Nama : Taufik Hidayat
Program : Golang for Back End Programmer

Summary REST API dengan Golang

Golang adalah bahasa pemrograman yang sangat mudah dipahami, cepat dalam performance. Merupakan compiled language, memiliki garbage collection, dan yang paling penting concurrency. Kelebihan golang saat membuat REST API dibanding pada bahasa pemrograman lain adalaha karena golang memiliki built-in package untuk membangun REST API dengan mudah. REST adalah singkatan dari Representational State Transfer adalah sebuah mekanisme transfer secara modern untuk berkomunikasi dengan database dan server via HTTP.

Untuk menginstall golang, kita bisa download melalui https://go.dev. Pada Linux kita bisa menggunakan package manager dengan perintah sebagai berikut:

```
sudo pacman -S go # Arch Linux
sudo apt install golang # Debian dan Ubuntu
sudo dnf install golang # Fedora dan RHEL
brew install go # macOS
```

Untuk memastikan apakah go sudah terinstall atau belum, bisa dengan menjalankan perintah berikut:

```
go version
```

Apabila muncul output seperti berikut:

```
go version go1.20.2 linux/amd64
```

Itu berarti golang versi 1.20.2 sudah terinstall di PC kita.

Pada golang versi lama, kita harus menyimpan project kita di folder `$GOPATH`, namun pada golang versi baru yang sudah mendukung fitur module, kita bisa menyimpan project golang di mana saja. Entry point pada golang berada pada function `main()`. Untuk menghandle request pada REST API golang, kita menggunakan function yang bernama `HandleFunc()` yang berasal dari package `net/http`. Setelah menghandle dynamic request kita bisa meng-serve static file dengan function `FileServer()` yang berasal dari package `net/http`. Agar program REST API kita dapat melisten sebuah port, maka kita bisa menggunakan function `ListenAndServe()` yang berasal dari package `net/http`. Untuk mencompile program REST ini, kita bisa menjalankan perintah `go build`. Untuk menjalankan program go tersebut secara langsung, kita bisa menggunakan perintah `go run .`

Untuk menginstall library, kita bisa menggunakan perintah `go get`, sebagai contoh untuk menginstall library gorilla mux kita bisa menjalankan command tersebut:

```
go get github.com/gorilla/mux
```

Model adalah representasi data yang akan dimasukkan ke dalam database. Untuk membuat sebuah model pada golang, kita bisa menggunakan struct. Contohnya sebagai berikut:

```
type Roll struct {
	ID          string `json:"id"`
	Name        string `json:"name"`
	Description string `json:"description"`
	Ingredient  string `json:"ingredient"`
}
```

Untuk menyimpan data tanpa database, kita bisa menggunakan global variable bertipe data slice untuk menyimpan data pada memory.

```
var rolls []Roll
```

Untuk menangani sebuah request pada sebuah route, kita bisa menggunakan function handler seperti berikut:

```
func getRolls(w http.ResponseWriter, r *http.Request) {
	w.Header().Set("Content-Type", "application/json")
	json.NewEncoder(w).Encode(rolls)
}
```

Karena data rolls masih kosong, maka kita harus meng-generate data dummy terlebih dahulu.

```
rolls = append(rolls,
	Roll{
		ID:          "1",
		Name:        "Salmon Roll",
		Description: "crab stick, tamago sushi with salmon and cheese",
		Ingredient:  "Salmon, Nori, Soy Sauce, Rice",
	},
	Roll{
		ID:          "2",
		Name:        "California Roll",
		Description: "crab stick, tamago sushi with california and cheese",
		Ingredient:  "California, Nori, Soy Sauce, Rice",
	},
)
```

Untuk mendapatkan param dari URL, kita bisa menggunakan cara ini:

```
func getRoll(w http.ResponseWriter, r *http.Request) {
	w.Header().Set("Content-Type", "application/json")
	params := mux.Vars(r)
	for _, item := range rolls {
		if item.ID == params["id"] {
			json.NewEncoder(w).Encode(item)
			return
		}
	}
```

```
}
```

Untuk membuat sushi baru, kita bisa menggunakan cara seperti ini:

```
func createRoll(w http.ResponseWriter, r *http.Request) {
      w.Header().Set("Content-Type", "application/json")
      var newRoll Roll
      json.NewDecoder(r.body).Decode(&newRoll)
      newRoll.ID = strconv.Itoa(len(rolls) + 1)
      rolls = append(rolls, newRoll)
      json.NewEncoder(w).Encoder(newRoll)
}
```

Untuk meng-update sushi, kita bisa menggunakan cara seperti ini:

```
func updateRoll(w http.ResponseWriter, r *http.Request) {
      w.Header().Set("Content-Type", "application/json")
      params := mux.Vars(r)
      for i, item := range rolls {
            if item.ID == params["id"] {
                  rolls = append(rolls[:i], rolls[i+1]...)
                  var newRoll Roll
                  json.NewDecoder(r.body).Decode(&newRoll)
                  newRoll.ID = params["id"]
                  rolls = append(rolls, newRoll)
                  json.NewEncoder(w).Encode(newRoll)
                  return
            }
      }
}
```

Untuk meng-delete sushi kita bisa menggunakan cara seperti ini:

```
func deleteRoll(w http.ResponseWriter, r *http.Request) {
      w.Header().Set("Content-Type", "application/json")
      params := mux.Vars(r)
      for i, item := range rolls {
            if item.ID == params["id"] {
                  rolls = append(rolls[:i], rolls[i+1]...)
                  return
            }
      }
}
```

Setelah membuat semua function handler, jangan lupa untuk meregister function handler tersebut ke routing:

```
router := mux.NewRouter()
router.HandleFunc("/sushi", getRolls).Methods("GET")
router.HandleFunc("/sushi/{id}", getRoll).Methods("GET")
router.HandleFunc("/sushi", createRolls).Methods("POST")
router.HandleFunc("/sushi/{id}", updateRolls).Methods("PATCH")
router.HandleFunc("/sushi/{id}", deleteRolls).Methods("DELETE")
```