2016/2017

Universidade do Minho

Mestrado Integrado em Engenharia Informática

Engenharia de Sistemas de Software

Engenharia Web

1º Trabalho de Grupo (TP1)

Sistema de Monitorização de Parâmetros Ambientais

Jorge Pedro Simão, A64335
Pedro Rites Lima, A61061

13 de Março de 2017

Conteúdo

Capítulo 1

Introdução

1.1 Resumo

O presente relatório tem por objetivo a exposição do desenvolvimento do primeiro trabalho prático (TP1) da Unidade Curricular de Engenharia Web, baseado na implementação de um sistema de monitorização de parâmetros ambientais, possibilitando a simulação da comunicação entre um simulador de XDK® (cliente) e um servidor que armazena estes valores enviados pelos simuladores. As leituras efetuadas por diversos dispositivos XDK deverão ser acessíveis e alojadas em formato persistente de forma a que se encontrem disponíveis sem que o cliente perca dados enviados para o servidor. Após reflexão e interpretação do problema enunciado, o grupo de trabalho propôs-se ao desenvolvimento de uma aplicação em Ruby, com suporte de SQLite3 para armazenamento em base de dados, com o propósito de desenvolver um sistema Cliente-Servidor que visa reflectir de forma mais exacta a implementação do inicialmente proposto sendo que este relatório serve, assim, o propósito de apresentação do projeto em estado terminal, cujas funcionalidades procuram corresponder aquelas enunciadas primeiramente aquando do início do seu desenvolvimento.

1.2 Contextualização e Estrutura do Relatório

Este relatório visa explicitar e documentar o recurso à utilização da linguagem Ruby de forma a solucionar o problema proposto, aplicando rotinas aprendidas ao longo do decorrer da Unidade Curricular de Engenharia Web, pelo que procura aplicar especial foco na correcta implementação de um modelo Cliente-Servidor. Baseado no caso de estudo proposto, procura-se encontrar soluções viáveis e válidas para a correcta concretização do projecto. A estrutura deste relatório orienta-se pelo próprio desenvolvimento da aplicação proposta em enunciado, pelo que será exposto e

explicitado o projecto apresentado, a implementação de métodos cruciais, as suas especificidades e tipologia, entre outros aspectos. Este relatório contém, finalmente, uma ponderação crítica do trabalho desenvolvido assim como a comparação dos requisitos pretendidos inicialmente e os apresentados na aplicação final.

Capítulo 2

Desenvolvimento e Implementação da Solução

2.1 Conexão TCP

Após interpretação das funcionalidades especificadas em enunciado, foram depreendidas as características cruciais que especificam o sistema a ser desenvolvido. Desta forma, foi procurada a definição inicial de duas classes - *Server* e *Cliente* - estabelecendo inicialmente a comunicação entre si com recurso a *sockets* TCP e desenvolvendo o código necessário para o estabelecimento da conexão e *loop* do envio das leituras por parte do cliente.

Do lado do cliente, foi instanciado o ciclo de ligação ao cliente através de uma *socket* TCP à espera de conexões numa porta específica. O cliente ao ser instanciado conecta-se ao servidor através dessa porta, pelo que inicia o ciclo de simulação e envio de leituras simuladas de temperatura e acústica.

```
1 require "socket"
2 server = TCPServer.open(2000)
3 ...
4
5 loop {        # Servers run forever
6
7 connect = server.accept
8 line = connect.gets
9 case line.chop
```

```
10
11 ...
12
13 else
14   connect.puts "KO"
15   connect.close
16 end
```

O término da comunicação é estabelecido do lado do cliente através da *keyword* "CLOSE", sendo que é desencadeado o método *kill()* para cada uma das threads. O cálculo de leituras efetuadas é feito previamente ao fecho da ligação, sendo comunicado em ambos os terminais este valor.

```
1 puts "PARA TERMINAR ESCREVA: CLOSE"
2 if gets.chomp == "CLOSE" || gets.chomp == "close"
3   Thread.kill(temperatura)
4   Thread.kill(acustica)
5   server.puts "Sair"
6   total_leituras = @temp + @ruido
7   server.puts "TOTAL DE LEITURAS: #{total_leituras}"
8   puts "TOTAL DE LEITURAS: #{total_leituras}"
9   server.close
10 end
```

2.2 Simulação de Leituras

A simulação dos valores lidos pelos dispositivos é efetuada com recurso a *threads* responsáveis pela atribuição de novos valores no intervalo de tempo especificado. Assim, foi definido o método *simula()* invocado aquando do estabelecimento da conexão. Este método assegura a atribuição de novos valores com recurso ao método *rand()*, utilizando intervalos plausíveis no contexto de cada leitura.

De forma a estabelecer o envio de leituras nos intervalos de tempo especificados, foi utilizado ainda o método *sleep()* que possibilita "adormecer"a *thread* durante o tempo necessário antes da atribuição de novos valores. Finalmente, cada *thread* envia para o servidor a leitura simulada.

```
1 def simula(server)
```

```
2
3  @temp = 0
4  @ruido = 0
5  @latitude = rand(-90.000000000...90.000000000)
6  @longitude = rand(-180.000000000...180.000000000)
7      temperatura = Thread.new {
8      loop {
9      sleep(30)
10     @temp=@temp+1
11     value = rand(-40..80)
12     time = Time.now.getutc
13     server.puts "Temperatura #{value} #{@latitude} #{@longitude} #{time}"
14
15      }
16     }
17
18     acustica = Thread.new {
19     loop {
20     sleep(1)
21     @ruido = @ruido+ 1
22     value = rand(0..200)
23     time = Time.now.getutc
24     server.puts "Acustica #{value} #{@latitude} #{@longitude} #{time}"
25
26      }
27     }
28
29 ...
30
31     end
```

2.3 *Login* e Registo

De forma a suportar a conexão de diferentes clientes e identificação dos mesmos através um identificador único com o intuito de possibilitar posteriormente a implementação dos métodos de interrogação em SQLite, foi definida um método de interação entre o cliente e servidor previamente

ao envio de leituras, que permite a utilizador registar-se usando credenciais únicas armazenadas em formato persistente e efetuar o *login* usando estas credenciais e um identificador único gerado pelo servidor para identificar o dispositivo.

Servidor

```
when "login"
user = connect.gets.chop
pwd = connect.gets.chop
userC = logUser user,pwd
if userC != nil
  userID = userC.first
  connect.puts "OK"
  connectName = userNameG userID
  @activeUsers[userID] = UserLocation.new(connectName)
  puts "O cliente #{connectName} acabou de se Ligar"
  Thread.new {trataCliente connect,userID}

else
  connect.puts "KO"
  connect.close
end

when "registo"
  user = connect.gets.chop
  pwd = connect.gets.chop
  newID = addClient user,pwd
if newID != nil
  connect.puts "OK"
  connectName = userNameG newID
  @activeUsers[userID] = UserLocation.new(connectName)
  puts "O cliente #{connectName} acabou de se Ligar"
  Thread.new {trataCliente connect,newID}


else
  connect.puts "KO"
```

```
32    connect.close
33  end
```

Cliente

```
1  puts "O que deseja fazer? (login/registo)"
2
3  case gets.chomp
4  when "login"
5    server.puts "login"
6    puts "Introduza as suas credenciais!"
7    puts "Username:"
8    username = gets.chomp
9    server.puts "#{username}"
10   puts "Password:"
11   password = gets.chomp
12   server.puts "#{password}"
13   resposta = server.gets
14   case resposta.chomp
15   when "OK"
16
17
18     simula(server)
19   else
20
21     server.close
22   end
23
24 when "registo"
25   server.puts "registo"
26   puts "Introduza as credenciais desejadas:"
27   puts "Username:"
28   username = gets.chomp
29   server.puts "#{username}"
30   puts "Password:"
31   password = gets.chomp
32   server.puts "#{password}"
```

```
33    resposta = server.gets
34    case resposta.chomp
35    when "OK"
36
37      simula(server)
38    else server.close
39    end
40
41  else puts "Opcao invalida!"
42
43  end
```

2.4 Clientes Ativos

Os clientes ativos são armazenados numa estrutura *Hash* cuja chave corresponde ao identificador único gerado, sendo também guardadas as informações de localização GPS e nome do dispositivo. Para concretizar este processo, foram definidas as funções *logUser()*, *logOffUser()* e *trataCliente()* que asseguram a inserção das credenciais do utilizador na base de dados, a remoção deste da estrutura de clientes ativos aquando do término da conexão e o início da inserção das leituras recebidas pelo servidor na base de dados através da invocação do método *addReading()*, respetivamente.

```
1  def logUser(nameC,pwd)
2    statement = @db.prepare "SELECT ID FROM Users WHERE Name LIKE ? and Pwd
         LIKE ?"
3      statement.bind_param 1,nameC
4      statement.bind_param 2,pwd
5
6      rs = statement.execute
7      row = rs.next
8
9    rescue SQLite3::Exception => e
10     nil
11 end
12
```

```
13 def logOffUser(id,connectS)
14   nome = userNameG id
15   puts "O cliente #{nome} acabou de sair"
16   @activeUsers.delete(id)
17   connectS.close
18
19 end
20
21 def trataCliente(connect,id)
22   line = connect.gets
23
24   while line.chop != "Sair"
25
26     addReading(id,line)
27     line = connect.gets
28   end
29 contagem = connect.gets
30 puts "#{contagem}"
31 logOffUser id,connect
32
33 end
34
35 def addReading(id,line)
36   userloc = @activeUsers[id]
37   vals = line.split ' '
38   puts "id: #{id} vals: #{vals}\n"
39   @db.execute "INSERT INTO Readings ( U_id, Sensor, Valor, Latitude,
       Longitude, DateS, TimeS ) VALUES ( ?, ?, ?, ?, ?, ?, ? )", id, vals[0],
        vals[1], vals[2], vals[3], vals[4], vals[5]
40
41
42   rescue SQLite3::Exception => e
43     puts "Erro :#{e}"
44
45 end
```

2.5 Base de Dados

Para armazenar as informações dos utilizadores do sistema, assim como os valores das leituras efetuadas e posteriormente enviadas ao servidor, foi definida uma base de dados em SQLite3 - *WS.db*. Nesse sentido, foi instalada a *gem* respetiva, assim como as suas dependências de forma a garantir a funcionalidade desta *database engine*.

O servidor assegura a criação das tabelas aquando da sua instanciação, no caso destas não se encontrarem ainda definidas. A tabelas criadas - *Readings* e *Users* - estão relacionadas através da definição de uma chave estrangeira primeira tabela (*Users(Id)*) que permite a relação entre utilizadores e leituras armazenadas, facilitando a execução de algumas interrogações essenciais para a definição funcional do sistema. Segue-se a definição das interrogações que suportam as funcionalidades do sistema do lado do servidor, e os respetivos métodos que efetuam a sua execução.

```
require 'sqlite3'

# Criacao da base de dados
@db = SQLite3::Database.open "WS.db"

# Criacao das tabelas e esquema da base de dados
@db.execute "CREATE TABLE IF NOT EXISTS Users(ID INTEGER PRIMARY KEY
    AUTOINCREMENT, Name TEXT NOT NULL , Pwd TEXT NOT NULL);"

@db.execute "CREATE TABLE IF NOT EXISTS Readings(ID INTEGER PRIMARY KEY
    AUTOINCREMENT, U_id INTEGER NOT NULL, Sensor TEXT, Valor TEXT ,Latitude
    TEXT, Longitude TEXT, DateS TEXT, TimeS TEXT,FOREIGN KEY(U_Id) REFERENCES
    Users(Id));"

# Insercao de novo utilizador na base de dados - addClient()
@db.execute "INSERT INTO Users(Name,Pwd) VALUES(?,?)",nameC,pwd

# Selecao do identificador unico atraves das credenciais de utilizador -
    logUser()
@db.prepare "SELECT ID FROM Users WHERE Name LIKE ? and Pwd LIKE ?"

# Selecao do nome de utilizador atraves do identificador unico - userNameG()
@db.prepare "SELECT Name FROM Users WHERE ID LIKE ?"
```

```
19
20 # Insercao de leitura na tabela Readings - addReading()
21 @db.execute "INSERT INTO Readings ( U_id, Sensor, Valor, Latitude, Longitude
       , DateS, TimeS ) VALUES ( ?, ?, ?, ?, ?, ?, ? )", id, vals[0], vals[1],
       vals[2], vals[3], vals[4], vals[5]
22
23 # Listagem de leituras de um dispositivo especifico atraves de um
       identificador unico - listReads()
24 @db.prepare "SELECT * FROM Readings where U_id = ?"
```

Capítulo 3

Conclusão e Considerações Finais

O relatório apresentado consiste na recapitulação e explicitação do processo de implementação de uma solução para o sistema proposto em enunciado, sendo que os objectivos propostos pelo enunciado do trabalho prático foram, na nossa visão, atingidos com sucesso na totalidade.
Esta abordagem prática da unidade curricular de Engenharia Web procura a rotina de métodos de programação em Ruby, pelo que requeriu um estudo das bibliotecas de métodos disponíveis assim como a correcta definição e implementação de classes. Após um estudo das principais características desta linguagem, procurou-se uma correcta interpretação do problema proposto e visou-se o desenvolvimento de uma solução fiável que reflectisse as funcionalidades especificadas.
Considera-se que a principal dificuldade residiu na interpretação da *database engine* SQLite3, e na correcta definição das interrogações ao servidor. Ainda assim, a acessibilidade de documentação disponibilizada por esta ferramenta permitiu alcançar os objectivos propostos.
É apresentada em estado terminal um sistema capaz de responder aos requisitos enunciados, possibilitando a simulação da comunicação entre dispositivos XDK® e um servidor responsável pelo armazenamento destes valores em formato persistente. Ainda, assegura-se a disponibilidade para mais que um cliente em simultâneo, a listagem dos clientes ativos num determinado instante e leituras efetuadas, assim como o estabelecimento de uma comunicação intuitiva e que ultimamente disponibiliza a execução das principais funcionalidades deste sistema.

Capítulo 4

Bibliografia e Referências

- Tutorials Point, "Ruby Tutorial"[ONLINE]
  Available at: `https://www.tutorialspoint.com/ruby/`
  (Acedido em Março de 2016).

- Tutorials Point, "Ruby Socket Programming"[ONLINE]
  Available at: `https://www.tutorialspoint.com/ruby/ruby_socket_programming.htm`
  (Acedido em Março de 2016).

- Ruby-Doc.org, "TCPSocket"[ONLINE]
  Available at: `http://ruby-doc.org/stdlib-1.9.3/libdoc/socket/rdoc/TCPSocket.html`
  (Acedido em Março de 2016).

- Ruby Gems, "sqlite3 1.3.11"[ONLINE]
  Available at: `https://rubygems.org/gems/sqlite3/versions/1.3.11`
  (Acedido em Março de 2016).

- Ruby Forge, "SQLite Ruby"[ONLINE]
  Available at: `http://sqlite-ruby.rubyforge.org/`
  (Acedido em Março de 2016).

- Skorks, "Installing And Using SQLite With Ruby On Windows"[ONLINE]
  Available at: `http://www.skorks.com/2009/08/installing-and-using-sqlite-with-ruby-on-windows/`
  (Acedido em Março de 2016).