Restful server em Golang

O pacote “net/http” já permite criar servidores HTTP. O método “ListenAndServe” inicia um servidor HTTP com um determinado endereço e manipulador. O manipulador é a função que recebe o request e o response e lida com eles. Vejamos um exemplo de servidor do pacote net/http:

```
http.Handle("/foo", fooHandler)

http.HandleFunc("/bar", func(w http.ResponseWriter, r *http.Request) {
      fmt.Fprintf(w, "Hello, %q", html.EscapeString(r.URL.Path))
})

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

Passamos um objeto function para o método “HandleFunc”, que será associado ao path “/bar”. A função recebe uma instância de ResponseWriter e de Request e lida com eles.

Gorilla Mux

O pacote gorilla/mux implementa request handlers. Ele faz a “multiplexação” do request de acordo com vários parâmetros:

- URL host

- path
- path prefix
- schemes
- headers
- query variables
- HTTP methods
- Custom matchers (podemos escrever o algoritmo de casamento de requests)

Utilizar o Gorilla/Mux é muito simples:

1. Instale o pacote com: `go get -u github.com/gorilla/mux` O flag “-u” serve para atualizar as dependências.
2. As funções “HandleFunc” associam funções às rotas REST, dependendo do método HTTP.

Vejamos um exemplo aqui:

```
router := mux.NewRouter()
router.HandleFunc("/api/note/{id}", ReadNote).Methods("GET")
router.HandleFunc("/api/note", WriteNote).Methods("POST")
err := http.ListenAndServe(fmt.Sprintf(":%s", serverPort), router)
```

Como podem ver, podemos associar pedaços da URL, com métodos HTTP e redirecionar para as funções corretas. E podemos utilizar variáveis de querystring.

Pegando dados

Podemos obter dados do querystring ou do corpo do request (se for POST). Vejamos um exemplo obtendo variáveis de querystring (extrapath).

Request GET com variável extra path:

```
func ReadNote(w http.ResponseWriter, r *http.Request) {
	vars := mux.Vars(r)
	id := vars["id"]
	if data, err := backend.GetKey(id); err == nil {
		WriteResponse(200, data, w)
	} else {
		if err.Error() == "not found" {
			WriteResponse(404, "Note not found", w)
		} else {
			WriteResponse(500, "Error", w)
		}
	}
```

```
}
```

Neste exemplo pegamos a coleção “mux.Vars” que tem as variáveis passadas na URL e obtivemos a variável “id”.

Request POST com JSON no corpo do request:

```
func WriteNote(w http.ResponseWriter, r *http.Request) {
	decoder := json.NewDecoder(r.Body)
	defer r.Body.Close()
	var note db.Note
	if err := decoder.Decode(&note); err != nil {
		WriteResponse(http.StatusBadRequest,
map[string]string{"error": err.Error()}, w)
		return
	}
	uuidString, err := backend.SaveKey(note.Text, note.OneTime)
	if err != nil {
		WriteResponse(http.StatusBadRequest,
map[string]string{"error": "invalid request"}, w)
	} else {
		WriteResponse(200, map[string]string{"code": uuidString}, w)
	}

}
```

Este segundo exemplo acima é de um request POST com dados JSON no corpo. Podemos criar um decoder JSON (pacote: “encoding/json”) para decodificar o corpo do request. Como vamos abrir um stream para ler o corpo, precisamos postergar o close do corpo do request com o comando “defer”.

Gerando a resposta HTTP

A resposta HTTP dependerá muito do content-type que você vai gerar. Se for JSON, então podemos simplesmente gerar um dicionário e usar o método marshal do JSON para gerar o conteúdo:

```
payload, _ := json.Marshal(body)
w.Write(payload)
```

Geralmente, eu crio uma func como esta para lidar com a geração da resposta:

```
func WriteResponse(status int, body interface{}, w http.ResponseWriter) {
	w.WriteHeader(status)
```

```
	w.Header().Set("Content-Type", "application/json")
	payload, _ := json.Marshal(body)
	w.Write(payload)
}
```

E a invoco deste jeito:

```
WriteResponse(200, map[string]string{"code": uuidString}, w)
```

ou:

```
WriteResponse(http.StatusBadRequest, map[string]string{"error": "invalid
request"}, w)
```

Servidor completo

Aqui está o esqueleto de um servidor REST simples. Há outros pacotes, mas o principal está na func “main()”:

```
package main

import (
	"encoding/json"
	"fmt"
	"net/http"
	"os"

	"com.blocopad/blocopad_poc/internal/backend"
	"com.blocopad/blocopad_poc/internal/db"
	"github.com/gorilla/mux"
)

func WriteResponse(status int, body interface{}, w http.ResponseWriter) {
	w.WriteHeader(status)
	w.Header().Set("Content-Type", "application/json")
	payload, _ := json.Marshal(body)
	w.Write(payload)
}

// HTTP Handlers

func ReadNote(w http.ResponseWriter, r *http.Request) {
	vars := mux.Vars(r)
	id := vars["id"]
	if data, err := backend.GetKey(id); err == nil {
		WriteResponse(200, data, w)
```

```
	} else {
		if err.Error() == "not found" {
			WriteResponse(404, "Note not found", w)
		} else {
			WriteResponse(500, "Error", w)
		}
	}
}

func WriteNote(w http.ResponseWriter, r *http.Request) {
	decoder := json.NewDecoder(r.Body)
	defer r.Body.Close()
	var note db.Note
	if err := decoder.Decode(&note); err != nil {
		WriteResponse(http.StatusBadRequest,
map[string]string{"error": err.Error()}, w)
		return
	}
	uuidString, err := backend.SaveKey(note.Text, note.OneTime)
	if err != nil {
		WriteResponse(http.StatusBadRequest,
map[string]string{"error": "invalid request"}, w)
	} else {
		WriteResponse(200, map[string]string{"code": uuidString}, w)
	}

}

func main() {
	serverPort := "8080"
	if port, hasValue := os.LookupEnv("API_PORT"); hasValue {
		serverPort = port
	}
	databaseUrl := "localhost:6379"
	if dbUrl, hasValue := os.LookupEnv("API_DB_URL"); hasValue {
		databaseUrl = dbUrl
	}
	databasePassword := ""
	if dbPassword, hasValue := os.LookupEnv("API_DB_PASSWORD"); hasValue
{
		databasePassword = dbPassword
	}

	db.DatabaseUrl = databaseUrl
	db.DatabasePassword = databasePassword

	router := mux.NewRouter()
```

```
	router.HandleFunc("/api/note/{id}", ReadNote).Methods("GET")
	router.HandleFunc("/api/note", WriteNote).Methods("POST")
	err := http.ListenAndServe(fmt.Sprintf(":%s", serverPort), router)
	fmt.Println(err)

}
```