GO tutorial

Nguyen Minh Duc
Bui Tan Dat

Agenda

1. Introduction
2. Basic GO
   - Hello world
   - Functions
   - Variables
   - Convert type
   - Loop
   - Defer
   - Array
   - Slice
   - Range
   - Method
   - Interface
3. Exercises
   - CRUD with mux and SQL
   - CRUD with gin and Mongo
   - CRUD with echo and gorm
4. Real project samples
5. GO service generator (low code)

1. Introduction

- **History**
  - Project starts at Google in 2007 (by Griesemer, Pike, Thompson)
  - Open source release in November 2009
  - More than 250 contributors join the project
  - Version 1.0 release in March 2012

- **Go is a new, general-purpose programming language**
  - Compiled
  - Statically typed
  - Concurrent
  - Simple
  - Productive

2. Basic GO

- Every Go program is made up of packages
- Start running in package main
  - Run here https://go.dev/tour/welcome/1
  - In this sample, use “fmt” package to print a string to console

```
package main

import "fmt"

func main() {
    fmt.Println( a...: "Hello, world")
}
```

2. Function

- **Support to return multiple values**
  - Exercise 1: write a function, to add 2 integer numbers
    - Run here https://go.dev/tour/basics/5
  - Exercise 2: write a function, to swap 2 strings
    - Run here https://go.dev/tour/basics/7

```go
package main

import "fmt"

func add(x, y int) int {
    return x + y
}

func main() {
    fmt.Println(add( x: 42,  y: 13))
}
```

```go
package main

import "fmt"

func swap(x, y string) (string, string) {
    return y, x
}

func main() {
    a, b := swap( x: "hello",  y: "world")
    fmt.Println(a, b)
}
```

2. Variables

- **Declare variables**
  - Exercise 1: use “var” to declare 3 variables
    - Run here https://go.dev/tour/basics/8
  - Exercise 2: use short variables declarations
    - Run here https://go.dev/tour/basics/10

```go
package main

import "fmt"

var c, python, java bool

func main() {
    var i int
    fmt.Println(i, c, python, java)
}
```

```go
package main

import "fmt"

func main() {
    var i, j int = 1, 2
    k := 3
    c, python, java := true, false, "no!"

    fmt.Println(i, j, k, c, python, java)
}
```

2. Convert type

- **Use expression T(v) to convert some number**
  - Run here https://go.dev/tour/basics/13

```
package main

import (
    "fmt"
    "math"
)

func main() {
    var x, y int = 3, 4
    var f float64 = math.Sqrt(float64(x*x + y*y))
    var z uint = uint(f)
    fmt.Println(x, y, z)
}
```

2. Loop

- **Only one loop construct, the for loop**
  - **Exercise 1: total from 1 to 10**
    - Run here https://go.dev/tour/flowcontrol/1
  - **Exercise 2: total from 1 to 10, with "while" loop logic**
    - Run here https://go.dev/tour/flowcontrol/3

```go
package main

import "fmt"

func main() {
    sum := 0
    for i := 0; i < 10; i++ {
        sum += i
    }
    fmt.Println(sum)
}
```

```go
package main

import "fmt"

func main() {
    sum := 1
    for sum < 1000 {
        sum += sum
    }
    fmt.Println(sum)
}
```

2. Defer

- **A defer statement defers the execution of a function until the surrounding function return**
  - **Exercise 1: print “hello world” with defer**
    - **Run here https://go.dev/tour/flowcontrol/12**
    - **Result**
      - hello
      - world

```go
package main

import "fmt"

func main() {
    defer fmt.Println(a...: "world")

    fmt.Println(a...: "hello")
}
```

2. Pointer

- **Struct fields can be accessed through a struct pointer**
  - **Exercise 1: access a struct with a struct pointer**
    - Run here https://go.dev/tour/moretypes/4

```go
package main

import "fmt"

type Vertex struct {
    X int
    Y int
}

func main() {
    v := Vertex{X: 1, Y: 2}
    p := &v
    p.X = 1e9
    fmt.Println(v)
}
```

2. Array

- **Arrays cannot be resized**
  - **Exercise 1: create arrays with string & int**
    - Run here https://go.dev/tour/moretypes/6

```go
package main

import "fmt"

func main() {
    var a [2]string
    a[0] = "Hello"
    a[1] = "World"
    fmt.Println(a[0], a[1])
    fmt.Println(a)

    primes := [6]int{2, 3, 5, 7, 11, 13}
    fmt.Println(primes)
}
```

2. Slice

- A dynamically-sized
  - Exercise 1: create a slice from an array
    - Run here https://go.dev/tour/moretypes/7
  - Exercise 2: create a slice with make function
    - Run here https://go.dev/tour/moretypes/13

```go
package main

import "fmt"

func main() {
    primes := [6]int{2, 3, 5, 7, 11, 13}

    var s []int = primes[1:4]
    fmt.Println(s, primes)
}
```

```go
package main
import "fmt"
func main() {
    a := make([]int, 5)
    printSlice( s: "a", a)
    b := make([]int, 0, 5)
    printSlice( s: "b", b)
    c := b[:2]
    printSlice( s: "c", c)
    d := c[2:5]
    printSlice( s: "d", d)
}
func printSlice(s string, x []int) {
    fmt.Printf( format: "%s len=%d cap=%d %v\n", s, len(x), cap(x), x)
}
```

2. Range

- **The range form of the for loop iterates over a slice or map**
  - **Exercise 1: loop a slice and print 2 powers**
    - Run here https://go.dev/tour/moretypes/16

```
package main

import "fmt"

var pow = []int{1, 2, 4, 8, 16, 32, 64, 128}

func main() {
    for i, v := range pow {
        fmt.Printf( format: "2**%d = %d\n", i, v)
    }
}
```

2. Method

- **Method is a function of struct**
  - Exercise 1: Create a method, with struct receiver
    - Run here https://go.dev/tour/methods/1

```go
package main

import (
    "fmt"
    "math"
)

func main() {
    v := Vertex{X: 3, Y: 4}
    fmt.Println(v.Abs())
}
```

```go
type Vertex struct {
    X, Y float64
}

func (v Vertex) Abs() float64 {
    return math.Sqrt(x: v.X*v.X + v.Y*v.Y)
}
```

2. Interface

- **An interface type is defined as a set of method signatures**
  - **Exercise 1: Create an interface, with abs float number**
    - Run here https://go.dev/tour/methods/1

```go
package main

import (
    "fmt"
    "math"
)

func main() {
    var a Abser
    f := MyFloat(-math.Sqrt2)
    v := Vertex{X: 3, Y: 4}

    a = f    // a MyFloat implements Abser
    a = &v // a *Vertex implements Abser

    // In the following line, v is a Vertex (not *Vertex)
    // and does NOT implement Abser.
    a = v

    fmt.Println(a.Abs())
}
```

```go
type Abser interface {
    Abs() float64
}

type MyFloat float64

func (f MyFloat) Abs() float64 {
    if f < 0 {
        return float64(-f)
    }
    return float64(f)
}

type Vertex struct {
    X, Y float64
}

func (v *Vertex) Abs() float64 {
    return math.Sqrt(X: v.X*v.X + v.Y*v.Y)
}
```

3. Exercise 1

- **Objectives**
  - Understand how to query data from RMS database, using "database/sql" package
  - Can insert, update, delete data
  - Use Mux to receive an http request, and return an http response

- Create a CRUD REST API with mux and My SQL, table users, with these fields fields id, username, email, phone, dateOfBirth, and methods GetAll, GetByID, Insert, Update, Delete
  - Refer to go-sql-tutorial

3. Exercise 2

- **Objectives**
  - Understand how to query data from Mongo
  - Can insert, update, delete data
  - Use gin to receive an http request, and return an http response

- Create a CRUD REST API with gin and MongoDB: collection user, with these fields id, username, email, phone, dateOfBirth, and methods: GetAll, GetByID, Insert, Update, Delete
  - Refer to go-mongo-tutorial and go-gin-sql-tutorial

3. Exercise 3

- **Objectives**
  - Understand how to query data from RMS database, using gorm
  - Can insert, update, delete data
  - Use echo to receive an http request, and return an http response

- Create a CRUD REST API with echo and gorm with My SQL: table users, with these fields id, username, email, phone, dateOfBirth, and methods: GetAll, GetByID, Insert, Update, Delete
  - Refer to gorm-tutorial and go-echo-sql-tutorial

4. Real project samples - overview

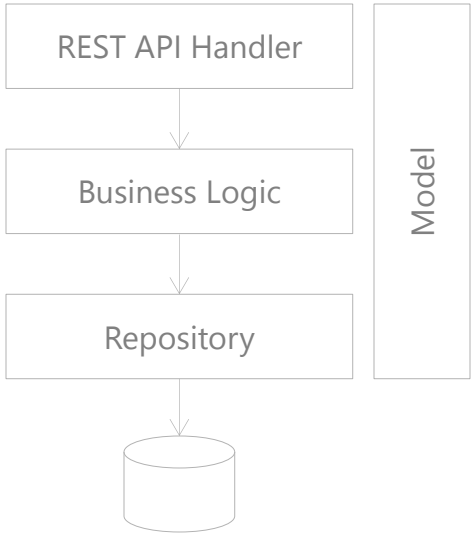
- **Layer Architecture Samples**
  - https://github.com/source-code-template/go-sql-layer-architecture-sample
  - https://github.com/source-code-template/go-mongo-layer-architecture-sample
- **Modular Samples**
  - https://github.com/source-code-template/go-sql-modular-sample
  - https://github.com/source-code-template/go-mongo-modular-sample
- **Message Queue Samples**
  - https://github.com/project-samples/go-subscription
  - https://github.com/project-samples/go-batch-subscription
- **go-admin**
  - https://github.com/project-samples/go-admin

4. Real project samples - 1

- **Layer Architecture Samples**
  - https://github.com/source-code-template/go-sql-layer-architecture-sample
  - https://github.com/source-code-template/go-mongo-layer-architecture-sample

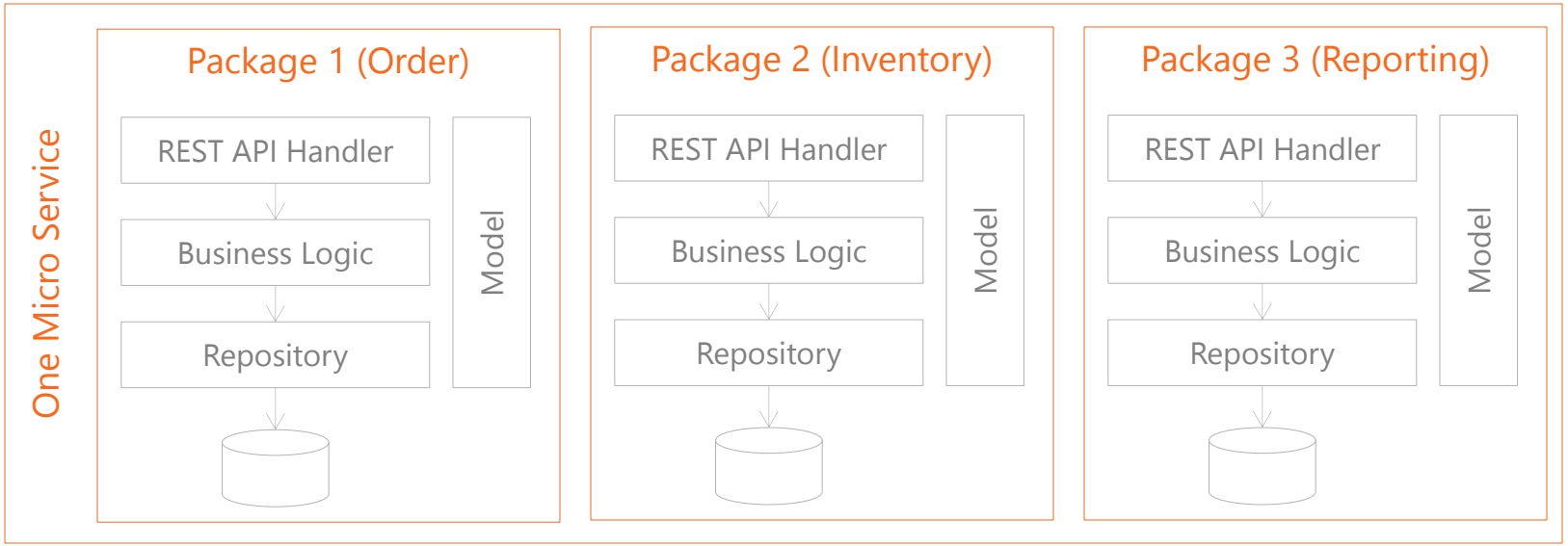
- **To build a REST API to support**
  - search, get by ID, create, update, delete
  - support "patch" method, using core-go/service
  - support "search" method, using core-go/search

- **Some standard features**
  - config: load config from yaml files
  - health check: to check health of SQL
  - logging: can use logrus or zap to log
  - tracing request and response at the middleware



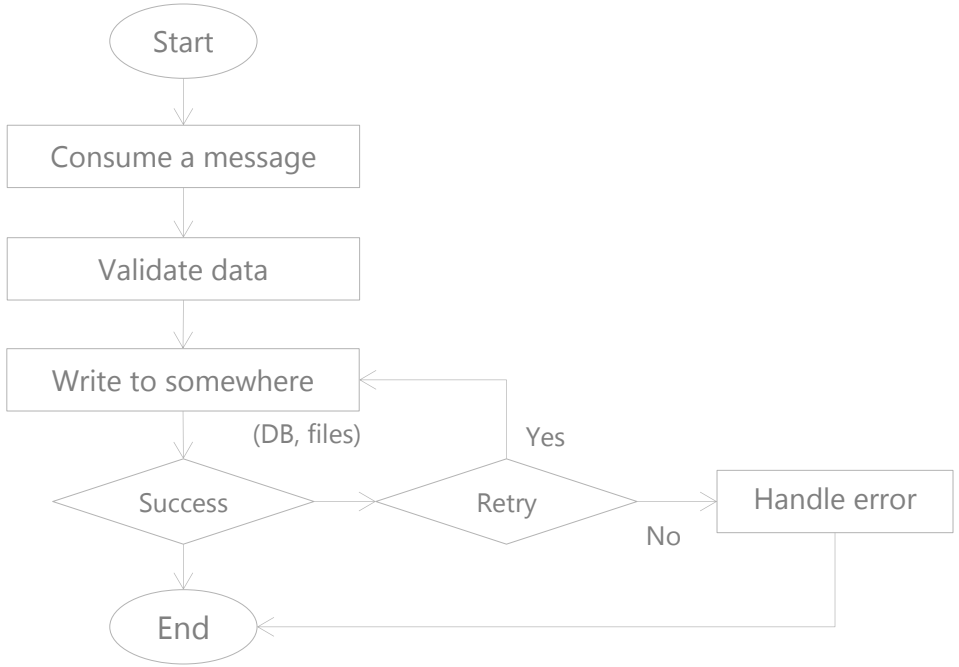
4. Real project samples - 2

- **Modular Samples**
  - https://github.com/source-code-template/go-sql-modular-sample
  - https://github.com/source-code-template/go-mongo-modular-sample

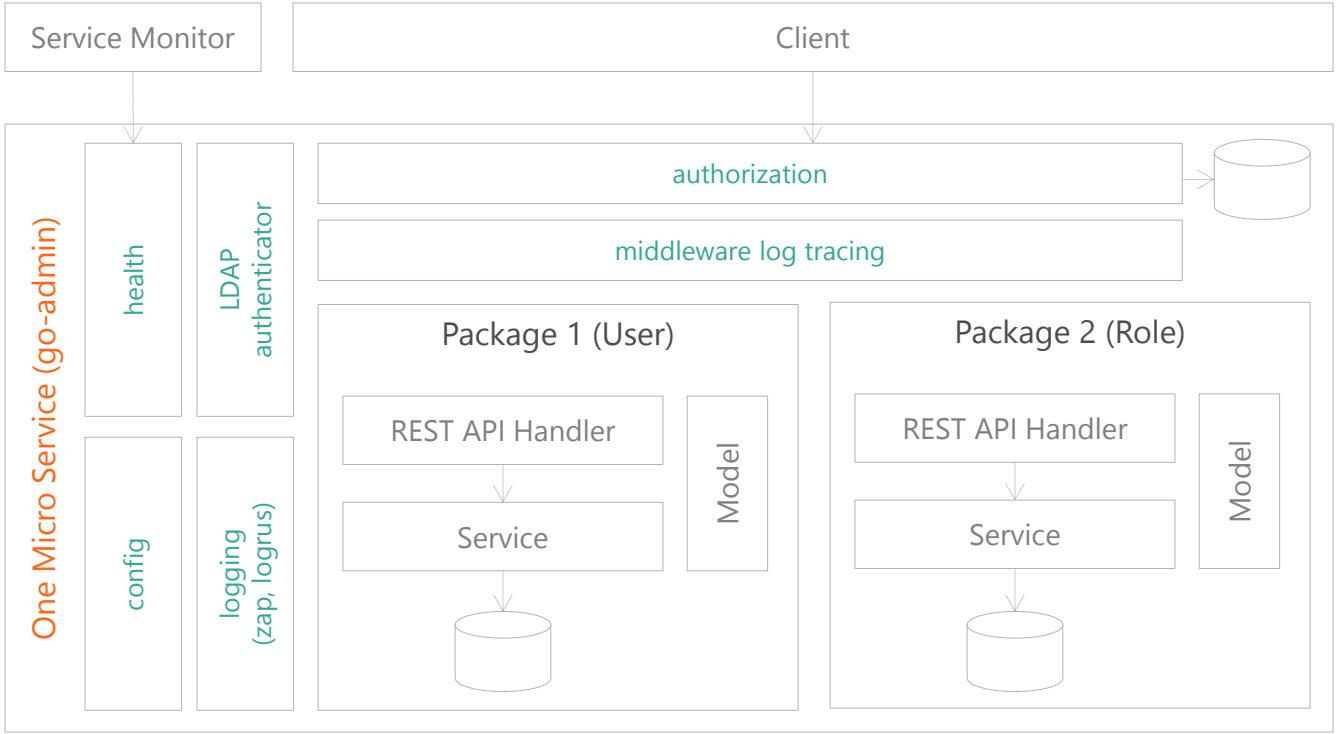


4. Real project samples - 3

- **Message Queue Samples**
  - **https://github.com/project-samples/go-subscription**
  - **https://github.com/project-samples/go-batch-subscription**
  - **Flow to consume a message**
  - **Support**
    - **Google Pub/Sub**
    - **Amazon SQS**
    - **RabbitMQ**
    - **Active MQ**
    - **IBM MQ**
    - **Kafka**
    - **NATS**



4. Real project samples – 4.1



4. Real project samples – 4.2

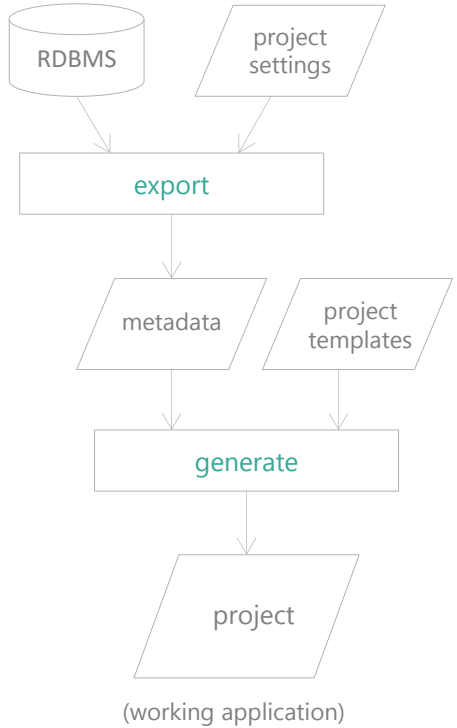
- **go-admin**
  - https://github.com/project-samples/go-admin

- **User and role management, with these features**
  - Authentication
    - Log in by LDAP
    - After logged in, get all privileges based on roles of that user
  - Security: Separate the "read" and "write" permissions for 1 role, using bitwise. For example
    - 001 (1 in decimal) is "read" permission
    - 010 (2 in decimal) is "write" permission
    - 100 (4 in decimal) is "delete" permission
    - "read" and "write" permission will be "001 | 010 = 011" (011 is 3 in decimal)

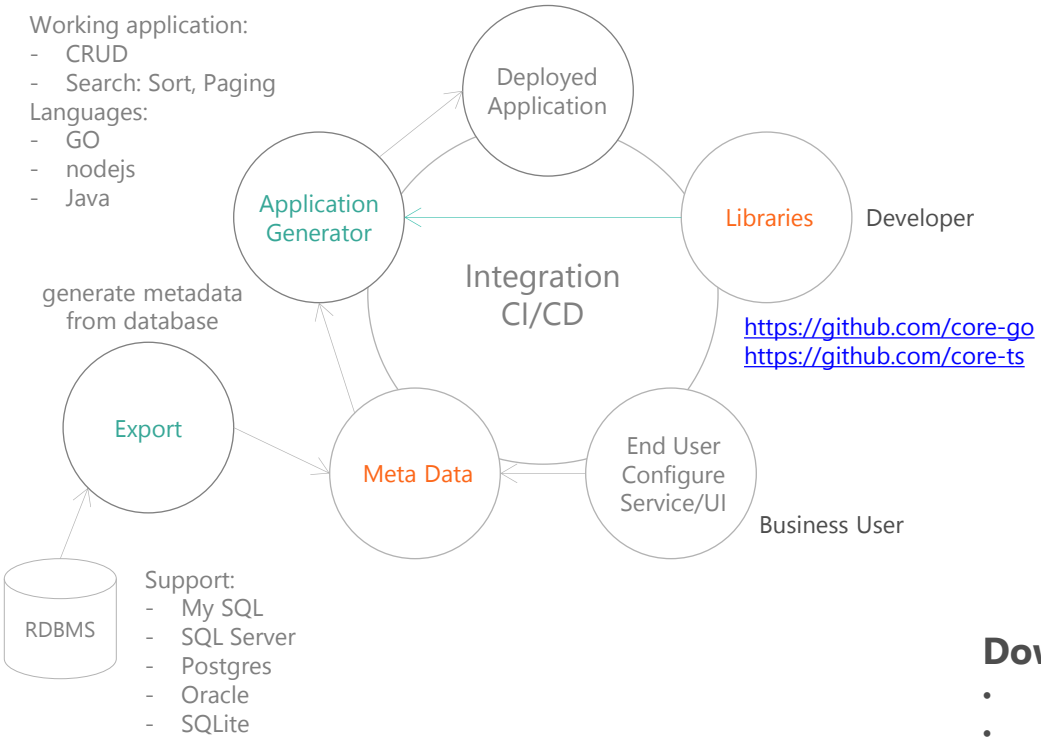
- **Some standard features**
  - config: load config from yaml files
  - health check: to check health of SQL
  - logging: can use logrus or zap to log, support to switch between logrus or zap
  - tracing request and response at the middleware

5. GO project generator - Components

- **Command line**
  - **export**
    - input: database, project settings
    - output: metadata
  - **generate**
    - input: metadata, project templates
    - output: project (working application)
- **GUI**
  - **generator**
    - GUI, include “export” and “generate”
- **Download**
    - https://github.com/lowcode-tech/windows



5. GO project generator – Business View



Download

- https://github.com/lowcode-tech/windows
- https://github.com/lowcode-tech/mac
- https://github.com/lowcode-tech/linux

5. GO project generator – Output Samples

- **Download**
  - https://github.com/lowcode-tech/windows
  - https://github.com/lowcode-tech/mac
  - https://github.com/lowcode-tech/linux
- **Output Samples**
  - GO Layer Architecture Sample
    - https://github.com/source-code-template/mongo-layer-architecture-sample
    - https://github.com/source-code-template/go-sql-layer-architecture-sample
  - GO Modular Sample
    - https://github.com/source-code-template/go-mongo-modular-sample
    - https://github.com/source-code-template/go-sql-modular-sample
  - nodejs Layer Architecture Sample
    - https://github.com/source-code-template/mongo-layer-architecture-sample
    - https://github.com/source-code-template/sql-layer-architecture-sample
  - nodejs Modular Sample
    - https://github.com/source-code-template/mongo-layer-architecture-sample
    - https://github.com/source-code-template/sql-layer-architecture-sample
  - nodejs Simple Modular Sample
    - https://github.com/source-code-template/mongo-simple-modular-sample
    - https://github.com/source-code-template/sql-simple-modular-sample

Questions and Anwers

Thank you

Refer to https://github.com/go-tutorials/overview