**­­­­Laravel installation steps**

1. Download and Install laragon
2. Run port to run laragon-> http://127.0.0.1
3. Download and Install composer
4. Then run this command in cmd to install composer-> composer global require laravel/installer
5. Then run this command in cmd to install project-> laravel new firstproj
6. run port <http://127.0.0.1:8000>
7. <http://localhost/phpmyadmin>.

**Starting Laravel Project**

1. Command to connect with database run this-> mysql -u root
2. Create database by running this comd->

CREATE DATABASE youtube; or create database youtube;

use youtube;

show tables;

select \* from ideas;

exit

clear

1. To migrate tables run this command->

php artisan migrate

php artisan migrate:rollback

php artisan migrate:fresh

php artisan migrate:refresh --step=2

1. php artisan make:migration create\_ideas\_table

php artisan make:model Idea

php artisan make:controller IdeaController

**This command makes model, migration and controller**

1. php artisan make:model Comment -m -c

**resource Controller**

1. php artisan make:controller UserController –r

**Laravel Build in Authentication system using Breeze also use (Jatstream)**

 Install Laravel Breeze via Composer:

composer require laravel/breeze --dev

 Install Breeze scaffolding:

php artisan breeze:install

 Install the front-end dependencies:

npm install

 Compile the assets:

npm run dev

 Run the migrations to create the necessary database tables:

php artisan migrate

Now, you can access the authentication system at /login, /register, etc.

To install **Spatie's Role and Permission** package in Laravel, follow the steps below:

**Step 1: Install the package via Composer**

Run the following command to install the Spatie Role and Permission package:

composer require spatie/laravel-permission

**Step 2: Publish the configuration file**

After installation, you need to publish the package's configuration file by running the following Artisan command:

php artisan vendor:publish --provider="Spatie\Permission\PermissionServiceProvider"

This will create a config/permission.php file where you can customize the package's settings.

**Step 3: Run the migrations**

The package requires a set of database tables to store roles and permissions. Run the migrations to create these tables:

php artisan migrate

ADD In User Model

use Spatie\Permission\Traits\HasRoles;

class User extends Authenticatable

{

use HasRoles

}

**After installation of Spatie**

class AdminSeeder extends Seeder

{

    /\*\*

     \* Run the database seeds.

     \*/

    public function run(): void

    {

        $user = User::create([

            'name' => 'Adminn',

            'email' => 'admin@gmail.com',

            'email\_verified\_at' => now(),

            'password' => bcrypt('password'),  // Make sure to hash the password

        ]);

        $user->assignRole(['writer','admin']);

    }

}

class RoleSeeder extends Seeder

{

    /\*\*

     \* Run the database seeds.

     \*/

    public function run(): void

    {

        Role::create(['name' => 'admin']);

        Role::create(['name' => 'user']);

        Role::create(['name' => 'writer']);

    }

}

class DatabaseSeeder extends Seeder

{

    /\*\*

     \* Seed the application's database.

     \*/

    public function run(): void

    {

            $this->call(RoleSeeder::class);

            $this->call(AdminSeeder::class);

    }

}

After creating AdminSeeder and RoleSeeder run this command

php artisan migrate:fresh –seed

After that we create

PermissionController

RoleController

### ****Check for Roles and Permissions in Blade****

In your Blade templates, you can use the @role, @can, and @hasrole directives provided by the Spatie package to define what content certain users are allowed to see or access.

Here are some examples:

#### Using @role directive:

@role('admin')

<p>This is visible to users with the admin role.</p>

@endrole

#### Using @can directive:

@can('edit posts')

<p>This is visible to users who have the 'edit posts' permission.</p>

@endcan

#### Using @hasrole directive:

@hasrole('writer')

<p>This is visible to users with the writer role.</p>

@endhasrole

#### Combining roles and permissions:

@role('admin')

<p>Only visible to admins.</p>

@elsecan('edit posts')

<p>Visible to users who can edit posts.</p>

@endrole

You can also check multiple roles and permissions with @hasanyrole or @hasallroles:

@hasanyrole('admin|writer')

<p>Visible to users with either the admin or writer role.</p>

@endhasanyrole

@hasallroles('admin|writer')

<p>Visible only to users who have both admin and writer roles.</p>

@endhasallroles

@role('admin')

    <p>You are an editor.</p>

    @can('delete post')

        <p>You can also publish posts.</p>

    @endcan

@endrole

@role('admin')

    <p>Admin access granted.</p>

        @elsecan('delete post')

            <p>You are not an admin, but you can edit posts.</p>

        @else

    <p>You do not have the necessary permissions to view this content.</p>

@endrole

@role('admin')

    <p>This content is only for users with the "admin" role.</p>

@endrole

**Seeder Commands**

### 1. ****Creating a Factory****

Factories are used to define how fake data should be generated for your models.

#### Step 1: Create a Factory

Run the following command to create a new factory:

php artisan make:factory UserFactory --model=User

This will create a factory for the User model. The factory file is stored in the database/factories directory.

#### Step 2: Define the Factory

Open the factory file, database/factories/UserFactory.php, and define the model's default state. Laravel uses the Faker library to generate fake data.

Example of UserFactory:

use App\Models\User;

use Illuminate\Database\Eloquent\Factories\Factory;

use Illuminate\Support\Str;

class UserFactory extends Factory

{

protected $model = User::class;

public function definition()

{

return [

'name' => $this->faker->name,

'email' => $this->faker->unique()->safeEmail,

'email\_verified\_at' => now(),

'password' => bcrypt('password'), // Default password

'remember\_token' => Str::random(10),

];

}

}

#### Step 3: Use the Factory

Use the factory in your seeders to create sample data.

Example of creating 10 users:

public function run(): void

{

Item::factory()->count(10)->create();

}

### 2. ****Creating a Seeder****

Seeders are used to populate your database with initial data.

#### Step 1: Create a Seeder

Run the following command to create a seeder:

php artisan make:seeder UserSeeder

This will create a seeder file in database/seeders.

#### Step 2: Define the Seeder

Open the UserSeeder.php file and define how you want to seed the database. You can use the factory here to generate fake data.

Example of UserSeeder:

use App\Models\User;

use Illuminate\Database\Seeder;

class UserSeeder extends Seeder

{

public function run()

{

User::factory()->count(10)->create();

}

}

#### Step 3: Run the Seeder

After defining the seeder, you can run it with the following command:

php artisan db:seed --class=UserSeeder

(This command run only the gien User Seeder)

Alternatively, to seed all defined seeders at once, register your seeders in DatabaseSeeder.php:

public function run()

{

$this->call(UserSeeder::class);

$this->call(ItemSeeder::class);

}

Then run:

php artisan db:seed

(this command run all the seeder defined in DatabaseSeedr)

### 3. ****Bonus: Running Migrations and Seeders Together****

You can refresh your migrations and seed data in one go:

php artisan migrate:fresh –seed

(This command run the all seeder and migrations simultaneously)

This will drop all tables, run all migrations, and then seed the database.

### Summary

* **Factories** define how to generate fake data for models.
* **Seeders** populate the database with data, often using factories.

These tools are crucial for testing and populating databases in Laravel.

**Git command**

**steps for first time configure**

1. git init

2. git add .

3. git commit -m "first commit"

4. git branch -M main

5. git remote add origin https://github.com/Muhammad-Abbas-55/First-Laravel-Project.git

6. git push -u origin main

**second time and onward**

1. git add .

2. git commit -m "asd"

3.git push

**Install Node.js (if you haven't already):**

a. React uses Node.js and npm (Node Package Manager) to manage dependencies.

b. **Download and install Node.js** from **nodejs.org**. This will also **install npm** automatically.

c. Open your terminal/command prompt and navigate to the directory where you want to create the project.

**React Commands**

1. **Use npx to create a new React app**:**:** npx create-react-app my-app

2. **Navigate into the project directory**: cd my-app

3. **Start the development server:** npm start

**Cloning an Existing React Project from GitHub:**

1. **Clone the repository**: Run the following command to clone a project:

git clone <https://github.com/username/repository-name.git>

2. cd repository-name

3. npm install

4. npm start

**Extensions of VS codes**

1. Laravel goto view (it opens the page by clicking the name)
2. Tabnine (Generate code using AI)
3. Prettier - Code formatter
4. Simple React Snippets (auto suggest code for react)
5. PHP Constructor (it able to write multiple cursors)
6. Path Autocomplete (give auto path suggestion)
7. Live Server (Five Server)
8. Live Server
9. Laravel Extra Intellisense (better intellisense for laravel projects.)
10. Laravel Extension Pack (A collection of extensions useful for Laravel development)
11. Laravel Blade Snippets (Laravel blade snippets and syntax highlight support ie: ! it write html code)
12. Laravel Blade formatter
13. ES7+ React/Redux/React-Native snippets (Extensions for React, React-Native and Redux in JS/TS with ES7+ syntax. Customizable. Built-in integration with prettier.)