https://bit.ly/shiju-go

Shiju Varghese’s Masterclass: Professional Go

Assignments

1: Go Basics and Functions

Objective: To get familiarise Go basics

- Define a package level variable of type map[string]string
- Create functions for add, update, get, getAll and delete on map

(package level variable of type map)

```
// Declare package level variable for storing map
var data map[string]string
// init function will be automatically invoked before main function
// init function is used to initialise package level variables
func init() {
    data = make(map[string]string) // Initialise map with make
}
func add(k,v string) {
 // ToDo: Check if key exists
   data [k] = v
}
```

2: Package and Data Models

Objective: Write idiomatic Go code with packages, struct and interface

Principles

- SOLID and Clean Architecture
- Explicit Dependencies: Methods and classes should explicitly require (typically through method parameters or constructor parameters) any collaborating objects they need in order to function correctly.

- Declarative Composition: Removes the dependent logic from the composition process.
- "Be conservative in what you send, be liberal in what you accept" – Robustness principle
- "Accept interfaces, return structs" -– A Go Proverb

<u>Create package named domain</u>

- Create a struct named Customer.
- Create an interface named CustomerStore to specify behaviours for CRUD on Customer.

```
type Customer struct {
	ID, Name, Email string
}
type CustomerStore interface {
	Create (Customer) error
	Update  (string, Customer) error
	Delete(string) error
	GetById(string) (Customer, error)
	GetAll() ([]Customer, error)
}
```

<u>Create package named mapstore</u>

- Implement interface CustomerStore into a struct MapStore to persist Customer data into a Map store (map[string]Customer)

```
  type MapStore struct {
      store map[string]domain.Customer // An in-memory store
       with a map
  }

// Factory method gives a new instance of MapStore
// This is for caller packages, not for mapstore
func NewMapStore() *MapStore {
  return &MapStore { store: make(map[string]domain.Customer)}
}

// Implement interface methods of domain.CustomerStore
```

Create package main

- Create CustomerController struct

```
type CustomerController struct {
                // Explicit dependency and declarative programming
that hides dependent logic
           store domain.CustomerStore // It can be any Store
including MapStore
      }
      func (c CustomerController ) Add (c domain.Customer) {
        err:= c.store.Create(c)
           if err!=nil {
                fmt.Println("Error:", err)
                return
           }
         fmt.Println("New Customer has been created")
     }

func main() {
   controller := CustomerController{ //FAcade
            store : mapstore.NewMapStore(), // Inject the
dependency
// store: = mongodb.NewMongoStore()
}

customer := Customer {
    ID : "cust101",
   Name: "JPM",
}

controller.Add(customer) // Create new Customer
```

- By using CustomerController, make CRUD operations on Customer into a in-memory map store.

3: HTTP Programming

Objective: Write RESTful APIs with Gorilla Mux package
From assignment 2:

- Create a package named controller and move the CustomerController type into this package in order to implement handlers (Eg: func (ctl CustomerController) Post(w http.ResponseWriter, r *http.Request))
- for CRUD on Customer entity
- Create a package named router and configure all routes into it.
- Create a package main and start a new http server from it.

4: TDD/BDD Unit Tests

Objective: Write TDD/BDD style unit tests

- Write BDD style unit tests for assignment 3