REST API sederhana (CRUD) dengan konsep Todos

REST singkatan bahasa Inggris dari *Representational State Transfer*, adalah suatu gaya arsitektur perangkat lunak untuk untuk pendistibusian sistem hipermedia seperti www. Istilah ini diperkenalkan pertama kali pada tahun 2000 pada disertasi doktoral Roy Fielding. Pada arsitektur REST, REST server menyediakan *resources* (sumber daya/data) dan REST client mengakses dan menampilkan *resource* tersebut untuk penggunaan selanjutnya. Disini kita menggunakan Package gorilla/mux sebagai implementasi sebuah request router dan dispatcher (operator) untuk mencocokkan permintaan yang masuk ke handler masing-masing. Penerapan REST API dengan konsep Todos ini juga menggunakan Go Swag.

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. Dengan swagger, memperbaharui/memelihara dokumentasi API sangat mudah. Pengembang hanya perlu menambahkan/mengubah anotasi dalam kode, dan perubahannya digabungkan saat dokumen API dibuat berikutnya. Swaggo dan go-swagger adalah dua framework paling populer yang tersedia untuk menghasilkan dokumen.

Langkah-langkah pengerjaannya:

1. Instal library dengan menjalankan perintah:

```
go get -u github.com/swaggo/swag/cmd/swag
go get -u github.com/swaggo/http-swagger
go get -u github.com/alecthomas/template
```

2. Generate Swagger Docs

Ada 3 langkah :

1. Add Anotasi Kode

a. General API pada main.go

```
// @title Hacktiv8 TODOS API
// @version 1.0
// @description This is API for completing final project 1 hacktiv8
// @license.name Apache 2.0
// @license.url http://www.apache.org/licenses/LICENSE-2.0.html
// @host localhost:8000
// @BasePath /
func main() {
    mux := mux.NewRouter()
    router.New(mux)

    fmt.Println("http listen on http://localhost:8000")
    http.ListenAndServe(":8000", mux)
}
```

b. Menambahkan dokumentasi project-level

Web Service API adalah sebuah web yang menerima request dari client dan menghasilkan response. Selanjutnya buat fungsi (c *Controller) untuk handle endpoint/controller. Didalam fungsi tersebut ada proses deteksi jenis request untuk mencari tahu apakah jenis request adalah GET, POST dan lainnya.

[Controller : CreateTodo]

```
// CreateTodo
// @Tags Todos
// @Summary Create a new Todo to the database
// @Description Create a new Todo
// @Accept json
// @Produce json
// @Param todo body model.TodoRequest true "create a new todo"
// @Success 200 {object} helper.BaseResponse{data=model.Todo}
// @Failure 400 {object} helper.BaseResponse
// @Router /todos [POST]
func (c *Controller) CreateTodo(w http.ResponseWriter, r *http.Request) {
    body, err := ioutil.ReadAll(r.Body)
    if err != nil {
        helper.Response(w, helper.BaseResponse{Status: http.StatusBadRequest,
Message: err.Error(), Data: nil})
        return
    }

    todo := new(model.Todo)
    err = json.Unmarshal(body, todo)
    if err != nil {
        helper.Response(w, helper.BaseResponse{Status: http.StatusBadRequest,
Message: err.Error(), Data: nil})
        return
    }
```

```
    createdTodo := c.listTodos.Add(*todo)
    helper.Response(w, helper.BaseResponse{Status: http.StatusOK, Message:
"success create data", Data: createdTodo})
}
```

[Controller : GetAllTodos]

```
// GetAllTodos
// @Tags Todos
// @Summary get all todos .
// @Description get all todso from the database
// @Accept json
// @Produce json
// @Success 200 {object} helper.BaseResponse{data=[]model.Todo}
// @Failure 400 {object} helper.BaseResponse{}
// @Router /todos [GET]
func (c *Controller) GetAllTodos(w http.ResponseWriter, r *http.Request) {

    resultTodo := c.listTodos.GetAll()

    respon := helper.BaseResponse{
        Status:  http.StatusOK,
        Message: "success get all data",
        Data:    resultTodo,
    }
    helper.Response(w, respon)
}
```

[Controller : GetTodoByid]

```
// GetTodoById
// @Tags Todos
// @Summary get a todo for a given todo id.
// @Description get a todo with id from the database
// @Accept json
// @Produce json
// @Param ID path int true "ID of the todo"
// @Success 200 {object} helper.BaseResponse{data=model.Todo}
// @Failure 400 {object} helper.BaseResponse{}
// @Router /todos/{id} [GET]
func (c *Controller) GetTodoById(w http.ResponseWriter, r *http.Request) {

    // handling path ID
    vars := mux.Vars(r)
    todoID, err := strconv.Atoi(vars["id"])
    if err != nil {
```

```
        response := helper.BaseResponse{
            Status:  http.StatusBadRequest,
            Message: err.Error(),
            Data:    nil,
        }
        helper.Response(w, response)
        return
    }

    resultTodo, err := c.listTodos.GetTodoById(todoID)
    if err != nil {
        response := helper.BaseResponse{
            Status:  http.StatusOK,
            Message: err.Error(),
            Data:    nil,
        }
        helper.Response(w, response)
        return
    }

    response := helper.BaseResponse{
        Status:  http.StatusOK,
        Message: "success",
        Data:    resultTodo,
    }
    helper.Response(w, response)
}
```

[Controller : UpdateTodoByid]

```
// UpdateTodoById
// @Tags Todos
// @Summary Update todo with given todo id.
// @Description Update the todo corresponding to the input id
// @Accept json
// @Produce json
// @Param todo body model.TodoRequest true "update"
// @Param ID path int true "ID of the todo"
// @Success 200 {object} helper.BaseResponse{data=model.Todo}
// @Failure 400 {object} helper.BaseResponse{}
// @Router /todos/{id} [PUT]
func (c *Controller) UpdateTodoById(w http.ResponseWriter, r *http.Request) {

    // handling path ID
    vars := mux.Vars(r)
    todoID, err := strconv.Atoi(vars["id"])
    if err != nil {
```

```
        helper.Response(w, helper.BaseResponse{Status: http.StatusBadRequest,
Message: err.Error(), Data: nil})
        return
    }

    // read body request
    readBody, errRead := ioutil.ReadAll(r.Body)
    if errRead != nil {
        helper.Response(w, helper.BaseResponse{Status: http.StatusBadRequest,
Message: err.Error(), Data: nil})
        return
    }
    var todo model.Todo
    // unmarshal to struct
    err = json.Unmarshal(readBody, &todo)

    if err != nil {
        helper.Response(w, helper.BaseResponse{Status: http.StatusBadRequest,
Message: err.Error(), Data: nil})
        return
    }
    // call database function update
    todo.ID = todoID
    err = c.listTodos.UpdateTodoById(todo)
    if err != nil {
        helper.Response(w, helper.BaseResponse{Status: http.StatusNotFound,
Message: err.Error(), Data: nil})
        return
    }

    // successreturn
    helper.Response(w, helper.BaseResponse{Status: http.StatusOK, Message:
"success update", Data: todo})
}
```

[Controller : DeleteTodoByid]

```
// DeleteTodoById
// @Tags Todos
// @Summary delete todo with given todo id.
// @Description delete the todo corresponding to the input id
// @Accept json
// @Produce json
// @Param ID path int true "ID of the todo"
// @Success 200 {object} helper.BaseResponse{}
// @Failure 400 {object} helper.BaseResponse{}
// @Router /todos/{id} [DELETE]
func (c *Controller) DeleteTodoById(w http.ResponseWriter, r *http.Request) {
```

```
    // handling path ID
    vars := mux.Vars(r)
    todoID, err := strconv.Atoi(vars["id"])
    if err != nil {
        helper.Response(w, helper.BaseResponse{Status: http.StatusBadRequest,
Message: err.Error(), Data: nil})
        return
    }

    err = c.listTodos.DeleteTodoById(todoID)
    if err != nil {
        helper.Response(w, helper.BaseResponse{Status: http.StatusNotFound,
Message: err.Error(), Data: nil})
        return
    }

    // success
    helper.Response(w, helper.BaseResponse{Status: http.StatusOK, Message:
"success Delete", Data: nil})
}
```

Jika request adalah GET,POST dan sebagainya, maka data yang di-encode ke JSON dijadikan sebagai reponse.

```
package helper

import (
    "encoding/json"
    "net/http"
)

type BaseResponse struct {
    Status  int         `json:"status"`
    Message string      `json:"message"`
    Data    interface{} `json:"data"`
}

func Response(w http.ResponseWriter, data BaseResponse) {
    w.Header().Set("Access-Control-Allow-Origin", "*")
    w.Header().Set("Content-Type", "application/json")
    w.WriteHeader(data.Status)
    dataJson, _ := json.Marshal(data)
    w.Write(dataJson)
}
```

Folder helper mengatur masalah response. Statement w.Header().Set("Content-Type", "application/json") digunakan untuk menentukan tipe response, yaitu sebagai JSON. Sedangkan w.Write() digunakan untuk mendaftarkan data sebagai response.

Selebihnya, jika tidak valid maka di set sebagai error yang diletak di folder database

```
package database

import (
    "errors"

    "github.com/Melvitasari3/golang-todos-crud/model"
)

type ListTodos struct {
    List []model.Todo
}

func (l *ListTodos) Add(todo model.Todo) model.Todo {
    id := 1
    if len(l.List) != 0 {
        id = l.List[len(l.List)-1].ID + 1
    }
    todo.ID = id
    l.List = append(l.List, todo)
    return todo
}

func (l *ListTodos) GetAll() []model.Todo {
    return l.List
}

func (l *ListTodos) GetTodoById(id int) (model.Todo, error) {
    for _, val := range l.List {
        if val.ID == id {
            return val, nil
        }
    }
    return model.Todo{}, errors.New("not found")
}

func (l *ListTodos) UpdateTodoById(todo model.Todo) error {
    for idx, val := range l.List {
        if val.ID == todo.ID {
            l.List[idx].Name = todo.Name
            return nil
        }
    }
    return errors.New("id not found")
}

func (l *ListTodos) DeleteTodoById(id int) error {
    var index *int
```

```
    for i, val := range l.List {
        if val.ID == id {
            index = &i
            break
        } else {
            index = nil
        }
    }

    if index == nil {
        return errors.New("id not found")
    }

    l.List[*index] = l.List[len(l.List)-1]
    l.List[len(l.List)-1] = model.Todo{}
    l.List = l.List[:len(l.List)-1]

    return nil
}
```

Buat juga Values for Model

```
package model

type Todo struct {
    ID   int    `json:"id" example:"1"`
    Name string `json:"name" example:"Todos API"`
}

type TodoRequest struct {
    Name string `json:"name" example:"Todos API"`
}
```

2. Swagger Specs (swagger.json)

```
swag init -g main.go
```

```
2021/11/18 11:33:44 Generate swagger docs....
2021/11/18 11:33:44 Generate general API Info, search dir:./
2021/11/18 11:34:20 Generating model.TodoRequest
2021/11/18 11:34:20 Generating helper.BaseResponse
2021/11/18 11:34:20 Generating model.Todo
2021/11/18 11:34:20 create docs.go at docs\docs.go
2021/11/18 11:34:20 create swagger.json at docs\swagger.json
2021/11/18 11:34:20 create swagger.yaml at docs\swagger.yaml
```

3. Serving Swagger UI yang digenerate di folder controller

```
import (
    "encoding/json"
    "io/ioutil"
    "net/http"
    "strconv"

    "github.com/gorilla/mux"

    "github.com/Melvitasari3/golang-todos-crud/database"
    "github.com/Melvitasari3/golang-todos-crud/helper"
    "github.com/Melvitasari3/golang-todos-crud/model"
)
```

FYI

Untuk menspesifikasikan router pada semua API, akan mendefine router di main method menggunakan Pathprefix method.

```
func New(mux *mux.Router) {
    var listTodos database.ListTodos

    ctrl := controller.New(&listTodos)

    mux.HandleFunc("/todos", ctrl.CreateTodo).Methods("POST")

    mux.HandleFunc("/todos", ctrl.GetAllTodos).Methods("GET")
    // Handle get todo by id
    mux.HandleFunc("/todos/{id}", ctrl.GetTodoById).Methods("GET")
    // Handle update todo by id
    mux.HandleFunc("/todos/{id}", ctrl.UpdateTodoById).Methods("PUT")

    mux.HandleFunc("/todos/{id}", ctrl.DeleteTodoById).Methods("DELETE")

    // routing swagger API
    mux.PathPrefix("/swagger/").Handler(httpSwagger.Handler(
        httpSwagger.URL("http://localhost:8000/swagger/doc.json"), //The url
pointing to API definition
        httpSwagger.DeepLinking(true),
        httpSwagger.DocExpansion("none"),
        httpSwagger.DomID("#swagger-ui"),
    ))
}
```

Setelah selesai semua. eksekusi dengan jalankan perintah berikut :

```
go run main.go
```