Error handling and logging in CodeIgniter

Topics : Codeigniter
Written on March 01, 2024

Error handling and logging are critical aspects of any web application to ensure smooth operation, identify issues, and debug problems effectively. CodeIgniter provides various mechanisms for error handling and logging. Here's how you can implement error handling and logging in your CodeIgniter application:

1. Error Reporting:

Set the desired error reporting level in your `index.php` file located in the root directory of your CodeIgniter application:

```
error_reporting(E_ALL);
```

2. Error Logging Configuration:

Configure error logging settings in your `config.php` file located in `application/config/config.php`:

```
 $config['log_threshold'] = 1; // Error logging threshold (0 = no logging, 1 = errors, 2 = debug)
$config['log_path'] = APPPATH . 'logs/'; // Log file directory
$config['log_file_extension'] = 'log'; // Log file extension
```

3. Error Logging:

Use CodeIgniter's logging library to log errors, warnings, and other messages to files. You can log messages using different log levels (e.g., DEBUG, INFO, ERROR) and categorize logs for easier debugging:

```
$this->load->library('logging');
$this->logging->log_message('error', 'An error occurred.');
```

4. Exception Handling:

Implement exception handling in your application to catch and handle runtime errors gracefully. Use try-catch blocks to handle exceptions and log detailed error messages, stack traces, and context information for easier debugging:

```
 try {
    // Code that may throw an exception
} catch (Exception $e) {
    log_message('error', $e->getMessage());
}
```

5. Displaying Errors:

Configure error display settings in your `config.php` file to control how errors are displayed in your application:

```
 $config['log_errors'] = TRUE; // Log PHP errors
$config['display_errors'] = FALSE; // Display PHP errors (set to FALSE in production)
$config['error_reporting'] = E_ALL; // Error reporting level
```

6. Custom Error Pages:

Create custom error pages for different HTTP error codes (e.g., 404 Not Found, 500 Internal Server Error) to provide a better user experience and help users navigate errors:

```
$route['404_override'] = 'errors/error_404'; $route['500_override'] = 'errors/error_500';
```

7. Error Handling in Controllers:

Handle errors and exceptions in your controllers by wrapping controller methods with try-catch blocks and logging errors as needed:

```
 try {
    // Controller method logic
} catch (Exception $e) {
    log_message('error', $e->getMessage());
}
```

8. Error Logging and Monitoring:

Regularly monitor error logs and review error messages to identify and troubleshoot issues in your application. Set up alerts or notifications to be notified of critical errors in real-time.

9. Debugging Tools:

Use debugging tools like Xdebug, CodeIgniter Profiler, or browser developer tools to inspect and debug PHP code, database queries, and client-side issues effectively.

10. Testing and Validation:

Test error handling and logging functionality thoroughly in different environments (e.g., development, staging, production) to ensure that errors are logged correctly and handled gracefully.