Go Track
Week 8 Practical

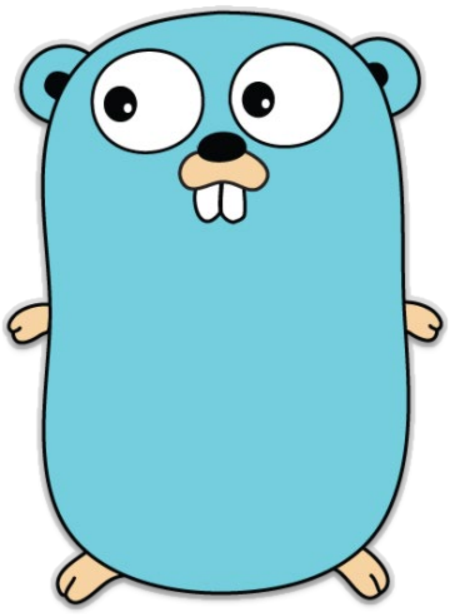


Table of Contents

Table of Contents 2
Lab 1. Parsing JSON 3
Lab 2. Consuming Web Services 9
Lab 3. Developing REST API 18
Lab 4. Connecting to a REST API 33
Lab 5. Accessing MySQL Databases 36
Lab 6. Getting Started with Docker 43
Lab 7. Docker Management 44
Lab 8. Using the Ubuntu Docker Image 49
Lab 9. Powering a Web Server using NginX 54
Lab 10. Using MySQL in Docker 61
Lab 11. Running Go REST API in Docker 67

NGEE ANN POLYTECHNIC

Lab 1. Parsing JSON

Description	*In this lab, you will learn how to parse JSON strings in Go.*
What You Will Learn	• How to decode a JSON string to a struct • How to decode a JSON string to an array • How to decode embedded JSON objects • How to map custom attribute names in JSON • How to map unstructured JSON data
Duration	50 Minutes

1. Create a folder named **JSON** within the **GoMS1** folder. Create a file named **main.go** and save it inside the **JSON** folder.

Decoding JSON To Struct

1. Populate the **main.go** file as follows:

```
package main

import (
    "fmt"
    "encoding/json"
)

type People struct {
    Firstname string              // first char must be capitalized
    Lastname string               // first char must be capitalized
}

func main() {
    var person People

    jsonString := `{"firstname":"Wei-Meng", "lastname":"Lee"}`
    err := json.Unmarshal([]byte(jsonString), &person)

    fmt.Println(person)           // {Wei-Meng Lee}
    fmt.Println(err)              // <nil>
}
```

In order for the attributes in the JSON string to map correctly to the fields in the struct, the struct and all fields in the struct need to be exported (accessible outside package). Therefore, you need to capitialize the first character of each field in the struct, as well as the struct itself.

The **Unmarshal()** function parses the JSON-encoded data and stores the result in the value pointed to by the second argument

2. Type the following commands in Terminal:

```
$ cd ~/GoMS1/JSON/
$ go run main.go
{Wei-Meng Lee}
<nil>
```

Decoding JSON to Arrays

1. Add the following statements in bold to the **main.go** file:

```
package main

import (
    "fmt"
    "encoding/json"
)

type People struct {
    Firstname string                    // must be capitalized
    Lastname string                     // must be capitalized
}

func main() {
    var persons []People

    jsonString :=
    `[
        {
            "firstname":"Wei-Meng",
            "lastname":"Lee"
        },
        {
            "firstname":"Mickey",
            "lastname":"Mouse"
        }
    ]`

    json.Unmarshal([]byte(jsonString), &persons)

    fmt.Println(persons)                 // [{Wei-Meng Lee} {Mickey Mouse}]
}
```

2. Type the following commands in Terminal:

```
$ go run main.go
```

Decoding Embedded Objects

1. Add the following statements in bold to the **main.go** file:

```
package main

import (
    "fmt"
    "encoding/json"
)

type People struct {
    Firstname string   // must be capitalized
    Lastname string    // must be capitalized
    Details struct {
        Height int
        Weight float32
    }
}

func main() {
    var persons []People
    jsonString :=
    `[
        {
            "firstname":"Wei-Meng",
            "lastname":"Lee",
            "details": {
                "height":175,
                "weight":70.0
            }
        },
        {
            "firstname":"Mickey",
            "lastname":"Mouse",
            "details": {
                "height":105,
                "weight":85.5
            }
        }
    ]`

    json.Unmarshal([]byte(jsonString), &persons)

    for _, v := range persons {
        fmt.Println(v.Firstname)
        fmt.Println(v.Lastname)
        fmt.Println(v.Details.Height)
        fmt.Println(v.Details.Weight)
    }
}
```

2. Type the following commands in Terminal:

```
$ go run main.go
Wei-Meng
Lee
175
70
Mickey
Mouse
105
85.5
```

Mapping Custom Attribute Names

1. Add the following statements in bold to the **main.go** file:

```
package main

import (
    "fmt"
    "encoding/json"
)

type People struct {
    Firstname string   // must be capitalized
    Lastname string    // must be capitalized
    Details struct {
        Height int
        Weight float32
    }
}

type Rates struct {
    Base string   `json:"base currency"`
    Symbol string `json:"destination currency"`
}

func main() {
    var person []People
    jsonString :=
    `[
        ...
    ]`

    json.Unmarshal([]byte(jsonString), &person)
    for _, v := range person {
        fmt.Println(v.Firstname)
        fmt.Println(v.Lastname)
        fmt.Println(v.Details.Height)
        fmt.Println(v.Details.Weight)
    }

    jsonString2 :=
    `{
        "base currency":"EUR",
        "destination currency":"USD"
     }`

    var rates Rates
    json.Unmarshal([]byte(jsonString2), &rates)
    fmt.Println(rates.Base)                            // EUR
    fmt.Println(rates.Symbol)                          // USD
}
```

You can map attributes from your JSON file to your struct using this syntax:

```
type Rates struct {
    Base string   `json:"base currency"`
    Symbol string `json:"destination currency"`
}
```

struct tag

2. Type the following commands in Terminal:

```
$ go run main.go
```

Mapping Unstructured Data

1. Add the following statements in bold to the **main.go** file:

```
package main

import (
    "fmt"
    "encoding/json"
)

type People struct {
    ...
}

type Rates struct {
    ...
}

func main() {
    ...

    jsonString2 :=
    `{
        "base currency":"EUR",
        "destination currency":"USD"
     }`

    var rates Rates
    json.Unmarshal([]byte(jsonString2), &rates)
    fmt.Println(rates.Base)
    fmt.Println(rates.Symbol)

    jsonString3 :=
    `{
        "success": true,
        "timestamp": 1588779306,
        "base": "EUR",
        "date": "2020-05-06",
        "rates": {
            "AUD": 1.683349,
            "CAD": 1.528643,
            "GBP": 0.874757,
            "SGD": 1.534513,
            "USD": 1.080054
        }
    }`

    var result map[string] interface{}
    /*
    Each string corresponds to a JSON property, and its mapped
    interface{} type corresponds to the value, which can be of
    any type.

    The type is asserted from this interface{} type as is needed in the code.
    */

    json.Unmarshal([]byte(jsonString3), &result)

    fmt.Println(result["success"])
    currRates := result["rates"]    // value of rates is actually an interface{},
                                    // which could be anything - a map, a
```

```
                                        // string, or an int.
    fmt.Println(currRates)              // map[AUD:1.683349 CAD:1.528643
                                        // GBP:0.874757 SGD:1.534513 USD:1.080054]

    SGD := currRates.(map[string] interface{})["SGD"] // you need to assert it to
                                                      // a map with expected
                                                      // key/value types
    fmt.Println(SGD)
}
```

2. Type the following commands in Terminal:

```
$ go run json.go
```

```
true
map[AUD:1.683349 CAD:1.528643 GBP:0.874757 SGD:1.534513 USD:1.080054]
1.534513
```

The statement:
`var result map[string] interface{}`

Means that **result** is a variable of type **map**, with *key* of type **string**, and *value* of type **interface** (which can be of any type)

The statement:

`rates.(map[string] interface{})`

asserts the **rates** variable into a **map** type with keys of type **string**

NGEE ANN POLYTECHNIC

Lab 2. Consuming Web Services

Description	*In this lab, you will learn how to consume web services and parse their returning results.*
What You Will Learn	• How to fetch data from web services • How to fetch data from multiple web services • How to decode JSON data • How to use a channel to pass data back from a goroutine
Duration	60 Minutes

Fetching Data from Web Services

1. Create a folder named **ConsumeWS** within the **GoMS1** folder. Create a file named **main.go** and save it inside the **ConsumeWS** folder. Populate the file with the following statements:

Fixer is a foreign exchange rate and currency conversion JSON API. We are going to fetch some data from it. You need to apply for your **Fixer** access key at http://fixer.io.

```
package main

import (
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
)

func main() {
    url := "http://data.fixer.io/api/latest?access_key=<access_key>"

    if resp, err := http.Get(url); err == nil {
        defer resp.Body.Close()
        if body, err := ioutil.ReadAll(resp.Body); err == nil {
            fmt.Println(string(body))
        } else {
            log.Fatal(err)
        }
    } else {
        log.Fatal(err)
    }
    fmt.Println("Done")
}
```

2. In Terminal, type the following commands:

```
$ cd ~/GoMS1/ConsumeWS
$ go run main.go
{"success":true,"timestamp":1588640707,"base":"EUR","date":"2020-05-
05","rates":{"AED":4.008572,"AFN":83.517906,"ALL":123.646112,"AMD":527.23302,"ANG
":1.962621,"AOA":610.377524,"ARS":72.831602,"AUD":1.692717,"AWG":1.964347,"AZN":1
.861229,"BAM":1.955927,"BBD":2.207309,"BDT":92.862687,"BGN":1.957847,"BHD":0.4120
88,"BIF":2068.423339,"BMD":1.091304,"BND":1.548732,"BOB":7.538915,"BRL":6.048772,
"BSD":1.093254,"BTC":0.000123,"BTN":82.679326,"BWP":13.331859,"BYN":2.683533,"BYR
":21389.557313,"BZD":2.203609,"CAD":1.534188,"CDF":1898.868473,"CHF":1.053064,"CL
F":0.033159,"CLP":914.949113,"CNY":7.707222,"COP":4352.840391,"CRC":620.523018,"C
UC":1.091304,"CUP":28.919555,"CVE":110.288639,"CZK":27.066301,"DJF":193.94638,"DK
K":7.460806,"DOP":59.174685,"DZD":140.312069,"EGP":17.187274,"ERN":16.369851,"ETB
":36.081835,"EUR":1,"FJD":2.457945,"FKP":0.875742,"GBP":0.875668,"GEL":3.502692,"
GGP":0.875742,"GHS":6.341659,"GIP":0.875742,"GMD":55.763339,"GNF":10445.738682,"G
TQ":8.438209,"GYD":229.239132,"HKD":8.463335,"HNL":27.126815,"HRK":7.581395,"HTG"
:115.484752,"HUF":354.122664,"IDR":16411.028718,"ILS":3.846825,"IMP":0.875742,"IN
R":82.949888,"IQD":1305.084494,"IRR":45949.353033,"ISK":159.504567,"JEP":0.875742
,"JMD":156.228487,"JOD":0.773701,"JPY":116.28393,"KES":116.333444,"KGS":86.090241
,"KHR":4488.36211,"KMF":493.951454,"KPW":982.232534,"KRW":1333.508328,"KWD":0.337
376,"KYD":0.911045,"KZT":467.463111,"LAK":9806.085136,"LBP":1653.181756,"LKR":207
.163144,"LRD":216.487353,"LSL":20.418525,"LTL":3.222337,"LVL":0.660119,"LYD":1.54
1976,"MAD":10.791213,"MDL":19.595769,"MGA":4149.835155,"MKD":61.618046,"MMK":1530
.49264,"MNT":3044.314515,"MOP":8.730631,"MRO":389.595541,"MUR":43.652004,"MVR":16
.860494,"MWK":805.082746,"MXN":26.306095,"MYR":4.708985,"MZN":74.056075,"NAD":20.
418531,"NGN":426.35057,"NIO":36.885907,"NOK":11.250492,"NPR":132.287351,"NZD":1.7
99795,"OMR":0.420131,"PAB":1.093434,"PEN":3.690082,"PGK":3.760606,"PHP":55.198696
,"PKR":175.022748,"PLN":4.548064,"PYG":7118.951803,"QAR":3.973164,"RON":4.827896,
"RSD":117.577022,"RUB":81.532444,"RWF":1027.770848,"SAR":4.099129,"SBD":9.022892,
"SCR":19.137102,"SDG":60.376395,"SEK":10.726334,"SGD":1.544238,"SHP":0.875742,"SL
L":10563.822075,"SOS":632.956083,"SRD":8.138975,"STD":24064.764671,"SVC":9.566473
,"SYP":559.93762,"SZL":20.43531,"THB":35.347974,"TJS":11.201801,"TMT":3.819564,"T
ND":3.14568,"TOP":2.528336,"TRY":7.692819,"TTD":7.386465,"TWD":32.50554,"TZS":252
4.735932,"UAH":29.481157,"UGX":4154.890621,"USD":1.091304,"UYU":46.222909,"UZS":1
1078.577181,"VEF":10.899396,"VND":25635.494661,"VUV":132.381199,"WST":3.019545,"X
AF":655.989426,"XAG":0.073647,"XAU":0.000642,"XCD":2.949303,"XDR":0.798439,"XOF":
655.989425,"XPF":119.265891,"YER":273.150719,"ZAR":20.188043,"ZMK":9823.065247,"Z
MW":20.033091,"ZWL":351.399871}}
```

The result printed is a JSON string

Decoding the JSON Response

1. Add the following statements in bold to the **main.go** file:

```
package main

import (
    "encoding/json"
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
)

type Result struct {
    Success   bool
    Timestamp int
    Base      string
    Date      string
    Rates     map[string]float64
}

type Error struct {
    Success bool
    Error   struct {
        Code  int
        Type  string
        Info  string
    }
}

func main() {
    url := "http://data.fixer.io/api/latest?access_key=<access_key>"

    if resp, err := http.Get(url); err == nil {
        defer resp.Body.Close()
        if body, err := ioutil.ReadAll(resp.Body); err == nil {
            var result Result
            json.Unmarshal(body, &result)
            if result.Success {
                for i, v := range result.Rates {
                    fmt.Println(i, v)
                }
            } else {
                var err Error
                json.Unmarshal(body, &err)
                fmt.Println(err.Error.Info)
            }
        } else {
            log.Fatal(err)
        }
    } else {
        log.Fatal(err)
    }
    fmt.Println("Done")
}
```

2. In Terminal, type the following commands:

```
$ cd ~/GoMS1/ConsumeWS
$ go run main.go
DOP 68.689041
JOD 0.833654
LVL 0.711216
...
CLF 0.032964
LKR 216.774437
RON 4.877364
Done
```

Note that the currencies are not sorted. Everytime, you may observe that the currencies appear in different order whenever you run the program again.

Sorting the Currencies

1. Add the following statements in bold to the **main.go** file:

```
func main() {
    url := "http://data.fixer.io/api/latest?access_key=<access_key>"

    if resp, err := http.Get(url); err == nil {
        defer resp.Body.Close()
        if body, err := ioutil.ReadAll(resp.Body); err == nil {
            var result Result
            json.Unmarshal(body, &result)
            if result.Success {
                // create a slice to store all keys
                keys := make([]string, 0,
                    len(result.Rates))

                // get all the keys---
                for k := range result.Rates {
                    keys = append(keys, k)
                }

                // sort the keys
                sort.Strings(keys)

                // print the keys and values in
                // alphabetical order
                for _, k := range keys {
                    fmt.Println(k, result.Rates[k])
                }

                /*
                    for i, v := range result.Rates {
                        fmt.Println(i, v)
                    }
                */
            } else {
                var err Error
                json.Unmarshal(body, &err)
                fmt.Println(err.Error.Info)
            }
        } else {
            log.Fatal(err)
        }
    } else {
        log.Fatal(err)
    }
    fmt.Println("Done")
}
```

2. In Terminal, type the following commands:

```
$ cd ~/GoMS1/ConsumeWS
$ go run main.go
AED 4.318876
AFN 90.418651
ALL 124.132913
...
ZMK 10583.430716
ZMW 23.927428
ZWL 378.600803
Done
```

Now the currencies are sorted!

Refactoring the Code for JSON Decoding

In this section, you will refactor your code for JSON decoding so that you can consume different web services and decode their responses.

1. Rewrite **main.go** as follows:

```
package main

import (
    "encoding/json"
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
)

var apis map[int]string

func fetchData(API int) {
    url := apis[API]
    if resp, err := http.Get(url); err == nil {
        defer resp.Body.Close()
        if body, err := ioutil.ReadAll(resp.Body); err == nil {

            var result map[string]interface{}

            json.Unmarshal(body, &result)

            switch API {
            case 1:
                if result["success"] == true {
                    fmt.Println(result["rates"].(map[string]interface{})["USD"])
                } else {
                    fmt.Println(result["error"].(map[string]interface{})["info"])
                }
            case 2:
                if result["main"] != nil {
                    fmt.Println(result["main"].(map[string]interface{})["temp"])
                } else {
                    fmt.Println(result["message"])
                }
            }
        } else {
            log.Fatal(err)
        }
    } else {
        log.Fatal(err)
    }
}

func main() {
    apis = make(map[int]string)

    apis[1] = "http://data.fixer.io/api/latest?access_key=<access_key>"
    apis[2] =
      "http://api.openweathermap.org/data/2.5/weather?q=SINGAPORE&appid=<app_id>"

    fetchData(1)
    fetchData(2)
}
```

You need to apply for your OpenWeatherMap App ID at http://api.openweathermap.org
Go to your username>My API Keys to obtain your general API key for open weather.

2. In Terminal, type the following commands:

```
$ cd ~/GoMS1/ConsumeWS
$ go run main.go
1.175206
303.15
```

Fetching From Several Web Services Concurrently

1. Modify the **main.go** file as follows:

```
func main() {
    apis = make(map[int]string)

    apis[1] = "http://data.fixer.io/api/latest?access_key=<access_key>"
    apis[2] =
      "http://api.openweathermap.org/data/2.5/weather?q=SINGAPORE&appid=<app_id>"

    go fetchData(1)
    go fetchData(2)

    fmt.Scanln()
}
```

2. In Terminal, type the following commands:

```
$ go run main.go
303.15
1.175206
```

Be sure to call the **Scanln()** function before the end of the **main()** function; otherwise, the application will exit before the goroutines have the chance to finish execution

Alternatively, **sync.Waitgroup** together with its **Wait()** function can be used.

Returning Goroutines' Results to the main() Function

Sometimes you need to return data from a goroutine back to the calling function. A good way to do that would be to use a channel. The following code snippet shows how to send data into a channel from a goroutine, and then retrieve the data from the main calling function:

1. Modify the **main.go** file as follows:

```
package main

import (
    "encoding/json"
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
)

var (
       apis map[int]string
       c chan map[int]interface{} // channel to store map[int]interface{}
)

func fetchData(API int) {
    url := apis[API]
    if resp, err := http.Get(url); err == nil {
        defer resp.Body.Close()
        if body, err := ioutil.ReadAll(resp.Body); err == nil {

            var result map[string]interface{}

            json.Unmarshal(body, &result)

            var re = make(map[int]interface{})

            switch API {
            case 1:
                if result["success"] == true {
                    re[API] = result["rates"].(map[string]interface{})["USD"]
                } else {
                    re[API] = result["error"].(map[string]interface{})["info"]
                }
                // store the result into the channel
                c <- re
                fmt.Println("Result for API 1 stored")

            case 2:
                if result["main"] != nil {
                    re[API] = result["main"].(map[string]interface{})["temp"]
                } else {
                    re[API] = result["message"]
                }
                // store the result into the channel
                c <- re
                fmt.Println("Result for API 2 stored")
            }
        } else {
            log.Fatal(err)
        }
    } else {
        log.Fatal(err)
    }
}

func main() {
    c = make(chan map[int]interface{})

    apis = make(map[int]string)

    apis[1] = "http://data.fixer.io/api/latest?access_key=<access_key>"
    apis[2] =
      "http://api.openweathermap.org/data/2.5/weather?q=SINGAPORE&appid=<app_id>"

    go fetchData(1)
    go fetchData(2)

    // we expect two results in the channel
    for i := 0; i < 2; i++ {
        fmt.Println(<-c)
    }
    fmt.Println("Done!")

    fmt.Scanln()
}
```

2. In Terminal, type the following commands:

```
$ go run network.go
Result for API 2 stored
map[2:303.15]
Result for API 1 stored
map[1:1.175206]
Done!
```

Exercises

1. Using Scanln() to ensure that main will only exit after the goroutines have finished execution is not the best way of doing it.

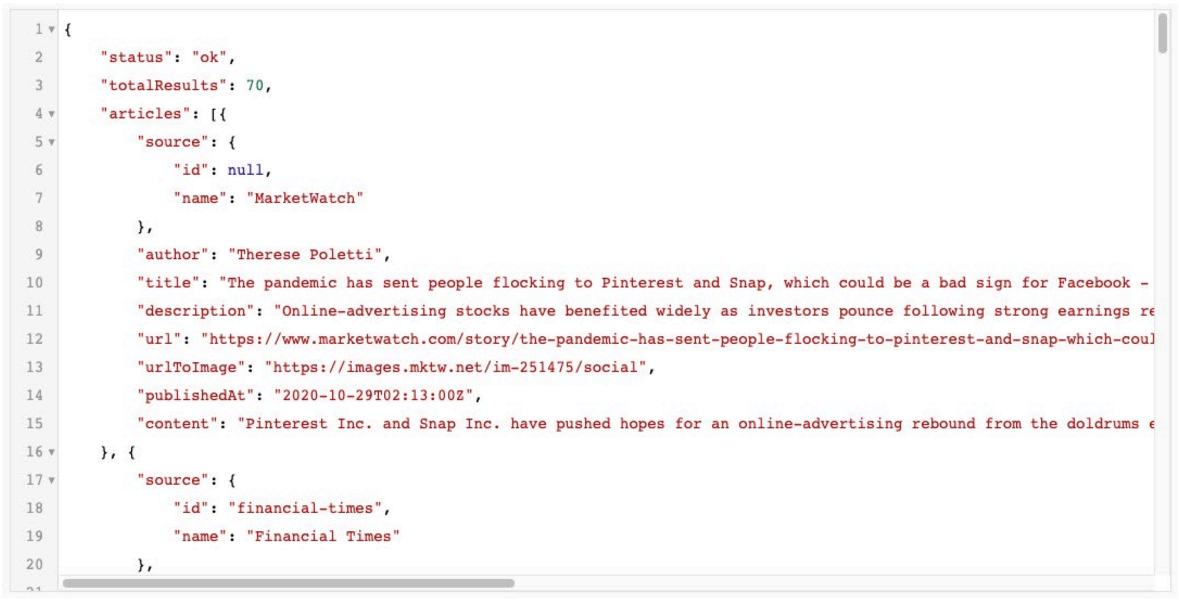
   You have learnt how to make sure of WaitGroup when studying concurrency in Go Advanced. Now improve on the previous example to incorporate the use of WaitGroup.

2. Given the following URL:

https://newsapi.org/v2/top-headlines?country=us&category=business&apiKey=<*api_key*>

You can apply for a free News API key from https://newsapi.org.

The response from this web service has the following format:



```
{
    "status": "ok",
    "totalResults": 70,
    "articles": [{
        "source": {
            "id": null,
            "name": "MarketWatch"
        },
        "author": "Therese Poletti",
        "title": "The pandemic has sent people flocking to Pinterest and Snap, which could be a bad sign for Facebook -
        "description": "Online-advertising stocks have benefited widely as investors pounce following strong earnings re
        "url": "https://www.marketwatch.com/story/the-pandemic-has-sent-people-flocking-to-pinterest-and-snap-which-coul
        "urlToImage": "https://images.mktw.net/im-251475/social",
        "publishedAt": "2020-10-29T02:13:00Z",
        "content": "Pinterest Inc. and Snap Inc. have pushed hopes for an online-advertising rebound from the doldrums e
    }, {
        "source": {
            "id": "financial-times",
            "name": "Financial Times"
        },
```

Write a Go program to display the list of news sources. Include the following fields:

- Name of the news source
- Title of the news
- Description of the news

NGEE ANN POLYTECHNIC

Lab 3. Developing REST API

Description	*In this lab, you will learn how to develop a REST API using Go.*
What You Will Learn	• How to create a REST API using the gorilla/mux package • How to register routes for your REST API • How to extract key/value pairs from query string • How to specify request methods • How to store information on the REST API • How to add course to the REST API • How to modify course information on the REST API • How to delete course on the REST API • How to retrieve course information on the REST API • How to secure the REST API using an access token
Duration	90 Minutes

Installing the gorilla/mux Package

1. Type the following command in Terminal:

```
$ go get -u github.com/gorilla/mux
```

The **github.com/gorilla/mux** package implements a request router and dispatcher.

Creating the Base REST API

1. Create a folder named **REST** within the **GoMS1** folder.

2. Create a file named **main.go** and save it inside the **REST** folder.

3. Populate the **main.go** file with the following statements:

```
package main

import (
    "fmt"
    "log"
    "net/http"

    "github.com/gorilla/mux"
)

func home(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Welcome to the REST API!")
}

func main() {
    router := mux.NewRouter()
    router.HandleFunc("/api/v1/", home)

    fmt.Println("Listening at port 5000")
    log.Fatal(http.ListenAndServe(":5000", router))
}
```

4. Type the following command in Terminal:

```
$ cd ~/GoMS1/REST
$ go run main.go
Listening at port 5000
```

5. Type the following command in *another* Terminal window:

```
$ curl http://localhost:5000/api/v1/
Welcome to the REST API!
```

Not familiar with curl? Use the online curl command line builder at: https://curlbuilder.com

You can also use the **Postman** app (https://www.getpostman.com) to connect to REST services graphically

Registering Additional Routes

1. Add the following statements in bold to the **main.go** file:

```
package main

import (
    "fmt"
    "log"
    "net/http"

    "github.com/gorilla/mux"
)

func home(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Welcome to the REST API!")
}

func allcourses(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "List of all courses")
}

func course(w http.ResponseWriter, r *http.Request) {
    params := mux.Vars(r)
    fmt.Fprintf(w, "Detail for course " + params["courseid"])
}

func main() {
    router := mux.NewRouter()

    router.HandleFunc("/api/v1/", home)
    router.HandleFunc("/api/v1/courses", allcourses)
    router.HandleFunc("/api/v1/courses/{courseid}", course)

    fmt.Println("Listening at port 5000")
    log.Fatal(http.ListenAndServe(":5000", router))
}
```

2. Restart the **main.go** (Ctrl-C to stop):

```
$ go run main.go
```

3. Type the following command into the other Terminal window:

```
$ curl http://localhost:5000/api/v1/courses
List of all courses

$ curl http://localhost:5000/api/v1/courses/IOS101
Detail for course IOS101
```

To pass a variable into a path, use the ***{variable}*** syntax:
`"/api/v1/courses/{courseid}"`

Specifying Query String

1. Add the following statements in bold to the **main.go** file:

```
package main

import (
    "fmt"
    "log"
    "net/http"

    "github.com/gorilla/mux"
)

func home(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Welcome to the REST API!")
}

func allcourses(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "List of all courses")

    // returns the key/value pairs in the query string as a map object
    kv := r.URL.Query()

    for k, v := range kv {
        fmt.Println(k, v)     // print out the key/value pair
    }
}

func course(w http.ResponseWriter, r *http.Request) {
    params := mux.Vars(r)
    fmt.Fprintf(w, "Detail for course "+params["courseid"])
}

func main() {
    router := mux.NewRouter()
    router.HandleFunc("/api/v1/", home)

    router.HandleFunc("/api/v1/courses", allcourses)
    router.HandleFunc("/api/v1/courses/{courseid}", course)

    fmt.Println("Listening at port 5000")
    log.Fatal(http.ListenAndServe(":5000", router))
}
```

2. Restart the **main.go** (Ctrl-C to stop):

```
$ go run main.go
```

3. Type the following command into the other Terminal window:

```
$ curl "http://localhost:5000/api/v1/courses?country=US&state=CA"
```

For Mac and Windows users, remember to enclose the URL using a pair of double quotes

You will now see the following printed on the server side:

```
$ go run main.go
Listening at port 5000
country [US]
state [CA]
```

Specifying Request Methods

1. Add the following statements in bold to the **main.go** file:

```
package main

import (
    "fmt"
    "log"
    "net/http"

    "github.com/gorilla/mux"
)

func home(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Welcome to the REST API!")
}

func allcourses(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "List of all courses")
    kv := r.URL.Query()

    for k, v := range kv {
        fmt.Println(k, v)
    }
}

func course(w http.ResponseWriter, r *http.Request) {
    params := mux.Vars(r)
    fmt.Fprintf(w, "Detail for course "+params["courseid"])
    fmt.Fprintf(w, "\n")
    fmt.Fprintf(w, r.Method)
}

func main() {
    router := mux.NewRouter()
    router.HandleFunc("/api/v1/", home)

    router.HandleFunc("/api/v1/courses", allcourses)
    router.HandleFunc("/api/v1/courses/{courseid}", course).Methods(
        "GET", "PUT", "POST", "DELETE")

    fmt.Println("Listening at port 5000")
    log.Fatal(http.ListenAndServe(":5000", router))
}
```

You can register each HTTP method with individual functions, or use a single function

2. Restart the **main.go** (Ctrl-C to stop):

```
$ go run main.go
```

3. Type the following command into the other Terminal window:

```
$ curl http://localhost:5000/api/v1/courses/IOT201
Detail for course IOT201
GET

$ curl -X POST http://localhost:5000/api/v1/courses/IOT201
Detail for course IOT201
POST

$ curl -X PUT http://localhost:5000/api/v1/courses/IOT201
Detail for course IOT201
PUT

$ curl -X DELETE http://localhost:5000/api/v1/courses/IOT201
Detail for course IOT201
DELETE
```

MINI-REFERENCE: CURL OPTIONS

Option	Purpose
`-X`	specify HTTP request method e.g. POST
`-H`	specify request headers e.g. "Content-type: application/json"
`-d`	specify request data e.g. '{"message":"Hello Data"}'
`--data-binary`	specify binary request data e.g. @file.bin
`-i`	shows the response headers
`-u`	specify username and password e.g. "admin:secret"
`-v`	enables verbose mode which outputs info such as request and response headers and errors

Storing Course Information

1. Add the following statements in bold to the **main.go** file:

```
package main

import (
    "encoding/json"
    "fmt"
    "log"
    "net/http"

    "github.com/gorilla/mux"
)

type courseInfo struct {
    Title string `json:"Title"`
}

// used for storing courses on the REST API
var courses map[string]courseInfo

func home(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Welcome to the REST API!")
}

func allcourses(w http.ResponseWriter, r *http.Request) {
    // fmt.Fprintf(w, "List of all courses")
    kv := r.URL.Query()

    for k, v := range kv {
        fmt.Println(k, v)
    }
    // returns all the courses in JSON
    json.NewEncoder(w).Encode(courses)
}

func course(w http.ResponseWriter, r *http.Request) {
    params := mux.Vars(r)
    fmt.Fprintf(w, "Detail for course "+params["courseid"])
    fmt.Fprintf(w, "\n")
    fmt.Fprintf(w, r.Method)
}

func main() {
    router := mux.NewRouter()
    router.HandleFunc("/api/v1/", home)

    router.HandleFunc("/api/v1/courses", allcourses)
    router.HandleFunc("/api/v1/courses/{courseid}", course).Methods(
        "GET", "PUT", "POST", "DELETE")

    fmt.Println("Listening at port 5000")
    log.Fatal(http.ListenAndServe(":5000", router))
}
```

Creating New Courses

1. Add the following statements in bold to the **main.go** file:

```
package main

import (
    "encoding/json"
    "fmt"
    "log"
    "net/http"

    "github.com/gorilla/mux"
)

type courseInfo struct {
    Title string `json:"Title"`
}

var courses map[string]courseInfo

func home(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Welcome to the REST API!")
}

func allcourses(w http.ResponseWriter, r *http.Request) {
    // fmt.Fprintf(w, "List of all courses")
    kv := r.URL.Query()

    for k, v := range kv {
        fmt.Println(k, v)
    }
    // returns all the courses in JSON
    json.NewEncoder(w).Encode(courses)
}

func course(w http.ResponseWriter, r *http.Request) {
    params := mux.Vars(r)

    /*
    fmt.Fprintf(w, "Detail for course "+params["courseid"])
    fmt.Fprintf(w, "\n")
    fmt.Fprintf(w, r.Method)
    */

    if r.Header.Get("Content-type")=="application/json" {

        // POST is for creating new course
        if r.Method == "POST" {

            // read the string sent to the service
            var newCourse courseInfo
            reqBody, err := ioutil.ReadAll(r.Body)

            if err == nil {
                // convert JSON to object
                json.Unmarshal(reqBody, &newCourse)

                if newCourse.Title == "" {
                    w.WriteHeader(
                        http.StatusUnprocessableEntity)
                    w.Write([]byte(
                       "422 - Please supply course " +
                       "information " + "in JSON format"))
                    return
                }
                // check if course exists; add only if
                // course does not exist
                if _, ok := courses[params["courseid"]];
                    !ok {
                    courses[params["courseid"]] = newCourse
                    w.WriteHeader(http.StatusCreated)
                    w.Write([]byte("201 - Course added: " +
                        params["courseid"]))
                } else {
                    w.WriteHeader(http.StatusConflict)
                    w.Write([]byte(
                        "409 - Duplicate course ID"))
                }
            } else {
                w.WriteHeader(
                    http.StatusUnprocessableEntity)
                w.Write([]byte("422 - Please supply course information " +
                               "in JSON format"))
            }
        }
    }
}

func main() {
    // instantiate courses
    courses = make(map[string]courseInfo)

    router := mux.NewRouter()
    router.HandleFunc("/api/v1/", home)

    router.HandleFunc("/api/v1/courses", allcourses)
    router.HandleFunc("/api/v1/courses/{courseid}", course).Methods(
        "GET", "PUT", "POST", "DELETE")

    fmt.Println("Listening at port 5000")
    log.Fatal(http.ListenAndServe(":5000", router))
}
```

The list of HTTP response code can be found here:

https://golang.org/pkg/net/http/#ResponseWriter

Updating Courses

1. Add the following statements in bold to the **main.go** file:

```
package main

import (
    "encoding/json"
    "fmt"
    "io/ioutil"
    "log"
    "net/http"

    "github.com/gorilla/mux"
)

type courseInfo struct {
    Title string `json:"Title"`
}

var courses map[string]courseInfo

func home(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Welcome to the REST API!")
}

func allcourses(w http.ResponseWriter, r *http.Request) {
    // fmt.Fprintf(w, "List of all courses")
    kv := r.URL.Query()

    for k, v := range kv {
        fmt.Println(k, v)
    }
    // returns all the courses in JSON
    json.NewEncoder(w).Encode(courses)
}

func course(w http.ResponseWriter, r *http.Request) {
    params := mux.Vars(r)
    fmt.Fprintf(w, "Detail for course "+params["courseid"])
    fmt.Fprintf(w, "\n")
    fmt.Fprintf(w, r.Method)

    if r.Header.Get("Content-type") == "application/json" {

        //  POST is for creating new course
        if r.Method == "POST" {
            ...
        }

        //---PUT is for creating or updating
        // existing course---
        if r.Method == "PUT" {
            var newCourse courseInfo
            reqBody, err := ioutil.ReadAll(r.Body)

            if err == nil {
                json.Unmarshal(reqBody, &newCourse)

                if newCourse.Title == "" {
                    w.WriteHeader(
                        http.StatusUnprocessableEntity)
                    w.Write([]byte(
                        "422 - Please supply course " +
                        " information " +
                        "in JSON format"))
                    return
                }

                // check if course exists; add only if
                // course does not exist
                if _, ok := courses[params["courseid"]]; !ok {
                    courses[params["courseid"]] =
                        newCourse
                    w.WriteHeader(http.StatusCreated)
                    w.Write([]byte("201 - Course added: " +
                        params["courseid"]))
                } else {
                    // update course
                    courses[params["courseid"]] = newCourse
                    w.WriteHeader(http.StatusAccepted)
                    w.Write([]byte("202 - Course updated: " +
                                    params["courseid"]))
                }
            } else {
                w.WriteHeader(
                    http.StatusUnprocessableEntity)
                w.Write([]byte("422 - Please supply " +
                    "course information " +
                    "in JSON format"))
            }
        }
    }
}

func main() {
    // instantiate courses
    courses = make(map[string]courseInfo)

    router := mux.NewRouter()
    router.HandleFunc("/api/v1/", home)

    router.HandleFunc("/api/v1/courses", allcourses)
    router.HandleFunc("/api/v1/courses/{courseid}", course).Methods(
        "GET", "PUT", "POST", "DELETE")

    fmt.Println("Listening at port 5000")
    log.Fatal(http.ListenAndServe(":5000", router))
}
```

Retrieving Courses

1. Add the following statements in bold to the **main.go** file:

```
func course(w http.ResponseWriter, r *http.Request) {
    params := mux.Vars(r)
    fmt.Fprintf(w, "Detail for course "+params["courseid"])
    fmt.Fprintf(w, "\n")
    fmt.Fprintf(w, r.Method)

    if r.Method == "GET" {
        if _, ok := courses[params["courseid"]]; ok {
            json.NewEncoder(w).Encode(
                courses[params["courseid"]])
        } else {
            w.WriteHeader(http.StatusNotFound)
            w.Write([]byte("404 - No course found"))
        }
    }

    if r.Header.Get("Content-type") == "application/json" {
        ...
    }
}
```

Deleting Courses

1. Add the following statements in bold to the **main.go** file:

```
func course(w http.ResponseWriter, r *http.Request) {
    params := mux.Vars(r)
    fmt.Fprintf(w, "Detail for course "+params["courseid"])
    fmt.Fprintf(w, "\n")
    fmt.Fprintf(w, r.Method)

    if r.Method == "GET" {
        if _, ok := courses[params["courseid"]]; ok {
            json.NewEncoder(w).Encode(
                courses[params["courseid"]])
        } else {
            w.WriteHeader(http.StatusNotFound)
            w.Write([]byte("404 - No course found"))
        }
    }

    if r.Method == "DELETE" {
        if _, ok := courses[params["courseid"]]; ok {
            delete(courses, params["courseid"])
            w.WriteHeader(http.StatusAccepted)
           w.Write([]byte("202 - Course deleted: " +
                              params["courseid"]))
        } else {
            w.WriteHeader(http.StatusNotFound)
            w.Write([]byte("404 - No course found"))
        }
    }

    if r.Header.Get("Content-type") == "application/json" {
        ...
    }
}
```

Testing the API

1. Restart the **main.go** (Ctrl-C to stop):

```
$ go run main.go
```

2. Type the following command into the other Terminal window:

```
$ curl http://localhost:5000/api/v1/courses
{}
```

There are no courses at this moment

```
$ curl -H "Content-Type:application/json" -X POST
http://localhost:5000/api/v1/courses/IOS101 -d "{\"title\":\"iOS Programming\"}"
201 - Course added: IOS101
```

Course **IOS101** has been added

```
$ curl -H "Content-Type:application/json" -X POST
http://localhost:5000/api/v1/courses/IOS101 -d "{\"title\":\"iOS Programming\"}"
409 - Duplicate course ID
```

Course **IOS101** already exists, so error **409** is returned

```
$ curl http://localhost:5000/api/v1/courses
{"IOS101":{"Title":"iOS Programming"}}
```

Only one course is stored at this moment

```
$ curl -H "Content-Type: application/json" -X PUT
http://localhost:5000/api/v1/courses/IOS102 -d "{\"title\":\"Swift
Programming\"}"
201 - Course added: IOS102
```

Another course **IOS102** is added

```
$ curl -H "Content-Type: application/json" -X PUT
http://localhost:5000/api/v1/courses/IOS102 -d "{\"title\":\"SwiftUI
Programming\"}"
```

Course title is modified; response **204** is returned (no content)

```
$ curl http://localhost:5000/api/v1/courses
{"IOS101":{"Title":"iOS Programming"},"IOS102":{"Title":"SwiftUI Programming"}}
```

Just to make sure the title for **IOS102** is updated

```
$ curl http://localhost:5000/api/v1/courses/IOS102
{"Title":"SwiftUI Programming"}
```

Retrieve course **IOS102**

```
$ curl -X DELETE http://localhost:5000/api/v1/courses/IOS102
```

Delete course **IOS102**

```
$ curl http://localhost:5000/api/v1/courses/IOS102
404 - No course found
```

Course **IOS102** is no longer available, so error **404**

```
$ curl http://localhost:5000/api/v1/courses
{"IOS101":{"Title":"iOS Programming"}}
```

We are finally back with just one course

Securing the REST API using Access Token

1. Add the following statements in bold to the **main.go** file:

```
package main

import (
    "encoding/json"
    "fmt"
    "io/ioutil"
    "log"
    "net/http"

    "github.com/gorilla/mux"
)

type courseInfo struct {
    Title string `json:"Title"`
}

var courses map[string]courseInfo

func validKey(r *http.Request) bool {
    v := r.URL.Query()
    if key, ok := v["key"]; ok {
        if key[0] == "2c78afaf-97da-4816-bbee-9ad239abb296" {
            return true
        } else {
            return false
        }
    } else {
        return false
    }
}

func home(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Welcome to the REST API!")
}

func allcourses(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "List of all courses")
    kv := r.URL.Query()
    for k, v := range kv {
        fmt.Println(k, v)
    }
    // returns all the courses in JSON
    json.NewEncoder(w).Encode(courses)
}

func course(w http.ResponseWriter, r *http.Request) {
    if !validKey(r) {
        w.WriteHeader(http.StatusNotFound)
        w.Write([]byte("401 - Invalid key"))
        return
    }

    params := mux.Vars(r)
    ...
}
```

In real life, the key should be queried from a database

2. Restart the **main.go** (Ctrl-C to stop):

```
$ go run main.go
```

3. Type the following command into the other Terminal window:

```
$ curl http://localhost:5000/api/v1/courses
{}
```

```
$ curl -H "Content-Type:application/json" -X POST
http://localhost:5000/api/v1/courses/IOS101 -d "{\"title\":\"iOS Programming\"}"
401 - Invalid key
```

Access key is not supplied

```
$ curl -H "Content-Type:application/json" -X POST
http://localhost:5000/api/v1/courses/IOS101?key=2c78afaf-97da-4816-bbee-
9ad239abb296 -d "{\"title\":\"iOS Programming\"}"
201 - Course added: IOS101
```

```
$ curl http://localhost:5000/api/v1/courses/IOS101?key=2c78afaf-97da-4816-bbee-
9ad239abb296
{"Title":"iOS Programming"}
```

NGEE ANN
POLYTECHNIC

Lab 4. Connecting to a REST API

Description	*In this lab, you will learn how to write a Go program to connect to the REST API created in the previous lab.*
What You Will Learn	• How to send a request to a REST API using the various methods – GET, PUT, POST, and DELETE • How to add a course to the REST API • How to modify course information on the REST API • How to retrieve course information on the REST API • How to delete a course on the REST API
Duration	40 Minutes

1. Create a folder named **UseREST** within the **GoMS1** folder.

2. Create a file named **main.go** and save it inside the **UseREST** folder.

3. Populate the **main.go** file with the following statements:

```
package main

import (
    "bytes"
    "encoding/json"
    "fmt"
    "io/ioutil"
    "net/http"
)

const baseURL = "http://localhost:5000/api/v1/courses"
const key = "2c78afaf-97da-4816-bbee-9ad239abb296"

func getCourse(code string) {
    url := baseURL
    if code != "" {
        url = baseURL + "/" + code + "?key=" + key
    }
    response, err := http.Get(url)

    if err != nil {
        fmt.Printf("The HTTP request failed with error %s\n", err)
    } else {
        data, _ := ioutil.ReadAll(response.Body)
        fmt.Println(response.StatusCode)
        fmt.Println(string(data))
        response.Body.Close()
    }
}

func addCourse(code string, jsonData map[string]string) {
    jsonValue, _ := json.Marshal(jsonData)

    response, err := http.Post(baseURL+"/"+code+"?key="+key,
        "application/json", bytes.NewBuffer(jsonValue))

    if err != nil {
        fmt.Printf("The HTTP request failed with error %s\n", err)
    } else {
        data, _ := ioutil.ReadAll(response.Body)
        fmt.Println(response.StatusCode)
        fmt.Println(string(data))
        response.Body.Close()
    }
}
```

```
func updateCourse(code string, jsonData map[string]string) {
    jsonValue, _ := json.Marshal(jsonData)

    request, err := http.NewRequest(http.MethodPut,
        baseURL+"/"+code+"?key="+key,
        bytes.NewBuffer(jsonValue))

    request.Header.Set("Content-Type", "application/json")

    client := &http.Client{}
    response, err := client.Do(request)

    if err != nil {
        fmt.Printf("The HTTP request failed with error %s\n", err)
    } else {
        data, _ := ioutil.ReadAll(response.Body)
        fmt.Println(response.StatusCode)
        fmt.Println(string(data))
        response.Body.Close()
    }
}

func deleteCourse(code string) {

    request, err := http.NewRequest(http.MethodDelete,
        baseURL+"/"+code+"?key="+key, nil)

    client := &http.Client{}
    response, err := client.Do(request)

    if err != nil {
        fmt.Printf("The HTTP request failed with error %s\n", err)
    } else {
        data, _ := ioutil.ReadAll(response.Body)
        fmt.Println(response.StatusCode)
        fmt.Println(string(data))
        response.Body.Close()
    }
}

func main() {
    jsonData := map[string]string{"title": "Applied Go Programming"}
    addCourse("IOT210", jsonData)
    // 201
    // 201 - Course added: IOT210

    jsonData = map[string]string{"title": "Applied Python Programming"}
    addCourse("IOT201", jsonData)
    // 201
    // 201 - Course added: IOT201

    jsonData = map[string]string{"title": "Go Concurrency Programming"}
    updateCourse("IOT210", jsonData)
    // 202
    // 202 - Course updated: IOT210

    getCourse("") // get all courses
    // 200
    // {"IOT201":{"Title":"Applied Python Programming"},
    //  "IOT210":{"Title":"Go Concurrency Programming"}}

    getCourse("IOT210") // get a specific course
    // 200
    // {"Title":"Go Concurrency Programming"}

    deleteCourse("IOT201")
    // 202
    //202 – Course deleted: IOT201

    getCourse("") // get all courses
    // 200
    // {"IOT210":{"Title":"Go Concurrency Programming"}}
}
```

The code to make **PUT** and **DELETE** requests is slightly different from **GET**/**POST** calls – the **http.Client** interface has convenient high-level methods for **GET** and **POST** but not for other http methods like **PUT** , **DELETE**, etc.

This is because the Go developers felt it's unrealistic to provide high-level methods for all HTTP verbs

This key "**2c78afaf-97da-4816-bbee-9ad239abb296**" is from the previous lab

4. Restart the **main.go** (Ctrl-C to stop) for the REST API created in the previous lab:

```
$ cd ~/GoMS1/REST
$ go run main.go
Listening at port 5000
```

5. Type the following commands in another Terminal window:

```
$ cd ~/GoMS1/UseREST
$ go run main.go
201
201 - Course added: IOT210
201
201 - Course added: IOT201
202
202 - Course updated: IOT201
200
{"IOT201":{"Title":"Applied Python Programming"},"IOT210":{"Title":"Go
Concurrency Programming"}}

200
{"Title":"Go Concurrency Programming"}

202
202 - Course deleted: IOT201

200
{"IOT210":{"Title":"Go Concurrency Programming"}}
```

Lab 5. Accessing MySQL Databases

Description	*In this lab, you will learn how to write a Go program to communicate with a MySQL database server.*
What You Will Learn	• How to create a new user account in MySQL • How to create a new database in MySQL • How to create a table and a new record • How to use Go to open a MySQL database connection • How to retrieve a record • How to insert a record • How to modify a record • How to delete a record
Duration	60 Minutes

Downloading and Installing the Package

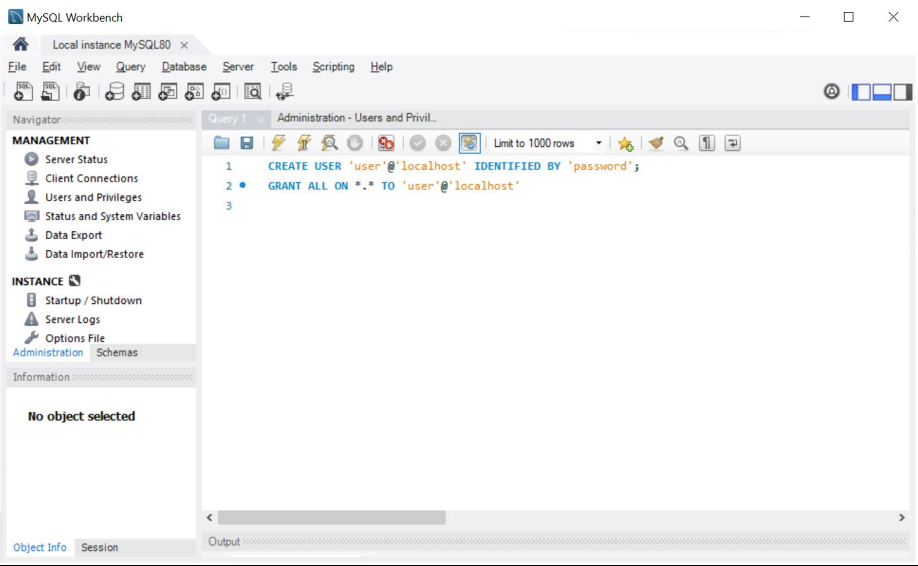
1. In Terminal, type the following command:

```
$ go get "github.com/go-sql-driver/mysql"
```

Creating a New User Account

1. Launch the **MySQL Workbench**, create a new MySQL Connection, and the following window will appear. Type the following command in the **Query 1** window and then click the *lightning* icon:

```
CREATE USER 'user'@'localhost' IDENTIFIED BY 'password';
GRANT ALL ON *.* TO 'user'@'localhost'
```



```
CREATE USER 'user'@'localhost' IDENTIFIED BY 'password';
GRANT ALL ON *.* TO 'user'@'localhost'
```

This will create an account named **user** with the password '**password**'

Creating a Database

1. Type the following command in the **Query1** window and then click the lightning icon:

```
CREATE database my_db;

USE my_db;

CREATE TABLE Persons (ID varchar (5) NOT NULL PRIMARY KEY, FirstName VARCHAR(30),
LastName VARCHAR(30), Age INT);

INSERT INTO Persons (ID, FirstName, LastName, Age) VALUES ("0001", "Jake", "Lee",
25);
```

The above creates a database named **my_db**, followed by creating a table named **Persons** within this newly created database

Finally, a record is inserted into the **Persons** database

Opening a Database Connection

1. Create a folder named **Database** within the **GoMS1** folder.
2. Create a file named **main.go** and save it inside the **database** folder.
3. Populate the **main.go** file with the following statements:

```
package main

import (
    "fmt"
    "database/sql"
    _ "github.com/go-sql-driver/mysql"
)

func main() {
    // Use mysql as driverName and a valid DSN as dataSourceName:
    db, err := sql.Open("mysql", "user:password@tcp(127.0.0.1:3306)/my_db")

    // handle error
    if err != nil {
        panic(err.Error())
    }

    // defer the close till after the main function has finished executing
    defer db.Close()

    fmt.Println("Database opened")



}
```

The “**github.com/go-sql-driver/mysql**” (**Go-MySQL-Driver**) is an implementation of Go's **database/sql/driver** interface. You only need to import the driver and then use the full database/sql APIs

See the following URL for more details:
https://www.calhoun.io/why-we-import-sql-drivers-with-the-blank-identifier/

You should import the **Go-MySQL-Driver** and then use it via the **database/sql** package

4. In Terminal, type the following command:

```
$ cd ~/GoMS1/Database
$ go run main.go
Database opened
```

Retrieving Records

1. Add the following statements in bold to the **main.go** file:

```
package main

import (
    "fmt"
    "database/sql"
    _ "github.com/go-sql-driver/mysql"
)

type Person struct {                    // map this type to the record in the table
    ID int
    FirstName string
    LastName string
    Age int
}

func GetRecords(db *sql.DB) {
    results, err := db.Query("Select * FROM my_db.Persons")

    if err != nil {
        panic(err.Error())
    }

    for results.Next() {
        // map this type to the record in the table
        var person Person
        err = results.Scan(&person.ID, &person.FirstName,
                           &person.LastName, &person.Age)
        if err != nil {
            panic(err.Error())
        }

        fmt.Println(person.ID, person.FirstName,
            person.LastName, person.Age)
    }
}

func main() {
    fmt.Println("Go MySQL Tutorial")
    db, err := sql.Open("mysql", "user:password@tcp(127.0.0.1:3306)/my_db")

    // handle error
    if err != nil {
        panic(err.Error())
    } else {
        fmt.Println("Database opened")
    }

    GetRecords(db)

    // defer the close till after the main function has finished executing
    defer db.Close()
}
```

2. In Terminal, type the following command:

```
$ go run main.go
Database opened
1 Jake Lee 25
```

Inserting Records

1. Add the following statements in bold to the **main.go** file:

```
package main

import (
    "fmt"
    "database/sql"
    _ "github.com/go-sql-driver/mysql"
)

type Person struct {
    ID int
    FirstName string
    LastName string
    Age int
}

func InsertRecord(db *sql.DB, ID int, FN string, LN string, Age int) {
    query := fmt.Sprintf("INSERT INTO Persons VALUES (%d, '%s', '%s', %d)",
        ID, FN, LN,Age)

    _, err := db.Query(query)

    if err != nil {
        panic(err.Error())
    }
}

func GetRecords(db *sql.DB) {
    ...
}

func main() {
    fmt.Println("Go MySQL Tutorial")
    db, err := sql.Open("mysql", "user:password@tcp(127.0.0.1:3306)/my_db")

    // handle error
    if err != nil {
        panic(err.Error())
    } else {
        fmt.Println("Database opened")
    }

    InsertRecord(db, 2, "Wallace","Tan", 55)
    GetRecords(db)

    // defer the close till after the main function has finished executing
    defer db.Close()
}
```

2. In Terminal, type the following command:

```
$ go run main.go
Database opened
1 Jake Lee 25
2 Wallace Tan 55
```

Editing Records

1. Add the following statements in bold to the **main.go** file:

```
package main

import (
    "fmt"
    "database/sql"
    _ "github.com/go-sql-driver/mysql"
)

type Person struct {
    ID int
    FirstName string
    LastName string
    Age int
}

func EditRecord(db *sql.DB, ID int, FN string, LN string, Age int) {
    query := fmt.Sprintf(
        "UPDATE Persons SET FirstName='%s', LastName='%s', Age=%d WHERE ID=%d",
        FN, LN, Age, ID)
    _, err := db.Query(query)
    if err != nil {
        panic(err.Error())
    }
}

func InsertRecord(db *sql.DB, ID int, FN string, LN string, Age int) {
    ...
}

func GetRecords(db *sql.DB) {
    ...
}

func main() {
    fmt.Println("Go MySQL Tutorial")
    db, err := sql.Open("mysql", "user:password@tcp(127.0.0.1:3306)/my_db")

    // handle error
    if err != nil {
        panic(err.Error())
    } else {
        fmt.Println("Database opened")
    }

    // InsertRecord(db, 2, "Wallace","Tan", 55)
    EditRecord(db, 2, "Taylor", "Swift", 23)

    GetRecords(db)

    // defer the close till after the main function has finished executing
    defer db.Close()
}
```

2. In Terminal, type the following command:

```
$ go run main.go
Database opened
1 Jake Lee 25
2 Taylor Swift 23
```

Deleting Records

1. Add the following statements in bold to the **main.go** file:

```
package main

import (
    "fmt"
    "database/sql"
    _ "github.com/go-sql-driver/mysql"
)

type Person struct {
    ID int
    FirstName string
    LastName string
    Age int
}

func DeleteRecord(db *sql.DB, ID int) {
    query := fmt.Sprintf(
        "DELETE FROM Persons WHERE ID='%d'", ID)
    _, err := db.Query(query)
    if err != nil {
        panic(err.Error())
    }
}

func EditRecord(db *sql.DB, ID int, FN string, LN string, Age int) {
    ...
}

func InsertRecord(db *sql.DB, ID int, FN string, LN string, Age int) {
    ...
}

func GetRecords(db *sql.DB) {
    ...
}

func main() {
    fmt.Println("Go MySQL Tutorial")
    db, err := sql.Open("mysql", "user:password@tcp(127.0.0.1:3306)/my_db")

    // handle error
    if err != nil {
        panic(err.Error())
    } else {
        fmt.Println("Database opened")
    }

    // InsertRecord(db, 2, "Michael","Jackson", 55)
    // EditRecord(db, 2, "Taylor", "Swift", 23)
    DeleteRecord(db, 2)

    GetRecords(db)
    // defer the close till after the main function has finished executing
    defer db.Close()
}
```

2. In Terminal, type the following command:

```
$ go run main.go
Database opened
1 Jake Lee 25
```

NGEE ANN POLYTECHNIC

Lab 6. Getting Started with Docker

Description	*In this lab, you will learn how to get started with Docker.*
What You Will Learn	• How to install Docker on your computer • How to check the version of Docker installed
Duration	20 minutes

Downloading and Installing Docker

1. Using a Web browser, go to https://docs.docker.com/docker-for-windows/install/ and download Docker for the OS you are using.

Getting Version of Docker

```
$ docker version
Client: Docker Engine - Community
 Version:           19.03.8
 API version:       1.40
 Go version:        go1.12.17
 Git commit:        afacb8b
 Built:             Wed Mar 11 01:21:11 2020
 OS/Arch:           darwin/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          19.03.8
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.12.17
  Git commit:       afacb8b
  Built:            Wed Mar 11 01:29:16 2020
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          v1.2.13
  GitCommit:        7ad184331fa3e55e52b890ea95e65ba581ae3429
 runc:
  Version:          1.0.0-rc10
  GitCommit:        dc9208a3303feef5b3839f4323d9beb36df0a9dd
 docker-init:
  Version:          0.18.0
  GitCommit:        fec3683
```

```
$ docker --version
Docker version 19.03.8, build afacb8b
```

NGEE ANN POLYTECHNIC

Lab 7. Docker Management

Description	*In this lab, you will learn how to manage your Docker images and containers.*
What You Will Learn	• How to create a container from a Docker image • How to view all the containers on your computer • How to run multiple containers from the same image • How to remove a container • How to remove all exited containers • How to list the last container executed • How to stop all containers • How to remove all containers • How to list all the Docker images • How to remove Docker images • How to download Docker images • How to get help in Docker
Duration	60 minutes

Creating a Container from an Image

1. In Terminal, type the following command:

```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest: sha256:6a65f928fb91fcfbc963f7aa6d57c8eeb426ad9a20c7ee045538ef34847f44f1
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

The **`docker run hello-world`** command downloads the **hello-world** *Docker Image* from *Docker Hub* and then creates a *Docker Container* using this image; it then assigns a random name to the container and then starts it

The above container then exits

To explicitly name the container, use the **`--name`** option, like this:

```
$ docker run --name helloworld hello-world
```

Viewing all Containers

1. Type the following command in Terminal:

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
```

The **docker ps** command shows all the *running* containers

2. Type the following command in Terminal:

```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS                      PORTS        NAMES
3f3b353cecbc        hello-world         "/hello"                 5 minutes ago
Exited (0) 5 minutes ago                 blissful_ardinghelli
```

The **docker ps –a** command shows *all* the containers

The container name (**`blissful_ardinghelli`**) is randomly assigned

Creating and Running Multiple Containers using the Same Image

1. Type the following command in Terminal:

```
$ docker run hello-world
```

2. Observe that you will get the similar output as you have seen in the previous section.

3. Type the following command in Terminal:

```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
241e627cc3fa        hello-world         "/hello"            19 seconds ago
Exited (0) 19 seconds ago                       nervous_kare
3f3b353cecbc        hello-world         "/hello"            8 minutes ago
Exited (0) 8 minutes ago                        blissful_ardinghelli
```

The **docker run** command will always create a new container from an image

Removing a Container

1. Type the following command in Terminal:

```
$ docker rm c251a834b184
c251a834b184
```

The **docker rm <*container id*>** command removes the specified container

Removing all Exited Containers

```
$ docker rm $(docker ps -a -q -f status=exited)
```

-q option only returns numeric IDs
-f option filters output based on specified conditions

For Windows users, use the following command in **PowerShell**:

```
docker rm @(docker ps -aq)
```

Listing the Last Container Executed

```
$ docker ps -lq
```

Stopping all Containers

```
$ docker stop $(docker ps -a -q)
```

For Windows users, use the following command in **PowerShell**:

```
docker ps -q | % { docker stop $_ }
```

Removing All Containers

```
$ docker rm $(docker ps -a -q)
```

For Windows users, use the following command in **PowerShell**:

```
docker ps -a -q | % { docker rm $_ }
```

Listing the Docker Images

1. Type the following command in Terminal:

```
$ docker images
REPOSITORY                 TAG                 IMAGE ID            CREATED
SIZE
hello-world                latest              bf756fb1ae65        4 months ago
13.3kB
```

The **docker images** command list all the images on your local computer

Removing Docker Images

1. Type the following command in Terminal:

```
$ docker rmi hello-world
Untagged: hello-world:latest
Untagged: hello-
world@sha256:6a65f928fb91fcfbc963f7aa6d57c8eeb426ad9a20c7ee045538ef34847f44f1
Deleted: sha256:bf756fb1ae65adf866bd8c456593cd24beb6a0a061dedf42b26a993176745f6b
Deleted: sha256:9c27e219663c25e0f28493790cc0b88bc973ba3b1686355f221c38a36978ac63
```

The **docker rmi <*repository*>** command deletes the specified image

If there are container(s) using the image that you are deleting, there will be an error, like this:
Error response from daemon: conflict: unable to remove repository reference "hello-world" (must force) - container c69e0a39d107 is using its referenced image c54a2cc56cbb

Downloading Docker Images to Local Computer

```
$ docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
c04b14da8d14: Pull complete
Digest: sha256:0256e8a36e2070f7bf2d0b0763dbabdd67798512411de4cdcf9431a1feb60fd9
Status: Downloaded newer image for hello-world:latest
```

Getting Help

1. To get help from Docker, type the following command in Terminal:

```
$ docker help
```

2. Type the following command to get help on the **run** command:

```
$ docker help run
```

Lab 8. Using the Ubuntu Docker Image

Description	*In this lab, you will learn how to use the Ubuntu Docker Image*
What You Will Learn	• How to run a Ubuntu Docker container • How to install curl on the Ubuntu container • How to run a command in a running container
Duration	20 minutes

Ubuntu is a Debian-based Linux operating system that runs from the desktop to the cloud, to all your internet connected things.

1. Type the following command in Terminal to download the Ubuntu image, create a container and then start it:

```
$ docker run -it ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
d51af753c3d3: Already exists
fc878cd0a91c: Already exists
6154df8ff988: Already exists
fee5db0ff82f: Already exists
Digest: sha256:747d2dbbaaee995098c9792d99bd333c6783ce56150d1b11e333bbceed5c54d7
Status: Downloaded newer image for ubuntu:latest

root@b560195c0fe5:/# apt-get update
Get:1 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:2 http://security.ubuntu.com/ubuntu focal-security InRelease [107 kB]
...
Get:16 http://archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 Packages
[1079 B]
Get:17 http://archive.ubuntu.com/ubuntu focal-backports/universe amd64 Packages
[2903 B]
Fetched 13.9 MB in 4s (3386 kB/s)
Reading package lists... Done

root@b560195c0fe5:/# apt-get install -y curl
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  ca-certificates krb5-locales libasn1-8-heimdal libbrotli1 libcurl4 libgssapi-
krb5-2 libgssapi3-heimdal libhcrypto4-heimdal libheimbase1-heimdal
  libheimntlm0-heimdal libhx509-5-heimdal libk5crypto3 libkeyutils1 libkrb5-26-
...
Setting up libcurl4:amd64 (7.68.0-1ubuntu2) ...
Setting up curl (7.68.0-1ubuntu2) ...
Processing triggers for libc-bin (2.31-0ubuntu9) ...
Processing triggers for ca-certificates (20190110ubuntu1) ...
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.

root@b560195c0fe5:/# curl
curl: try 'curl --help' or 'curl --manual' for more information

root@b560195c0fe5:/# exit
```

```
exit
```

The **-it** option instructs Docker to allocate a pseudo-TTY connected to the container's stdin; creating an interactive **bash** shell in the container

You first run the **apt-get update** command to update the package cache on the container

You then install the **curl** application on the **Ubuntu** container

When a container exits, all the changes that you have made to it are lost

2. Type the following command in Terminal:

```
$ docker run -it ubuntu bash

root@dbfa4381b077:/# curl
bash: curl: command not found

root@dbfa4381b077:/# exit
exit
```

In the above, a new container is started, and hence **curl** cannot be found

3. To get back the previous container, type the following commands:

```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                        PORTS               NAMES
dbfa4381b077        ubuntu              "bash"              About a minute ago
Exited (127) 33 seconds ago                       vigorous_bhaskara
b560195c0fe5        ubuntu              "bash"              11 minutes ago
Exited (0) 2 minutes ago                          elastic_mcnulty
```

The above shows the two most recent Docker containers

Locate the container that you had used to install curl and take note of its name

4. Type the following command in Terminal to start the container:

```
$ docker start elastic_mcnulty
elastic_mcnulty
```

5. Type the following command in Terminal to connect to the container:

```
$ docker exec -it elastic_mcnulty bash
root@b560195c0fe5:/# curl
curl: try 'curl --help' or 'curl --manual' for more information
root@b560195c0fe5:/#
```

The **exec** option runs a command in a running container

You should now be able to find the curl application on the container

Exercises

1. Install nano and vim on your Ubuntu container

Commiting the Last Container into a New Docker Image

1. Type the following command in Terminal:

```
$ docker commit $(docker ps -lq) ubuntu_curl
```

Windows users need to run the above command in PowerShell

The above command commits the last container into a new Docker image

2. Type the following command in Terminal:

```
$ docker images
REPOSITORY                              TAG                 IMAGE ID            CREATED
SIZE
ubuntu_curl                             latest              b98199311660        3 minutes
ago         116MB
...
```

You should now find a new Docker image named **ubuntu_curl**

3. Type the following command in Terminal:

```
$ docker run -it ubuntu_curl bash
root@51e1a0e3fbb1:/# curl
curl: try 'curl --help' or 'curl --manual' for more information
```

NGEE ANN POLYTECHNIC

Lab 9. Powering a Web Server using NginX

Description	*In this lab, you will learn how to start a web server using NginX.*
What You Will Learn	• How to start nginx • How to create your own web page and serve it using nginx • How to start and stop a container • How to modify a Docker image
Duration	30 minutes

1. Type the following command in Terminal:

```
$ docker run -d -p 88:80 --name webserver nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
afb6ec6fdc1c: Pull complete
b90c53a0b692: Pull complete
11fa52a0fdc0: Pull complete
Digest: sha256:30dfa439718a17baafefadf16c5e7c9d0a1cde97b4fd84f63b69e13513be7097
Status: Downloaded newer image for nginx:latest
8438d0ac513097baa2f74b52a24ef6e88aa69f01640a102405c52bd47e0f65a1
```

The **–d** option (or **--detach**) runs the container in background

The **–p** option publishes the exposed port to specified port; here *88:80* means all traffic to port 88 on the client is forwarded to port 80 in the container

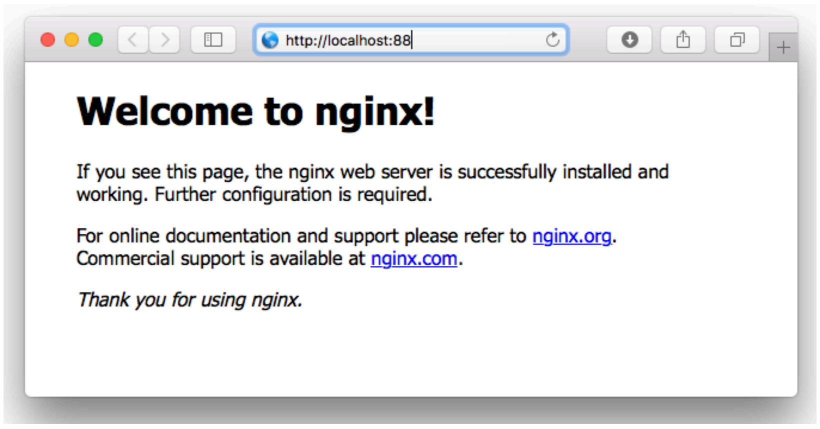
The **--name** option assigns a name to the container

The **nginx** is an image from Docker Hub; **nginx** [engine x] is an HTTP and reverse proxy server, a mail proxy server, and a generic TCP/UDP proxy server, originally written by Igor Sysoev.

```
$ docker port webserver
80/tcp -> 0.0.0.0:88
```

The port option list the port mappings for a container

2. Using a web browser, type the URL: http://localhost:88 and you should see the following:



Welcome to nginx!

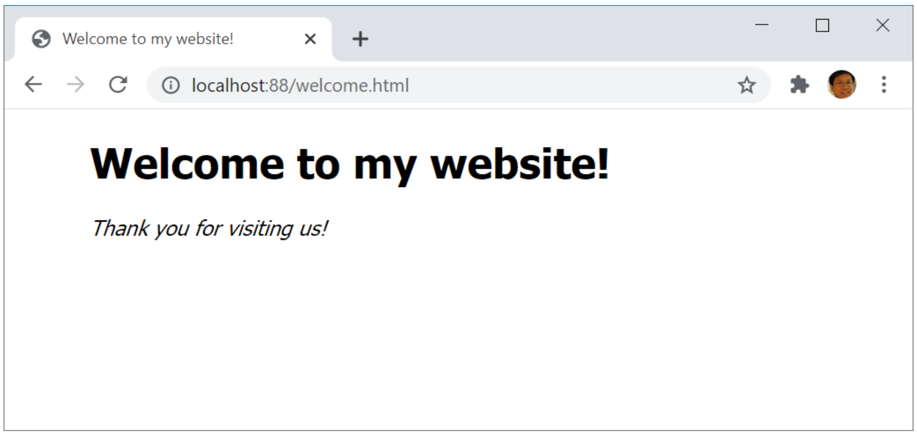
If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Exercises

1. Create a page named **welcome.html** to display some welcome messages.



Welcome to my website!

Thank you for visiting us!

Stopping a Container

1. Type the following command in Terminal:

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS              PORTS                NAMES
8438d0ac5130        nginx               "nginx -g 'daemon of…"   3 minutes ago
Up 3 minutes        0.0.0.0:88->80/tcp   webserver
```

The **docker ps** command shows all running containers

2. Type the following command in Terminal:

```
$ docker stop 8438d0ac5130
8438d0ac5130
```

The **docker stop** command stops the specified container

You can also stop a container by its name:
`docker stop webserver`

To stop and remove a running container, use the command **docker rm –f <*container id*>**

Starting a Container

1. Type the following command in Terminal:

```
$ docker start 8438d0ac5130
8438d0ac5130
```

The **docker start** command starts the specified container

Modifying the Docker Image

1. Type the command in Terminal:

```
$ docker stop 8438d0ac5130
```

2. In the **~/GoMS1** folder, create a new file named **index.html** and populate it with the following statements:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Hello World</title>
    <style>
        h1{
            font-weight:lighter;
            font-family: Helvetica, sans-serif;
        }
    </style>
</head>
<body>
    <h1>
        Hello, Docker!
    </h1>
</body>
</html>
```

3. In the **~/MSG01** folder, create a new file named **Dockerfile** and populate it with the following statements:

```
FROM nginx
COPY index.html /usr/share/nginx/html/index.html
```

4. In Terminal, type the following command:

```
$ cd ~/GoMS1
$ docker build -t nginx .
Sending build context to Docker daemon  8.704kB
Step 1/2 : FROM nginx
 ---> 6af5193097be
Step 2/2 : COPY index.html /usr/share/nginx/html/index.html
 ---> 1077f8f61699
Successfully built 1077f8f61699
Successfully tagged nginx:latest
```

The above command will copy the local **index.html** file into the Docker image as **/usr/share/nginx/html/index.html**

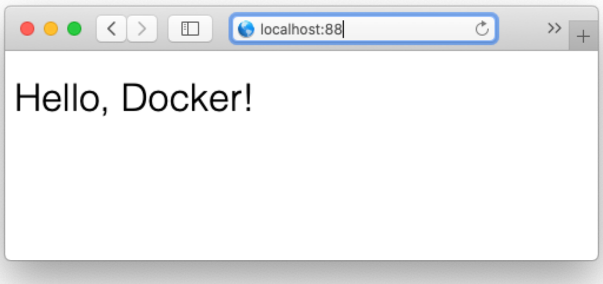
The instructions to copy the file is in **Dockerfile**

The changes is made to the Docker image, and it will affect all containers created from this image

5. In Terminal, type the following command:

```
$ docker run -d -p 88:80 nginx
```

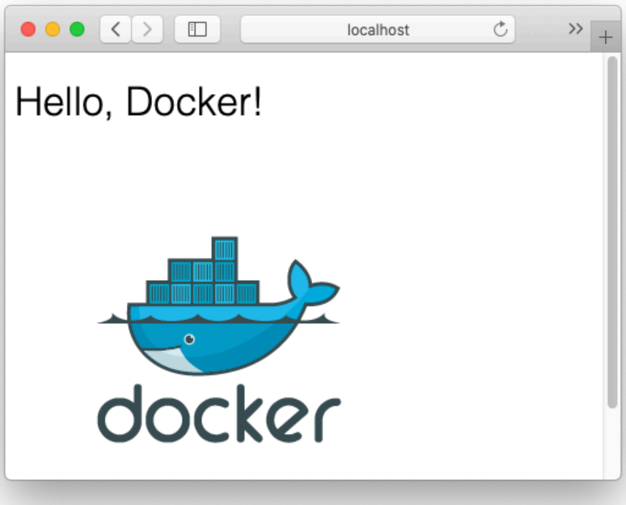
3. Using a web browser, type the URL: http://localhost:88 and you should see the following:



Hello, Docker!

Exercises

1. Modify the **index.html** file to include the following image:



Lab 10. Using MySQL in Docker

Description	*In this lab, you will learn how to use MySQL in Docker.*
What You Will Learn	• How to run a container with MySQL installed • How to inspect a container • How to connect to the MySQL container using a local MySQL client • How to create a database and table in MySQL • How to insert a record into a table • How to retrieve a record from a table • How to connect to the MySQL container using another MySQL container • How to examine the storage area for the MySQL database • How to specify the connection strings for your Go program to connect to a MySQL container
Duration	60 minutes

MySQL is the world's most popular open source database.

Creating the Container

1. Type the following command in Terminal:

```
$ docker run --name My-mysql -p 3306 -e MYSQL_ROOT_PASSWORD=password -d
mysql:latest
Unable to find image 'mysql:latest' locally
latest: Pulling from library/mysql
afb6ec6fdc1c: Pull complete
0bdc5971ba40: Pull complete
97ae94a2c729: Pull complete
f777521d340e: Pull complete
1393ff7fc871: Pull complete
a499b89994d9: Pull complete
7ebe8eefbafe: Pull complete
597069368ef1: Pull complete
ce39a5501878: Pull complete
7d545bca14bf: Pull complete
211e5bb2ae7b: Pull complete
5914e537c077: Pull complete
Digest: sha256:a31a277d8d39450220c722c1302a345c84206e7fd4cdb619e7face046e89031d
Status: Downloaded newer image for mysql:latest
b11df0806a6cae8f19183b082cbf33b8cc15a4deebe1b05c7c770b5362a37e9c
```

The above command loads the latest version of the MySQL Docker image, creates a container and then runs it

It gives the container the name of **My-mysql**

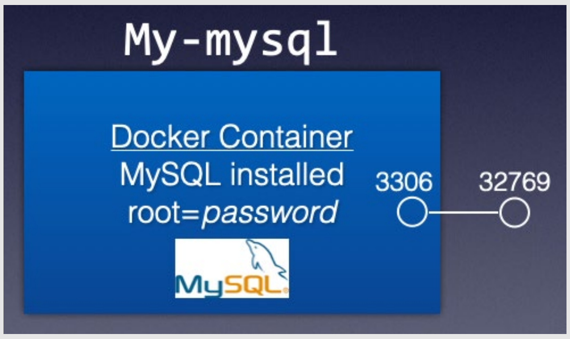
It sets the environment variable **MYSQL_ROOT_PASSWORD** to ***password***; which is the password of the **root** account

It also expose port 3306 so that you can interact with MySQL through this port

2. Type the following command in Terminal:

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS              PORTS                            NAMES
b11df0806a6c        mysql:latest        "docker-entrypoint.s…"   32 seconds ago
Up 30 seconds       33060/tcp, 0.0.0.0:32769->3306/tcp   My-mysql
```

Observe that the port **32769** is mapped to **3306**; this means that if you want to interact with MySQL, you need to connect to this port – **32769** (randomly assigned); *you may see a different port number*



If you want to map the container to a specific port, use this option when starting the container:

-p *32769*:3306

3. You can also use the following command to list the port mappings on the My-mysql container:

```
$ docker port My-mysql
3306/tcp -> 0.0.0.0:32769
```

Inspecting the Docker Container

1. Type the following command in Terminal:

```
$ docker inspect My-mysql
[
    {
        "Id": "e3a05e2c8df96120702c6648911be5bd1b27ec7632e13fcbd7b38743f5f5846a",
        "Created": "2020-06-05T09:45:52.589650696Z",
        "Path": "docker-entrypoint.sh",
        "Args": [
            "mysqld"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 10163,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2020-06-05T09:45:52.870891798Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
...
        "Mounts": [
            {
                "Type": "volume",
                "Name":
"71e8528f7fe9704929f1f8221c28ee14c445660151819a27befc272da69b50ff",
                "Source":
"/var/lib/docker/volumes/71e8528f7fe9704929f1f8221c28ee14c445660151819a27befc272d
a69b50ff/_data",
                "Destination": "/var/lib/mysql",
                "Driver": "local",
                "Mode": "",
                "RW": true,
                "Propagation": ""
            }
        ],
...
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID":
"3d80dfe5a271a661a87fb7e5dfedb597fcc1b1ae71e09891911afbec24799f2a",
                    "EndpointID":
"a977e1008f38caf3915ee8ace0049d0ce61a3dad162d6fde9f9cefeb5faa12dd",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.3",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:03",
                    "DriverOpts": null
                }
            }
        }
    }
]
```

Observe the value of "**Source**", which is: "**/var/lib/docker/volumes/71e8528f7fe9704929f1f8221c28ee14c445660151819a27befc272da69b50ff/_data**". This refers to the actual directory on the host; for a **Linux** machine, this directory refers to an actual directory on the local computer; on a **Mac** and **Windows** directory this is a directory on the Docker VM

The "**Destination**" key has the value of "**/var/lib/mysql**". This refers to the (logical) directory in the Docker container

Connecting to the MySQL Container using a Local MySql Client

This section assumes that you already have the **mysql** client installed on your local computer.

You can download and install MySQL from: https://dev.mysql.com/downloads/mysql/

During the installation stage, you can choose to only install the client

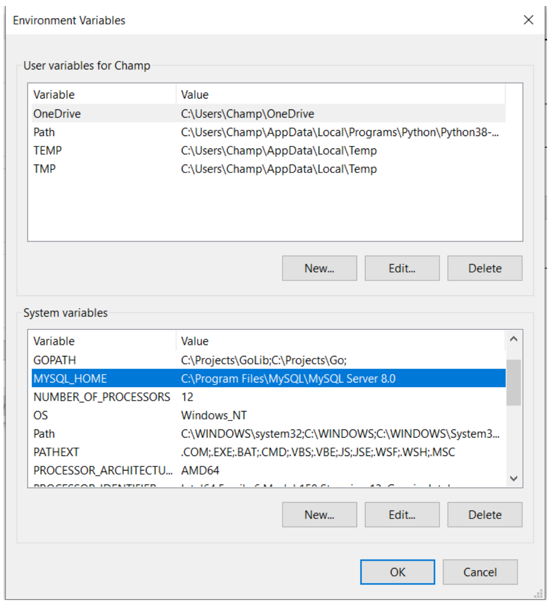
1. Type the following command in Terminal:

```
$ mysql -P 32769 --protocol=tcp -u root -p
Enter password: <password>
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.20 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Replace the 32769 with the port number assigned to your **MySQL** container

For Windows user, the **mysql.exe** utility is located in
C:\Program Files\MySQL\MySQL Server 8.0\bin\

Need to add it to environment variable path



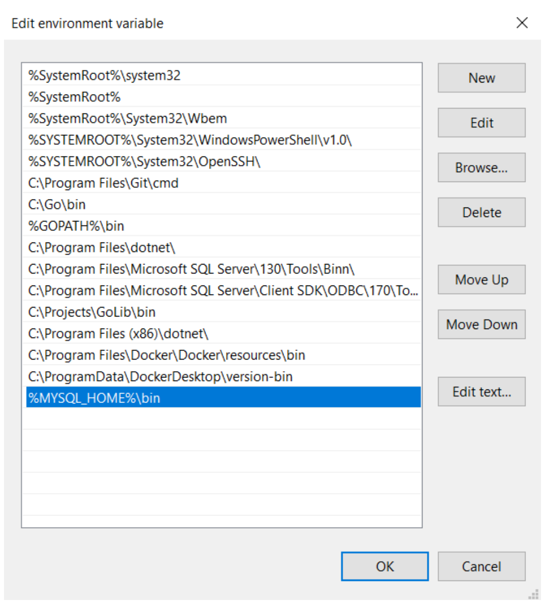
Environment Variables

User variables for Champ

Variable	Value
OneDrive	C:\Users\Champ\OneDrive
Path	C:\Users\Champ\AppData\Local\Programs\Python\Python38-...
TEMP	C:\Users\Champ\AppData\Local\Temp
TMP	C:\Users\Champ\AppData\Local\Temp

System variables

Variable	Value
GOPATH	C:\Projects\GoLib;C:\Projects\Go;
MYSQL_HOME	C:\Program Files\MySQL\MySQL Server 8.0
NUMBER_OF_PROCESSORS	12
OS	Windows_NT
Path	C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System3...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTU...	AMD64



Edit environment variable

```
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
C:\Program Files\Git\cmd
C:\Go\bin
%GOPATH%\bin
C:\Program Files\dotnet\
C:\Program Files\Microsoft SQL Server\130\Tools\Binn\
C:\Program Files\Microsoft SQL Server\Client SDK\ODBC\170\To...
C:\Projects\GoLib\bin
C:\Program Files (x86)\dotnet\
C:\Program Files\Docker\Docker\resources\bin
C:\ProgramData\DockerDesktop\version-bin
%MYSQL_HOME%\bin
```

Creating a Database, Table, and Inserting a Row

1. Type the following in the mysql client prompt:

```
mysql> CREATE database my_db;
Query OK, 1 row affected (0.01 sec)

mysql> USE my_db;
Database changed

mysql> CREATE TABLE Persons (ID varchar (5) NOT NULL PRIMARY KEY, FirstName
VARCHAR(30), LastName VARCHAR(30), Age INT);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO Persons (ID, FirstName, LastName, Age) VALUES ("0001", "Wei-
Meng", "Lee", 25);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM Persons;
+------+-----------+----------+------+
| ID   | FirstName | LastName | Age  |
+------+-----------+----------+------+
| 0001 | Wei-Meng  | Lee      |   25 |
+------+-----------+----------+------+
1 row in set (0.00 sec)

mysql>
```

Connecting your local Go program to a MySQL Docker Container

To connect a local *Go* program to the MySQL Docker container, make sure you change the connection string to use the port exposed by the container:

```
    db, err := sql.Open("mysql", "root:password@tcp(127.0.0.1:32769)/my_db")
```

NGEE ANN POLYTECHNIC

Lab 11. Running Go REST API in Docker

Description	*In this lab, you will learn how to backup and restore Docker Images.*
What You Will Learn	• How to download a Go Docker container • How to load a Go Docker container with a REST API • How to run a REST API from a Docker container • How to backup Docker images • How to restore Docker images
Duration	40 minutes

Preparing the Docker Image to Run the REST API

1. Type the following in Terminal:

```
$ docker run -it golang
root@fc151e2109dd:/go# pwd
/go
root@fc151e2109dd:/go# cd /go
root@fc151e2109dd:/go# ls
bin  src
root@fc151e2109dd:/go# cd src
root@fc151e2109dd:/go/src# pwd
/go/src
root@fc151e2109dd:/go/src#
```

The above step downloads the **golang** Docker image

The source directory for the Go Docker image is **/go/src**

This is where you will copy your **main.go** file to

2. Create a new file named **Dockerfile** and save it in the **REST** folder:

```
~/GoMS1
    |__REST
       |__main.go
       |__Dockerfile
```

3. Populate the **Dockerfile** with the following:

```
FROM golang

WORKDIR /go/src

RUN go get -u github.com/gorilla/mux

COPY . .

CMD ["go", "run", "main.go"]
```

The **Dockerfile** specifies the instructions for creating the Docker image:

Use the **golang** Docker image

Set the working directory to **/go/src**

Execute the following command in the container (it installs the **gorilla** library): **`go get -u github.com/gorilla/mux`**

Copy all the files in the current directory (**~/GoMS1/REST**) to the working directory (**/go/src**)

Execute the Go program – **`go run main.go`**

4. Type the following commands in Terminal:

```
$ cd ~/GoMS1/REST
$ docker build -t golang .
Sending build context to Docker daemon  10.24kB
Step 1/5 : FROM golang
latest: Pulling from library/golang
Digest: sha256:4d58164df51901679307e547ffed19837e63a1d1b9ed80453b8d88749b6033f4
Status: Downloaded newer image for golang:latest
 ---> 05feda542433
Step 2/5 : WORKDIR /go/src
 ---> Running in c5699e8d0145
Removing intermediate container c5699e8d0145
 ---> 94e3f3f18e39
Step 3/5 : RUN go get -u github.com/gorilla/mux
 ---> Running in 0086cd0c3d8d
Removing intermediate container 0086cd0c3d8d
 ---> e1d5e9b5da12
Step 4/5 : COPY . .
 ---> 850edc459a24
Step 5/5 : CMD ["go", "run", "main.go"]
 ---> Running in 4674af9c799e
Removing intermediate container 4674af9c799e
 ---> 9bbad2fc853f
Successfully built 9bbad2fc853f
Successfully tagged golang:latest
```

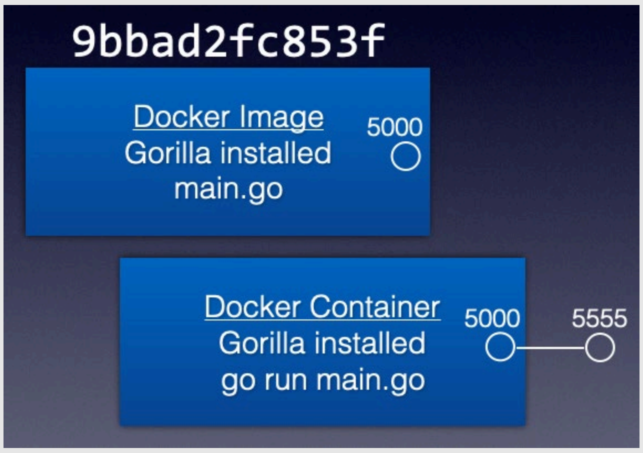
golang is the Docker image name

The above command creates a Docker image with the configuration specified in the **Dockerfile**

5. Type the following command in Terminal:

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
golang              latest              9bbad2fc853f        31 seconds ago
811MB
...
```

```
$ docker run -p 5555:5000 9bbad2fc853f
```



The above runs the **golang** Docker image as a container

The **main.go** program is automatically run and is listening at port **5000** (internally) and is mapped to an external port **5555**

6. In a *new* Terminal window, type the following command:

```
$ curl http://localhost:5555/api/v1/courses
{}
```

The above output confirms that the REST API within the Docker container is working

Backing Up Docker Images

1. Type the following command in Terminal:

```
$ docker images
REPOSITORY           TAG                  IMAGE ID             CREATED              SIZE
golang               latest               a787fb80d096         21 hours ago         821MB
```

```
$ docker save -o my_golang.tar golang:latest
```

```
$ ls
my_golang.tar
```

You have now made a backup copy of the **golang** Docker image

Restoring Docker Images

1. Copy all the .tar files created in the previous section onto another computer with Docker installed.

2. Type the following commands in Terminal:

```
$ docker load -i my_golang.tar
8c02234b8605: Loading layer [==================================================>]
119.2MB/119.2MB                    7cc1c2d7e744: Loading layer
[==================================================>]  17.11MB/17.11MB
760e8d95cf58: Loading layer [==================================================>]
17.85MB/17.85MB                    88cfc2fcd059: Loading layer
[==================================================>]    150MB/150MB
2f6bb5712e22: Loading layer [==================================================>]
...
375.8kB/375.8kB                    8ea1fcb733ab: Loading layer
[==================================================>]  805.9kB/805.9kB
1f9260d8837c: Loading layer [==================================================>]
13.31kB/13.31kB                    Loaded image: golang:latest
```

3. You can now start the **golang** Docker image as a container.

```
$ docker run -p 5555:5000 9bbad2fc853f
```