Go: All you ever wanted to know but didn't dare to ask

Details

1. Keynote (introduction to golang - short presentation)
2. First steps with go - Jeshta Bhoyedhur
3. Building web APIs - Nadim Bundhoo
4. Going serverless - Yusuf Satar
5. Devops using go - Jules Giovanni

Special Guest: **Natalie Pistunovich**, GDE for Go

Go Installation

1. Download Go binary, unzip and move to appropriate folder.

```
> wget https://dl.google.com/go/go1.13.5.linux-amd64.tar.gz

> sudo tar -xvf go1.13.5.linux-amd64.tar.gz
> sudo mv go $HOME/Apps
```

2. Set up the environment

```
> export GOROOT=$HOME/Apps/go
> export GOPATH=$HOME/Code/Go
> export PATH=$GOPATH/bin:$GOROOT/bin:$PATH
```

- All the above environment will be set for your current session only. To make it permanent add above commands in `~/.profile` file.
- Reload the profile: `$ source .profile`

3. Verify Installation

- Verify the version

```
> go version

go version go1.13.5 linux/amd64
```

- Verify the environment

```
> go env

GO111MODULE=""
GOARCH="amd64"
GOBIN=""
GOCACHE="/home/mushtaaq/.cache/go-build"
GOENV="/home/mushtaaq/.config/go/env"
GOEXE=""
GOFLAGS=""
GOHOSTARCH="amd64"
GOHOSTOS="linux"
GONOPROXY=""
GONOSUMDB=""
GOOS="linux"
GOPATH="/home/mushtaaq/Code/go"
GOPRIVATE=""
GOPROXY="https://proxy.golang.org,direct"
GOROOT="/home/mushtaaq/Apps/go"
GOSUMDB="sum.golang.org"
GOTMPDIR=""
...
```

Go tools are awesome

We'll be using Visual Studio Code. If you have never coded Go using this editor, it will automatically detect that you are coding in Go; hence will recommend you to install the `Go` extension. It may also recommend you to install `Go tools`

The standard `Go tools` are:

- go get
- go build / go install
- go test
- go env
- go list
- go fmt
- go vet
- go doc
- go mod

Hello World

```go
package main

import "fmt"

func main {
    fmt.Println("Hello world!")
}
```

Go basic types

```go
bool

string

int  int8  int16  int32  int64
uint uint8 uint16 uint32 uint64 uintptr

byte // alias for uint8

rune // alias for int32
     // represents a Unicode code point

float32 float64

complex64 complex128
```

Variables declared without an explicit initial value are given their **zero** value.

The zero value is:

- `0` for numeric types,
- `false` for the boolean type, and
- `""` (the empty string) for strings

Structs

A struct is a collection of fields.

```go
// Product is a model for products
type Product struct {
    ID       int    `json:"id"`
    Product  string `json:"product"`
    Image    string `json:"image"`
    Quantity int    `json:"quantity"`
}

var products []Product
```

Struct fields are accessed using a dot, for eaxample `cart[0].Product`

Struct values encode as **JSON** objects. Each exported struct field becomes a member of the object, using the field name as the object key, unless the field is omitted for one of the reasons given below.

The encoding of each struct field can be customized by the format string stored under the " `json` " key in the struct field's tag. The format string gives the name of the field, possibly followed by a comma-separated list of options. The name may be empty in order to specify options without overriding the default field name.

The " `omitempty` " option specifies that the field should be omitted from the encoding if the field has an empty value, defined as *false*, *0*, a *nil pointer*, a *nil interface* value, and any *empty* array, slice, map, or string.

As a special case, if the field tag is " `-` ", the field is always omitted. Note that a field with name "-" can still be generated using the tag "-,".

Seeding

Any time you declare an `init()` function, Go will load and run it prior to anything else in that package.

We'll use the `init()` function to initialise the `products` variable and seed it with some data.

```go
func init() {
    var id int

    id = len(products) + 1
    products = append(products, Product{
        ID:       id,
        Product:  "Product 1",
        Image:    "/images/products/1.jpg",
        Quantity: 1,
    })
    // ....
}
```

Register URL paths and handlers

We'll have endpoints to:

- **list** all products
- **get** a product (by id)
- **create** a product

- **update** a product
- **delete** a product

```
r := mux.NewRouter()

r.HandleFunc("/products", listProducts).Methods("GET")
r.HandleFunc("/product/{id}", getProduct).Methods("GET")
r.HandleFunc("product", createProduct).Methods("POST")
r.HandleFunc("/product/{id}", updateProduct).Methods("PUT")
r.HandleFunc("/product/{id}", deleteProduct).Methods("DELETE")

// ListenAndServe listens on the TCP network address and then calls Serve with handler
http.ListenAndServe(":8080", r)
```

We need to implement the handlers. From the docs, we have:

```
func (*Router) HandleFunc[Top]

func (r *Router) HandleFunc(path string, f func(http.ResponseWriter, *http.Request)) *
```

```
func listProducts(w http.ResponseWriter, r *http.Request)  {}
func getProduct(w http.ResponseWriter, r *http.Request)    {}
func createProduct(w http.ResponseWriter, r *http.Request) {}
func updateProduct(w http.ResponseWriter, r *http.Request) {}
func deleteProduct(w http.ResponseWriter, r *http.Request) {}
```

We'll have to implement these handlers.

Putting it all together

```
package main

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

    "github.com/gorilla/mux"
)

// Product is a model for products
type Product struct {
    ID       int    `json:"id"`
```

```go
	Product  string `json:"product"`
	Image    string `json:"image"`
	Quantity int    `json:"quantity"`
}

var products []Product

func init() {
	var id int

	id = len(products) + 1
	products = append(products, Product{
		ID:       id,
		Product:  "Product 1",
		Image:    "/images/products/1.jpg",
		Quantity: 1,
	})
	id = len(products) + 1
	products = append(products, Product{
		ID:       id,
		Product:  "Product 2",
		Image:    "/images/products/2.jpg",
		Quantity: 1,
	})
	id = len(products) + 1
	products = append(products, Product{
		ID:       id,
		Product:  "Product 3",
		Image:    "/images/products/3.jpg",
		Quantity: 1,
	})
	id = len(products) + 1
	products = append(products, Product{
		ID:       id,
		Product:  "Product 4",
		Image:    "/images/products/4.jpg",
		Quantity: 1,
	})
	id = len(products) + 1
	products = append(products, Product{
		ID:       id,
		Product:  "Product 5",
		Image:    "/images/products/5.jpg",
		Quantity: 1,
	})
}

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

	router.HandleFunc("/products", listProducts).Methods("GET")
	router.HandleFunc("/product", createProduct).Methods("POST")
	router.HandleFunc("/product/{id}", getProduct).Methods("GET")
	router.HandleFunc("/product/{id}", updateProduct).Methods("PUT")
	router.HandleFunc("/product/{id}", deleteProduct).Methods("DELETE")
```

```go
    fmt.Println("Listening on http://localhost:8080")
    if err := http.ListenAndServe(":8080", router); err != nil {
        log.Fatal(err)
    }
}

func listProducts(w http.ResponseWriter, r *http.Request) {
    w.Header().Set("Content-type", "application/json")
    json.NewEncoder(w).Encode(products)
}

func createProduct(w http.ResponseWriter, r *http.Request) {
    var p Product

    json.NewDecoder(r.Body).Decode(&p)

    p.ID = len(products) + 1
    products = append(products, p)

    w.Header().Set("Content-type", "application/json")
    json.NewEncoder(w).Encode(p)
}

func getProduct(w http.ResponseWriter, r *http.Request) {
    w.Header().Set("Content-type", "application/json")

    id := getID(r)
    for _, p := range products {
        if p.ID == id {
            json.NewEncoder(w).Encode(p)
            return
        }
    }
}

func updateProduct(w http.ResponseWriter, r *http.Request) {
    w.Header().Set("Content-type", "application/json")

    var p Product

    id := getID(r)
    json.NewDecoder(r.Body).Decode(&p)

    for i := range products {
        if products[i].ID == id {
            products[i].Product = p.Product
            products[i].Image = p.Image
            products[i].Quantity = p.Quantity

            json.NewEncoder(w).Encode(products[i])
            return
        }
    }
}
```

```go
func deleteProduct(w http.ResponseWriter, r *http.Request) {
    var prods []Product

    id := getID(r)
    for _, p := range products {
        if p.ID != id {
            prods = append(prods, p)
        }
    }

    products = prods
    w.Header().Set("Content-type", "application/json")
    json.NewEncoder(w).Encode(products)
}

func getID(r *http.Request) int {
    params := mux.Vars(r)
    i, _ := strconv.Atoi(params["id"])
    return i
}
```

Testing

```
# List all products
GET http://localhost:8080/products

# Get a product by ID
GET http://localhost:8080/product/5

// Create a product
POST http://localhost:8080/product
Content-Type: application/json

{
    "product": "Product X",
    "image": "/images/products/x.jpg",
    "quantity": 1
}

// Update a product
PUT http://localhost:8080/product/5
Content-Type: application/json

{
    "product": "Product 12",
    "quantity": 1
}

# Delete a product
DELETE http://localhost:8080/product/5
```