Tworzenie Aplikacji Internetowych

Laboratorium 11

Technologie po stronie serwera – Język Go – Serwer HTTP – HTML oraz JSON

Jednym z najprostszych sposobów uruchomienia serwera jest wywołanie funkcji **http.ListenAndServe** - pierwszym parametrem jest adres (można go pominąć) i port, drugi jest opcjonalny i będzie omówiony w kolejnych laboratoriach:

```
http.ListenAndServe(":8080", nil)
```

W momencie uruchomienia powyższej funkcji kod programu **zatrzymuje się** na tym poleceniu i rozpoczyna obsługę żądań kierowanych do serwera. Obsługę żądań należy wcześniej zaprogramować - jednym z najprostszych sposobów jest obsługa wzorców (*ang.* pattern). W momencie zgodności adresu strony z danym wzorcem następuje wywołanie konkretnej funkcji programu. Do podłączenia wzorców służy metoda **http.HandleFunc**. Przedstawia to poniższy przykład:

```
package main

import (
    "fmt"
    "net/http"
)

func indexFunc(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "<html><body>STRONA GŁÓWNA</body></html>")
}

func itemFunc(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "<html><body>STRONA ITEM<br>ADRES: ")
    fmt.Fprintf(w, r.RequestURI)
    fmt.Fprintf(w, "<br>METODA: ")
    fmt.Fprintf(w, r.Method)
    fmt.Fprintf(w, "</body></html>")
}

func main() {
    http.HandleFunc("/", indexFunc) // wzorzec /*
    http.HandleFunc("/item/", itemFunc) // wzorzec /item/*
    http.ListenAndServe("localhost:8080", nil)
}
```

W ramach przetestowania działania serwera należy uruchomić powyższy przykład i przetestować w przeglądarce adresy:

- localhost:8080/
- localhost:8080/item/
- localhost:8080/items
- localhost:8080/item/15

Wymuszenie zatrzymania serwera można uzyskać przez skrót w terminalu Ctrl + C

W powyższym przykładzie zawartość stron jest generowana "ręcznie". Istnieje jednak wiele innych możliwości przesyłania treści stron internetowych oraz plików, np.:

- Zwracanie statycznych plików:

```
http.ServeFile(w, r, "pages/strona.html")
```

- Zwracanie szablonów plików .html (w takich plikach można umieszczać treść dynamicznie, na podstawie danych przekazanych z języka Go – może być to dowolna struktura danych)

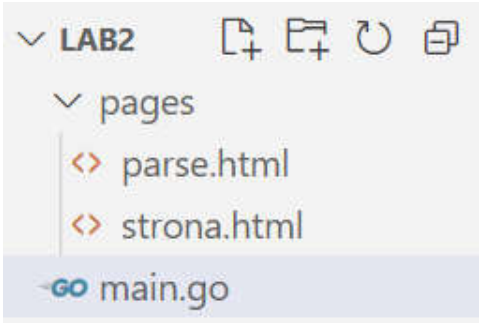
```
tmpl, _ := template.ParseFiles("pages/parse.html")
tmpl.Execute(w, [data])
```

W powyższym kodzie [data] oznacza opcjonalną strukturę danych przekazywanych do strony. Znak _ oznacza że drugi parameter zwracany przez ParseFiles nie będzie wykorzystywany.

Podmianę danych umożliwia specjalna składnia umieszczana na stronie .html – kod pomiędzy nawiasami **{{** oraz **}}**. Dostęp do pól struktury odbywa się przez zapis: .NazwaPola (pola struktury muszą być publiczne – pisane z dużych liter).

Do przetestowania powyższych funkcjonalności należy dodać do folderu roboczego folder **pages** oraz umieścić w nim pliki: strona.html (o dowolnej treści HTML) i parse.html o następującej zawartości:

```
<!DOCTYPE html>
<html>
    <head></head>
    <body>
        <table>
            <tr>
                <th>Imię</th>
                <th>Indeks</th>
            </tr>
            <tr>
                <td>{{ .Name }}</td>
                <td>{{ .Index }}</td>
            </tr>
        </table>
    </body>
</html>
```



Następnie należy dodać w funkcji main obsługę dwóch kolejnych wzorców (oczywiście przed rozpoczęciem startu serwera):

```
    http.HandleFunc("/strona/", stronaFunc)
    http.HandleFunc("/parse/", parseFunc)
```

Do kodu programu należy również dodać strukturę student (będzie zawierała pola .Name oraz .Index, które będą przekazywane do strony parse.html) oraz funkcje stronaFunc i parseFunc. Implementację tych elementów przedstawia następujący kod źródłowy:

```go
func stronaFunc(w http.ResponseWriter, r *http.Request) {
    // zwrócenie statycznej strony strona.html
    http.ServeFile(w, r, "pages/strona.html")
}

type student struct {
    Name  string
    Index int
}

func parseFunc(w http.ResponseWriter, r *http.Request) {
    // zwrócenie strony o dynamicznej zawartości
    tmpl, _ := template.ParseFiles("pages/parse.html")
    tmpl.Execute(w, &student{"Jan", 12345})
}
```

W celu sprawdzenia działania powyższej funkcjonalności należy przetestować adresy w przeglądarce:

- localhost:8080/strona/
- localhost:8080/parse/

Powyższe przykłady zwracają użytkownikowi strony internetowe. Serwer HTTP nie jest do tego ograniczony – może on zwracać pliki dowolnego typu. Najlepszym przykładem jest zwracanie plików z danymi w formacie **JSON**. Takie pliki można później wykorzystać przez każdą inną technologię obsługującą ten standard lub pozwalającą zaprogramować jego obsługę (np. JavaScript, C#, Rust).

W języku Go każdą strukturę danych można przekonwertować na format JSON, służy do tego polecenie Marshall z biblioteki json. Domyślnie parsowane są wszystkie publiczne pola struktury (pola nazwane z dużej litery) i posiadają one nazwy odpowiadające nazwom struktury. Nazwy te można jednak zmodyfikować (lub nawet usunąć konkretne pola) za pomocą dodatkowej składni. Całość pozwoli zrozumieć poniższy przykład:

```go
type rms struct {
    N1 float64 `json:"number1"` // ta właściwość zostanie zwrócona jako number1
    N2 float64 `json:"number2"` // ta właściwość zostanie zwrócona jako number2
    N3 float64 `json:"-"`       // ta właściwość nie zostanie zwrócona
}

func GenerateRandom() *rms { // zwraca powyższą strukturę z 3 losowymi liczbami
    return &rms{rand.Float64(), rand.Float64(), rand.Float64()}
}

func randomFunc(w http.ResponseWriter, r *http.Request) {
    w.Header().Set("Content-Type", "application/json") // nagłówek JSON
    numbers := GenerateRandom()                        // wygenerowanie liczb
    data, _ := json.Marshal(numbers)                   // konwersja na JSON
    w.Write(data)                                      // zwrócenie danych JSON
}

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

Uruchomić przykład i wyświetlić **w różnych przeglądarkach** stronę localhost:8080/random/

Zadanie 1

Utworzyć serwer HTTP, który pod adresem localhost:8080/test/ wyświetli z wykorzystaniem szablonów stronę internetową **dane.html** o następującej treści:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Lista studentów</title>
</head>
<body>
    <h3>Lista studentów: </h3>
    {{ . }}
</body>
</html>
```

Przetestować stronę przesyłając do niej puste dane (nil):

```
tmpl.Execute(w, nil)
```

Sprawdzić działanie strony w przeglądarce, podejrzeć kod źródłowy i jak został wstawiony pusty wskaźnik.

Następnie do strony dane.html zamiast wskaźnika nil przekazać tablicę obiektów typu Student. W tym celu należy utworzyć strukturę student zawierającą pola: Imie (string), Nazwisko (string), Indeks (int), Email (string) oraz utworzyć zmienną globalną zawierającą kolekcję kilku studentów:

```
var studenci []Student = []Student{
    {"Jan", "Kowalski", 12345, "test@test"},
    {"Marek", "Nowak", 30000, "to@tamto"},
    {"Anna", "Zdyb", 23232, "anna@zdyb"},
}
```

Sprawdzić działanie strony.

Template w języku Go oprócz wyświetlania obiektu na stronie posiadają bardzo rozbudowaną składnie, w której można sprawdzać warunki, wywoływać właściwości, wywoływać funkcję, tworzyć definicję i podszablony, oraz iterować kolekcję. Do przeiterowania kolekcji służy składnia:

```
{{ range [kolekcja] }}
  {{ . }}
{{ end }}
```

Kropka wewnątrz kolekcji oznacza tym razem obiekt kolekcji (w taki sposób działa zagnieżdżanie). Wykorzystując powyższy zapis można zmodyfikować wyświetlanie studentów {{ . }} na następującą składnię:

```
    <h3>Lista studentów: </h3>
    <ol>
    {{ range . }}
        <li>{{ .Imie }}, {{ .Indeks }}</li>
    {{ end }}
    </ol>
```

Cel zadania: zamiast powyższej listy numerowanej dane studentów wyświetlić w tabeli. Oprócz imienia i indeksu wyświetlić pozostałe pola (nazwisko oraz adres e-mail).

Zadanie 2

Rozszerzyć zadanie 1 tak, aby pod adresem /json/ zwracało tablicę ze zmiennej globalnej "studenci" w formacie JSON. Dodatkowo zmienić **formatowanie** struktury tak, aby struktura JSON nie zawierała pola Email.

Zadanie 3

Napisać obsługę /zad3/ w taki sposób, aby zwracana była w sposób losowy jedna z trzech różnych stron internetowych (liczby losowe: rand.Intn(doilu)).

Zadanie 4 (na plusa)

Samodzielnie sprawdzić jak rozszerzyć serwer HTTP o serwer plików, pozwalający na stronie wyświetlać m.in. pliki .png oraz .css pochodzące z tego samego serwera. Utworzyć stronę HTML, która wyświetli dowolny plik graficzny oraz załączy dowolny plik styli CSS udostępniany przez utworzony serwer plików.

Zadanie 5 (na dwa plusy)

Język Go pozwala również połączyć się z innymi serwerami, wysłać żądanie GET zwracające dane w formacie JSON, które można następnie przekonwertować na dane języka Go. Przykład działania pokazuje następna strona instrukcji.

Celem zadania jest rozszerzenie przykładu z następnej strony w taki sposób, aby otrzymane dane wyświetlać na stronie internetowej w taki sposób, aby były one wyświetlane dla IP podanego w nazwie strony, tzn. http://localhost:8080/pogoda/87.99.119.110 powinno wyświetlić dane dla tego IP (itd.).

```go
package main

import (
    "encoding/json"
    "fmt"
    "io"
    "log"
    "net/http"
    "time"
)

type jsonIPdata struct { // stuktura danych do której przekażemy dane JSON
    Query       string  `json:"query"`
    Status      string  `json:"status"`
    Country     string  `json:"country"`
    CountryCode string  `json:"countrycode"`
    Region      string  `json:"region"`
    RegionName  string  `json:"regionname"`
    City        string  `json:"city"`
    Zip         string  `json:"zip"`
    Lat         float64 `json:"lat"`
    Lon         float64 `json:"lon"`
}

func main() {
    client := http.Client{Timeout: time.Second}       // utworzenie klienta HTTP
    url := "http://ip-api.com/json/87.99.119.110"     // strona z API z danych
    r, e := http.NewRequest(http.MethodGet, url, nil) // żądanie HTTP GET
    if e != nil {
        log.Fatal(e)
    }
    res, err := client.Do(r) // wywołanie żądania HTTP GET
    if err != nil {
        log.Fatal(err)
    }
    body, err := io.ReadAll(res.Body) // odczytanie odpowiedzi z HTTP GET
    if err != nil {
        log.Fatal(err)
    }
    ipData := jsonIPdata{}              // utworzenie struktury na otrzymane dane
    err = json.Unmarshal(body, &ipData) // parsowanie danych JSON na strukturę
    if err != nil {
        log.Fatal(err)
    }
    fmt.Println(ipData) // wyświetlenie wyniku w terminalu
}
```