Error Handling: Custom Error Classes

1. Problem Statement

A cloud-based application leverages MongoDB Atlas for its backend data storage and management. As the application scales, it faces challenges in robustly handling API errors—such as invalid requests, resource not found, rate limiting, or internal server errors—when interacting with the Atlas Administration API. Without proper error handling, users receive cryptic or unhelpful error messages, making troubleshooting difficult and degrading the user experience.

For example, when a user attempts to list projects but exceeds their API quota, the application should clearly indicate the issue and guide the user or administrator toward resolution, rather than failing silently or crashing. Similarly, if a requested resource does not exist, the application should return a meaningful 404 response.

This scenario underscores the need for comprehensive error handling that leverages custom error classes and appropriate HTTP status codes, especially when using the Atlas Go SDK for MongoDB Atlas.

2. Learning Objectives

By the end of this tutorial, you will be able to:

- **Understand the importance and structure of error handling in MongoDB Atlas Go SDK applications.**
- **Fetch and interpret API error objects returned by the Atlas Go SDK.**
- **Check for specific error codes and HTTP status codes in your error handling logic.**
- **Implement custom error handling logic to map API errors to meaningful client responses.**
- **Mock errors for testing and debugging purposes.**
- **Differentiate between client (4xx) and server (5xx) errors and respond accordingly.**

3. Concept Introduction with Analogy

Imagine your application as a customer service agent at a large company. When customers (users) make requests, sometimes things go wrong: a form is incomplete (invalid request), a file is missing (resource not found), or the system is temporarily unavailable (server error).

Error handling is like training the agent to:

- **Recognize different types of issues** (e.g., missing information, system outages).
- **Provide clear, actionable feedback** to the customer.
- **Escalate unresolved issues** to the technical team with detailed logs.

Custom error classes and HTTP status codes are the tools that help the agent categorize each problem and respond appropriately.

4. Technical Deep Dive

Error Representation in Atlas Go SDK:
Errors returned by the Atlas Go SDK are represented by the `ApiErrorObject` struct. This object contains fields such as `Error` (HTTP status code), `ErrorCode` (Atlas-specific error code), `Detail` (detailed error message), and `Reason` (brief description).

Fetching and Casting Errors:
To obtain detailed error information, use the `admin.AsError(err)` method to cast a returned error to an `ApiErrorObject`:

```
apiError, ok := admin.AsError(err)
if ok {
    fmt.Println(apiError)
}
```

Checking for Specific Error Codes:
To check for a specific Atlas error code (e.g., `MAXIMUM_INDEXES_FOR_TENANT_EXCEEDED`):

```
if admin.IsErrorCode(err, "MAXIMUM_INDEXES_FOR_TENANT_EXCEEDED") {
    // Handle exceeded index limit
}
```

Checking for HTTP Status Codes:
To check for a specific HTTP status code (e.g., 404 Not Found):

```
if ok && apiError.GetError() == 404 {
    // Handle not found
}
```

Mocking Errors:
For testing, you can mock errors by creating an instance of `GenericOpenAPIError` and setting custom error details:

```
apiError := admin.GenericOpenAPIError{}
apiError.SetModel(admin.ApiError{
    Detail:    admin.PtrString("Error when listing clusters"),
    Error:     admin.PtrInt(400),
    ErrorCode: admin.PtrString("CLUSTERS_UNREACHABLE"),
    Reason:    admin.PtrString("Clusters unreachable"),
})
apiError.SetError("Mocked error")
```

---

5. Step-by-Step Data Modeling & Code Walkthrough

1. Initialize the SDK Client:

```
sdk, err := admin.NewClient(
    admin.UseDigestAuth(apiKey, apiSecret))
if err != nil {
    log.Fatalf("Failed to initialize client: %v", err)
}
```

2. Make an API Request and Handle Errors:

```
projects, response, err := admin.ProjectsApi.ListProjects(ctx).Execute()
if err != nil {
    apiError, ok := admin.AsError(err)
    if ok {
        switch apiError.GetError() {
        case 404:
            log.Printf("Resource not found: %v", apiError.GetDetail())
            // Return 404 to client
        case 400:
            log.Printf("Bad request: %v", apiError.GetDetail())
            // Return 400 to client
        case 500:
            log.Printf("Internal server error: %v", apiError.GetDetail())
            // Return 500 to client
        default:
            log.Printf("Unhandled error: %v", apiError.GetDetail())
            // Return generic error to client
        }
    } else {
        log.Printf("Unknown error: %v", err)
        // Return 500 to client
    }
}
```

3. Check for Specific Error Codes:

```
if admin.IsErrorCode(err, "MAXIMUM_INDEXES_FOR_TENANT_EXCEEDED") {
    // Handle exceeded index limit
}
```

4. Mock Errors for Testing:

```
apiError := admin.GenericOpenAPIError{}
apiError.SetModel(admin.ApiError{
    Detail:    admin.PtrString("Error when listing clusters"),
    Error:     admin.PtrInt(400),
    ErrorCode: admin.PtrString("CLUSTERS_UNREACHABLE"),
    Reason:    admin.PtrString("Clusters unreachable"),
})
apiError.SetError("Mocked error")
// Use apiError as a mock error in tests
```

---

6. Challenge

Your Turn!

- **Implement error handling** for a request to list clusters in MongoDB Atlas.
- **Check for a specific error code** (e.g., `CLUSTERS_UNREACHABLE`) and return a custom error message to the client.
- **Mock an error** for a 404 response and test your error handling logic.
- **Extend your error handler** to support additional HTTP status codes (e.g., 403 Forbidden, 429 Too Many Requests).

---

7. Common Pitfalls & Best Practices

Pitfall	Best Practice
Not checking for specific error codes	Use `IsErrorCode` to handle known error cases
Ignoring HTTP status codes	Map API errors to appropriate 4xx/5xx status codes
Returning raw error messages to clients	Provide user-friendly, actionable error messages
Not logging detailed error information	Log errors for debugging and monitoring
Failing to mock errors for testing	Use `GenericOpenAPIError` to simulate error scenarios

8. Optional: Programmer’s Workflow Checklist

- **Initialize the Atlas Go SDK client with proper authentication.**
- **Make API requests and handle errors using `AsError`.**
- **Check for specific error codes and HTTP status codes.**
- **Return meaningful error messages and status codes to clients.**
- **Log detailed error information for debugging.**
- **Mock errors for testing and validation.**
- **Document error handling logic and expected responses for your team.**