LARAVEL 10

Part 4: Authentication, Sessions, send emails

INSTALL LARAVEL UI

```
composer require laravel/ui
```

SCAFFOLDING

```
php artisan ui bootstrap --auth
```

You can stop the vite and use the normal bootstrap in

resources → views → auth → layouts → app.blade.php or resources → views → layouts → app.blade.php

```
{{-- @vite(['resources/sass/app.scss', 'resources/js/app.js']) --}}

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC" crossorigin="anonymous">

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
crossorigin="anonymous"></script>
```

CONFIGURE EMAIL

You can use your real email configuration or test through mailtrap.io or mailhog

Inside .env file configure the email settings

Go to Models → User.php and uncomment

```
use Illuminate\Contracts\Auth\MustVerifyEmail;
```

Change class to

```
class User extends Authenticatable implements MustVerifyEmail
```

CONFIGURE SMTP EMAIL

Add email verifier inside route file, and be sure to change the auth route to

```
Auth::routes(['verify'=>true]);
```

PAGES FOR ONLY VERIFIED EMAIL ACCOUNTS

Inside your route page use the middleware verified as below example

```
Route::get('/home', [HomeController::class, 'index'])-
>middleware('verified')->name('home');
```

ADD MORE COLUMNS TO THE REGISTRATION

- Go to the register controller and inside the validator method add the new columns
- Inside the create method add the mobile
- Go to the user DB schema, and add the mobile
- Add to the user Model fillable array
- Migrate
- Edit the register blade file

LOGIN USING MOBILE OR EMAIL

From login controller add below method

```
public function credentials(Request $request){
        if(is_numeric($request->email)){
            return ['mobile'=>$request->email, 'password'=>$request->password];
        }elseif(filter_var($request->email, FILTER_VALIDATE_EMAIL)){
            return ['email'=>$request->email, 'password'=>$request->password];
        }
    }
```

SESSIONS

Session default configuration can be found inside the .env file

```
SESSION_DRIVER=file

SESSION_LIFETIME=120 // in minutes
```

Then

Config → session.php

CREATE AND GET SESSIONS

Create Session

```
session()->put('test', 'First Laravel session’);
```

Get Session

```
session('test');
```

DELETE SESSION

```
session()->forget('test’);
```

Or delete all sessions at all

```
session()->flush();
```

FLASH SESSION

- Used to create a session that can be used for one time only

```
session()->flash('test1', 'First Laravel session');
```

MORE DETAILS ABOUT THE SESSIONS

https://laravel.com/docs/10.x/session

SEND EMAILS

From terminal

```
php artisan make:mail DemoMail
```

File path can be found on

app/Mail/DemoMail.php

SEND EMAILS

Create blade file for your email content

https://laravel.com/docs/10.x/mail

https://laravel-school.com/posts/how-to-send-email-in-laravel-10/

SEND EMAILS

Go to the email file app/Mail/DemoMail.php be sure to add the blade file to the content method

```
public function content(): Content
    {
        return new Content(
            view: 'emails.demo',
        );
    }
```

SEND EMAILS

Inside your controller

```
use Mail;

use App\Mail\DemoMail;
```

And at your method use the below as example

```
Mail::to('your_email@gmail.com')->send(new DemoMail());
```

SEND EMAILS – ADVANCED EXAMPLE

https://www.laravelia.com/post/laravel-10-available-mail-options-with-send-example

MULTI LANGUAGE WEBSITE

By default, Laravel static text translations are stored in the /lang folder. But in Laravel 10 that lang folder is not included in the beginning.

Running the following artisan command will add it:

```
php artisan lang:publish
```

MULTI LANGUAGE WEBSITE

You will find a folder called lang/en

You can copy en folder and rename to ar

Translate the keys inside it

MULTI LANGUAGE WEBSITE

Add the translations to the blade files, example as below

```
<label for="email">{{ __('messages.email') }}:</label>
```

Note: messages is the translation php file, and email is the array key

MULTI LANGUAGE WEBSITE

To change the default language go to config → app.php

Change to the required language

```
'locale' => 'en',
```

MULTI LANGUAGE WEBSITE

Install mcamara to manage the langauges

https://github.com/mcamara/laravel-localization

```
composer require mcamara/laravel-localization
```

Then show and publish configuration

```
php artisan vendor:publish --provider="Mcamara\LaravelLocalization\LaravelLocalizationServiceProvider"
```

MULTI LANGUAGE WEBSITE

Goto config → laravellocalization.php to set the languages

MULTI LANGUAGE WEBSITE

Register Middleware

app/Http/Kernel.php

Add the code from https://github.com/mcamara/laravel-localization#installation

MULTI LANGUAGE WEBSITE

Add route group for the languages as below

```
Route::group(
[
		'prefix' => LaravelLocalization::setLocale(),
		'middleware' => [ 'localeSessionRedirect', 'localizationRedirect', 'localeViewPath' ]
], function(){
//		Your routes
});
```

TRANSLATE ERROR MESSAGES

Inside your controller manage the messages as below, and be sure to add the translation to the messages file.

```
$messages=[
            'title.required'=> __('messages.titleRequired'),
            'title.string'=>__('messages.titleString'),
            'description.required'=> __('messages.descRequired'),
        ];
```

TASK SCHEDULAR

```
php artisan make:command Expiration
```

https://laravel.com/docs/10.x/scheduling

TASK SCHEDULAR – EXPIRE USERS

Add a new column to the users table

Boolean → expired

TASK SCHEDULAR – EXPIRE USERS

Inside the Expiration.php

```
use App\Models\User;

public function handle()
    {
        $users = User::where('expired', 0)->get();

        foreach($users as $user){
            $user->update(['expired'=>1]);
        }

    }
```

TASK SCHEDULAR – EXPIRE USERS

Inside the kernel, run app → console → kernel.php

```
use App\Console\commands\Expiration;
protected function schedule(Schedule $schedule): void
    {
        $schedule->command('user:expiration')->everyMinute();
    }
```

ADD NEW CUSTOM ROUTE

https://www.linkedin.com/pulse/simple-steps-create-custom-route-file-laravel-avaneesh-verma

LOGIN USING SOCIAL MEDIA

From cmd install the package using the composer

```
Composer require Laravel/socialite
```

https://laravel.com/docs/10.x/socialite

LOGIN USING SOCIAL MEDIA

`Create your api app keys for the social media platforms required, example`

https://developers.facebook.com/

LOGIN USING SOCIAL MEDIA

Goto config → services.php and add credentials for each platforms used, example

```
'facebook' => [
        'client_id' => env('FACEBOOK_CLIENT_ID'),
        'client_secret' => env('FACEBOOK_CLIENT_SECRET'),
        'redirect' => 'http://example.com/callback-url',
    ],
```

LOGIN USING SOCIAL MEDIA

Goto config → services.php and add credentials config for each platform used, example

```
'facebook' => [
        'client_id' => env('FACEBOOK_CLIENT_ID'),
        'client_secret' => env('FACEBOOK_CLIENT_SECRET'),
        'redirect' => env('FACEBOOK_CALLBACK'),
    ],
```

LOGIN USING SOCIAL MEDIA

Inside.env file add the app credentials, example below

```
FACEBOOK_CLIENT_ID=your_id
FACEBOOK_CLIENT_SECRET=your_secret_key
FACEBOOK_CALLBACK=callback_url
```

LOGIN USING SOCIAL MEDIA

Add routes for redirect and callback as below

```
Route::get('/auth/redirect', function () {
    return Socialite::driver('facebook')->redirect();
})->name('facebookRedirect');

Route::get('/auth/callback', function () {
    $user = Socialite::driver('facebook')->user();
});
```

LOGIN USING SOCIAL MEDIA

Add link for social media registration in your blade file, example

```
<div class="col-md-6">
    <a href="{{ route('facebookRedirect') }}">Login with
facebook</a>
</div>
```

CLONE YOUR GITHUB TO ANOTHER COMPUTER

Clone the repository on your second computer using the following command:

```
git clone https://github.com/your-username/your-repository.git
```

Replace your-username and your-repository with your GitHub username and repository name.

CLONE YOUR GITHUB TO ANOTHER COMPUTER

- Navigate to the Project Directory
- Install Dependencies

```
composer install
```

CLONE YOUR GITHUB TO ANOTHER COMPUTER

- Configure Environment

```
cp .env.example .env
```

Edit the .env file with the correct configuration for your second computer.

CLONE YOUR GITHUB TO ANOTHER COMPUTER

- Generate Application Key

```
php artisan key:generate
```

- Run Migrations and Seeders (if applicable)
- Remember, the .env file and some configuration details may be specific to each environment, so ensure that your .env file on the second computer is configured appropriately for that system.

PUBLISH YOUR PROJECT

https://www.youtube.com/watch?v=dpJDV25tptw&ab_channel=TheCodeholic

WHAT YOU SHOULD TAKE CARE ABOUT AFTER PUBLISHING

1. Environment Configuration:

- Update .env File:

Ensure that the .env file on the live server contains the correct configurations for the production environment, including database settings, application key, and other environment-specific variables.

- Debug Mode:

Set APP_DEBUG=false in the .env file to disable debugging in the production environment.

WHAT YOU SHOULD TAKE CARE ABOUT AFTER PUBLISHING

2. Security:

- App Key:

Generate a new application key using the following command to ensure the security of encrypted data:

```
php artisan key:generate --ansi
```

WHAT YOU SHOULD TAKE CARE ABOUT AFTER PUBLISHING

- Secure Your .env File:

Ensure that your .env file is not accessible from the web. Move it to a directory outside the public web root or restrict access to it.

- Secure Database Credentials:

Use strong and secure database credentials. Avoid using default usernames and passwords.

- HTTPS:

Enable HTTPS to encrypt data in transit. Obtain and install an SSL certificate for your domain.

- Update Dependencies:

Keep all dependencies, including Laravel and its packages, up to date to benefit from security updates.

WHAT YOU SHOULD TAKE CARE ABOUT AFTER PUBLISHING

3. Performance:

- Optimize Autoloader:

Run the following command to optimize the Composer autoloader:

```
composer install --optimize-autoloader --no-dev
```

WHAT YOU SHOULD TAKE CARE ABOUT AFTER PUBLISHING

- Optimize Configuration Files:

Run the following command to cache configuration files for better performance:

```
php artisan config:cache
```

WHAT YOU SHOULD TAKE CARE ABOUT AFTER PUBLISHING

- Optimize Route Cache:

If you have a large number of routes, run the following command to cache them:

```
php artisan route:cache
```

WHAT YOU SHOULD TAKE CARE ABOUT AFTER PUBLISHING

4. Database:

- Run Migrations:

Run migrations on the live server to create database tables:

```
php artisan migrate --force
```

WHAT YOU SHOULD TAKE CARE ABOUT AFTER PUBLISHING

- **Seed the Database:**

If your application uses seeders to populate the database with initial data, run:

```
php artisan db:seed --force
```

WHAT YOU SHOULD TAKE CARE ABOUT AFTER PUBLISHING

5. Error Handling:

- Custom Error Pages:

Create custom error pages for a better user experience in case of errors. Customize the resources/views/errors directory.

WHAT YOU SHOULD TAKE CARE ABOUT AFTER PUBLISHING

6. Caching:

- Clear Caches:

Clear various caches before deploying:

```
php artisan cache:clear

php artisan config:clear
```

WHAT YOU SHOULD TAKE CARE ABOUT AFTER PUBLISHING

7. File Permissions:

- Set Proper File Permissions:

Ensure that directories like storage and bootstrap/cache have the correct write permissions.

```
chmod -R 775 storage bootstrap/cache
```

KOICA
Korea International Cooperation Agency

السويد
Sverige

ECTD
Egyptian Council
For Training & Development

العمل اللائق للمرأة

منظمة العمل الدولية

هيئة الأمم المتحدة للمرأة

المجلس القومي للمرأة
THE NATIONAL COUNCIL FOR WOMEN