https://bit.ly/shiju-go | https://shijuvar.medium.com

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 making insert, update, delete and get items to and from the map (package level variable of type map) with the following signature:

- addItem (k,v string)
- updateItem (k,v string)
- getById (k string)
- getAll()
- deleteItem (k string)

```
// 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 addItem(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

Create package named domain

- 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)
}
```

Create package named mapstore

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

```
// MapStore is an implementation of CustomerStore interface
```

```
  type MapStore struct {
      // An in-memory store with a map
      // Use Customer.ID as the key of map
       store map[string]domain.Customer
  }

// Factory method gives a new instance of MapStore
// This is for caller packages to create MapStore instances
func NewMapStore() *MapStore {
  return &MapStore { store: make(map[string]domain.Customer)}
}

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

Create package main

- Create CustomerController struct

```
// Organises the CRUD operations at UI layer
type CustomerController struct {
// Explicit dependency that hides dependent logic
   store domain.CustomerStore // CustomerStore value
 }

 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{
     store : mapstore.NewMapStore(), // Inject the dependency
   // store : mongodb.NewMongoStore(), // with another database
}


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