



UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

BACHARELADO EM ENGENHARIA DE SOFTWARE

Sistema de disponibilidade de aluguel de roupas

**Alexandre Dantas dos Santos
Bergony Bandeira de Melo Silva**

**Natal - RN
Julho de 2023**

Sumário

1. Introdução	3
2. Arquitetura, implementação e execução	3
2.1 Arquitetura	3
2.2 Implementação	5
2.3 Execução	9
3. Conclusão	14

1 - Introdução

Neste relatório, apresentaremos o desenvolvimento de um sistema de monitoramento para disponibilidade de aluguel de roupas em lojas especializadas. O sistema permite aos usuários se inscreverem para receber notificações sobre três estilos de roupas: esportivo, tradicional e de festa. Essas notificações são enviadas por meio da plataforma *Fiware*, utilizando-se de suas ferramentas, bem como também das ferramentas *Orion* e *Wirecloud*.

O trabalho descreve a arquitetura do sistema, a implementação do projeto, a execução e conclui com uma descrição geral do assunto tratado nos tópicos abordados.

2 - Arquitetura, implementação e execução

2.1 - Arquitetura

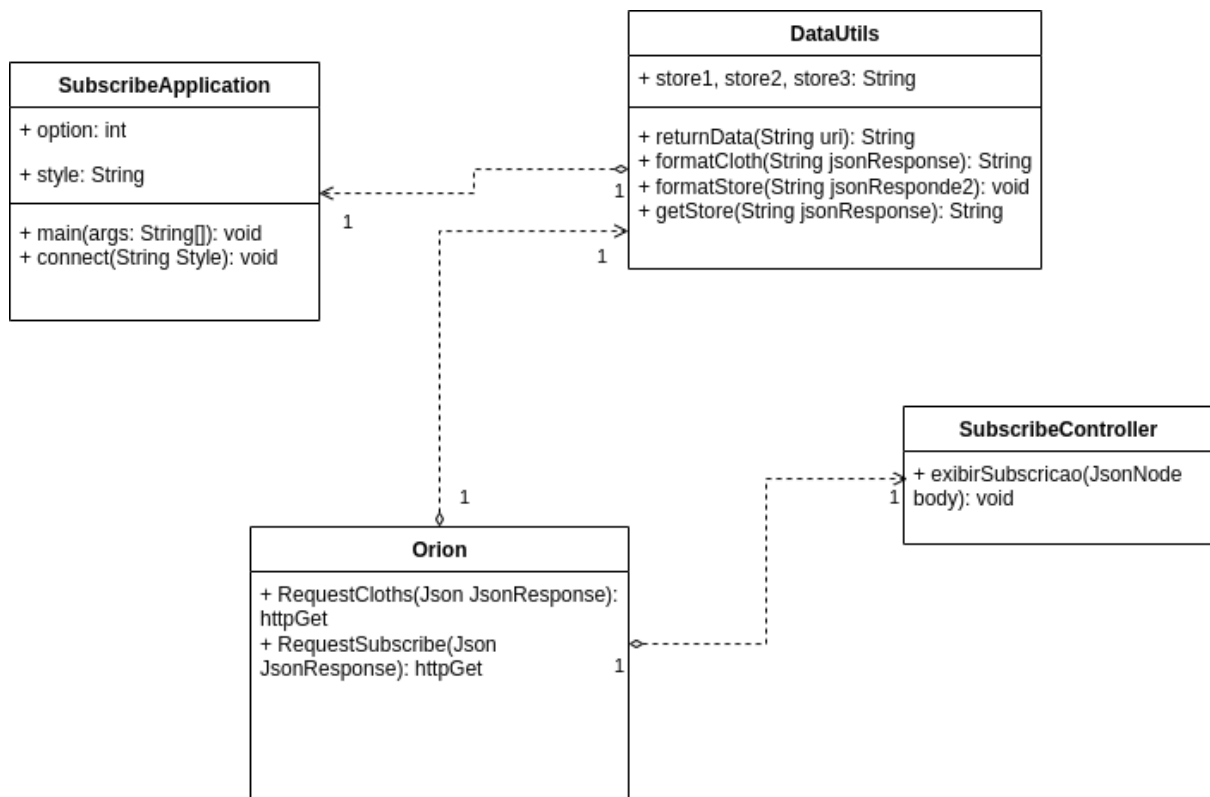
O código fonte do projeto pode ser acessado através do seguinte repositório no GitHub: <https://github.com/alexandresantosrn/FiwareProject>.

Em seguida, serão apresentados os detalhes gerais da arquitetura do projeto, bem como o seu processo de desenvolvimento e execução.

A arquitetura do projeto foi desenvolvida em cima do contexto de aplicação de *Middleware*, onde temos a integração entre a aplicação cliente utilizando-se diretamente dos serviços disponibilizados pela plataforma *Fiware*.

Para a implementação do projeto, utilizamos várias ferramentas. Inicialmente, utilizamos o *Orion* junto com o *Postman* para realizar as requisições, permitindo a criação, atualização e subscrição de entidades necessárias. Também utilizamos o *Docker* para armazenar as instâncias do *Orion* disponibilizadas nos tutoriais do projeto. Para auxiliar na construção da interface, utilizamos o *Wirecloud*. Por fim, um projeto Java (desenvolvido em *Spring Boot*) foi utilizado como cliente da aplicação, permitindo exibir as notificações para os usuários inscritos para recebimento de informações dos trajes desejados.

Na figura 1, temos a representação da arquitetura do projeto. Nesta visão temos a presença da classe: *SubscribeApplication*, responsável pelas chamadas de comunicação junto a aplicação *Orion*. Há também a presença das classes: *DataUtils* e *SubscribeApplication*. A classe *SubscribeApplication* atua no carregamento das informações do cliente, dando as opções de escolha do tipo de traje desejado. Este processo é realizado junto ao método *main()*. Em seguida é realizado o repasse das informações para o método: *connect()* que estabelece a conexão junto ao *Orion*, utilizando-se também do método *returnData()* da classe *DataUtils.java*.



(Figura 1: diagrama de classes do projeto)

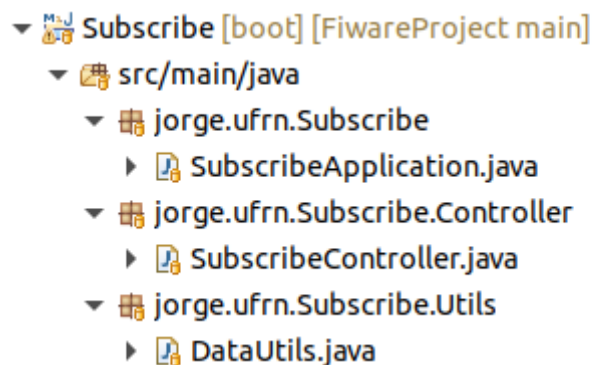
De forma bem resumida, o projeto funciona da seguinte forma. Inicialmente a aplicação se comunica com o Orion por meio das classes: *SubscribeApplication.java* e *DataUtils.java* para trazer as informações dos trajes dos tipos selecionados por meio de requisições *Http* do tipo *Get*. E em paralelo, caso novos trajes se encontrem disponíveis, a aplicação exibirá em tela acerca da disponibilidade desses trajes. Este procedimento é realizado por meio da classe: *SubscribeController.java*, junto ao método: *exibirSubscricao()*.

2.2 - Implementação

Quanto à implementação, o projeto foi desenvolvido a partir do projeto disponibilizado na turma virtual da disciplina, construído inicialmente pelo usuário: Jorge Pereira. O projeto foi também desenvolvido utilizando-se da versão 17 do Java, em conjunto com o framework *Spring Boot*.

A aplicação Java responde aos pedidos do usuário, apresentando as informações dos tipos de trajes desejados de acordo com a necessidade do cliente. Internamente, a aplicação também trata os *JSONs* que trazem as informações dos trajes presentes nas lojas.

A estrutura do projeto encontra-se detalhada na figura 2. Para execução do projeto temos a existência das seguintes classes: *SubscribeApplication.java*, *SubscribeController.java* e *DataUtils.java*.



(Figura 2: estrutura do projeto)

A classe *SubscriberConstoller.java* possui o método: *exibirSubscricao()* que é bastante importante para exibir em tela as notificações dos trajes disponíveis. O código de parte desta classe encontra-se presente na figura 3.

```

import java.text.SimpleDateFormat;

@RestController
@RequestMapping("/subscribe")
public class SubscribeController {

    @PostMapping
    public void exibirSubscricao(@RequestBody JsonNode body) {

        Date dataAtual = new Date();
        SimpleDateFormat formato = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
        String dataHoraFormatada = formato.format(dataAtual);

        String store = body.findValue("store").findValue("value").textValue();
        store = DataUtils.getStore(store);

        String msg = " \n0 traje: " + body.findValue("description").findValue("value")
            + " encontra-se disponível para aluguel na Loja: " + store + "." + "(" + dataHoraFormatada + ")";

        System.out.println(msg);
    }
}

```

(Figura 3: classe SubscribeController.java)

Em especial, conforme pode-se observar na figura 3, destaca-se o método estático: *exibirSubscrição()*, que realiza o estabelecimento da conexão com as notificações disparadas pelo *Orion* e exibe em tela as informações de localização do traje e da loja associada.

Já a classe: *SubscribeApplication.java* possui dois métodos utilizados na interface de acesso ao usuário: *main()* e *connect()*. O método *connect()* é apresentado na figura 4.

```

private static void connect(String style) throws ClientProtocolException, IOException {

    Date dataAtual = new Date();
    SimpleDateFormat formato = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
    String dataHoraFormatada = formato.format(dataAtual);

    String uriCloth = "http://127.0.0.1:1026/v2/entities?type=Cloth&q=category==" + style + ";available==true";
    String uriStore = "http://127.0.0.1:1026/v2/entities?type=Store";

    String jsonResponse = DataUtils.returnData(uriCloth);
    String jsonResponse2 = DataUtils.returnData(uriStore);
    DataUtils.formatStore(jsonResponse2);

    if (jsonResponse.equals("")) {
        System.out.println("Não há roupas disponíveis para a categoria de trajes desejada.");
        System.out.println(
            "Caso algum traje do tipo selecionado: " + style + " esteja disponível, este será listado abaixo.");
    } else {
        System.out.println(
            "Caso algum traje do tipo selecionado: " + style + " esteja disponível, este será listado abaixo.");
        String cloth = DataUtils.formatCloth(jsonResponse);
        System.out.println(cloth + "(" + dataHoraFormatada + ")");
    }
}

```

(Figura 4: método connect())

A função deste método é atuar no estabelecimento da conexão junto ao *Orion*. O método monta uma *uri* para efetivação da conexão e recebe as *strings* dos trajes e roupas através de uma requisição *HttpGet*. Em seguida, após tratamento são exibidos em tela as informações do traje do tipo selecionado.

Na figura 5 vemos parte da implementação do método: *main()* da classe: *SubscribeApplication.java*.

```
public static void main(String[] args) throws ClientProtocolException, IOException {
    SpringApplication.run(SubscribeApplication.class, args);

    int option = 100;

    try (Scanner in = new Scanner(System.in)) {

        while (option != 0 && option != 1 && option != 2 && option != 3 && option != 4) {

            System.out.println("-----");
            System.out.println("-----");
            System.out.println("Caro cliente, informe a opção de traje preferido para aluguel:");
            System.out.println("1 - Roupas esportivas.");
            System.out.println("2 - Roupas tradicionais.");
            System.out.println("3 - Roupas para festas.");
            System.out.println("0 - Sair.");
            System.out.print("Opção desejada: ");
            option = in.nextInt();

            switch (option) {

                case 0:
                    System.out.println(" \n" + "Até logo pessoal!!");
                    break;
                case 1:
                    style = "Esportivo";
                    connect(style);
                    break;
                case 2:
                    style = "Tradicional";
                    connect(style);
                    break;
                case 3:
                    style = "Festa";
                    connect(style);
                    break;
                default:
                    if (option != 0)
                        System.out.println("Opção inválida. Selecione uma das opções disponíveis!");
            }
        }
    }
}
```

(Figura 5: método *main()* - parte inicial)

Neste trecho são utilizadas as variáveis para utilização junto ao menu de perguntas do usuário: *option* e *style*. A variável *style* armazena o tipo de traje desejado pelo usuário.

Este método contém também o esqueleto do menu de perguntas apresentado para o cliente, onde são exibidas as opções de seleção disponíveis para o usuário. Cada opção de escolha é trabalhada com a estrutura *switch(case)*, onde é realizada também a conexão com o método: *connect()* já descrito anteriormente.

O projeto possui também a classe auxiliar: *DataUtils.java* necessária para tratar algumas informações desenvolvidas no projeto.

Esta classe possui uma estrutura mais complexa, onde temos a existência de diversos métodos úteis. Em especial destaca-se o método: *returnData()* necessário pelo retorno das informações do *JSON* a partir das requisições realizadas junto ao *Orion*.

```
public static String returnData(String uri) throws ClientProtocolException, IOException {
    String jsonResponse = "";

    HttpClient httpClient = HttpClients.createDefault();
    HttpGet httpGet = new HttpGet(uri);
    HttpResponse response = httpClient.execute(httpGet);
    HttpEntity entity = response.getEntity();
    jsonResponse = EntityUtils.toString(entity);

    return jsonResponse;
}
```

(Figura 6: método returnData())

Outro método bastante importante junto a classe é o método: *formatCloth()*. Tal método tem um papel importante em formatar as informações da roupa para serem exibidas em uma mensagem para o cliente.

```
public static String formatCloth(String jsonResponse) throws ClientProtocolException, IOException {
    String traje = "";
    String store = getStore(jsonResponse);
    String word = jsonResponse.substring(184);

    for (int i = 0; i < word.length(); i++) {
        char caractere = word.charAt(i);
        if (caractere != ' ') {
            traje += caractere;
        } else {
            break;
        }
    }

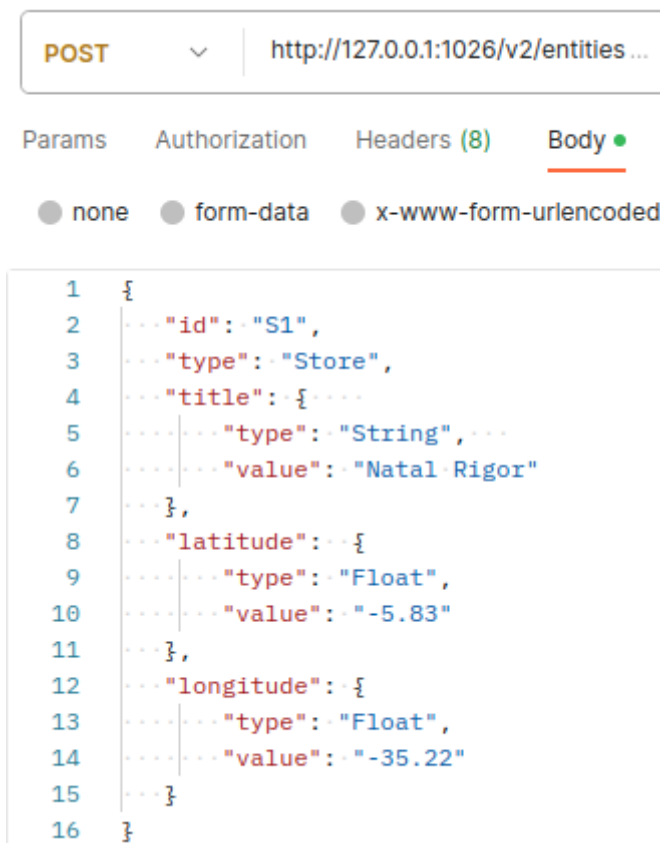
    String msg = "0 traje: " + traje + " encontra-se disponível para aluguel na Loja: " + store + ".";

    return msg;
}
```

(Figura 7: método formatCloth())

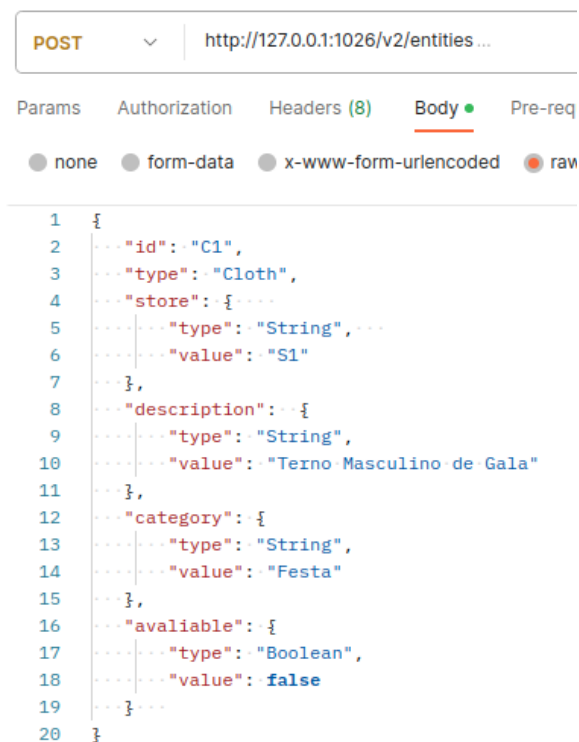
2.3 - Execução

O exemplo de execução abaixo, diz respeito ao registro de uma loja, contendo duas roupas do tipo: Festa. Para simulação, as requisições para o back-end serão realizadas por meio da ferramenta: *Postman*. Nas figuras a seguir serão apresentadas as requisições que realizam os registros para as lojas e roupas, com informações detalhadas do seu corpo e dados.



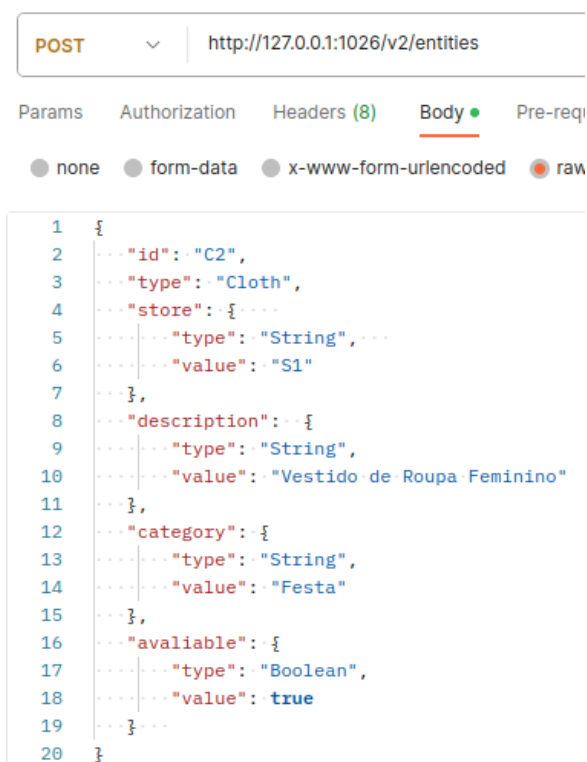
(Figura 8: representação do cadastro da loja)

A seguir, são detalhados os registros das roupas:



```
1  {
2    "id": "C1",
3    "type": "Cloth",
4    "store": {
5      "type": "String",
6      "value": "S1"
7    },
8    "description": {
9      "type": "String",
10     "value": "Terno Masculino de Gala"
11   },
12   "category": {
13     "type": "String",
14     "value": "Festa"
15   },
16   "avaliabile": {
17     "type": "Boolean",
18     "value": false
19   }
20 }
```

(Figura 9: representação do cadastro da roupa)



```
1  {
2    "id": "C2",
3    "type": "Cloth",
4    "store": {
5      "type": "String",
6      "value": "S1"
7    },
8    "description": {
9      "type": "String",
10     "value": "Vestido de Roupa Feminino"
11   },
12   "category": {
13     "type": "String",
14     "value": "Festa"
15   },
16   "avaliabile": {
17     "type": "Boolean",
18     "value": true
19   }
20 }
```

(Figura 10: representação do cadastro da roupa)

Ao executar a aplicação, será apresentado o menu de interação junto ao cliente. Inicialmente, será apresentado para o usuário as opções de seleção do tipo de traje, conforme descrito na figura 17.

```
-----  
-----  
Caro cliente, informe a opção de traje preferido para aluguel:  
1 - Roupas esportivas.  
2 - Roupas tradicionais.  
3 - Roupas para festas.  
0 - Sair.  
Opção desejada:
```

(Figura 11: tela de seleção do tipo de traje)

Caso o usuário selecione a opção: “0” será dada a opção para este sair da aplicação. Já as opções: 1, 2 e 3 correspondem aos tipos de trajes desejados. Neste caso, correspondente às categorias: Esportivo, tradicional e festa.

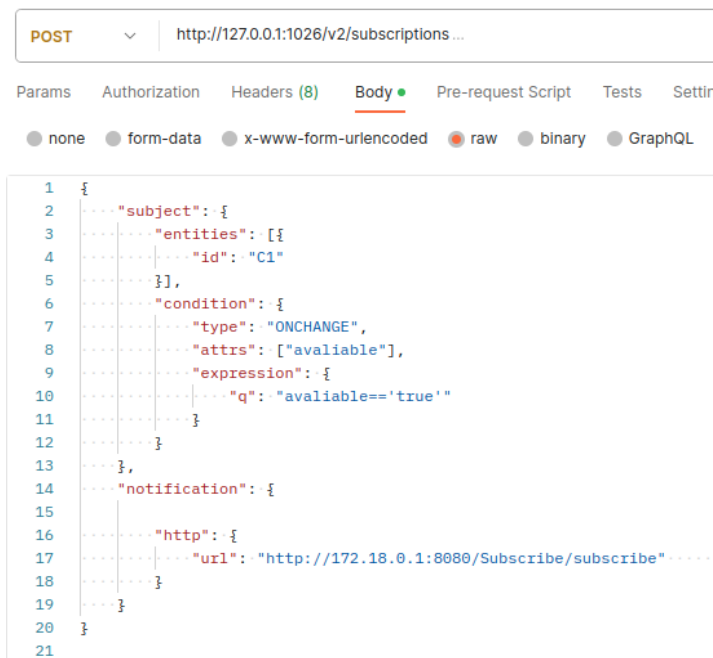
Após seleção do tipo de traje desejado, serão apresentados os trajes disponíveis para o tipo de roupa desejada. Este retorno é realizado a partir de uma requisição *httpGET* sobre o serviço disponibilizado pelo *Orion*.

```
Caro cliente, informe a opção de traje preferido para aluguel:  
1 - Roupas esportivas.  
2 - Roupas tradicionais.  
3 - Roupas para festas.  
0 - Sair.  
Opção desejada: 3  
Caso algum traje do tipo selecionado: Festa esteja disponível, este será listado abaixo.  
0 traje: Vestido de Noiva Feminino encontra-se disponível para aluguel na Loja: Natal Rigor.(01/07/2023 20:07:07)
```

(Figura 12: tela com apresentação do tipo de traje informado)

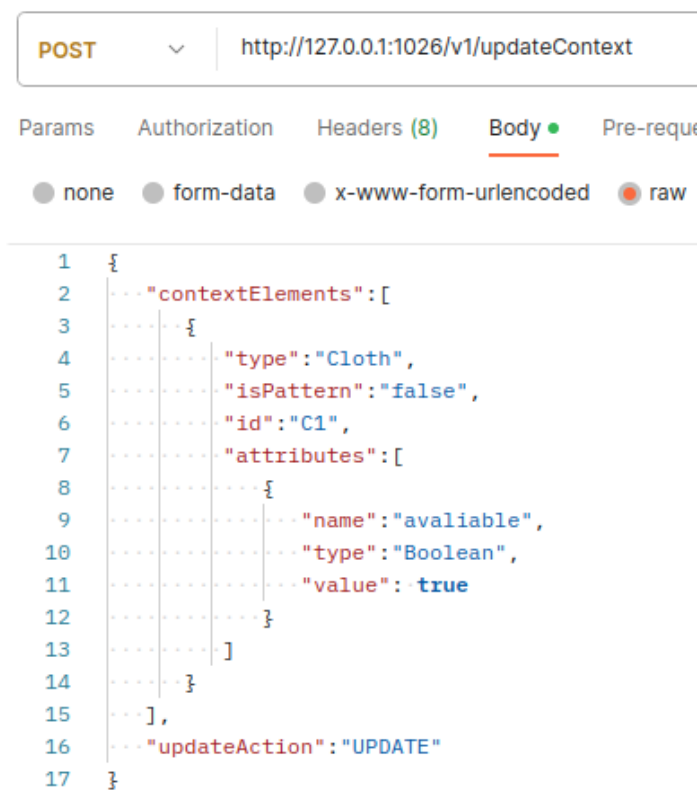
De forma assíncrona, caso um novo traje encontre-se disponível para aluguel, o sistema irá notificar para o *user* em tela da disponibilidade daquele traje. Esta notificação é “disparada” quando algum traje tem seu status alterado para disponível = true. Tal alteração é realizada por uma operação manual disponibilizada junto ao *Postman*.

A seguir, são detalhados os registros de subscrição e atualização das roupas:



```
1 {
2   "subject": {
3     "entities": [
4       {
5         "id": "C1"
6       }
7     ],
8     "condition": {
9       "type": "ONCHANGE",
10      "attrs": ["avaliabile"],
11      "expression": {
12        "q": "avaliabile==true"
13      }
14    },
15    "notification": {
16      "http": {
17        "url": "http://172.18.0.1:8080/Subscribe/subscribe"
18      }
19    }
20 }
21 }
```

(Figura 13: representação da subscrição dos trajes)



```
1 {
2   "contextElements": [
3     {
4       "type": "Cloth",
5       "isPattern": "false",
6       "id": "C1",
7       "attributes": [
8         {
9           "name": "avaliabile",
10          "type": "Boolean",
11          "value": true
12        }
13      ]
14    }
15  ],
16  "updateAction": "UPDATE"
17 }
```

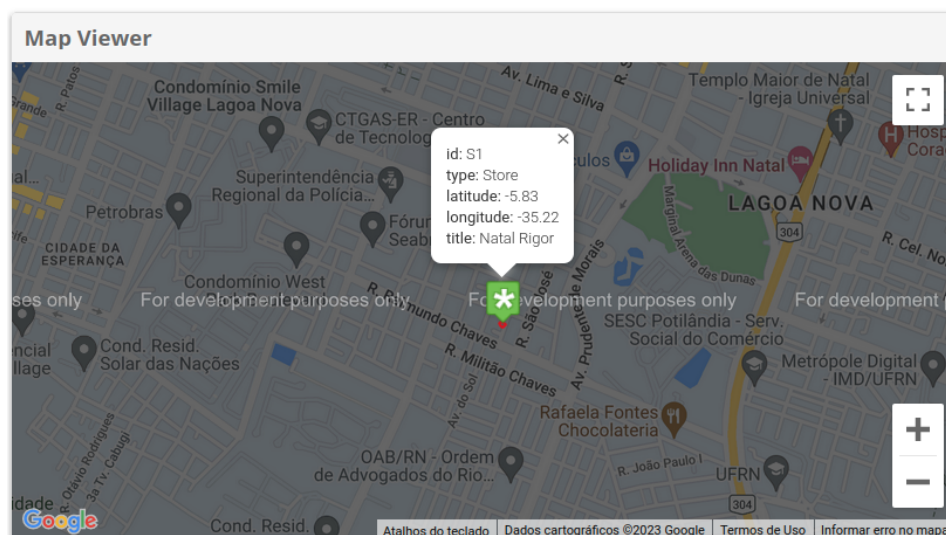
(Figura 14: representação da atualização dos trajes)

A seguir, a tela que exibe a notificação de disponibilidade do traje disparada para o usuário:

```
-----  
Caro cliente, informe a opção de traje preferido para aluguel:  
1 - Roupas esportivas.  
2 - Roupas tradicionais.  
3 - Roupas para festas.  
0 - Sair.  
Opção desejada: 3  
Caso algum traje do tipo selecionado: Festa esteja disponível, este será listado abaixo.  
O traje: Vestido de Noiva Feminino encontra-se disponível para aluguel na Loja: Natal Rigor.(01/07/2023 20:07:07)  
2023-07-01 20:13:10.753 INFO 9321 --- [nio-8080-exec-1] o.a.c.c.c.[./localhost].[/Subscribe] : Initializing Spring DispatcherServlet  
2023-07-01 20:13:10.753 INFO 9321 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'  
2023-07-01 20:13:10.756 INFO 9321 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 3 ms  
  
O traje: "Terno Masculino de Gala" encontra-se disponível para aluguel na Loja: Natal Rigor.(01/07/2023 20:13:11)
```

(Figura 15: tela de apresentação dos trajes)

Por fim, através do *Wirecloud* é possível também obter os dados da localização da loja, por meio do seguinte mapa interativo:



(Figura 16: tela de apresentação da loja no mapa)

Conclusão

O projeto aqui desenvolvido proporcionou um melhor aprofundamento sobre o entendimento no uso das ferramentas utilizadas no contexto do IOT, em especial sobre o uso da plataforma *Fireware* e de suas especificações: *Orion* e *Wirecloud*.

Foi encontrada uma pequena dificuldade na configuração dos ambientes, em especial no uso do docker, que funcionou apenas com a mudança de uso para uma segunda máquina. A nível de programação houve uma dificuldade na estruturação do programa, em especial na modelagem da sua arquitetura, e também na manipulação dos JSONs utilizados pela aplicação.