





Sistemas de Informação Distribuídos

Licenciaturas em Engenharia Informática e Informática e Gestão de Empresas
2017-2018, Segundo Semestre

Monitorização de Culturas em Laboratório

Mongo DB e Android

Identificação do grupo autor da especificação (Etapa A): G11

Número	Nome	Foto
74232	Alberto Ramos	
72948	André Carvalho	
73105	Bruno Carreira	
74486	Daniel Reis	

Identificação do grupo autor da implementação (Etapas B):

Número	Nome	Foto
62109	André Vieira	
16482	Paulo Vieira	
69565	Rodolfo Arnaldo	
73553	Rui Tomé	
65345	Tiago Rodrigues	

Instruções

Estas instruções são de cumprimento obrigatório. Relatórios que não cumpram as indicações serão penalizados na nota final.

- Podem (e em várias situações será necessário) ser adicionadas novas páginas ao relatório, mas não podem ser removidas páginas. Se uma secção não for relevante, fica em branco, não pode ser removida;
- Todas as secções têm que iniciar-se no topo de página (colocar uma quebra de página antes);
- A paginação tem de ser sequencial e não ter falha;
- O índice tem de estar actualizado;
- Na folha de rosto (anterior) têm de constar toda a informação solicitada, nomeadamente todas as fotografias de todos os elementos dos dois grupos. É obrigatório que caiba tudo numa única página;
- A formatação das “zonas” (umas sombreadas outras não sombreadas) não pode ser alterada;
- O grupo que primeiro edita o documento (Etapa A) apenas escreve na secção 1.1, e o outro grupo apenas em todas as outras secções.

Índice

1	Mongo DB.....	7
1.1	Descrição Geral do Procedimento.....	7
1.2	Estrutura da Base de Dados Mongo.....	8
1.3	Periodicidade de Leitura de Sensores e Escrita no Mongo.....	9
1.4	Estrutura da Base de Dados Sybase.....	10
1.5	Periodicidade de Leitura de Mongo e Escrita no Sybase.....	11
1.6	Triggers, SP ou eventos no Sybase (caso relevante).....	12
1.7	Utilizadores relevantes no Sybase e respectivos privilégios.....	13
1.8	Avaliação Global da Qualidade das Especificações do próprio grupo.....	14
1.9	Implementação.....	15
1.9.1	Código Mongo Implementado (dentro do java).....	15
1.9.2	Divergências face ao especificado.....	17
1.9.3	Código SQL Implementado.....	18
1.9.4	Divergências face ao especificado.....	28
2	Android e Php.....	29
2.1	Esquema da BD Lite Geral.....	29
2.2	Layout Implementado no Android.....	31

Monitorização de Culturas em Laboratório

Um laboratório de investigação de um departamento biológico necessita de um sistema para monitorizar a evolução de culturas. Nomeadamente pretende acompanhar a temperatura e humidade a que as culturas estão sujeitas, bem como detectar/antecipar potenciais problemas.

Cada cultura tem um único investigador responsável e apenas ele pode actualizar e consultar os dados de medições das suas culturas. Esta *protecção de dados* é um aspecto importante do sistema.

Sobre cada cultura são regularmente efectuadas (manualmente) medições com base num conjunto de variáveis que variam consoante a cultura. Para cada cultura o sistema conhece o intervalo de valores normal para cada variável, logo, o sistema poderá emitir alertas caso surja um valor anormal.

Por exemplo, para as culturas hidropónicas de pimento e tomate, fazem-se medições do nível de concentração de mercúrio e chumbo. Se, por exemplo, a concentração de chumbo no pimento reduzir significativamente – menos de 25 mg/litro – significa que a planta ajuda a absorver os metais indesejáveis. (*Culturas = pimento e tomate (hidropónico), variáveis = mercúrio, chumbo.*)

Outro exemplo. Numa solução onde convivem bactérias e antibióticos, se o número de bactérias cresce pouco então é porque são sensíveis ao antibiótico (logo, sabemos como as matar se forem prejudiciais). Se o número de colónias de bactérias *Bacillus subtilis*, colocadas junto de antibiótico penicilina, aumentar em mais de 30% em 2 horas é porque o antibiótico não é eficaz. (*Cultura = Bacillus subtilis, variável = penicilina.*)

Existe um sensor que periodicamente lê a temperatura e humidade no laboratório. Os dados são registados na base de dados (classe HumidadeTemperatura), e pretende-se que sejam utilizados para emitir alertas (o sistema sabe o intervalo de valores de humidade e temperatura ideal para cada cultura) e para tentar *explicar* eventuais valores anómalos de variáveis (por exemplo, “detecta-se que sempre que a temperatura desce bruscamente – mais do que 5 graus em menos de uma hora – a concentração de ferro no pimento apresenta valores anormalmente baixos”).

Cada investigador deverá ter a possibilidade de, através de um telemóvel, monitorizar a evolução da temperatura e humidade (não apenas a última leitura, mas a evolução da última hora ou horas) e receber alertas relativos a variações bruscas nos valores das variáveis das suas culturas.

É necessário guardar no sybase o registo de todas as operações de escrita sobre todas as tabelas (qua dados foram alterados/inseridos/apagados, quando e por quem) e registo de operações de consulta sobre a tabela Medições. Esse registo de alterações (*log*) é *exportado* incrementalmente (apenas informação nova) e periodicamente para uma base de dados autónoma (mysql). Através dessa base de dados (apenas de consulta) um auditor pode analisar se ocorreram utilizações abusivas dos dados (por exemplo, verificar se um investigador tentou

ler medições de culturas que não as suas, quem é que alterou limites de Temperatura de uma cultura, etc.).

Diagrama de Use Case Global

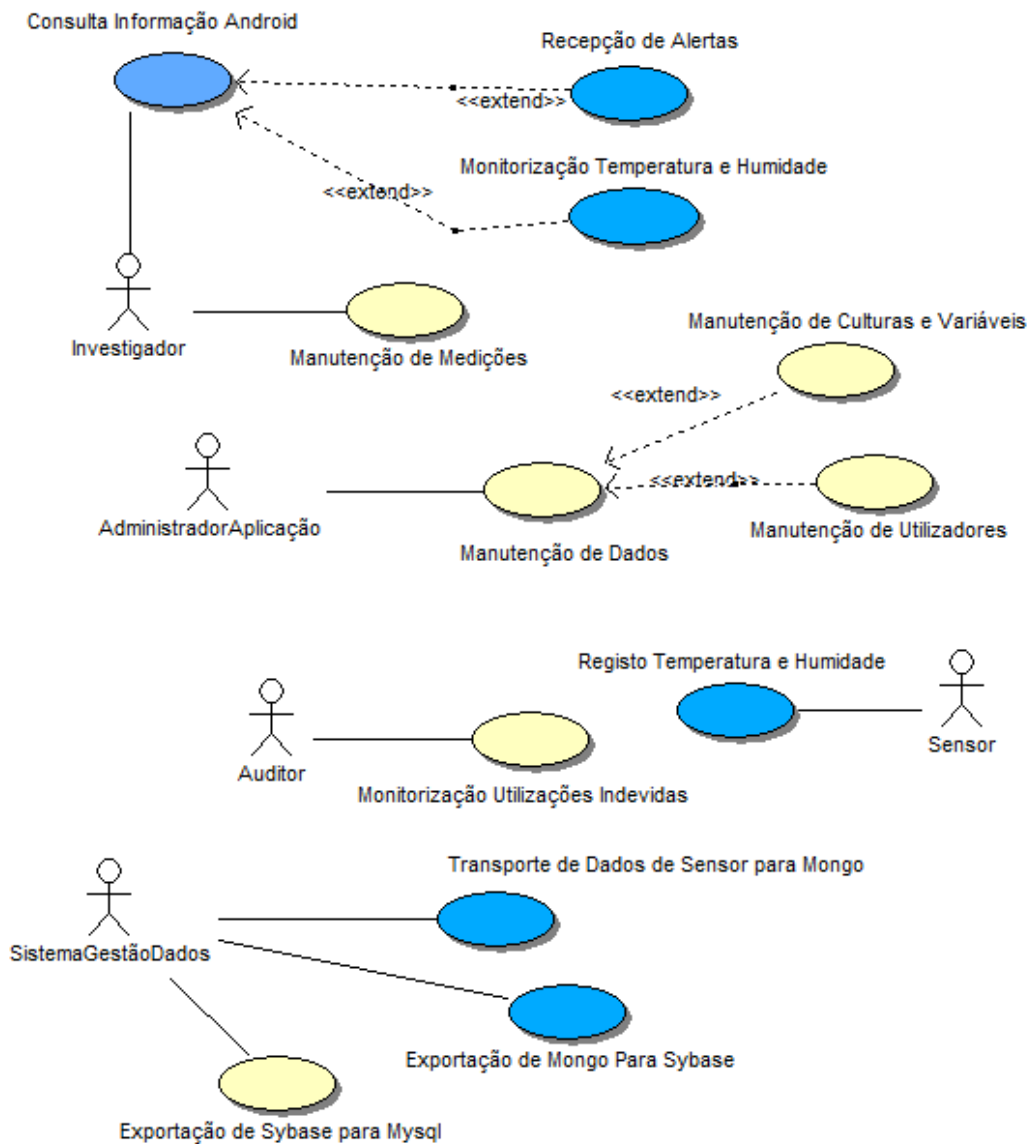
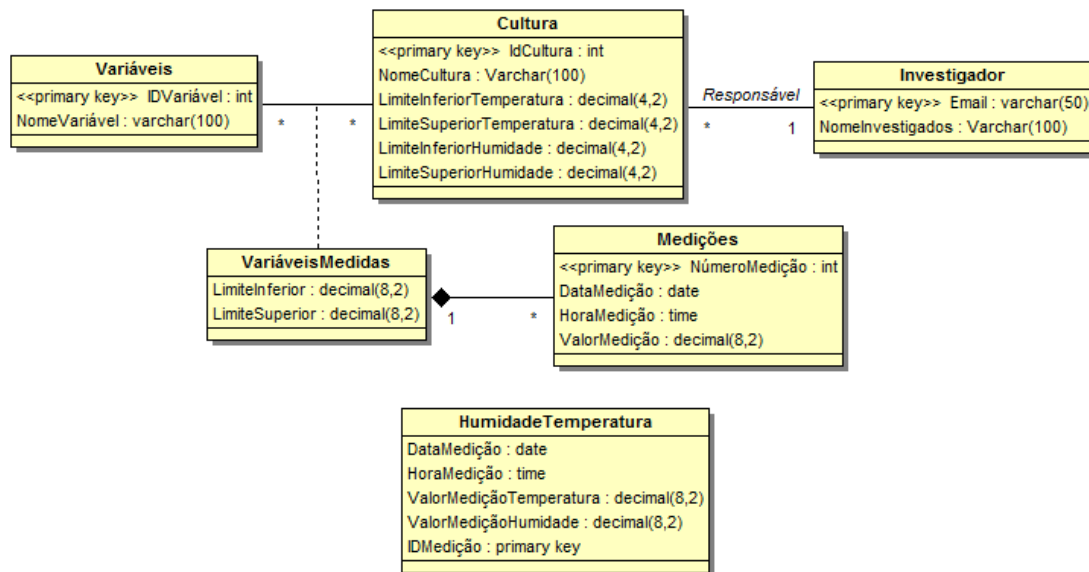


Diagrama de Classes de Suporte à Base de Dados Sql Anywhere

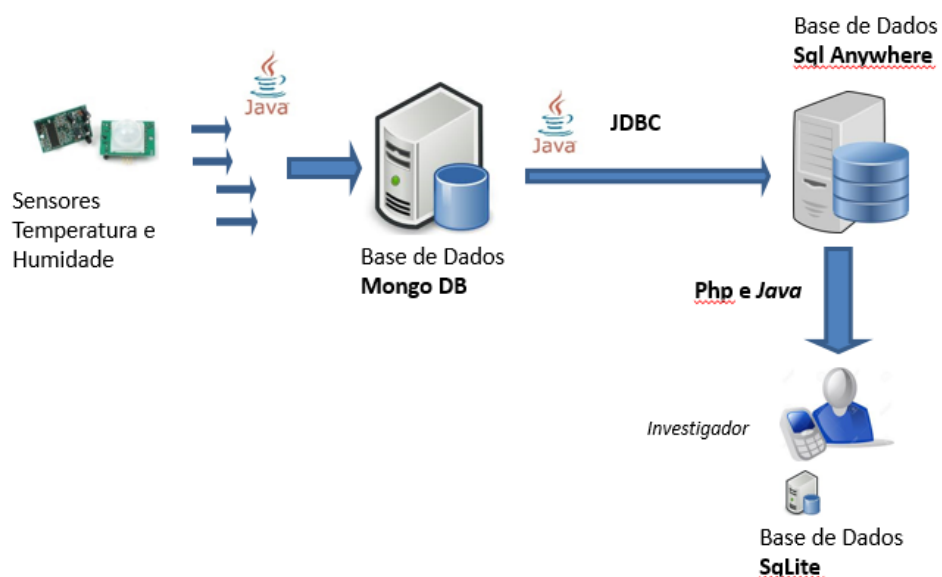


Sensor

Exemplo Mensagens

```
{"sensor": "1", "datapassagem": "2016/12/12", "horapassagem": "18:45:24"}
```

Esquema de Importação e Migração



1 Mongo DB

1.1 Descrição Geral do Procedimento

Com o objetivo de ligar os Sensores à base de dados *MongoDB*, criámos um programa em Java que serviu como uma ponte de comunicação entre as duas plataformas.

O programa que criámos, baseado no código fornecido pelo docente, possui duas classes. A primeira, denominada *Sensor_to_Java*, é uma *Thread* que está sempre à escuta da informação enviada pelos Sensores. Sempre que uma mensagem proveniente de um Sensor é recebida, é executado um método que corre uma outra *Thread*. Esta *Thread*, que chamámos de *Java_to_MongoDB*, decifra a mensagem recebida e importa-a para o *MongoDB*.

Nota: para o correto funcionamento do nosso programa, além das bibliotecas fornecidas pelo docente (jackson-all-1.9.0.jar e org.eclipse.paho.client.mqttv3-1.1.0.jar) é necessário utilizar-se uma outra:

mongo-java-driver-3.8.0-beta2.jar - necessária para fazer a ligação do Java ao *MongoDB*.

A informação do MongoDB deve ser exportada de forma incremental para a base de dados SQL Anywhere. Para tal, no programa em Java valida-se a informação importada do Mongo e, caso esta seja válida, exporta-se para a base de dados SQL Anywhere.

Nota: para possibilitar o acesso do Java ao Sybase, é necessário instalar o seguinte ficheiro “.jar”:
sajdbc4.jar

1.2 Estrutura da Base de Dados Mongo

Atribuímos o nome de “db_grupo11” à base de dados do *MongoDB*.

Esta base de dados possui uma única coleção, denominada *humidadeTemperatura*, que guarda a seguinte informação:

- Valor da Medição da Temperatura (**temperature**)
- Valor da Medição da Humidade (**humidity**)
- Data da Medição (**date**)
- Horas da Medição (**time**)

É também relevante referir que não se utiliza o método *Sharding*.

Este método é utilizado pelo *MongoDB* para suportar implementações de coleções com uma grande quantidade de dados, dividindo-os por várias máquinas. Dado que não estamos a trabalhar com uma quantidade excessiva de dados, não se justifica utilizar este método.

Utilizando o comando *find().pretty()*, apresentamos na figura abaixo algumas linhas que exemplificam a informação guardada na coleção acima descrita:

```
{
  "_id" : ObjectId("5af4438fb9df3b3350260746"),
  "temperature" : "12.0",
  "humidity" : "13.5",
  "date" : "10/05/2018",
  "time" : "16:35:10"
}
{
  "_id" : ObjectId("5af4cbb6b9df3b401477c06b"),
  "temperature" : "12.2",
  "humidity" : "13.3",
  "date" : "10/05/2018",
  "time" : "16:40:10"
}
```

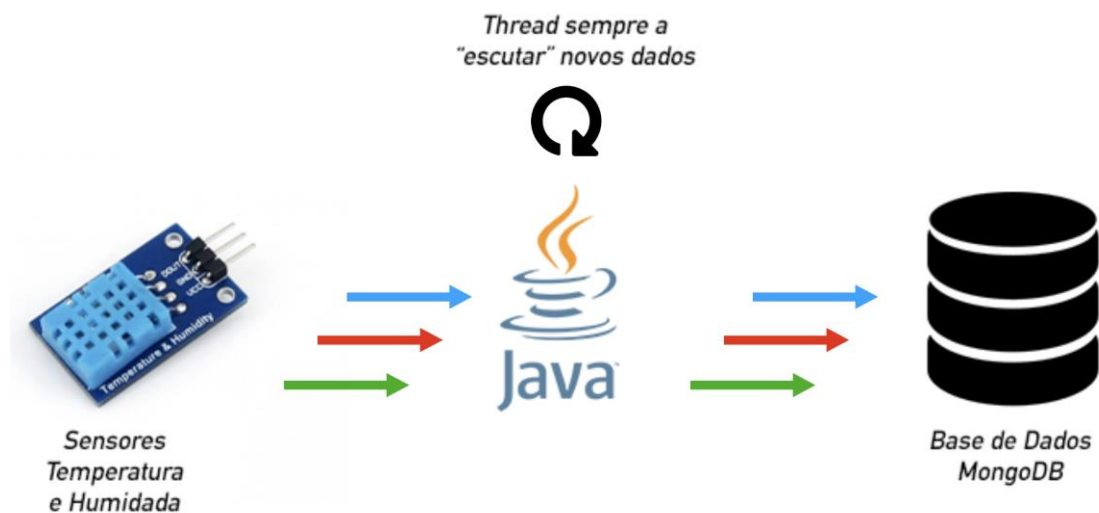

1.3 Periodicidade de Leitura de Sensores e Escrita no Mongo

O programa Java vai, em tempo real, receber a informação dos sensores e exportá-la para o *MongoDB*.

Mal o sensor efetue uma medição, envia, no momento, a informação com os dados recolhidos para o programa Java, que tem uma *Thread* sempre à espera de receber novos dados.


Sempre que a *Thread* que está à escuta recebe uma nova medição (proveniente do sensor), inicia uma outra *Thread*. Esta *Thread* temporária, converte os dados da medição recebida para um objeto do tipo *Document* e insere-os na coleção *humidadeTemperatura* na base de dados do *MongoDB*.


Nota: é também relevante referir que o QoS escolhido tem o valor 0. Escolheu-se este valor porque, como a periodicidade de envio de dados da parte do sensor é muito curta, a perda de uma medição não é muito problemática. A escolha do QoS1 foi excluída para evitar receber-se mensagens duplicadas. Assim, não é necessário validar se cada mensagem recebida é ou não duplicada. Excluímos também o QoS2 porque a robustez oferecida pelo mesmo, não justifica o acrescido processamento computacional.



1.4 Estrutura da Base de Dados Sybase

Nesta etapa, considerámos apenas relevantes as tabelas Cultura e HumidadeTemperatura (pois são as únicas tabelas que contêm informação significativa ou que serão atualizadas pelos dados do sensor)

Cultura (DBA)	
 IdCultura	integer
NomeCultura	varchar(100)
LimiteInferiorTemperatura	decimal(8,2)
LimiteSuperiorTemperatura	decimal(8,2)
LimiteInferiorHumidade	decimal(8,2)
LimiteSuperiorHumidade	decimal(8,2)
EmailInvestigador	varchar(50)

HumidadeTemperatura (DBA)	
 IdMedicao	integer
DataMedicao	date
HoraMedicao	time
ValorMedicaoTemperatura	decimal(8,2)
ValorMedicaoHumidade	decimal(8,2)

Autro-increment

Nota: decidimos eliminar a tabela Log_HumidadeTemperatura, especificada na etapa anterior, devido à sua redundância.

Apresentamos, nas figuras abaixo, algumas linhas que exemplificam os dados guardados nas tabelas acima referidas:

	IdCultura	NomeCultura	LimiteInferiorTemperatura	LimiteSuperiorTemperatura	LimiteInferiorHumidade	LimiteSuperiorHumidade	EmailInvestigador
1	1	testecult1	0,00	10,00	10,00	50,00	testeinv1@gmail.com

IDMedicao	DataMedicao	HoraMedicao	ValorMedicaoHumidade	ValorMedicaoTemperatura	
1	16	2018-05-10	17:17:44.103	13,50	12,00

1.5 Periodicidade de Leitura de Mongo e Escrita no Sybase

A leitura da informação do *Mongo* e exportação da mesma para o *Sybase*, será feita com um valor temporal configurável. Isto é, através de uma interface gráfica, é possível definir de quanto em quanto tempo é que se pretende receber novos dados.

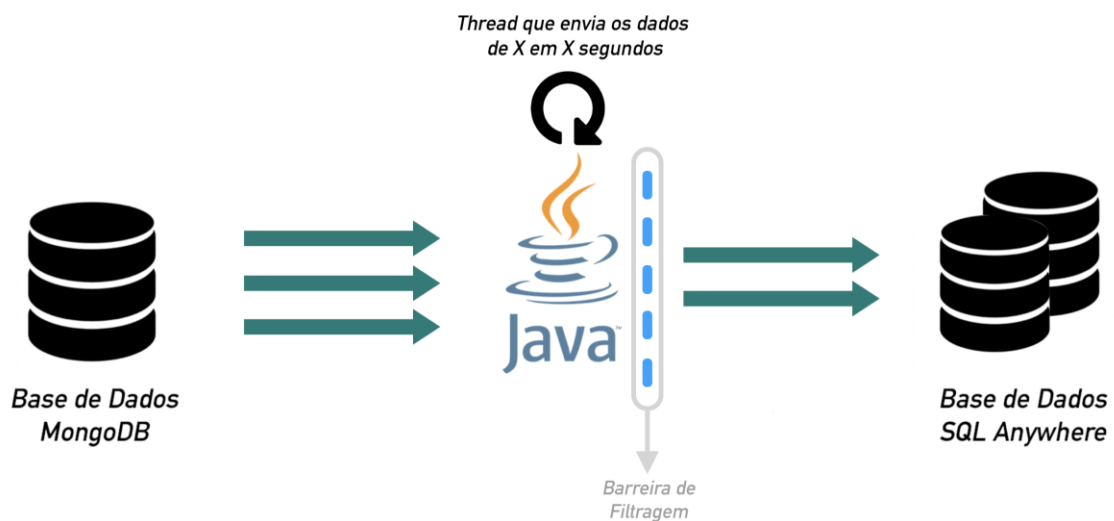
No fundo, o que será definido é a periodicidade com que o programa Java escreve a informação proveniente do Mongo no *Sybase*.

O programa em Java não só deve servir de ponte entre a plataforma *Mongo* e o *Sybase*, como também deve servir de filtro. Isto é, deve validar a toda informação proveniente do *MongoDB* funcionando, no fundo, como uma “Barreira de Filtragem” para dados corrompidos e/ou incoerências de valores.

Exemplo:

Validar se os dados da temperatura estão em decimal e se os mesmos não são demasiado discrepantes, quando comparados com dados sequentes e subsequentes.

No exemplo ilustrado na figura em baixo, um dos dados estava corrompido ou não era coerente, e acabou por ser descartado pela barreira de filtragem.



1.6 Triggers, SP ou eventos no Sybase (caso relevante)

Nome Trigger	Tabela	Tipo de Operação (I, U, D)	Evento (A, B)
tr_alert	HumidadeTemperatura	I	A

O único *trigger* especificado, será acionado após qualquer inserção na tabela HumidadeTemperatura e tem como objetivo verificar se os dados da humidade e da temperatura estão contidos nos intervalos definidos na tabela Cultura (através da análise e comparação dos campos LimiteInferiorTemperatura, LimiteInferiorHumidade, LimiteSuperiorTemperatura, LimiteSuperiorHumidade). Caso os dados não estejam no intervalo delimitado, é lançado um alerta.

1.7 Utilizadores relevantes no Sybase e respectivos privilégios

Tabela	Tipo de Utilizador
	Mongo_User
HumidadeTemperatura	Insert

Este utilizador foi criado com o objetivo de aumentar a segurança geral da base de dados do *Sybase*.

O programa em Java consegue aceder ao *Sybase* utilizando as credenciais deste *Mongo_User*. Este utilizador é somente utilizado para se inserir os dados provenientes do *MongoDB* na tabela *HumidadeTemperatura*.

1.8 Avaliação Global da Qualidade das Especificações do próprio grupo

Avaliação (A,B,C,D,E) : D

Utilize a seguinte escala:

A: - 1 - 5 valores B: 6 - 9 valores C: 10 - 13 Valores D: 14 - 17 valores E: 18 - 20 valores

Análise crítica (clareza, completude, rigor):

Relativamente à especificação fornecida pelo G11, considera-se que a mesma é, na generalidade dos nos aspetos abordados, **suficientemente clara, com as seguintes exceções:**

- Considera-se **menos claro** o modo como serão desencadeados o alertas e como se possibilitará que os mesmos sejam obtidos pelo Android, uma vez que não se procede à criação de uma tabela para o efeito no Sybase, sendo apenas referido que o trigger "tr_alert" verifica "se os dados da humidade e da temperatura estão contidos nos intervalos definidos na tabela Cultura" e que "caso os dados não estejam no intervalo delimitado, é lançado um alerta.";
- **Não está clara a pertinência** de se dispor uma thread a correr nos seguintes termos: "sempre que a Thread que está à escuta recebe uma nova medição (proveniente do sensor), inicia uma outra Thread. Esta Thread temporária, converte os dados da medição recebida para um objeto do tipo Document e insere-os na coleção *humidadeTemperatura* na base de dados do MongoDB".

No que diz respeito à **completude** e ao **rigor**, consideram-se insuficientes os seguintes aspetos:

- Regressando ao modo como está especificada a geração dos alertas, entendemos **não ser suficientemente rigoroso, nem completo**, o conteúdo da frase "caso os dados não estejam no intervalo delimitado, é lançado um alerta", até porque não está de acordo com os requisitos, onde se previa a deteção de valores anómalos, mediante observação de um padrão de medições e não apenas de valores limite por Cultura;
- Ainda nos parece **menos completo** o aspeto de não ser feita na especificação qualquer distinção entre alarmes específicos de uma Cultura e alarmes genéricos, respeitantes, por exemplo, à operacionalidade do sistema de ar condicionado.

1.9 Implementação

1.9.1 Código Mongo Implementado (dentro do java)

Como explicado na especificação o processo J2 é o responsável pela escrita dos dados recebidos do sensor na base de dados MongoDB. Depois de receber os dados do sensor, o processo J2 manipula a informação e passa de uma “string” com 4 campos para o uma “string” com 3 campos, onde os campos data e hora são aglutinados para se aproximar o mais possível do formato de um “timestamp”.

Para isso inicia-se a ligação ao mongo db:

```
//inicia a ligação ao mongodb a partir das configurações
//definidas no ficheiro j2.conf
MongoClientURI connectionString = new MongoClientURI(
    "mongodb://" + confFile.getMongoIp() +
    ":" + confFile.getMongoPort());
MongoClient mongoClient = new MongoClient(connectionString);

//conexão à base de dados "sid_2018"
MongoDatabase database = mongoClient.getDatabase("sid_2018");

//conexão à collection "sensor_messages"
collection = database.getCollection("sensor_messages");
```

Assim que se liga ao MQTT brooker e faz a subscrição ao tópico do sensor, a informação é recebida, tratada de maneira a garantir a sua validade e enviada para o MongoDB através do comando “insertOne”, como mostrado no excerto abaixo.

Escrita de uma nova entrada no MongoDB:

```
//adiciona ao objeto Document doc, os dados para que sejam
//adicionados à collection
Document doc = new Document("date", timestamp).append(
    "temperature", checkTemperature).append(
    "humidity", checkHumidity);
collection.insertOne(doc);
```

Passando para o processo J1, este estava encarregue de copiar a informação presente no MongoDB para o Sybase e se a operação fosse bem-sucedida apagar as respetivas entradas do MongoDB. Para isso recorremos ao uso de um iterador com o nome de “cursor”, que através do comando abaixo, cria um iterador a partir de todos os dados presentes no MongoDB num dado instante. É feita ainda uma cópia deste iterador para ser novamente utilizado no processo de eliminação dos dados já copiados. Garantindo assim que os dados recebidos do sensor entre o instante que fizemos o find(), e o instante em que os dados são eliminados, permanecem no mongodb para que sejam exportados na próxima iteração.

Leitura do MongoDB:

```
cursor = collection.find().iterator();
MongoCursor<Document> cursorTemp = cursor;

while (cursor.hasNext()) {
    String next = cursor.next().toJson();
    //constrói string para adicionar à query SQL para inserir
    //dados no Sybase
}
```

Eliminação de entradas do MongoDB já passadas para Sybase:

```
while (cursorTemp.hasNext()) {
    collection.deleteOne(cursorTemp.next());
}
```


1.9.2 Divergências face ao especificado

Durante a implementação da especificação apresentada, verificou-se que era mais simples implementar o procedimento de inserção dos dados da *collection* no MongoDB, com recurso à definição de um *cursor*.

Esta forma de implementação permite proceder às operações de inserção e remoção na *collection* do mongoDB, através da execução de um ciclo (definido em Java), destinado a iterar sobre todos os elementos do *cursor* e converter os recebidos de imediato no formato json (a partir da junção *.toJson*). O *cursor* contém todos os elementos existentes na *collection* do MongoDB num dado instante.

1.9.3 Código SQL Implementado

1.9.3.1 Inserção de medições no Sybase

O código SQL relevante inclui o necessário para inserir no SQL Anywhere os dados lidos da collection definida no MongoDB. Trata-se de um comando Insert no formato:

```
INSERT INTO HumidadeTemperatura
(dataHoraMedicao, valorMedicaoTemperatura, valorMedicaoHumidade)
VALUES (timestamp1, temperature1, humidity1),
        (timestamp2, temperature2, humidity2),...;
```

Sendo necessário testar o sucesso, ou insucesso, de cada operação de inserção, mediante avaliação do respetivo SQLSTATE, tal como especificado, o método para realizar esta inserção inclui a utilização da classe SQLException e do método getSQLState(), tal como abaixo descrito:

```
//recebe e a string com os dados a inserir no formato
//(ts, temperature, humidity),(ts, temperature, humidity),...
public void insertToSybase(String sqlString) throws
    SQLException {

    try {
        //constrói a query
        String sqlQuery = "INSERT INTO HumidadeTemperatura" +
            "(dataHoraMedicao,valorMedicaoTemperatura,valorMedicaoHumidade) "
            + " VALUES " + sqlString;

        //cria o objeto statement para que seja executada a query
        Statement stmt = con.createStatement();

        //executa a query
        stmt.executeQuery(sqlQuery);

        //fecha o statement
        stmt.close();
    }
    catch (SQLException sqe) {
        //caso dê erro, é disparada a exceção, terminando o
        //programa
        System.out.println("Exceção de SQL inesperada: " +
            sqe.toString() + ", sqlstate = " + sqe.getSQLState());
        System.exit(1);
    }
}
```

1.9.3.2 Adaptação dos utilizadores e permissões

Face à introdução dos alertas no sistema, foram adicionadas permissões de select aos Investigadores à *view* nomeada de “AlertasPorInvestigador”, que irá garantir a segurança na consulta de alertas para cada Investigador, assegurando que cada Investigador apenas tem acesso aos alertas pertencentes às suas culturas, como verificado em outros views.

Esta nova tabela irá também resultar em uma permissão extra de *select* para o grupo dos Administradores, que por sua vez, irá incluir o SuperAdministrador. Abaixo, encontra-se o quadro com as permissões extra acrescentadas:

Permissões	Administrador Principal	Administradores	Investigadores	MongoDB
Tabelas				
AlertaHumidadeTemperatura	Select, HardDelete	Select	-	-
Views				
v_AlertasPorInvestigador	-	-	Select	-
Store Procedures dos Alertas	-	-	-	-

Comandos SQL:

```
GRANT SELECT ON DBA.AlertasHumidadeTemperatura TO  
"Administradores";
```

```
GRANT SELECT ON AlertasPorInvestigador TO "Investigadores";
```

```
GRANT SELECT ON DBA.AlertasHumidadeTemperatura TO  
"SuperAdministrador";
```

1.9.3.3 Implementação de Triggers

Trigger para chamar os vários procedimentos de geração de alertas

```
//Trigger para verificação geral de possíveis alertas nos dados
CREATE TRIGGER "tr_ins_HumiTemp" AFTER INSERT
ORDER 1 ON HumidadeTemperatura
REFERENCING NEW AS new_name
FOR EACH ROW
BEGIN
    DECLARE noDataAlert INTEGER;
    DECLARE readErrorAlert INTEGER;

    //Chama o SP que verifica se os dados possuem informação dos
    //sensores, atualiza a tabela de alertas e retorna o resultado
    noDataAlert = CALL sp_NoDataAlert(
        "medicao" = new_name.idMedicao);

    //Se os sensores tiverem valores:
    IF noDataAlert <> 3
    THEN
        // SP que verifica se os valores dos sensores estão com
        //valores anormais, atualiza alertas e retorna o resultado
        readErrorAlert = CALL sp_ReadErrorAlert(
            "medicao" = new_name.idMedicao,
            "noDataAlert" = noDataAlert);

        //SP que verifica se os limites max e min de humidade,
        //a evolução da variação por cada cultura e atualiza alertas
        IF readErrorAlert <> 1 AND readErrorAlert <> 3
        THEN CALL sp_HumiAlert("medicao" = new_name.idMedicao)
        ENDIF;

        //SP que verifica se os limites max e min de temperatura,
        //a evolução da variação por cada cultura e atualiza alertas
        IF readErrorAlert <> 2 AND readErrorAlert <> 3
        THEN CALL sp_TempAlert("medicao" = new_name.idMedicao)
        ENDIF;
    ENDIF;
END;
```

Trigger para impedir inundação de alertas iguais num intervalo de um minuto

Este trigger tem a função de evitar que um alerta que se mantenha ativo inunde a tabela de alertas de 5 em 5 segundos. Desta forma, ele verifica se nos últimos 2 minutos existiu um alerta igual e, em caso positivo, rejeita a inserção desse alerta.

```
//Trigger para verificação geral de possíveis alertas nos dados
CREATE TRIGGER "tr_beforeInsAlertas" BEFORE INSERT
ORDER 1 ON AlertasHumidadeTemperatura
REFERENCING NEW AS new_name
FOR EACH ROW
BEGIN
  IF new_name.idCultura IS NULL
  THEN
    //Se já existir um alerta geral igual nos últimos 2 min ele
    //rejeita o atual
    IF EXISTS(
      SELECT idAlerta FROM AlertasHumidadeTemperatura
      WHERE tipoAlerta = new_name.tipoAlerta AND
        dataHora > dateadd(mi,-2,now()))
    THEN ROLLBACK TRIGGER
    ENDIF;
  ENDIF;

  //Se já existir um alerta específico igual nos últimos 2 min
  //ele rejeita o atual
  IF EXISTS(
    SELECT idAlerta FROM AlertasHumidadeTemperatura
    WHERE tipoAlerta = new_name.tipoAlerta AND
      dataHora > dateadd(mi,-2,now())
      AND idCultura = new_name.idCultura)
    THEN ROLLBACK TRIGGER
    ENDIF
  END;
END;
```

1.9.3.4 Implementação de Stored Procedures

Para a detecção e geração de alertas foram criadas 4 stored procedures, cada uma responsável por um conjunto de alertas conforme a tabela abaixo:

Tipo de Alerta	Stored Procedure responsável
NoDataAlert	sp_NoDataAlert
NoTempAlert	sp_NoDataAlert
NoHumiAlert	sp_NoDataAlert
ReadTempErrorAlert	sp_ReadErrorAlert
ReadHumiErrorAlert	sp_ReadErrorAlert
HighTempAlert	sp_TempAlert
LowTempAlert	sp_TempAlert
IncTempAlert	sp_TempAlert
DecTempAlert	sp_TempAlert
HighHumiAlert	sp_HumiAlert
LowHumiAlert	sp_HumiAlert
IncHumiAlert	sp_HumiAlert
DecHumiAlert	sp_HumiAlert

O objetivo de cada SP é o seguinte:

Nome SP	Parâmetro Entrada	Parâmetro Saída	Descrição
sp_NoDataAlert	Registro da tabela HumidadeTemperatura	Resultado da detecção de alertas: 0 – Com valores 1 – Sem temperatura 2 – Sem humidade 3 – Sem valores	Deteta e regista se existe ausência de leitura de temperatura e/ou humidade
sp_ReadErrorAlert	Registro da tabela HumidadeTemperatura e output do sp_NoDataAlert	Resultado da detecção de alerta: 0 – Valores sem erro 1 – Erro na temperatura 2 – Erro na humidade 3 – Erro nos dois	Deteta e regista se o valor dos sensores é inválido (variação > 20 na humidade e temperatura)
sp_TempAlert	Registro da tabela HumidadeTemperatura	Resultado da detecção de alerta: 0 – Sem alertas 1 – Com alertas	Deteta e regista valores limite ou variações de temperatura
sp_HumiAlert	Registro da tabela HumidadeTemperatura	Resultado da detecção de alerta: 0 – Sem alertas 1 – Com alertas	Deteta e regista valores limite ou variações de humidade

SP para verificação de existência de valores nos sensores:

```
CREATE PROCEDURE "DBA"."sp_NoDataAlert"(IN medicao INTEGER)
BEGIN
    DECLARE valorTemp DECIMAL(8,2);
    DECLARE valorHumi DECIMAL(8,2);
    DECLARE lastAlert INTEGER;

    SELECT valorMedicaoTemperatura INTO valorTemp
        FROM HumidadeTemperatura WHERE idMedicao = medicao;
    SELECT valorMedicaoHumidade INTO valorHumi
        FROM HumidadeTemperatura WHERE idMedicao = medicao;

    //Verifica se os campos humidade e temperatura são null e se
    //já existem valores null nos últimos 2 minutos
    SELECT count(idMedicao) INTO lastAlert
        FROM HumidadeTemperatura
        WHERE idMedicao > medicao - 12
        AND valorMedicaoHumidade = NULL
        AND valorMedicaoTemperatura = NULL;

    IF valorTemp IS NULL AND valorHumi IS NULL AND lastAlert = 0
    THEN
        INSERT INTO AlertasHumidadeTemperatura (
            tipoAlerta, idCultura, dataHora, valorReg)
            VALUES ('NoDataAlert', NULL, now(), NULL);
        RETURN 3;
    ENDIF;

    //Verifica se o campo temperatura somente é null e se
    //já existem valores null da temperatura nos últimos 2 minutos
    SELECT count(idMedicao) INTO lastAlert
    FROM HumidadeTemperatura
    WHERE idMedicao > medicao - 12
    AND valorMedicaoTemperatura = NULL;

    IF valorTemp IS NULL AND lastAlert = 0
    THEN
        INSERT INTO AlertasHumidadeTemperatura (
            tipoAlerta, idCultura, dataHora, valorReg)
            VALUES ('NoTempAlert', NULL, now(), NULL);
        RETURN 1;
    ENDIF;

    (...) //O mesmo código para os valores de humidade

    RETURN 0;
END;
```

SP para verificação de existência de valores de pico nos sensores:

```
CREATE PROCEDURE "DBA"."sp_ReadErrorAlert"(IN medicao INTEGER,
IN noDataAlert INTEGER)
BEGIN
    DECLARE valorTempNow DECIMAL(8,2);
    DECLARE valorTempBefore DECIMAL(8,2);
    DECLARE valorHumiNow DECIMAL(8,2);
    DECLARE valorHumiBefore DECIMAL(8,2);

    //Vai buscar o valor atual e o anterior
    SELECT valorMedicaoTemperatura INTO valorTempNow
    FROM HumidadeTemperatura WHERE idMedicao = medicao;
    SELECT valorMedicaoTemperatura INTO valorTempBefore
    FROM HumidadeTemperatura WHERE idMedicao = medicao - 1;

    (...) //O mesmo para a humidade

    //Verifica se as diferenças de temperatura e humidade
    //ultrapassam os 20 valores e recusa em caso positivo
    IF abs(valorTempNow - valorTempBefore) >= 20
        AND abs(valorHumiNow - valorHumiBefore) >= 20
        AND noDataAlert = 0
    THEN
        INSERT INTO AlertasHumidadeTemperatura
            (tipoAlerta, idCultura, dataHora, valorReg)
        VALUES ('ReadTempErrorAlert', NULL, now(), valorTempNow);
        INSERT INTO AlertasHumidadeTemperatura
            (tipoAlerta, idCultura, dataHora, valorReg)
        VALUES ('ReadHumiErrorAlert', NULL, now(), valorHumiNow);
        RETURN 3;
    ENDIF;

    //Verifica se as diferenças de temperatura somente
    //ultrapassa os 20 valores e recusa em caso positivo
    IF abs(valorTempNow - valorTempBefore) >= 20
        AND noDataAlert <> 1 AND noDataAlert <> 3
    THEN
        INSERT INTO AlertasHumidadeTemperatura
            (tipoAlerta, idCultura, dataHora, valorReg)
        VALUES ('ReadTempErrorAlert', NULL, now(), valorTempNow);
        RETURN 1;
    ENDIF;

    (...) //O mesmo para a humidade

    RETURN 0;
END;
```


SP para verificação dos limites de temperatura e humidade e a variação dos mesmos:

(implementado um SP para a temperatura e um para a humidade)

```
CREATE PROCEDURE "DBA"."sp_TempAlert"(IN medicao INTEGER)
BEGIN
    DECLARE numAmostras INTEGER DEFAULT 60;
    DECLARE idMedicaoInicial INTEGER;
    (...) //Declaração das variáveis abaixo
    DECLARE numNulls INTEGER;
    DECLARE valorTemp DECIMAL(8,2);
    DECLARE thresholdTemp DECIMAL(8,2) DEFAULT 3;

    //Procura valores limites das culturas para deteção de
    //alertas de proximidade dos limites
    SELECT valorMedicaoTemperatura INTO valorTemp
    FROM HumidadeTemperatura WHERE idMedicao = medicao;

    INSERT INTO AlertasHumidadeTemperatura(
        tipoAlerta, dataHora, idCultura, valorReg)
    SELECT 'LowTempAlert', now(), idCultura, valorTemp
    FROM Cultura
    WHERE limiteInferiorTemperatura > valorTemp - thresholdTemp;

    INSERT INTO AlertasHumidadeTemperatura(
        tipoAlerta, dataHora, idCultura, valorReg)
    SELECT 'HighTempAlert', now(), idCultura, valorTemp
    FROM Cultura
    WHERE limiteSuperiorTemperatura < valorTemp + thresholdTemp;

    //Inicia o cálculo do declive dos dados através do método de
    //regressão linear para deteção de alertas de incremento ou
    //decremento de temperatura.
    SET idMedicaoInicial = medicao - numAmostras;
    IF (idMedicaoInicial >= 0)
    THEN
        SELECT sum(valorMedicaoTemperatura),
        sum(valorMedicaoTemperatura * (idMedicao - idMedicaoInicial)),
        sum(idMedicao - idMedicaoInicial),
        sum(Power(idMedicao - idMedicaoInicial, 2))
        INTO somaTempY, somaTempXY, somaX, somaQuadX
        FROM HumidadeTemperatura WHERE idMedicao > idMedicaoInicial
        AND valorMedicaoTemperatura IS NOT NULL;

        //rejeita os valores a null
        SELECT count(idMedicao) INTO numNulls
        FROM HumidadeTemperatura
        WHERE valorMedicaoTemperatura IS NULL;
```

```

//Calcula o declive da variação de temperatura e humidade
SET decliveTemp = ((numAmostras - numNulls) *
    somaTempXY - somaTempY * somaX) /
    ((numAmostras - numNulls) * somaQuadX - Power(somaX,2));

IF decliveTemp > 0.1
THEN
    INSERT INTO AlertasHumidadeTemperatura (
        tipoAlerta, idCultura, dataHora, valorReg)
    VALUES ('IncTempAlert', NULL, now(), valorTemp);
ENDIF;

IF decliveTemp < -0.1
THEN
    INSERT INTO AlertasHumidadeTemperatura (
        tipoAlerta, idCultura, dataHora, valorReg)
    VALUES ('DecTempAlert', NULL, now(), valorTemp);
ENDIF;
ENDIF;
END;

```

1.9.3.5 Implementação de Views

Para que os pedidos sobre a tabela dos alertas fossem filtrados consoante o investigador e as suas respetivas culturas, foi criada uma view que retorna somente os alertas gerais e os alertas pertencentes a culturas desse investigador.

```
CREATE VIEW "DBA"."AlertasPorInvestigador"() AS
SELECT idAlerta, tipoAlerta, AlertasHumidadeTemperatura.dataHora,
       AlertasHumidadeTemperatura.idCultura, valorReg
FROM AlertasHumidadeTemperatura, Cultura, Investigador
WHERE AlertasHumidadeTemperatura.idCultura IS NULL OR (
       Cultura.idCultura = AlertasHumidadeTemperatura.idCultura
       AND Cultura.idInvestigador = Investigador.idInvestigador
       AND Investigador.email = user_name()
       AND Cultura.deleted = 0)
```

1.9.4 Divergências face ao especificado

No que diz respeito ao código SQL implementado, não foram assinaladas divergências relevantes face à especificação do G23. A especificação elaborada pelo G11 não foi considerada para efeitos da implementação, tendo em consideração que não trazia qualquer adição de valor nas partes que foram especificadas pelos dois grupos e, além disso, continha algumas insuficiências e aspetos menos claros, conforme descrito no parágrafo 1.10.

2 Android e Php

2.1 Esquema da BD Lite Geral

O esquema utilizado na BD do SQLite, apesar de ser um modelo relacional, não possui quaisquer relações entre as tabelas. Como se pode ver nas tabelas abaixo, estas são uma cópia semelhante das tabelas Cultura, HumidadeTemperatura e AlertasHumidadeTemperatura implementado no Sybase (com as devidas diferenças inerentes ao sistema).

Visto os dados transitarem do Sybase, não foram definidas chaves primárias nas tabelas auto-numeradas, por forma a não entrar em conflito com os ids transitados.

Nesse sentido, as três tabelas criadas são as seguintes:

Cultura:

Nome	Tipo	Descrição
idCultura	integer	Identificação única do alerta
nomeCultura	text	Nome da cultura
limiteSuperiorTemperatura	real	Limite superior da temperatura
limiteInferiorTemperatura	real	Limite inferior da temperatura
limiteSuperiorHumidade	real	Limite superior da humidade
limiteInferiorHumidade	real	Limite inferior da humidade

HumidadeTemperatura:

Nome	Tipo	Descrição
idMedicao	integer	Identificação da medição
dataHoraMedicao	datetime	Data e hora da medição
valorMedicaoTemperatura	real	Valor da temperatura
valorMedicaoHumidade	real	Valor da humidade

AlertasHumidadeTemperatura:

Nome	Tipo	Descrição
idAlerta	integer	Identificação única do alerta
tipoAlerta	text	Tipo de alerta ativado
idCultura	integer	Referência à cultura se necessário
dataHora	datetime	Data e hora de ativação do alerta
valorReg	real	Valor de temperatura ou humidade

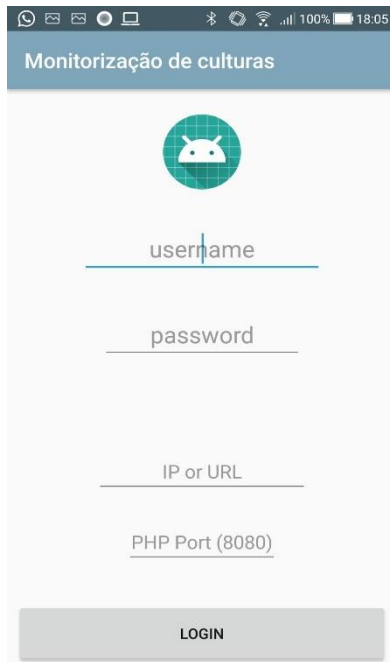
A migração de dados entre o Sybase e o SQLite é feita de forma manual ou temporizada conforme referido no capítulo abaixo.

Os pedidos dos dados são efetuados através de scripts PHP colocados num server web apache no lado do Sybase. Esses scripts encontram-se especificados no capítulo 2.3.

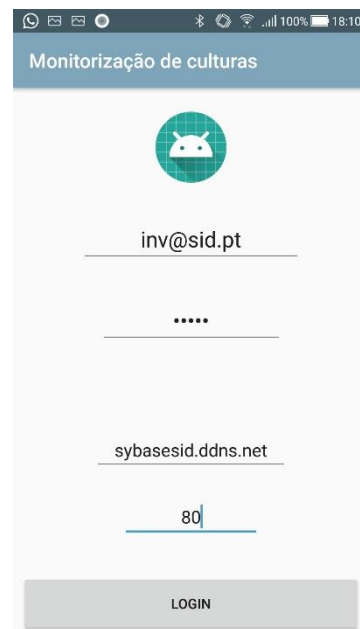
2.2 Layout Implementado no Android

De seguida será apresentado um resumo de todas as ações possíveis durante a utilização da aplicação (incluindo todos os layouts/activities).

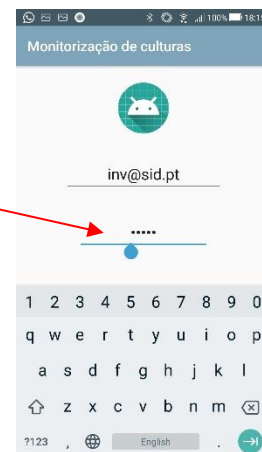
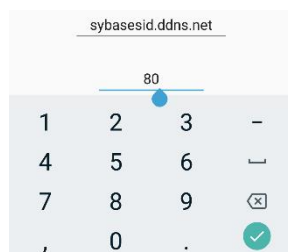
LoginActivity:



Inicialmente (quando a app é instalada), todos os campos de login permanecem vazios. No momento em que o login é bem-sucedido os dados são então guardados na memória do telemóvel.



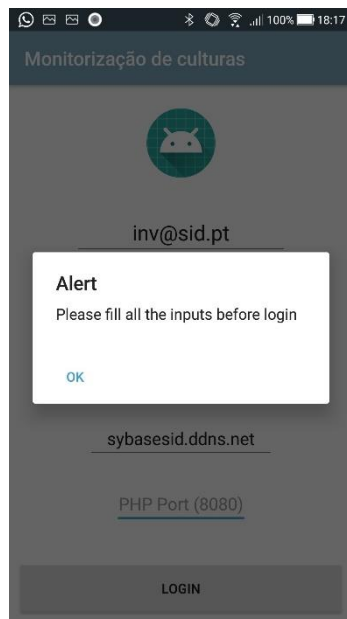
Para otimização da App, o campo password foi modificado para que fique oculto, para que seja mais segura. O campo de inserção da porta foi também mudado para que fosse apenas possível inserir números. A fonte foi aumentada para que se tornasse mais visível. Por último foi adicionado a opção a todos os inputs de passar diretamente para o campo seguinte, em vez de fazer “enter” no campo após.



Mensagens de alerta:

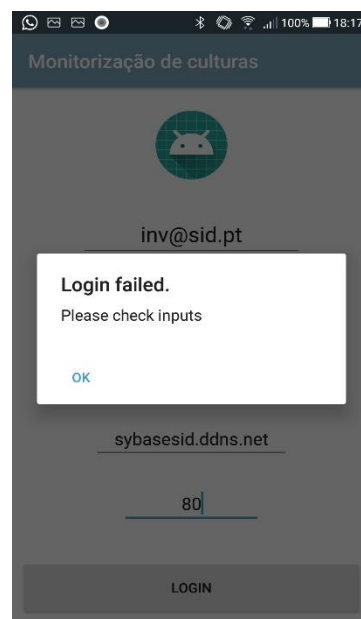
Na LoginActivity existem dois tipos de alertas que podem aparecer caso o login não seja bem-sucedido, garantindo que o utilizador não avança para a MainActivity sem um login válido, ao servidor e ao Sybase, ao contrário do que acontecia na aplicação original.

O seguinte alerta aparece caso algum dos campos de login estiver vazio:



O seguinte alerta aparece quando:

- O servidor está em baixo
- O login não é válido no Sybase



Para que o login fosse testado antes de o utilizador poder avançar para a MainActivity, foi adicionado um script chamada tryLogin.php (como explicado mais à frente), que retorna em caso de sucesso no login na base de dados Sybase o seguinte conteúdo json:

```
[{"valid":1}]
```

E em caso de insucesso:

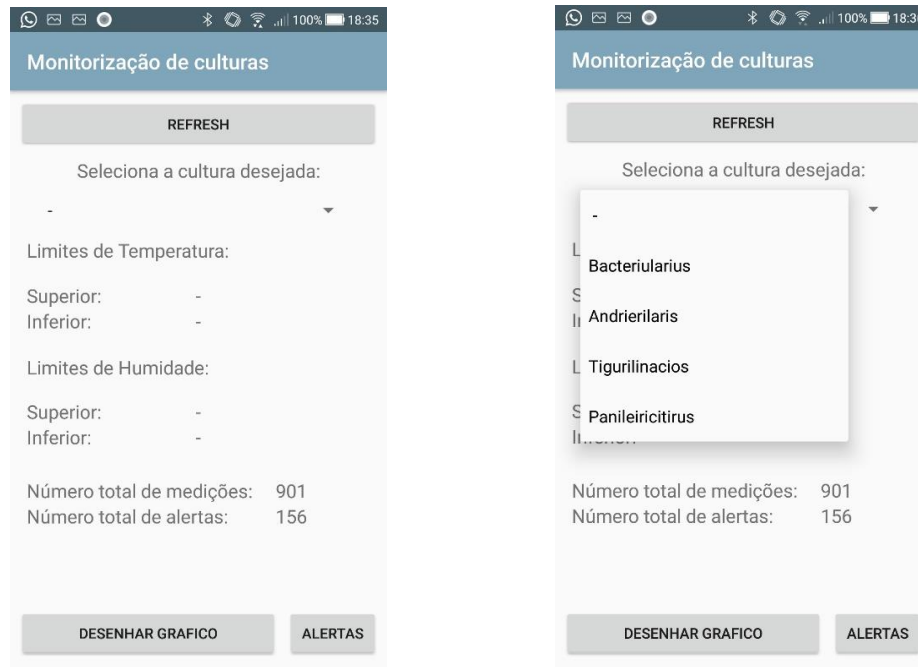
```
[{"valid":0}]
```

MainActivity:

Quando a MainActivity é inicializada, 3 pedidos são feitos ao servidor PHP:

- getHumidadeTemperatura.php
- getCultura.php
- getAlertas.php

Para estes três pedidos são enviados o username e password, para efetuar a conexão no Sybase. No caso dos alertas a query SQL pede os dados dos últimos dois dias e no caso da humidade temperatura, pede sempre os dados para o dia corrente. Depois de recebida a informação, é inserida no SQLite, e insere as culturas na select box, e as informações no ecrã:



Nas imagens anteriores, os dados de limite de temperatura e de humidade não são mostrados porque não existe uma cultura selecionada. Ao selecionar uma Cultura da selectbox, é possível verificar que a informação é atualizada automaticamente para essa cultura específica. Esta informação é retirada apenas do SQLite, não sendo preciso assim fazer quaisquer pedidos ao servidor PHP:

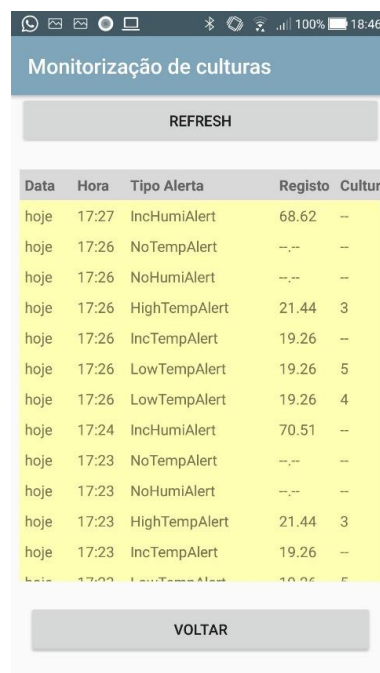


O botão de refresh atualiza automaticamente todo o conteúdo da base de dados SQLite com a informação proveniente do servidor PHP. Como na aplicação original, o botão de desenhar gráfico leva à GraphicActivity, e o botão de alertas leva à AlertasActivity.

Através da utilização de um temporizador, a aplicação faz uma atualização dos dados com o Sybase de minuto a minuto. Essa ação é semelhante ao botão refresh.

AlertasActivity:

Ao entrarmos na activity, deparamos com os alertas de uma dada cultura, ou de todas as culturas caso nenhuma esteja selecionada na selectbox:



Data	Hora	Tipo Alerta	Registo	Cultur
hoje	17:27	IncHumiAlert	68.62	--
hoje	17:26	NoTempAlert	--	--
hoje	17:26	NoHumiAlert	--	--
hoje	17:26	HighTempAlert	21.44	3
hoje	17:26	IncTempAlert	19.26	--
hoje	17:26	LowTempAlert	19.26	5
hoje	17:26	LowTempAlert	19.26	4
hoje	17:24	IncHumiAlert	70.51	--
hoje	17:23	NoTempAlert	--	--
hoje	17:23	NoHumiAlert	--	--
hoje	17:23	HighTempAlert	21.44	3
hoje	17:23	IncTempAlert	19.26	--
hoje	17:23	LowTempAlert	19.26	5

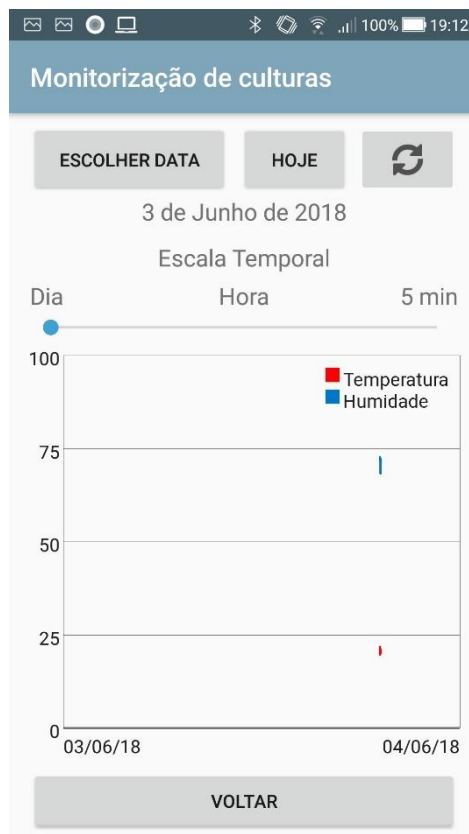
O botão de refresh faz o pedido ao server PHP, getAlertas.php atualizando assim apenas a tabela de alertas no SQLite. Toda a informação relevante ao alerta consegue mostrar toda a informação, para que não seja preciso arrastar no ecrã para os lados, ao contrário do que acontecia na versão original da app. A informação do alerta, contém, a data (“hoje para o dia de hoje”, ou o dia em que ocorreu), a hora do dia em que ocorreu, o tipo de alerta como especificado, o registo como o valor que despoletou o alerta (o valor de humidade ou temperatura), e a cultura (“--” caso nenhuma cultura selecionada).

O fundo amarelo indica que o alerta ainda não foi lido. Para que o alerta seja considerado lido, é necessário clicar nele como mostrado a imagem ao lado. Esta informação é guardada em memória, e caso saia e volte a entrar na aplicação ou/e na janela alertas, os alertas que já foram clicados anteriormente permanecessem a cinzento.



Data	Hora	Tipo Alerta	Registo	Cultur
hoje	18:56	ReadHumiErrorAlert	120.21	--
hoje	18:56	HighTempAlert	21.35	3
hoje	18:56	NoTempAlert	--	--
hoje	18:56	LowTempAlert	19.02	5
hoje	18:56	LowTempAlert	19.02	4
hoje	18:56	IncHumiAlert	68.27	--
hoje	18:56	IncTempAlert	21.38	--
hoje	18:55	ReadTempErrorAlert	71.93	--
hoje	18:55	NoHumiAlert	--	--
hoje	18:54	HighTempAlert	21.17	3
hoje	18:54	ReadHumiErrorAlert	69.07	--
hoje	18:54	IncTempAlert	20.19	--
hoje	18:54	LowTempAlert	20.19	4

GraphicActivity:



Botão escolher data: o utilizador ao clicar neste botão, vai ao layout onde pode escolher uma data específica. Como pode ser verificado pela imagem, foram bloqueados os dias que sucedem o dia corrente, não permitindo assim a sua seleção. Ao clicar ok, o conteúdo da base de dados SQLite e o conteúdo do gráfico são automaticamente atualizados.



Botão Hoje: ao clicar neste botão, o utilizador é automaticamente reencaminhado para o dia corrente e a informação no gráfico é atualizada. Esta alteração é visível, quando o utilizador escolhe outro dia através do botão “Escolher data” como explicado anteriormente.

Botão refresh: este botão, faz apenas o refresh ao gráfico, não fazendo qualquer pedido ao servidor PHP, melhorando consideravelmente a rapidez da aplicação.



É também possível alterar a escala do gráfico, para “Dia”, “Hora” e “5 min”:



2.3 PHP

De seguida será descrita muito brevemente a função de cada ficheiro:

tryLogin.php:

Como dito anteriormente, este é o ficheiro responsável por validar o login do utilizador quando tenta entrar na aplicação. Em caso de sucesso no login na base de dados Sybase o seguinte conteúdo json é impresso:

```
[{"valid":1}]
```

E em caso de insucesso:

```
[{"valid":0}]
```

Pseudo-Código PHP:

```
//tenta a conexão
startConnection();
//verifica se a conexão foi bem-sucedida
if( ! $this->dbconnection ) {
    echo [{"valid":0}];
} else {
    echo [{"valid":1}];
}
```

Databasehandle.php:

Esta class é utilizada por todas as outras classes presentes no projeto, e é o script responsável por interagir diretamente com o servidor. Para isso conta com algumas funções, como:

```
//inicia a conexão
startConnection($connection_string){ . . . }
//executa uma query e devolve o resultado em array
query($query) { . . . }
//encerra a conexão em segurança
closeConnection() { . . . }
```

getAlertas.php / getCultura.php / getHumidade_Temperatura.php:

Nestes primeiros 2 ficheiros, começa-se pela verificação se o utilizador é o utilizador “dba” ou outro (investigador). Caso seja DBA a query é feita diretamente à tabela, caso contrário, é feita à View:

getCultura.php:

```
if($this->username == "dba") {  
    $queryInput = "SELECT * FROM Cultura";  
} else {  
    $queryInput = "SELECT * FROM DBA.CulturaPorInvestigador";  
}
```

getAlertas.php:

```
$queryInput = "";  
if($this->username == "dba") {  
    $queryInput = "SELECT * FROM AlertasHumidadeTemperatura  
    WHERE dataHora " . $this->last2DaysQuery() . "  
    ORDER BY idAlerta DESC";  
} else {  
    $queryInput = "SELECT * FROM DBA.AlertasPorInvestigador  
    WHERE dataHora " . $this->last2DaysQuery() . "  
    ORDER BY idAlerta DESC" ;  
}
```

Na class getHumidade_Temperatura.php, o select é sempre feito á tabela:

```
$queryInput = "SELECT * FROM DBA.HumidadeTemperatura  
    WHERE dataHoraMedicao " . $this->last2DaysQuery() . "  
    ORDER BY dataHoraMedicao" ;
```

A função, presente em duas das três classes:

last2DaysQuery();

Retorna a string a adicionar na query, para que dê os últimos dois dias, caso o parâmetro “datepickerDate”, não seja enviado. Caso contrário retorna com a data escolhida, e dois dias anteriores, na forma:

BETWEEN '2018-06-03 00:00:00' AND '2018-06-03 23:59:59'

Pode-se ainda dizer, que todos os scripts do projeto são baseados no seguinte template, criado por nós para que fosse possível aplicar a todas as classes:

```

<?php
ini_set('max_execution_time', '0');
ini_set('memory_limit', '800M');
date_default_timezone_set('Europe/Lisbon');

define('ROOT', dirname(__FILE__) . "/");
include(ROOT . '/databasehandle.php');

class getSomething
{
    public $dbconnection = null;
    private $username;
    private $password;

    public function __construct()
    {
        if(isset($_POST['username']) && isset($_POST['password']))
        {
            $this->username = $_POST['username'];
            $this->password = $_POST['password'];

            $queryInput = "";
            if($this->username == "dba") {
                $queryInput = "SELECT * FROM ";
            } else {
                $queryInput = "SELECT * FROM DBA.";
            }

            $this->startConnection();

            if( ! $this->dbconnection ) {
                $this->dbconnection->endExecution();
            } else {
                $res = $this->dbconnection->query($queryInput);
                echo json_encode($res);
            }
        } else {
            echo "not a post, or missing parameters";
        }
    }

    private function startConnection() {
        $this->dbconnection = new databasehandle("uid=" . $this->username . ";pwd=" . $this->password);
    }
}

new getSomething();
?>

```

2.4 Scripts adicionais

2.4.1 Simulador de sensor javascript

Para dar suporte a todo o projeto, criamos ainda um pequeno script em JavaScript, que ao ser executado no browser na página do Paho onde as simulações são executadas, comporta-se como um sensor verdadeiro, enviando a data e hora no formato desejado, e com valores aleatórios de humidade e temperatura, entre valores máximos e mínimos configuráveis, a cada 5 segundos.

Para correr o script é aconselhável usar a extensão na aplicação Google Chrome, chamada de:



Depois de copiar e colar o código na extensão, basta clicar save e o site inicia a simulação dos sensores, enviando mensagens para o tópico também configurável no código.

Código JavaScript:

```
//clica automaticamente no botão connect
$("#clientConnectButton").click();

//preenche o tópico para onde irá enviar mensagens
$("#publishTopicInput").val("sid23_lab_2018");

//a cada 5 segundos corre:
setInterval(function() {

    //recebe a hora atual
    var hour = getHourString();

    //recebe o dia atual
    var date = getDateString();

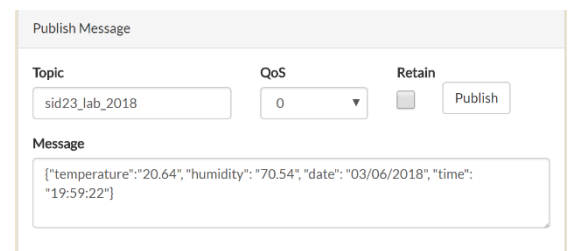
    //recebe os valores aleatórios entre o (mínimo,máximo)
    var humidity = getRandomValue(68,72);
    var temperature = getRandomValue(19,21);

    //publica a mensagem na textbox a enviar
    $("#publishMessageInput").text('{ "temperature":"' + temperature
    + '", "humidity": "' + humidity
    + '", "date": "' + date
    + '", "time": "' + hour
    + '"}');

    //imprime a mensagem enviada
    console.log($("#publishMessageInput").val());

    //clica no botão enviar para enviar a mensagem
    $("#publishButton").click();

}, 5000);
```



2.4.2 Script bat para a execução dos programas Java:

Este script, a quando dentro da pasta dos processos, onde se encontram os ficheiros j1.jar e j2.jar, e respetivos ficheiros de configuração, pode ser executado para que duas janelas sejam abertas e mantidas com a execução de cada um dos jars.

```
cd C:\Users\sid18\Desktop\WORKSPACE\project_final\processes
start cmd /k java -jar j1.jar
start cmd /k java -jar j2.jar
```

2.4.3 VM's adicionadas nesta etapa:

Para esta etapa foi adicionada ao sistema mais uma máquina virtual para o MongoDB. Esta máquina suporta os ficheiros Java para a receção dos dados do sensor, e envio para o Sybase, bem como a base de dados Mongo.

Na máquina Sybase, foi instalado o Xampp v5.6 para a execução do servidor PHP, com certas modificações para que fosse possível fazer a correta ligação ao Sybase, através dos dlls e edição do ficheiro php.ini.

Todo o restante sistema foi mantido das etapas iniciais do trabalho.