": "2024-01-30T14:08:06.000000Z",          "id": 6      }  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | Prijava korisnika |
| HTTP metoda | POST |
| URL | /api/login |
| URL parametri | ?email=korisnik@gmail.com&password=password |
| HTTP body parametri | (nema) |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | {      "message": "Login successful",      "token": "10|isCOm8BedOz0ykBCehxNxzgtDH27RGuJyLLClqCo48f05a3d",      "user": {          "id": 6,          "name": "korisnik",          "email": "korisnik@gmail.com",          "email\_verified\_at": **null**,          "created\_at": "2024-01-30T14:08:06.000000Z",          "updated\_at": "2024-01-30T14:08:06.000000Z",          "date\_of\_birth": "2024-01-01",          "bio": "biografija"      }  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | Odjava korisnika |
| HTTP metoda | POST |
| URL | /api/logout |
| URL parametri | nema |
| HTTP body parametri | (nema) |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | {      "message": "Logout successful"  } |
| Format izlaznih parametara | application/json |

# Opis tehnologija korišćenih u aplikaciji

Laravel je PHP web aplikacijski okvir koji je odabran zbog svoje elegantne sintakse, robustnih funkcionalnosti i snažne zajednice. Laravel nudi bogat set alata i biblioteka koje olakšavaju razvoj, uključujući Eloquent ORM za rad sa bazom podataka, middleware za filtriranje HTTP zahteva, i Blade templating engine za efikasniju obradu pogleda. Laravel takođe pruža ugrađenu podršku za autentifikaciju, autorizaciju, rutiranje i sesije, čime se ubrzava razvoj sigurnih i visoko performantnih web aplikacija.

React je deklarativna, efikasna i fleksibilna JavaScript biblioteka za izgradnju korisničkih interfejsa, razvijena od strane Facebook-a. Odabrali smo React zbog njegove sposobnosti da omogući brz razvoj interaktivnih UI komponenata, koristeći koncept virtualnog DOM-a za minimiziranje potrebe za manipulacijom stvarnog DOM-a i time poboljšanje performansi aplikacije. React takođe podržava JSX, sintaksno proširenje za JavaScript, koje omogućava pisanje UI komponenata koristeći sintaksu sličnu HTML-u unutar JavaScript koda, čineći kod čistijim i lakšim za razumevanje.

Axios je obećavajuća JavaScript biblioteka koja se koristi za izvršavanje HTTP zahteva iz browsera i Node.js aplikacija. Izabrali smo Axios zbog njegove jednostavnosti upotrebe, podrške za Promise API, i mogućnosti da se lako konfiguriše za potrebe naše aplikacije. Axios omogućava slanje asinhronih HTTP zahteva ka RESTful API endpointima, podržavajući globalne konfiguracije, presretanje zahteva i odgovora, te automatsko transformisanje JSON podataka. Korišćenje Axiosa u našoj aplikaciji olakšava komunikaciju između frontend-a i backend-a, čineći razmenu podataka brzom i efikasnom.

# Korisničko uputstvo

Pocetna stranica

Slika na kojoj se nalazi tekst, snimak ekrana, Font, softver

Opis je automatski generisan

Stranice za login i registraciju

Slika na kojoj se nalazi tekst, snimak ekrana, softver, Operativni sistem

Opis je automatski generisan

Slika na kojoj se nalazi tekst, snimak ekrana, dizajn

Opis je automatski generisan

Stranica za prikaz tabele sa dokumentima

Stranica za unos novog dokumenta

Slika na kojoj se nalazi tekst, snimak ekrana, Font, softver

Opis je automatski generisan

Forma za editovanje dokumenata

Slika na kojoj se nalazi tekst, snimak ekrana, Font, broj

Opis je automatski generisan

Stranica za prikaz dokumenata sa spoljnog apija

Slika na kojoj se nalazi tekst, snimak ekrana, Font, broj

Opis je automatski generisan

# Prikaz delova koda

U ovom poglavlju ćemo prikazati neke fajlove I njihovo objašnjenje.

### Laravel

U ovom radu imamo sledece migracije:

Slika na kojoj se nalazi tekst, snimak ekrana, Font

Opis je automatski generisan

Slika 1 prikaz svih migracija

Migracija create\_documents\_table kreira tabelu documents sa kolonama za title, content, file\_path, author\_id, downloads, is\_public, category\_id, i tags.

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

return new class extends Migration

{

    /\*\*

     \* Run the migrations.

     \*

     \* @return void

     \*/

    public function up()

    {

        Schema::create('documents', function (Blueprint $table) {

            $table->id();

            $table->string('title');

            $table->text('content');

            $table->string('file\_path')->nullable();

            $table->unsignedBigInteger('author\_id');

            $table->unsignedBigInteger('category\_id');

            $table->integer('downloads')->default(0);

            $table->boolean('is\_public')->default(true);

            $table->text('tags');

            $table->timestamps();

            $table->softDeletes();

        });

    }

    /\*\*

     \* Reverse the migrations.

     \*

     \* @return void

     \*/

    public function down()

    {

        Schema::dropIfExists('documents');

    }

};

Migracija add\_foreign\_key\_to\_documents\_table dodaje strani ključ category\_id u tabelu documents, povezujući je sa tabelom categories.

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

return new class extends Migration

{

    /\*\*

     \* Run the migrations.

     \*

     \* @return void

     \*/

    public function up()

    {

        Schema::table('documents', function (Blueprint $table) {

            $table->foreign('author\_id')->references('id')->on('users');

            $table->foreign('category\_id')->references('id')->on('categories');

        });

    }

    /\*\*

     \* Reverse the migrations.

     \*

     \* @return void

     \*/

    public function down()

    {

        Schema::table('documents', function (Blueprint $table) {

            $table->dropForeign('author\_id');

        });

    }

};

Migracija add\_columns\_to\_tags\_table dodaje nove kolone color i slug u tabelu tags.

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

return new class extends Migration

{

    /\*\*

     \* Run the migrations.

     \*

     \* @return void

     \*/

    public function up()

    {

        Schema::table('tags', function (Blueprint $table) {

            $table->string('slug')->unique(); // Polje za jedinstveni slug

        });

    }

    /\*\*

     \* Reverse the migrations.

     \*

     \* @return void

     \*/

    public function down()

    {

        Schema::table('tags', function (Blueprint $table) {

            $table->dropColumn('slug');

        });

    }

};

Migracija add\_foreign\_key\_to\_comments\_table uključuje dodavanje stranih ključeva document\_id i user\_id u tabelu comments, povezujući je sa tabelama documents i users.

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

return new class extends Migration

{

    /\*\*

     \* Run the migrations.

     \*

     \* @return void

     \*/

    public function up()

    {

        Schema::table('comments', function (Blueprint $table) {

            $table->foreign('document\_id')->references('id')->on('documents');

            $table->foreign('user\_id')->references('id')->on('users');

        });

    }

    /\*\*

     \* Reverse the migrations.

     \*

     \* @return void

     \*/

    public function down()

    {

        Schema::table('comments', function (Blueprint $table) {

            $table->dropForeign('document\_id');

            $table->dropForeign('document\_id');

        });

    }

};

Migracija drop\_description\_column\_from\_categories\_table uklanja kolonu description iz tabele categories.

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

return new class extends Migration

{

    /\*\*

     \* Run the migrations.

     \*

     \* @return void

     \*/

    public function up()

    {

        Schema::table('categories', function (Blueprint $table) {

            $table->dropColumn('description');

        });

    }

    /\*\*

     \* Reverse the migrations.

     \*

     \* @return void

     \*/

    public function down()

    {

        Schema::table('categories', function (Blueprint $table) {

            $table->text('description');

        });

    }

};

Migracija add\_columns\_to\_users\_table dodaje kolone date\_of\_birth i bio u tabelu users.

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

return new class extends Migration

{

    /\*\*

     \* Run the migrations.

     \*

     \* @return void

     \*/

    public function up()

    {

        Schema::table('users', function (Blueprint $table) {

            $table->date('date\_of\_birth');

            $table->text('bio');

        });

    }

    /\*\*

     \* Reverse the migrations.

     \*

     \* @return void

     \*/

    public function down()

    {

        Schema::table('users', function (Blueprint $table) {

            $table->dropColumn('date\_of\_birth');

            $table->dropColumn('bio');

        });

    }

};

Migracija rename\_columns\_in\_comments\_table menja imena nekih kolona u tabeli comments.

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

return new class extends Migration

{

    /\*\*

     \* Run the migrations.

     \*

     \* @return void

     \*/

    public function up()

    {

        Schema::table('comments', function (Blueprint $table) {

            $table->renameColumn('tekst','content');

        });

    }

    /\*\*

     \* Reverse the migrations.

     \*

     \* @return void

     \*/

    public function down()

    {

        Schema::table('comments', function (Blueprint $table) {

            $table->renameColumn('content','tekst');

        });

    }

};

U ovom radu su kreirani sledeci kontroleri

Slika na kojoj se nalazi tekst, snimak ekrana, Font

Opis je automatski generisan

Slika 2 prikaz kreiranih kontrolera

U ovom AuthControlleru, imamo tri glavne metode za rad sa korisnicima: register, login i logout.

* Register: Ova metoda prima zahtev za registraciju korisnika. zahtev mora sadrzati korisnikovo ime, email, lozinku, datum rođenja i biografiju. Metoda koristi validaciju da bude sigurna da su svi potrebni podaci uneti i da su validni. Ako validacija ne prodje, vraca se odgovor sa greškama. Ako validacija prodje, korisnik se kreira u bazi podataka sa unetim podacima.
* Login: Ova metoda prima zahtev za prijavu korisnika koji mora sadrzati email i lozinku. Metoda koristi validaciju da bude sigurna da su svi potrebni podaci uneti i da su validni. Ako validacija ne prodje, vraca se odgovor sa greškama. Ako validacija prodje, korisnik se prijavljuje u aplikaciju. Ako prijava ne uspe, vraca se odgovor sa greškom. Ako prijava uspe, kreira se token za autentifikaciju i vraca se zajedno sa korisnikovim podacima.
* Logout: Ova metoda brise token za autentifikaciju trenutno prijavljenog korisnika. Nakon brisanja tokena, korisnik se automatski odjavljuje iz aplikacije.

<?php

namespace App\Http\Controllers;

use App\Models\User;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Auth;

use Illuminate\Support\Facades\Hash;

use Illuminate\Support\Facades\Validator;

class AuthController extends Controller

{

    public function register(Request $request)

    {

        $validator = Validator::make($request->all(), [

            'name' => 'required|string|max:255',

            'email' => 'required|string|email|max:255|unique:users',

            'password' => 'required|string|min:8|confirmed',

            'date\_of\_birth' => 'required|date',

            'bio' => 'required|string'

        ]);

        if ($validator->fails()) {

            return response()->json($validator->errors(), 422);

        }

        $user = User::create([

            'name' => $request->name,

            'email' => $request->email,

            'password' => Hash::make($request->password),

            'date\_of\_birth' => $request->date\_of\_birth,

            'bio' => $request->bio

        ]);

        return response()->json(['message' => 'User successfully registered', 'user' => $user], 201);

    }

    public function login(Request $request)

    {

        $validator = Validator::make($request->all(), [

            'email' => 'required|string|email|max:255',

            'password' => 'required|string|min:8'

        ]);

        if ($validator->fails()) {

            return response()->json($validator->errors(), 422);

        }

        if (!Auth::attempt($request->only('email', 'password'))) {

            return response()->json(['message' => 'Unauthorized'], 401);

        }

        $user = User::where('email', $request->email)->firstOrFail();

        $token = $user->createToken('auth\_token')->plainTextToken;

        return response()->json(['message' => 'Login successful', 'token' => $token,'user'=> $user]);

    }

    public function logout(Request $request)

    {

        $request->user()->currentAccessToken()->delete();

        return response()->json(['message' => 'Logout successful']);

    }

    public function changePassword(Request $request)

    {

        $validator = Validator::make($request->all(), [

            'current\_password' => 'required|string|min:8',

            'new\_password' => 'required|string|min:8|confirmed',

        ]);

        if ($validator->fails()) {

            return response()->json($validator->errors(), 422);

        }

        if (!Hash::check($request->current\_password, $request->user()->password)) {

            return response()->json(['message' => 'The current password is incorrect.'], 403);

        }

        $request->user()->update([

            'password' => Hash::make($request->new\_password),

        ]);

        return response()->json(['message' => 'Password successfully changed.']);

    }

}

Dodatno, u ovom AuthControlleru imamo i metodu za promenu lozinke. Ova metoda prima zahtev za promenu lozinke.

Kontroler PasswordResetController ima dve funkcije koje omogućavaju korisnicima da resetuju svoju lozinku. Prva funkcija sendResetLinkEmail služi za slanje linka za resetovanje lozinke na email koji korisnik unese. Ova funkcija prvo validira email korisnika, a zatim koristi Laravelovu funkciju Password::sendResetLink za slanje linka. U zavisnosti od rezultata, vraća odgovarajući JSON odgovor sa porukom uspeha ili greške. Druga funkcija reset omogućava korisniku da postavi novu lozinku koristeći token dobijen putem emaila. Nakon validacije ulaznih podataka, koristi se Password::reset za promenu lozinke. Ako je resetovanje uspešno, korisnikov remember\_token se ažurira, a lozinka se čuva u hashiranom obliku u bazi podataka, nakon čega se vraća odgovor uspeha ili greške.

<?php

namespace App\Http\Controllers;

use App\Models\User;

use Carbon\Carbon;

use Illuminate\Auth\Events\PasswordReset;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\DB;

use Illuminate\Support\Facades\Hash;

use Illuminate\Support\Facades\Password;

use Illuminate\Support\Str;

class PasswordResetController extends Controller

{

    public function sendResetLinkEmail(Request $request)

    {

      $request->validate(['email' => 'required|email']);

        $status = Password::sendResetLink(

            $request->only('email')

        );

        return $status === Password::RESET\_LINK\_SENT

                    ? response()->json(['message' => \_\_($status)])

                    : response()->json(['email' => \_\_($status)], 422);

    }

    public function reset(Request $request)

    {

        $request->validate([

            'token' => 'required',

            'email' => 'required|email',

            'password' => 'required|min:8|confirmed',

        ]);

        $status = Password::reset(

            $request->only('email', 'password', 'password\_confirmation', 'token'),

            function ($user, $password) {

                $user->forceFill([

                    'password' => Hash::make($password)

                ])->setRememberToken(Str::random(60));

                $user->save();

                event(new PasswordReset($user));

            }

        );

        return $status === Password::PASSWORD\_RESET

                    ? response()->json(['message' => \_\_($status)])

                    : response()->json(['email' => \_\_($status)], 422);

    }

}

DocumentController sadrži nekoliko metoda za upravljanje dokumentima. Metoda index vraća sve dokumente iz baze podataka. show vraća određeni dokument na osnovu ID-a. store služi za kreiranje novog dokumenta, gde se prvo vrši validacija podataka, a zatim čuva dokument u bazi sa opcionalnim uploadom fajla. Metoda update omogućava ažuriranje postojećeg dokumenta, uključujući i zamenu fajla ukoliko je novi fajl poslat. destroy briše dokument, uz proveru da li je dokument već soft-deleted. search omogućava filtriranje dokumenata po različitim kriterijumima kao što su naslov, sadržaj, autor, kategorija i tagovi. download omogućava preuzimanje fajla dokumenta, uz proveru da li fajl postoji. Takođe, postoji pomoćna funkcija getMimeType koja vraća MIME tip fajla na osnovu njegove ekstenzije.

<?php

namespace App\Http\Controllers;

use App\Models\Document;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Storage;

use Illuminate\Support\Facades\Validator;

class DocumentController extends Controller

{

    public function index()

    {

        $documents = Document::all();

        return response()->json($documents);

    }

    public function show($id)

    {

        $document = Document::findOrFail($id);

        return response()->json($document);

    }

    public function store(Request $request)

    {

        $validator = Validator::make($request->all(), [

            'title' => 'required|max:255',

            'content' => 'required',

            'author\_id' => 'required|exists:users,id',

            'category\_id' => 'required|exists:categories,id',

            'tags' => 'required|array',      //Ovo pravilo zahteva da polje 'tags' bude prisutno u zahtevu (ne smije biti prazno ili null) i da bude niz (array).

            'tags.\*' => 'distinct|exists:tags,id',  //Ovo pravilo se primenjuje na svaki element niza 'tags'.  //distinct' proverava da svaki tag u nizu bude jedinstven, tj. da se ne ponavlja više puta.

            'file' => 'required|file', // Pravilo za validaciju uploadovanog fajla

         //   'is\_public' => 'required',

            'downloads' => 'required|numeric',

        ]);

        if ($validator->fails()) {

            return response()->json($validator->errors(), 422);

        }

        $documentData = $validator->validated();

        // Obrada uploada fajla

        if ($request->hasFile('file')) {

            $file = $request->file('file');

            $filePath = $file->store('documents', 'public'); // Sačuvaj fajl i vrati putanju

            $documentData['file\_path'] = $filePath;

        }

        $documentData['is\_public'] = 0;

        // Kreiraj dokument i postavi kategoriju

        $document = Document::create($documentData);

        // Kako su tagovi predstavljeni kao niz, treba ih konvertovati u string pre snimanja

       // $document->tags()->sync($request->tags);   //Ova linija koda se koristi za uspostavljanje veze many-to-many između Document i Tag. Metoda sync prima niz ID-ova tagova i ažurira odgovarajuću pivot tabelu, uklanjajući sve postojeće veze i dodajući nove. S obzirom na vašu strukturu, ovo je ispravno za rad sa tagovima.

        return response()->json($document, 201);

    }

    public function update(Request $request, $id)

    {

        $document = Document::findOrFail($id);

        $validator = Validator::make($request->all(), [

            'title' => 'sometimes|required|max:255',

            'content' => 'sometimes|required',

            'author\_id' => 'sometimes|required|exists:users,id',

            'category\_id' => 'sometimes|required|exists:categories,id',

            'tags' => 'sometimes|required|array',

            'tags.\*' => 'distinct|exists:tags,id',

            'file' => 'sometimes|file',

            'is\_public' => 'sometimes|boolean',

            'downloads' => 'sometimes|numeric',

        ]);

        if ($validator->fails()) {

            return response()->json($validator->errors(), 422);

        }

        $documentData = $validator->validated();

        // Obrada uploada fajla

        if ($request->hasFile('file')) {

            // Pre nego što sačuvamo novi fajl, trebalo bi da obrisemo stari

            Storage::delete($document->file\_path);

            $file = $request->file('file');

            $filePath = $file->store('documents', 'public'); // Sačuvaj fajl i vrati putanju

            $documentData['file\_path'] = $filePath;

        }

        // Konvertovanje niza tagova u string pre snimanja, ako su tagovi prosleđeni

        if (isset($documentData['tags'])) {

            $documentData['tags'] = implode(',', $request->tags);

        }

        $document->update($documentData);

        return response()->json($document);

    }

    public function destroy($id)  //posto nas model koristi softDeletes, onda metoda destroy mora da bude implementirana na malo drugaciji nacin

    {

        $document = Document::withTrashed()->findOrFail($id);

        if ($document->trashed()) {

            //  dokument je već obrisan

            return response()->json(['message' => 'Document has already been deleted.'], 404);

        } else {

            $document->delete();

            return response()->json(null, 204);

        }

    }

    public function search(Request $request)

{

    // Počinjemo sa kreiranjem osnovnog upita

    $query = Document::query();

    // Filtriramo po naslovu ako je prosleđen kao parametar

    if ($request->has('title')) {

        $query->where('title', 'like', '%' . $request->title . '%');

    }

    // Filtriramo po sadržaju ako je prosleđen kao parametar

    if ($request->has('content')) {

        $query->where('content', 'like', '%' . $request->content . '%');

    }

    // Filtriramo po autoru ako je prosleđen kao parametar

    if ($request->has('author\_id')) {

        $query->where('author\_id', $request->author\_id);

    }

    // Filtriramo po kategoriji ako je prosleđen kao parametar

    if ($request->has('category\_id')) {

        $query->where('category\_id', $request->category\_id);

    }

    // Filtriramo po tagovima ako su prosleđeni kao parametar

    if ($request->has('tags')) {

        $tags = $request->tags;

        $query->whereHas('tags', function ($query) use ($tags) {

            $query->whereIn('id', $tags);

        });

    }

    // Filtriramo po dostupnosti ako je prosleđen kao parametar

    if ($request->has('is\_public')) {

        $query->where('is\_public', $request->is\_public);

    }

    // Dobijamo filtrirane dokumente

    $documents = $query->get();

    // Vraćamo dokumente kao JSON odgovor

    return response()->json($documents);

    }

    public function download($id)

    {

        $document = Document::findOrFail($id);

        $filePath = storage\_path('app/public/' . $document->file\_path);

        if (!file\_exists($filePath)) {

            return response()->json(['message' => 'File not found.'], 404);

        }

        $fileName = basename($filePath);

        $fileExtension = pathinfo($filePath, PATHINFO\_EXTENSION);

        // Određivanje MIME tipa na osnovu ekstenzije fajla

        $mimeType = $this->getMimeType($fileExtension);

        // Preuzimanje fajla sa ispravnim MIME tipom

        return response()->download($filePath, $fileName, [

            'Content-Type' => $mimeType,

            'Content-Disposition' => 'attachment; filename="' . $fileName . '"'

        ]);

    }

    // Pomoćna funkcija za dobijanje MIME tipa

    private function getMimeType($extension)

    {

        $mimeTypes = [

            'pdf' => 'application/pdf',

            'jpg' => 'image/jpeg',

            'jpeg' => 'image/jpeg',

            'png' => 'image/png',

            'txt' => 'text/plain',

        ];

        return $mimeTypes[$extension] ?? 'application/octet-stream';

    }

}

Prva ruta je GET /api/documents/search, koja koristi DocumentController za pretragu dokumenata po kriterijumima kao što su naslov, kategorija i oznake. Druga ruta, GET /api/documents/{document}/download, omogućava preuzimanje dokumenta i takođe koristi DocumentController.

Za resetovanje lozinke postoje dve rute: POST /api/forgot-password i POST /api/reset-password, koje koriste PasswordResetController za slanje linka za resetovanje lozinke i sam proces resetovanja lozinke.

Takođe, postoje rute za autentifikaciju korisnika: POST /api/register i POST /api/login koje koriste AuthController za registraciju novih korisnika i prijavljivanje postojećih korisnika.

Unutar grupe koja zahteva autentifikaciju korisnika (auth:sanctum), nalaze se dodatne rute. Postoji ruta POST /api/logout za odjavljivanje korisnika, kao i više ruta za upravljanje dokumentima, komentarima, oznakama i kategorijama kroz odgovarajuće kontrolere (DocumentController, CommentController, TagController, CategoryController). Na kraju, postoji ruta POST /api/changePassword za promenu korisničke lozinke. Ove rute omogućavaju kreiranje, čitanje, ažuriranje i brisanje (CRUD operacije) nad pomenutim resursima.

<?php

use App\Http\Controllers\AuthController;

use App\Http\Controllers\CategoryController;

use App\Http\Controllers\CommentController;

use App\Http\Controllers\DocumentController;

use App\Http\Controllers\PasswordResetController;

use App\Http\Controllers\TagController;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Route;

/\*

|--------------------------------------------------------------------------

| API Routes

|--------------------------------------------------------------------------

|

| Here is where you can register API routes for your application. These

| routes are loaded by the RouteServiceProvider within a group which

| is assigned the "api" middleware group. Enjoy building your API!

|

\*/

Route::get('/documents/search', [DocumentController::class, 'search']);  //GET /api/documents/search?title=izveštaj&category\_id=3&tags[]=1&tags[]=5&is\_public=1

Route::get('/documents/{document}/download', [DocumentController::class, 'download']);

Route::post('/forgot-password', [PasswordResetController::class, 'sendResetLinkEmail']);

Route::post('/reset-password', [PasswordResetController::class, 'reset']);

Route::post('/register', [AuthController::class, 'register']);

Route::post('/login', [AuthController::class, 'login']);

Route::middleware('auth:sanctum')->group(function () {

    Route::post('/logout', [AuthController::class, 'logout']);

    Route::apiResource('documents', DocumentController::class);

        Route::apiResource('comments', CommentController::class);

        Route::apiResource('tags', TagController::class);

        Route::apiResource('categories', CategoryController::class);

    Route::post('/changePassword', [AuthController::class, 'changePassword']);

});

### React

Komponenta Register je odgovorna za proces registracije korisnika u web aplikaciji. Ova komponenta sadrži formu sa poljima za unos osnovnih informacija, kao što su ime, email adresa, lozinka, datum rođenja i biografija korisnika.

Kada korisnik popuni ova polja sa svojim podacima i pritisne dugme "Register", komponenta aktivira funkciju handleSubmit. Ova funkcija ima ključnu ulogu u procesu registracije jer šalje Axios POST zahtev prema određenom API endpointu (u ovom slučaju, 'http://127.0.0.1:8000/api/register') kako bi se korisnički podaci poslali na server u svrhu registracije.

Funkcija handleChange se poziva svaki put kada korisnik unese ili izmeni neku od vrednosti u poljima forme. Korišcenjem handleChange, komponenta prati promene unosa i ažurira formData stanje kako bi reflektovalo trenutne vrednosti unosa.

import React, { useState } from 'react';

import './Register.css';

import InputField from './InputField';

import axios from 'axios';

import { useNavigate } from 'react-router-dom';

const Register = () => {

  const [formData, setFormData] = useState({

    name: 'ana',

    email: 'anaa@gmail.com',

    password: 'anaana1234',

    password\_confirmation: 'anaana1234',

    date\_of\_birth: '2000-01-01',

    bio: 'ana bio'

  });

  const navigate = useNavigate();

  const handleChange = (e) => {

    setFormData({ ...formData, [e.target.name]: e.target.value });

  };

  const handleSubmit = async (e) => {

    e.preventDefault();

    try {

      const response = await axios.post('http://127.0.0.1:8000/api/register', formData);

      console.log(response.data);

      navigate('/login');

    } catch (error) {

      console.error(error);

    }

  };

  return (

    <div className="register-container">

      <div className="register-card">

        <h2>Register</h2>

        <form className="register-form" onSubmit={handleSubmit}>

          <InputField

            label="Name"

            type="text"

            id="name"

            name="name"

            value={formData.name}

            onChange={handleChange}

            required

          />

          <InputField

            label="Email"

            type="email"

            id="email"

            name="email"

            value={formData.email}

            onChange={handleChange}

            required

          />

          <InputField

            label="Password"

            type="password"

            id="password"

            name="password"

            value={formData.password}

            onChange={handleChange}

            required

          />

          <InputField

            label="Confirm Password"

            type="password"

            id="confirmPassword"

            name="password\_confirmation"

            value={formData.confirmPassword}

            onChange={handleChange}

            required

          />

          <InputField

            label="Date of Birth"

            type="date"

            id="dateOfBirth"

            name="date\_of\_birth"

            value={formData.dateOfBirth}

            onChange={handleChange}

            required

          />

          <InputField

            label="Bio"

            type="textarea"

            id="bio"

            name="bio"

            value={formData.bio}

            onChange={handleChange}

            required

          />

          <button type="submit" className="register-button">Register</button>

        </form>

      </div>

    </div>

  );

};

export default Register;

Komponenta Login je sluzi za prijavu korisnika u aplikaciju. Ova komponenta prikazuje formu sa dva polja za unos - jedno za unos email adrese i drugo za unos lozinke.

Kada korisnik unese svoje podatke i pritisne dugme "Login", komponenta aktivira funkciju handleSubmit. Ova funkcija šalje Axios POST zahtjev prema određenom API endpointu (u ovom slučaju, 'http://127.0.0.1:8000/api/login') kako bi korisnički podaci bili poslani na server radi provere identiteta. Ako su korisnički podaci tačni i uspešno autentifikovani na serveru, server će odgovoriti sa tokenom koji se koristi za autentifikaciju korisnika u budućim zahtevima prema zaštićenim resursima. Takođe, korisnički ID se takođe čuva u sesiji radi dalje identifikacije.

U slučaju neuspešnog prijavljivanja, greške će biti uhvaćene i prikazane kao poruke u konzoli.

import React, { useState } from 'react';

import './Login.css';

import { FaUser } from "react-icons/fa";

import InputField from './InputField';

import axios from 'axios';

import { useNavigate } from 'react-router-dom';

const Login = ({setToken}) => {

  let navigate=useNavigate();

  const [formData, setFormData] = useState({

    email: 'ana@gmail.com',

    password: 'anaana1234',

  });

  const handleChange = (e) => {

    setFormData({ ...formData, [e.target.name]: e.target.value });

  };

  const handleSubmit = async (e) => {

    e.preventDefault();

    try {

      const response = await axios.post('http://127.0.0.1:8000/api/login', formData);

      const { token } = response.data;

      console.log(response.data);

      setToken(token)

      sessionStorage.setItem('token', token);

      sessionStorage.setItem('id',response.data.user.id);

      console.log('Login successful');

      navigate('/docs');

    } catch (error) {

      console.error(error);

    }

  };

  return (

    <div className="login-container">

      <div className="login-card">

        <div className="login-icon">

          <FaUser />

        </div>

        <form className="login-form" onSubmit={handleSubmit}>

          <InputField

            label="Email Address"

            type="email"

            id="email"

            name="email"

            required

            value={formData.email}

            onChange={handleChange}

          />

          <InputField

            label="Password"

            type="password"

            id="password"

            name="password"

            required

            value={formData.password}

            onChange={handleChange}

          />

          <button type="submit" className="login-button">Login</button>

        </form>

        <div className="login-footer">

          <a href="#signup">Signup</a>

          <a href="#forgot-password">Forgot Password?</a>

        </div>

      </div>

    </div>

  );

};

export default Login;

Ove tri komponente (ispod) predstavljaju korisnički definisane kuke (custom hooks) u React aplikaciji. Kuke su funkcije koje olakšavaju ponovno korišćenje i organizaciju koda za zajedničke funkcionalnosti. Na primer useCategories omogućava aplikaciji da dohvati i koristi informacije o kategorijama iz API-ja.

Koristi useState kako bi čuvala listu kategorija i useEffect kako bi izvršila HTTP GET zahtev ka određenom API endpointu za kategorije.

Kada se podaci uspešno dohvate, setuje ih u stanje categories koje se može koristiti u komponentama koje koriste ovu kuku.

import { useState, useEffect } from 'react';

import axios from 'axios';

const useTags = () => {

    const [tags, setTags] = useState([]);

    useEffect(() => {

        axios.get('http://127.0.0.1:8000/api/tags')

            .then(response => {

                setTags(response.data);

            })

            .catch(error => {

                console.error('Error fetching tags:', error);

            });

    }, []);

    return {tags,setTags};

};

export default useTags;

import { useState, useEffect } from 'react';

import axios from 'axios';

const useDocuments = () => {

    const [documents, setDocuments] = useState([]);

    const [error, setError] = useState(null);

    useEffect(() => {

        axios.get('http://127.0.0.1:8000/api/documents')

            .then(response => {

                setDocuments(response.data);

            })

            .catch(error => {

                console.error('There was an error fetching the documents:', error);

                setError(error);

            });

    }, []);

    return { documents,setDocuments, error };

};

export default useDocuments;

import { useState, useEffect } from 'react';

import axios from 'axios';

const useCategories = () => {

    const [categories, setCategories] = useState([]);

    useEffect(() => {

        axios.get('http://127.0.0.1:8000/api/categories')

            .then(response => {

                setCategories(response.data);

            })

            .catch(error => {

                console.error('Error fetching categories:', error);

            });

    }, []);

    return {categories,setCategories};

};

export default useCategories;

DocumentsTable komponenta ima više funkcionalnosti, uključujući prikaz, pretragu, uređivanje, preuzimanje i brisanje dokumenata. Koristi se za upravljanje dokumentima unutar aplikacije.

Komponenta prvo koristi tri custom kuke (useDocuments, useCategories i useTags) kako bi dohvatila podatke o dokumentima, kategorijama i oznakama sa servera.

U tabeli se prikazuju podaci o dokumentima, kao što su ID, naslov, sadržaj, autor, kategorija, oznake, status i akcije (preuzimanje, brisanje i uređivanje). Tabela je paginirana, što znači da se prikazuje određeni broj dokumenata po stranici, a korisnici mogu da se kreću kroz stranice koristeći paginaciju.

Korisnicima je omogućeno filtriranje dokumenata po kategoriji i oznakama, što olakšava pronalaženje određenih dokumenata.

Takođe, komponenta omogućava uređivanje dokumenata putem modala. Kada se izabere dokument za uređivanje, otvara se modal sa formom za izmenu podataka o dokumentu. Nakon što se izmene podaci i sačuvaju, dokument se ažurira na serveru i prikazuje se ažuriran prikaz u tabeli.

Osim uređivanja, komponenta omogućava i preuzimanje dokumenata. Kada korisnik klikne na dugme za preuzimanje, dokument se preuzima sa servera i spašava na računar korisnika.

Takođe, komponenta omogućava i brisanje dokumenata. Kada korisnik klikne na dugme za brisanje, dokument se briše sa servera i uklanja iz tabele.

import React, { useState, useEffect } from 'react';

import axios from 'axios';

import './DocumentsTable.css';

import DocumentTableRow from './DocumentTableRow';

import useDocuments from '../customHooks/useDocuments';

import useCategories from '../customHooks/useCategories';

import useTags from '../customHooks/useTags';

import DocumentEditModal from './update/DocumentEditModal';

const DocumentsTable = () => {

    const { documents,setDocuments, error } = useDocuments();

    const [isEditModalOpen, setIsEditModalOpen] = useState(false);

    const [currentDocument, setCurrentDocument] = useState(null);

    const [filterByTags, setFilterByTags] = useState([]);

    const [filterByCategory, setFilterByCategory] = useState('');

    const [filteredDocuments, setFilteredDocuments] = useState([]);

    const {categories,setCategories} = useCategories();

    const {tags,setTags} = useTags();

    const [currentPage, setCurrentPage] = useState(1);

    const documentsPerPage = 3;

    const handleEdit = (document) => {

        setCurrentDocument(document);

        setIsEditModalOpen(true);

    };

    const handleSaveEdit = (documentId, editFormData) => {

        const token = sessionStorage.getItem('token');

        const config = {

            headers: {

                'Authorization': `Bearer ${token}`,

                'Content-Type': 'application/json' // Ako šaljete JSON

            }

        };

        const updatedData = {

            ...editFormData,

            tags: editFormData.tags.map(Number)

        };

        console.log(updatedData.tags)

        axios.put(`http://127.0.0.1:8000/api/documents/${documentId}`, updatedData, config)

            .then(response => {

                // Ažuriranje prikaza dokumenata

                setDocuments(documents.map(doc => doc.id === documentId ? {...doc, ...response.data} : doc));

                alert('Document updated successfully');

                setIsEditModalOpen(false); // Zatvaranje modala

            })

            .catch(error => {

                console.error('Error updating document:', error);

            });

    };

    // const [documents, setDocuments] = useState([]);

    // useEffect(() => {

    //     axios.get('http://127.0.0.1:8000/api/documents')

    //         .then(response => {

    //             setDocuments(response.data);

    //         })

    //         .catch(error => {

    //             console.error('There was an error fetching the documents:', error);

    //         });

    // }, []);

    const handleDownload = (documentId, fileName) => {

        axios({

            url: `http://127.0.0.1:8000/api/documents/${documentId}/download`,

            method: 'GET',

            responseType: 'blob', // Važno za preuzimanje fajlova  \*/////Opcija responseType: 'blob' u Axios zahtevu označava da želite da odgovor koji primate od servera bude predstavljen kao "Blob" objekt u JavaScriptu. "Blob" (Binary Large Object) je tip objekta koji se koristi za predstavljanje binarnih podataka, kao što su slike, audio datoteke, ili u ovom slučaju, datoteke koje preuzimate sa servera.

        })

        .then((response) => {

            const blob = new Blob([response.data], { type: response.headers['content-type'] });  //Ova linija koda stvara objekt Blob (Binary Large Object) koji predstavlja binarni podatak. response.data pretpostavlja se da sadrži binarni podatak koji je došao kao odgovor na HTTP zahtjev. response.headers['content-type'] koristi se za postavljanje tipa sadržaja (MIME tip) za taj objekt Blob, na osnovu informacija iz zaglavlja odgovora.

            const url = window.URL.createObjectURL(blob);  //Ova linija koda koristi window.URL.createObjectURL metodu kako bi se stvorila URL adresa koja se odnosi na Blob objekt stvoren u prethodnoj liniji. Ova URL adresa će biti privremeno dostupna i može se koristiti za preuzimanje sadržaja.

            const link = document.createElement('a'); //Ova linija koda stvara novi HTML element <a> (hipertekstualna veza) koji će se koristiti za kreiranje linka za preuzimanje datoteke.

            link.href = url;  //Ova linija koda postavlja href atribut HTML elementa <a> na prethodno stvorenu URL adresu, čime se veza povezuje s Blob objektom i omogućava preuzimanje sadržaja.

            link.setAttribute('download', fileName);  //Ova linija koda postavlja atribut download na HTML elementu <a> i dodjeljuje mu vrijednost fileName. To omogućava korisnicima da preuzmu sadržaj sa zadanim imenom datoteke kada kliknu na vezu.

            document.body.appendChild(link); //Ova linija koda dodaje HTML element <a> u telo (body) HTML dokumenta, čime se omogućava njegovo prikazivanje na stranici.

            link.click(); //pokrece preuzimanje fajla

            link.parentNode.removeChild(link); //Na kraju, ova linija koda uklanja HTML element <a> iz dokumenta kako bi se oslobodili resursi i očistila stranica od tog elementa nakon preuzimanja.

        })

        .catch((error) => {

            console.error('There was an error downloading the file:', error);

        });

    };

    const handleDelete = (documentId) => {

        const token = sessionStorage.getItem('token');

        const config = {

            headers: {

                'Authorization': `Bearer ${token}`

            }

        };

        axios.delete(`http://127.0.0.1:8000/api/documents/${documentId}`, config)

            .then(() => {

                setDocuments(documents.filter(document => document.id !== documentId));

            })

            .catch((error) => {

                console.error('Došlo je do greške prilikom brisanja dokumenta:', error);

            });

    };

    useEffect(() => {

        const filterDocuments = () => {

            let updatedFilteredDocuments = [...documents];

            if (filterByCategory) {

                updatedFilteredDocuments = updatedFilteredDocuments.filter(document => document.category\_id.toString() === filterByCategory);

            }

            if (filterByTags.length > 0) {

                updatedFilteredDocuments = updatedFilteredDocuments.filter(document =>

                    filterByTags.every(tag => document.tags.map(tag => tag.toString()).includes(tag))

                );

            }

            setFilteredDocuments(updatedFilteredDocuments);

        };

        filterDocuments();

    }, [documents, filterByTags, filterByCategory]);   // Ovde primenjujemo filtere svaki put kada se promeni stanje `documents`, `filterByTags`, ili `filterByCategory`

    if (error) {

        return <div>Došlo je do greške: {error.message}</div>;

    }

    const indexOfLastDocument = currentPage \* documentsPerPage;

    const indexOfFirstDocument = indexOfLastDocument - documentsPerPage;

    const currentDocuments = filteredDocuments.slice(indexOfFirstDocument, indexOfLastDocument);

    const paginate = (pageNumber) => setCurrentPage(pageNumber);

    const pageNumbers = [];

    for (let i = 1; i <= Math.ceil(filteredDocuments.length / documentsPerPage); i++) {

      pageNumbers.push(i);

    }

    const renderPageNumbers = pageNumbers.map(number => {

        return (

          <li key={number} className='page-item'>

            <button onClick={() => paginate(number)}   className='page-link'>

              {number}

            </button>

          </li>

        );

      });

    return (

        <div className="documents-container">

            <div className="filter-container">

                <div className="input-group">

                    <label htmlFor="filter-category">Filter by Category</label>

                    <select id="filter-category" value={filterByCategory} onChange={(e) => setFilterByCategory(e.target.value)}>

                    <option value="">All Categories</option>

                    {categories.map((category) => (

                        <option key={category.id} value={category.id}>

                        {category.name}

                        </option>

                    ))}

                    </select>

                </div>

                <div className="input-group">

                    <label htmlFor="filter-tags">Filter by Tags</label>

                    <select id="filter-tags" multiple value={filterByTags} onChange={(e) => {

                    const options = e.target.options;

                    const value = [];

                    for (let i = 0, l = options.length; i < l; i++) {

                        if (options[i].selected) {

                        value.push(options[i].value);

                        }

                    }

                    setFilterByTags(value);

                    }}>

                    {tags.map((tag) => (

                        <option key={tag.id} value={tag.id}>

                        {tag.name}

                        </option>

                    ))}

                    </select>

                </div>

                </div>

            <table className="documents-table">

                <thead>

                    <tr>

                        <th>ID</th>

                        <th>Title</th>

                        <th>Content</th>

                        <th>Author ID</th>

                        <th>Category ID</th>

                        <th>Tags</th>

                        <th>Public</th>

                        <th>Download</th>

                        <th>Delete</th>

                        <th>Edit</th>

                    </tr>

                </thead>

                <tbody>

                    {currentDocuments.map((document) => (

                        <DocumentTableRow

                        key={document.id}

                        document={document}

                        onDelete={handleDelete}

                        onDownload={handleDownload}

                        onEdit={handleEdit}

                        />

                    ))}

                    </tbody>

            </table>

            <div>

            <ul className='pagination'>

                {renderPageNumbers}

            </ul>

            </div>

            {isEditModalOpen && currentDocument && (

                <DocumentEditModal

                    document={currentDocument}

                    categories={categories}

                    tags={tags}

                    onSave={handleSaveEdit}

                    onClose={() => setIsEditModalOpen(false)}

                />

            )}

        </div>

    );

};

export default DocumentsTable;

U kodu je definisana funkcionalna komponenta Statistike, koja ima za cilj prikazivanje statističkih podataka o broju fajlova u različitim kategorijama kroz bar grafikon. Prilikom prvog renderovanja komponente, koristi se useEffect hook kako bi se izvršila akcija slanja GET zahteva ka serveru kako bi se dobili statistički podaci. Ovi podaci se zatim čuvaju u stanju komponente korišćenjem useState hook-a.

Nakon što se podaci o statistici dobiju od servera, koriste se za formiranje podataka neophodnih za prikaz bar grafikona. Na x-osi grafikona prikazuju se imena kategorija, dok se na y-osi nalazi broj fajlova u svakoj kategoriji. Opcije grafikona, kao što su podešavanja y-ose da počinje od nule i druga podešavanja, takođe su definisane kako bi se grafikon pravilno prikazao.

Konačno, podaci i opcije se prosleđuju komponenti Bar iz biblioteke react-chartjs-2 kako bi se prikazao bar grafikon sa statistikama. Ovaj kod omogućava interaktivno prikazivanje statističkih podataka o broju fajlova u različitim kategorijama, što omogućava bolje razumevanje distribucije fajlova u sistemu.

import React, { useState, useEffect } from 'react';

import axios from 'axios';

import { Bar } from 'react-chartjs-2';

import { Chart, CategoryScale, LinearScale, BarElement, Title, Tooltip, Legend } from 'chart.js';

Chart.register(CategoryScale, LinearScale, BarElement, Title, Tooltip, Legend);

const Statistike = () => {

  const [statistics, setStatistics] = useState([]);

  useEffect(() => {

    const token = sessionStorage.getItem('token');

    const headers = {

      'Authorization': `Bearer ${token}`

    };

    axios.get('http://127.0.0.1:8000/api/statistics', { headers })

      .then(response => {

        setStatistics(response.data);

      })

      .catch(error => {

        console.error('Error fetching statistics:', error);

      });

  }, []);

  const chartData = {

    labels: statistics.map(category => category.category\_name),

    datasets: [

      {

        label: 'Broj fajlova',

        data: statistics.map(category => category.document\_count),

        backgroundColor: 'rgba(75,192,192,0.2)',

        borderColor: 'rgba(75,192,192,1)',

        borderWidth: 1,

      },

    ],

};

const options = {

    maintainAspectRatio: false,

    scales: {

        y: {

            beginAtZero: true,

            ticks: {

                stepSize: 1,

            },

        },

        x: {

        },

    },

};

  return (

    <div className="container" style={{width:"80vh",marginLeft:"10vh"}}>

      <h2>Statistike</h2>

      <div className="chart-container">

      <Bar

        data={chartData}

        options={options}

        />

      </div>

    </div>

  );

};

export default Statistike;

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

import { QueryClient, QueryClientProvider } from 'react-query';

const queryClient = new QueryClient();

ReactDOM.render(

  <QueryClientProvider client={queryClient}>

    <App />

  </QueryClientProvider>,

  document.getElementById('root')

);

// If you want to start measuring performance in your app, pass a function

// to log results (for example: reportWebVitals(console.log))

// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals

reportWebVitals();

U gornjem kodu, primenili smo kesiranje podataka u React aplikaciji koristeći React Query biblioteku. Ovaj proces se sastoji od nekoliko ključnih koraka. Prvo, uvezli smo neophodne komponente i funkcionalnosti iz React Query biblioteke. Zatim smo kreirali instancu QueryClient, koja predstavlja centralno mesto za upravljanje kesiranim podacima. Nakon toga, omotali smo glavnu komponentu aplikacije u QueryClientProvider, što omogućava pristup QueryClient-u u celoj aplikaciji. Kroz ovaj korak, prosledili smo instancu QueryClient-a kao prop client unutar QueryClientProvider komponente. Konačno, pozvali smo funkciju ReactDOM.render() kako bismo renderovali aplikaciju, pri čemu smo povezali instancu QueryClient-a sa aplikacijom putem QueryClientProvider komponente. Na taj način, obezbedili smo da aplikacija koristi kesiranje podataka pruženo od strane React Query biblioteke, što može poboljšati performanse i efikasnost aplikacije.

# Link ka github repozitorijumu

<https://github.com/elab-development/internet-tehnologije-projekat-documentmanagementsystem_2020_0272>

# Reference

* React. (n.d.). Learn React [Web stranica]. <https://react.dev/learn>
* Laravel. (n.d.). Laravel 10.x Documentation [Web stranica]. https://laravel.com/docs/10.x
* W3C. (s.d.). HTML i CSS specifikacije. https://www.w3.org/standards/webdesign/htmlcss
* MySQL. (s.d.). Zvanična web stranica MySQL baze podataka. https://www.mysql.com/
* <https://medium.com/@ITesic/uvod-u-react-ekosistem-8ccfad0a1030>