# Exercício 04

Um serviço web que fornece a hora atual do servidor para os clientes.


- Carlos Veeck <chvmv>
- Rodrigo Sales <rdgs>
- Vinícius Seabra <vsll>

# Desafio Proposto

Implementar um Serviço Web capaz de aceitar requisições HTTP, processá-las e enviar respostas.

Desenvolvemos 2 sistemas: um usando Sockets TCP e outro utilizando Go RPC.

# Sockets:

# Código Servidor:

```go
package main

import (
    "errors"
    "fmt"
    "io"
    "net/http"
    "os"
    "time"
)

func getTime(w http.ResponseWriter, r *http.Request) {
    fmt.Printf("got /time request\n")
    currentTime := time.Now().Format(time.RFC1123)
    io.WriteString(w, fmt.Sprintf("Server Time: %s\n", currentTime))
}
func main() {

    http.HandleFunc("/time", getTime)

    err := http.ListenAndServe(":5555", nil)
    if errors.Is(err, http.ErrServerClosed) {
        fmt.Printf("servidor fechou\n")
    } else if err != nil {
        fmt.Printf("error ao inicializar o servidor: %s\n", err)
        os.Exit(1)
    }
}
```

# Fução getTime

É uma função que trata todas os HTTP requests enviados para a rota "/time".

# Dois principais parâmetros:

'w': objeto do tipo http.ResponseWriter utilizado para escrever a resposta para o cliente.

'r': objeto do tipo http.Request, contendo informações da requisição do cliente.

# http.ListenAndServe

Função utilizada para criar um servidor web HTTP na máquina local.

# Código Cliente:

```go
package main

import (
    "fmt"
    "io"
    "log"
    "net/http"
    "time"
)

func main() {
    // Marca o início do tempo de requisição
    start := time.Now()

    // Faz uma requisição GET para o servidor
    resp, err := http.Get("http://localhost:5555/time")
    if err != nil {
        log.Fatalf("Erro ao fazer a requisição: %v", err)
    }
    defer resp.Body.Close()

    // Marca o fim do tempo de requisição
    elapsed := time.Since(start)

    // Lê o corpo da resposta
    body, err := io.ReadAll(resp.Body)
    if err != nil {
        log.Fatalf("Erro ao ler a resposta: %v", err)
    }

    // Imprime a resposta
    fmt.Printf("Resposta do servidor: %s\n", body)
    fmt.Printf("Tempo de resposta: %s\n", elapsed)
}
```

# http.Get

Função utilizada para fazer uma requisição GET para o servidor.

# Go RPC:

# Código Servidor:

```go
package main
import (
    "log"
    "net"
    "net/http"
    "net/rpc"
    "time"
)
type Args struct{}
type TimeServer string
func (t *TimeServer) GiveServerTime(args *Args, reply *string) error {

    *reply = time.Now().Format(time.RFC1123)
    return nil
}
func main() {

    timeserver := new(TimeServer)

    rpc.Register(timeserver)
    rpc.HandleHTTP()

    l, e := net.Listen("tcp", ":2233")
    if e != nil {
        log.Fatal("listen error:", e)
    }
    http.Serve(l, nil)
}
```

# GiveServerTime

Função responsável por preencher uma string com o tempo atual de servidor.

# new(TimeServer), rpc.Register e rpc.HandleHTTP

Utilizada para criar, registrar e configurar um RPC Server com HTTP.

# http.Serve

Inicia o servidor HTTP para lidar com as chamadas RPC.

# Código Cliente:

```go
package main

import (
    "log"
    "net/rpc"
    "time"
)

type Args struct {
}

func main() {
    var reply string
    args := Args{}
    start := time.Now() // Marca o início do tempo de requisição
    client, err := rpc.DialHTTP("tcp", "localhost"+":2233")
    if err != nil {
        log.Fatal("dialing:", err)
    }

    err = client.Call("TimeServer.GiveServerTime", args, &reply)
    if err != nil {
        log.Fatal("arith error:", err)
    }
    elapsed := time.Since(start) // Marca o fim do tempo de requisição

    log.Printf("Resposta do servidor: %s", reply)
    log.Printf("Tempo de resposta: %s", elapsed)
}
```

# Args e reply:

Declaramos uma variável reply para armazenar a resposta do servidor e uma variável args do tipo Args para fornecer os argumentos para a chamada RPC (que, neste caso, são vazios).

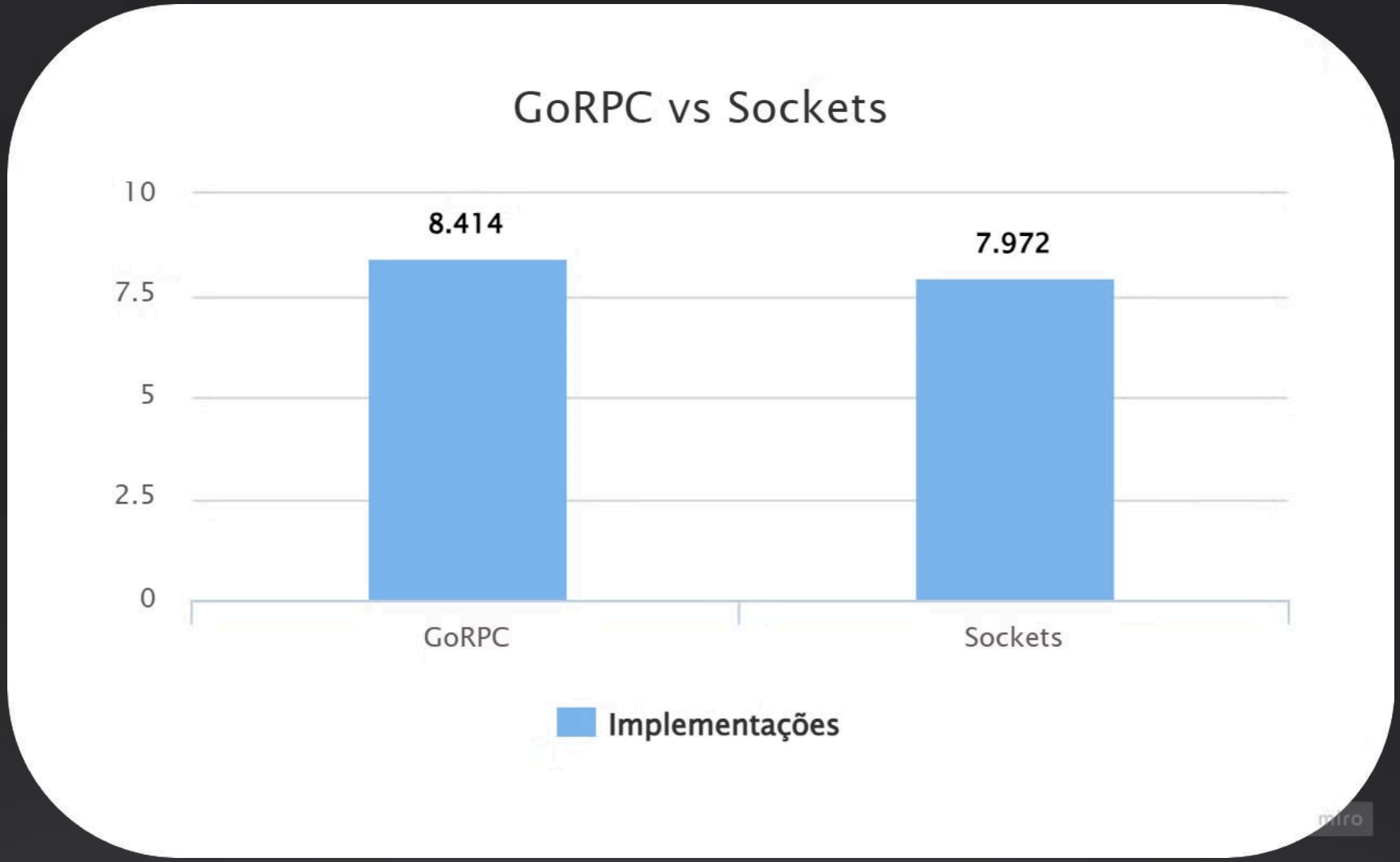
# rpc.DialHTTP:

Estabelece uma conexão com o servidor RPC.

# client.Call:

Utilizada para fazer a chamada RPC para o método desejado, nesse caso GiveServerTime.

# Análise de desempenho:




** Tempo em ms*

# Comparação de Desempenho

1 **Sockets TCP**

Maior eficiência

2 **Go RPC**

Simplicidade de implementação e abstração

3 **Escolha depende**

Requisitos do projeto e preferências da equipe

# Muito obrigado!