Relatório das atividades delegadas pela TrackSale
Lucas Alcântara

Para a resolução das tarefas foi utilizada a linguagem GO.

Tarefa principal: Desenvolva um teste automatizado para a API pública de uma base de dados de cervejas. De forma geral, você deve escrever os testes de forma que acessem o *endpoint* **/beers** com o método **GET**, verificar e informar se os tipos de dados esperados são equivalentes aos que são retornados. (Cabe a você pesquisar na documentação da API quais seriam estes dados).

Na tarefa principal, o primeiro passo a ser feito foi a conexão da minha aplicação com a API BreweryDB, que foi feita da seguinte forma:

```
response,err:=http.Get("http://api.brewerydb.com/v2/beers/?key=8c8de43d4fb51642
8fc3baba71e8eca6&name=****")
if err != nil {
  fmt.Print(err.Error())
  os.Exit(1)
}
```

Sendo na primeira linha onde leio os dados e os guardo na variável response. as linhas seguintes são em caso de haver alguma falha com a API. O passo seguinte é ler o fluxo de bytes requisitados da API usando a função da linguagem ioutil.ReadAll e converte-la numa string (responseData)

```
responseData, err := ioutil.ReadAll(response.Body)
if err != nil {
  log.Fatal(err)
}
```

O próximo passo na aplicação foi criar alguns structs para que pudesse mapear a estrutura retornada pela API.no tipo JSON.

```
//struct responsável para mapear toda a estrutura
type Response struct {
  Page    int    `json:"totalResults"`
  Beer []Beer `json:"data"`
}

//struct responsável para mapear todas as cervejas disponiveis
type Beer struct {

  EntryNo string `json:"id"`
  Name string `json:"name"`
  ABV string `json:"abv"`
  IBU string `json:"ibu"`
  IsOrganic string `json:"isOrganic"`
  Status string `json:"status"`
  Data string `json:"createDate"`


}
```

Feito isso, foi feito o desmembramento do JSON nas structs criadas.

```
var responseObject Response
json.Unmarshal(responseData, &responseObject)
```

Em seguida foi feito as funções assert para poder verificar os dados recebidos pelos que foram especificados pela API e conferir se o retorno de dados está correto. Eu criei duas funções, a primeira chamada verificaString e a segunda verificaNum, sendo a primeira para campos de texto e a segunda campos inteiros.

```
func verificaString(text string){

  _, err := strconv.ParseInt(text, 10, 64)
  if err != nil {
  }else{
     fmt.Printf("data error: expected only string")
     fmt.Print(" - ")
     fmt.Println(text)
     os.Exit(0)
  }
}
```

A lógica dessa função está em tentar converter um campo em float, sendo que se for bem sucedido significa que o campo pode ser um int ou float, se for mal sucedido significa que o campo apenas uma string.

```
func verificaNum(text string){
  if text != "" {
     _, err := strconv.ParseFloat(text, 64)
     if err != nil {
        fmt.Printf("data error: expected only numbers")
        fmt.Printf(" - ")
        fmt.Println(text)
        os.Exit(0)
     }else{
     }
  }
}
```

A lógica nesta função é inversa a primeira, ao tentar converter um campo em float e falhar significa que o texto é apenas uma string, e se for bem sucedido significa que o campo é um int ou float. Com as duas funções assert prontas, foi feito um laço no programa principal onde foi conferido os campos retornados pela consulta na API.

```
for i := 0; i < len(responseObject.Beer); i++ {
  verificaString(responseObject.Pokemon[i].EntryNo)
  verificaString(responseObject.Pokemon[i].Name)
  verificaNum(responseObject.Pokemon[i].ABV)
  verificaNum(responseObject.Pokemon[i].IBU)
  verificaString(responseObject.Pokemon[i].IsOrganic)
  verificaString(responseObject.Pokemon[i].Status)
```

```
  verificaString(responseObject.Pokemon[i].Data)
}
```

E assim, foi terminada a tarefa principal. Segue abaixo o código completo da tarefa principal.

```
package main

import (
  "encoding/json"
  "fmt"
  "io/ioutil"
  "log"
  "net/http"
  "os"
  "strconv"
)


type Response struct {
  Page    int    `json:"totalResults"`
  Beer []Beer `json:"data"`
}

type Beer struct {

  EntryNo string `json:"id"`
  Name string `json:"name"`
  ABV string `json:"abv"`
  IBU string `json:"ibu"`
  IsOrganic string `json:"isOrganic"`
  Status string `json:"status"`
  Data string `json:"createDate"`


}

func main() {

response,err:=http.Get("http://api.brewerydb.com/v2/beers/?key=8c8de43d4fb51642
8fc3baba71e8eca6&name=****")

if err != nil {
     fmt.Print(err.Error())
     os.Exit(1)
  }

  responseData, err := ioutil.ReadAll(response.Body)
  if err != nil {
     log.Fatal(err)
  }

  var responseObject Response
  json.Unmarshal(responseData, &responseObject)
```

```
  fmt.Println(responseObject.Page)
  fmt.Println(len(responseObject.Beer))

  for i := 0; i < len(responseObject.Beer); i++ {
     verificaString(responseObject.Beer[i].EntryNo)
     verificaString(responseObject.Beer[i].Name)
     verificaNum(responseObject.Beer[i].ABV)
     verificaNum(responseObject.Beer[i].IBU)
     verificaString(responseObject.Beer[i].IsOrganic)
     verificaString(responseObject.Beer[i].Status)
     verificaString(responseObject.Beer[i].Data)
  }

}

  func verificaString(text string){

     i, err := strconv.ParseFloat(text, 64)
     if err != nil {
        fmt.Println(i)
     }else{
        fmt.Println("data error")
        os.Exit(0)
     }
  }

  func verificaNum(text string){
     i, err := strconv.ParseFloat(text, 64)
     if err != nil {
        fmt.Println("data error")
        os.Exit(0)
     }else{
        fmt.Println(i)
     }
  }
```

Tarefa extra número 1: No mesmo repositório da tarefa principal, desenvolva outro teste com qualquer coisa que tenha te chamado a atenção no *endpoint* **/beers** (usando o método **GET**).

Para esta tarefa, eu decidir acessar o endpoint **GET:** /beer/:beerId/breweries, usando o método GET, para realizar esta tarefa eu criei dois structs para comportar os dados retornados pela API. Segue abaixo o código dos structs:

```
type ResponseB struct {
  BeerB []BeerB `json:"data"`
}

type BeerB struct {
  EntryNo string `json:"id"`
  Name string `json:"name"`
  Descri string `json:"description"`
  Year string `json:"established"`
}
```

Com as structs prontas, foi necessário repetir os passos da tarefa principal para poder recuperar os dados da API, segue abaixo o código fonte:

```
responseb, err := http.Get(out)
if err != nil {
  fmt.Print(err.Error())
  os.Exit(1)
}

responseDatab, err := ioutil.ReadAll(responseb.Body)
if err != nil {
  log.Fatal(err)
}

var responseObjectb ResponseB
json.Unmarshal(responseDatab, &responseObjectb)
```

Para o funcionamento correto desta tarefa, eu tive que concatenar três strings para poder fazer a pesquisa na API, a variável `out` representa o resultado da concatenação, pois neste caso, eu teria que pesquisar cada query com uma id de uma beer da tarefa principal. Segue abaixo o código usado para concatenar as strings:

```
var link string = "http://api.brewerydb.com/v2/beer/"
var codigo string = responseObject.Beer[i].EntryNo
var rlink string = "/breweries/?key=8c8de43d4fb516428fc3baba71e8eca6"
out := fmt.Sprint(link, codigo, rlink)
```

Feito a concatenação das strings, e recuperado os novos dados da API, eu chamei novamente os métodos assert para verificar o retorno esperado de cada campo recuperado. Segue abaixo o código desta etapa.

```
for j := 0; j < len(responseObjectb.BeerB); j++ {
  verificaString(responseObjectb.BeerB[j].EntryNo)
  //fmt.Println(responseObjectb.BeerB[j].EntryNo)
  verificaString(responseObjectb.BeerB[j].Name)
  //fmt.Println(responseObjectb.BeerB[j].Name)
  verificaString(responseObjectb.BeerB[j].Descri)
  //fmt.Println(responseObjectb.BeerB[j].Descri)
  verificaNum(responseObjectb.BeerB[j].Year)
  //fmt.Println(responseObjectb.BeerB[j].Year)
}
```

E desta forma, foi terminada a tarefa extra número 1, e abaixo, segue o código fonte adicionado a tarefa principal.

```
type ResponseB struct {
  BeerB []BeerB `json:"data"`
}
```

```
type BeerB struct {
  EntryNo string `json:"id"`
  Name string `json:"name"`
  Descri string `json:"description"`
  Year string `json:"established"`
}


func main() {
  response, err :=
http.Get("http://api.brewerydb.com/v2/beers/?key=8c8de43d4fb516428fc3baba71e8ec
a6&name=****")
  if err != nil {
     fmt.Print(err.Error())
     os.Exit(1)
  }

  responseData, err := ioutil.ReadAll(response.Body)
  if err != nil {
     log.Fatal(err)
  }

  var responseObject Response
  json.Unmarshal(responseData, &responseObject)

  fmt.Println(responseObject.Page)
  fmt.Println(len(responseObject.Beer))

  for i := 0; i < len(responseObject.Beer); i++ {
     verificaString(responseObject.Beer[i].EntryNo)
     verificaString(responseObject.Beer[i].Name)
     verificaNum(responseObject.Beer[i].ABV)
     verificaNum(responseObject.Beer[i].IBU)
     verificaString(responseObject.Beer[i].IsOrganic)
     verificaString(responseObject.Beer[i].Status)
     verificaString(responseObject.Beer[i].Data)
  }
  //----------inicio Tarefa extra 1----------------------//

  for i := 0; i < len(responseObject.Beer); i++ {
     var link string = "http://api.brewerydb.com/v2/beer/"
     var codigo string = responseObject.Beer[i].EntryNo
     var rlink string = "/breweries/?key=8c8de43d4fb516428fc3baba71e8eca6"
     out := fmt.Sprint(link, codigo, rlink)
     fmt.Println(out)

     responseb, err := http.Get(out)
     if err != nil {
        fmt.Print(err.Error())
        os.Exit(1)
     }
```

```
responseDatab, err := ioutil.ReadAll(responseb.Body)
     if err != nil {
        log.Fatal(err)
     }

     var responseObjectb ResponseB
     json.Unmarshal(responseDatab, &responseObjectb)

     for j := 0; j < len(responseObjectb.BeerB); j++ {
        verificaString(responseObjectb.BeerB[j].EntryNo)
        //fmt.Println(responseObjectb.BeerB[j].EntryNo)
        verificaString(responseObjectb.BeerB[j].Name)
        //fmt.Println(responseObjectb.BeerB[j].Name)
        verificaString(responseObjectb.BeerB[j].Descri)
        //fmt.Println(responseObjectb.BeerB[j].Descri)
        verificaNum(responseObjectb.BeerB[j].Year)
        //fmt.Println(responseObjectb.BeerB[j].Year)
     }
  }
}
```

Tarefa extra número 2: Crie, em um repositório separado do anterior, um webserver em **Go** ou **PHP** com um *endpoint* que retorne: em 90% das vezes em que for chamado, um número randômico; e, em 10% das vezes em que for chamado, uma *string* (também randômica).

Nesta tarefa, foi necessário a criação de um web server com o GO. segue abaixo o código referente a criação do webserver,

```
func sayhelloName(w http.ResponseWriter, r *http.Request) {
  r.ParseForm()
  fmt.Println(r.Form)
  fmt.Println("path", r.URL.Path)
  fmt.Println("scheme", r.URL.Scheme)
  fmt.Println(r.Form["url_long"])
  for k, v := range r.Form {
     fmt.Println("key:", k)
     fmt.Println("val:", strings.Join(v, ""))
  }
}
```

Depois de criado o webserver, foi usado os mesmos passos feitos na tarefa principal para recuperar as informações do endpoint na API. Segue abaixo o código.

```
type Response struct {
  Page   int    `json:"totalResults"`
  Beer []Beer `json:"data"`
}

type Beer struct {
  EntryNo string `json:"id"`
  Name string `json:"name"`
  ABV string `json:"abv"`
  IBU string `json:"ibu"`
```

```
  IsOrganic string `json:"isOrganic"`
  Status string `json:"status"`
  Data string `json:"createDate"`
}

response,err:=http.Get("http://api.brewerydb.com/v2/beers/?key=8c8de43d4fb5164
28fc3baba71e8eca6&name=****")
if err != nil {
  fmt.Print(err.Error())
  os.Exit(1)
}
responseData, err := ioutil.ReadAll(response.Body)
if err != nil {
  log.Fatal(err)
}
var responseObject Response
json.Unmarshal(responseData, &responseObject)

for i := 0; i < len(responseObject.Beer); i++ {
  fmt.Fprintf(w, strconv.Itoa(i))
  fmt.Fprintf(w, " - ")
  fmt.Fprintln(w, responseObject.Beer[i].Name) // send data to client side
}
```

Através de um cálculo simples de probabilidade pude definir quantas vezes será necessária a chamada de uma string e de um int. A fórmula usada foi P = Na/N, onde P é a probabilidade, Na o número de casos favoráveis e N o número de casos totais. Feito o cálculo com a possibilidade total de 50 casos, cheguei a probabilidade de 45 casos em que retorna um número randômico e 5 casos em que retorna uma string randômica. Abaixo segue o código que usa dessa probabilidade, onde ele gera um número aleatório entre 0 e 1000 e baseado no resto da divisão por 2, ele determina qual será a probabilidade a ser chamada.

```
var prob int
var contI int = 45
var contS int = 5

for i := 0; i < len(responseObject.Beer); i++ {

  if contS > 0 {
     prob = rand.Intn(1000)
  }else{
     prob = 0
  }


  fmt.Println(prob)
  if  prob%2 == 0 {
     if(contI > 0){
        fmt.Fprintf(w, strconv.Itoa(i))
        fmt.Fprintf(w, " - ")
        if responseObject.Beer[i].ABV == "" {
```

```
            fmt.Fprintln(w, responseObject.Beer[i].IBU)
         }else if responseObject.Beer[i].IBU == ""{
            fmt.Fprintln(w, responseObject.Beer[i].ABV)
         }
         contI--
      }
   }else if  prob%2 != 0 {
      if(contS > 0){
         fmt.Fprintf(w, strconv.Itoa(i))
         fmt.Fprintf(w, " - ")
         fmt.Fprintln(w, responseObject.Beer[i].Name)
         contS--
      }
   }
}
```

E assim, foi terminada a tarefa extra numero 2 . Segue abaixo o seu código completo.

```
package main

import (
  "fmt"
  "net/http"
  "strings"
  "log"
  "encoding/json"
  "os"
  "io/ioutil"
  "strconv"
  "math/rand"
  "time"
)

type Response struct {
  Page   int     `json:"totalResults"`
  Beer []Beer `json:"data"`
}

// A Pokemon Struct to map every pokemon to.
type Beer struct {

  EntryNo string `json:"id"`
  Name string `json:"name"`
  ABV string `json:"abv"`
  IBU string `json:"ibu"`
  IsOrganic string `json:"isOrganic"`
  Status string `json:"status"`
  Data string `json:"createDate"`


}


func sayhelloName(w http.ResponseWriter, r *http.Request) {
```

```
  r.ParseForm()  // parse arguments, you have to call this by yourself
  fmt.Println(r.Form)  // print form information in server side
  fmt.Println("path", r.URL.Path)
  fmt.Println("scheme", r.URL.Scheme)
  fmt.Println(r.Form["url_long"])
  for k, v := range r.Form {
     fmt.Println("key:", k)
     fmt.Println("val:", strings.Join(v, ""))
  }

  ///-------------------------------------------------///
  response, err :=
http.Get("http://api.brewerydb.com/v2/beers/?key=8c8de43d4fb516428fc3baba71e8ec
a6&name=****")
  if err != nil {
     fmt.Print(err.Error())
     os.Exit(1)
  }

  responseData, err := ioutil.ReadAll(response.Body)
  if err != nil {
     log.Fatal(err)
  }

  var responseObject Response
  json.Unmarshal(responseData, &responseObject)

  fmt.Println(responseObject.Page)
  fmt.Println(len(responseObject.Beer))


  var prob int
  var contI int = 45
  var contS int = 5

  for i := 0; i < len(responseObject.Beer); i++ {

     if contS > 0 {
        prob = rand.Intn(1000)
     }else{
        prob = 0
     }


     fmt.Println(prob)
     if  prob%2 == 0 {
        if(contI > 0){
           fmt.Fprintf(w, strconv.Itoa(i))
           fmt.Fprintf(w, " - ")
           if responseObject.Beer[i].ABV != "" {
              fmt.Fprintln(w, responseObject.Beer[i].ABV)
           }else if responseObject.Beer[i].IBU != ""{
              fmt.Fprintln(w, responseObject.Beer[i].IBU)
           }
```

```
            contI--
         }
      }else if  prob%2 != 0 {
         if(contS > 0){
            fmt.Fprintf(w, strconv.Itoa(i))
            fmt.Fprintf(w, " - ")
            fmt.Fprintln(w, responseObject.Beer[i].Name)
            contS--
         }
      }
   }

}

func main() {
   rand.Seed(time.Now().UnixNano())
   http.HandleFunc("/", sayhelloName) // set router
   err := http.ListenAndServe(":9126", nil) // set listen port
   if err != nil {
      log.Fatal("ListenAndServe: ", err)
   }
}
```

Conclusão: Sendo este o primeiro contato em uma API, tive que pesquisar muito e correr atrás para poder aprender e desenvolver as atividades , demonstrando assim esforço em querer fazer parte da equipe.

Dentre as dificuldades que tive pode-se dizer que as principais foram trabalhar com a API, visto que ela tomou a maior parte do tempo em aprender, uma outra dificuldade enfrentada foi na hora de criar o webserver com GO, acabou que eu criei apenas uma página bem simples em que eu escrevia a saída da tarefa no navegador.

Fiz as atividades de forma simples e rápido, peço desculpas se o código estiver bagunçado e agradeço a oportunidade de fazer parte desse processo seletivo. Espero ter mostrado minhas capacidades e desde já, foi um prazer conhecê-los.