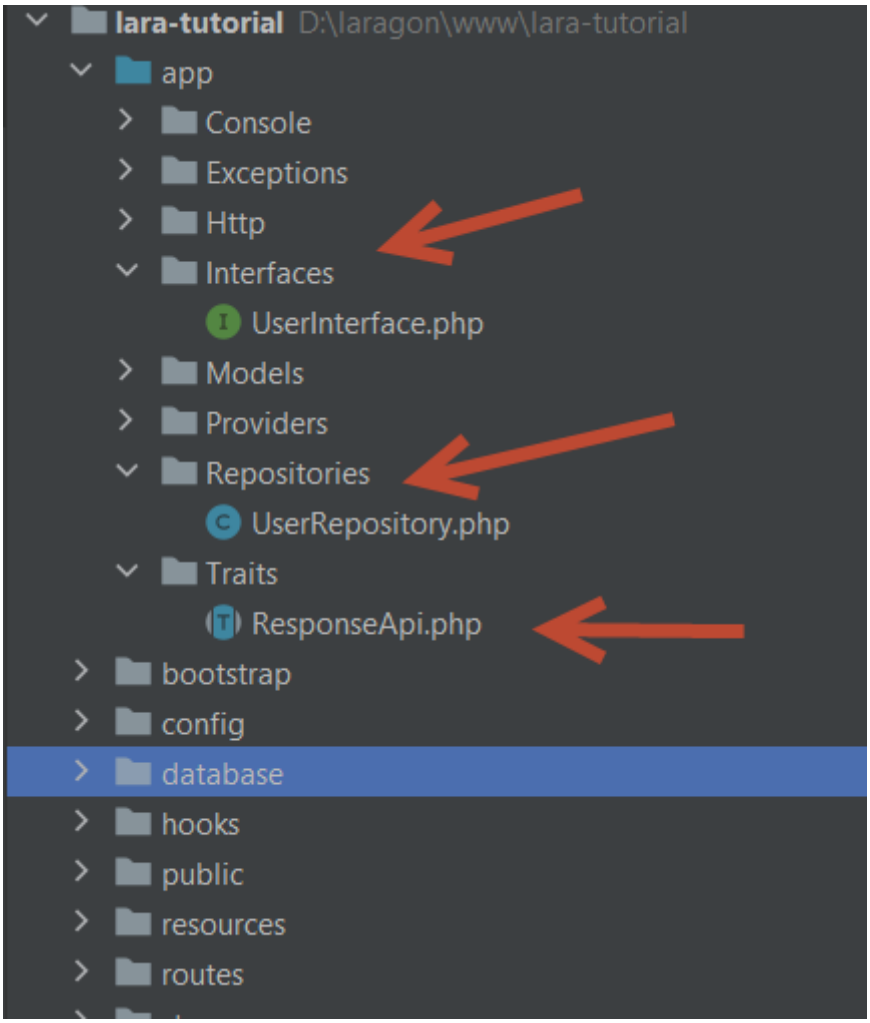
Let's Implementing the Repository Pattern!
Let's do it! First thing first, you gonna make three folders inside your **app** folder in your project, which are **Repositories**, **Interfaces**, and **Traits**. So inside your app folder will gonna look like this.



1. Trait

   I named **ResponseAPI.php**, place the file in your app/Traits folder. The code, you can write it down below.

```
<?php

namespace App\Traits;

use Illuminate\Http\JsonResponse;

trait ResponseApi
{
    /**
     * @param string $message
     * @param array|object|null $data
     * @param int $statusCode
     * @param bool $isSuccess
     */
    public function coreResponse(string $message, $data, int $statusCode,
$isSuccess = true)
    {
        if (! $message) {
```

```
            return response()->json(['message' => 'Message is required !'],
500);
        }

        if ($isSuccess) {
            return response()->json([
                'message' => $message,
                'error' => false,
                'code' => $statusCode,
                'results' => $data,
            ], $statusCode);
        } else {
            return response()->json([
                'message' => $message,
                'error' => true,
                'code' => $statusCode,
            ], $statusCode);
        }
    }

    /**
     * @param string $message
     * @param array|object $data
     * @param int $statusCode
     * @return JsonResponse
     */
    public function success(string $message, $data, int $statusCode = 200)
    {
        return $this->coreResponse($message, $data, $statusCode);
    }

    /**
     * @param string $message
     * @param int $statusCode
     * @param false $isSuccess
     * @return JsonResponse
     */
    public function error(string $message, int $statusCode = 500, $isSuccess
= false)
    {
        return $this->coreResponse($message, null, $statusCode, $isSuccess);
    }
}
```

2. Interface code

I named **UserInterface.php**, place the file in your app/Interfaces folder. The code, you can write it down below.

```
<?php

namespace App\Interfaces;

use App\Http\Requests\UserRequest;

interface UserInterface
{
```

```
    /**
     * @return mixed
     */
    public function getAllUsers();

    /**
     * @param int $id
     * @return mixed
     */
    public function getUserById(int $id);

    /**
     * @param UserRequest $request
     * @return mixed
     */
    public function createUser(UserRequest $request);

    /**
     * @param UserRequest $request
     * @param int $id
     * @return mixed
     */
    public function updateUser(UserRequest $request, int $id);

    /**
     * @param int $id
     * @return mixed
     */
    public function deleteUser(int $id);
}
```

3. Repositories

   I named **UserRepostitory.php**, place the file in your **app/Repositories** folder. The code, you can write it down below.

```
<?php

namespace App\Repositories;

use App\Http\Requests\UserRequest;
use App\Interfaces\UserInterface;
use App\Models\User;
use App\Traits\ResponseApi;
use Illuminate\Http\JsonResponse;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Str;

class UserRepository implements UserInterface
{
    use ResponseApi;

    /**
     * @return JsonResponse|mixed
     */
```

```
    public function getAllUsers()
    {
        // TODO: Implement getAllUsers() method.
        try {
            $user = User::all();

            return $this->success('All users', $user);
        } catch (\Exception $exception) {
            return $this->error($exception->getMessage(), $exception-
>getCode());
        }
    }

    /**
     * @param int $id
     * @return JsonResponse|mixed
     */
    public function getUserById(int $id)
    {
        // TODO: Implement getUserById() method.
        try {
            $user = User::query()->findOrFail($id);

            if (! $user) {
                return $this->error('No user with ID $id', 404);
            }

            return $this->success('user detail get by user Id', $user);
        } catch (\Exception $exception) {
            return $this->error($exception->getMessage(), $exception-
>getCode());
        }
    }

    /**
     * @param UserRequest $request
     * @param int $id
     * @return JsonResponse|mixed
     */
    public function updateUser(UserRequest $request, int $id)
    {
        // TODO: Implement requestUser() method.

        DB::beginTransaction();
        try {
            $user = User::query()->findOrFail($id);
            if (! $user) {
                return $this->error('No user with id $id', 404);
            }
            $user->name = $request->name;
            $user->email = preg_replace('/\s+/', '', strtolower($request-
>email));
            $user->save();
            DB::commit();

            return $this->success('Update successfully', $user);
        } catch (\Exception $exception) {
            DB::rollBack();
```

```
            return  $this->error($exception->getMessage(), $exception->getCode());
        }
    }

    /**
     * @param UserRequest $request
     * @return JsonResponse
     */
    public function createUser(UserRequest $request)
    {
        DB::beginTransaction();
        try {
            $user = new User;
            $user->name = $request->name;
            $user->email = preg_replace('/\s+/', '', strtolower($request->email));
            $user->password = Hash::make($request->password);
            $user->remember_token = Str::random(10);
            $user->save();
            DB::commit();

            return $this->success('User create successfully', $user);
        } catch (\Exception $exception) {
            DB::rollBack();

            return $this->error($exception->getMessage(), $exception->getCode());
        }
    }

    /**
     * @param int $id
     * @return JsonResponse|mixed
     */
    public function deleteUser(int $id)
    {
        // TODO: Implement deleteUser() method.

        DB::beginTransaction();
        try {
            $user = User::query()->findOrFail($id);
            if (! $user) {
                return $this->error("No user with id $id", 404);
            }
            $user->delete();
            DB::commit();

            return $this->success('User deleted', $user);
        } catch (\Exception $exception) {
            DB::rollBack();

            return $this->error($exception->getMessage(), $exception->getCode());
        }
    }
}
```

4. Controller

```
<?php

namespace App\Http\Controllers;

use App\Http\Requests\UserRequest;
use App\Interfaces\UserInterface;

class UserController extends Controller
{
    protected $userInterface;

    /**
     * UserController constructor.
     * @param UserInterface $userInterface
     */
    public function __construct(UserInterface $userInterface)
    {
        $this->userInterface = $userInterface;
    }

    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        return $this->userInterface->getAllUsers();
    }

    /**
     * @param UserRequest $request
     */
    public function store(UserRequest $request)
    {
        return $this->userInterface->createUser($request);
    }

    /**
     * Display the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function show(int $id)
    {
        return $this->userInterface->getUserById($id);
    }

    /**
     * @param UserRequest $request
     * @param int $id
     */
```

```
    public function update(UserRequest $request, int $id)
    {
        return $this->userInterface->updateUser($request, $id);
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function destroy(int $id)
    {
        return $this->userInterface->deleteUser($id);
    }
}
```

5. Request code

```
php artisan make: request UserRequest
```

```
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class UserRequest extends FormRequest
{
    /**
     * @var mixed
     *
     * /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            'name' => 'required|max:50',
            // I want to check the user if exists first
            // If exists, will pass the email for unique validation
            // This is efficient, rather than we creating another validation
for create or update
            'email' => request()->route('user')
```

```
                ? 'required|email|max:255|unique:users,email,'.request()-
>route('user')
                : 'required|email|max:255|unique:users,email',
            'password' => request()->route('user') ? 'nullable' :
'required|max:50',
        ];
    }

}
```

6. Registering a new Provider

   We're gonna registering a Provider right? We're just simply creating a new file inside **app/Provider**s folder.
   I named **RepositoryServiceProvider.php**, place the file in your app/Providers folder. The code, you can write it down below.

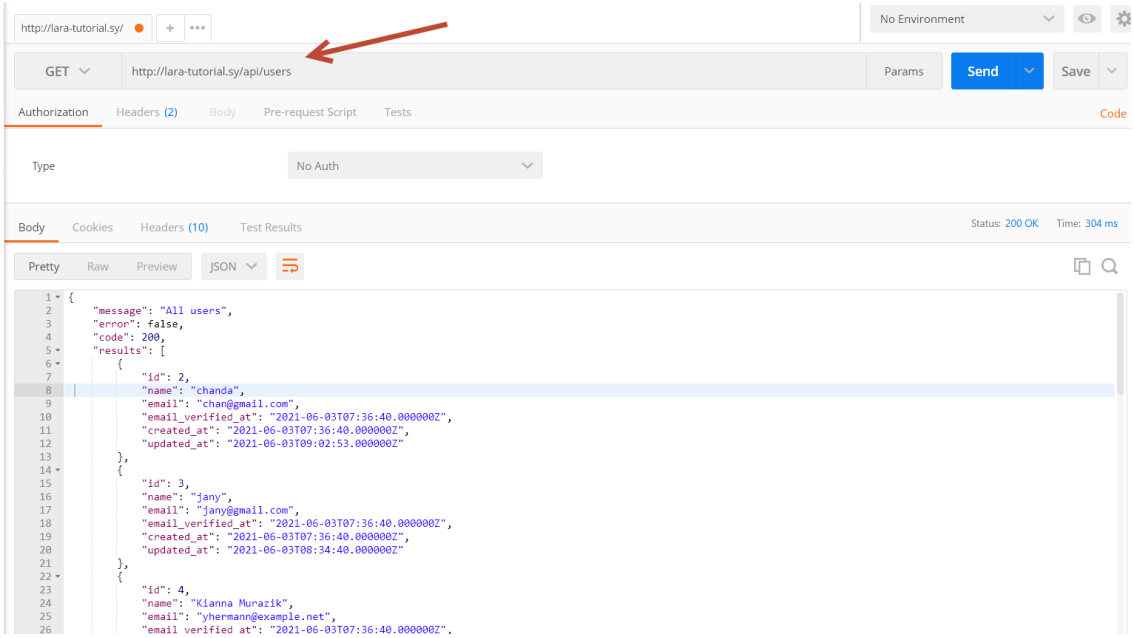
```
<?php

namespace App\Providers;

use Illuminate\Support\ServiceProvider;

class RepositoryServiceProvider extends ServiceProvider
{
    public function register()
    {
        // TODO: Change the autogenerated stub
        $this->app->bind(
            'App\Interfaces\UserInterface',
            'App\Repositories\UserRepository'
        );

        // Register another new interface and repository
//         $this->app->bind(
//             'App\Interfaces\NewInterfaceExampleName',
//             'App\Repositories\NewRepositoryExampleName'
//         );
    }
}
```

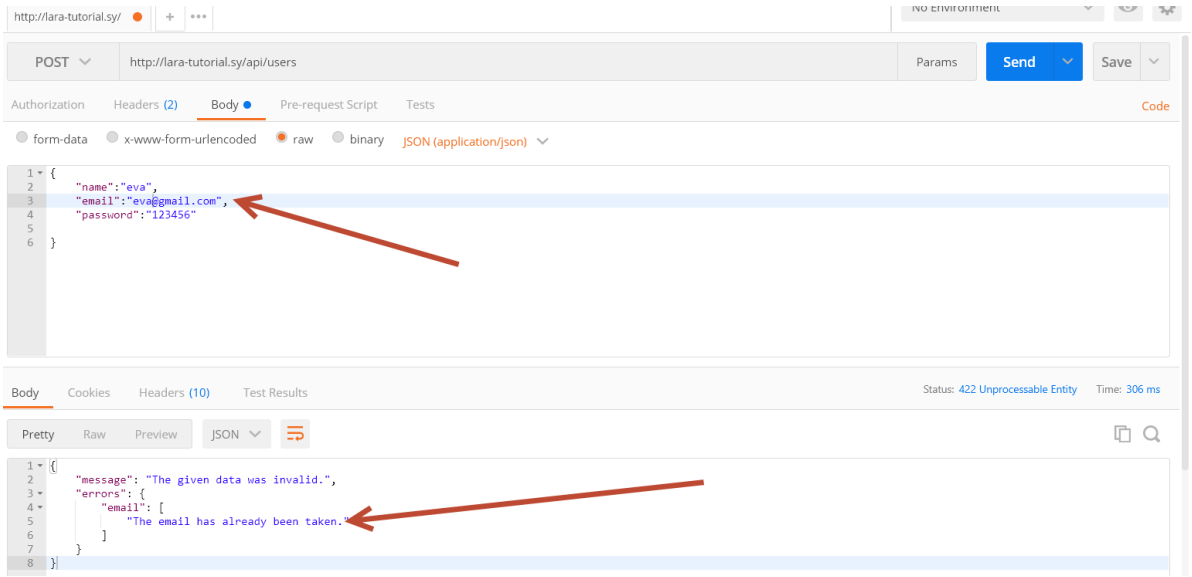
   After that, you can register the Provider in **config/app.php** file. Go to providers array, and place this code.

```
  'providers' => [

        // repository service provider

        App\Providers\RepositoryServiceProvider::class,
```

7. route

```
Route::resource('users', UserController::class);
```


http://lara-tutorial.sy/
No Environment
GET
http://lara-tutorial.sy/api/users
Params
Send
Save
Authorization
Headers (2)
Body
Pre-request Script
Tests
Code
Type
No Auth
Body
Cookies
Headers (10)
Test Results
Status: 200 OK
Time: 304 ms
Pretty
Raw
Preview
JSON
{
"message": "All users",
"error": false,
"code": 200,
"results": [
{
"id": 2,
"name": "chanda",
"email": "chan@gmail.com",
"email_verified_at": "2021-06-03T07:36:40.000000Z",
"created_at": "2021-06-03T07:36:40.000000Z",
"updated_at": "2021-06-03T09:02:53.000000Z"
},
{
"id": 3,
"name": "jany",
"email": "jany@gmail.com",
"email_verified_at": "2021-06-03T07:36:40.000000Z",
"created_at": "2021-06-03T07:36:40.000000Z",
"updated_at": "2021-06-03T08:34:40.000000Z"
},
{
"id": 4,
"name": "Kianna Murazik",
"email": "yhermann@example.net",
"email_verified_at": "2021-06-03T07:36:40.000000Z",


http://lara-tutorial.sy/
POST
http://lara-tutorial.sy/api/users
Params
Send
Save
Authorization
Headers (2)
Body
Pre-request Script
Tests
Code
form-data
x-www-form-urlencoded
raw
binary
JSON (application/json)
{
"name":"eva",
"email":"eva@gmail.com",
"password":"123456"
}
Body
Cookies
Headers (10)
Test Results
Status: 422 Unprocessable Entity
Time: 306 ms
Pretty
Raw
Preview
JSON
{
"message": "The given data was invalid.",
"errors": {
"email": [
"The email has already been taken."
]
}
}