Go

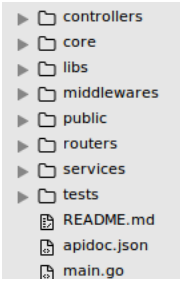
Best practices on project structure

Restfull Projects API with Gin

Gin is a web framework written in Golang. It features a martini-like API with much better performance, up to 40 times faster. If you need performance and good productivity, you will love Gin.

There will be 8 packages + main.go

1. controllers
2. core
3. libs
4. middlewares
5. public
6. routers
7. services
8. tests
9. main.go



---

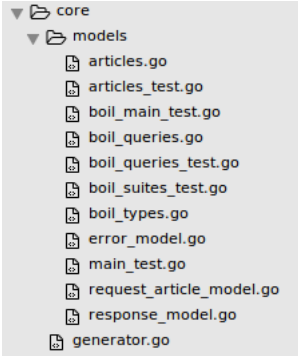
controllers

Controllers package will store all the API logic. Whatever your API, your logic will happen here

---

core

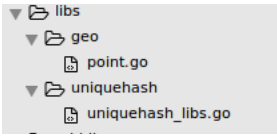
Core package will store all your created models, ORM, etc



---

libs

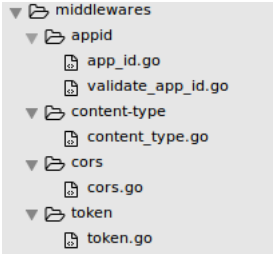
This package will store any library that used in projects. But only for manually created/imported library, that not available when using `go get package_name` commands. Could be your own hashing algorithm, graph, tree etc.



---

middlewares

This package store every middleware that used in project, could be creation/validation of cors,device-id , auth etc



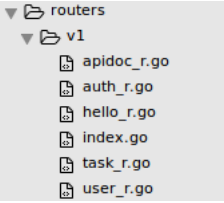
---

public

This pacakge will store every public and static files, could be html, css, javascript ,images, etc

routers

This package will store every routes in your REST API.



See sample code how to assign the routes.

auth_r.go

```
import (
    auth "simple-api/controllers/v1/auth"
    "gopkg.in/gin-gonic/gin.v1"
)

func SetAuthRoutes(router *gin.RouterGroup) {

/**
 * @api {post} /v1/auth/login Login
 * @apiGroup Users
 * @apiHeader {application/json} Content-Type Accept application/json
 * @apiParam {String} username User username
 * @apiParam {String} password User Password
 * @apiParamExample {json} Input
 *    {
 *      "username": "your username",
 *        "password"     : "your password"
 *    }
 * @apiSuccess {Object} authenticate Response
 * @apiSuccess {Boolean} authenticate.success Status
 * @apiSuccess {Integer} authenticate.statuscode Status Code
 * @apiSuccess {String} authenticate.message Authenticate Message
 * @apiSuccess {String} authenticate.token Your JSON Token
 * @apiSuccessExample {json} Success
 *    {
 *        "authenticate": {
 *               "statuscode": 200,
 *              "success": true,
 *           "message": "Login Successfully",
 *              "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV
9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ"
 *            }
 *      }
 * @apiErrorExample {json} List error
 *    HTTP/1.1 500 Internal Server Error
 */

    router.POST("/auth/login" , auth.Login)
}
```

If you see, the reason I separate the handler is, to easy us to manage each routers. So I can create comments about the API , that with apidoc will generate this into structured documentation. Then I will call the function in index.go in current package

index.go

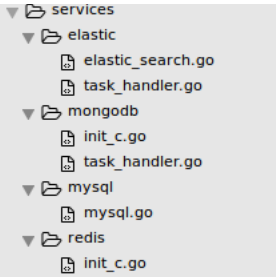
```
package v1


import (
    "gopkg.in/gin-gonic/gin.v1"
    token "simple-api/middlewares/token"
    appid "simple-api/middlewares/appid"
)
func InitRoutes(g *gin.RouterGroup)  {
    g.Use(appid.AppIDMiddleWare())
    SetHelloRoutes(g)
    SetAuthRoutes(g) // SetAuthRoutes invoked
    g.Use(token.TokenAuthMiddleWare())  //secure the API From this line to bottom with JSON Auth
    g.Use(appid.ValidateAppIDMiddleWare())
    SetTaskRoutes(g)
    SetUserRoutes(g)
}
```

---

services

This package will store any configuration and setting to used in project from any used service, could be mongodb,redis,mysql, elasticsearch, etc.



---

main.go

The main entrance of the API. Any configuration about the dev environment settings, systems,port, etc will configured here.

Example:
main.go

```
package main
import (
    "fmt"
    "net/http"
    "gopkg.in/gin-gonic/gin.v1"
    "articles/services/mysql"
    "articles/routers/v1"
    "articles/core/models"
)

var router  *gin.Engine;

func init() {
    mysql.CheckDB()
    router = gin.New();
    router.NoRoute(noRouteHandler())
    version1:=router.Group("/v1")
    v1.InitRoutes(version1)
```



```
func main() {
    fmt.Println("Server Running on Port: ", 9090)
    http.ListenAndServe(":9090",router)
}

func noRouteHandler() gin.HandlerFunc{
    return  func(c *gin.Context) {
    var statuscode     int
    var message        string         = "Not Found"
    var data         interface{} = nil
    var listError [] models.ErrorModel = nil
    var endpoint    string = c.Request.URL.String()
    var method        string = c.Request.Method

    var tempEr models.ErrorModel
    tempEr.ErrorCode     = 4041
    tempEr.Hints         = "Not Found !! \n Routes In Valid. You enter on invalid Page/Endpoint"
    tempEr.Info            = "visit http://localhost:9090/v1/docs to see the available routes"
    listError             = append(listError,tempEr)
    statuscode             = 404
    responseModel := &models.ResponseModel{
        statuscode,
        message,
        data,
        listError,
        endpoint,
        method,
    }
    var content gin.H = responseModel.NewResponse();
    c.JSON(statuscode,content)
    }
}
```

ps: Every code in this example, come from different projects

---

see sample projects on github

Previous Next