

Protocol Hörschinger Rene

Link to [Github](#)

Step 1

just following the instructions and hardcode the username/password for the database in main_test.go

Step 2

After starting postgres db in docker, all tests run smooth

```
→ go-mux git:(main) x go test -v
=== RUN   TestEmptyTable
--- PASS: TestEmptyTable (0.02s)
=== RUN   TestGetNonExistentProduct
--- PASS: TestGetNonExistentProduct (0.01s)
=== RUN   TestCreateProduct
--- PASS: TestCreateProduct (0.01s)
=== RUN   TestGetProduct
--- PASS: TestGetProduct (0.01s)
=== RUN   TestUpdateProduct
--- PASS: TestUpdateProduct (0.01s)
=== RUN   TestDeleteProduct
--- PASS: TestDeleteProduct (0.01s)
=== RUN   TestMyNewFeature
--- PASS: TestMyNewFeature (0.00s)
PASS
ok      github.com/ReneH98/go-mux      0.268s
→ go-mux git:(main) x
```

In the app.Initialize I added the code:

```
err = a.DB.Ping()
if err != nil {
    panic(err)
}
```

That way, the connection is tested on the startup, which then resulted in the error: "panic: pq: SSL is not enabled on the server"

In order to fix this, I needed the environment variables which only seem to work if they get sourced inside a .sh file

```
$ source exports.sh
```

Step 3 developing a new feature

In order to start the webserver, I adjusted the run function in app.go

```
func (a *App) Run(addr string) {  
    log.Fatal(http.ListenAndServe(":8010", a.Router))  
}
```

Now I can build the project, run it and access the webserver over a browser:



The image contains three terminal screenshots. The first shows the command `go build .` being executed in a `go-mux` directory, with the output `init done`. The second shows the command `curl localhost:8010/` being executed, returning `"Hello World!"`. The third shows the command `curl localhost:8010/product -X POST -d '{"name":"test","price":3}'` being executed, returning `{"id":1,"name":"test","price":3}`.

```
→ go-mux git:(main) x go build .  
→ go-mux git:(main) x ./go-mux  
init done  
→ ~ git:(master) x curl localhost:8010/product -X POST -d '{"name":"test","price":3}'  
{"id":1,"name":"test","price":3}
```

As a feature, I want to support endpoints to get the highest and lowest priced items (or an error if there are no items in the db). Of course I could've done it with the SQL statement, but I wanted to play a little with GO, arrays, structs, etc.

Here I tested delete, products, post new products, highest and lowest endpoints:

```

renehorschinger@MBP-von-Rene: ~
~ (-zsh)  %1  ~ (-zsh)  %2  +

→ ~ git:(master) x curl localhost:8010/product/3 -X DELETE
{"result":"success"}%

→ ~ git:(master) x curl localhost:8010/products
[]%

→ ~ git:(master) x curl localhost:8010/price/highest
{"error":"No items in db"}%

→ ~ git:(master) x curl localhost:8010/price/lowest
{"error":"No items in db"}%

→ ~ git:(master) x curl localhost:8010/product -X POST -d '{"name":"product 1","price":2}'
{"id":4,"name":"product 1","price":2}%

→ ~ git:(master) x curl localhost:8010/product -X POST -d '{"name":"product 2","price":4}'
{"id":5,"name":"product 2","price":4}%

→ ~ git:(master) x curl localhost:8010/product -X POST -d '{"name":"product 3","price":3}'
{"id":6,"name":"product 3","price":3}%

→ ~ git:(master) x curl localhost:8010/product -X POST -d '{"name":"product 4","price":10}'
{"id":7,"name":"product 4","price":10}%

→ ~ git:(master) x curl localhost:8010/product -X POST -d '{"name":"product 5","price":7}'
{"id":8,"name":"product 5","price":7}%

→ ~ git:(master) x curl localhost:8010/products
[{"id":4,"name":"product 1","price":2},{ "id":5,"name":"product 2","price":4},{ "id":6,"name":"product 3","price":3},{ "id":7,"name":"product 4","price":10},{ "id":8,"name":"product 5","price":7}]%

→ ~ git:(master) x curl localhost:8010/price/highest
{"id":7,"name":"product 4","price":10}%

→ ~ git:(master) x curl localhost:8010/price/lowest
{"id":4,"name":"product 1","price":2}%

→ ~ git:(master) x

```

I also added some tests for the highest and lowest endpoint:

```

renehorschinger@MBP-von-Rene: ~/Hagenberg/MC_Master/2_Semester/ContinuousDel...
~ (-zsh)  %1  ..e/ue02/go-mux (-zsh)  %2  +

→ go-mux git:(main) x go test -v
=== RUN   TestEmptyTable
--- PASS: TestEmptyTable (0.01s)
=== RUN   TestGetNonExistentProduct
--- PASS: TestGetNonExistentProduct (0.00s)
=== RUN   TestCreateProduct
--- PASS: TestCreateProduct (0.01s)
=== RUN   TestGetProduct
--- PASS: TestGetProduct (0.01s)
=== RUN   TestUpdateProduct
--- PASS: TestUpdateProduct (0.01s)
=== RUN   TestDeleteProduct
--- PASS: TestDeleteProduct (0.01s)
=== RUN   TestLowestPrice
--- PASS: TestLowestPrice (0.01s)
=== RUN   TestHighestPrice
--- PASS: TestHighestPrice (0.01s)
PASS
ok      github.com/ReneH98/go-mux    0.276s
→ go-mux git:(main) x

```

