Relatório Técnico - Especificações do Projeto

Sistema de monitorização de clientes e produtos de uma loja física e automatização de compras

IES - Introdução à Engenharia de Software

93346 - Alexandra de Carvalho

92972 - Gonçalo Matos

93195 - Hugo Almeida

91322 - Isadora Loredo

Aveiro, 14 de janeiro de 2021

Índice

1. Introdução

Um dos principais objetivos do projeto consiste na exploração de conceitos ligados à monitorização e controlo de sistemas automatizados.

Para isto, serão aplicadas práticas de Engenharia de Software que vão desde a especificação de um produto até à implementação final do mesmo, aplicando práticas de trabalho colaborativo utilizando ferramentas como o *Github* para gestão e armazenamento de código e *Jira* para gestão de tarefas entre a equipa.

Será, ainda, abordada a construção de uma arquitetura que vá de encontro aos requisitos do projeto, sendo discutida a escolha de cada componente.

2. Equipa

O sistema foi desenvolvido no âmbito da Unidade Curricular de Introdução à Engenharia de Software, sendo necessário distribuir o trabalho pelos participantes do grupo. Deste modo, e de acordo com os requisitos do projeto, foi feita a seguinte atribuição:

- Alexandra de Carvalho - **Architect**
- Gonçalo Matos - **DevOps Master**
- Hugo Almeida - **Team Manager**
- Isadora Loredo - **Product Owner**

Apesar destas funções, todos os elementos participaram no desenvolvimento das tarefas.

3. Práticas

3.1 *Backlog*

Com o objetivo de gerir o trabalho de desenvolvimento do sistema foi utilizado o software Jira, de acordo com a metodologia Scrum, baseada em Agile. Foram criados vários Sprints, usualmente de duas semanas, com as tarefas da iteração distribuídas pelos membros do grupo.

3.2 *Workflow* com *Feature-branching*

Tal como requerido, todo o desenvolvimento através de *GIT* foi estruturado com *features* por *branch*. As *branches* eram nomeadas de acordo com o projeto que estava a ser desenvolvido, o tipo de trabalho que se estava a fazer (adição de novas funcionalidades ou *bug fixes)* e o nome da *feature*. Desta forma, o nome foi sempre algo como ***projservice/feature/feature_name***.

Sempre que uma *feature* era terminada, a criação de uma *Pull Request* era obrigatória, permitindo a revisão de código entre os membros e a descrição da tarefa em questão.

3.3 *Deployment*

Todo o *deployment* foi feito utilizando o princípio da segregação de responsabilidades, dividindo todos os componentes em *containers* de *Docker*. Para reunir todos os componentes e permitir a rápida execução na *VM*, recorreu-se ao *Docker Compose*.

Por outro lado, para criar uma *VM*, o grupo recorreu ao serviço da *Google Cloud* descrito no *README* do repositório de modo a carregar, executar e disponibilizar todo o sistema através do *Docker Compose*.

O sistema encontra-se disponível no seguinte *link*: 35.246.117.113

4. Conceito do produto

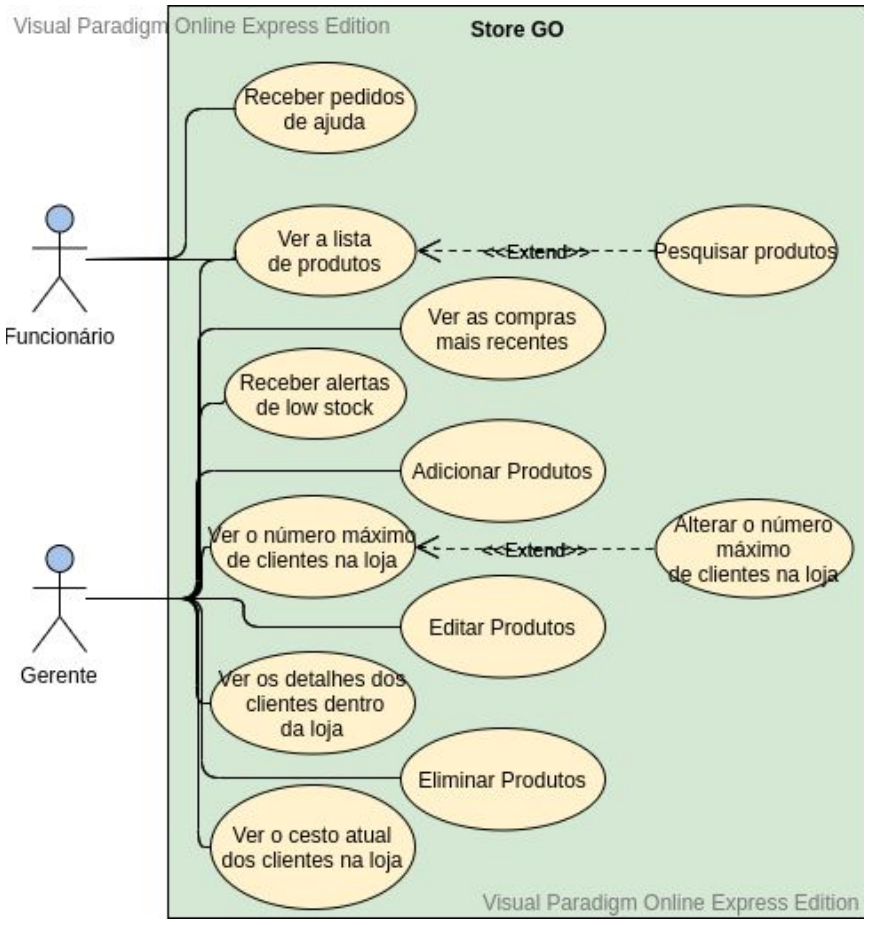
4.1 Visão

O sistema tem a finalidade de simular uma loja automatizada, ou seja, um estabelecimento que proporciona uma experiência de compras sem caixas de pagamento, ao reconhecer os produtos que um cliente retirou da prateleira e efetuar a cobrança no momento em que ele deixa a loja.

Ao chegar ao supermercado é detectada a entrada do cliente na loja e através dos sensores distribuídos pelo espaço são adicionados produtos ao seu carrinho virtual ou removidos se este os voltar a pousar. Terminadas as compras, não é preciso fazer nada: apenas sair da loja. Uma vez do lado de fora da loja a compra é finalizada.

Esta aplicação é similar ao supermercado inteligente da Amazon com o conceito “Just Walk Out“, sem filas e sem *checkouts*.

Tudo isto será visualizado através da aplicação *web* para Funcionários e Gerentes, cada um com diferentes permissões e ações possíveis. Estes são alguns dos casos de uso:



4.2 Personas

Foram definidos dois atores principais do sistema:

Gerente

- Amélia Rodrigues tem 32 anos e é **gerente** da StoreGO

Funcionário

- Pedro Paulo tem 26 anos e é **funcionário** da StoreGO da Amélia

4.3 Cenários Principais

Monitorização dos clientes atuais na Loja

Um **Gerente** da loja, através da aplicação *web* vai conseguir visualizar em tempo real as informações relativas aos **clientes que se encontram na loja** bem como os seus **cestos de compras**. Além disso, este poderá limitar o número de clientes em simultâneo dentro da loja.

Notificações de alterações de estado

O sistema, consoante a ocorrência de eventos (por exemplo limite de pessoas atingido, falta de *stock* e pedidos de ajuda de clientes) irá **notificar os seus atores**. Caso seja um **Gerente**, este receberá em tempo real notificações relativas a falta de *Stock* e limite de pessoas atingido. Quanto ao **Funcionário**, este receberá apenas notificações de pedidos de ajuda provenientes dos clientes.

Um **Gerente** consegue ainda visualizar todo o histórico de notificações que o sistema gerou.

Monitorização de Produtos

É possível visualizar todos os produtos incluindo informações relativas ao *stock* atual, categoria, etc. O **Gerente** consegue editar, adicionar e remover produtos bem como repor *Stock*. O **Funcionário** apenas consegue visualizar a listagem de produtos, fazer pesquisa e repor *Stock*.

Alteração de Preferências

Tanto o **Funcionário** como o **Gerente** vão poder alterar informações relativas à sua conta e escolher que tipo de notificações pretendem receber dentro do tipo de notificações permitido.

Gestão de Pedidos de Ajuda

Um **Funcionário** terá acesso à lista de pedidos de ajuda dos clientes que se encontram pendentes, podendo alterar o seu estado indicando que o pedido foi resolvido.

Histórico de Compras

Como **Gerente**, é possível aceder a todo o histórico de compras realizadas na loja.

Gestão dos Clientes do Sistema

O **Gerente** tem permissões para aceder à listagem de clientes da loja e por sua vez, ao histórico de compras desses mesmos clientes.

5. Notas de Arquitetura

5.1 Principais requisitos e Restrições

A escolha da arquitetura centra-se no endereçamento das seguintes questões:

- O sistema deve ser capaz de gerar e consumir dados automaticamente, simulando o que aconteceria numa loja em funcionamento.
- Devido às interações que os utilizadores podem realizar, o script *python* necessita de consumir os novos dados atualizados, de modo a garantir a congruência dos dados gerados. Por exemplo, quando é alterado o limite de pessoas dentro da loja não faz sentido serem geradas mais pessoas e novos eventos para essas.
- Deve-se garantir que os dados gerados pelo simulador não são misturados no *message broker* com os dados recebidos.
- O sistema não pode permitir acesso por terceiros a dados confidenciais, necessitando de um sistema de autenticação.
- O sistema estará alojado numa máquina virtual.

5.2 Vista da Arquitetura

Aplicação Web: Escolheu-se a biblioteca de *JavaScript React* para construir a aplicação web do sistema devido às vantagens de desenvolvimento que esta apresenta face ao desenvolvimento de aplicações dinâmicas.

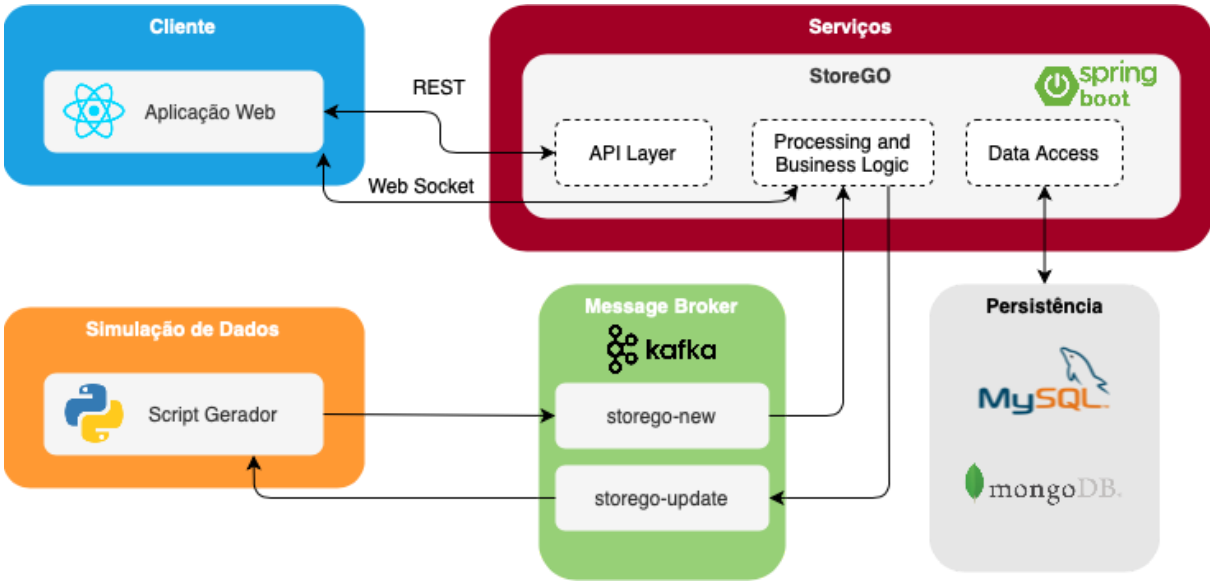
StoreGO: Será construído um serviço baseado em *Spring Boot* que será o ponto essencial do sistema. Este será composto por uma camada *REST API*, que permitirá consultar diversos dados do sistema para serem posteriormente apresentados na aplicação *web*. Existe ainda a camada do modelo de dados, que permite o mapeamento dos dados armazenados na base de dados para objetos *Java* a serem tratados e processados na camada de lógica de negócio.

Simulação de Dados: Foi definido que os dados consumidos pelo sistema seriam gerados através de um script em Python, que consumirá as novas atualizações de dados, essenciais ao seu funcionamento, através de um tópico do *Message Broker.* Os dados gerados serão publicados num outro tópico *Kafka* de modo a serem enviados para o sistema.

Message Broker: Foi escolhido *Kafka* como *message broker* uma vez que permite um sistema rápido e escalável de produtores-consumidores com um grande volume de dados. Existirão dois tópicos com diferentes objetivos. Um deles servirá para enviar dados que foram gerados de modo a serem guardados e processados pelo sistema. O outro

servirá para atualizar os dados do gerador visto que ao longo do tempo poderão existir dados essenciais à geração que sofrerão alterações por parte dos utilizadores.

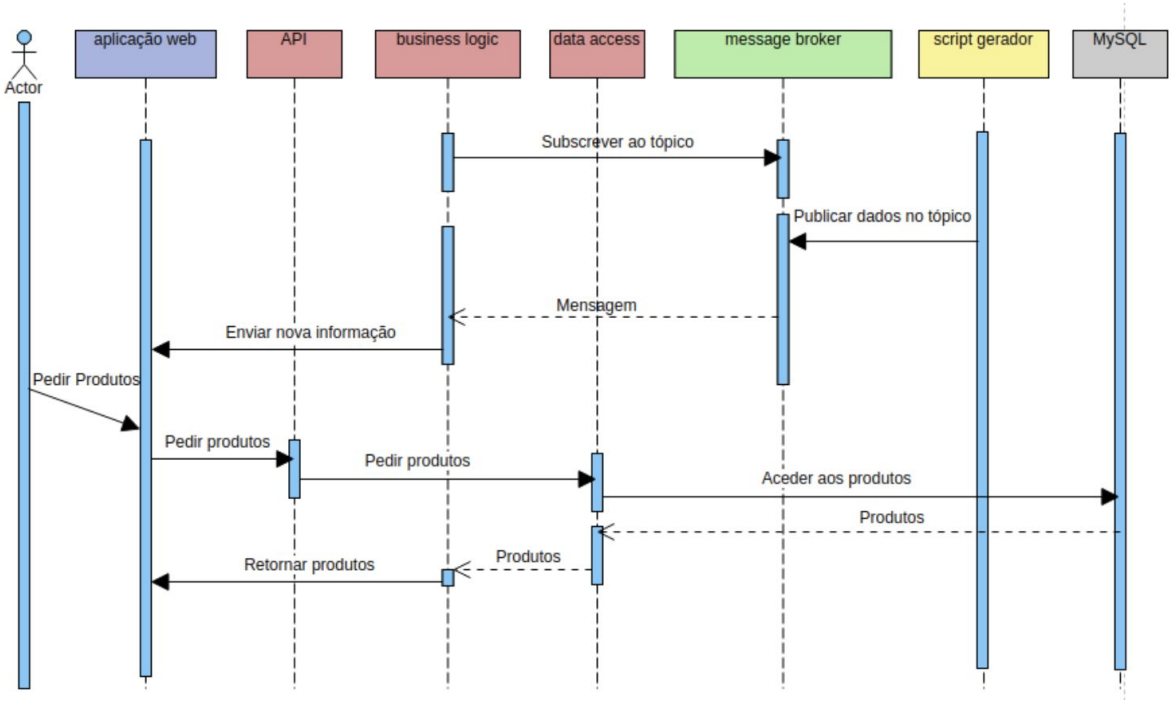
Base de Dados: Utilizado MySQL para armazenar de forma relacional os principais dados do sistema. Será também utilizado o MongoDB para armazenar informações como logs de notificações que não necessitam de estar armazenados de forma relacional.



Exemplo de Interação entre Módulos

1. Os dados são gerados através do *Script* gerador, sendo publicados num tópico de *Kafka*
2. Estes dados são consumidos pela lógica de negócio implementada em *Spring Boot,* armazenando-os na base de dados
3. Através da ligação *Web Socket*, são enviadas à aplicação web novas notificações (por exemplo, limite de clientes na loja atingido)
4. O cliente através da aplicação *web* vê, por exemplo, os produtos disponíveis, sendo feito um pedido à *REST-API* para este propósito
5. Através da *REST-API* são enviados os dados pedidos, com um acesso à base de dados e passagem pela lógica de negócio
6. O gerente através da aplicação *web* dá ordem de reposição de stock, sendo enviada essa informação à API através de um pedido PUT
7. Ao receber este pedido, as informações são alteradas na base de dados e é publicado no tópico correto os novos dados atualizados

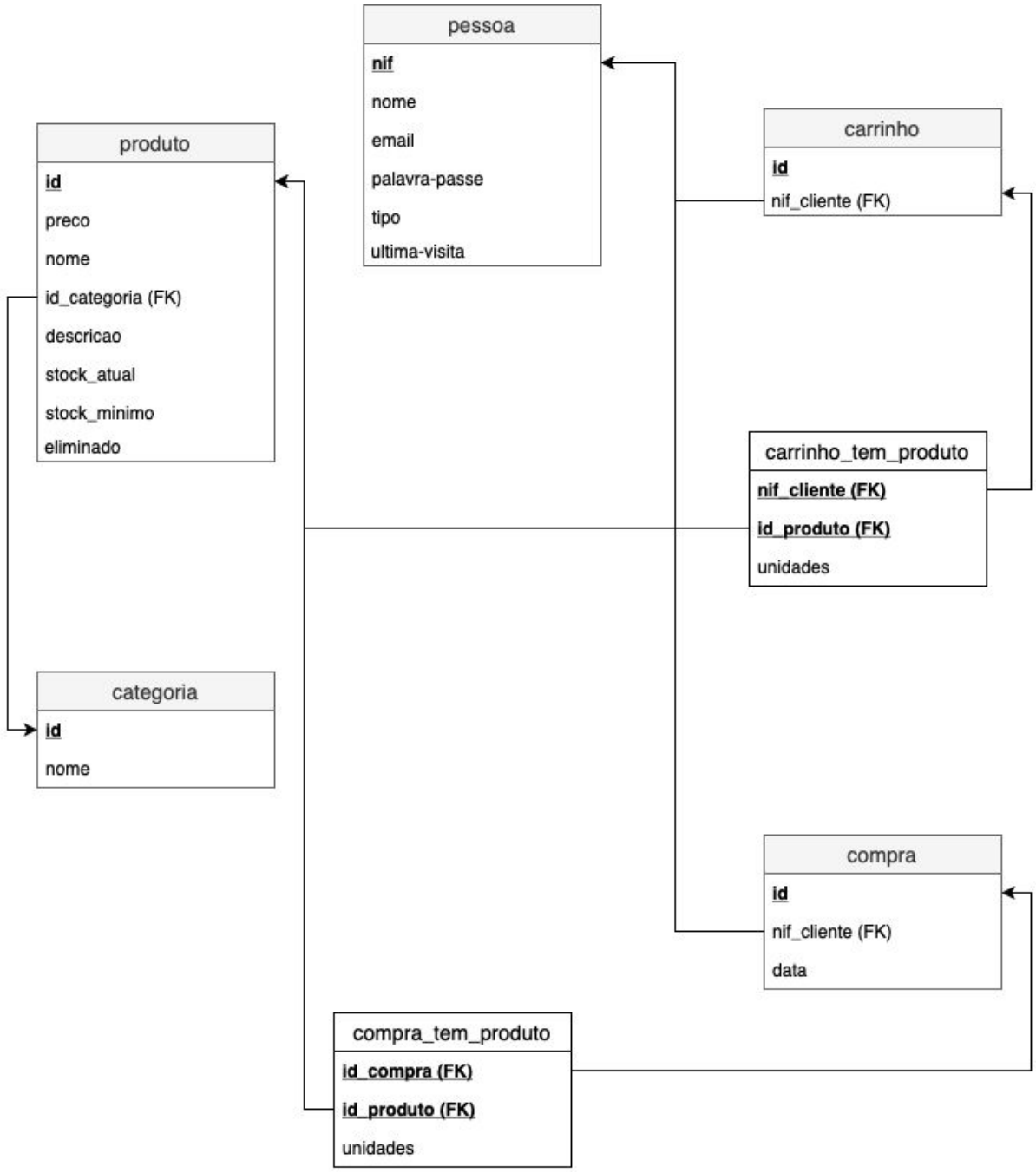
8. O *Script Gerador* ao consumir estes dados através desse tópico, irá gerar novos dados de acordo, mantendo a congruência



6. Perspetiva da Informação

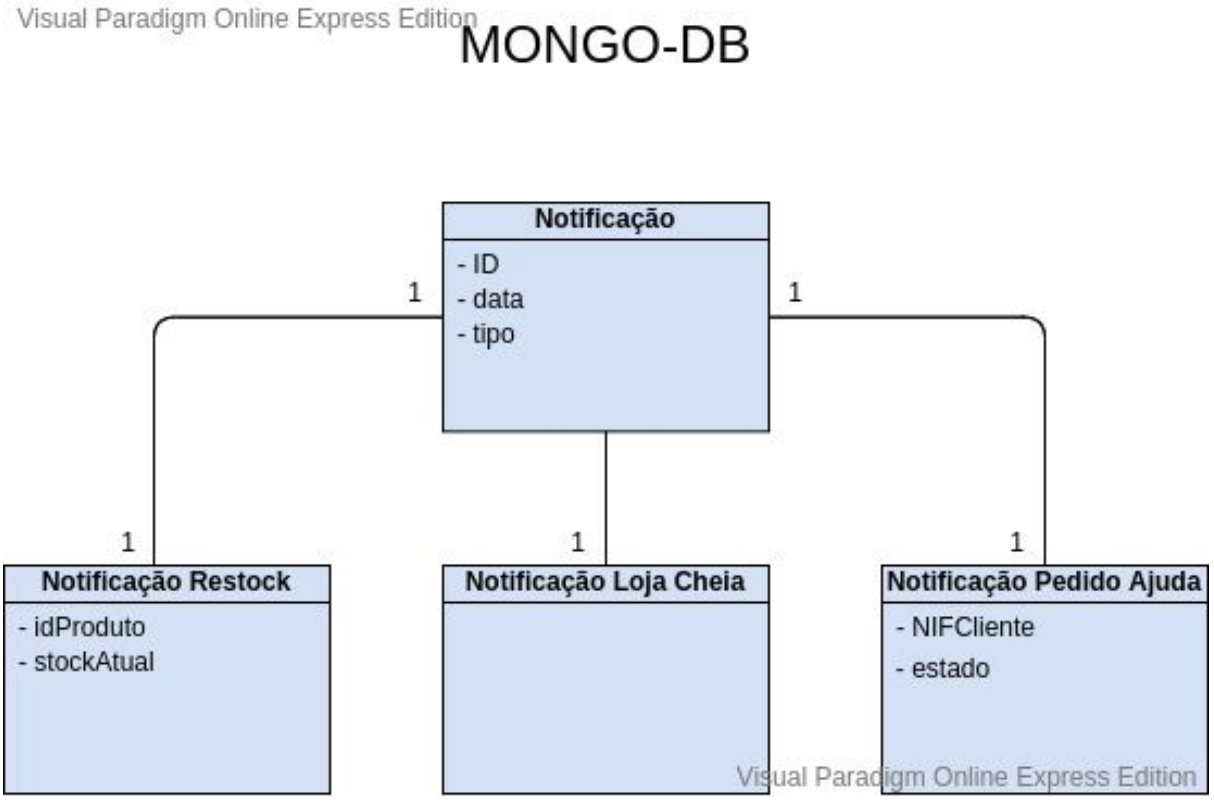
Tendo em consideração as necessidades do projeto e visto que grande parte das entidades estão de alguma forma relacionadas entre si, fez sentido optar pela utilização de uma base de dados relacional, *MySQL*, para estas entidades. A escolha de *MySQL* deve-se à sua utilização anteriormente no contexto das aulas práticas de IES no entanto poderia-se ter optado por outras como *PostgreSQL*, *MariaDB*, entre outras.

Para a base de dados relacional, existirão as seguintes entidades:



Uma vez que as notificações serão guardadas e dado que estas apresentam uma maior simplicidade face às entidades descritas anteriormente, optou-se por documentá-las através de uma base de dados documental, *MongoDB*, devido ao conhecimento adquirido previamente em outras disciplinas.

A estrutura de cada documento, usando as possibilidades dos documentos, não é necessariamente a mesma, dependendo de cada tipo de notificação:



7. Conclusão

Para terminar, pensa-se que, de acordo com as metas estabelecidas pelo docente, o trabalho foi bem sucedido. O foco principal foi a monitorização e deteção de eventos em tempo real ainda que, devido à dificuldade de acesso a recursos físicos, nomeadamente sensores para monitorização de pessoas e produtos, e devido à realidade pandémica, se tenha tornado impossível testar o projeto num cenário real. Para ultrapassar estas barreiras, implementou-se um *script* gerador cujo objetivo assenta na detecção e recolha de dados, simulando, assim, o fluxo normal que a loja apresenta num contexto real.

A arquitetura implementada é uma arquitetura monolítica onde apenas contém um serviço a ser consumido. Na eventualidade do serviço em questão falhar, o cliente, através da aplicação *web,* fica sem dados na sua totalidade uma vez que todos provém deste único serviço. No entanto, como trabalho futuro, dada a complexidade do fluxo do sistema, seria interessante a divisão do serviço em múltiplos serviços de modo a garantir que a falha de um serviço não afetasse a disponibilização de outros dados não dependentes desse serviço. Por exemplo, os históricos poderiam ficar associados a um serviço, geração de notificações a outro serviço, operações de gestão outro serviço, etc.

Há que referir ainda que a metodologia de trabalho implementada facilitou a gestão de tarefas a serem desenvolvidas, permitindo aos elementos do grupo perceberem o ponto de situação do projeto de forma rápida e clara.