Milad Khodabandehloo Follow
Jun 8, 2020 · 3 min read · Listen

Test suite for unit testing GORM in Go

Today most of frameworks that are used for backend development provide a test suite that makes it easy to test different aspects of an application such as **database interactions**. because testing an application is very important.

> *Jacob Kaplan-Moss, one of Django's original developers, says Code without tests is broken by design.*

Let's talk about how to test **database interactions** when you use GORM in **Go programming language** web frameworks such as Gin-Gonic.

Prerequisite

Let's take simple model `Blog` for example.

```
import "github.com/jinzhu/gorm"

type Blog struct {
   gorm.Model
   Title   string
   Content string
}

func GetModels() []interface{} {
 return []interface{}{&Blog{}}
}
```

function `GetModels` returns ar 52 f the database models.

Before getting started with how we can implement our desired test suite, we must add `testify` package in our project.

We use `suite` of testify to ease testing setup. If you are not yet familiar with `suite`, checkout the quote from the testify below.

> The `suite` package provides functionality that you might be used to from more common object oriented languages. With it, you can build a testing suite as a struct, build setup/teardown methods and testing methods on your struct, and run them with 'go test' as per normal.

Suite

Below is how we implement our test suite.

```
type ExampleSuite struct {
 suite.Suite
 db *gorm.DB
 tx *gorm.DB
}

func (t *ExampleSuite) SetupSuite() {
 db, err := gorm.Open("mysql", "root:my-secret-pw@/test?
charset=utf8&parseTime=True&loc=Local")
 if err != nil {
  log.Fatalln(err)
 }
 t.db = db
 for _, model := range GetModels() {
  t.db.AutoMigrate(model)
 }
}

func (t *ExampleSuite) TearDownSuite() {
 defer t.db.Close()
 for _, model := range GetModels() {
  t.db.DropTableIfExists(model)
 }
}

func (t *ExampleSuite) SetupTest() {
 t.tx = t.db.Begin()
}

func (t *ExampleSuite) TearDownTest() {
 t.tx.Rollback()
}
```

In the above code `ExampleSuite` *is the struct that would be our desired test suite.*

`SetupSuite` method attached to our test suite is the code that will run once before running tests. In this method we connect GROM to our test database and by `AutoMigrate` create the schema of our application database in the test database.

`TearDownSuite` is the code that will run once when all of the tests are finished so in this table we will erase the test tables in test database and then release the connection.

`SetupTest` is the code that will run before each test. In this method we start a database transaction.

`TearDownTest` is the code that will run after each test. In this method we roll back the transaction started in the `SetupTest` because database content must be isolated in a test from each others.

Testing

After implementing our test suite, now it's time to write our unit tests.

Let's have two unit tests for example.

```
func (t *ExampleSuite) TestExampleOne() {
 err := t.tx.Create(Blog{
  Title:   "some_title",
  Content: "some_content",
 }).Error
 require.NoError(t.T(), err)
}

func (t *ExampleSuite) TestExampleTwo() {
 err := t.tx.Create(Blog{
  Title:   "some_title",
  Content: "some_content",
 }).Error
 require.NoError(t.T(), err)
 var count int
 t.tx.Model(Blog{}).Count(&count)
 assert.Equal(t.T(), 1, count)
}
```

after attaching our desired unit test to the suite we can put this code in `main_test.go` and then run the following command.

```
go test -v
```

```
func TestSuite(t *testing.T) {
 s := new(ExampleSuite)
 suite.Run(t, s)
}
```

Here is repository for abovementioned code if it’s hard for you to read codes in seprated pieces.

Go Golang Gorm Go Programming Go Programming Language