ROLES AND PERMISSIONS WITH LOCK

- Charles Griffin
- Software Engineer at Indatus

WHAT THIS TALK IS ABOUT

- Roles and permissions that aren't painful
- Using Lock as a Laravel package to accomplish painless roles and permissions
- Under the hood of Lock (drivers, service providers, extending core functionality)
- How to build custom roles and permissions for your specific domain

WHY ROLES AND PERMISSIONS ARE SO PAINFUL

- What should permissions be applied to. Should they be at the controller level? Or the model level? ie) Only admins can create users. Or only admins can visit this url.

- Role Hierarchies. ie) A super admin can do everything an admin can do PLUS edit users. A standard admin is really just a guest with ability to edit their own posts.

- Most approaches are prone to leaving unintentional security laps. ie) Forgetting that this user can only view their own tasks.

- Many moving parts

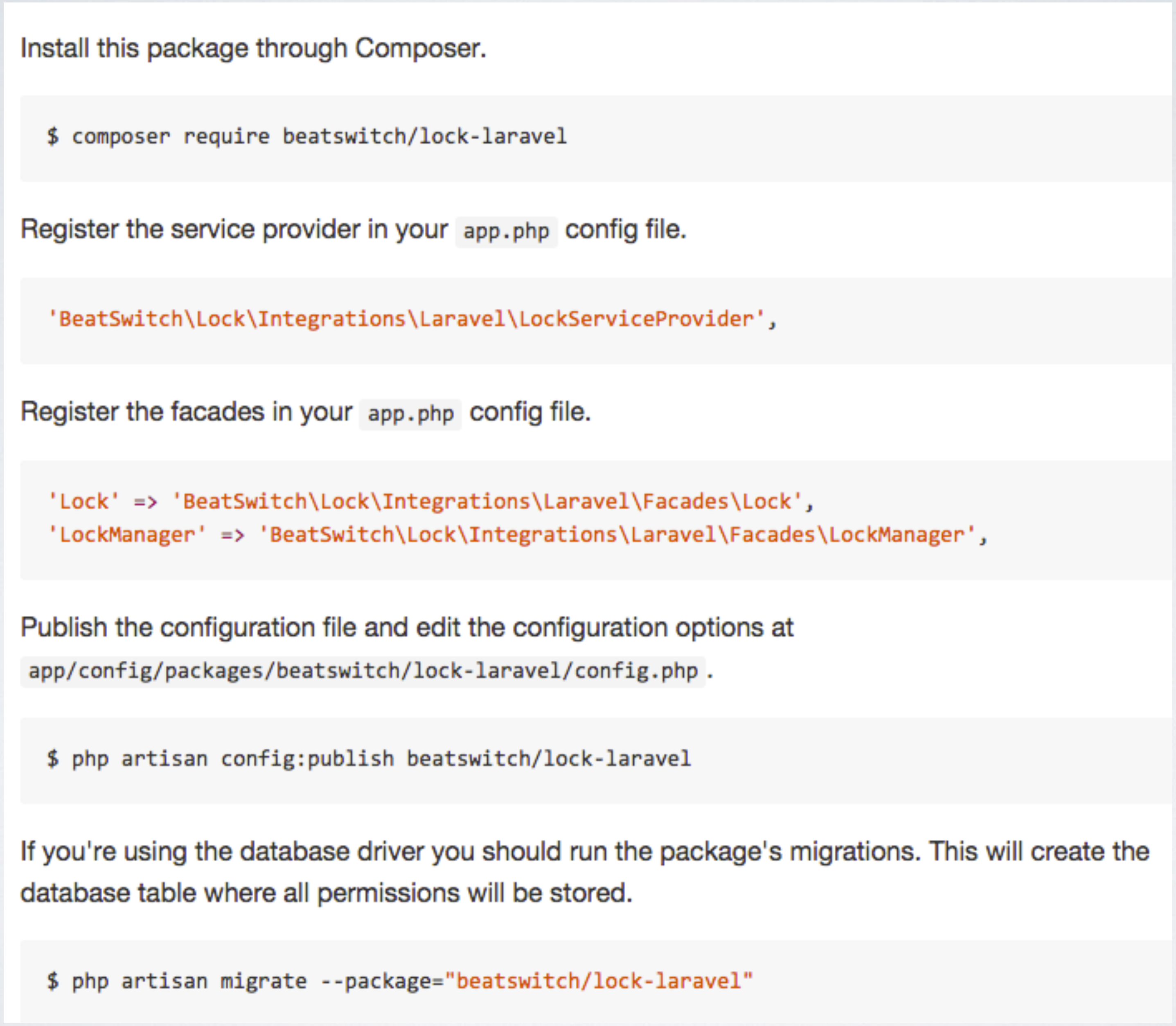
WHAT WOULD MAKE ROLES AND PERMISSIONS EASIER

- Everything starts with the database schema. It would be nice to use a schema that is flexible enough to handle the many combinations of roles and permissions.
- Syntactical sugar to go with that impressively designed schema.
- Abstraction away from those hairy queries so we can just focus on the domain problems.
- Many different ways to apply permissions. Ie) entity level, feature level, entity identifiers.

THIS IS WHERE LOCK COMES IN

- Flexible ACL permissions for multiple identities (callers)
- Static or persistent drivers to store permissions
- Roles
- Conditions (Asserts)
- Easily implement ACL functionality on your caller or role with a trait

INSTALLATION



Install this package through Composer.

```
$ composer require beatswitch/lock-laravel
```

Register the service provider in your `app.php` config file.

```
'BeatSwitch\Lock\Integrations\Laravel\LockServiceProvider',
```

Register the facades in your `app.php` config file.

```
'Lock' => 'BeatSwitch\Lock\Integrations\Laravel\Facades\Lock',
'LockManager' => 'BeatSwitch\Lock\Integrations\Laravel\Facades\LockManager',
```

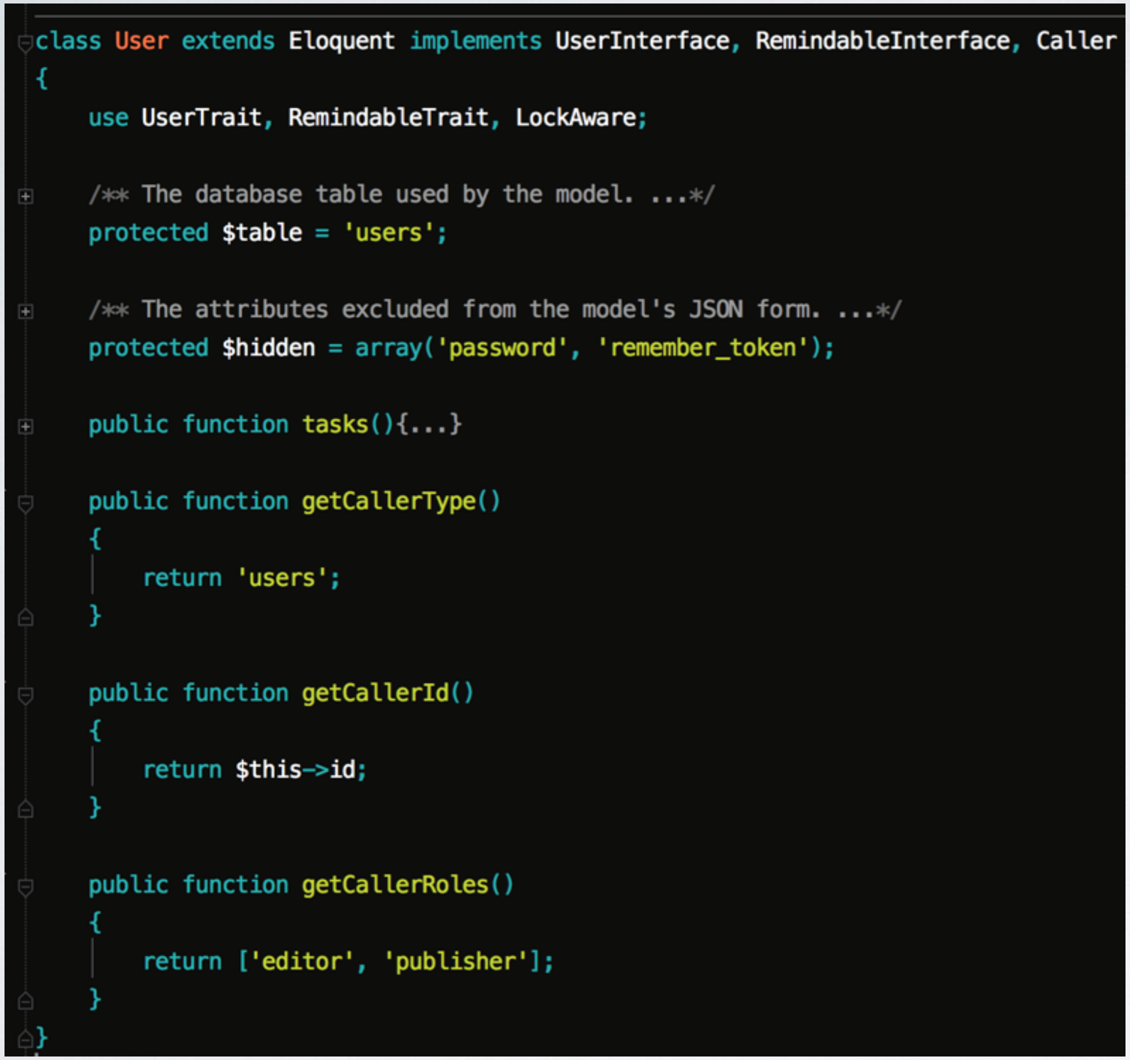
Publish the configuration file and edit the configuration options at `app/config/packages/beatswitch/lock-laravel/config.php`.

```
$ php artisan config:publish beatswitch/lock-laravel
```

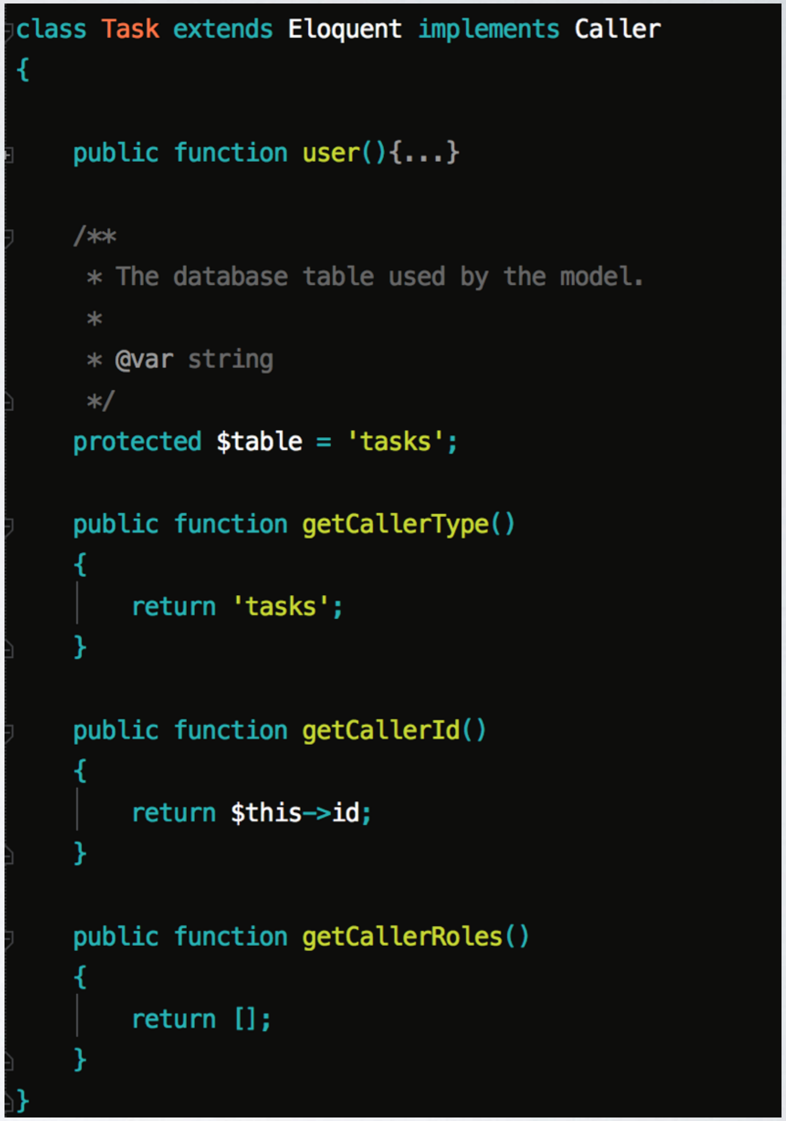
If you're using the database driver you should run the package's migrations. This will create the database table where all permissions will be stored.

```
$ php artisan migrate --package="beatswitch/lock-laravel"
```

ADD CALLER INTERFACES

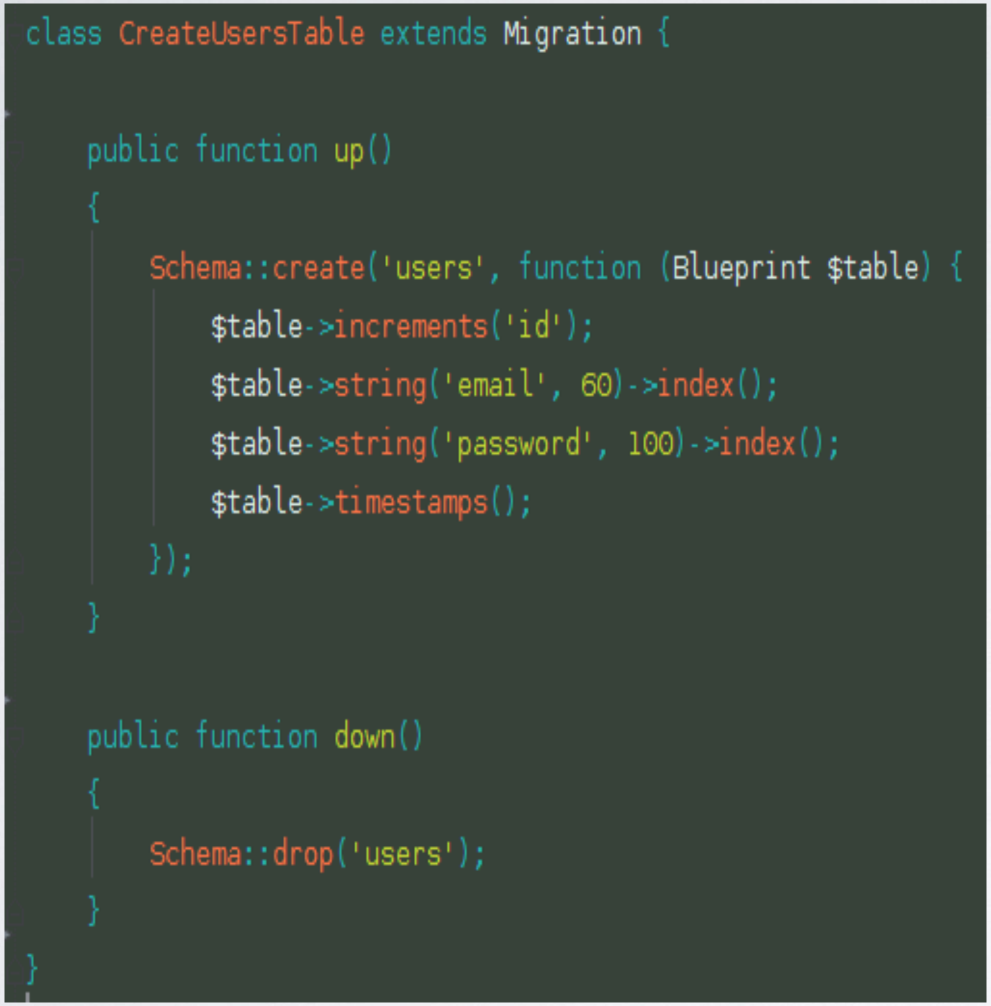


```php
class User extends Eloquent implements UserInterface, RemindableInterface, Caller
{
    use UserTrait, RemindableTrait, LockAware;

    /** The database table used by the model. ...*/
    protected $table = 'users';

    /** The attributes excluded from the model's JSON form. ...*/
    protected $hidden = array('password', 'remember_token');

    public function tasks(){...}

    public function getCallerType()
    {
        return 'users';
    }

    public function getCallerId()
    {
        return $this->id;
    }

    public function getCallerRoles()
    {
        return ['editor', 'publisher'];
    }
}
```

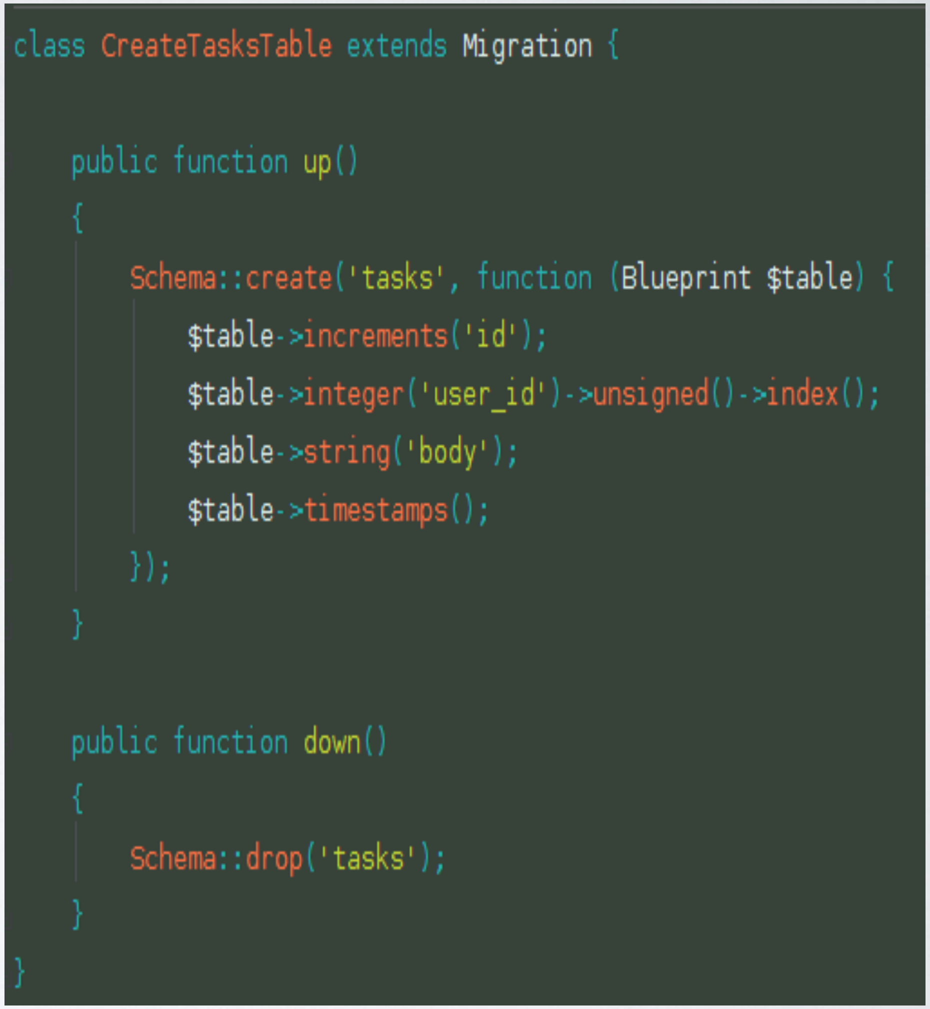


```php
class Task extends Eloquent implements Caller
{

    public function user(){...}

    /**
     * The database table used by the model.
     *
     * @var string
     */
    protected $table = 'tasks';

    public function getCallerType()
    {
        return 'tasks';
    }

    public function getCallerId()
    {
        return $this->id;
    }

    public function getCallerRoles()
    {
        return [];
    }
}
```

MIGRATIONS

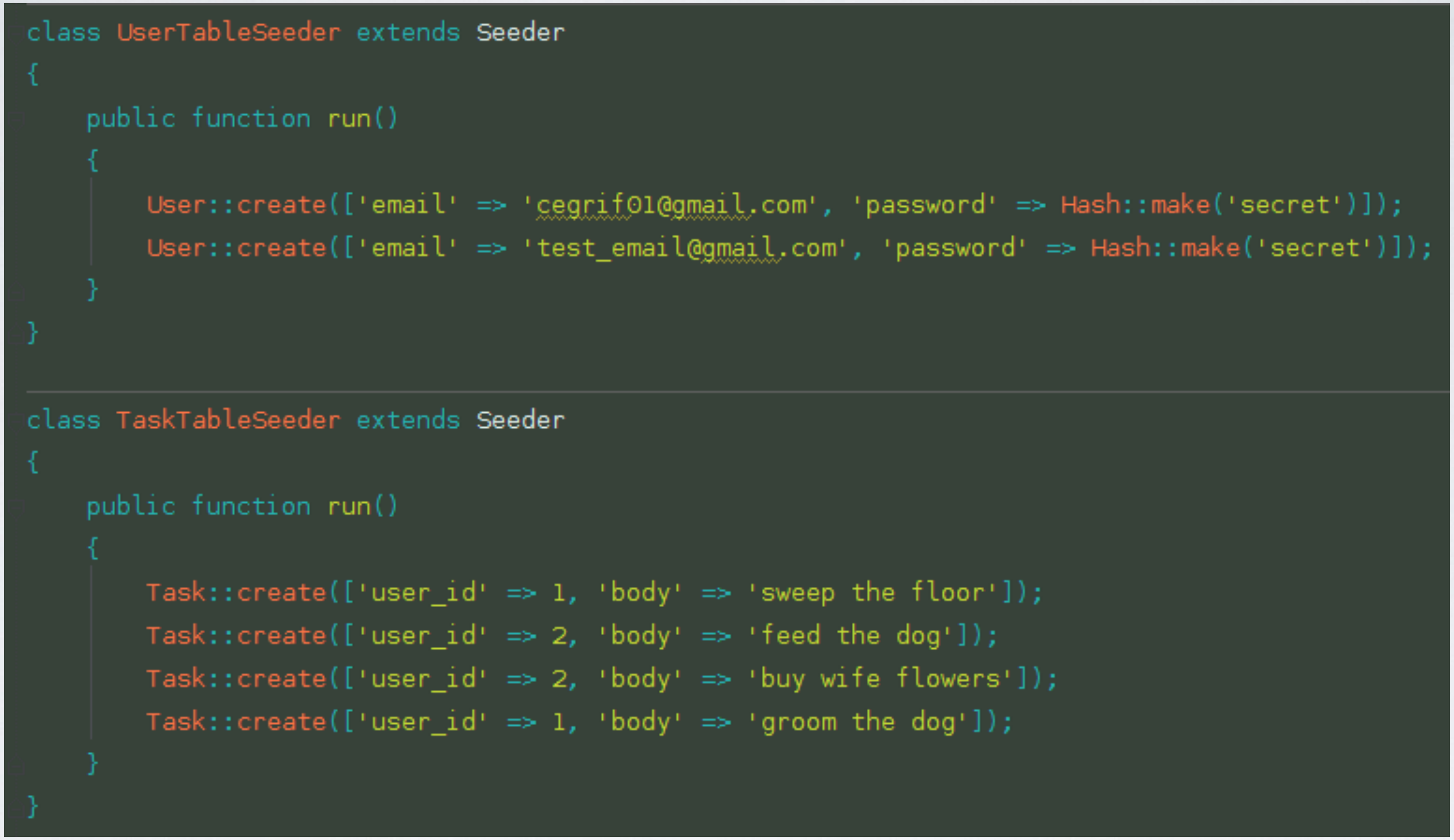


```php
class CreateUsersTable extends Migration {

    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->increments('id');
            $table->string('email', 60)->index();
            $table->string('password', 100)->index();
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::drop('users');
    }
}
```



```php
class CreateTasksTable extends Migration {

    public function up()
    {
        Schema::create('tasks', function (Blueprint $table) {
            $table->increments('id');
            $table->integer('user_id')->unsigned()->index();
            $table->string('body');
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::drop('tasks');
    }
}
```

SEEDING THE DATABASE



```
class UserTableSeeder extends Seeder
{
    public function run()
    {
        User::create(['email' => 'cegrif01@gmail.com', 'password' => Hash::make('secret')]);
        User::create(['email' => 'test_email@gmail.com', 'password' => Hash::make('secret')]);
    }
}
```

```
class TaskTableSeeder extends Seeder
{
    public function run()
    {
        Task::create(['user_id' => 1, 'body' => 'sweep the floor']);
        Task::create(['user_id' => 2, 'body' => 'feed the dog']);
        Task::create(['user_id' => 2, 'body' => 'buy wife flowers']);
        Task::create(['user_id' => 1, 'body' => 'groom the dog']);
    }
}
```

SEEDING THE DATABASE

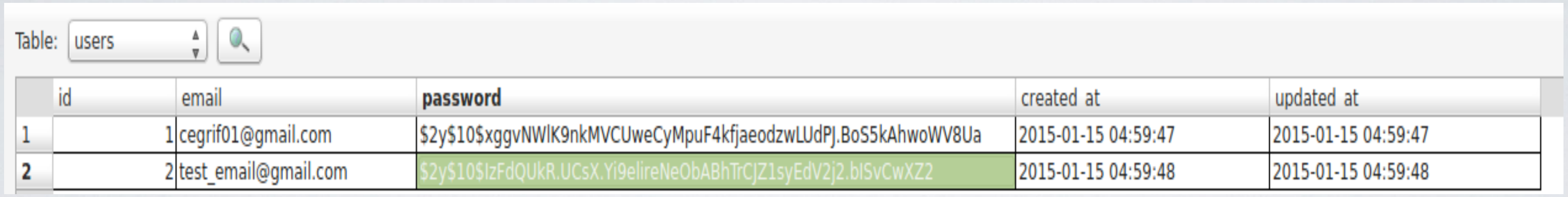


Table: users

	id	email	password	created at	updated at
1	1	cegrif01@gmail.com	$2y$10$xggvNWlK9nkMVCUweCyMpuF4kfjaeodzwLUdPJ.BoS5kAhwoWV8Ua	2015-01-15 04:59:47	2015-01-15 04:59:47
2	2	test_email@gmail.com	$2y$10$IzFdQUkR.UCsX.Yi9elireNeObABhTrCJZ1syEdV2j2.bISvCwXZ2	2015-01-15 04:59:48	2015-01-15 04:59:48

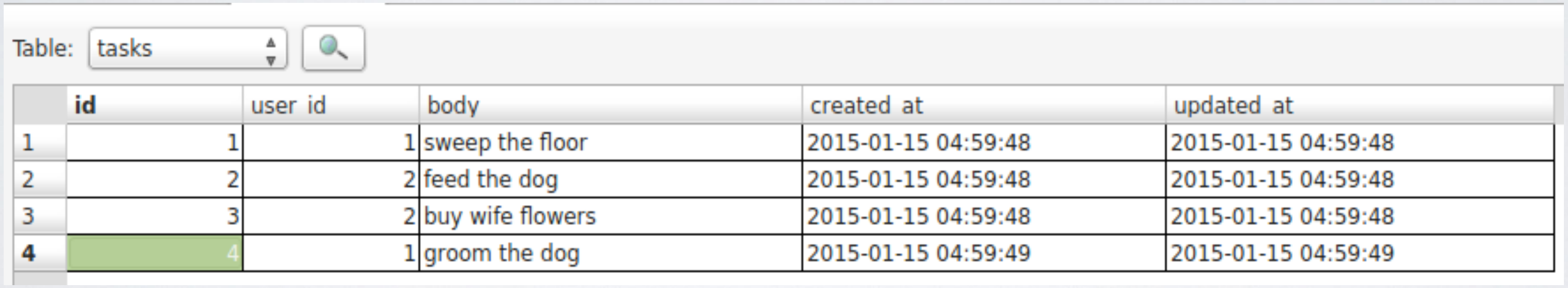
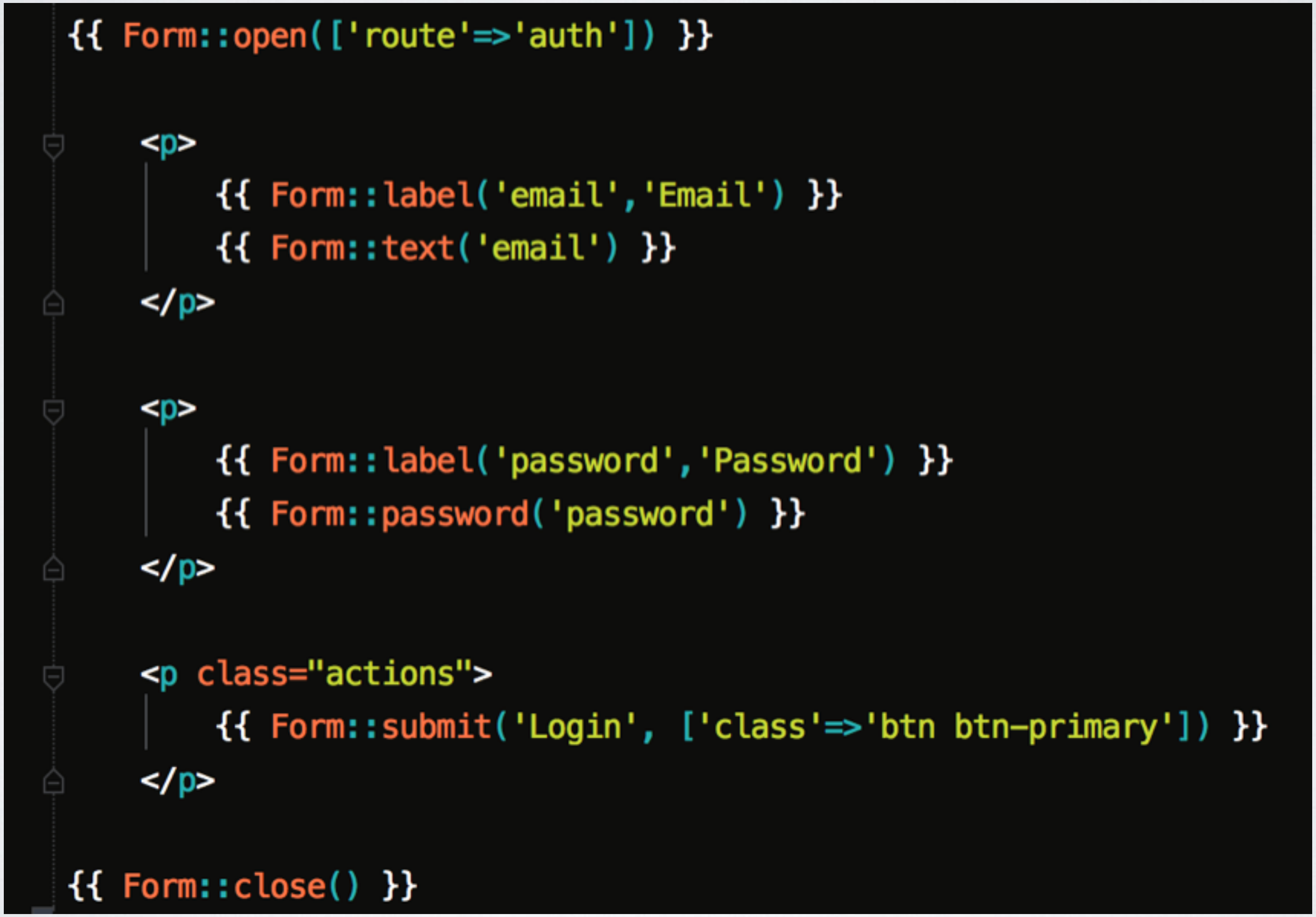


Table: tasks

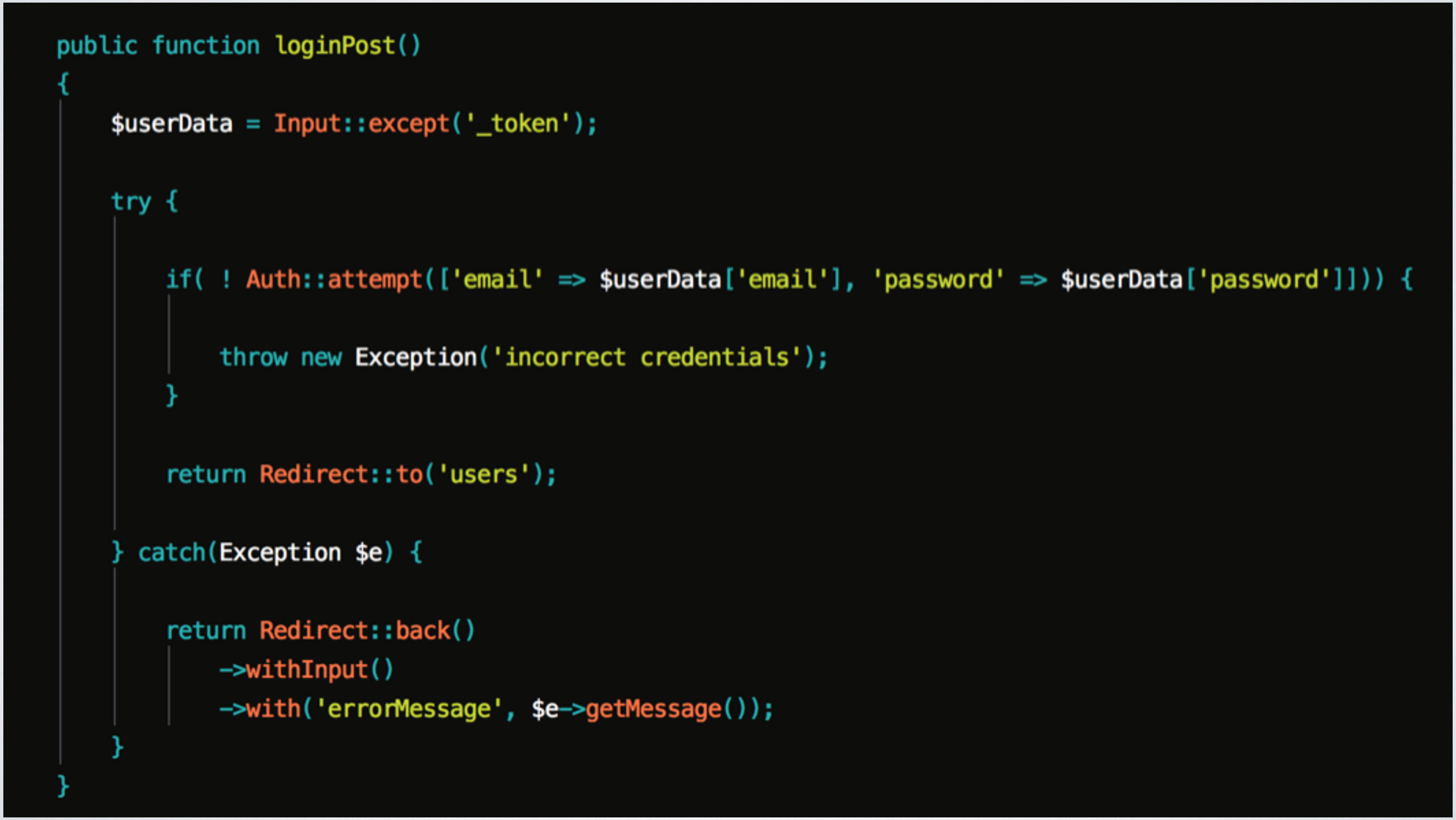
	id	user id	body	created at	updated at
1	1	1	sweep the floor	2015-01-15 04:59:48	2015-01-15 04:59:48
2	2	2	feed the dog	2015-01-15 04:59:48	2015-01-15 04:59:48
3	3	2	buy wife flowers	2015-01-15 04:59:48	2015-01-15 04:59:48
4	4	1	groom the dog	2015-01-15 04:59:49	2015-01-15 04:59:49

CREATE LOGIN FORM



```
{{ Form::open(['route'=>'auth']) }}

    <p>
        {{ Form::label('email','Email') }}
        {{ Form::text('email') }}
    </p>

    <p>
        {{ Form::label('password','Password') }}
        {{ Form::password('password') }}
    </p>

    <p class="actions">
        {{ Form::submit('Login', ['class'=>'btn btn-primary']) }}
    </p>

{{ Form::close() }}
```

SESSIONS CONTROLLER



```php
public function loginPost()
{
    $userData = Input::except('_token');

    try {

        if( ! Auth::attempt(['email' => $userData['email'], 'password' => $userData['password']])) {

            throw new Exception('incorrect credentials');
        }

        return Redirect::to('users');

    } catch(Exception $e) {

        return Redirect::back()
            ->withInput()
            ->with('errorMessage', $e->getMessage());
    }
}
```

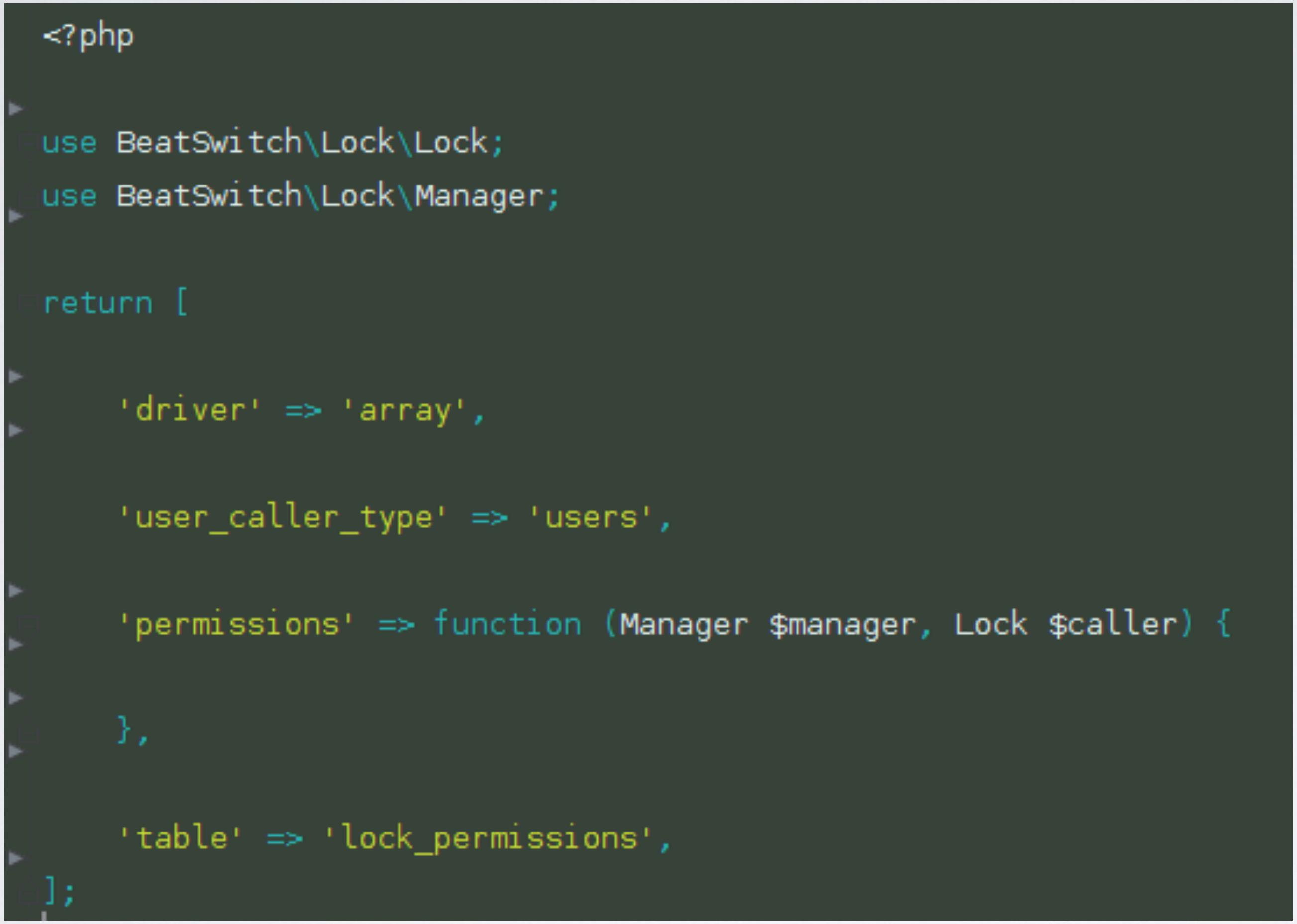
OUR INITIAL GOAL

- The first user (cegrif01@gmail.com) is an admin that can view all tasks.
- The second user (test_email@gmail.com) is a standard user that can only view their own tasks.

ARRAY DRIVER

- Default driver for Lock
- Don't spend too much time here because the database is what you really want
- The permissions key is a closure that sets up your roles and permissions in memory.
- On every request, the roles and permissions need to be recalculated according to what's in the permissions closure.

IN THE CONFIG

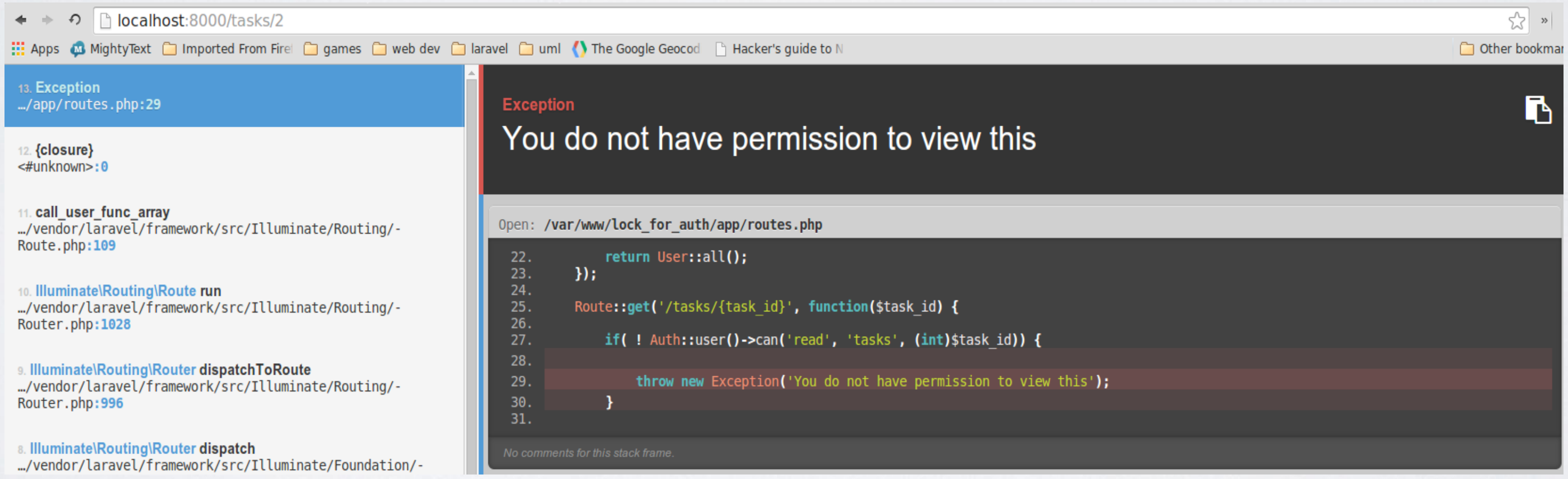


```php
<?php

use BeatSwitch\Lock\Lock;
use BeatSwitch\Lock\Manager;

return [

    'driver' => 'array',

    'user_caller_type' => 'users',

    'permissions' => function (Manager $manager, Lock $caller) {

    },

    'table' => 'lock_permissions',
];
```

*Non-commented version of Lock config

ALL PERMISSIONS DENIED BY DEFAULT

```
Route::get('/tasks/{task_id}', function($task_id) {

    if( ! Auth::user()->can('read', 'tasks', (int)$task_id)) {

        throw new Exception('You do not have permission to view this');
    }

    return Task::find($task_id);
});
```



13. Exception
…/app/routes.php:29

12. {closure}
<#unknown>:0

11. call_user_func_array
…/vendor/laravel/framework/src/Illuminate/Routing/-
Route.php:109

10. Illuminate\Routing\Route run
…/vendor/laravel/framework/src/Illuminate/Routing/-
Router.php:1028

9. Illuminate\Routing\Router dispatchToRoute
…/vendor/laravel/framework/src/Illuminate/Routing/-
Router.php:996

8. Illuminate\Routing\Router dispatch
…/vendor/laravel/framework/src/Illuminate/Foundation/-

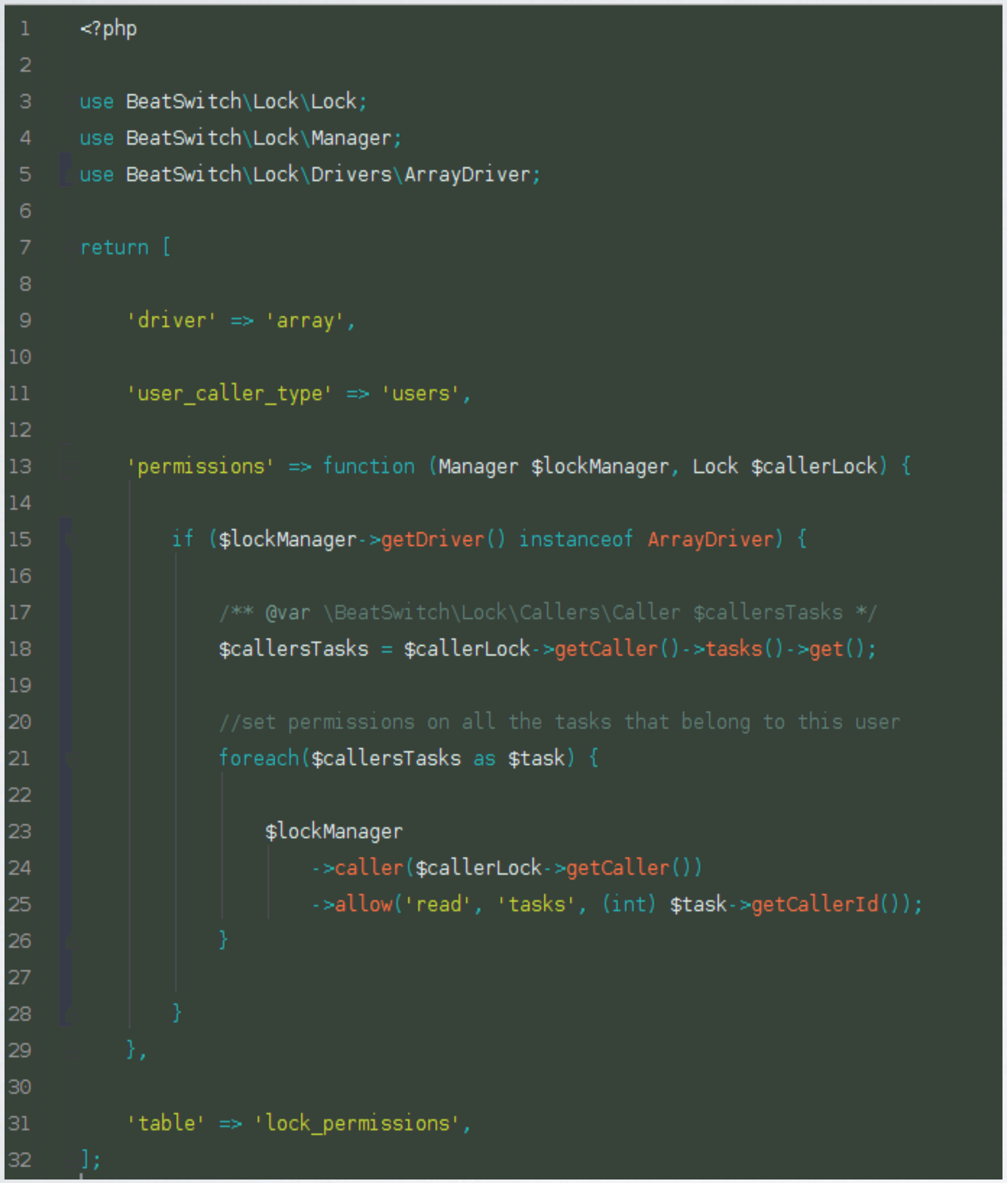
Exception
You do not have permission to view this

Open: /var/www/lock_for_auth/app/routes.php

```
        return User::all();
    });

    Route::get('/tasks/{task_id}', function($task_id) {

        if( ! Auth::user()->can('read', 'tasks', (int)$task_id)) {

            throw new Exception('You do not have permission to view this');
        }

```

No comments for this stack frame.

The closure gets ran every page load to make sure permissions are being applied



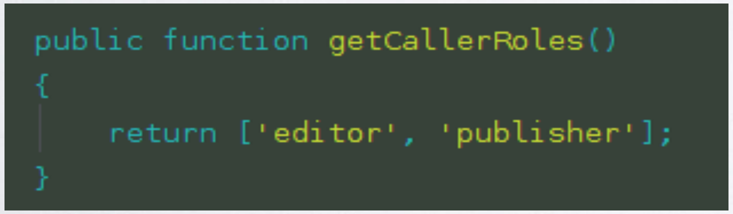
```php
<?php

use BeatSwitch\Lock\Lock;
use BeatSwitch\Lock\Manager;
use BeatSwitch\Lock\Drivers\ArrayDriver;

return [

    'driver' => 'array',

    'user_caller_type' => 'users',

    'permissions' => function (Manager $lockManager, Lock $callerLock) {

        if ($lockManager->getDriver() instanceof ArrayDriver) {

            /** @var \BeatSwitch\Lock\Callers\Caller $callersTasks */
            $callersTasks = $callerLock->getCaller()->tasks()->get();

            //set permissions on all the tasks that belong to this user
            foreach($callersTasks as $task) {

                $lockManager
                    ->caller($callerLock->getCaller())
                    ->allow('read', 'tasks', (int) $task->getCallerId());
            }

        }
    },

    'table' => 'lock_permissions',
];
```

*First param manages the driver and callers

* Second param getting passed in is the authenticated user

MY FINDINGS AFTER USING THE ARRAY DRIVER

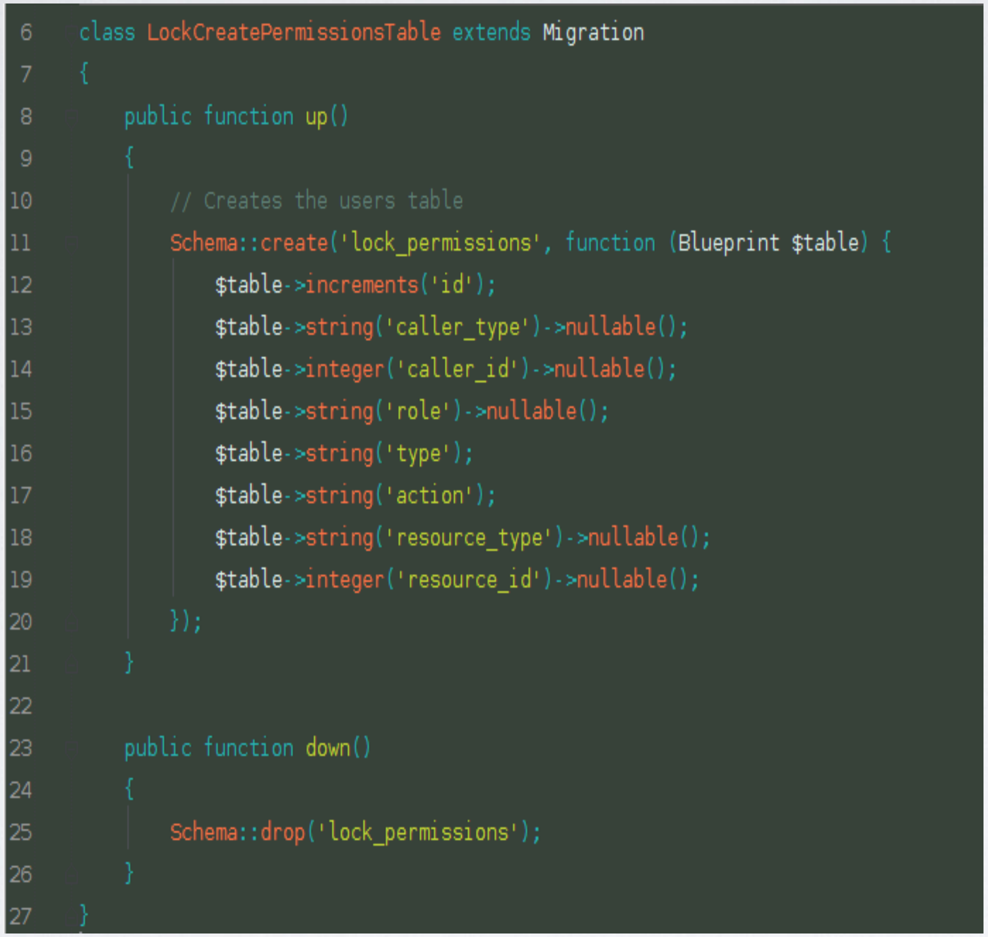
- Great for prototyping
- Not very optimal for production due to lack of persistence
- Roles are still unclear at this point
- There's no way to attach individual roles to each user at run time.



```
public function getCallerRoles()
{
    return ['editor', 'publisher'];
}
```

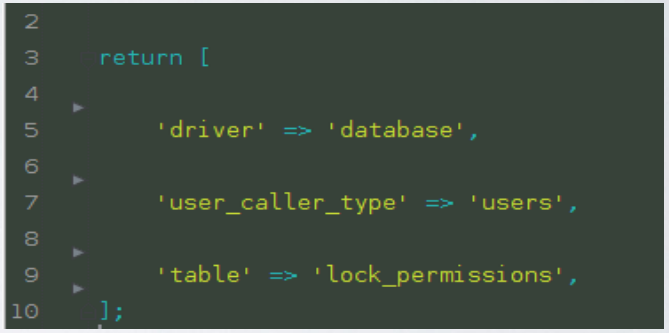
PERSISTENCE PLEASE

php artisan migrate --package="beatswitch/lock-laravel"



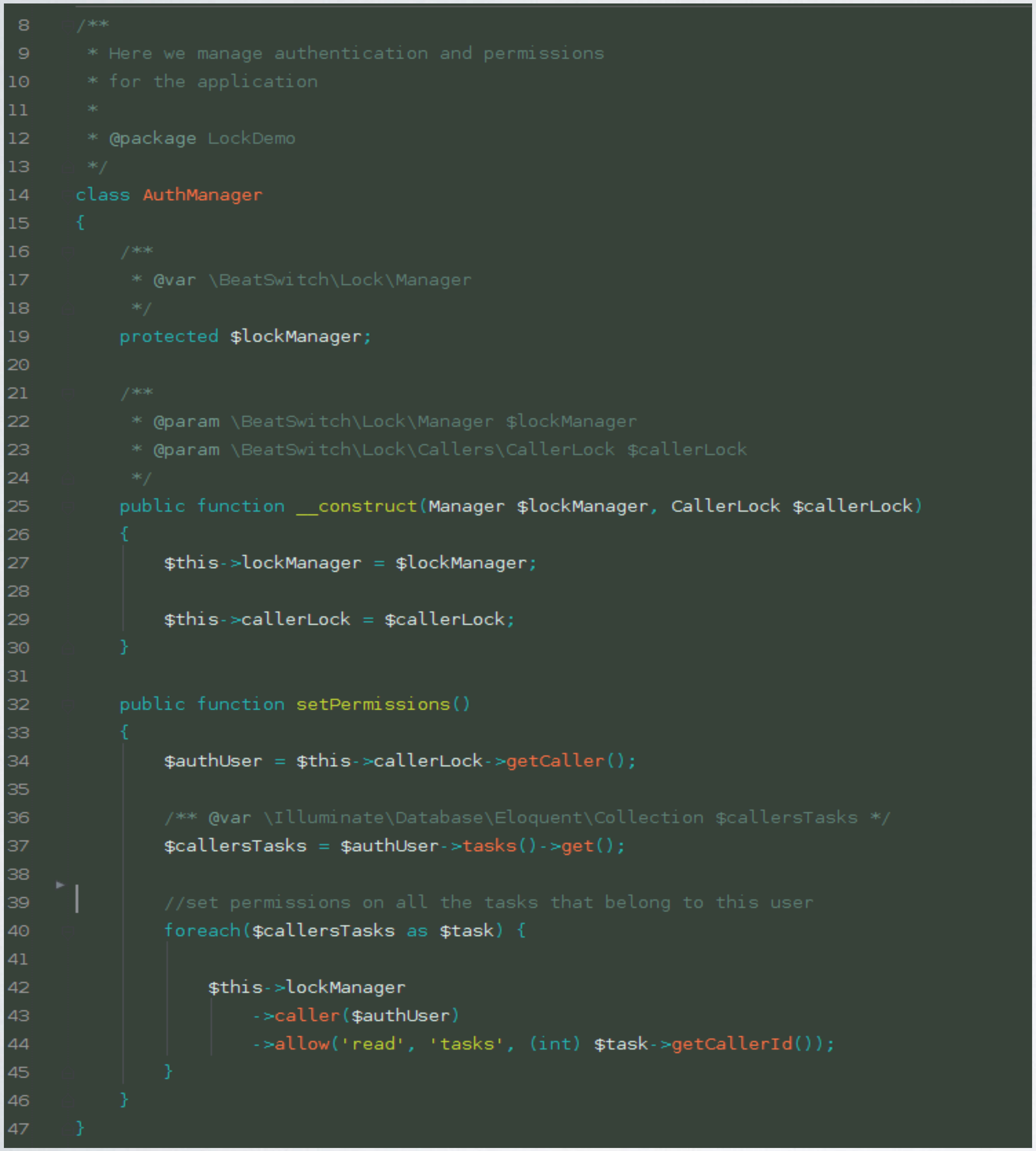
```
class LockCreatePermissionsTable extends Migration
{
    public function up()
    {
        // Creates the users table
        Schema::create('lock_permissions', function (Blueprint $table) {
            $table->increments('id');
            $table->string('caller_type')->nullable();
            $table->integer('caller_id')->nullable();
            $table->string('role')->nullable();
            $table->string('type');
            $table->string('action');
            $table->string('resource_type')->nullable();
            $table->integer('resource_id')->nullable();
        });
    }

    public function down()
    {
        Schema::drop('lock_permissions');
    }
}
```

*Be sure to remove permissions closure



```

return [

    'driver' => 'database',

    'user_caller_type' => 'users',

    'table' => 'lock_permissions',
];
```

SETTING PERMISSIONS

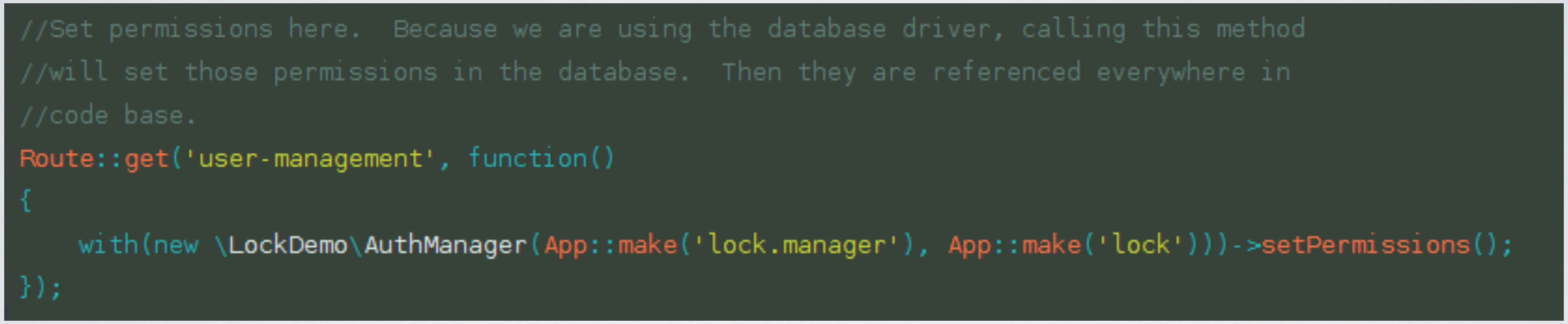


```php
/**
 * Here we manage authentication and permissions
 * for the application
 *
 * @package LockDemo
 */
class AuthManager
{
    /**
     * @var \BeatSwitch\Lock\Manager
     */
    protected $lockManager;

    /**
     * @param \BeatSwitch\Lock\Manager $lockManager
     * @param \BeatSwitch\Lock\Callers\CallerLock $callerLock
     */
    public function __construct(Manager $lockManager, CallerLock $callerLock)
    {
        $this->lockManager = $lockManager;

        $this->callerLock = $callerLock;
    }

    public function setPermissions()
    {
        $authUser = $this->callerLock->getCaller();

        /** @var \Illuminate\Database\Eloquent\Collection $callersTasks */
        $callersTasks = $authUser->tasks()->get();

        //set permissions on all the tasks that belong to this user
        foreach($callersTasks as $task) {

            $this->lockManager
                ->caller($authUser)
                ->allow('read', 'tasks', (int) $task->getCallerId());
        }
    }
}
```

Lock knows who the currently logged in user is due to it's service provider. More on this later.

Since Task implements caller interface, we can grab it's id

Apply permissions



```
//Set permissions here.  Because we are using the database driver, calling this method
//will set those permissions in the database.  Then they are referenced everywhere in
//code base.
Route::get('user-management', function()
{
    with(new \LockDemo\AuthManager(App::make('lock.manager'), App::make('lock')))->setPermissions();
});
```

Verify correct permissions

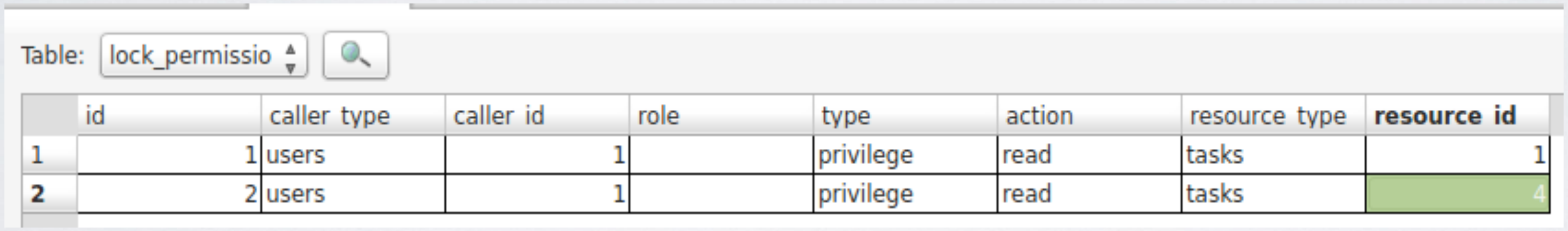
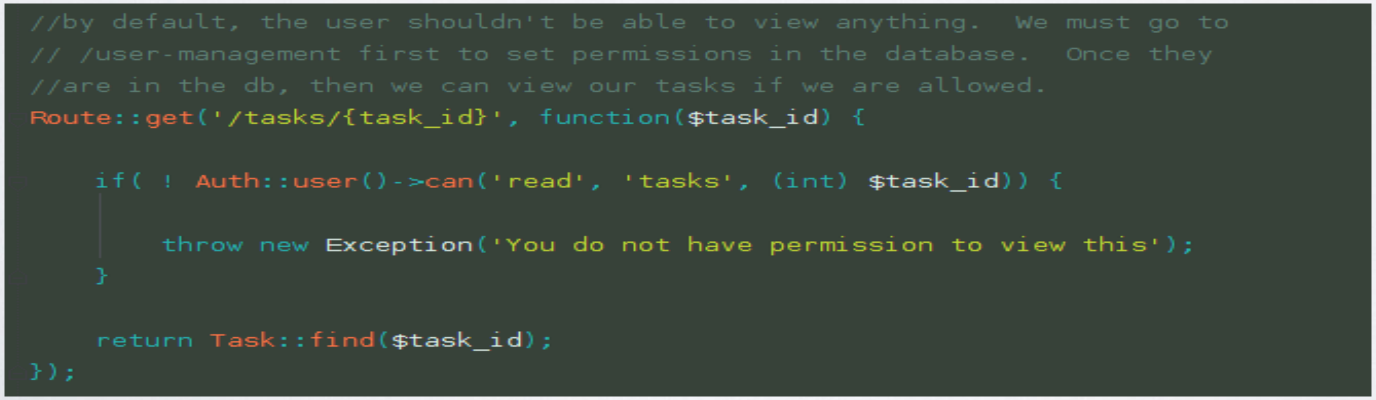


Table: lock_permissio

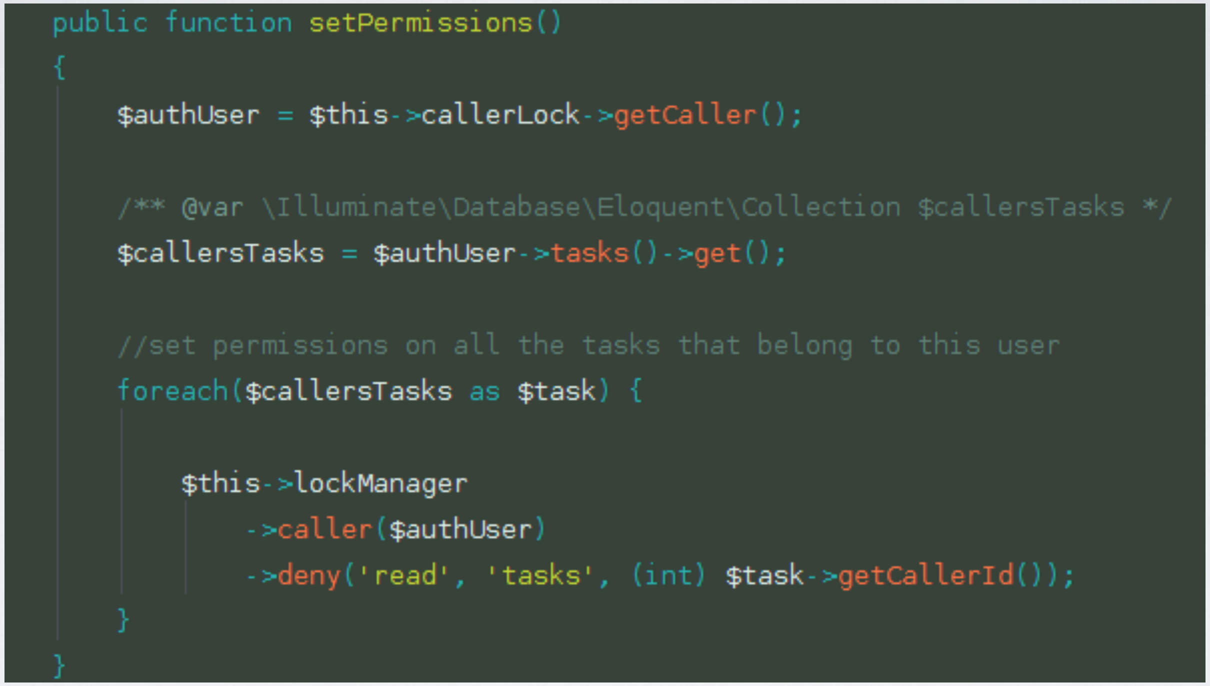
	id	caller type	caller id	role	type	action	resource type	resource id
1	1	users	1		privilege	read	tasks	1
2	2	users	1		privilege	read	tasks	4

Enforce permissions



```
//by default, the user shouldn't be able to view anything.  We must go to
// /user-management first to set permissions in the database.  Once they
//are in the db, then we can view our tasks if we are allowed.
Route::get('/tasks/{task_id}', function($task_id) {

    if( ! Auth::user()->can('read', 'tasks', (int) $task_id)) {

        throw new Exception('You do not have permission to view this');
    }

    return Task::find($task_id);
});
```

PRIVILEGE AND RESTRICTION



```php
public function setPermissions()
{
    $authUser = $this->callerLock->getCaller();

    /** @var \Illuminate\Database\Eloquent\Collection $callersTasks */
    $callersTasks = $authUser->tasks()->get();

    //set permissions on all the tasks that belong to this user
    foreach($callersTasks as $task) {

        $this->lockManager
            ->caller($authUser)
            ->deny('read', 'tasks', (int) $task->getCallerId());
    }
}
```

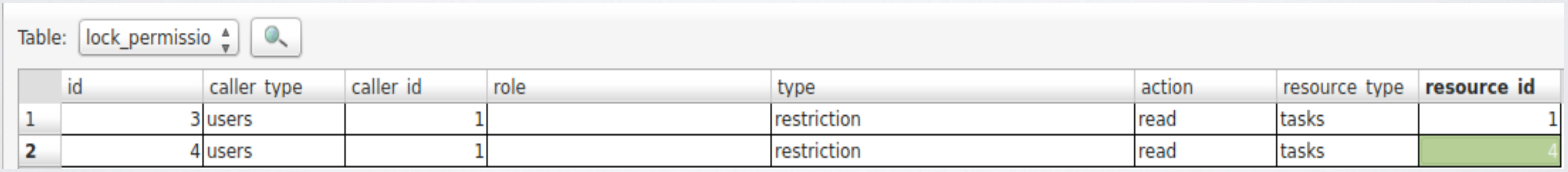


Table: lock_permissio

	id	caller type	caller id	role	type	action	resource type	resource id
1	3	users	1		restriction	read	tasks	1
2	4	users	1		restriction	read	tasks	4

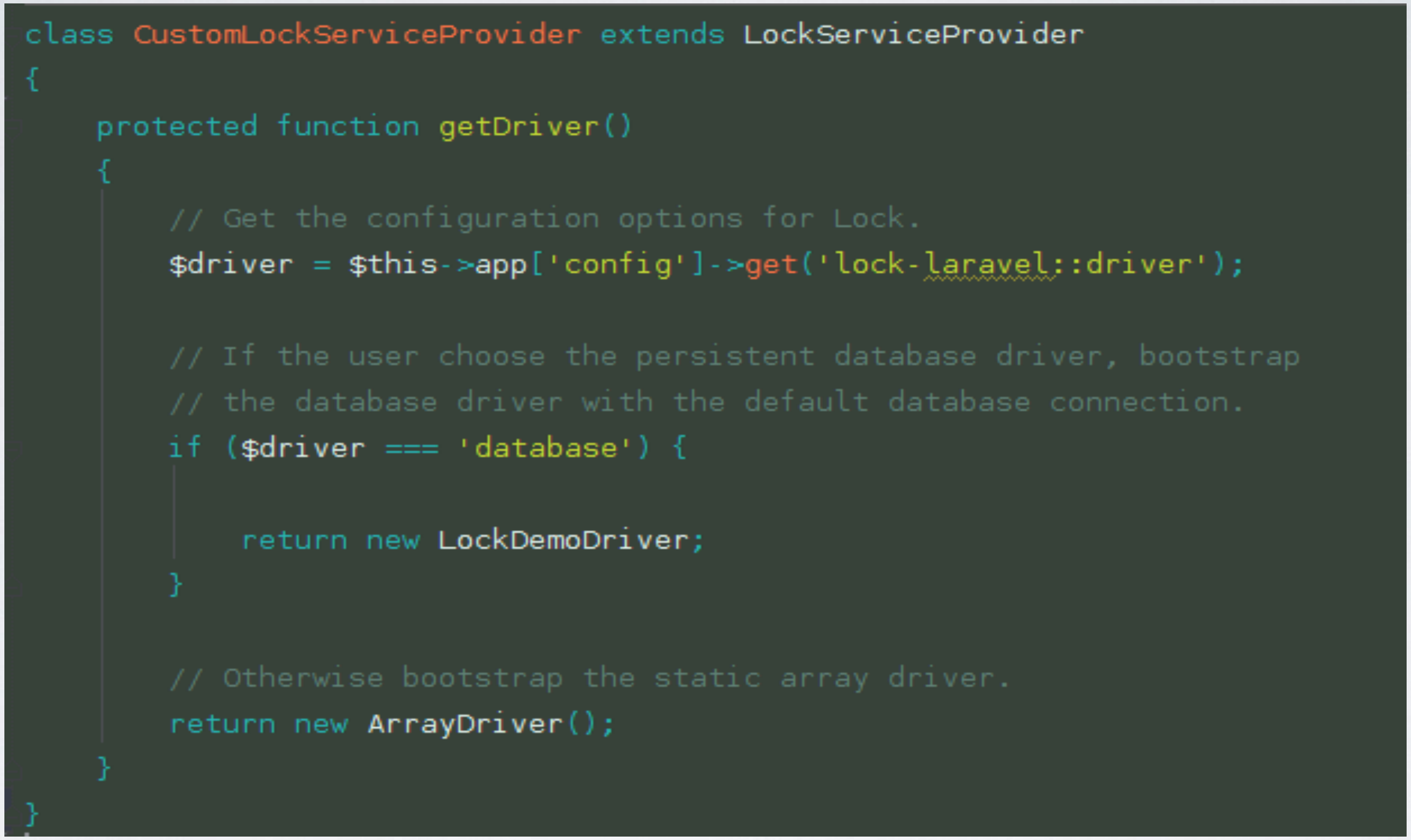
MY FINDINGS AFTER DEFAULT DB

- Roles still didn't make sense because all the examples show hard coded getCallerRoles() return.
- Can't attach roles to individual users
- Out of the box, Lock fails to accomplish my goals :(
- At this point I'm wondering if I can create my own schema, own driver, and make roles available on a per user basis

THE OPEN/CLOSED PRINCIPLE

- Adherence to this principle is really redeeming
- Let's look at how the DatabaseDriver and Lock classes talk to each other
- We can change our implementation of the Database Driver and Lock doesn't care.
- At this point, I'm excited again.

OVERRIDING THE LOCK SERVICE PROVIDER

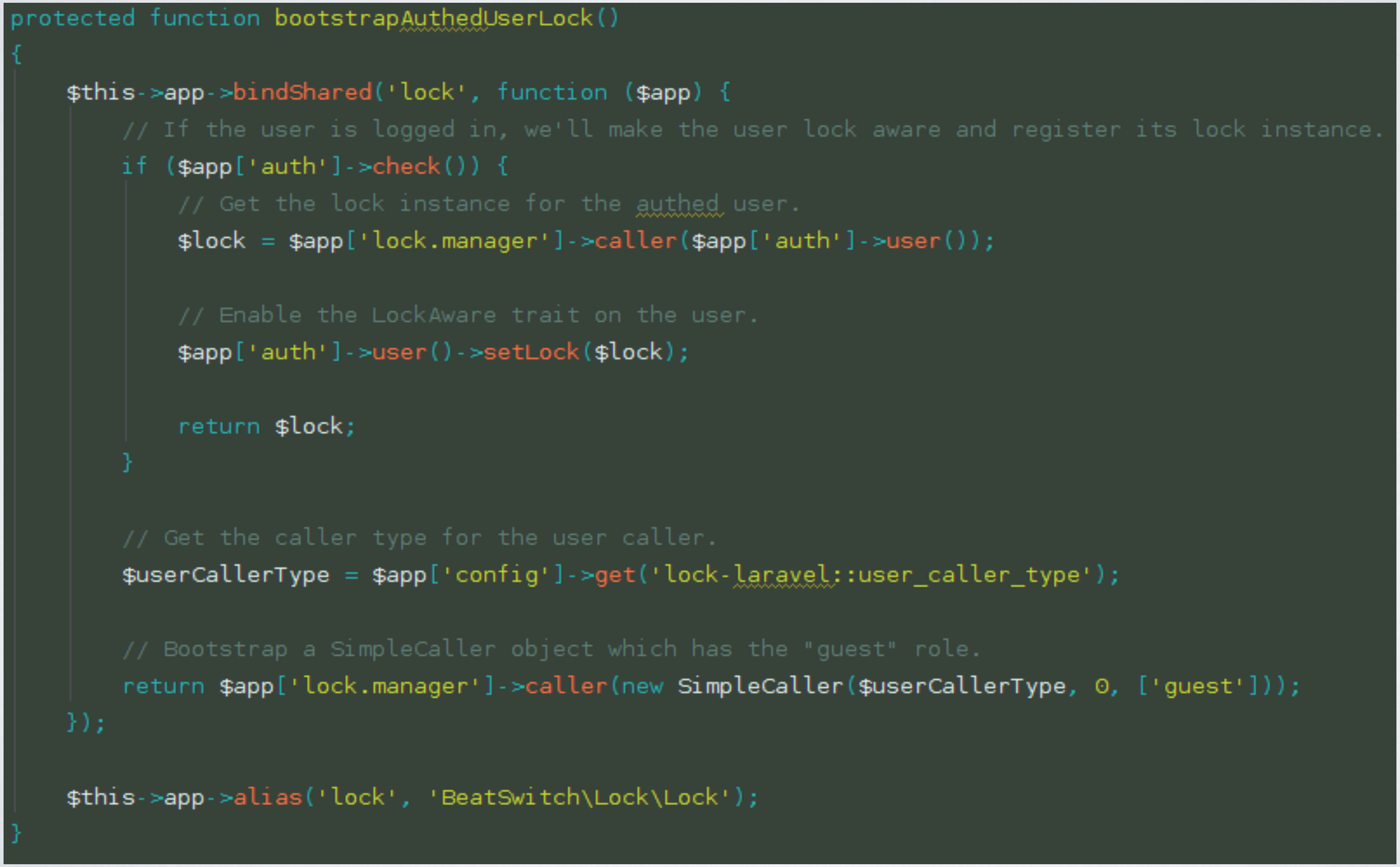


```php
class CustomLockServiceProvider extends LockServiceProvider
{
    protected function getDriver()
    {
        // Get the configuration options for Lock.
        $driver = $this->app['config']->get('lock-laravel::driver');

        // If the user choose the persistent database driver, bootstrap
        // the database driver with the default database connection.
        if ($driver === 'database') {

            return new LockDemoDriver;
        }

        // Otherwise bootstrap the static array driver.
        return new ArrayDriver();
    }
}
```

*Don't forget to switch out service providers in app.php providers array

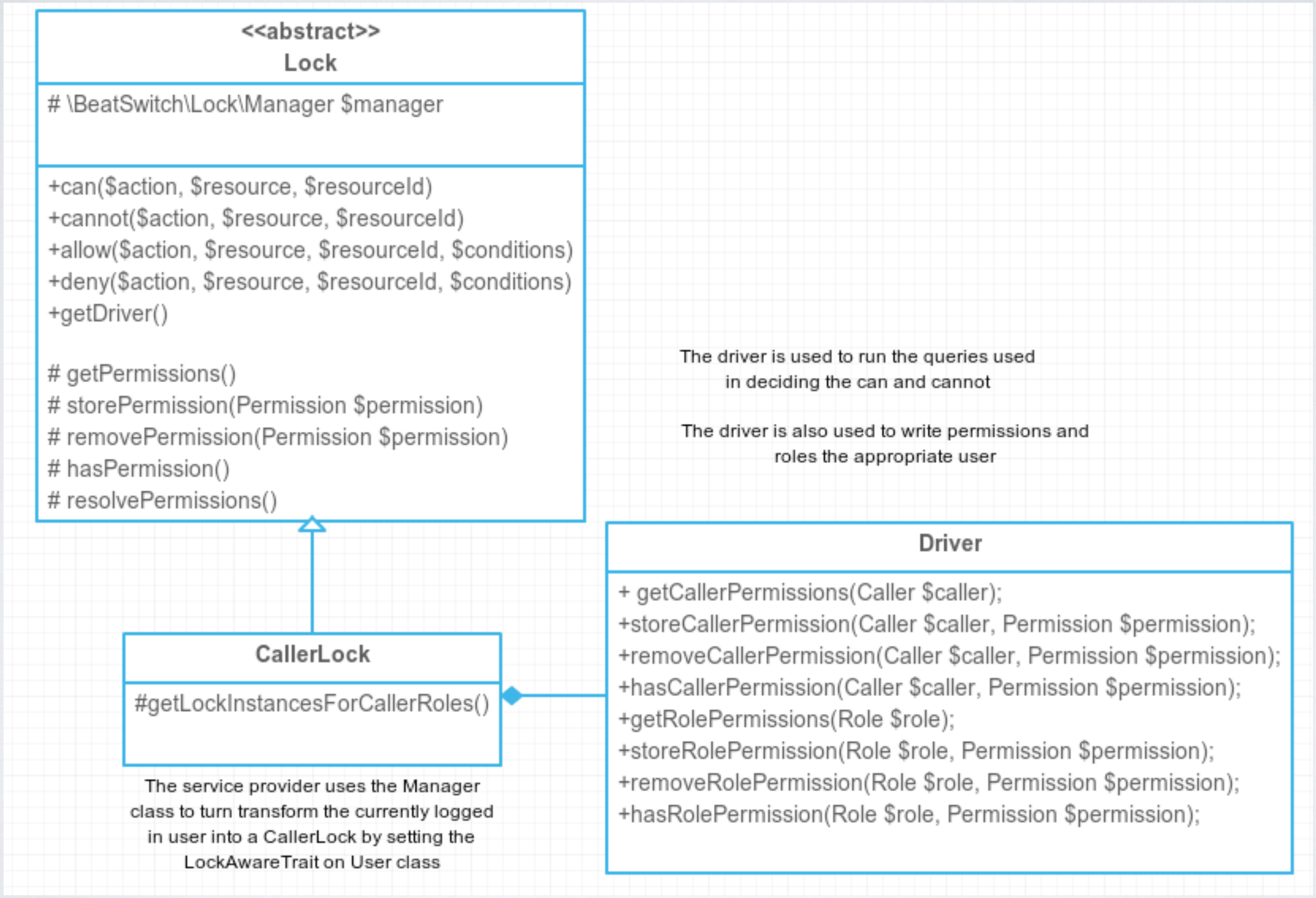
LOGGED IN USER GETS SPECIAL TRAIT

Allows for Auth::user()->can() and Auth::user()->cannot()



```php
protected function bootstrapAuthedUserLock()
{
    $this->app->bindShared('lock', function ($app) {
        // If the user is logged in, we'll make the user lock aware and register its lock instance.
        if ($app['auth']->check()) {
            // Get the lock instance for the authed user.
            $lock = $app['lock.manager']->caller($app['auth']->user());

            // Enable the LockAware trait on the user.
            $app['auth']->user()->setLock($lock);

            return $lock;
        }

        // Get the caller type for the user caller.
        $userCallerType = $app['config']->get('lock-laravel::user_caller_type');

        // Bootstrap a SimpleCaller object which has the "guest" role.
        return $app['lock.manager']->caller(new SimpleCaller($userCallerType, 0, ['guest']));
    });

    $this->app->alias('lock', 'BeatSwitch\Lock\Lock');
}
```

LET'S LOOK AT THE DRIVER

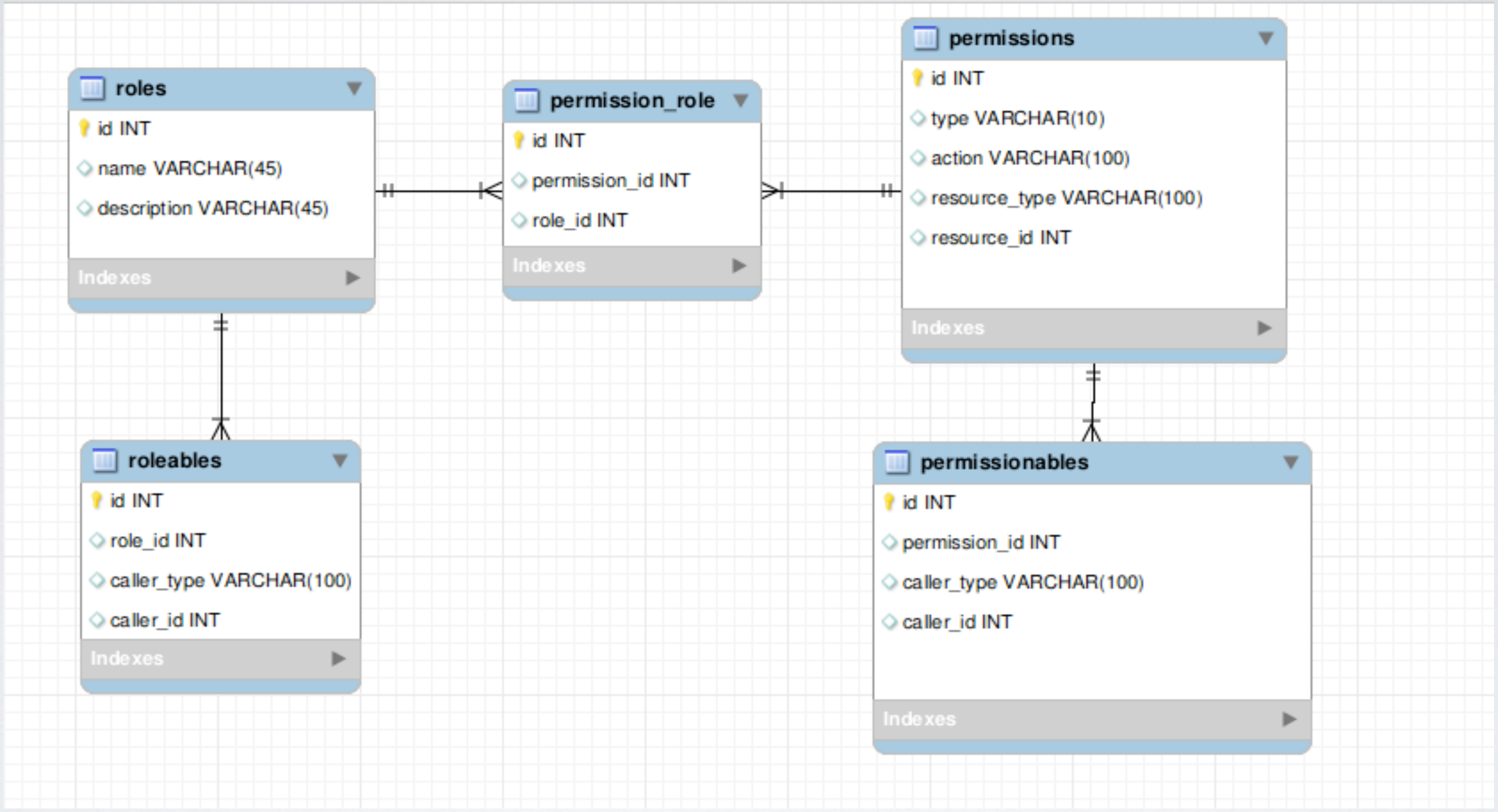

<<abstract>>
Lock
# \BeatSwitch\Lock\Manager $manager
+can($action, $resource, $resourceId)
+cannot($action, $resource, $resourceId)
+allow($action, $resource, $resourceId, $conditions)
+deny($action, $resource, $resourceId, $conditions)
+getDriver()
# getPermissions()
# storePermission(Permission $permission)
# removePermission(Permission $permission)
# hasPermission()
# resolvePermissions()
CallerLock
#getLockInstancesForCallerRoles()
Driver
+ getCallerPermissions(Caller $caller);
+storeCallerPermission(Caller $caller, Permission $permission);
+removeCallerPermission(Caller $caller, Permission $permission);
+hasCallerPermission(Caller $caller, Permission $permission);
+getRolePermissions(Role $role);
+storeRolePermission(Role $role, Permission $permission);
+removeRolePermission(Role $role, Permission $permission);
+hasRolePermission(Role $role, Permission $permission);

The driver is used to run the queries used in deciding the can and cannot

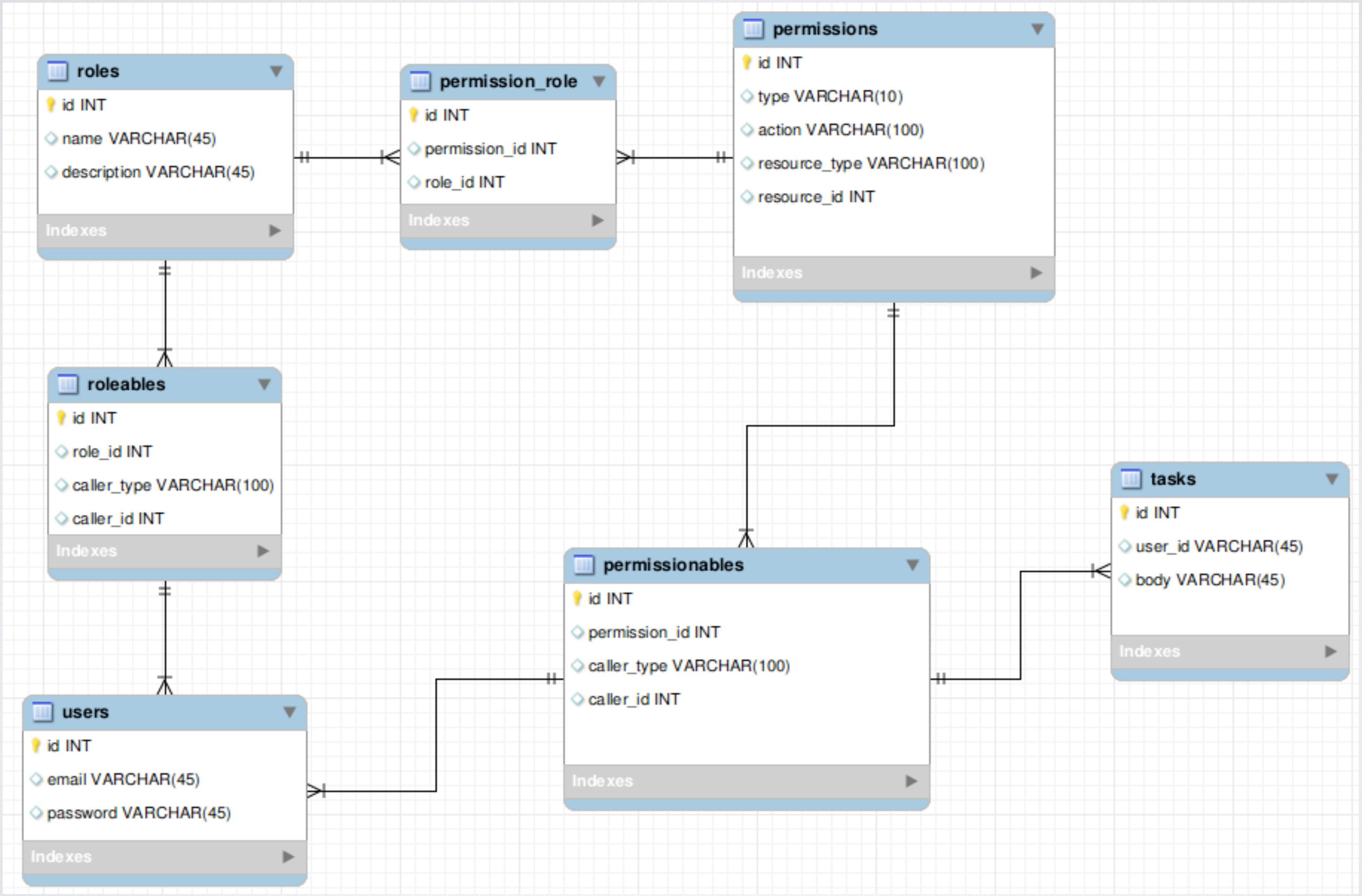
The driver is also used to write permissions and roles the appropriate user

The service provider uses the Manager class to turn transform the currently logged in user into a CallerLock by setting the LockAwareTrait on User class

BASE ERD DIAGRAM

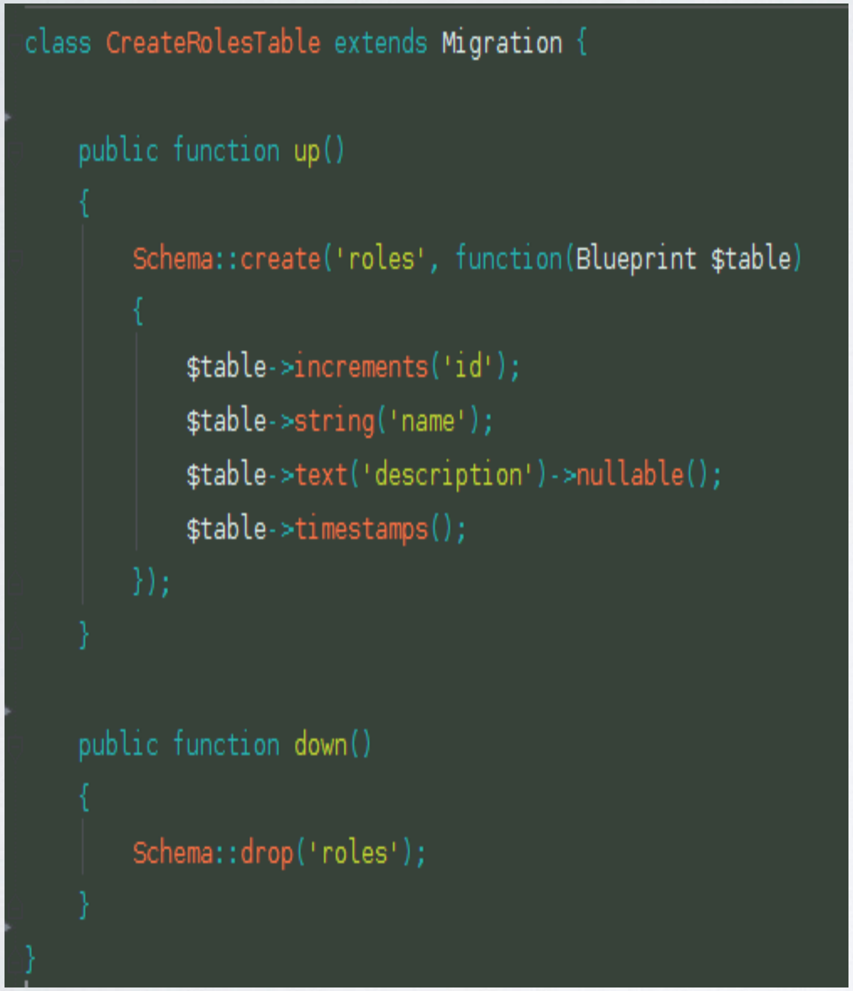


ATTACHING CALLERS TO OUR BASE

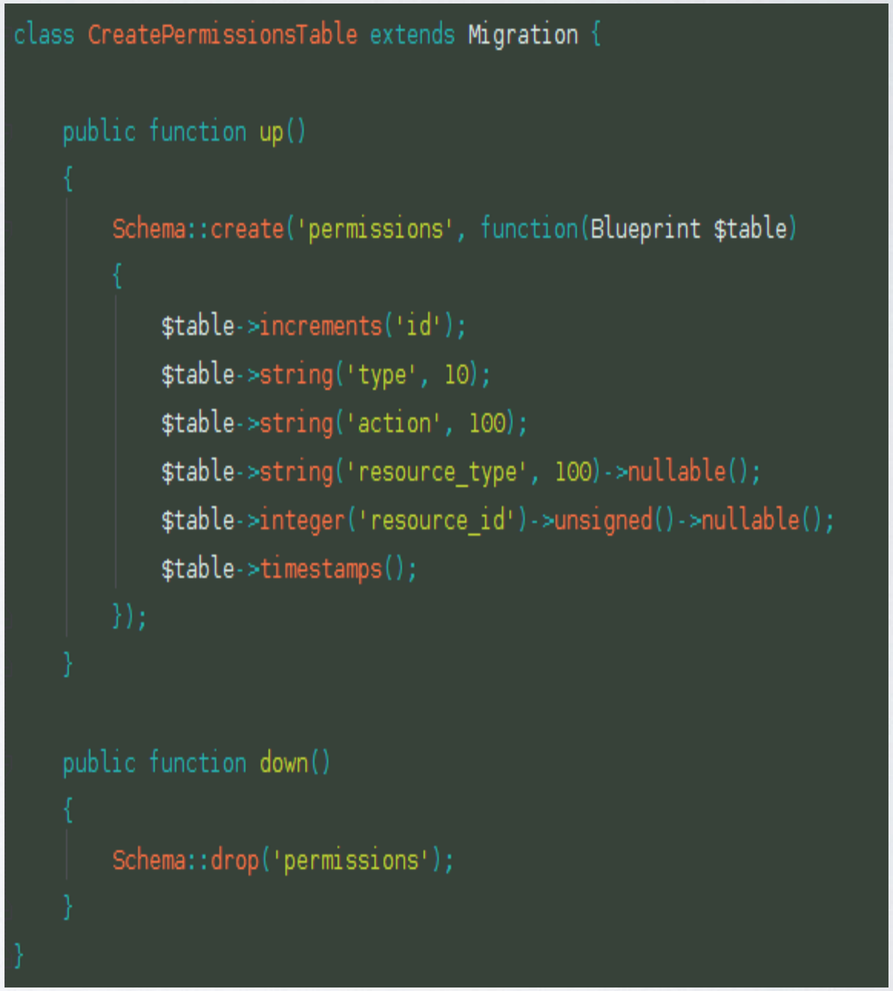


LET'S DO ROLES AND PERMISSIONS BETTER

Let's completely scrap the lock_permissions table and create our own design

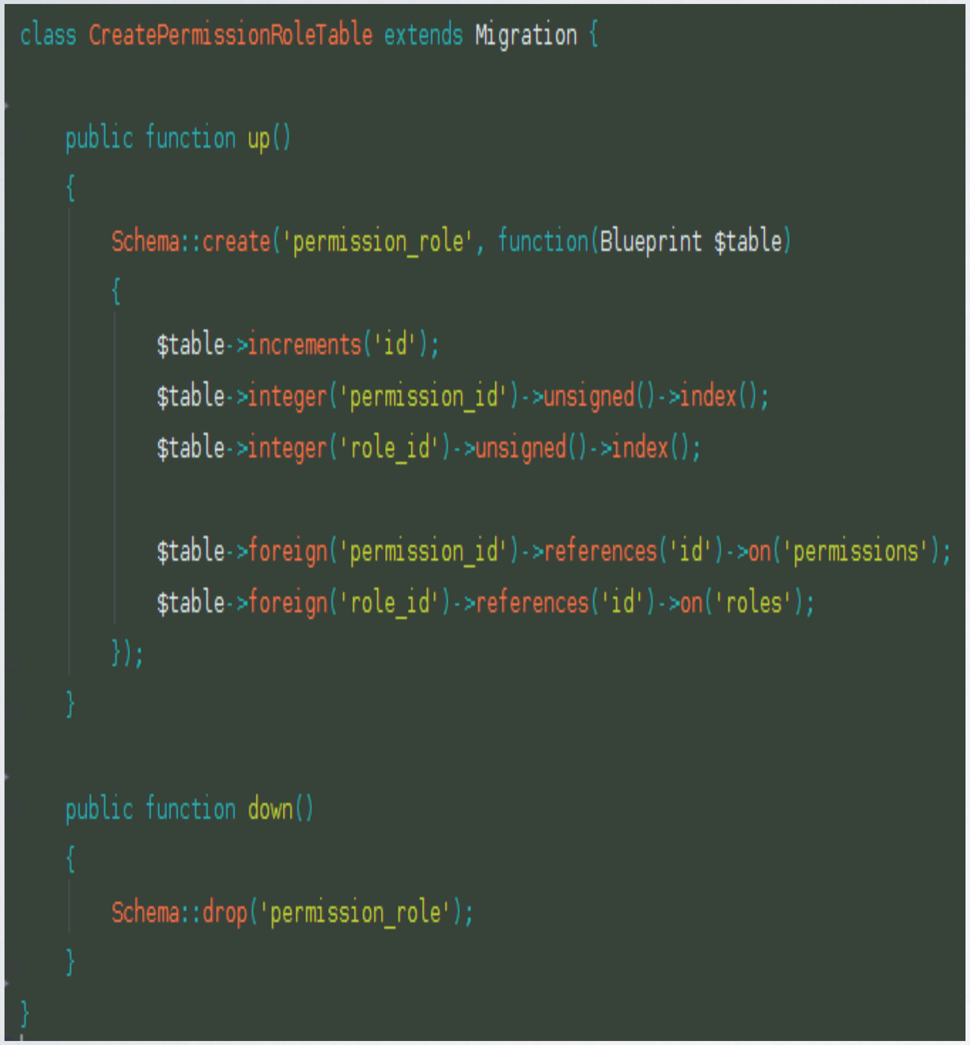


```php
class CreateRolesTable extends Migration {

    public function up()
    {
        Schema::create('roles', function(Blueprint $table)
        {
            $table->increments('id');
            $table->string('name');
            $table->text('description')->nullable();
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::drop('roles');
    }
}
```

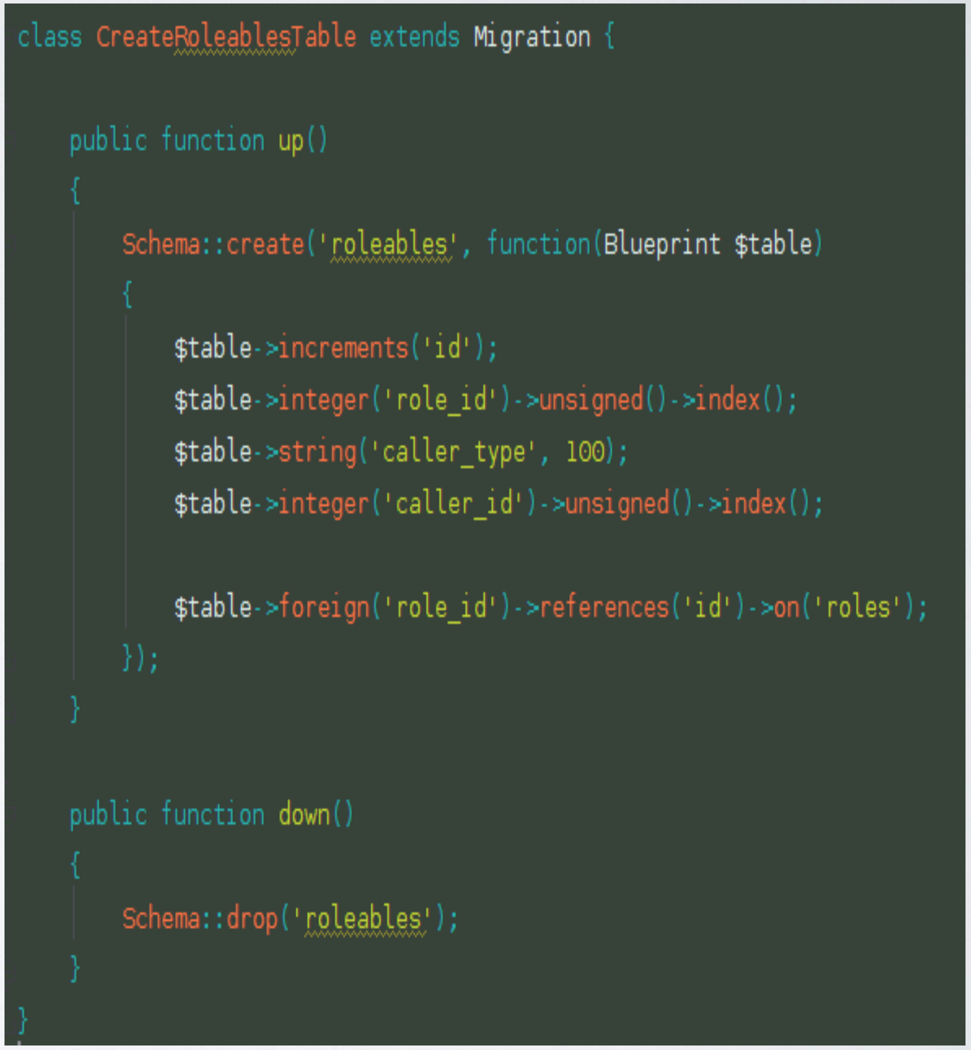


```php
class CreatePermissionsTable extends Migration {

    public function up()
    {
        Schema::create('permissions', function(Blueprint $table)
        {
            $table->increments('id');
            $table->string('type', 10);
            $table->string('action', 100);
            $table->string('resource_type', 100)->nullable();
            $table->integer('resource_id')->unsigned()->nullable();
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::drop('permissions');
    }
}
```

PERMISSIONS AND ROLEABLES

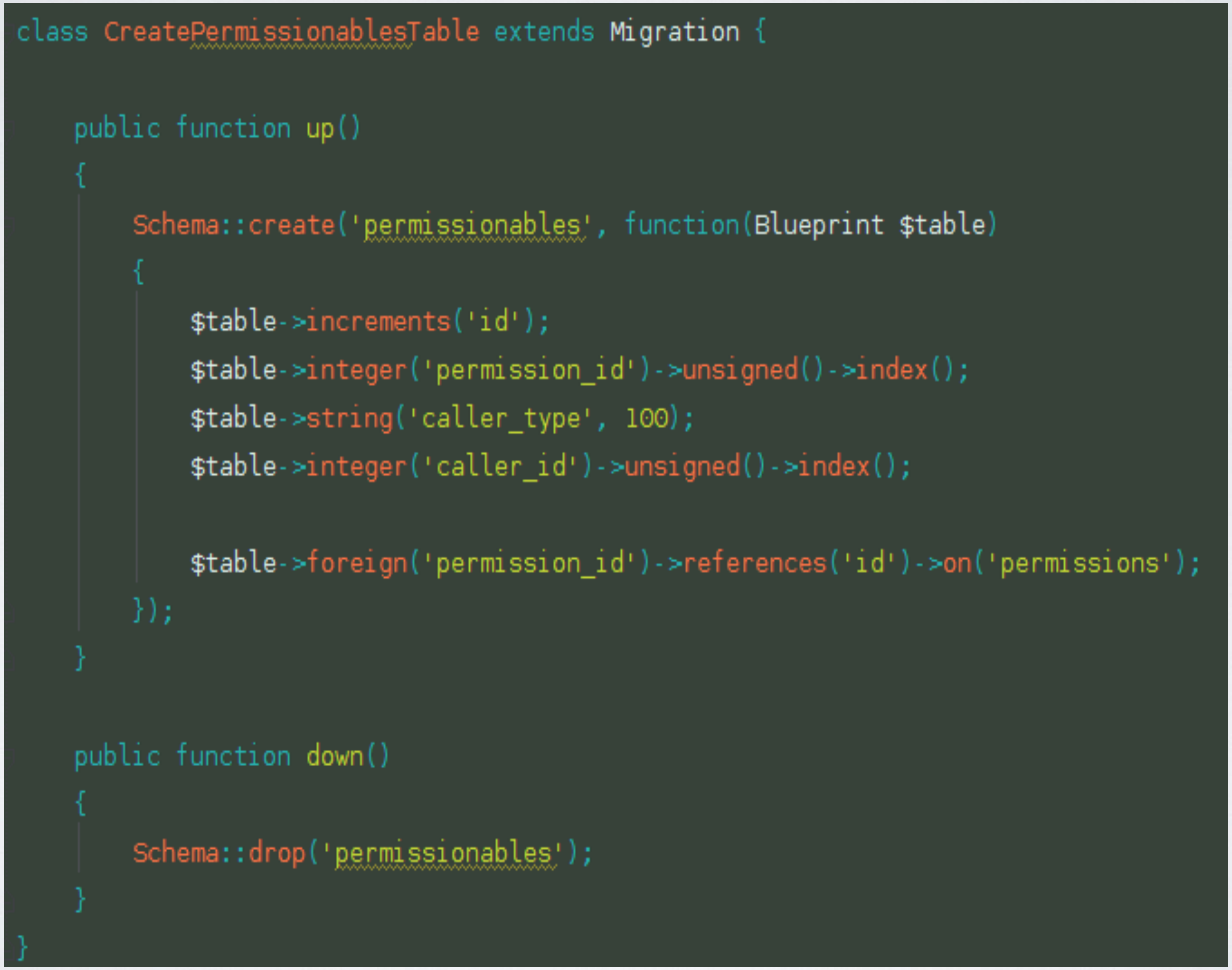


```php
class CreatePermissionRoleTable extends Migration {

    public function up()
    {
        Schema::create('permission_role', function(Blueprint $table)
        {
            $table->increments('id');
            $table->integer('permission_id')->unsigned()->index();
            $table->integer('role_id')->unsigned()->index();

            $table->foreign('permission_id')->references('id')->on('permissions');
            $table->foreign('role_id')->references('id')->on('roles');
        });
    }

    public function down()
    {
        Schema::drop('permission_role');
    }
}
```



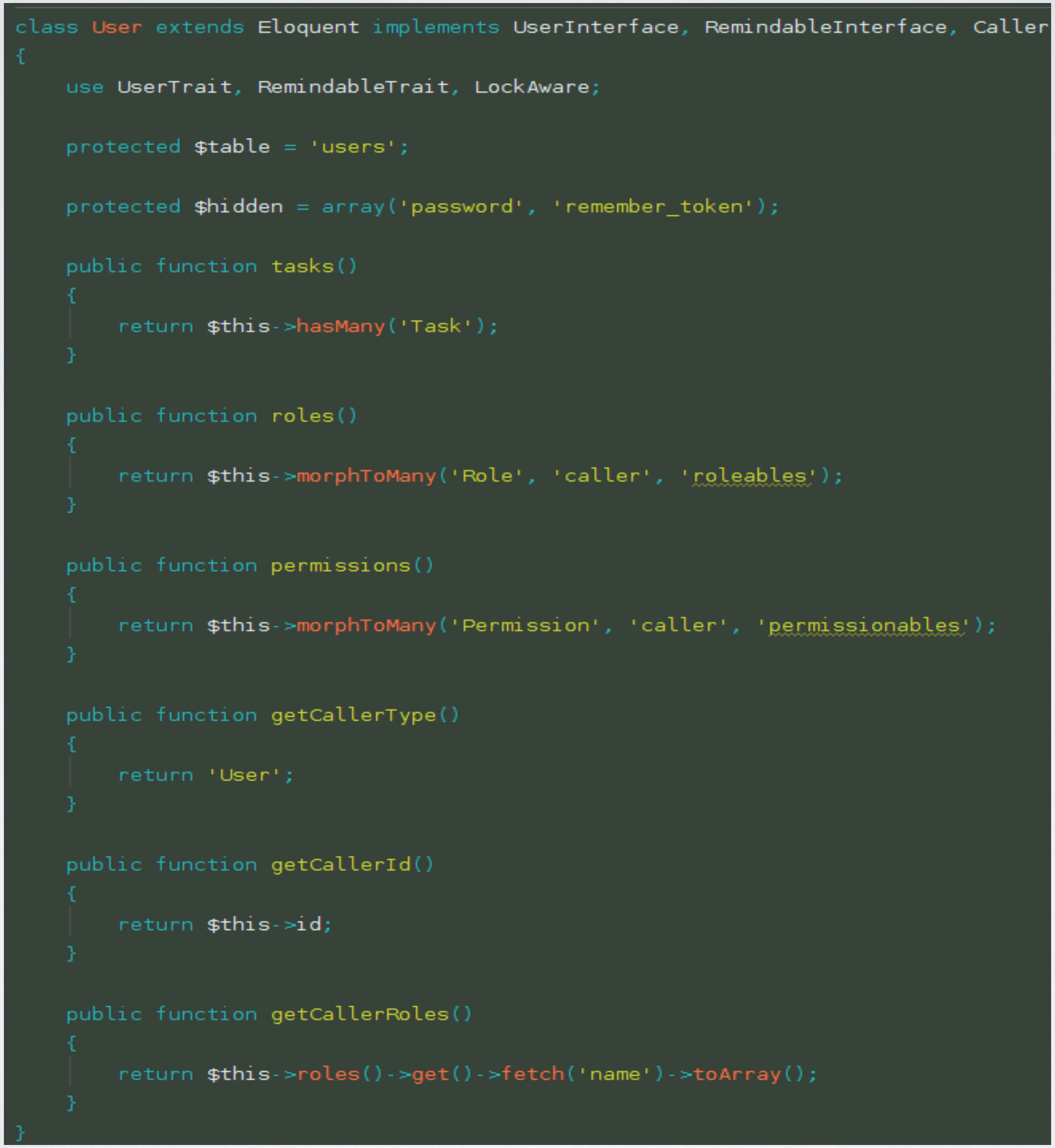
```php
class CreateRoleablesTable extends Migration {

    public function up()
    {
        Schema::create('roleables', function(Blueprint $table)
        {
            $table->increments('id');
            $table->integer('role_id')->unsigned()->index();
            $table->string('caller_type', 100);
            $table->integer('caller_id')->unsigned()->index();

            $table->foreign('role_id')->references('id')->on('roles');
        });
    }

    public function down()
    {
        Schema::drop('roleables');
    }
}
```

PERMISSIONABLES



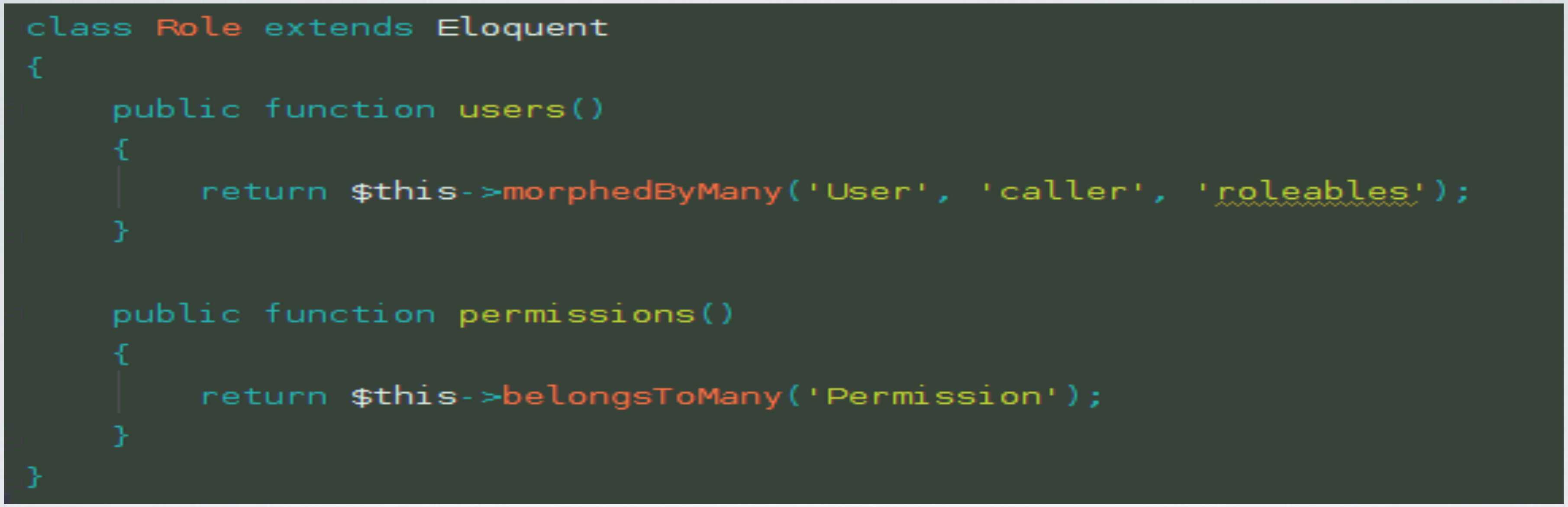
```
class CreatePermissionablesTable extends Migration {

    public function up()
    {
        Schema::create('permissionables', function(Blueprint $table)
        {
            $table->increments('id');
            $table->integer('permission_id')->unsigned()->index();
            $table->string('caller_type', 100);
            $table->integer('caller_id')->unsigned()->index();

            $table->foreign('permission_id')->references('id')->on('permissions');
        });
    }

    public function down()
    {
        Schema::drop('permissionables');
    }
}
```

USER CLASS

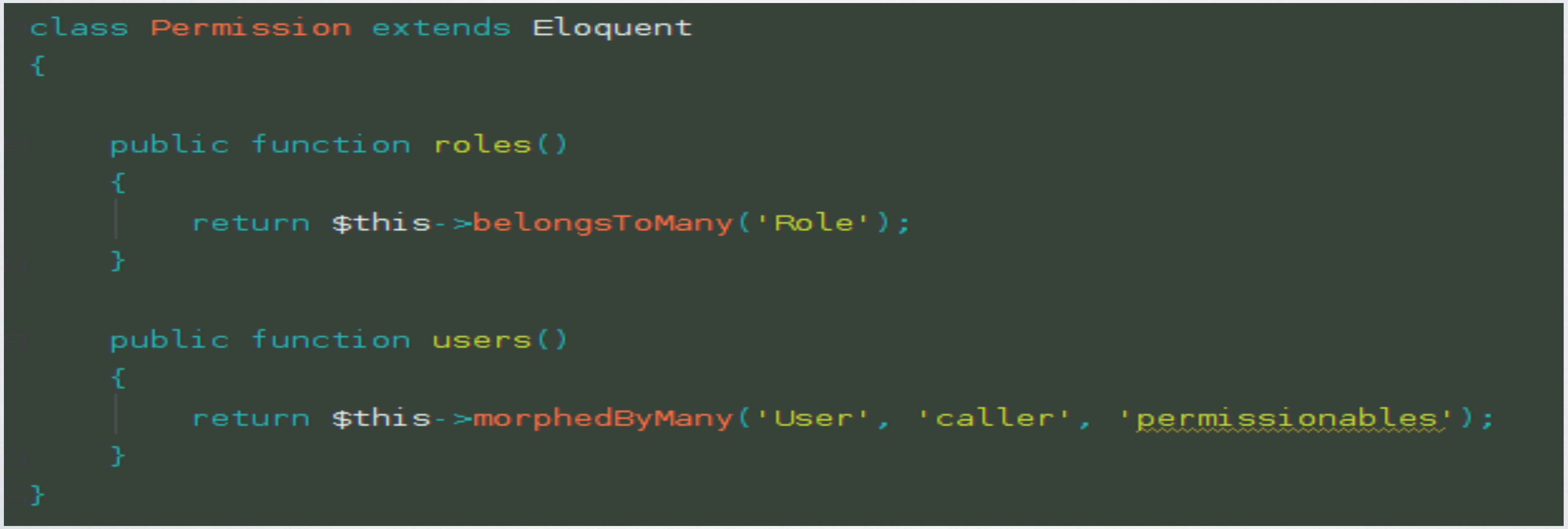


```php
class User extends Eloquent implements UserInterface, RemindableInterface, Caller
{
    use UserTrait, RemindableTrait, LockAware;

    protected $table = 'users';

    protected $hidden = array('password', 'remember_token');

    public function tasks()
    {
        return $this->hasMany('Task');
    }

    public function roles()
    {
        return $this->morphToMany('Role', 'caller', 'roleables');
    }

    public function permissions()
    {
        return $this->morphToMany('Permission', 'caller', 'permissionables');
    }

    public function getCallerType()
    {
        return 'User';
    }

    public function getCallerId()
    {
        return $this->id;
    }

    public function getCallerRoles()
    {
        return $this->roles()->get()->fetch('name')->toArray();
    }
}
```

ROLE AND PERMISSION MODELS



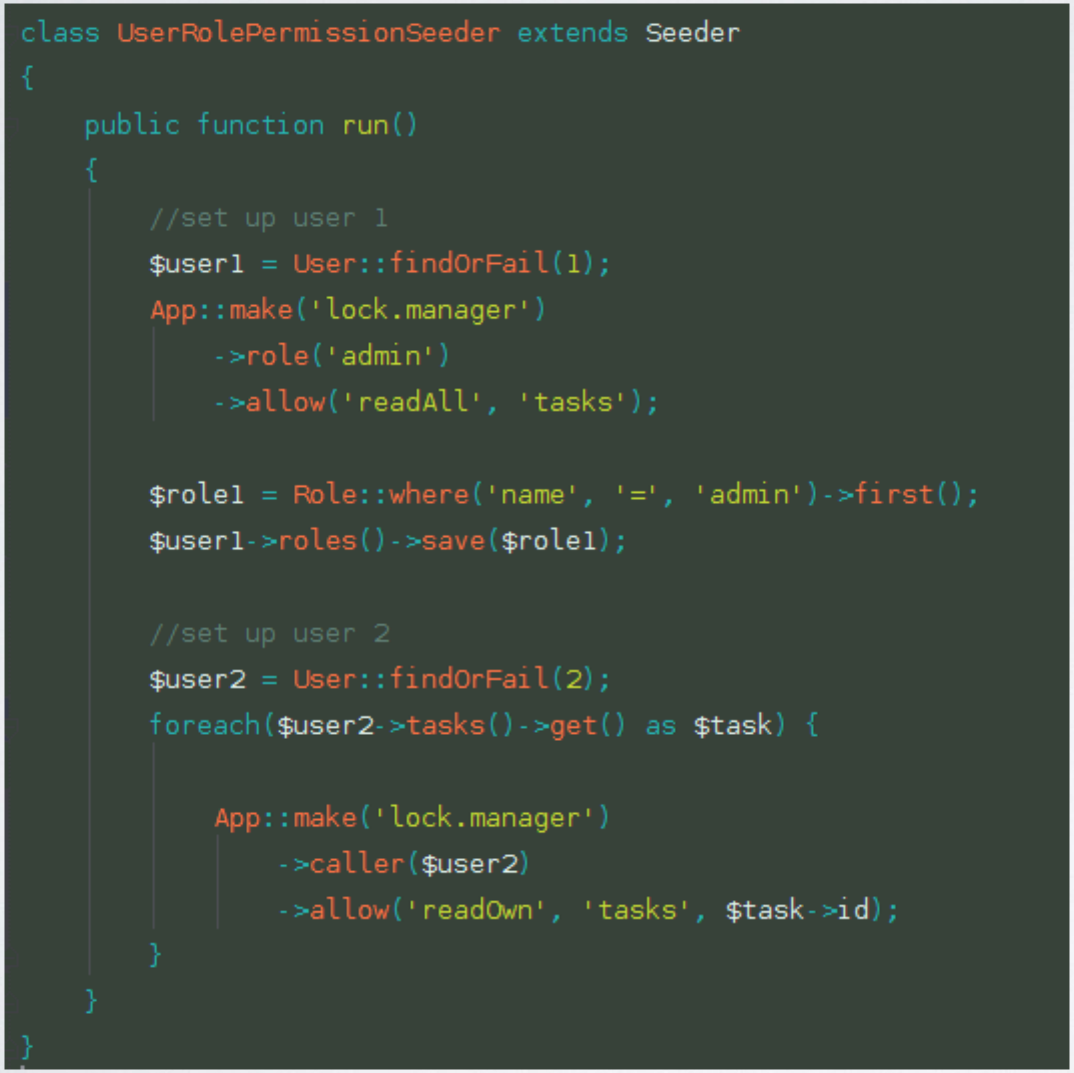
```
class Role extends Eloquent
{
    public function users()
    {
        return $this->morphedByMany('User', 'caller', 'roleables');
    }

    public function permissions()
    {
        return $this->belongsToMany('Permission');
    }
}
```



```
class Permission extends Eloquent
{

    public function roles()
    {
        return $this->belongsToMany('Role');
    }

    public function users()
    {
        return $this->morphedByMany('User', 'caller', 'permissionables');
    }
}
```

LET'S SEED ROLES AND PERMISSIONS

*In reality, you won't use a seeder to set permissions



```
class UserRolePermissionSeeder extends Seeder
{
    public function run()
    {
        //set up user 1
        $user1 = User::findOrFail(1);
        App::make('lock.manager')
            ->role('admin')
            ->allow('readAll', 'tasks');

        $role1 = Role::where('name', '=', 'admin')->first();
        $user1->roles()->save($role1);

        //set up user 2
        $user2 = User::findOrFail(2);
        foreach($user2->tasks()->get() as $task) {

            App::make('lock.manager')
                ->caller($user2)
                ->allow('readOwn', 'tasks', $task->id);
        }
    }
}
```

Notice how I had to tie a user to a permission directly for the second user. This is because there's no good way to tie a role to an individual resource.

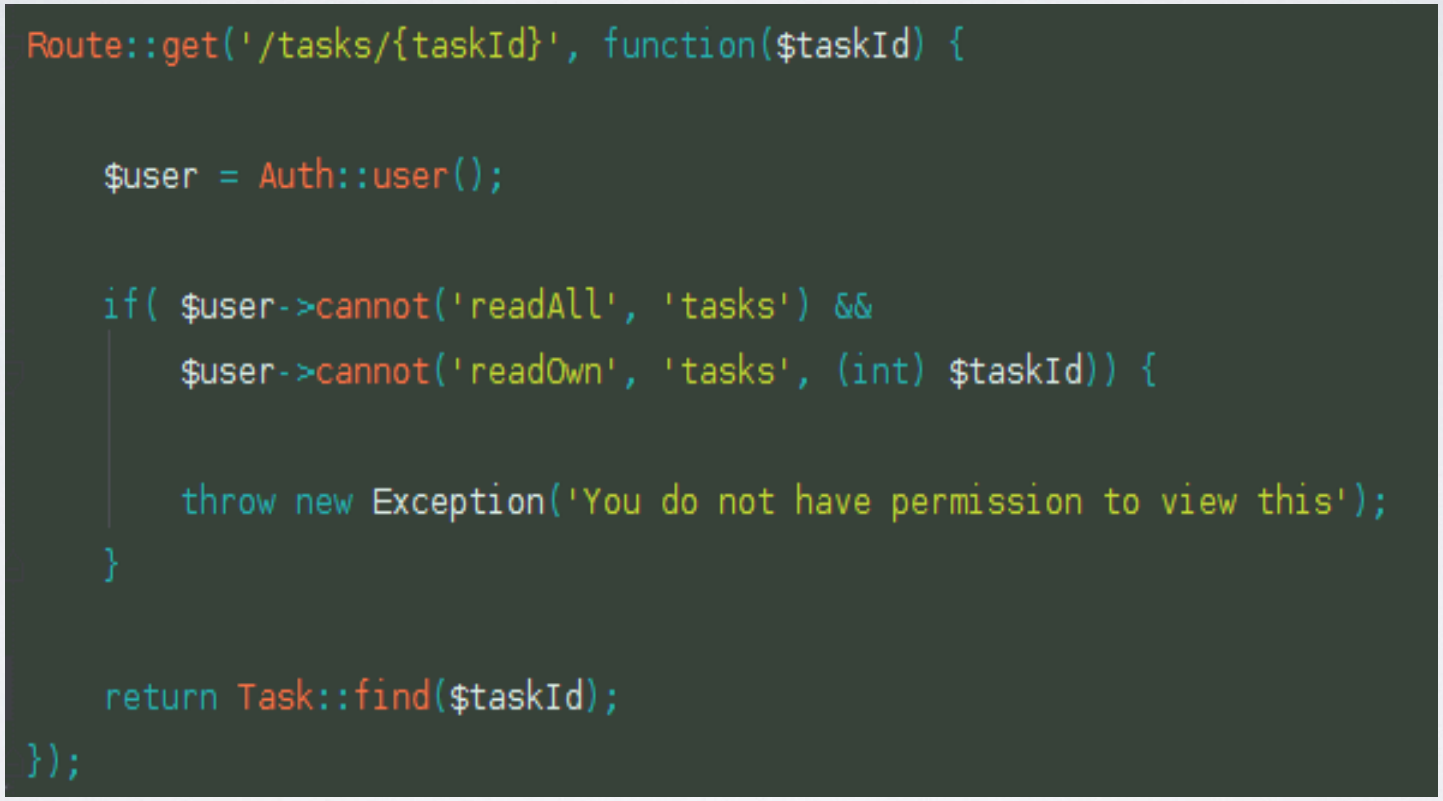
However we can assign a permission to an individual resource.

For example:

I can say prevent the wrong user from viewing task with an id of 1.

However, there's no way to say prevent the wrong role from viewing a task with an id of 1.

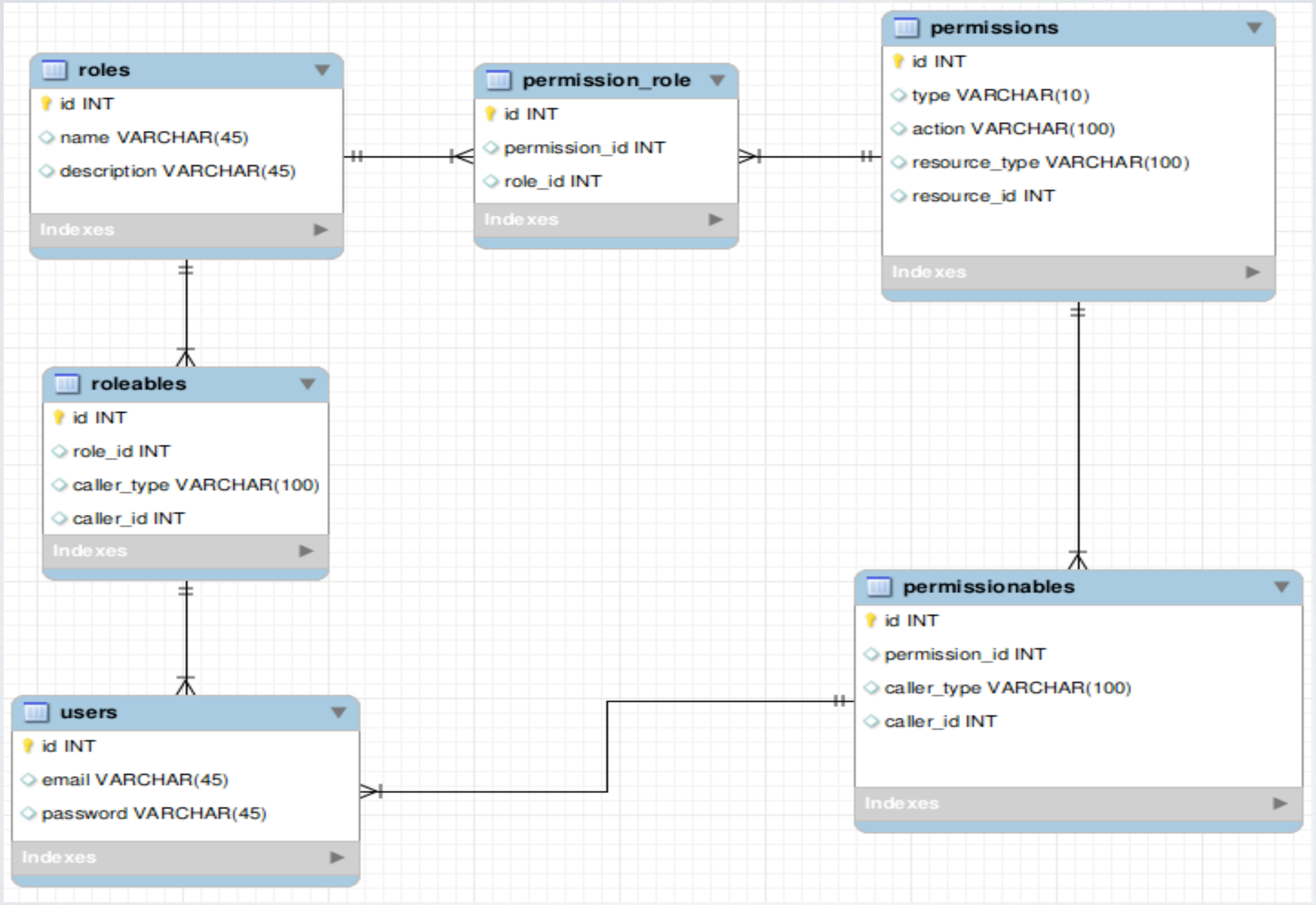
THERE'S A LOT OF POWER IN THIS CODE. IT'S ABLE TO CHECK THE LOGGED IN USER'S ROLES AND THEN GRAB THE PERMISSIONS FROM THOSE ROLES.



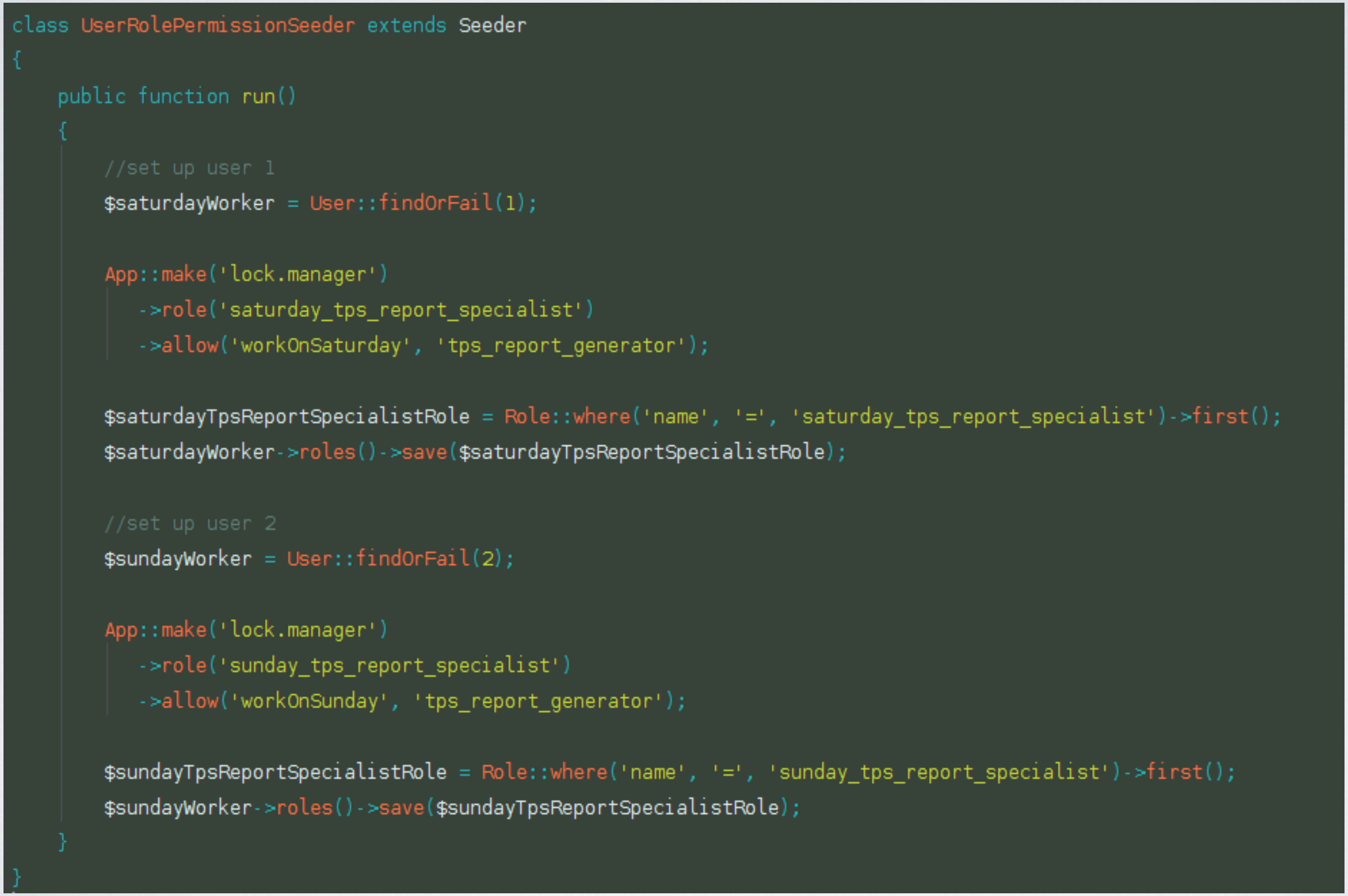
```php
Route::get('/tasks/{taskId}', function($taskId) {

    $user = Auth::user();

    if( $user->cannot('readAll', 'tasks') &&
        $user->cannot('readOwn', 'tasks', (int) $taskId)) {

        throw new Exception('You do not have permission to view this');
    }

    return Task::find($taskId);
});
```

LET'S BUILD A WEEKEND WORK DETECTOR

RESOURCE_TYPE ON PERMISSIONS DOESN'T HAVE TO BE A LARAVEL MODEL



ATTACHING ROLES AND PERMISSIONS TO A FEATURE



```php
class UserRolePermissionSeeder extends Seeder
{
    public function run()
    {
        //set up user 1
        $saturdayWorker = User::findOrFail(1);

        App::make('lock.manager')
            ->role('saturday_tps_report_specialist')
            ->allow('workOnSaturday', 'tps_report_generator');

        $saturdayTpsReportSpecialistRole = Role::where('name', '=', 'saturday_tps_report_specialist')->first();
        $saturdayWorker->roles()->save($saturdayTpsReportSpecialistRole);

        //set up user 2
        $sundayWorker = User::findOrFail(2);

        App::make('lock.manager')
            ->role('sunday_tps_report_specialist')
            ->allow('workOnSunday', 'tps_report_generator');

        $sundayTpsReportSpecialistRole = Role::where('name', '=', 'sunday_tps_report_specialist')->first();
        $sundayWorker->roles()->save($sundayTpsReportSpecialistRole);
    }
}
```

Create a role with permissions then attach that role to a user

ADDING SECURITY ON A FEATURE CONT.

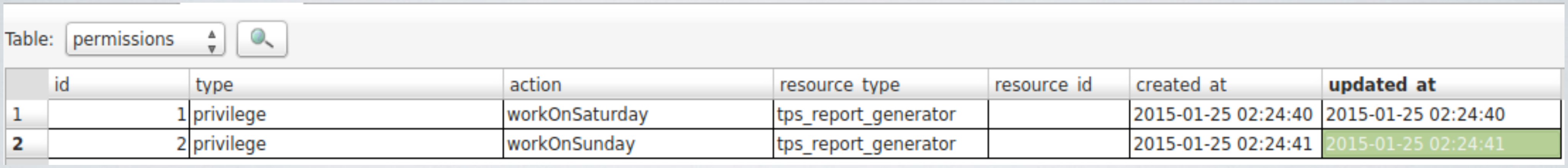


Table: permissions

	id	type	action	resource type	resource id	created at	updated at
1	1	privilege	workOnSaturday	tps_report_generator		2015-01-25 02:24:40	2015-01-25 02:24:40
2	2	privilege	workOnSunday	tps_report_generator		2015-01-25 02:24:41	2015-01-25 02:24:41

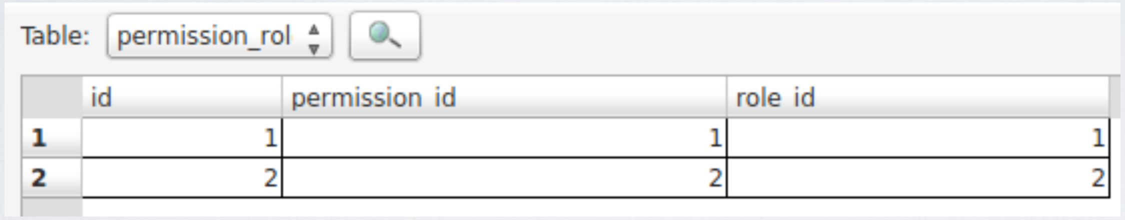


Table: permission_rol

	id	permission id	role id
1	1	1	1
2	2	2	2

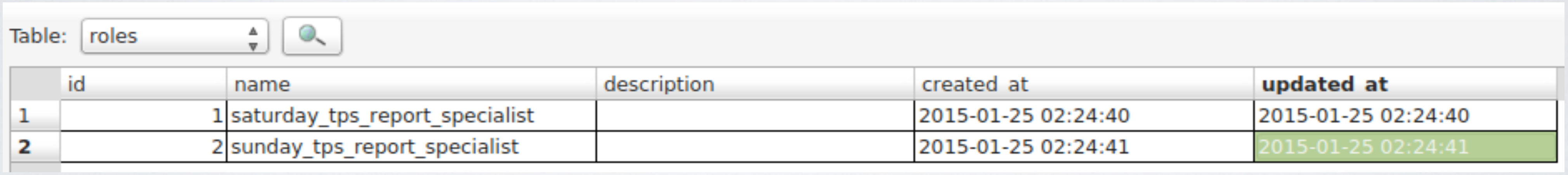


Table: roles

	id	name	description	created at	updated at
1	1	saturday_tps_report_specialist		2015-01-25 02:24:40	2015-01-25 02:24:40
2	2	sunday_tps_report_specialist		2015-01-25 02:24:41	2015-01-25 02:24:41

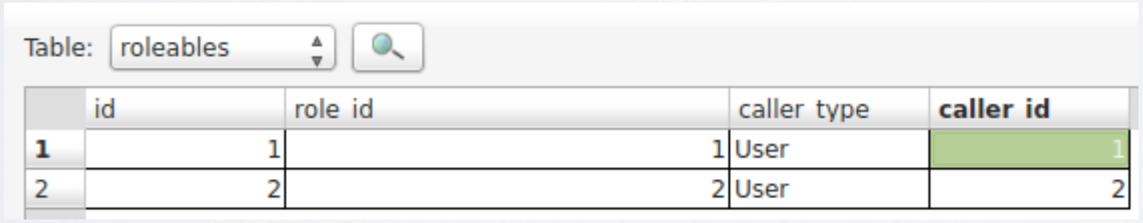


Table: roleables

	id	role id	caller type	caller id
1	1	1	User	1
2	2	2	User	2

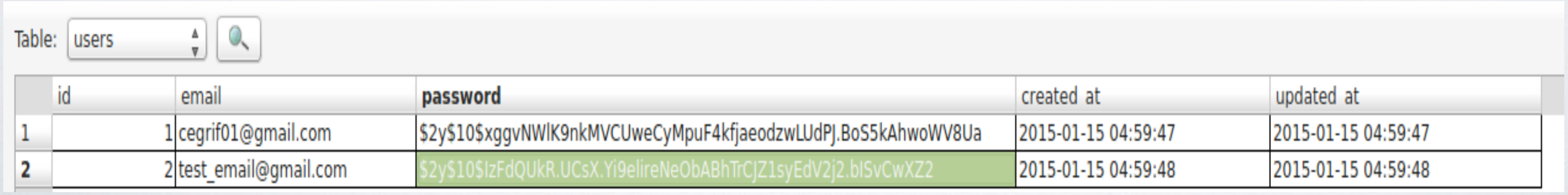
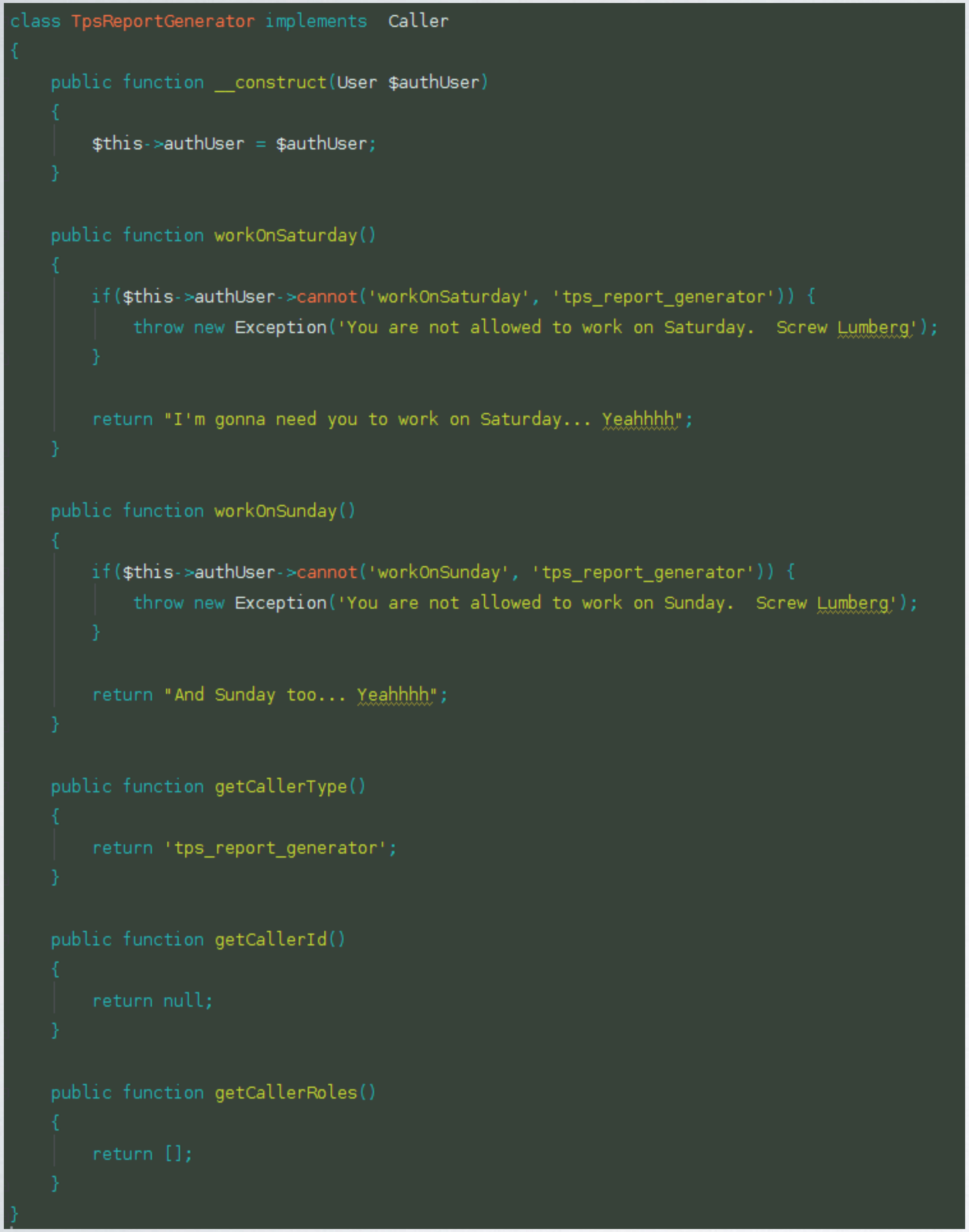


Table: users

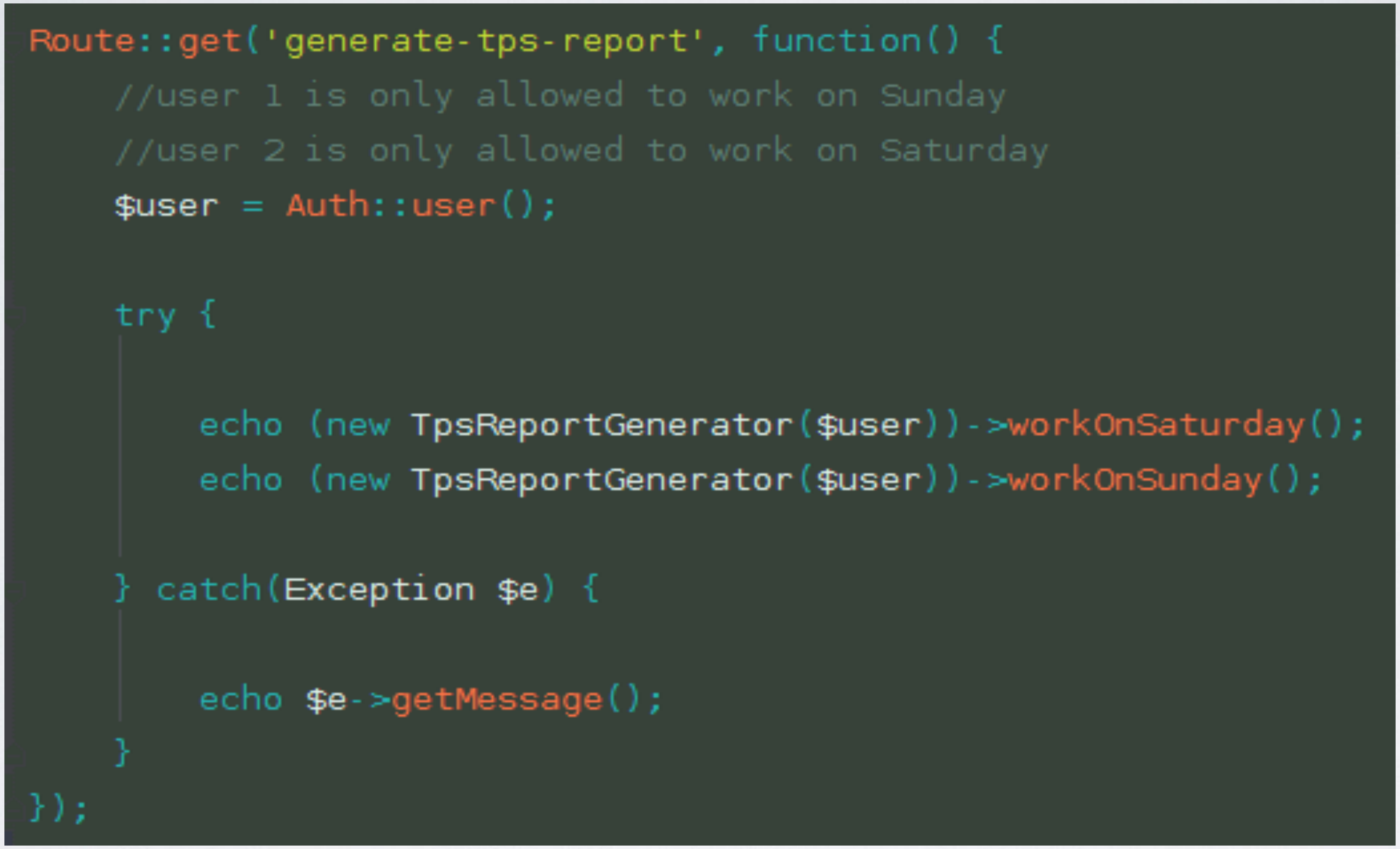
	id	email	password	created at	updated at
1	1	cegrif01@gmail.com	$2y$10$xggvNWlK9nkMVCUweCyMpuF4kfjaeodzwLUdPJ.BoS5kAhwoWV8Ua	2015-01-15 04:59:47	2015-01-15 04:59:47
2	2	test_email@gmail.com	$2y$10$IzFdQUkR.UCsX.Yi9elireNeObABhTrCJZ1syEdV2j2.bISvCwXZ2	2015-01-15 04:59:48	2015-01-15 04:59:48



```php
class TpsReportGenerator implements  Caller
{
    public function __construct(User $authUser)
    {
        $this->authUser = $authUser;
    }

    public function workOnSaturday()
    {
        if($this->authUser->cannot('workOnSaturday', 'tps_report_generator')) {
            throw new Exception('You are not allowed to work on Saturday.  Screw Lumberg');
        }

        return "I'm gonna need you to work on Saturday... Yeahhhh";
    }

    public function workOnSunday()
    {
        if($this->authUser->cannot('workOnSunday', 'tps_report_generator')) {
            throw new Exception('You are not allowed to work on Sunday.  Screw Lumberg');
        }

        return "And Sunday too... Yeahhhh";
    }

    public function getCallerType()
    {
        return 'tps_report_generator';
    }

    public function getCallerId()
    {
        return null;
    }

    public function getCallerRoles()
    {
        return [];
    }
}
```

As long as the correct roles are added to the user, we can decide which user should be able to work on Saturday or Sunday

INVOKING SECURITY ON FEATURE



```
Route::get('generate-tps-report', function() {
    //user 1 is only allowed to work on Sunday
    //user 2 is only allowed to work on Saturday
    $user = Auth::user();

    try {

        echo (new TpsReportGenerator($user))->workOnSaturday();
        echo (new TpsReportGenerator($user))->workOnSunday();

    } catch(Exception $e) {

        echo $e->getMessage();
    }
});
```

An exception will be thrown if we try to work a user on the wrong day

QUESTIONS???