Chapter: Running and Debugging (30 mins)

Chapter Objectives

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

- Run BDD tests using `go test -v`
- Understand what undefined steps are and how to resolve them
- Interpret common `godog` error messages and failure scenarios
- Debug failing steps efficiently in Go

Theoretical Concepts

BDD tests in Go using `godog` are executed via the `go test` command when integrated with `godog.TestSuite`.

Why Use `go test -v`?

- Runs BDD scenarios just like unit tests
- Provides verbose, human-readable output
- Captures success, failure, and skipped scenarios clearly

> ☑ `godog run` is **deprecated**. Use `go test` instead for long-term support and better test integration.

Running the Tests

Navigate to your project root and run:

```
go test -v
```

You'll see something like:

```
=== RUN   TestFeatures
Feature: User Login

  Scenario: Successful login
    Given a registered user with username "john" and password "secret"
    When the user logs in with username "john" and password "secret"
    Then the login should be successful

1 scenarios (1 passed)
3 steps (3 passed)
```

📌 Output Breakdown

Section	Description
`=== RUN`	Go's native test block
`Feature:`	The name of your Gherkin feature
`Scenario:`	Test case in natural language
`Steps:`	Execution of each Gherkin step and whether it passed
`(x passed)`	Summary of test success

❌ Handling Undefined Steps

If you run your test with unimplemented steps, you'll see:

```
1 scenarios (1 undefined)
3 steps (3 undefined)
```

And code snippets like:

```go
func iDoSomething() error {
    return godog.ErrPending
}
```

☑ How to Fix

1. Copy the generated function stub
2. Paste it in your step definitions file
3. Write the actual Go logic inside
4. Register the step using `ctx.Step(...)`

🐞 Debugging Failures

When a test fails, you'll see output like:

```
Then the login should be successful
    login_steps.go:29
    Error: expected login to be successful but it failed
--- FAIL: TestFeatures
```

🔍 Interpretation

- The step failed because your Go logic returned an error
- You can trace the error back to the exact line and message
- Use `fmt.Printf(...)` or Go's `log` package to add debug output in your steps

---

💡 Real-World Analogy

> Think of Gherkin as the test case written by a business analyst.
> You're the developer ensuring that what's described is exactly what happens.
> If the user says "Login should succeed," but the test says "Login failed," you need to figure out why the logic didn't meet the expectation.

🛠 Example: Add Debug Print

```go
func (s *Suite) loginShouldBeSuccessful() error {
    fmt.Printf("Verifying login for: %s\n", s.enteredUsername)
    if !s.loginSuccess {
        return fmt.Errorf("login failed for user %s", s.enteredUsername)
    }
    return nil
}
```

Then rerun the test and inspect the debug output:

```
go test -v
```

🔄 Common Errors and Fixes

Error	**Cause**	**Fix**
`undefined step`	Missing step function	Define and register it
`panic: nil pointer dereference`	Uninitialized struct or map	Initialize before use
`expected X but got Y`	Assertion mismatch	Recheck input or logic
`exit status 1`	Generic failure	Check stack trace and test logic

🎯 Interview Questions

1. What is the purpose of `go test -v` in a godog project?
2. What causes steps to be reported as undefined?
3. How can you debug a failing BDD test in Go?
4. What does `godog.ErrPending` mean?

5. How do you interpret a failing scenario with multiple steps?

📺 Curated YouTube Videos

1. BDD Testing with godog: Run and Debug
2. Debugging Go Applications (GoLand/VSCode)
3. Effective Logging in Go
4. Understanding go test Output

📚 Additional Resources

- godog GitHub
- Go test flags
- Go error handling best practices

✅ Summary

- Use `go test -v` for executing BDD tests with detailed output
- Handle undefined steps by implementing and registering them
- Debug failures using verbose error messages, `fmt.Print`, and step-by-step trace
- Understand common causes of failure and how to fix them

> In the next chapter, we'll explore Scenario Outlines, Background steps, and reusable step organization strategies.