Gestão Inteligente de Stocks

Ana Rita Ferreira dos Santos
Inês Lima Amil Soares
Nuno Manuel Olival Veloso

Orientadores Matilde Pós-de-Mina Pato
Nuno Miguel Soares Datia

Relatório final realizado no âmbito de Projeto e Seminário,
do curso de licenciatura em Engenharia Informática e de Computadores
Semestre de Verão 2017/2018

Julho de 2018

Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores

Gestão Inteligente de Stocks

42142 Ana Rita Ferreira dos Santos
42162 Inês Lima Amil Soares
42181 Nuno Manuel Olival Veloso

Orientadores: Matilde Pós-de-Mina Pato
Nuno Miguel Soares Datia

Relatório final realizado no âmbito de Projeto e Seminário,
do curso de licenciatura em Engenharia Informática e de Computadores
Semestre de Verão 2017/2018

Julho de 2018

Resumo

A gestão de stocks ajuda a controlar, de forma otimizada, os investimentos em stock de uma casa. Com um sistema de gestão apropriado é possível obter diversos benefícios, tais como, acabar com o esquecimento de produtos no fundo dos locais de armazenamento ou fora da validade. A gestão de stocks envolve três decisões principais:

- decidir quando comprar os produtos,
- determinar a quantidade a comprar de cada produto e,
- garantir um stock mínimo de segurança para cada produto.

Estas decisões assumem uma dinâmica que se repete ao longo do tempo. Pretende-se desenvolver um sistema de gestão automática de stocks, onde a recolha de informação é feita por sensores. O sistema é constituído, para além dos sensores, por uma aplicação móvel e web, e um servidor que implementa um algoritmo de previsão de stocks e que disponibiliza uma API *Web*. Este sistema simplifica não só o controlo de stocks, como também a análise dos padrões de consumo e reposição de uma casa. Assim consegue-se auxiliar os utilizadores a manter o stock adequado às suas necessidades, bem como alerta-los para a proximidade do fim da validade e/ou stock dos produtos.

Palavras-chave: IoT; Gestão de Stock; Previsão de consumo; Sensorização em casa

Lista de Figuras

Lista de Tabelas

Índice

Capítulo 1

Introdução

O conceito Sistema de Gestão de Stocks (GES) compreende o registo de todos os movimentos de stocks que, através da disponibilização de ferramentas informáticas, permitem efetuar a gestão administrativa de stocks, gestão de armazéns e a gestão económica de stocks, com controlo em tempo-real do stock existente. Assim, é possível responder eficazmente às necessidades de fluxo de stocks, que envolve três decisões principais: i) decidir quando comprar, ii) escolher quanto comprar, e iii) determinar a quantidade de stock de segurança. Estas decisões assumem uma dinâmica repetitiva ao longo do tempo, e tornam-se complexas quando estão envolvidos muitos fatores na tomada das mesmas. O GES é complementado por sistema de alertas de rotura de stock mínimo e por um conjunto de mecanismos que permitem efetuar o apuramento de custos vitais para uma gestão eficiente da organização. Todas as organizações, seja qual for o sector onde operam, partilham o mesmo problema: como efetuar a manutenção e controle de stock. Este problema não reside apenas nas empresas como indústria, distribuição, operadores logísticos, retalho. Numa outra escala, como as nossas casas, a gestão de stocks também merece destaque. Quem nunca se deparou com produtos fora de validade ou legumes e frutas muito velhas?

Para além da gestão de stocks, automatizar a recolha de informação sobre o que compramos, quando compramos, e quando consumimos permite identificar situações onde compramos quase sempre os mesmos produtos, esquecendo outros que não comemos há muito tempo. Estes padrões tornam a alimentação pouco variada, podendo contribuir para eventuais problemas de saúde que possam surgir. A razão prende-se, essencialmente, com as alterações nas rotinas diárias das famílias. A gestão de stocks é uma tarefa repetitiva e estruturada, para a qual já existem soluções como as listas de compras, nomeadamente *OutOfMilk*[1] e *Bring*[2] são exemplos dessas soluções no formato de aplicações *mobile*. No entanto, estas soluções não permitem o controlo de stocks e "aprendizagem" dos hábitos dos seus utilizadores.

A ideia geral deste projeto consiste na gestão "inteligente" de stocks, ligado à *Internet of Things*(IoT). Através de uma aplicação *mobile* e *web* com suporte inteligente de um algoritmo

[1] https://www.outofmilk.com/
[2] https://www.getbring.com/#!/app

de previsão de stocks, é possível fazer a gestão de produtos numa casa. Tendo por base a automatização da recolha de dados recorrendo a sensores, simplifica-se, não só, o controlo de stocks, como também, a análise dos padrões de consumo e reposição numa casa. Desta forma, auxilia-se os utilizadores a manter o stock adequado às suas necessidades, bem como alertá-los para a proximidade do fim da validade e/ou quantidade dos produtos.

Este trabalho vai no sentido de responder a questões como: "De que forma podemos evitar transtornos causados na altura de reabastecer a nossa despensa?", "Como proceder ao controlo de stocks de alimentos e outros produtos?", "Como impedir artigos fora de prazo?". Se se entender que uma casa funciona como uma empresa e existem quantidades mínimas recomendadas, é possível gerar uma nota de encomenda com os produtos em falta ou prestes a terminar para o utilizador poder consultar e exercer a compra.

1.1 Contexto

Uma boa gestão de stocks de mercadorias é de extrema importância porque tem reflexos imediatos nos resultados de uma empresa, o que permite manter os clientes satisfeitos não só a nível da quantidade como da qualidade. Para manter o stock ideal não basta bom senso e intuição, é necessário conhecer o fluxo de vendas, utilizar ferramentas adequadas de gestão de informação sobre movimentos e eventuais constrangimentos no fornecimento. Nas nossas casas, o problema é idêntico apenas numa escala diferente. Organizar a despensa como se de uma empresa se tratasse possibilita uma melhor logística de custos e tempo. Ao elaborar uma lista de compras, onde se vai anotando os produtos que se tem, o que está a acabar e o que se tem de comprar, passa por uma solução indispensável. Que por vezes se torna numa tarefa que "não é para todos".

Perante este problema, pretende-se desenvolver um sistema, utilizando uma solução digital, aplicação *mobile* e de *web*, que tem como objetivo ajudar os portugueses nesta repetitiva tarefa que é adotar e manter, ao longo do tempo, a sua despensa sem faltas. Através desta solução, o individuo terá sempre presente informação útil e prática, com possibilidade de utilizar um formato de lembretes e de registar as tendências para uma futura investigação no que diz respeito aos hábitos de consumo.

Destaca-se ainda o facto de, no contexto atual, existir um aumento na facilidade de acesso às novas tecnologias, nomeadamente à *internet*. Em plena era da informação a proliferação dos meios de comunicação e da própria *internet* permitiu que os utilizadores se liguem à rede 24 horas, por dia, através de telemóveis, portáteis, *tablets* e outros. A cada dia que passa assiste-se a uma mudança do comportamento do consumidor nesta área, graças à utilização dos dispositivos móveis dos "8 aos 80"anos. Conforme os dados divulgados em Dezembro de 2016 pelo Gabinete de Estatísticas da União Europeia (Eurostat) [1].

Atualmente, a evolução tecnológica move-se a uma velocidade nunca vista. Segundo um

estudo da Accenture [2], estima-se que, esta tendência venha adicionar 14,2 mil milhões de dólares à economia global até 2030. A IoT surgiu nos últimos anos e está a alterar os nossos hábitos, sem nos darmos conta, já que está presente em televisões, frigoríficos e outros aparelhos. Casa inteligente é uma das áreas mais proeminentes de aparelhos inteligentes [3], e a cozinha é um dos lugares onde tais aparelhos são usados. Em Junho de 2000, a LG lançou o primeiro frigorífico ligado à internet, o "Internet Digital DIOS"[1]. Também conhecido por frigorífico inteligente foi programado para detetar que tipos de produtos são armazenados e manter o controle de stock por meio de leitura de código de barras ou RFID (Radio Frequency IDentification). Este frigorífico foi um produto mal sucedido porque os consumidores o viram como um produto desnecessário e devido ao preço excessivo (à época, mais de $20,000) para além da falta de capacidade de comunicar aos utilizadores para futuras compras quando existem vários e a comunicação é informal. Desde essa altura, começaram a surgir muitas soluções com mais funcionalidades. Na CES 2018[2], a Samsung mostrou o novo Family Hub com assistente pessoal inteligente Bixby, integração SmartThings com outros aparelhos de casa entre outras funcionalidades. A Siemens Home Connect é outra solução entre outros fabricantes de electrodomésticos. Atualmente, os preços são mais convidativos. O Family Hub não atinge os $4,000[3]. Outra solução é o módulo Intelligent Refrigerator [4]. Este foi projetado para converter qualquer frigorífico existente num aparelho inteligente e económico usando Inteligência Artificial. O frigorífico é capaz de detetar e monitorizar o seu conteúdo remotamente, tem uma funcionalidade de indicar itens que não são consumidos há muito tempo. A principal funcionalidade é manter, com mínimo esforço, uma lista de stocks de alimentos assim que necessário. Como resultado, o utilizador é notificado todos os dias sobre a quantidade, idade e itens descarregados.

1.2 Metas e Objetivos

Face ao exposto, a existência de um sistema de gestão de stocks na agenda de tarefas de uma organização doméstica poderá ser uma mais valia. Para concretizar esse sistema foi necessário equacionar os objetivos específicos que respondessem à seguinte questão:
"Quais as características e funcionalidades que o sistema devera ter para que seja útil para os utilizadores e se diferencia das restantes?"

Deste modo, definiram-se os seguintes objetivos:

- Implementar uma interface com o utilizador para dispositivos móveis;
- Implementar uma interface com o utilizador para dispositivos *desktop*;
- Implementar a componente servidora de um sistema de gestão de stocks;

[1] Acedido a 21 Junho 2018, https://www.itweb.co.za/content/KA3WwqdlozkqrydZ
[2] 51ª edicão da *Consumer Electronics Show*, Las Vegas, EUA
[3] Acedido a 21 Junho 2018, https://www.samsung.com/us/home-appliances/refrigerators/s/

- Implementar um algoritmo de previsão de stocks.

1.3 Abordagem do Projeto

Este trabalho divide-se em duas partes principais. A primeira com o enquadramento teórico, em que se fez uma revisão da literatura focando os principais temas associados ao projeto, nomeadamente, gestão de stocks, a utilização das novas tecnologias. Reviu-se também estratégias de usabilidade e promoção de literatura na construção das aplicações móveis bem como a regulamentação existente e possibilidade de certificação. Ainda nesta parte, efetuou-se investigação exploratória de suporte à elaboração do projeto, assim como análise e discussão dos resultados obtidos.

Na segunda parte encontra-se todo o trabalho desenvolvido para o projeto. Inicialmente definiram-se os requisitos fundamentais ao sistema de gestão de stocks a desenvolver. Aqui, foi também importante comparar as funcionalidades que se pretendiam implementar com as de outros sistemas já existentes, de forma a garantir elementos inovadores na solução. Posteriormente, definiu-se a arquitetura do sistema tal como todas as partes envolvidas. Foi ainda necessário estabelecer o modo como o utilizador iria interagir com o sistema, através do desenho das interfaces gráficas.

1.4 Estrutura do Relatório

Este relatório está organizado em 4 capítulos.

O capítulo 2 introduz o sistema de gestão de stocks desenvolvido, como também, formaliza o problema indicando os requisitos, as entidades para a resolução do mesmo e, ainda, apresenta a solução implementada.

O capítulo 3 aborda a modelagem dada aos dados, as aplicações de interação direta com o utilizador, a API *Web* bem como, todas as suas particularidades. Por fim, é também explicado o desenvolvimento e implementação do algoritmo de previsão de stocks.

Conclusões, testes e diretrizes relativas a trabalho futuro são disponibilizadas no capítulo 4.

No Anexo A definem-se as tabelas de domínio de cada uma das entidades do projeto.

Capítulo 2

Sistema de Gestão de Stocks

O sistema de gestão de stocks desenvolvido neste projeto, denominado Smart Stocks, é apresentado neste capítulo, bem como os requisitos funcionais, não funcionais e opcionais. São ainda expostos conceitos básicos importantes para uma melhor compreensão do sistema desenvolvido.

2.1 Conceitos Básicos de Gestão de Stocks

Em qualquer gestão de stocks é necessário realizar uma catalogação, de forma regular, dos bens ou propriedades tangíveis existentes. A essa lista dá-se o nome de **Inventário** [5]. Quando há uma forma automatizada de dar entrada e saída dos bens, o inventário serve para determinar erros e não-conformidades. Cada bem é identificado no stock por um código, cujo nome se dá ***Stock Keeping Unit* (SKU)** (Unidade de Manutenção de Stock, em Português) [6]. Esse código de identificação é muitas vezes retratado como um código de barras legível por máquinas que ajuda a rastrear o item para inventários. Note-se que um SKU não identifica o bem tangível, mas antes todos os bens que contêm as mesmas características. No exemplo 2.1.1. está ilustrado um SKU.

Exemplo 1
Imagine-se um armário com:

- 2 pacotes de leite magro da marca X;
- 2 pacotes de leite magro da marca Y;
- 1 pacote de leite meio gordo da marca X.

Esse armário contém 3 SKUs, uma vez que um SKU se distingue pelo tamanho, cor, sabor, marca, etc.

Os itens que se mantêm em stock físico na loja são identificados como **Stock Item (Item em Stock, em Português)** [7]. O item em stock tem uma quantidade associada, e de cada vez que uma venda é feita a sua quantidade é reduzida. Ler mais em [8]. A taxonomia de

classificação que subdividem um setor ("yet another market construct") nos diferentes tipos de produtos para os quais existe demanda denomina-se por **Category (Categoria, em Português)**. Quanto mais especializada for uma categoria, mais especializado é o produto.

Nota: Neste projeto apenas se consideram as categorias de maior dimensão, são elas, por exemplo, Laticínios, Bebidas, Frescos, Congelados, entre outras.

Como distinção de produtos entre empresas, recorre-se a um símbolo de identificação, marca, logótipo, nome, palavra e/ou frase, a que se dá o nome de **Brand (Marca, em Português)** [9]. Quando os estrategistas de marca falam sobre **Segmentation (Segmento, em Português)**, referem-se à segmentação do consumidor/audiência. A maneira antiga de abordar isto era através da demografia (idade, sexo, etnia, faixa de renda, urbano-rural, etc.). Agora a segmentação é VALS (valores, atitudes e estilo de vida). Ler mais em [7].

Nota: Neste projeto o segmento é a quantidade presente numa embalagem, i.e., para um pacote de leite de 1L, o segmento é 1L.

A **Variety (Variedade, em Português)** é confusa porque pode ser difícil de entender onde a especialização do segmento termina e a especialização em prol da variedade começa. A variedade é sobre a personalização de um produto para se adequar ao caráter do consumidor individual. Ver exemplo 2.1.2. Ler mais em [7].

Exemplo 2
Note-se um pacote de leite com as caraterísticas, quantidade líquida igual a 1L, da marca X e do tipo UHT magro. Então, identificar-se-ia da seguinte forma:

- Categoria: Laticínios
- Produto: Leite
- Marca: X
- Segmento: 1L
- Variedade: UHT Magro

2.2 Sistema Smart Stocks

Smart Stocks é um sistema que visa dar suporte à gestão de stocks domésticos. Para tal é necessário recolher determinadas informações, tais como, as características da casa a gerir, as particularidades dos membros co-habitantes da casa e ainda os padrões de consumo e reposição. De forma a facilitar tal tarefa são disponibilizadas listas geridas pelo sistema. Por exemplo, lista de compras e lista dos itens em stock na casa, cuja consistência é garantida às custas dos movimentos de entrada e saída dos itens nos diversos locais de armazenamento.

2.2.1 Requisitos

Como forma de assegurar as funcionalidades pretendidas para o sistema, são exigidos os seguintes requisitos funcionais e opcionais.

Requisitos Funcionais

- Informar o utilizador dos itens existentes, a sua validade e a sua quantidade;
- Alertar sobre os produtos que estão perto do fim;
- Gerar listas de compras com os produtos em falta;
- Possibilidade de criar listas com produtos e suas quantidades;
- Partilhar listas entre utilizadores da mesma casa;
- Especificar alergias dos membros da casa.

Requisitos Não Funcionais

- Desenhar interfaces com o utilizador de fácil utilização;
- Desenhar interfaces com o utilizador com design apelativo;
- Internacionalizar as interfaces com o utilizador.

Requisitos Opcionais

- Criar listas de produtos quase a expirar;
- Criar listas de produtos indesejados (Lista Negra);
- Criar listas de contenção em situações de emergência (Lista SOS);
- Inserir refeições extraordinárias de eventos a realizar num futuro próximo, para acrescentar alimentos não básicos à lista de compras.
- Notificar o utilizador da proximidade do fim da data de validade de um determinado produto.
- Notificar o utilizador para a rotura iminente da quantidade de um determinado produto.

2.3 Arquitetura da Solução

Nesta secção pretende-se abordar de forma geral a solução implementada para resolver o problema apresentado no capítulo 1.

2.3.1 Abordagem

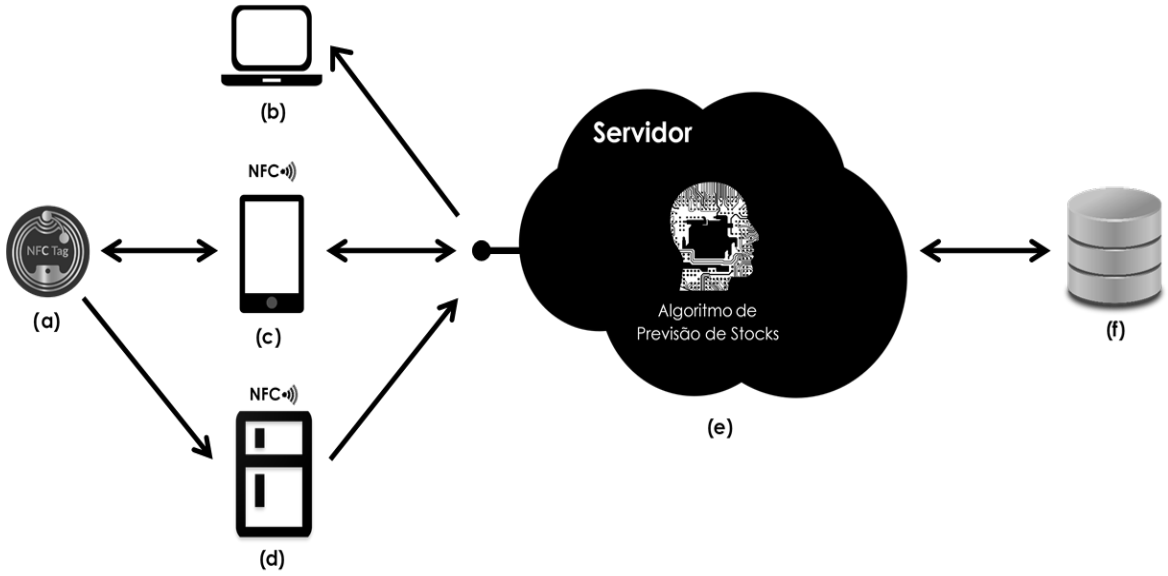


Figura 2.1: Arquitetura Geral do Projeto

Após uma ida às compras, os itens adquiridos com rótulos não tradicionais, Figura 2.1(a), são armazenados nos seus respetivos locais, Figura 2.1(d). Como forma de automatizar a recolha de informação relativa quer aos artigos obtidos quer às suas caraterísticas, utilizam-se rótulos digitais e sensores.

Ao guardar os artigos nos locais de armazenamento, os seus rótulos devem ser lidos por dispositivos de hardware, conjunto sensor mais leitor de rótulos digitais, presentes no local, de forma a que a informação e identificação do item, bem como, o tipo de movimento (entrada ou saída) possam ser enviados para a componente servidora, Figura 2.1(e). Assim, estes dados são posteriormente tratados e armazenados de forma persistente na Base de Dados (BD), Figura 2.1(f). A componente servidora é responsável por retornar dados para as aplicações cliente, Figura 2.1(b, c). É ainda nesta que está presente o algoritmo de previsão de stocks utilizado para efetuar a previsão quanto à duração de cada um dos itens em stock, assim como, o controlo da gestão de stocks.

No contexto da gestão de stocks assume-se a existência de duas formas de apresentação para os itens em stock: avulsos e embalados. Os primeiros são conservados em sistemas de arrumação identificados com *tags* programáveis por *smartphones*, 2.1(c). Os detalhes dos itens são especificados pelo utilizador e carregados para a *tag*. Já os segundos contêm os seus rótulos digitais com o detalhe guardado pelos embaladores.

2.3.2 Rótulos dos Itens em Stock

Requisitos Legais dos Rótulos

Segundo o Regulamento (UE) nº1169/2011 [10] os rótulos devem conter a seguinte lista de informação:

- Denominação da venda;
- Listas de ingredientes;
- Quantidades de ingredientes ou das categorias de ingredientes;
- Quantidade líquida;
- Data de durabilidade mínima (DDM)/ Data limite de consumo (DLM);
- Condições especiais de conservação e utilização;
- Nome ou firma e endereço fabricante, do acondicionador ou do vendedor;
- País de origem ou de proveniência;
- Instruções de utilização;
- Referência ao teor alcoométrico volúmico adquirido.

Rótulos Tradicionais *vs* Rótulos Digitais

Os códigos de barras são amplamente utilizados na identificação de produtos, quer seja dentro da própria organização, quer seja quando a empresa produtora pretende vender os seus produtos no mercado. Neste último caso, a codificação dos produtos, sendo uma obrigação do mercado, segue as normas da organização GS1[1], organização responsável pelo sistema de Normas Globais de Identificação e Codificação de bens e serviços mais utilizado no mundo. A GS1 Portugal é a entidade competente geradora e reguladora da atribuição dos códigos de barras em Portugal. É garantido que não existem dois produtos com o mesmo código de barras em circulação quer a nível nacional, quer a nível global.

São três os componentes relevantes identificados por um código de barras: o país de origem, a empresa fabricante e o produto produzido, ilustrado na Figura 2.2 (um exemplo fictício).

[1] https://www.gs1.org/

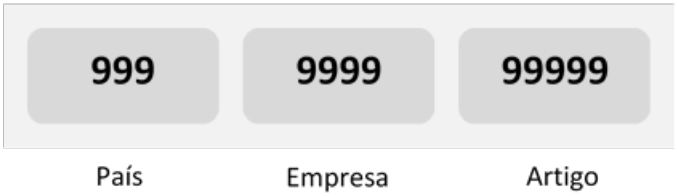


Figura 2.2: Exemplo proposto de código de barras

A gestão de stocks do sistema Smart Stocks necessita de saber o nome do produto, a marca, a variedade, o segmento, a data de validade, os alergénios, a quantidade e, opcionalmente, as condições de conservação. Logo, a informação presente nos códigos de barras é insuficiente para o correto funcionamento do sistema. Como tal, existiu a necessidade de encontrar uma nova abordagem que solucionasse o problema. Uma solução possível passa pela utilização de um leitor de imagens ou de objetos 3D, capaz de ler a informação presente no rótulo tradicional. No entanto, uma outra hipótese é a utilização de rótulos digitais, por exemplo, recorrendo a *tags Near-Field Communication* (NFC) [11] ou *Radio-frequency Identification* (RFID) [12]. Como os produtos avulsos têm de ser rotulados, decidiu-se usar *tags* programáveis por *smartphones*. Este processo torna-se prático e acessível a muitos. Ora, uma vez que esta tecnologia é utilizada para os produtos avulsos e sendo uma das soluções possíveis para os produtos embalados, então, uniformiza-se a automatização utilizando a mesma tecnologia nas duas circunstâncias.

Comparação entre *Tags* NFC e RFID

Tabela 2.1: Comparação da tecnologia NFC com a tecnologia RFID

	NFC	**RFID**
Gama de frequência	13,56MHz	125kHz - 960MHz
Comunicação	Unidirecional ou bidirecional (P2P)	Unidirecional
Distância	até 5cm	até 100m
Componentes	Leitor NFC e *tag* NFC	Leitor RFID, *tag* RFID e uma antena
Suporte em *smartphones*	Na maioria dos *smartphones*	Não
Ativo/Passivo	Passivo, ativado na presença de um leitor NFC	Passivo, ativado na presença de um leitor RFID e Ativo, a *tag* tem uma fonte de energia própria.
Uso aplicacional	Propriedade desenvolvidas para pagamentos móveis seguros	Usado globalmente para gestão de stocks, manipulação de bagagem no aeroporto, identificação de gado entre outros
Capacidade das *tags*	64 Bytes até 1024 Bytes	96 Bits até 512 Bits ou até 4k ou 8k Bytes

Conforme se pode observar na tabela 2.1, a comunicação com as *tags* NFC pode ser bidirecional o que torna mais fácil e intuitiva a transmissão entre telemóveis e *tags* NFC. Desta forma, optou-se pela utilização de *tags* NFC para os produtos avulsos em que é preciso usar o telemóvel para as programar, e, atualmente, os telemóveis só têm suporte para a tecnologia NFC. Assim os locais de armazenamento têm obrigatoriamente de dispor de leitor de *tags* NFC. Contudo, se os embaladores decidirem utilizar RFID, o hardware pode também ter leitor RFID. O levantamento de requisitos feito, juntamente com a dimensão dos campos da base de dados, e sabendo que o formato utilizado na escrita das *tags* é *Comma-separated values* [13], estimou-se que a capacidade de armazenamento mínima das *tags* é aproximadamente

324 Bytes.

2.3.3 Dispositivos de *Hardware*

Posto que a componente de *hardware*, sensores e leitores de *tags*, não foi âmbito do projeto, mas é parte integrante e essencial para o correto funcionamento do sistema desenvolvido, foi fundamental efetuar análise e pesquisa alusiva ao assunto. Por isso, estudou-se como se deveria proceder para incorporar os dispositivos *hardware* no sistema Smart Stocks.

Assim, para adicionar um dispositivo de *hardware* usar-se-ia um *QRCode* [14]. Este *QRCode* deveria conter informação acerca do dispositivo, como o seu identificador e possivelmente o local de armazenamento a que se destina. Este ao ser comprado e sem ter sido ainda instalado no local de armazenamento, seria adicionado à casa, pelo utilizador, recorrendo a uma funcionalidade extra da aplicação móvel. O utilizador passaria o *scanner QRCode* sobre o *QRCode* presente no dispositivo de *hardware* e assim a aplicação móvel comunicaria com a API *Web* para associar aquele dispositivo à casa do utilizador. Caso este utilizador tenha várias casas, este teria de, previamente, especificar a qual casa pretendia associar o dispositivo. Uma vez associado, o dispositivo necessita de comunicar com a API *Web* de forma a registar os movimentos presentes ocorridos naquele local de armazenamento. Para tal, quando o dispositivo for instalado e ligado, este faria um *ping* à API *Web* a sinalizar que está ativo e enviando a sua informação, nomeadamente o seu identificador. A API *Web* respondia com o link ao qual o dispositivo *hardware* teria de enviar a informação dos movimentos que deteta.

2.3.4 Arquitetura por Camadas

O sistema de gestão de stocks é composto por 2 blocos principais: o bloco do lado do cliente e o bloco do lado do servidor, que se relacionam. A representação destes blocos é apresentada na Figura 2.3.

A arquitetura do projeto segue uma arquitetura por camadas, dado que este padrão permite individualizar cada camada [15]. Assim, estas tornam-se independentes umas das outras, fornecendo não só abstração sobre as camadas inferiores, mas também, oferecendo a possibilidade de testar e/ou substituir cada uma das camadas de forma independente, desde que seja mantido o contrato.

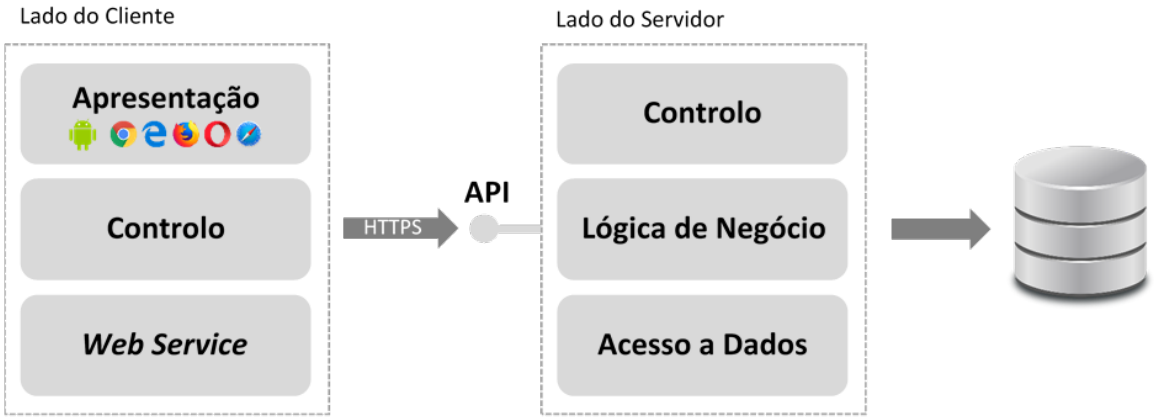


Figura 2.3: Arquitetura por Camadas do Projeto

No lado do cliente existem três camadas: a camada Apresentação que é responsável por representar os dados solicitados pelo utilizador; o Controlo que está encarregue de despoletar ações na camada do *Web Service* de forma a satisfazer as solicitações do utilizador; e assim o *Web Service* interage com a API *Web*.

As camadas que compõem o lado do servidor são: o Controlo que processa pedidos e retorna uma resposta; a camada da Lógica de Negócio que é responsável por satisfazer as regras de negócio; e por fim o Acesso a Dados que efetua leituras e escritas sobre a BD.

Tecnologias Inerentes à Solução

O lado do servidor incluí três camadas e expõe uma API *Web*. A Camada de Acesso a Dados (DAL) é produzida com a linguagem de programação *Java*, usando a *Java Persistent API* (JPA), e é responsável pelas leituras e escritas sobre a Base de Dados (BD). A BD é externa ao servidor, utilizando para isso o Sistema de Gestão de Base de Dados (SGBD) *PostgreSQL*. A Camada da Lógica de Negócio (BLL) é responsável pela aplicação das regras de negócio. A implementação desta camada é também realizada com linguagem *Java*. Os *controllers* foram desenvolvidos em *Java* com a *framework* da *Spring*, chamada de *Spring Boot*[1]. A API *Web* disponibiliza recursos em diferentes *hypermedias*. Para a implementação do algoritmo de previsão de stocks usou-se a linguagem *R*.

Do lado do cliente existem dois modos de interação: usando uma aplicação móvel ou usando uma aplicação web. A aplicação móvel está disponível para a plataforma *Android*, e foi desenvolvida na linguagem *Kotlin*. A aplicação web é compatível com a maioria dos *browsers*, e é implementada utilizando a linguagem *JavaScript* com o auxilio da biblioteca *React*[2].

[1] https://spring.io/projects/spring-boot
[2] https://reactjs.org

2.4 Entidades

Em seguida identificam-se as diversas entidades relevantes que compõem o sistema de informação, que permite gerir os itens em stock numa dada casa.

Casa

- Cada casa é caracterizada por um identificador único, um nome, atribuído por um utilizador no momento de registo da casa.
- Deve ser possível saber o número de bebés, crianças, adultos e seniores que vivem nessa casa.
- Uma casa está associada a um ou mais utilizadores, podendo um utilizador ter várias casas.
- Existe um ou mais administradores de uma casa.
- A casa pode ter vários itens em stock.
- Para cada casa existem vários locais de armazenamento dos itens, por exemplo armários, frigoríficos, etc.
- Em cada casa deve ser possível conhecer as alergias assim como quantos membros possuem essa alergia (os membros não precisam necessariamente de estar registados).

Utilizador

- Uma pessoa é representada por um utilizador que é caracterizado por um email ou por um nome de utilizador, pelo nome da pessoa, a sua idade e uma *password*.

Listas

- Cada lista é composta por um identificador único e um nome.
- Uma lista pode ter vários produtos.
- Existem dois tipos de listas: de sistema e de utilizador.
- As listas de sistema são comuns a todos os utilizadores registados, contudo são particulares a cada casa.
- Um utilizador pode criar as suas listas, partilhando-as com outros utilizadores da casa ou tornando-as secretas.

Categoria

- Uma categoria é identificada univocamente por um número ou por um nome.

Produtos

- Um produto é constituído por um identificador único, um nome, se é ou não comestível, e a validade perecível.
- Para os produtos presentes numa lista pode ser possível saber a sua marca e a quantidade.
- Um produto pertence a uma categoria, podendo uma categoria ter vários produtos.
- Um produto pode ser concretizado por diversos itens em stock na casa.

Item em Stock

- Um item em stock é a concretização de um produto que existe numa casa. É identificado univocamente por um número ou por uma marca, uma variedade e um segmento, é também caracterizado por uma descrição, o local de conservação, a quantidade e as datas de validade.
- Para cada item deve ser possível saber os seus movimentos de entrada e saída de um local de armazenamento.
- Deve também ser possível saber os alergénios de cada item presente na casa.

Movimento

- Para cada movimento deve ser possível saber o tipo do movimento (entrada ou saída), a data em que ocorreu e a quantidade de produtos.

Local de armazenamento

- Cada local de armazenamento é caraterizado por um identificador único, a temperatura e um nome.
- Deve ser possível saber a quantidade de cada item presente no local.
- Um local de armazenamento pode ter vários itens em stock presentes na casa e estar associado a diversos movimentos.

Capítulo 3

Implementação do Sistema de Gestão de Stocks

O presente capítulo retrata a implementação do sistema de gestão de stocks, *Smart Stocks*. Este subdivide-se nas seguintes secções: 3.1) Modelo de Dados, 3.2) Aplicações Cliente, 3.3) Componente Servidora, 3.4) Algoritmo de Previsão de Stocks.

3.1 Modelo de Dados

3.1.1 Base de Dados

Os dados são armazenados de forma persistente numa Base de Dados (BD). A BD utilizada é relacional uma vez que não se preveem alterações ao esquema das tabelas, não carecendo do dinamismo oferecido por uma BD documental, por exemplo.

A escolha de qual o melhor Sistema de Gestão de Base de Dados (SGBD) assentava em três possibilidades, *SQL Server*, *PostgreSQL* e *MySQL*. O primeiro apesar de ser uma ferramenta com a qual o grupo estava familiarizado foi automaticamente excluído visto que um dos requisitos pretendidos era usar ferramentas *open source*, caraterística não presente neste sistema. Os restantes sistemas são *open source* e têm uma elevada compatibilidade com os principais fornecedores de serviços *cloud*. Pelo que a verdadeira distinção se prende com as vantagens oferecidas pelo sistema *PostgreSQL*:

- O *PostgreSQL* é compatível com as propriedades *Atomicity, Consistency, Isolation, Durability* (ACID), garantindo assim que todos os requisitos sejam atendidos;
- O *PostgreSQL* aborda a concorrência de uma forma eficiente com a sua implementação de *Multiversion Concurrency Control* (MVCC), que alcança níveis muito altos de concorrência;
- O *PostgreSQL* possui vários recursos dedicados à extensibilidade. É possível adicionar novos tipos, novas funções, novos tipos de índice, etc.

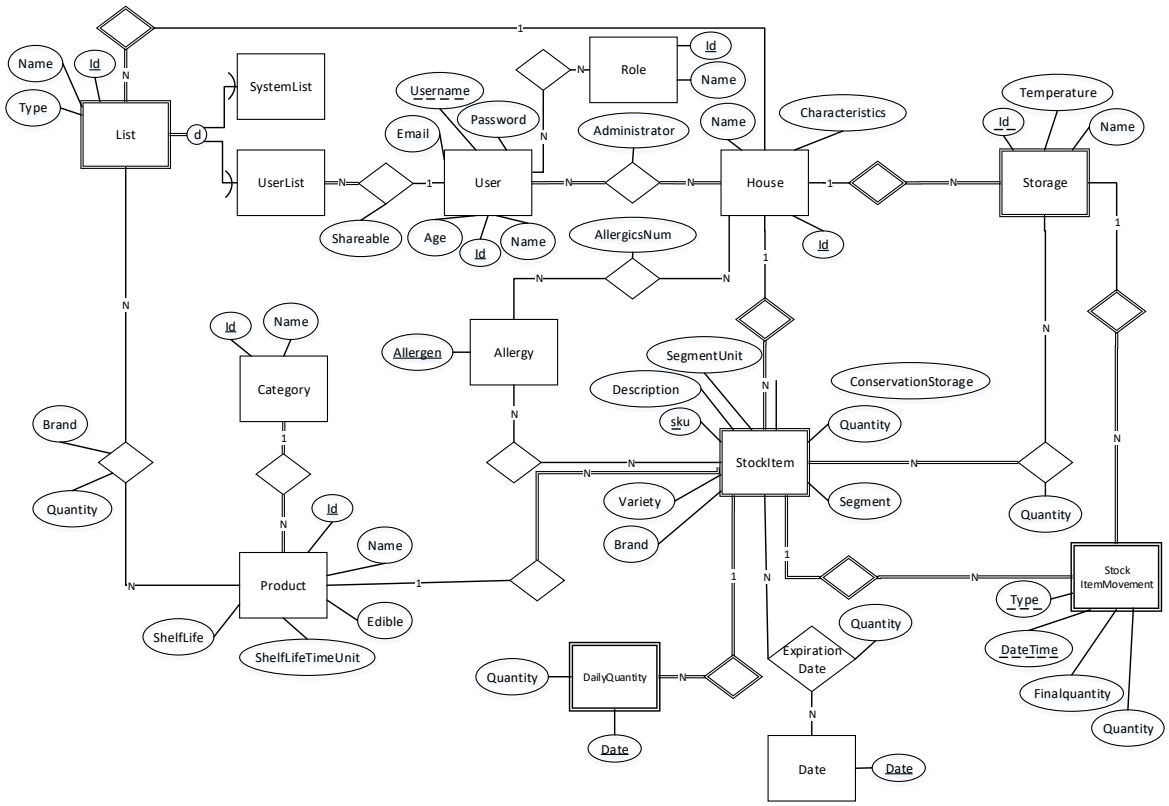


Figura 3.1: Modelo Entidade-Associação

Assim sendo, foi escolhido o Sistema de Gestão de Base de Dados Relacional de Objetos (SGBDRO) *PostgreSQL*, como já anteriormente mencionado, na secção 2.3.4 do capítulo 2.

Implementação

Na BD foram desenvolvidas funções que garantem a consistência dos dados, por um lado na inserção de entidades cujos *IDs* sejam incrementais ou gerados consoante o desejado, por outro lado na remoção de entidades que se relacionam com outras.

Decidiu-se usar funções na BD em vez de criar métodos em *Java*, pois se imaginarmos um cenário onde a aplicação servidora esteja distribuída, existe um problema no controlo da concorrência na geração dos *ID*s. Tendo em conta que a BD não é distribuída então o problema descrito não existe.

3.1.2 Modelo Entidade-Associação

Com Figura 3.1 é possível entender as relações das entidades referidas na secção 2.4 do capítulo 2. Para um maior detalhe do domínio de cada uma das entidades consultar o anexo A.1.

3.1.3 Particularidades do Modelo de Dados

Alergias dos Membros de uma Casa

O sistema Smart Stocks disponibiliza um leque de funcionalidades, porém o foco do projeto prendeu-se com a parte relativa à gestão de stocks, deixando a parte da administração de casas e utilizadores para segundo plano. Contudo pensaram-se em alternativas, para implementar no futuro, que permitam que a gestão de utilizadores e casas seja mais pessoal e funcional.

Um dos aspetos a ter em conta, seria a manutenção da consistência das alergias numa casa, aquando um utilizador é removido desta. Este aspeto é importante, pois o modelo de dados desenvolvido não garante essa coerência, visto que as alergias estão associadas a uma casa e aos seus membros e não a um utilizador do sistema. Tomou-se esta decisão, uma vez que faria mais sentido, na altura, as alergias estarem associadas a uma casa e ao seu conjunto de membros, que podem ou não estar registados no sistema. A razão desta escolha baseia-se na possibilidade de registar as alergias dos co-habitantes da casa, mesmo que estes não sejam utilizadores do sistema.

Um bom exemplo que apoia esta abordagem, é o caso das crianças, desta forma, os seus pais registam as alergias dos mais pequenos e conseguem melhor controlar o plano alimentar familiar. Imagine-se uma casa com três habitantes, dois dos quais intolerantes à lactose, e apenas um está registado no sistema. O contra da solução é não saber qual dos membros detém a alergia, todavia, sabe-se que a mesma existe e assim é possível alertar o utilizador do sistema. Para concluir, esta solução apesar das suas vantagens, está acompanhada de um problema, que é a saída de alguém da casa, ficando a informação relativa às alergias inconsistente. Uma possível solução passa por notificar o administrador da casa para esse facto, e alertá-lo para a necessidade de retificar os dados.

Internacionalização

A internacionalização não é suportada ao nível da base de dados, não tendo por tanto tradução das nomenclaturas das categorias nem dos produtos, nem qualquer informação destes.

3.2 Aplicações Cliente

Disponibilizam-se aplicações cliente em dispositivos móveis *Android* (*smartphones* e *tablets*) e dispositivos *desktop*, através do *browser*. Na implementação das mesmas teve-se em atenção conceitos como *responsive*, pois tal permite, não só, uma melhor experiência de utilizador, como também, uma interface mais apelativa aos diversos dispositivos suportados.

Realizou-se uma pesquisa pelas aplicações já existentes de maneira a estar cientes do que se pode encontrar no mercado e comparar com o que o sistema Smart Stock oferece, como se

Tabela 3.1: Comparação do sistema Smart Stocks com outros

	Smart Stocks	**OutOfMilk**	**Bring!**
Aplicação Móvel	✓	✓	✓
Aplicação Web	✓	✓	✓
Listas de Compras	✓	✓	✓
Listas de Tarefas	✗	✓	✗
Partilha de Listas	✓	✓	✓
Adicionar Itens às Listas	✓	✓	✓
Adicionar Detalhes aos Itens	✗	✓	✗
Produtos Organizados por Categorias	✓	✓	✓
Integração com Dispositivos Inteligentes	✓	✗	✗
Sincronização em Tempo Real	✓	✓	✓
Gestão Automática da Despensa	✓	✗	✗
Gestão de Múltiplas Casas por Utilizador	✓	✗	✗
Leitura de Rótulos Digitais	✓	✗	✗
Previsão de Stocks	✓	✗	✗

pode observar na tabela 3.1.

3.2.1 Aplicação Móvel

Neste momento, a aplicação móvel está disponível apenas para a plataforma *Android*, visto que, segundo um estudo do mercado de *smartphones* de 2017 [16], 86.2% da quota de mercado pertence a utilizadores de *Android*, logo, nesta primeira fase faz mais sentido dirigir a aplicação ao maior número de utilizadores.

De seguida, analisando a distribuição das diversas versões da plataforma *Android* [17], optou-se por permitir compatibilidade desde a API 15 até à API atual, 27, acumulando assim 99.7% dos dispositivos móveis.

Interface com o Utilizador

Antes de desenvolver a interface com o utilizador foi efetuada pesquisa relativa às boas práticas de desenho de componentes *Android*, leram-se diretrizes presentes no Material Design [18], e ainda, investigaram-se diversas aplicações móveis de forma a extrair particularidades que melhorassem a experiência de utilização. Nas Figuras 3.2 a 3.8 apresentam-se os esboços iniciais das *user interfaces*.

Já durante o desenvolvimento da componente gráfica priorizaram-se técnicas de *responsiveness*, i.e., adequação dos vários *layouts* às múltiplas dimensões dos dispositivos suportados.

Figura 3.2: Esboço 1

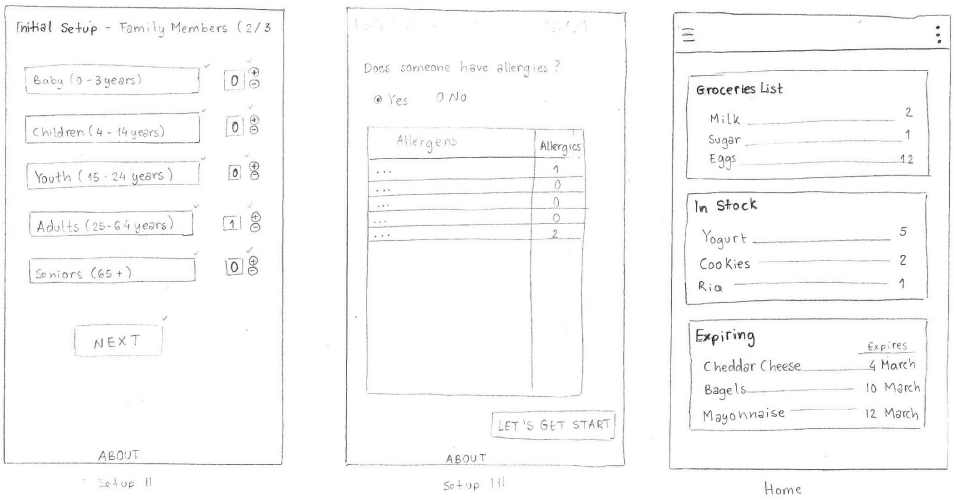


Figura 3.3: Esboço 2

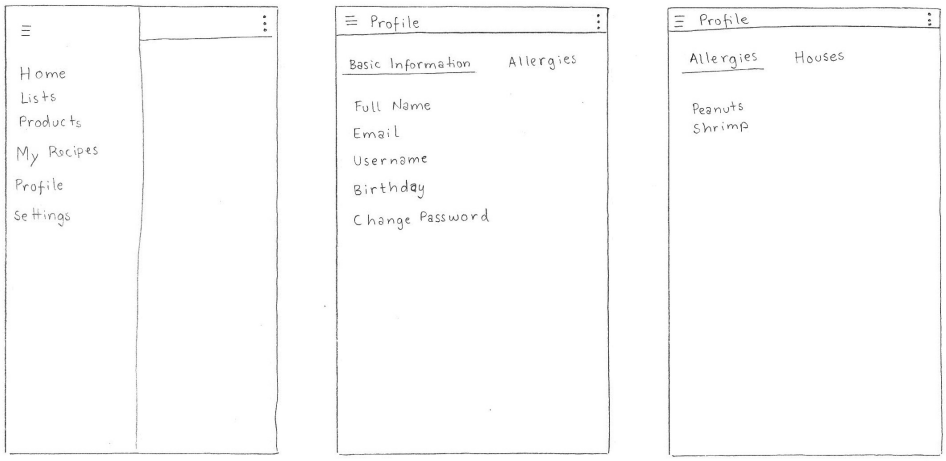
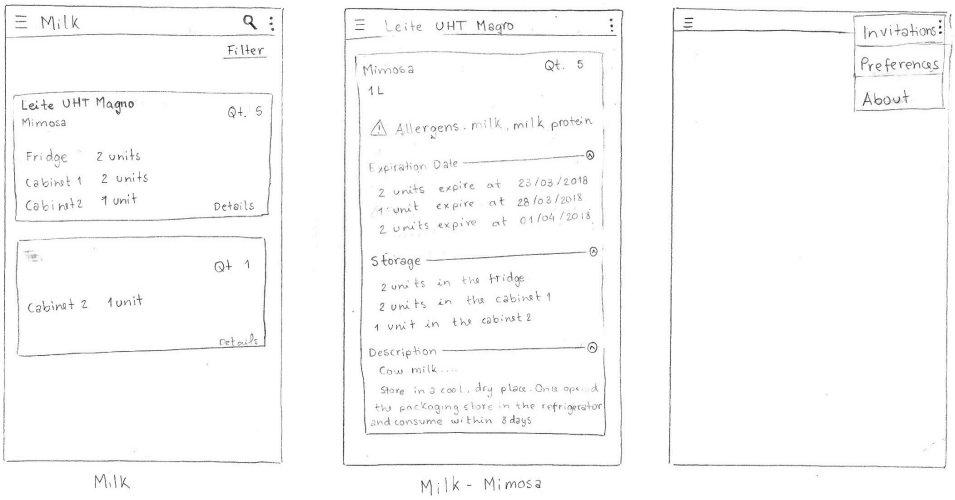


Figura 3.4: Esboço 3



Milk

Milk - Mimosa

Figura 3.5: Esboço 4

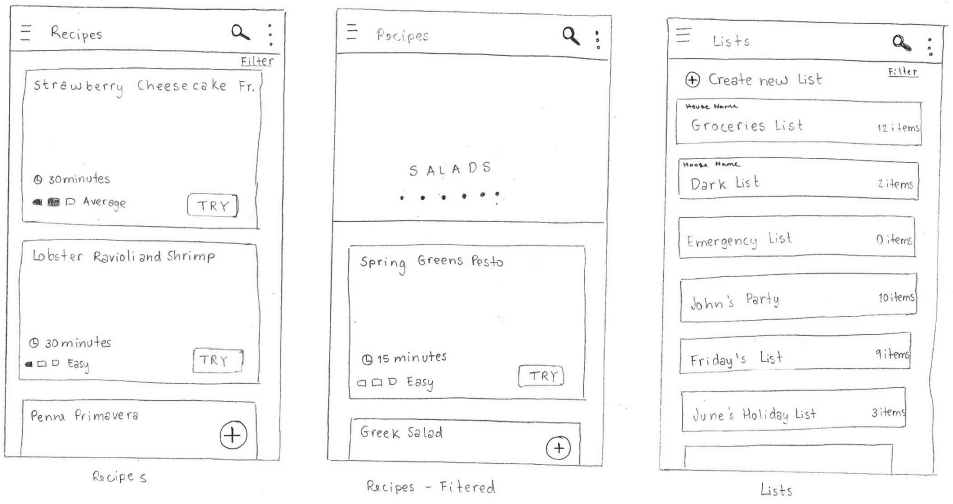


Figura 3.6: Esboço 5

Figura 3.7: Esboço 6

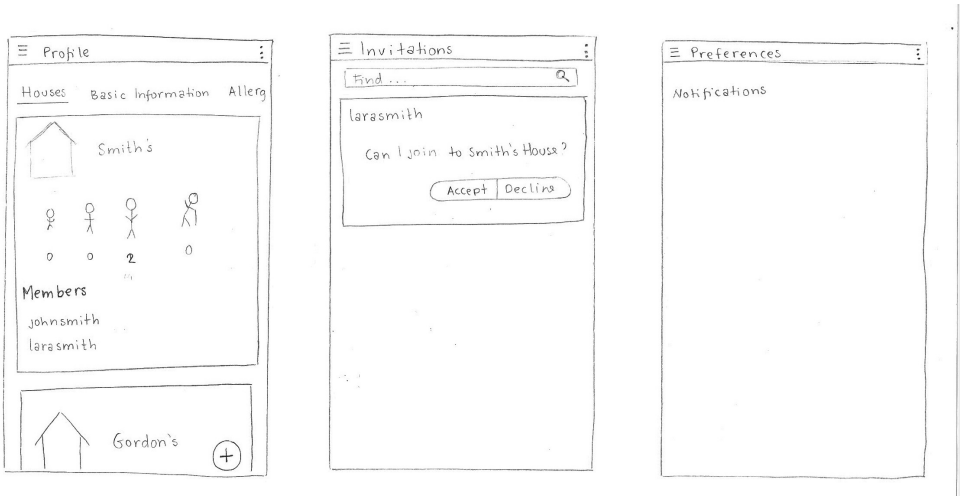


Figura 3.8: Esboço 7

Padrões de Desenho

MVC *vs* MVP *vs* MVVM

Os padrões de desenho em *Android* têm vindo a evoluir, de forma a tornar os componentes independentes e facilmente testáveis. Um dos primeiros padrões a surgir foi o padrão *Model-View-Controller* (MVC), porém nos últimos anos, emergiram dois novos padrões, a destacar o *Model-View-Presenter* (MVP) e o *Model-View-ViewModel* (MVVM). Estes padrões elevaram-se com o propósito de evitar a dispersão da lógica de negócio entre as camadas de dados e as gráficas, logo, reduzindo a duplicação de código. Assim como, facilitam a manutenção dos diversos módulos e a realização de testes unitários. Outro fator de relevo é o facto de permitirem trocar rapidamente de repositório de dados, seja este uma base de dados ou APIs.

Na tabela 3.2 apresenta-se uma comparação entre os três padrões, pretendendo-se tornar percetível as vantagens e desvantagens de cada um.

Tabela 3.2: Comparação dos Padrões MVC, MVP e MVVM

	MVC	**MVP**	**MVVM**
Separação entre o *Model* e a *View*	✓	✓	✓
Separação entre a *View* e o *Controller*	✗	n.a.	n.a.
Separação entre a *View* e o *Presenter*	n.a.	✓	n.a.
Separação entre a *View* e o *ViewModel*	n.a.	n.a.	✓
Facilidade em testar o *Model*	✓	✓	✓
Facilidade em testar a *View*	✓	✓	✓
Facilidade em testar o *Controller*	✗	n.a.	n.a.
Facilidade em testar o *Presenter*	n.a.	✓	n.a.
Facilidade em testar o *ViewModel*	n.a.	n.a.	✓
Facilidade na manutenção do *Controller*	✗	n.a.	n.a.
Facilidade na manutenção do *Presenter*	n.a.	✗	n.a.
Facilidade na manutenção do *ViewModel*	n.a.	n.a.	✗

n.a. - Não aplicável

Service Locator

Ao contrário do padrão atualmente muito utilizado, da injeção de dependências, o *Service Locator* [19] permite obter instâncias de serviços, através de um único ponto central na aplicação. É ainda possível trocar de implementações em *runtime* consoante o dispositivo onde a aplicação está a ser utilizada.

Internacionalização

Na interface com o utilizador são disponibilizados dois idiomas, Português e Inglês. A internacionalização de uma aplicação em *Android* é relativamente simples, uma vez que existem recursos para cada língua suportada [20], sendo tarefa do *Android* determinar qual o recurso de strings a utilizar consoante a língua local associada ao telemóvel.

Segurança

A segurança na aplicação móvel *Android* é assegurada de duas formas, recorrendo às *Shared Preferences* e realizando a autenticação através dos pedidos à API *Web*. Armazenar *passwords* no *Android* não é uma tarefa simples, uma vez que este não disponibiliza muitos locais onde se possa armazenar informação sensível. As hipóteses consistiram em armazenar nas *Shared Preferences*, numa base de dados *SQLite* ou no *Keystore* do dispositivo. Existem diferentes formas de efetuar o armazenamento das *passwords*, podendo estas serem guardadas em claro, encriptadas com uma chave simétrica, usando o *Android Keystore*, ou encriptadas com uma chave assimétrica [21].

As *passwords* armazenadas em claro nas *Shared Preferences* ou numa base de dados *SQLite* não é de todo uma boa ideia, pois se um atacante conseguir aceder ao telemóvel obtém acesso ao ficheiro das *Shared Preferences* ou à base de dados, conseguindo, assim, obter a *password* armazenada. Encriptando a *password* com uma chave simétrica, também, não se trata de uma boa ideia pois a chave iria ficar armazenada no Android Package (APK). Se um atacante fizer o *decompile* do APK pode encontrar a chave e assim desencriptar a password. Para além de que a encriptação da *password* consistiria num custo adicional.

A outra opção passaria por usar o *Android Keystore* para encriptar as palavras-chave utilizando uma chave assimétrica. Porém, mais uma vez um atacante com acesso privilegiado (*root*), visto que o *Android* se baseia num sistema *Linux*, conseguiria localizar a chave privada e assim aceder à *password*. Uma opção mais viável seria armazenar a chave privada remotamente e sempre que fosse necessário desencriptar a *password*, esta seria remetida para o servidor que compreende a chave privada para a desencriptação. Esta opção, também, não é a melhor pois necessita de aceder à *password* do utilizador sempre que se realizar um pedido à API *Web*, visto que é necessário enviar no *header* HTTP *Authorization* as credenciais do utilizador. Outro contra, é no momento do *login* ser necessário fazer um pedido ao servidor fornecedor da chave privada para obter a *password* e realizar o *login* na API *Web*, o que

limitaria o futuro da aplicação móvel no sentido de só ser possível o uso desta com Internet.

Existe ainda outra solução, que seria usar o *Smart Lock for Passwords* da *Google* [22]. Esta foi, igualmente, descartada pela razão anterior de que é necessário ter acesso à Internet para obter a *password*.

A solução implementada foi armazenar as *passwords* em claro nas *Shared Preferences*, no entanto, este é, sem dúvida, um ponto a melhorar futuramente, por exemplo, passando por utilizar um servidor de autorização do sistema Smart Stocks, em que os utilizadores podem iniciar sessão e em caso de sucesso seria armazenado um *token* na aplicação móvel.

As credenciais são solicitadas ao utilizador no *loggin* e no registo. No caso destes serem sucedidos, a sessão fica iniciada ao longo do tempo, dado que as suas credenciais ficam registadas nas *Shared Preferences*.

O início de sessão é realizado efetuando um pedido à API *Web*, este pedido é autenticado através do *header Authorization* e fazendo uso das credenciais inicialmente solicitadas. Em caso do pedido suceder significa que o *loggin* pode ser efetuado com sucesso, caso contrário verifica-se se pode ter sido um erro de credenciais incorretas para assim puder informar o utilizador, ou se se trata de um outro problema.

Implementação

A aplicação móvel foi desenvolvida com a linguagem *Kotlin*, pois esta linguagem é:

- concisa,
- *null-safe*, por omissão,
- interoperável, existindo enumeras bibliotecas da JVM, do *Android* e dos *browsers*,
- suportada por vários IDEs,
- empresas como o *Pinterest*, a *Uber*, a *Evernote*, etc. estão a utilizar linguagem Kotlin em diversas componentes das suas aplicações e programas,
- entre muitas outras vantagens [23].

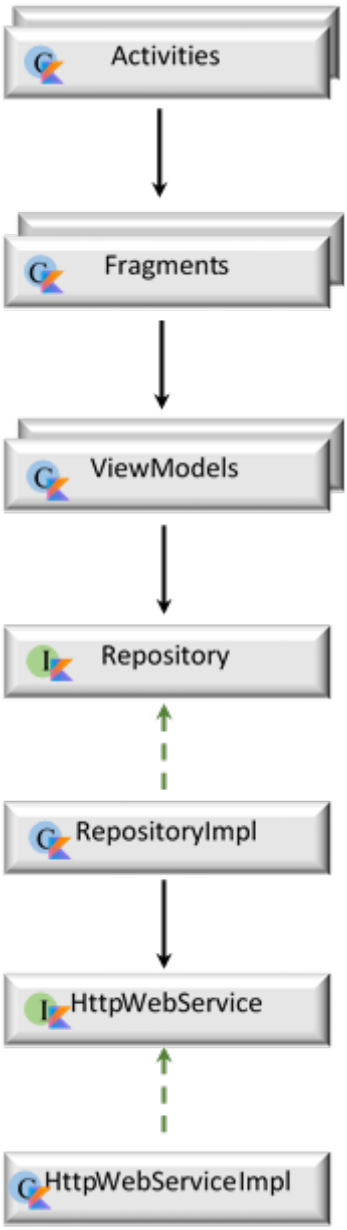


Figura 3.9: Arquitetura da Aplicação Móvel

Na Figura 3.9 expõe-se o *Unified Modeling Language* (UML) geral dos componentes da aplicação móvel. Assim de uma forma geral, a arquitetura é composta por *Activities*, sendo a principal a controladora das transações entre fragmentos. Estes últimos utilizam *ViewModels* na obtenção, inserção e remoção de dados, registando-se como observadores da resposta. Os *ViewModels* recorrem ao Repositório para efetuarem as operações básicas CRUD. É no repositório que se definem os métodos HTTP a utilizar e todas as partes integrantes de um pedido HTTP. Por fim o pedido é efetuado por uma implementação de *HttpWebService*, responsável por adicionar os pedidos a uma fila. Para a realização de pedidos à API *Web* utilizou-se a biblioteca *Volley* [24], esta biblioteca permite realizar pedidos de forma rápida,

simples e assíncrona a *REST-Clients*.

3.2.2 Aplicação *Web*

A aplicação *web* foi pensada como uma aplicação de consulta, onde facilmente os utilizadores podem aceder, através dos diversos *browsers* existentes, aos dados relativos às suas casas. Primeiramente pensou-se em desenvolver a aplicação recorrendo ao ambiente *Node.js* com o auxílio da *framework Express*, contudo esta deve ser utilizada para desenvolver API *Web* o que não era o pretendido. Deste modo, optou-se pelo o uso da biblioteca *React*, que é amplamente empregue na criação de *user interfaces* [25].

Internacionalização

A aplicação *Web* apenas está disponível no idioma Português.

Segurança

À semelhança da aplicação móvel a autenticação é efetuada nos pedidos através do *header Authorization*. A sessão apenas permanece enquanto a janela estiver aberta, para este efeito utiliza-se a propriedade do *Session Storage*.

3.3 Componente Servidora

A API *Web* é a interface exposta pela componente servidora baseada em *Hypertext Transfer Protocol* (HTTP) [26]. Esta disponibiliza informação e funcionalidades fornecidas pelo sistema Smart Stocks, através de *endpoints* públicos e privados.

A decisão de utilizar a *framework Spring Boot* para implementar o servidor deve-se ao facto de ser uma ferramenta *open source*, capaz de criar rapidamente aplicações com auto-configuração, através de anotações. Outra vantagem é a integração com tecnologias de persistência de dados, como a *Java Persistent API* (JPA). Por fim, por questões de conhecimento e de experiência anterior com esta *framework*.

O servidor processa pedidos HTTP por parte das aplicações cliente. Antes dos pedidos serem processados são sujeitos ao controlo de segurança, que é responsável por validar as credenciais presentes no *header Authorization*.

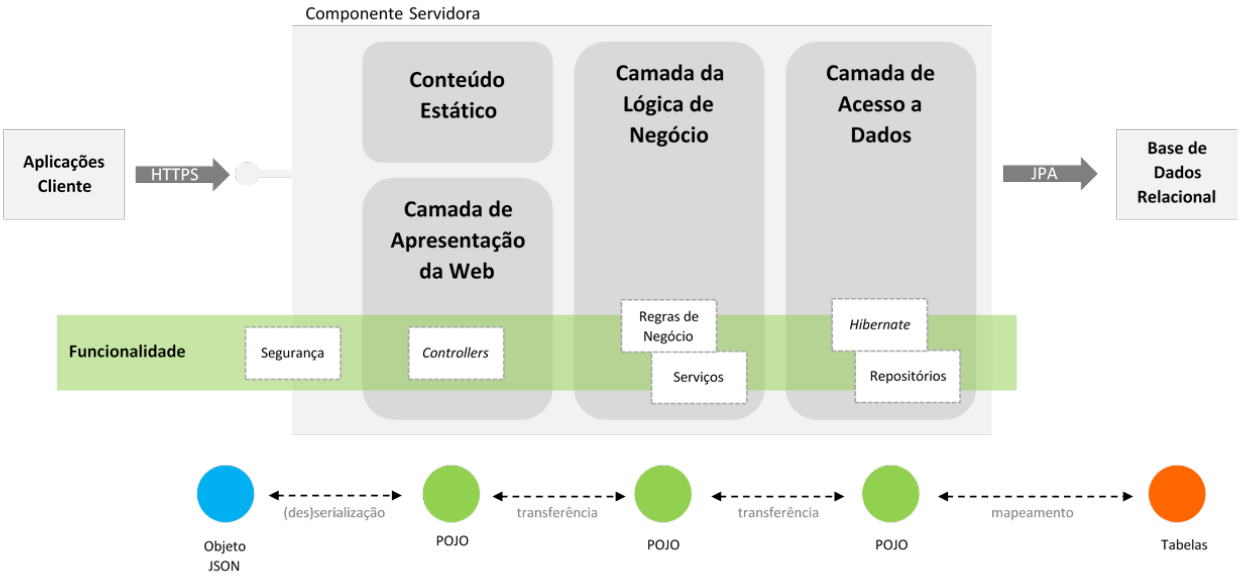


Figura 3.10: Arquitetura da Componente Servidora

Como se pode observar na Figura 3.10 o servidor é constituído por 4 blocos, que se enumeram de seguida. O Conteúdo Estático está encarregue de providenciar os dados inalteráveis, i.e., que são independentes do pedido, como por exemplo, as mensagens de erro e a página inicial, que contém informação relativa à API e os links para os recursos. A Camada de Apresentação da Web é responsável por processar os pedidos aos diferentes *endpoints* e produzir uma resposta. É na Camada da Lógica de Negócio que são verificadas as regras de negócio a aplicar aos pedidos, esta camada constitui assim o aglomerado de serviços que acedem aos respetivos repositórios. Por último, a Camada de Acesso a Dados garante o acesso à base de dados relacional, com uma implementação de JPA, através de repositórios específicos a cada entidade.

À receção de um pedido no servidor os objetos JSON são desserializados em *InputModel* (representações gerais do objeto), de seguida são transformados em objetos POJO, os quais fluem ao longo das camadas descendentes de todo o processo. No final, estes objetos são mapeados em tuplos das tabelas presentes na base de dados. Já no extremo oposto do processamento da resposta ascendente, o objeto POJO é transformado em *OutputModel* (representações específicas do objeto segundo *hypermedia* utilizada) para assim ser serializado num objeto JSON, de forma a poder retornar a resposta ao cliente.

Internacionalização

A API desenvolvida ao longo do projeto tem suporte multi-línguas. Esta funcionalidade é importante pois permite que as aplicações cliente usufruam de mensagens de erro na língua em que for feito o pedido.

Para suportar a funcionalidade de internacionalização faz-se uso do *header* HTTP *Accept-Language*. No entanto, poderia ser utilizado um parâmetro na *query string* para identificar a linguagem pretendida. Porém, segundo a Google [27] não se deve usar *query string* pelo simples facto de que alguns intermediários, por exemplo, sistemas de *cache*, não observam a *query string*, não distinguindo assim os recursos. Uma outra desvantagem em usar um parâmetro na *query string* deve-se a esta não ser *standard*.

Os idiomas disponibilizados são inglês e português, sendo a linguagem inglês a linguagem por omissão. Para aceder às mensagens, o *Spring Boot* disponibiliza uma implementação da interface *MessageSource*, que contém um método para aceder a uma mensagem numa determinada língua.

Logging

Sendo uma grande parte do tempo dos programadores gasta na monitorização, resolução de problemas e *debugging* é interessante manter ficheiros de *Linear Observations Guide* (Log) na manutenção de uma aplicação.

Para cada pedido HTTP recebido na API, é feito *logging* do IP e do porto remoto, do método HTTP, do *uri* do pedido HTTP e de todos os *headers* HTTP à exceção do *header Authorization*. Já na resposta do pedido é feito registo do *status code*, assim como todos os *headers*.

Com a criação de registos sobre a execução das aplicações, a tarefa de inspecionar o software e atuar adequadamente perante erros é facilitada. Visto que, se consegue perceber quais os pedidos e respostas que foram efetuados, de maneira a replicar e resolver eventuais falhas.

Documentação

É de extrema importância ter a API documentada, tal permite que futuros clientes possam empregar-se dessa mesma documentação e interagir com a API.

O desenvolvimento da documentação foi executado com o auxilio do Swagger [28]. Com esta biblioteca especificam-se os diferentes *endpoints* que existem e é gerada uma página HTML com a coletânea de *endpoints*, a partir dos quais é possível realizar testes. Para cada *endpoint* apresentam-se os possíveis *status codes* HTTP a comporem a resposta, bem como, um exemplo de uma representação de um recurso em caso de sucesso. Permite-se ainda, autenticar os pedidos que assim o exigem. Ao testar um *endpoint*, é retornada a resposta e os *headers* HTTP da mesma. Para os pedidos que contêm corpo no pedido HTTP é concedido um exemplo deste e permite-se a inserção de valores para o *endpoint* a ser testado. No caso dos pedidos que necessitam de enviar valores na *path* do *endpoint* é disponibilizada uma caixa de texto para inserir o valor variável.

A página HTML que permite testar os endpoints individualmente está presente em **/v1/documentation/swagger-ui.html**

3.3.1 Segurança

Sendo o Smart Stocks um sistema de gestão de stocks domésticos é de extrema importância assegurar a confidencialidade e a segurança dos dados de cada casa. Como tal, o acesso a recursos ou a manipulação dos mesmos só pode suceder de forma autenticada e autorizada. A solução para a segurança passou por diferentes fases.

Numa primeira fase foi pensado em usar um servidor de autorização já implementado. Para tal ponderou-se fazer-se uso do projeto *MITREid Connect* [29], por questões de familiaridade. Este projeto pode ser usado como *IdProvider*, segundo o protocolo *OpenId Connect* [30], ou como servidor de autorização, segundo o protocolo *OAuth2.0* [31]. Esta opção foi rejeitada uma vez que este projeto não contém uma página de registo para novos utilizadores. Realizou-se uma pesquisa de implementações de servidores de autorização disponíveis para *Java*. Os projetos encontrados foram o *APIs* [32] e *Tokens* [33]. Contudo, estes também não disponibilizam uma página de registo para novos utilizadores, logo a ideia de usar um projeto de um servidor de autorização foi igualmente descartada .

A segunda fase passou por desenvolver um servidor de autorização. No entanto, desenvolver um servidor de autorização é complexo, pelo que existem diversos aspetos de segurança importantes para a realização deste tipo de servidores. Com auxílio do *Spring Security* este processo torna-se mais simples. Porém, ainda existe um aspeto fundamental, que é assinar os *tokens*. Para estes seria útil empregar *Json Web Token* (JWT) [34], permitindo revogá-los. Todavia, implementar estes servidores é moroso e pouco trivial.

A terceira e última fase passou por utilizar outro mecanismo mais básico, o *Basic Scheme* [35]. Ao assumir a utilização do protocolo *Hypertext Transfer Protocol Secure* (HTTPS) [36] (TLS) [37] cria-se um canal com comunicação segura que permite assim utilizar *Basic Scheme*. Com auxílio do *Spring Security* a implementação é facilitada e tem suporte a papéis (*roles*). No sistema Smart Stocks existem dois papéis, o papel *USER* e o papel *ADMIN*. Os utilizadores registados pelo sistema, ficam automaticamente com o papel *USER*. De momento, o papel *ADMIN* não tem qualquer privilégio, mas no futuro este terá funcionalidades de administração do sistema Smart Stocks. Esta foi a solução encontrada, no entanto é um dos aspetos a melhorar futuramente.

Cross-Origin Resource Sharing

Ao desenvolver uma API que possa ser acessível através de pedidos que tenham um endereço diferente do endereço da API, por exemplo, feitos por código *JavaScript* a ser executado num *browser* usando AJAX [38], é preciso configurar a API para responder de forma a que o resultado possa ser visível no *browser*. A API *XMLHttpRequest* [39] usa a

política de *same-origin*. A especificação *World Wide Web Consortium* (W3C) para *Cross-Origin Resource Sharing* (CORS) [40], permite que os resultados se tornem visíveis, evitando assim a política de segurança imposta pelos *browsers*. Assim deu-se permissão para todos os endereços e todos os *headers* HTTP, de forma a que os utilizadores da API possam aceder livremente a esta através do *browser*.

Armazenamento de *Passwords*

É considerada má prática o armazenamento de informação sensível, tal como, as de *passwords* em claro na base de dados, então estas devem ser encriptadas antes de serem armazenadas.

Para a encriptação é feito o *hash* da palavra-chave adicionando um valor aleatório. A este valor aleatório dá-se o nome de *salt*. Com o uso de *salt* previne-se que palavras-chave iguais sejam armazenadas na base de dados com valores diferentes.

Caso alguém consiga aceder à base de dados não consegue descobrir a palavra-passe de um determinado utilizador, isto é garantido encriptando a palavra-chave. Para encriptar as palavras-chave usou-se uma implementação do algoritmo *BCrypt* [41] fornecido pelo *Spring Security*. Outras opções passariam por usar o algoritmo *MD5*, ou *SHA*, mas estes estão *deprecated* por serem algoritmos com fraca classificação. Na implementação do *Spring Security* o *salt* é gerado internamente, sendo o valor por omissão de iterações igual a 10, isto significa que é feito o *hash* 2^{10} iterações.

3.3.2 *Controllers*

Os *controllers* identificam os pedidos e direciona-os para os serviços adequados ao seu processamento, retornando no final a resposta apropriada. Os formatos de resposta utilizam *hypermedia* [42], e são de um de três tipos, *Json Home* [43], *Siren* [44] e *Problem Details* [45].

A escolha do uso de *hypermedia* apoia-se em questões evolutivas da API em termos de hiperligações, ou seja, caso os *endpoints* dos recursos sejam alterados a aplicação cliente não sofre alterações.

Hoje em dia é comum encontrar uma API que se classifique no nível dois, segundo o modelo de maturidade definido pelo Leonard Richardson [46]. Pertencer ao nível dois significa que é retornada informação relativa ao recurso pedido, não existindo nenhuma meta informação associada ao recurso. Logo, para se conhecer as operações disponíveis e permitidas sobre esse recurso, ou ter conhecimento dos links para os recursos relacionados com o recurso retornado, é indispensável a consulta da documentação da API.

Já no nível três, do modelo supramencionado, estas informações são retornadas em conjunto com a representação do recurso, estando presente a representação da entidade e meta informação associada ao recurso retornado. Uma outra vantagem do nível três é que as aplicações cliente apenas contém um link *hardcoded*, sendo este o url para a página inicial,

denominado link base. Tornando-se responsabilidade da API embeber os links nas representações. Assim, se algum link sofrer alterações, nenhuma aplicação cliente fica comprometida.

Existem desvantagens no nível três comparado com o nível dois. Por exemplo, as mensagens de resposta HTTP serem de maior dimensão, usando uma maior largura de banda. Uma outra desvantagem é as aplicações cliente necessitarem de mapear as representações recebidas pela API e fornecerem uma *user interface* com base na representação recebida, o que aumenta o grau de complexidade no desenvolvimento.

Concluiu-se que usar controlos com *hypermedia* seria mais vantajoso para o projeto, pois as regras de negócio são definidas pela API e estão embebidas nas representações em *hypermedia*.

Exemplo 1
Imagine-se o caso em que cada utilizador apenas pode criar até cinco listas numa casa. Enquanto o número de listas de um utilizador não perfizer cinco, a representação da coleção de listas permite que sejam adicionadas mais listas, ao incluir nesta a ação de adicionar uma lista. No entanto, quando este limite for atingido, esta ação é excluída da representação, evitando assim que as aplicações cliente que recebem a representação sejam tentadas a adicionar uma nova lista. Deste modo, é evitado um pedido desnecessário à API *Web*, que certamente iria falhar.

Formato dos Erros

De forma a uniformizar os erros expostos pela API, utilizou-se *hypermedia Problem Details*. Esta *hypermedia* permite dar ao utilizador mais informação sobre o erro e como resolvê-lo caso seja um erro de cliente. Todos os pedidos realizados à API em que ocorra um erro, é retornada uma representação no formato *Problem Details*.

3.3.3 Lógica de Negócio

É fundamental fazer cumprir as regras, restrições e toda a lógica da gestão dos dados para o correto funcionamento do sistema. Assim, este controlo foi depositado na Camada da Lógica de Negócio (BLL) e também no modelo desenvolvido. Esta decisão permite, não só, concentrar a gestão dos dados, como também, controlar numa camada intermédia os dados a obter, atualizar, remover ou inserir, antes de realizar o acesso/escrita dos mesmos. O que estabelece um conjunto de operações disponíveis e coordena a resposta do aplicativo em cada operação.

Cada serviço estabelece um conjunto de operações disponíveis, conforme a definição de Randy Stafford [47]. Desta maneira, consegue-se isolar as diversas operações referentes a cada entidade.

Para a implementação desta camada criaram-se serviços para cada entidade. Estes expõem funcionalidades e aplicam as regras de negócio necessárias. É de salientar que um

serviço está fortemente ligado a um ou mais *repositories*, e estes podem estar associados a um ou mais serviços.

3.3.4 Acesso a Dados

Uma vez armazenados os dados de forma persistente é indispensável realizar escritas e leituras sobre os mesmos. Como forma de abstrair as restantes camadas do SGBD, é necessário definir interfaces entre as diferentes camadas. Para tal, desenvolveu-se a Camada de Acesso a Dados (DAL).

São várias as formas de realizar o acesso à base de dados, sendo uma delas com a utilização do JDBC *driver*. Este expõe uma série de interfaces para realizar o acesso, como o *PreparedStatment*, o *ResultSet*, o *CallableStatment*, entre outras. Primeiramente, pensou-se desenvolver o acesso à base de dados através destas interfaces, tendo o programador de se preocupar com a escrita das *queries*, os contextos, o mapeamento do resultado das *queries* em classes *Plain Old Java Objects* (POJO), as transações, a implementação dos padrões de *lazy-loading*, de *repository* e de *unit of work*. Por contexto entendem-se as ligações estabelecidas e a manutenção destas através de um *pool* de ligações. Desenvolver um sistema Object-relational mapping (ORM) não é simples e toma algum tempo. Não é âmbito do projeto desenvolver este sistema.

Uma solução alternativa seria usar *Java Database Connectivity Template* (Jdbc *Template*). Esta biblioteca fornece uma abstração maior sobre a gestão de ligações, não tendo o programador de se preocupar com isso. O contra desta solução seria a escrita de *queries*, o mapeamento dos resultados em classes POJO e a implementação dos padrões descritos anteriormente. É um contra pois seria código repetitivo, visto que para cada entidade seria necessário escrever *queries* e mapear os resultados em classes POJO para obter os dados dessa tabela, ou só pelo identificador da entidade, para inserir uma entidade, para atualizar ou apagar. No fundo, garantir as operações *Create, Read, Update e Delete* (CRUD), entre outras.

A solução encontrada foi usar a *Java Persistent API* (JPA). Esta API permite uma maior abstração quando comparada com Jdbc *Template*, devido a não ser requerida a escrita de *queries* nem o mapeamento em classes *Java*, e esta, já implementar os padrões mencionados anteriormente. Para ter acesso às operações CRUD é necessário criar uma interface, que estenda de uma interface do JPA. Assim, simplifca-se a implementação das operações básicas. Caso sejam precisas outras operações, é possível declarar métodos nessa interface com nomes específicos do JPA, para este gerar as *queries* automaticamente. Cada entidade presente na BD é mapeada numa classe em Java, que representa o modelo de domínio da mesma. Esta classe tem várias anotações da JPA para referir a Chave-Primária, Chave-Estrangeira, associações entre entidades, etc. Em conjunto estas classes *Java* formam o modelo utilizado entre as camadas internas do lado do servidor. A JPA é a especificação, para a implementação

foi usado *Hibernate*.

Dado o extenso modelo do projeto, e não sendo o principal foco do projeto escrever e mapear os resultados esta foi a solução encontrada.

3.4 Algoritmo de Previsão de Stocks

Uma vez que para uma boa gestão de stocks existe a necessidade de prever a duração de cada um dos itens em stock, com base no historial de consumo e reposição da casa. De forma a garantir este requisito inerente ao sistema Smart Stocks, usou-se um algoritmo de previsão de stocks.

Para a realização deste algoritmo, teve-se a preocupação de analisar os vários tipos de modelo existentes e adotar o mais adequado ao sistema. Durante a pesquisa, deparou-se com a existência de dois tipos de modelo de previsão, sendo eles, Modelos Quantitativos e Qualitativos [48]. Os primeiros baseiam-se em análises numéricas dos dados históricos, enquanto que os qualitativos privilegiam dados subjetivos baseados na opinião de especialistas. Como se deseja realizar este algoritmo com base no histórico de consumo e reposição, optou-se pelo Modelo Quantitativo.

Dos sub-modelos disponíveis elegeu-se um modelo de séries temporais visto que este pode incluir nos seus dados um conjunto de elementos, por exemplo, sazonalidade, tendências, influências cíclicas ou comportamento aleatório. Decidiu-se, então, utilizar o método da média móvel devido a esta ser simples e fácil de implementar e ser suficiente para o desenvolvimento do projeto. No entanto, dentro do método da média móvel existem diversos tipos, dos quais escolheu-se o da média móvel ponderada, que é uma variação do modelo da média móvel simples em que os valores dos períodos mais recentes recebem um peso maior que os valores correspondentes aos períodos anteriores, podendo assim dar maior relevância aos dados mais atuais.

3.4.1 Implementação

Aplicou-se o método da média móvel ponderada para um período mínimo de 3 semanas. Este procedimento começa por atribuir pesos aos dados consoante a semana a que se referem, sendo que as mais recentes têm maior peso.

Como se tratam de dados diários e para evitar a descompensação de valores provocada por dias com consumo excecionais amortecem-se os valores. Assim os resultados tornam-se mais homogéneos e com um comportamento mais sazonal.

Com base nestes dois aspetos construiu-se um dia típico, que servirá de base para uma previsão. Seguiu-se o seguinte procedimento:

1º Passo

A primeira fase consiste em aplicar o método da média móvel ponderada aos dados, para isso usou-se a seguinte expressão:

$$Pdiax' = T1 \times Rdia1 + T2 \times Rdia2 + T3 \times Rdia3 \tag{3.1}$$

Onde os valores de Tx correspondem às taxas definidas como pesos para os dados diários ($Rdiax$) da semana x, sendo a semana mais recente a com o valor superior. As taxas escolhidas foram:

$$T1 = 10\%,\ T2 = 30\%\ e\ T3 = 60\%$$

A escolha destes valores foi realizada de modo a atribuir determinados pesos a cada elemento, garantindo que a soma de todos os pesos seja igual a 100%, tendo em atenção em dar um peso maior às semanas mais recentes.

2º Passo

Aos valores obtidos anteriormente aplicou-se um método de amortização de forma a homogeneizar e harmonizar a previsão. Assim, o valor obtido para um dia da semana resulta da soma do seu valor e do valor do dia da semana anterior e da seguinte, multiplicados por taxas, ou seja:

$$Pdiax'' = Tant \times PdiaxAnt' + Tdiax \times Pdiax' + Tseg \times PdiaxSeg' \tag{3.2}$$

Onde os valores de $Tant$, $Tdiax$ e $Tseg$ correspondem às taxas definidas como pesos para os dias da semana anterior, atual e seguinte, respetivamente. Os valores $PdiaxAnt'$, $Pdiax'$ e $PdiaxSeg'$ são os valores previstos no 1º passo do método de previsão para os dias da semana anterior, atual e seguinte, respetivamente. As taxas escolhidas foram:

$$Tant = Tseg = 25\%\ e\ Tdiax = 50\%.$$

A escolha destes valores foi realizada de modo a atribuir determinados pesos a cada elemento, garantindo que a soma de todos os pesos seja igual a 100%, tendo em atenção dar um peso maior ao dia da semana a calcular.

3.4.2 Integração no Sistema Smart Stocks

De forma a tornar o sistema Smart Stocks independente de um só algoritmo de previsão de stocks, estipulou-se uma interface para separar o contrato da implementação. De forma geral, a informação mínima exigida a cada dia para se obter a previsão de uma semana é: o nome do produto, a sua quantidade e a respetiva data do dia em questão. Esta informação compõe um objeto, que pode ser utilizado pelas várias implementações do algoritmo de previsão de stocks.

Foi criada uma *task*, a ser executada periodicamente [1], cuja função é percorrer todos os itens na base de dados, obter o seu histórico de quantidades e realizar a previsão para a semana seguinte. De seguida, os dados obtidos da previsão são analisados, e caso a quantidade prevista esteja abaixo de um determinado limite [2], este item é inserido na lista de compras gerida pelo sistema Smart Stocks.

[1] Para efeitos de teste, a periodicidade da *task* é de 5 minutos. No entanto, num cenário real esta tarefa seria executada no final de cada dia.

[2] Considera-se como stock mínimo de segurança, a quantidade mínima abaixo da qual um produto é inserido na lista de compras. Atualmente, esse limite está definido como duas unidades. No entanto, este é um ponto a melhorar. Esta melhoria poderia consistir no utilizador poder definir os seus próprios valores limite, através de configurações nas aplicações cliente.

Capítulo 4

Conclusões

Neste capítulo apresentam-se as conclusões relativas ao desempenho e trabalho realizado pelo grupo e, ainda, as diretrizes relativas a trabalho futuro.

4.1 Sumário

A possibilidade de visualização em tempo real do estado do stock de uma casa é um auxílio enorme. No entanto, transmitir essa informação de forma simples, clara e percetível aos utilizadores é requerente de um prévio trabalho de pesquisa, para se poder conhecer o que os utilizadores procuram, de maneira a oferecer serviços adequados às suas necessidades, tornando-os acessíveis e dinamiza-los de forma simples e prática. Com a finalidade de implementação do sistema Smart Stocks usou-se um modelo de dados, *PostgreSQL*, para o armazenamento dos dados indispensáveis à gestão de stocks. Implementou-se uma API *Web*, independente do modelo de dados e das aplicações cliente, responsável por disponibilizar os dados às aplicações clientes, móvel e web. A aplicação móvel é responsável, não só, por permitir a visualização da informação fornecida pela API *Web*, tal como a aplicação web, como também, por interagir com o sistema e escrever nas *tags*.

Realizaram-se testes à resistência das *Tags* NFC e testes unitários às funções utilitárias desenvolvidas. De forma a realizar testes ao sistema Smart Stocks num ambiente real, desenvolveu-se uma aplicação para dispositivos móveis *Android*, cuja principal função é simular um dispositivo de *hardware*, como os que estão presentes em armários e frigoríficos que integram o sistema desenvolvido neste projeto.

4.2 Testes

4.2.1 Testes à Resistência das *Tags* NFC

Existem diferentes condições de conservação para os diversos locais de armazenamento, podendo estes variar em termos de temperatura, humidade, pressão, entre outros. Assim, os alimentos são expostos a circunstâncias distintas aquando armazenados em casa. Por

exemplo, no caso dos frigoríficos as capacidades de congelação variam consoante o número de estrelas que possuem, de acordo com a DECO PROTESTE [49]:

- 1 estrela: até - 6ºC; conserva congelados até 1 semana.
- 2 estrelas: até - 12ºC; conserva congelados até 1 mês.
- 3 estrelas: permite conservar alimentos previamente congelados até 1 ano.
- 4 estrelas: entre - 18º e - 24ºC; são os únicos que permitem congelar alimentos.

O período máximo de conservação de cada alimento varia conforme a classificação de estrelas de um determinado frigorífico/congelador, segundo a DECO PROTESTE [50]. Nos modelos de quatro estrelas os tempos de conservação são os seguintes:

- Frutas e hortícolas cozidas e arrefecidas: 12 meses.
- Bifes de vaca sem gordura, salsichas frescas, frango e peru: 10 meses.
- Queijos de pasta mole ou semimole: 8 meses.
- Pão, massa folhada ou quebrada, bolachas e crepes, manteiga, carne de porco pouco gorda, coelho, lebre e caça, peixe magro ou meio gordo: 6 meses.
- Massas para pão e pizzas, tartes de fruta, peixe gordo, marisco, sopas, sobras de pratos cozinhados: 3 meses.
- Hambúrgueres, carne picada, carne de porco gorda: 2 meses.
- Bolos com creme: 1 mês.

Como tal, foi importante testar a resistência das *tags* NFC às várias condições de conservação, de forma a garantir a eficácia do sistema de gestão de stocks nos diversos espaços de cada casa. Realizaram-se testes em três locais de armazenamento distintos, um frigorífico, um congelador e um armário. Os testes compreenderam a escrita na *tag* e sua arrumação no respetivo local durante um período de tempo estipulado. Após este período de tempo a *tag* foi retirada e verificado o seu estado e ainda retificadas as funcionalidades de leitura e de escrita.

- Teste às *tags* NFC no frigorífico: Duração: 1 mês Estrelas: Temperatura: Humidade: média Resultado: Leitura e escrita funcionais.
- Teste às *tags* NFC no congelador: Duração: 1 mês Estrelas: Temperatura: Humidade: baixa Resultado: Leitura e escrita funcionais.

- Teste às *tags* NFC no armário: Duração: 1 mês Temperatura: 15-20 graus Celsius Humidade: alta Resultado: Leitura e escrita funcionais.

Em conclusão, para curtos períodos de tempo pode-se garantir a resistência das *tags*, bem como, a eficácia do sistema de gestão de stocks desenvolvido. Contudo, seria imprescindível uma avaliação extensa, com maior gama de períodos de tempo e múltiplas condições, para assim assegurar o sucesso da utilização do sistema Smart Stocks.

4.2.2 Testes ao Funcionamento do Sistema Smart Stocks

De forma a poder testar o correto funcionamento do sistema Smart Stocks, simulou-se um dispositivo de hardware por meio de uma aplicação móvel desenvolvida para o propósito. O intuito desta, para além de testar o algoritmo de previsão de stock, é o de simular movimentos, quer sejam de entrada, quer sejam de saída, dos itens numa casa.

Esta aplicação deve ser usada por dispositivos moveis *Android* que suportem a tecnologia NFC. Requerem desta funcionalidade pois é através desta que é possível realizar as leituras das *tags* NFC, para obter a informação do item que está a ser movimentado. De maneira a simular um movimento de entrada ou de saída, existe um *switch* na interface do utilizador que tem o propósito de indicar o tipo do movimento a ser testado. Existe, ainda, a possibilidade de especificar a casa e o identificador do local de armazenamento envolvidos. A aplicação ao ler a informação da *tag*, se é um movimento de entrada ou de saída, o identificador da casa e do local de armazenamento, envia o aglomerado de informação para a API *Web* e esta irá manusear e armazenar os dados na base de dados.

Com a utilização desta aplicação torna-se fácil testar o algoritmo desenvolvido, pois é apenas necessário realizar movimentos de entrada e saída, num determinado armazenamento de uma determinada casa, para se perceber se algoritmo se encontra a funcionar ou não corretamente, inserindo na lista de compras do sistema Smart Stocks os produtos em vias de acabar.

4.2.3 Testes ao Algoritmo de Previsão de Stocks

A desenvolver...

4.2.4 Testes Unitários

Os testes unitários servem para garantir o funcionamento de um componente, por exemplo, se um método funciona, ou seja, se é retornando o que é esperado, dando garantias ao programador de que a sua aplicação continua a funcionar mesmo após serem efetuadas alterações ou substituídas peças componentes, sendo, assim importante a realização dos testes.

Estão disponíveis testes unitários para as funções utilitárias. No entanto, a Camada da Lógica de Negócio (BLL) carece de testes. Considerou-se que para realizar estes testes

seria necessário demasiado tempo para estes ficarem realmente bem feitos, testando todas as particularidades, como por exemplo, garantir que todas as restrições de integridade estariam a funcionar corretamente, e testando casos em que seria suposto funcionar e noutros que não seria suposto funcionar. A Camada de Acesso a Dados (DAL) carece também de testes aos métodos desenvolvidos. Os métodos gerados pela JPA não necessitam de testes uma vez que se confia nessa biblioteca. Para realizar os testes seria necessário simular o acesso à base de dados, usando uma base de dados em memória (*mock*). O *Spring Boot* tem integração com uma base de dados em memória chamada *H2 Database Engine* [51]. Esta base de dados será construída com a informação presente nas classes do modelo, as anotações JPA, pelo que facilitaria simular a base de dados. Contudo a realização destes testes demorariam mais tempo que o disponível para serem realizados da melhor forma.

4.3 Trabalho Futuro

As principais vertentes para um trabalho futuro centram-se em adquirir um certificado para API, de modo a que o sistema possa ir para produção. Igualmente importante, é a aquisição de funcionalidades de administração do sistema Smart Stocks para o papel *ADMIN*, de forma a criar uma aplicação de *back-office*, que se ocupasse da gestão das categorias e produtos, multi-línguas, adição de novas listas de sistema, assim como, novas funcionalidades e/ou regras de negócio. Melhorar a gestão dos utilizadores de uma casa, de maneira, por exemplo, para quando um utilizador sair de uma casa, o administrador receber uma notificação e ter de verificar e atualizar os dados básicos da casa, como o caso das alergias e do número de pessoas alérgicas da casa. Realizar um sistema de cores que nas aplicações cliente serviria como um aviso visual do estado dos itens em stock, tanto para a sua quantidade, como para a sua validade, aproveitando a análise dos resultado do algoritmo de previsão de stocks. Seria ainda interessante, permitir que o utilizador definisse os seus próprios limites mínimos dos produtos a ter em stock de maneira a tornar o sistema mais pessoal a cada pessoa e a cada casa. Outra melhoria a incluir, seria a criação de um sistema de notificações, de modo a deixar os utilizador sempre informados de qualquer alteração que aconteça, fossem elas por *e-mail* ou recorrendo às notificações do *Android*. E ainda, realizar testes unitários que testem a grande maioria das particularidades e casos. Por fim, uma melhoria à qualidade da segurança oferecida, de forma a garantir a confidencialidade máxima dos dados dos utilizadores e das suas casas.

Referências

[1] Eurostat. Internet use by individuals. http://ec.europa.eu/eurostat/documents/2995521/7771139/9-20122016-BP-EN.pdf, Dezembro 2016. [Online, Acedido a 23/05/2018].

[2] Mark Purdy and Ladan Davarzani (Accenture). The growth game-changer: How the industrial internet of things can drive progress and prosperity. https://www.accenture.com/t20150708T025453__w__/fr-fr/_acnmedia/Accenture/Conversion-Assets/DotCom/Documents/Local/fr-fr/PDF_5/Accenture-Industrial-Internet-Things-Growth-Game-Changer.pdf, 2015. [Online, Acedido a 12/06/2018].

[3] M. Miller. *The Internet of Things: How Smart TVs, Smart Cars, Smart Homes, and Smart Cities are Changing the World*. Que, 2015. [Online, Acedido a 20/04/2018].

[4] AS Shweta. Intelligent refrigerator using artificial intelligence. In *Intelligent Systems and Control (ISCO), 2017 11th International Conference on*, pages 464–468. IEEE, 2017.

[5] Business Dictionary. What is inventory? definition and meaning - businessdictionary.com. http://www.businessdictionary.com/definition/inventory.html. [Online, Acedido a 04/26/2018].

[6] Investopedia. Stock keeping unit (sku). https://www.investopedia.com/terms/s/stock-keeping-unit-sku.asp. [Online, Acedido a 26/04/2018].

[7] Thad Scheer. Category, segment, and brand – what's the difference? – sphere of influence : Analytics studio. https://sphereoi.com/studios/category-segment-and-brand-whats-the-difference/. [Online, Acedido on 25/03/2018].

[8] Gazelle Point of Sale Support. Stock item and non-stock item : Gazelle point-of-sale support. http://support.phostersoft.com/support/solutions/articles/17907-stock-item-and-non-stock-item. [Online, Acedido a 26/04/2018].

[9] Investopedia. Brand. https://www.investopedia.com/terms/b/brand.asp. [Online, Acedido a 26/04/2018].

[10] Regulamento (UE) nº1169/2011. Rotulagem de géneros alimentícios. https://www.asae.gov.pt/perguntas-frequentes1/rotulagem-de-generos-alimenticios-.aspx, Abril 2017. [Online, Acedido a 10/03/2018].

[11] What is nfc? - nfc forum — nfc forum. https://nfc-forum.org/what-is-nfc/. [Online, Acedido a 15/04/2018].

[12] Rfid readers and rfid tags for any specialty rfid needs - rfid, inc. http://rfidinc.com/. [Online, Acedido a 15/04/2018].

[13] Y. Shafranovich. Rfc 4180 - common format and mime type for comma-separated values (csv) files. urlhttps://tools.ietf.org/html/rfc4180, Outubro 2005. [Online, Acedido a 21/05/2018].

[14] What is a qr code? — qrcode.com — denso wave. http://www.qrcode.com/en/about/. [Online, Acedido a 07/03/2018].

[15] Waqar Haque, Robert Lucas, and Paul Stokes. Architectural design and analysis of an n-tier enterprise application. In *Proceedings of the Third IASTED European Conference on Internet and Multimedia Systems and Applications*, EurolMSA '07, pages 128–134, Anaheim, CA, USA, 2007. ACTA Press. [Online, Acedido a 22/06/2018].

[16] Jayanti Katariya. Apple vs android — a comparative study 2017 – androidpub. https://android.jlelse.eu/apple-vs-android-a-comparative-study-2017-c5799a0a1683, Março 2017. [Online, Acedido a 16/03/2018].

[17] Distribution dashboard — android developers. https://developer.android.com/about/dashboards/, Maio 2018. [Online, Acedido a 16/03/2018].

[18] Homepage - material design. https://material.io/. [Online, Acedido a 24/04/2018].

[19] Daniel Novak. Service locator pattern in android – inloopx – medium. https://medium.com/inloopx/service-locator-pattern-in-android-af3830924c69, Janeiro. [Online, Acedido a 20/03/2018].

[20] Support different languages and cultures — android developers. https://developer.android.com/training/basics/supporting-devices/languages, Abril 2018. [Online, Acedido a 27/04/2018].

[21] Godfrey Nolan. Best place to store a password in your android app. https://www.androidauthority.com/where-is-the-best-place-to-store-a-password-in-your-android-app-597197/, Março 2015. [Online, Acedido a 03/07/2018].

[22] Smart lock for passwords on android — google developers. https://developers.google.com/identity/smartlock-passwords/android/. [Online, Acedido a 02/05/2018].

[23] Magnus Vinther. Why you should totally switch to kotlin – magnus vinther – medium. https://medium.com/@magnus.chatt/why-you-should-totally-switch-to-kotlin-c7bbde9e10d5, Maio 2017. [Online, Acedido a 16/03/2018].

[24] Volley overview — android developers. https://developer.android.com/training/volley/, Abril 2018. [Online, Acedido a 27/04/2018].

[25] Software developer Mauricio Klein. What is the difference between express and react, in terms of node.js development? - quora. https://www.quora.com/What-is-the-difference-between-Express-and-React-in-terms-of-node-js-development, Abril 2016. [Online, Acedido a 28/06/2018].

[26] R. Fielding and J. Reschke. Rfc 7231 - hypertext transfer protocol (http/1.1): Semantics and content. https://tools.ietf.org/html/rfc7231, Junho 2014. [Online, Acedido a 15/04/2018].

[27] Leverage browser caching — pagespeed insights — google developers. https://web.archive.org/web/20161206092632/https://developers.google.com/speed/docs/insights/LeverageBrowserCaching, Janeiro 2018. [Online, Acedido a 21/02/2018].

[28] The best apis are built with swagger tools — swagger. https://swagger.io/. [Online, Acedido a 01/07/2018].

[29] Justin Richer, Amanda Anganes, Michael Jett, Michael Walsh, Steve Moore, Mike Derryberry, William Kim, and Mark Janssen. Mitreid connect: An openid connect reference implementation in java on the spring platform. https://github.com/mitreid-connect/OpenID-Connect-Java-Spring-Server, 2018. [Online, Acedido a 21/06/2018].

[30] Openid connect — openid. http://openid.net/connect/, Fevereiro 2014. [Online, Acedido a 21/06/2018].

[31] Oauth 2.0 — oauth. https://oauth.net/2/. [Online, Acedido a 21/06/2018].

[32] Oauth-apis/apis: Oauth authorization as a service. https://github.com/OAuth-Apis/apis, 2016. [Online, Acedido a 22/06/2018].

[33] zalando/tokens: Java library for conveniently verifying and storing oauth 2.0 service access tokens. https://github.com/zalando/tokens, 2015. [Online, Acedido a 22/06/2018].

[34] Json web token introduction - jwt.io. https://jwt.io/introduction/. [Online, Acedido a 23/06/2018].

[35] J. Reschke. Rfc 7617 - the 'basic' http authentication scheme. https://tools.ietf.org/html/rfc7617, Setembro 2015. [Online, Acedido a 15/04/2018].

[36] E. Rescorla and E. A. Schiffman. Rfc 2660 - the secure hypertext transfer protocol. https://tools.ietf.org/html/rfc2660, Agosto 1999. [Online, Acedido a 15/04/2018].

[37] T. Dierks and E. Rescorla. Rfc 5246 - the transport layer security (tls) protocol version 1.2. https://tools.ietf.org/html/rfc5246, Agosto 2008. [Online, Acedido a 25/06/2018].

[38] Angel-luis. Ajax - developer guides — mdn. https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX, Janeiro 2018. [Online, Acedido a 24/04/2018].

[39] Xmlhttprequest standard. https://xhr.spec.whatwg.org/, Junho 2018. [Online, Acedido a 16/06/2018].

[40] Anne van Kesteren. Cross-origin resource sharing. https://www.w3.org/TR/cors/, Janeiro 2014. [Online, Acedido a 24/04/2018].

[41] Bcrypt (spring security 4.2.5.release api). https://docs.spring.io/spring-security/site/docs/4.2.5.RELEASE/apidocs/org/springframework/security/crypto/bcrypt/BCrypt.html. [Online, Acedido a 29/05/2018].

[42] Mike Stowe. Api best practices: Hypermedia (part 4.1) — mulesoft blog. https://blogs.mulesoft.com/dev/api-dev/api-best-practices-hypermedia-part-1/. [Online, Acedido a 28/05/2018].

[43] Home documents for http apis draft-nottingham-json-home-06. https://tools.ietf.org/html/draft-nottingham-json-home-06, Fevereiro 2017. [Online, Acedido a 15/04/2018].

[44] Kevin Swiber. Siren: a hypermedia specification for representing entities. https://github.com/kevinswiber/siren. [Online, Acedido a 15/04/2018].

[45] M. Nottingham, Akamai, and E. Wilde. Rfc 7807 - problem details for http apis. https://tools.ietf.org/html/rfc7807, Março 2016. [Online, Acedido a 15/04/2018].

[46] Martin Fowler. Richardson maturity model. https://martinfowler.com/articles/richardsonMaturityModel.html, Março 2010. [Online, Acedido a 17/06/2018].

[47] Randy Stafford. P of eaa: Service layer. https://www.martinfowler.com/eaaCatalog/serviceLayer.html. [Online, Acedido a 28/05/2018].

[48] Dulce Almeida and Lídia Teixeira. Microsoft word - relatoriofinalissimo.doc. https://web.fe.up.pt/~ee00190/RelatorioProjectoEstagio.pdf. [Online, Acedido a 28/05/2018].

[49] Funções dos frigoríficos de a a z. https://www.deco.proteste.pt/eletrodomesticos/frigorificos/dicas/funcoes-dos-frigorificos-de-a-a-z, Setembro 2015. [Online, Acedido a 06/07/2018].

[50] Conservar alimentos no congelador. https://www.deco.proteste.pt/alimentacao/seguranca-alimentar/dicas/conservar-alimentos-no-congelador, Abril 2016. [Online, Acedido a 06/07/2018].

[51] H2 database engine. http://www.h2database.com/html/main.html. [Online, Acedido a 28/05/2018].

Anexo A

Modelo de Dados

A.1 Domínio dos Atributos

Tabela A.1: Domínio dos Atributos da Entidade House.

Entidade	**Atributo**	**Domínio**	**Tipo Variável (PostgreSQL)**	**Restrições**	**Nullable**
House	house_id	Número inteiro auto-incrementado	bigserial	-	não
	house_name	Cadeia de caracteres de comprimento variável	character varying(35)	até 35 caracteres	não
	house_characteristics	Objeto JSON	json	-	não

Tabela A.2: Domínio dos Atributos da Entidade Users.

Entidade	**Atributo**	**Domínio**	**Tipo Variável (PostgreSQL)**	**Restrições**	**Nullable**
Users	users_id	Número inteiro auto-incrementado	bigserial	-	não
	users_username	Cadeia de caracteres de comprimento variável	character varying(30)	até 30 caracteres	não
	users_email	Cadeia de caracteres de comprimento variável	character varying(254)	até 254 caracteres	não
	users_age	Número inteiro	smallint	user_age in [0, 150]	não
	users_name	Cadeia de caracteres de comprimento variável	character varying(70)	até 70 caracteres	não
	users_password	Cadeia de caracteres de comprimento variável	character varying(50)	até 50 caracteres	não

Tabela A.3: Domínio dos Atributos da Entidade Role.

Entidade	Atributo	Domínio	Tipo Variável (PostgreSQL)	Restrições	Nullable
Role	role_id	Número inteiro auto-incrementado	smallserial	-	não
	role_username	Cadeia de caracteres de comprimento variável	character varying(50)	-	não

Tabela A.4: Domínio dos Atributos da Entidade Usersrole.

Entidade	Atributo	Domínio	Tipo Variável (PostgreSQL)	Restrições	Nullable
Usersrole	users_id	Número inteiro	bigint	users_id > 0	não
	role_id	Número inteiro	smallserial	role_id > 0	não

Tabela A.5: Domínio dos Atributos da Entidade Allergy.

Entidade	Atributo	Domínio	Tipo Variável (PostgreSQL)	Restrições	Nullable
Allergy	allergy_allergen	Cadeia de caracteres de comprimento variável	character varying(75)	até 75 caracteres	não

Tabela A.6: Domínio dos Atributos da Entidade List.

Entidade	Atributo	Domínio	Tipo Variável (PostgreSQL)	Restrições	Nullable
List	house_id	Número inteiro	bigint	house_id > 0	não
	list_id	Número inteiro auto-incrementado	smallint	-	não
	list_name	Cadeia de caracteres de comprimento variável	character varying(35)	até 35 caracteres	não
	list_type	Cadeia de caracteres de comprimento variável	character varying(7)	list_type in ['system', 'user']	não

Tabela A.7: Domínio dos Atributos da Entidade SystemList.

Entidade	**Atributo**	**Domínio**	**Tipo Variável (PostgreSQL)**	**Restrições**	**Nullable**
System List	house_id	Número inteiro	bigint	house_id > 0	não
	list_id	Número inteiro	smallint	list_id > 0	não

Tabela A.8: Domínio dos Atributos da Entidade UserList.

Entidade	**Atributo**	**Domínio**	**Tipo Variável (PostgreSQL)**	**Restrições**	**Nullable**
User List	house_id	Número inteiro	bigint	house_id > 0	não
	list_id	Número inteiro	smallint	list_id > 0	não
	users_id	Número inteiro	bigint	users_id > 0	não
	list_shareable	Booleano	boolean	-	sim

Tabela A.9: Domínio dos Atributos da Entidade Category.

Entidade	**Atributo**	**Domínio**	**Tipo Variável (PostgreSQL)**	**Restrições**	**Nullable**
Category	category_id	Número inteiro auto-incrementado	serial	-	não
	category_name	Cadeia de caracteres de comprimento variável	character varying(35)	até 35 caracteres	não

Tabela A.10: Domínio dos Atributos da Entidade Product.

Entidade	**Atributo**	**Domínio**	**Tipo Variável (PostgreSQL)**	**Restrições**	**Nullable**
Product	category_id	Número inteiro	integer	category_id > 0	não
	product_id	Número inteiro auto-incrementado	serial	-	não
	product_name	Cadeia de caracteres de comprimento variável	character varying(35)	até 35 caracteres	não
	product_edible	Booleano	boolean	-	não
	product_shelfLife	Número inteiro	smallint	product_shelfLife > 0	não
	product_shelfLifeTimeUnit	Cadeia de caracteres de comprimento variável	character varying(5)	product_shelfLifeTimeUnit in ['day', 'week', 'month', 'year']	não

Tabela A.11: Domínio dos Atributos da Entidade StockItem.

Entidade	Atributo	Domínio	Tipo Variável (PostgreSQL)	Restrições	Nullable
StockItem	house_id	Número inteiro	bigint	house_id > 0	não
	stockItem_sku	Cadeia de caracteres de comprimento variável	character varying(128)	até 128 caracteres	não
	product_id	Número inteiro	integer	product_id > 0	não
	stockItem_brand	Cadeia de caracteres de comprimento variável	character varying(35)	até 35 caracteres	não
	stockItem_segment	Número décimal	real	stockItem_segment > 0	não
	stockItem_variety	Cadeia de caracteres de comprimento variável	character varying(35)	até 35 caracteres	não
	stockItem_quantity	Número inteiro	smallint	stockItem_quantity > 0	não
	stockItem_segmentUnit	Cadeia de caracteres de comprimento variável	character varying(5)	stockItem_segmentUnit in ['kg', 'dag', 'hg', 'g', 'dg', 'cg', 'mg', 'kl', 'hl', 'dal', 'l', 'dl', 'cl', 'ml', 'oz', 'lb', 'pt', 'fl oz', 'units']	não
	stockItem_description	Cadeia de caracteres de comprimento variável	text	-	sim
	stockItem_conservationStorage	Cadeia de caracteres de comprimento variável	character varying(128)	até 128 caracteres	não

Tabela A.12: Domínio dos Atributos da Entidade Storage.

Entidade	Atributo	Domínio	Tipo Variável (PostgreSQL)	Restrições	Nullable
Storage	house_id	Número inteiro	bigint	house_id > 0	não
	storage_id	Número inteiro auto-incrementado	smallint	-	não
	storage_name	Cadeia de caracteres de comprimento variável	character varying(35)	até 35 caracteres	não
	storage_temperature	Intervalo de números decimais	numrange	-	não

Tabela A.13: Domínio dos Atributos da Entidade UserHouse.

Entidade	Atributo	Domínio	Tipo Variável (PostgreSQL)	Restrições	Nullable
UserHouse	house_id	Número inteiro	bigint	house_id > 0	não
	users_id	Número inteiro	bigint	users_id > 0	não
	userHouse_administrator	Booleano	boolean	-	sim

Tabela A.14: Domínio dos Atributos da Entidade StockItemStorage.

Entidade	Atributo	Domínio	Tipo Variável (PostgreSQL)	Restrições	Nullable
StockItemStorage	house_id	Número inteiro	bigint	house_id > 0	não
	stockItem_sku	Cadeia de caracteres de comprimento variável	character varying(128)	até 128 caracteres	não
	storage_id	Número inteiro	smallint	storage_id > 0	não
	stockItemStorage_quantity	Número inteiro	smallint	stockItemStorage_quantity > 0	não

Tabela A.15: Domínio dos Atributos da Entidade StockItemMovement.

Entidade	Atributo	Domínio	Tipo Variável (PostgreSQL)	Restrições	Nullable
StockItemMovement	house_id	Número inteiro	bigint	house_id > 0	não
	stockItem_sku	Cadeia de caracteres de comprimento variável	character varying(128)	até 128 caracteres	não
	storage_id	Número inteiro	smallint	storage_id > 0	não
	stockItemMovement_type	Booleano	boolean	-	não
	stockItemMovement_dateTime	Data e Horas	timestamp	-	não
	stockItemMovement_quantity	Número inteiro	smallint	stockItemMovement_quantity > 0	não
	stockItemMovement_finalquantity	Número inteiro	smallint	stockItemMovement_finalquantity >= 0	não

Tabela A.16: Domínio dos Atributos da Entidade HouseAllergy.

Entidade	Atributo	Domínio	Tipo Variável (PostgreSQL)	Restrições	Nullable
HouseAllergy	house_id	Número inteiro	bigint	house_id > 0	não
	allergy_allergen	Cadeia de caracteres de comprimento variável	character varying(75)	até 75 caracteres	não
	houseAllergy_alergicsNum	Número inteiro	smallint	houseAllergy_alergicsNum > 0	não

Tabela A.17: Domínio dos Atributos da Entidade ListProduct.

Entidade	Atributo	Domínio	Tipo Variável (PostgreSQL)	Restrições	Nullable
ListProduct	house_id	Número inteiro	bigint	house_id > 0	não
	list_id	Número inteiro	smallint	list_id > 0	não
	product_id	Número inteiro	integer	product_id > 0	não
	listProduct_brand	Cadeia de caracteres de comprimento variável	character varying(35)	até 35 caracteres	sim
	listProduct_quantity	Número inteiro	smallint	listProduct_quantity > 0	não

Tabela A.18: Domínio dos Atributos da Entidade StockItemAllergy.

Entidade	Atributo	Domínio	Tipo Variável (PostgreSQL)	Restrições	Nullable
StockItemAllergy	house_id	Número inteiro	bigint	house_id > 0	não
	stockItem_sku	Cadeia de caracteres de comprimento variável	character varying(128)	até 128 caracteres	não
	allergy_allergen	Cadeia de caracteres de comprimento variável	character varying(75)	até 75 caracteres	não

Tabela A.19: Domínio dos Atributos da Entidade Date.

Entidade	Atributo	Domínio	Tipo Variável (PostgreSQL)	Restrições	Nullable
Date	date_date	Data (AAAA/MM/DD)	date	-	não

Tabela A.20: Domínio dos Atributos da Entidade ExpirationDate.

Entidade	Atributo	Domínio	Tipo Variável (PostgreSQL)	Restrições	Nullable
ExpirationDate	house_id	Número inteiro	bigint	house_id > 0	não
	stockItem_sku	Cadeia de caracteres de comprimento variável	character varying(128)	até 128 caracteres	não
	date_date	Data (AAAA/MM/DD)	date	-	não
	date_quantity	Número inteiro	smallint	date_quantity >= 0	não

Tabela A.21: Domínio dos Atributos da Entidade Invitations.

Entidade	Atributo	Domínio	Tipo Variável (PostgreSQL)	Restrições	Nullable
Invitations	house_id	Número inteiro	bigint	house_id > 0	não
	users_id	Número inteiro	bigint	users_id > 0	não

Tabela A.22: Domínio dos Atributos da Entidade DailyQuantity.

Entidade	Atributo	Domínio	Tipo Variável (PostgreSQL)	Restrições	Nullable
DailyQuantity	house_id	Número inteiro	bigint	house_id > 0	não
	stockItem_sku	Cadeia de caracteres de comprimento variável	character varying(128)	até 128 caracteres	não
	dailyquantity_date	Data (AAAA/MM/DD)	date	-	não
	dailyquantity_quantity	Número inteiro	smallint	dailyquantity_quantity >= 0	não