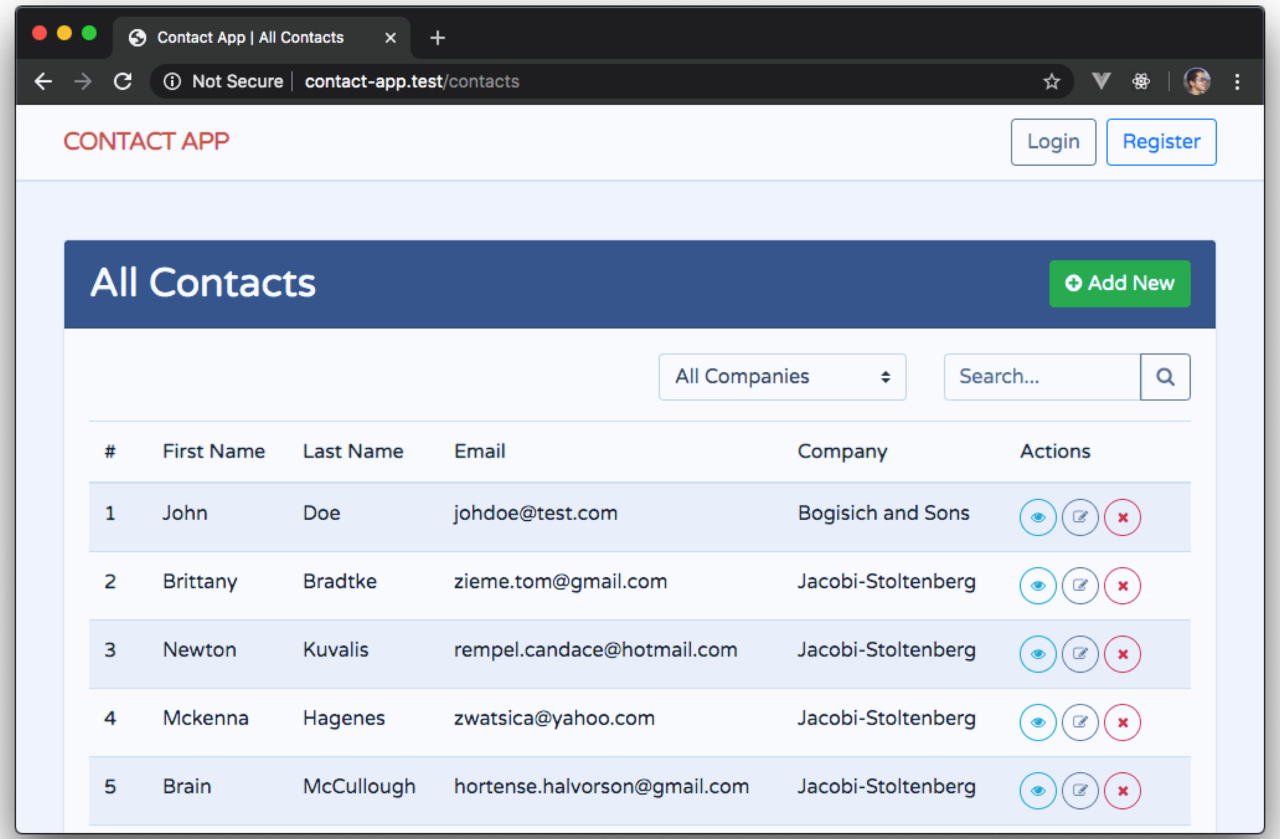
Protecting Routes

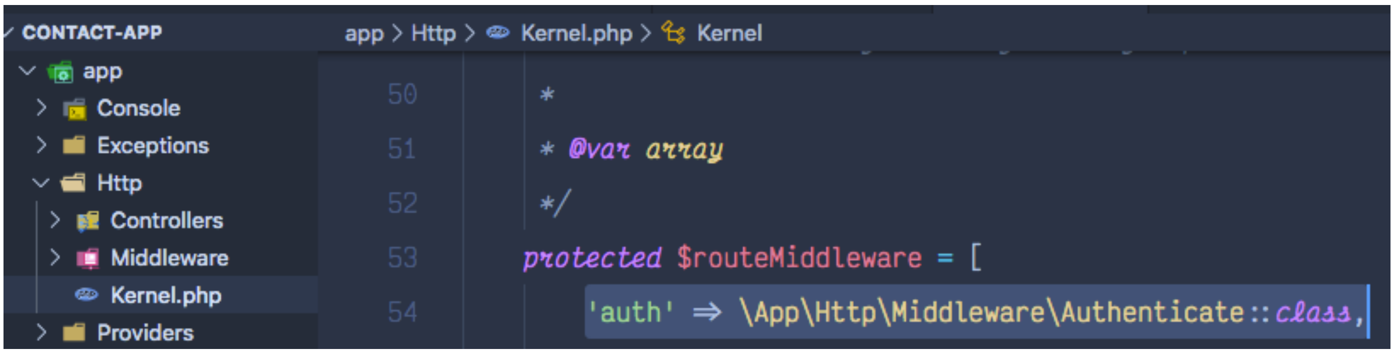
In this lesson we will see how to protect our routes from being access by unauthenticated user. We have integrated Authentication in our application. But any user can still access the `contacts` page without login.



#	First Name	Last Name	Email	Company	Actions
1	John	Doe	johdoe@test.com	Bogisich and Sons	
2	Brittany	Bradtke	zieme.tom@gmail.com	Jacobi-Stoltenberg	
3	Newton	Kuvalis	rempel.candace@hotmail.com	Jacobi-Stoltenberg	
4	Mckenna	Hagenes	zwatsica@yahoo.com	Jacobi-Stoltenberg	
5	Brain	McCullough	hortense.halvorson@gmail.com	Jacobi-Stoltenberg	

So, in order to protect that route we can use `auth` middleware which is defined at `Illuminate\Auth\Middleware\Authenticate`.

This middleware is already registered in the `Kernel.php` file. You can find it `routeMiddleware` property inside `app\Http\Kernel.php`.



```
     *
     * @var array
     */
    protected $routeMiddleware = [
        'auth' => \App\Http\Middleware\Authenticate::class,
```

To protect your routes you can follow one of these two options:

1. Attach the `auth` middleware in your route definition.
2. Call the `auth` middleware from your controller's constructor.

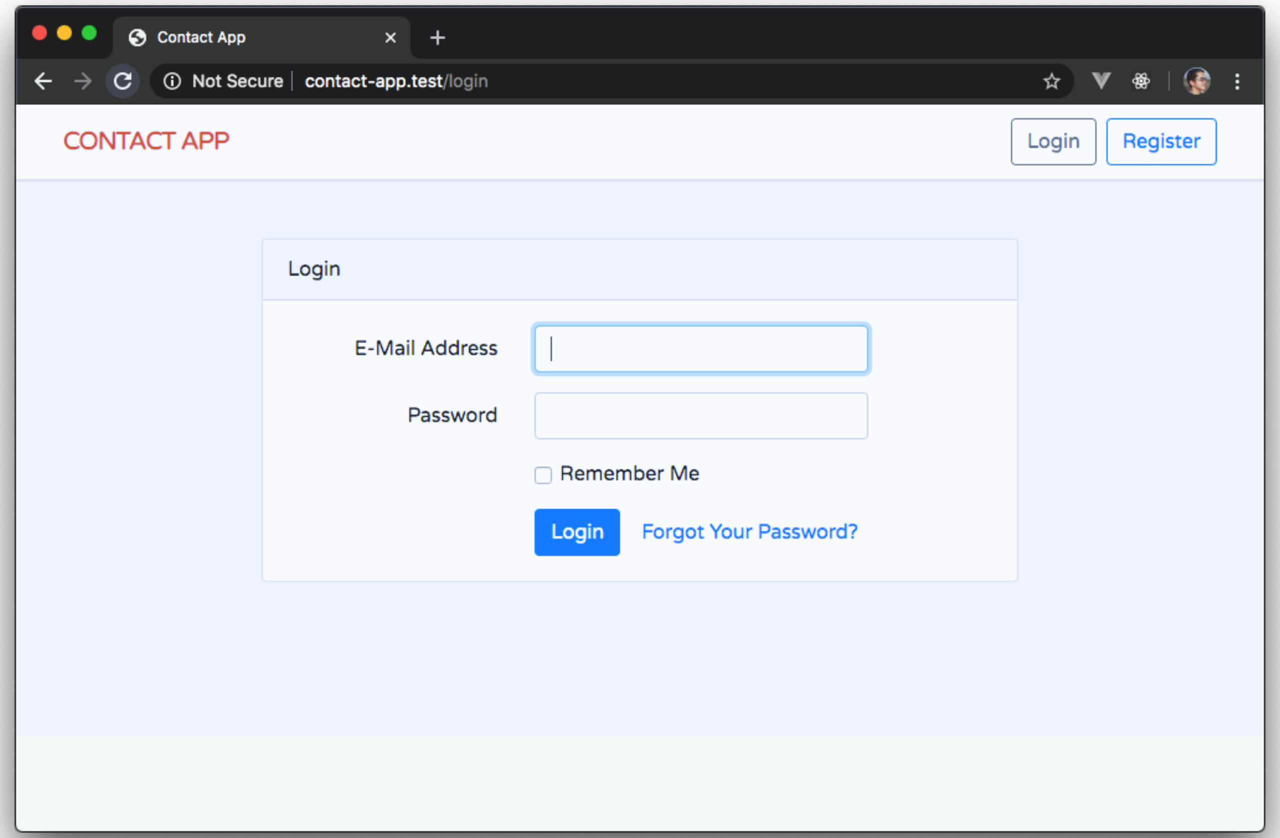
1. Attaching the `auth` middleware in routes definition

To attach the `auth` middleware you can add in your route definition with `middleware` and pass the `auth` middleware.

In `web.php` let's firstly protect the `contacts.index` with `auth` middleware like so:

```
Route::get('/contacts', [ContactController::class, 'index'])
     ->name('contacts.index')
     ->middleware('auth');
```

Now, if you're not signed in and try to access the **All Contacts** page. You'll be redirected to the `login` page.



You can do the similar thing to other contact routes to protect them from being accessed by unauthenticated user like so:

```
Route::post('/contacts', [ContactController::class, 'store'])
     ->name('contacts.store')
     ->middleware('auth');
Route::get('/contacts/create', [ContactController::class, 'create'])
     ->name('contacts.create')
     ->middleware('auth');
Route::get('/contacts/{id}', [ContactController::class, 'show'])
     ->name('contacts.show')
     ->middleware('auth');
Route::put('/contacts/{id}', [ContactController::class, 'update'])
     ->name('contacts.update')
     ->middleware('auth');
Route::delete('/contacts/{id}', [ContactController::class, 'destroy'])
     ->name('contacts.destroy')
     ->middleware('auth');
Route::get('/contacts/{id}/edit', [ContactController::class, 'edit'])
     ->name('contacts.edit')
     ->middleware('auth');
```

Although this way is valid, but here we find repetitive `middleware` calling in our route definition. We can make these much simpler by assigning the `auth` middleware in route `group`, then move our existing routes inside.

```
Route::middleware('auth')->group(function () {
    Route::get('/contacts', [ContactController::class, 'index'])-
>name('contacts.index');
    Route::post('/contacts', [ContactController::class, 'store'])-
>name('contacts.store');
    Route::get('/contacts/create', [ContactController::class, 'create'])-
>name('contacts.create');
    Route::get('/contacts/{id}', [ContactController::class, 'show'])-
>name('contacts.show');
    Route::put('/contacts/{id}', [ContactController::class, 'update'])-
>name('contacts.update');
    Route::delete('/contacts/{id}', [ContactController::class, 'destroy'])-
>name('contacts.destroy');
    Route::get('/contacts/{id}/edit', [ContactController::class, 'edit'])-
>name('contacts.edit');
});
```

2. Calling the `auth` middleware in controller's constructor

Before we call the `auth` middleware in our constructor's controller, make sure you're not call the `middleware('auth')` in your route definition.

Open the `ContactController`. Then define a constructor, add the `middleware` call, then pass in the `auth` middleware.

```php
class ContactController extends Controller
{
    public function __construct()
    {
        $this->middleware('auth');
    }

    // other methods definition
    // ...
}
```

By doing this way the `auth` middleware will get applied to all methods that defined in the controller. This because constructor will automatically called when you instantiate an object.

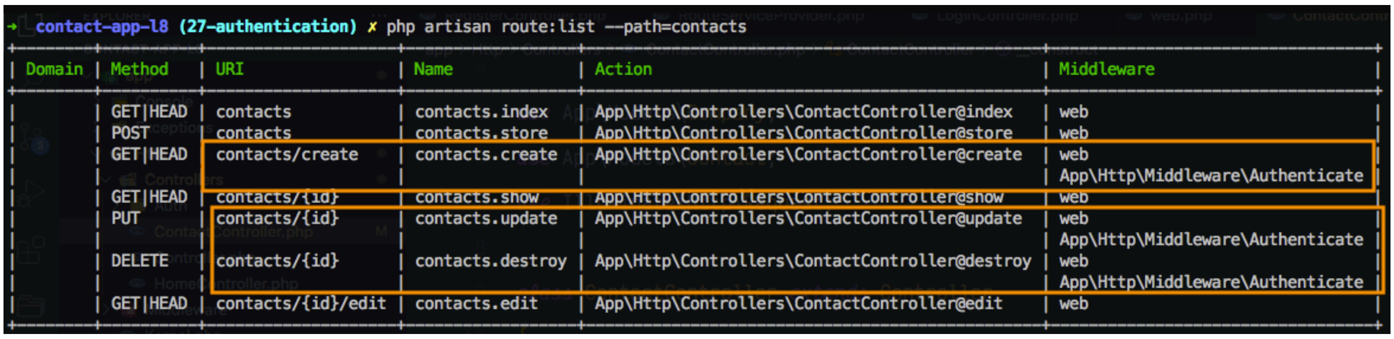
But you can utilise `except()` or `only()` method to apply the middleware on a certain method explicitly.

2.1. Using middleware `only` method

We can use middleware `only` method to *only* apply a certain middlware to the given methods. For example if you want to apply the `auth` middleware only on `create`, `update` and `delete` methods, you can do like this:

```php
public function __construct()
{
    $this->middleware('auth')->only('create', 'update', 'destroy');
}
```

Now if you see your routes in your terminal:



```
→ contact-app-l8 (27-authentication) ✗ php artisan route:list --path=contacts
+--------+----------+--------------------+------------------+-----------------------------------------------------+------------------------------------+
| Domain | Method   | URI                | Name             | Action                                              | Middleware                         |
+--------+----------+--------------------+------------------+-----------------------------------------------------+------------------------------------+
|        | GET|HEAD | contacts           | contacts.index   | App\Http\Controllers\ContactController@index        | web                                |
|        | POST     | contacts           | contacts.store   | App\Http\Controllers\ContactController@store        | web                                |
|        | GET|HEAD | contacts/create    | contacts.create  | App\Http\Controllers\ContactController@create       | web                                |
|        |          |                    |                  |                                                     | App\Http\Middleware\Authenticate   |
|        | GET|HEAD | contacts/{id}      | contacts.show    | App\Http\Controllers\ContactController@show         | web                                |
|        | PUT      | contacts/{id}      | contacts.update  | App\Http\Controllers\ContactController@update       | web                                |
|        |          |                    |                  |                                                     | App\Http\Middleware\Authenticate   |
|        | DELETE   | contacts/{id}      | contacts.destroy | App\Http\Controllers\ContactController@destroy      | web                                |
|        |          |                    |                  |                                                     | App\Http\Middleware\Authenticate   |
|        | GET|HEAD | contacts/{id}/edit | contacts.edit    | App\Http\Controllers\ContactController@edit         | web                                |
+--------+----------+--------------------+------------------+-----------------------------------------------------+------------------------------------+
```

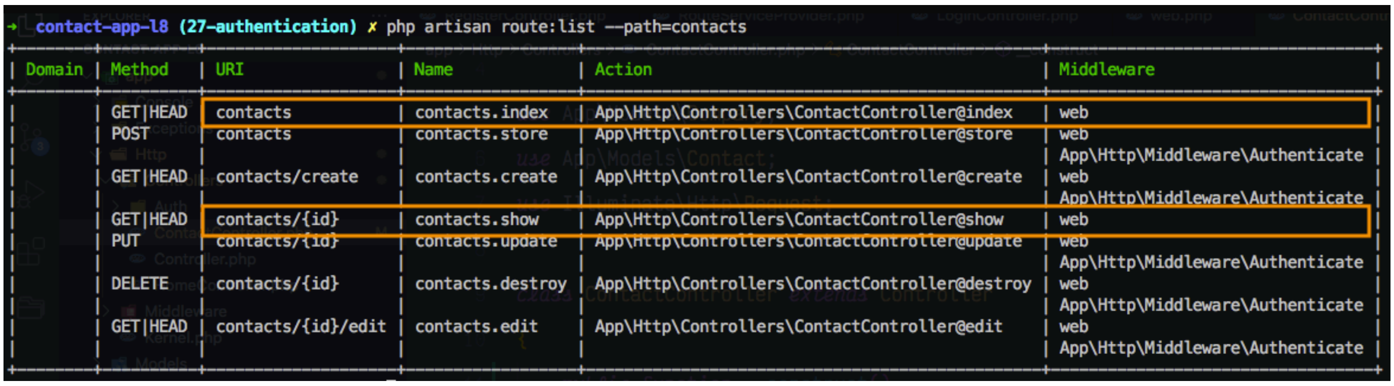
The `auth` middleware only applied on `contacts.create`, `contacts.update` and `contacts.destroy` routes.

2.2. Using middleware `except` method

We can use middleware `except` method to exclude the given methods from being applied by a certain middleware. If you want to exclude the `auth` middleware on let's say `index` and `show` methods you can do like so:

```php
public function __construct()
{
    $this->middleware('auth')->except('index', 'show');
}
```

If you now see your routes in your terminal:



```
→ contact-app-l8 (27-authentication) ✗ php artisan route:list --path=contacts
+--------+----------+--------------------+------------------+------------------------------------------------+----------------------------------+
| Domain | Method   | URI                | Name             | Action                                         | Middleware                       |
+--------+----------+--------------------+------------------+------------------------------------------------+----------------------------------+
|        | GET|HEAD | contacts           | contacts.index   | App\Http\Controllers\ContactController@index   | web                              |
|        | POST     | contacts           | contacts.store   | App\Http\Controllers\ContactController@store   | web                              |
|        |          |                    |                  |                                                | App\Http\Middleware\Authenticate |
|        | GET|HEAD | contacts/create    | contacts.create  | App\Http\Controllers\ContactController@create  | web                              |
|        |          |                    |                  |                                                | App\Http\Middleware\Authenticate |
|        | GET|HEAD | contacts/{id}      | contacts.show    | App\Http\Controllers\ContactController@show    | web                              |
|        | PUT      | contacts/{id}      | contacts.update  | App\Http\Controllers\ContactController@update  | web                              |
|        |          |                    |                  |                                                | App\Http\Middleware\Authenticate |
|        | DELETE   | contacts/{id}      | contacts.destroy | App\Http\Controllers\ContactController@destroy | web                              |
|        |          |                    |                  |                                                | App\Http\Middleware\Authenticate |
|        | GET|HEAD | contacts/{id}/edit | contacts.edit    | App\Http\Controllers\ContactController@edit    | web                              |
|        |          |                    |                  |                                                | App\Http\Middleware\Authenticate |
+--------+----------+--------------------+------------------+------------------------------------------------+----------------------------------+
```

The `auth` middleware applied to all `contacts.*`, except `contacts.index` and `contacts.show` route.

So that's how you could protect your routes using `auth` middleware. You can apply it on the route definition or on the controller's constructor.