Utilizando API First com golang e goswagger

API-first approach é uma nova prática para construção de APIs, na qual desenhamos primeiro a API e depois geramos o código básico para implementá-la.

Esta abordagem garante que o código gerado será compatível com a API e já gera a documentação para você.

Vamos começar com uma api muito simples que retorna as medidas captadas por um sensor remoto IoT:

```
measurement{
      sensor          string
      timeTaken    data/hora
      value             string
}
```

Quando você solicitar a URL “/measurements” deve receber a lista de medidas:

```
HTTP/1.1 200 OK
Content-Type: application/network.golang.demoapi-list.v1+json
Date: Wed, 23 Nov 2022 20:09:06 GMT
Content-Length: 224

[{"sensor":"sensor1","timeTaken":"2011-11-01T10:10:10.000Z","value":
"333"},{"sensor":"sensor2","timeTaken":"2011-11-01T10:10:10.000Z","v
alue":"333"},{"sensor":"sensor3","timeTaken":"2011-11-01T10:10:10.00
0Z","value":"333"}]
```

Desenvolver isso em Golang é simples. Podemos utilizar o pacote “net/http” com ou sem Gorilla/Mux para gerar esta API. Mas você pode fazer isso em 15 minutos? É claro que não.

O **goswagger** permite fazer isso de forma bem simples, prática e documentacional.

Podemos instalar o goswagger ou podemos simplesmente executá-lo a partir de um contêiner **Docker**. E foi o que fizemos, para não instalar nada na máquina. No site deles eles explicam como fazer isso.

Gerando o swagger.yml

A primeira coisa que você deve fazer é criar uma pasta para seu módulo Golang:

```
go mod init <url>/projeto
```

No meu caso usei:

```
go mod init network.golang/measurements
```

Agora podemos criar a descrição da nossa API, que é um arquivo swagger.yml, dentro dessa pasta. É possível fazer isso à mão, mas eu recomendo usar o próprio pacote goswagger:

```
swagger init spec \
  --title "Sensor readings" \
  --description "Return sensor readings from database" \
  --version 1.0.0 \
  --scheme http \
  --consumes application/network.golang.demoapi-measurement.v1+json \
  --produces application/network.golang.demoapi-list.v1+json
```

O arquivo gerado é bem simples e não tem as definições de dados e nem as rotas (paths):

```
consumes:
    - application/network.golang.demoapi-measurement.v1+json
info:
    description: Return sensor readings from database
    title: Sensor readings
    version: 1.0.0
paths: {}
```

```
produces:
    - application/network.golang.demoapi-list.v1+json
schemes:
    - http
swagger: "2.0"
```

Vamos acrescentar os tipos de dados na seção “definitions:”, logo ao fim do arquivo:

```
definitions:
    measurement:
        type: object
        required:
            - timeTaken
            - sensor
            - value
        properties:
            timeTaken:
                type: string
                format: date-time
            sensor:
                type: string
                minLength: 8
            value:
                type: string
```

Ok, o que temos aqui? Um tipo de dados chamado “measurement”, com 3 atributos, todos obrigatórios (required):

- timeTaken: A data/hora em que a medida foi tomada, string
- sensor: O identificador do sensor, string
- value: O valor medido, também string

E precisamos criar rotas para acessar essas informações, o que fazemos na seção “paths:”:

```
paths:
    /measurements:
        get:
            tags:
                - measurements
            responses:
                200:
                    description: List of measurements
                    schema:
                        type: array
                        items:
                            $ref: "#/definitions/measurement"
```

Muito simples. Temos uma rota “/measurements” que é consumida por um método HTTP GET. A resposta de status 200 deve retornar um vetor de elementos do tipo “measurement” (utilizamos o $ref para referenciar o tipo de dados que criamos).

Pronto! A API está definida. Agora, podemos gerar o código.

Geração do código Golang

Para gerar o código Golang do servidor, podemos usar o comando:

```
swagger generate server -A measurements -f ./swagger.yml
```

Ele gerará um projeto completo, que pode até ser servido:

```
go run cmd/measurements-server/main.go --port=8080
```

Se tentarmos acessar as medidas, tomaremos um erro 501 not implemented. Então, precisamos configurar a função que responderá a esta rota. Isto é feito no arquivo “configure_measurements.go”, dentro da pasta “restAPI”.

Ele gerou vários “responders” para nossa API, os tipos de dados etc.

No arquivo “configure_measurements.go” precisamos criar o handler para isso: “api.MeasurementsGetMeasurementsHandler”. Você notará que ele testa se esse handler é **nil**, gerando o erro 501. Basta substituir esse comando por outro:

```
api.MeasurementsGetMeasurementsHandler =
measurements.GetMeasurementsHandlerFunc(
func(params measurements.GetMeasurementsParams) middleware.Responder
{
  return measurements.NewGetMeasurementsOK().WithPayload(items)
})
```

O método “WithPayload” foi criado para nós no tipo “measurements.NewGetMeasurementsOK”, e devemos passar um vetor de ponteiros para elementos “Measurement”, então, criamos algumas variáveis para isso como globais no “configure_measurements.go”:

```
// Fake database:
var (
      sensor1 = "sensor1"
      sensor2 = "sensor2"
      sensor3 = "sensor3"
      data = strfmt.DateTime(time.Date(2011, 11, 01, 10, 10, 10, 0,
                  time.UTC))
      medida = "333"
      items = []*models.Measurement{
      &models.Measurement{&sensor1, &data, &medida},
      &models.Measurement{&sensor2, &data, &medida},
      &models.Measurement{&sensor3, &data, &medida},
      }
)
```

Na realidade, utilizaríamos um database para ler essas informações, mas aqui, para demonstrar ou para criar um serviço de teste, vamos apenas criar um vetor hardcoded.

Pronto! Agora você já pode acessar sua API.

Consumindo a API gerada

Suba o servidor com o comando:

```
go run cmd/measurements-server/main.go --port=8080
```

E rode um cURL:

```
curl -i http://localhost:8080/measurements
HTTP/1.1 200 OK
Content-Type: application/network.golang.demoapi-list.v1+json
Date: Wed, 23 Nov 2022 20:09:06 GMT
Content-Length: 224

[{"sensor":"sensor1","timeTaken":"2011-11-01T10:10:10.000Z","value":
"333"},{"sensor":"sensor2","timeTaken":"2011-11-01T10:10:10.000Z","v
alue":"333"},{"sensor":"sensor3","timeTaken":"2011-11-01T10:10:10.00
0Z","value":"333"}]
```

Pronto! API funcionando CQD (como queríamos demonstrar). Porém, tem mais!!!!

Acessando a documentação

Como isso é Swagger, você pode acessar a documentação da API com:

http://localhost:8080/docs

E verá algo assim: