ITMD:469/569 - Week 6 Lab 2

Travis Smith

Lab Summary

Now that you understand how Go web servers work behind the scenes, we can utilize the net/http package to easily create a Go web server and handler functions. We can use a router to handle the concurrency aspect and set up different routes for the web server, transforming it into a RESTful API that returns data to the user in JSON format, similar to the Covid API from last week.

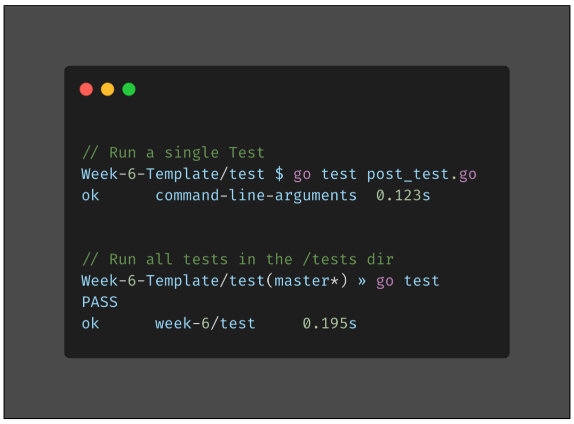
Please use the provided template code on Blackboard.

Testing

Testing may be challenging for some requests that require data submission to the web server. In the template code, three testing programs are included in the test directory to demonstrate how unit tests are created in Go. Take a look at them to understand how to write tests. The DeleteCourse test showcases a successful test, as the handler function has already been implemented. In your homework, you will create a test case for the GetCourseByCRN() handler.

Go tests can be executed either by using the "go test filename.go" command in the terminal or by using your IDE if it supports it.

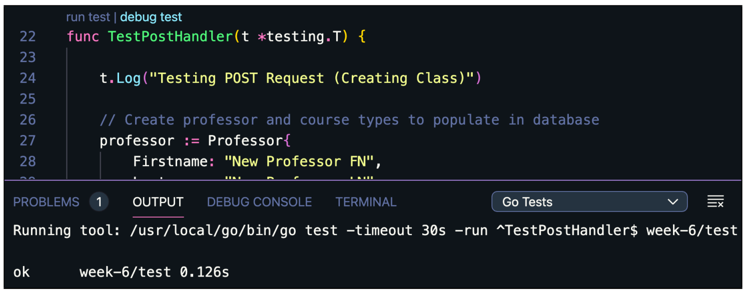
Running Go test from the terminal:



```
// Run a single Test
Week-6-Template/test $ go test post_test.go
ok      command-line-arguments  0.123s


// Run all tests in the /tests dir
Week-6-Template/test(master*) » go test
PASS
ok      week-6/test     0.195s
```

Testing using built-in integration by clicking "run test" above the func keyword:



```go
func TestPostHandler(t *testing.T) {

    t.Log("Testing POST Request (Creating Class)")

    // Create professor and course types to populate in database
    professor := Professor{
        Firstname: "New Professor FN",
```

```
Running tool: /usr/local/go/bin/go test -timeout 30s -run ^TestPostHandler$ week-6/test

ok      week-6/test 0.126s
```

Steps

- In the main.go file, the mux router and the handler functions have already been defined for you. You do not need to modify anything in main.go for this lab. You will be working in the handlers package, implementing these handler functions. These functions are called whenever a request is made to the corresponding handler URL.
- This REST API spins up a web server that handles requests to retrieve, update, create, and delete courses. The courses are already created for you in the coursesDatabase package as an array of Course types, accessible through coursesDatabase.Courses.
- Some of the handler functions have already been implemented for you to demonstrate how to decode the request into a Course type, loop through the courses "database," retrieve request parameters from the URL, and so on.
- Your task is to implement the UpdateCourse handler, CreateCourse handler, and the GetCourseByCRN handler. The DeleteCourse and GetAllCourses handlers have already been completed for you. Note that most of the logic in each handler will be very similar, so you can refer to the already implemented handlers for guidance on each step of the logic.
- You can test your code using the provided testing programs. They will perform the UpdateCourse, CreateCourse, and DeleteCourse requests and provide feedback on whether they were executed correctly.
- Handler Steps:
  1. Implement the GetCourseByCRN() handler logic (GET REQUEST)
     – Set the application/json header.
     – Retrieve the parameters from the request.
     – Loop through every course in coursesDatabase.Courses.
     – Check if the course's CRN matches the CRN in the parameters.
     – If there is a match, encode the course into the response writer to return the requested course to the requester.
  2. Implement the CreateCourse() handler logic (POST REQUEST)
     – Set the application/json header.

   - Create a new Course instance.
   - Decode the response body into the Course instance.
   - Append the new course to courseDatabase.Courses.
   - Encode the new course using the response writer to return the response back to the caller.

3. Implement the UpdateCourse() handler logic (PUT REQUEST)
   - Set the application/json header.
   - Retrieve the parameters from the request.
   - Loop through every course in courseDatabase.Courses using the range keyword and check if the course's CRN is equal to the CRN parameter in the request.
   - If there is a match, modify courseDatabase.Courses to remove the old course.
   - Create a new Course instance.
   - Decode the response body into the Course instance.
   - Set the new course's CRN equal to the CRN in the request parameters.
   - Append the new course to courseDatabase.Courses.
   - Encode the new course using the response writer to return the response back to the requester.
   - This is very similar to the DeleteCourse handler, but instead of removing the course altogether, we are updating the course's CRN.

Submission

Please compress your source code into a ZIP file and upload it to Blackboard. Additionally, attach a separate screenshot demonstrating your code and the go tests running successfully.