

더 쉽게 개발하고 싶었어요



GIN-RESTFUL

목차

- ▶ 발표자 소개
- ▶ gin-restful 소개
- ▶ gin을 이용한 rest api 구현
- ▶ gin-restful을 사용한 rest api 구현
- ▶ gin-restful의 구현방법
- ▶ 마무리

발표자 소개

만드는 것이 즐거운 개발자

- ▶ 이름: 황선우
- ▶ 나이: 만 17세
- ▶ 대덕소프트웨어마이스터고등학교 2학년
정보보안과 재학
- ▶ java, kotlin, golang, python 등 사용



GIN-RESTFUL 소개



gin framework 소개

GO언어 웹 프레임워크

- ▶ 웹 mvc 프레임워크

- ▶ revel

- ▶ buffalo

- ▶ beego

- ▶ 마이크로 웹 프레임워크

- ▶ gin

- ▶ echo

- ▶ iris



시연

GIN으로
RESTFUL API 구현해보기


```
1  package main
2
3  import (
4      "github.com/gin-gonic/gin"
5      "log"
6  )
7
8  ► func main() {
9      r := gin.Default()
10     r.GET(relativePath: "/api/samples", func(c *gin.Context) {
11         c.JSON(code: 200, gin.H{"message": "Hello, Gin!"})
12     })
13     log.Fatalln(r.Run(addr...: ":5000"))
14 }
```

```
1 {
2     "message": "Hello, Gin!"
3 }
```

GIN FRAMEWORK 의 장점

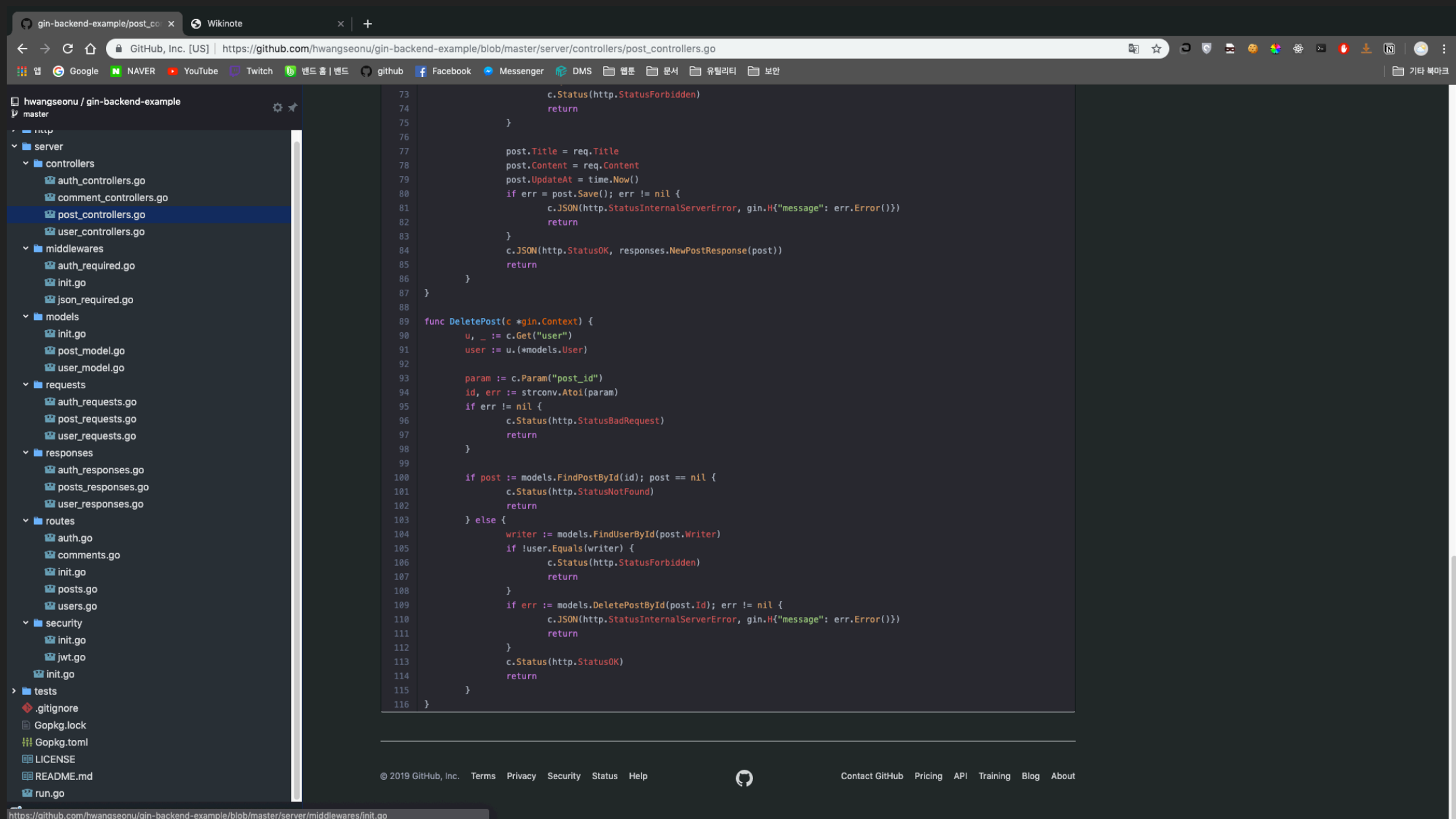
- ▶ Go 언어로 웹 서비스를 쉽게 개발할 수 있다.
- ▶ 다른 언어나 프레임워크로 만든 웹에 비해 성능이 좋다
- ▶ Go 언어의 장점인 병행프로그래밍을 활용할 수 있다.
- ▶ 자유로운 구조로 프로젝트를 구성할 수 있다.

GIN FRAMEWORK 의 단점

- ▶ Response를 전달한 후에 따로 return을 해주지 않으면 함수가 종료되지 않는다.
- ▶ Handle 함수의 형태가 고정되어 있다.
- ▶ *gin.Context 에서 요청에 대한 정보를 함수 내부에서 가져와야 하므로 코드가 길어진다.
- ▶ 프로젝트의 규모가 커지면 프로젝트의 구조와 코드가 복잡해진다.

GIN 프레임워크를 활용한 프로젝트

▶ <https://github.com/hwangseonu/gin-backend-example>



RETURN

► <https://github.com/hwangseonu/gin-backend-example>

```
func UpdatePost(c *gin.Context) {
    body, _ := c.Get("body")
    req := body.(*requests.CreatePostRequest)
    u, _ := c.Get("user")
    user := u.(*models.User)

    param := c.Param("post_id")
    id, err := strconv.Atoi(param)
    if err != nil {
        c.Status(http.StatusBadRequest)
        return
    }

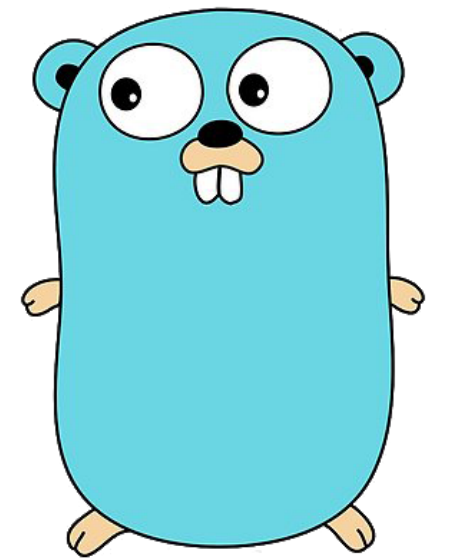
    if len(req.Title) <= 0 || len(req.Content) <= 0 {
        c.Status(http.StatusBadRequest)
        return
    }

    if post := models.FindPostById(id); post == nil {
        c.Status(http.StatusNotFound)
        return
    } else {
        writer := models.FindUserById(post.Writer)
        if !user.Equals(writer) {
            c.Status(http.StatusForbidden)
            return
        }

        post.Title = req.Title
        post.Content = req.Content
        post.UpdateAt = time.Now()
        if err = post.Save(); err != nil {
            c.JSON(http.StatusInternalServerError, gin.H{"message": err.Error()})
            return
        }
        c.JSON(http.StatusOK, responses.NewPostResponse(post))
        return
    }
}
```

그래서 만들었습니다.

더 쉽게 개발하고 싶었어요



GIN-RESTFUL

GIN-RESTFUL이란

- ▶ Flask restful extension 에서 영감을 받아 만든 gin 웹 프레임워크의 extension
- ▶ gin으로 restful api를 쉽게 작성할 수 있도록 하기 위해 제작
- ▶ 자원을 구조체로, 자원에 대한 행위를 메서드로 정의함으로써 api를 제작할 수 있음
/:string0/:int1/:string2
- ▶ 함수의 매개변수를 파싱하여 url 규칙을 자동으로 생성해줌
- ▶ 어떤 자원에 대한 요청이 들어 왔을 때 해당하는 자원의 메서드를 실행

GIN-RESTFUL 살펴보기

▶ <https://github.com/hwangseonu/gin-restful>

```
package main

import (
    "github.com/gin-gonic/gin"
    "github.com/hwangseonu/gin-restful"
    "net/http"
)

//restful api 를 구현하기 위한 SampleResource 구조체입니다.
//gin-restful 의 Resource 구조체 포인터를 일대일합니다.
//각 Handler 마다 다른 Middleware 들을 적용할 수 있습니다.
type SampleResource struct {
    *gin_restful.Resource
}

//Json Body 를 테스트하기 위한 구조체입니다.
type Data struct {
    Name string `json:"name" validate:"required,notblank"`
}

//SampleResource 의 Url 로 GET 요청이 들어왔을 때 실행되는 Handler 입니다.
//gin.H(json) 와 status code 를 반환합니다.
//path variable 인 name 을 json 에 담아 반환합니다.
func (r SampleResource) Get(name string) (gin.H, int) {
    return gin.H{
        "name": name,
    }, http.StatusOK
}

//SampleResource 의 Url 로 POST 요청이 들어왔을 때 실행되는 Handler 입니다.
//json body 를 Data 구조체로 받습니다.
//gin.H 과 status code 를 반환합니다.
//요청으로 받은 payload 를 그대로 반환합니다.
func (r SampleResource) Post(c *gin.Context, json Data) (Data, int) {
    return json, 200
}

//Middleware 테스트용 Sample Middleware 입니다.
//콘솔에 "Hello, World" 를 출력합니다.
func SampleMiddleware(c *gin.Context) {
    println("Hello, World")
}

//Api 인스턴스를 "/" 주소로 생성합니다.
//SampleResource 의 인스턴스를 생성하여 Api "/samples" 주소로 등록합니다.
//SampleResource GET handler 에 SampleMiddleware 를 등록합니다.
//gin 서버를 5000 포트에서 실행합니다.
func main() {
    r := gin.Default()
    v1 := gin_restful.NewApi(r, "/")
    res := SampleResource{gin_restful.InitResource()}
    res.AddMiddleware(SampleMiddleware, http.MethodGet)
    v1.AddResource(res, "/samples")
    _ = r.Run(":5000")
}
```

GIN-RESTFUL

```
1  package main
2
3  import (
4      "github.com/gin-gonic/gin"
5      "github.com/hwangseonu/gin-restful"
6  )
7
8  type Sample struct {}
9
10 func (s Sample) Get() gin.H {
11     return gin.H{"message": "Hello, Gin!"}
12 }
13
14 func main() {
15     r := gin.Default()
16     api := gin_restful.NewApi(r, prefix: "/api")
17     api.AddResource(Sample{}, url: "/samples")
18     _ = r.Run(addr...: ":5000")
19 }
```

```
1 {
2     "message": "Hello, Gin!"
3 }
```

GIN-RESTFUL EXAMPLE

▶ <https://github.com/hwangseonu/gin-restful>

```
package main

import (
    "github.com/gin-gonic/gin"
    "github.com/hwangseonu/gin-restful"
    "net/http"
)

//restful api 를 구현하기 위한 SampleResource 구조체입니다.
//gin-restful 의 Resource 구조체 포인터를 임베딩합니다.
//각 Handler 마다 다른 Middleware 들을 적용할 수 있습니다.
type SampleResource struct {
    *gin_restful.Resource
}

//Json Body 를 테스트하기 위한 구조체입니다.
type Data struct {
    Name string `json:"name" validate:"required,notblank"`
}

//SampleResource 의 Url 로 GET 요청이 들어왔을 때 실행되는 Handler 입니다.
//gin.H(json) 와 status code 를 반환합니다.
//path variable 인 name 을 json 에 담아 반환합니다.
func (r SampleResource) Get(name string) (gin.H, int) {
    return gin.H{
        "name": name,
    }, http.StatusOK
}

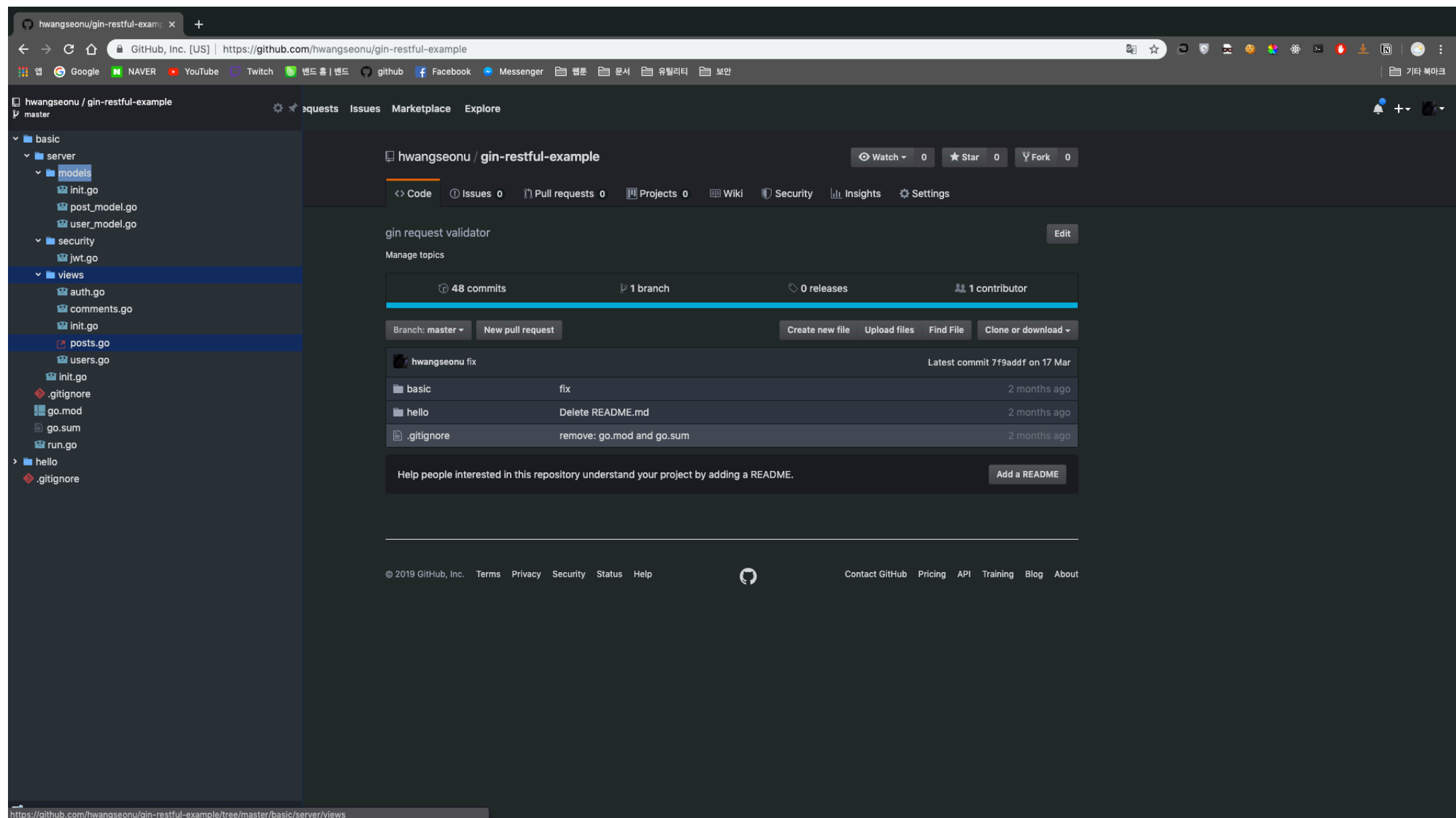
//SampleResource 의 Url 로 POST 요청이 들어왔을 때 실행되는 Handler 입니다.
//json body 를 Data 구조체로 받습니다.
//gin.H 과 status code 를 반환합니다.
//요청으로 받은 payload 를 그대로 반환합니다.
func (r SampleResource) Post(c *gin.Context, json Data) (Data, int) {
    return json, 200
}

//Middleware 테스트용 Sample Middleware 입니다.
//콘솔에 "Hello, World" 를 출력합니다.
func SampleMiddleware(c *gin.Context) {
    println("Hello, World")
}

//Api 인스턴스를 "/" 주소로 생성합니다.
//SampleResource 의 인스턴스를 생성하여 Api "/samples" 주소로 등록합니다.
//SampleResource GET handler 에 SampleMiddleware 를 등록합니다.
//gin 서버를 5000 포트에서 실행합니다.
func main() {
    r := gin.Default()
    v1 := gin_restful.NewApi(r, "/")
    res := SampleResource{gin_restful.InitResource()}
    res.AddMiddleware(SampleMiddleware, http.MethodGet)
    v1.AddResource(res, "/samples")
    _ = r.Run(":5000")
}
```

GIN-RESTFUL EXAMPLE

▶ <https://github.com/hwangseonu/gin-restful-example>



GIN-RESTFUL EXAMPLE

```
func UpdatePost(c *gin.Context) {
    body, _ := c.Get("body")
    req := body.(*requests.CreatePostRequest)
    u, _ := c.Get("user")
    user := u.(*models.User)

    param := c.Param("post_id")

func (r Posts) Patch(c *gin.Context, id int, req CreatePostRequest) (gin.H, int) {
    user := c.MustGet("user").(*models.UserModel)
    p := models.FindPostById(id)
    if p == nil {
        return gin.H{"message": "cannot find post by id"}, http.StatusNotFound
    }
    if p.Writer != user.Id {
        return gin.H{"message": "cannot access this resource"}, http.StatusForbidden
    }
    p.Title = req.Title
    p.Content = req.Content
    p.UpdateAt = time.Now()
    return PostResponse(p), http.StatusOK
}

    c.JSON(http.StatusInternalServerError, gin.H{"message": err.Error()})
    return
}
    c.JSON(http.StatusOK, responses.NewPostResponse(post))
    return
}
}
```

GIN-RESTFUL은 어떻게 구현되었을까?

GIN-RESTFUL

- ▶ <https://github.com/hwangseonu/gin-restful>
- ▶ reflect 사용

 .gitignore


 LICENSE

 README.md


 api.go

 error.go

 go.mod

 go.sum

 init.go

 resource.go

 utils.go

API.GO

```
//Api 구조체는 Resource 인스턴스들을 관리하고 gin 서버에 등록하기 위한 구조체입니다.
//NewApi 함수로 인스턴스를 생성하여 사용합니다.
//AddResource 함수로 Api 인스턴스에 Resource 를 등록할 수 있습니다.
```

```
type Api struct {
    App      *gin.RouterGroup
    Prefix    string
    Resources map[string]interface{}
}
```

```
//Api 구조체의 인스턴스를 인스턴스를 생성하여 포인터로 반환하는 함수입니다.
```

```
//Api 인스턴스에 등록된 Resource 를 gin 서버에 등록시키는 메서드입니다.
```

```
func (a *Api) registerResource(resource interface{}, url string) {
    for i := 0; i < reflect.TypeOf(resource).NumMethod(); i++ {
        value := reflect.ValueOf(resource)
        method := reflect.TypeOf(resource).Method(i)
        if !isHttpMethod(method.Name) {
            continue
        }
        args := parseArgs(method)
        url := createUrl(url, args)
        g := a.App.Group(url, parseMiddlewares(resource, method.Name)...)
        g.Handle(strings.ToUpper(method.Name), "", createHandlerFunc(value, method, args))
    }
}
```

```
//string, int, float, bool 타입의 인자는 url 에서 파싱하여 전달합니다.
```

```
//*gin.Context 타입의 인자는 해당 요청의 context 로 채워집니다.
```

```
//구조체 타입의 인자는 하나만 존재할 수 있으며 요청의 body 를 파싱하여 채워집니다.
```

```
func (a *Api) AddResource(resource interface{}, url string) {
    if a.App != nil {
        a.registerResource(resource, url)
    } else {
        a.Resources[url] = resource
    }
}
```


RESOURCE.GO

```
type Resource struct {
    Middlewares map[string][]gin.HandlerFunc
}

//Resource 구조체를 초기화하여 포인터로 반환해주는 함수입니다.
//새로운 Resource 구조체를 정의하여 사용할 때 Resource 를 임베딩 하기 위해 사용합니다.
func InitResource() *Resource {
    return &Resource{
        Middlewares: make(map[string][]gin.HandlerFunc, 0),
    }
}

//Resource 인스턴스에 각 http method 에 사용할 Middleware 를 등록하는 메서드입니다.
//methods 는 사용가능한 http method 의 이름과 같아야 합니다.
func (r *Resource) AddMiddleware(middleware gin.HandlerFunc, methods ...string) {
    for _, m := range methods {
        m = strings.ToUpper(m)
        middlewares := r.Middlewares[m]
        middlewares = append(middlewares, middleware)
        r.Middlewares[m] = middlewares
    }
}
```

UTILS.GO

- ▶ <https://github.com/hwangseonu/gin-restful/blob/master/Utils.go>
- ▶ 내용이 너무 많아요ㅠㅠ;

`/:string0/:int1`

아쉬운점 & 부족한점

아쉬운점

- ▶ restful api 더 편하게 할 수 있도록 하지 못한 점
- ▶ xml 등과 같은 Json 외의 데이터 형식을 지원하지 못한 점
- ▶ multipart를 지원하지 않는 점
- ▶ 함수의 인자로 int, float64, string, bool 밖에는 지원하지 못하는 점

Q & A

무엇이든 물어보세요



**THANKS FOR
YOUR
ATTENTION**