



Universidade Autónoma de Lisboa  
Departamento de Ciências e Tecnologia

# Projeto

## Sistemas Distribuídos e Paralelos

2020/2021

- **Docente:** André Sabino
- **Curso:** Licenciatura em Engenharia Informática
- **Turno:** Diurno
- **Trabalho elaborado por:**

Bruno Silva                      30003769

David Monteiro                30003043

## Introdução

Este relatório tem como objetivo apresentar a construção de uma aplicação para gestão de transporte de itens de uma empresa, de forma a consolidar a matéria lecionada durante este semestre, na unidade curricular de Sistemas Distribuídos e Paralelos.

Neste relatório está relatado e descrito as decisões tomadas para a construção desta aplicação, e alguns conceitos que são importantes para demonstrar a importância dos Sistemas Distribuídos e Paralelos assim como a sua aplicação prática.

Para este projeto foi utilizada a linguagem de programação Java, de consulta estruturada SQL e linguagem de marcação HTML, com recurso à IDE IntelliJ IDEA Ultimate 2020.3, ao SGBD PostgreSQL, ao servidor de aplicações JAVA EE Wildfly e por fim a Docker *containers*.

Este projeto modela e implementa um sistema distribuído com os seguintes nós:

- Base de dados para persistência de informação;
- API REST para disponibilização da funcionalidade;
- Aplicação web, i.e., interface HTML para interação com o utilizador.

## Arquitetura Geral

Como se verifica na arquitetura (Ver Figura 1 – Diagrama da Arquitetura Geral), existe uma Aplicação Web para interagir com utilizador, que deve consultar a informação de negócio de forma a fazer pedidos/consultas (GET) à API REST, no entanto, não pode estar diretamente ligada à base de dados, pois o principal objetivo é criar uma aplicação que dependa da API REST. Depois de fazer os pedidos para consultar a informação de negócio a API REST devolve a informação requerida para a aplicação WEB e esta mostra ao cliente o conteúdo em HTML.

O único ponto de acesso à base de dados é pela API REST, através de JDBC.

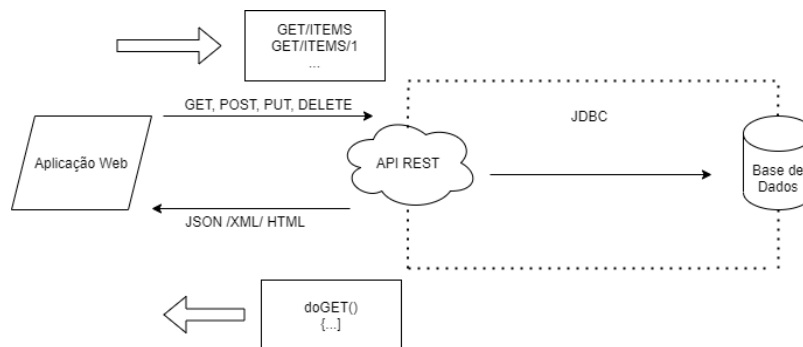


Figura 1- Diagrama da Arquitetura Geral

## Base de Dados

É um conjunto de dados relacionados de acordo com uma ou várias regras e um objetivo específico, consiste numa coleção de dados estruturados e armazenados de forma persistente.

Um sistema de gestão de base de dados (SGBD) é uma aplicação ou um conjunto de aplicações informáticas utilizadas para definir, aceder e gerir os dados existentes numa base de dados.

Neste projeto decidimos usar como SGBD, PostgreSQL, pois era assegurado apoio neste SGBD, de fácil utilização e compatível com *containers* Docker.

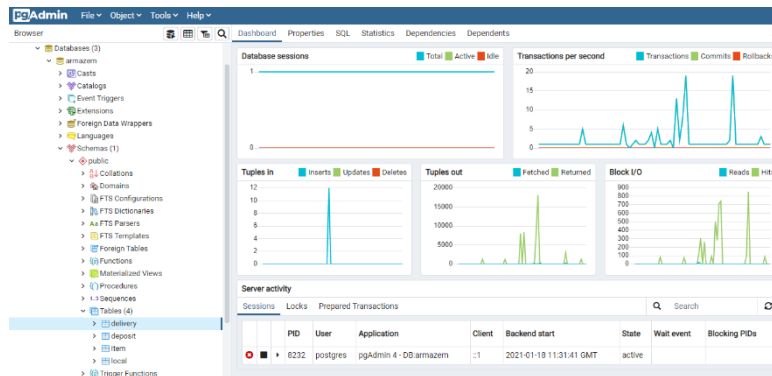


Figura 2- Pg admin do PostgreSQL

Na Base de Dados foram criadas 4 tabelas:

- Item
- Delivery
- Deposit
- Local

## item

Esta tabela tem como atributos:

- **id: Integer NOT NULL** – Id do item do tipo inteiro. É a primary key desta tabela e não pode ser um campo nulo.
- **name: character varying NOT NULL** - Nome do item do tipo character varying e não pode ser um campo nulo.
- **description: character varying** – É a descrição do item do tipo character varying.

```
CREATE TABLE item
(
    id integer NOT NULL,
    name character varying NOT NULL,
    description character varying,
    CONSTRAINT "PK_Item" PRIMARY KEY (id)
);
```

Figura 3- Tabela item

## deposit

Esta tabela tem como atributos:

- **id: Integer NOT NULL** – Id do deposit do tipo inteiro. É a primary key desta tabela e também é a foreign key referenciando “id” da tabela item. Não pode ser um campo nulo
- **quantity: Integer** – Quantidade do Deposit do tipo inteiro.

```
CREATE TABLE deposit
(
    id integer NOT NULL,
    quantity integer,
    CONSTRAINT "PK_id" PRIMARY KEY (id),
    CONSTRAINT "FK_itemId" FOREIGN KEY (id)
        REFERENCES item (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
);
```

*Figura 4- Tabela deposit*

## local

- **id: Integer NOT NULL** – Id do local do tipo inteiro. É a primary key desta tabela e não pode ser um campo nulo.
- **name: character varying** - Nome do local do tipo character varying.

```
CREATE TABLE local
(
    id integer NOT NULL,
    name character varying,
    CONSTRAINT "PK_Local" PRIMARY KEY (id)
);
```

*Figura 5-Tabela local*

## delivery

Esta tabela tem como atributos:

- **id: Integer NOT NULL** – Id do delivery do tipo inteiro. É a primary key desta tabela e não pode ser um campo nulo.
- **quantity: Integer NOT NULL** – Quantidade do Deposit com o tipo inteiro e não pode ser um campo nulo.
- **local\_id: Integer NOT NULL** - É a foreign key referenciando “id” da tabela local e não pode ser um campo nulo.
- **item\_id: Integer NOT NULL** - É a foreign key referenciando “id” da tabela item e não pode ser um campo nulo.

```
CREATE TABLE delivery
(
    id integer NOT NULL,
    quantity integer NOT NULL,
    local_id integer NOT NULL,
    item_id integer NOT NULL,
    CONSTRAINT "PK_Delivery" PRIMARY KEY (id),
    CONSTRAINT "FK_Delivery_Item" FOREIGN KEY (item_id)
        REFERENCES item (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
    CONSTRAINT "FK_Delivery_Local" FOREIGN KEY (local_id)
        REFERENCES local (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
);
```

Figura 6- Tabela delivery

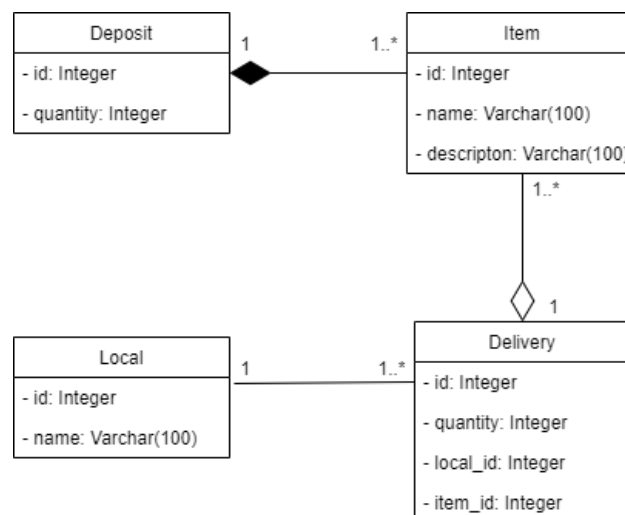


Figura 7- Diagrama de Classes

## Especificação das classes

Nome Classe:	Item		
Descrição da Classe	Classe que contém informações sobre cada item		
Atributos	id	Número de identificação dos itens	Integer
	name	Nome dos itens	Varchar
	description	Descrição dos itens	Varchar
Multiplicidade	Um item está contido em um depósito Nenhum item ou vários podem estar agregados a uma entrega		

Nome Classe:	Delivery		
Descrição da Classe	Classe que contém informações sobre cada		
Atributos	id	Nº de identificação das entregas.	Integer
	quantity	Quantidade das	Integer
	Local_id	Nº de identificação do local	Integer
	Item_id	Nº de identificação do item	Integer
Multiplicidade	Uma entrega pode ter um ou vários itens. Uma entrega tem um local		

Nome Classe:	Deposit		
Descrição da Classe	Classe que contém informações sobre o depósito		
Atributos	id	Número de identificação do depósito	Integer
	quantity	Quantidade dos depósitos	Integer
Multiplicidade	Um depósito pode conter vários itens		

Nome Classe:	Local		
Descrição da Classe	Classe que contém informações sobre o local		
Atributos	id	Número de identificação do local	Integer
	name	Nome dos locais	Varchar
Multiplicidade	Um local pode ter várias entregas		

## API REST

REST é um padrão arquitetural que modela aplicações de recursos na web.

API REST, também chamada de API RESTful, é uma interface de programação de aplicações que segue conformidade com as restrições da arquitetura REST. A sigla REST significa Representational State Transfer (Transferência Representativa de Estado).

O padrão assume que cada tipo de recurso na web tem um URL, e que cada instância de recurso tem um identificador. A interface com a aplicação é feita através de um esquema de URL's. De acordo com o URL utilizado, o padrão permite distinguir entre itens e coleções.

O padrão arquitetural propõe que:

- Os pedidos HTTP contenham toda a informação necessária para obter a resposta;
- Tudo é um recurso;
  - Cada recurso deverá poder ligar-se a outro recurso através de links.
- O acesso é uniforme;
  - O tipo de operação a efetuar num recurso, ou coleção de recursos, é definido por um padrão de verbos conhecido;
  - Todos os sistemas que sigam o padrão REST devem interpretar os verbos da mesma forma.

Nestas condições, propõe o seguinte significado para os operadores do pedido HTTP:

**POST** - Cria um novo recurso, e retorna o URL com identificador no cabeçalho.

**GET** - Obtém o recurso se o URL tiver identificador, ou obtém a listagem de recursos.

**DELETE** - Elimina o recurso com o identificador indicado.

**PATCH/PUT** - Altera o recurso com o identificador indicado.



## A lista de operações da API REST

- **O registo de um novo item, com a indicação do seu nome:**

Para esta operação é necessário o verbo POST que criará um novo item, neste caso foi criado um novo item com id=2, nome do item = “peca1”, descrição do item=“descrição peca1”.

A URL correspondente a essa operação é <http://localhost:8080/items>.

```
C:\Users\bruno>curl -X POST -d "id=2&name=peca1&description=descricao peca 1" http://localhost:8080/items
{"id":2,"name":"peca1","description":"descricao peca 1"}
C:\Users\bruno>
```

*Figura 8- Registo de um novo item*

- **O registo do depósito de um item, com a indicação da quantidade:**

Para esta operação precisamos do verbo POST que criará um registo do depósito do novo item e indicar a sua quantidade.

Como se vê na figura 9, foi registado o depósito do id do item 2 e a quantidade=10.

A URL correspondente a essa operação é <http://localhost:8080/deposits>.

```
C:\Users\bruno>curl -X POST -d "id=2&quantity=10" http://localhost:8080/deposits
{"id":2,"quantity":10}
C:\Users\bruno>
```

*Figura 9- Registo do depósito de um item*

- **O registo de uma entrega de itens, com a indicação da quantidade de cada item, e a descrição do local de entrega:**

Para esta operação é necessário o verbo POST, que vai criar um registo de uma entrega de itens.

Como se pode ver na figura 10, criou-se o registo de uma entrega de itens com o id do item=2, quantidade=10, o id do local=12 e o id da entrega=1. A URL correspondente a essa operação é <http://localhost:8080/deliveries>.

```
C:\Users\bruno>curl -X POST -d "id=1&quantity=10&local_id=12&item_id=2" http://localhost:8080/deliveries
{"id":1,"quantity":10,"local_id":12,"item_id":2}
C:\Users\bruno>
```

*Figura 10- Registo de uma entrega de itens*

- **A consulta da coleção de itens registados:**

Para esta operação precisa-se do verbo GET, para obter a coleção de itens ou um item específico que está registado.

Como se vê nas figuras 11 e 12 está-se a obter uma coleção de itens e uma coleção de itens em específico.

A URL correspondente a essa operação é <http://localhost:8080/items/2>.

```
C:\Users\bruno>curl -X GET http://localhost:8080/items/2
{"id":2,"name":"peca1","quantity":0,"description":"descricao peca 1"}
C:\Users\bruno>
```

*Figura 11-Consulta da coleção de itens*

A URL correspondente a essa operação é <http://localhost:8080/items>

```
C:\Users\bruno>curl -X GET http://localhost:8080/items
{"id":6,"name":"teste","quantity":100,"description":"fd"}{"id":2,"name":"peca1","quantity":10,"description":"descricao peca 1"}
C:\Users\bruno>
```

*Figura 12- Consulta da coleção de itens*

- **A consulta das entregas registadas:**

Para esta operação é necessário o verbo GET, para obter a coleção das entregas registadas.

Como se vê na figura 13, obtêm-se duas entregas com os respetivos id, quantidade, id do local e id do item.

A URL correspondente a essa operação é <http://localhost:8080/deliveries>.

```
C:\Users\bruno>curl -X GET http://localhost:8080/deliveries
{"id":6,"quantity":10,"local_id":12,"item_id":6}{ "id":1,"quantity":10,"local_id":12,"item_id":2}
C:\Users\bruno>
```

*Figura 13- Consulta das entregas registadas*

- **A alteração de descrição de itens:**

Para esta operação é preciso o verbo PUT, para alterar a descrição de um item em específico, como se constata na figura 14, alteramos a descrição do item com o id=10.

A URL correspondente a essa operação é <http://localhost:8080/items/10>.

```
C:\Users\bruno>curl -X PUT -d "description=descricao mudou" http://localhost:8080/items/10  
{ "Alterado o nome do item id":10 }
```

*Figura 14- Alteração de descrição de itens*

- **A alteração do local de destino de uma entrega:**

Para esta operação é preciso o verbo PUT, onde se altera a descrição do local de destino de uma entrega, como se vê na figura 15, altera-se o local da entrega com o id=6.

A URL correspondente a essa operação é <http://localhost:8080/deliveries/6>.

```
C:\Users\bruno>curl -X PUT -d "local_id=36" http://localhost:8080/deliveries/6  
{ "id":6, "quantity":10, "local_id":36, "item_id":6 }
```

*Figura 15- Alteração do local de destino*

- **A eliminação de um item, caso nunca tenha sido registado um depósito ou entrega com este:**

Para esta operação é necessário o verbo DELETE, que vai eliminar um item que nunca foi registado num depósito ou entrega.

Como vemos nesta Figura aparece a informação que foi eliminado o item com o id=1. A URL correspondente a essa operação é <http://localhost:8080/items/1>.

```
C:\Users\bruno>curl -X DELETE http://localhost:8080/items/1  
{ "Eliminado item com o id":1 }  
C:\Users\bruno>
```

*Figura 16- Eliminação de um item*

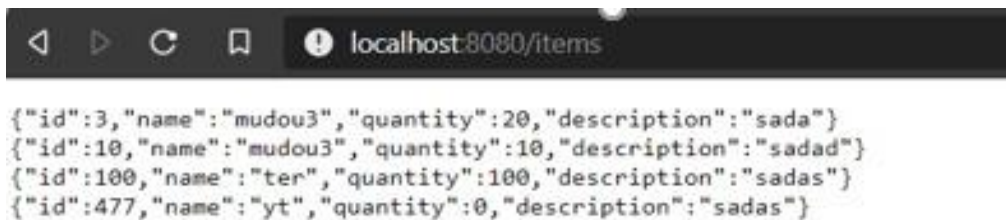
## Wildfly

Wildfly, também conhecido como JBoss, é um servidor de aplicação Java EE desenvolvido em Java e pode ser executado em qualquer Sistema Operativo, 32 ou 64 bits que tenha suporte Java.

Aqui neste tópico vamos mostrar os outputs da API REST em JSON no Wildfly.

Só iremos fazer uso do verbo GET pois queremos consultar a informação armazenada na base de dados.

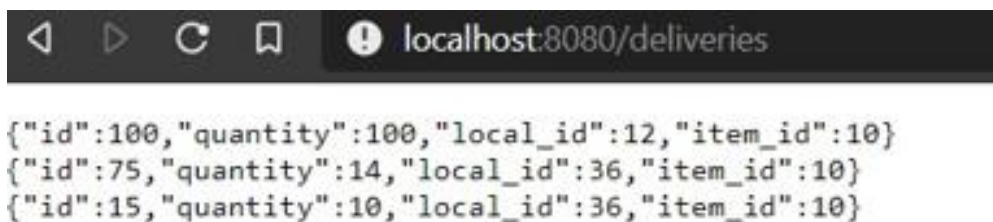
- **Consulta da coleção de todos os itens:**



```
{ "id": 3, "name": "mudou3", "quantity": 20, "description": "sada" }  
{ "id": 10, "name": "mudou3", "quantity": 10, "description": "sada" }  
{ "id": 100, "name": "ter", "quantity": 100, "description": "sadas" }  
{ "id": 477, "name": "yt", "quantity": 0, "description": "sadas" }
```

Figura 17- Consulta da coleção de itens


- **Consulta da coleção de todas as entregas:**



```
{ "id": 100, "quantity": 100, "local_id": 12, "item_id": 10 }  
{ "id": 75, "quantity": 14, "local_id": 36, "item_id": 10 }  
{ "id": 15, "quantity": 10, "local_id": 36, "item_id": 10 }
```

Figura 18- Consulta da coleção das entregas

- **Consulta da coleção de todos os depósitos:**



```
{ "id": 10, "quantity": 10 } { "id": 3, "quantity": 20 } { "id": 100, "quantity": 100 }
```

*Figura 19- Consulta da coleção de depósitos*

- **Consulta de um item específico :**



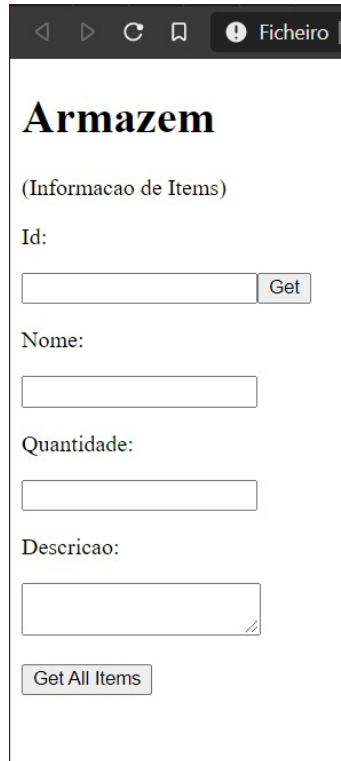
```
{ "id": 10, "name": "mudou3", "quantity": 10, "description": "sadam" }
```

*Figura 20- Consulta de um item específico*

## Web App

No último tópico, criámos a web app que tem interação com o utilizador e serve o conteúdo em HTML, o acesso à informação como já foi dito em cima é feito através da API REST.

A figura seguinte mostra a página da Web App.



The screenshot shows a web browser window with a dark header bar containing navigation icons and the text 'Ficheiro'. The main content area has a title 'Armazem' in a large, bold, serif font. Below the title is a subtitle '(Informacao de Items)'. The form consists of several input fields and buttons: an 'Id:' label followed by a text input field and a 'Get' button; a 'Nome:' label followed by a text input field; a 'Quantidade:' label followed by a text input field; a 'Descricao:' label followed by a larger text input field with a small icon in the bottom right corner; and a 'Get All Items' button at the bottom.

Figura 21- Output da nossa Web App HTML