CR1

// Sample test to show how to write a basic unit test.

package example1

import (

"net/http"

"testing"

)

const succeed = "\u2713"

const failed = "\u2717"

// TestDownload validates the http Get function can download content.

func TestDownload(t \*testing.T) {

url := "https://www.ardanlabs.com/blog/index.xml"

statusCode := 200

t.Log("Given the need to test downloading content.")

{

testID := 0

t.Logf("\tTest %d:\tWhen checking %q for status code %d", testID, url, statusCode)

{

resp, err := http.Get(url)

if err != nil {

t.Fatalf("\t%s\tTest %d:\tShould be able to make the Get call : %v", failed, testID, err)

}

t.Logf("\t%s\tTest %d:\tShould be able to make the Get call.", succeed, testID)

defer resp.Body.Close()

if resp.StatusCode == statusCode {

t.Logf("\t%s\tTest %d:\tShould receive a %d status code.", succeed, testID, statusCode)

} else {

t.Errorf("\t%s\tTest %d:\tShould receive a %d status code : %d", failed, testID, statusCode, resp.StatusCode)

}

}

}

}

CR2

// Sample test to show how to write a basic unit table test.

package example2

import (

"net/http"

"testing"

)

const succeed = "\u2713"

const failed = "\u2717"

// TestDownload validates the http Get function can download content and

// handles different status conditions properly.

func TestDownload(t \*testing.T) {

tt := []struct {

url string

statusCode int

}{

{"https://www.ardanlabs.com/blog/index.xml", http.StatusOK},

{"http://rss.cnn.com/rss/cnn\_topstorie.rss", http.StatusNotFound},

}

t.Log("Given the need to test downloading different content.")

{

for testID, test := range tt {

t.Logf("\tTest %d:\tWhen checking %q for status code %d", testID, test.url, test.statusCode)

{

resp, err := http.Get(test.url)

if err != nil {

t.Fatalf("\t%s\tTest %d:\tShould be able to make the Get call : %v", failed, testID, err)

}

t.Logf("\t%s\tTest %d:\tShould be able to make the Get call.", succeed, testID)

defer resp.Body.Close()

if resp.StatusCode == test.statusCode {

t.Logf("\t%s\tTest %d:\tShould receive a %d status code.", succeed, testID, test.statusCode)

} else {

t.Errorf("\t%s\tTest %d:\tShould receive a %d status code : %v", failed, testID, test.statusCode, resp.StatusCode)

}

}

}

}

}

CR3

// Sample test to show how to mock an HTTP GET call internally.

package example3

import (

"encoding/xml"

"fmt"

"net/http"

"net/http/httptest"

"testing"

)

const succeed = "\u2713"

const failed = "\u2717"

// feed is mocking the XML document we expect to receive.

var feed = `<?xml version="1.0" encoding="UTF-8"?>

<rss>

<channel>

<title>Going Go Programming</title>

<description>Golang : https://github.com/goinggo</description>

<link>http://www.goinggo.net/</link>

<item>

<pubDate>Sun, 15 Mar 2015 15:04:00 +0000</pubDate>

<title>Object Oriented Programming Mechanics</title>

<description>Go is an object oriented language.</description>

<link>http://www.goinggo.net/2015/03/object-oriented</link>

</item>

</channel>

</rss>`

// Item defines the fields associated with the item tag in

// the buoy RSS document.

type Item struct {

XMLName xml.Name `xml:"item"`

Title string `xml:"title"`

Description string `xml:"description"`

Link string `xml:"link"`

}

// Channel defines the fields associated with the channel tag in

// the buoy RSS document.

type Channel struct {

XMLName xml.Name `xml:"channel"`

Title string `xml:"title"`

Description string `xml:"description"`

Link string `xml:"link"`

PubDate string `xml:"pubDate"`

Items []Item `xml:"item"`

}

// Document defines the fields associated with the buoy RSS document.

type Document struct {

XMLName xml.Name `xml:"rss"`

Channel Channel `xml:"channel"`

URI string

}

// mockServer returns a pointer to a server to handle the mock get call.

func mockServer() \*httptest.Server {

f := func(w http.ResponseWriter, r \*http.Request) {

w.WriteHeader(200)

w.Header().Set("Content-Type", "application/xml")

fmt.Fprintln(w, feed)

}

return httptest.NewServer(http.HandlerFunc(f))

}

// TestDownload validates the http Get function can download content and

// the content can be unmarshaled and clean.

func TestDownload(t \*testing.T) {

statusCode := http.StatusOK

server := mockServer()

defer server.Close()

t.Log("Given the need to test downloading content.")

{

testID := 0

t.Logf("\tTest %d:\tWhen checking %q for status code %d", testID, server.URL, statusCode)

{

resp, err := http.Get(server.URL)

if err != nil {

t.Fatalf("\t%s\tTest %d:\tShould be able to make the Get call : %v", failed, testID, err)

}

t.Logf("\t%s\tTest %d:\tShould be able to make the Get call.", succeed, testID)

defer resp.Body.Close()

if resp.StatusCode != statusCode {

t.Fatalf("\t%s\tTest %d:\tShould receive a %d status code : %v", failed, testID, statusCode, resp.StatusCode)

}

t.Logf("\t%s\tTest %d:\tShould receive a %d status code.", succeed, testID, statusCode)

var d Document

if err := xml.NewDecoder(resp.Body).Decode(&d); err != nil {

t.Fatalf("\t%s\tTest %d:\tShould be able to unmarshal the response : %v", failed, testID, err)

}

t.Logf("\t%s\tTest %d:\tShould be able to unmarshal the response.", succeed, testID)

if len(d.Channel.Items) == 1 {

t.Logf("\t%s\tTest %d:\tShould have 1 item in the feed.", succeed, testID)

} else {

t.Errorf("\t%s\tTest %d:\tShould have 1 item in the feed : %d", failed, testID, len(d.Channel.Items))

}

}

}

}

CR4

// go test -run TestSendJSON -race -cpu 16

// Sample test to show how to test the execution of an internal endpoint.

package handlers\_test

import (

"encoding/json"

"net/http"

"net/http/httptest"

"testing"

"github.com/ardanlabs/gotraining/topics/go/testing/tests/example4/handlers"

)

const succeed = "\u2713"

const failed = "\u2717"

func init() {

handlers.Routes()

}

// TestSendJSON testing the sendjson internal endpoint.

func TestSendJSON(t \*testing.T) {

url := "/sendjson"

statusCode := 200

t.Log("Given the need to test the SendJSON endpoint.")

{

r := httptest.NewRequest("GET", url, nil)

w := httptest.NewRecorder()

http.DefaultServeMux.ServeHTTP(w, r)

testID := 0

t.Logf("\tTest %d:\tWhen checking %q for status code %d", testID, url, statusCode)

{

if w.Code != 200 {

t.Fatalf("\t%s\tTest %d:\tShould receive a status code of %d for the response. Received[%d].", failed, testID, statusCode, w.Code)

}

t.Logf("\t%s\tTest %d:\tShould receive a status code of %d for the response.", succeed, testID, statusCode)

var u struct {

Name string

Email string

}

if err := json.NewDecoder(w.Body).Decode(&u); err != nil {

t.Fatalf("\t%s\tTest %d:\tShould be able to decode the response.", failed, testID)

}

t.Logf("\t%s\tTest %d:\tShould be able to decode the response.", succeed, testID)

if u.Name == "Bill" {

t.Logf("\t%s\tTest %d:\tShould have \"Bill\" for Name in the response.", succeed, testID)

} else {

t.Errorf("\t%s\tTest %d:\tShould have \"Bill\" for Name in the response : %q", failed, testID, u.Name)

}

if u.Email == "bill@ardanlabs.com" {

t.Logf("\t%s\tTest %d:\tShould have \"bill@ardanlabs.com\" for Email in the response.", succeed, testID)

} else {

t.Errorf("\t%s\tTest %d:\tShould have \"bill@ardanlabs.com\" for Email in the response : %q", failed, testID, u.Email)

}

}

}

}

CR5

// go test -run ExampleSendJSON

// Sample to show how to write a basic example.

package handlers\_test

import (

"encoding/json"

"fmt"

"log"

"net/http"

"net/http/httptest"

)

// ExampleSendJSON provides a basic example example.

func ExampleSendJSON() {

r := httptest.NewRequest("GET", "/sendjson", nil)

w := httptest.NewRecorder()

http.DefaultServeMux.ServeHTTP(w, r)

var u struct {

Name string

Email string

}

if err := json.NewDecoder(w.Body).Decode(&u); err != nil {

log.Println("ERROR:", err)

}

fmt.Println(u)

// Output:

// {Bill bill@ardanlabs.com}

}

CR6

// go test -v

// go test -run TestDownload/statusok -v

// go test -run TestDownload/statusnotfound -v

// go test -run TestParallelize -v

// Sample test to show how to write a basic sub unit table test.

package example2

import (

"net/http"

"testing"

)

const succeed = "\u2713"

const failed = "\u2717"

// TestDownload validates the http Get function can download content and

// handles different status conditions properly.

func TestDownload(t \*testing.T) {

tt := []struct {

name string

url string

statusCode int

}{

{"statusok", "https://www.ardanlabs.com/blog/index.xml", http.StatusOK},

{"statusnotfound", "http://rss.cnn.com/rss/cnn\_topstorie.rss", http.StatusNotFound},

}

t.Log("Given the need to test downloading different content.")

{

for i, test := range tt {

tf := func(t \*testing.T) {

t.Logf("\tTest %d:\tWhen checking %q for status code %d", i, test.url, test.statusCode)

{

resp, err := http.Get(test.url)

if err != nil {

t.Fatalf("\t%s\tTest %d:\tShould be able to make the Get call : %v", failed, i, err)

}

t.Logf("\t%s\tTest %d:\tShould be able to make the Get call.", succeed, i)

defer resp.Body.Close()

if resp.StatusCode == test.statusCode {

t.Logf("\t%s\tTest %d:\tShould receive a %d status code.", succeed, i, test.statusCode)

} else {

t.Errorf("\t%s\tTest %d:\tShould receive a %d status code : %v", failed, i, test.statusCode, resp.StatusCode)

}

}

}

t.Run(test.name, tf)

}

}

}

// TestParallelize validates the http Get function can download content and

// handles different status conditions properly but runs the tests in parallel.

func TestParallelize(t \*testing.T) {

type tableTest struct {

name string

url string

statusCode int

}

tt := []tableTest{

{"statusok", "https://www.ardanlabs.com/blog/index.xml", http.StatusOK},

{"statusnotfound", "http://rss.cnn.com/rss/cnn\_topstorie.rss", http.StatusNotFound},

}

t.Log("Given the need to test downloading different content.")

{

for testID, test := range tt {

tf := func(testID int, test tableTest) func(t \*testing.T) {

return func(t \*testing.T) {

t.Parallel()

t.Logf("\tTest %d:\tWhen checking %q for status code %d", testID, test.url, test.statusCode)

{

resp, err := http.Get(test.url)

if err != nil {

t.Fatalf("\t%s\tTest %d:\tShould be able to make the Get call : %v", failed, testID, err)

}

t.Logf("\t%s\tTest %d:\tShould be able to make the Get call.", succeed, testID)

defer resp.Body.Close()

if resp.StatusCode == test.statusCode {

t.Logf("\t%s\tTest %d:\tShould receive a %d status code.", succeed, testID, test.statusCode)

} else {

t.Errorf("\t%s\tTest %d:\tShould receive a %d status code : %v", failed, testID, test.statusCode, resp.StatusCode)

}

}

}

}

t.Run(test.name, tf(testID, test))

}

}

}