User Social Login

Ruta: (post) api/v1/user/loginsocial (middlewares que observan esta ruta [`AuthenticationChannel`]), la cual apunta al controlador UserController, método loginSocial.

Resumen general de funcionamiento

El front hará un callback contra un servicio de authenticación en principio google, pero esto se extenderá a otros servicios como linkedin, twitter (x), twitch, etc.

Este callback devolverá al front entre otras cosas un token oauth2, **OJO!!** no un token JWT, este token es el que tiene que enviar al back junto conn un param driver del servicio(ver ejemplo a pie de este párrafo) el cual a través del paquete de Socialite volverá a hacer una llamada a google con este token y google devolverá la info del user donde cogeremos su correo y lo cotejaremos para ver que esta en nuestra base de datos y aprobar el login en esencia.

Front hace callback contra google y lo que tiene que enviar al back es:

param = socialToken value =
`ya29.a0AXooCgtoeYxGac76iv7El3FNasgIlXanf1m2uo3zj1UmsGgd3YAbaRjZrUprSe4fJ0cTbHEFvg7BuKlYRL7niD0SxU3Q9QDsx9thHpGnJjbDaA`
`ilsEflEFXZR5xYIk6bAFTiw9fMC64D2tSsuEefKvy3uPQRXk3xSAaCgYKAdgSARASFQHGX2MisWQT4jP5a-**********`

param = driver value = google (si fuese twitch pues twitch, etc.)

Una vez explicada la funcionalidad principal, entendamos como funciona el método.

Primero que todo, nos encontramos una clase proporcionada por Laravel para crear una instancia que compruebe los parámetro que incluyamos que tengan con los requisitos que definamos, esto quiere que decir que en esta api deberá llegar un parámetro socialToken y otro de driver decimos que es requerido para de esta manera si no va incluido lo capture como un error (img 1).

```php
$validator = Validator::make($request->all(), [
    'socialToken' => 'required',
    'driver' => 'required',
]);


if ($validator->fails()) {
    $error = 1;
    $message_validator = $message_validator->merge($validator->errors());
}
```

img 1

Primer check correcto? aseguramos que el el parámetro enviado de driver no sea distinto de google, si no, error (img 2) (esto podrá cambiar a medida que se meten más servicios, si se usa twitter pues se comprobará también || != twitter etc.).

```php
if ($error == 0) {
    $driver = $request->input('driver');
    if ($driver != 'google') {
        $error = 1;
        $message_validator = $message_validator->merge(['driver' => ['Driver not implemented.']]);
    }
}
```

img 2

Todo ok? pues comprobamos el token ahora y probramos que Socialite con el token dado y el driver nos devuelva el usuario que corresponde a ese token, de esta info ademas extraemos su id en dicha plataforma, si sale mal pues error (img 3).

```php
if ($error == 0) {
    $socialToken = $request->input('socialToken');
    try {
        $user = Socialite::driver($driver)->userFromToken($socialToken);
        $provider = $user->getId();
    } catch (\Exception $e) {
        $error = 1;
        $message_validator = $message_validator->merge(['token' => [$e->getMessage()]]);
    }
}
```

img 3

En este punto si todo fue ok, tenemos un objeto de usuario con toda su data de la plataforma en cuestión, lo que tenemos que hacer a continuación es que dicho usuario exista en nuestra base de datos lo cual lo realizaremos a través de la comparación del correo que nos ha proporcionado el servicio con un correo que tengamos dentro de la tabla lp_users de nuestra base de datos por tanto si el usuario de google con correo german.incentro lo cotejamos con nuestra base de datos y no hay ningún usuario que tenga ese correo o si lo hay pero la id de la plataforma del servicio no corresponde con la que tiene el user (en caso de existir, la primera vez que logee un user este campo estará vacio y se rellenará con la id del servicio), dará error (img 4).

```php
if ($error == 0) {
    $data_user = User::where('email', $user->getEmail())->first();

    if ($data_user) {
        if ($driver == 'google') {
            if ($data_user->google_id && $data_user->google_id != $provider) {
                $error = 2;
                $message_validator = $message_validator->merge(['token' => ['The google id field is not valid.']]);
            }
        }
    } else {
        $error = 2;
        $message_validator = $message_validator->merge(['token' => ['User unauthorized.']]);
    }
```

img 4

Ahora nos queda comprobar que la compañía a la que pertenece el user esté activa, el usuario no esté bloqueado o no esté bloqueado temporalmente (img 5).

```php
if ($error == 0) {
    $data_company = Company::where('id', $data_user->company_id)
        ->where('lp_companies.date_start', '<=', Carbon::now()->format('Y-m-d'))
        ->where('lp_companies.date_end', '>=', Carbon::now()->format('Y-m-d'))
        ->first();

    if (!$data_company) {
        $error = 2;
        $message_validator = $message_validator->merge(['token' => ['Invalid company.']]);
    }
}

if ($error == 0) {
    if ($data_user->bloqued_login == 1) {
        $error = 3;
        $message_validator = $message_validator->merge(['token' => ['User bloqued.']]);
    }
}

if ($error == 0) {
    if ($data_user->bloqued_to >= Carbon::now()) {
        $error = 3;
        $message_validator = $message_validator->merge(['token' => ['User bloqued temporarily.']]);
    }
}
```

img 5

A continuación realizamos la creación del UserToken, UserRememberToken ya explidos en sus respectivas apis, imagen de referencia en la api de social login para asociarla en dicha referencia en esas apis(img 6)

```php
if ($error == 0) {
    $token = base64_encode(Str::random(200));
    $remember_token = base64_encode(Str::random(200));
    $time_expired = intval(Setting::where('name', 'token_validate')->first()->value);
    $expired_at = Carbon::now()->addMinutes($time_expired);

    $refresh_token = base64_encode(Str::random(200));
    $refresh_expired_at = Carbon::now()->addMinutes(2 * $time_expired);

    UserToken::create([
        'user_id' => $data_user->id,
        'token' => $token,
        'expired_at' => $expired_at,
        'refresh_token' => $refresh_token,
        'refresh_expired_at' => $refresh_expired_at,
    ]);

    UserRememberToken::create([
        'user_id' => $data_user->id,
        'token' => $remember_token,
        'ip' => $request->ip(),
    ]);
```

img 6

Ahora si, si el driver es google y el google_id asociado al user en la base de datos es false , le añadimos la id del usuario en esa plataforma que guardamos en la variable provider.

Cambiamos los datos del usuario de la base de datos, social register a 1 (true), last login a la fecha de ahora, fault login a 0 y email verificado a 1 (true), guardamos cambios y generamos el array con la info que le daremos al front(img 7).

```php
        }

        $data_user->social_register = 1;
        $data_user->last_login = Carbon::now();
        $data_user->faults_login = 0;
        $data_user->verified_email = 1;
        $data_user->save();

        $array['data'][] = [
            'user_id' => $data_user->id,
            'token' => 'Bearer ' . $token,
            'remember_token' => $remember_token,
            'refresh_token' => $refresh_token
        ];
```

img 7

La respuesta que recibirá el front tras el correcto log in (img 8).

img 8