

Go Programming Microservice Into The Specialization

Sesi 5 : JSON AND URL PARSING IN GO
PROGRAMMING





Decoding & Parsing JSON Data

Decoding JSON To Struct

Pada sesi kali ini, kita akan mempelajari cara mendecode data JSON kepada sebuah struct. Caranya seperti pada gambar di sebelah kanan.

Pada line 15, kita membuah data JSON sederhana menggunakan tanda *backtick* ```. Kemudian pada line 25, kita menggunakan function `json.Unmarshal` untuk mendecode data JSON kepada struct `Employee`. Argumen pertama dari function `json.Unmarshal` menerima sebuah nilai dengan tipe data *slice of byte*. Pada argument pertama itulah kita meletakkan data JSON nya tetapi harus kita ubah terlebih dahulu menjadi tipe data *slice of byte*. Kemudian pada argumen kedua, kita meletakkan pointer dari variable `result` agar setelah data JSON berhasil di decode, datanya akan disimpan kepada variable `result`.

```
1 package main
2
3 import (
4     "encoding/json"
5     "fmt"
6 )
7
8 type Employee struct {
9     FullName string `json:"full_name"`
10    Email    string `json:"email"`
11    Age      int   `json:"age"`
12 }
13
14 func main() {
15     var jsonString = `
16     {
17         "full_name": "Airell Jordan",
18         "email": "airell@mail.com",
19         "age": 23
20     }
21 `
22
23     var result Employee
24
25     var err = json.Unmarshal([]byte(jsonString), &result)
26     if err != nil {
27         fmt.Println(err.Error())
28         return
29     }
30
31     fmt.Println("full_name:", result.FullName)
32     fmt.Println("email:", result.Email)
33     fmt.Println("age:", result.Age)
34 }
```

Decoding JSON To Struct

Pada struct *Employee* yang telah kita buat diawal pada line 8 - 12. Terdapat sebuah format tulisan seperti ``json:"full_name"`, `json:"email"`, dan `json:"age"`. Tulisan-tulisan tersebut disebut sebagai tag. Tag kita gunakan ketika ingin mendecode data-data seperti JSON, form data, hingga xml kemudian kita simpan data decode tersebut kepada field-field struct nya.`

Tag yang kita buat harus kita sesuaikan dengan field pada data JSONnya. Jika di perhatikan pada line 17, terdapat field bernama *full_name*. Sedangkan pada line 9, field pada struct *Employee* nya bernama *FullName*. Karena field *full_name* pada data JSON akan kita simpan ke pada field *FullName* pada struct *Employee*, maka kita perlu menyesuaikan *tag* pada field *FullName* menjadi ``json:"full_name"`.`

```
full_name: Airell Jordan  
email: airell@mail.com  
age: 23
```

```
1 package main  
2  
3 import (  
4     "encoding/json"  
5     "fmt"  
6 )  
7  
8 type Employee struct {  
9     FullName string `json:"full_name"`  
10    Email      string `json:"email"`  
11    Age        int    `json:"age"`  
12 }  
13  
14 func main() {  
15     var jsonString = `  
16     {  
17         "full_name": "Airell Jordan",  
18         "email": "airell@mail.com",  
19         "age": 23  
20     }  
21     `,  
22  
23     var result Employee  
24  
25     var err = json.Unmarshal([]byte(jsonString), &result)  
26     if err != nil {  
27         fmt.Println(err.Error())  
28         return  
29     }  
30  
31     fmt.Println("full_name:", result.FullName)  
32     fmt.Println("email:", result.Email)  
33     fmt.Println("age:", result.Age)  
34 }
```

Decoding JSON To Map

Kita juga bisa men-decode data JSON kepada sebuah tipe data *map*. Caranya seperti pada gambar pertama di sebelah kanan. Kita tidak perlu membuat *tag* seperti yang kita lakukan pada sebuah struct.

Jika kita jalankan, maka hasilnya akan seperti pada gambar kedua.

```
1 package main
2
3 import (
4     "encoding/json"
5     "fmt"
6 )
7
8 func main() {
9     var jsonString = `
10     {
11         "full_name": "Airell Jordan",
12         "email": "airell@mail.com",
13         "age": 23
14     }
15 `
16
17     var result map[string]interface{}
18
19     var err = json.Unmarshal([]byte(jsonString), &result)
20     if err != nil {
21         fmt.Println(err.Error())
22         return
23     }
24
25     fmt.Println("full_name:", result["full_name"])
26     fmt.Println("email:", result["email"])
27     fmt.Println("age:", result["age"])
28 }
```

```
full_name: Airell Jordan
email: airell@mail.com
age: 23
```

Decoding JSON To Empty Interface

Kita juga bisa men-decode data JSON kepada sebuah tipe data *empty interface*. Caranya seperti pada gambar pertama di sebelah kanan.

Perlu diingat disini bahwa ketika kita ingin mengakses field-fieldnya, maka harus dilakukan *type assertion* dari *empty interface* menjadi tipe data *map[string]interface{}*.

Jika kita jalankan, maka hasilnya akan seperti pada gambar kedua.

```
full_name: Airell Jordan  
email: airell@mail.com  
age: 23
```

```
1 package main  
2  
3 import (  
4     "encoding/json"  
5     "fmt"  
6 )  
7  
8 func main() {  
9     var jsonString = `  
10     {  
11         "full_name": "Airell Jordan",  
12         "email": "airell@mail.com",  
13         "age": 23  
14     }  
15     `
```

```
16  
17     var temp interface{}  
18  
19     var err = json.Unmarshal([]byte(jsonString), &temp)  
20     if err != nil {  
21         fmt.Println(err.Error())  
22         return  
23     }  
24  
25     var result = temp.(map[string]interface{})  
26  
27     fmt.Println("full_name:", result["full_name"])  
28     fmt.Println("email:", result["email"])  
29     fmt.Println("age:", result["age"])  
30 }
```

Decoding Array Of JSON To Slice Of Struct

Kita juga bisa men-decode data *array of JSON* kepada sebuah tipe data *slice of struct*. Caranya seperti pada gambar pertama di sebelah kanan.

Jika kita jalankan, maka hasilnya akan seperti pada gambar kedua.

```
Index 1: {FullName:Airell Jordan Email:airell@mail.com Age:23}  
Index 2: {FullName:Ananda RHP Email:ananda@mail.com Age:23}
```

```
1 package main  
2  
3 import (  
4     "encoding/json"  
5     "fmt"  
6 )  
7  
8 type Employee struct {  
9     FullName string `json:"full_name"`  
10    Email    string `json:"email"`  
11    Age      int    `json:"age"`  
12 }  
13  
14 func main() {  
15     var jsonString = `[  
16         {  
17             "full_name": "Airell Jordan",  
18             "email": "airell@mail.com",  
19             "age": 23  
20         },  
21         {  
22             "full_name": "Ananda RHP",  
23             "email": "ananda@mail.com",  
24             "age": 23  
25         }  
26     ],  
27     ]  
28  
29     var result []Employee  
30  
31     var err = json.Unmarshal([]byte(jsonString), &result)  
32     if err != nil {  
33         fmt.Println(err.Error())  
34         return  
35     }  
36  
37     for i, v := range result {  
38         fmt.Printf("Index %d: %+v\n", i+1, v)  
39     }  
40 }
```

Encoding Objek ke JSON

Data yang biasanya di lemparkan sebagai response backend berupa JSON, maka dari itu kita juga dapat mengubah data non JSON seperti slice of struct atau object agar dapat di return menjadi data JSON.

Untuk mengubah objek ke data JSON dapat gunakan cara encoding objek ke JSON dengan **json.Marshal()**. Fungsi dari **json.Marshal()** bekerja dengan menerima variable atau value yang bertipe data any atau apapun itu untuk diubah menjadi **byte**. Fungsi tersebut juga mereturn 2 value yang berupa **byte** dan **error**.

```
> go run encode-objek-ke-json.go
[{"Name":"john wick","Age":27}, {"Name":"ethan hunt", "Age":32}]
```

```
package main

import "encoding/json"
import "fmt"

You, 7 seconds ago | 1 author (You)
type User struct {
    FullName string `json:"Name"`
    Age      int
}

func main() {
    var object = []User{{"john wick", 27}, {"ethan hunt", 32}}

    var jsonData, err = json.Marshal(object)

    if err != nil {
        fmt.Println(err.Error())
        return You, 1 second ago · Uncommitted changes
    }

    var jsonString = string(jsonData)
    fmt.Println(jsonString)
}
```




URL Parsing

URL Parsing

Data string url bisa dikonversi kedalam bentuk `url.URL`. Dengan menggunakan tipe tersebut akan ada banyak informasi yang bisa kita manfaatkan, diantaranya adalah jenis protokol yang digunakan, path yang diakses, query, dan lainnya.

Contohnya seperti pada gambar di sebelah kanan.

Function `url.Parse` digunakan untuk parsing string ke bentuk url. Mengembalikan 2 data, variabel objek bertipe `url.URL` dan error (jika ada). Melalui variabel objek tersebut pengaksesan informasi url akan menjadi lebih mudah, contohnya seperti nama host bisa didapatkan lewat `u.Host`, protokol lewat `u.Scheme`, dan lainnya.

Selain itu, query yang ada pada url akan otomatis diparsing juga, menjadi bentuk `map[string][]string`, dengan key adalah nama elemen query, dan value array string yang berisikan value elemen query.

```
1 package main
2
3 import (
4     "fmt"
5     "net/url"
6 )
7
8 func main() {
9     urlString = "http://developer.com:80/hello?name=airell&age=23"
10    var u, e = url.Parse(urlString)
11    if e != nil {
12        fmt.Println(e.Error())
13        return
14    }
15
16    fmt.Printf("url: %s\n", urlString)
17
18    fmt.Printf("protocol: %s\n", u.Scheme)
19    fmt.Printf("host: %s\n", u.Host)
20    fmt.Printf("path: %s\n", u.Path)
21
22    var name = u.Query()["name"][0]
23    var age = u.Query()["age"][0]
24    fmt.Printf("name: %s, age: %s\n", name, age)
25 }
26
```

URL Parsing

Ketika kita jalankan, maka hasilnya akan seperti pada gambar di bawah.

```
url: http://developer.com:80/hello?name=airell&age=23
protocol: http
host: developer.com:80
path: /hello
name: airell, age: 23
```



Swagger

Setelah kita belajar membuat REST API,

langkah selanjutnya adalah membagikannya dengan Consumers dan memudahkan mereka untuk memahami dan menggunakan API.

Membangun API hanyalah setengah dari pekerjaan 😊 Separuh lainnya memungkinkan klien untuk menggunakan, menjelajahi, dan menguji API tanpa terlalu banyak kerumitan.

Tanpa dokumentasi API yang jelas, konsumen akan kesulitan melakukan semua hal di atas, yang menyebabkan kerugian serius dalam produktivitas pengembang.

KENAPA SWAGGER ?

Swagger UI memberikan cara yang nyaman bagi consumer untuk menjelajahi API.

Selain itu, API berkembang dari waktu ke waktu dan dokumentasi harus mencerminkan perubahan yang sesuai (Jumlah bug yang muncul karena komunikasi perubahan API yang tidak tepat (atau tidak ada) terlalu tinggi!). Dengan Swagger, memperbarui / memelihara dokumentasi API sangat mudah - pengembang hanya perlu menambahkan / mengubah anotasi dalam kode, dan perubahannya digabungkan saat dokumen API dibuat berikutnya.

Set Up Swagger

Buat struktur folder project golang kalian dengan DDD dan gunakan juga gin framework pada golang kalian dengan melakukan install library gin seperti berikut :

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

Install *library* swagger dengan jalankan perintah berikut :

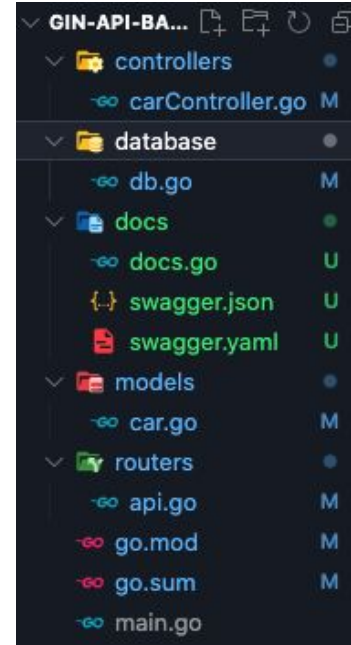
```
go get -u github.com/swaggo/swag/cmd/swag
```

```
go get -u github.com/swaggo/gin-swagger
```

```
go get -u github.com/swaggo/files
```

```
go get -u github.com/alecthomas/template
```

Gambar disamping berikut adalah susunan folder dan file yang akan kita buat untuk dapat mengimplementasikan swagger dengan menggunakan gin framework, untuk folder docs tidak perlu dibuat manual folder filenya karena akan tergenerate otomatis nanti, cara nya akan kita bahas step by step pada slide berikutnya ya



Generate Swagger Docs

Kita Bagi seluruh proses dokumentasi API menjadi 3 Langkah :

1. Add Anotasi Kode

Tambahkan Deskripsi Kode untuk :

A. Controllers setiap endpoint

```
12 // GetAllCars godoc
13 // @Summary Get details
14 // @Description Get details of all car
15 // @Tags cars
16 // @Accept json
17 // @Produce json
18 // @Success 200 {object} models.Car You, 1 second ago
19 // @Router /orders [get]
20 func GetAllCars(c *gin.Context){
21     var db = database.GetDB()
22
23     var cars []models.Car
24     err := db.Find(&cars).Error
25
26     if err != nil {
27         fmt.Println("Error getting car datas :", err.Error())
28     }
29
30     c.JSON(http.StatusOK, gin.H{"data": cars})
31 }
```

Mari kita tambahkan dokumentasi per Endpoint, gambar disamping kita tambahkan dokumentasi untuk endpoint get all cars.

Sebagai referensi, saya coba menggunakan [Method : GET] ⇒ GetAllCars API

Pada bagian @Success kita harus menuliskan model struct kita agar endpoint yg didokumentasikan ini tersambung dengan struct yg kita buat juga.

Generate Swagger Docs

```
33 // GetOneCars godoc
34 // @Summary Get details for a given Id
35 // @Description Get details of car corresponding to the input Id
36 // @Tags cars
37 // @Accept json
38 // @Produce json
39 // @Param Id path int true "ID of the car"
40 // @Success 200 {object} models.Car
41 // @Router /cars/{Id} [get]
42 func GetOneCars(c * gin.Context){
43     var db = database.GetDB()
44
45     var carOne models.Car
46     // err := db.Table("Car").Where("Id = ?", c.Param("id")).First(&car).Error
47     err := db.First(&carOne, "Id = ?", c.Param("id")).Error
48     if err != nil {
49         c.JSON(http.StatusBadRequest, gin.H{"error": "Record not found!"})
50         return
51     }
52
53     c.JSON(http.StatusOK, gin.H{"data One": carOne})
54 }
```

Mari kita tambahkan dokumentasi untuk API get car by id

Sebagai referensi, saya coba menggunakan [Method : GET] ⇒ GetOneCars API

Pada dokumentasi dengan parameter kita harus masukan anotasi @param dengan format contoh seperti ini

```
@Param [param_name] [param_type] [data_type] [required/mandatory] [description]
```


Generate Swagger Docs

```
56 // CreateCars godoc
57 // @Summary Post details for a given Id
58 // @Description Post details of car corresponding to the input Id
59 // @Tags cars
60 // @Accept json
61 // @Produce json
62 // @Param models.Car body models.Car true "create car"
63 // @Success 200 {object} models.Car
64 // @Router /cars [post]
65 func CreateCars(c *gin.Context) {
66     var db = database.GetDB()
67     // Validate input
68     var input models.Car
69     if err := c.ShouldBindJSON(&input); err != nil {
70         c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
71         return
72     }
73
74     // Create book
75     carinput := models.Car{Merk: input.Merk, Harga: input.Harga, Typecars: input.Typecars}
76     db.Create(&carinput)
77
78     c.JSON(http.StatusOK, gin.H{"data": carinput})
79 }
```

Mari kita tambahkan dokumentasi untuk API create car

Sebagai referensi, saya coba menggunakan [Method : POST] ⇒ CreateCars API

Pada dokumentasi dengan parameter kita harus masukan anotasi @param dengan format contoh seperti ini

```
@Param [param_name] [param_type] [data_type] [required/mandatory] [description]
```

Dokumentasi method post dengan swagger perlu dituliskan param *models.Car* **body** *models.Car* **true** agar terdokumentasi sebagai parameter yang membaca body request

Generate Swagger Docs

```
81 // UpdateCars godoc
82 // @Summary Update car identified by the given Id
83 // @Description Update the car corresponding to the input Id
84 // @Tags cars
85 // @Accept json
86 // @Produce json
87 // @Param id path int true "ID of the car to be updated"
88 // @Success 200 {object} models.Car
89 // @Router /cars/{id} [patch] You, 1 second ago • Uncommitted changes
90 func UpdateCars(c *gin.Context){
91     var db = database.GetDB()
92
93     var car models.Car
94     err := db.First(&car, "Id = ?", c.Param("id")).Error
95     if err != nil {
96         c.JSON(http.StatusBadRequest, gin.H{"error": "Record not found!"})
97         return
98     }
99     // Validate input
100     var input models.Car
101     if err := c.ShouldBindJSON(&input); err != nil {
102         c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
103         return
104     }
```

Mari kita tambahkan dokumentasi untuk API update car

Sebagai referensi, saya coba menggunakan [Method : PATCH] ⇒ UpdateCars API

Pada dokumentasi update ini parameter yg kita gunakan berupa **id** yang merupakan panggilan dari gin.Context param().

Generate Swagger Docs

```
111 // DeleteCars godoc
112 // @Summary Delete car identified by the given id
113 // @Description Delete the order corresponding to the input id
114 // @Tags cars
115 // @Accept json
116 // @Produce json
117 // @Param id path int true "ID of the car to be deleted"
118 // @Success 204 "No Content"
119 // @Router /cars/{id} [delete]
120 func DeleteCars(c *gin.Context){
121     var db = database.GetDB()
122     // Get model if exist
123     var carDelete models.Car
124     err := db.First(&carDelete, "Id = ?", c.Param("id")).Error;
125     if err != nil {
126         c.JSON(http.StatusBadRequest, gin.H{"error": "Record not found!"})
127         return
128     }
129
130     db.Delete(&carDelete)
131
132     c.JSON(http.StatusOK, gin.H{"data": true})
133 }
```

Mari kita tambahkan dokumentasi untuk API delete car

Sebagai referensi, saya coba menggunakan [Method : DELETE] ⇒ DeleteCars API

Pada dokumentasi update ini parameter yg kita gunakan berupa **id** yang merupakan panggilan dari gin.Context param().

Generate Swagger Docs

Tambahkan Deskripsi Kode untuk :
B. Model/entity setiap struct

```
1 package models
2
3
4 // Car represents the model for an car
  You, 4 hours ago | 2 authors (hacktiv8 and others)
5 type Car struct{      hacktiv8, 5 months ago
6     Pemilik string `gorm:"varchar(100)"`
7     Merk string `gorm:"varchar(100)"`
8     Harga int `gorm:"integer(11)"`
9     Typecars string `gorm:"varchar(100)"`
10    Id uint `gorm:"primaryKey"`
11 }
```

Anotasi deskripsi kode pada struct harus di represents sebagai model tertentu yang nantinya harus disamakan pada pembuatan anotasi di controller kita juga contoh

// Car represents the model for an car

Car pada kata pertama menunjukan pada type Car struct{} yang kita buat dibawahnya.

Generate Swagger Docs

Tambahkan Deskripsi Kode untuk :

C. Routing setiap endpoint & starting server

```
You, 4 hours ago | 2 authors (You and others)
1 package routers
2
3 import (
4     "gin-api/controllers"
5
6     "github.com/gin-gonic/gin"
7
8     _ "gin-api/docs" // docs is generated by Swag CLI, you have to import it.
9
10    ginSwagger "github.com/swaggo/gin-swagger"
11
12    swaggerfiles "github.com/swaggo/files"
13 )
```

Perlu diingat kita harus import library yang sudah kita go get atau download di awal tadi.

GinSwagger sebagai alias yang dapat memanggil package gin-swagger,

SwaggerFiles sebagai alias yang dapat digunakan untuk menampilkan file html dari swagger.

_"gin-api/docs" adalah untuk mengimport file docs yang tergenerate otomatis dari swagger (gin-api dipanggil dari nama module di go.mod).

```
15 // @title Car API
16 // @version 1.0
17 // @description This is a sample service for managing cars
18 // @termsOfService http://swagger.io/terms/
19 // @contact.name API Support
20 // @contact.email soberkoder@swagger.io
21 // @license.name Apache 2.0
22 // @license.url http://www.apache.org/licenses/LICENSE-2.0.html
23 // @host localhost:8080
24 // @basePath /
25 func StartServer() *gin.Engine{
26     router := gin.Default()
27
28     // Read
29     router.GET("/cars/:id", controllers.GetOneCars)
30     // Create
31     router.POST("/cars", controllers.CreateCars)
32     // Read All
33     router.GET("/cars", controllers.GetAllCars)
34     // Update
35     router.PATCH("/cars/:id", controllers.UpdateCars)
36     // DELETE
37     router.DELETE("/cars/:id", controllers.DeleteCars)
38
39     router.GET("/swagger/*any", ginSwagger.WrapHandler(swaggerfiles.Handler))
40
41     return router
42 }
```

Generate Swagger Docs

2. Generate folder Docs Swagger

```
> swag init -g routers/api.go
2022/11/23 16:08:55 Generate swagger docs....
2022/11/23 16:08:55 Generate general API Info, search dir:./
2022/11/23 16:08:55 Generating models.Car
2022/11/23 16:08:55 create docs.go at docs/docs.go
2022/11/23 16:08:55 create swagger.json at docs/swagger.json
2022/11/23 16:08:55 create swagger.yaml at docs/swagger.yaml
```

Mari kita generate folder dengan isi swagger.json dan swagger.yaml nya

Fyi :

Swagger.json dan swagger.yaml merupakan konfigurasi dari swagger yang nantinya akan ditampilkan pada website, untuk mengubahnya teman2 tidak dapat mengubah secara manual melainkan setiap ada perubahan anotasi kode swagger perlu dilakukan generate ulang dengan
`swag init -g [nama-file-yang-ada-swagger]`

Apabila ada kendala pada running perintah swag init dapat ketikan
`export PATH=$(go env GOPATH)/bin:$PATH` terlebih dahulu sebelum melakukan generate

Serve & Run Swagger

3. Serving Swagger

Mari kita server dengan menggunakan go run main.go dan output pada terminal akan seperti pada gambar dibawah

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

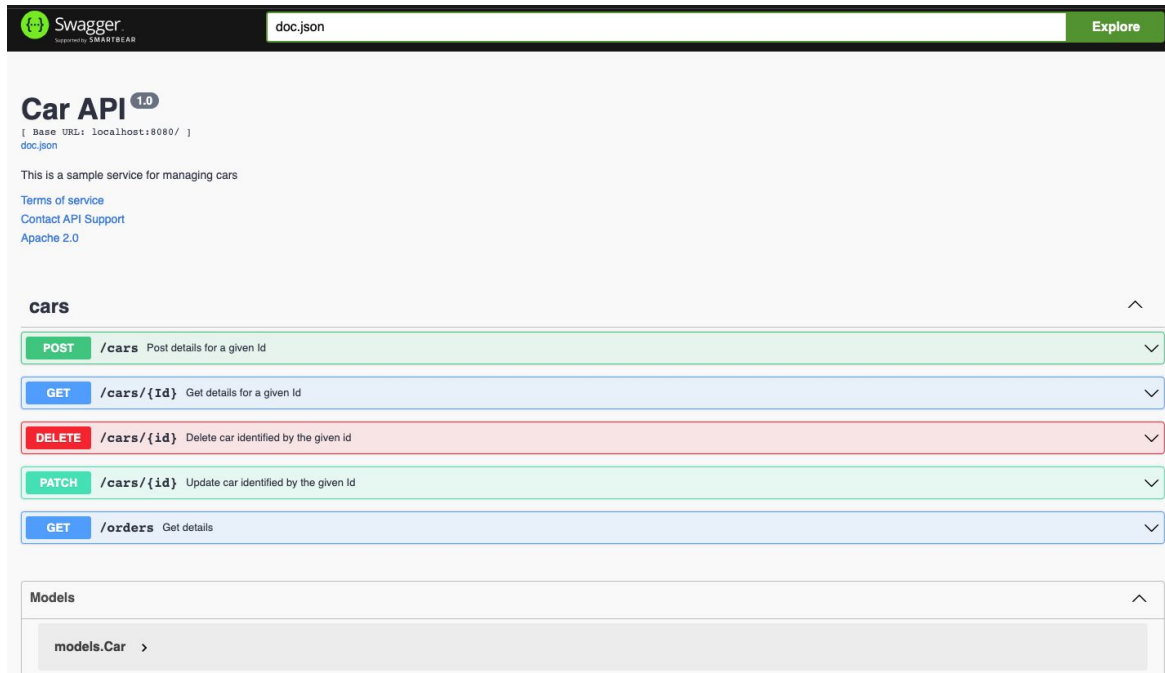
[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.
- using env:   export GIN_MODE=release
- using code:  gin.SetMode(gin.ReleaseMode)

[GIN-debug] GET    /cars/:id      -> gin-api/controllers.GetOneCars (3 handlers)
[GIN-debug] POST   /cars          -> gin-api/controllers.CreateCars (3 handlers)
[GIN-debug] GET    /cars          -> gin-api/controllers.GetAllCars (3 handlers)
[GIN-debug] PATCH  /cars/:id      -> gin-api/controllers.UpdateCars (3 handlers)
[GIN-debug] DELETE /cars/:id      -> gin-api/controllers.DeleteCars (3 handlers)
[GIN-debug] GET    /swagger/*any   -> github.com/swagger/gin-swagger.CustomWrapHandler.func1 (3 handlers)
[GIN-debug] [WARNING] You trusted all proxies, this is NOT safe. We recommend you to set a value.
Please check https://pkg.go.dev/github.com/gin-gonic/gin#readme-don-t-trust-all-proxies for details.
[GIN-debug] Listening and serving HTTP on :8080
```

Serve & Run Swagger

3. Serving Swagger

Hasil :



The screenshot displays the Swagger UI for a "Car API" version 1.0. The interface includes a top navigation bar with the Swagger logo, a search bar containing "doc.json", and an "Explore" button. Below the header, the API title "Car API 1.0" is shown, along with the base URL "[Base URL: localhost:8080/]" and a link to "doc.json". A description states, "This is a sample service for managing cars," followed by links for "Terms of service", "Contact API Support", and "Apache 2.0".

The main section, titled "cars", lists several API endpoints:

- POST** `/cars`: Post details for a given id
- GET** `/cars/{id}`: Get details for a given id
- DELETE** `/cars/{id}`: Delete car identified by the given id
- PATCH** `/cars/{id}`: Update car identified by the given id
- GET** `/orders`: Get details

Below the endpoints, there is a "Models" section showing a single model: `models.Car` with a right-pointing arrow.