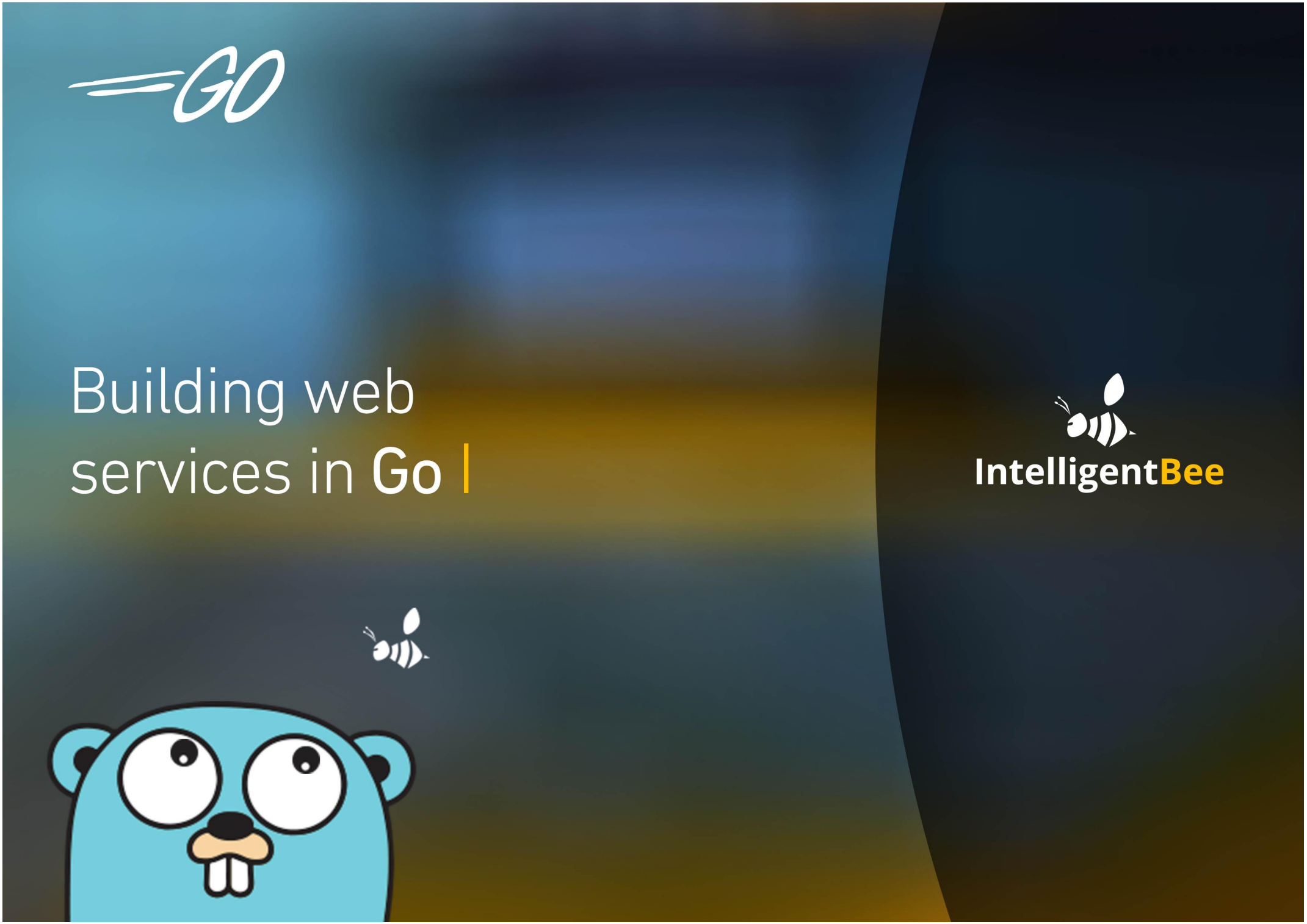
GO
Building web
services in Go |
IntelligentBee

Summary

- We can write HTTP endpoints
- How we can read & write JSON
- We can respond with an error
- We can log events in our app
- We can extract duplicate controller code into Middlewares
- We can display HTML pages using templates
- We can save and retrieve state into a database

1.0 Web services

A **web service** is any piece of software that makes itself available over the internet and uses a standardized messaging system.

Let's see a how we can make a Web service in Go:

1.1 Hello, world!

```
package main

import (
    "fmt"
    "net/http"
)

func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hi there, I love %s!", r.URL.Path[1:])
}

func main() {
    http.HandleFunc("/", handler)
    http.ListenAndServe(":8080", nil)
}
```

1.2 Routing

Processing HTTP requests with Go is primarily about two things: ServeMuxes and Handlers.

• A ServeMux is essentially a HTTP request router (or multiplexor).
It compares incoming requests against a list of predefined URL paths, and calls the associated handler for the path whenever a match is found.

• Handlers are responsible for reading the client request, processing data and writing response headers and bodies back to the client.

1.2 Routing

Native Multiplexer: `net/http.ServeMux`

- Simple but fast
- No route patterns

Most used router: `github.com/gorilla/mux`

- Lots of features and patterns for routing
- Easy to use
- Consumes a lot of resources

Fastest & popular router: `github.com/julienschmidt/httprouter`

- Simple routing with variables
- Scales better than ServeMux

1.2 Simple Example with Httprouter

```
package main

import (
    "fmt"
    "net/http"
    "log"

     "github.com/julienschmidt/httprouter"
)

func Index(w http.ResponseWriter, r *http.Request, _ httprouter.Params) {
    fmt.Fprint(w, "Welcome!\n")
}

func Hello(w http.ResponseWriter, r *http.Request, ps httprouter.Params) {
    fmt.Fprintf(w, "hello, %s!\n", ps.ByName("name"))
}

func main() {
    router := httprouter.New()
    router.GET("/", Index)
    router.GET("/hello/:name", Hello)

    log.Fatal(http.ListenAndServe(":8080", router))
}
```

2.0 JSON encoding & field tags:

```
import "encoding/json"

type StructA struct{
      A string    `json:"ab"`
      B int       `json:"b"`
      C string    `json:"c,omitempty"`
}

func main() {
      arr, _ := json.Marshal([]string{“apple”, “lettuce”})
      fmt.Println(string(arr)) // [“apple”,“lettuce”]

      mapA := map[string]int{"apple": 5, "lettuce": 7}
      mapB, _ := json.Marshal(mapA)
      fmt.Println(string(mapB)) //  {"apple":5,"lettuce":7}

      structA := StructA{A: "test", B: 5}
      structB, _ := json.Marshal(structA)
      fmt.Println(string(structB)) //  {"ab":"test,"b":5}
}
```

2.1 JSON decoding:

```
import (
	"encoding/json"
	"fmt"
)

type StructA struct {
	A string `json:"ab"`
	B int    `json:"b"`
	C string `json:"c,omitempty"`
}
func main() {
	arr := []string{}
	_ = json.Unmarshal([]byte(`["gopher","con"]`), &arr)
	fmt.Println(arr) // [gopher con]

	mapA := map[string]int{}
	_ = json.Unmarshal([]byte(`{"apple": 5, "lettuce": 7}`), &mapA)
	fmt.Println(mapA) //  map[apple:5 lettuce:7]

	structA := StructA{}
	_ = json.Unmarshal([]byte(`{"ab":"test","b":5}`), &structA)
	fmt.Println(structA) //  {test 5 }
}
```

2.2 Responding with JSON

```
import (
	"encoding/json"
	"net/http"
	"log"

	"github.com/julienschmidt/httprouter"
)
func function(w http.ResponseWriter, req *http.Request, ps httprouter.Params) {
	// read json from request body
	body := req.Body
	dec := json.NewDecoder(&body)
	var v map[string]interface{}
	if err := dec.Decode(&v); err != nil {
		log.Println(err)
		return
	}

	// respond with json
	w.Header().Set("Content-Type", "application/json")
	w.WriteHeader(http.StatusOK)
	enc := json.NewEncoder(w)
	if err := enc.Encode(v); err != nil {
		log.Println(err)
		return
	}
}
```

2.3 Responding with Error

```
import (
	"encoding/json"
	"errors"
	"log"
	"net/http"

	"github.com/julienschmidt/httprouter"
)

func function(w http.ResponseWriter, req *http.Request, ps httprouter.Params) {
	err := errors.New("Error description") // generate an error

	enc := json.NewEncoder(w)
	if err != nil {
		w.Header().Set("Content-Type", "application/json")
		w.WriteHeader(http.StatusInternalServerError)
		err := enc.Encode(map[string]string{
			"error": err.Error(),
		})

		return
	}
}
```

3.0 Logging

Golang provides you with a native logging library simply called “log”. This logger is perfectly suited to track simple behaviors such as adding a timestamp before an error message by using the available flags. But it doesn’t have leveled logs, you have to manually add prefixes like debug, info, warn, and error.

```
err := somethingHard()
if err != nil {
     log.Printf("oops, something was too hard: %s\n", err.Error())
     return err
}
```

Logging example

```
2017/09/22 09:06:33 Log message
2017/09/22 09:06:33 Panic oops, something was too hard: Error description
2017/09/22 09:06:33 Recovered from panic
```

3.1 Logging & Errors

```
defer func() {
     if r := recover(); r != nil {
          log.Println("Recovered", r)
     }
}()

err := somethingVeryHard()
if err != nil {
     log.Panicf("oops, something was too hard: %s\n", err.Error())
     return err
}

err := somethingSuperHard()
if err != nil {
     log.Fatalf("oops, something was too hard: %s\n", err.Error())
     return err
}
```

4.0 Middlewares

Before processing the request, we will often need to

- log the request,
- convert app errors into HTTP 500 errors,
- authenticate users, etc.

And we need to do most of these things for each handler or maybe for only some handlers.

4.1 Middlewares

Negroni is BYOR (Bring your own Router) so we can use it with **`httprouter`**.

```
router := httprouter.New()
router.GET("/", Index)
router.GET("/hello/:name", Hello)

n := negroni.Classic() // Includes some default middlewares
n.UseHandler(router)

log.Fatal(http.ListenAndServe(":8080", n))
```

4.1 Middlewares

negroni.Classic() provides some default middleware that is useful for most applications:

- negroni.Recovery - Panic Recovery Middleware.
- negroni.Logger - Request/Response Logger Middleware.
- negroni.Static - Static File serving under the "public" directory.

```
func MyMiddleware(rw http.ResponseWriter, r *http.Request, next http.HandlerFunc) {
     // do some stuff before
     next(rw, r)
     // do some stuff after
}

...
n := negroni.New()
n.Use(negroni.HandlerFunc(MyMiddleware))
...
```

4.2 Route specific Middlewares

```
router := httprouter.New()
router.GET(“/login”, loginHandler)

// add middleware for a specific route
nIndex := negroni.New()
nIndex.Use(negroni.handleFunc(auth))
nIndex.UseHandleFunc(index)
router.Handler(“GET”, “/”, nIndex)

// add middleware for a specific route & its params
server := negroni.Classic()
server.UseHandler(router)
server.Run(“:8080")
```

5.0 HTML Templates

Serving HTML is an important job for some web applications. Rendering HTML templates is almost as easy as rendering JSON using the 'html/template' package from the standard library.

```
templates/index.html:
<html>
    <h1>{{ .Title }}</h1>
    <h3>by {{ .Author }}</h3>
</html>
```

5.0 HTML Templates

```
import (
    "html/template"
    "net/http"
)

type Book struct {
    Title string
    Author string
}

func main() {
    http.HandleFunc("/", ShowBooks)
    http.ListenAndServe(":8080", nil)
}
```

5.0 HTML Templates

```
func ShowBooks(w http.ResponseWriter, r *http.Request) {
      book := Book{
          "Building Web Apps with Go",
          "Jeremy Saenz"
      }
      indexTemplate := "templates/index.html"
      tmpl, err := template.ParseFiles(indexTemplate)
      if err != nil {
          http.Error(w, err.Error(), http.StatusInternalServerError)
          return
      }
      if err := tmpl.Execute(w, book); err != nil {
          http.Error(w, err.Error(), http.StatusInternalServerError)
      }
}
```

6.0 Database

The "database/sql" package is Go's lightweight standard interface for relational databases, with support for backend drivers, prepared statements and transactions.

In order to use the sql package we need to include a specific database driver:

- MySQL: `https://github.com/go-sql-driver/mysql/`
- Postgres (pure Go): `https://github.com/jackc/pgx`
- SQLite: `https://github.com/mattn/go-sqlite3`
- Oracle: `https://github.com/mattn/go-oci8`

6.1 Database examples

```
import (
    "database/sql"
    "log"

    _ "github.com/go-sql-driver/mysql"
)
```

Create a database connection:

```
db, err := sql.Open("mysql", "root:@tcp(:3306)/test")
if err != nil {
    log.Fatal(err)
}
defer db.Close()
```

6.1 Database examples

Create a table:

```
_, err = db.Exec("CREATE TABLE IF NOT EXISTS test.hello(id INT(6) UNSIGNED AUTO_INCREMENT
PRIMARY KEY, world varchar(50))")
if err != nil {
    log.Fatal(err)
}
```

Insert data:

```
res, err := db.Exec("INSERT INTO test.hello(world) VALUES('hello world!')")
if err != nil {
    log.Fatal(err)
}
```

6.1 Database examples

Select data:

```
rows, err := db.Query("SELECT * FROM test.hello")
if err != nil {
      log.Fatal(err)
}
for rows.Next() {
      var id int
      var world string
      if err := rows.Scan(&id, &world); err != nil {
            log.Fatal(err)
      }
      log.Printf("found row containing %d, %q", id, world)
}
rows.Close()
```

Example

```
2017/09/22 10:21:34 found row containing 1, "hello world!"
2017/09/22 10:21:34 found row containing 2, "here we GO"
2017/09/22 10:21:34 found row containing 3, "let’s start the challenge"
```

Thank you.

```
fmt.Println("questions?")
```