Authentication

Built-in Authentication System: Laravel provides an out-of-the-box solution for user authentication, including registration, login, and password reset.
Easy Customization: Customize views, validation rules, and user redirections to fit specific project needs.
User Providers and Guards: Define how users are retrieved from your database and how they are authenticated.
Password Hashing and Encryption: Secure user passwords using robust hashing algorithms.
Password Reset Functionality: Simplified password reset process with secure token generation and email notifications.
Email Verification: Built-in support for new user email verification to ensure email address authenticity.
Middleware for Access Control: Protect routes with middleware to restrict access to authenticated users or guests.
Socialite for OAuth: Integration with Laravel Socialite for OAuth authentication with platforms like Facebook, Google, and Twitter.
Two-Factor Authentication: Extendable to support two-factor authentication for increased security.
Remember Me Functionality: Provides 'Remember Me' feature for convenient, yet secure, long-term authentication.
API Authentication with Laravel Sanctum: Secure API token handling for SPA (Single Page Application) or mobile applications.
Event-Based Authentication Actions: Trigger events on login, logout, and registration to perform custom actions.

Built-in Authentication

- **Quick Setup**: Laravel provides a quick way to scaffold all of the routes and views you need for authentication using:
  - `composer require laravel/ui`
  - `php artisan ui vue --auth`
- **Features**:
  - **Login/Registration**: Ready-to-use views and controllers for user registration and login.
  - **Password Reset**: Automated password reset functionality.
  - **Email Verification**: Easy to implement email verification for new accounts.
- **Customization**: Built-in auth system is fully customizable - from the validation rules to the user-facing views.
- **Security**: Secure session and token-based authentication.
- **Application Starter Kits**:
  - Laravel Breeze.
  - Laravel Jetstream.
  - Laravel Fortify.

Custom Authentication

- **When to Use**: When built-in features don't match specific requirements.
- **Steps for Custom Authentication**:
  - **Define Routes**: Create custom routes for login, registration, etc.
  - **Create Controllers**: Implement your own logic for user authentication.
  - **Managing User Sessions**: Utilize Laravel's session drivers to manage authenticated user sessions.
- **Custom User Providers**: For entirely different authentication backends, you can define custom user providers.

Custom Authentication Example

```php
// In LoginController.php
public function authenticate(Request $request)
{
    $credentials = $request->validate([
        'email' => 'required|email',
        'password' => 'required',
    ]);

    if (Auth::attempt($credentials)) {
        $request->session()->regenerate();
        return redirect()->intended('dashboard');
    }

    return back()->withErrors([
        'email' => 'The provided credentials do not match our records.',
    ]);
}
```

Maharah
Code Your Future

Creating Middleware

- **Definition**: Middleware acts as a filter for HTTP requests entering your application.
- **Creation**: Use Artisan command **`php artisan make:middleware ExampleMiddleware`**.
- **Implementation**: Middleware is defined in the **`handle`** method.

```php
// app/Http/Middleware/ExampleMiddleware.php
public function handle($request, Closure $next)
{
    // Custom logic here
    return $next($request);
}
```

Performs actions or checks before the request is handled by the application.

Registering Middleware

- **Global Middleware**: Register in **app/Http/Kernel.php** in the **$middleware** array. Applied to all HTTP requests.
- Route Middleware: Register in **app/Http/Kernel.php** in the **$routeMiddleware** array. Apply to specific routes.
- Group Middleware: Useful for grouping middleware under a key for easy assignment to routes.

```
// In app/Http/Kernel.php
protected $routeMiddleware = [
   'example' =>
\App\Http\Middleware\ExampleMiddleware::class,
];
```

Security Best Practices

- **Validation**: Always validate user input to prevent SQL injections and cross-site scripting (XSS).
- **CSRF Protection**: Utilize CSRF tokens in forms to prevent cross-site request forgery.
- **Password Hashing**: Use Laravel's built-in functions for password hashing.
- **Limiting Exposure**: Minimize the exposure of sensitive data and use environment files for configuration.
- **HTTPS**: Enforce HTTPS in production to secure data transmission.
- **Regular Updates**: Keep Laravel and its dependencies updated for the latest security patches.

Error Handling

- **Exception Handling**: Laravel's exception handler is located in app/Exceptions/Handler.php.
- **Custom Error Pages**: Create custom error views in resources/views/errors/.
- **Logging**: Utilize Laravel's logging capabilities to log errors.
- **Reporting and Rendering Methods**: Customize how exceptions are reported and rendered.
- **Handling Validation Errors**: Automatically handled by Laravel, with easy ways to display validation errors in views.

```php
// Custom render method in app/Exceptions/Handler.php
public function render($request, Exception $exception)
{
    if ($exception instanceof CustomException) {
        return response()->view('errors.custom', [], 500);
    }

    return parent::render($request, $exception);
}
```

Exercise: Add User Registration and Login

03

Mission

Enhance the book rental library to allow user registration and login. Implement middleware to restrict certain actions (like renting a book) to logged-in users. Focus on security aspects to protect user data and handle errors gracefully.

Artisan Console

- **What is Artisan?**: Artisan is Laravel's command-line interface that provides a number of helpful commands for developing a Laravel application.
- **Key Features**:
  - Automates repetitive tasks like database migrations, seeding, and testing.
  - Simplifies common tasks related to Laravel development.
- **Accessing Artisan**: Run **`php artisan`** from the command line in your project directory to see a list of available commands.

Common Artisan Commands

- **Starting a Development Server**: `php artisan serve`
- **Creating Controllers, Models, and Other Classes**:

  `php artisan make:controller ControllerName`
  - `php artisan make:model ModelName`
- **Database Migrations and Seeding**:
  - `php artisan migrate`
  - `php artisan db:seed`
- **Tinker (Interactive Shell)**: `php artisan tinker`
- **Clearing Cache**: `php artisan cache:clear`
- **Viewing Routes**: `php artisan route:list`

Localization and Internationalization

- **Purpose**: Laravel provides support for localization (L10n) and internationalization (I18n) to create multilingual applications.
- **Language Files**: Store translation strings in files within **`resources/lang`** directory.
- Using Translation Strings:
  - Use **`__('key')`** or **`@lang('key')`** in Blade templates.
- **Locale Switching**: Set the App locale to change languages dynamically.

```
// resources/lang/en/messages.php
return [
    'welcome' => 'Welcome to our application',
];

// Using in a Blade file
<p>{{ __('messages.welcome') }}</p>
```

Mail Configuration

- **Configuration File**: Mail settings are located in config/mail.php. Customize settings to use different mail drivers like SMTP, Mailgun, etc.
- **Environment Variables**: Set mail configuration in **.env** file for security and flexibility.
- **Drivers and Services**: Supports SMTP, Mailgun, Postmark, Amazon SES, and others.

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null
MAIL_FROM_ADDRESS="example@example.com"
MAIL_FROM_NAME="${APP_NAME}"
```

Sending Emails

- **Mailables**: Use Artisan to create a Mailable class with **`php artisan make:mail ExampleMail`**.
- **Writing Email Content**: Customize the email content in the Mailable class.
- **Sending Email**: Use the Mail facade to send the email.

```php
// In ExampleMail.php
public function build()
{
    return $this->from('example@example.com')
                ->view('emails.example');
}

// Sending the email
Mail::to('recipient@example.com')->send(new ExampleMail());
```

Events and Listeners

- **Events**: Used to announce that something has happened in the application.
- **Listeners**: Contains the logic that responds to the event.
- Creating Events and Listeners: Use Artisan commands like **`php artisan make:event ExampleEvent`** and **`php artisan make:listener ExampleListener`**.
- **Registering Events and Listeners**: In **`EventServiceProvider`**.
- **Example Usage**:
  - An **`OrderShipped`** event and a **`SendShipmentNotification`** listener.
- Dispatching Events:

```
event(new OrderShipped($order));
```

Task Scheduling

Laravel provides a powerful task scheduling feature that allows you to automate recurring tasks within your application.

- **Definition**

  Task scheduling allows you to define commands or closures to be executed at specific intervals or times, such as running a cleanup script daily or sending reminder emails weekly.

- **Configuration**

  Laravel's task scheduling is managed through the `App\Console\Kernel` class. You define your scheduled tasks within the schedule method, specifying the desired frequency and the command or closure to be executed.

- **Cron Expression Syntax**

  Laravel uses the standard cron expression syntax to define the schedule. This syntax allows you to specify precise intervals, such as every minute, hourly, daily, weekly, monthly, or custom intervals.

Task Scheduling Example

```
// Define a scheduled task
protected function schedule(Schedule $schedule)
{
    $schedule->command('cleanup:files')->daily();
    $schedule->command('send:reminders')->weekly();
}

// Run the scheduled tasks
 php artisan schedule:run
```

File Storage

Laravel offers a flexible and efficient file storage system, providing seamless integration with various storage providers.

- **Configuration**

  Laravel's file storage configuration is managed through the `config/filesystems.php` file. You can specify the default storage driver, as well as configure multiple disks with different settings and providers.

- **Storage Methods**

  Laravel provides a unified API for working with files, allowing you to perform common operations such as reading, writing, deleting, and moving files across different storage providers.

- **Supported Drivers**

  Laravel supports various storage drivers out of the box, including local storage, Amazon S3, FTP, SFTP, and more. You can easily switch between drivers based on your application's needs.

File Storage Example

```php
// Storing a file
Storage::disk('s3')->put('file.txt', $contents);

// Retrieving a file
$contents = Storage::disk('s3')->get('file.txt');

// Deleting a file
Storage::disk('s3')->delete('file.txt');
```

Exercise: Localization, Notifications, Events, and File Uploads

04

Mission

Enhance the book rental library application by implementing various advanced features. Add support for different languages, send email notifications to users, utilize event-driven programming for various actions, and manage file storage for digital assets like book covers or digital copies.