



Gin Framework

+

Gin Framework

Introduction

Gin adalah sebuah framework untuk bahasa Go yang digunakan untuk keperluan http routing. Dengan menggunakan Gin, maka akan sangat mempermudah kita dalam membuat Rest API yang sudah pasti memerlukan fitur routing.

Kali ini kita akan membuat sebuah Rest API sederhana yang akan melakukan CRUD data mobil.

Untuk itu, maka buatlah sebuah folder dengan nama belajar-gin, dan jalankan perintah go mod init belajar-gin pada folder tersebut.


```
mkdir belajar-gin  
cd belajar-gin  
go mod init belajar-gin
```



Gin Framework

Installation

Kemudian jalankan perintah `go get -u github.com/gin-gonic/gin` pada terminal untuk menginstall framework Gin.

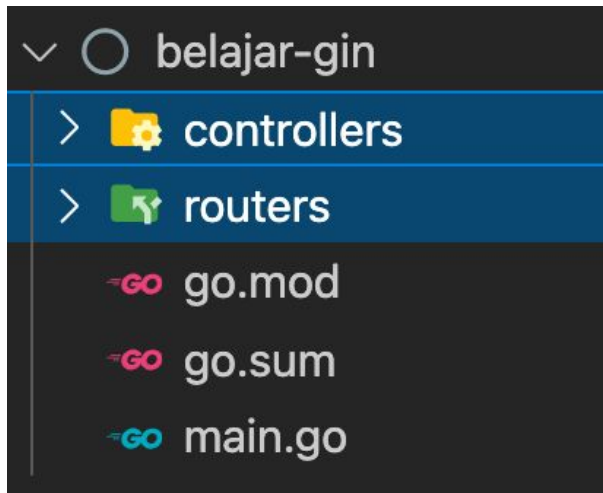


```
go get -u github.com/gin-gonic/gin
```

Setting Up Environment

Buat suatu file bernama main.go pada root direktori. File main.go itu nantinya akan menjadi entry point dari servernya.

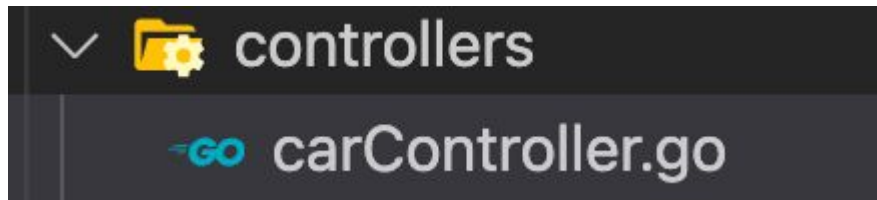
Kemudian buatlah 2 folder dengan nama routers dan juga controllers. Folder routers akan menjadi tempat kita dalam menaruh konfigurasi dari routingnya, sedangkan controllers akan menjadi tempat kita untuk menaruh endpoint-endpoint yang kita perlukan.



Setting Up Controllers

Pada folder controllers, buatlah sebuah file dengan nama carController.go.

Kemudian pada file tersebut, buat lah sebuah struct dengan nama Car yang akan kita jadikan sebagai struktur data dari data-data mobilnya, dan kita juga memerlukan sebuah variable global dengan tipe data list yang akan menampung seluruh data mobilnya.



```
10  type Car struct {  
11      CarID string `json:"car_id"`  
12      Brand  string `json:"brand"`  
13      Model  string `json:"model"`  
14      Price  int   `json:"price"`  
15  }  
16  
17  var CarDatas = []Car{}
```



Gin Framework

Create Car

Sekarang kita akan membuat sebuah endpoint untuk keperluan create data atau membuat data mobil baru.

Untuk itu pada file `carController.go`, kita perlu mengimport dulu framework Gin yang telah kita install dengan cara seperti pada gambar dibawah ini.

Kita juga perlu mengimport package `net/http` untuk penentuan status code dan package `fmt` untuk memformat data string.

```
1  package controllers
2
3  import (
4      "fmt"
5      "net/http"
6
7      "github.com/gin-gonic/gin"
8  )
```



Gin Framework

Create Car

Kemudian buatlah sebuah function dengan nama CreateCar yang menerima satu parameter dengan tipe data *gin.Context.

```
19 func CreateCar(ctx *gin.Context) {  
20  
21 }
```

Function CreateCar akan menjadi endpoint untuk proses create data. Function ini perlu menerima sebuah parameter dengan tipe data *gin.Context yang merupakan sebuah tipe data yang telah disediakan oleh framework Gin.

*gin.Context mempunyai berbagai macam method yang dapat kita gunakan untuk mendapat request body dari client, mengirim response dan lain-lain.

Gin Framework

Create Car

Ikutilah kode-kode pada gambar di sebelah kanan untuk melengkapi proses dari membuat data mobil baru.

```
19 func CreateCar(ctx *gin.Context) {
20     var newCar Car
21
22     if err := ctx.ShouldBindJSON(&newCar); err != nil {
23         ctx.AbortWithError(http.StatusBadRequest, err)
24         return
25     }
26     newCar.CarID = fmt.Sprintf("c%d", len(CarDatas)+1)
27     CarDatas = append(CarDatas, newCar)
28
29     ctx.JSON(http.StatusCreated, gin.H{
30         "car": newCar,
31     })
32 }
```

ShouldBindJSON pada baris 22 merupakan sebuah method dari tipe data `*gin.Context` yang digunakan untuk mem-binding data JSON yang dikirimkan oleh client sebagai request body kepada server. Method ShouldBindJSON menerima sebuah parameter yang dimana kita perlu meletakkan pointer dari variable yang akan menampung hasil dari data binding tersebut.

Method ShouldBindJSON akan mereturn sebuah error jika memang terjadi sebuah error, maka dari itu kita perlu memvalidasi terlebih dahulu jika terjadi sebuah error.

Gin Framework

Create Car

```
19 func CreateCar(ctx *gin.Context) {
20     var newCar Car
21
22     if err := ctx.ShouldBindJSON(&newCar); err != nil {
23         ctx.AbortWithError(http.StatusBadRequest, err)
24         return
25     }
26     newCar.CarID = fmt.Sprintf("c%d", len(CarDatas)+1)
27     CarDatas = append(CarDatas, newCar)
28
29     ctx.JSON(http.StatusCreated, gin.H{
30         "car": newCar,
31     })
32 }
```

Kemudian method `AbortWithError` pada baris 23 digunakan untuk melempar error jika memang ada error yang terjadi. Method `AbortWithError` menerima 2 parameter. Parameter pertama adalah status error nya, kemudian parameter kedua adalah data error nya.



Gin Framework

Create Car

```
19 func CreateCar(ctx *gin.Context) {
20     var newCar Car
21
22     if err := ctx.ShouldBindJSON(&newCar); err != nil {
23         ctx.AbortWithError(http.StatusBadRequest, err)
24         return
25     }
26     newCar.CarID = fmt.Sprintf("c%d", len(CarDatas)+1)
27     CarDatas = append(CarDatas, newCar)
28
29     ctx.JSON(http.StatusCreated, gin.H{
30         "car": newCar,
31     })
32 }
```

Kemudian jika tidak terjadi error ketika proses data binding, maka data request body yang telah ditampung oleh variable `newCar`, akan kita tambahkan pada variable global `CarDatas`.

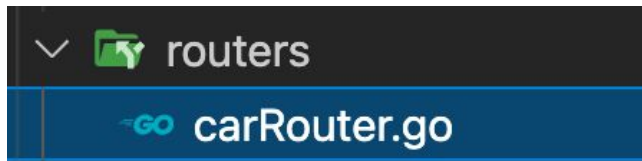
Pada baris ke 26, kita mencoba untuk mengenerate id untuk tiap data mobil baru.

Setelah selesai ditampung, maka kita mengirim data response kepada client dengan menggunakan method `JSON`. Method `JSON` digunakan untuk mengirim response kepada client dengan format data JSON. Method ini menerima 2 parameter. Parameter pertama adalah status response nya, dan parameter kedua adalah data response yang di kirimkan kepada client.

Create Car (Router)

Setelah membuat endpoint untuk membuat data mobil baru, maka kita perlu membuat routing yang akan menghubungkan client kepada endpoint tersebut.

Buatlah sebuah file dengan nama `carRoute.go` pada folder `routes`. Kemudian pada file tersebut, isilah dengan kode-kode seperti pada gambar kedua dibawah.



```
1 package routers
2
3 import (
4     "belajar-gin/controllers"
5
6     "github.com/gin-gonic/gin"
7 )
8
9 func StartServer() *gin.Engine {
10     router := gin.Default()
11
12     router.POST("/cars", controllers.CreateCar)
13
14     return router
15 }
```



Gin Framework

Create Car (Router)

```
1 package routers
2
3 import (
4     "belajar-gin/controllers"
5
6     "github.com/gin-gonic/gin"
7 )
8
9 func StartServer() *gin.Engine {
10     router := gin.Default()
11
12     router.POST("/cars", controllers.CreateCar)
13
14     return router
15 }
```

Function `StartServer` akan kita gunakan untuk menjalankan server dari aplikasi kita. Function ini mengembalikan suatu data dengan tipe data struct `*gin.Engine` yang berasal dari Gin, dan kita gunakan untuk menjalankan server, sebagai multiplexer dari routing dan lain-lain.

Pada baris ke 10, variable `router` kita gunakan sebagai penampung untuk engine dari router yang kita dapatkan dari pemanggilan function `gin.Default`.



Gin Framework

Create Car (Router)

```
1 package routers
2
3 import (
4     "belajar-gin/controllers"
5
6     "github.com/gin-gonic/gin"
7 )
8
9 func StartServer() *gin.Engine {
10     router := gin.Default()
11
12     router.POST("/cars", controllers.CreateCar)
13
14     return router
15 }
```

Pada baris ke 12, kita menggunakan method POST untuk menghubungkan client dengan endpoint CreateCar. Kita memberikan 2 parameter terhadap method POST ini. Parameter pertama adalah route path nya, sedangkan parameter kedua memerlukan handler atau endpoint nya. Endpoint harus merupakan sebuah function yang menerima satu parameter dengan tipe data *gin.Context.

Karena kita sudah membuat function bernama CreateCar yang menerima satu parameter yang sesuai dengan kriteria method POST, maka kita dapat menjadi function CreateCar untuk parameter kedua dari method POST.

Update Car

Sekarang kita akan membuat proses untuk mengupdate data mobil.

Buatlah sebuah function dengan nama UpdateCar pada file carController.go seperti pada gambar di sebelah kanan.

Pada baris 35, method `ctx.Param` merupakan sebuah method yang digunakan untuk mendapat request parameter yang dikirimkan oleh client.

Method `ctx.Param` menerima satu parameter dimana kita perlu meletakkan nama parameter yang kita buat pada router nya nanti.

```
34 func UpdateCar(ctx *gin.Context) {
35     carID := ctx.Param("carID")
36     condition := false
37     var updatedCar Car
38
39     if err := ctx.ShouldBindJSON(&updatedCar); err != nil {
40         ctx.AbortWithError(http.StatusBadRequest, err)
41         return
42     }
43
44     for i, car := range CarDatas {
45         if carID == car.CarID {
46             condition = true
47             CarDatas[i] = updatedCar
48             CarDatas[i].CarID = carID
49             break
50         }
51     }
52
53     if !condition {
54         ctx.AbortWithStatusJSON(http.StatusNotFound, gin.H{
55             "error_status": "Data Not Found",
56             "error_message": fmt.Sprintf("car with id %v not found", carID),
57         })
58         return
59     }
60
61     ctx.JSON(http.StatusOK, gin.H{
62         "message": fmt.Sprintf("car with id %v has been successfully updated", carID),
63     })
64 }
```



Update Car

Kemudian sebelum kita betul-betul mengupdate data mobilnya, pada baris 44 - 50, kita melooping data-data mobil yang ditampung oleh variable CarDatas guna untuk mencari terlebih dahulu data mobil yang ingin di update oleh client nantinya.

Kita mencari data mobilnya berdasarkan id data mobil yang dikirimkan client melalui request parameter. Jika data mobilnya tidak dapat, maka kita langsung melempar error dengan status Data Not Found(404).

Namun jika data mobilnya ada, maka kita langsung mengupdate data mobilnya dan menghentikan proses loopingnya.

```
34 func UpdateCar(ctx *gin.Context) {
35     carID := ctx.Param("carID")
36     condition := false
37     var updatedCar Car
38
39     if err := ctx.ShouldBindJSON(&updatedCar); err != nil {
40         ctx.AbortWithError(http.StatusBadRequest, err)
41         return
42     }
43
44     for i, car := range CarDatas {
45         if carID == car.CarID {
46             condition = true
47             CarDatas[i] = updatedCar
48             break
49         }
50     }
51
52     if !condition {
53         ctx.AbortWithStatusJSON(http.StatusNotFound, gin.H{
54             "error_status": "Data Not Found",
55             "error_message": fmt.Sprintf("car with id %v not found", carID),
56         })
57         return
58     }
59
60     ctx.JSON(http.StatusOK, gin.H{
61         "message": fmt.Sprintf("car with id %v has been successfully updated", carID),
62     })
63 }
```



Update Car (Router)

Sekarang kita perlu mendaftarkan endpoint UpdateCar pada function StartServer yang terletak pada file carRouter.go di dalam folder routers.

```
9  func StartServer() *gin.Engine {
10      router := gin.Default()
11
12      router.POST("/cars", controllers.CreateCar)
13
14      router.PUT("/cars/:carID", controllers.UpdateCar)
15
16      return router
17  }
```



GetCar

Buatlah sebuah function bernama GetCar yang akan kita gunakan untuk endpoint mendapatkan satu data mobil.

Buatlah function GetCar tersebut dengan kode-kode seperti pada gambar di sebelah kanan.

```
66 func GetCar(ctx *gin.Context) {
67     carID := ctx.Param("carID")
68     condition := false
69     var carData Car
70
71     for i, car := range CarDatas {
72         if carID == car.CarID {
73             condition = true
74             carData = CarDatas[i]
75             break
76         }
77     }
78
79     if !condition {
80         ctx.AbortWithStatusJSON(http.StatusNotFound, gin.H{
81             "error_status": "Data Not Found",
82             "error_message": fmt.Sprintf("car with id %v not found", carID),
83         })
84         return
85     }
86
87     ctx.JSON(http.StatusOK, gin.H{
88         "car": carData,
89     })
90 }
```



Gin Framework

GetCar (Router)

Kemudian kita perlu mendaftarkan endpoint GetCar pada function StartServer seperti endpoint-endpoint sebelumnya.

```
9  func StartServer() *gin.Engine {
10      router := gin.Default()
11
12      router.POST("/cars", controllers.CreateCar)
13
14      router.PUT("/cars/:carID", controllers.UpdateCar)
15
16      router.GET("/cars/:carID", controllers.GetCar)
17
18      return router
19  }
```



Gin Framework

DeleteCar

Dan sekarang, kita akan membuat endpoint terakhir kita yaitu endpoint untuk menghapus satu data mobil.

Buatlah sebuah function dengan nama DeleteCar seperti pada gambar di sebelah kanan.

```
92 func DeleteCar(ctx *gin.Context) {
93     carID := ctx.Param("carID")
94     condition := false
95     var carIndex int
96
97     for i, car := range CarDatas {
98         if carID == car.CarID {
99             condition = true
100             carIndex = i
101             break
102         }
103     }
104
105     if !condition {
106         ctx.AbortWithStatusJSON(http.StatusNotFound, gin.H{
107             "error_status": "Data Not Found",
108             "error_message": fmt.Sprintf("car with id %v not found", carID),
109         })
110         return
111     }
112
113     copy(CarDatas[carIndex:], CarDatas[carIndex+1:])
114     CarDatas[len(CarDatas)-1] = Car{}
115     CarDatas = CarDatas[:len(CarDatas)-1]
116
117     ctx.JSON(http.StatusOK, gin.H{
118         "message": fmt.Sprintf("car with id %v has been successfully deleted", carID),
119     })
120 }
```



Gin Framework

DeleteCar (Router)

Kemudian function DeleteCar perlu kita registrasikan pada function StartServer.

```
9 func StartServer() *gin.Engine {
10     router := gin.Default()
11
12     router.POST("/cars", controllers.CreateCar)
13
14     router.PUT("/cars/:carID", controllers.UpdateCar)
15
16     router.GET("/cars/:carID", controllers.GetCar)
17
18     router.DELETE("/cars/:carID", controllers.DeleteCar)
19
20     return router
21 }
```



Gin Framework

Entry Point

```
1  package main
2
3  import "belajar-gin/routers"
4
5  func main() {
6      var PORT = ":8080"
7
8      routers.StartServer().Run(PORT)
9  }
```

Pada file main.go, ikutilah kode diatas untuk menjalankan server nya.



Test Endpoint With Postman(Create Data)

Mari kita coba mengirim request melalui client dengan menggunakan Postman. Jalankan server dengan perintah `go run main.go`.

```
└─ go run main.go
[GIN-debug] [WARNING] Creating an Engine instance with the Logger and Recovery middleware already attached.

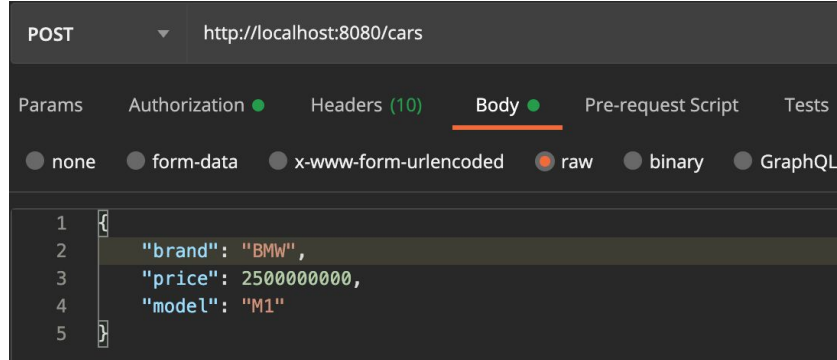
[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.
- using env:   export GIN_MODE=release
- using code:  gin.SetMode(gin.ReleaseMode)

[GIN-debug] POST    /cars                --> belajar-gin/controllers.CreateCar (3 handlers)
[GIN-debug] PUT     /cars/:carID         --> belajar-gin/controllers.UpdateCar (3 handlers)
[GIN-debug] GET      /cars/:carID         --> belajar-gin/controllers.GetCar (3 handlers)
[GIN-debug] DELETE  /cars/:carID         --> belajar-gin/controllers.DeleteCar (3 handlers)
[GIN-debug] Listening and serving HTTP on :8080
```



Test Endpoint With Postman(Create Data)

Kemudian kirimkan request seperti pada gambar dibawah ini untuk proses membuat data mobil baru.



Test Endpoint With Postman(Create Data)

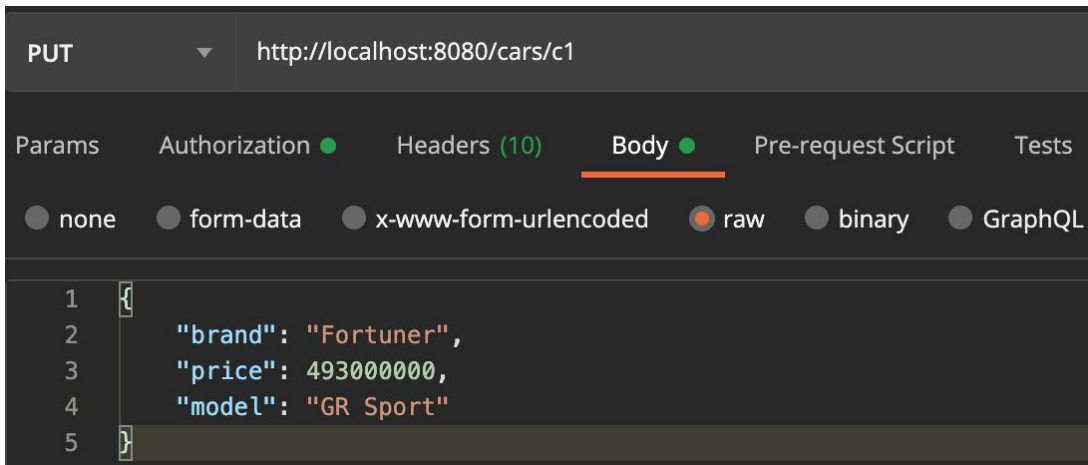
Dan response yang di dapatkan oleh client akan seperti pada gambar dibawah ini.

```
1  {  
2    "car": {  
3      "car_id": "c1",  
4      "brand": "BMW",  
5      "model": "M1",  
6      "price": 25000000000  
7    }  
8  }
```



Test Endpoint With Postman(Update Data)

Mari kita kirimkan request untuk mengupdate data mobil yang sudah kita tambahkan sebelumnya.



Test Endpoint With Postman(Update Data)

Dan response yang di dapatkan oleh client akan seperti pada gambar dibawah ini.

```
1  {  
2    "message": "car with id c1 has been successfully updated"  
3  }
```

Test Endpoint With Postman(Get Data)

Mari kita kirimkan request untuk mendapatkan data mobil yang telah kita update sebelumnya.

GET



http://localhost:8080/cars/c1



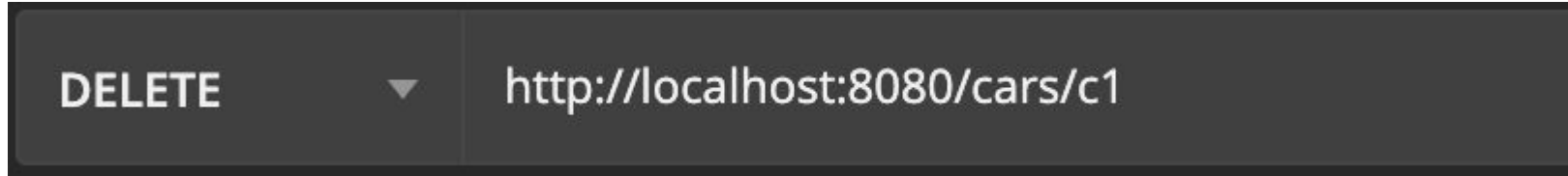
Test Endpoint With Postman(Get Data)

Dan response yang di dapatkan oleh client akan seperti pada gambar dibawah ini.

```
1  {
2    "car": {
3      "car_id": "c1",
4      "brand": "Fortuner",
5      "model": "GR Sport",
6      "price": 493000000
7    }
8  }
```

Test Endpoint With Postman(Delete Data)

Untuk yang terakhir, kita akan mengirim request untuk menghapus data mobil yang telah kita tambahkan sebelumnya.



Test Endpoint With Postman(Delete Data)

Dan response yang di dapatkan oleh client akan seperti pada gambar dibawah ini.

```
1  {  
2    "message": "car with id c1 has been successfully deleted"  
3  }
```

