

Laravel

STRUCTURE

DEVELOPMENT

Documentation

INDEX

1. Models
 - a. Table name
 - b. Fillable
 - c. Sort records
 - d. Define filterable parameters
 - e. Fetch records
 - f. Upload file
 - g. Store / Update records
 - h. Destroy record
2. Controllers
 - a. Constructor
 - b. Index
 - c. Show
 - d. Create
 - e. Edit
 - f. Store
 - g. Update
 - h. Destroy
3. Requests
4. Views
5. Site setting
6. Templates
7. Admin menus
8. Roles
9. Permissions

CONTENT

1. **Models:**

- a. Define table name.

```
protected $table = 'users';
```

- b. Define fillable column names.

```
/**
 * The attributes that are mass assignable.
 *
 * @var array
 */
protected $fillable = [
    'username',
    'first_name',
    'last_name',
    'email',
    'phone',
    'password',
    'status',
    'verified',
];
```

- c. Define blank orderBy variable.

```
public $orderBy = [];
```

- d. Define filterable columns list.

```
public function getFilters()
{
    $roleModel      = new \App\Role();
    $userMinRole    = $this->myRoleMinLevel(\Auth::user()->id);
    $roles          = $roleModel->getListing([
        'level_gte' => $userMinRole,
        'orderBy'   => 'roles__level'
    ])
    ->pluck('title', 'slug')
    ->all();

    return [
        'reset' => route('users.index'),
        'fields' => [
            'name'      => [
                'type'    => 'text',
                'label'   => 'Name'
            ],
            'role_slug' => [
                'type'    => 'select',
                'label'   => 'Role',
                'options' => $roles
            ],
            'status'    => [
                'type'    => 'select',
                'label'   => 'Status',
                'options' => [
                    1 => 'Enabled',
                    2 => 'Disabled'
                ]
            ],
            'created_at' => [
                'type'    => 'date',
                'label'   => 'Created At'
            ]
        ]
    ];
}
```

e. Create a function that will fetch that model's list.

```

public function getListing($srch_params = [], $offset = 0) {
    $listing = self::select(
        $this->table . ".*"
    )
    ->when(
        isset($srch_params['with']),
        function ($q) use ($srch_params) {
            return $q->with($srch_params['with']);
        })
    ->when(
        isset($srch_params['name']),
        function ($q) use ($srch_params) {
            return $q->where(
                \DB::raw(
                    "CONCAT({$this->table}.first_name, ' ',
{$this->table}.last_name) LIKE '%{$srch_params['name']}%"
                )
            );
        })
    ->when(
        isset($srch_params['email']),
        function ($q) use ($srch_params) {
            return $q->where(
                $this->table . ".email",
                "LIKE",
                "%{$srch_params['email']}%"
            );
        })
    ->when(
        isset($srch_params['role']),
        function ($q) use ($srch_params) {
            return $q->join(
                "user_roles AS ur",
                function ($join) use ($srch_params) {
                    $join->on(
                        $this->table . ".id",
                        "Ur.user_id"
                    );
                }
            );
        })
    );
}

```

```
        ->join(
            "roles AS r",
            function ($join) use ($srch_params) {
                $join->on("r.id", "ur.role_id")
                ->where("r.slug", $srch_params['role']);
            });
    })
    ->when(
        isset($srch_params['created_at']),
        function ($q) use ($srch_params) {
            return $q->whereDate(
                $this->table . ".created_at",
                $srch_params['created_at']
            );
        })
    ->when(
        isset($srch_params['status']),
        function ($q) use ($srch_params) {
            return $q->where(
                $this->table . ".status",
                $srch_params['status']
            );
        });

    if (isset($srch_params['id'])) {
        return $listing->where(
            $this->table . '.id',
            $srch_params['id']
        )
        ->first();
    }

    if (isset($srch_params['count'])) {
        return $listing->count();
    }

    if (isset($srch_params['orderBy'])) {
```

Laravel Developer Documentation

```
$this->orderBy = \App\Helpers\Helper::manageOrderBy(
    $srch_params['orderBy']
);

foreach ($this->orderBy as $key => $value) {
    $listing->orderBy($key, $value);
}
} else {
    $listing->orderBy($this->table . '.id', 'DESC');
}

if ($offset) {
    $listing = $listing->paginate($offset);
} else {
    $listing = $listing->get();
}

return $listing;
}
```

If you want to filter a record, please put a unique key in `$srch_params` array and its value, and make condition in the model

```
->when(
    isset($srch_params['status']),
    function ($q) use ($srch_params) {
        return $q->where(
            $this->table . ".status",
            $srch_params['status']
        );
    }
);
```

And call `getList` function from your controller.

```
$srch_params = $request->all();
$srch_params['status'] = 1;
$this->_data['data'] = $this->_model->getList(
    $srch_params,
    $this->_offset
);
```

```
);
```

If you pass the `$this->_offset` it will paginate, otherwise it will return all the data in respect to the search parameters which you have passed.

If you pass `id` in the `$srch_params` array it will return back a single object of the record rather than a list.

If you want to sort your record put `orderBy` in your `getListing` function.

```
if (isset($srch_params['orderBy'])) {
    $this->orderBy = \App\Helpers\Helper::manageOrderBy(
        $srch_params['orderBy']
    );
    foreach ($this->orderBy as $key => $value) {
        $listing->orderBy($key, $value);
    }
} else {
    $listing->orderBy($this->table . '.id', 'DESC');
}
```

To call this `orderBy` from the controller use below code.

```
$srch_params          = $request->all();
$srch_params['orderBy'] = 'users__first_name,-r__id';
$this->_data['data']    = $this->_model->getListing(
    $srch_params,
    $this->_offset
);
```

First you have to declare the sorting order by default it is doing Ascending order, if you want to fetch records in Descending order put a minus (-) before column name, then you need to declare the table name, or table alias name then put double underscores (__) then the column name. You may sort multiple columns by putting a comma (,) between two column names.

In the above example the listing is first order by users (table) first_name in ascending order, then role (r as table alias) id in descending order.

f. Relation with file or (profile picture)

First you need to define an unique fileType in App\File model


```
/**
 * var fileType
 * used for various files, type will be store in
 * database, location: your desired upload location.
 *
 * please create a new entity, if you want to
 * upload for a new module.
 */
public static $fileType = [
    'user_avatar' => [
        'type' => 1,
        'location' => 'u'
    ]
];
```

Define an unique type, and a location.

Then you need to add two functions in your model.

```
public function getAvatarAttribute() {
    return File::file($this->profile_pic);
}

public function profile_pic() {
    $entityType = isset(File::$fileType['user_avatar']['type']) ?
File::$fileType['user_avatar']['type'] : 0;
    return $this->hasOne(
        'App\File',
        'entity_id',
        'Id'
    )
    ->where('entity_type', $entityType);
}
```

The **profile_pic** function is for creating a relationship between your model and the File model. The **getAvatarAttribute** function is for checking file existence, and if the file type is image, it will generate a thumbnail.

- g. Create a function that will store or update data

```
public function store($input = [], $id = 0, $request = null)
{
    $status = 200;
    if ($id) {
        $data = $this->getListing(['id' => $id]);

        if (!$data) {
            return \App\Helpers\Helper::resp('Not a valid data', 400);
        }

        $data->update($input);
    } else {
        $status = 201;
        $data = $this->create($input);
    }

    return \App\Helpers\Helper::resp(
        'Changes has been successfully saved.',
        $status,
        $data
    );
}
```

First of all it is checking whether the id is given or not. If it is yes, then it will fetch that record. If the id given is wrong, then it will return with a response.

```
if (!$data) {
    return \App\Helpers\Helper::resp('Not a valid data', 400);
}
```

h. Create a function that will destroy data.

```
public function remove($id = null)
{
    $data = $this->getListing([
        'id' => $id,
```

```
'id_greater_than' => \Auth::user()->id
]);

if(!$data) {
    return \App\Helpers\Helper::resp('Not a valid data', 400);
}

$data->delete();

return \App\Helpers\Helper::resp(
    'Record has been successfully deleted.',
    200,
    $data
);
}
```

- i. Upload an Image. Please first follow point# 1.f before uploading an image.

```
public function uploadAvatar($data = [], $id, $request)
{
    $avatar = $data->profile_pic;
    $file = \App\File::upload($request, 'avatar', 'user_avatar', $data->id);

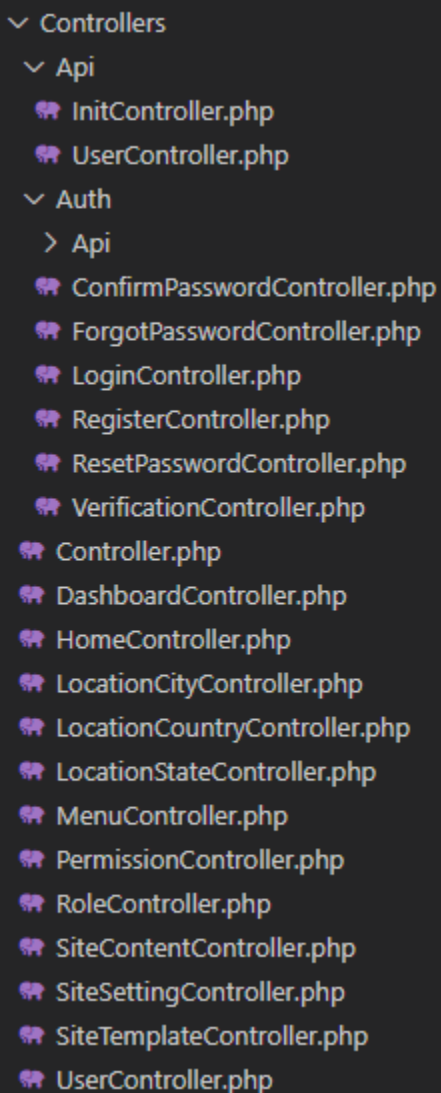
    // if file has successfully uploaded
    // and previous file exists, it will
    // delete old file.
    if($file && $avatar){
        \App\File::deleteFile($avatar, true);
    }

    return \App\Helpers\Helper::resp(
        'Changes has been successfully saved.',
        200,
        $file
    );
}
```

If you want to delete the old file then put 3 lines after file upload.

```
if($file && $avatar){  
    \App\File::deleteFile($avatar, true);  
}
```

2. **Controllers:** All controller files are stored under the App\Http\Controllers folder, and all API controller files are stored under the App\Http\Controllers\Api folder. There are some pre-installed controllers that exist to build this application.



The screenshot shows a file explorer view of the `App\Http\Controllers` directory. It contains several sub-directories and files:

- `Controllers` (expanded)
 - `Api` (expanded)
 - `InitController.php`
 - `UserController.php`
 - `Auth` (expanded)
 - `Api` (expanded)
 - `ConfirmPasswordController.php`
 - `ForgotPasswordController.php`
 - `LoginController.php`
 - `RegisterController.php`
 - `ResetPasswordController.php`
 - `VerificationController.php`
 - `Controller.php`
 - `DashboardController.php`
 - `HomeController.php`
 - `LocationCityController.php`
 - `LocationCountryController.php`
 - `LocationStateController.php`
 - `MenuController.php`
 - `PermissionController.php`
 - `RoleController.php`
 - `SiteContentController.php`
 - `SiteSettingController.php`
 - `SiteTemplateController.php`
 - `UserController.php`

Make the same name for API controllers. Such as `UserController`.

Please extend your controller from `App\Http\Controllers\Controller`.

```
use App\Http\Controllers\Controller;  
class UserController extends Controller
```

Use a constructor for the class before you start your application.

```
public function __construct($parameters = array())
{
    parent::__construct($parameters);
    $this->_module      = 'User';
    $this->_routePrefix = 'users';
    $this->_model       = new User();
}
```

```
$this->_module      = 'User';
```

This is your module name, Used to show in the page title and breadcrumb.

```
$this->_routePrefix = 'users';
```

Make it the same as your route defined. And create a folder under your views folder with the same name.

```
$this->_model       = new User();
```

Create an object of the default model for this controller.

We followed Laravel's resource mechanism to build this application. The functions that are used to run the application are described below.

A. index: This function is used to show the list of records.

```
/**
 * Display a listing of the resource.
 *
 * @return \Illuminate\Http\Response
 */
public function index(Request $request)
{
    $this->initIndex();
    $srch_params      = $request->all();
    $this->_data['data'] = $this->_model->getListing(
        $srch_params,
        $this->_offset
    );
    $this->_data['filters'] = $this->_model->getFilters();
    $this->_data['orderBy'] = $this->_model->orderBy;
```

```
return view(
    'admin.' . $this->_routePrefix . '.index',
    $this->_data
)
->with('i', ($request->input('page', 1) - 1) * $this->_offset);
}
```

`$this->initIndex();`

This function is used to initialize your listing. Such as,

- a. it will fetch permissions to show hide buttons (icons) from the listing page (UI),
- b. Generating breadcrumb. Etc.

For general controller index function. If you want to get extra details, please put it in your index function. Such as I want to merge another permission form another controller.

```
$manageRole = \App\Permission::checkModulePermissions(
    ['manageRole'],
    'PermissionController'
);
$this->_data['permission'] = array_merge(
    $this->_data['permission'],
    $manageRole
);
```

3. Requests: