**🧩 The Core Idea**

When you build a **Spring MVC** or **Spring REST** application,  
Spring Boot already provides a **predefined controller** behind the scenes to handle **errors and exceptions** — even if *you* don’t define any.

That’s why, when something goes wrong (like a 404 or 500),  
you see a **default white error page** — called the **“Whitelabel Error Page.”**

**⚙️ 1. What is the “Predefined Controller”?**

Spring Boot provides a built-in class called  
BasicErrorController  
(from the package org.springframework.boot.autoconfigure.web.servlet.error).

This class acts as a **default global error handler** for all exceptions and HTTP errors.

**📘 What It Does**

* It **intercepts** any unhandled exceptions or HTTP errors  
  (like 404 Not Found, 500 Internal Server Error, 400 Bad Request, etc.)
* It returns:
  + A **“Whitelabel Error Page”** (for web browser requests)
  + A **JSON error response** (for REST API calls)

**💡 2. The Whitelabel Error Page**

If you run a Spring Boot app and hit a wrong URL like:

http://localhost:8080/unknown-url

You’ll see something like:

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

There was an unexpected error (type=Not Found, status=404).

✅ This is generated automatically by Spring Boot’s internal **BasicErrorController**.  
It’s a *simple default HTML page* used when your app doesn’t define a custom error handler.

**⚡ 3. Why Is It Called “Whitelabel”?**

* “Whitelabel” just means a **generic template** (no branding, no styling).
* It’s meant as a **placeholder page** during development — not a final production UI.

You can think of it as Spring saying:

“You didn’t define your own error page, so here’s a simple default one until you do.”

**🧰 4. How It Works (Flow)**

Let’s see how Spring Boot handles an error internally:

1. Request comes to controller (e.g., /students/999)
2. Exception occurs → StudentNotFoundException
3. Spring tries to find a handler for it:
   * If you have @RestControllerAdvice → that runs first.
   * If not, Spring forwards the request to /error.
4. /error is automatically handled by BasicErrorController.
5. That controller returns:
   * JSON (if REST request)
   * HTML “Whitelabel Error Page” (if browser request)

**🧠 5. Default JSON Response Example**

If you call a REST API endpoint that doesn’t exist:

**GET** http://localhost:8080/wrong

You’ll get this response in Postman:

{

"timestamp": "2025-10-21T20:25:44.123+00:00",

"status": 404,

"error": "Not Found",

"path": "/wrong"

}

✅ This is **generated automatically** by the BasicErrorController.  
No need to write any code for it — it’s built into Spring Boot.

**⚙️ 6. Customizing the Default Error Behavior**

You can override this default handling in several ways:

**🧩 Option 1 — Create a @RestControllerAdvice**

(Recommended for APIs)

@RestControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(Exception.class)

public ResponseEntity<Map<String, Object>> handleAll(Exception ex) {

Map<String, Object> body = new HashMap<>();

body.put("error", "Custom Error");

body.put("message", ex.getMessage());

return new ResponseEntity<>(body, HttpStatus.INTERNAL\_SERVER\_ERROR);

}

}

**🧩 Option 2 — Create Custom HTML Error Page**

(Recommended for web apps)

Add:

src/main/resources/templates/error.html

Spring Boot will automatically show this HTML file instead of the default Whitelabel page.

**🧩 Option 3 — Disable Whitelabel Page**

In application.properties:

server.error.whitelabel.enabled=false

This turns off the default white error page (useful when you have your own custom one).

**🧱 7. Summary Table**

| **Concept** | **Description** |
| --- | --- |
| **BasicErrorController** | The built-in controller that handles all unhandled errors. |
| **Whitelabel Error Page** | The default simple HTML error page shown for web requests. |
| **Default JSON Response** | Shown for REST API requests when an error occurs. |
| **Customizing** | Use @RestControllerAdvice for APIs or custom HTML page for web. |

**🔄 Real-Life Analogy**

Think of the **BasicErrorController** like a **default receptionist** in a company.

* If a visitor comes to an **unknown department** (wrong URL),  
  the receptionist (BasicErrorController) steps in and says:  
  “Sorry, that department doesn’t exist. Here’s some basic info.”
* But if you have your own **customer service team** (@RestControllerAdvice),  
  they’ll take care of such cases with a **better, customized message.**

✅ **In Short:**

Spring Boot provides a **predefined controller (BasicErrorController)**  
that handles all unhandled exceptions or HTTP errors by default,  
showing a **Whitelabel Error Page** or **default JSON response**.

You can override this behavior using your own  
@RestControllerAdvice, @ExceptionHandler, or error.html file.

**🧩 What is Spring Boot DevTools?**

**Spring Boot DevTools** is a **developer productivity tool**.  
It helps you **see your code changes instantly** without needing to **manually stop and restart** your Spring Boot application every time you change a file.

In simple words:

It gives **auto-reload** and **hot deployment** features during development.

**⚙️ What It Does (Key Features)**

**1️⃣ Automatic Restart (Hot Reload)**

* When you make changes to **Java code**, **HTML**, or **templates**,  
  DevTools automatically **restarts** your Spring Boot app.
* No need to stop and run manually each time.

📌 Example:  
You edit your controller method → save it → Spring Boot automatically restarts in a second or two → your browser or Postman immediately reflects the new change.

**2️⃣ LiveReload Support**

* DevTools includes a **LiveReload server** that works with browsers.
* It can **auto-refresh** your web pages when changes are detected (useful in web projects with Thymeleaf, React, or Angular).

**3️⃣ Automatic Cache Disable in Development**

* It **disables caching** of static resources (like CSS, JS, templates) while developing.
* So you don’t need to clear your browser cache to see changes.

**4️⃣ Faster Development Cycle**

* Saves a lot of time in development phase.
* You focus on coding instead of repeatedly restarting the server.

**🧰 How to Use It**

**📍 Step 1: Add Dependency in pom.xml**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

✅ scope=runtime → It will **not** be included in production JAR/WAR.  
✅ optional=true → Ensures it is used only in **development**, not in **production**.

**📍 Step 2: Enable Automatic Build in IDE**

**In Eclipse or IntelliJ:**

* When you save your Java file, DevTools restarts automatically.  
  (If not, press **Ctrl + F9** in IntelliJ or **Project → Build Automatically** in Eclipse.)

**📍 Step 3: Run the Application Normally**

You can run it as usual:

mvn spring-boot:run

Now, modify any file (for example, controller or HTML).  
The application restarts automatically in 1–2 seconds — no manual restart needed.

**⚡ Example Scenario**

Without DevTools 👎

* You change @GetMapping("/hello") → stop server → recompile → start server again → test again.

With DevTools 👍

* Just edit → save → app restarts automatically → test immediately.

**⚠️ Important Notes**

| **Point** | **Description** |
| --- | --- |
| 🧱 Only for Development | Don’t use in Production — it’s meant for developers only. |
| 💾 Works on Save | Auto-restart triggers when files are saved (not on typing). |
| 🧩 Restart vs Reload | It doesn’t just reload classes; it actually restarts the app with a smart restart classloader. |
| 🌍 LiveReload Browser Plugin | Optional, helps auto-refresh web pages when files change. |

**💡 Real-Life Analogy**

Think of DevTools as an **auto-refresh button** for your backend code.  
When you make any change and hit *save*, it’s like someone automatically clicks *Restart Application* for you — making development smooth and fast. 🚀

**✅ Quick Summary**

| **Feature** | **Purpose** |
| --- | --- |
| Automatic Restart | Reload app on code changes |
| LiveReload | Auto-refresh browser |
| Cache Disable | Prevent stale resources |
| Dev-only Dependency | Excluded from production builds |