Project Management WebsIte

In this tutorial, you are going to learn how to build single page application (SPA) using Laravel 11 and React by building the awesome project management website. In my opinion this is the perfect project if you want to learn building some apps using laravel and react this is going to be the perfect project to put in your portfolio.

On Laravel Side, you are going to learn

Laravel is a powerful and popular open-source PHP framework used for building web applications. Laravel follows the Model-View-Controller (MVC) architectural pattern, which helps to separate the concerns of an application's logic, presentation, and data management.

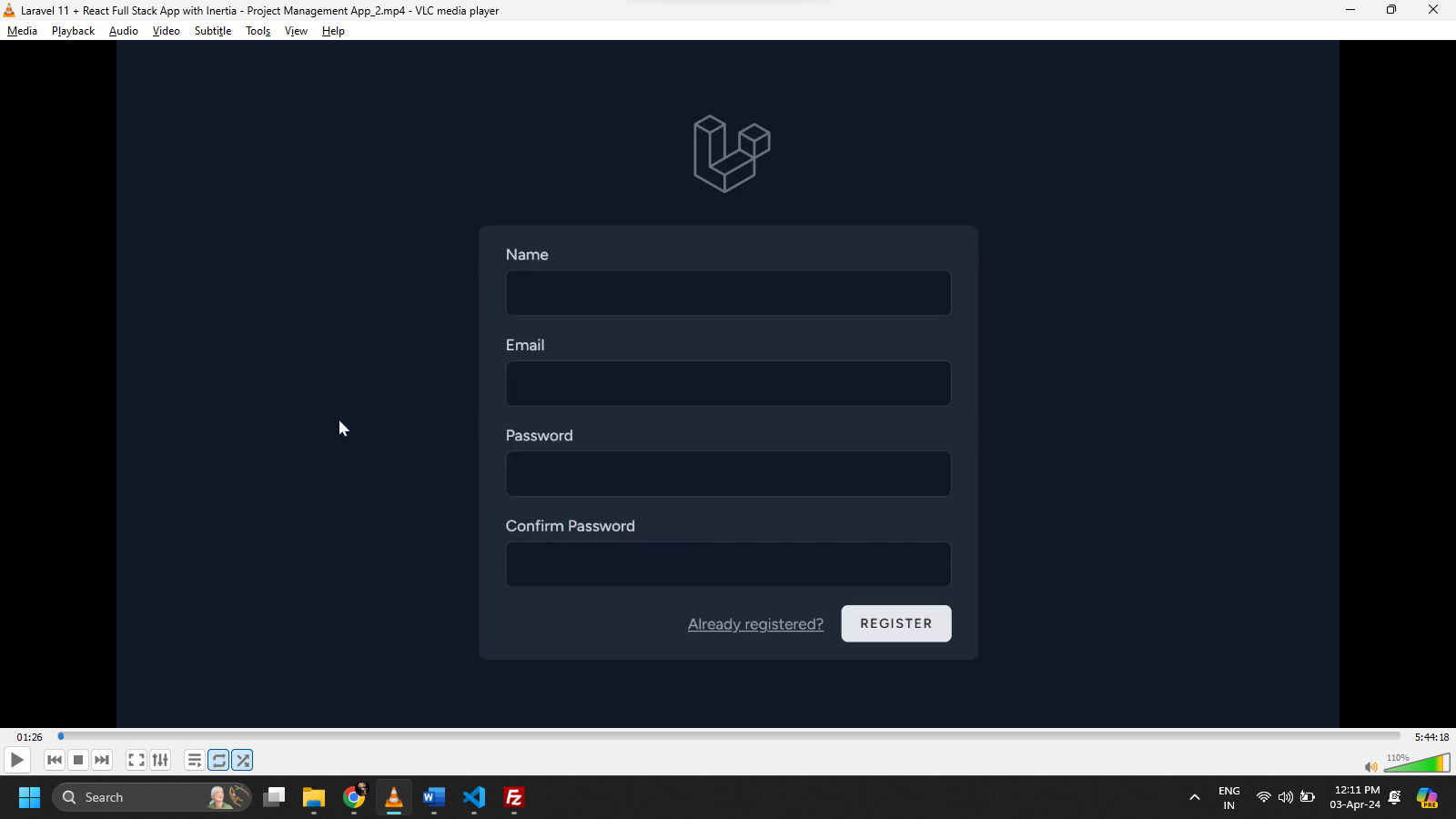
* How to setup the project
* Models
  + In Laravel, a model represents the structure and behavior of data within an application. It typically corresponds to a database table and allows developers to interact with the database using object-oriented methods. Models contain methods for querying, updating, and deleting data, providing an abstraction layer between the application and the database.
* Migrations
  + Migrations in Laravel are version control for database schemas. They allow developers to define and modify the structure of database tables using PHP code, rather than SQL. Migrations provide a convenient way to create and update database tables, columns, indexes, and constraints, ensuring that changes to the database schema can be easily tracked and shared among team members.
* Controllers
  + Controllers are responsible for handling incoming HTTP requests and executing appropriate actions to fulfill those requests. They act as an intermediary between the routes defined in the application and the logic needed to process the requests. Controllers contain methods (or actions) that perform specific tasks, such as retrieving data from the database, manipulating data, or returning responses to the client.
* Resource Controllers
  + Resource controllers in Laravel are a specialized type of controller that provides a convenient way to handle common CRUD (Create, Read, Update, Delete) operations for a resource, such as articles, users, or products. By defining a single resource controller, developers can automatically generate routes for all standard CRUD operations, reducing the need for manual route configuration.
* Requests
  + Requests in Laravel are objects that encapsulate incoming HTTP requests, allowing developers to easily retrieve and validate request data. Laravel provides various types of requests, such as HTTP requests (e.g., GET, POST, PUT, DELETE), form requests, and AJAX requests. Form requests are particularly useful for validating incoming data against predefined rules before processing it in the controller, helping to ensure the security and integrity of the application's data. By separating the validation logic into dedicated request classes, developers can keep their controllers clean and focused on handling business logic.
* Factories
  + Factories in Laravel are used to generate fake data for testing and seeding the database. They provide a convenient way to create model instances with randomized or predefined attributes, making it easier to simulate real-world scenarios and test application functionality. Factories can be used in conjunction with database seeders to populate the database with sample data for testing and development purposes.
* Seeders
  + Seeders in Laravel are classes designed to populate the database with initial data. They are typically used in combination with migrations to set up the initial state of the database, such as creating default users, roles, or other necessary data. Seeders are especially useful for setting up a consistent development environment and for automating the process of populating the database with test data.

On React Side, you are going to learn

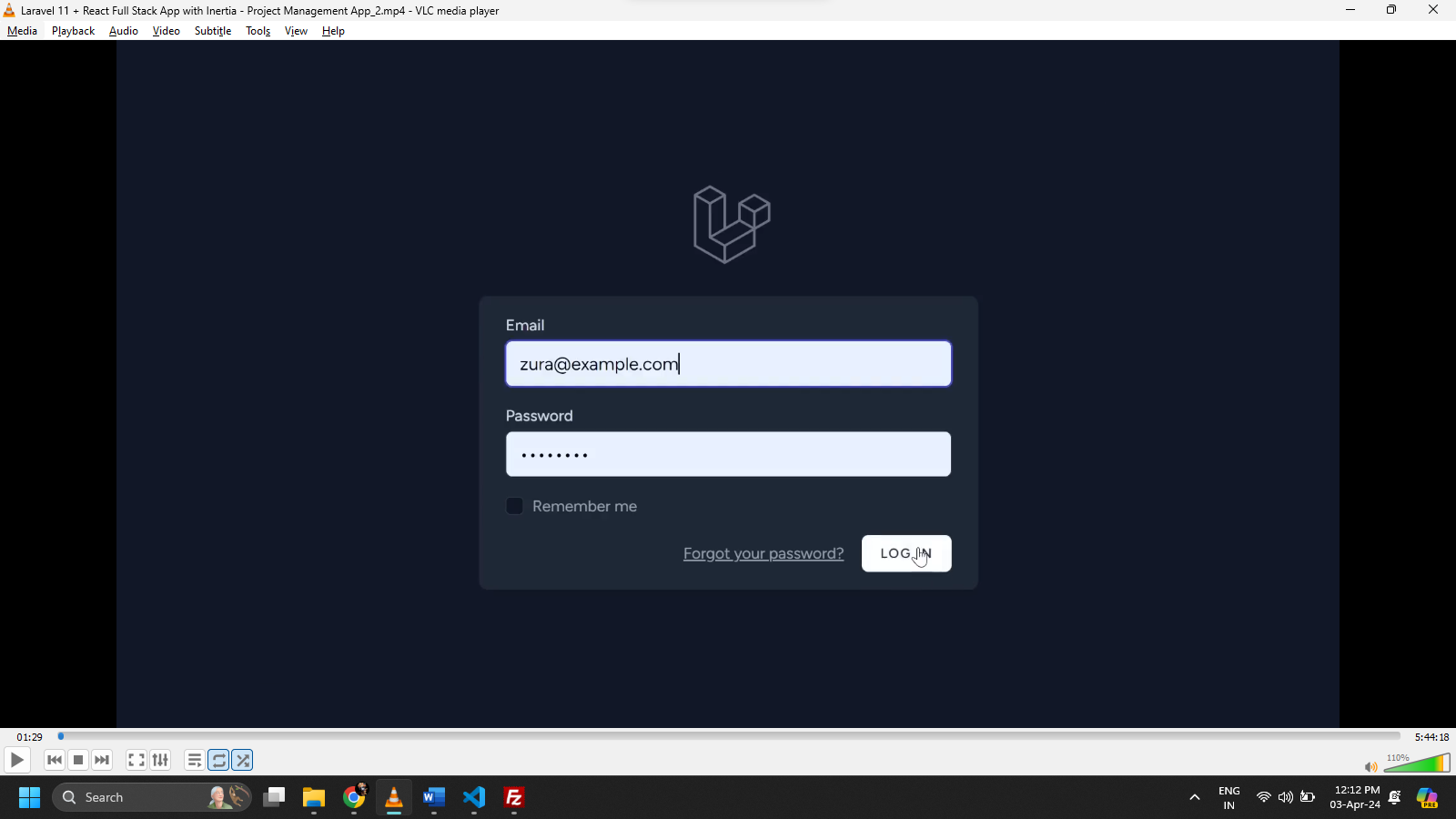
* How to retrieve the data from the Laravel
* Submit the data
* Upload Files
* Implement Pagination
* Implement Sorting
* Implement Filtering

Screenshots

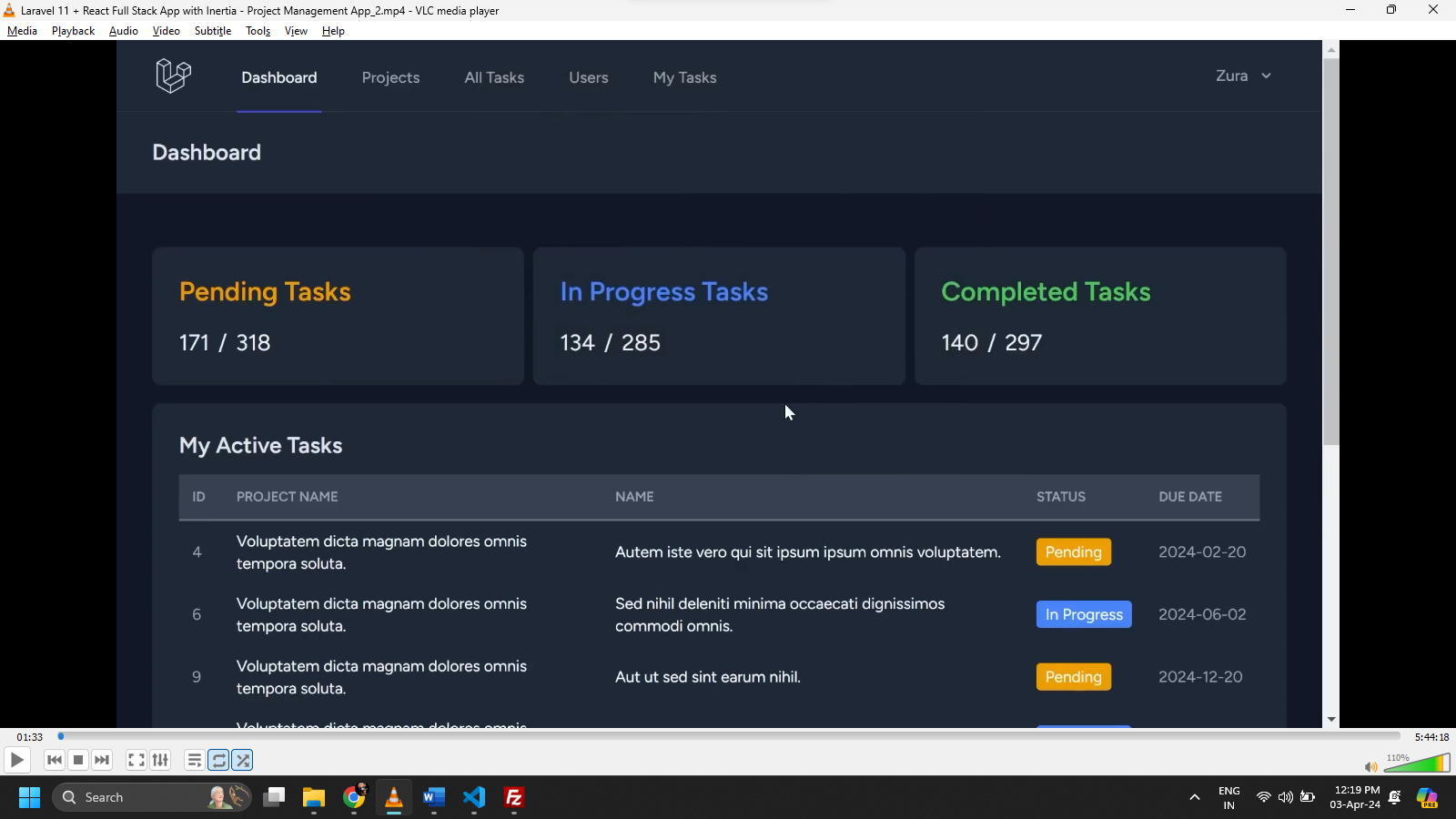
Register Page



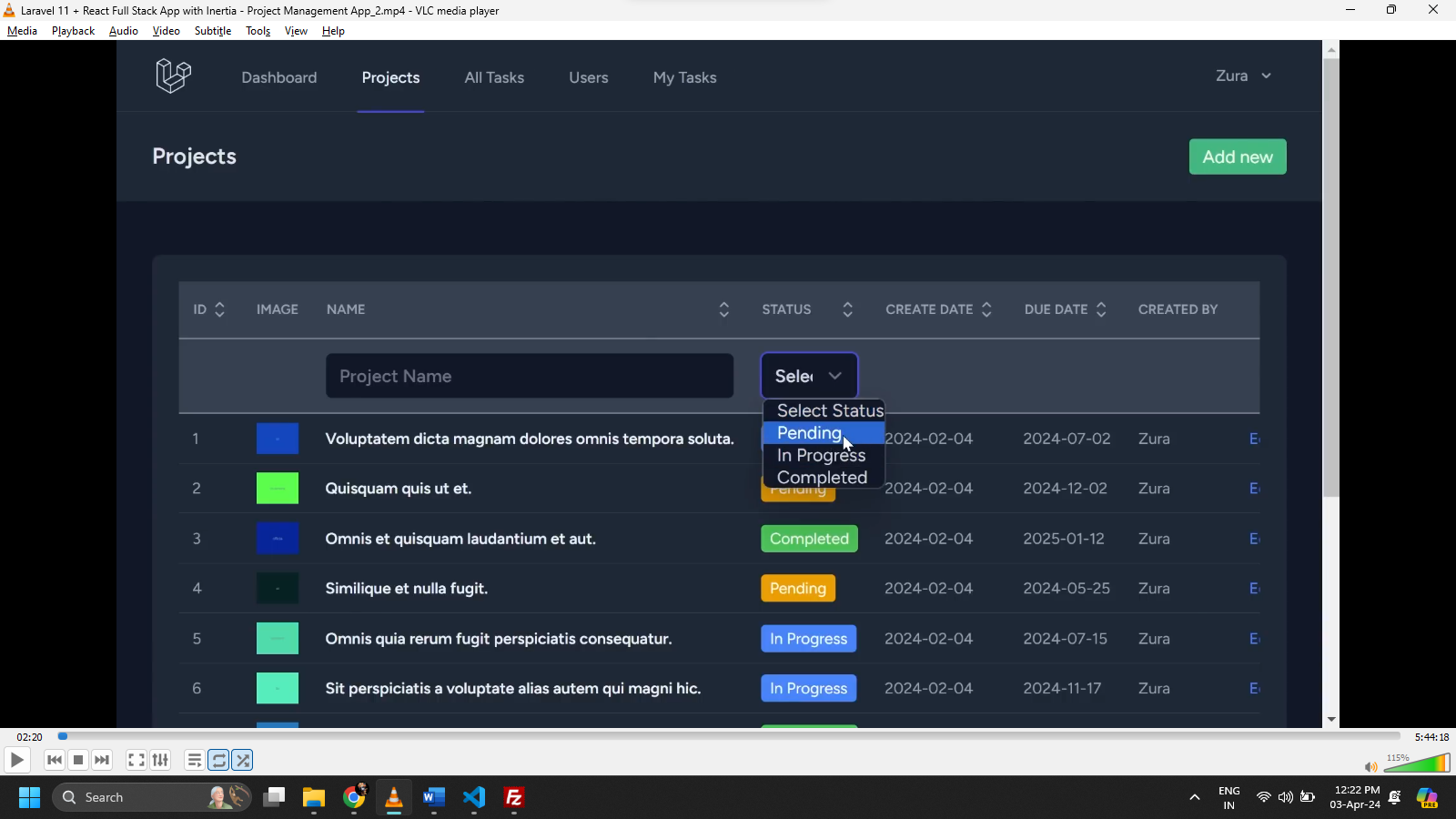
Login Page



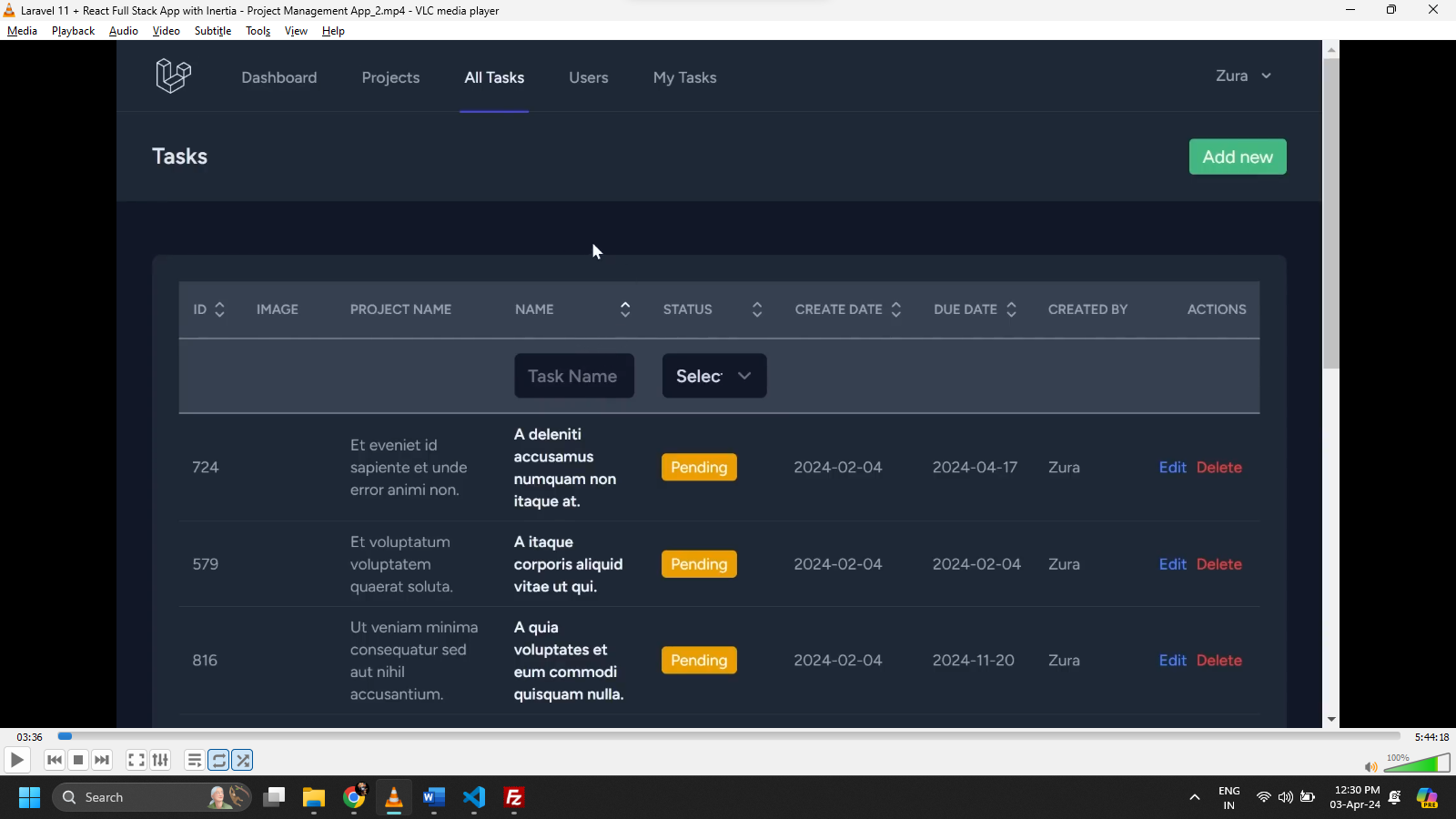
Dashboard Page



Projects Page



All Tasks Page



Working Environment

* XAMPP
* Composer
* Node.js
* Visual Studio Code
* Git
* Laravel Documentation

Laravel Installation: -

After you have installed XAMPP, Composer, Node.js, and Git, you can create a new Laravel project via Composer's create-project command:

composer create-project laravel/laravel example-app

composer require laravel/breeze –dev

php artisan breeze:install

Or, you may create new Laravel projects by globally installing the Laravel installer via Composer:

composer global require laravel/installer

laravel new example-app

composer require laravel/breeze –dev

php artisan breeze:install

*Once the project has been created, start Laravel's local development server using Laravel Artisan's serve command:*

cd example-app

php artisan serve

Breeze with React

Database: sqlite

Enables Email Verification to avoid spammers:

In UserModel **implements** **MustVerifyEmail**

<?php

namespace App\Models;

use Illuminate\Contracts\Auth\MustVerifyEmail;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Foundation\Auth\User as Authenticatable;

use Illuminate\Notifications\Notifiable;

class User extends Authenticatable implements MustVerifyEmail

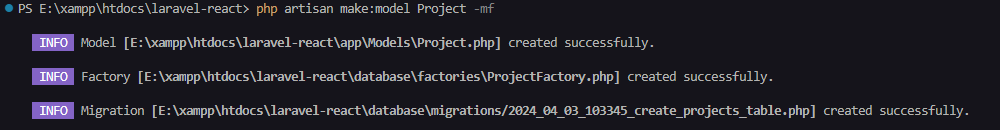
{

    use HasFactory, Notifiable;

}

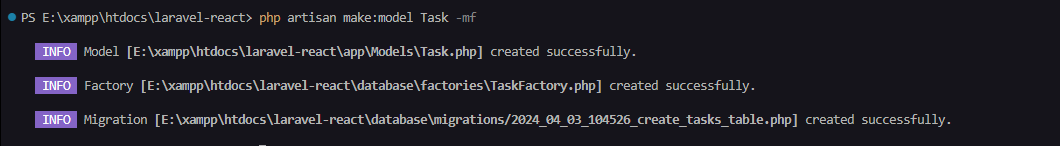
Creating Project Model with factory and migration using command

php artisan make:model Project -mf



Creating Task Model with factory and migration using command

php artisan make:model Task -mf



In Project migration, adding the required columns to define the scheme for projects table

public function up(): void

    {

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

            $table->id();

            $table->string('name');

            $table->longText('description')->nullable();

            $table->timestamp('due\_date')->nullable();

            $table->string('status');

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

            $table->foreignId('created\_by')->constrained('users');

            $table->foreignId('updated\_by')->constrained('users');

            $table->timestamps();

        });

    }

In Task migration, adding the required columns to define the scheme for tasks table

public function up(): void

    {

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

            $table->id();

            $table->string('name');

            $table->longText('description')->nullable();

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

            $table->timestamp('due\_date')->nullable();

            $table->string('status');

            $table->string('priority');

            $table->foreignId('assigned\_user\_id')->constrained('users');

            $table->foreignId('created\_by')->constrained('users');

            $table->foreignId('updated\_by')->constrained('users');

            $table->foreignId('project\_id')->constrained('projects');

            $table->timestamps();

        });

    }

In ProjectFactory, added the columns to generate fake data.

class ProjectFactory extends Factory

{

    public function definition(): array

    {

        return [

            'name' => fake()->sentence(),

            'description' => fake()->realText(),

            'due\_date' => fake()->dateTimeBetween('now', '+1 year'),

            'status' => fake()->randomElement (['pending', 'in\_progress', 'completed']),

            'image\_path' => fake()->imageUrl(),

            'created\_by' => 1,

            'updated\_by' => 1,

            'created\_at' => time(),

            'updated\_at' => time(),

        ];

    }

}

In TaskFactory, added the columns to generate fake data.

class TaskFactory extends Factory

{

    public function definition(): array

    {

        return [

            'name' => fake()->sentence(),

            'description' => fake()->realText(),

            'due\_date' => fake()->dateTimeBetween('now', '+1 year'),

            'status' => fake()->randomElement(['pending', 'in\_progress', 'completed']),

            'priority' => fake()->randomElement(['low','medium','high']),

            'image\_path' => fake()->imageUrl(),

            'assigned\_user\_id' => 1,

            'created\_by' => 1,

            'updated\_by' => 1,

            'created\_at' => time(),

            'updated\_at' => time(),

        ];

    }

}

In Project Model, Define a one-to-many relationship with the Task model by adding tasks()

class Project extends Model

{

    use HasFactory;

    public function tasks() : HasMany

    {

        return $this->hasMany(Task::class);

    }

}

In DatabaseSeeder, updated the UserFactory and added the ProjectFactory to create 30 projects and each project with 30 tasks

class DatabaseSeeder extends Seeder

{

    /\*\*

     \* Seed the application's database.

     \*/

    public function run(): void

    {

        User::factory()->create([

            'name' => 'Madhura',

            'email' => 'madhura@example.com',

            'password' => bcrypt('Password@123'),

            'email\_verified\_at' => time()

        ]);

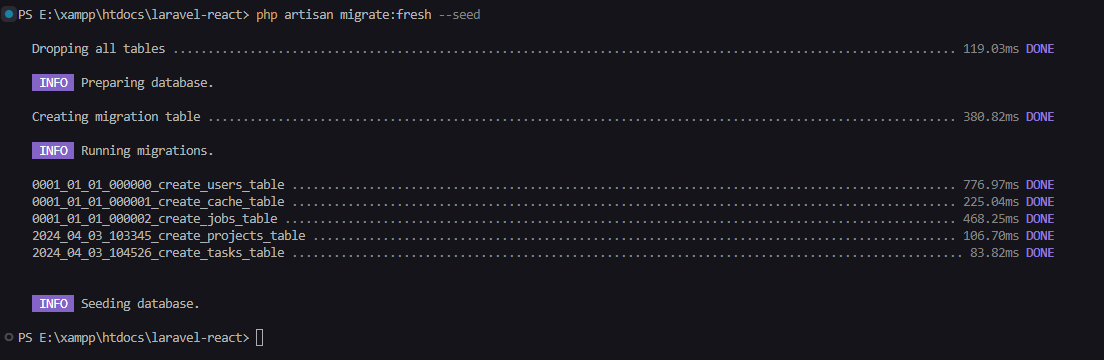
        Project::factory()->count(30)->hasTasks(30)->create();

    }

}

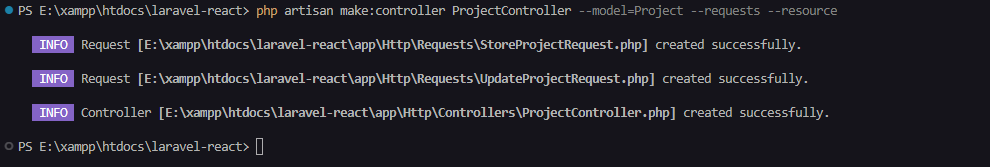
Process of migration(creating database and tables) and seeding(inserting records) using command:-

php artisan migrate:fresh –seed



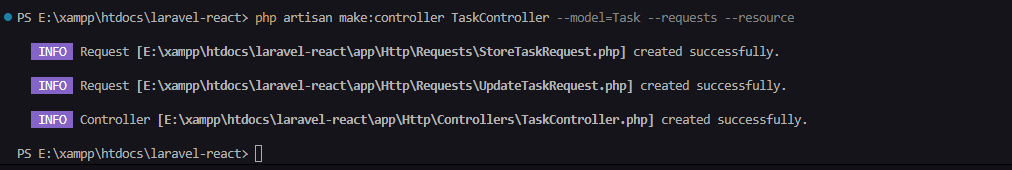
Creating Project Controller using command

php artisan make:controller ProjectController --model=Project --requests --resource



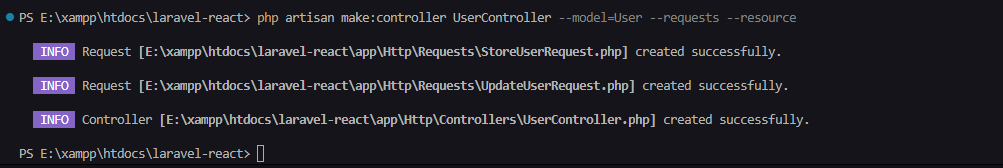
Creating Task Controller using command

php artisan make:controller TaskController --model=Task --requests --resource



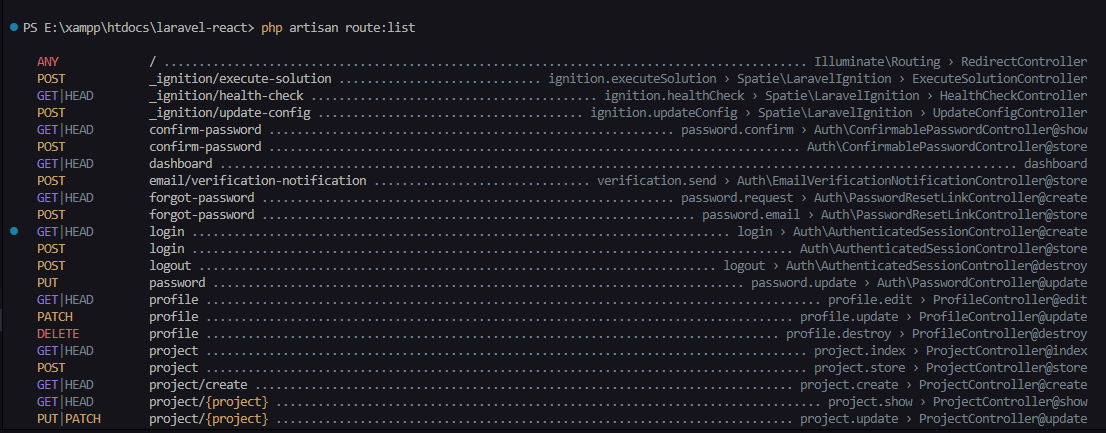
Creating User Controller using command

php artisan make:controller UserController --model=User --requests --resource



To Get the route list by using following command

php artisan route:list



Updating the routes/web.php as like below

<?php

use Inertia\Inertia;

use Illuminate\Support\Facades\Route;

use Illuminate\Foundation\Application;

use App\Http\Controllers\TaskController;

use App\Http\Controllers\UserController;

use App\Http\Controllers\ProfileController;

use App\Http\Controllers\ProjectController;

Route::redirect('/', 'dashboard');

Route::middleware(['auth', 'verified'])->group(function () {

    Route::get('/dashboard', fn () => Inertia::render('Dashboard'))->name('dashboard');

    //Controllers

    Route::resource('project', ProjectController::class);

    Route::resource('task', TaskController::class);

    Route::resource('user', UserController::class);

});

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

    Route::get('/profile', [ProfileController::class, 'edit'])->name('profile.edit');

    Route::patch('/profile', [ProfileController::class, 'update'])->name('profile.update');

    Route::delete('/profile', [ProfileController::class, 'destroy'])->name('profile.destroy');

});

require \_\_DIR\_\_.'/auth.php';

Adding the navigation link in resources/js/Layouts/AuthenticatedLayout.jsx

<div className="hidden space-x-8 sm:-my-px sm:ms-10 sm:flex">

    <NavLink href={route('dashboard')} active={route().current('dashboard')}>

        Dashboard

    </NavLink>

    <NavLink href={route('project.index')} active={route().current('project.index')}>

        Projects

    </NavLink>

    <NavLink href={route('task.index')} active={route().current('task.index')}>

        All Tasks

    </NavLink>

    <NavLink href={route('user.index')} active={route().current('user.index')}>

        Users

    </NavLink>

</div>

Create an Index.jsx file within the js\Pages\Project directory to serve as a view page for the Project controller.

import AuthenticatedLayout from "@/Layouts/AuthenticatedLayout";

import { Head } from "@inertiajs/react";

export default function Index*({ auth, projects })* {

    return (

        <AuthenticatedLayout

            user={auth.user}

            header={<h2 className="font-semibold text-xl text-gray-800 dark:text-gray-200 leading-tight">Projects</h2>}

        >

            <Head title="Projects" />

            <div className="py-12">

                <div className="max-w-7xl mx-auto sm:px-6 lg:px-8">

                    <div className="bg-white dark:bg-gray-800 overflow-hidden shadow-sm sm:rounded-lg">

                        <div className="p-6 text-gray-900 dark:text-gray-100">Projects</div>

                    </div>

                </div>

            </div>

        </AuthenticatedLayout>

    )

}

The index() method of the ProjectController has been modified to render the "Project/Index" view utilizing the Inertia.js framework.

class ProjectController extends Controller

{

    public function index()

    {

        return inertia("Project/Index", []);

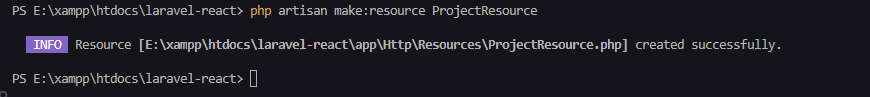
    }

}

Creating ProjectResource using command

php artisan make:resource ProjectResource

By creating a resource class, you can customize the data that gets returned when accessing your model instances through API endpoints. This allows you to shape the data and include only the necessary fields or additional information required by the client.



Updating **ProjectResource** with the necessary fields required by the client

class ProjectResource extends JsonResource

{

    public function toArray(*Request $request*): array

    {

        return [

            'id' => $this->id,

            'name' => $this->name,

            'description' => $this->description,

            'created\_at' => (new Carbon($this->created\_at))->format('Y-m-d'),

            'due\_date' => (new Carbon($this->de\_at))->format('Y-m-d'),

            'status' => $this->status,

            'image\_path' => $this->image\_path,

            'createdBy' => $this->createdBy,

            'updatedBy' => $this->updatedBy,

        ];

    }

}

In Project Model, Define a belongs to relationship with the User model by adding createdBy() and updatedBy()

class Project extends Model

{

    use HasFactory;

    public function createdBy(): BelongsTo

{

        return $this->belongsTo(User::class, 'created\_by');

    }

    public function updatedBy(): BelongsTo

    {

        return $this->belongsTo(User::class, 'updated\_by');

    }

}

The index() method of the ProjectController has been updated to render the "Project/Index" view with a collection of project data. Here's the modified method:

class ProjectController extends Controller

{

    public function index()

    {

        $query = Project::query();

        // Retrieve projects data and paginate the results

        $projects = $query->paginate(10);

        // Return the "Project/Index" view along with the projects data collection

        return inertia("Project/Index", [

            "projects" => ProjectResource::collection($projects),

        ]);

    }

}

Update an Index.jsx file within the js\Pages\Project directory to serve as a view page for the Project controller.

import AuthenticatedLayout from "@/Layouts/AuthenticatedLayout";

import { Head } from "@inertiajs/react";

export default function Index*({ auth, projects })* {

    return (

        <AuthenticatedLayout

            user={auth.user}

            header={<h2 className="font-semibold text-xl text-gray-800 dark:text-gray-200 leading-tight">Projects</h2>}

        >

            <Head title="Projects" />

            <div className="py-12">

                <div className="max-w-7xl mx-auto sm:px-6 lg:px-8">

                    <div className="bg-white dark:bg-gray-800 overflow-hidden shadow-sm sm:rounded-lg">

                        <div className="p-6 text-gray-900 dark:text-gray-100">

                            <pre>{JSON.stringify(projects, undefined, 2)}</pre>

                        </div>

                    </div>

                </div>

            </div>

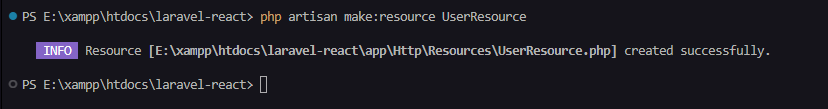
        </AuthenticatedLayout>

    )

}

Creating UserResource using command

php artisan make:resource UserResource



Updating and Updating UserResource with the necessary fields required by the client

class UserResource extends JsonResource

{

    public function toArray(*Request $request*): array

    {

        return [

            'id' => $this->id,

            'name' => $this->name,

            'email' => $this->email,

            'created\_at' => (new Carbon($this->created\_at))->format('Y-m-d'),

        ];

    }

}

Updating the **ProjectResource** to include **UserResource** in order to pass the required fields from the User model

class ProjectResource extends JsonResource

{

    public function toArray(*Request $request*): array

    {

        return [

            'id' => $this->id,

            'name' => $this->name,

            'description' => $this->description,

            'created\_at' => (new Carbon($this->created\_at))->format('Y-m-d'),

            'due\_date' => (new Carbon($this->due\_date))->format('Y-m-d'),

            'status' => $this->status,

            'image\_path' => $this->image\_path,

            'createdBy' => new UserResource($this->createdBy),

            'updatedBy' => new UserResource($this->updatedBy),

        ];

    }

}

Update an Index.jsx file within the js\Pages\Project directory to display a data in table format

<div className="p-6 text-gray-900 dark:text-gray-100">

<pre>{JSON.stringify(projects, undefined, 2)}</pre>

</div>

To

<table className="w-full text-sm text-left rtl:text-right text-gray-500 dark:text-gray-400">

    <thead className="text-xs text-gray-700 uppercase bg-gray-50 dark:bg-gray-700 dark:text-gray-400 border-b-2 border-gray-500">

        <tr className="text-nowrap">

            <th className="px-3 py-3">ID</th>

            <th className="px-3 py-3">Image</th>

            <th className="px-3 py-3">Name</th>

            <th className="px-3 py-3">Status</th>

            <th className="px-3 py-3">Created Date</th>

            <th className="px-3 py-3">Due Date</th>

            <th className="px-3 py-3">Created By</th>

            <th className="px-3 py-3 text-right">Actions</th>

        </tr>

    </thead>

    <tbody>

        {projects.data.map(project => (

            <tr className="bg-white border-b dark:bg-gray-800 dark:border-gray-700" key={project.id}>

                <th className="px-3 py-2">{project.id}</th>

                <td className="px-3 py-2">

<img src={project.image\_path} style={{ width: 60 }} alt="image" />

</td>

                <td className="px-3 py-2">{project.name}</td>

                <td className="px-3 py-2">{project.status}</td>

                <td className="px-3 py-2 text-nowrap">{project.created\_at}</td>

                <td className="px-3 py-2 text-nowrap">{project.due\_date}</td>

                <td className="px-3 py-2">{project.createdBy.name}</td>

                <td className="px-3 py-2">

                    <Link href={route('project.edit', project.id)}

                        className="font-medium text-blue-600 dark:text-blue-500 hover:underline mx-1">

                        Edit

                    </Link>

                    <Link href={route('project.destroy', project.id)}

                        className="font-medium text-red-600 dark:text-red-500 hover:underline mx-1">

                        Delete

                    </Link>

                </td>

            </tr>

        ))}

    </tbody>

</table>

Creating Pagination.jsx file within the js\Components directory.

import { Link } from "@inertiajs/react";

export default function Pagination*({ links })* {

    return (

        <nav className="text-center mt-4">

            {links.map(link => (

                <Link

                    preserveScroll

                    href={link.url || ""}

                    key={link.label}

                    className={

                        "inline-block py-2 px-3 rounded-lg text-gray-200 text-xs " +

                        (link.active ? "bg-gray-950 " : " ") +

                        (!link.url

                            ? "!text-gray-500 cursor-not-allowed "

                            : "hover:bg-gray-950")

                    }

                    dangerouslySetInnerHTML={{ \_\_html: link.label }}></Link>

            ))}

        </nav>

    )

}

Now include the pagination component in Project/Index.jsx below the table tag

<div className="p-6 text-gray-900 dark:text-gray-100">

<table></table>

     <Pagination links={projects.meta.links}/>

</div>

And import Pagination Component Like this

import Pagination from "@/Components/Pagination";

Creating constants.jsx file within the js directory to map the class and text for the options status and priority of projects and tasks

export const PROJECT\_STATUS\_CLASS\_MAP = {

    'pending': 'bg-amber-500',

    'in\_progress': 'bg-blue-500',

    'completed': 'bg-green-500',

}

export const PROJECT\_STATUS\_TEXT\_MAP = {

    'pending': 'Pending',

    'in\_progress': 'In Progress',

    'completed': 'Completed',

}

export const TASK\_STATUS\_CLASS\_MAP = {

    'pending': 'bg-amber-500',

    'in\_progress': 'bg-blue-500',

    'completed': 'bg-green-500',

}

export const TASK\_STATUS\_TEXT\_MAP = {

    'pending': 'Pending',

    'in\_progress': 'In Progress',

    'completed': 'Completed',

}

export const TASK\_PRIORITY\_CLASS\_MAP = {

    'low': 'bg-gray-500',

    'medium': 'bg-amber-500',

    'high': 'bg-red-500',

}

export const TASK\_PRIORITY\_TEXT\_MAP = {

    'low': 'Low',

    'medium': 'Medium',

    'high': 'High',

}

Update an Index.jsx file within the js\Pages\Project directory to add PROJECT\_STATUS\_TEXT\_MAP constant

<td className="px-3 py-2">

{PROJECT\_STATUS\_TEXT\_MAP[project.status]}

</td>

Update an Index.jsx file within the js\Pages\Project directory to add PROJECT\_STATUS\_CLASS\_MAP constant

<td className="px-3 py-2">

<span className={"px-2 py-1 rounded text-white "+ PROJECT\_STATUS\_CLASS\_MAP[project.status]}>

{PROJECT\_STATUS\_TEXT\_MAP[project.status]}

</span>

</td>

Implement Searching Option

SelectInput Component(SelectInput.jsx) Created by copying and modifying TextInput Component of Breeze

import { forwardRef, useRef } from 'react';

export default forwardRef(function TextInput*({ className = '', children, ...props }, ref)* {

    const input = ref ? ref : useRef();

    return (

        <select

            {...props}

            className={

                'border-gray-300 dark:border-gray-700 dark:bg-gray-900 dark:text-gray-300 focus:border-indigo-500 dark:focus:border-indigo-600 focus:ring-indigo-500 dark:focus:ring-indigo-600 rounded-md shadow-sm ' +

                className

            }

            ref={input}

        >

            {children}

        </select>

    );

});

Modifying the index() method of ProjectController to pass the queryParams and change the query according to request

public function index()

    {

        $query = Project::query();

        if (request("name")) {

            $query->where("name", "like", "%" . request("name") . "%");

        }

        if (request("status")) {

            $query->where("status", request("status"));

        }

        $projects = $query->paginate(10);

        return inertia("Project/Index", [

            "projects" => ProjectResource::collection($projects),

            "queryParams" => request()->query() ?: null,

        ]);

    }

Now paste the below code between </thead> <tbody> tags of Project/Index.jsx

<thead className="text-xs text-gray-700 uppercase bg-gray-50 dark:bg-gray-700 dark:text-gray-400 border-b-2 border-gray-500">

    <tr className="text-nowrap">

        <th className="px-3 py-3"></th>

        <th className="px-3 py-3"></th>

        <th className="px-3 py-3">

            <TextInput

                defaultValue={queryParams.name}

                className="w-full"

                placeholder="Project Name"

                onBlur={(e) => searchFeildChanged('name', e.target.value)}

                onKeyPress={(e) => onKeyPress('name', e)}

            />

        </th>

        <th className="px-3 py-3">

            <SelectInput

                defaultValue={queryParams.status}

                className="w-full"

                onChange={(e) => searchFeildChanged('status', e.target.value)}

            >

                <option value="">Select Status</option>

                <option value="pending">Pending</option>

                <option value="in\_progress">In Progress</option>

                <option value="completed">Completed</option>

           </SelectInput>

        </th>

        <th className="px-3 py-3"></th>

        <th className="px-3 py-3"></th>

        <th className="px-3 py-3"></th>

        <th className="px-3 py-3"></th>

    </tr>

</thead>

And import and update these lines before return statement

import SelectInput from "@/Components/SelectInput";

import TextInput from "@/Components/TextInput";

import { Head, Link, router } from "@inertiajs/react";

export default function Index*({ auth, projects, queryParams = null })* {

    queryParams = queryParams || {}

    const searchFeildChanged = *(name, value)* => {

        if (value) {

            queryParams[name] = value

        }

        else {

            delete queryParams[name]

        }

        router.get(route('project.index'), queryParams);

    };

    const onKeyPress = *(name, e)* => {

        if (e.key !== 'Enter') return;

        searchFeildChanged(name, e.target.value);

    };

Implement Sorting Option

Modifying the index() method of ProjectController to pass the queryParams and change the query according to request

public function index()

    {

        $query = Project::query();

        $sortField = request("sort\_feild", "created\_at");

        $sortDirection = request('sort\_direction', "desc");

        if (request("name")) {

            $query->where("name", "like", "%" . request("name") . "%");

        }

        if (request("status")) {

            $query->where("status", request("status"));

        }

        // Retrieve projects data and paginate the results

        $projects = $query->orderBy($sortField, $sortDirection)->paginate(10);

        // Return the "Project/Index" view along with the projects data collection

        return inertia("Project/Index", [

            "projects" => ProjectResource::collection($projects),

            "queryParams" => request()->query() ?: null,

        ]);

    }

Now update code of table columns headers which need to sort of Project/Index.jsx

<thead className="text-xs text-gray-700 uppercase bg-gray-50 dark:bg-gray-700 dark:text-gray-400 border-b-2 border-gray-500">

<tr className="text-nowrap">

        <th onClick={*(e)* => sortChanged('id')} className="px-3 py-3">ID</th>

             <th className="px-3 py-3">Image</th>

             <th onClick={*(e)* => sortChanged('name')} className="px-3 py-3">Name</th>

              <th onClick={*(e)* => sortChanged('status')} className="px-3 py-3">Status</th>

              <th onClick={*(e)* => sortChanged('created\_at')} className="px-3 py-3">Created Date</th>

              <th onClick={*(e)* => sortChanged('due\_date')} className="px-3 py-3">Due Date</th>

             <th className="px-3 py-3">Created By</th>

             <th className="px-3 py-3 text-right">Actions</th>

    </tr>

</thead>

Write the function definition for sortChanged()

const sortChanged = *(name)* => {

        if (name === queryParams.sort\_feild) {

            if (queryParams.sort\_direction === 'asc') {

                queryParams.sort\_direction = 'desc';

            } else {

                queryParams.sort\_direction = 'asc';

            }

        } else {

            queryParams.sort\_feild = name;

            queryParams.sort\_direction = 'asc';

        }

        router.get(route('project.index'), queryParams);

};

Install Heroicons for React using

npm install @heroicons/react -S

Now update code of table columns headers of Project/Index.jsx for including heroicons

import { ChevronUpIcon, ChevronDownIcon } from "@heroicons/react/16/solid";

Step 1:

<th onClick={*(e)* => sortChanged('id')} className="px-3 py-3">

       ID

       <ChevronUpIcon className="w-4" />

       <ChevronDownIcon className="w-4" />

</th>

Step 2:

<th onClick={*(e)* => sortChanged('id')}

className="px-3 py-3 flex items-center justify-between">

       ID

       <ChevronUpIcon className="w-4" />

       <ChevronDownIcon className="w-4" />

</th>

Step 3:

<th onClick={*(e)* => sortChanged('id')}

className="px-3 py-3 flex items-center justify-between">

       ID

<div>

      <ChevronUpIcon className="w-4" />

        <ChevronDownIcon className="w-4" />

</div>

</th>

Step 4:

<th onClick={*(e)* => sortChanged('id')}

className="px-3 py-3 flex items-center justify-between">

       ID

<div>

      <ChevronUpIcon className="w-4" />

        <ChevronDownIcon className="w-4 -mt-2" />

</div>

</th>

Step 5:

<th onClick={*(e)* => sortChanged('id')}

className="px-3 py-3 flex items-center justify-between gap-1">

       ID

<div>

      <ChevronUpIcon className="w-4" />

        <ChevronDownIcon className="w-4 -mt-2" />

</div>

</th>

Step 6:

<th onClick={*(e)* => sortChanged('id')}>

<div className="px-3 py-3 flex items-center justify-between gap-1 cursor-pointer">

        ID

<div>

      <ChevronUpIcon className="w-4" />

        <ChevronDownIcon className="w-4 -mt-2" />

</div>

</div>

</th>

Final Version:

<th onClick={*(e)* => sortChanged('id')}>

<div className="px-3 py-3 flex items-center justify-between gap-1">

        ID

        <div>

            <ChevronUpIcon className={

                 "w-4 " +

                (queryParams.sort\_feild === 'id' && queryParams.sort\_direction === 'asc' ? 'text-white' : '')

              }

              />

            <ChevronDownIcon className={

                "w-4 -mt-2 " +

                  (queryParams.sort\_feild === 'id' && queryParams.sort\_direction === 'desc' ? 'text-white' : '')

            }

            />

          </div>

       </div>

</th>

Now we have to create the TableHeading.jsx Components

import React from 'react';

import { ChevronUpIcon, ChevronDownIcon } from "@heroicons/react/16/solid";

const TableHeading = ({

    name,

    sortable = true,

    sort\_feild = null,

    sort\_direction = null,

    sortChanged = *()* => {},

    children,

}) => {

    return (

        <th onClick={*(e)* => sortChanged(name)}>

            <div className="px-3 py-3 flex items-center justify-between gap-1 cursor-pointer">

                {children}

                {sortable && (

                    <div>

                        <ChevronUpIcon className={

                            "w-4 " +

                            (sort\_feild === name && sort\_direction === 'asc' ? 'text-white' : '')

                        }

                        />

                        <ChevronDownIcon className={

                            "w-4 -mt-2 " +

                            (sort\_feild === name && sort\_direction === 'desc' ? 'text-white' : '')

                        }

                        />

                    </div>

                )}

            </div>

        </th>

    )

}

export default TableHeading

Project/Index.jsx from this

<th onClick={*(e)* => sortChanged('id')}>

<div className="px-3 py-3 flex items-center justify-between gap-1">

        ID

        <div>

            <ChevronUpIcon className={

                 "w-4 " +

                (queryParams.sort\_feild === 'id' && queryParams.sort\_direction === 'asc' ? 'text-white' : '')

              }

              />

            <ChevronDownIcon className={

                "w-4 -mt-2 " +

                  (queryParams.sort\_feild === 'id' && queryParams.sort\_direction === 'desc' ? 'text-white' : '')

            }

            />

          </div>

       </div>

</th>

to

<TableHeading

       name="id"

       sort\_feild={queryParams.sort\_feild}

       sort\_direction={queryParams.sort\_direction}

       sortChanged={sortChanged}

>

ID

</TableHeading>

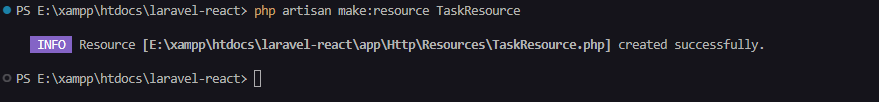
Now we have to do all the things again to display the tasks

Step 1:

Copy and paste Project\Index.jsx to Task\Index.jsx and the find & replace the contents of projects to tasks.

Step 2:

Create TaskResource and update the content



class TaskResource extends JsonResource

{

    public function toArray(*Request $request*): array

    {

        return [

            'id' => $this->id,

            'name' => $this->name,

            'description' => $this->description,

            'created\_at' => (new Carbon($this->created\_at))->format('Y-m-d'),

            'due\_date' => (new Carbon($this->due\_date))->format('Y-m-d'),

            'status' => $this->status,

            'priority' => $this->priority,

            'image\_path' => $this->image\_path,

            'project' => new ProjectResource($this->project),

            'assignedUser' => $this->assignedUser ? new UserResource($this->assignedUser) : null,

            'createdBy' => new UserResource($this->createdBy),

            'updatedBy' => new UserResource($this->updatedBy),

        ];

    }

}

Copy and paste index() method from ProjectController to index() of TaskController and then find and replace projects to tasks

class TaskController extends Controller

{

public function index()

    {

        $query = Task::query();

        $sortField = request("sort\_feild", "created\_at");

        $sortDirection = request('sort\_direction', "desc");

        if (request("name")) {

            $query->where("name", "like", "%" . request("name") . "%");

        }

        if (request("status")) {

            $query->where("status", request("status"));

        }

        $tasks = $query->orderBy($sortField, $sortDirection)->paginate(10);

        return inertia("Task/Index", [

            "tasks" => TaskResource::collection($tasks),

            "queryParams" => request()->query() ?: null,

        ]);

    }

}

Update the Task Model to define the relationships:

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Relations\BelongsTo;

use Illuminate\Database\Eloquent\Factories\HasFactory;

class Task extends Model

{

    use HasFactory;

    public function project(): BelongsTo

    {

        return $this->belongsTo(Project::class);

    }

    public function assignedUser(): BelongsTo

    {

        return $this->belongsTo(User::class, 'assigned\_user\_id');

    }

    public function createdBy(): BelongsTo

    {

        return $this->belongsTo(User::class, 'created\_by');

    }

    public function updatedBy(): BelongsTo

    {

        return $this->belongsTo(User::class, 'updated\_by');

    }

}

Update an Index.jsx file within the js\Pages\Project directory added link tag for name to go to the route project.show from

<td className="px-3 py-2">{project.name}</td>

to

<td className="px-3 py-2 hover:underline hover:text-white text-nowrap">

<Link href={route('project.show', project.id)}>

{project.name}

</Link>

</td>

Update the show() method from ProjectController

public function show(*Project $project*)

{

       return inertia("Project/Show", [

            "project" => new ProjectResource($project)

       ]);

}

Now Create Show.jsx file inside resource/js/Pages/Project (using rafce)

import React from 'react'

const Show = *()* => {

  return (

    <div>Show</div>

  )

}

export default Show

now update the file as like below

import AuthenticatedLayout from '@/Layouts/AuthenticatedLayout'

import { Head } from '@inertiajs/react'

import React from 'react'

const Show = *({ auth, project })* => {

    return (

        <AuthenticatedLayout

            user={auth.user}

            header= {

                <h2 className="font-semibold text-xl text-gray-800 dark:text-gray-200 leading-tight">

                    {`Project "${project.name}"`}

                </h2>}

        >

            <Head title={`Project "${project.name}"`} />

            <div className="py-12">

                <div className="max-w-7xl mx-auto sm:px-6 lg:px-8">

                    <div className="bg-white dark:bg-gray-800 overflow-hidden shadow-sm sm:rounded-lg">

                        <div className="p-6 text-gray-900 dark:text-gray-100">

                           <pre>{JSON.stringify(project)}</pre>

                        </div>

                    </div>

                </div>

            </div>

        </AuthenticatedLayout>

    )

}

export default Show

ProjectResource Updated to remove default outer wrapping(data)

class ProjectResource extends JsonResource

{

public static $wrap = false;

…

…

}

ProjectController Show() method updated to display the project details using Project/Show.jsx

public function show(*Project $project*)

    {

        return inertia("Project/Show", [

            "project" => new ProjectResource($project)

        ]);

    }

Updating Project/Show.jsx to display the selected project details

<div className="py-12">

                <div className="max-w-7xl mx-auto sm:px-6 lg:px-8">

                    <div className="bg-white dark:bg-gray-800 overflow-hidden shadow-sm sm:rounded-lg">

                        <div className="p-6 text-gray-900 dark:text-gray-100">

                           <pre>{JSON.stringify(project)}</pre>

                        </div>

                    </div>

                </div>

            </div>

Instead of above code we are replacing with new code

<div className="py-12">

    <div className="max-w-7xl mx-auto sm:px-6 lg:px-8">

        <div className="bg-white dark:bg-gray-800 overflow-hidden shadow-sm sm:rounded-lg">

            <div>

                <img src={project.image\_path} alt={project.name} className='w-full h-64 object-cover' />

            </div>

            <div className="p-6 text-gray-900 dark:text-gray-100">

                <div className='grid gap-1 grid-cols-2 mt-2'>

                    <div>

                        <div>

                            <label className='font-bold text-lg'>Project ID</label>

                            <p className='mt-1'>{project.id}</p>

                        </div>

                        <div className='mt-4'>

                            <label className='font-bold text-lg'>Project Name</label>

                            <p className='mt-1'>{project.name}</p>

                        </div>

                        <div className='mt-4'>

                            <label className='font-bold text-lg'>Project Status</label>

                            <p className='mt-1'>

                                <span className={"px-2 py-1 rounded text-white " + PROJECT\_STATUS\_CLASS\_MAP[project.status]}>

                                    {PROJECT\_STATUS\_TEXT\_MAP[project.status]}

                                </span>

                            </p>

                        </div>

                        <div className='mt-4'>

                            <label className='font-bold text-lg'>Created By</label>

                            <p className='mt-1'>{project.createdBy.name}</p>

                        </div>

                    </div>

                    <div>

                        <div>

                            <label className='font-bold text-lg'>Created Date</label>

                            <p className='mt-1'>{project.created\_at}</p>

                        </div>

                        <div className='mt-4'>

                            <label className='font-bold text-lg'>Due Date</label>

                            <p className='mt-1'>{project.due\_date}</p>

                        </div>

                        <div className='mt-4'>

                            <label className='font-bold text-lg'>Updated By</label>

                            <p className='mt-1'>{project.updatedBy.name}</p>

                        </div>

                    </div>

                </div>

                <div className='mt-4'>

                    <label className='font-bold text-lg'>Project Description</label>

                    <p className='mt-1'>{project.description}</p>

                </div>

            </div>

        </div>

    </div>

</div>

Now below this we going insert task list in table format belongs to this project

<div className="py-12">

    <div className="max-w-7xl mx-auto sm:px-6 lg:px-8">

        <div className="bg-white dark:bg-gray-800 overflow-hidden shadow-sm sm:rounded-lg">

            <div className="p-6 text-gray-900 dark:text-gray-100">

                TaskList Table goes here

            </div>

        </div>

    </div>

</div>

Now we have create Tasks Table as reusable so we need to create TasksTable.jsx file inside Task Directory

Using (rafce) and move the table information from Task/Index.jsx to TasksTable.jsx like this

import React from 'react';

import TableHeading from "@/Components/TableHeading";

import { TASK\_STATUS\_CLASS\_MAP, TASK\_STATUS\_TEXT\_MAP } from "@/constants";

import Pagination from "@/Components/Pagination";

import SelectInput from "@/Components/SelectInput";

import TextInput from "@/Components/TextInput";

import { Link, router } from "@inertiajs/react";

const TasksTable = *({ tasks, queryParams = null })* => {

    queryParams = queryParams || {}

    const searchFeildChanged = *(name, value)* => {

        if (value) {

            queryParams[name] = value

        }

        else {

            delete queryParams[name]

        }

        router.get(route('task.index'), queryParams);

    };

    const onKeyPress = *(name, e)* => {

        if (e.key !== 'Enter') return;

        searchFeildChanged(name, e.target.value);

    };

    const sortChanged = *(name)* => {

        if (name === queryParams.sort\_feild) {

            if (queryParams.sort\_direction === 'asc') {

                queryParams.sort\_direction = 'desc';

            } else {

                queryParams.sort\_direction = 'asc';

            }

        } else {

            queryParams.sort\_feild = name;

            queryParams.sort\_direction = 'asc';

        }

        router.get(route('task.index'), queryParams);

    };

    return (

        <>

            <div className="overflow-auto">

                <table className="w-full text-sm text-left rtl:text-right text-gray-500 dark:text-gray-400">

                    <thead className="text-xs text-gray-700 uppercase bg-gray-50 dark:bg-gray-700 dark:text-gray-400 border-b-2 border-gray-500">

                        <tr className="text-nowrap">

                            <TableHeading

                                name="id"

                                sort\_feild={queryParams.sort\_feild}

                                sort\_direction={queryParams.sort\_direction}

                                sortChanged={sortChanged}

                            >

                                ID

                            </TableHeading>

                            <th className="px-3 py-3">Image</th>

                            <TableHeading

                                name="name"

                                sort\_feild={queryParams.sort\_feild}

                                sort\_direction={queryParams.sort\_direction}

                                sortChanged={sortChanged}

                            >

                                Name

                            </TableHeading>

                            <TableHeading

                                name="status"

                                sort\_feild={queryParams.sort\_feild}

                                sort\_direction={queryParams.sort\_direction}

                                sortChanged={sortChanged}

                            >

                                Status

                            </TableHeading>

                            <TableHeading

                                name="created\_at"

                                sort\_feild={queryParams.sort\_feild}

                                sort\_direction={queryParams.sort\_direction}

                                sortChanged={sortChanged}

                            >

                                Created Date

                            </TableHeading>

                            <TableHeading

                                name="due\_date"

                                sort\_feild={queryParams.sort\_feild}

                                sort\_direction={queryParams.sort\_direction}

                                sortChanged={sortChanged}

                            >

                                Due Date

                            </TableHeading>

                            <th className="px-3 py-3">Created By</th>

                            <th className="px-3 py-3 text-right">Actions</th>

                        </tr>

                    </thead>

                    <thead className="text-xs text-gray-700 uppercase bg-gray-50 dark:bg-gray-700 dark:text-gray-400 border-b-2 border-gray-500">

                        <tr className="text-nowrap">

                            <th className="px-3 py-3"></th>

                            <th className="px-3 py-3"></th>

                            <th className="px-3 py-3">

                                <TextInput

                                    defaultValue={queryParams.name}

                                    className="w-full"

                                    placeholder="Project Name"

                                    onBlur={*(e)* => searchFeildChanged('name', e.target.value)}

                                    onKeyPress={*(e)* => onKeyPress('name', e)}

                                />

                            </th>

                            <th className="px-3 py-3">

                                <SelectInput

                                    defaultValue={queryParams.status}

                                    className="w-full"

                                    onChange={*(e)* => searchFeildChanged('status', e.target.value)}

                                >

                                    <option value="">Select Status</option>

                                    <option value="pending">Pending</option>

                                    <option value="in\_progress">In Progress</option>

                                    <option value="completed">Completed</option>

                                </SelectInput>

                            </th>

                            <th className="px-3 py-3"></th>

                            <th className="px-3 py-3"></th>

                            <th className="px-3 py-3"></th>

                            <th className="px-3 py-3"></th>

                        </tr>

                    </thead>

                    <tbody>

                        {tasks.data.map(task => (

                            <tr className="bg-white border-b dark:bg-gray-800 dark:border-gray-700" key={task.id}>

                                <th className="px-3 py-2">{task.id}</th>

                                <td className="px-3 py-2"><img src={task.image\_path} style={{ width: 60 }} alt="image" /></td>

                                <td className="px-3 py-2">{task.name}</td>

                                <td className="px-3 py-2">

                                    <span className={"px-2 py-1 rounded text-white " + TASK\_STATUS\_CLASS\_MAP[task.status]}>

                                        {TASK\_STATUS\_TEXT\_MAP[task.status]}

                                    </span>

                                </td>

                                <td className="px-3 py-2 text-nowrap">{task.created\_at}</td>

                                <td className="px-3 py-2 text-nowrap">{task.due\_date}</td>

                                <td className="px-3 py-2">{task.createdBy.name}</td>

                                <td className="px-3 py-2">

                                    <Link href={route('task.edit', task.id)}

                                        className="font-medium text-blue-600 dark:text-blue-500 hover:underline mx-1">

                                        Edit

                                    </Link>

                                    <Link href={route('task.destroy', task.id)}

                                        className="font-medium text-red-600 dark:text-red-500 hover:underline mx-1">

                                        Delete

                                    </Link>

                                </td>

                            </tr>

                        ))}

                    </tbody>

                </table>

            </div>

            <Pagination links={tasks.meta.links} />

        </>

    )

}

export default TasksTable

Task/Index.jsx will look like this

import AuthenticatedLayout from "@/Layouts/AuthenticatedLayout";

import { Head } from "@inertiajs/react";

import TasksTable from "./TasksTable";

export default function Index*({ auth, tasks, queryParams = null })* {

    return (

        <AuthenticatedLayout

            user={auth.user}

            header={<h2 className="font-semibold text-xl text-gray-800 dark:text-gray-200 leading-tight">Tasks</h2>}

        >

            <Head title="Tasks" />

            <div className="py-12">

                <div className="max-w-7xl mx-auto sm:px-6 lg:px-8">

                    <div className="bg-white dark:bg-gray-800 overflow-hidden shadow-sm sm:rounded-lg">

                        <div className="p-6 text-gray-900 dark:text-gray-100">

                            <TasksTable tasks={tasks} queryParams={queryParams} />

                        </div>

                    </div>

                </div>

            </div>

        </AuthenticatedLayout>

    )

}

Next Step is to import TaskTables Component to show.jsx to display the list of the tasks belongs to the projects.

<div className="py-12">

    <div className="max-w-7xl mx-auto sm:px-6 lg:px-8">

        <div className="bg-white dark:bg-gray-800 overflow-hidden shadow-sm sm:rounded-lg">

            <div className="p-6 text-gray-900 dark:text-gray-100">

                Table goes here

            </div>

        </div>

    </div>

</div>

Now we have to replace the Table goes here with TasksTable Component

<div className="py-12">

    <div className="max-w-7xl mx-auto sm:px-6 lg:px-8">

        <div className="bg-white dark:bg-gray-800 overflow-hidden shadow-sm sm:rounded-lg">

            <div className="p-6 text-gray-900 dark:text-gray-100">

                <TasksTable tasks={tasks} queryParams={queryParams} />

            </div>

        </div>

    </div>

</div>

Like that we have to import and add extra params

import TasksTable from '../Task/TasksTable'

const Show = *({ auth, project, tasks, queryParams })* => {}

Now we have to update the ProjectController show() method file to display the tasks related to table

public function show(Project $project)

{

    $query = $project->tasks();

    $sortField = request("sort\_feild", "created\_at");

    $sortDirection = request('sort\_direction', "desc");

    if (request("name")) {

        $query->where("name", "like", "%" . request("name") . "%");

    }

    if (request("status")) {

        $query->where("status", request("status"));

    }

    $tasks = $query->orderBy($sortField, $sortDirection)->paginate(10);

    return inertia("Project/Show", [

        "project" => new ProjectResource($project),

        "tasks" => TaskResource::collection($tasks),

        "queryParams" => request()->query() ?: null,

    ]);

}

Next Step is to include the project name in TasksTable to know the tasks belongs to which project

Under first <thead> tag

<th className="px-3 py-3">Project Name</th>

Under second <thead> tag

<th className="px-3 py-3">

    <TextInput defaultValue={queryParams.name} className="w-full" placeholder="Project Name" onBlur={(e)=>

        searchFeildChanged('project\_name', e.target.value)}

        onKeyPress={(e) => onKeyPress('project\_name', e)}

        />

</th>

Under <tbody> tag

<td className="px-3 py-2 text-nowrap">{task.project.name}</td>

To do that in TasksTable component

const TasksTable = *({ tasks, queryParams = null, hideProjectColumn = false })* => {}

then we have to add Project Columns

Under first <thead> tag

{!hideProjectColumn && (

<th className="px-3 py-3">Project Name</th>

)}

Under second <thead> tag

{!hideProjectColumn && (

<th className="px-3 py-3">

    <TextInput defaultValue={queryParams.name} className="w-full" placeholder="Project Name" onBlur={(e)=>

        searchFeildChanged('project\_name', e.target.value)}

        onKeyPress={(e) => onKeyPress('project\_name', e)}

        />

</th>

)}

Under <tbody> tag

{!hideProjectColumn && (

<td className="px-3 py-2 text-nowrap">{task.project.name}</td>

)}

Now we have to hide the project column in show() method of project Controller so in Project\Show.jsx

<TasksTable tasks={tasks} queryParams={queryParams} hideProjectColumn={true} />

In TaskController index() method to enable search of project in task table

$query = Task::query()->with('project');

if (request("project\_name")) {

    $query->whereHas('project', function ($query) {

        $query->where("name", "like", "%" . request("project\_name") . "%");

    });

}

Content width increased from 7xl to 9xl in Project/Show.jsx, Project/Index.jsx, Task/Index.jsx

<div className="max-w-9xl mx-auto sm:px-6 lg:px-8">

Creating Projects

Now to Create the **Add New** Button in Project/Index.jsx

from

header={

<h2 className="font-semibold text-xl text-gray-800 dark:text-gray-200 leading-tight">Projects</h2>

}

to

header={

    <div className="flex items-center justify-between">

        <h2 className="font-semibold text-xl text-gray-800 dark:text-gray-200 leading-tight">Projects</h2>

       <Link className="bg-emerald-500 text-white px-3 py-1 rounded hover:bg-emerald-600" href={route('project.create')}> Add New</Link>

    </div>

}

Next step is to create a Create.jsx file in Project Folder

import React from 'react'

const Create = *()* => {

  return (

    <div>Create</div>

  )

}

export default Create

And Link it in ProjectController to return the Create ReactCompontent

public function create()

{

return inertia("Project/Create");

}

Now Update the Create.jsx of Project to add authenticated and form block

import Authenticated from '@/Layouts/AuthenticatedLayout'

import { Head } from '@inertiajs/react'

import React from 'react'

const Create = *({ auth })* => {

    return (

        <Authenticated

            user={auth.user}

            header={

                <h2 className="font-semibold text-xl text-gray-800 dark:text-gray-200 leading-tight"> Create New Project</h2>

            }

        >

            <Head title="Projects" />

            <div className="py-12">

                <div className="max-w-9xl mx-auto sm:px-6 lg:px-8">

                    <div className="bg-white dark:bg-gray-800 overflow-hidden shadow-sm sm:rounded-lg">

                        <div className="p-6 text-gray-900 dark:text-gray-100">

                            Form

                        </div>

                    </div>

                </div>

            </div>

        </Authenticated>

    )

}

export default Create

Create TexAreaInput Component:-

import { forwardRef, useEffect, useRef } from 'react';

export default forwardRef(function TextAreaInput*({ className = '', isFocused = false, children, ...props }, ref)* {

    const input = ref ? ref : useRef();

    useEffect(*()* => {

        if (isFocused) {

            input.current.focus();

        }

    }, []);

    return (

        <textarea

            {...props}

            className={

                'border-gray-300 dark:border-gray-700 dark:bg-gray-900 dark:text-gray-300 focus:border-indigo-500 dark:focus:border-indigo-600 focus:ring-indigo-500 dark:focus:ring-indigo-600 rounded-md shadow-sm ' +

                className

            }

            ref={input}

        >

            {children}

        </textarea>

    );

});

Now Update Create.jsx Component of Project to add the form elements

import InputError from '@/Components/InputError'

import InputLabel from '@/Components/InputLabel'

import SelectInput from '@/Components/SelectInput'

import TextAreaInput from '@/Components/TextAreaInput'

import TextInput from '@/Components/TextInput'

import Authenticated from '@/Layouts/AuthenticatedLayout'

import { Head, useForm } from '@inertiajs/react'

import React from 'react'

const Create = *({ auth })* => {

    const { data, setData, post, errors, reset } = useForm({

        image: '',

        name: '',

        description: '',

        status: '',

        due\_date: ''

    });

    const onSubmit = *(e)* => {

        e.preventDefault();

        post(route('project.store'));

    };

    return (

        <Authenticated

            user={auth.user}

            header={

                <h2 className="font-semibold text-xl text-gray-800 dark:text-gray-200 leading-tight"> Create New Project</h2>

            }

        >

            <Head title="Projects" />

            <div className="py-12">

                <div className="max-w-9xl mx-auto sm:px-6 lg:px-8">

                    <div className="bg-white dark:bg-gray-800 overflow-hidden shadow-sm sm:rounded-lg">

                        <form onSubmit={onSubmit} className="p-4 sm:p-8 bg-white dark:bg-gray-800 shadow sm:rounded-lg">

                            <div>

                                <InputLabel

                                    htmlFor="project\_image\_path"

                                    value="Project Image"

                                />

                                <TextInput

                                    id="project\_image\_path"

                                    type="file"

                                    name="image"

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('image', e.target.files[0])}

                                />

                                <InputError message={errors.image} className="mt-2" />

                            </div>

                            <div className='mt-4'>

                                <InputLabel

                                    htmlFor="project\_name"

                                    value="Project Name"

                                />

                                <TextInput

                                    id="project\_name"

                                    type="text"

                                    name="name"

                                    value={data.name}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('name', e.target.value)}

                                />

                                <InputError message={errors.name} className="mt-2" />

                            </div>

                            <div className='mt-4'>

                                <InputLabel

                                    htmlFor="project\_description"

                                    value="Project Description"

                                />

                                <TextAreaInput

                                    id="project\_description"

                                    name="description"

                                    value={data.description}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('description', e.target.value)}

                                />

                                <InputError message={errors.description} className="mt-2" />

                            </div>

                            <div className='mt-4'>

                                <InputLabel

                                    htmlFor="project\_due\_date"

                                    value="Project Deadline"

                                />

                                <TextInput

                                    id="project\_due\_date"

                                    type="date"

                                    name="due\_date"

                                    value={data.due\_date}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('due\_date', e.target.value)}

                                />

                                <InputError message={errors.due\_date} className="mt-2" />

                            </div>

                            <div className='mt-4'>

                                <InputLabel

                                    htmlFor="project\_status"

                                    value="Project Status"

                                />

                                <SelectInput

                                    id="project\_status"

                                    name="status"

                                    value={data.status}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('status', e.target.value)}

                                >

                                    <option value="">Select Status</option>

                                    <option value="pending">Pending</option>

                                    <option value="in\_progress">In Progress</option>

                                    <option value="completed">Completed</option>

                                </SelectInput>

                                <InputError message={errors.status} className="mt-2" />

                            </div>

                        </form>

                    </div>

                </div>

            </div>

        </Authenticated>

    )

}

export default Create

Add the cancel and submit button in Create.jsx Component of Project

<div className='mt-4 text-right'>

    <Link href={route("project.index")} className='bg-gray-100 py-1 px-3 text-gray-800 rounded hover:bg-gray-300 mr-2'>

        Cancel

    </Link>

    <button className='bg-emerald-500 text-white px-3 py-1 rounded hover:bg-emerald-600'>

        Submit

    </button>

</div>

Project Model Updated to get by adding fillable

protected $fillable = ['image\_path', 'name', 'description', 'status', 'due\_date', 'created\_by', 'updated\_by'];

In ProjectController, store() method is updated to

public function store(*StoreProjectRequest $request*)

    {

        $data = $request->validated();

        $data['created\_by'] = Auth::id();

        $data['updated\_by'] = Auth::id();

        Project::create($data);

        return to\_route('project.index');

    }

StoreProjectRequest is updated

public function authorize(): bool

    {

        return true;

    }

public function rules(): array

    {

        return [

            'name' => ['required', 'max:255'],

            'image' => ['nullable', 'image'],

            'description' => ['nullable', 'string'],

            'due\_date' => ['nullable', 'date'],

            'status' => ['required', Rule::in(['pending', 'in\_progress', 'completed'])]

        ];

    }

Now we send the success message after project created successfully

In ProjectController -> store()

return to\_route('project.index');

To

return to\_route('project.index')->with("success", "Project created successfully.");

In ProjectController -> index()

return inertia("Project/Index", [

            "projects" => ProjectResource::collection($projects),

            "queryParams" => request()->query() ?: null,

        ]);

To

return inertia("Project/Index", [

            "projects" => ProjectResource::collection($projects),

            "queryParams" => request()->query() ?: null,

            "success" => session("success"),

        ]);

Now Update the Index.jsx of Project to display the success message

export default function Index*({ auth, projects, queryParams = null })* {}

To

export default function Index*({ auth, projects, queryParams = null, success })* {}

<div className="max-w-9xl …

{success && (

<div className="bg-emerald-500 py-2 px-4 text-white rounded mb-4">

     {success}

 </div>

)}

ProjectController store() method is updated to upload image and update image\_path

public function store(*StoreProjectRequest $request*)

    {

        $data = $request->validated();

        $image = $data['image'] ?? null;

        $data['created\_by'] = Auth::id();

        $data['updated\_by'] = Auth::id();

        if($image){

           $data['image\_path'] = $image->store('project/'.Str::random(), 'public');

        }

        Project::create($data);

        return to\_route('project.index')->with("success", "Project created successfully.");

    }

ProjectResource updated

'image\_path' => $this->image\_path,

To

'image\_path' => $this->image\_path ? Storage::url($this->image\_path) : '',

StoreProjectRequest updated

'description' => ['string'],

To

'description' => ['nullable','string'],

Project Deleting

Update Project/Index.jsx

<Link

href={route('project.destroy', project.id)}

      className="font-medium text-red-600 dark:text-red-500 hover:underline mx-1"

>

Delete

</Link>

To

<button

onClick={*(e)* => deleteProject(project)}

className="font-medium text-red-600 dark:text-red-500 hover:underline mx-1"

>

Delete

</button>

Add deleteProject()

const deleteProject = *(project)* => {

        if (!window.confirm('Are you sure want to delete the project?')) {

            return;

        }

        router.delete(route('project.destroy', project.id))

};

ProjectController destroy() method is updated for deleting the project

public function destroy(*Project $project*)

    {

        $projectName = $project->name;

if ($project->image\_path) {

            Storage::disk('public')->deleteDirectory(dirname($project->image\_path));

        }

        $project->delete();

        return to\_route('project.index')->with("success", "Project " . $projectName . " deleted successfully.");

    }

Editing Project

In ProjectController, edit() method updated:-

public function edit(*Project $project*)

    {

        return inertia("Project/Edit", [

            "project" => new ProjectResource($project),

        ]);

    }

Next step is to create Edit Component for Project by copy paste Create Component of Project and do some modifications

const Edit = *({ auth, project })* => {

    const { data, setData, post, errors, reset } = useForm({

        image: '',

        name: project.name || '',

        description: project.description || '',

        status: project.status || '',

        due\_date: project.due\_date || '',

\_method: 'PUT'

    });

const onSubmit = *(e)* => {

        e.preventDefault();

        post(route("project.update", project.id));

    };

return (…)

}

 <form onSubmit={onSubmit} ….

{project.image\_path &&

<div className='mb-4'>

          <img src={project.image\_path} alt={project.name} className='w-full h-64' />

      </div>

}

export default Edit

Now like create we need to all things to update the project

In ProjectController, update() method updated:-

public function update(*UpdateProjectRequest $request, Project $project*)

    {

        $data = $request->validated();

        $image = $data['image'] ?? null;

        $data['updated\_by'] = Auth::id();

        if ($image) {

            if ($project->image\_path) {

                Storage::disk('public')->deleteDirectory(dirname($project->image\_path));

            }

            $data['image\_path'] = $image->store('project/' . Str::random(), 'public');

        }

        $project->update($data);

        return to\_route('project.index')->with("success", "Project " . $project->name . " updated successfully.");

    }

Then in UpdateProjectRequest: -

public function authorize(): bool

    {

        return true;

    }

public function rules(): array

    {

        return [

            'name' => ['required', 'max:255'],

            'image' => ['nullable', 'image'],

            'description' => ['nullable', 'string'],

            'due\_date' => ['nullable', 'date'],

            'status' => ['required', Rule::in(['pending', 'in\_progress', 'completed'])]

        ];

    }

Filtering the task by specific user for that we need to create the components of User by copy and paste the Project folder and rename to User folder

And then replace all the words of PROJECT, Project, project to USER, User, user respectively

Then update the Index.jsx of User Folder by removing unwanted columns and update the required column like this

import Pagination from "@/Components/Pagination";

import TextInput from "@/Components/TextInput";

import AuthenticatedLayout from "@/Layouts/AuthenticatedLayout";

import { Head, Link, router } from "@inertiajs/react";

import TableHeading from "@/Components/TableHeading";

export default function Index*({ auth, users, queryParams = null, success })* {

    queryParams = queryParams || {}

    const searchFeildChanged = *(name, value)* => {

        if (value) {

            queryParams[name] = value

        }

        else {

            delete queryParams[name]

        }

        router.get(route('user.index'), queryParams);

    };

    const onKeyPress = *(name, e)* => {

        if (e.key !== 'Enter') return;

        searchFeildChanged(name, e.target.value);

    };

    const sortChanged = *(name)* => {

        if (name === queryParams.sort\_feild) {

            if (queryParams.sort\_direction === 'asc') {

                queryParams.sort\_direction = 'desc';

            } else {

                queryParams.sort\_direction = 'asc';

            }

        } else {

            queryParams.sort\_feild = name;

            queryParams.sort\_direction = 'asc';

        }

        router.get(route('user.index'), queryParams);

    };

    const deleteUser = *(user)* => {

        if (!window.confirm('Are you sure want to delete the user?')) {

            return;

        }

        router.delete(route('user.destroy', user.id))

    };

    return (

        <AuthenticatedLayout

            user={auth.user}

            header={

                <div className="flex items-center justify-between">

                    <h2 className="font-semibold text-xl text-gray-800 dark:text-gray-200 leading-tight">Users</h2>

                    <Link className="bg-emerald-500 text-white px-3 py-1 rounded hover:bg-emerald-600" href={route('user.create')}> Add New</Link>

                </div>

            }

        >

            <Head title="Users" />

            <div className="py-12">

                <div className="max-w-9xl mx-auto sm:px-6 lg:px-8">

                    {success && (

                        <div className="bg-emerald-500 py-2 px-4 text-white rounded mb-4">

                            {success}

                        </div>

                    )}

                    <div className="bg-white dark:bg-gray-800 overflow-hidden shadow-sm sm:rounded-lg">

                        <div className="p-6 text-gray-900 dark:text-gray-100">

                            <div className="overflow-auto">

                                <table className="w-full text-sm text-left rtl:text-right text-gray-500 dark:text-gray-400">

                                    <thead className="text-xs text-gray-700 uppercase bg-gray-50 dark:bg-gray-700 dark:text-gray-400 border-b-2 border-gray-500">

                                        <tr className="text-nowrap">

                                            <TableHeading

                                                name="id"

                                                sort\_feild={queryParams.sort\_feild}

                                                sort\_direction={queryParams.sort\_direction}

                                                sortChanged={sortChanged}

                                            >

                                                ID

                                            </TableHeading>

                                            <TableHeading

                                                name="name"

                                                sort\_feild={queryParams.sort\_feild}

                                                sort\_direction={queryParams.sort\_direction}

                                                sortChanged={sortChanged}

                                            >

                                                Name

                                            </TableHeading>

                                            <TableHeading

                                                name="email"

                                                sort\_feild={queryParams.sort\_feild}

                                                sort\_direction={queryParams.sort\_direction}

                                                sortChanged={sortChanged}

                                            >

                                                Email

                                            </TableHeading>

                                            <TableHeading

                                                name="created\_at"

                                                sort\_feild={queryParams.sort\_feild}

                                                sort\_direction={queryParams.sort\_direction}

                                                sortChanged={sortChanged}

                                            >

                                                Created Date

                                            </TableHeading>

                                            <th className="px-3 py-3 text-right">Actions</th>

                                        </tr>

                                    </thead>

                                    <thead className="text-xs text-gray-700 uppercase bg-gray-50 dark:bg-gray-700 dark:text-gray-400 border-b-2 border-gray-500">

                                        <tr className="text-nowrap">

                                            <th className="px-3 py-3"></th>

                                            <th className="px-3 py-3">

                                                <TextInput

                                                    defaultValue={queryParams.name}

                                                    className="w-full"

                                                    placeholder="User Name"

                                                    onBlur={*(e)* => searchFeildChanged('name', e.target.value)}

                                                    onKeyPress={*(e)* => onKeyPress('name', e)}

                                                />

                                            </th>

                                            <th className="px-3 py-3">

                                                <TextInput

                                                    defaultValue={queryParams.email}

                                                    className="w-full"

                                                    placeholder="User Email"

                                                    onBlur={*(e)* => searchFeildChanged('email', e.target.value)}

                                                    onKeyPress={*(e)* => onKeyPress('email', e)}

                                                />

                                            </th>

                                            <th className="px-3 py-3"></th>

                                            <th className="px-3 py-3"></th>

                                        </tr>

                                    </thead>

                                    <tbody>

                                        {users.data.map(user => (

                                            <tr className="bg-white border-b dark:bg-gray-800 dark:border-gray-700" key={user.id}>

                                                <th className="px-3 py-2 text-nowrap">{user.id}</th>

                                                <td className="px-3 py-2 text-nowrap">{user.name}</td>

                                                <td className="px-3 py-2 text-nowrap">{user.email}</td>

                                                <td className="px-3 py-2 text-nowrap">{user.created\_at}</td>

                                                <td className="px-3 py-2 text-nowrap">

                                                    <Link href={route('user.edit', user.id)}

                                                        className="font-medium text-blue-600 dark:text-blue-500 hover:underline mx-1">

                                                        Edit

                                                    </Link>

                                                    <button onClick={*(e)* => deleteUser(user)}

                                                        className="font-medium text-red-600 dark:text-red-500 hover:underline mx-1">

                                                        Delete

                                                    </button>

                                                </td>

                                            </tr>

                                        ))}

                                    </tbody>

                                </table>

                            </div>

                            <Pagination links={users.meta.links} />

                        </div>

                    </div>

                </div>

            </div>

        </AuthenticatedLayout>

    )

}

Now update the index() of UserController like this

public function index()

    {

        $query = User::query();

        $sortField = request("sort\_feild", "created\_at");

        $sortDirection = request('sort\_direction', "desc");

        if (request("name")) {

            $query->where("name", "like", "%" . request("name") . "%");

        }

        if (request("email")) {

            $query->where("email", "like", "%" . request("email") . "%");

        }

        $users = $query->orderBy($sortField, $sortDirection)->paginate(10);

        return inertia("User/Index", [

            'users' => UserResource::collection($users),

            "queryParams" => request()->query() ?: null,

            "success" => session("success"),

        ]);

    }

Now to display the Add New User form update the component Create.jsx of User

import InputError from '@/Components/InputError'

import InputLabel from '@/Components/InputLabel'

import TextInput from '@/Components/TextInput'

import Authenticated from '@/Layouts/AuthenticatedLayout'

import { Head, Link, useForm } from '@inertiajs/react'

import React from 'react'

const Create = *({ auth })* => {

    const { data, setData, post, errors, reset } = useForm({

        name: '',

        email: '',

        password: '',

        password\_confirmation: ''

    });

    const onSubmit = *(e)* => {

        e.preventDefault();

        post(route('user.store'));

    };

    return (

        <Authenticated

            user={auth.user}

            header={

                <h2 className="font-semibold text-xl text-gray-800 dark:text-gray-200 leading-tight"> Create New User</h2>

            }

        >

            <Head title="Users" />

            <div className="py-12">

                <div className="max-w-9xl mx-auto sm:px-6 lg:px-8">

                    <div className="bg-white dark:bg-gray-800 overflow-hidden shadow-sm sm:rounded-lg">

                        <form onSubmit={onSubmit} className="p-4 sm:p-8 bg-white dark:bg-gray-800 shadow sm:rounded-lg">

                            <div>

                                <InputLabel

                                    htmlFor="user\_name"

                                    value="User Name"

                                />

                                <TextInput

                                    id="user\_name"

                                    type="text"

                                    name="name"

                                    value={data.name}

isFocused={true}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('name', e.target.value)}

                                />

                                <InputError message={errors.name} className="mt-2" />

                            </div>

                            <div className='mt-4'>

                                <InputLabel

                                    htmlFor="user\_email"

                                    value="User Email"

                                />

                                <TextInput

                                    id="user\_email"

                                    type="text"

                                    name="email"

                                    value={data.email}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('email', e.target.value)}

                                />

                                <InputError message={errors.email} className="mt-2" />

                            </div>

                            <div className='mt-4'>

                                <InputLabel

                                    htmlFor="user\_password"

                                    value="Password"

                                />

                                <TextInput

                                    id="user\_password"

                                    type="password"

                                    name="password"

                                    value={data.password}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('password', e.target.value)}

                                />

                                <InputError message={errors.password} className="mt-2" />

                            </div>

                            <div className='mt-4'>

                                <InputLabel

                                    htmlFor="user\_password\_confirmation"

                                    value="Password Confirmation"

                                />

                                <TextInput

                                    id="user\_password\_confirmation"

                                    type="password"

                                    name="password\_confirmation"

                                    value={data.password\_confirmation}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('password\_confirmation', e.target.value)}

                                />

                                <InputError message={errors.password\_confirmation} className="mt-2" />

                            </div>

                            <div className='mt-4 text-right'>

                                <Link

                                    href={route("user.index")}

                                    className='bg-gray-100 py-1 px-3 text-gray-800 rounded hover:bg-gray-300 mr-2'

                                >

                                    Cancel

                                </Link>

                                <button className='bg-emerald-500 text-white px-3 py-1 rounded hover:bg-emerald-600'>

                                    Submit

                                </button>

                            </div>

                        </form>

                    </div>

                </div>

            </div>

        </Authenticated>

    )

}

export default Create

Now update the create() method of UserController

public function create()

    {

        return inertia("User/Create");

    }

Now update the StoreUserRequest to validate user form data

public function authorize(): bool

    {

        return true;

    }

public function rules(): array

    {

        return [

            'name' => ['required', 'string', 'max:255'],

            'email' => ['required', 'email', 'max:255', 'unique:users,email'],

            'password' => [

                'required',

                'confirmed',

                Password::min(8)->letters()->symbols()

            ],

        ];

    }

And Update the store() method of UserController

public function store(*StoreUserRequest $request*)

    {

        $data = $request->validated();

        $data['password'] = bcrypt($data['password']);

        User::create($data);

        return to\_route('user.index')->with("success", "User created successfully.");

    }

To display the edit form update the edit()method of UserContoller

public function edit(*User $user*)

    {

        return inertia("User/Edit", [

            "user" => new UserResource($user),

        ]);

    }

And update the component edit.jsx of User

import InputError from '@/Components/InputError'

import InputLabel from '@/Components/InputLabel'

import TextInput from '@/Components/TextInput'

import Authenticated from '@/Layouts/AuthenticatedLayout'

import { Head, Link, useForm } from '@inertiajs/react'

import React from 'react'

const Edit = *({ auth, user })* => {

    const { data, setData, post, errors, reset } = useForm({

        name: user.name || '',

        email: user.email || '',

        password: '',

        password\_confirmation: '',

        \_method: 'PUT'

    });

    const onSubmit = *(e)* => {

        e.preventDefault();

        post(route("user.update", user.id));

    };

    return (

        <Authenticated

            user={auth.user}

            header={

                <h2 className="font-semibold text-xl text-gray-800 dark:text-gray-200 leading-tight">

                    Edit User : "{user.name}"

                </h2>

            }

        >

            <Head title="Users" />

            <div className="py-12">

                <div className="max-w-9xl mx-auto sm:px-6 lg:px-8">

                    <div className="bg-white dark:bg-gray-800 overflow-hidden shadow-sm sm:rounded-lg">

                        <form onSubmit={onSubmit} className="p-4 sm:p-8 bg-white dark:bg-gray-800 shadow sm:rounded-lg">

                            <div>

                                <InputLabel

                                    htmlFor="user\_name"

                                    value="User Name"

                                />

                                <TextInput

                                    id="user\_name"

                                    type="text"

                                    name="name"

                                    value={data.name}

                                    isFocused={true}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('name', e.target.value)}

                                />

                                <InputError message={errors.name} className="mt-2" />

                            </div>

                            <div className='mt-4'>

                                <InputLabel

                                    htmlFor="user\_email"

                                    value="User Email"

                                />

                                <TextInput

                                    id="user\_email"

                                    type="text"

                                    name="email"

                                    value={data.email}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('email', e.target.value)}

                                />

                                <InputError message={errors.email} className="mt-2" />

                            </div>

                            <div className='mt-4'>

                                <InputLabel

                                    htmlFor="user\_password"

                                    value="Password"

                                />

                                <TextInput

                                    id="user\_password"

                                    type="password"

                                    name="password"

                                    value={data.password}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('password', e.target.value)}

                                />

                                <InputError message={errors.password} className="mt-2" />

                            </div>

                            <div className='mt-4'>

                                <InputLabel

                                    htmlFor="user\_password\_confirmation"

                                    value="Password Confirmation"

                                />

                                <TextInput

                                    id="user\_password\_confirmation"

                                    type="password"

                                    name="password\_confirmation"

                                    value={data.password\_confirmation}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('password\_confirmation', e.target.value)}

                                />

                                <InputError message={errors.password\_confirmation} className="mt-2" />

                            </div>

                            <div className='mt-4 text-right'>

                                <Link

                                    href={route("user.index")}

                                    className='bg-gray-100 py-1 px-3 text-gray-800 rounded hover:bg-gray-300 mr-2'

                                >

                                    Cancel

                                </Link>

                                <button className='bg-emerald-500 text-white px-3 py-1 rounded hover:bg-emerald-600'>

                                    Submit

                                </button>

                            </div>

                        </form>

                    </div>

                </div>

            </div>

        </Authenticated>

    )

}

export default Edit

Add the $wrap = false to UserResource

public static $wrap = false;

Update the UserController update() method to edit the user data

public function update(*UpdateUserRequest $request, User $user*)

    {

        $data = $request->validated();

        $password = $data['password'] ?? null;

        if ($password) {

            $data['password'] = bcrypt($data['password']);

        } else {

            unset($data['password']);

        }

        $user->update($data);

        return to\_route('user.index')->with("success", "User " . $user->name . " updated successfully.");

    }

UpdateUserRequest updated to authorize and validate the edit form values

public function authorize(): bool

    {

        return true;

    }

public function rules(): array

    {

        $user = $this->route('user');

        return [

            'name' => ['required', 'string', 'max:255'],

            'email' => [

                'required',

                'email',

                'max:255',

                Rule::unique('users')->ignore($user->id),

            ],

            'password' => [

                'nullable',

                'confirmed',

                Password::min(8)->letters()->symbols()

            ],

        ];

    }

And Update the store() method of UserController to enable email\_verified\_at

 public function store(*StoreUserRequest $request*)

    {

        $data = $request->validated();

        $data['email\_verified\_at'] = time();

        $data['password'] = bcrypt($data['password']);

        User::create($data);

        return to\_route('user.index')->with("success", "User created successfully.");

    }

And in User Model add the email\_verified\_at property to fillable

protected $fillable = [

        'name',

        'email',

        'password',

        'email\_verified\_at'

    ];

Now UserController destroy() method updated to delete the user

public function destroy(*User $user*)

    {

        $userName = $user->name;

        $user->delete();

        return to\_route('user.index')->with("success", "User " . $userName . " deleted successfully.");

    }

Now we do the same thing for Task

we need to copy and paste from Project folder (create.jsx, edit,jsx, show.jsx) to Taks folder

And then replace all the words of PROJECT, Project, project to TASK, Task, task respectively and don’t change anything in TasksTable.jsx

And then copy and paste methods content from ProjectController to TaskController and replace all the words of PROJECT, Project, project to TASK, Task, task respectively

Now Update to add the Add New Button to Task/Index.jsx

<AuthenticatedLayout

user={auth.user}

header={<h2 className="font-semibold text-xl text-gray-800 dark:text-gray-200 leading-tight">Tasks</h2>}

        >

To

<AuthenticatedLayout

user={auth.user}

      header={

                <div className="flex items-center justify-between">

                    <h2 className="font-semibold text-xl text-gray-800 dark:text-gray-200 leading-tight">Tasks</h2>

                    <Link className="bg-emerald-500 text-white px-3 py-1 rounded hover:bg-emerald-600" href={route('task.create')}> Add New</Link>

                </div>

            }

        >

TaskController, create() method updated to pass projects and users to Task/Create.jsx

public function create()

    {

        $projects = Project::query()->orderBy('name', 'asc')->get();

        $users = User::all();

        return inertia("Task/Create", [

            'projects' => ProjectResource::collection($projects),

            'users' => UserResource::collection($users),

        ]);

    }

Task/Create.jsx updated to add additional form elements

import InputError from '@/Components/InputError'

import InputLabel from '@/Components/InputLabel'

import SelectInput from '@/Components/SelectInput'

import TextAreaInput from '@/Components/TextAreaInput'

import TextInput from '@/Components/TextInput'

import Authenticated from '@/Layouts/AuthenticatedLayout'

import { Head, Link, useForm } from '@inertiajs/react'

import React from 'react'

const Create = *({ auth, projects, users })* => {

    const { data, setData, post, errors, reset } = useForm({

        image: '',

        name: '',

        description: '',

        status: '',

        priority: '',

        project\_id: '',

        assigned\_user\_id: '',

        due\_date: ''

    });

    const onSubmit = *(e)* => {

        e.preventDefault();

        post(route('task.store'));

    };

    return (

        <Authenticated

            user={auth.user}

            header={

                <h2 className="font-semibold text-xl text-gray-800 dark:text-gray-200 leading-tight"> Create New Task</h2>

            }

        >

            <Head title="Tasks" />

            <div className="py-12">

                <div className="max-w-9xl mx-auto sm:px-6 lg:px-8">

                    <div className="bg-white dark:bg-gray-800 overflow-hidden shadow-sm sm:rounded-lg">

                        <form onSubmit={onSubmit} className="p-4 sm:p-8 bg-white dark:bg-gray-800 shadow sm:rounded-lg">

                            <div>

                                <InputLabel

                                    htmlFor="project\_id"

                                    value="Project"

                                />

                                <SelectInput

                                    id="project\_id"

                                    name="project\_id"

                                    value={data.project\_id}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('project\_id', e.target.value)}

                                >

                                    <option value="">Select Project</option>

                                    {projects.data.map(project => (

                                        <option value={project.id} key={project.id} >{project.name}</option>

                                    ))}

                                </SelectInput>

                                <InputError message={errors.project\_id} className="mt-2" />

                            </div>

                            <div className='mt-4'>

                                <InputLabel

                                    htmlFor="task\_image\_path"

                                    value="Task Image"

                                />

                                <TextInput

                                    id="task\_image\_path"

                                    type="file"

                                    name="image"

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('image', e.target.files[0])}

                                />

                                <InputError message={errors.image} className="mt-2" />

                            </div>

                            <div className='mt-4'>

                                <InputLabel

                                    htmlFor="task\_name"

                                    value="Task Name"

                                />

                                <TextInput

                                    id="task\_name"

                                    type="text"

                                    name="name"

                                    value={data.name}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('name', e.target.value)}

                                />

                                <InputError message={errors.name} className="mt-2" />

                            </div>

                            <div className='mt-4'>

                                <InputLabel

                                    htmlFor="task\_description"

                                    value="Task Description"

                                />

                                <TextAreaInput

                                    id="task\_description"

                                    name="description"

                                    value={data.description}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('description', e.target.value)}

                                />

                                <InputError message={errors.description} className="mt-2" />

                            </div>

                            <div className='mt-4'>

                                <InputLabel

                                    htmlFor="task\_due\_date"

                                    value="Task Deadline"

                                />

                                <TextInput

                                    id="task\_due\_date"

                                    type="date"

                                    name="due\_date"

                                    value={data.due\_date}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('due\_date', e.target.value)}

                                />

                                <InputError message={errors.due\_date} className="mt-2" />

                            </div>

                            <div className='mt-4'>

                                <InputLabel

                                    htmlFor="task\_status"

                                    value="Task Status"

                                />

                                <SelectInput

                                    id="task\_status"

                                    name="status"

                                    value={data.status}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('status', e.target.value)}

                                >

                                    <option value="">Select Status</option>

                                    <option value="pending">Pending</option>

                                    <option value="in\_progress">In Progress</option>

                                    <option value="completed">Completed</option>

                                </SelectInput>

                                <InputError message={errors.status} className="mt-2" />

                            </div>

                            <div className='mt-4'>

                                <InputLabel

                                    htmlFor="task\_priority"

                                    value="Task Priority"

                                />

                                <SelectInput

                                    id="task\_priority"

                                    name="priority"

                                    value={data.priority}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('priority', e.target.value)}

                                >

                                    <option value="">Select Priority</option>

                                    <option value="low">Low</option>

                                    <option value="medium">Medium</option>

                                    <option value="high">High</option>

                                </SelectInput>

                                <InputError message={errors.priority} className="mt-2" />

                            </div>

                            <div className='mt-4'>

                                <InputLabel

                                    htmlFor="assigned\_user\_id"

                                    value="Assigned User"

                                />

                                <SelectInput

                                    id="assigned\_user\_id"

                                    name="assigned\_user\_id"

                                    value={data.assigned\_user\_id}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('assigned\_user\_id', e.target.value)}

                                >

                                    <option value="">Select User</option>

                                    {users.data.map(user => (

                                        <option value={user.id} key={user.id} >{user.name}</option>

                                    ))}

                                </SelectInput>

                                <InputError message={errors.assigned\_user\_id} className="mt-2" />

                            </div>

                            <div className='mt-4 text-right'>

                                <Link

                                    href={route("task.index")}

                                    className='bg-gray-100 py-1 px-3 text-gray-800 rounded hover:bg-gray-300 mr-2'

                                >

                                    Cancel

                                </Link>

                                <button className='bg-emerald-500 text-white px-3 py-1 rounded hover:bg-emerald-600'>

                                    Submit

                                </button>

                            </div>

                        </form>

                    </div>

                </div>

            </div>

        </Authenticated>

    )

}

export default Create

TaskResource image\_path updated

 'image\_path' => $this->image\_path,

To

 'image\_path' => $this->image\_path ? Storage::url($this->image\_path) : '',

StoreTaskRequest Updated

public function authorize(): bool

    {

        return true;

    }

public function rules(): array

    {

        return [

            'name' => ['required', 'max:255'],

            'image' => ['nullable', 'image'],

            'description' => ['nullable','string'],

            'due\_date' => ['nullable', 'date'],

            'project\_id' => ['required', 'exists:projects,id'],

            'assigned\_user\_id' => ['required', 'exists:users,id'],

            'status' => ['required', Rule::in(['pending', 'in\_progress', 'completed'])],

            'priority' => ['required', Rule::in(['low', 'medium', 'high'])]

        ];

    }

And in Task model fillable property added

 protected $fillable = [

        'name',

        'description',

        'image\_path',

        'due\_date',

        'status',

        'priority',

        'assigned\_user\_id',

        'created\_by',

        'updated\_by',

        'project\_id'

    ];

Now we need to update the Edit.jsx of Task

import InputError from '@/Components/InputError'

import InputLabel from '@/Components/InputLabel'

import SelectInput from '@/Components/SelectInput'

import TextAreaInput from '@/Components/TextAreaInput'

import TextInput from '@/Components/TextInput'

import Authenticated from '@/Layouts/AuthenticatedLayout'

import { Head, Link, useForm } from '@inertiajs/react'

import React from 'react'

const Edit = *({ auth, task, projects, users })* => {

    const { data, setData, post, errors, reset } = useForm({

        image: '',

        name: task.name || '',

        description: task.description || '',

        status: task.status || '',

        due\_date: task.due\_date || '',

        project\_id: task.project\_id || '',

        priority: task.priority || '',

        assigned\_user\_id: task.assigned\_user\_id || '',

        \_method: 'PUT'

    });

    const onSubmit = *(e)* => {

        e.preventDefault();

        post(route("task.update", task.id));

    };

    return (

        <Authenticated

            user={auth.user}

            header={

                <h2 className="font-semibold text-xl text-gray-800 dark:text-gray-200 leading-tight">

                    Edit Task : "{task.name}"

                </h2>

            }

        >

            <Head title="Tasks" />

            <div className="py-12">

                <div className="max-w-9xl mx-auto sm:px-6 lg:px-8">

                    <div className="bg-white dark:bg-gray-800 overflow-hidden shadow-sm sm:rounded-lg">

                        <form onSubmit={onSubmit} className="p-4 sm:p-8 bg-white dark:bg-gray-800 shadow sm:rounded-lg">

                            <div className='mb-4'>

                                <InputLabel

                                    htmlFor="task\_project\_id"

                                    value="Project"

                                />

                                <SelectInput

                                    id="task\_project\_id"

                                    name="project\_id"

                                    value={data.project\_id}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('project\_id', e.target.value)}

                                >

                                    <option value="">Select Project</option>

                                    {projects.data.map(project => (

                                        <option value={project.id} key={project.id}>{project.name}</option>

                                    ))}

                                </SelectInput>

                                <InputError message={errors.project\_id} className="mt-2" />

                            </div>

                            {task.image\_path &&

                                <div className='mb-4'>

                                    <img src={task.image\_path} alt={task.name} className='w-full h-64' />

                                </div>

                            }

                            <div>

                                <InputLabel

                                    htmlFor="task\_image\_path"

                                    value="Task Image"

                                />

                                <TextInput

                                    id="task\_image\_path"

                                    type="file"

                                    name="image"

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('image', e.target.files[0])}

                                />

                                <InputError message={errors.image} className="mt-2" />

                            </div>

                            <div className='mt-4'>

                                <InputLabel

                                    htmlFor="task\_name"

                                    value="Task Name"

                                />

                                <TextInput

                                    id="task\_name"

                                    type="text"

                                    name="name"

                                    value={data.name}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('name', e.target.value)}

                                />

                                <InputError message={errors.name} className="mt-2" />

                            </div>

                            <div className='mt-4'>

                                <InputLabel

                                    htmlFor="task\_description"

                                    value="Task Description"

                                />

                                <TextAreaInput

                                    id="task\_description"

                                    name="description"

                                    value={data.description}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('description', e.target.value)}

                                />

                                <InputError message={errors.description} className="mt-2" />

                            </div>

                            <div className='mt-4'>

                                <InputLabel

                                    htmlFor="task\_due\_date"

                                    value="Task Deadline"

                                />

                                <TextInput

                                    id="task\_due\_date"

                                    type="date"

                                    name="due\_date"

                                    value={data.due\_date}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('due\_date', e.target.value)}

                                />

                                <InputError message={errors.due\_date} className="mt-2" />

                            </div>

                            <div className='mt-4'>

                                <InputLabel

                                    htmlFor="task\_status"

                                    value="Task Status"

                                />

                                <SelectInput

                                    id="task\_status"

                                    name="status"

                                    value={data.status}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('status', e.target.value)}

                                >

                                    <option value="">Select Status</option>

                                    <option value="pending">Pending</option>

                                    <option value="in\_progress">In Progress</option>

                                    <option value="completed">Completed</option>

                                </SelectInput>

                                <InputError message={errors.status} className="mt-2" />

                            </div>

                            <div className='mt-4'>

                                <InputLabel

                                    htmlFor="task\_priority"

                                    value="Task Priority"

                                />

                                <SelectInput

                                    id="task\_priority"

                                    name="priority"

                                    value={data.priority}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('priority', e.target.value)}

                                >

                                    <option value="">Select Priority</option>

                                    <option value="low">Low</option>

                                    <option value="medium">Medium</option>

                                    <option value="high">High</option>

                                </SelectInput>

                                <InputError message={errors.priority} className="mt-2" />

                            </div>

                            <div className='mt-4'>

                                <InputLabel

                                    htmlFor="assigned\_user\_id"

                                    value="Assigned User"

                                />

                                <SelectInput

                                    id="assigned\_user\_id"

                                    name="assigned\_user\_id"

                                    value={data.assigned\_user\_id}

                                    className="mt-1 block w-full"

                                    onChange={*(e)* => setData('assigned\_user\_id', e.target.value)}

                                >

                                    <option value="">Select User</option>

                                    {users.data.map(user => (

                                        <option value={user.id} key={user.id}>{user.name}</option>

                                    ))}

                                </SelectInput>

                                <InputError message={errors.assigned\_user\_id} className="mt-2" />

                            </div>

                            <div className='mt-4 text-right'>

                                <Link

                                    href={route("task.index")}

                                    className='bg-gray-100 py-1 px-3 text-gray-800 rounded hover:bg-gray-300 mr-2'

                                >

                                    Cancel

                                </Link>

                                <button className='bg-emerald-500 text-white px-3 py-1 rounded hover:bg-emerald-600'>

                                    Submit

                                </button>

                            </div>

                        </form>

                    </div>

                </div>

            </div>

        </Authenticated>

    )

}

export default Edit

Next update the TaskController edit()method

public function edit(*Task $task*)

    {

        $projects = Project::query()->orderBy('name', 'asc')->get();

        $users = User::all();

        return inertia("Task/Edit", [

            "task" => new TaskResource($task),

            'projects' => ProjectResource::collection($projects),

            'users' => UserResource::collection($users),

        ]);

    }

In TaskResource, added project\_id, assigned\_user\_id and also set $wrap = false;

 public static $wrap = false;

'project\_id' => $this->project\_id,

'assigned\_user\_id' => $this->assigned\_user\_id,

Now Update the UpdateTaskRequest

public function authorize(): bool

    {

        return true;

    }

public function rules(): array

    {

        return [

            'name' => ['required', 'max:255'],

            'image' => ['nullable', 'image'],

            'description' => ['nullable','string'],

            'due\_date' => ['nullable', 'date'],

            'project\_id' => ['required', 'exists:projects,id'],

            'assigned\_user\_id' => ['required', 'exists:users,id'],

            'status' => ['required', Rule::in(['pending', 'in\_progress', 'completed'])],

            'priority' => ['required', Rule::in(['low', 'medium', 'high'])]

        ];

    }

Update the TasksTable.jsx to delete the the task

Replace the Action Columns with

<td className="px-3 py-2 text-nowrap">

    <Link href={route('task.edit', task.id)}

        className="font-medium text-blue-600 dark:text-blue-500 hover:underline mx-1"

    >

        Edit

    </Link>

    <button onClick={(e)=> deleteTask(task)}

        className="font-medium text-red-600 dark:text-red-500 hover:underline mx-1">

        Delete

    </button>

</td>

And add the deleteTask function

 const deleteTask = *(task)* => {

        if (!window.confirm('Are you sure want to delete the task?')) {

            return;

        }

        router.delete(route('task.destroy', task.id))

    };

To Display Notifications

In TaskController index()

return inertia("Task/Index", [

            "tasks" => TaskResource::collection($tasks),

            "queryParams" => request()->query() ?: null,

            'success' => session('success'),

        ]);

Task/Index.jsx

export default function Index*({ auth, tasks, queryParams = null, success })* {

}

<TasksTable tasks={tasks} queryParams={queryParams} success={success} />

Task/TasksTable.jsx

const TasksTable = *({ tasks, queryParams = null, hideProjectColumn = false, success })* => {

}

<>

            {success && (

                <div className="bg-emerald-500 py-2 px-4 text-white rounded mb-4">

                    {success}

                </div>

            )}

Now Update the Task/TasksTable to display the link on task name to show task details

<td className="px-3 py-2 hover:underline hover:text-white text-nowrap">

   <Link href={route('task.show', task.id)}>

         {task.name}

   </Link>

</td>

TaskController show() method to updated to show the details of Task

public function show(*Task $task*)

    {

        return inertia("Task/Show", [

            "task" => new TaskResource($task),

        ]);

    }

Task/Show.jsx is updated

import AuthenticatedLayout from '@/Layouts/AuthenticatedLayout'

import { TASK\_STATUS\_CLASS\_MAP, TASK\_STATUS\_TEXT\_MAP } from '@/constants'

import { Head } from '@inertiajs/react'

import React from 'react'

const Show = *({ auth, task})* => {

    return (

        <AuthenticatedLayout

            user={auth.user}

            header={

                <h2 className="font-semibold text-xl text-gray-800 dark:text-gray-200 leading-tight">

                    {`Task "${task.name}"`}

                </h2>}

        >

            <Head title={`Task "${task.name}"`} />

            <div className="py-6">

                <div className="max-w-9xl mx-auto sm:px-6 lg:px-8">

                    <div className="bg-white dark:bg-gray-800 overflow-hidden shadow-sm sm:rounded-lg">

                        <div>

                            <img src={task.image\_path} alt={task.name} className='w-full h-64 object-cover' />

                        </div>

                        <div className="p-6 text-gray-900 dark:text-gray-100">

                            <div className='grid gap-1 grid-cols-2 mt-2'>

                                <div>

                                    <div>

                                        <label className='font-bold text-lg'>Task ID</label>

                                        <p className='mt-1'>{task.id}</p>

                                    </div>

                                    <div className='mt-4'>

                                        <label className='font-bold text-lg'>Task Name</label>

                                        <p className='mt-1'>{task.name}</p>

                                    </div>

                                    <div className='mt-4'>

                                        <label className='font-bold text-lg'>Task Status</label>

                                        <p className='mt-1'>

                                            <span className={"px-2 py-1 rounded text-white " + TASK\_STATUS\_CLASS\_MAP[task.status]}>

                                                {TASK\_STATUS\_TEXT\_MAP[task.status]}

                                            </span>

                                        </p>

                                    </div>

                                    <div className='mt-4'>

                                        <label className='font-bold text-lg'>Created By</label>

                                        <p className='mt-1'>{task.createdBy.name}</p>

                                    </div>

                                </div>

                                <div>

                                    <div>

                                        <label className='font-bold text-lg'>Created Date</label>

                                        <p className='mt-1'>{task.created\_at}</p>

                                    </div>

                                    <div className='mt-4'>

                                        <label className='font-bold text-lg'>Due Date</label>

                                        <p className='mt-1'>{task.due\_date}</p>

                                    </div>

                                    <div className='mt-4'>

                                        <label className='font-bold text-lg'>Updated By</label>

                                        <p className='mt-1'>{task.updatedBy.name}</p>

                                    </div>

                                </div>

                            </div>

                            <div className='mt-4'>

                                <label className='font-bold text-lg'>Task Description</label>

                                <p className='mt-1'>{task.description}</p>

                            </div>

                        </div>

                    </div>

                </div>

            </div>

        </AuthenticatedLayout>

    )

}

export default Show

Now Update the Task/Show.jsx to display additional components

import { TASK\_PRIORITY\_CLASS\_MAP, TASK\_PRIORITY\_TEXT\_MAP, TASK\_STATUS\_CLASS\_MAP, TASK\_STATUS\_TEXT\_MAP } from '@/constants'

import { Head, Link } from '@inertiajs/react'

<div className='mt-4'>

    <label className='font-bold text-lg'>Task Priority</label>

    <p className='mt-1'>

        <span className={"px-2 py-1 rounded text-white " + TASK\_PRIORITY\_CLASS\_MAP[task.priority]}>

            {TASK\_PRIORITY\_TEXT\_MAP[task.priority]}

        </span>

    </p>

</div>

<div>

    <label className='font-bold text-lg'>Project Name</label>

    <p className='mt-1 hover:underline hover:text-white'>

        <Link href={route('project.show', task.project.id)}>

            {task.project.name}

        </Link>

    </p>

</div>

<div className='mt-4'>

    <label className='font-bold text-lg'>Assigned to User</label>

    <p className='mt-1'>{task.assignedUser.name}</p>

</div>

Update the route/web.php

Route::get('task/my-tasks', [TaskController::class, 'myTasks'])->name('task.myTasks');

In TaskController, create myTasks() method

public function myTasks()

    {

        $user = auth()->user();

        $query = Task::query()->with('project')->where('assigned\_user\_id', $user->id);

        $sortField = request("sort\_feild", "created\_at");

        $sortDirection = request('sort\_direction', "desc");

        if (request("project\_name")) {

            $query->whereHas('project', function (*$query*) {

                $query->where("name", "like", "%" . request("project\_name") . "%");

            });

        }

        if (request("task\_name")) {

            $query->where("name", "like", "%" . request("task\_name") . "%");

        }

        if (request("status")) {

            $query->where("status", request("status"));

        }

        // Retrieve tasks data and paginate the results

        $tasks = $query->orderBy($sortField, $sortDirection)->paginate(10);

        // Return the "Task/Index" view along with the tasks data collection

        return inertia("Task/Index", [

            "tasks" => TaskResource::collection($tasks),

            "queryParams" => request()->query() ?: null,

            'success' => session('success'),

        ]);

    }

In AuthenticatedLayout.jsx, add the link for myTasks

<NavLink

href={route('task.myTasks')}

active={route().current('task.myTasks')}

>

     My Tasks

</NavLink>

Update the Task/TasksTable.jsx

<th className="px-3 py-3">Assigned To</th>

<th className="px-3 py-3"></th>

<td className="px-3 py-2">{task.assignedUser.name}</td>

DashboardController Created by

php artisan make:controller DashboardController

Route/web.php updated

Route::get('/dashboard', [DashboardController::class, 'index'])->name('dashboard');

DashboardController, index() method updated to pass counts to dashboard component

public function index()

    {

        $user = auth()->user();

        $totalPendingTasks = Task::query()->where('status', 'pending')->count();

        $myPendingTasks = Task::query()->where('status', 'pending')->where('assigned\_user\_id', $user->id)->count();

        $totalInProgressTasks = Task::query()->where('status', 'in\_progress')->count();

        $myInProgressTasks = Task::query()->where('status', 'in\_progress')->where('assigned\_user\_id', $user->id)->count();

        $totalCompletedTasks = Task::query()->where('status', 'completed')->count();

        $myCompletedTasks = Task::query()->where('status', 'completed')->where('assigned\_user\_id', $user->id)->count();

        return Inertia('Dashboard', compact(

            'totalInProgressTasks',

            'myInProgressTasks',

            'totalCompletedTasks',

            'myCompletedTasks',

            'totalPendingTasks',

            'myPendingTasks'

        ));

    }

Now Update the Dashboard Component to display the Task Stat widgets

import AuthenticatedLayout from '@/Layouts/AuthenticatedLayout';

import { Head } from '@inertiajs/react';

export default function Dashboard*({ auth, totalPendingTasks, myPendingTasks, totalCompletedTasks, myCompletedTasks, totalInProgressTasks, myInProgressTasks })* {

    return (

        <AuthenticatedLayout

            user={auth.user}

            header={<h2 className="font-semibold text-xl text-gray-800 dark:text-gray-200 leading-tight">Dashboard</h2>}

        >

            <Head title="Dashboard" />

            <div className="py-12">

                <div className="max-w-9xl mx-auto sm:px-6 lg:px-8 grid grid-cols-3 gap-5">

                    <div className="bg-white dark:bg-gray-800 overflow-hidden shadow-sm sm:rounded-lg">

                        <div className="p-6 text-gray-900 dark:text-gray-100">

                            <h3 className='text-amber-500 text-xl font-semibold'>Pending Tasks</h3>

                            <p className='text-xl mt-4'>

                                <span className='mr-2'>{myPendingTasks}</span>/

                                <span className='ml-2'>{totalPendingTasks}</span>

                            </p>

                        </div>

                    </div>

                    <div className="bg-white dark:bg-gray-800 overflow-hidden shadow-sm sm:rounded-lg">

                        <div className="p-6 text-gray-900 dark:text-gray-100">

                            <h3 className='text-blue-500 text-xl font-semibold'>In Progrss Tasks</h3>

                            <p className='text-xl mt-4'>

                                <span className='mr-2'>{myInProgressTasks}</span>/

                                <span className='ml-2'>{totalInProgressTasks}</span>

                            </p>

                        </div>

                    </div>

                    <div className="bg-white dark:bg-gray-800 overflow-hidden shadow-sm sm:rounded-lg">

                        <div className="p-6 text-gray-900 dark:text-gray-100">

                            <h3 className='text-green-600 font-semibold'>Completed Tasks</h3>

                            <p className='text-xl mt-4'>

                                <span className='mr-2'>{myCompletedTasks}</span>/

                                <span className='ml-2'>{totalCompletedTasks}</span>

                            </p>

                        </div>

                    </div>

                </div>

            </div>

        </AuthenticatedLayout>

    );

}

Dashboard.jsx updated to display the active tasks

import { TASK\_STATUS\_CLASS\_MAP, TASK\_STATUS\_TEXT\_MAP } from '@/constants';

import { Head, Link } from '@inertiajs/react';

export default function Dashboard({ auth, totalPendingTasks, myPendingTasks, totalCompletedTasks, myCompletedTasks, totalInProgressTasks, myInProgressTasks, activeTasks }) {}

<div className="max-w-9xl mx-auto sm:px-6 lg:px-8 mt-4">

    <div className="bg-white dark:bg-gray-800 overflow-hidden shadow-sm sm:rounded-lg">

        <div className="p-6 text-gray-900 dark:text-gray-100">

            <h3 className='text-gray-500 dark:text-gray-200 text-xl font-semibold'>My Active Tasks</h3>

            <table className="w-full text-sm text-left rtl:text-right text-gray-500 dark:text-gray-400 mt-4">

                <thead className="text-xs text-gray-700 uppercase bg-gray-50 dark:bg-gray-700 dark:text-gray-400 border-b-2 border-gray-500">

                    <tr>

                        <th className="px-3 py-3">ID</th>

                        <th className="px-3 py-3">Project Name</th>

                        <th className="px-3 py-3">Task Name</th>

                        <th className="px-3 py-3">Status</th>

                        <th className="px-3 py-3">DeadLine</th>

                    </tr>

                </thead>

                <tbody>

                    {activeTasks.data.map((task) => (

                        <tr key={task.id}>

                            <td className="px-3 py-2">{task.id}</td>

                            <td className="px-3 py-2 hover:underline hover:text-white text-nowrap">

                                <Link href={route('project.show', task.project.id)}>

                                    {task.project.name}

                                </Link>

                            </td>

                            <td className="px-3 py-2 hover:underline hover:text-white text-nowrap">

                                <Link href={route('task.show', task.id)}>

                                    {task.name}

                                </Link>

                            </td>

                            <td className="px-3 py-2">

                                <span className={"px-2 py-1 rounded text-white " + TASK\_STATUS\_CLASS\_MAP[task.status]}>

                                    {TASK\_STATUS\_TEXT\_MAP[task.status]}

                                </span>

                            </td>

                            <td className="px-3 py-2">{task.due\_date}</td>

                        </tr>

                    ))}

                </tbody>

            </table>

        </div>

    </div>

</div>

DashboardController, index() method updated to pass the activeTasks

$activeTasks = Task::query()->whereNot('status', 'completed')->where('assigned\_user\_id', $user->id)->orderBy('due\_date','desc')->limit(10)->get();

$activeTasks = TaskResource::collection($activeTasks);

return Inertia('Dashboard', compact(

            'totalInProgressTasks',

            'myInProgressTasks',

            'totalCompletedTasks',

            'myCompletedTasks',

            'totalPendingTasks',

            'myPendingTasks',

            'activeTasks'

        ));

Task/Show.jsx

<AuthenticatedLayout

            user={auth.user}

            header={

                <div className='flex items-center justify-between'>

                    <h2 className="font-semibold text-xl text-gray-800 dark:text-gray-200 leading-tight">

                        {`Task "${task.name}"`}

                    </h2>

                    <Link className="bg-emerald-500 text-white px-3 py-1 rounded hover:bg-emerald-600"

                        href={route('task.edit', task.id)}

                    >

                        Edit

                    </Link>

                </div>

            }

        >

Project/Show.jsx

<AuthenticatedLayout

            user={auth.user}

            header={

                <div className='flex items-center justify-between'>

                    <h2 className="font-semibold text-xl text-gray-800 dark:text-gray-200 leading-tight">

                        {`Project "${project.name}"`}

                    </h2>

                    <Link className="bg-emerald-500 text-white px-3 py-1 rounded hover:bg-emerald-600"

                        href={route('project.edit', project.id)}

                    >

                        Edit

                    </Link>

                </div>

            }

        >