ExoSuite Users API Documentation

Rechercher dans cet espace de documentation

Pages proposées

Content by label

There is no content with the specified labels

Pages mises à jour récemment

Database Diagram
Aug 07, 2019 • updated by Loïc Lopez • view change

Routes documentation
Mar 14, 2019 • updated by Loïc Lopez • view change

Routes documentation
Mar 06, 2019 • updated by Stanislas Deneubourg • view change

Routes
Jan 24, 2019 • updated by Stanislas Deneubourg • view change

Sub-resource of a $ITEM group
Jan 24, 2019 • updated by Loïc Lopez • view change

ExoSuite Guideline

Used shortcuts:

$ITEM will be a ressource:

```
$ITEM -> time
```

$METHOD will be an HTTP method:

```
$METHOD -> POST
```

$COLUMN will be a name of a sql column:

```
$COLUMN -> name
```

$TABLE will be a name of a sql table:

```
$TABLE -> times
```

Guideline:

- Store an Enum in database
- Eloquent: Mutators
- Available Methods Eloquent: Collections
- Automatically generate Uuid in Eloquent Model
- One To Many Relationship
    - belongsTo relationship
    - The Create Method with belongsTo relationship
- Many To Many Relationships

Tests Guideline and available assertions

- Authenticate
- Testing API
- Testing Database records

Eloquent Methods

For all available methods see https://laravel.com/docs/5.7/eloquent

`find($id)` takes an id and returns a single model. If no matching model exist, it returns `null`.

`findOrFail($id)` takes an id and returns a single model. If no matching model exist, it throws an error[1].

`first()` returns the first record found in the database. If no matching model exist, it returns `null`.

`firstOrFail()` returns the first record found in the database. If no matching model exist, it throws an error[1].

`get()` returns a collection of models matching the query.

`pluck($column)` returns a collection of just the values in the given column.

`toArray()` converts the model/collection into a simple PHP array.

`where$COLUMN() example : Times::whereName('Paris');`

```
Laravel will automatically transform this call into sql:
     where name = 'Paris' ( same call as Times::where('name', '=', 'Paris') )
```

[1] The error thrown by the `findOrFail` and `firstOrFail` methods is a `ModelNotFoundException`. If you don't catch this exception yourself, Laravel will respond with a 404, which is what you want most of the time.

Laravel Helpers

```
Laravel includes a variety of global "helper" PHP functions.
Many of these functions are used by the framework itself;
```

See: https://laravel.com/docs/master/helpers#available-methods

API Authentication (Passport)

See: https://laravel.com/docs/master/passport

Authorization

See: https://laravel.com/docs/master/authorization

Routes

Routes must be under a $ITEM group:

Norm for route names :

Exception:

- **Route::get('/id/{id}') will be named as follow: ->name('get_$ITEM_by_id');**
- **Route::get('/', 'TimeController@index')name('get_$ITEMs'); will be prefixed with sign of the plural 's' as follow: ->name('get_times');**

For all other routes, their names will be of the following format: name('$METHOD_$ITEM').

For example, if there is a route that does a POST on a Time item , then it's name will be : ->name('post_time');

```
Route::group('$ITEM', function() {
        Route::get('/id/{$ITEM}', 'TimeController@show')->name('get_$ITEM_by_id');
        Route::get('/', 'TimeController@index')->name('get_$ITEMs');
        Route::patch('/{$ITEM}', 'TimeController@update')->name('patch_$ITEM');
        Route::post('/', 'TimeController@store')->name('post_$ITEM');
        Route::delete('/{$ITEM}', 'TimeController@destroy')->name('delete_$ITEM');
});
```

Take care about naming routes, because it will be used in our unit tests, and will ensure we don't have to modify all the concerned test code if we ever change the route.

Complete guideline:

- Bind a route for a $ITEM
- Sub-resource of a $ITEM group

Complete example with a Time group:

Full CRUD methods

```
Route::group('time', function() {
        Route::get('/id/{time}', 'TimeController@show')->name('get_time_by_id');
        Route::get('/', 'TimeController@index')->name('get_times');
        Route::patch('/{time}', 'TimeController@update')->name('patch_time');
        Route::post('/', 'TimeController@store')->name('post_time');
        Route::delete('/{time}', 'TimeController@destroy')->name('delete_time');
});
```

Bind a route for a $ITEM

In order to make a route working with a path parameter like:

```
 Route::patch('/{time}', 'TimeController@update')->name('patch_time');
```

You will need to open app/Enums/BindType.php and add your ressource:

```
final class BindType extends Enum
{
    const $ITEM = "$item (in lowercase)";
}
```

And open app/Providers/RouteServiceProvider.php and add your bind:

```
/**
* Define your route model bindings, pattern filters, etc.
*
* @return void
*/
public function boot()
{
   Route::model(BindType::$ITEM, $ITEM::class);
}
```

Sub-resource of a $ITEM group

Imagine that we have checkpoints resources associated with times resources then:

```
Route::prefix('checkpoint')->group(function () {
   Route::get('/', 'CheckpointController@index')name('get_checkpoints');
   Route::prefix('{checkpoint}/time')->group(function () {
       Route::get('/', 'TimeController@index')name('get_times');
   });
});
```

For an example see: exosuite-users-api/routes/api.php#61

Create a Controller

Command:

```
php artisan make:controller $ITEMController --api
```

Example:

```
php artisan make:controller TimeController --api
```

Generated Controller

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class TimeController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        //
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        //
    }

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

    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, $id)
    {
        //
    }

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

Create an Enum

Generate Enum:

```
 php artisan make:enum Roles
```

Generated Enum:

```
<?php

namespace App\Enums;

use BenSampo\Enum\Enum;

final class Roles extends Enum
{
    const OptionOne = 0;
    const OptionTwo = 1;
    const OptionThree = 2;
}
```

Complete ExoSuite API Controller

```
<?php

namespace App\Http\Controllers;

use App\Http\Requests\Times\CreateTimeRequest;
use App\Http\Requests\Times\DeleteTimeRequest;
use App\Http\Requests\Times\GetTimeRequest;
use App\Http\Requests\Times\UpdateTimeRequest;
use Webpatser\Uuid\Uuid;

class TimeController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\JsonResponse
     */
    public function index()
    {
        return $this->ok([]);
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param CreateTimeRequest $request
     * @return \Illuminate\Http\JsonResponse
     */
    public function store(CreateTimeRequest $request)
    {
        return $this->noContent();
    }

    /**
     * Display the specified resource.
     *
     * @param GetTimeRequest $request
     * @param Uuid $id
     * @return \Illuminate\Http\JsonResponse
     */
    public function show(GetTimeRequest $request, Uuid $id)
    {
        return $this->ok([]);
    }

    /**
     * Update the specified resource in storage.
     *
     * @param UpdateTimeRequest $request
     * @param Uuid $id
     * @return \Illuminate\Http\JsonResponse
     */
    public function update(UpdateTimeRequest $request, Uuid $id)
    {
        return $this->noContent();
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param DeleteTimeRequest $request
     * @param Uuid $id
     * @return \Illuminate\Http\JsonResponse
     */
    public function destroy(DeleteTimeRequest $request, Uuid $id)
    {
        return $this->noContent();
```

```
    }
}
```

Controller methods for a sub-resource

Exemple of a PATCH Request:

```
PATCH /checkpoint/a257df24-1f6b-11e9-ab14-d663bd873d93/time/ab6b2d32-1f6b-11e9-ab14-d663bd873d93
{
 data ....
}
```

Take care about the order of function parameters!!

For example let's update the '*update*' method of the controller Time which is a sub-resource of a Checkpoint resource:

```
public function update(UpdateTimeRequest $request, Checkpoint $checkpoint, Time $time)
{
  ....
}
```

For an exemple of a sub-resource update function see: app/Http/Controllers/MessageController.php#51 where Message is a sub-resource of the Group resource.

POST Request

Error codes:

`They will be handled automatically by Request Class.`

Success returns

Return 201 with data

```
public function store(CreateTimeRequest $request)
{
        // data processing
    return $this->created($times);
}
```

Return 201 with data and location

```
public function store(CreateTimeRequest $request)
{
        // data processing
    return $this->created($times, "/times/{$times->id}");
}
```

Return 204 (no content)

```
public function store(CreateTimeRequest $request)
{
        // data processing
    return $this->noContent();
}
```

Gideline:

- Generate POST Request Class
- Update store method
- Get validated data
- Complete store method

Generate POST Request Class

Create a Request Class:

Artisan

```
php artisan make:request $ITEM\\Create$ITEMRequest
```

Example:

Artisan

```
php artisan make:request Time\\CreateTimeRequest
```

Generated Request Class

```
<?php

namespace App\Http\Requests\Time;

use Illuminate\Foundation\Http\FormRequest;

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

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            //
        ];
    }
}
```

Update store method

You will change Illuminate\Http\Request by generated Request Class "Create$ITEMRequest"

Update store method

```
<?php

namespace App\Http\Controllers;

use App\Http\Requests\Time\CreateTimeRequest;

class TimesController extends Controller
{

    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(CreateTimeRequest $request)
    {
        //
    }
}
```

Get validated data

Get validated data

```
<?php

namespace App\Http\Controllers;

use App\Http\Requests\Time\CreateTimeRequest;

class TimesController extends Controller
{

    /**
     * Store a newly created resource in storage.
     *
     * @param  CreateTimeRequest $request
     * @return \Illuminate\Http\Response
     */
    public function store(CreateTimeRequest $request)
    {
        $times = $request->validated();
    }
}
```

Get validated data and create ressource

```
<?php

namespace App\Http\Controllers;

use App\Http\Requests\Time\CreateTimeRequest;
use App\Models\Time;

class TimesController extends Controller
{

    /**
     * Store a newly created resource in storage.
     *
     * @param  CreateTimeRequest  $request
     * @return \Illuminate\Http\Response
     */
    public function store(CreateTimeRequest $request)
    {
        $time = Time::create($request->validated());
    }
}
```

Complete store method

Create and return 201 without data

```
public function store(CreateTimesRequest $request)
{
        $times = Times::create($request->validated());
    return $this->created();
}
```

Create and return 201 with data

```
public function store(CreateTimesRequest $request)
{
        $times = Times::create($request->validated());
    return $this->created($times);
}
```

Create and return 201 with data and location

```
public function store(CreateTimesRequest $request)
{
        $times = Times::create($request->validated());
    return $this->created($times, "/times/{$times->id}");
}
```

GET Request

Success returns:

Return 200 with data

```
public function index()
{
        // get data
    return $this->ok($times);
}
```

Methods:

- Method: Get all ressources
- Method: Get by id

Method: Get all ressources

Index method

```
/**
 * Display a listing of the resource.
 *
 * @return \Illuminate\Http\Response
 */
public function index()
{
  //logic to retrieve data
  return $this->ok($times);
}
```

Method: Get by id

Create a Request Class:

Artisan

```
php artisan make:request $ITEM\\Get$ITEMRequest
```

Example:

Artisan

```
php artisan make:request Time\\GetTimeRequest
```

Generated Request Class

```
<?php

namespace App\Http\Requests\Time;

use Illuminate\Foundation\Http\FormRequest;

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

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            //
        ];
    }
}
```

CAUTION: You will need to extends to RouteParamRequest in order to verify if the id passed as argument exists in database or all others actions

Modified/Final Get Request Class

```
<?php

namespace App\Http\Requests\Times;

use App\Http\Requests\Abstracts\RouteParamRequest;

/**
 * Class GetTimeRequest
 * @property string id
 * @package App\Http\Requests
 */
class GetTimesRequest extends RouteParamRequest
{
    /**
     * 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 [
            'id' => 'exists:$TABLE' // this will check if the id exists in $TABLE
        ];
    }
}
```

See https://laravel.com/docs/5.7/eloquent#retrieving-single-models and Eloquent Methods for findOrFail or firstOrFail

Show method

```
/**
* Display the specified resource.
*
* @param  GetTimeRequest  $request
* @param  Uuid $id
* @return \Illuminate\Http\JsonResponse
*/
public function show(GetTimesRequest $request, Uuid $id)
{
    $time = Time::findOrFail($id); // or firstOrFail
    return $this->ok($time);
}
```

PATCH Request

Error codes:

They will be handled automatically by Request Class.

Success returns:

Return 200 with data

```
public function store(CreateTimeRequest $request)
{
        // data processing
    return $this->ok($data);
}
```

Return 204 (no content)

```
public function store(CreateTimeRequest $request)
{
        // data processing
    return $this->noContent();
}
```

Guideline:

- Generate PATCH Request Class
- Update the update controller method

Generate PATCH Request Class

Create a Request Class:

Artisan

```
php artisan make:request $ITEM\\Update$ITEMRequest
```

Example:

Artisan

```
php artisan make:request Time\\UpdateTimeRequest
```

Generated Request Class

```
<?php

namespace App\Http\Requests\Time;

use Illuminate\Foundation\Http\FormRequest;

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

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            //
        ];
    }
}
```

Update the update controller method

```
/**
* Update the specified resource in storage.
*
* @param  UpdateTimesRequest  $request
* @param  Uuid  $id
* @return \Illuminate\Http\JsonResponse
*/
public function update(UpdateTimesRequest $request, Uuid $id)
{
  //
}
```

DELETE Request

Error codes:

They will be handled automatically by Request Class.

Success returns:

Return 200 with data

```
public function destroy(DeleteTimeRequest $request)
{
        // data processing
    return $this->ok($data);
}
```

Return 204 (no content)

```
public function destroy(DeleteTimeRequest $request)
{
        // data processing
    return $this->noContent();
}
```

Guideline:

- Generate DELETE Request Class
- Update the delete method

Generate DELETE Request Class

Create a Request Class:

Artisan

```
php artisan make:request $ITEM\\Delete$ITEMRequest
```

Example:

Artisan

```
php artisan make:request Time\\DeleteTimeRequest
```

Generated Request Class

```
<?php

namespace App\Http\Requests\Times;

use Illuminate\Foundation\Http\FormRequest;

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

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            //
        ];
    }
}
```

Update the delete method

```
/**
* Update the specified resource in storage.
*
* @param  DeleteTimesRequest  $request
* @param  Uuid  $id
* @return \Illuminate\Http\JsonResponse
*/
public function destroy(DeleteTimesRequest $request, Uuid $id)
{
  // logic to delete ressource
  return $this->noContent();
}
```

Database, Tables and Eloquent Models

Generate model and migration

```
php artisan make:model Models\\$ITEM -m
```

Example:

```
php artisan make:model Models\\Time -m
```

To chose the best relationship read:

- **https://laravel.com/docs/master/eloquent-relationships**
- **One To Many Relationship**
- **Many To Many Relationships**

See datatypes:

- **https://laravel.com/docs/master/migrations#columns**

Guideline:

- Required relationships loads (Eager Loading)
- Store an Enum in database
- Eloquent: Mutators
- Available Methods Eloquent: Collections
- Automatically generate Uuid in Eloquent Model
- One To Many Relationship
    - belongsTo relationship
    - The Create Method with belongsTo relationship
- Many To Many Relationships

Required relationships loads (Eager Loading)

Eager Loading must be used to load mutliple tables at the same time.

```
When accessing Eloquent relationships as properties, the relationship data is "lazy loaded".
This means the relationship data is not actually loaded until you first access the property.
However, Eloquent can "eager load" relationships at the time you query the parent model.
```

See: https://laravel.com/docs/master/eloquent-relationships#eager-loading

Sometimes you may need to eager load a relationship after the parent model has already been retrieved. For example, this may be useful if you need to dynamically decide whether to load related models:

Example:

```
if ($someCondition) {
    $user->load('times');
}
```

See: https://laravel.com/docs/master/eloquent-relationships#lazy-eager-loading

Store an Enum in database

Example:

```
$table->string('level');
```

Let's imagine you have an *enum* :

```
final class NotificationType extends Enum
{
    const FOLLOW = 'follow';
    const NEW_MESSAGE = 'new_message';
}
```

Migration:

```
$table->string('notification_type');
```

Eloquent: Mutators

Accessors and mutators allow you to format Eloquent attribute values when you retrieve or set them on model instances.
For example, you may want to use the Laravel encrypter to encrypt a value while it is stored in the database, and then automatically decrypt the attribute when you access it on an Eloquent model.

To define an accessor, create a `getFooAttribute` method on your model where `Foo` is the "studly" cased name of the column you wish to access.

In this example, we'll define an accessor for the `full_name` attribute. The accessor will automatically be called by Eloquent when attempting to retrieve the value of the `full_name`attribute:

User Model:

```
/**
 * Get the user's full name.
 *
 * @return string
 */
public function getFullNameAttribute()
{
    return "{$this->first_name} {$this->last_name}";
}
```

Get value of Mutator:

```
$user->full_name
```

See: https://laravel.com/docs/master/eloquent-mutators

Available Methods Eloquent: Collections

All multi-result sets returned by Eloquent are instances of the Illuminate\Database\Eloquent\Collection object, including results retrieved via the get method or accessed via a relationship.

See: https://laravel.com/docs/master/eloquent-collections#available-methods

Automatically generate Uuid in Eloquent Model

Update your model as follow:

```
<?php

namespace App;

use App\Models\Traits\Uuids;
use Illuminate\Database\Eloquent\Model;

class Time extends Model
{
    use Uuids;

    public $incrementing = false;
}
```

One To Many Relationship

Eloquent will automatically determine the proper foreign key column on the model. By convention, Eloquent will take the "snake case" name of the owning model and suffix it with `_id`.

So, for this example, Eloquent will assume the foreign key on the Time model is `user_id`.

```
php artisan make:model Models\\Time -m
```

Generated migration

```
class CreateTimesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('times', function (Blueprint $table) {
            $table->increments('id');
            $table->timestamps();
        });
    }

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

Update the migration:

updated migration

```
class CreateTimesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('times', function (Blueprint $table) {
            $table->increments('id');
            $table->uuid('user_id');
            $table->foreign('user_id')->references('id')->on('users');
            $table->timestamps();
        });
    }

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

See :

- **https://laravel.com/docs/master/eloquent-relationships#one-to-many**

Guideline:

- belongsTo relationship
- The Create Method with belongsTo relationship

belongsTo relationship

When updating a `belongsTo` relationship, you may use the `associate` method. This method will set the foreign key on the child model:

```
$time = Time::find($id);

$user->times()->associate($time);

$user->save();
```

When removing a `belongsTo` relationship, you may use the `dissociate` method. This method will set the relationship's foreign key to `null`:

```
$user->times()->dissociate();

$user->save()
```

See:

- **https://laravel.com/docs/master/eloquent-relationships#updating-belongs-to-relationships**

The Create Method with belongsTo relationship

see https://laravel.com/docs/master/eloquent-relationships#the-create-method

Example:

```
$comment = user->times()->create([
    'message' => 'A new times.',
]);
```

Many To Many Relationships

For example, let's imagine a user can have many roles and a role can have many users:

Create a role table with a model

```
php artisan make:model Models\\Role -m
```

Create a pivot table

```
php artisan make:migration create_role_users_table
```

Implemented with roles see :

- **2018_10_24_222444_create_roles_table.php**
- **2018_10_24_222559_create_role_users_table.php**
- **User.php**
- **Role.php**

References:

- **https://laravel.com/docs/master/eloquent-relationships#many-to-many**
- **https://laravel.com/docs/master/eloquent-relationships#updating-many-to-many-relationships**

Tests Guideline and available assertions

Unit Tests (Unit directory):

```
Unit Tests are written from a programmers perspective.
They are made to ensure that a particular method (or a unit) of a class performs a set of specific tasks.
```

Functional Tests (Feature directory):

Functional Tests are written from the user's perspective. They ensure that the system is functioning as users are expecting it to.

See:

- **https://laravel.com/docs/5.7/http-tests**
- **https://laravel.com/docs/5.7/database-testing**

Available assertions:

- **https://laravel.com/docs/5.7/http-tests#available-assertions**
- **https://laravel.com/docs/5.7/database-testing#available-assertions**

Guideline:

- Authenticate
- Testing API
- Testing Database records

Authenticate

Use:

```
Passport::actingAs();
```

See: https://laravel.com/docs/master/passport#testing

Testing API

Request class: `use Illuminate\Http\Request;`

Response class: `use Illuminate\Http\Response;`

Example for get array of times:

```
$response = $this->json(Request::METHOD_GET, route('get_times'));
```

Assert a status:

```
$response->assertStatus(Response::HTTP_CREATED)
```

Testing Database records

Example:

```
public function testDatabase()
{
    // Make call to application...

    $this->assertDatabaseHas('users', [
        'email' => 'sally@example.com'
    ]);
}
```

See: https://laravel.com/docs/5.7/database-testing

Update composer command for windows

```
composer update --ignore-platform-reqs
```