UNIVERSIDADE AUTÓNOMA DE LISBOA
LUÍS DE CAMÕES

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

ShappList

Autores: David Santos Cordeiro Teixeira Pinto, Henrique Filipe Chuva, Pedro Jorge de Almeida Correia, Ricardo Filipe Vicente Machado

Orientador: Daniel Silvestre

Número dos candidatos: 20160740, 20160715, 20160536, 20160746

Julho de 2019

Lisboa

(Página em Branco)

1 Resumo

Shapplist foi criada com o objetivo de proporcionar aos clientes uma maneira simples e intuitiva de criar múltiplas listas de compras, podendo estas ser partilhadas com amigos, colegas de trabalho ou familiares através de um sistema de convites. Os produtos disponíveis são igualmente um aspeto fulcral, pois estes são recolhidos a partir dos principais supermercados do país – Pingo Doce, Continente e Jumbo – sendo assim possível verificar qual o supermercado onde a compra de todos os itens é mais barata ou verificar produtos com desconto.

A aplicação foi desenvolvida tendo em conta a possibilidade de acesso através dos mais variados dispositivos – computador, tablet ou telemóvel. Assim, esta passa a designar-se por *web app*, capaz de correr em diversos browsers e sistemas operativos. O uso de micro-serviços na estrutura de base da aplicação permite garantir um alto nível de disponibilidade e redundância. Estes micro-serviços estão isolados entre si, o que permite eliminar pontos singulares de falha, tornar eficiente a adição de novas funcionalidades à aplicação e permitir a gestão e modificação de cada um dos micro-serviços sem ser necessário "desligar" toda a aplicação. Para cada um dos micro-serviços foram criadas máquinas virtuais de modo a ter um controlo completo sobre a execução de toda a aplicação e todas foram configuradas de modo a ser rápida e fácil a replicação da aplicação num novo sistema.

A organização da base de dados visa trabalhar sobre pontos fulcrais como a disponibilidade, redundância e acessibilidade dos dados, assim como fornecer um alto nível de eficiência na pesquisa e agrupamento de dados, usando uma arquitetura bem estruturada e modularizada.

Palavras-chave: Web app; Simples; Intuitiva; Inovadora.

2 Abstract

Shapplist was created with the purpose of providing customers a simple and intuitive way to create shopping lists, while having the ability to share these with friends, co-workers or family members through an invite system. The available products are of equal importance, as these are collected from the main supermarkets in the country – Pingo Doce, Continente and Jumbo – allowing users to check which supermarket is best or which products are on sale.

The application was developed with the purpose of it being able to be accessed by all kinds of devices, like computers, tablets or mobile phones. As such, it is considered a web app, capable of running on a variety of browsers and operative systems. The use of microservices as the design for the application's architecture grants it a high level of availability and redundancy. These microservices are isolated from each other, not allowing the possibility of single points of failure, making more efficient the addition of new functionalities and enabling the ability to modify each of the microservices without the need to "shutdown" the entire application. Virtual machines were created for each one of the microservices in order to have full control over the applications' execution. All virtual machines were setup in such a way as to allow the easy and fast replication of the application in a new system.

The organization of the database emphasizes key points such as availability, redundancy and accessibility to the data, as well as providing a high level of efficiency on the execution of queries and clustering of data, using a well-defined and modularized structure.

Keywords: Web app; Simple; Intuitive; Innovative.

3 Índice

4 Lista de Quadros/Gráficos

5 Lista de Fotografias/Ilustrações

6 Lista de Siglas e Acrónimos

AJAX	Asynchronous JavaScript and XML
API	Application programming interface
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
JSON	JavaScript Object Notation
MySQL	My Structured Query Language
URL	Uniform Resource Locator
OVF	Open Virtualization Format
UML	Unified Modeling Language
VM	Virtual Machine

7 Glossário

Atom	Editor de código
Adobe Photoshop CC	Software de edição de imagem
Android Studio	Software que permite criar aplicações móveis
Auto_Increment	Variável que guarda um valor numérico para atribuição automática numa tabela
Back-end	Conjunto de funcionalidades auxiliares à execução da aplicação
Balanceamento de carga	Permite equilíbrio da carga de trabalho por vários servidores
Biblioteca	Conjunto de recursos para auxílio de programação
Bootstrap	Biblioteca de CSS
Browser	Aplicação para o acesso à World Wide Web
Caminho/path	Especifica a localização de um ficheiro no sistema de ficheiros
Checkboxes	Caixas de confirmação
Checked	Confirmação da checkbox
Container	Sistema virtual de execução de *docker images*
Cookie	Informação guardado no *browser* do utilizador referente a um único endereço IP/*Domain*

Deployed	Execução de uma aplicação num determinado ambiente
Diretório	Sistema de ficheiros de um computador
Django	Framework baseada em *Python*
Docker	Tecnologia que permite o *deployment* automatizado de uma aplicação
Dockerfile	Documento texto que contém todos os comandos a ser executados para criar uma *docker image*
Docker Image	Ficheiro com múltiplas especificações a serem executadas
Domain	Tradução de um endereço IP para um nome
Escalabilidade	Capacidade de uma aplicação crescer ao longo da sua vida
Endereço IP	Endereço numérico que identifica uma máquina numa rede
Fast-add	Termo usado para definir a adição rápida dos produtos
Float	Tipo de dado que representa um conjunto finito de números racionais
Foreign key	Coluna numa tabela que realiza a ligação a outra tabela, servindo de apontador à chave primária da outra tabela
Framework	Plataforma virtual que facilita o desenvolvimento de uma aplicação, fornecendo diversas funcionalidades base
Front-end	Código que corresponde à componente visual da aplicação
Hardware	Componentes físicos de um sistema
Hold	Termo utilizado para definir a funcionalidade de um longo clique
Hub	Ponto central de uma área/secção
Hyperlink	Sequência de caracteres que permite redirecionar um utilizador de uma página HTML para outra
Int	Tipo de dado que representa um conjunto finito de números inteiros
JavaScript	Linguagem de programação de alto nível
Landing page	Página de introdução de uma aplicação
Layout	Grafismos das páginas HTML
Login	Ponto de entrada para uma aplicação utilizando credenciais
Micro-serviços	Divisão de uma aplicação em partes mais pequenas
Mobile	Termo utilizado para definir telemóvel
Mode-shop	Termo utilizado para definir o modo de visualização no supermercado

Nginx	Servidor *web* que permite balanceamento de carga e redirecionamento de pedidos *web*
Package	Conjunto de *software*
POST	Entrega de uma representação do recurso alvo ao servidor de modo a criar um novo recurso
Primary Key	Coluna numa tabela que serve como identificador para essa tabela
Python	Linguagem de programação de alto nível
PythonAnywhere	Serviço que permite a gestão e desenvolvimento de uma aplicação web, através de um *browser*
Queries	Consulta efetuada sobre os dados de uma base de dados, de acordo com certas condições
Repositório	Sistema de armazenamento de ficheiros, podendo ser *online* ou local
Script	Ficheiro que executa uma sequência de funções
Scroll	Movimento vertical/horizontal numa página
Sign in	Termo sinónimo de login
Tablet	Computador móvel que funciona através do toque
Tinyint	Tipo de dado semelhante a int mas com tamanho menor
Tolerância a faltas	Capacidade de um sistema permanecer em operação em caso de erro
Token	Conjunto de caracteres aleatórios que identificam um utilizador
Tuplo	Resposta a uma query, conjunto de dados que representa uma entrada na base de dados
Ubuntu	Sistema operativo
Unix time	Número de segundos desde a meia noite de 1 de Janeiro de 1970
User	Termo utilizado para o utilizador de uma aplicação
User-friendly	Termo utilizado para descrever uma aplicação fácil e intuitiva
Username	Termo utilizado para o utilizador de uma aplicação
Varchar	Tipo de dado que representa um conjunto de caracteres de texto
Views	Funções python da framework Django que recebem um pedido web e devolvem uma resposta
Web app	Aplicação acedida através de um browser
XMLHttpRequest	Objetos que enviam dados entre uma página HTML e um servidor

AJAX	Tecnologia que permite a sincronização da troca de informação de uma página HTML com um servidor
API	Conjunto de funções que permitem a uma aplicação aceder a informação de outro sistema
CSS	Linguagem que define o estilo dos documentos HTML
HTML	Linguagem utilizada para a exposição de documentos em browsers
JSON	Formato de dados em forma de par chave e valor
MySQL	Sistema de manuseamento de bases de dados
OVF	Formato que permite rapidamente executar uma máquina virtual mantendo todas as suas configurações originais
UML	Linguagem que permite construir e visualizar a estrutura de uma base de dados
URL	Especifica um endereço na World Wide Web
VM	Ambiente virtual que simula um sistema computacional

8 Introdução

Este projeto tem como objetivo a criação de uma web app – ShappList – que tem como principal funcionalidade a criação de listas de compras, podendo estas ser partilhadas e editadas por outros utilizadores convidados.

Os produtos adicionáveis às listas foram recolhidos a partir dos sites dos três principais supermercados do país – Pingo Doce [1], Continente [2] e Jumbo [3] – sendo possível adicionar às listas os produtos vendidos nestes supermercados com os valores de venda destes, bem como os produtos em desconto.

A criação das listas culmina no cálculo final do preço dos artigos dessa mesma lista, sendo possível perceber qual o supermercado mais vantajoso, isto é, o que permite uma maior poupança para o consumidor.

O projeto está separado em 3 blocos principais: *front-end* – área de apresentação da *web app* (aplicação multiplataforma – computador, tablet e telemóvel), *back-end* – microserviços; e base de dados.

8.1 Front-End

O design da Shapplist teve em conta, desde muito cedo, o conceito de *user-friendly*, ou seja, funcionalidade e apresentação focadas na facilidade de entendimento por parte dos utilizadores, sem nunca prejudicar os objetivos fulcrais da aplicação. Ter uma aplicação intuitiva foi algo que foi traçado como linha de desenvolvimento de todo o projeto. Para o desenvolvimento de todo o design e layout da aplicação foram usadas as linguagens *HTML* e *CSS*, com o auxílio de *JavaScript*. No *HTML*, foi também utilizado *Bootstrap*, para adaptar a linguagem para o universo móvel, permitindo que as diversas páginas da aplicação se adaptem ao dispositivo em que estão a ser exibidas.

8.2 Back-End

Para a criação do *back-end* foram traçados dois objetivos principais: a escalabilidade de toda a aplicação e a modularidade do código que a suporta. Foi escolhida a *framework* Django de modo a criar uma aplicação *web*, baseada em *Python*, que satisfizesse os nossos objetivos.

O *back-end* é dividido em micro-serviços, onde cada um é responsável por um certo número de funcionalidades. Este agrupamento é definido tendo em conta a que "parte" da aplicação se referem (funcionalidades de autenticação, manuseamento de produtos, gestão das listas, etc.). Esta divisão em micro-serviços facilita a adição de novas funcionalidades à aplicação (escalabilidade) e a identificação e correção de erros. Permite também uma maior tolerância a faltas de cada serviço e de toda a aplicação em si (devido ao isolamento de cada micro-serviço).

Após a criação de cada micro-serviço, estes são *deployed* em máquinas virtuais isoladas e únicas, o que permite uma alocação apropriada tendo em conta as necessidades de cada um dos micro-serviços. Estas máquinas virtuais permitem uma maior flexibilidade à aplicação, devido ao uso do formato *OVF*, que permite uma fácil e rápida criação de novos sistemas virtuais em caso de erro ou de adição de novos serviços.

8.3 Base de dados

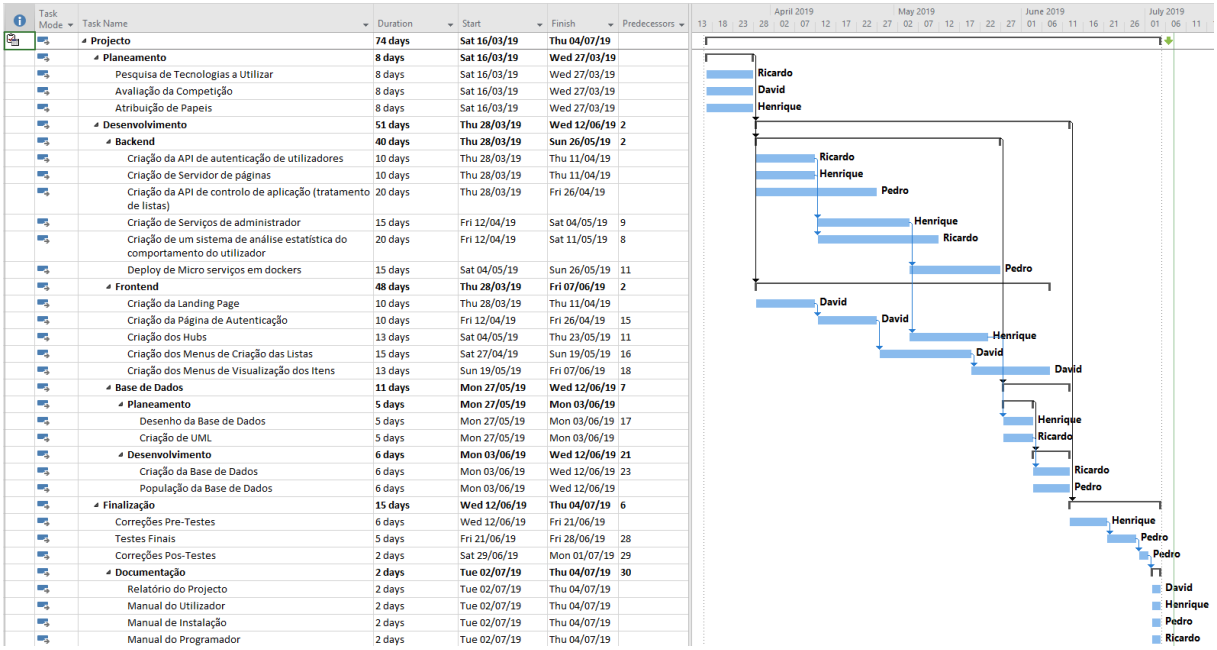
A base de dados foi concebida por forma a englobar duas características importantes para o bom funcionamento da aplicação: o acesso fácil e seguro aos dados; e a garantia de futura expansibilidade. Para tal, o desenho da base de dados teve, desde início, estes dois conceitos como base a partir dos quais a divisão das tabelas está organizada. A modularização dos dados desempenha um papel importante no aumento do nível de extensibilidade, possibilitando aumentos de acordo com as necessidades futuras. A organização em diversas tabelas facilita também o acesso aos dados, diminuindo a complexidade das *queries* efetuadas e aumentando a eficiência destas.

O uso de *MySQL* para a criação da base de dados justifica-se na familiaridade que o grupo possui com o sistema.

9 Planeamento do Projeto

A aplicação foi planeada ao pormenor desde uma fase embrionária. Ao longo de todo o processo de planeamento, foram definidas as várias ideias principais a desenvolver, bem como ideias que foram consideradas como secundárias e as que não fariam parte do produto final, pelo simples facto de não se enquadrarem na linha de pensamento traçado para a *app*.

Após todas as tarefas terem sido definidas, foi usado o programa *Microsoft Project* para fazer a planificação das tarefas, definir o tempo previsto de execução de cada uma e atribuir um ou mais membros do grupo para a realização das tarefas (figura 1).



Task Mode	Task Name	Duration	Start	Finish	Predecessors
	Projecto	**74 days**	**Sat 16/03/19**	**Thu 04/07/19**	
	Planeamento	**8 days**	**Sat 16/03/19**	**Wed 27/03/19**	
	Pesquisa de Tecnologias a Utilizar	8 days	Sat 16/03/19	Wed 27/03/19	
	Avaliação da Competição	8 days	Sat 16/03/19	Wed 27/03/19	
	Atribuição de Papeis	8 days	Sat 16/03/19	Wed 27/03/19	
	Desenvolvimento	**51 days**	**Thu 28/03/19**	**Wed 12/06/19**	2
	Backend	**40 days**	**Thu 28/03/19**	**Sun 26/05/19**	2
	Criação da API de autenticação de utilizadores	10 days	Thu 28/03/19	Thu 11/04/19	
	Criação de Servidor de páginas	10 days	Thu 28/03/19	Thu 11/04/19	
	Criação da API de controlo de aplicação (tratamento de listas)	20 days	Thu 28/03/19	Fri 26/04/19	
	Criação de Serviços de administrador	15 days	Fri 12/04/19	Sat 04/05/19	9
	Criação de um sistema de análise estatística do comportamento do utilizador	20 days	Fri 12/04/19	Sat 11/05/19	8
	Deploy de Micro serviços em dockers	15 days	Sat 04/05/19	Sun 26/05/19	11
	Frontend	**48 days**	**Thu 28/03/19**	**Fri 07/06/19**	2
	Criação da Landing Page	10 days	Thu 28/03/19	Thu 11/04/19	
	Criação da Página de Autenticação	10 days	Fri 12/04/19	Fri 26/04/19	15
	Criação dos Hubs	13 days	Sat 04/05/19	Thu 23/05/19	11
	Criação dos Menus de Criação das Listas	15 days	Sat 27/04/19	Sun 19/05/19	16
	Criação dos Menus de Visualização dos Itens	13 days	Sun 19/05/19	Fri 07/06/19	18
	Base de Dados	**11 days**	**Mon 27/05/19**	**Wed 12/06/19**	7
	Planeamento	**5 days**	**Mon 27/05/19**	**Mon 03/06/19**	
	Desenho da Base de Dados	5 days	Mon 27/05/19	Mon 03/06/19	17
	Criação de UML	5 days	Mon 27/05/19	Mon 03/06/19	
	Desenvolvimento	**6 days**	**Mon 03/06/19**	**Wed 12/06/19**	21
	Criação da Base de Dados	6 days	Mon 03/06/19	Wed 12/06/19	23
	População da Base de Dados	6 days	Mon 03/06/19	Wed 12/06/19	
	Finalização	**15 days**	**Wed 12/06/19**	**Thu 04/07/19**	6
	Correções Pre-Testes	6 days	Wed 12/06/19	Fri 21/06/19	
	Testes Finais	5 days	Fri 21/06/19	Fri 28/06/19	28
	Correções Pos-Testes	2 days	Sat 29/06/19	Mon 01/07/19	29
	Documentação	**2 days**	**Tue 02/07/19**	**Thu 04/07/19**	30
	Relatório do Projecto	2 days	Tue 02/07/19	Thu 04/07/19	
	Manual do Utilizador	2 days	Tue 02/07/19	Thu 04/07/19	
	Manual de Instalação	2 days	Tue 02/07/19	Thu 04/07/19	
	Manual do Programador	2 days	Tue 02/07/19	Thu 04/07/19	

Figura 1 – Planeamento de tarefas no Microsoft Project

10 Front-End

O *front-end* engloba todas as funcionalidades visuais da aplicação, bem como o *layout* desta. Este foi concebido com base na metodologia *user-friendly*, que promove uma aplicação intuitiva, acessível e simples. Tal como referido anteriormente, as linguagens utilizadas no desenvolvimento do front-end foram *HTML5* e *CSS*. Em conjunto com estas, a biblioteca *Bootstrap* [4] contribuiu para o dinamismo da aplicação relativamente à caraterística de multiplataforma, uma vez que esta permite uma adaptabilidade da *app* consoante o dispositivo que a está a exibir, mantendo a formatação original.

Para além disso, a linguagem *JavaScript* permitiu, em conjunto com as anteriores, criar uma aplicação dinâmica e fluída a nível visual, culminando em páginas que permitem ao utilizador realizar diversas funções (procurar produtos por nome ou categoria e adicioná-los a uma lista) sem ter de sair da mesma página, melhorando assim toda a experiência de utilização. É também através dos *scripts* que são realizadas funções de acesso a micro-serviços que auxiliam toda a aplicação a funcionar, como por exemplo: acrescentar produtos às listas, convidar amigos ou familiares para partilhar uma lista, aceitar convites, editar informações da conta, etc.

A ferramenta utilizada para o desenvolvimento do código das várias páginas foi o *Atom* [5]. Este editor foi o escolhido pois permitiu, através de algumas das suas funcionalidades, tornar o código mais fácil de compreender, mais rápido de desenvolver e pela facilidade em exportar para o *browser*. A edição das imagens utilizadas em toda a *app*, desde os produtos até aos logótipos ShappList, foram desenhados no *Adobe Photoshop CC 2018*.

10.1 Planeamento

Na fase de planeamento, foi definido que teriam de existir duas componentes principais: a aplicação com todas as funcionalidades e uma *landing page*. Nesta última, o utilizador pode vero que a aplicação oferece e como funciona. Para além disto, é através desta secção que o *user* se pode registar e efetuar o *login* para entrar na área da aplicação.

No registo seria pedido ao utilizador que introduzisse o seu nome de utilizador, palavra-passe, e-mail e número de telemóvel. É com base nestes dois últimos dados que se define cada *user* como único.

Adicionalmente, seria através do número de telemóvel que seriam feitos os convites para as listas de compras, ou seja, para um *user* adicionar outro a uma lista, é necessário enviar um convite para o número de telemóvel inserido na opção "Adicionar Participante".

Na secção do *login*, serão pedidos o *e-mail* e a palavra-passe correspondente para a entrada na área pessoal da aplicação.

Depois de efetuado o *sign in*, foi definido que tinham de existir três áreas distintas: as listas, os produtos e as definições.

A secção das listas teria de conter toda a informação relativa às listas de compras, como a criação, visualização, edição e eliminação destas. Aqui, o utilizador teria de ter acesso às listas criadas por si e aquelas em que estava como convidado. Poderia criar listas novas com nomes customizados e adicionar-lhes produtos e participantes. Aos produtos, poderia aumentar e minuir a quantidade destes, bem como apagá-los. Finalmente, seria possível calcular o custo final da lista e verificar qual o supermercado mais acessível dos três disponibilizados.

A área dos produtos seria a mais importante de toda a aplicação, pois permite satisfazer não só a maior parte dos utilizadores que pretendem criar listas de compras, mas também aqueles que usam a aplicação para visualizar os melhores preços nos diversos supermercados, fazendo assim da aplicação um *hub* para os preços praticados pelos supermercados disponibilizados, bem como os descontos nestes. Aqui, poderiam ser adicionados produtos às várias listas através de ferramentas de visualização como filtros por categoria ou ordem alfabética, que tornam a visualização destes mais fácil e acessível.

Por fim, a secção das definições do utilizador permitiria que este possa não só editar os seus dados pessoais, bem como aceitar ou rejeitar convites relativos a listas para os quais foi convidado.

10.2 Desenvolvimento

Na fase de desenvolvimento, todas as funcionalidades previstas na fase de planeamento foram criadas e desenvolvidas, com o acréscimo de algumas que se tornaram essenciais para o bom funcionamento da aplicação.

10.2.1 Página de introdução da aplicação – *landing page*

Figura 2 – Página de Introdução da Aplicação – *landing page*

Na página de introdução da aplicação (figura 2), foram criadas várias secções: Caraterísticas, Guia, Sobre Nós e Contactos. Esta página foi desenhada tendo como alvo principal os utilizadores que utilizam o dispositivo móvel, pois podem visualizar toda a informação sem nunca sair desta através do *scroll* da página ou das categorias na barra de navegação, , tornando mais acessível e intuitivo a utilização. Como referido anteriormente, esta página é meramente informativa para dar a conhecer ao utilizador a aplicação e as suas funcionalidades principais.

10.2.2 Página de registo de novos utilizadores

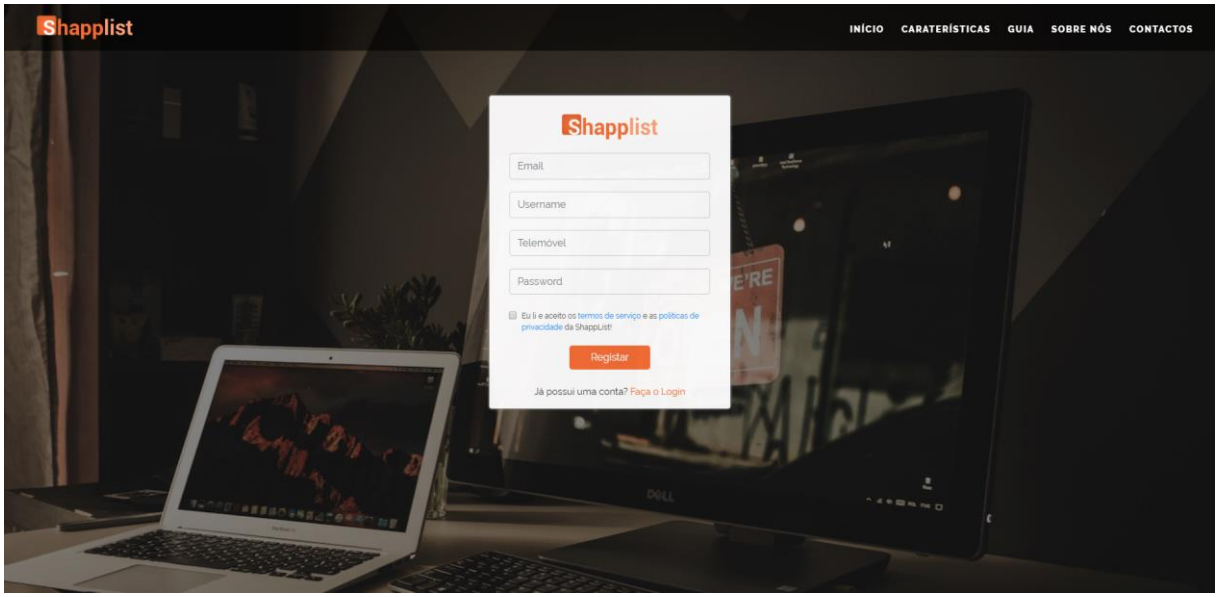


Figura 3 – Página de registo de novos utilizadores

Na zona de registo (figura 3), o utilizador pode registar-se e criar a sua nova conta pessoal. Através dos dados pedidos, este pode entrar na sua conta, ser convidado por amigos ou familiares para as listas destes, etc. Os termos e as políticas de segurança são dois documentos descritivos sobre os termos de utilização responsável da aplicação e as políticas de segurança utilizadas pela aplicação relativamente às informações pessoais dos utilizadores, respetivamente.

10.2.3 Página de entrada para utilizadores existentes – *login*

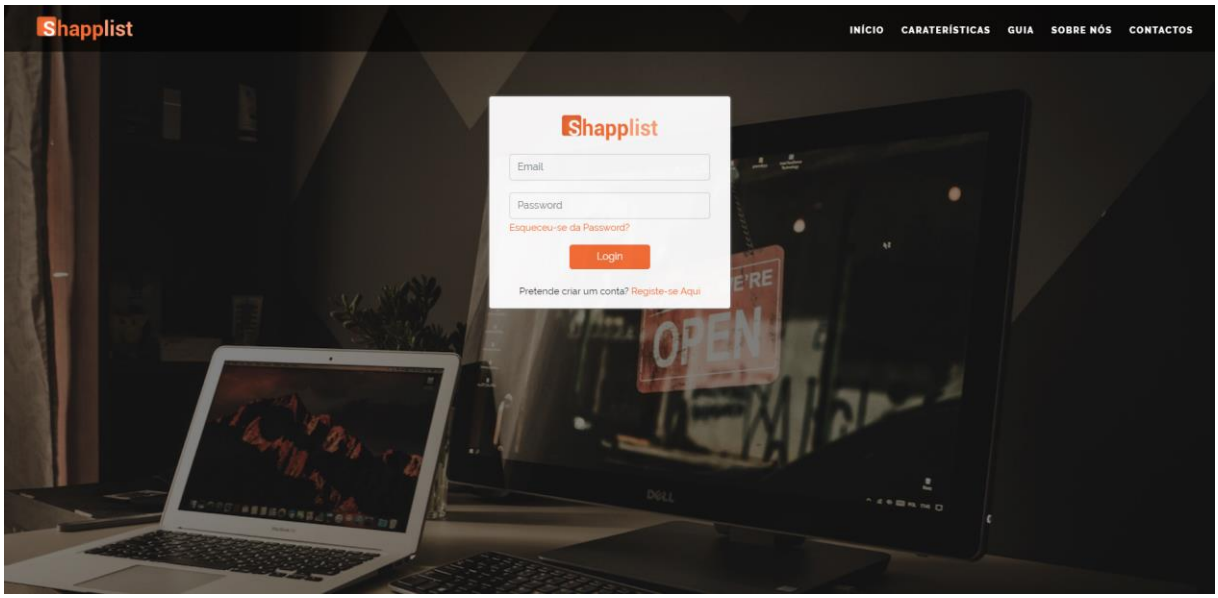


Figura 4 – Página de entrada para utilizadores existentes – *login*

No caso de os utilizadores já terem criado uma conta anteriormente, podem dar entrada na área pessoal da aplicação, a partir da página de *login* (figura 4), inserindo o email e a palavra-passe correspondente, escolhidas na criação da sua conta pessoal.

10.2.4 Página dos produtos disponibilizados – *hub* central da aplicação

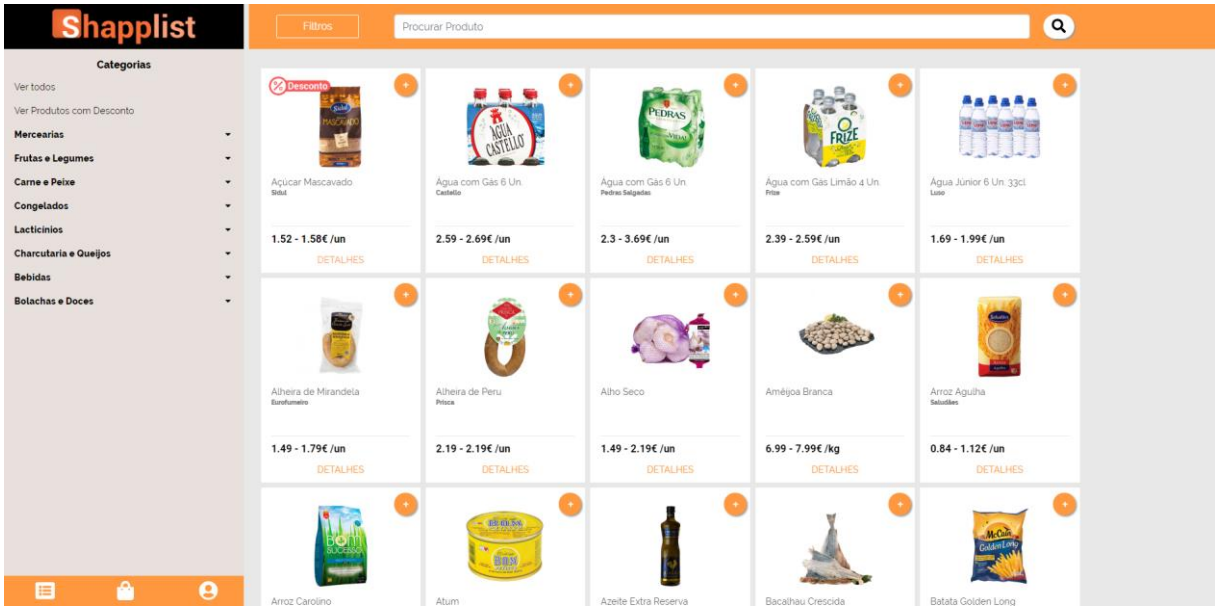


Figura 5 – Página dos produtos disponibilizados – *hub* central da aplicação

A página dos produtos (figura 5) é a página central de toda a área pessoal da aplicação. Nesta é possível visualizar todos os produtos que estão inseridos na base de dados. Estes foram escolhidos através de uma pesquisa realizada nos sites dos vários supermercados considerados para esta aplicação: Pingo Doce, Continente e Jumbo.

Os produtos estão categorizados em oito principais categorias: Mercearias, Frutas e Legumes, Carne e Peixe, Congelados, Lacticínios, Charcutaria e Queijos, Bebidas e Bolachas e Doces. Para além destes, existem ainda duas outras: ver todos os produtos – Ver Todos; e ver os produtos com Desconto nos supermercados – Ver Produtos com Desconto. Estas categorias permitem assim, filtrar os produtos de acordo com as necessidades do *user* e do que este procura para adicionar às suas listas.

O utilizador pode acrescentar produtos às suas listas ou às listas em que está como convidado de duas formas. A primeira, denominada por "*fast-add*", que se carateriza pela adição dos vários produtos de forma rápida, sem mudar de página e apenas através do um clique no botão "+" exibido em cada um dos produtos, colmatando a necessidade de utilizadores menos interessados nos preços dos produtos e mais interessados na criação rápida da lista de compras.

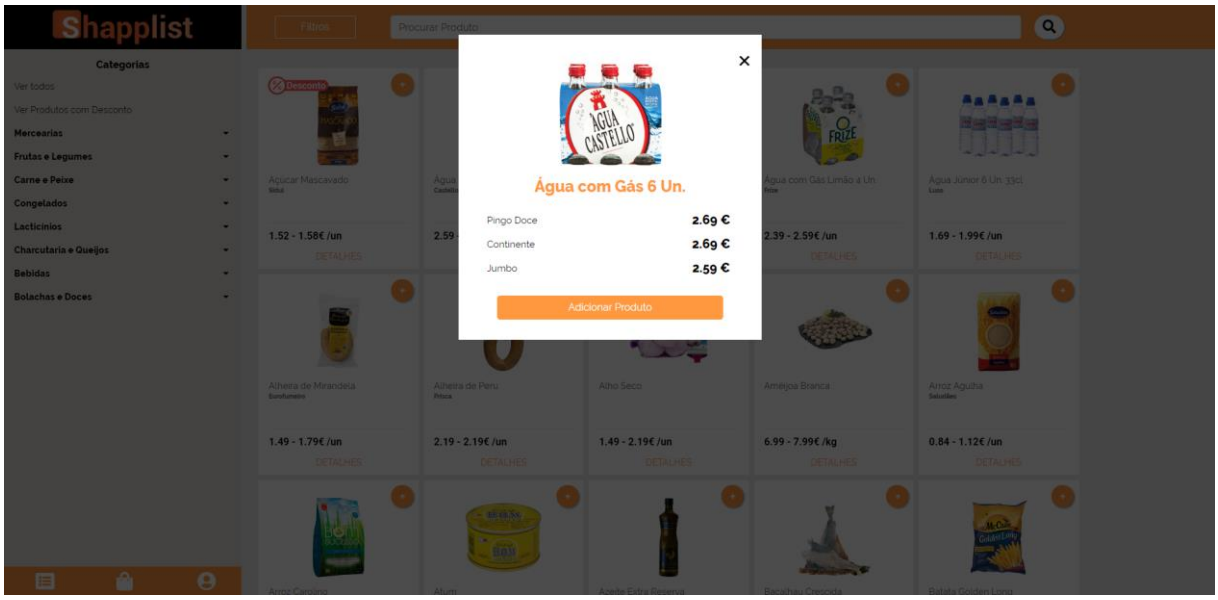


Figura 6 – Produto específico selecionado

A segunda opção é feita pela visualização dos preços dos produtos (figura 6), discriminados por supermercado. Desta forma, o utilizador tem a possibilidade de verificar os preços específicos para o produto escolhido em cada um dos supermercados e verificar, caso o produto esteja referenciado com "Desconto", qual o novo preço do produto bem como qual o supermercado em que este se encontra descontado.

Finalmente, caso o utilizador queira adicionar o produto a uma das suas listas, através do *overlay* "Adicionar A" (figura 7) é possível escolher qual a lista (ou listas) à qual quer adicionar o produto.

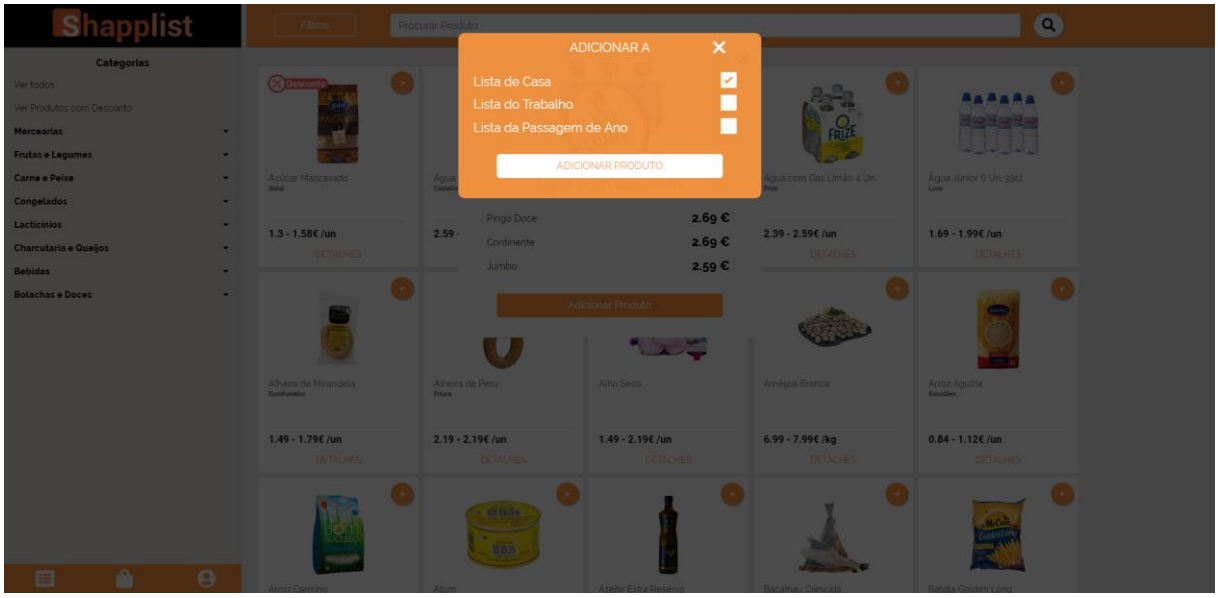


Figura 7 – Escolher uma lista (ou listas) para adicionar o produto escolhido

Quando os produtos são adicionados com sucesso, é mostrada ao utilizador uma mensagem temporizada que o informa da concretização da adição do produto à lista. Esta informação aparece na parte inferior do ecrã, próximo da barra de navegação.

No topo da página, existem dois componentes importantes para a visualização: o botão dos filtros e a barra da pesquisa. O primeiro permite auxiliar a filtragem dos produtos. Os filtros que podem ser escolhidos pelo utilizador são: "Preço mais baixo", "Preço mais alto", "De A a Z" e "De Z a A". Com o auxílio destes filtros, aliados à filtragem por categorias, o utilizador consegue visualizar com maior precisão os produtos que procura. A barra de pesquisa permite pesquisar pelo nome do produto ou pela marca deste. À medida que o utilizador for inserindo caracteres, a página vai mostrando os que satisfazem a pesquisa.

Por fim, existe uma barra de navegação no fundo da página à esquerda. Esta é semelhante em todas as páginas, sendo apenas a localização desta diferente nesta página, devido ao resto do layout. A partir dos botões de navegação, é possível ao utilizador mudar entre as várias secções dentro da aplicação: Listas, Produtos e Definições.

10.2.5 Página de visualização das listas

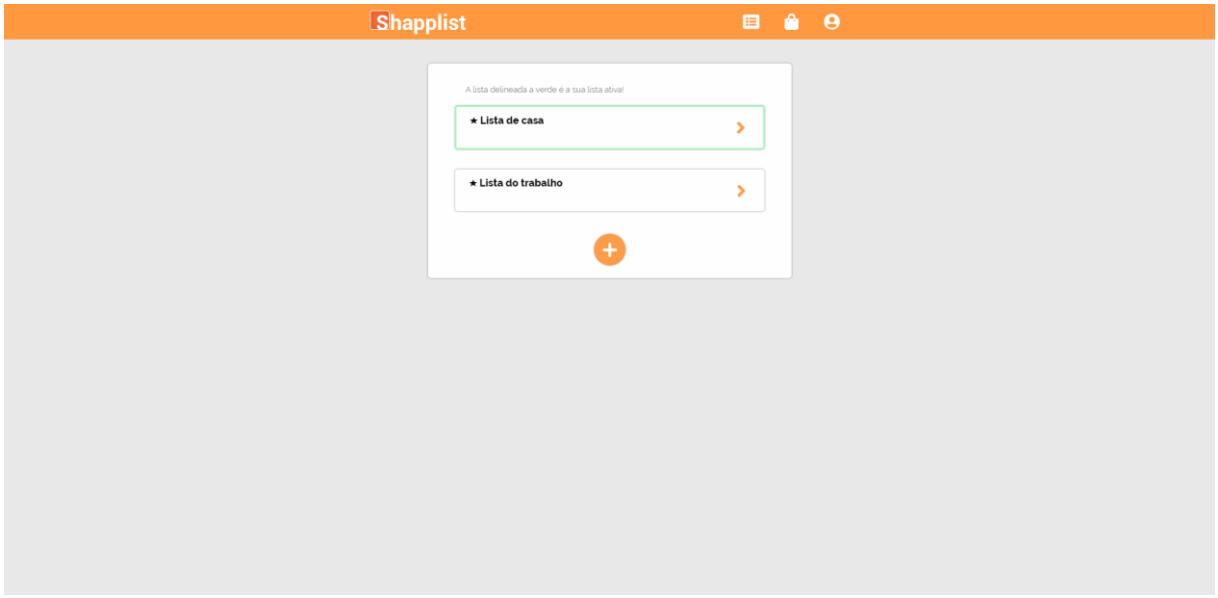


Figura 8 – Página de visualização das listas

A página das listas (figura 8) carateriza-se pela visualização de todas as listas criadas pelo utilizador, diferenciadas pela "★" antes do nome definido para cada uma das listas, bem como as listas para as quais o utilizador foi convidado. A lista ativa, usada pelo utilizador para

a adição rápida de produtos, encontra-se delineada a verde. Se a lista tiver mais do que um participante (para além do utilizador que a criou), por baixo do nome desta aparecerá o nome dos participantes dessa lista.

O utilizador pode também criar novas listas através do botão "+", localizado depois das listas.

Este *hub* é o ponto central para o *user* aceder às listas. Aquando da criação da primeira lista, essa será a sua lista "ativa". A lista ativa será utilizada para o processo de "*fast-add*", supramencionado na secção dos produtos. Para modificar a lista ativa, o utilizador tem de aceder a outra e defini-la como tal.

10.2.6 Página de criação de novas listas

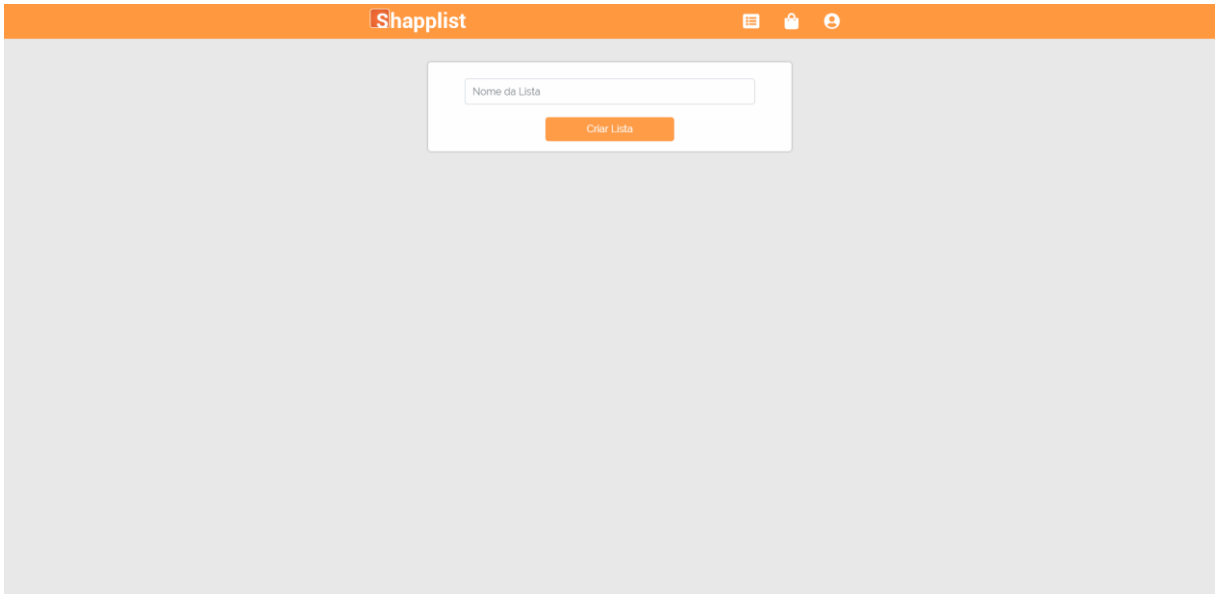


Figura 9 – Página de criação de novas listas

A página de criação de novas listas (figura 9) permite ao utilizador criar novas listas. Caso o utilizador pretenda adicionar participantes, poderá fazê-lo na página individual referente a essa lista, utilizando o número do telemóvel do utilizador a convidar.

10.2.7 Página individual de cada lista

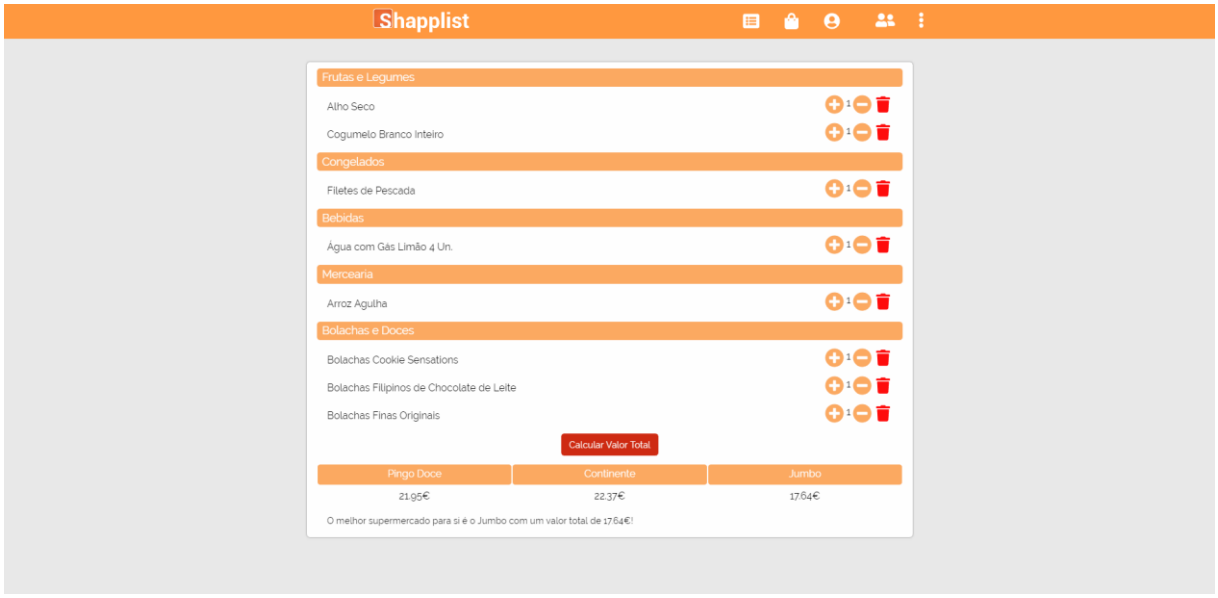


Figura 10 – Página individual de cada lista

A página referente a cada lista (figura 10) permite ao utilizador visualizar toda a informação relacionada com cada lista específica.

Os produtos adicionados à lista podem aqui ser visualizados por categoria. Depois de adicionados, é possível ao utilizador aumentar ou diminuir a quantidade dos produtos selecionados. Estes também podem ser eliminados da lista, no caso de já não ser em necessários ao utilizador.

O valor total da lista vai sendo atualizado à medida que são adicionados produtos novos e aumentada ou diminuída a quantidade de cada um. Este valor é calculado para informar o utilizador qual o supermercado mais vantajoso para os produtos selecionados. Se algum dos produtos não se encontrar num dos supermercados, este não será contabilizado para o valor final da lista, pelo que tem de ser reportado ao utilizador qual o produto que não existe e em que supermercado. Assim, o utilizador consegue definir qual o melhor supermercado para satisfazer as suas necessidades de acordo com o valor final, bem como com a disponibilidade dos artigos.

No topo da página, para além dos ícones de navegação mencionados, existem também dois novos botões: “Participantes” e “Definições da Lista”.

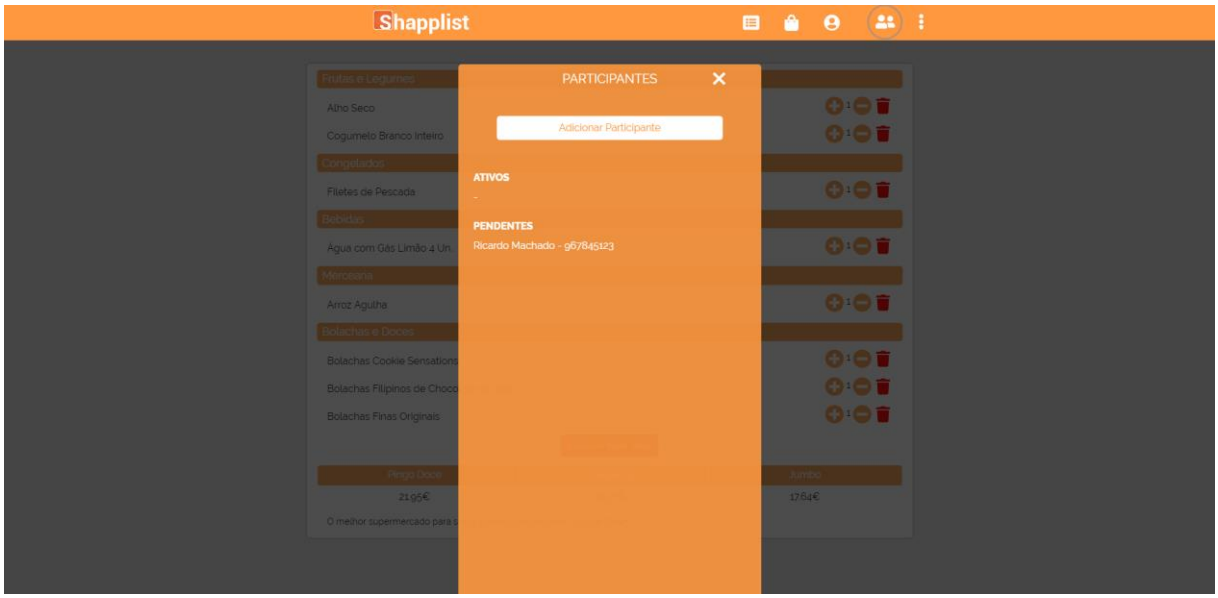


Figura 11 – Secção relativa aos participantes

A secção dos participantes (figura 11) permite ao utilizador visualizar os participantes ativos na lista, bem como os que foram convidados, mas ainda não aceitaram/rejeitaram o convite que lhes foi feito, denominados de pendentes). Os *users* ativos na lista podem ser removidos pelo administrador e criador da lista.

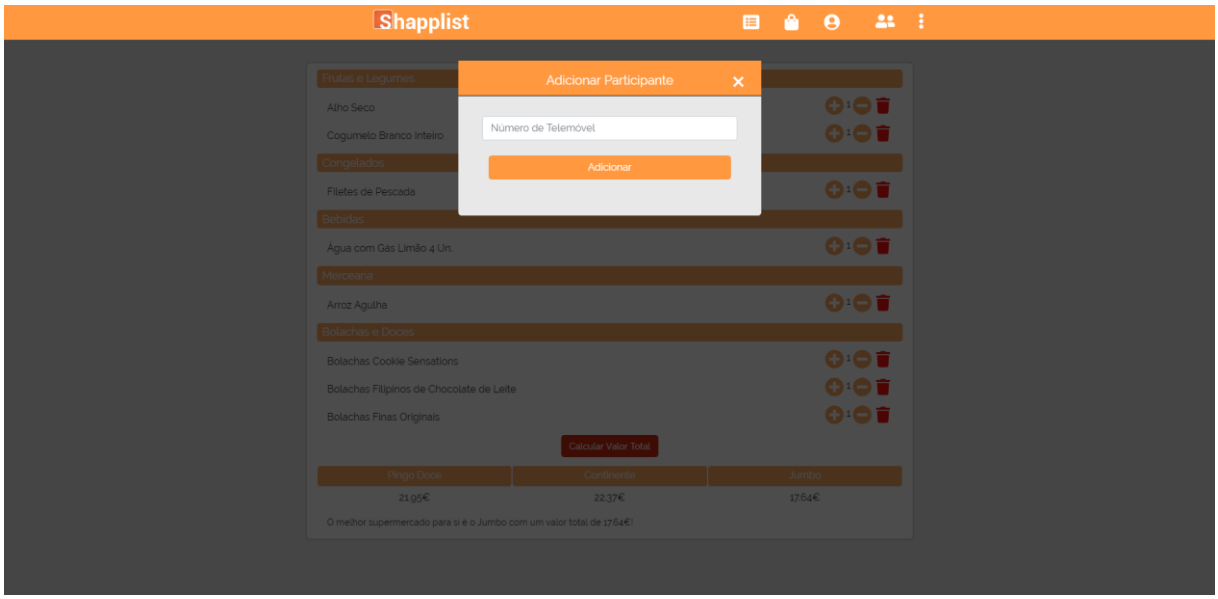


Figura 12 – Secção para adicionar novos participantes

Novos participantes podem também ser adicionados aqui (ver figura 12), como referido anteriormente, pela inserção do número de telemóvel associado à conta do utilizador convidado.

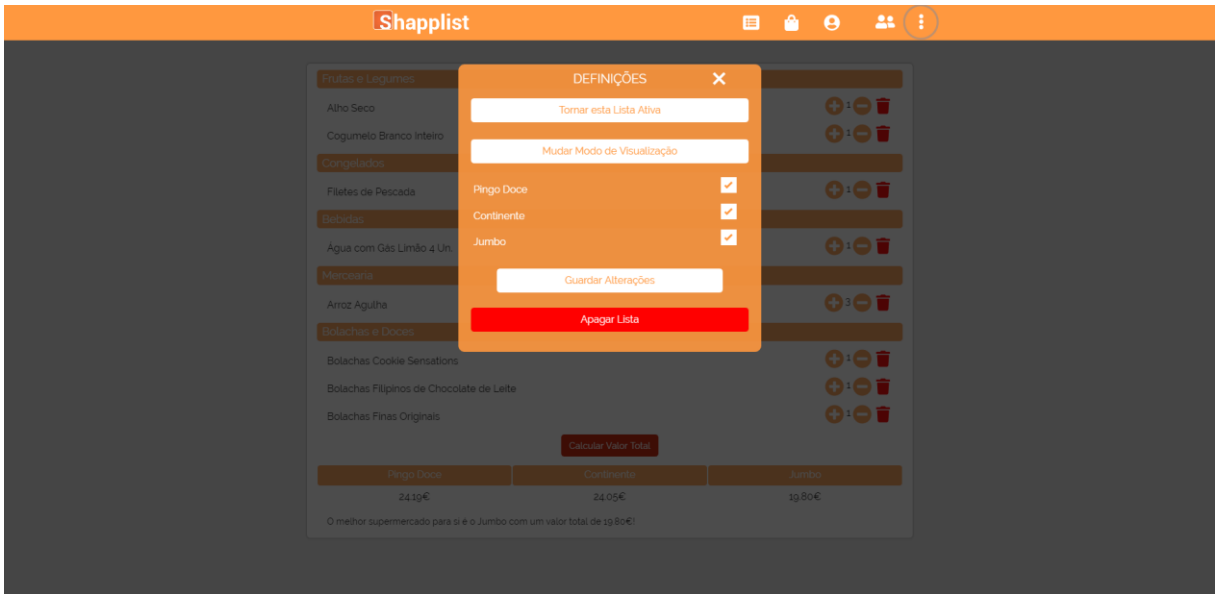


Figura 13 – Secção de definições da cada lista

A área de definições da lista (figura 13) possibilita ao utilizador tornar a lista ativa, no caso de pretender que esta se torne na sua lista ativa para o "*fast-add*". No caso de a lista visualizada ser a lista ativa, essa opção não irá estar disponível.

O utilizador tem a possibilidade de definir se para o cálculo final da lista pretende contabilizar todos os supermercados disponíveis ou limitá-los à sua escolha. Através de três *checkboxes*, em que o valor por defeito é *checked* (verificado), o utilizador pode retirar um ou dois dos supermercados que não sejam opção para finalizar as suas compras, independentemente dos preços praticados nestes. Assim, o *user* pode obter o valor final mais em conta consoante os supermercados escolhidos, não sendo calculado o(s) valor(es) final para os não contemplados. Para confirmar a configuração, apenas é necessário confirmar as alterações.

O administrador da lista é o único utilizador que pode apagar as listas por si criadas. Por outro lado, os utilizadores convidados nas diversas listas apenas podem sair da lista, não tendo permissões para as apagar.

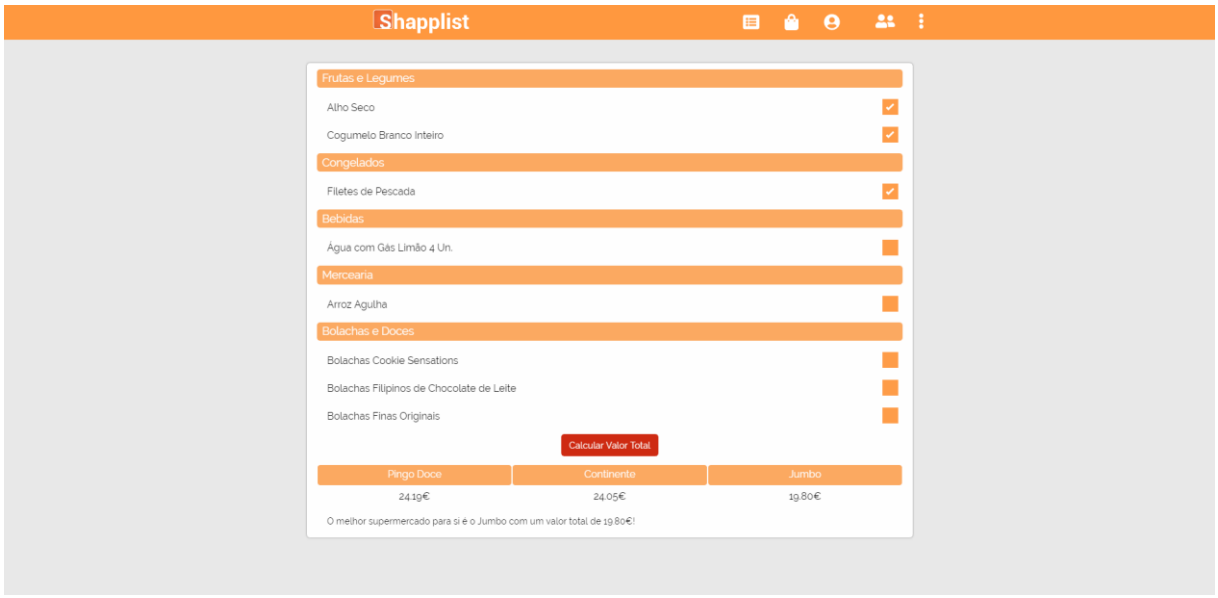


Figura 14 – Modo de visualização das listas para compras

Por fim, foi definido que para além das listas servirem para calcular o valor final mais em conta para o utilizador, deveriam também permitir a confirmação dos artigos já colocados no cesto/carrinho do supermercado, "riscando-os" da lista. Para isso, foi criado o "*Modo Shop*" (figura 14) que permite ao *user* mudar o modo de visualização para, em vez de aumentar/diminuir a quantidade dos produtos ou apagá-los, poder confirmar os produtos à medida que os coloca no cesto.

10.2.8 Página de definições pessoais do utilizador

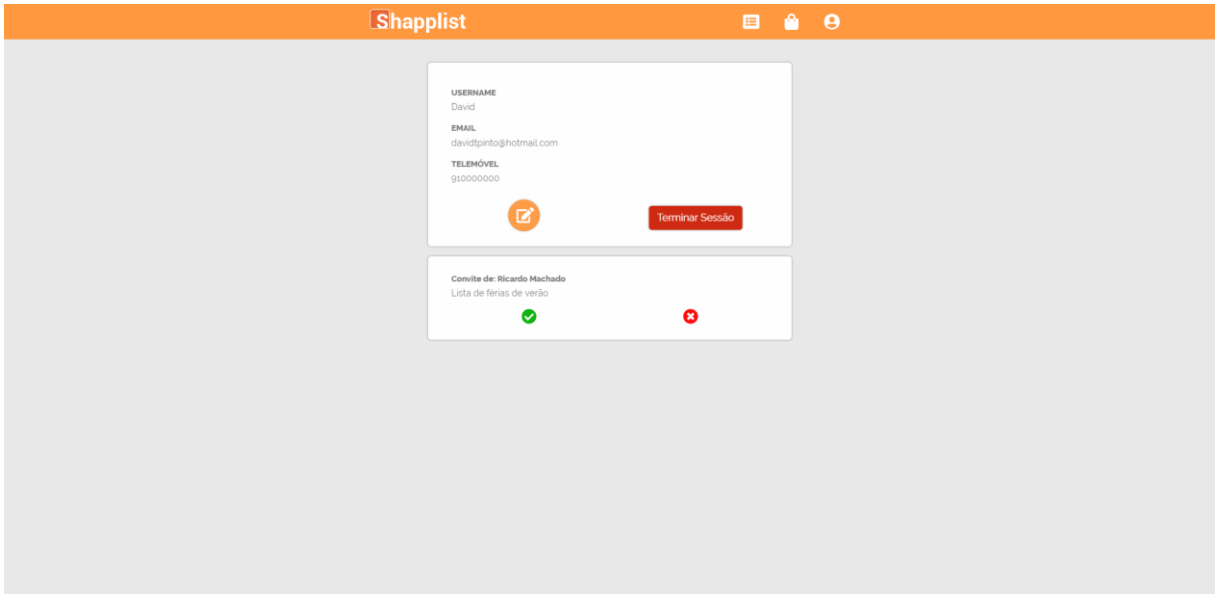


Figura 15 – Página de definições pessoais de cada utilizador

A página de definições (figura 15) permite ao utilizador verificar as suas informações pessoais como: *Username* ou nome de utilizador, o *e-mail* fornecido e utilizado para fazer *login* e ainda o telemóvel. Alguns destes dados podem ser editados.

Aqui, é possível ao utilizador terminar a sessão. Quando a sessão é terminada o utilizador será reencaminhado para a página de *login* da aplicação. Caso o *user* queira reentrar, terá de fornecer novamente os dados de *sign in* – o *e-mail* e a palavra-passe. Por outro lado, se o utilizador quiser fechar a aplicação, mas manter a sessão, apenas tem de a fechar e a sessão será mantida durante algum tempo até o *token* da sessão expirar.

Nesta página, foi definido que o utilizador poderia visualizar os convites feitos pelos outros utilizadores que o convidem para as suas listas, podendo estes ser aceites ou rejeitados. No caso de não existirem convites pendentes, será apresentada uma mensagem informativa ao *user*.

10.2.9 Página de edição dos dados pessoais do utilizador

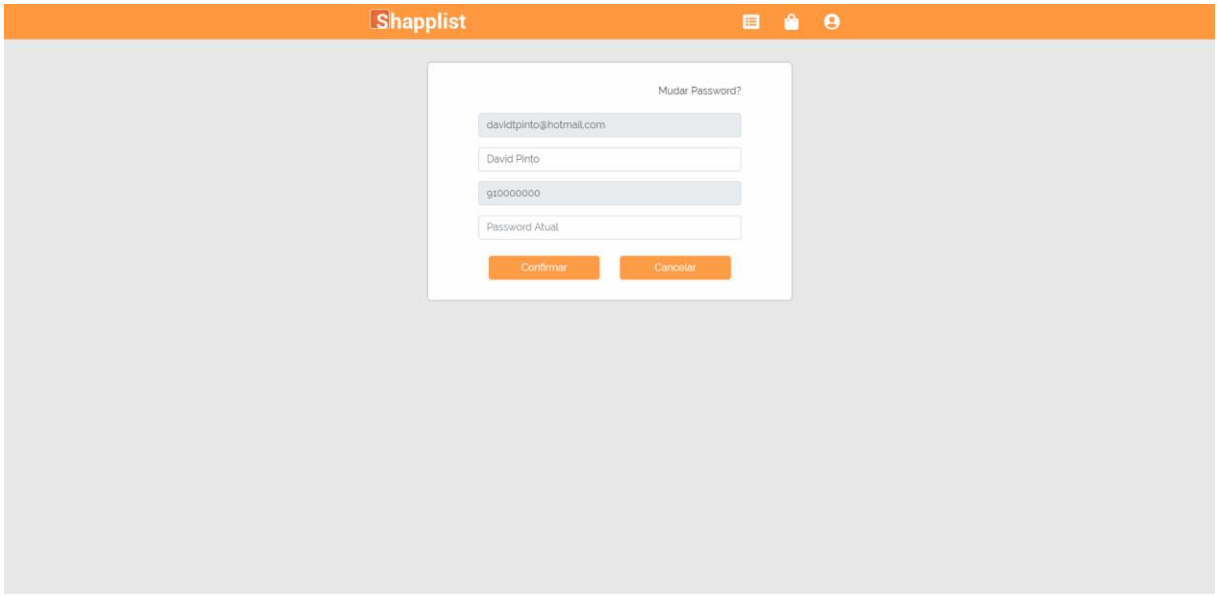


Figura 16- Página de edição dos dados pessoais do utilizador

A página de edição dos dados pessoais (figura 16) permite que o utilizador edite alguns dos campos definidos a quando do seu registo. Os campos editáveis são o nome de utilizador e a palavra-passe, sendo que, em ambos os casos, a verificação da veracidade da palavra-passe atual funciona como forma de guardar a alteração das informações.

10.2.10. Aplicação em formato "*mobile*"

Tal como referido anteriormente, todo a aplicação foi desenvolvida tendo como principal objetivo ser multiplataforma, ou seja, independentemente do dispositivo em que estaria a ser exibida, estaria 100% funcional, sem problemas de visualização ou das funcionalidades acima descritas. Para tal, foi utilizado, em conjunto com as linguagens *HTML5* e *CSS*, a biblioteca *Bootstrap* que permitiu uma maior flexibilidade de todas as páginas relativamente aos dispositivos utilizados.

As funcionalidades são praticamente as mesmas ao longo de todas as páginas entre o modo computador e o modo telemóvel, salvo algumas exceções.

As categorias, ao contrário do que acontece na versão da aplicação no computador, estão compressas num único botão, no topo do ecrã (figura 17), que permite ao utilizador visualizar toda a página de forma fluída.

Figura 17 – Página de introdução da aplicação – versão telemóvel

As páginas relativas ao registo e ao *login* estão semelhantes às referidas anteriormente, na secção da versão do computador.

A página dos produtos é uma das que tem mais alterações visuais relativamente à versão do computador (figura 18). Devido ao facto de o tamanho de ecrã disponível ser menor do que

anteriormente, foi definido que os produtos disponíveis deveriam ser o ponto principal da página, estando os filtros e as categorias "escondidos", podendo o utilizador aceder-lhes através dos respetivos botões.

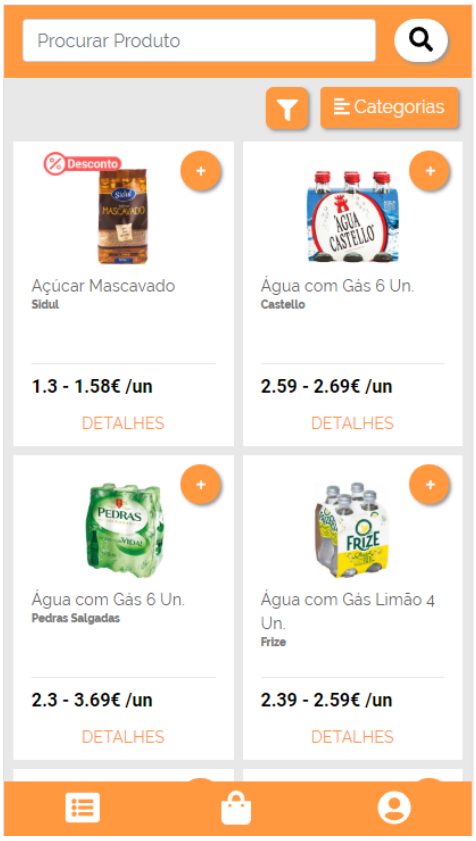


Figura 18 – Página dos produtos – versão telemóvel

Tal como anteriormente, os produtos podem ser adicionados de modo rápido, através dos botões "+" disponibilizados em cada um destes, ou através dos "Detalhes" de cada produto, sendo posteriormente possível ver os preços discriminados por supermercado e, caso existam, com os descontos destacados, bem como adicionar a uma ou mais listas.

Figura 19 – Página individual referente a cada lista

As páginas de visualização das listas e de criação de novas listas são idênticas às explicadas anteriormente.

A página relativa a cada lista individual (figura 19) é a que sofreu mais alterações, a par da página dos produtos. Tendo em conta a fluidez e simplicidade traçadas para toda a aplicação, foi definido que o botão para apagar cada produto seria suprimido. Por isso, a funcionalidade para apagar qualquer um dos artigos passou a ser efetuado através de um *hold* do produto a apagar, ou seja, o longo clique é feito e é apresentado um botão para eliminar o produto da lista.

Todas as restantes funcionalidades desta página, bem como das páginas de definições pessoais do utilizador e edição destas são idênticas às referidas anteriormente.

Todas as páginas estão adaptadas para os vários tipos de visualização, não só os referidos, mas também dispositivos com resoluções intermédias, como os *tablets*. Estas foram desenvolvidas para, através do *CSS* e do *HTML5*, se adaptarem aos dispositivos sem nunca ser necessário criar páginas semelhantes para as diferentes resoluções, facilitando assim toda a operação auxiliar da aplicação.

10.3 Fase de testes e Problemas encontrados

Na fase de testes foram concebidos e realizados diversos testes ao *front-end*. Estes permitiram ter uma visão concreta, não só dos erros nas páginas, bem como das funcionalidades do ponto de vista dos utilizadores, algo que foi definido como fulcral e fundamental para todo o desenvolvimento do projeto.

Os testes foram feitos ao longo do todo o processo de criação e desenvolvimento das várias páginas, levando a que estas sejam produto de várias fases, de vários *layouts* testados, de várias tentativas e erros.

As páginas relativas à visualização dos produtos e das listas individuais são as principais páginas de toda a aplicação, pelo que são as que têm mais funcionalidades nelas contidas. Muitas são fruto de tentativas de colmatar necessidades pensadas aquando da criação das páginas, e, do ponto de vista do utilizador, são ferramentas que fazem sentido no âmbito da aplicação bem como no dia-a-dia. Foram desenvolvidas e aperfeiçoadas ao longo de todo o projeto, com o objetivo principal de satisfazer todo o tipo de utilizadores que possam vir a usar a aplicação, desde os *users* mais atentos aos valores dos produtos e aos respetivos descontos, até aos que apenas pretendem criar rapidamente uma lista de compras.

Alguns dos problemas encontrados relativos ao *front-end* e à visualização da *web app* prenderam-se pelo facto de a forma definida para criação da aplicação não ser a mais usual, ou seja, utilizar as linguagens de programação de *web* e torná-las numa *web app*. Esta abordagem permitiu facilitar a adaptação entre o modo de computador e de telemóvel. Contrariamente ao que acontece em programas de criação de aplicações de telemóvel, como por exemplo o *Android Studio* [6], em que as funcionalidades estão à distância de um clique, estas, por vezes, são bastante difíceis de replicar usando apenas as linguagens *web* conjugadas com *Javascript*. Um destes casos, foi a eliminação individual de cada produto nas listas através do *hold*, que é uma funcionalidade bastante usual nas aplicações *mobile*.

Em suma muitas das funcionalidades inicialmente planeadas, bem como aquelas que foram sendo pensadas e desenvolvidas ao longo de todo o projeto, foram realizadas com sucesso.

11 Back-End

O *back-end* representa todo o código que permite o bom funcionamento de toda a aplicação. Este tem a capacidade de dinamicamente fornecer as páginas *HTML* necessárias a cada secção da aplicação e fazer as ligações necessárias à base de dados, de modo a disponibilizar a informação necessária às funções *JavaScript* que são executadas no *front-end*.

A *web framework* Django foi escolhida para o desenvolvimento de todos os componentes *back-end* da aplicação. Esta escolha foi feita tendo em conta a experiência e a familiaridade já existente do grupo com a *framework*, mas também com as próprias características e vantagens que esta opção providencia. Baseia-se na linguagem de programação *Python*, especificamente a versão 3.7, que permite um ambiente de programação altamente flexível e versátil, o que facilitou muito o desenvolvimento das funcionalidades do *back-end*. Para além disto, Django permite também uma interligação eficiente e simplificada com as funcionalidades do *front-end*, uma implementação nativa de sistemas de segurança (como a proteção a ataques à integridade da base de dados) e funcionalidades que permitem o envio de *e-mails* após o registo de um novo utilizador [7].

Um dos objetivos principais para o desenvolvimento do *back-end* era a criação de uma aplicação escalável e modular, para tal foi feita a divisão do projeto em quatro aplicações Django diferentes, onde cada uma é responsável por uma determinada quantidade de funções. Esta divisão foi realizada de modo a isolar o mais possível cada uma das aplicações, alcançando assim uma arquitetura baseada em micro-serviços. Este isolamento permite uma melhor disponibilidade da aplicação, maior eficiência na identificação e correção de erros, tolerância a faltas e uma maior facilidade de adição de novas funcionalidades, sendo este último um dos pontos fulcrais do desenvolvimento do projeto – escalabilidade [8].

Para cada um dos micro-serviços criados, foram criadas máquinas virtuais onde estes são executados. Para tal foi utilizada a tecnologia *Docker* que fornece um sistema de repositório *online* onde é possível guardar ficheiros (denominados *Docker Images*). Estes ficheiros permitem definir várias especificações, como qual micro-serviço executar e as dependências a serem instaladas [10]. Quando é executada uma *Docker Image* é criado um ambiente virtual (denominado *container)* onde todas as especificações são executadas. Este processo permite uma generalização e simplicidade na criação de novas *VMs*, pois todo processo é feito a partir de um único comando. Para além da tecnologia *Docker*, foram também utilizados os serviços do servidor *web Nginx*. Este permite criar uma primeira "camada" entre o *back-end* e o *front-*

end, fornecendo funcionalidades de balanceamento de carga nas *VMs* e redirecionamento de pedidos e centralizando toda a aplicação a um único endereço IP/*domain* facilitando qualquer tipo de pedidos *web* e a passagem de *cookies* entre os diversos micro-serviços, visto que cada uma é referente a um único *domain* [11].

11.1 Planeamento

No planeamento do *back-end* foi delineada a criação de 4 serviços individuais: *pageServer*, *authService*, *productService* e *listService*.

O serviço *pageServer* seria responsável por corretamente identificar e fornecer as páginas *HTML* corretas ao utilizador tendo em conta a "secção" da aplicação em que se encontra.

O serviço *authService* seria responsável por todas as funcionalidades relacionadas com o registo de utilizadores e a autenticação dos dados deste, quando necessário.

O serviço *productService* seria responsável por fornecer a informação necessária em relação aos produtos, guardada na base de dados, de modo a estes serem representados dinamicamente na aplicação, consoante os filtros escolhidos pelo utilizador.

Finalmente, seria criado o serviço *listService*. Este seria o mais complexo e extenso de toda a aplicação, pois é o que iria conter a grande maioria das funcionalidades da aplicação, tais como: a criação de listas, adição de participantes, adição de produtos e a gestão destes dentro da lista, entre outras.

Para realizar o *deployment* de todos os micro-serviços, será construída uma rede de máquinas virtuais (VMs), utilizando o formato *OVF* [9], de modo a obter uma configuração escalável, modular e tolerante a faltas. O número de *VMs* dedicado a cada micro-serviço é inteiramente dependente das necessidades de cada um destes, sendo que a adição de novas instâncias é bastante simples e eficiente. Será também utilizada a tecnologia *Docker* para a distribuição dos vários micro-serviços, o que permite uma automatização em termos de configurações de rede, dependências em termos de *packages* necessárias para a aplicação e a fácil iniciação de cada micro-serviço através de um único comando [10].

11.2 Desenvolvimento

Na fase de desenvolvimento, os serviços descritos na fase de planeamento foram criados e implementados com sucesso. A finalidade de cada um dos serviços reteve os conceitos originais propostos, com a adição de novas funcionalidades ao longo do desenvolvimento do *back-end*, consoante as necessidades e problemas que foram surgindo.

Para a funcionalidade do projeto, é necessária uma interação fiável e eficiente entre o *front-end* e o *back-end*. Para alcançar este equilíbrio é utilizada uma combinação de *AJAX* e objetos *JSON*. *AJAX* permite a sincronização de funções *JavaScript*, executadas nas páginas *HTML*, e objetos *XMLHttpRequest*, que são usados para enviar os pedidos de informação ao *back-end* [12], sendo que estes pedidos são feitos com o método POST [13]. Estes pedidos são enviados em objetos *JSON*, que têm um formato semelhante ao de um dicionário, onde cada campo é descrito com um par chave e valor [14] (figura 20).

```
function sendAjaxDeleteList() {
    //esta função envia um pedido AJAX ao servidor para eliminar a lista, utilizando o ID desta
    //a função só vai estar disponível ao criador da lista
    console.log('Creating ajax object');
    var ajaxRequest = new XMLHttpRequest();
    //após receber a resposta da API, o seguinte código é executado
    ajaxRequest.onreadystatechange = function () {
        if (ajaxRequest.readyState == 4) {   // XMLHttpRequest.DONE == 4
            if (ajaxRequest.status == 200) {
                var responseData = JSON.parse(ajaxRequest.responseText);
            }
            else if (ajaxRequest.status == 400) {
                console.log('There was an error 400');
            }
            else {
                console.log('something else other than 200 was returned');
            }
        }
    };
    console.log('Sending ajax');
    //envio para a api específica que trata deste pedido
    ajaxRequest.open("POST", "https://shapplist.pythonanywhere.com/api/listServices/api_deleteList", true);
    ajaxRequest.setRequestHeader("content-type", "application/json; charset=utf-8");
    //data é enviada no formato de objetos JSON, semelhantes a dicionários, usando par chave e valor
    data = JSON.stringify({ 'listID': localListID });

    ajaxRequest.send(data);
}
```

Figura 20 - Exemplo de um pedido AJAX enviado pelo *front-end*, neste caso para eliminar uma lista

Os pedidos do *front-end* são recebidos pelo *back-end* nos micro-serviços específicos, dependendo da finalidade da informação. Dentro destes micro-serviços existem múltiplas *APIs* responsáveis pela lógica de cada funcionalidade do micro-serviço (figura 21). Estas *APIs* são os pontos de convergência de toda a aplicação, na medida que recebem os pedidos do *front-end* e devolvem a informação necessária [15].

```python
def api_deleteList(request):
    #informação recebida pelo front-end
    dict_data = json.loads(request.body.decode('utf-8'))
    #identificador da lista utilizado para eliminar da base de dados
    listID = dict_data['listID']
    data = {}

    #query que elimina a lista a partir do ID desta
    if(query_deleteListByID(listID)):
        #query que define uma nova lista como default após a eliminação
        if(query_updateDefaultListOnDelete(listID)):
            data['status'] = 200 # 200 representa sucesso
            data['message'] = 'List successfully deleted'
        else:
            data['status'] = 400 # 400 representa erro
            data['message'] = 'Could not update default list'
    else:
        data['status'] = 400
        data['message'] = 'Could not delete list'

    #informação é devolvida ao front-end usando objeto JSON
    return JsonResponse(data)
```

Figura 21 - API no back-end responsável por responder ao pedido de eliminação de uma lista

A identificação de um utilizador é feita a partir de uma *token*. Estas são cadeias de caracteres aleatórios, utilizando o nome escolhido pelo utilizador para a criação da sua conta. As *tokens* (figura 23) são guardadas nas *cookies* (figura 22) do *browser* do utilizador [16].

Figura 22 - Cookie guardada no browser

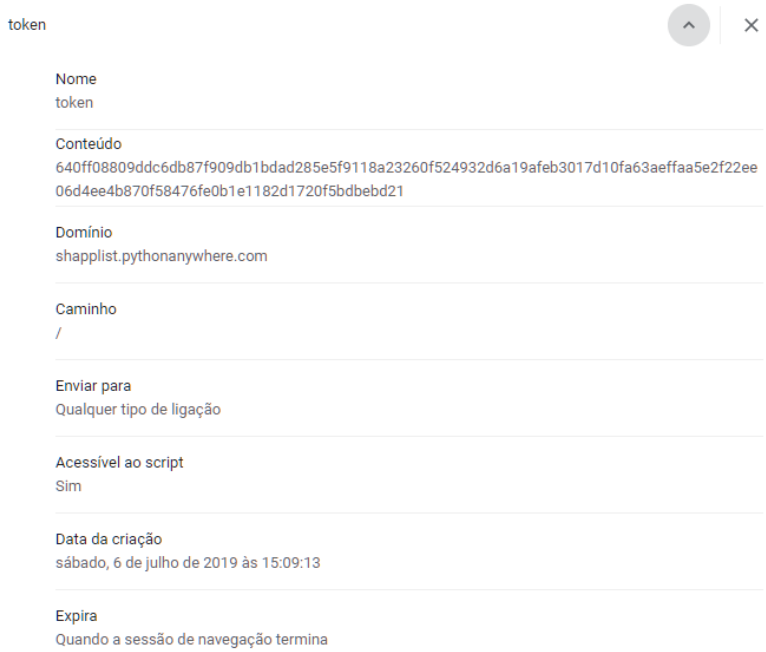


Figura 23 - Conteúdo da cookie definido pela token

A existência destas permite à aplicação identificar que o utilizador está *signed in*, redirecionando-o de volta para a página de entrada caso não este não tenha a *cookie*. As *tokens* são também guardadas na base de dados de modo a não ser necessário a criação de uma nova sempre que o utilizador acede à aplicação. Para impedir que a base de dados fique sobrecarregada com entradas sempre que uma *token* nova é criada, estas são periodicamente removidas por um *script* que verifica a data de criação da *token*. Este intervalo tempo é definido usando *Unix time* [17], sendo que as *tokens* são removidas a cada 72 horas. Caso o utilizador faça o *sign in* na aplicação antes das 72 horas, a data de criação da *token* é reposta.

Como foi referido na fase de planeamento, o *back-end* é dividido em 4 micro-serviços: *pageServer*, *authService*, *productService* e *listService*.

11.2.1 Serviço *pageServer*

O serviço *pageServer* é responsável por identificar a página *HTML* necessária para cada secção da aplicação e apresentá-la ao utilizador. Este comportamento é definido nos ficheiros *urls.py* e *views.py* da *framework* Django. Esta ligação direta entre os dois ficheiros permite à aplicação atualizar rapidamente as páginas *HTML* consoante as ações do utilizador. Cada *URL* da aplicação é descrito no ficheiro *urls.py*, que identifica a *view* correspondente no ficheiro *views.py* [18]. Dentro deste é feita uma avaliação do *URL* pedido e é encontrada a página *HTML* específica, que é devolvida ao utilizador. Dentro de cada uma das *views* deste serviço é também feita uma verificação que analisa a *cookie* do utilizador, averiguando se esta contém ou não uma *token* (figura 24). Esta verificação impede que o utilizador aceda à aplicação sem uma *token*, mas também permite que este não tenha de fazer o processo de *sign in* sempre que pretende usar a aplicação.

```python
def products(request):
    accept = False

    #caso o utilizador tenha uma token é feita uma verificação ao tempo de expiração desta
    if(not request.COOKIES.get('')):
        temp_token = request.COOKIES.get('token')
        if(temp_token != None ):
            #check_token verifica se a token ultrapassou o tempo de expiração
            accept = check_token(temp_token)

    #se o utilizador tiver uma token e esta ainda nao tiver expirado é imediatamente redirecionado para a pagina de produtos
    if(accept):
        template = loader.get_template('pageServer/tempProducts.html')

    #se o utilizador não tiver uma token, ou esta ja tenha expirado é redirecionado para a pagina de login
    else:
        template = loader.get_template('pageServer/login.html')


    context = {
        'debug' : 'debug'
    }

    return HttpResponse(template.render(context, request))

```

Figura 24 - Função que verifica a token do utilizador quando este tenta aceder à página dos produtos pelo URL desta

11.2.2 Serviço *authService*

O serviço *authService* trata de todas as funcionalidades relativas à autenticação do utilizador, registo de novos utilizadores, visualização das suas informações pessoais e envio de *e-mail* após a criação de uma nova conta para confirmar o ato. Este serviço trata da verificação dos dados do utilizador quando este tenta realizar o *sign* in, comparando os dados introduzidos com os que já existem na base de dados (figura 25). Caso o utilizador tenha introduzido corretamente as suas credencias, este será redirecionado para a página dos produtos e irá também receber uma *cookie* com a sua *token*. Quando o utilizador termina a sua sessão, esta *token* é apagada.

```python
def api_login(request):

    #dicionário enviado como resposta para o front-end
    responseData = {}
    if(request.method =="POST"):

        #informação recebida pelo front-end
        dict_data = json.loads(request.body.decode('utf-8'))
        temp_email = dict_data['email']
        temp_password = dict_data['password']
        #query que verifica se o utilizador introduziu corretamente as suas credenciais
        userID = check_info_query(temp_email,temp_password)

        if(not userID):
            responseData['updatedDefaultList'] = False
            responseData['status'] = 400
            responseData['message'] = 'User creds wrong '

        elif(userID):
            #obtenção da lista ativa do utilizador, caso ela exista, ou seja, é diferente de 0
            temp_val = query_getUserDefaultList(userID)
            if( temp_val != 0):
                responseData['updatedDefaultList'] = True
                responseData['defaultList'] = temp_val
            else:
                responseData['updatedDefaultList'] = False

            responseData['status'] = 200
            responseData['message'] = 'User creds ok '
            #criação da token do utilizador a partir do seu email e identificador
            responseData['token'] = create_token(temp_email,userID)
        else:
            print('Query error')

    else:
        responseData['status'] = 500
        responseData['message'] = 'Not POST'

    #resposta é enviada de volta para o front-end
    return JsonResponse(responseData)
```

Figura 25 - API que analisa a informação introduzida pelo utilizador quando este tenta realizar *o sign in*

Quando é realizado o registo de um novo utilizador, antes de esta informação ser enviada e registada na base de dados, é primeiro avaliada a integridade dos campos "*e-mail*" e "número de telefone". O *e-mail* fornecido é avaliado, verificando se tem o caracter "@", se tem uma extensão permitida (google.com, outlook.pt, etc.) e se contém caracteres inválidos ("%", "/", etc). A avaliação ao número de telemóvel é menos complexa, pois é apenas verificado se o número fornecido corresponde a um número de telemóvel com o indicativo "9", que representa os serviços de comunicações móveis em Portugal [19], e se o número inserido tem nove dígitos.

Com estas verificações feitas a informação introduzida é enviada ao *back-end* e esta é comparada com a que já existe na base de dados. Não é permitido que dois utilizadores tenham o mesmo *e-mail* ou número de telemóvel, portanto, caso a informação enviada no registo não respeite esta regra, ela é recusada e o utilizador recebe um aviso. Se as informações forem

válidas, o registo do utilizador é feito, este recebe uma *token* e é redirecionado para a página principal da aplicação, a página dos produtos. Após um registo com sucesso, o utilizador irá receber no seu correio eletrónico um *e-mail* com um *hyperlink* para confirmar a criação da sua conta (figura 26). Este *link* é dinamicamente criado para cada utilizador diferente utilizando a sua *token* pessoal.

```python
def api_signup(request):
    responseData = {}
    #informação recebida pela base de dados
    dict_data = json.loads(request.body.decode('utf-8'))
    #informação introduzida pelo utilizador para o registo
    temp_username = dict_data['username']
    temp_password = dict_data['password']
    temp_email = dict_data['email']
    temp_phoneNum = dict_data['phoneNum']
    new_data = {}

    if(request.method =="POST"):
        #query que verifica se email ou número de telemovel ja está em uso
        if(not query_checkIfUserWithMailOrPhone(temp_email,temp_phoneNum)):
            #query que adiciona utilizador à tabela userss
            if(create_user(temp_username,temp_password,temp_email)):
                #query que obtém o ID do utilizador para ser introduzido na tabela users_extended
                new_data = get_user_by_email(temp_email)
                #query que adiciona o user à tabela users_extended
                if(create_user_extended(new_data['userID'],temp_email,temp_phoneNum)):
                    responseData['status'] = 200
                    responseData['message'] = "User succesfully created"
                    #token criada a partir do nome do utilizador e do seu ID
                    responseData['token'] = create_token(temp_username, new_data['userID'])
                    #token criada para criar um hyperlink para ativação da conta
                    temp_ver_token = generateHash(temp_username, temp_email)

                    #envio do email de confirmação de conta
                    if(query_addVerificationTokenToDatabase(temp_ver_token, new_data['userID'])):
                        support_sendEmail(temp_email, temp_ver_token, temp_username)

                else:
                    if(delete_user(new_data['userID'])):
                        responseData['status'] = 400
                        responseData['message'] = "Could not create user extended "
            else:
                responseData['status'] = 400
                responseData['message'] = "Could not create user "

        elif(query_checkIfUserWithMailOrPhone(temp_email,temp_phoneNum)):
            responseData['status'] = 300
            responseData['message'] = "User creds already exist "
    else:
        responseData['status'] = 500
        responseData['message'] = 'Not POST '


    return JsonResponse(responseData)
```

Figura 26 - API responsável pelo registo de um novo utilizador e pelo envio do email de confirmação de conta

Por último, o serviço *authService* permite também ao utilizador ter uma área pessoal com as informações da sua conta, permitindo inclusive, a alteração do seu "*username*" e "palavra-passe".

11.2.3 Serviço *productService*

O serviço *productService* permite ao *front-end* apresentar ao utilizador todos os produtos contidos na base de dados de um modo dinâmico e eficiente. Este permite ao utilizador pesquisar produtos manualmente, filtrar por preço, por nome, utilizar a procura por categorias ou subcategorias e adicionar produtos às suas listas pessoais. A pesquisa manual, utilizando a barra de pesquisa da aplicação, é tratada de um modo flexível pelo *back-end*, permitindo ao utilizador procurar pelo nome específico de um produto ou pelo nome de uma marca conhecida (figura 27).

```
def select_query(product_dict):
    #função que define a query a ser executada na base de dados
    #a query pode ser feita pelo nome do produto caso seja uma pesquisa manual
    #pode ser feita pelo menu das categorias e sub categorias
    #os resultados são devolvidos em ordem ao nome ou ao preço sendo que pode ser ordenado de modo ascendente ou descendente
    query = 'SELECT uniqueID,itemName,itemPrice,priceType,itemCategory,itemSubCategory,imgPath,discountValue,storeID FROM products WHERE '
    boolSearch = False
    boolCategory = False

    if(product_dict['searchbar_keyword'] != ''):
        sub_stringSearch = 'itemName like \'%' + product_dict['searchbar_keyword'] + '%\''
        boolSearch = True

    if(product_dict['sub_category'] != ''):
        sub_stringCategory = 'itemSubCategory = \'' + product_dict['sub_category'] + '\''
        boolCategory = True

    elif(product_dict['category'] != ''):
        sub_stringCategory = 'itemCategory = \'' + product_dict['category'] + '\''
        boolCategory = True
    elif(product_dict['searchbar_keyword'] == ''): #caso o utilizador utilize a opção "Ver todos"
        sub_stringSearch = 'itemName like \'%%\''
        boolSearch = True


    if(boolSearch and boolCategory):
        finalQuery = query + sub_stringSearch + ' AND ' + sub_stringCategory
    elif(boolSearch):
        finalQuery = query + sub_stringSearch
    elif(boolCategory):
        finalQuery = query + sub_stringCategory

    finalQuery = finalQuery + 'ORDER BY '+product_dict['orderBy']+' '+ product_dict['clause']+ ';'
    print(finalQuery)
    return finalQuery
```

Figura 27 - Função no *back-end* que define que filtros utilizar para obter os produtos

Cada uma destas pesquisas é resultado de inquisições à base de dados de modo a não sobrecarregar a máquina do utilizador. Estes pedidos são respondidos com toda a informação necessária para o utilizador avaliar e comparar os produtos disponíveis, analisando os preços nas diversas lojas e descontos disponíveis (figura 28).

```python
def executeQuery(query):
    cursor_id = connection.cursor()
    #execução da query na base de dados dependendo dos filtros utilizados
    cursor_id.execute(query)
    rows_id = cursor_id.fetchall()

    data = {}
    data['totalSize'] = len(rows_id)
    data['products'] = {}
    query_resultSize = len(rows_id)
    found_products = {}
    total_products = 0
    lastAdded = 0
    for i in range(query_resultSize):
        temp_dict = {}

        if rows_id[i][1] not in found_products.values():

            temp_dict['id'] = []
            temp_dict['storeID'] = []
            temp_dict['preco'] = []
            temp_dict['desconto'] = []

            temp_dict['id'].append(rows_id[i][0])
            temp_dict['nome'], temp_dict['marca'] = treatBrand(rows_id[i][1]) #separação do campo itemName do produto em nome e marca
            temp_dict['preco'].append(rows_id[i][2])
            temp_dict['tipo_preco'] = rows_id[i][3]
            temp_dict['categoria'] = rows_id[i][4]
            temp_dict['sub_categoria'] = rows_id[i][5]
            temp_dict['path'] = rows_id[i][6]
            temp_dict['desconto'].append(rows_id[i][7])
            temp_dict['storeID'].append(rows_id[i][8])

            data['products']['prod' + str(lastAdded)] = temp_dict
            found_products['prod' + str(lastAdded)] = rows_id[i][1]
            lastAdded +=1
            total_products +=1
        else: #caso seja um produto repetido, mas de um supermercado diferente
            temp_nome,temp_marca = treatBrand(rows_id[i][1])
            existent_key = getKeyByValue(data,temp_nome,temp_marca)
            data['products'][existent_key]['id'].append(rows_id[i][0])
            data['products'][existent_key]['preco'].append(rows_id[i][2])
            data['products'][existent_key]['desconto'].append(rows_id[i][7])
            data['products'][existent_key]['storeID'].append(rows_id[i][8])

    data['totalSize'] = total_products
    return data
```

Figura 28 - Obtenção dos produtos e ordenação destes num dicionário que será devolvido ao *front-end*

11.2.4 Serviço *listService*

Por fim, foi criado o serviço *listService*, como foi descrito na fase de desenvolvimento este é o serviço mais complexo e extenso de todo o projeto, visto que é responsável pela grande maioria das funcionalidades que caracterizam a aplicação. Estas funcionalidades permitem ao utilizador uma personalização das suas listas pessoais, podendo atribuir nomes à sua escolha, eliminá-las, enviar convites a outros participantes de modo a que estes possam visualizar a lista, adicionar e remover produtos ou definir a quantidade destes e que supermercados a considerar para o cálculo do valor total da lista.

Antes da criação de uma nova lista é feita uma rápida verificação à integridade do nome escolhido, de modo a verificar se este contém caracteres inválidos, e, caso isto não se verifique, a lista é criada e o utilizador pode adicionar produtos e participantes a esta (figura 29).

```python
def api_createList(request):
    #informação recebida pelo front-end
    dict_data = json.loads(request.body.decode('utf-8'))
    #informação introduzida pelo utilizador
    listname = dict_data['listname']
    defaultLevel = dict_data['defaultLevel']
    #token do utilizador, guardada na cookie do seu bwoser
    userToken = request.COOKIES.get('token')
    #obtenção do identificador do utilizador a partir da token
    userID = tokenToUser(userToken)
    #query que verifica se o utilizador está em alguma lista
    #se o utilizador nao estiver em nenhuma lista, a lista a ser criada será definida como a sua lista ativa
    noLists = query_checkIfUserInAnyList(userID)

    if(createList(listname,userID,defaultLevel)):
        print('List created')
        if(not noLists):
            if(setDefaultListOnCreate(userID)):
                data = {'status' : 'success', 'updatedDefaultList' : True, 'defaultList' : query_getListByParticipant(userID)}
            else:
                data = {'status' : 'failed'}
        else:
            data = {'status' : 'success', 'updatedDefaultList' : False}
    else:
        data = {'status' : 'failed'}

    return JsonResponse(data)
```

Figura 29 - API responsável por adicionar à base de dados a nova lista criada pelo utilizador

Quando o utilizador acede à página que apresenta todas as suas listas, são feitas verificações que permitem ao *front-end* identificar quais são as listas criadas pelo utilizador, os participantes de todas as suas listas e qual destas é a sua lista ativa. Como foi descrito no *front-end* a adição de produtos pode ser feita através da funcionalidade "*fast-add"* ou através dos detalhes de cada produto. A funcionalidade "*fast-add"* é realizada através da verificação da variável "*activeList*", que identifica uma das listas do utilizador, como sendo a sua lista ativa. Esta é armazenada no próprio *browser* do utilizador [20] (figura 30), de modo a poder ser utilizada em diversas secções da aplicação. Esta variável existe sempre desde o momento em que o utilizador cria a sua conta, sendo que tem o valor "0" caso este não tenha nenhuma lista criada ou não seja participante na lista de outro utilizador. A variável "*activeList*" é também guardada na base de dados, de modo a ser possível manter a mesma entre sessões, sendo que esta é fornecida ao utilizador sempre que este realiza o *sign in*.

Key	Value
allLists	95,128
acceptedStores	2
activeList	95

Figura 30 - Variáveis guardadas no armazenamento local do *browser*

Com a sua lista criada, o utilizador pode agora convidar outros para visualizar a sua lista. Estes convites são executados a partir da verificação do número de telemóvel. Tal como foi feito na fase de registo, é verificada a integridade do número introduzido. Após esta verificação o número é recebido pelo serviço no *back-end* que por sua vez verifica se o número

de telemóvel está associado a algum utilizador e, caso esteja, cria um convite para este aceitar ou recusar (figura 31).

```python
def api_getParticipant(request):
    #informação recebida pelo front-end
    dict_data = json.loads(request.body.decode('utf-8'))
    #informação introduzida pelo utilizador
    temp_phone = dict_data['phoneNum']
    listID = dict_data['listID']
    #obtenção do identificador do utilizador a partir da cookie
    userToken = request.COOKIES.get('token')
    selfID = tokenToUser(userToken)
    data={}

    #query que obtém o identificador do utilizador que vai receber o convite
    userID = query_getParticipant(temp_phone)

    if(selfID != userID):
        if(not userID):
            data['status'] = 300
            data['message'] = 'Phone number does not exist '

        elif(userID):
            data['status'] = 200
            data['message'] = 'User found'
            #query que verifica se o utilizador a ser convidada já é participante da lista
            if(not query_checkIfAlreadyParticipant(userID,listID)):
                #query que verifica se o convite jáe xiste
                if(query_checkDupeInvite(userID,listID)):
                    #query que adiciona o convite
                    if(query_addInvite(userID,listID)):
                        data['status'] = 200
                        data['message'] = 'Invite added'

                    else:
                        data['status'] = 400
                        data['message'] = 'Could not create invite'
                else:
                    data['status'] = 200
                    data['message'] = 'Invite already exists'

            else:
                data['status'] = 600
                data['message'] = 'User already in list as participant'
    else:
        data['status'] = 500
        data['message'] = 'Sending invite to self'

    return JsonResponse(data)
```

Figura 31 - API responsável pela receção de novos convites

Os convites podem ser visualizados na zona das definições pessoais do utilizador. Caso o utilizador decida aceitar o convite, este é adicionado aos participantes da lista e pode agora aceder a esta. Se o utilizador recusar o convite, este é indicado como recusado na base de dados (figura 32).

```python
def api_replyToUserInvites(request):
    #informação recebida pelo front-end
    dict_data = json.loads(request.body.decode('utf-8'))
    #obtenção do identificador do utilizador a partir da token guardada nacookie
    userToken = request.COOKIES.get('token')
    userID = tokenToUser(userToken)
    #resposta do utilizador ao convite para se juntar a uma lista
    inviteReply = dict_data['reply']
    listID = dict_data['listID']
    data = {}

    #se o utilizador aceitar o convite, é enviado um True
    #caso este recuse é enviado um False
    if(inviteReply):
        #query que adiciona o utilizador à lista
        if(query_updateParticipants(userID,listID)):
            #query que atualiza o estado do pedido para 'Accepted' ou para 'Refused'
            if(query_updateStatus(True,userID,listID)):
                data['status'] = 200
                data['message'] = 'Invite status now accepted'
            else:
                data['status'] = 400
                data['message'] = 'Could not update status'
        else:
            data['status'] = 400
            data['message'] = 'Could not add new participant'

    elif(not inviteReply):

        if(query_updateStatus(False,userID,listID)):
            data['status'] = 200
            data['message'] = 'Invite status now refused'
        else:
            data['status'] = 400
            data['message'] = 'Could not update status'

    return JsonResponse(data)
```

Figura 32 - API responsável por tratar a resposta ao convite para uma lista

É possível inclusive remover participantes da lista no entanto, não é possível remover o criador desta (figura 33).

```python
def api_removeUserFromList(request):
    #informação recebida pelo front-end
    dict_data = json.loads(request.body.decode('utf-8'))
    #informação que determina de que lista remover o utilizador
    listID = dict_data['listID']
    userID = dict_data['userID']
    data={}

    #query que verifica se o utilizador a ser removido é o dono da lista
    if(not query_checkIfRemovingOwner(listID,userID)):
        #query que remove o utilizador da lista
        if(query_removeUserFromList(listID,userID)):
            data['status'] = 200
            data['message'] = 'Successfully removed from list'
        else:
            data['status'] = 400
            data['message'] = 'Could not leave list'

    elif(query_checkIfRemovingOwner(listID,userID)):
        data['status'] = 300
        data['message'] = 'Trying to delete owner of list'
    else:
        data['status'] = 400
        data['message'] = 'Error checking owner'


    return JsonResponse(data)
```

Figura 33 - API responsável pela remoção de um utilizador

Quando o utilizador pretende eliminar uma lista da sua aplicação é primeiro feita uma verificação que identifica se o utilizador é o criador da lista. Caso isto se verifique a lista é eliminada da aplicação de todos os que a podem ver e editar (figura 34). Caso o utilizador não seja o criador da lista, este apenas consegue remover-se a si próprio (figura 35).

```python
def api_deleteList(request):
    #informação recebida do front-end
    dict_data = json.loads(request.body.decode('utf-8'))
    listID = dict_data['listID']
    data = {}

    #query que elimina a lista
    if(query_deleteListByID(listID)):
        #query que verifica se existem utilizadores com a lista ativa que foi eliminada
        #todos estes utilizadores vão ter a sua defaultList mudada para 0
        if(query_updateDefaultListOnDelete(listID)):
            data['status'] = 200
            data['message'] = 'List successfully deleted and defaultLists changed to 0'
        else:
            data['status'] = 400
            data['message'] = 'Could not delete list'
    else:
        data['status'] = 400
        data['message'] = 'Could not delete list'

    return JsonResponse(data)
```

Figura 34 - API que elimina uma lista caso a pessoa seja o criador dela

```python
def api_leaveList(request):
    #informação recebida pelo front-end
    dict_data = json.loads(request.body.decode('utf-8'))
    listID = dict_data['listID']
    userToken = request.COOKIES.get('token')
    userID = tokenToUser(userToken)
    data = {}

    #query que retira o utilizador dos participantes da lista
    if(query_leaveList(listID,userID)):
        data['status'] = 200
        data['message'] = 'Successfully left list'
    else:
        data['status'] = 400
        data['message'] = 'Could not leave list'

    return JsonResponse(data)
```

Figura 35 - API que remove um utilizador convidado para uma lista

Após a adição de produtos à lista, os participantes desta podem aumentar ou diminuir a quantidade de cada produto individual através de um simples botão, é também possível remover o produto inteiramente da lista. Todos estes pedidos são tratados no *back-end*, que faz o pedido à base de dados para atualizar o conteúdo da lista consoante os pedidos dos utilizadores (figura 36).

```python
def api_addProducts(request):
    #informação recebida pelo front-end
    dict_data = json.loads(request.body.decode('utf-8'))
    #informação que determina que lista e produto atualizar
    listID = dict_data['listID']
    productID = dict_data['productID']
    #variável que determina se vai ser adicionada ou removida quantidade
    tag = dict_data['tag']
    productQuantity = 1
    userToken = request.COOKIES.get('token')
    addedBy = tokenToUser(userToken)

    #query que verifica o produto já está na lista
    if(not query_verifyIfProductInList(listID, productID)):
        #query que adiciona o produto à lista
        if(addProduct(listID,productID,productQuantity,addedBy)):
            data = {'status' : 'success'}
        else:
            data = {'status' : 'failed'}
    #se o produto já existir na lista a variável tag determina se é adicionada ou removida quantidade
    else:
        if(tag):
            if(addQuantityToProduct(listID, productID)):
                data = {'status' : 'success'}
            else:
                data = {'status' : 'failed'}
        else:
            if(removeQuantityFromProduct(listID, productID)):
                data = {'status' : 'success'}
            else:
                data = {'status' : 'failed'}


    return JsonResponse(data)
```

Figura 36 - API responsável por adicionar produtos à lista, ou atualizar a quantidade desta

Quando o utilizador adiciona um novo produto, altera a quantidade deste ou remove-o inteiramente, o valor total da lista é recalculado. Este cálculo avalia o preço de cada produto nos diversos supermercados e faz o somatório destes valores para cada uma das superfícies comerciais. O resultado destas somas é apresentado ao utilizador, separado por supermercado, junto de um pequeno texto que indica qual a melhor opção para o utilizador. No caso de o utilizador escolher produtos que não existem em todas as lojas, irá também aparecer uma secção que indica que supermercados têm produtos em falta e quais são. Por último, o utilizador tem a possibilidade de alterar que supermercados quer permitir para o cálculo do valor total. Estas preferências são armazenadas no próprio *browser* de modo a que o utilizador não tenha de as mudar sempre que volta a entrar na página da lista.

11.2.5 Máquinas Virtuais

Com os diversos micro-serviços criados, para a sua execução foram criadas diversas máquinas virtuais, que comunicam entre si através de uma rede privada. Esta arquitetura permite uma melhor organização de cada micro-serviço e a otimização dos recursos de computação necessários para cada um destes micro-serviços.

Foi criada uma máquina virtual que serve como base para todas as restantes, sendo que esta contém o sistema operativo *Ubuntu* (versão 14.04) e o *software* referente à tecnologia *Docker*. Com a máquina virtual base criada, esta foi replicada seis vezes, onde quatro *VMs* são alocadas para os quatro micro-serviços e as restantes duas são utilizadas para conter a base de dados e o servidor *Nginx*. Este processo foi facilitado devido ao uso do formato *OVF*, utilizado para a criação da primeira *VM*.

O servidor *Nginx* tem o papel de redirecionar pedidos *web* à aplicação, de modo a que seja possível a comunicação com as *VMs* de cada micro-serviço. Para tal é analisado o *subdomain* e o caminho do pedido recebido e é escolhido o servidor correto para tratar o pedido.

Para cada micro-serviço, foi criado um ficheiro, denominado de "requirements.txt", onde são guardadas as *packages Python* necessárias para o funcionamento de cada um. Este ficheiro será utilizado no início da execução de cada *Docker image*, de modo a rapidamente instalar as *packages* mencionadas. Foi também criado o ficheiro *Dockerfile* onde é configurada a *Docker image*, definindo a *docker image* base, neste caso *Python 3.6*, o diretório a ser executado e o comando de execução. Após este processo é criada a *docker image* no sistema de ficheiros local. Com a *docker image* criada, é feita uma ligação entre esta e o serviço de sistema de ficheiros *online*, providenciado pelo serviço *Docker*. Seguidamente, esta é enviada para o repositório referido anteriormente. Com este processo concluído, é possível aceder ao repositório e executar a *docker image* em qualquer *VM*.

11.3 Fase de Testes

Durante todo o desenvolvimento do projeto, todas as funcionalidades da aplicação foram constantemente avaliadas e melhoradas, testando ao pormenor cada uma destas. Para facilitar todo este processo, foi utilizado os serviços do *PythonAnywhere* [21]. Graças a este *website*, foi possível ter um ambiente para executar e testar todas as funcionalidades do projeto, com maior ênfase no comportamento do *back-end* uma vez que este funciona à base de pedidos pela internet.

Para o testo do funcionamento dos micro-serviços quando executadas nas máquinas virtuais, foi criada uma máquina virtual adicional com o sistema operativo *Ubuntu* de modo a ser possível, através de um *browser*, aceder à rede interna na qual se encontram os micro-serviços e testar todas as suas funcionalidades.

12 Base de dados

A base de dados, criada em *MySQL*, desempenha o papel de guardar e fornecer dados à aplicação, de maneira rápida e eficiente, para o uso correto desta. A nossa base de dados agrupa os seus dados em tabelas, estando estas ligadas entre si através de chaves. A ligação da base de dados ao *back-end*, que por sua vez processa os dados e os apresenta ao *front-end*, é feita com base em *views*. Este método de procura e recolha de informação é particularmente eficiente com o uso de *primary keys* (PK) e *foreign keys* (FK). Estas interligam as tabelas, facilitando o processo de pesquisa, que por sua vez aumenta a eficiência e rapidez da aplicação.

12.1 Planeamento

Para o planeamento da base de dados e a criação do seu UML, recorrer-se-á à ferramenta *online* dbdiagram.io [22] (figura 37). Com esta é possível criar uma representação visual da base de dados, o que irá facilitar o desenvolvimento desta.

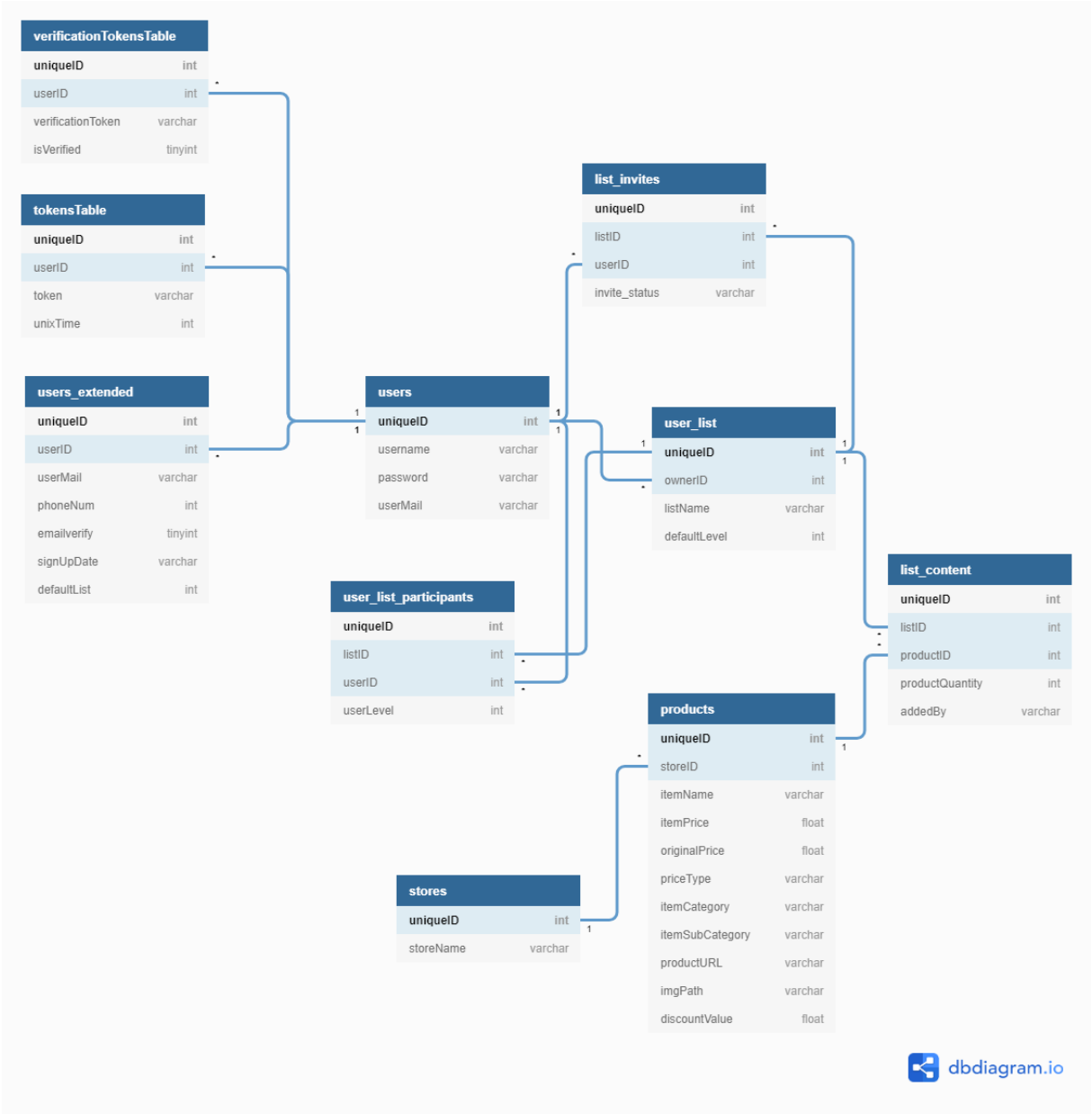


Figura 37- UML da base de dados

O desenho da base de dados vai basear-se em três grandes secções: os utilizadores, os produtos e as listas. Devido à natureza da aplicação, as ligações entre tabelas terão de ser estruturadas por forma a diminuir a complexidade das *queries* necessárias, criando uma rede modularizada com ênfase na eficiência.

Na secção dos utilizadores, uma tabela será criada para conter informações referentes aos utilizadores e os seus registos. A secção dos produtos irá servir para guardar informações referentes aos produtos específicos, como nome, preço, a que loja pertence, entre outras. E por fim, iremos ter ainda uma tabela na secção das listas, que irá conter informações sobre os conteúdos de cada lista, como o produto adicionado e quem o adicionou, assim como

informações sobre a lista em si, como o nome, o utilizador que a criou e o nível predefinido de acesso.

12.2 Desenvolvimento

A base de dados foi criada seguindo a estrutura delineada durante a fase de planeamento, para facilidade de acesso aos dados e possível extensibilidade. A sua ligação com o *back-end*, por intermédio das *views*, desempenha um papel fundamental no acesso aos dados.

Todas as tabelas possuem um atributo *uniqueID*, usado como chave primária para distinguir os tuplos entre si. Este atributo é do tipo *int*, e é atualizado com cada nova entrada: sempre que é feita uma nova adição às tabelas, o valor do *uniqueID* é declarado como nulo, é verificado o valor atual do parâmetro *AUTO_INCREMENT* e este é utilizado para atribuir o novo *uniqueID*. Após este processo o valor do parâmetro *AUTO_INCREMENT* é incrementado por um.

Tabela 1 – Tabela *stores*

stores		
PK	uniqueID	Int
	storeName	Varchar

A tabela *stores* (tabela 1) foi criada para conter informação sobre os três tipos de supermercados. Esta contém o "*storeName*", que representa o nome do supermercado (Pingo Doce, Continente ou Jumbo).

Tabela 2 – Tabela *products*

products		
PK	uniqueID	Int
FK	storeID	Int
	itemName	Varchar
	itemPrice	Float
	originalPrice	Float
	priceType	Varchar
	itemCategory	Varchar
	itemSubCategory	Varchar
	productURL	Varchar

	imgPath	Varchar
	discountValue	Float

A tabela *products* (tabela 2) está interligada à tabela "*stores*" através da *foreign key* "*storeID*". Esta tabela serve para demonstrar as funcionalidades da aplicação, apesar dos produtos selecionados não representarem toda a oferta de mercado. Nela estão contidas informações sobre cada produto, incluindo o nome ("*itemName*"), o preço original ("*originalPrice*"), o preço atual ("*itemPrice*"), o valor referente ao desconto caso exista – ("*discountValue*"), a categoria ("*itemCategory*"), o *URL* para a página de cada produto no *site* do respetivo supermercado ("*productURL*"), o caminho para a sua imagem no sistema de ficheiros local ("*imgPath*"), e o identificador do supermercado a que pertence ("*storeID*").

Se o produto estiver em desconto, o valor atual é calculado através da multiplicação do preço original (preço definido como base do produto) pelo respetivo desconto (valor definido para cada um dos supermercados).

Tabela 3 – Tabela *list_content*

list_content		
PK	uniqueID	Int
FK	listID	Int
FK	productID	Int
	productQuantity	Int
	addedBy	Varchar

A tabela *list_content* (tabela 3), através do "*productID*", está interligada à tabela *products*. Esta foi criada para conter informações dos produtos adicionados a cada uma das listas. Para isso, possui um identificador único, "*listID*", que identifica cada uma das listas; um "*productID*", que guarda o identificador de cada produto; um "*productQuantity*", que guarda a quantidade de cada produto na lista e um "*addedBy*", que representa o utilizador que adicionou esse produto.

Tabela 4 – Tabela *user_list*

user_list		
PK	uniqueID	Int
FK	ownerID	Int
	listName	Varchar
	defaultLevel	Int

A tabela *user_list* (tabela 4) encontra-se ligada à tabela *list_content* e tem como objetivo guardar todas as informações específicas de cada uma das listas. O atributo "*ownerID*" identifica o utilizador que criou a lista. O "*listName"* guarda o nome customizado de cada uma das listas O "*defaultLevel*" especifica o nível padrão de acesso à lista, que pode ser 0 (*read only* ou ver apenas), 1 (adicionar sugestões) ou 2 (controlo dos produtos).

Tabela 5 – Tabela *list_invites*

list_invites		
PK	uniqueID	Int
FK	listID	Int
FK	userID	Int
	invite_status	Varchar

A tabela *list_invites* (tabela 5), ligada à tabela *user_list*, guarda dados adicionais sobre as listas, mais especificamente dados relacionados com os convites enviados aos utilizadores para participarem na lista. O atributo "*listID*", que serve de *foreign key* para ligar à tabela *user_list*, identifica cada uma das listas; o "*userID*" identifica cada um dos utilizadores, servindo também de *foreign key* para ligar a *list_invites* à tabela *users*; o "*invite_status*" representa o estado de cada convite.

Tabela 6 – Tabela *user_list_participants*

user_list_participants		
PK	uniqueID	Int
FK	listID	Int
FK	userID	Int
	userLevel	Int

A tabela *user_list_participants* (tabela 6), também ligada às tabelas *user_list* e *users*, é utilizada para manter informações sobre as listas que estão relacionados com os seus participantes. O atributo "*listID*" retém o identificador de cada lista; o "*userID*" corresponde à identificação de cada utilizador; e o "*userLevel*" guarda o nível de acesso que cada utilizador possui na respetiva lista.

Tabela 7 – Tabela *users*

users		
PK	uniqueID	Int
	username	Varchar
	password	Varchar
	userMail	Varchar

A tabela *users* (tabela 7) foi criada para guardar os dados dos utilizadores relacionados com os registos e entradas. O atributo "*username*" guarda os nomes personalizados de cada utilizador; "*password*" guarda a palavra-passe associada a cada conta, necessária para a entrada na aplicação; e o "*userMail*" contém o endereço de *e-mail* associado a cada conta de utilizador, usado em conjunto com a palavra-passe para dar entrada na aplicação.

Tabela 8 – Tabela *verificationTokensTable*

verificationTokensTable		
PK	uniqueID	Int
FK	userID	Int
	verificationToken	Vachar
	isVerified	Tinyint

A tabela *verificationTokensTable* (tabela 8) foi criada para conter informações relacionadas com os *tokens* usados para verificar os registos/entradas dos utilizadores. O atributo "*userID*" identifica cada utilizador e serve de *foreign key* na ligação com a tabela *users*; a *verificationToken* guarda o valor aleatório de cada *token* atribuído; e o atributo "*isVerified*" contém os valores "1" ou "0", que indicam se o utilizador se encontra verificado ou não, respetivamente.

Tabela 9 – Tabela *users-extended*

users_extended		
PK	uniqueID	Int
FK	userID	Int
	userMail	Varchar
	phoneNum	Int
	emailVerify	Tinyint
	signUpDate	Varchar
	defaultList	int

A tabela *users_extended* (tabela 9) contém os dados pessoais dos utilizadores. O atributo "*userID*" identifica cada utilizador e serve também de *foreign key* na ligação com a tabela *users*; "*userMail*" contém os endereços de correio eletrónico correspondentes a cada conta de utilizador; "*phoneNum*" guarda os números de telemóvel de cada utilizador; "*emailVerify*" representa o estado de verificação da conta do utilizador, e contém um valor de "1" ou "0", indicando se o utilizador se encontra verificado ou não, respetivamente; "*signUpDate*" especifica a data de registo de cada utilizador; e a "*defaultList*" guarda o valor da lista considerada ativa para cada utilizador.

12.3 População da base de dados

A população de praticamente toda a base de dados é feita de forma automática e de acordo com as necessidades dos utilizadores, como por exemplo: registo de novos *users*, criação de listas, convites a outros utilizadores, etc. Os únicos dados inseridos manualmente por parte

do administrador passam pela adição de novos supermercados e, principalmente, de novos produtos.

Os produtos inseridos na base de dados são fruto de uma recolha nos vários sites dos supermercados selecionados e de uma posterior comparação dos preços entre estes. Estes foram adicionados através de um *script* que permitiu a inserção de todos os produtos e das informações referentes a estes de uma só vez, facilitando a inserção, bem como testando a capacidade da base de dados para receber grandes quantidades de dados.

13 Conclusão

O projeto e todas as ideias inicialmente planeadas foram, de um modo geral, conseguidas e desenvolvidas com sucesso. Nas diversas fases de planeamento dos tópicos acima mencionados, muitas foram as ideias e alterações feitas que foram surgindo à medida que o projeto foi desenvolvido. Todas as ideias inicialmente propostas para o produto final estão na aplicação, tendo sido acrescentadas várias funcionalidades que resultam numa *web app* 100% funcional, intuitiva e simples.

Todas as fases do projeto foram encaradas pelo grupo com muita responsabilidade, procurando sempre encontrar um equilíbrio entre o que tinha sido proposto e o resultado de todo esse planeamento e idealização.

Relativamente ao *front-end*, muitas foram as mudanças e alterações feitas ao longo de todo o projeto, devido principalmente à linha de pensamento traçada para o projeto e ao equilíbrio entre as diversas funcionalidades do *back-end*. O *design* final de todas as páginas e o ambiente da aplicação são aspetos que merecem destaque, pois, inicialmente, foram concebidos para serem simples e assim ficaram no fim do projeto. No entanto, existiram vários percalços no design gráfico da aplicação, como a falta de experiência no design de uma aplicação direcionada para universo móvel, mas também a adaptação que tem de existir para uma transição fluída entre os vários dispositivos, sem nunca se perder a essência caraterística da Shapplist. Através de uma extensa pesquisa, utilizando *apps* semelhantes e documentação na área, foi possível tornar esta aplicação adaptável a todos os ambientes de visualização, tendo como ideias principais a simplicidade aliada a uma ferramenta que permitisse ao utilizador "largar o papel e a caneta", utilizando-a para fazer as suas pesquisas de produtos e criação de listas de compras.

Em relação ao *back-end*, a tecnologia utilizada para a construção dos diversos micro-serviços foi alterada. Originalmente seria utilizado .NET Core [23] para construir as funcionalidades do *back-end*, no entanto, devido a constrições de tempo, o grupo sentiu que não seria favorável o tempo despendido para a aprendizagem desta nova tecnologia descartando esta opção em favor da *framework* Django, com a qual já existia bastante familiaridade. Com a *framework* escolhida, as dificuldades encaradas no desenvolvimento do *back-end* deveram-se essencialmente a encontrar as soluções ideais para realizar a troca de informação entre os diversos micro-serviços e as páginas *HTML* que iriam utilizar estes dados. Para resolver este problema, experimentou-se a combinação da tecnologia *AJAX* e o formato *JSON*. Esta solução provou ser um sucesso, tornando-se um dos pilares para o desenvolvimento de todo o projeto. Tendo todas as tecnologias escolhidas, o desenvolvimento do *back-end* e todos os seus micro-serviços teve um percurso bem definido, sendo que os únicos obstáculos encontrados posteriormente estariam relacionados com a otimização da lógica de cada uma das funcionalidades. A construção da rede de máquinas virtuais, na qual são executados os diversos micro-serviços, também foi alcançada com sucesso. O formato *OVF* forneceu uma opção ideal para a criação de máquinas virtuais, pois elimina a necessidade de configurar um sistema operativo de raiz sempre que se pretendia criar uma nova márquina virtual. A utilização da tecnologia *Docker* facilitou o processo de replicação das *VMs*, uma vez que permite uma automatização da instalação das dependências necessárias e a execução do micro-serviço em questão. Esta permite também uma melhor organização devido à tecnologia *container*.

14 Trabalho futuro

Relativamente ao trabalho futuro, existem ainda muitas funcionalidades e alterações que podem ser feitas na aplicação e em tudo o que a envolve. Muitas foram as ideias que foram surgindo, no entanto, por um ou mais motivos acabaram por não fazer parte do produto final. Acima de tudo, e de acordo com aquilo que já foi referido antes, a aplicação visa principalmente ser uma ferramenta simples e intuitiva, pelo que muitas funcionalidades poderiam torná-la demasiado complexa para a maioria dos utilizadores.

Na área do *front-end*, a maior parte do *design* gráfico e das funcionalidades foi conseguido na aplicação. Os menus, os vários *overlays* e o ambiente envolvente são pontos bastantes bem conseguidos, salvo algumas exceções. Algumas das funcionalidades pensadas para adicionar à aplicação são: introdução de mais formas de calcular o valor final da lista, de

acordo com exceções que podem ser definidas pelos utilizadores; adicionar um sistema de GPS que possibilitasse ao utilizador perceber qual dos supermercados se encontra mais perto de si; um sistema de leitura de códigos de barras para inserção rápida dos produtos nas listas; criação de um sistema de permissões/níveis que iria permitir ao criador de uma lista definir os comportamentos dos participantes nesta, como restringir a adição ou remoção de produtos.

A página individual das listas é uma das páginas onde existem mais ideias para a tornar uma ferramenta mais personalizável por parte do utilizador, visto ser a página com a maior margem para progressão e desenvolvimento. Para isso, foi definido que no futuro poderia ser adicionado um sistema de exceções que permitisse aos utilizadores presentes na lista criar diversas limitações na mesma. Uma das limitações idealizadas foi a de especificar, de entre os produtos, quais correspondem às necessidades de um utilizador específico, ou seja, definir que um dos artigos é um pedido específico de um dos utilizadores e que no valor final da lista, caso necessário, o valor referente a esse artigo seria mostrado à parte para destacar quanto teria o utilizador que o pediu pagar. Outra funcionalidade seria a introdução de uma função, provavelmente em *MATLAB* [24], que fizesse automaticamente o cálculo final do valor total da lista com base em limitações definidas pelo utilizador, como, por exemplo, através da definição de um número específico de supermercados, não especificando quais, ou ainda definir que um produto específico só possa ser contabilizado a partir de um único supermercado. Existiram ainda outras ideias, como a criação de uma zona de partilha de mensagens escritas entre os utilizadores ou notas sobre as listas, que, no entanto, apesar de colmatar a necessidade um certo número de utilizadores, foram definidas como ideias que não se encontravam na linha de pensamento traçada para a aplicação.

Em relação ao *back-end* existem várias otimizações que podem ser feitas, mas que não foram encaradas como prioridade para a funcionalidade da aplicação e, devido a restrições de tempo, não foi possível a sua implementação. Uma das que se pode considerar como crucial seria a proteção da integridade da informação armazenada na base de dados, em particular os dados pessoais dos utilizadores. A solução para este problema passaria pela encriptação da informação de cada utilizador, tendo especial atenção aos campos "*e-mail*" e "*password*". Outra otimização passaria pela implementação de uma opção de recuperação da palavra-passe, caso o utilizador se esqueça da sua, onde seria enviado um e-mail com um *hyperlink* a partir do qual poderia ser possível definir uma nova palavra-passe. A implementação de *two factor authentication* (2FA) [25] iria aumentar ainda mais o nível de segurança da conta do utilizador, sendo que para alterar as suas informações pessoais, o utilizador iria de ter de introduzir um código aleatório enviado para o seu número de telemóvel. Caso o utilizador não consiga ter

acesso ao seu *e-mail*, a utilização de 2FA e o envio do código aleatório poderiam garantir o acesso à zona de criação de uma nova palavra-passe. A própria criação da palavras-passe também poderia ser melhorada, obrigando o utilizador a escolher uma que respeitasse um conjunto de regras que a tornasse mais segura, como a utilização de números e caracteres especiais.

Todas as funcionalidades dependentes de código *JavaScript* poderiam ser otimizadas de modo a diminuir a probabilidade de erros. Atualmente, a granda maioria destas são guardadas nas próprias páginas *HTML*. Isto é problemático devido ao modo como o *browser* executa estas páginas. O processamento das páginas *HTML* começa pela execução do código *HTML*, o componente visual da página, e só após este é analisado e guardado o código *JavaScript* de modo a poder ser executado. Caso o utilizador tente usar uma funcionalidade da aplicação antes do código *JavaScript* ser processado, esta não será executada ou pode devolver um erro. A solução deste problema, passaria por remover todo o código *JavaScript* das páginas *HTML* e guardá-lo em ficheiros separados no *back-end*, de modo a ser o primeiro componente a analisar quando uma página *HTML* é fornecida ao utilizador.

Por fim, seria necessário definir uma forma normalizada para introduzir novos produtos na base de dados e todas as informações associadas a estes. Com isto, as duas opções possíveis seriam: a criação de uma *API* por parte do grupo que permitisse aos diversos supermercados adicionar e remover produtos, permitindo também definir descontos para estes; ou a utilização de uma *API* disponibilizada pelos diversos supermercados, que permitisse atualizar a base de dados da aplicação ShappList sempre que estes façam alterações nos seus produtos.

15 Bibliografia

[1] – https://mercadao.pt/store/pingo-doce/search?page=0 - acesso em 19 Mar. 2019;

[2] – https://www.continente.pt/pt-pt/public/Pages/homepage.aspx - acesso em 20 Mar. 2019;

[3] – https://www.jumbo.pt/Frontoffice/ - acesso em 20 Mar. 2019;

[4] – https://bootstrapious.com/p/bootstrap-sidebar - acesso em 21 Mar. 2019;

[5] – https://www.djangoproject.com/ - acesso em 25 Mar. 2019;

[6] – https://developer.android.com/studio - acesso em 30 Mar 2019;

[7] – https://docs.djangoproject.com/en/2.2/ - acesso em 29 Mar 2019;

[8] – https://microservices.io/ - acesso em 29 Mar. 2019;

[9] – https://www.techopedia.com/definition/4518/open-virtualization-format-ovf - acesso em 3 Abr. 2019;

[10] – https://docs.docker.com/get-started/ - acesso em 4 Abr. 2019;

[11] –http://nginx.org/en/docs/http/ngx_http_proxy_module.html #proxy_cookie_domain - acesso em 7 Abri. 2019;

[12] – https://www.w3schools.com/js/js_ajax_intro.asp - acesso em 5 Mai. 2019;

[13] – https://www.w3schools.com/tags/ref_httpmethods.asp - acesso em 5 Mai. 2019;

[14] – https://www.w3schools.com/js/js_json_intro.asp - acesso em 5 Mai. 2019;

[15] – https://www.digitalocean.com/community/tutorials/how-to-use-web-apis-in-python-3 - acesso em 6 Mai. 2019;

[16] – https://www.w3schools.com/js/js_cookies.asp - acesso em 7 Mai. 2019;

[17] – https://www.unixtimestamp.com/ - acesso em 9 Mai. 2019;

[18] – https://docs.djangoproject.com/en/2.2/intro/tutorial03/ - acesso em 3 Mai. 2019;

[19] – https://pt.wikipedia.org/wiki/N%C3%BAmeros_de_telefone_em_Portugal – acesso em 28 Mai. 2019;

[20] – https://www.w3schools.com/jsref/prop_win_localstorage.asp - acesso em 3 Jun. 2019;

[21] – https://help.pythonanywhere.com/pages/FollowingTheDjangoTutorial - 8 Abr. 2019;

[22] – https://dbdiagram.io/d – acesso em 22 Mar. 2019

[23] – https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-2.2&tabs=visual-studio – acesso em 22 Mar. 2019;

[24] – https://www.mathworks.com/?s_tid=gn_logo – acesso em 5 Jun. 2019;

[25] – https://searchsecurity.techtarget.com/definition/two-factor-authentication - acesso em 15 Mai. 2019;

16 Apêndices

UNIVERSIDADE AUTÓNOMA DE LISBOA
LUÍS DE CAMÕES

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

ShappList

Manual de Utilizador

Autores: David Santos Cordeiro Teixeira Pinto, Henrique Filipe Chuva, Pedro Jorge de Almeida Correia, Ricardo Filipe Vicente Machado

Orientador: Daniel Silvestre

Número dos candidatos: 20160740, 20160715, 20160536, 20160746

Julho de 2019

Lisboa

(Página em Branco)

17 Índice

18 Lista de Fotografias/Ilustrações

19 Introdução

O manual de utilizador da aplicação consiste num documento que os utilizadores podem consultar para tornar a experiência de usar a aplicação ShappList de uma forma mais fluída e simples. A aplicação foi desenhada e desenvolvida com um principal objetivo em mente: ser fácil de utilizar e intuitiva para que a use como uma ferramenta do seu dia-a-dia. Todas as funcionalidades nela presente, foram idealizadas para facilitar a experiência de visualizar os produtos a comprar de uma forma rápida e eficaz, podendo adicioná-los às listas criadas, apenas através de um clique.

O *slogan* "A nova forma de fazer compras" demonstra o foco principal durante todo o processo de criação da aplicação. A ideia de colmatar a necessidade existente em ter um app que fizesse os *users* "largar o papel e a caneta" foi algo traçado desde o primeiro dia.

A aplicação *ShappList*, tal como referido anteriormente, baseia-se nos princípios da simplicidade e facilidade de compreender como tudo funciona. Para tal, esta foi criada com a particularidade de se adaptar consoante o dispositivo que o utilizador estiver a usar, ou seja, a aplicação reconhece se está a ser exibida num telemóvel, tablet ou no computador. Existem então, dois modos principais de visualizar a app: modo computador e modo telemóvel. A seguir serão explicados como funciona cada um destes e como o utilizador pode usufruir de todas as funcionalidades presentes na aplicação.

20 Web app ShappList

20.1 Página introdutória da aplicação ShappList

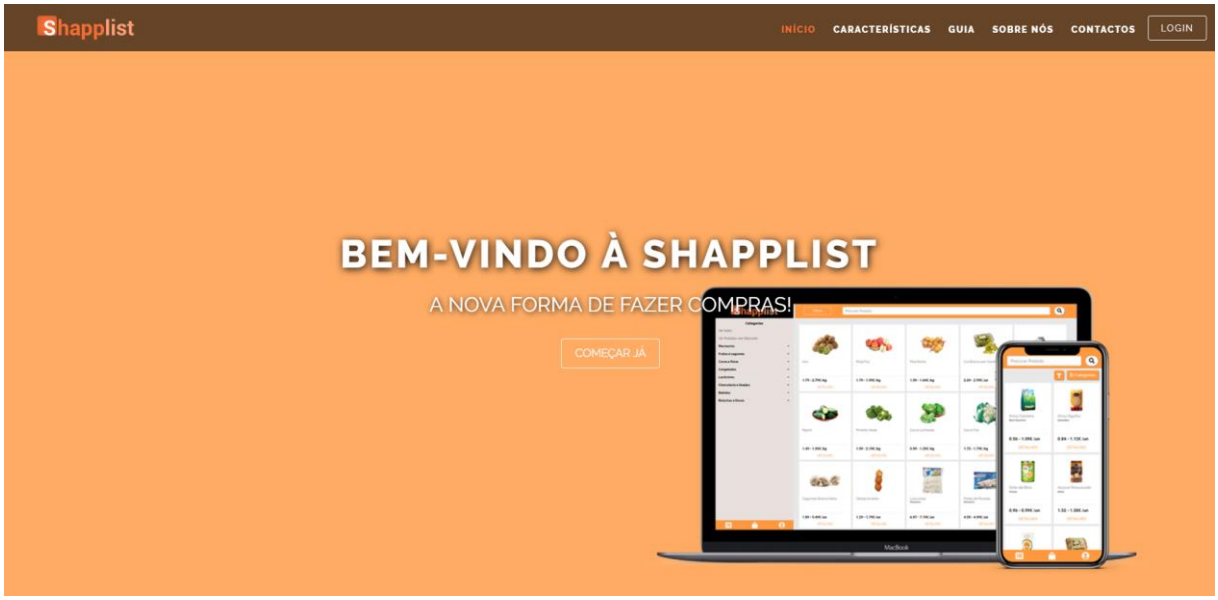


Figura 38 – Página de introdução da aplicação – *landing page*

Na página de introdução da aplicação, é apresentado ao utilizador uma amostra de algumas das funcionalidades e de informações referentes a esta. Esta página é meramente informativa, no entanto o utilizador pode começar a perceber melhor como tudo funciona, retendo também um dos principais focos da aplicação – a adaptabilidade entre dispositivos – através da imagem introdutória à aplicação.

Figura 39 – Barra de navegação da área pública da aplicação

A barra de navegação permite ao utilizador percorrer toda a página introdutória de uma forma rápida e intuitiva. Esta foi adaptada para o *user* nunca sair da mesma página, tendo apenas

de percorrê-la para saber toda a informação relativa à aplicação. Aqui o utilizador tem acesso a um resumo das caraterísticas da web app; a um pequeno e rápido guia de como usar a aplicação; algumas informações sobre a criação da aplicação e uma zona de contactos. É possível, a partir desta barra de navegação, aceder às áreas de Login (entrada na zona pessoal da aplicação) e Registo (criação de uma nova conta de utilizador).

Figura 40 – Página introdutória da aplicação – início – versão de telemóvel

Figura 41 – Página introdutória da aplicação - caraterísticas – versão de telemóvel

No modo telemóvel, a grande diferença é que a barra de navegação está condensada no botão no topo do ecrã. Quando clicado, todas as opções anteriormente faladas são demonstradas ao utilizador, para que este as use como forma de percorrer a página de uma forma mais rápida.

20.2 Página de entrada para utilizadores existentes – login

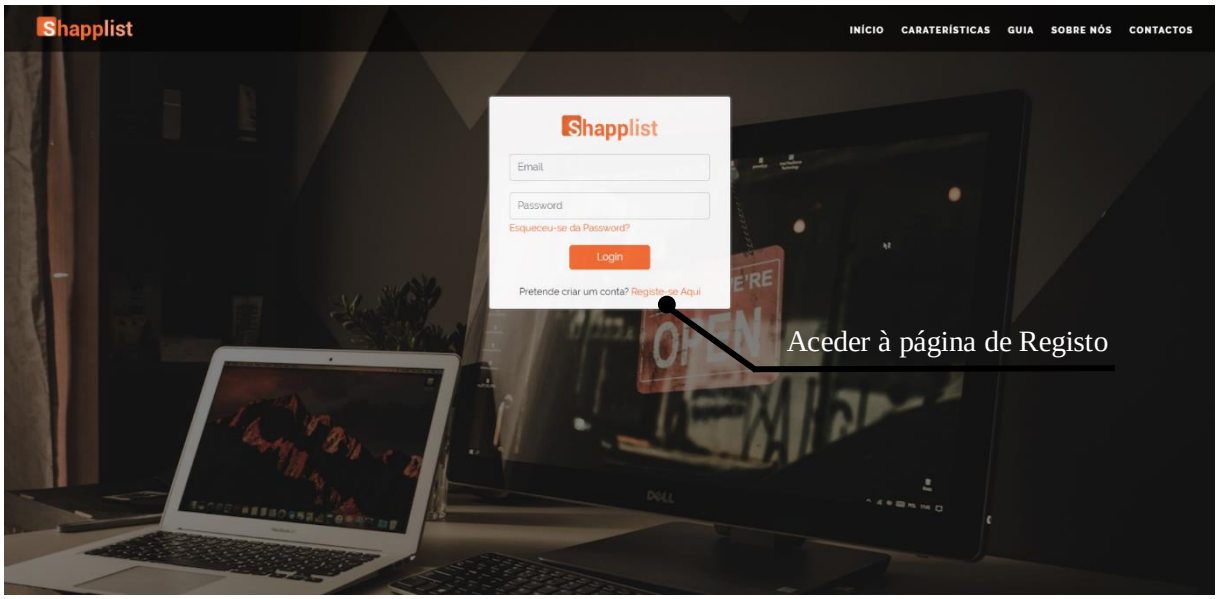


Figura 42 – Página de entrada para utilizadores existentes – *login*

Os utilizadores já registados podem, a partir da página login, dar entrada na sua zona pessoal na aplicação. Esta é feita através na introdução do email inserido a quando do registo e a palavra-passe correspondente. Caso as credenciais providenciadas forem as corretas, o utilizador será reencaminhado para a página dos produtos, dentro da sua área pessoal. Por outro lado, se o email e/ou a palavra-passe estiverem incorretos, uma mensagem de erro será demonstrada ao utilizador, informando-o desse incompatibilidade entre os dados inseridos.

A partir desta página, é possível aceder à página do Registo, no caso de ser um utilizador novo que não tenha criado ainda uma conta.

20.3 Página de criação de conta para novos utilizadores - registo

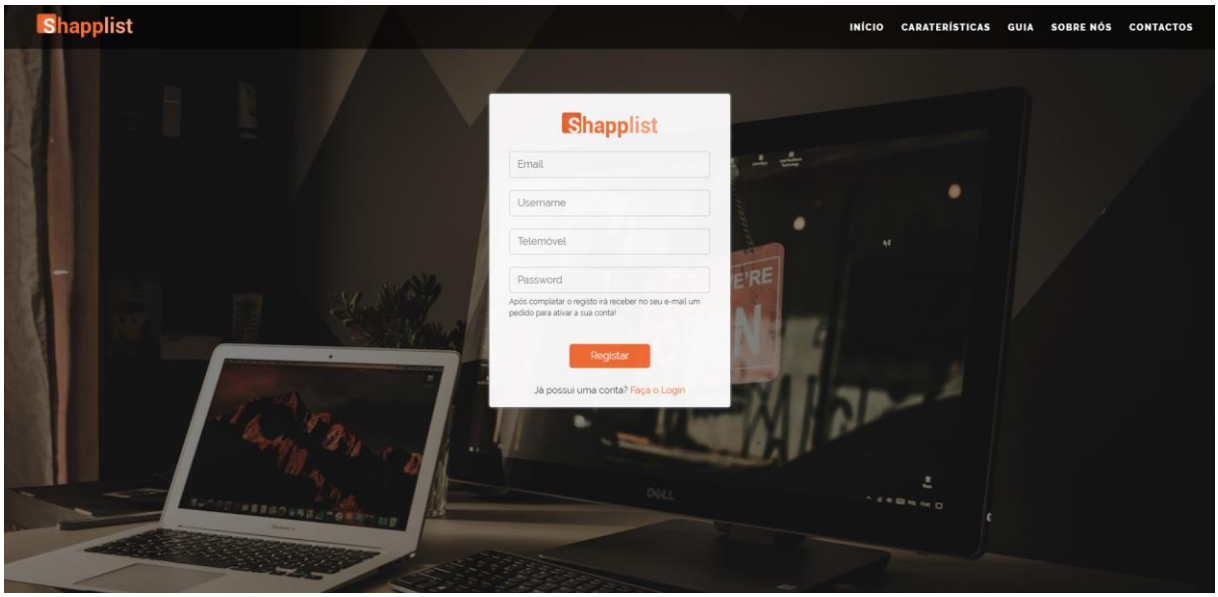


Figura 43 – Página de registo de novos utilizadores

Na zona de registo, o utilizador pode registar-se e criar a sua nova conta pessoal. Através dos dados pedidos, a sua conta de utilizador é criada. Os dados pedidos para a criação de conta são: email – endereço eletrónico pessoal para onde será enviado um email de confirmação para aceder à conta e que será futuramente utilizado, como visto anteriormente, para aceder à área pessoal do utilizador, em conjunto com a palavra passe; *username* – nome personalizado que vai identificar o *user* dentro da aplicação para os seus amigos e familiares, caso esteja em listas partilhadas; telemóvel – usado para os amigos ou familiares convidarem o utilizador para as suas listas; e palavra-passe – palavra secreta para garantir um acesso seguro e apenas pelo utilizador à sua área pessoal.

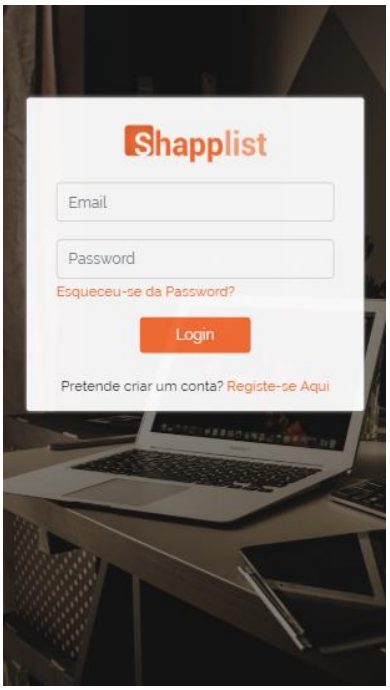


Figura 44 – Página de login – versão de telemóvel

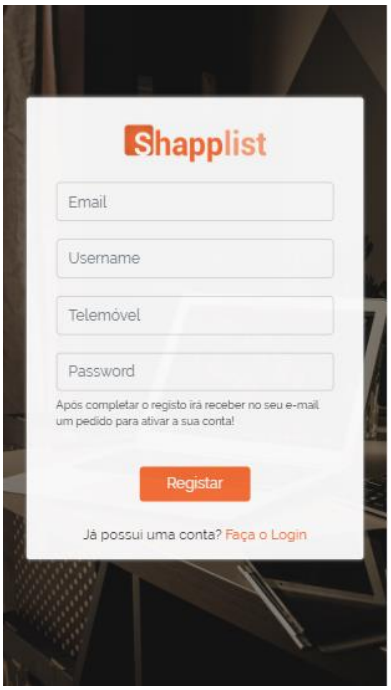


Figura 45 – Página de registo – versão de telemóvel

Em ambas as páginas, na versão de telemóvel, o design é idêntico às respetivas páginas na versão apresentada antes. A pequena diferença é a de que a barra de navegação é suprimida, aumentando assim o espaço de visualização disponível.

20.4 Página dos produtos disponibilizados – *hub* central da zona pessoal da aplicação

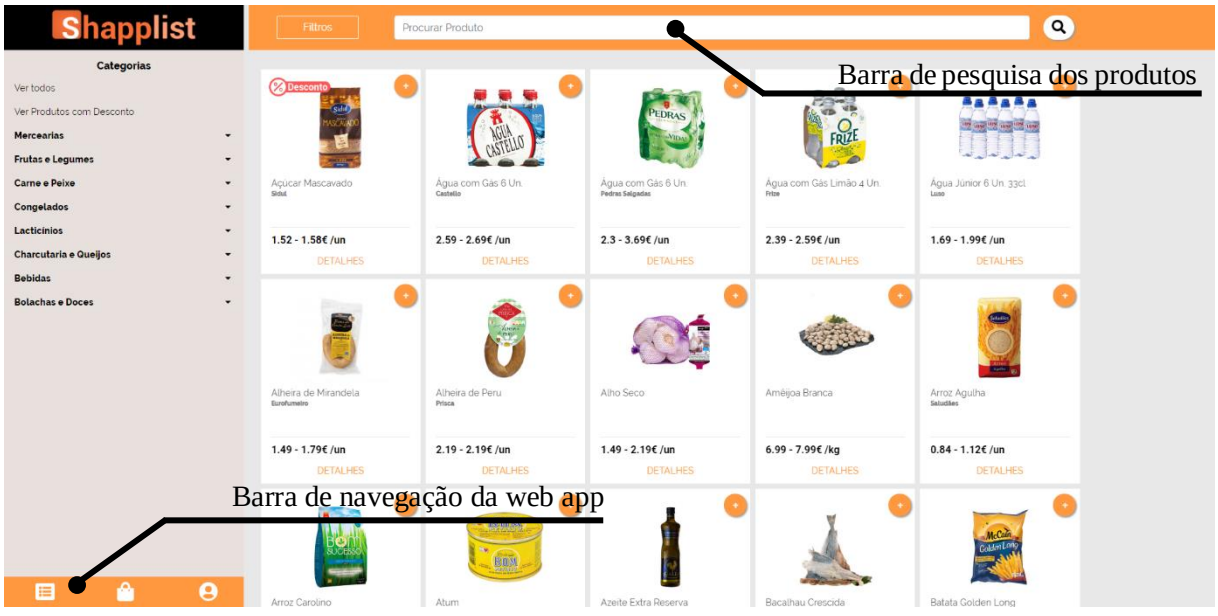


Figura 46 – Página dos produtos disponibilizados – *hub* central da zona pessoal da aplicação

A página dos produtos é a página central de toda a área pessoal da aplicação. Nesta é possível visualizar todos os produtos que se encontram disponíveis.

No canto inferior esquerdo, está localizada a barra de navegação que possui três botões que redirecionam o utilizador para: a zona das listas, a zona dos produtos (atual) e a zona das definições de utilizador, respetivamente.

No topo, destaca-se a barra de pesquisa de produtos pelo nome ou pela marca destes. À medida que o utilizador for escrevendo, a aplicação vai mostrando os produtos filtrados que possuem, no nome ou na marca, os carateres inseridos. À esquerda desta, encontra-se o botão dos filtros que permite ao utilizador ordenar os produtos apresentados por: ordem alfabética – ascendente e descendente – e por preço – do maior para o menor e vice-versa.

À esquerda dos produtos, está a secção das categorias. Esta permite ao utilizador filtrar os produtos de acordo com a categoria geral em que estes estão inseridos. Existem oito principais categorias: mercearias, frutas e legumes, carne e peixe, congelados, lacticínios, charcutaria e queijos, bebidas e bolachas e doces. Cada um destas tem ainda várias subcategorias que permitem fazer uma filtragem melhorada, permitindo a demonstração de uma pequena amostra de produtos. Para além destas, existem duas categorias gerais: ver todos e ver produtos com desconto.

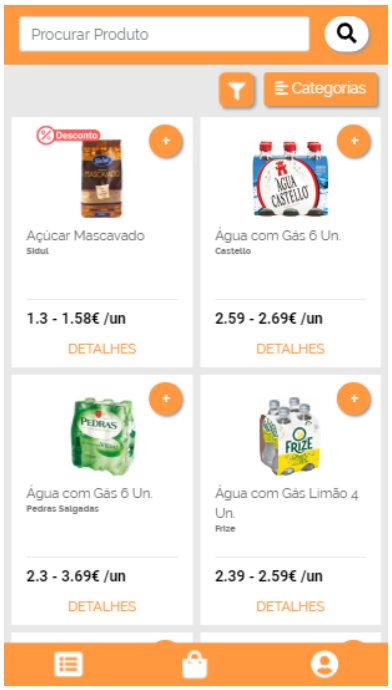


Figura 47 – Página dos produtos – versão de telemóvel

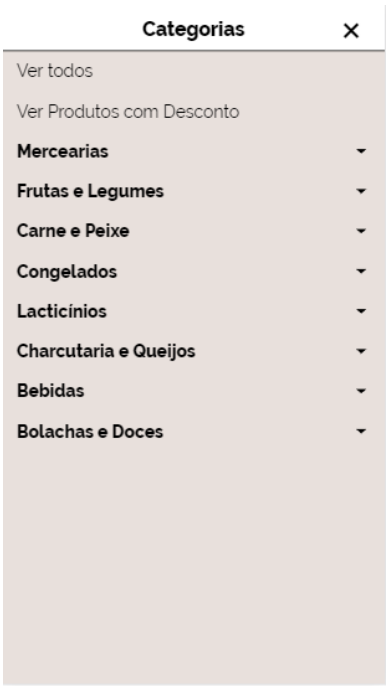


Figura 48 – Página dos produtos – secção das categorias – versão de telemóvel

A página dos produtos, no modo de telemóvel, possui algumas alterações comparativamente com a do modo computador. As categorias encontram-se agora, tal como

acontece com a barra de navegação na para introdutória, condensadas num botão que ao ser clicado, abre uma aba lateral onde são apresentadas todas as categorias – ver figura 11.

Outra diferença existente é a barra de navegação que, apesar de possuir os mesmos três ícones, preenche agora toda a parte inferior do ecrã.

Os produtos podem ser adicionados às listas de duas formas: através do botão "Detalhes" ou através do botão "+", localizado no topo direito de cada um dos produtos.

Este último permite adicionar o produto à lista de uma forma rápida, pelo que esta funcionalidade é denominada de "*fast-add*". Para isso, o utilizador apenas precisa de criar uma lista ou estar inserido numa lista como convidado, sendo que uma das listas é considerada a lista ativa (ver área das listas). Esta ferramenta permite assim que o utilizador possa adicionar múltiplos artigos à distância de um clique em cada produto que queira acrescentar, sem nunca ter de sair da página e/ou abrir ou fechar janelas de confirmação. O utilizador, após adicionar com o produto à sua lista, visualizará uma pequena mensagem temporizada a informar que o produto foi adicionado com sucesso.

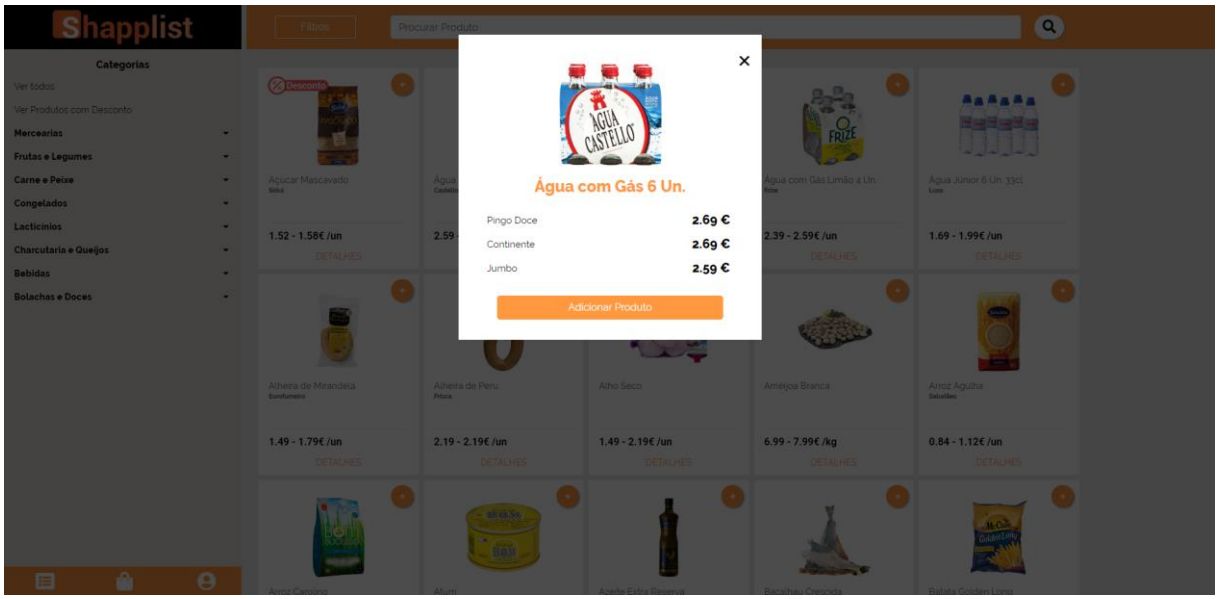


Figura 49 – Produto específico selecionado

O método "tradicional" de adicionar os produtos, através do botão "Detalhes", permite ao *user*, ter um maior controlo sobre os produtos a adicionar, podendo visualizar os preços destes descriminados em cada um dos supermercados. Caso o preço do produto esteja a vermelho, significa que esse produto está descontado relativamente ao preço original do mesmo nesse supermercado. Outra forma de visualizar que o produto está descontado, é através da imagem informativa no topo esquerdo de cada produto.

Finalmente, caso o utilizador queira adicionar o produto a uma das suas listas, através de “Adicionar A”, é possível escolher qual a lista (ou listas) à qual quer adicionar o produto.

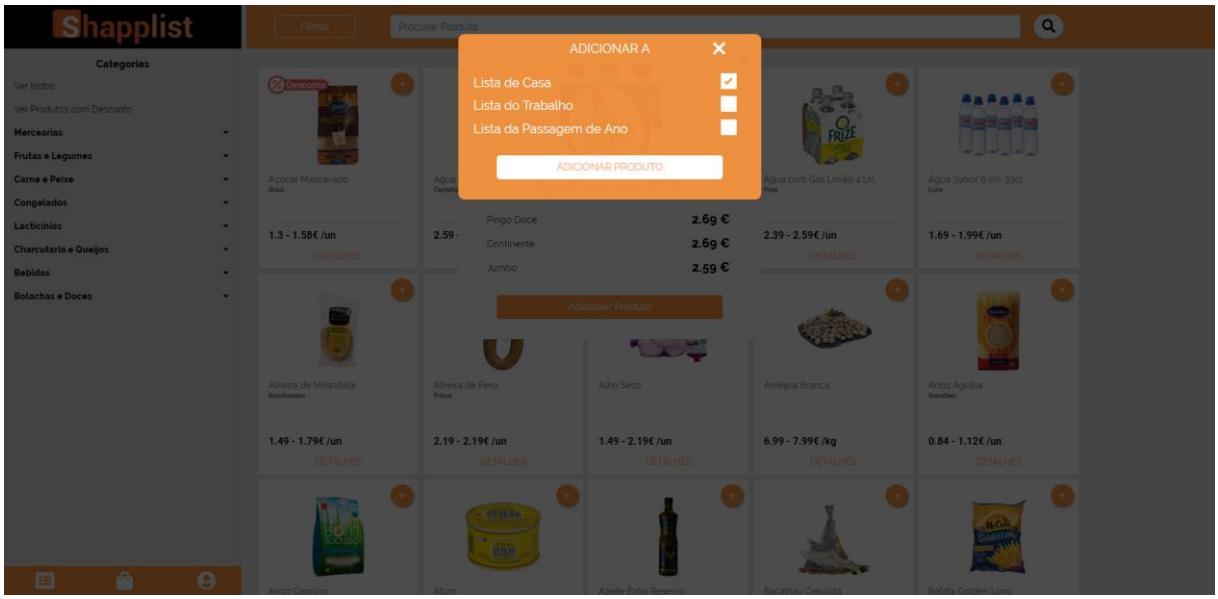


Figura 50 – Escolher uma lista (ou listas) para adicionar o produto escolhido

Este permite ao utilizador escolher qual a lista, ou listas, à qual quer adicionar o produto escolhido. A lista que está ativa será sempre apresentada como verificada para o produto ser adicionado, no entanto, caso não seja essa a lista a que o utilizador pretende adicionar o produto, poderá retirar essa confirmação. Cada produto pode ser adicionado a uma ou mais listas. Este processo é terminado quando o utilizador clica no botão “Adicionar Produto”, recebendo também aqui a mensagem temporizada de que o produto foi adicionado com sucesso.

Figura 51 – Página dos produtos – produto individual selecionado – versão de telemóvel

Figura 52 – Página dos produtos – escolher lista para adicionar o produto – versão de telemóvel

A visualização dos produtos e a escolha da lista à qual os adicionar é idêntica entre os dois modos de visualização.

20.5 Página das listas

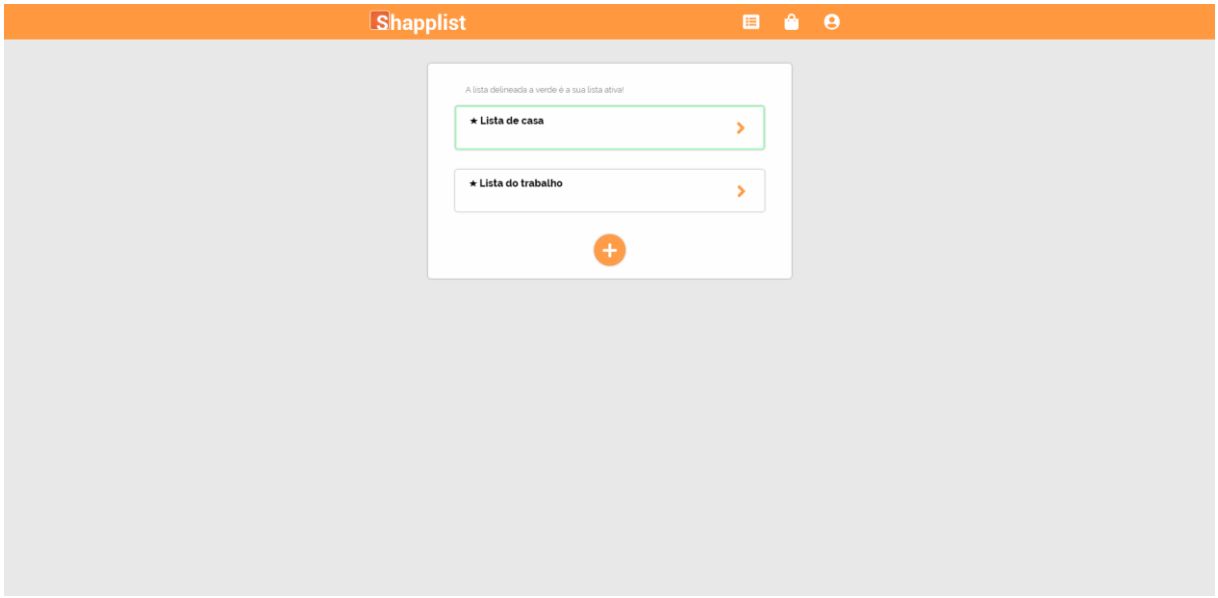


Figura 53 – Página de visualização das listas

A página das listas permite ao utilizador visualizar todas as listas criadas por si, diferenciadas pela "★" antes do nome definido para cada uma das listas, bem como as listas para as quais o utilizador foi convidado. A lista ativa, que tem como principal objetivo permitir

ao utilizador usar a função de adição rápida de produtos, encontra-se delineada a verde. Se a lista tiver mais do que um participante (o utilizador que a criou), por baixo do nome desta aparecerá o nome dos participantes dessa lista.

O utilizador pode também criar novas listas através do botão "+", localizado depois das listas.

No topo direito, ao contrário do que acontece na página dos produtos, está a barra de navegação com os ícones representativos das várias áreas da aplicação.

20.6 Página de criação de novas listas

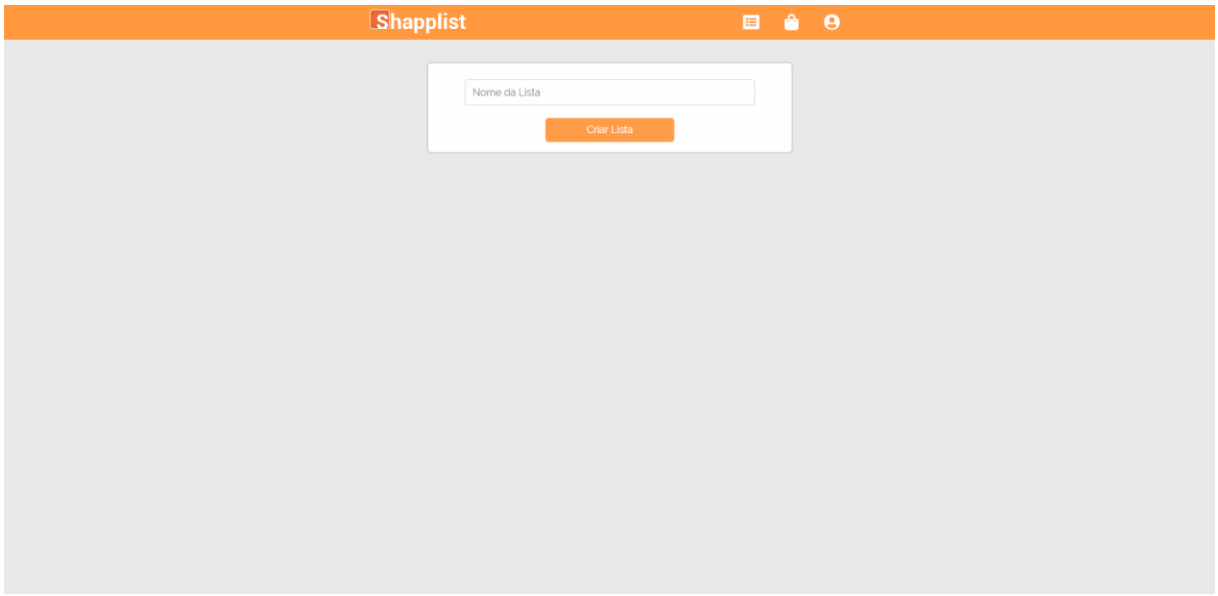


Figura 54 – Página de criação de novas listas

A página de criação de novas listas permite ao utilizador criar novas listas. Caso o utilizador pretenda adicionar participantes a esta, poderá fazê-lo na página individual referente a essa lista, utilizando o número do telemóvel do utilizador a convidar.

Figura 55 – Página das listas – versão de telemóvel

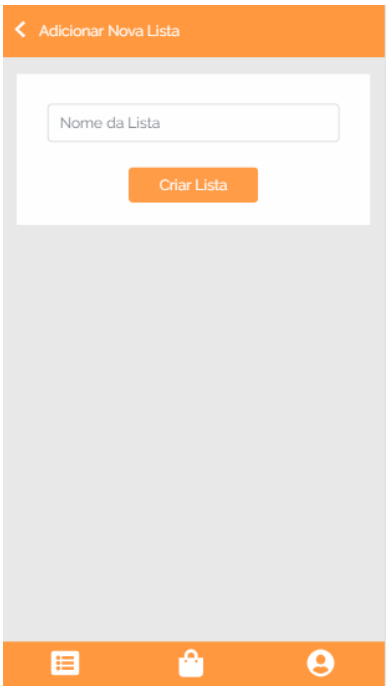


Figura 56 – Página de criação de novas listas – versão de telemóvel

20.7 Página individual de cada lista

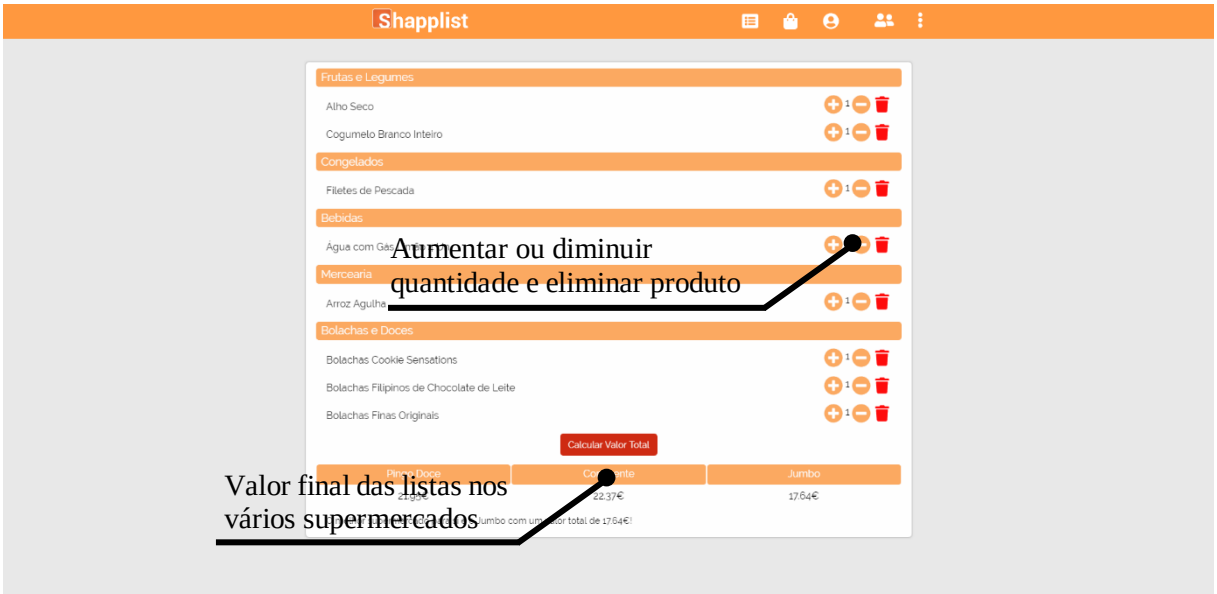


Figura 57 – Página individual de cada lista

A página referente a cada lista permite ao utilizador visualizar toda a informação relacionada com cada lista específica.

Os produtos adicionados à lista podem aqui ser visualizados por categoria. Depois de adicionados à lista, aqui é possível ao utilizador aumentar ou diminuir a quantidade dos

produtos selecionados. Estes também podem ser eliminados da lista, no caso de já não ser necessário ao utilizador.

O valor total da lista vai sendo atualizado à medida que são adicionados produtos novos ou é aumentada ou diminuída a quantidade de cada um. Cada lista, por defeito, contabiliza os valores totais nos supermercados disponíveis, sendo que, caso o utilizador não pretenda contabilizar um destes, pode retirá-lo na secção de definições da lista. O valor final da lista é calculado para informar o utilizador de qual o supermercado mais vantajoso para os produtos selecionados.

Se algum dos produtos adicionados à lista não se encontrar num dos supermercados, este não será contabilizado para o valor final da lista, pelo que é apresentado ao utilizador qual o produto que não existe e em que supermercado.

No topo da página, para além dos ícones de navegação já referenciados, existem também dois novos botões: "Participantes" e "Definições da Lista".

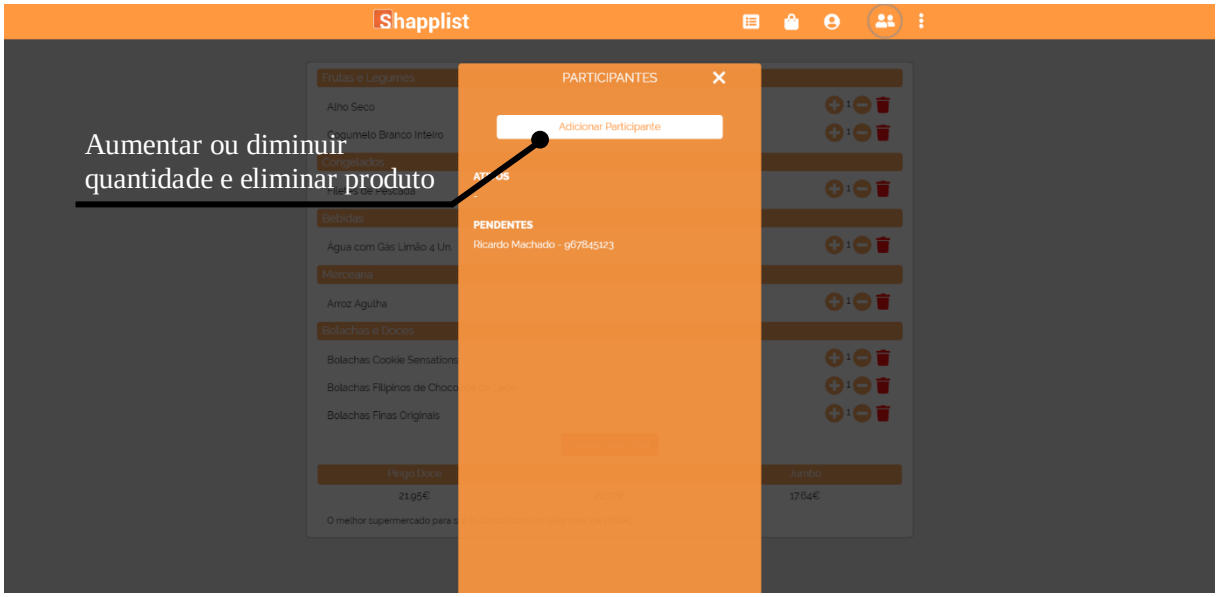


Figura 58 – Secção relativa aos participantes

A secção dos participantes permite ao utilizador visualizar os participantes ativos na lista, bem como os que foram convidados, mas ainda não aceitaram/rejeitaram o convite que lhes foi feito.

Os *users* ativos na lista podem ser removidos pelo administrador e criador da lista.

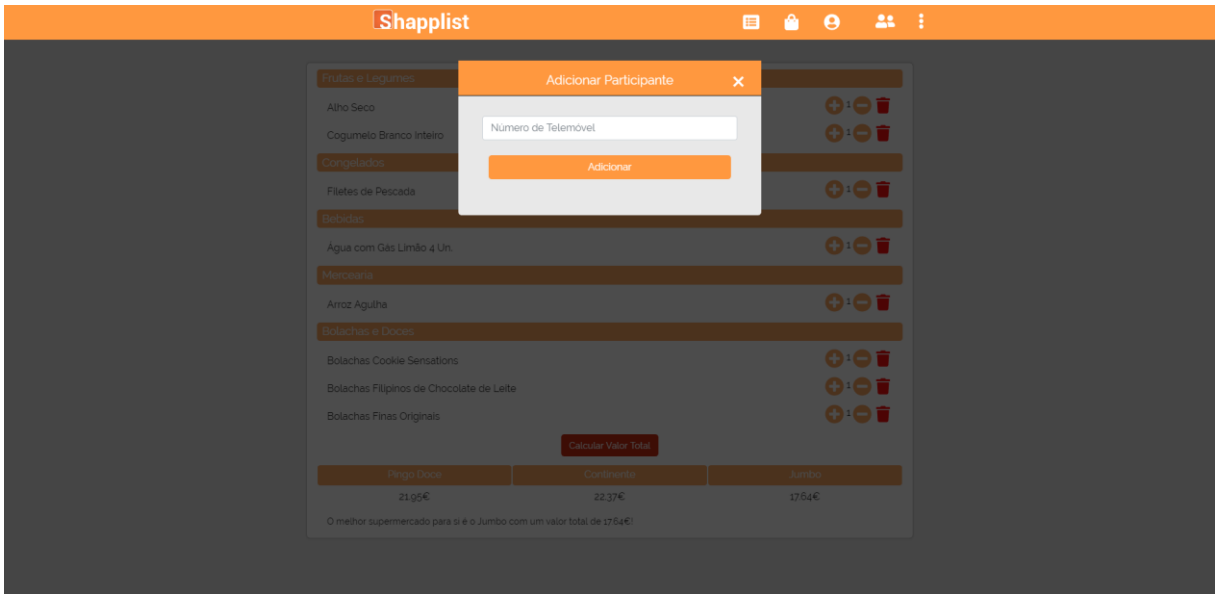


Figura 59 – Secção para adicionar novos participantes

Novos participantes podem também ser adicionados aqui, como referido anteriormente, pela inserção do número de telemóvel associado à conta do utilizador convidado.

Caso o número inserido não exista, será apresentada uma mensagem que informa o utilizador que o convite não foi enviado. Por outro lado, se o convite for enviado, o *user* será informado que este convite foi enviado com sucesso.

Figura 60 – Página individual da lista

Figura 61 – Página individual da lista - participantes

A página individual de cada lista, na versão mobile, tem uma grande diferença relativamente ao modo de visualização do computador, pois, para eliminar o produto, é agora

necessário fazer um longo clique no produto a eliminar e confirmar essa eliminação na janela que é apresentada ao utilizador.

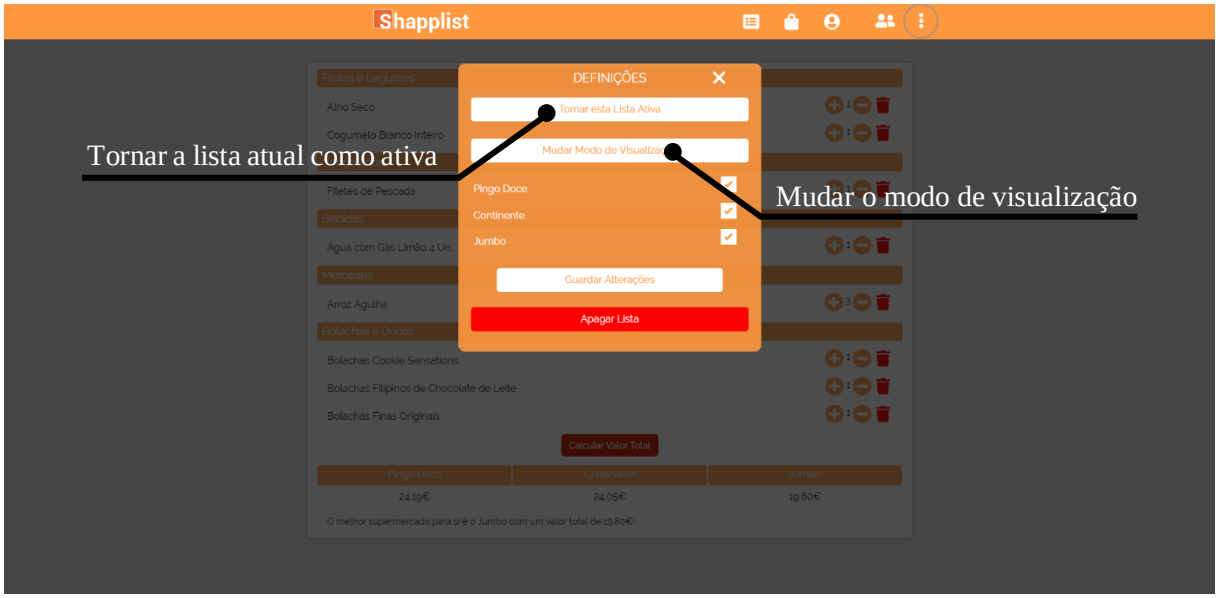


Figura 62 – Secção de definições da cada lista

A área de definições da lista possibilita ao utilizador tornar a lista ativa, no caso de pretender que esta se torne na sua lista ativa para o "*fast-add*". No caso de a lista visualizada ser a lista ativa, essa opção não irá estar disponível.

O utilizador tem a possibilidade de definir se, para o cálculo final da lista, pretende contabilizar todos os supermercados disponíveis ou limitá-los à sua escolha. Através das três *checkboxes* disponíveis, em que o valor por defeito é *checked* (verificado), o utilizador pode retirar um ou dois dos supermercados que não sejam opção para finalizar as suas compras, independentemente dos preços praticados nestes. Para confirmar a configuração, apenas é necessário confirmar as alterações.

O administrador da lista é o único utilizador que pode apagar as listas por si criadas. Por outro lado, os utilizadores convidados nas diversas listas apenas podem sair da lista, não tendo permissões para as apagar.

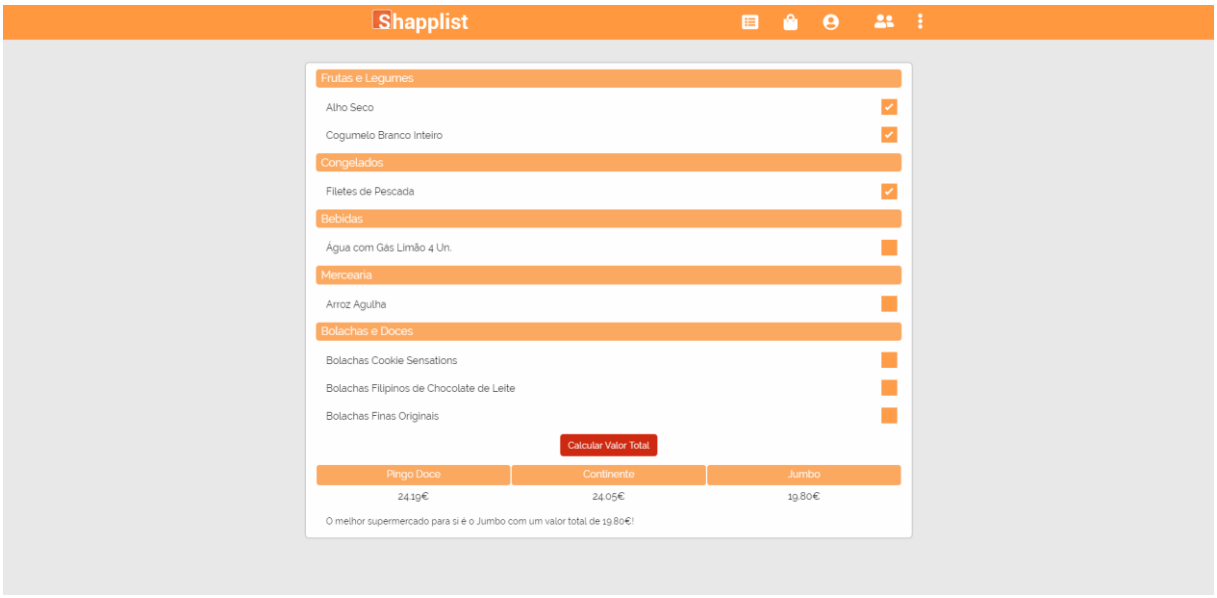


Figura 63 – Modo de visualização das listas para compras

O utilizador tem a possibilidade de mudar o modo de visualização da lista, ou seja, escolher um dos dois tipos disponíveis: modo compras e modo lista. O modo de compras carateriza-se pela possibilidade de, à medida que o utilizador for colocando os artigos no cesto/carrinho no supermercado, possa também ir confirmando os artigos já adquiridos, "riscando-os" da lista. O modo de lista, que é o modo ativo por defeito e foi o já apresentado, permite ao *user* aumentar/diminuir a quantidade dos produtos ou apagá-los.

Figura 64 – Página individual da lista - definições

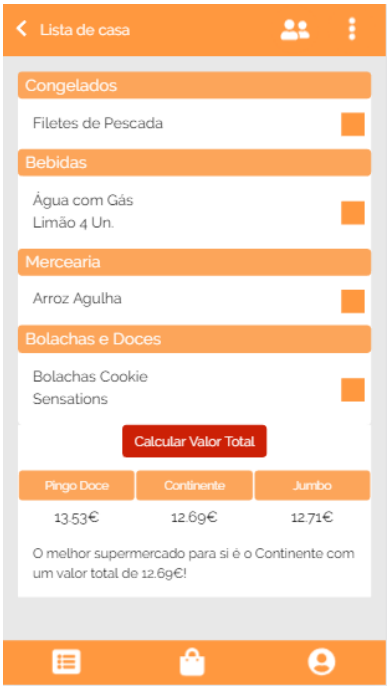


Figura 65 – Página individual da lista – modo de compras

20.8 Página de definições pessoais do utilizador

Figura 66 – Página de definições pessoais de cada utilizador

A página de definições permite ao utilizador verificar as suas informações pessoais como: *username* ou nome de utilizador, o email fornecido e utilizado para fazer *login* e ainda o telemóvel, usado pelos outros utilizadores para convidarem amigos e familiares para as listas.

Aqui, é possível ao utilizador terminar a sessão. Quando a sessão é terminada o utilizador será reencaminhado para a página de *login* da aplicação. Caso o *user* queira reentrar, terá de fornecer novamente os dados de *sign in* – o email e a palavra-passe correspondente. Por outro lado, se o utilizador quiser fechar a aplicação, mas manter a sessão, apenas tem de a fechar e a sessão será mantida durante algum tempo até a sessão expirar.

Nesta página, foi definido que o utilizador pode visualizar os convites feitos pelos outros utilizadores que o convidem para as suas listas, podendo estes convites ser aceites ou rejeitados.

No caso de não existirem convites pendentes, será apresentada uma mensagem informativa ao utilizador.

20.9 Página de edição dos dados pessoais do utilizador

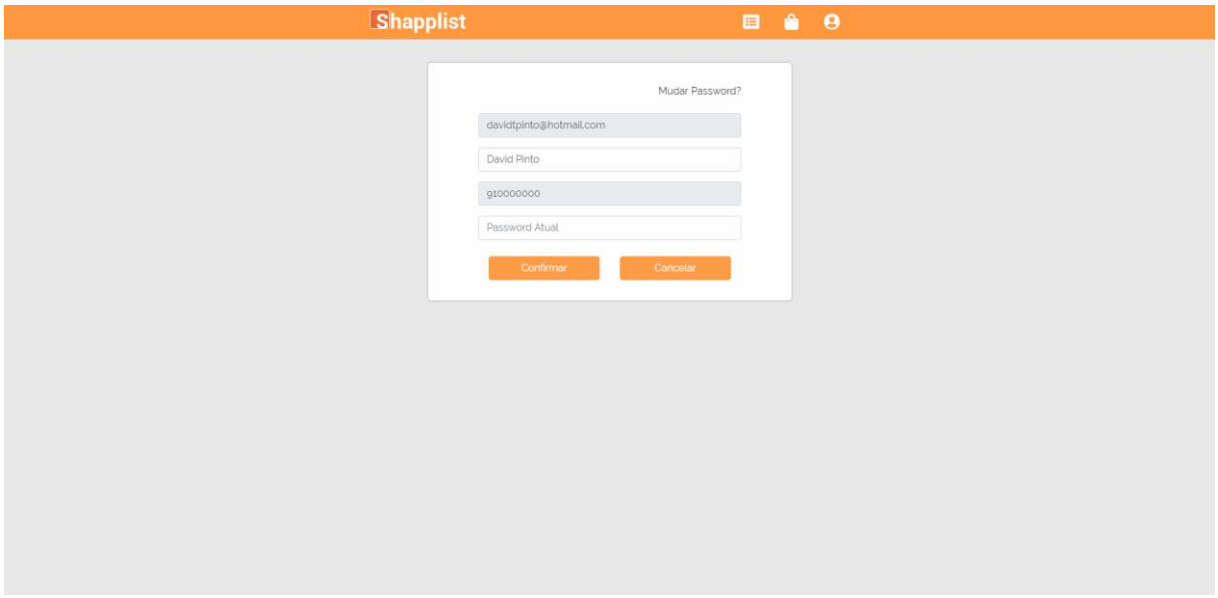


Figura 67 - Página de edição dos dados pessoais do utilizador

A página de edição dos dados pessoais permite que o utilizador edite algumas das usas informações como o nome de utilizador e a palavra-passe, sendo que, em ambos os casos, a verificação da veracidade da palavra-passe atual funciona como forma de guardar a alteração das informações.

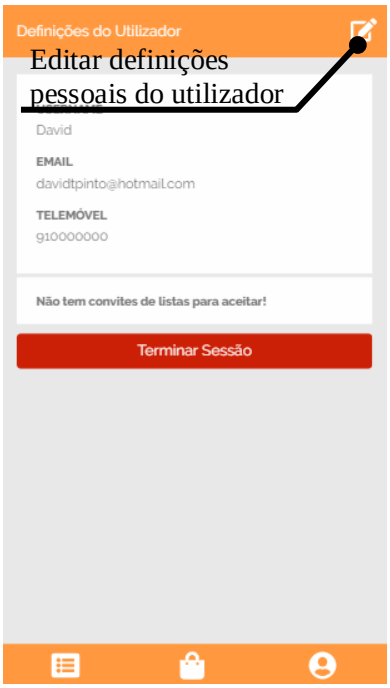


Figura 68 – Página de definições do utilizador

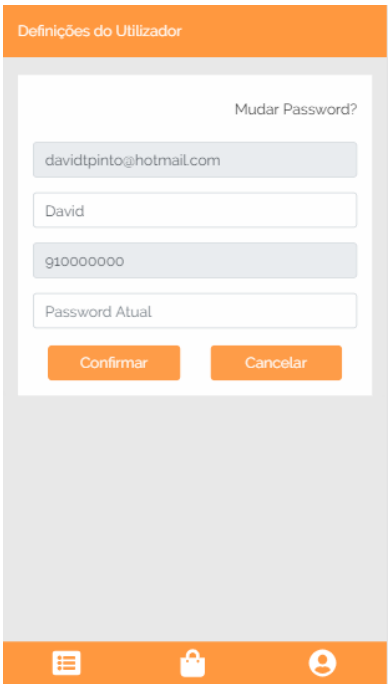


Figura 69 – Página de edição das definições do utilizador

UNIVERSIDADE AUTÓNOMA DE LISBOA
LUÍS DE CAMÕES

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

ShappList

Manual de Instalação

Autores: David Santos Cordeiro Teixeira Pinto, Henrique Filipe Chuva, Pedro Jorge de Almeida Correia, Ricardo Filipe Vicente Machado

Orientador: Daniel Silvestre

Número do/a candidato/a: 20160740, 20160715, 20160536, 20160746

Julho de 2019

Lisboa

(Página em Branco)

21 Índice

22 Web app ShappList – Passos para a instalação

A *web app* ShappList foi desenvolvida com o objetivo de manter sempre alta disponibilidade e sem pontos singulares de falha, que pudessem colocar em causa todo o funcionamento da aplicação. Para isso foram desenvolvidas 6 *VMs*, que garantem toda a interoperabilidade da aplicação e do servidor adjacente.

22.1 Guia de Instalação das *virtual machines*

Primeiramente, é necessário instalar as *VMs*. Para isso, é necessário o programa *Virtual Box* que vai possibilitar criar as máquinas virtuais e definir os parâmetros necessários para funcionarem corretamente. Depois da instalação do *Virtual Box*, e antes da criação das *VMs* internas da rede, é necessária a criação de uma rede *NAT* (*NAT Network*). Para isto, seguem-se os seguintes passos:

1. "*Tool*";
2. "*Preferences*";
3. "Network";
4. "*Add New Nat Network*";

Depois deste processo, a network criada deverá ser editada, sendo que o nome utilizado neste projeto é "*ShappNetwork*" e a network CIDR é "192.168.11.0/24". A network não deverá, no entanto, possuir um servidor DHCP (também definido nesta janela).

No que toca a cada *VM*, estas são criadas a partir dos seguintes passos:

1. "*File*" (topo da janela do Virtual Box);
2. *Import Appliance* (Ctrl + I) – é escolhida a OVF da *VM* a importar;
3. *Next* (2 vezes);

De reforçar que todas a *VMs* criadas são definidas do mesmo modo. Para fazer o *download* do software adicional destas e para configurar todas a especificações, é necessário manter uma ligação à internet.

As *virtual machines* criadas para o projeto possuem um *username* e uma palavra correspondente associadas. Neste caso, as credenciais usadas são:

- Username – "*template*";
- Palavra-passe – "*qwerty*";

Das seis máquinas virtuais criadas, foram definidos dois grupos: o primeiro possui as VMs *nginxServer* e *databaseServer*; e o segundo possui as *pageServer*, *productService*, *listService* e *authService*.

As *VMs nginxServer* e *databaseServer*, após executadas, não necessitam de configuração adicional e estão prontas a ser executadas. As *VMs pageServer*, *productService*, *listService* e *authService*, após executadas, requerem um comando específico para executar o serviço disponibilizado:

- *pageServer*: docker run -p 80:4466 shapplist/shapplistservices:pageserverdeploy_v10;
- *listService*: docker run -p 80:4444 shapplist/shapplistservices:listservicedeploy_v5;
- *productService*: docker run -p 80:44555 shapplist/shapplistservices:productservicedeploy_v4;
- *authService*: docker run -p 80:4433 shapplist/shapplistservices:authservicedeploy_v4;

Finalmente, quando todas as máquinas estiverem configuradas, a execução das VMs deve seguir a seguinte ordem:

1. *databaseServer*;
2. *nginxServer*;
3. *authServer*, *productServer* e *listServer*;
4. *pageServer*;

UNIVERSIDADE AUTÓNOMA DE LISBOA
LUÍS DE CAMÕES

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

ShappList

Manual de Programador

Autores: David Santos Cordeiro Teixeira Pinto, Henrique Filipe Chuva, Pedro Jorge de Almeida Correia, Ricardo Filipe Vicente Machado

Orientador: Daniel Silvestre

Número do/a candidato/a: 20160740, 20160715, 20160536, 20160746

Julho de 2019

Lisboa

(Página em Branco)

23 Índice

24 Lista de Fotografias/Ilustrações

25 Introdução

O manual do programador consiste num documento que retrata as funcionalidades essenciais da aplicação ShappList de modo a que seja possível dar a entender como manter ou melhorar o código da aplicação. Também neste manual será descrito criar e gerir as máquinas virtuais de modo a executar os micro-serviços.

26 *Web app* ShappList – Principais funções

A identificação de um utilizador é feita a partir de uma *token*. Estas são cadeias de caracteres aleatórios, utilizando o nome escolhido pelo utilizador para a criação da sua conta. As *tokens* (figura 1) são guardadas nas *cookies* (figura 2) do *browser* do utilizador.

Figura 70 - Cookie guardada no browser

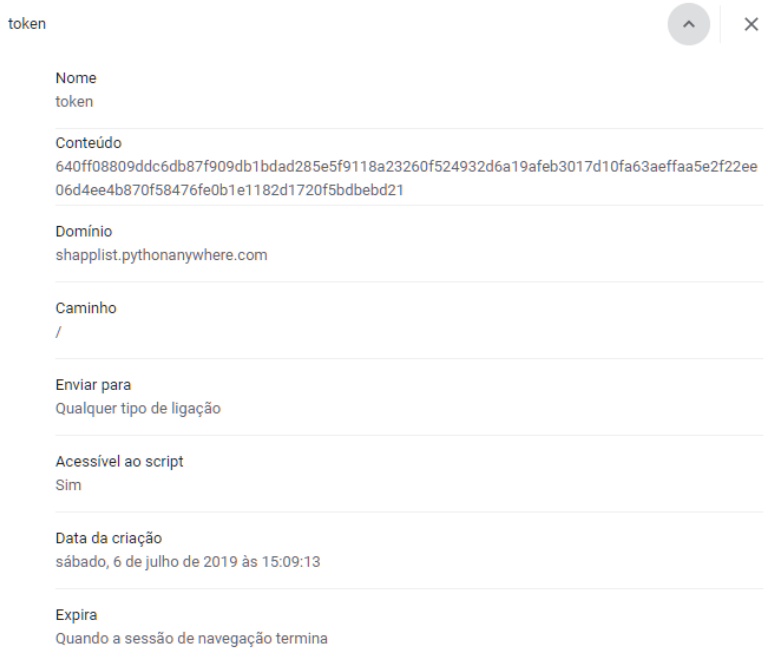


Figura 71 - Conteúdo da cookie definido pela token

A existência destas permite à aplicação identificar que o utilizador está *signed in*, redirecionando-o de volta para a página de entrada caso não este não tenha a *cookie*. As *tokens* são também guardadas na base de dados de modo a não ser necessário a criação de uma nova sempre que o utilizador acede à aplicação. Para impedir que a base de dados fique

sobrecarregada com entradas sempre que uma *token* nova é criada, estas são periodicamente removidas por um *script* que verifica a data de criação da *token*. Este intervalo tempo é definido usando *Unix time*, sendo que as *tokens* são removidas a cada 72 horas. Caso o utilizador faça o *sign in* na aplicação antes das 72 horas, a data de criação da *token* é reposta.

Como foi referido na fase de planeamento, o *back-end* é dividido em 4 micro-serviços: *pageServer*, *authService*, *productService* e *listService*.

26.1 Serviço *pageServer*

O serviço *pageServer* é responsável por identificar a página *HTML* necessária para cada secção da aplicação e apresentá-la ao utilizador. Este comportamento é definido nos ficheiros *urls.py* e *views.py* da *framework* Django. Esta ligação direta entre os dois ficheiros permite à aplicação atualizar rapidamente as páginas *HTML* consoante as ações do utilizador. Cada *URL* da aplicação é descrito no ficheiro *urls.py*, que identifica a *view* correspondente no ficheiro *views.py*. Dentro deste é feita uma avaliação do *URL* pedido e é encontrada a página *HTML* específica, que é devolvida ao utilizador. Dentro de cada uma das *views* deste serviço é também feita uma verificação que analisa a *cookie* do utilizador, averiguando se esta contém ou não uma *token* (figura 3). Esta verificação impede que o utilizador aceda à aplicação sem uma *token*, mas também permite que este não tenha de fazer o processo de *sign in* sempre que pretende usar a aplicação.

```python
def products(request):
    accept = False

    #caso o utilizador tenha uma token é feita uma verificação ao tempo de expiração desta
    if(not request.COOKIES.get('')):
        temp_token = request.COOKIES.get('token')
        if(temp_token != None ):
            #check_token verifica se a token ultrapassou o tempo de expiração
            accept = check_token(temp_token)

    #se o utilizador tiver uma token e esta ainda nao tiver expirado é imediatamente redirecionado para a pagina de produtos
    if(accept):
        template = loader.get_template('pageServer/tempProducts.html')

    #se o utilizador não tiver uma token, ou esta ja tenha expirado é redirecionado para a pagina de login
    else:
        template = loader.get_template('pageServer/login.html')


    context = {
        'debug' : 'debug'
    }

    return HttpResponse(template.render(context, request))

```

Figura 72 - Função que verifica a token do utilizador quando este tenta aceder à página dos produtos pelo URL desta

26.2 Serviço *authService*

O serviço *authService* trata de todas as funcionalidades relativas à autenticação do utilizador, registo de novos utilizadores, visualização das suas informações pessoais e envio de *e-mail* após a criação de uma nova conta para a confirmar.

Este serviço trata da verificação dos dados do utilizador quando este tenta realizar o *sign* in, comparando os dados introduzidos com os que já existem na base de dados (figura 4). Caso o utilizador tenha introduzido corretamente as suas credencias, este será redirecionado para a página dos produtos e irá também receber uma *cookie* com a sua *token*. Quando o utilizador termina a sua sessão, esta *token* é apagada.

```python
def api_login(request):

    #dicionário enviado como resposta para o front-end
    responseData = {}
    if(request.method =="POST"):

        #informação recebida pelo front-end
        dict_data = json.loads(request.body.decode('utf-8'))
        temp_email = dict_data['email']
        temp_password = dict_data['password']
        #query que verifica se o utilizador introduziu corretamente as suas credenciais
        userID = check_info_query(temp_email,temp_password)

        if(not userID):
            responseData['updatedDefaultList'] = False
            responseData['status'] = 400
            responseData['message'] = 'User creds wrong '

        elif(userID):
            #obtenção da lista ativa do utilizador, caso ela exista, ou seja, é diferente de 0
            temp_val = query_getUserDefaultList(userID)
            if( temp_val != 0):
                responseData['updatedDefaultList'] = True
                responseData['defaultList'] = temp_val
            else:
                responseData['updatedDefaultList'] = False

            responseData['status'] = 200
            responseData['message'] = 'User creds ok '
            #criação da token do utilizador a partir do seu email e identificador
            responseData['token'] = create_token(temp_email,userID)
        else:
            print('Query error')

    else:
        responseData['status'] = 500
        responseData['message'] = 'Not POST'

    #resposta é enviada de volta para o front-end
    return JsonResponse(responseData)
```

Figura 73 - API que analisa a informação introduzida pelo utilizador quando este tenta realizar *o sign in*

Quando é realizado o registo de um novo utilizador, antes de esta informação ser enviada e registada na base de dados, é primeiro avaliada a integridade dos campos "*e-mail*" e "número de telefone". O *e-mail* fornecido é avaliado, verificando se tem o caracter "@", se tem uma extensão permitida (google.com, outlook.pt, etc.) e se contém caracteres inválidos ("%", "/", etc). A avaliação do número de telefone é menos complexa, pois é apenas verificado se o número fornecido corresponde a um número de telemóvel com o indicativo "9", que representa os serviços de comunicações móveis em Portugal e se o número inserido tem nove dígitos.

Com estas verificações feitas é então enviada a informação introduzida ao *back-end* e esta é comparada com a que já existe na base de dados. Não é permitido que dois utilizadores tenham o mesmo *e-mail* ou número de telefone, portanto, caso a informação enviada no registo não respeite esta regra, ela é recusada e o utilizador recebe um aviso. Se as informações forem válidas, o registo do utilizador é feito, este recebe uma *token* e é redirecionado para a página principal da aplicação, a página dos produtos. Após um registo com sucesso, o utilizador irá receber no seu correio eletrónico um *e-mail* com um *hyperlink* para confirmar a criação da sua conta (figura 5). Este link é dinamicamente criado para cada utilizador diferente utilizando a sua *token* pessoal.

```python
def api_signup(request):
    responseData = {}
    #informação recebida pela base de dados
    dict_data = json.loads(request.body.decode('utf-8'))
    #informação introduzida pelo utilizador para o registo
    temp_username = dict_data['username']
    temp_password = dict_data['password']
    temp_email = dict_data['email']
    temp_phoneNum = dict_data['phoneNum']
    new_data = {}

    if(request.method =="POST"):
        #query que verifica se email ou número de telemovel ja está em uso
        if(not query_checkIfUserWithMailOrPhone(temp_email,temp_phoneNum)):
            #query que adiciona utilizador à tabela userss
            if(create_user(temp_username,temp_password,temp_email)):
                #query que obtém o ID do utilizador para ser introduzido na tabela users_extended
                new_data = get_user_by_email(temp_email)
                #query que adiciona o user à tabela users_extended
                if(create_user_extended(new_data['userID'],temp_email,temp_phoneNum)):
                    responseData['status'] = 200
                    responseData['message'] = "User succesfully created"
                    #token criada a partir do nome do utilizador e do seu ID
                    responseData['token'] = create_token(temp_username, new_data['userID'])
                    #token criada para criar um hyperlink para ativação da conta
                    temp_ver_token = generateHash(temp_username, temp_email)

                    #envio do email de confirmação de conta
                    if(query_addVerificationTokenToDatabase(temp_ver_token, new_data['userID'])):
                        support_sendEmail(temp_email, temp_ver_token, temp_username)

                else:
                    if(delete_user(new_data['userID'])):
                        responseData['status'] = 400
                        responseData['message'] = "Could not create user extended "
            else:
                responseData['status'] = 400
                responseData['message'] = "Could not create user "

        elif(query_checkIfUserWithMailOrPhone(temp_email,temp_phoneNum)):
            responseData['status'] = 300
            responseData['message'] = "User creds already exist "
    else:
        responseData['status'] = 500
        responseData['message'] = 'Not POST '


    return JsonResponse(responseData)
```

Figura 74 - API responsável pelo registo de um novo utilizador e pelo envio do email de confirmação de conta

Por último, o serviço *authService* permite também ao utilizador ter uma área pessoal com as informações da sua conta, permitindo a alteração do seu "*username*" e "palavra-passe".

26.3 Serviço *productService*

O serviço *productService* permite ao *front-end* apresentar ao utilizador todos os produtos contidos na base de dados de um modo dinâmico e eficiente. Este permite ao utilizador pesquisar produtos manualmente, filtrar por preço, por nome, utilizar a procura por categorias ou subcategorias e adicionar produtos às suas listas pessoais. A pesquisa manual, utilizando a barra de pesquisa da aplicação, é tratada de um modo flexível pelo *back-end*, permitindo ao utilizador procurar pelo nome específico de um produto ou pelo nome de uma marca conhecida (figura 6).

```
def select_query(product_dict):
    #função que define a query a ser executada na base de dados
    #a query pode ser feita pelo nome do produto caso seja uma pesquisa manual
    #pode ser feita pelo menu das categorias e sub categorias
    #os resultados são devolvidos em ordem ao nome ou ao preço sendo que pode ser ordenado de modo ascendente ou descendente
    query = 'SELECT uniqueID,itemName,itemPrice,priceType,itemCategory,itemSubCategory,imgPath,discountValue,storeID FROM products WHERE '
    boolSearch = False
    boolCategory = False

    if(product_dict['searchbar_keyword'] != ''):
        sub_stringSearch = 'itemName like \'%' + product_dict['searchbar_keyword'] + '%\''
        boolSearch = True

    if(product_dict['sub_category'] != ''):
        sub_stringCategory = 'itemSubCategory = \'' + product_dict['sub_category'] + '\''
        boolCategory = True

    elif(product_dict['category'] != ''):
        sub_stringCategory = 'itemCategory = \'' + product_dict['category'] + '\''
        boolCategory = True
    elif(product_dict['searchbar_keyword'] == ''): #caso o utilizador utilize a opção "Ver todos"
        sub_stringSearch = 'itemName like \'%%\''
        boolSearch = True


    if(boolSearch and boolCategory):
        finalQuery = query + sub_stringSearch + ' AND ' + sub_stringCategory
    elif(boolSearch):
        finalQuery = query + sub_stringSearch
    elif(boolCategory):
        finalQuery = query + sub_stringCategory

    finalQuery = finalQuery + 'ORDER BY '+product_dict['orderBy']+' '+ product_dict['clause']+ ';'
    print(finalQuery)
    return finalQuery
```

Figura 75 - Função no *back-end* que define que filtros utilizar para obter os produtos

Cada uma destas pesquisas é resultado de inquirições à base de dados de modo a não sobrecarregar a máquina do utilizador. Estes pedidos à base de dados são respondidos com toda a informação necessária (figura 7) para o utilizador avaliar e comparar os produtos disponíveis, analisando os preços nas diversas lojas e descontos disponíveis.

```python
def executeQuery(query):
    cursor_id = connection.cursor()
    #execução da query na base de dados dependendo dos filtros utilizados
    cursor_id.execute(query)
    rows_id = cursor_id.fetchall()

    data = {}
    data['totalSize'] = len(rows_id)
    data['products'] = {}
    query_resultSize = len(rows_id)
    found_products = {}
    total_products = 0
    lastAdded = 0
    for i in range(query_resultSize):
        temp_dict = {}

        if rows_id[i][1] not in found_products.values():

            temp_dict['id'] = []
            temp_dict['storeID'] = []
            temp_dict['preco'] = []
            temp_dict['desconto'] = []

            temp_dict['id'].append(rows_id[i][0])
            temp_dict['nome'], temp_dict['marca'] = treatBrand(rows_id[i][1]) #separação do campo itemName do produto em nome e marca
            temp_dict['preco'].append(rows_id[i][2])
            temp_dict['tipo_preco'] = rows_id[i][3]
            temp_dict['categoria'] = rows_id[i][4]
            temp_dict['sub_categoria'] = rows_id[i][5]
            temp_dict['path'] = rows_id[i][6]
            temp_dict['desconto'].append(rows_id[i][7])
            temp_dict['storeID'].append(rows_id[i][8])

            data['products']['prod' + str(lastAdded)] = temp_dict
            found_products['prod' + str(lastAdded)] = rows_id[i][1]
            lastAdded +=1
            total_products +=1
        else: #caso seja um produto repetido, mas de um supermercado diferente
            temp_nome,temp_marca = treatBrand(rows_id[i][1])
            existent_key = getKeyByValue(data,temp_nome,temp_marca)
            data['products'][existent_key]['id'].append(rows_id[i][0])
            data['products'][existent_key]['preco'].append(rows_id[i][2])
            data['products'][existent_key]['desconto'].append(rows_id[i][7])
            data['products'][existent_key]['storeID'].append(rows_id[i][8])

    data['totalSize'] = total_products
    return data
```

Figura 76 - Obtenção dos produtos e ordenação destes num dicionário que será devolvido ao *front-end*

26.4 Serviço *listService*

O serviço *listService* é o serviço mais complexo e extenso de todo o projeto, visto que este é responsável pela grande maioria das funcionalidades que caracterizam a aplicação. Estas funcionalidades permitem ao utilizador uma personalização das suas listas pessoais, podendo atribuir nomes à sua escolha, eliminá-las, enviar convites a outros participantes de modo a que estes possam visualizar e utilizar a lista, adicionar e remover produtos ou definir a quantidade destes e que supermercados a considerar para o cálculo do valor total da lista.

Antes da criação de uma nova lista é feita uma rápida verificação à integridade do nome escolhido, de modo a verificar se este contém caracteres inválidos, caso isto não se verifique a lista é criada e o utilizador pode agora adicionar produtos e participantes a esta (figura 8).

```python
def api_createList(request):
    #informação recebida pelo front-end
    dict_data = json.loads(request.body.decode('utf-8'))
    #informação introduzida pelo utilizador
    listname = dict_data['listname']
    defaultLevel = dict_data['defaultLevel']
    #token do utilizador, guardada na cookie do seu bwoser
    userToken = request.COOKIES.get('token')
    #obtenção do identificador do utilizador a partir da token
    userID = tokenToUser(userToken)
    #query que verifica se o utilizador está em alguma lista
    #se o utilizador nao estiver em nenhuma lista, a lista a ser criada será definida como a sua lista ativa
    noLists = query_checkIfUserInAnyList(userID)

    if(createList(listname,userID,defaultLevel)):
        print('List created')
        if(not noLists):
            if(setDefaultListOnCreate(userID)):
                data = {'status' : 'success', 'updatedDefaultList' : True, 'defaultList' : query_getListByParticipant(userID)}
            else:
                data = {'status' : 'failed'}
        else:
            data = {'status' : 'success', 'updatedDefaultList' : False}
    else:
        data = {'status' : 'failed'}

    return JsonResponse(data)
```

Figura 77 - API responsável por adicionar à base de dados a nova lista criada pelo utilizador

Quando o utilizador acede à página que apresenta todas as suas listas, são feitas verificações que permitem ao *front-end* identificar quais são as listas criadas pelo utilizador, os participantes de todas as suas listas e qual destas é a sua lista ativa. Como foi descrito no *front-end*, a adição de produtos pode ser feita através da funcionalidade *fast-add* ou através dos detalhes de cada produto. A funcionalidade *fast-add* é realizada através da verificação da variável "*activeList*", que identifica uma das listas do utilizador, como sendo a sua lista ativa. Esta é armazenada no próprio *browser* do utilizador (figura 9), de modo a poder ser utilizada em diversas secções da aplicação. Esta variável existe sempre desde o momento em que o

utilizador cria a sua conta, sendo que tem o valor "0" caso este não tenha nenhuma lista criada, ou não seja participante na lista de outro utilizador. A variável "*activeList*" é também guardada na base de dados, de modo a ser possível manter a mesma entre sessões, sendo que esta é fornecida ao utilizador sempre que este realiza o *sign in*.

Key	Value
allLists	95,128
acceptedStores	2
activeList	95

Figura 78 - Variáveis guardadas no armazenamento local do *browser*

Com a sua lista criada, o utilizador pode agora convidar outros para visualizar a sua lista. Estes convites são executados a partir da verificação do número de telefone. Como foi feito na fase de registo, é verificada a integridade do número introduzido. Após esta verificação o número é recebido pelo serviço no *back-end* que verifica se o número de telefone está associado a algum utilizador e, caso esteja, cria um convite para este aceitar ou recusar (figura 10).

```python
def api_getParticipant(request):
    #informação recebida pelo front-end
    dict_data = json.loads(request.body.decode('utf-8'))
    #informação introduzida pelo utilizador
    temp_phone = dict_data['phoneNum']
    listID = dict_data['listID']
    #obtenção do identificador do utilizador a partir da cookie
    userToken = request.COOKIES.get('token')
    selfID = tokenToUser(userToken)
    data={}

    #query que obtém o identificador do utilizador que vai receber o convite
    userID = query_getParticipant(temp_phone)

    if(selfID != userID):
        if(not userID):
            data['status'] = 300
            data['message'] = 'Phone number does not exist '

        elif(userID):
            data['status'] = 200
            data['message'] = 'User found'
            #query que verifica se o utilizador a ser convidada já é participante da lista
            if(not query_checkIfAlreadyParticipant(userID,listID)):
                #query que verifica se o convite jáe xiste
                if(query_checkDupeInvite(userID,listID)):
                    #query que adiciona o convite
                    if(query_addInvite(userID,listID)):
                        data['status'] = 200
                        data['message'] = 'Invite added'

                    else:
                        data['status'] = 400
                        data['message'] = 'Could not create invite'
                else:
                    data['status'] = 200
                    data['message'] = 'Invite already exists'

            else:
                data['status'] = 600
                data['message'] = 'User already in list as participant'
    else:
        data['status'] = 500
        data['message'] = 'Sending invite to self'

    return JsonResponse(data)
```

Figura 79 - API responsável pela receção de novos convites

Os convites podem ser visualizados na zona das definições pessoais do utilizador. Caso o utilizador decida aceitar o convite, este é adicionado aos participantes da lista e pode agora aceder a esta. Se o utilizador recusar o convite, este é indicado como recusado na base de dados (figura 11).

```python
def api_replyToUserInvites(request):
    #informação recebida pelo front-end
    dict_data = json.loads(request.body.decode('utf-8'))
    #obtenção do identificador do utilizador a partir da token guardada nacookie
    userToken = request.COOKIES.get('token')
    userID = tokenToUser(userToken)
    #resposta do utilizador ao convite para se juntar a uma lista
    inviteReply = dict_data['reply']
    listID = dict_data['listID']
    data = {}

    #se o utilizador aceitar o convite, é enviado um True
    #caso este recuse é enviado um False
    if(inviteReply):
        #query que adiciona o utilizador à lista
        if(query_updateParticipants(userID,listID)):
            #query que atualiza o estado do pedido para 'Accepted' ou para 'Refused'
            if(query_updateStatus(True,userID,listID)):
                data['status'] = 200
                data['message'] = 'Invite status now accepted'
            else:
                data['status'] = 400
                data['message'] = 'Could not update status'
        else:
            data['status'] = 400
            data['message'] = 'Could not add new participant'

    elif(not inviteReply):

        if(query_updateStatus(False,userID,listID)):
            data['status'] = 200
            data['message'] = 'Invite status now refused'
        else:
            data['status'] = 400
            data['message'] = 'Could not update status'

    return JsonResponse(data)
```

Figura 80 - API responsável por tratar a resposta ao convite para uma lista

É possível remover participantes da lista, no entanto não é possível remover o criador desta (figura 12).

```python
def api_removeUserFromList(request):
    #informação recebida pelo front-end
    dict_data = json.loads(request.body.decode('utf-8'))
    #informação que determina de que lista remover o utilizador
    listID = dict_data['listID']
    userID = dict_data['userID']
    data={}

    #query que verifica se o utilizador a ser removido é o dono da lista
    if(not query_checkIfRemovingOwner(listID,userID)):
        #query que remove o utilizador da lista
        if(query_removeUserFromList(listID,userID)):
            data['status'] = 200
            data['message'] = 'Successfully removed from list'
        else:
            data['status'] = 400
            data['message'] = 'Could not leave list'

    elif(query_checkIfRemovingOwner(listID,userID)):
        data['status'] = 300
        data['message'] = 'Trying to delete owner of list'
    else:
        data['status'] = 400
        data['message'] = 'Error checking owner'


    return JsonResponse(data)
```

Figura 81 - API responsável pela remoção de um utilizador

Quando o utilizador pretende eliminar uma lista da sua aplicação, primeiro é feita uma verificação que identifica se o utilizador é o criador da lista. Caso isto se verifique, a lista é eliminada da aplicação de todos os que a podem ver e editar (figura 13). Caso o utilizador não seja o criador da lista, este apenas consegue remover-se a si próprio (figura 14).

```python
def api_deleteList(request):
    #informação recebida do front-end
    dict_data = json.loads(request.body.decode('utf-8'))
    listID = dict_data['listID']
    data = {}

    #query que elimina a lista
    if(query_deleteListByID(listID)):
        #query que verifica se existem utilizadores com a lista ativa que foi eliminada
        #todos estes utilizadores vão ter a sua defaultList mudada para 0
        if(query_updateDefaultListOnDelete(listID)):
            data['status'] = 200
            data['message'] = 'List successfully deleted and defaultLists changed to 0'
        else:
            data['status'] = 400
            data['message'] = 'Could not delete list'
    else:
        data['status'] = 400
        data['message'] = 'Could not delete list'

    return JsonResponse(data)
```

Figura 82 - API que elimina uma lista caso a pessoa seja o criador dela

```python
def api_leaveList(request):
    #informação recebida pelo front-end
    dict_data = json.loads(request.body.decode('utf-8'))
    listID = dict_data['listID']
    userToken = request.COOKIES.get('token')
    userID = tokenToUser(userToken)
    data = {}

    #query que retira o utilizador dos participantes da lista
    if(query_leaveList(listID,userID)):
        data['status'] = 200
        data['message'] = 'Successfully left list'
    else:
        data['status'] = 400
        data['message'] = 'Could not leave list'

    return JsonResponse(data)
```

Figura 83 - API que remove um utilizador convidado para uma lista

Após a adição de produtos à lista, os participantes desta podem aumentar ou diminuir a quantidade de cada produto individual através de um simples botão, é também possível remover o produto inteiramente da lista. Todos estes pedidos são tratados no *back-end*, que faz o pedido à base de dados para atualizar o conteúdo da lista consoante os pedidos dos utilizadores (figura 15).

```python
def api_addProducts(request):
    #informação recebida pelo front-end
    dict_data = json.loads(request.body.decode('utf-8'))
    #informação que determina que lista e produto atualizar
    listID = dict_data['listID']
    productID = dict_data['productID']
    #variável que determina se vai ser adicionada ou removida quantidade
    tag = dict_data['tag']
    productQuantity = 1
    userToken = request.COOKIES.get('token')
    addedBy = tokenToUser(userToken)

    #query que verifica o produto já está na lista
    if(not query_verifyIfProductInList(listID, productID)):
        #query que adiciona o produto à lista
        if(addProduct(listID,productID,productQuantity,addedBy)):
            data = {'status' : 'success'}
        else:
            data = {'status' : 'failed'}
    #se o produto já existir na lista a variável tag determina se é adicionada ou removida quantidade
    else:
        if(tag):
            if(addQuantityToProduct(listID, productID)):
                data = {'status' : 'success'}
            else:
                data = {'status' : 'failed'}
        else:
            if(removeQuantityFromProduct(listID, productID)):
                data = {'status' : 'success'}
            else:
                data = {'status' : 'failed'}


    return JsonResponse(data)
```

Figura 84 - API responsável por adicionar produtos à lista, ou atualizar a quantidade desta

Quando o utilizador adiciona um novo produto, altera a quantidade deste ou remove-o inteiramente, o valor total da lista é recalculado. Este cálculo avalia o preço de cada produto nos diversos supermercados e faz uma soma pesada destes valores para cada uma das superfícies comercias. O resultado destas somas é apresentado ao utilizador, separado por supermercado, junto de um pequeno texto que indica qual a melhor opção para o utilizador. No caso de o utilizador escolher produtos que não existem em todas as lojas, irá também aparecer uma secção que indica que supermercados têm produtos em falta e quais são. Por último, o utilizador tem a possibilidade de alterar que supermercados quer permitir para o cálculo do valor total. Estas preferências são armazenadas no próprio *browser* de modo a que o utilizador não tenha de as mudar sempre que volta a entrar na página da lista.

26.5 Máquinas Virtuais

Para este projeto, foi definido que teriam de ser criadas seis máquinas virtuais, sendo que foram definidos dois grupos: o primeiro possui as VMs *nginxServer* e *databaseServer*; e o segundo possui as *pageServer*, *productService*, *listService* e *authService*.

As VMs são distribuídas com 1024 MB de RAM, sendo o recomendado para um micro-serviço normal este valor está entre 512 MB e 1024 MB, enquanto que para a base de dados situa-se entre os 2048 MB e 4096 MB.

A única interface de rede que cada VM contém é uma interface *NAT network*, proprietária da *Virtual Box*, sendo que o nome da rede "*ShappNetwork*" (com possibilidade de ser alterado, sendo que teria de ser também alterado em todas as *VMs*) e o CIDR é "192.168.11.0/24", sendo que este não pode ser modificado. Esta rede não contém um servidor DHCP, sendo que cada micro-serviço necessita de um endereço IP estático, sem a possibilidade de alteração em caso de engano.

Na *VM* que contém a base de dados, o motor/software utilizado é *MySQL*, sendo que o *username* é "ShappList" e a palavra-passe correspondente é "*qwerty*". A base de dados encontra-se configurada para aceitar qualquer ligação vinda de dentro da rede.

26.5.1 O software Docker

O *software Docker* definido para o projeto tem como *username* "*shapplist*" e o repositório com todos os serviços do projeto é denominado "*shapplistservices*". A configuração base do *software* é:

- A versão *Python* utilizada é a 3.6, devido a problemas relacionados com packages essenciais na versão 3.7;
- O diretório definido por defeito de cada serviço é /usr/src/<nomedoserviço>;
  As especificações das portas utilizadas são definidas em baixo:
- O servidor de *Django* executado no *Docker* contém a diretiva "--noreload", devido a problemas entre o funcionamento da *Docker image* resultante e o módulo de *Django statreloader*;

Notas dos requerimentos de *Python* - É necessário instalar as *packages* de *Python* *mysqlclient* e de *django-cors-headers*. A primeira foi instalada devido ao facto de existir o conector entre *MySQL* e *Python* na versão de *Django* a ser utilizada. No caso da segunda, esta é instalada devido ao facto de ser necessário fazer *requests* de múltiplas origens diferentes para o funcionamento de cada *API*.

26.5.1.1 Definição de portas de micro-serviços:

Os micro-serviços criados encontram-se em *VMs* separadas, tendo cada uma um IP diferente e estabelecida uma porta para cada micro-serviço. Estas são as portas que estão a ser utilizadas:

- *authService* – porta 4433;
- *listService* - porta 4444;
- *productService* - porta 4455;
- *pageServer* - porta 4466;

Devido a dificuldades na utilização do sistema de versões fornecido pelo próprio *Docker*, as versões são definidas no fim do nome de cada *Docker image*, separado por um "_" (ex: authservicedeploy_vX, onde "X" representa a versão). Notar também que o nome de cada *Docker image* é constituído pelo nome do micro-serviço, seguido do sufixo "*deploy*", para versões prontas a ser utilizadas.

26.6 Nomenclatura do caminho (*path*) de cada micro-serviço

Durante o projeto, foi estabelecida uma nomenclatura para facilitar o trabalho de reencaminhamento do *nginx* para cada micro-serviço, estabelecendo-se caminhos pré-definidos:

- Front-end (páginas visuais) – "/" (não tem qualquer *path* associado);
- Para *APIs* – "/api/<nomedaapi>/<apiendpoint>, sendo o nome da *API* o nome do micro-serviço e o *endpoint* representa a funcionalidade do serviço a ser requisitada;