Feb 21, 2021 - 2 min read

Creating Custom User Provider for Laravel

Whenever you need to connect your application to 3rd party apis for data sourcing or just plain user sourcing generally you would need to create a User Provider for the job. Althought offical docs have some information on the matter they are a bit abstract that I needed to do extra fiddling around the matter.

First we need to create a model for our users this class needs to extend either Illuminate\Auth\GenericUser which implements Authenticatable or we need to implement Illuminate\Contracts\Auth\Authenticatable ourselves.

Afterwards we need to create a custom providers to source our users. I will be implementing this through contract way.

```php
<?php

namespace App\Providers;

use Illuminate\Contracts\Auth\Authenticatable;
use Illuminate\Contracts\Auth\UserProvider as UserProviderContract;

class CustomProvider implements UserProviderContract
{
  public function retrieveById($identifier) {}

  public function retrieveByCredentials(array $credentials) {}

  public function validateCredentials(Authenticatable $user, array $credentials) {}

  public function retrieveByToken($identifier, $token) {}

  public function updateRememberToken(Authenticatable $user, $token) {}
}
```

> CustomProvider.php

There will be 5 methods we need to implement for our provider.

`retrieveById` will be called on every request where the session cookie has information about user id. In normal circumtances this won't be a problem because accessing the database would be somewhat fast. But if we want to call another service for this every call would be slower by default.

Instead we can store user information on session. Whether it is on a local database, redis or anothor medium. retrieveById() method would provide a dummy User model, which is used to extract session information which we will use to populate user information in a middleware.

```php
<?php
...
public function retrieveById($identifier)
{
  return new User(['id' => $identifier]);
}
```

`retrieveByCredentials(array $credentials)` This will be your prelogin/login action. In normal provider this will search for user in the configured database. But in our case we will send these credentials to another party to be verified. Which will provide us with information about user if successfull or errors if not.

```php
<?php
...
public function retrieveByCredentials(array $credentials)
{
  return new User(/* user information or errors. */);
}
```

```
}
```

`validateCredentials(Authenticatable $user, array $credentials)` This is where you validate whether or not our users password matches what is provided in the login form. Since the user created in retrieveByCrendials already know whether or not it is successfull about this authentication. We can ask for confirmation if user is successfully logged on or not.

```php
<?php
...
public function validateCredentials(Authenticatable $user, array $credentials) {
  try {
    return $user->is_successful;
  } catch (\Exception $e) {
    return false;
  }
}
```

Rest of the methods will help you handle remember tokens.

Because we are using session to store our user information accross requests we need to a store and retrieve it at some point in request lifecycle.

We will be storing this information after an user is logged in at `LoginController` and restore it in `EnsureFrontendRequestHasUser` middleware.

```php
<?php
...
protected function authenticated(Request $request, $user)
{
    session(['user' => $user]);
}
...
```

> LoginController.php

```php
<?php
...
public function handle(Request $request, Closure $next)
{
    $user = session('user');
    if (isset($user) && isset($user->is_succesful)) {
        Auth::setUser($user);
    }
    return $next($request);
}
...
```

> EnsureFrontendRequestHasUser.php

Only thing left is registering our custom providers.

```php
<?php
...
public function boot()
{
  ...
  Auth::provider('custom', function($app, array $config) {
    return new CustomProvider();
  });
}
```

> AuthUserProvider.php

```php
<?php
...
'providers' => [
    'users' => [
        'driver' => 'custome',
        'model' => App\Models\User::class,
    ],
```

```
    ],

```

> auth.php

After all this done you can login using your custom provider.

#Laravel

599 words

2021-02-21

← NEXT

Controlling Monitor Settings with ddcutil