Distributed Systems

Project 1 Discussion

Paul Krzyzanowski
TA: Yuanzhen Gu
Rutgers University
Fall 2015
The Assignment

• Write a simple REST based Web Service:
  – Add/change/query student information

• You may choose either XML or JSON as for data representation

• Support **POST GET PUT DELETE** to perform CRUD operations

• Create the project in the Go Programming Language (golang.org)

• Write a client to test your REST based Web Service, Usage provided

• Store and retrieve data from a backend database

• You may work on this assignment either individually or in a group of up to four members.

• Start early!
Assignment Details – student info

• Student information includes NetID, Name, Major, the Year enrolled, the Grade for Distributed Systems course, and the Rating (course grade)

• Syntax

Student struct{
    NetID string `json: "id`"
    Name string `json: "name"`"
    Major string `json: "major"`"
    Year int `json: "year"`"
    Grade int `json: "grade"`"
    Rating string `json: "rating"`"
}
POST Operation

• Add new student info by running your client. Client usage is:

```
$ go run test.go  -url= http://localhost:1234/Student   \ 
   -method=Create  -data='{"NetID":"147001234", "Name":"Mike","Major":"Computer Science","Year":2015,"Grade":90,"Rating":"D"}'
```

GET Operation

• List a specific student’s info. Client usage is:

```
$ go run test.go  -url="http://localhost:1234/Student/getstudent?name=Mary"  \ 
   -method=list
```

DELETE Operation

Students enrolled before a specific year will be removed. For example, you can delete students who enrolled before year 2010 with:

```
$ go run test.go  -url="http://localhost:1234/Student  "  \ 
   -method=remove  -year=2010
```
Assignment Details – CRUD operations Cont’

UPDATE Operation

The Rating (course grade) field is initially set as "D", and been updated after an average grade is set:

- If Grade > Average + 10 then set student's rating to "A"
- Average-10 < Grade < Average+10 then set the student’s rating to "B"
- Average-20 < Grade < Average-10 then set the student’s rating to "C"
- Others just no need update the Rating

The program should get all the grade info, do the calculations, then update the rating field:

$ go run test.go -url="http://localhost:1234/Student" -method=update

LIST Operation

- Finally, show all the students’ info with:

$ go run test.go -url="http://localhost:1234/Student/listall" -method=list
Language & Prerequisite

• You must use Go for this assignment
  – Install Go on your laptop and make sure your Go environment correctly
  – Or log onto an iLab machine
    • We showed how to set up your workstation and run your Go program on iLab machines in an earlier recitation class

• Test your Web Server with Hello World
  – Make sure your Go environment set up is correctly
  – Start the HTTP server on port 1234
  – Build your program
  – Run it within your Go Workspace
How to do this assignment: Step 0

• Make sure you can use Go to write a RESTful Web Service
  – Review recitation notes for Introduction to Go programming
  – Read http://thenewstack.io/make-a-restful-json-api-go/

• Install Go on your laptop or log onto an iLab machine
  – Set up your Go environment correctly

• Write a "Hello World" program and make sure your web service works well on the Go workstation
Step 1 Import packages

• RESTful web applications are all HTTP-based, so we can use the `net/http` package to implement it

• Need a `router` package
  – There are couple of third party routers
  – Example using mux router from the Gorilla Web Toolkit:

```go
Import(
  "net/http"

  "github.com/gorilla/mux"
)
```
Step 2 Create handler functions

- http requests are typically handled by functions of type `http.HandlerFunc`

- The handler functions contains two parameters:
  - an `http.ResponseWriter` which is used to write our response to the http request
  - an `*http.Request` which we use to parse the contents of the incoming request

```go
func myHandlerFunction(w http.ResponseWriter, r *http.Request) {
  // read from r and write to w
}
```
Step 3a Create routes

• Create a list of routes that link your handler functions to a URL and the HTTP request and feed these routes to a router

```go
type Route struct {
    Name    string
    Method  string
    HandlerFunc http.HandlerFunc
}

type Routes []Route
```
Step 3b Retrieve variables

- Retrieve the variable from the handler function by using `mux.Vars"variable"]`

```go
func (ls *LinkShortnerAPI) UrlShow(w http.ResponseWriter, r *http.Request) {
    // retrieve the variable from the request
    vars := mux.Vars(r)
}

"mux" is the name of the [Gorilla package](https://godoc.org/gorilla/mux) that includes the router.
```
Step 4 JSON Parsing

• In Go, the package to encode & decode JSON formats is called `encoding/json`

```go
Student struct{
    NetID string `json: "id"`
    Name string `json: "name"`
    Major string `json: "major"`
    Year int `json: "year"`
    Grade int `json: "grade"`
    Rating string `json: "rating"
}
```

• Start web server on localhost port, say 1234

```go
func main() {
    router := Newrouter()
    log.Fatal(http.ListenAndServe(":1234", router))
}
```
Step 5 Add Backend Database

- Use MongoDB as an example
  - MongoDB is a popular scalable document store NOSQL database
  - When writing code in Go, we use the mgo package to interface with MongoDB
  - MongoDB uses a binary form of JSON called BSON to store data in the documents. This means that data stored in a MongoDB document could be easily modeled like any JSON document.

- mgo uses the concept of "sessions" to connect to the database
  - A session could be considered a socket connection from a socket pool

```go
type MongoConnection struct {
    originalSession *mgo.Session
}
```
Step 6 Update code

• Update Student info syntax

Import "gopkg.in/mgo.v2/bson"

Student struct{

    NetID bson.ObjectId `json: "id" bson:"_id"`
    Name string `json: "name" bson:"name"`
    Major string `json: "major" bson:"major"`
    Year int `json: "year" bson:"year"`
    Grade int `json: "grade" bson:"grade"`
    Rating string `json: "rating" bson:"rating"`
}

• Update your code for Post, Get, Update, Delete functions accordingly
Documentation

• Document your work NEATLY

• For your submission, explain:
  – The files you’re submitting and what they do
  – How to compile & run (test with a non-technical person!)
  – List of tests that you ran to validate the software
The end