Laravel

Intermediate

Caspian Consultancy & Services

Agenda

What we going to learn

- Roles & Permissions
  - spatie/laravel-permission
  - Create Blog with ACL
  - Using Seeders
- Export/Import Excel or CSV files
- Create PDF files
- Create Chart

Laravel Role & Permission

Introduction

We will implement a Laravel 8 spatie user roles and permissions.

Spatie role permission composer package provide way to create ACL in Laravel 8. They provide how to assign role to user, how to assign permission to user and how to assign permission assign to roles.

Create CMS

Access Control Level (ACL) Management

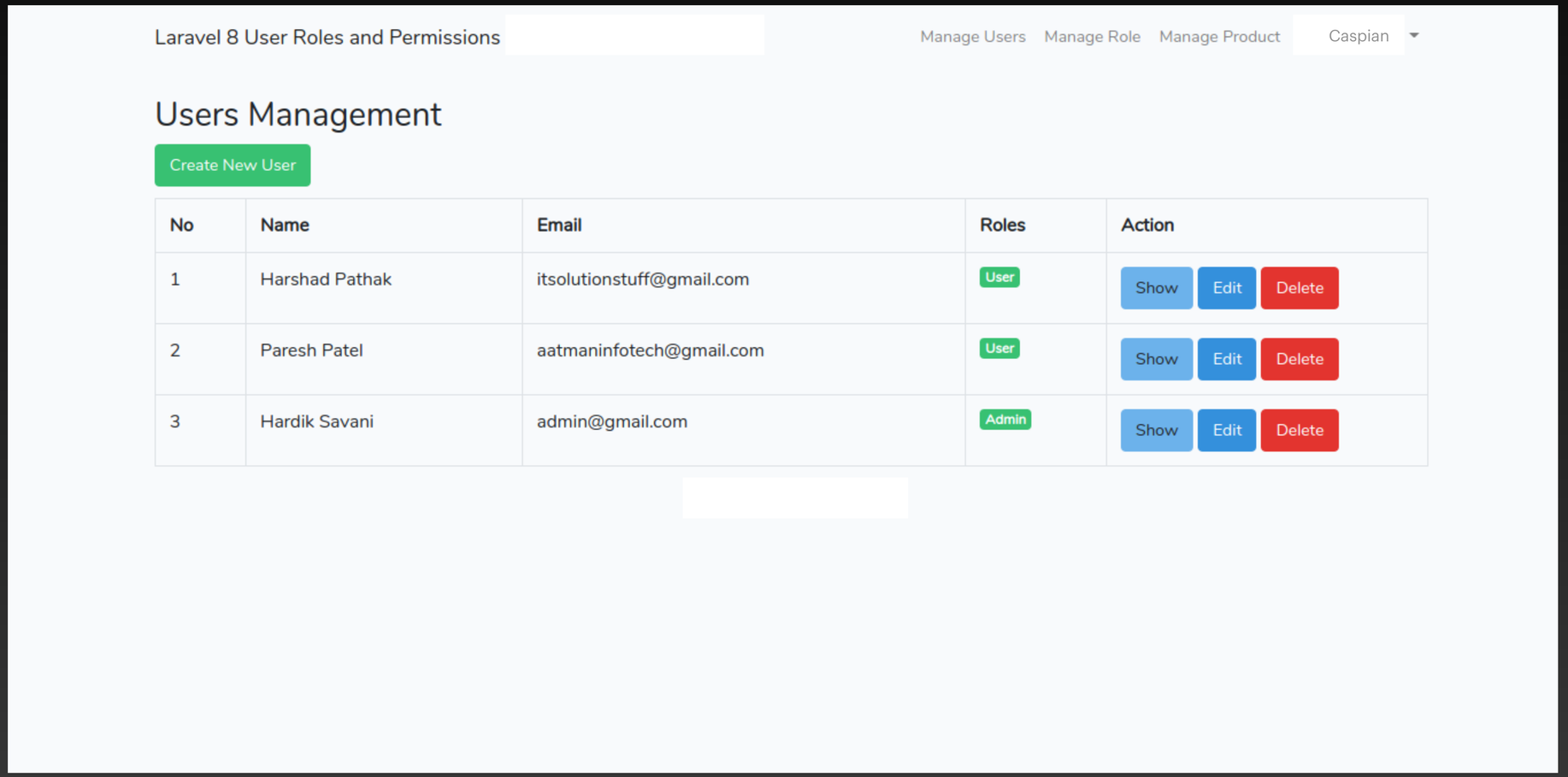
In this examples we will created three modules as listed below:

- User Management
- Role Management
- Product Management

After register user, you don't have any roles, so you can edit your details and assign admin role to you from User Management. After that you can create your own role with permission like role-list, role-create, role-edit, role-delete, product-list, product-create, product-edit, product-delete. you can check with assign new user and check that.

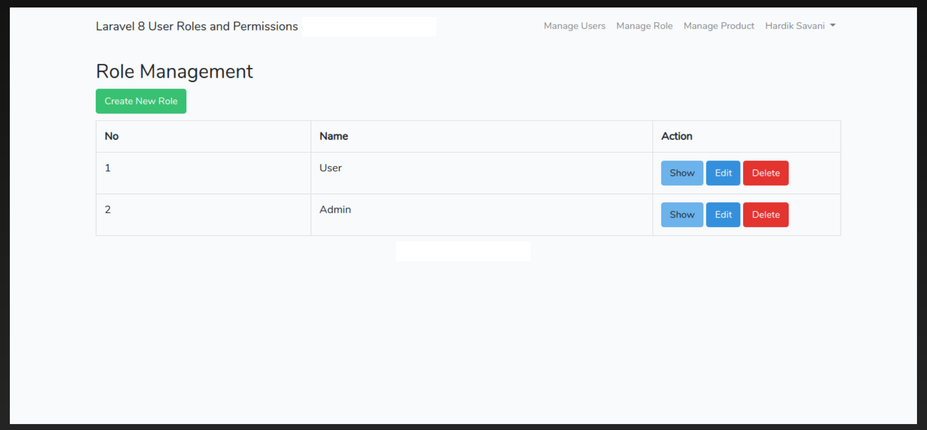
CMS

Access Control Level (ACL) Management

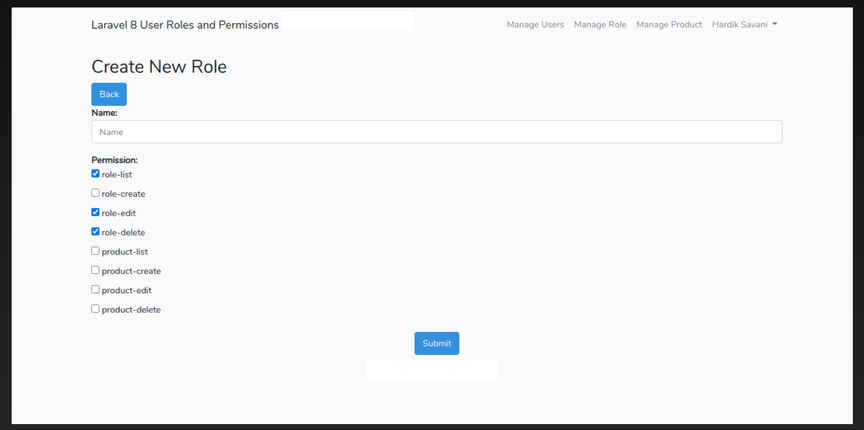


CMS
ACL Features

List Role

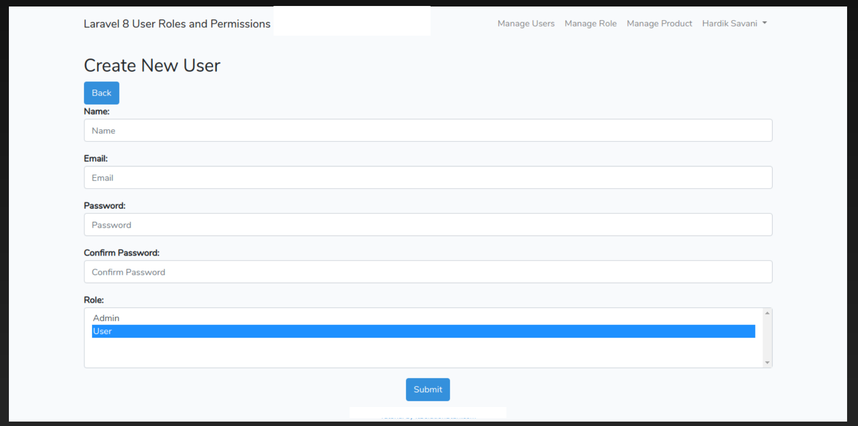


Create Role

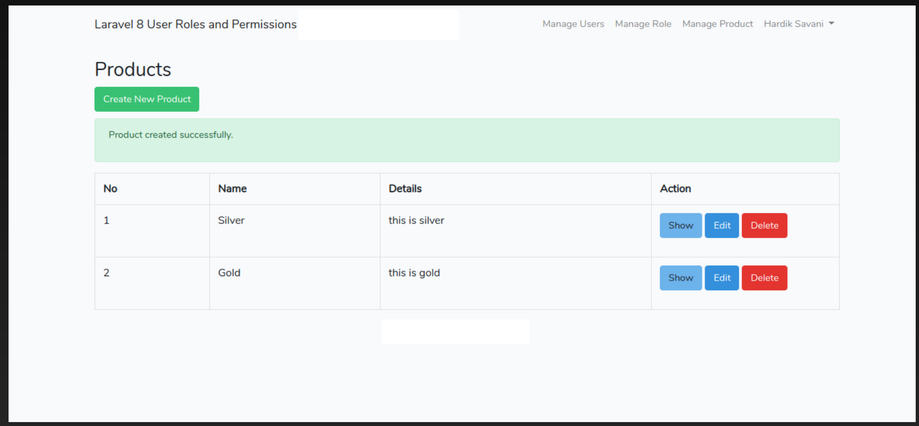


CMS
ACL Features

Create User



List Products



Create a Blog with ACL

Step 1
Laravel Project Installation

- Terminal: composer create-project --prefer-dist laravel/laravel blog6 or
- Terminal: Laravel new blog6 or
- Terminal: curl -s "https://laravel.build/blog6" | bash

Step 2
Install Composer Packages

- Terminal: composer require spatie/laravel-permission
- Terminal: composer require laravelcollective/html
- Next, Now open config/app.php file and add service provider and aliases.
- In the config/app.php, add Spatie

```
'providers' => [
                ....
                Spatie\Permission\PermissionServiceProvider::class,
],
```

- Terminal: php artisan vendor:publish --provider="Spatie\Permission\PermissionServiceProvider"
- After that, Terminal: php artisan migrate
  (You will encounter an Error. Why?)

Step 3
Create Product Migration

- Terminal: php artisan make:migration create_products_table
- Terminal: php artisan migrate

```php
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateProductsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('products', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->text('detail');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('products');
    }
}
```

Step 4
Create Models

- App/Models/User.php

```php
<?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;
use Spatie\Permission\Traits\HasRoles;

class User extends Authenticatable
{
    use HasFactory, Notifiable, HasRoles;

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name',
        'email',
        'password',
    ];

    /**
     * The attributes that should be hidden for arrays.
     *
     * @var array
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];

    /**
     * The attributes that should be cast to native types.
     *
     * @var array
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
    ];
}
```

- App/Models/Product.php

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Product extends Model
{
    use HasFactory;

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name', 'detail'
    ];
}
```

Step 5
Add Middleware

- Spatie package provide it's in-built middleware that way we can use it simply and that is display as below:
  - Role
  - Permission
- So, we have to add middleware in Kernel.php file this way :

```
....
protected $routeMiddleware = [
    ....
    'role' => \Spatie\Permission\Middlewares\RoleMiddleware::class,
    'permission' => \Spatie\Permission\Middlewares\PermissionMiddleware::class,
    'role_or_permission' => \Spatie\Permission\Middlewares\RoleOrPermissionMiddleware::class,
]
....
```

Step 6
Create Authentication

- Terminal: composer require laravel/ui
- Then, Terminal: php artisan ui bootstrap --auth
- Install npm using below:
  - Terminal: npm install
  - Terminal: npm run dev

Note: Now you need to run npm command, otherwise you can not see better layout of login and register page using css.

Step 7
Create Routes

- We require to add several routes for users module, products module and roles module. We will also use middleware with permission for roles and products route

```php
<?php

use Illuminate\Support\Facades\Route;

use App\Http\Controllers\HomeController;
use App\Http\Controllers\RoleController;
use App\Http\Controllers\UserController;
use App\Http\Controllers\ProductController;

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

Route::get('/', function () {
    return view('welcome');
});

Auth::routes();

Route::get('/home', [HomeController::class, 'index'])->name('home');

Route::group(['middleware' => ['auth']], function() {
    Route::resource('roles', RoleController::class);
    Route::resource('users', UserController::class);
    Route::resource('products', ProductController::class);
});
```

Step 8
Add Controllers

- We need 3 Controllers:
  - UserController
  - ProductController
  - RoleController
- How to Create Controller: Terminal: php artisan make:controller -r or php artisan make:model ModelName -mcr (Model, Migration & Resource)

UserController

Step 8

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use App\Models\User;
use Spatie\Permission\Models\Role;
use DB;
use Hash;
use Illuminate\Support\Arr;

class UserController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index(Request $request)
    {
        $data = User::orderBy('id','DESC')->paginate(5);
        return view('users.index',compact('data'))
            ->with('i', ($request->input('page', 1) - 1) * 5);
    }

    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create()
    {
        $roles = Role::pluck('name','name')->all();
        return view('users.create',compact('roles'));
    }
```

UserController

Step 8

```php
/**
 * Show the form for creating a new resource.
 *
 * @return \Illuminate\Http\Response
 */
public function create()
{
    $roles = Role::pluck('name','name')->all();
    return view('users.create',compact('roles'));
}

/**
 * Store a newly created resource in storage.
 *
 * @param  \Illuminate\Http\Request  $request
 * @return \Illuminate\Http\Response
 */
public function store(Request $request)
{
    $this->validate($request, [
        'name' => 'required',
        'email' => 'required|email|unique:users,email',
        'password' => 'required|same:confirm-password',
        'roles' => 'required'
    ]);

    $input = $request->all();
    $input['password'] = Hash::make($input['password']);

    $user = User::create($input);
    $user->assignRole($request->input('roles'));

    return redirect()->route('users.index')
                    ->with('success','User created successfully');
}
```

UserController

Step 8

```php
/**
 * Display the specified resource.
 *
 * @param  int  $id
 * @return \Illuminate\Http\Response
 */
public function show($id)
{
    $user = User::find($id);
    return view('users.show',compact('user'));
}

/**
 * Show the form for editing the specified resource.
 *
 * @param  int  $id
 * @return \Illuminate\Http\Response
 */
public function edit($id)
{
    $user = User::find($id);
    $roles = Role::pluck('name','name')->all();
    $userRole = $user->roles->pluck('name','name')->all();

    return view('users.edit',compact('user','roles','userRole'));
}
```

UserController

Step 8

```php
/**
 * Update the specified resource in storage.
 *
 * @param  \Illuminate\Http\Request  $request
 * @param  int  $id
 * @return \Illuminate\Http\Response
 */
public function update(Request $request, $id)
{
    $this->validate($request, [
        'name' => 'required',
        'email' => 'required|email|unique:users,email,'.$id,
        'password' => 'same:confirm-password',
        'roles' => 'required'
    ]);

    $input = $request->all();
    if(!empty($input['password'])){
        $input['password'] = Hash::make($input['password']);
    }else{
        $input = Arr::except($input,array('password'));
    }

    $user = User::find($id);
    $user->update($input);
    DB::table('model_has_roles')->where('model_id',$id)->delete();

    $user->assignRole($request->input('roles'));

    return redirect()->route('users.index')
                    ->with('success','User updated successfully');
}
```

UserController

Step 8

```
    /**
     * Remove the specified resource from storage.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function destroy($id)
    {
        User::find($id)->delete();
        return redirect()->route('users.index')
                        ->with('success','User deleted successfully');
    }
}
```

ProductController

Step 8

```php
class ProductController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    function __construct()
    {
         $this->middleware('permission:product-list|product-create|product-edit|product-delete', ['only'
         $this->middleware('permission:product-create', ['only' => ['create','store']]);
         $this->middleware('permission:product-edit', ['only' => ['edit','update']]);
         $this->middleware('permission:product-delete', ['only' => ['destroy']]);
    }
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        $products = Product::latest()->paginate(5);
        return view('products.index',compact('products'))
            ->with('i', (request()->input('page', 1) - 1) * 5);
    }
```

ProductController

Step 8

```php
/**
 * Show the form for creating a new resource.
 *
 * @return \Illuminate\Http\Response
 */
public function create()
{
    return view('products.create');
}

/**
 * Store a newly created resource in storage.
 *
 * @param  \Illuminate\Http\Request  $request
 * @return \Illuminate\Http\Response
 */
public function store(Request $request)
{
    request()->validate([
        'name' => 'required',
        'detail' => 'required',
    ]);

    Product::create($request->all());

    return redirect()->route('products.index')
                    ->with('success','Product created successfully.');
}
```

ProductController

Step 8

```php
/**
 * Display the specified resource.
 *
 * @param  \App\Product  $product
 * @return \Illuminate\Http\Response
 */
public function show(Product $product)
{
    return view('products.show',compact('product'));
}

/**
 * Show the form for editing the specified resource.
 *
 * @param  \App\Product  $product
 * @return \Illuminate\Http\Response
 */
public function edit(Product $product)
{
    return view('products.edit',compact('product'));
}
```

ProductController

Step 8

```
    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \App\Product  $product
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, Product $product)
    {
         request()->validate([
            'name' => 'required',
            'detail' => 'required',
        ]);

        $product->update($request->all());

        return redirect()->route('products.index')
                        ->with('success','Product updated successfully');
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  \App\Product  $product
     * @return \Illuminate\Http\Response
     */
    public function destroy(Product $product)
    {
        $product->delete();

        return redirect()->route('products.index')
                        ->with('success','Product deleted successfully');
    }
}
```

RoleController

Step 8

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use Spatie\Permission\Models\Role;
use Spatie\Permission\Models\Permission;
use DB;

class RoleController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    function __construct()
    {
         $this->middleware('permission:role-list|role-create|role-edit|role-delete', ['only' => ['index',
         $this->middleware('permission:role-create', ['only' => ['create','store']]);
         $this->middleware('permission:role-edit', ['only' => ['edit','update']]);
         $this->middleware('permission:role-delete', ['only' => ['destroy']]);
    }
```

RoleController

Step 8

```php
/**
 * Display a listing of the resource.
 *
 * @return \Illuminate\Http\Response
 */
public function index(Request $request)
{
    $roles = Role::orderBy('id','DESC')->paginate(5);
    return view('roles.index',compact('roles'))
        ->with('i', ($request->input('page', 1) - 1) * 5);
}

/**
 * Show the form for creating a new resource.
 *
 * @return \Illuminate\Http\Response
 */
public function create()
{
    $permission = Permission::get();
    return view('roles.create',compact('permission'));
}
```

RoleController

Step 8

```php
/**
 * Store a newly created resource in storage.
 *
 * @param  \Illuminate\Http\Request  $request
 * @return \Illuminate\Http\Response
 */
public function store(Request $request)
{
    $this->validate($request, [
        'name' => 'required|unique:roles,name',
        'permission' => 'required',
    ]);

    $role = Role::create(['name' => $request->input('name')]);
    $role->syncPermissions($request->input('permission'));

    return redirect()->route('roles.index')
                    ->with('success','Role created successfully');
}
/**
 * Display the specified resource.
 *
 * @param  int  $id
 * @return \Illuminate\Http\Response
 */
public function show($id)
{
    $role = Role::find($id);
    $rolePermissions = Permission::join("role_has_permissions","role_has_permissions.permission_id","
        ->where("role_has_permissions.role_id",$id)
        ->get();

    return view('roles.show',compact('role','rolePermissions'));
}
```

RoleController

Step 8

```php
/**
 * Show the form for editing the specified resource.
 *
 * @param  int  $id
 * @return \Illuminate\Http\Response
 */
public function edit($id)
{
    $role = Role::find($id);
    $permission = Permission::get();
    $rolePermissions = DB::table("role_has_permissions")->where("role_has_permissions.role_id",$id)
        ->pluck('role_has_permissions.permission_id','role_has_permissions.permission_id')
        ->all();

    return view('roles.edit',compact('role','permission','rolePermissions'));
}

/**
 * Update the specified resource in storage.
 *
 * @param  \Illuminate\Http\Request  $request
 * @param  int  $id
 * @return \Illuminate\Http\Response
 */
public function update(Request $request, $id)
{
    $this->validate($request, [
        'name' => 'required',
        'permission' => 'required',
    ]);

    $role = Role::find($id);
    $role->name = $request->input('name');
    $role->save();

    $role->syncPermissions($request->input('permission'));

    return redirect()->route('roles.index')
                    ->with('success','Role updated successfully');
}
```

RoleController

Step 8

```
    /**
     * Remove the specified resource from storage.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function destroy($id)
    {
        DB::table("roles")->where('id',$id)->delete();
        return redirect()->route('roles.index')
                        ->with('success','Role deleted successfully');
    }
}
```

Add Blade Layout Files

Step 9

We need 4 groups of Blade Layout files:

- **Theme Layout**
  - app.blade.php
- **User Module**
  - index.blade.php
  - create.blade.php
  - edit.blade.php
  - show.blade.php
- **Role Module**
  - index.blade.php
  - create.blade.php
  - edit.blade.php
  - show.blade.php
- **Product Module**
  - index.blade.php
  - create.blade.php
  - edit.blade.php
  - show.blade.php

Create Seeders

Step 10

- In this step we will create seeder for permissions.
- We create using seeder as listed below, but if you can add more permission as you want for:
  - role-list
  - role-create
  - role-edit
  - role-delete
  - product-list
  - product-create
  - product-edit
  - product-delete

How to Create Seeder for Permission

Step 10

- Terminal: php artisan make:seeder PermissionTableSeeder
- Add the code in **database/seeds/PermissionTableSeeder.php**
- Then Terminal: php artisan db:seed --class=PermissionTableSeeder

```php
<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;
use Spatie\Permission\Models\Permission;

class PermissionTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $permissions = [
           'role-list',
           'role-create',
           'role-edit',
           'role-delete',
           'product-list',
           'product-create',
           'product-edit',
           'product-delete'
        ];

        foreach ($permissions as $permission) {
             Permission::create(['name' => $permission]);
        }
    }
}
```

How to Create Seeder for Admin User

Step 10

- Terminal: php artisan make:seeder CreateAdminUserSeeder
- Add the code in **database/seeds/CreateAdminUserSeeder.php**
- Then Terminal: php artisan db:seed --class=CreateAdminUserSeeder

```php
<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;
use App\Models\User;
use Spatie\Permission\Models\Role;
use Spatie\Permission\Models\Permission;

class CreateAdminUserSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $user = User::create([
            'name' => 'Hardik Savani',
            'email' => 'admin@gmail.com',
            'password' => bcrypt('123456')
        ]);

        $role = Role::create(['name' => 'Admin']);

        $permissions = Permission::pluck('id','id')->all();

        $role->syncPermissions($permissions);

        $user->assignRole([$role->id]);
    }
}
```

Complete
Let’s run

- Terminal: php artisan serve
- Access on your Browser: http://localhost:8000/

Import/Export Excel & CSV

Import Export Excel and CSV

Introduction

- We will simple create import data to csv, xls file and also we can import data to database using csv file in Laravel 8 application.
- We will use maatwebsite/excel composer package for import and export task. maatwebsite/excel provide easy way to import and export using database model.

Step 1
Create Laravel Installation

- Terminal: composer create-project --prefer-dist laravel/laravel exceldemo or
- Terminal: Laravel new exceldemo or
- Terminal: curl -s “https://laravel.build/exceldemo" | bash

Step 2
Install maatwebsite/excel Package

- Terminal: composer require maatwebsite/excel
- Open **config/app.php** file and add service provider and alias.

```
'providers' => [
                ....
                Maatwebsite\Excel\ExcelServiceProvider::class,
 ],
'aliases' => [
                ....
                'Excel' => Maatwebsite\Excel\Facades\Excel::class,
],
```

Step 3
Create Dummy Records

- In this step, we have to require "users" table with some dummy records, so we can simply import and export. So first you have to run default migration that provided by Laravel using following command:
- Terminal: php artisan migrate
- Use Tinker to create some dummy users:
- Terminal: php artisan tinker
- User::factory()->count(20)->create()

Step 4
Add Routes

- Open your "routes/web.php" file and add following route:

```php
<?php

use Illuminate\Support\Facades\Route;

use App\Http\Controllers\MyController;

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

Route::get('importExportView', [MyController::class, 'importExportView']);
Route::get('export', [MyController::class, 'export'])->name('export');
Route::post('import', [MyController::class, 'import'])->name('import');
```

Step 5
Create Import Class

- In maatwebsite 3 version provide way to built import class and we have to use in a controller.
- Terminal: php artisan make:import UsersImport --model=User

Step 6
Create Import Class

- In **app/Imports/UsersImport.php**, add these codes:

```php
<?php

namespace App\Imports;

use App\Models\User;
use Maatwebsite\Excel\Concerns\ToModel;
use Maatwebsite\Excel\Concerns\WithHeadingRow;

class UsersImport implements ToModel, WithHeadingRow
{
    /**
    * @param array $row
    *
    * @return \Illuminate\Database\Eloquent\Model|null
    */
    public function model(array $row)
    {
        return new User([
            'name'     => $row['name'],
            'email'    => $row['email'],
            'password' => \Hash::make($row['password']),
        ]);
    }
}
```

Step 7
Create Export Class

- Terminal: php artisan make:export UsersExport --model=User

```php
<?php

namespace App\Exports;

use App\Models\User;
use Maatwebsite\Excel\Concerns\FromCollection;

class UsersExport implements FromCollection
{
    /**
    * @return \Illuminate\Support\Collection
    */
    public function collection()
    {
        return User::all();
    }
}
```

Step 8
Create Controller

- Create a new controller as MyController in **app/Http/Controllers/MyController.php**. This controller will manage all importExportView, export and import request and return response.
- Terminal: php artisan make:controller MyController

Step 8

Create Controller Functions

- In **app/Http/Controllers/MyController.php**, add these codes:

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Exports\UsersExport;
use App\Imports\UsersImport;
use Maatwebsite\Excel\Facades\Excel;

class MyController extends Controller
{
    /**
    * @return \Illuminate\Support\Collection
    */
    public function importExportView()
    {
       return view('import');
    }

    /**
    * @return \Illuminate\Support\Collection
    */
    public function export()
    {
        return Excel::download(new UsersExport, 'users.xlsx');
    }

    /**
    * @return \Illuminate\Support\Collection
    */
    public function import()
    {
        Excel::import(new UsersImport,request()->file('file'));

        return back();
    }
}
```

Step 9
Create View

- Create a new blade View to import an Excel file, **resources/views/import.blade.php**.

```
<!DOCTYPE html>
<html>
<head>
    <title>Laravel 8 Import Export Excel to database Example - ItSolutionStuff.com</title>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.1.3
</head>
<body>

<div class="container">
    <div class="card bg-light mt-3">
        <div class="card-header">
            Laravel 8 Import Export Excel to database Example - ItSolutionStuff.com
        </div>
        <div class="card-body">
            <form action="{{ route('import') }}" method="POST" enctype="multipart/form-data">
                @csrf
                <input type="file" name="file" class="form-control">
                <br>
                <button class="btn btn-success">Import User Data</button>
                <a class="btn btn-warning" href="{{ route('export') }}">Export User Data</a>
            </form>
        </div>
    </div>
</div>

</body>
</html>
```

Complete
Open your Browser

- Terminal: php artisan serve
- Access on your Browser: http://localhost:8000/importExportView

Create PDF

Create PDF

Introduction

PDF is one of basic requirement when you are working with Government project or e commerce website. You may need to create pdf file for report or invoice etc. So, here i will give you very simple example for create pdf file with Laravel.

We will use dompdf package for Laravel.

Step 1

Create a new Laravel project Installation

- Terminal: composer create-project --prefer-dist laravel/laravel pdfdemo or
- Terminal: Laravel new pdfdemo or
- Terminal: curl -s “https://laravel.build/pdfdemo" | bash

Step 2
Install dompdf Package

- Terminal: composer require barryvdh/laravel-dompdf
- After successfully install package, open **config/app.php** file and add service provider and alias.

```
'providers' => [
        ....
        Barryvdh\DomPDF\ServiceProvider::class,
],

'aliases' => [
        ....
        'PDF' => Barryvdh\DomPDF\Facade::class,
]
```

Step 3
Add Route

- Open your **routes/web.php** file and add following route.

```php
<?php

use Illuminate\Support\Facades\Route;

use App\Http\Controllers\PDFController;

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

Route::get('generate-pdf', [PDFController::class, 'generatePDF']);
```

Step 4
Add Controller

- We require to create new controller PDFController that will manage generatePDF method of route.
- Create PDFController
- Terminal: php artisan make:controller PDFController

Step 4
Add Controller

- Add these codes:

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use PDF;

class PDFController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function generatePDF()
    {
        $data = [
            'title' => 'Welcome to ItSolutionStuff.com',
            'date' => date('m/d/Y')
        ];

        $pdf = PDF::loadView('myPDF', $data);

        return $pdf->download('itsolutionstuff.pdf');
    }
}
```

Step 5
Create View File

- Create myPDF.blade.php (resources/views/myPDF.blade.php) for layout of pdf file and put following code:

```
<!DOCTYPE html>
<html>
<head>
    <title>Hi</title>
</head>
<body>
    <h1>{{ $title }}</h1>
    <p>{{ $date }}</p>
    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
    tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
    quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
    consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
    cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
    proident, sunt in culpa qui officia deserunt mollit anim id est laborum.</p>
</body>
</html>
```

Complete
Open your Browser

- Terminal: php artisan serve
- Access on your Browser: http://localhost:8000/generatepdf

Create BarChart

Step 1

Create a new Laravel project

- Terminal: composer create-project --prefer-dist laravel/laravel bardemo or
- Terminal: Laravel new bardemo or
- Terminal: curl -s “https://laravel.build/bardemo" | bash

Step 2
Create a new Database

- Create Database: laravel
- You can use phpMyAdmin or any other database tool
- To connect database with application, Open .env file from application root. Search for DB_ and update your details.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=database_name
DB_USERNAME=database_user_name
DB_PASSWORD=database_password
```

Step 3
Create Model & Migration

- Terminal: php artisan make:model Student -m
- It will create two files:
  - Model – Student.php at /app/Models folder
  - Migration file – 2021_05_04_145928_create_students_table.php at / database/migrations folder.

Step 3
Create Model & Migration

1. Add these codes to the Migration file:

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateStudentsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('students', function (Blueprint $table) {
            $table->id();
            $table->string("name", 120);
            $table->integer("term1_marks");
            $table->integer("term2_marks");
            $table->integer("term3_marks");
            $table->integer("term4_marks");
            $table->text("remarks");
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('students');
    }
}
```

2. In the Student file:

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Student extends Model
{
    use HasFactory;

    public $timestamps = false;
}
```

3. Terminal: php artisan migrate

Step 4
Create Data Seeder

- Next, creating a seeder files to seed some dummy data for table.
- Terminal: php artisan make:seeder StudentSeeder
- Add these codes in the seeder file:
- Terminal: php artisan db:seed --class=StudentSeeder

```php
<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;

use DB;

class StudentSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $faker = \Faker\Factory::create();

        for ($loop = 0; $loop < 5; $loop++) {

            DB::table("students")->insert([
                "name" => $faker->name(),
                "term1_marks" => $faker->numberBetween(45, 95),
                "term2_marks" => $faker->numberBetween(45, 95),
                "term3_marks" => $faker->numberBetween(45, 95),
                "term4_marks" => $faker->numberBetween(45, 95),
                "remarks" => $faker->randomElement(["Good", "Excellent", "Needs Improvement", "Better", "Poor"])
            ]);
        }
    }
}
```

Step 5
Create Controller

- Terminal: php artisan make:controller StudentController
- Create an Index function in the StudentController and add these codes:

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Student;

class StudentController extends Controller
{
    public function index()
    {
        $students = Student::all();

        $dataPoints = [];

        foreach ($students as $student) {

            $dataPoints[] = array(
                "name" => $student['name'],
                "data" => [
                    intval($student['term1_marks']),
                    intval($student['term2_marks']),
                    intval($student['term3_marks']),
                    intval($student['term4_marks']),
                ],
            );
        }

        return view("bar-graph", [
            "data" => json_encode($dataPoints),
            "terms" => json_encode(array(
                "Term 1",
                "Term 2",
                "Term 3",
                "Term 4",
            )),
        ]);
    }
}
```

Step 6
Create View

- Go to /resources/views folder and create a file with name bar-graph.blade.php
- Open bar-graph.blade.php and write this complete code into it.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Bar Chart in Laravel 8 - Online Web Tutor</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
</head>
<body>

<div class="container">
  <h2 style="text-align:center;">Bar Chart in Laravel 8 - Online Web Tutor</h2>
  <div class="panel panel-primary">
    <div class="panel-heading">Bar Chart in Laravel 8</div>
    <div class="panel-body">
        <div id="bar-chart"></div>
    </div>
  </div>
</div>

<script src="https://code.highcharts.com/highcharts.js"></script>
<script src="https://code.highcharts.com/modules/exporting.js"></script>
<script src="https://code.highcharts.com/modules/export-data.js"></script>
<script src="https://code.highcharts.com/modules/accessibility.js"></script>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Bar Chart in Laravel 8 - Online Web Tutor</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
</head>
<body>

<div class="container">
  <h2 style="text-align:center;">Bar Chart in Laravel 8 - Online Web Tutor</h2>
  <div class="panel panel-primary">
    <div class="panel-heading">Bar Chart in Laravel 8</div>
    <div class="panel-body">
        <div id="bar-chart"></div>
    </div>
  </div>
</div>

<script src="https://code.highcharts.com/highcharts.js"></script>
<script src="https://code.highcharts.com/modules/exporting.js"></script>
<script src="https://code.highcharts.com/modules/export-data.js"></script>
<script src="https://code.highcharts.com/modules/accessibility.js"></script>
```

```
<script>
    $(function(){
        Highcharts.chart('bar-chart', {
            chart: {
                type: 'column'
            },
            title: {
                text: 'Student Term Wise Marks'
            },
            xAxis: {
                categories: <?= $terms ?>,
                crosshair: true
            },
            yAxis: {
                min: 0,
                title: {
                    text: 'Marks'
                }
            },
            tooltip: {
                headerFormat: '<span style="font-size:10px">{point.key} Marks</span><table>',
                pointFormat: '<tr><td style="color:{series.color};padding:0">{series.name}: </td>' +
                    '<td style="padding:0"><b>{point.y}</b></td></tr>',
                footerFormat: '</table>',
                shared: true,
                useHTML: true
            },
            plotOptions: {
                column: {
                    pointPadding: 0.2,
                    borderWidth: 0
                }
            },
            series: <?= $data ?>
        });
    });
</script>

</body>
</html>
```

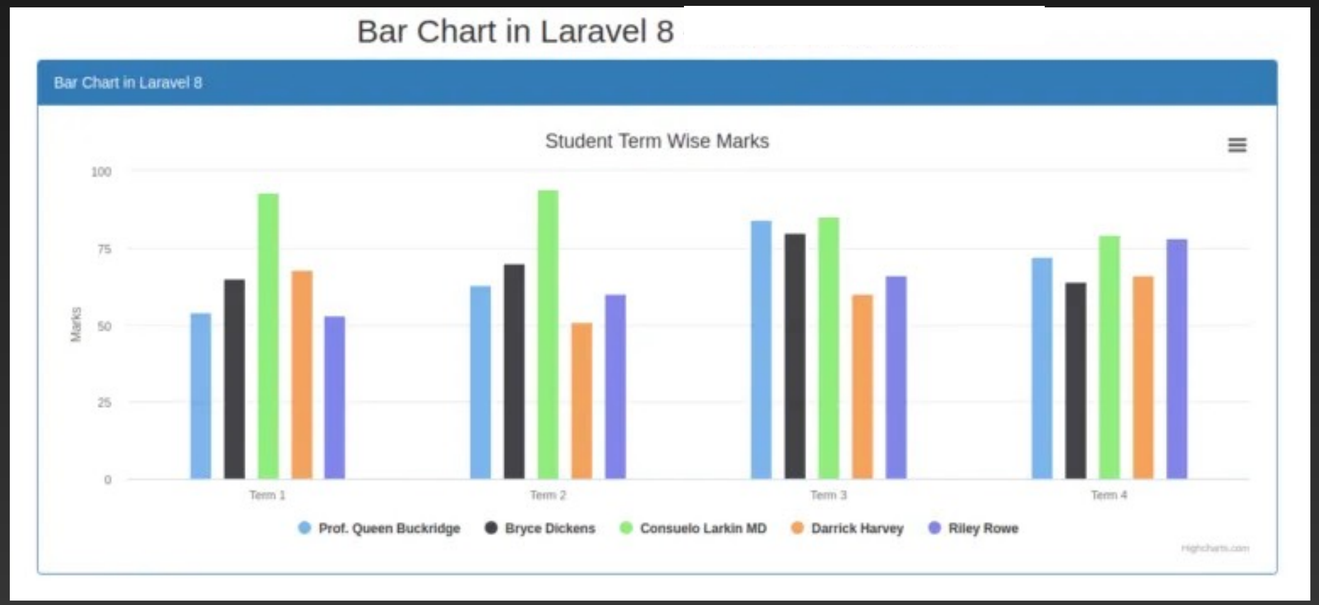
Step 7
Create Route

- Open **web.php** from /routes folder and add this route into it.
- Add code: Route::get('bar-graph', [StudentController::class, 'index']);

```
# Add this to header
use App\Http\Controllers\StudentController;

//...

Route::get('bar-graph', [StudentController::class, 'index']);
```

Complete
Open your Browser

- Terminal: php artisan serve
- Access on your Browser: http://localhost:8000/



Create PushNotifications

Introduction
Laravel & FCM

- Web Push Notification in Laravel, we will explore Firebase Cloud Messaging (FCM). It is one of the easiest to setup and you will get instant Push Notifactions using Laravel.

Step 1

Create a new Laravel project

- Terminal: composer create-project --prefer-dist laravel/laravel notificationdemo or
- Terminal: Laravel new notificationdemo or
- Terminal: curl -s “**https://laravel.build/notificationdemo**” | bash

Step 2
Create Database

- Create Database: laravel
- You can use phpMyAdmin or any other database tool
- To connect database with application, Open .env file from application root. Search for DB_ and update your details.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=database_name
DB_USERNAME=database_user_name
DB_PASSWORD=database_password
```

Step 3
Install laravel Ui Scaffolding

- Terminal: composer require laravel/ui
- Terminal: php artisan ui bootstrap --auth
- Terminal: npm install && npm run dev

Step 4

Update Server Key Prop in User Table

- Terminal: php artisan make:migration add_column_device_key
- We need to add the new column property in the current User table inside the database
- In **database/migrations/ xxxx_add_column_device_key.php** file, add these codes.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class AddColumnDeviceKey extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::table('users', function (Blueprint $table) {
            $table->string('device_key')->nullable();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        //
    }
}
```

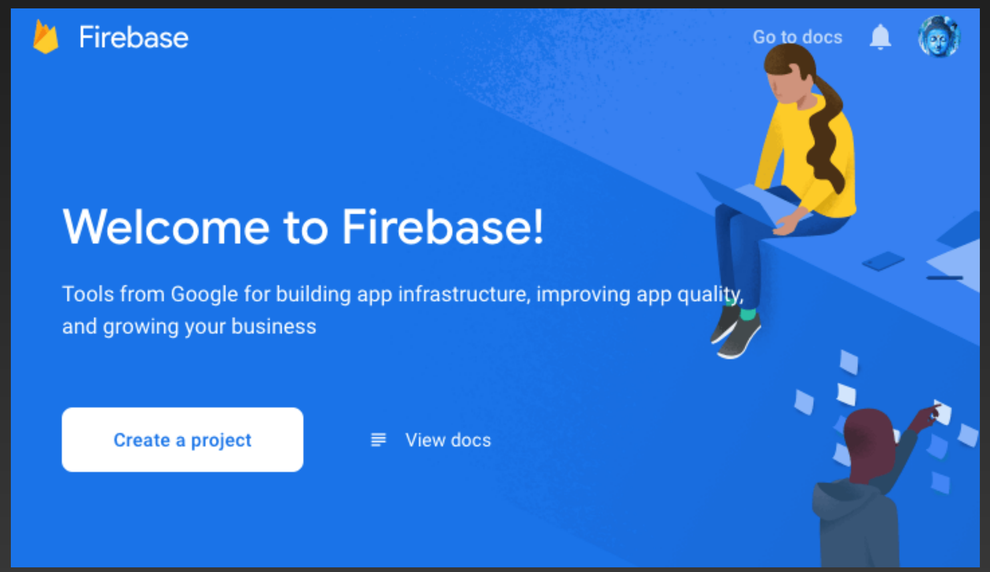
- After that, in the **app/Models/User.php** file and add the device key property.
- After all that has been done, migrate
- Terminal: php artisan migrate

```php
<?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
{
    use HasFactory, Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name',
        'email',
        'password',
        'device_key',
    ];

    /**
     * The attributes that should be hidden for arrays.
     *
     * @var array
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];

    /**
     * The attributes that should be cast to native types.
     *
     * @var array
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
    ];
}
```

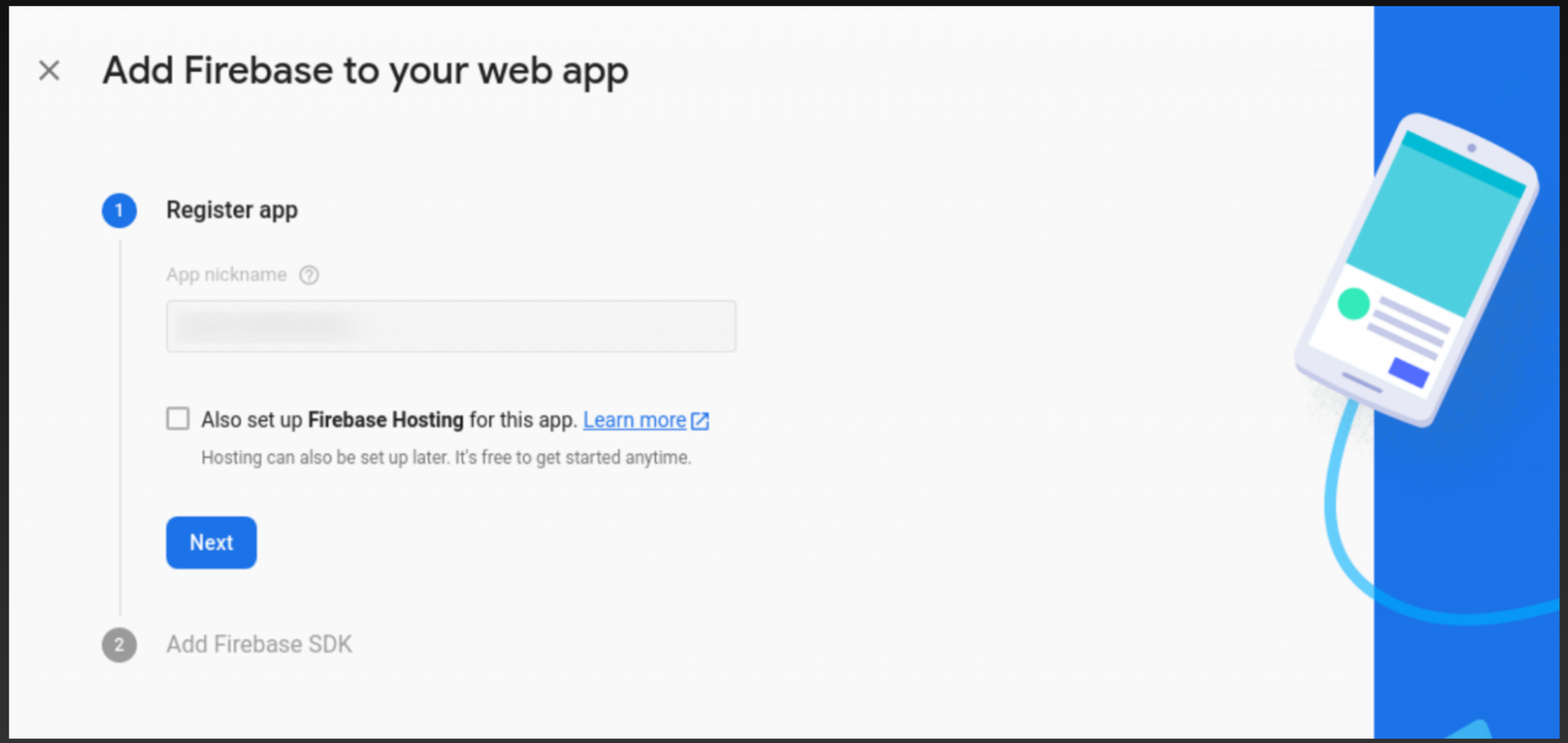
Step 5

Get Firebase Cloud Messaging (FCM) Server Key

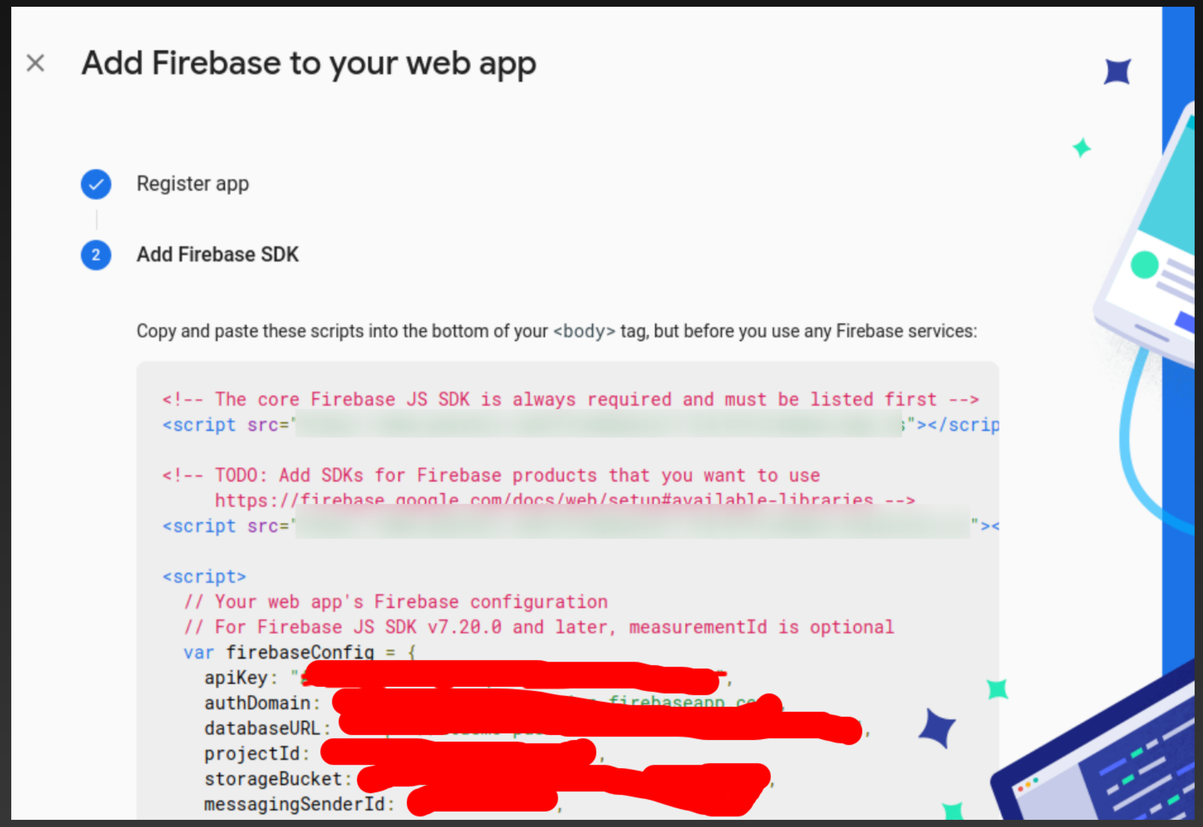
- This step will describe how to get the Firebase cloud messaging server key and Firebase web app’s configuration credentials.
- Goto Firebase, and Create a new Firebase Project.



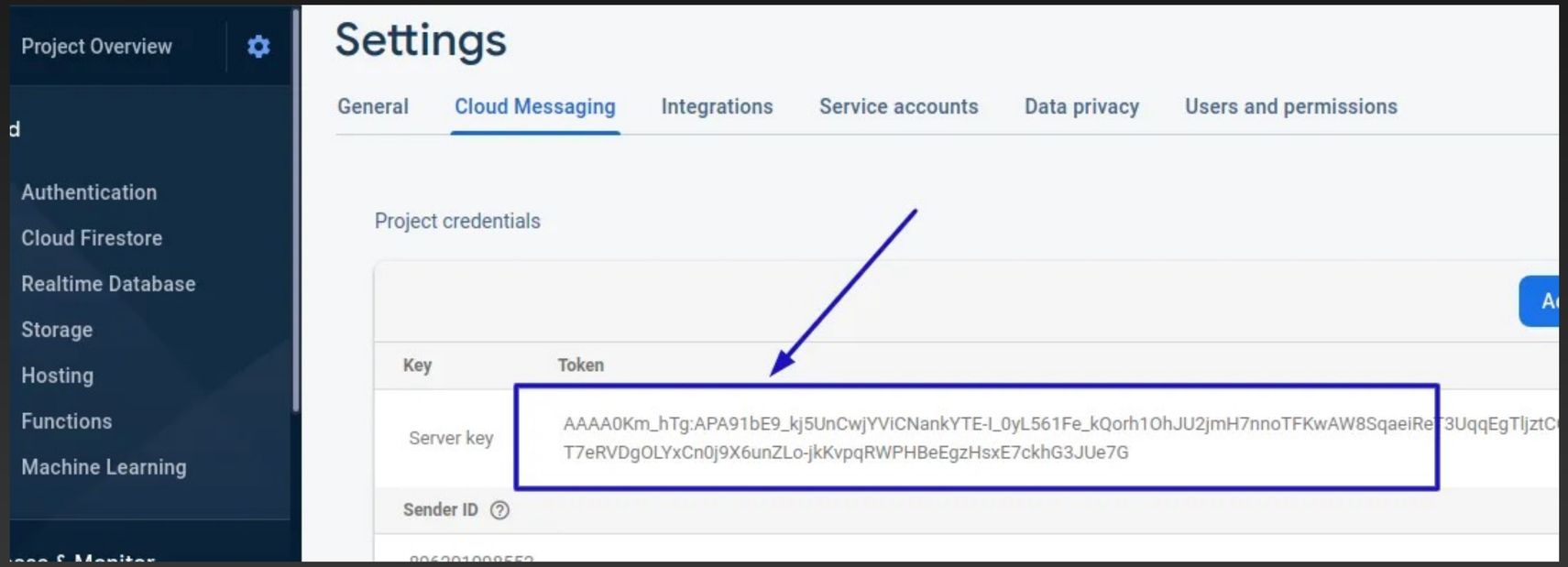
- Then, add the notification app name for adding Firebase to your web app.



- Copy the Firebase configuration keys, and this will help to connect Laravel to Firebase..



- Get into the Firebase project dashboard in the “Settings” page, click on the “Cloud Messaging” tab. You need to look for the “Server key”, copy this cloud messaging server key, and you have to paste this key in the $serverKey variable. That is defined in the web notification controller file’s sendWebNotification() function.



Step 6
Create Controller

- We need to create some functions in the controller file to handle web notifications, so first generate the controller.
- Terminal: php artisan make:controller WebNotificationController

Open WebNotificationController and place these fucntions inside:

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\User;

class WebNotificationController extends Controller
{

    public function __construct()
    {
        $this->middleware('auth');
    }

    public function index()
    {
        return view('home');
    }

    public function storeToken(Request $request)
    {
        auth()->user()->update(['device_key'=>$request->token]);
        return response()->json(['Token successfully stored.']);
    }
```

```php
    public function sendWebNotification(Request $request)
    {
        $url = 'https://fcm.googleapis.com/fcm/send';
        $FcmToken = User::whereNotNull('device_key')->pluck('device_key')->all();

        $serverKey = 'server key goes here';

        $data = [
            "registration_ids" => $FcmToken,
            "notification" => [
                "title" => $request->title,
                "body" => $request->body,
            ]
        ];
        $encodedData = json_encode($data);

        $headers = [
            'Authorization:key=' . $serverKey,
            'Content-Type: application/json',
        ];

        $ch = curl_init();

        curl_setopt($ch, CURLOPT_URL, $url);
        curl_setopt($ch, CURLOPT_POST, true);
        curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);
        curl_setopt($ch, CURLOPT_HTTP_VERSION, CURL_HTTP_VERSION_1_1);
        // Disabling SSL Certificate support temporarly
        curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
        curl_setopt($ch, CURLOPT_POSTFIELDS, $encodedData);

        // Execute post
        $result = curl_exec($ch);

        if ($result === FALSE) {
            die('Curl failed: ' . curl_error($ch));
        }

        // Close connection
        curl_close($ch);

        // FCM response
        dd($result);
    }
}
```

Step 7
Create Routes

- Import **WebNotificationController** controller in web.php,
  - use App\Http\Controllers\WebNotificationController;
- Then create three routes with Get and Post methods, so open and place the code in **routes/web.php** file.
  - Route::get('/push-notificaiton', [WebNotificationController::class, 'index'])->name('push-notificaiton');
  - Route::post('/store-token', [WebNotificationController::class, 'storeToken'])->name('store.token');
  - Route::post('/send-web-notification', [WebNotificationController::class, 'sendWebNotification'])->name('send.web-notification');

Step 8
Create View

- This step comprises setting up the home blade view file, integrate push notification code for the web app hence open **resources/views/home.blade.php** file, and replace with the following code.

```blade
@extends('layouts.app')

@section('content')
<div class="container">
    <div class="row justify-content-center">
        <div class="col-md-8">

                <button onclick="startFCM()"
                    class="btn btn-danger btn-flat">Allow notification
                </button>

            <div class="card mt-3">
                <div class="card-body">
                    @if (session('status'))
                    <div class="alert alert-success" role="alert">
                        {{ session('status') }}
                    </div>
                    @endif

                    <form action="{{ route('send.web-notification') }}" method="POST">
                        @csrf
                        <div class="form-group">
                            <label>Message Title</label>
                            <input type="text" class="form-control" name="title">
                        </div>
                        <div class="form-group">
                            <label>Message Body</label>
                            <textarea class="form-control" name="body"></textarea>
                        </div>
                        <button type="submit" class="btn btn-success btn-block">Send Notification</button>
                    </form>

                </div>
            </div>
        </div>
    </div>
</div>
```

```
<!-- The core Firebase JS SDK is always required and must be listed first -->
<script src="https://www.gstatic.com/firebasejs/8.3.2/firebase.js"></script>

<script>
    var firebaseConfig = {
        apiKey: 'api-key',
        authDomain: 'project-id.firebaseapp.com',
        databaseURL: 'https://project-id.firebaseio.com',
        projectId: 'project-id',
        storageBucket: 'project-id.appspot.com',
        messagingSenderId: 'sender-id',
        appId: 'app-id',
        measurementId: 'G-measurement-id',
    };

    firebase.initializeApp(firebaseConfig);
    const messaging = firebase.messaging();

    function startFCM() {
        messaging
            .requestPermission()
            .then(function () {
                return messaging.getToken()
            })
            .then(function (response) {
                $.ajaxSetup({
                    headers: {
                        'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
                    }
                });
                $.ajax({
                    url: '{{ route("store.token") }}',
                    type: 'POST',
                    data: {
                        token: response
                    },
                    dataType: 'JSON',
                    success: function (response) {
                        alert('Token stored.');
                    },
                    error: function (error) {
                        alert(error);
                    },
                });

            }).catch(function (error) {
                alert(error);
            });
    }
```

```
    messaging.onMessage(function (payload) {
        const title = payload.notification.title;
        const options = {
            body: payload.notification.body,
            icon: payload.notification.icon,
        };
        new Notification(title, options);
    });

</script>
@endsection
```

Step 9
Create Firebase (FCM) File

- Create a new **firebase-messaging-sw.js** file. This file holds the web push notification configurations.
- Add these codes

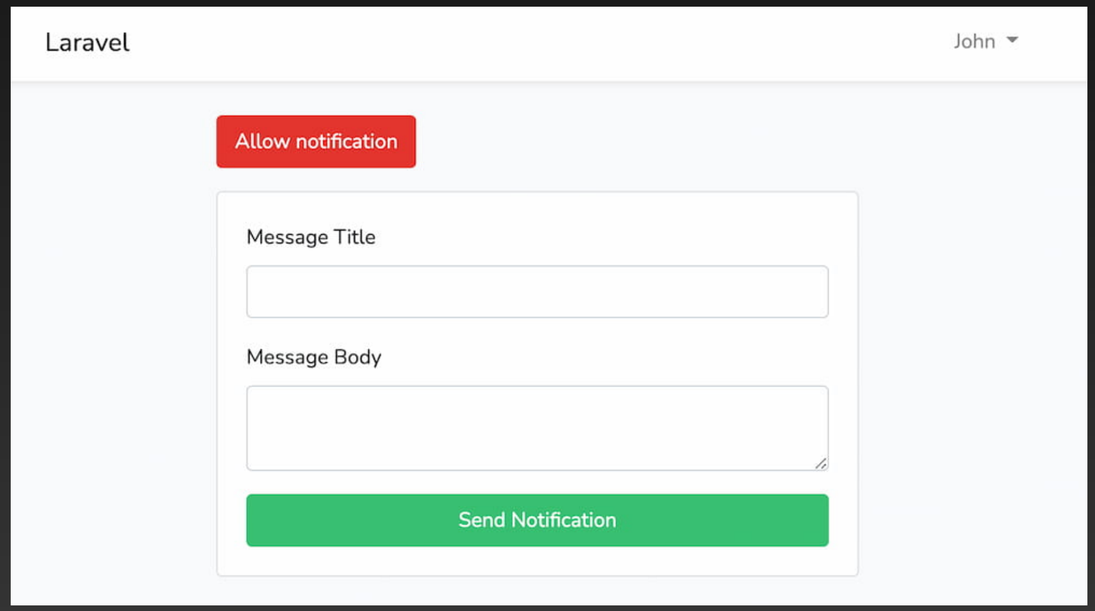
```
// Give the service worker access to Firebase Messaging.
// Note that you can only use Firebase Messaging here. Other Firebase libraries
// are not available in the service worker.importScripts('https://www.gstatic.com/fir
importScripts('https://www.gstatic.com/firebasejs/8.3.2/firebase-app.js');
importScripts('https://www.gstatic.com/firebasejs/8.3.2/firebase-messaging.js');

/*
Initialize the Firebase app in the service worker by passing in the messagingSenderId
*/
firebase.initializeApp({
    apiKey: 'api-key',
    authDomain: 'project-id.firebaseapp.com',
    databaseURL: 'https://project-id.firebaseio.com',
    projectId: 'project-id',
    storageBucket: 'project-id.appspot.com',
    messagingSenderId: 'sender-id',
    appId: 'app-id',
    measurementId: 'G-measurement-id',
});


// Retrieve an instance of Firebase Messaging so that it can handle background
// messages.
const messaging = firebase.messaging();
messaging.setBackgroundMessageHandler(function (payload) {
    console.log("Message received.", payload);

    const title = "Hello world is awesome";
    const options = {
        body: "Your notificaiton message .",
        icon: "/firebase-logo.png",
    };

    return self.registration.showNotification(
        title,
        options,
    );
});
```

Step 10

Send FCM Web Push Notification in Laravel

- Terminal: php artisan serve
- From your Browser, Register a new user. After creating a new account then you have to login into the app using your email and password.
  - # Register new user
    http://127.0.0.1:8000/register
  - # Sign-in to account
    http://127.0.0.1:8000/login

- After, you signed-in, then head over to following url:
- http://127.0.0.1:8000/push-notificaiton
- Next, you need to click on the allow notification button, it will generate the device id, also add push notification title and body in the given form.



- Here is the response for web push notification sent by you on the browser: