Go for Web Developers

Workshop Part II

22 Oct 2018

Tags: edmontongo, golang

Edmonton Go

@edmontongo

https://edmontongo.org

Workshop I Review

Created library that wraps the DarkSky weather API

- Darksky Signup https://darksky.net/dev/register
- HTTP GET request
- JSON decoding
- Testing

Workshop II Outline

Create a web API for our previously created library

- Using flags to configure app
- http.Handler and http.HandlerFunc
- Interfaces
- JSON encoding
- Refactoring
- Add tests for our API

Development Environment

Can use libraries such as Gin, to allow for livereloading

1. https://github.com/codegangsta/gin

```
go get github.com/codegangsta/gin
```

Hello World

- https://golang.org/dl/
- https://golang.org/doc/install

```
package main

import "fmt"

func main() {
    fmt.Println("Hello, World")
}
```

Flags

- Go accepted way of configuration
- Allows for default values
- Provides documentation → `./myapp -help`

```
import "flag"

func main() {
    var someVal string
    flag.StringVar(&someVal, "someval", "", "some config v
    flag.Parse()
}
```

Secrets

Add the following code

```
func main() {
    var secret string
    flag.StringVar(&secret, "secret", "", "DarkSky API key
    port := flag.String("port", "3001", "Port service will
    flag.Parse()

    // if you want to use Gin
    if len(secret) == 0 {
        secret = os.Getenv("secret")
    }
    if len(secret) == 0 {
        flag.PrintDefaults()
        os.Exit(1)
    }
    fmt.Printf("Listening on :%s\n", *port)
}
```

Test it

```
go run . # should fail
```

```
go run . -secret=foo
```

Web API Basics

```
package main

import (
    "net/http"
)

func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *ht
        // handle request
    })
    http.Handle("/foo", someHandler)
    if err := http.ListenAndServe(":3000", nil); err != ni
        fmt.Printf("failed to start server: %v", err)
        os.Exit(1)
    }
}
```

Handlers

What is a Handler?

- It's a Go interface that requires the type to have a `ServeHTTP(w, r)` method

Why use it?

- Makes it easier to inject dependencies

Listen and Serve

```
type locationHandler struct { }

func (lh locationHandler) ServeHTTP(w http.ResponseWriter,
    fmt.Fprintf(w, "Hello World")
}

func main() {
    //...
    locationHandler := locationHandler{}
    http.Handle("/", locationHandler)
    if err := http.ListenAndServe(":"+*port, nil); err !=
        fmt.Printf("failed to start server: %v", err)
        os.Exit(1)
    }
}
```

```
go run . -secret=abc123 # secret=abc123 gin .
```

Weather (aka Location) Handler

http://localhost:3000?lat=53.5&lng=-113.3

Querystring params

```
q := r.URL.Query()
lat, err := strconv.ParseFloat(q.Get("lat"), 10)
if err != nil {
    http.Error(w, "Invalid lat querystring param", http.St
    return
}
```

Exercise

Parse out the longitude value as well, then test it via the browser

Get the weather

```
var darkskySvc *darksky.DarkSky
func main() {
    // ...
    darkskySvc = darksky.New(secret)
    // http.Handle("/", ...)
}
```

```
import "github.com/edmontongo/darksky"
func (lh locationHandler) ServeHTTP(w http.ResponseWriter,
    //...
    weather, err := darkskySvc.Forecast(darksky.Location{
    if err != nil {
        http.Error(w, "failed to fetch weather details", h
        return
    }
}
```

Content-Type

Add the code to the locationHandler

```
accept := r.Header.Get("Accept")
contentType := "application/json"
if strings.Contains(accept, "text/html") {
    contentType = "text/html"
}
w.Header().Add("Content-Type", contentType)

switch contentType {
case "text/html":
    // render html here
case "application/json":
    // return json
}
```

JSON Marshaling

```
import "encoding/json"
```

```
b, err := json.Marshal(weather)
if err != nil {
    http.Error(w, "Oops something went wrong", http.Status
    return
}
w.Write(b)
```

Html templates

```
t, err := template.New("app.html").
    Funcs(template.FuncMap{
        "toCelsius": func(tempF float64) string {
            degC := math.Round((tempF - 32) * 5 / 9)
            return fmt.Sprintf("%.0fÂ°C", degC)
        },
    }).
    ParseFiles(
        "templates/app.html",
        "views/location.html",
    )
if err != nil { return }  // handle these errors
if err = t.Execute(w, weather); err != nil { return }
```

Add the HTML rendering code

```
<!-- templates/app.html -->
<!DOCTYPE html>
<html lang="en">
<head><meta charset="UTF-8"></head>
<body>
{{block "content" .}}
    This text will render if no `content` block is defined
{{end}}
</body></html>
```

```
<!-- views/location.html -->
{{define "content"}}
<h1>{{.Currently.Summary}}</h1>
<div>{{.Currently.Temperature | toCelsius}}</div>
{{end}}
```

Refactor

1. Remove global darkskySvc
2. Cache templates (won't cover this, but checkout the code on github)

Use Interfaces

```
type forecaster interface {
    Forecast(darksky.Location) (*darksky.Weather, error)
}

type locationHandler struct {
    forecastSvc forecaster  // add this line to existing h
}

func newLocationHandler(f forecaster) locationHandler {
    return locationHandler{ forecastSvc: f }
}
```

```
// remove global service
darkskySvc := darksky.New(secret)
locationHandler := newLocationHandler(darkskySvc)
```

API Tests

Mocks

```
// main_test.go
type forecasterMock struct {
    weather *darksky.Weather
    err     error
}

func (f forecasterMock) Forecast(darksky.Location) (*darks
    return f.weather, f.err
}
```

Tests

```
func TestHTMLResponse(t *testing.T) {
    forecastMock := forecasterMock{
        weather: &darksky.Weather{
            Currently: darksky.Currently{
                Summary: "hot", Temperature: 90,
            },
        },
    }
    lh := newLocationHandler(forecastMock)

    r, _ := http.NewRequest("GET", "/?lat=53.5&lng=-113.3'
    r.Header.Add("Accept", "text/html")
    w := httptest.NewRecorder()
    lh.ServeHTTP(w, r)

    if w.Code != 200 { t.Fatalf("Unexpected status code: %
    body := w.Body.String()
    if !strings.Contains(body, "hot") { t.Errorf("it shou
}
```

Questions?