Lab Monitor: Monitoramento Inteligente de Laboratórios

Disciplina: CKP7500 - Sistemas Distribuídos e Redes de Comunicação

Professor: Dr. Paulo Antonio Leal Rego

Alunos: Pedro Almir Oliveira (434728), Rubens Silva (434753) e Joseane Paiva (434718)

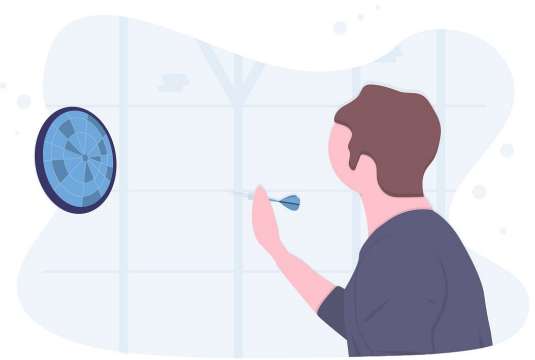
Universidade Federal do Ceará

MDCC Master and Doctorate Program in Computer Science

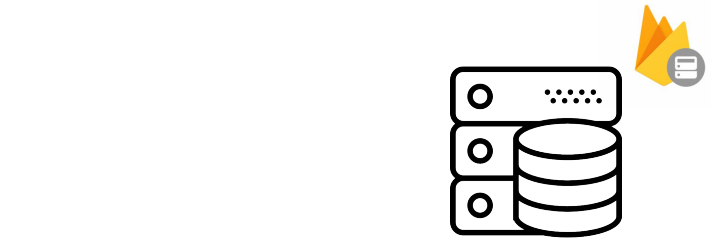
GREat Grupo de Redes de Computadores Engenharia de Software e Sistemas

Objetivo

“Desenvolver um **sistema distribuído** utilizando sockets, representação externa de dados e comunicação em grupo para **monitoramento e atuação em um ambiente inteligente**. Para representação externa dados, utilizamos **Protocol Buffer** e a comunicação em grupo foi feita com requisições **broadcast**.”



Funcionamento



Dados armazenados no Firebase

Client
Obter Dados dos Sensores
Acionar os atuadores (buzzer e led)

Sensores e atuadores com conexão a Internet

Gateway
Descoberta de Dispositivos
Responder Requisições do Cliente

protobuf
Protocol Buffers

Comandos:

- **Gateway**: java -jar gateway-sd-2.0.0.jar
- **Cliente**: java -jar client-sd-1.0.0.jar
  - sensors -ip 127.0.0.1 -port 7896
  - actuator -ip 127.0.0.1 -port 7896 -ledColor red -ledStatus on -buzzerFreq 220 -buzzerDuration 500 -buzzerTimes 5 -buzzerStatus pending

Mensagens Protobuf

- Inicialmente, criarmos mensagens para padronizar a representação de dados dos sensores <**SensorData**>;
- Depois, criamos as mensagens para organizar o acionamento dos atuadores. No nosso caso, utilizamos dois atuadores <**Led**> e <**Buzzer**>;
- Sempre que usuário precisa atuar, ele deve enviar uma requisição contendo a mensagem <**Actuators**>;

```
1  syntax = "proto2";
2  package labmonitor;
3
4  option java_package = "br.ufc.great.protoc";
5  option java_outer_classname = "LabMonitorProtos";
6
7  // [Sensors messages]
8  message Sensor {
9      required string name = 1;
10     required string value = 2;
11 }
12
13 message SensorsData {
14     repeated Sensor data = 1;
15     required string last_updated = 2;
16 }
17
18 // [Actuators messages]
19 message Led {
20     required string color = 1;
21     required string status = 2;
22 }
23
24 message Buzzer {
25     required int32 freq = 1;
26     required int32 duration = 2;
27     required int32 times = 3;
28     required string status = 4;
29 }
30
31 message Actuators {
32     optional Led led = 1;
33     optional Buzzer buzzer = 2;
34 }
```

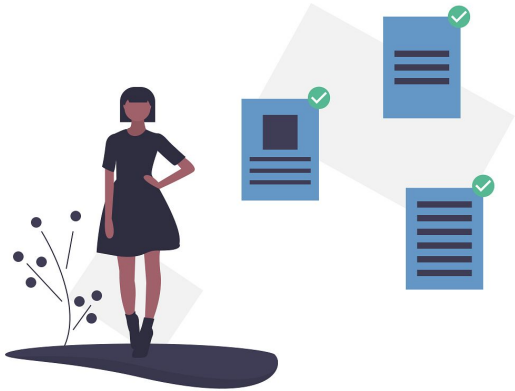
Mensagens Protobuf

- Por fim, temos a mensagem <**ClientRequest**> na qual o usuário pode informar se deseja apenas obter os dados dos sensores ou se deseja acionar os atuadores;
- Após o processamento da requisição, o gateway responde ao cliente com a mensagem <**ServerResponse**>;

```
36  // [Client Request messages]
37  message ClientRequest {
38      enum ClientRequestType {
39          GET_SENSORS_DATA = 0;
40          SET_ACTUATORS_VALUE = 1;
41      }
42      required ClientRequestType reqType = 1;
43      optional Actuators actuatorsValue = 2;
44  }
45
46  // [Server Response messages]
47  message ServerResponse {
48      enum ServerResponseType {
49          GET_SENSORS_DATA = 0;
50          SET_ACTUATORS_VALUE = 1;
51      }
52      enum ServerResponseStatus {
53          OK = 0;
54          ERROR = 1;
55      }
56      required ServerResponseType respType = 1;
57      required ServerResponseStatus respStatus = 2;
58      optional SensorsData sensorsData = 3;
59  }
```

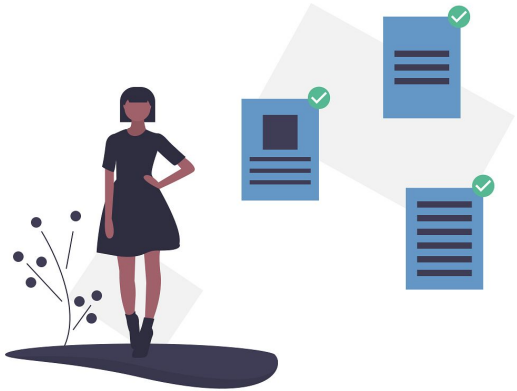
Decisões de Projeto

- Quanto aos **sensores**, decidimos utilizar sensores físicos em contraponto aos simulados para conferir um maior realismo ao trabalho. Com eles, podemos seguir com a proposta de monitoramento dos laboratórios.
    - A programação dos sensores foi feita em C;
- Quanto ao **gateway**, optamos por desenvolvê-lo em Java considerando a expertise da equipe. Os dos sensores ficam salvos no Firebase DB possibilitando acesso externo à rede do gateway;



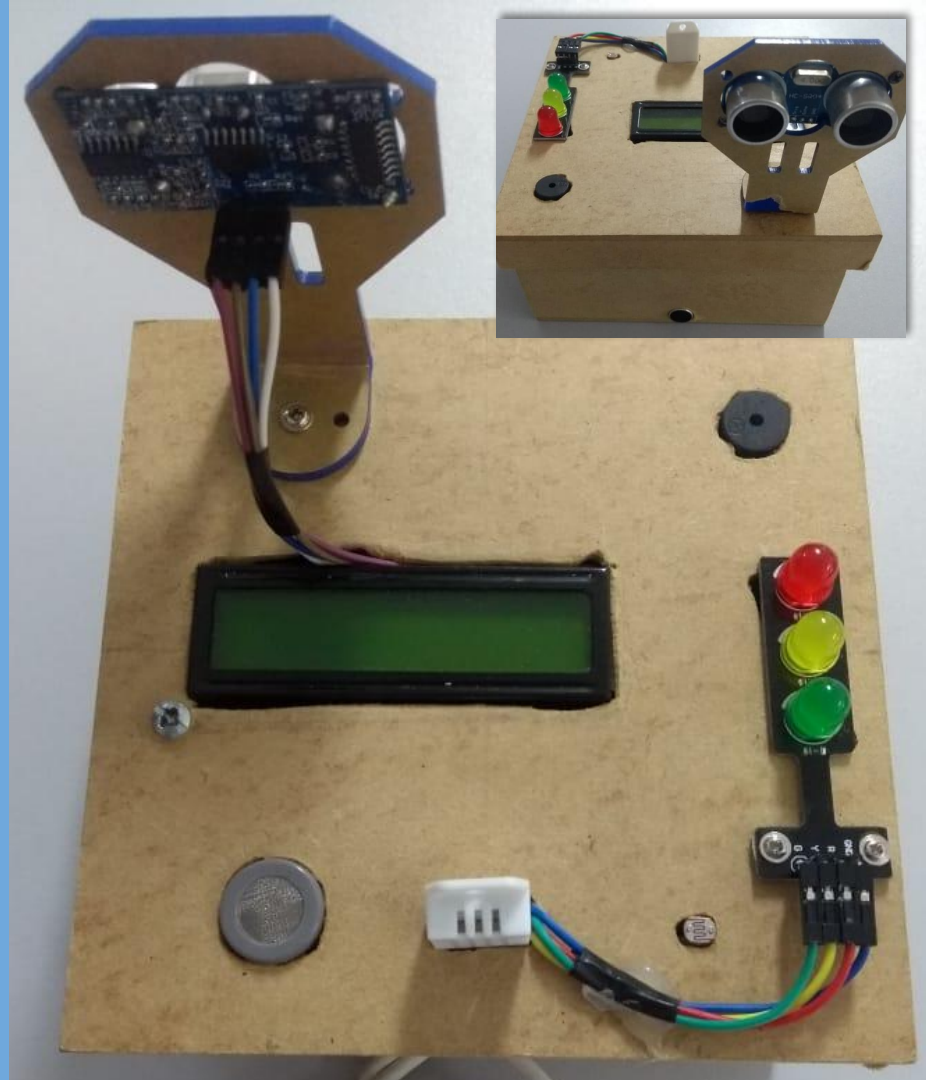
Decisões de Projeto

- Quanto ao **cliente**, inicialmente gostaríamos de desenvolver utilizando DART e Flutter. No entanto, após diversos testes com o plugin do protocol buffer para DART, tivemos que abortar esse ideia;
    - Protobuffer ainda deixa a desejar em termos de comunidade de desenvolvimento. Há poucos materiais complementares e a documentação deixa lacunas.
- Com isso, desenvolvemos um cliente Java como prova de conceito, mas que atende a todos os requisitos estabelecidos no trabalho;



Equipamento

COMPONENTE	
Esp32	DHT22 (Temperatura e Humidade)
LEDs	
Jumpers	KY38 (Ruído)
LCD + I2C	Ultrassom
Case	
LDR (Luminosidade)	
MQ7 (CO_2)	



Obrigado!

Dúvidas?