ERROR HANDLING

Error handling is a critical aspect of any web application, and Laravel provides a comprehensive set of tools and features for handling errors effectively. Here's an overview of various aspects of error handling in Laravel:

1. Exception Handling:

1. Exception Reporting:

Laravel automatically logs exceptions and provides detailed error pages during development. The logs are stored in the `storage/logs` directory.

2. Customizing Exception Handling

Customize exception handling in the `App\Exceptions\Handler` class. Override the `render` method to define how exceptions are displayed.

```php
 public function render($request, Throwable $exception)
{
        if ($exception instanceof MyCustomException) {
        return response()->view('errors.my_custom_error', [], 500);
        }

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

3. **HTTP Exceptions:**
Use Laravel's HTTP exception classes for common HTTP errors.

```php
abort(404, 'Resource not found.');
```

2. Logging:

1. Logging Errors:
Laravel logs exceptions automatically. You can view the logs in the `storage/logs` directory.

2. Custom Logging:
Customize logging by using Laravel's logging facilities. You can log messages at different levels (info, warning, error) using the `Log` facade.

```php
use Illuminate\Support\Facades\Log;

Log::error('This is an error message.');
```

3. Handling Validation Errors:

1. Validation Exception:

Laravel automatically redirects users back to the previous page with validation errors. If you want to customize this behavior, you can catch the `ValidationException` in your exception handler.

4. Handling AJAX Errors:

1. AJAX Error Responses:

When handling AJAX requests, you can return JSON responses for errors.

```php
return response()->json(['error' => 'Unauthorized'], 401);
```

2. Exception Handling for AJAX

Customize the exception handling for AJAX requests in the exception handler.

```php
public function render($request, Throwable $exception)
{
    if ($request->ajax()) {
    return response()->json(['error' => 'Something went wrong.'], 500);
    }

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

5. Error Views:

1. Custom Error Views:

Customize the error views located in the `resources/views/errors` directory.

```php
resources/views/errors/404.blade.php
```

2. Rendering Views for Exceptions

You can render specific views for different exceptions in the exception handler.

```php
public function render($request, Throwable $exception)
{
    if ($exception instanceof MyCustomException) {
    return response()->view('errors.my_custom_error', [], 500);
}

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

6. Logging Stack Traces:

1. Stack Trace in Logs

Laravel includes detailed stack traces in the logs for easier debugging.

2. Debug Mode:

In the `.env` file, set `APP_DEBUG=true` to display detailed error pages with stack traces in development.

7. Custom Error Pages:

1. Custom 404 Page:

Customize the 404 page by creating a `404.blade.php` file in the `resources/views/errors` directory.

2. Maintenance Mode:

Enable maintenance mode for planned downtime. Customize the view in the `resources/views/errors` directory.

```bash
php artisan down --message="Upgrading the application"
php artisan up
```

These are some of the key aspects of error handling in Laravel. By leveraging Laravel's built-in features and customization options, you can create a robust error handling strategy for your application, ensuring a smooth and user-friendly experience.