Chapter: Mini Project – End-to-End BDD with Go (GraphQL APIs)

🎯 Chapter Objective

In this mini project, you'll:

- Build a **GraphQL-based User Data Management Service** in Go
- Use **Behavior Driven Development (BDD)** to define, test, and validate behavior
- Learn to integrate **godog** for end-to-end scenario testing

---

🧠 Theoretical Concepts

📌 Why GraphQL?

GraphQL is a flexible query language for APIs that allows clients to specify exactly what data they need.

- ✅ No over-fetching or under-fetching
- ✅ Ideal for frontend-driven APIs
- ✅ Schema-driven and introspectable

📌 Why BDD with GraphQL?

BDD allows you to write **business-readable** specs using `.feature` files (Gherkin), which are mapped to Go functions.

Example Gherkin scenario:

```
Scenario: Create a new user
  When I send a GraphQL mutation to create a user with name "Alice" and email "alice@example.com"
  Then the response should contain user with name "Alice"
```

🏗 Project Goals

Build the following GraphQL operations:

Operation	Type	Description
`users`	`query`	Get all users
`user(id)`	`query`	Get user by ID
`createUser(name, email)`	`mutation`	Create a user

Operation	Type	Description
`updateUser(id, name, email)`	`mutation`	Update user
`deleteUser(id)`	`mutation`	Delete a user
`deleteAllUsers`	`mutation`	Delete all users

🛠 Setup Instructions

1. Initialize project

```bash
mkdir graphql-user-bdd
cd graphql-user-bdd
go mod init github.com/yourname/graphql-user-bdd
```

2. Install dependencies

```bash
go get github.com/graphql-go/graphql
go get github.com/graphql-go/handler
go get github.com/cucumber/godog@latest
```

✍ Step 1: Feature File

Create `features/user_graphql.feature`

```gherkin
Feature: GraphQL User Management

  Scenario: Create a new user
    When I send a GraphQL mutation to create a user with name "Alice" and email "alice@example.com"
    Then the response should contain user with name "Alice"

  Scenario: Fetch all users
    When I send a GraphQL query to fetch all users
    Then the response should include at least one user

  Scenario: Update existing user
    When I send a GraphQL mutation to update user with ID 1 to name "Alicia" and email "alicia@example.com"
    Then the response should contain user with name "Alicia"

  Scenario: Delete user by ID
    When I send a GraphQL mutation to delete user with ID 1
    Then the response status should be success
```

🧩 Step 2: GraphQL Schema (main.go)

Use `github.com/graphql-go/graphql` to define your schema.

Include:

- User Type
- Queries (`users`, `user`)
- Mutations (`createUser`, `updateUser`, `deleteUser`, `deleteAllUsers`)
- Start HTTP server on `localhost:8080/graphql`

> Would you like a working example `main.go` for this? Let me know!

🧪 Step 3: Step Definitions

Create `stepdefs/user_graphql_steps.go`

Example:

```go
func (s *userSuite) iSendGraphqlMutationToCreateUser(name, email string) error
{
    query := fmt.Sprintf(`{"query": "mutation { createUser(name: \"%s\", email:
\"%s\") { name } }"}`, name, email)
    resp, err := http.Post("http://localhost:8080/graphql", "application/json",
bytes.NewBuffer([]byte(query)))
    s.response = resp
    return err
}
```

Match Gherkin steps with Go functions using:

```go
ctx.Step(`^I send a GraphQL mutation to create a user with name "([^"]*)" and
email "([^"]*)"$`, s.iSendGraphqlMutationToCreateUser)
```

▶️ Step 4: Run the Server

```
go run main.go
```

In another terminal, run:

```
go test -v
```

BDD Testing Flow Recap

File	Role
`user_graphql.feature`	Defines human-readable scenarios
`user_graphql_steps.go`	Implements step logic in Go
`main.go`	Starts GraphQL API server
`godog_test.go`	Entry point for test suite

Interview Questions

1. What's the difference between GraphQL and REST?
2. How do you define a GraphQL mutation in Go?
3. How does BDD benefit API development?
4. How are `.feature` steps mapped to Go code in godog?
5. How would you test complex GraphQL responses in BDD?

Curated YouTube Videos

- GraphQL in Golang Crash Course
- GraphQL vs REST
- BDD with Godog and Go
- Testing GraphQL APIs

Resources

- godog GitHub
- graphql-go Docs
- Gherkin Syntax

Summary

- You wrote `.feature` specs describing a GraphQL API
- You implemented Go logic to run a GraphQL server and fulfill BDD steps
- You validated API behavior using Gherkin + godog
- You now understand how to test GraphQL endpoints using BDD

> Up next: Learn to use Scenario Outlines for testing multiple data sets with fewer steps.