universidade de aveiro
theoria poiesis praxis

Introdução à Engenharia de Software (2019/2020)

Greenhouse Management

Vinícius Ribeiro, 82773

Marta Ferreira, 88830

Luís Sousa Rêgo, 68826

Luís Silva, 88888

Fred Avô, 79900

Produto:

O “Greenhouse Management” procura automatizar o processo de produção em estufas, provendo ao usuário dados úteis e fiáveis em tempo real, alertas e controlo automático do sistema. Para isso, o sistema inclui software para a monitorização, além de um web site para consulta e gerenciamento dos aparelhos da estufa.

A aplicação Greenhouse Management permite ao dono da empresa de estufas analisar dados relativos às condições das estufas. Este tem acesso ao log de todas as ações realizadas pelos aparelhos da estufa e aos gráficos relativos aos dados captados pelos seus sensores.

Requisitos:

- Coletar dados;
- Registo permanente numa base de dados;
- Levantamentos estatísticos (gráficos a análises) sobre os dados;
- Alertas sobre perigos iminentes (por e-mail);
- Logs de todas as ações realizadas;
- Acesso e gestão dos dados através de plataforma RESTful;
- Portal WEB

Arquitetura:

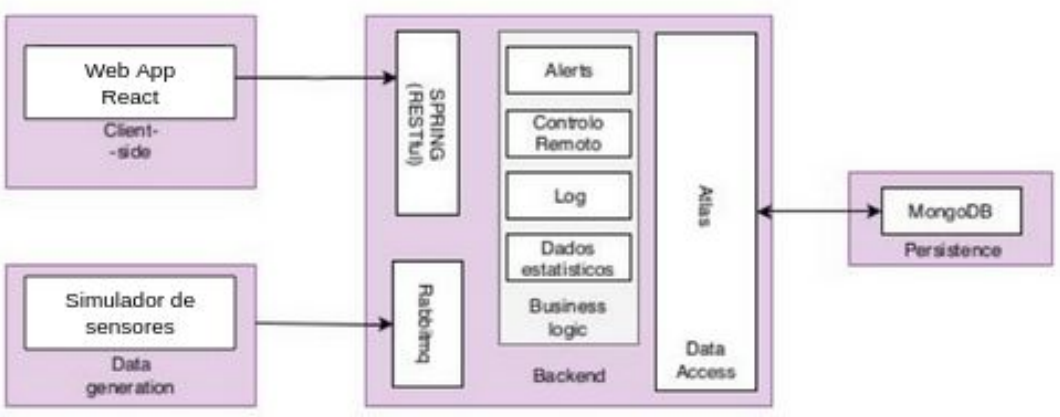

Diagrama geral da arquitetura

Simulador de sensores:

Os dados relativos às estufas que são mostrados ao usuário são gerados aleatoriamente dentro de uma gama de valores, gama esta que varia dependendo se os controlos estão ativos, isto é, a janela está aberta ou não e o aquecedor ou o sistema de rega ou a luz UV estão ligados ou não. Por exemplo, se o aquecedor está ligado, a temperatura interna gerada será crescente e vice versa.

Geração da mensagem:

O corpo da mensagem a ser enviada se encontra na classe CustomMessage.java.

```
public CustomMessage(@JsonProperty("Sensor") int sensor,
                     @JsonProperty("Temperatura interna") String tipo,
                     @JsonProperty("Dado coletado") double data,
                     @JsonProperty("Id da estufa") int estufa) {
  this.sensor = sensor;
  this.tipo = tipo;
  this.data = data;
  this.estufa= estufa;
}
```

Construtor da mensagem, utilizando o @JsonProperty

Para gerar a mensagem, o programa cria um objeto relativo ao CustomMessage, no @Component CustomMessageSender.java que verifica qual é o sensor que está a enviar o dado em questão, através do ficheiro ‘config.json ‘.

```
"sensores": [
 {
   "_id": "106",
   "tipo": "temp_int",
   "estufa": "6"
 },
 {
   "_id": "101",
   "tipo": "temp_int",
   "estufa": "1"
 },
 {
   "_id": "103",
   "tipo": "temp_int",
   "estufa": "3"
 },
```

Estrutura do config.json

```
for (JsonNode sensor : parser.path("sensores")) {
  if (sensor.path("tipo").asText().equals("temp_int")) {
     CustomMessage msg = new CustomMessage(sensor.path("_id").asInt(),
          sensor.path("tipo").asText(),
               ThreadLocalRandom.current().nextDouble(cenas.getAquecedores(sensor.path("_id").asText()).getmin(),
cenas.getAquecedores(sensor.path("_id").asText()).getmax()), sensor.path("estufa").asInt()); // temp_interna
                                    rabbitTemplate.convertAndSend(GreenHouseManagementApp.EXCHANGE_NAME,
GreenHouseManagementApp.ROUTING_KEY, msg);
   }
```

Código para gerar os dados

Imediatamente após gerar a mensagem, o RabbitMQ envia a mesma. As mensagens são geradas e enviadas a cada 5 segundos, usando a anotação @Scheduled do SpringBoot.

Receptor da mensagem:

Os dados dos sensores são recebidos no @Component CustomMessageListenner.java, na qual são tomadas ações dependendo dos valores recebidos de cada sensor. Nota-se que as ações são adicionadas ao log com o timestamp associado.

Usamos a anotação @RabbitListener para receber as mensagens em formato .json do RabbitMQ. Em seguida, o programa checa no corpo da mensagem qual o tipo do sensor que enviou a mensagem e coleta o dado, adicionando na estrutura adequada, dependendo do tipo. A estrutura escolhida para tal foi Multimap e ArrayListMultimap (com.google.common.collect), pois permite adicionar valores a chaves existentes sem sobrescrever o dado que já lá estava., útil no nosso caso, pois salvamos os dados em memória e escrevemos para a base de dados somente após certo tamanho.

O Listenner também inicia a conexão com a cloud Atlas do MongoDB, e insere os dados na mesma. Além disso, cria os métodos necessários para a API acessar os dados coletados pelo listenner.

```
@RabbitListener(queues = GreenHouseManagementApp.QUEUE_SPECIFIC_NAME)
public void receiveMessage(final Message message) throws IOException, JSONException, MessagingException {
  log.info("Dados do sensor: {}", message.toString());
  JSONObject obj = new JSONObject(new String(message.getBody()));
if (obj.getString("tipo").equals("temp_int")) {
  tempInt.put(Integer.parseInt(obj.getString("sensor")), obj.getString("data"));
  if (Double.parseDouble(obj.getString("data")) < 24) {
    if (!cenas.getAquecedores(obj.getString("sensor")).getState()) {
      cenas.getLigar().execute((cenas.getAquecedores(obj.getString("sensor"))).toString());
      Instant ts = Instant.now();
       Acoes.put(Integer.parseInt(obj.getString("sensor")), "Aquecedor ligou ás "+ formatter.format( ts ));
    }
  }
  if (Double.parseDouble(obj.getString("data")) > 26) {
    if (cenas.getAquecedores(obj.getString("sensor")).getState()) {
      cenas.getDesligar().execute((cenas.getAquecedores(obj.getString("sensor"))).toString());
      Instant ts = Instant.now();
       Acoes.put(Integer.parseInt(obj.getString("sensor")), "Aquecedor desligou ás "+ formatter.format( ts ));
    }
  }
  counter++;
```

Recepção dos dados e ação efetuada

```
if (counter > 10000) {
  try {
     for (int key : tempInt.keySet()) {
        for (int i = 0 ; i < counter ; i++) {
                                collecTempInt.updateOne(Filters.eq("sensor", key), Updates.addToSet("data",
tempInt.get(key).toArray()[i]), new UpdateOptions().upsert(true));
        }
            …
            counter = 0;
     }
```

A partir de certo limite, ocorre a escrita no banco de dados

```
public Multimap<Integer, String> getTempExt() { return tempExt; }
public Multimap<Integer, String> getTempInt() { return tempInt; }
```

Métodos acessados pela API

Controles:

Os controles foram organizados utilizando o padrão de software Command. Temos as classes para cada componente, a interface Command e a classe Invoker, que guarda e executa a lista de ações.

MongoDB:

Para este projeto, criamos um cluster no MongoDB Atlas. O banco de dados guarda batchs de dados gerados pelos sensores e os usuários do sistema.

```
private MongoClientURI connectionString = new MongoClientURI("mongodb+srv://dbAdmin:projetoies@greenhouse-vvjsz.mongodb.net/test?retryWrites=true&w=majority");
private MongoClient mongoClient = new MongoClient(connectionString);
private MongoDatabase db = mongoClient.getDatabase("GreenHouseDb");
```

Conexão ao Mongo

Para acessar o banco de dados: https://www.mongodb.com/cloud/atlas
Login: viniciusribeiro@ua.pt
Password: projetoies

API:

A API foi feita utilizando o Java, SpringBoot. A mesma fornece as operações básicas de uma plataforma REST. O código se encontra na classe DataController, que basicamente e um @RestContoller que possui vários @RequestMapping para diversos métodos.

```
@ApiOperation(value = "Get all internal temperatures from sensor")
@GetMapping(value = "/data/tempint")
public ResponseEntity<Object> getTempInt() {
  for (int key : listener.getTempInt().keySet()) {
     finalMapTempInt.put(key, listener.getTempInt().get(key).toArray());
  }
  return new ResponseEntity<>(finalMapTempInt, HttpStatus.OK);
}
```

Método get da API

E possivel adicionar sensores e edita-los pela API,que gera um ficheiro config.json a ser usado pelo gerador de mensagens e, também obter os dados gerados pelos sensores. Para mais informações sobre a API, consultar a documentação:
https://app.swaggerhub.com/apis-docs/GreenHouseManagement/green-house_management_api/1.0.0

Outros:

A classe SwaggerConfig.java e as anotações @Api servem para gerar a documentação da API.

Front-end:

Para o desenvolvimento do front-end optou-se pelo uso da *framework React* juntamente com *reactstrap* (*wrapper* que integra *bootstrap* directamente em

react) e *Apex-Charts* para a visualização de gráficos representantes da evolução dos valores recolhidos pelos vários sensores ao longo do tempo.

Estruturalmente o front-end está dividido em um router, 3 páginas, e uma barra de navegação.

- O router, implementado no ficheiro App.js, está responsável por mostrar ao utilizador a página correspondente ao url atual.
- A página *Infrastructures* mostra a lista de estufas que o cliente tem registadas na aplicação e está implementada no ficheiro infrastructures.js.
- A página *estufa* possibilita a consulta dos diversos gráficos de cada sensor existente na estufa escolhida. A estufa em questão está definida na segunda parte do url (e.g. /estufa/1 para a estufa 1).
- A página *Hardware Logs* implementa a visualização das últimas ações que cada sensor fez despoletar, como por exemplo o sensor da radiação desligar a luz artificial.

Existe ainda no front-end uma página de login que tinha como objectivo dar diferentes permissões a utilizadores com diferentes cargos, no entanto essa nuance acabou por não ser implementada e então o acesso à aplicação é indiferenciado.

Para iniciar a aplicação react do front-end executa-se o comando *$npm install* dentro da pasta “ghm-fe” para instalar as dependências necessárias e de seguida o comando *$npm start* para iniciar o servidor local, este deverá abrir no browser uma nova janela no url *localhost:3000.*