# Lab Exercise 10

Dr. Sarvar Abdullaev
`s.abdullaev@inha.uz`

April 19, 2019

The purpose of this lab is to learn using Laravel's authentication and authorization mechanisms in your own project.

## 1 Clone Project, Install Dependencies and Configure Database

Follow below steps in order to ensure that your project is set up correctly:

1. Clone newly created repository from accepted assignment to your local labs folder.
2. Open terminal inside that folder and run following command `composer install` to install all PHP dependencies of cloned project
3. Rename `.env.example` file to `.env` file. In command line run `mv .env.example .env` (Linux or MacOS) or `ren .env.example .env` (Windows)
4. Run following command afterwards: `php artisan key:generate`
5. Go to `https://remotemysql.com/signup.html` and provide some email address.It will create a free database account. Save details of your newly created database account into somewhere safe. You can login to your database using these credentials in this `https://remotemysql.com/phpmyadmin/`
6. In your Laravel project folder, open `.env` file and copy your remote database credentials to corresponding environment variables inside `.env` file, and save it.
7. In your laravel project folder, open `config\database.php` file and ensure that your `mysql` configuration is set as shown below:

```
'mysql' => [
            'driver' => 'mysql',
            'host' => env('DB_HOST', 'localhost'),
            'port' => env('DB_PORT', '3306'),
            'database' => env('DB_DATABASE', 'forge'),
            'username' => env('DB_USERNAME', 'forge'),
            'password' => env('DB_PASSWORD', ''),
            'charset' => 'utf8',
            'collation' => 'utf8_unicode_ci',
            'prefix' => '',
            'strict' => true,
            'engine' => null,
            'modes'  => [
                'ONLY_FULL_GROUP_BY',
                'STRICT_TRANS_TABLES','NO_ZERO_IN_DATE', 'NO_ZERO_DATE',
                'ERROR_FOR_DIVISION_BY_ZERO',
                'NO_ENGINE_SUBSTITUTION',
                ],
          ],
```

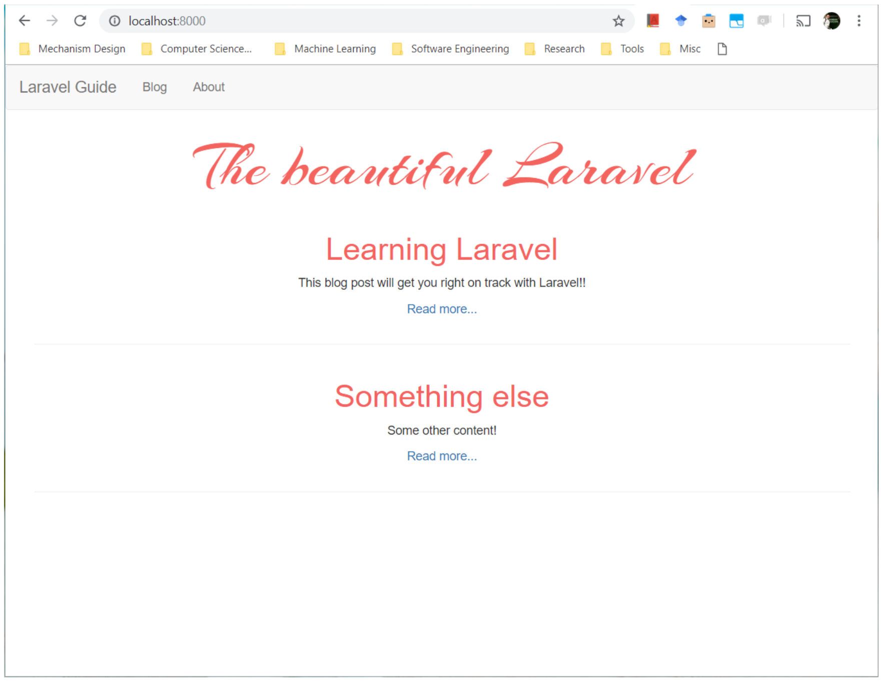
   Ensure that `modes` key is set as shown above. It is important because `https://remotemysql.com` does not grant full permission on your database, and your database connection driver should use specified modes.

8. Run `php artisan migrate` to create all necessary database tables.

9. In `.env` file, provide your `http://mailtrap.io` credentials in SMTP server settings. You should register to `http://mailtrap.ip` and find them in your default inbox.

```
MAIL_DRIVER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=[your_mailtrap_username]
MAIL_PASSWORD=[your_mailtrap_password]
MAIL_FROM_ADDRESS=sender@laravelblog.uz
MAIL_FROM_NAME=LaravelBlog
```

10. Once all dependencies are installed, run following command `php artisan serve`. This will start a Laravel's own development web server at `http://localhost:8000`. Open it in your browser. You should be able to see Figure 1 web page:



The beautiful Laravel

Learning Laravel

This blog post will get you right on track with Laravel!!

Read more...

Something else

Some other content!

Read more...

Figure 1: First View

11. Now your Laravel project is fully configured for using database.

## 2 Integrating Authentication to Your Project

Complete steps below to customize Laravel's authentication mechanism in your project:

1. Run `php artisan make:auth`. This will create all views, controllers, middleware and models necessary for supporting authentication in your project.

2. In browser, go to `http://localhost:8000/register` and you should be able to see a registration form shown in Figure 2. Fill it in and press *Register* button. Web-site will redirect you to default `\home` path, it was created automatically when you run previous command. We will remove it later, and redirect users to our `\admin` section.

3. Log out from default page and go to `http://localhost:8000/login`, below is the login form, you should be able to view. Try to sign in with your credentials.

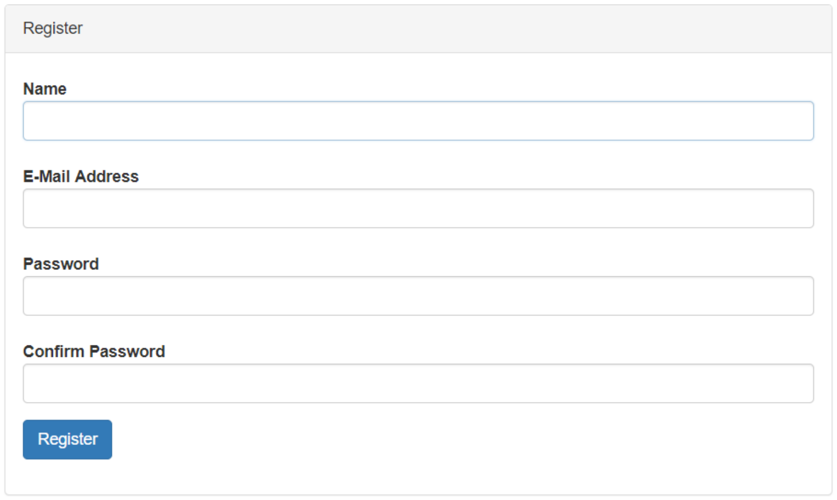


Figure 2: Register Form

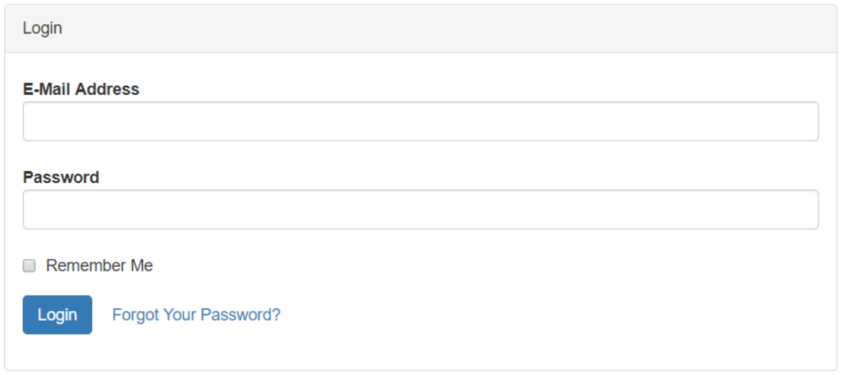


Figure 3: Login Form

4. Go to `https://remotemysql.com/login` and sign in to your database with credentials you have. In `users` table, you should be able to find newly registered user.
5. Open `views\partials\header.blade.php` file and place following snippet next to other links in your navigation menu.

```
@if(!Auth::check())

    <li><a href="{{ url('/login') }}">Login</a></li>
    <li><a href="{{ url('/register') }}">Register</a></li>

@else
    <li><a href="{{ route('admin.index') }}">Posts</a></li>
    <li>
        <a href="{{ url('/logout') }}"
            onclick="event.preventDefault();
                     document.getElementById('logout-form').submit();">
            Logout
        </a>

        <form id="logout-form" action="{{ url('/logout') }}" method="POST" style="display: none;">
            {{ csrf_field() }}
        </form>
    </li>
@endif
```

6. Open `views\auth\register.blade.php` and change extended layout to `layouts.master`
7. Open `views\auth\login.blade.php` and change extended layout to `layouts.master`
8. Open `views\auth\passwords\reset.blade.php` and change extended layout to `layouts.master`
9. Open `views\auth\passwords\email.blade.php` and change extended layout to `layouts.master`
10. Open `views\admin\index.blade.php` and change extended layout to `layouts.master`
11. Open `views\admin\edit.blade.php` and change extended layout to `layouts.master`
12. Open `views\admin\create.blade.php` and change extended layout to `layouts.master`
13. Remove following files:

```
views\layouts\admin.blade.php
views\partials\admin-header.blade.php
views\layouts\admin.blade.php
views\layouts\app.blade.php
views\home.blade.php
```

14. Open `app/Http/Controllers/Auth/LoginController.php` and set `protected $redirectTo='/admin'`. This should redirect users after successful sign in to Admin section.
15. Open `app/Http/Controllers/Auth/RegisterController.php` and set `protected $redirectTo='/admin'`. This should redirect users after successful registration to Admin section.
16. Open `app/Http/Controllers/Auth/ResetPasswordController.php` and set `protected $redirectTo='/admin'`. This should redirect users after successful reset of password to Admin section.
17. Open `app/Http/Middleware/RedirectIfAuthenticated.php` and set `redirect('/admin')`. This should redirect users to Admin section if they are authenticated.
18. Open `routes/web.php` and make following change for Admin-prefixed URLs. This should protect all `/admin` sublinks from unauthenticated users.

```
Route::group(['prefix' => 'admin', 'middleware'=>['auth']], function() {
...
}
```

19. Now your Laravel blog can authenticate users and redirect them to Admin section after successful sign in. It can also register users, recover passwords and remember users for long time. Admin section is also protected from unauthenticated users.

## 3 Integrating Authorization to Your Project

Complete steps below to customize Laravel's authorization mechanism in your project:

1. Open database migration file `create_posts_table` and add `user_id` field as follows:

   ```
   Schema::create('posts', function (Blueprint $table) {
               $table->increments('id');
               $table->timestamps();
               $table->string('title');
               $table->text('content');
               $table->integer('user_id');
           });
   ```

2. Open file `app\Post.php` and add following function:

   ```
   public function user(){
            return $this->belongsTo('App\User');
     }
   ```

3. Open file `app\User.php` and add following function:

   ```
   public function posts() {
         return $this->hasMany('App\Post');
   }
   ```

4. Open file `database\seeds\DatabaseSeeder.php` and comment PostTableSeeder. We will not add random posts through seader.

   ```
   //$this->call(PostTableSeeder::class);
     $this->call(TagTableSeeder::class);
   ```

5. Run `php artisan migrate:refresh` to recreate all tables with new fields

6. Run `php artisan db:seed` to populate you database with tags.

7. Open `app\Providers\AuthServiceProvider.php` and inside `boot()` method, create a new Gate.

   ```
   Gate::define('update-post', function($user, $post){
       return $user->id == $post->user_id;
   });
   ```

8. Open `PostController.php` and add following namespaces at the top

   ```
   use Auth;
   use Gate;
   ```

   And ensure that you controller now uses created Gate to check if the user is allowed to update record as follows. These are methods such as `postAdminUpdate` and `getAdminDelete`. Make sure that you place below gate after `$post = Post::find($request->input('id'));`, so you can pass `$post` variable to gate.

   ```
    if (Gate::denies('update-post', $post)) {
       return redirect()->back();
    }
   ```

9. In `PostController.php`, modify `$post->save()` to `Auth::user()->posts()->save($post);` inside `postAdminCreate(Request $request)` function. Do not make this change anywhere else. This way, it will create a post and assign it to a corresponding user.

10. After placing corresponding gates to your PostController methods, your application will be able to authorize users and prevent them from updating posts that do not belong to them.

## 4 Final Solution

You can compare your solution with the final solution here: `https://github.com/iuthub/ip2019-lab11/tree/solution`