Workshop de GoLang: 2021

1. Configurando el Ambiente

Herramientas

Se deben instalar las siguientes herramientas:

GoLang

Sitio de descarga: https://golang.org/dl/

Probar que está instalado con el siguiente comando:

```
$ go version
```

Visual Studio Code

1. Instalar el VS-Code: https://code.visualstudio.com/download
2. Ir al módulo de **Extensions** y buscar con el nombre `Go`
3. Ejecutar Command Palette (`Ctrl+Shift+P`) y seleccionar `Go: Install/Update Tools`
4. Configurar el *settings.json*

```
{
    ....
    "go.testOnSave": false,
    "go.lintOnSave": "package",
    "go.formatTool": "goimports",
    "go.testFlags": [
        "-v"
    ],
    "go.autocompleteUnimportedPackages": true,
    "[go]": {
        "editor.insertSpaces": true,
        "editor.formatOnSave": true,
        "editor.codeActionsOnSave": {
            "source.organizeImports": true
        }
    }
}
```

2. Creando un "Hola Mundo"

1. Crear el directorio de trabajo. Ejm: **workshop-go**
2. Crear el archivo **main.go**
3. Agregar las líneas para imprimir un "Hola Mundo".
4. Inicializamos el proyecto como un módulo.

```
$ go mod init workshop-go
```

Agregando Librerías

1. Se agrega una librería para el manejo de variables de entorno o inputs

```
$ go get github.com/namsral/flag
```

2. Cargamos una variable con `flag.String()` y `flag.Parse()`

Creando un servidor HTTP

1. Se utiliza la librería `net/http`
2. Se crea el handler para

```
func newHandler(w http.ResponseWriter, r *http.Request) {
  fmt.Fprintf(w, "Hello, %s!", r.URL.Path[1:])
}
```

3. Se crea el `ListenAndServe` para escuchar en la dirección (IP + Puerto) de entrada.

Utilizando la libreria de JSON

1. Crear el objeto a ser devuelto como `struct`
2. Establecer la cabecera con el `Content-Type` de salida:

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

3. Codificar la respuesta:

```
json.NewEncoder(w).Encode(payload)
```

Utilizando los Tags en las estructuras

1. En los campos de la estructura se deben colocar Tags

```
type MyObject struct {
  Date    time.Time  `json:"CreationTime"`
}
```

Utilizando una base de datos en Memoria

1. Importar la dependencia gomembd

```
$ go get github.com/hashicorp/go-memdb
```

2. Crear un archivo llamado `schema.go` y en el colocar lo siguiente:

```
var (
  instance *memdb.MemDB
  lock     = &sync.Mutex{}
)
```

3. Dentro del mismo archivo se inicializa la base de datos con la tabla a manejar:

```
func createSchema() *memdb.MemDB {
  schema := &memdb.DBSchema{
    Tables: map[string]*memdb.TableSchema{

    },
  }
  db, err := memdb.NewMemDB(schema)
  if err != nil {
    panic(err)
  }
  return db
}
```

4. Crear un método para obtener con el patrón Singleton

```
func GetOrCreateSchema() *memdb.MemDB {
  ...
  return instance
}
```

Crear un Handler para llamar a la Base de Datos

1. Extendemos el router HTTP utilizando uno mas avanzado:

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

2. Crear un nuevo router

```
router := mux.NewRouter()
```

3. Asociar un path a una función y su método HTTP para todos los casos

```
router.HandleFunc("/path", handleFunc).Methods("GET")
```

Asociando los handlers a los métodos del CRUD

1. Obtener el schema para cada método, `true` para la parte transaccional

```
  database := db.GetOrCreateSchema()
  tx := database.Txn(true)
```

2. Para insertar un registro se utiliza el Insert

```
tx.Insert("table", &model)
tx.Commit()
```

3. Para obtener un registro

```
aw, err := tx.First("table", "id", id)
```

Creando un Binario Ejecutable

1. Utilizar el `go build` para generar el binario

```
$ go build -ldflags="-s -w" -o main .
```

2. Ejecutar el binario con el siguiente comando

```
$ ./main
```

Contenerizando la aplicación en Docker

1. Crear el archivo Dockerfile

```
FROM golang:1.15.6-alpine AS builder

# Set necessary environmet variables needed for our image
# GOOS=windows|linux|darwin
ENV GO111MODULE=on \
    CGO_ENABLED=0 \
    GOOS=darwin \
```

```
    GOARCH=amd64

# Move to working directory /build
WORKDIR /build

# Copy and download dependency using go mod
COPY go.mod .
COPY go.sum .
RUN go mod download

# Copy the code into the container
COPY . .

# Build the application
RUN go build -ldflags="-s -w" -o main .

# Move to /dist directory as the place for resulting binary folder
WORKDIR /dist
# Copy binary from build to main folder
RUN cp /build/main .

# Build a small image
FROM scratch
# commented to use shell from previous image

COPY --from=builder /dist/main /

# Command to run
ENTRYPOINT ["/main"]
```