

## App Install & Go

Roberto Filipe Manso Barreto  
(nrº 21123, regime diurno)

Orientação de  
Luís Gonzaga Martins Ferreira

LICENCIATURA EM ENGENHARIA EM SISTEMAS INFORMÁTICOS  
ESCOLA SUPERIOR DE TECNOLOGIA  
INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE

## **Identificação do aluno**

Roberto Filipe Manso Barreto  
Aluno número 21123, regime diurno  
Licenciatura em Engenharia em Sistemas Informáticos

## **Orientação**

Luís Gonzaga Martins Ferreira

## **Informação sobre o Estágio**

Motorline Eletrocelos S.A  
Travessa do Sobreiro, 29 Rio Côvo (Sta. Eugénia) 4755-474 Barcelos  
Eng. Helder Remelhe

## Resumo

Este documento trata o processo de análise, especificação e implementação da solução Install&Go. Esta vem resolver o problema de comunicação entre a empresa Motorline e os seus técnicos, dado que, na eventualidade de um problema estes deverão telefonar para a empresa, o que gera sobrecarga do sistema.

Para a resolução deste problema foi desenvolvido uma plataforma de fórum, onde as empresas podem registar os seus técnicos. Aqui, podem expor questões, onde técnicos externos de outras empresas, e/ou técnicos Motorline poderão prestar auxílio. Se um técnico possuir um problema já resolvido, este poderá pesquisar pela solução dada no fórum.

O desenvolvimento desta solução proveu a aquisição de novas capacidades tecnológicas como o desenvolvimento *cross-platform* e as suas *frameworks*, sendo que destas foi explorado o *flutter*. Através da elaboração desta aplicação foi possível, para além das competências mencionadas anteriormente, assimilar capacidades como análise, especificação de projetos e comunicação com clientes. Por fim, foi desenvolvida completamente a solução conforme as necessidades e expectativas do cliente.



## Abstract

This document describes the analysis, specification and development process of the Install&Go solution. This solution solves the communication problem between Motorline and their professionals, since that, in the event of a problem, they must call the company, which generates an overload.

To solve this problem a forum platform was developed, where companies can register their professionals. Here they can submit questions, so that other professionals from other companies and/or Motorline can provide assistance. If a professional has a problem that has already been solved, he can search for the solution in the forum

The development of this solution provided the acquisition of new technological skills such as cross-platform development and its frameworks, of which flutter was explored. Through the elaboration of this application it was possible, besides the competences previously mentioned, to assimilate skills such as analysis, project specification and communication with clients. At the end, the solution was completely developed according to the client's needs and expectations.



## Agradecimentos

Em primeiro lugar, gostaria de agradecer à minha família, com destaque à minha mãe, ao meu padrinho, aos meus avós e à minha namorada, que com todo o carinho orientaram-me nesta caminhada.

Também, gostaria de salientar um caloroso agradecimento à empresa Motorline, uma vez que, fizeram-me sempre sentir um membro da equipa, com destaque ao supervisor Helder Remelhe que sempre esteve disponível para eventuais dúvidas que surgissem no desenvolvimento do projeto.

Por fim, mas não menos importante, gostaria de enfatizar toda a orientação, disponibilidade, conversa e ensinamentos proporcionados pelo professor doutor Luís Ferreira e também aos meus colegas de curso Henrique Cartucho e João Castro que mais que colegas, são amigos para a vida.



# Conteúdo

<b>1 Trabalho desenvolvido</b>	<b>1</b>
1.1 Web scraper . . . . .	1
1.1.1 Implementação web scraper . . . . .	1
1.1.1.1 Implementação no website . . . . .	6
1.1.1.2 Melhoria de implementação . . . . .	7
1.1.1.3 Diagrama de base de dados final . . . . .	10
1.1.1.4 Armazenamento de dados . . . . .	10
1.2 Backend . . . . .	11
1.2.1 Organização do projeto . . . . .	11
1.2.2 Definição de rotas base . . . . .	12
1.2.3 Middlewares . . . . .	12
1.2.3.1 Idioma de Comunicação . . . . .	12
1.2.3.2 Autenticação . . . . .	13
1.2.3.3 Validação de <i>Role</i> . . . . .	14
1.2.3.4 Logging de erros . . . . .	14
1.2.3.5 Logging de pedidos com morgan . . . . .	14
1.2.4 Controllers . . . . .	15
1.2.4.1 Estruturação dos controllers . . . . .	15
1.2.4.2 Execução da lógica de negócio . . . . .	15
1.2.4.3 Validação dos dados . . . . .	16
1.2.4.4 Formulação da resposta . . . . .	16
1.2.4.5 Processamento de erros . . . . .	16
1.2.5 Envio de <i>emails</i> . . . . .	17
1.2.6 Agendamento de tarefas . . . . .	17
1.2.7 Cifra de <i>passwords</i> . . . . .	18
1.2.7.1 Cifra de configurações do servidor . . . . .	18
1.2.8 Documentação Typedoc . . . . .	19

1.2.9	Documentação Swagger . . . . .	20
1.3	Frontend . . . . .	21
1.3.1	Organização do projeto . . . . .	21
1.3.2	Processo de aprendizagem . . . . .	22
1.3.3	Extensions . . . . .	22
1.3.4	Handlers . . . . .	22
1.3.5	Providers . . . . .	22
1.3.6	Helpers . . . . .	23
1.3.7	Gestão de utilizadores . . . . .	24
1.3.8	Fórum . . . . .	25
1.3.8.1	Filtragem de tópicos . . . . .	26
1.3.8.2	Carregamento de tópicos . . . . .	27
1.3.8.3	Detalhes de tópico . . . . .	28
1.3.9	Firestore . . . . .	29
1.3.10	Apresentação de Imagens . . . . .	30
1.3.11	Apresentação de Imagens em carrossel . . . . .	31
1.3.12	Carregamento de Imagens . . . . .	31
1.3.13	Vídeos . . . . .	32
1.3.14	Links . . . . .	32
1.3.15	Notificações . . . . .	33
1.3.16	Permissões . . . . .	34
1.3.17	Ios . . . . .	35

# Listas de Figuras

1.1	Estrutura dos dados obtidos . . . . .	2
1.2	Página geral de produtos . . . . .	3
1.3	Página de produtos de uma subcategoria . . . . .	3
1.4	Página de produtos de uma subcategoria distinta . . . . .	4
1.5	Página de detalhes de produto, secção inicial . . . . .	4
1.6	Página de detalhes de produto, secção de informações . . . . .	5
1.7	Página de detalhes de produto, secção de videos . . . . .	5
1.8	Resposta obtida aquando o pedido ao <i>url</i> da página <i>web</i> . . . . .	6
1.9	Urls com erro primeira interação . . . . .	7
1.10	Exemplo de página de acessórios . . . . .	7
1.11	Indicação das páginas com falha . . . . .	8
1.12	Exemplo de página de produto incomum . . . . .	8
1.13	Exemplo de página de produto com subprodutos . . . . .	9
1.14	Exemplo de página de serviço . . . . .	9
1.15	atualização da base de dados . . . . .	10
1.16	Exemplo de página de produto incomum . . . . .	11
1.17	Comportamento de middlewares . . . . .	12
1.18	Utilização de tokens . . . . .	13
1.19	Autenticação - Login e Registo . . . . .	16
1.20	Documentação <i>typedoc</i> . . . . .	19
1.21	Documentação de classe <i>typedoc</i> . . . . .	19
1.22	Documentação swagger . . . . .	20
1.23	Exemplo de documentação de serviço . . . . .	20
1.24	Organização do projeto . . . . .	21
1.26	Aviso de login a conta sem acesso . . . . .	24
1.28	Filtragem do forum . . . . .	26
1.29	Carregamento de tópicos . . . . .	27
1.30	Destaque de mensagens . . . . .	28



# **Lista de Tabelas**



# Siglas & Acrónimos

**API** Application Programming Interface. 7, 79

**JSON** JavaScript Object Notation. 7, 17

**MVC** Model View Controller. 16

**UC** Use Cases. 34

**US** User Stories. 31



# 1. Trabalho desenvolvido

## 1.1 Web scraper

Após uma reunião com o cliente foi compreendido que o catálogo de produtos Motorline não se encontra num servidor. Estes dados apenas estão diretamente no *website* da empresa, sendo assim, foi determinado a necessidade de criar um *web scraper*.

Durante a reunião concluiu-se que o *web scraper* iria apenas correr uma vez por mês para ser evitada a sobrelocação do servidor. Para agilizar a realização do *web scraper* foi entregue pela empresa a estrutura do *website* a seguir para serem obtidas as informações.

### 1.1.1 Implementação web scraper

A implementação e testagem do *web scraper*, sem sobrelocação do servidor, foi alcançada, uma vez que, préviamente foi descarregado todo o *website* localmente, o que permitiu simular o catálogo.

Para a implementação do *web scraper* optou-se por uma abordagem mais simples. Esta abordagem, consiste na realização de um pedido para ser obtida a página *web*, ler e guardar os dados.

A linguagem utilizada foi *python* devido à facilidade de lidar com abundantes quantidades de dados. O tratamento dos dados das páginas *web* foi realizado com o auxílio da biblioteca *bs4*, também conhecida como *beautiful soup*, esta permite alimentar com uma página *web* e de seguida realizar pesquisas sobre esta com base em *tags* e atributos dos elementos.

Após um estudo do catálogo foi compreendido que dados seriam necessários, sendo estes:

1. Categorias e subcategorias de produtos;
2. Produtos de cada categoria e subcategoria;
3. Documentação dos produtos;
4. Imagens e vídeos dos produtos;

Para guardar estes dados foi utilizado um dicionário, que contém como chaves as categorias de produtos. Para cada categoria contém mais um dicionário com as subcategorias de produtos. Cada categoria possui uma lista de produtos, sendo cada produto representado por um dicionário, que abrange como chaves os seus atributos. A utilização de dicionários e listas para guardar estes dados deve-se a que o objetivo será guardar estes em *JSON* e a transformação é simplificada com a utilização destas estruturas devido à proximidade com a estrutura *JSON*.

```
{
    "produtos": [
        {
            "nome": "nome do produto",
            "categoria": "categoria do produto",
            "subcategoria": "subcategoria do produto",
            "imagem-amostra": "imagem de amostra do produto",
            "imagens": [
                "links de imagens de produtos"
            ],
            "descricao": "descrição do produto",
            "documentacao": [
                {
                    "nome": "Nome da documentação",
                    "url": "link da documentação"
                }
            ],
            "placas-controlo": [
                {
                    "nome": "nome da placa de controlo",
                    "url": "documentação placa de controlo"
                }
            ],
            "informacao-geral": "imagem de informação geral",
            "desenho-tecnico": "imagem de desenho técnico",
            "videos": [
                {
                    "name": "nome do vídeo",
                    "url": "link do vídeo"
                }
            ]
        },
        "categorias": {
            "nome_categoria": {
                "nome_subcategoria": [
                    {
                        "nome": "nome produto",
                        "link": "link página do produto",
                        "imagem-amostra": "link imagem de amostra do produto"
                    }
                ]
            }
        }
    ]
}
```

Figura 1.1: Estrutura dos dados obtidos

Após uma análise da estrutura do *website* foi constatado que a página geral dos produtos possui todas as categorias, assim como, as subcategorias com *urls* para as páginas que contém todos os produtos das subcategorias.

Sendo assim, em primeiro lugar percebeu-se que cada conjunto(categoria, subcategorias) é uma secção, pelo que, são obtidas todas as secções das categorias. Para cada uma destas secções, é obtido o título que equivale ao nome da categoria e também todos os elementos clicáveis. Estes, correspondem às subcategorias que contêm o nome e um *url*, que redireciona para a página de produtos da subcategoria.



Figura 1.2: Página geral de produtos

Posto isto, já é possível identificar cada categoria e subcategoria, assim como, o *url* da página de produtos para cada subcategoria. Mas, após alguma análise dos dados foi percebido que estas não contêm acentuação visto que, existe uma formatação no *website*. Para resolver este problema, foram pesquisadas ferramentas capazes de corrigir erros ortográficos. Pelo que, descobriu-se que a biblioteca mais utilizada em *python* para resolver este problema é a biblioteca *spellchecker*. Esta ferramenta, por sua vez, é a mais utilizada dado à sua capacidade de correção dos erros ortográficos em diversas linguagens. Por fim, sempre que uma categoria e subcategoria é obtida, antes de ser guardada, é corrigida.

Aquando a finalização do processo anterior, cada *url* é aberto e a partir disto, são obtidos os *urls* de produtos e imagens de amostra dos produtos. Em cada página, foram obtidos todos os elementos pressionáveis existentes, sendo que cada um refere-se a um produto. Para obter o nome do produto correspondente, foi utilizado o nome contido no *url* do elemento, pois todos os produtos seguem a mesma estrutura, sendo esta, /produtos/nome-produto. Como em *urls* não é permitido utilizar acentuação e espaços, todos os nomes foram corrigidos com a mesma ferramenta de correção ortográfica mencionada anteriormente.

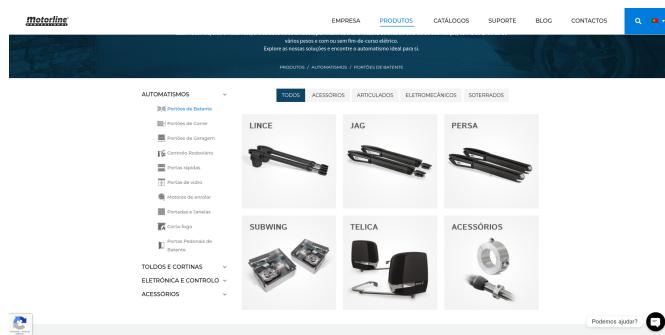


Figura 1.3: Página de produtos de uma subcategoria

Neste momento, depois de correr o código foi deduzido que existem algumas páginas de produtos em que este não conseguia obter dados, pelo que, um erro era atirado. Para compreender exatamente em que páginas de produtos surgia este erro, sempre que um erro era detetado, este *url* era adicionado a uma nova chave do dicionário mencionado anteriormente. Esta chave, tem o nome *misses* e contém todos os *urls* em que algum erro aconteceu. Então, nesta ocasião percebeu-se que nem todas as páginas de produtos são iguais, contudo, após uma reunião com o cliente este expôs que existem páginas de produtos e de detalhes de produtos que são muito diferentes das restantes.

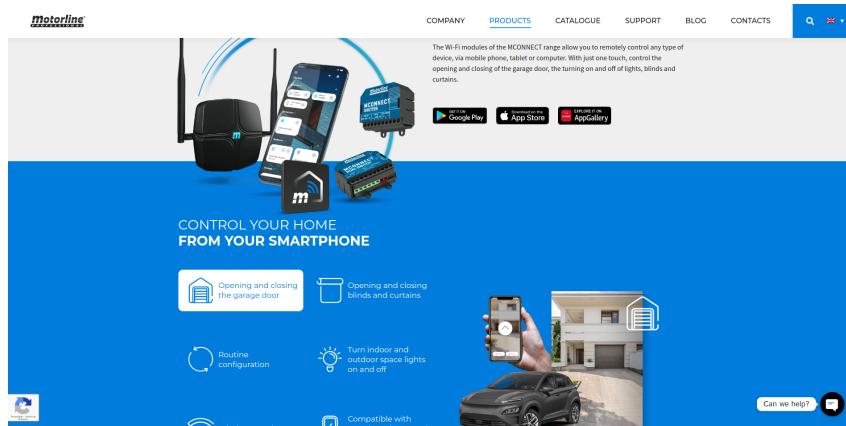


Figura 1.4: Página de produtos de uma subcategoria distinta

Com o objetivo de não atrasar o restante projeto foi determinado primeiramente, que todos os produtos que contêm páginas semelhantes deveriam ser obtidos. Para cada página de produto, foi obtido o título que corresponde ao nome do produto e o elemento que contém o *id* produto-descrição, que corresponde à sua descrição. As imagens dos produtos são disponibilizadas através de *urls* na secção da galeria do produto, pelo que, são obtidos todos os seus *urls*.

A documentação dos produtos pode ser disponibilizada através de *urls* para os manuais, ou, com uma lista *dropdown* com todos os manuais em formato *url*, o que permite, serem obtidos através de todos os *urls* da secção de documentação, assim como, os nomes e todas as opções do *dropdown* se este existir.

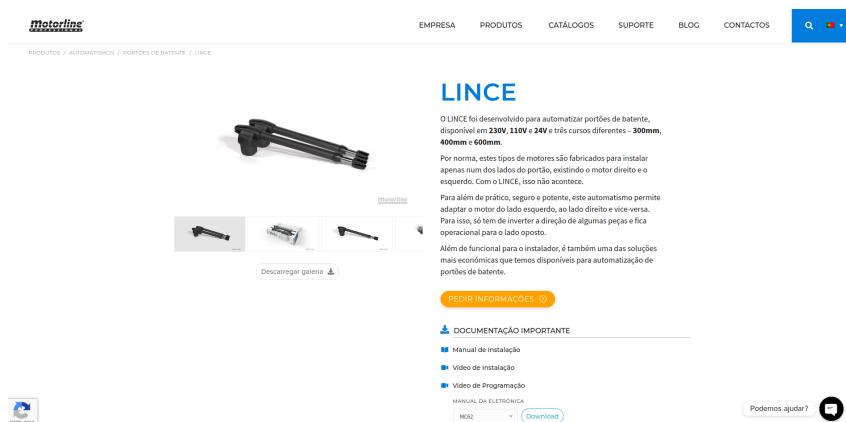


Figura 1.5: Página de detalhes de produto, secção inicial

As imagens de desenho técnico e informação geral encontram-se disponíveis na secção correspondente ao nome de cada uma, caso existam, são obtidas as imagens e os seus *urls*.

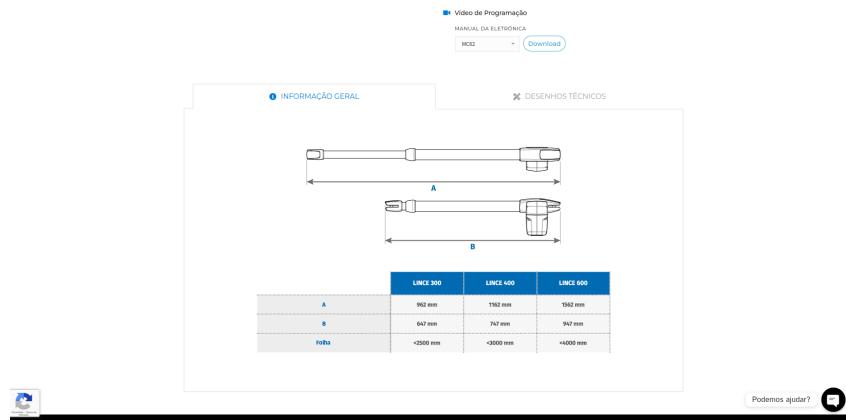


Figura 1.6: Página de detalhes de produto, secção de informações

Os vídeos de produtos estão disponíveis na secção de vídeos, sendo que, cada uma contém o nome e o vídeo. Estes são demonstrados através da utilização de um elemento *iframe*, este contém um *url* para o vídeo, mas após a tentativa de visualização deste *url*, constatou-se que não é possível obter o vídeo a partir deste. Sendo assim, foi investigada a plataforma *vimeo*. Esta, contém todos os vídeos de produtos, pelo que, para cada um é gerado um *id* único. Este vídeo poderá ser acedido através do *url* geral da plataforma seguido do *id*. O *id*, está colocado no elemento *iframe*, pelo que, é obtido e acrescentado ao *url* da plataforma, o que permite guardar todos os vídeos de produtos.

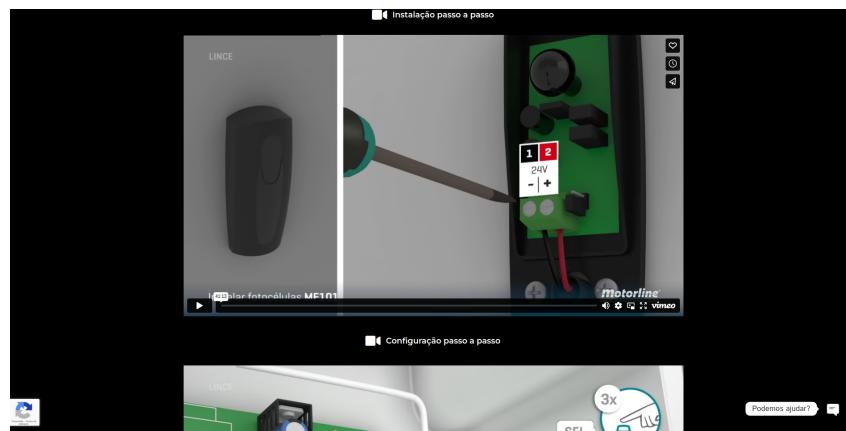


Figura 1.7: Página de detalhes de produto, secção de videos

### 1.1.1.1 Implementação no website

Logo que se verificou que pelo menos 80% dos produtos totais eram obtidos, decidiu-se testar no *website*. Para isto, foi utilizada a biblioteca *requests*, com a qual é realizado um pedido *GET* a cada *url* necessário para obter a página *web*. Assim que o código foi corrido e a resposta analisada compreendeu-se que o *website* bloqueia este tipo de solução como é indicado na figura 1.8

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access this resource.</p>
<p>Additionally, a 403 Forbidden
error was encountered while trying to use an ErrorDocument to handle the request.</p>
</body></html>
```

Figura 1.8: Resposta obtida aquando o pedido ao *url* da página *web*

Através da resposta obtida, foi compreendida a necessidade de alteração da abordagem, visto que, a anterior não poderia ser utilizada. Opcionalmente seria possível seguir a abordagem de simular a ação humana através da abertura de um navegador programaticamente e pesquisar pelo *url* desejado.

Após uma investigação descobriu-se que existem ferramentas que permitem controlar o dispositivo onde correm, mas estas impedem a utilização enquanto se encontram a correr. Também, foram descobertas ferramentas que apenas recebem o navegador a utilizar e abrem uma nova janela deste para realizar a pesquisa. O processo de obter os produtos é demorado, pelo que, optou-se pela segunda opção, uma vez que, seria possível continuar com o trabalho em paralelo com o processo de obter dados. Sendo assim, a ferramenta mais recomendada para realizar esta operação é a biblioteca *selenium*.

Com a utilização da biblioteca *selenium* indicou-se qual o navegador a utilizar. Neste caso foi escolhido o *chrome*, dado que, encontrava-se instalado no dispositivo. Após ser indicado qual o navegador a utilizar, é necessário para cada página referir qual o elemento a esperar que carregue, pois, por vezes existem elementos que demoram mais tempo a carregar do que a página. A página carregada poderá não conter o elemento a obter, pelo que foi implementado um tempo de espera máximo de 5 segundos, assim que este tempo expira, a operação é cancelada e o *url* é adicionado à lista de *urls* com erros.

### 1.1.1.2 Melhoria de implementação

Depois de completo o processo, foi decidido melhorar a implementação com a resolução dos erros encontrados em *urls* específicos. Para perceber exatamente quais os *urls* que possuem erros, foram direcionados os dados obtidos para um ficheiro *JSON*. Sendo assim, os *urls* com erros eram os indicados na figura 1.9.

```
"misses": [
    "https://motorline.pt/produto/acessorios-para-portoes-de-correr/",
    "https://motorline.pt/produto/acessorios-portoes-garagem/",
    "https://motorline.pt/produto/zuma/",
    "https://motorline.pt/produto/sigma-x/",
    "https://motorline.pt/produto/acessorios-barreira-eletromecanica/",
    "https://motorline.pt/produto/acessorios-para-portas-de-vidro/",
    "https://motorline.pt/produto/adaptador-tub/",
    "https://motorline.pt/produto/acessorios-motores-enrolar/",
    "https://motorline.pt/produto/cortina-corta-fogo-flama/",
    "https://motorline.pt/produto/acessorios-para-toldos/",
    "https://motorline.pt/produto/mbn25/",
    "https://motorline.pt/produto/acessorios-controlo-de-acessos/",
    "https://motorline.pt/produto/stop/",
    "https://motorline.pt/produto/acessorios-fotocelulas/"
]
```

Figura 1.9:Urls com erro primeira interação

Posteriormente a uma primeira análise percebeu-se que grande maioria os erros provêm de *urls* de acessórios de produtos, isto deve-se ao facto destes encontrarem-se na página de produtos de subcategoria e serem tratados como um produto, sendo assim, sempre que se trata de um url de acessórios seria necessário correr código para obter dados de acessórios ao invés de detalhes de produtos. Deste modo, compreendeu-se que nos *urls* a palavra accessórios está sempre contida, o que permite que sempre que esta é detetada num *url*, seja corrido o código referente a obter de acessórios. Para desenvolver este código foi primeiramente analisada a página de acessórios de produtos(Figura 1.10), esta contém para cada acessório um elemento do tipo artigo o qual detém uma imagem, titulo e descrição. Esta descrição por vezes possui *urls* para os produtos aos quais este acessório se refere. Sempre que estes *urls* são detetados, os nomes dos produtos são guardados para futuramente realizar a ligação entre os acessórios e os produtos, dado que, não existem produtos com nomes iguais.



Figura 1.10: Exemplo de página de acessórios

Na iteração seguinte determinou-se que a quantidade de *urls* com erros diminuiu, mas mesmo assim existiam produtos com erro, pelo que, estes foram analisados e percebeu-se que o erro ocorria, visto que, por vezes as páginas não continham os vídeos ou imagens de documentação. Para resolver este problema, o código foi alterado para somente obter estes dados se os elementos existirem na página. Após correr novamente o código foi compreendido que a quantidade de falhas obtidas diminuiu drasticamente (Figura 1.11), mas mesmo assim, ainda existiam quatro falhas a ocorrer e após uma análise foi determinado que estas ocorriam devido a uma página de subcategoria de produtos conter um serviço, um produto conter uma página de detalhes de produto com sub produtos, existir uma página de adaptadores de produtos e um produto conter uma página de detalhes diferente das demais.

```
"misses": [
    "https://motorline.pt/produtos/electronica-e-controlo/casa-inteligente/",
    "https://motorline.pt/produto/zuma/",
    "https://motorline.pt/produto/adaptador-tub/",
    "https://motorline.pt/produto/cortina-corta-fogo-flama/"
]
```

Figura 1.11: Indicação das páginas com falha

A página de adaptadores de produtos segue uma estrutura similar à dos acessórios, pelo que foi a primeira a ser abordada e resolvida através do código de obter detalhes de acessórios. Neste sentido este foi alterado para executar sempre que a palavra acessórios ou adaptadores encontra-se no *url*.

A resolução do problema de existirem serviços e subprodutos implica uma alteração no diagrama de entidade relação, pelo que, o problema do produto que contém uma página de detalhes diferente das demais foi resolvido primeiro. Este produto para além da dificuldade de ser uma página completamente diferente, as informações encontram-se espalhadas (Figura 1.12), o que leva a que estas tenham de ser combinadas para construir os detalhes do produto. Após obter-se estes dados foi determinado que as imagens do produto têm dados escritos que não se encontram na imagem original. Para a solução deste problema o cliente recomendou obter com *screenshots* e guardar num servidor de imagens.



Trata-se de uma cortina construída de forma compacta em perfis lacados, garantindo assim um maior aproveitamento do vão de passagem, sem alterar a estética do espaço. Rigorosamente testada, esta passou com distinção nos seguintes testes CE.

CERTIFICAÇÃO E QUALIDADE

Podemos ajudar?

Figura 1.12: Exemplo de página de produto incomum

O problema de existirem produtos com subprodutos foi resolvido através da alteração da base de dados, à qual foi adicionada uma nova ligação sobre si mesma na tabela produtos, o que permite que um subproduto possua uma ligação com produto principal. Após ser feita esta alteração, o código para obter os dados do catálogo foi alterado. Em primeiro lugar, foram obtidos todos os dados do produto principal e de seguida os dados específicos a cada subproduto, o que leva a que a organização do produto principal seja igual aos restantes, mas, com um dado extra com os subprodutos.

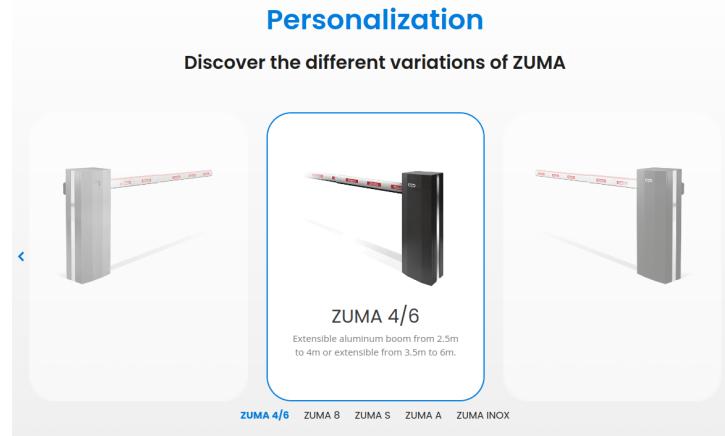


Figura 1.13: Exemplo de página de produto com subprodutos

O último problema a solucionar é a existência de serviços, pelo que, iniciou-se pela análise deste tipo de produto. Este possui descrição e imagens como os produtos, mas também vídeos direcionados a plataformas diferentes, produtos do serviço, registo de atualizações do serviço, planos de pagamento e cada plano contém diversas ofertas. Através da estrutura de um serviço foi adicionado à base de dados as tabelas de serviço, planos do serviço, ofertas dos planos, vídeos do serviço, informações de atualizações dos serviços, imagens do serviço e a ligação à tabela produtos, o que permite a identificação dos produtos do serviço. Aqui, foi identificada a necessidade de manter guardado os *links* para todas as imagens e vídeos na própria base de dados, visto que, existem imagens que não sabem ainda qual é o *id* do produto referente, pelo que não seria possível no futuro obtê-las sem alteração manual, o que levou a que estas tabelas também existam para os produtos. De seguida compreendeu-se que existem dados que não são necessários na descrição, o cliente recomendou guardar estes dados como *screenshots*.

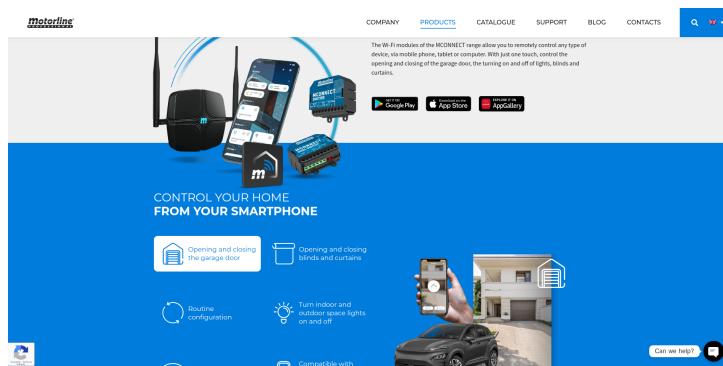


Figura 1.14: Exemplo de página de serviço

### 1.1.1.3 Diagrama de base de dados final

Após as alterações mencionadas à base de dados, o seu diagrama final encontra-se na Figura 1.15

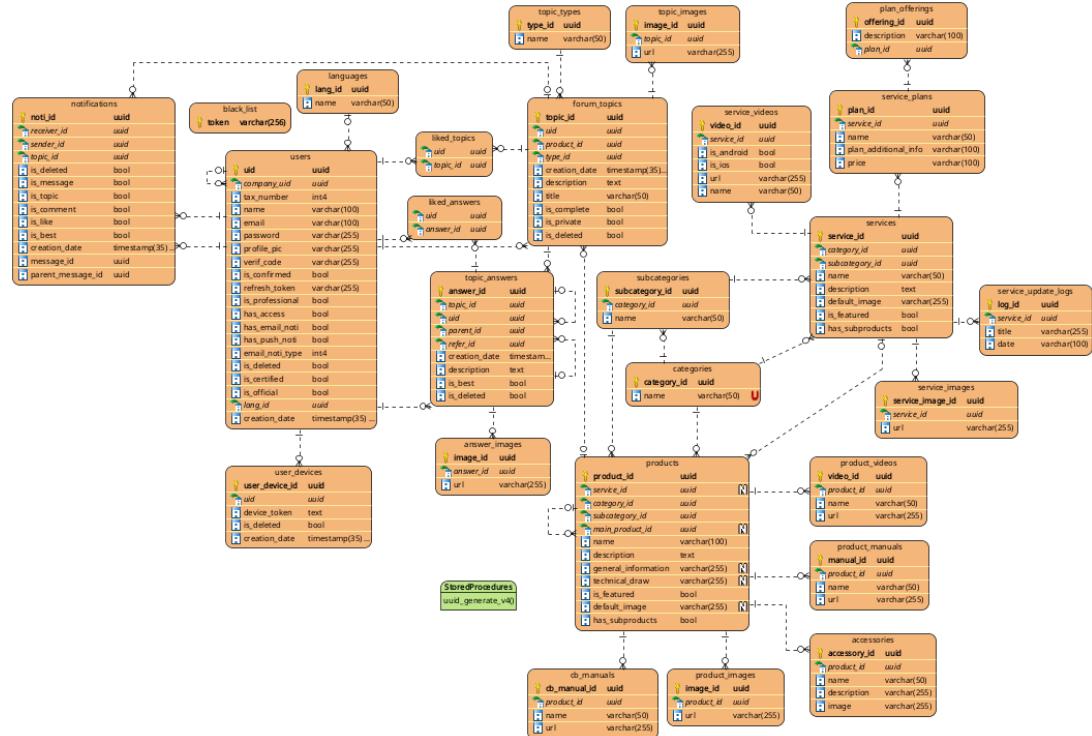


Figura 1.15: atualização da base de dados

### 1.1.1.4 Armazenamento de dados

Após obter-se os dados dos produtos, é necessário guardar na base de dados para serem disponibilizados ao *backend*. Para realizar esta operação existiam duas opções, criar um serviço para inserir produtos e realizar um pedido a este serviço, ou então, conectar diretamente o *web scraper* à base de dados. Não seria de grande interesse conectar diretamente à base de dados, pelo que, foi decidido criar um serviço que recebe um produto e o insere. O grande problema que surgiu com esta solução é que os pedidos ocorrem de forma sequencial, mas com pouco tempo de espera, o que levou a que o limite máximo de conexões com a base de dados fosse extrapolada. Isto acontece porque para cada serviço que recebe uma chamada é desenvolvida uma nova conexão à base de dados, todas as operações são realizadas e por fim a conexão é terminada, mas enquanto estas operações estão a decorrer, o servidor poderá receber mais pedidos, o que leva a que mais conexões sejam criadas, o que atinge assim rapidamente o limite de conexões da base de dados. Como solução para este problema foi acrescentado uma espera de 0.5 segundos a cada pedido. A inserção de serviços decorreu com o mesmo processo.

A inserção de categorias decorre através do envio das categorias a inserir num *array*, o que leva a que estas sejam inseridas todas num pedido. As subcategorias, visto que, não se sabe o *id* da categoria referente, foi utilizado o nome da categoria, pois este é único, sendo assim, é enviado o nome da categoria e as subcategorias referentes. Todas são inseridas com a referência para a sua categoria.

## 1.2 Backend

Após o desenvolvimento do *webscraper*, procedeu-se com o desenvolvimento do suporte *backend* do projeto.

### 1.2.1 Organização do projeto

Antes de iniciar a implementação definiu-se a estrutura do projeto a seguir. *MVC* foi a estrutura escolhida, uma vez que, é a mais comum e bem estabelecida. Sendo assim, a organização do projeto seguiu a seguinte estrutura:

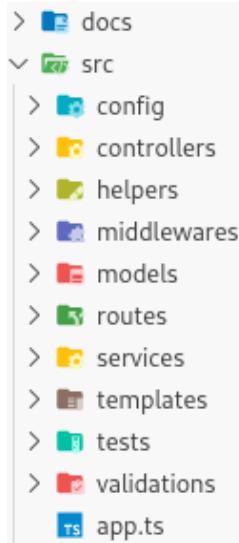


Figura 1.16: Exemplo de página de produto incomum

- **docs** - Documentação gerada;
- **src** - Base de todo o projeto;
- **config** - Ficheiros de configuração do projeto;
- **controllers** - Controladores para cada pedido;
- **helpers** - Ficheiros com funções gerais utilizadas regularmente;
- **middlewares** - Ficheiros com os middlewares da api;
- **models** - Classes criadas para representação de base de dados e outras entidades;
- **routes** - Rotas existentes;
- **services** - Serviços para cada pedido;
- **templates** - Templates de *email* a serem enviados;
- **tests** - Testes de código realizados;
- **validations** - Validações a realizar do modelo de negócio e dos dados;
- **app** - Ficheiro de início do projeto;

### 1.2.2 Definição de rotas base

Após ser definida a estrutura do projeto foram estruturadas as rotas base a existir, estas referem-se a cada ator. Para uma melhor organização das rotas e da aplicação de regras, foram definidos 3 *routers*, *user*, para utilizadores sem sessão, *professional*, para técnicos, e *company* para empresas. Para indicar qual o *router* a utilizar em cada pedido foi definido que:

- **http://baseurl:port/professional** - Encaminhar para *router* de técnicos;
- **http://baseurl:port/company** - Encaminhar para *router* de empresas;
- **Restantes** - Encaminhar para *router* de utilizadores;

### 1.2.3 Middlewares

Um *middleware* comporta-se como uma ligação entre duas partes e permite executar código.

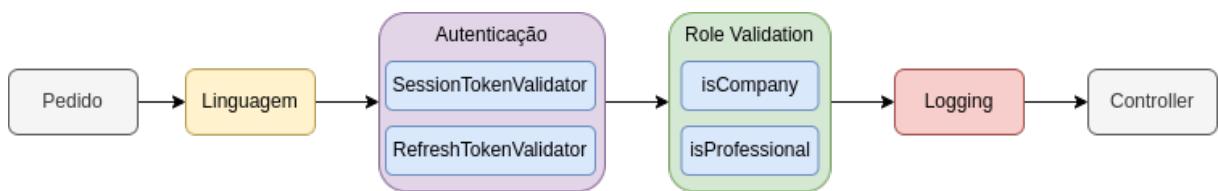


Figura 1.17: Comportamento de middlewares

#### 1.2.3.1 Idioma de Comunicação

O bem essencial para uma boa comunicação entre duas partes, é a utilização da mesma linguagem, pelo que, foi necessário perceber qual a linguagem a utilizar quando se responde a um pedido. Para este fim, foi desenvolvido um *middleware*, cujo objetivo é verificar se existe a chave *language* no cabeçalho do pedido. Caso esta exista, é obtida a linguagem e guardada nas variáveis locais do pedido. Na eventualidade desta não existir, a aplicação dará uma resposta em português por omissão. Este valor poderá ser futuramente alterado de forma simples.

### 1.2.3.2 Autenticação

A autenticação dos utilizadores foi implementada através de *Json Web Token*. Este tipo de autenticação, tem por base a utilização de *tokens* com tempo de expiração, o que significa que enquanto o *token* estiver válido, o utilizador poderá realizar pedidos. Assim que o *token* expirar, este terá de se autenticar novamente para obter um novo *token*. A utilização de *tokens* permite assegurar que os pedidos são realizados com *tokens*, gerados pela *API*, através de uma chave de assinatura de *token*, o que impede a utilização de *tokens* gerados por utilizadores.

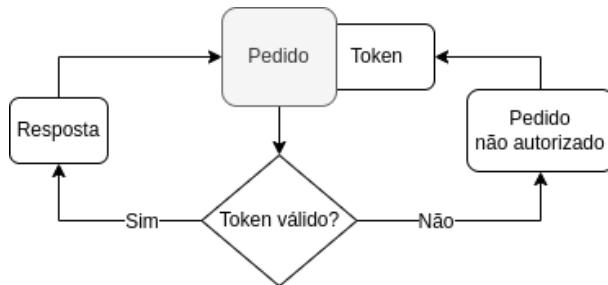


Figura 1.18: Utilização de tokens

A maior valência da utilização da técnica de autenticação mencionada anteriormente é a segurança. Isto acontece porque estes *tokens* têm geralmente uma duração muito curta, como por exemplo, quinze minutos. Sempre que um *token* de sessão expira o utilizador necessita realizar novamente o *login*, o que poderá tornar a utilização da aplicação imprática.

A solução deste problema sem a perda de segurança significativa veio pelo meio da utilização de *tokens* de duração maior em conjunto com os *tokens* de duração curta. Enquanto o *token* de grande duração se encontrar válido, novos *tokens* de curta duração são gerados para o utilizador, o que leva a que o utilizador nunca perca a sua sessão. Estes *tokens* de grande duração têm por nome *refresh tokens* e os *tokens* de curta duração têm por nome *session tokens*. O utilizador sempre que termina a sessão o *refresh token* deverá ser apagado.

Sempre que um utilizador realiza um pedido, o seu *token* de sessão deverá ser validado. Caso este seja válido, o seu *refresh token* deverá ser validado e apenas após esta verificação o utilizador estará autenticado. Na eventualidade de o *token* de sessão ou de *refresh* estiverem expirados, este estará sem autorização para realizar o pedido, mas, poderá solicitar um novo *token* de sessão enquanto o seu *refresh token* estiver válido. Isto acontece sem realizar novamente o *login* e sem o utilizador perceber.

Além das funcionalidades atrás mencionadas é possível também associar dados em formato *JSON* a um *token*. Esta funcionalidade foi utilizada para enviar o *id* e o cargo do utilizador a qual pertence este *token*.

### 1.2.3.3 Validação de *Role*

Com a finalidade de garantir que apenas utilizadores com cargos suficientes conseguem realizar as ações regradas, foi desenvolvido um *middleware* que valida se o utilizador que realizou o pedido tem permissões para o mesmo. Este *middleware*, interliga-se com o *middleware* anterior, pois, o cargo do utilizador em questão é enviado no *token*, pelo que, é obtido e realizada uma comparação com o cargo desejado. Para isto foram criados dois *middlewares*s diferentes, um valida o cargo das empresas e o outro o cargo dos técnicos. As empresas podem realizar operações de técnicos, dado que, no *middleware* de técnicos é verificado se o *token* corresponde a um utilizador empresa, ou a um utilizador técnico. Já no *middleware* de validação de empresa é verificado se o utilizador tem cargo de empresa.

### 1.2.3.4 Logging de erros

A identificação dos erros no servidor é difícil, uma vez que, os erros são tratados e os dados referentes não são guardados ou utilizados. Para resolver este problema foi determinado que sempre que um erro que não é customizado é detetado é registado um *log*, este contém informações sobre o pedido como data e hora, dados recebidos, a descrição original do erro e serviço referente.

Para implementar esta solução foi em primeiro lugar criado um *middleware* que sempre que deteta erros dentro do serviço executa um método. Este método, por sua vez, obtém os dados referentes ao pedido realizado, sendo estes a data e hora, os dados recebidos e o serviço pedido. Após se obter os dados do pedido é obtida a mensagem do erro e estes dados acrescentados numa nova linha no ficheiro de erros do servidor.

### 1.2.3.5 Logging de pedidos com morgan

Para realizar o *logging* de pedidos a serviços foi utilizado o *morgan*. Esta ferramenta precisa da indicação do ficheiro onde escrever os *logs*, o que leva à utilização de um ficheiro chamado *log*. O *morgan*, também precisa da indicação do tipo de *log* a realizar. Estes tipos são indicados pela ferramenta, o utilizado foi o *combined*, este é o tipo de *log* mais completo, visto que, é o que obtém mais dados, sendo estes, a data e hora do pedido, o tipo, o serviço, os dados recebidos, a resposta devolvida e também a descrição do sistema utilizado.

### 1.2.4 Controllers

Assim que um pedido consegue ultrapassar todos os *middlewares* sem ser impedido, este é redirecionado para um *controller*.

#### 1.2.4.1 Estruturação dos controllers

Para evitar a variação de código destes *controllers* em termos de estrutura, foi decidido desenhar uma estrutura de *controller* e aplicar perante o demais código. Esta segue as seguintes etapas:

1. Obter dados do pedido
2. Validar se os dados obrigatórios são obtidos
3. Validar o pedido perante o modelo de negócio
4. Executar a lógica do pedido
5. Formular a resposta e enviar
6. Em caso de erro este deverá ser capturado e processado para enviar um erro para o utilizador

Esta estrutura será sempre aplicada, pois, foram utilizados *snippets* de código que permitem criar um modelo de estrutura, apenas é necessário escrever a palavra-chave e toda a estrutura é aplicada, pelo que, é necessário de seguida efetuar as alterações perante o contexto.

#### 1.2.4.2 Execução da lógica de negócio

A execução da lógica de negócio passa por direcionar os dados para a ação correta. Esta ação geralmente resulta numa operação de base de dados. Inicialmente foi desenvolvida toda a validação de código e todas as operações de base de dados diretamente na execução da lógica de negócio. Após uma revisão desta organização de código com o professor orientador, foi decidido separar estas funcionalidades. Daí surgiu a componente de validação de dados, a de operações de base de dados e a de lógica de negócio, que implementa a componente de operações de base de dados. Sendo assim, para evitar que estas operações sobre a base de dados estejam em conjunto com o direcionamento dos dados, foram criados modelos para cada tabela. Cada modelo contém um conjunto de operações sobre a tabela correspondente. Estas operações, estão contidas sobre métodos que podem receber dados para executar na operação e devolver a resposta da mesma.

### 1.2.4.3 Validação dos dados

A validação dos dados é necessária para evitar erros a nível de servidor com dados em falta e também para aplicar as regras de negócio. Para realizar estas validações, é necessário em primeiro lugar verificar se todos os dados são recebidos, de seguida, são enviados para um *validator*. O *validator* executa todas as verificações necessárias a nível de regras de negócio, na possibilidade de alguma regra não ser cumprida, é atirado um erro.

### 1.2.4.4 Formulação da resposta

Como mencionado anteriormente, o bem mais importante numa boa comunicação é a utilização da mesma linguagem, pelo que, a resposta do servidor deverá utilizar a linguagem indicada pelo utilizador. Para realizar esta tradução foi utilizado o mesmo conceito que se usa em *Android*. Neste é desenvolvido um ficheiro que contém um conjunto de chaves e a cada corresponde um texto. Cada tradução tem de conter estas chaves para que seja possível obter o texto correto para cada uma. Sendo assim, foi utilizado um ficheiro *JSON* que dispõem das chaves das linguagens suportadas. A cada uma corresponde a um conjunto de outras chaves que com todas as traduções necessárias. Esta abordagem permite que de forma fácil, futuramente seja possível adicionar outras linguagens ao servidor.

```
"pt/PT": {
    "unauthorized_request": "Pedido não autorizado",
    "jwt_key_missing": "Chave jwt em falta",
    "undefined_token": "Token não definido",
    "invalid_language": "Linguagem inválida",
    "invalid_request": "Pedido inválido",
```

(a) Página de login

```
"en/UK": {
    "unauthorized_request": "Unauthorized Request",
    "jwt_key_missing": "Jwt key missing",
    "undefined_token": "Token undefined",
    "invalid_language": "Invalid communication language",
    "invalid_request": "Invalid Request", ..
```

(b) Página de registo

Figura 1.19: Autenticação - Login e Registo

O suporte a este ficheiro foi elaborado com uma operação que recebe a chave e a linguagem desejada. Este devolve o texto traduzido, dado que, na formulação da resposta a operação é executada com a indicação da chave do texto a enviar e a linguagem desejada, sendo que este é devolvido para o utilizador.

### 1.2.4.5 Processamento de erros

Como não é de interesse enviar erros do próprio servidor para o utilizador, foi decidido controla-los. Para isso foi concebido um erro customizado com base no erro da própria linguagem. Este recebe por parâmetro o código da tradução da mensagem. Esta abordagem permite evitar que sempre que um erro é lançado o sistema pare. Contudo, sempre que um erro é lançado pela base de dados, erro de código ou de biblioteca, o original é chamado, o que levou a que sempre que é detetado é devolvida uma mensagem de "erro de servidor", isto evita a que dados sensíveis e desnecessários para o utilizador sejam devolvidos.

### 1.2.5 Envio de *emails*

Como mencionado na apresentação da ferramenta tabular, não é permitido a utilização de acentuação na escrita de *emails*, pelo que, primeiramente todos os problemas foram resolvidos. Para permitir que os dados dos *emails* sejam personalizáveis, como por exemplo, dados de utilizador e *link* para clicar, estes *emails* são colocados em métodos que recebem por parâmetro os dados para serem colocados dentro do *html* do *email*, este método, por fim, devolve em *string* o conteúdo a enviar.

Para o envio de *emails* é necessário um servidor e um serviço de envio. Para vias de teste foi utilizado um servidor gratuito de *email* hospedado por *Mailjet*. Após a fase de testes foi alterado para o servidor de *email* da empresa, o que permitiu que seja identificado como empresa Motorline.

O serviço de envio de *emails* foi utilizado o *nodemailer*. A configuração do servidor de *emails* do *nodemailer* é realizada através de uma chamada ao objeto do servidor com a indicação das configurações que estão no ficheiro *.env*. Esta chamada ao servidor, por sua vez, devolve um objeto que fornece métodos para enviar *emails*, sendo este, criado e utilizado sempre que se deseja enviar *emails* no servidor.

Para evitar que sempre que se deseja enviar um *email* seja necessário indicar todos os dados de configuração do servidor, foi elaborado um método que cria e devolve o objeto do servidor sempre que necessário. Sendo assim, sempre que se deseja enviar um *email* é primeiramente obtido o objeto do servidor, de seguida é invocado o método para obter o conteúdo *html* do *email* e por fim, é utilizado o objeto do servidor para chamar o método de envio de *emails*, no qual, se terá de indicar o *email* do destinatário, o assunto e o conteúdo.

### 1.2.6 Agendamento de tarefas

Com a utilização da ferramenta *node-cron*, foi inicialmente programado para enviar o relatório de notificações, todos os dias, às 22 horas. Esta configuração foi realizada através da indicação da programação de horário de envio, para isso, é utilizada a estrutura segundo, minuto, hora, dia do mês, mês, dia da semana. Como o objetivo é enviar às 22 horas, os segundos e minutos foram indicados como 0 e as horas foram indicadas como 22, já o restante foi indicado com o símbolo "\*", que indica que o processo deverá ocorrer em todas as instâncias dos restantes valores, o que significa, todos os dias. A configuração final obtida foi "0 0 22 \* \* \*".

Quando o método do agendamento é invocado, são obtidos todos os utilizadores com a configuração de relatório de notificações ativa. Em primeiro lugar, para cada um destes, são obtidas todas as notificações do dia, de seguida é criado o objeto de servidor e invocado o método para obter o conteúdo de notificações através da indicação de todas as notificações do utilizador. Por fim, é enviado o *email* para o utilizador com todas as suas notificações e um *link* para aceder rapidamente ao ecrã de notificações.

### 1.2.7 Cifra de *passwords*

Aquando o registo de clientes é indicada a *password*, pelo que, esta deverá ser guardada sobre cifragem. Para isto, é utilizada a biblioteca *bcrypt*. No processo de registo, antes do envio dos dados para a base de dados, é invocado o método de cifra da *password* com a indicação do *salt* com valor de oito. Após a execução é obtida a *password* cifrada e enviada para a base de dados em conjunto com os restantes dados.

#### 1.2.7.1 Cifra de configurações do servidor

Para a cifra das configurações do servidor, foi executado o comando de cifra da biblioteca *secure-env*, com a indicação da *password*. Como resultado foi criado o ficheiro cifrado. O ficheiro original deverá ser eliminado ou, em caso de necessidade, guardado num local seguro para evitar que durante algum ataque seja obtido os dados do mesmo.

Após cifrar o ficheiro, o servidor foi configurado para quando inicia pedir ao utilizador para indicar a *password* do ficheiro. Para esta configuração foi utilizada a biblioteca *readline-sync* no modo de *password*. Quando o utilizador indica a *password*, na eventualidade de esta estar errada, o servidor irá falhar, pois a biblioteca de cifra tentará decifrar o ficheiro de configurações e falhará não concluindo o processo de inicialização. Caso contrário, este continuará o processo, no qual, o primeiro passo é a utilização da *password*, para decifrar o ficheiro de configurações e carregar as variáveis de ambiente.

### 1.2.8 Documentação Typedoc

Para a documentação de código foi utilizado *typedoc*. Durante a implementação do *typedoc* foi detetado que a categorização da documentação não estava funcional devido a um problema encontrado pelos programadores da ferramenta. Deste modo reduziu-se a versão para uma com a funcionalidade ativa, mas esta, não é compatível com a versão mais atualizada de *typescript*, pelo que, não foi possível explorar esta funcionalidade. Para contornar o problema foi explorada outra funcionalidade menos utilizada, esta permite converter qualquer documentação em módulos, estes podem ser categorizados, o problema é que é criado um modelo genérico do código, o qual não permite a fácil identificação das tipagens de *scripts*. Estes módulos permitem também a categorizar, o que leva a um maior nível de organização da documentação.

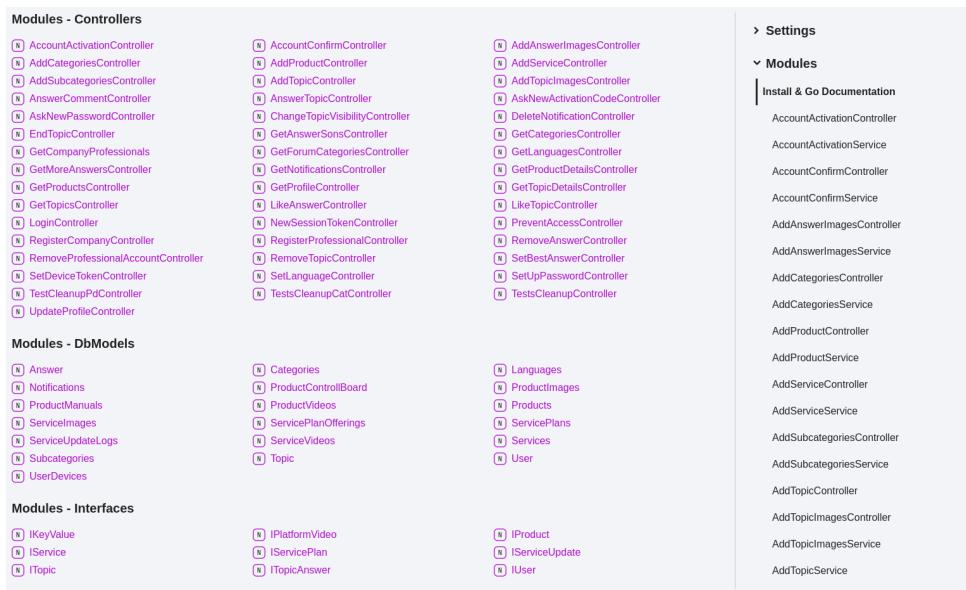


Figura 1.20: Documentação *typedoc*

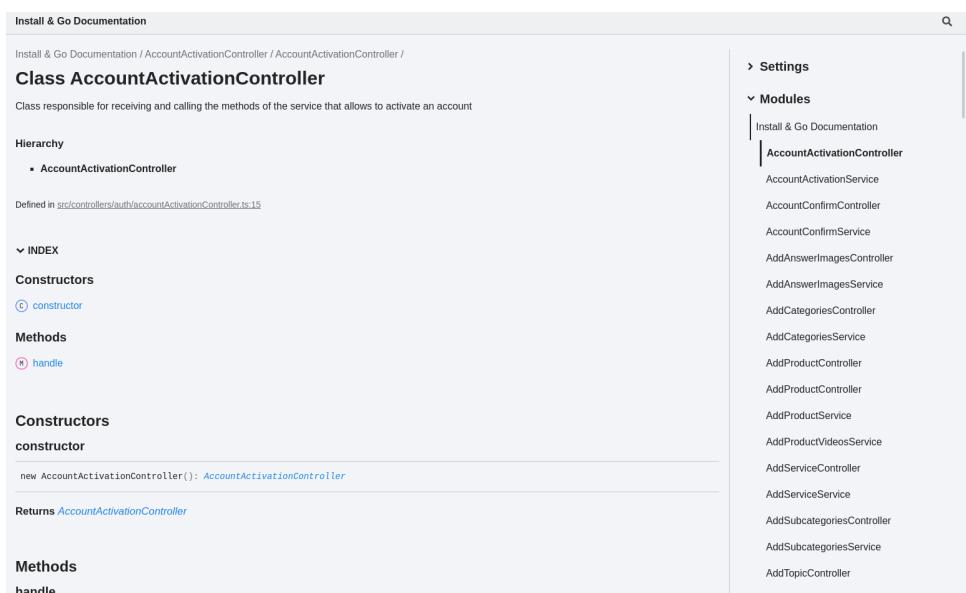


Figura 1.21: Documentação de classe *typedoc*

### 1.2.9 Documentação Swagger

Apesar do *swagger* disponibilizar a funcionalidade de gerar automaticamente documentação a partir de comentários de código, foram encontrados alguns problemas com esta funcionalidade, o que levou a que não fosse possível gerar a documentação. Portanto, optou-se por manter a documentação manualmente com o ficheiro *JSON*. Esta ferramenta oferece diversas funcionalidades como autenticação, definição de estruturas de dados para os serviços e exemplos de respostas. Estas funcionalidades foram exploradas o que gerou um bom suporte de documentação para qualquer utilizador.

The screenshot shows the Swagger UI interface for the 'Install&Go API'. At the top, it displays the title 'Install&Go API 1.0.0 OAS3' and a note 'Backend support for install&go app'. Below this, there's a 'Servers' dropdown set to 'http://82.48.213.162:7856/api/v1 - External docs' and an 'Authorize' button. The main area is divided into sections: 'Products', 'Services', and 'Categories'. Under 'Products', there are five items: 'GET /products get products' (blue), 'POST /products Add product' (green), 'GET /products/{productId} Get product details' (light blue), 'DELETE /tests/product/{productId}/cleanup Get product details' (red), and 'DELETE /tests/category/{catName}/{subName}/cleanup Get product details' (red). Under 'Services', there is one item: 'POST /services Add service' (green). Under 'Categories', there is one item: 'GET /categories Get categories' (light blue). Each item has a collapse/expand arrow icon to its right.

Figura 1.22: Documentação swagger

This screenshot provides a detailed view of the Swagger UI for a specific endpoint: `DELETE /tests/category/{catName}/{subName}/cleanup`. The top bar shows the method (`DELETE`), URL, and description ('Get product details'). Below this, the 'Parameters' section lists three required parameters: `catName` (path, type: string, example: 'catName'), `subName` (path, type: string, example: 'subName'), and `language` (header, type: string, examples: 'pt/PT', 'en/EN'). The 'Responses' section shows two possible outcomes: a successful `200 OK` response with a media type of `application/json` containing an example of 'Cleanup' (which is redacted in the screenshot) and an error `500 ERROR`. A 'Try it out' button is located in the top right corner of the main content area.

Figura 1.23: Exemplo de documentação de serviço

## 1.3 Frontend

Após o desenvolvimento do suporte *backend* prosseguiu-se para o desenvolvimento *frontend* do projeto.

### 1.3.1 Organização do projeto

Tal como o *backend*, o modelo a seguir no *frontend* foi o *MVC*.

Como recomendado de boas práticas de código limpo da *framework*, as cores do tema da aplicação foram colocadas num ficheiro separado para garantir a fácil troca do tema da aplicação. Outras aplicações de boas práticas de código limpo foram, sempre que possível, particionar o código das páginas em vários *widgets*, para deste modo, facilitar a navegação e também, a criação de *widgets* reutilizáveis que evitam a repetição de código e agilizam o desenvolvimento. Por fim, a estrutura do projeto foi organizada da seguinte forma:

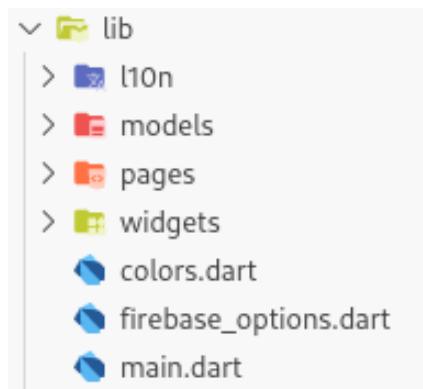


Figura 1.24: Organização do projeto

- **l10n** - Traduções da aplicação;
- **models** - Modelos de classes como *handlers*, *helpers*, *providers*, entre outros;
- **pages** - Páginas da aplicação;
- **widgets** - *Widgets* referentes às páginas;

### 1.3.2 Processo de aprendizagem

De forma a realizar a aprendizagem da ferramenta foi então procedido para a desenvolvedora da mesma, a *Google*, esta dispõe de uma lista de *workshops* e projetos a seguir para aprender as bases da ferramenta, sendo a lista a seguinte:

1. MDC-101 no Flutter: noções básicas dos componentes do Material Design | Google Codelabs
2. MDC-102 no Flutter: estrutura e layout do Material Design | Google Codelabs
3. MDC-103 Flutter: temas do Material Design com cores, formas, elevação e tipo | Google Codelabs
4. MDC-104 Flutter: componentes avançados do Material Design | Google Codelabs
5. Apps adaptáveis no Flutter | Google Codelabs

Após a realização destes *workshops*, foi possível interiorizar como funciona a base desta ferramenta e também encontrar fontes para pesquisa de *widgets* da comunidade a nível gráfico e funcional, estes provaram ser de grande auxílio no desenvolvimento da aplicação *frontend*.

### 1.3.3 Extensions

A linguagem de programação *dart*, assim como outras linguagens orientadas a objetos, permite alterar e adicionar métodos aos objetos base da linguagem. Para realizar estas ações foram criadas extensões do objeto, neste caso, uma extensão para o objeto *string* que permite capitalizar um texto.

### 1.3.4 Handlers

Os *handlers* são porções de código que possibilitam a execução de ações perante um evento, como por exemplo, a realização de uma ação num clique no ecrã. Neste caso, os *handlers* foram utilizados para detetar o estado da aplicação e realizar ações diante destes estados. Os estados da aplicação referem-se a se a aplicação se encontra em primeiro plano, segundo plano, a resumir ou então desligada. Com estes *handlers* é possível alterar o funcionamento da aplicação de acordo com estes estados, como por exemplo, tratar de notificações da aplicação.

### 1.3.5 Providers

Os *providers* são classes criadas para ajudar com comunicações externas, neste caso chamadas à *API*. Estes foram desenvolvidos conforme os diversos modelos de dados que se recebem, como por exemplo, uma tópico do fórum. Um *provider* oferece, na presença de um modelo, um conjunto de métodos para as diferentes chamadas necessárias à *API*. Como o exemplo anterior, numa tópico existem métodos para eliminar, adicionar, editar e obter dados, sendo que, cada método terá os seus requisitos do serviço que invoca.

Estes *providers* automaticamente detetam a linguagem da aplicação e realizam o pedido ao serviço com a utilização dessa linguagem.

### 1.3.6 Helpers

Os *helpers*, como o próprio nome indica, são classes que ajudam com a realização de tarefas. Neste caso, os *helpers* foram utilizados para o auxílio de mensagens de notificações. Estas mensagens deveriam conter o nome do utilizador e a ação, traduzida na linguagem da aplicação, pelo que, foi criada a classe de *helper* de notificação, que contém um método estático para obter a mensagem de uma notificação segundo a sua ação.

### 1.3.7 Gestão de utilizadores

Um dos requisitos da aplicação é que apenas as empresas têm a possibilidade de registar-se, pelo que, apenas estas poderão registar os seus técnicos. Dado este requisito foi elaborada a página de gestão de utilizadores, apenas acessível às empresas. Nesta página poderão pesquisar pelos seus técnicos ou registar novos técnicos, sendo que, têm a obrigatoriedade de indicar o nome, *email* e o tipo de técnico. Outras ações que as empresas poderão realizar, é a visualização do perfil de um técnico com a possibilidade de bloquear o acesso ou até apagar a conta do mesmo, sendo que, esta ação é irreversível.



(a) Aviso de impedir acesso à conta

(b) Aviso de remover conta

Sempre que um utilizador sem acesso ou com a conta apagada efetua o *login*, receberá uma mensagem de erro que menciona que não possui acesso à conta impedindo a continuação do processo.

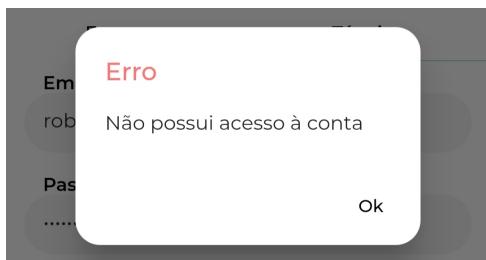
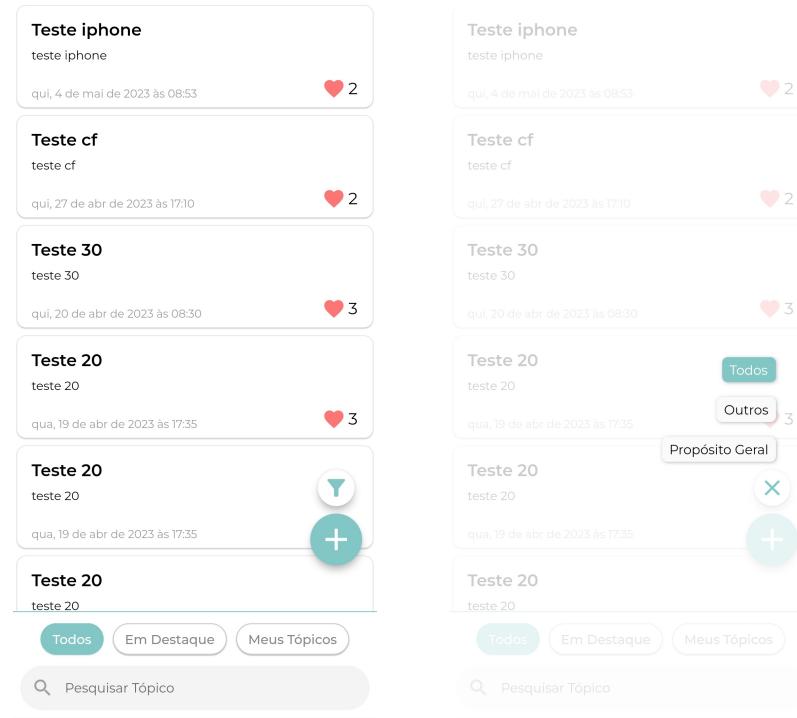


Figura 1.26: Aviso de login a conta sem acesso

### 1.3.8 Fórum

O desenvolvimento da página do fórum trouxe diversas dificuldades, entre elas, a gestão de filtros, pesquisas, a mostragem de tópicos e atualização das mesmas.



(a) Página de forum

(b) Filtragem de tipo

### 1.3.8.1 Filtragem de tópicos

O grande problema com a filtragem dos tópicos é que existem 3 tipos de filtros, o de categoria de tópico, o de tipo de tópico e o de pesquisa.

Se algum filtro for alterado, os seguintes deverão ser novamente executados para garantir que todos permanecem aplicados. Inicialmente, este tipo de filtragem não era realizado, o que levava a diversos problemas, tais como, na realização de uma pesquisa, esta não era efetuada sobre os tópicos filtradas, o que levava a que a pesquisa fosse efetuada por todas os tópicos.

Outro problema detetado foi a troca de categorias, que por vezes, os filtros adicionavam-se, o que levava a que estes não apresentassem tópicos.

Sendo assim, foram criados métodos para auxílio na filtragem, tais como, uma prioridade, onde cada método invoca o método seguinte de forma encadeada. Primeiramente aplica-se o filtro de categoria, de seguida este envia o resultado para o método de filtragem por tipo e por fim, se existir algum tipo de pesquisa, os tópicos provenientes do método anterior serão filtrados.

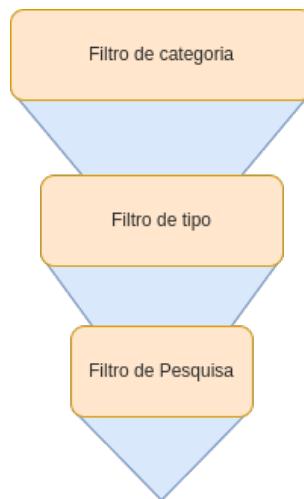


Figura 1.28: Filtragem do forum

### 1.3.8.2 Carregamento de tópicos

Inicialmente o carregamento de tópicos fazia-se por inteiro, desde carregamento de todos os tópicos, até todos os dados dos mesmos, visto que, não existiam muitos tópicos este não seria um problema para a *API*, mas, conforme os testes foram realizados a quantidade de tópicos existentes foi gradualmente aumentando, pelo que, foi possível visualizar o tempo de demora da resposta do servidor a aumentar, assim como, o desempenho da aplicação no fórum a piorar.

A resolução deste problema proveio com uma técnica de *sliding window*, na qual os tópicos mantêm-se carregados, sendo que, a própria *framework* consegue através da lista retirar de renderização os tópicos que o utilizador não consegue ver.

Esta solução foi implementada através da utilização de três valores, quantidade de tópicos a obter, índice inicial e data do primeiro tópico. O valor de quantidade de tópicos a obter, inicialmente dez, permite limitar a quantidade de tópicos que a *API* irá processar, o que reduz o tempo de resposta. O índice inicial, indica qual o índice do primeiro elemento que se deseja obter da lista. A data do primeiro tópico, mantém uma referência temporal para obter tópicos, o que garante que a lista que está a ser a visualizada é sempre a mesma.

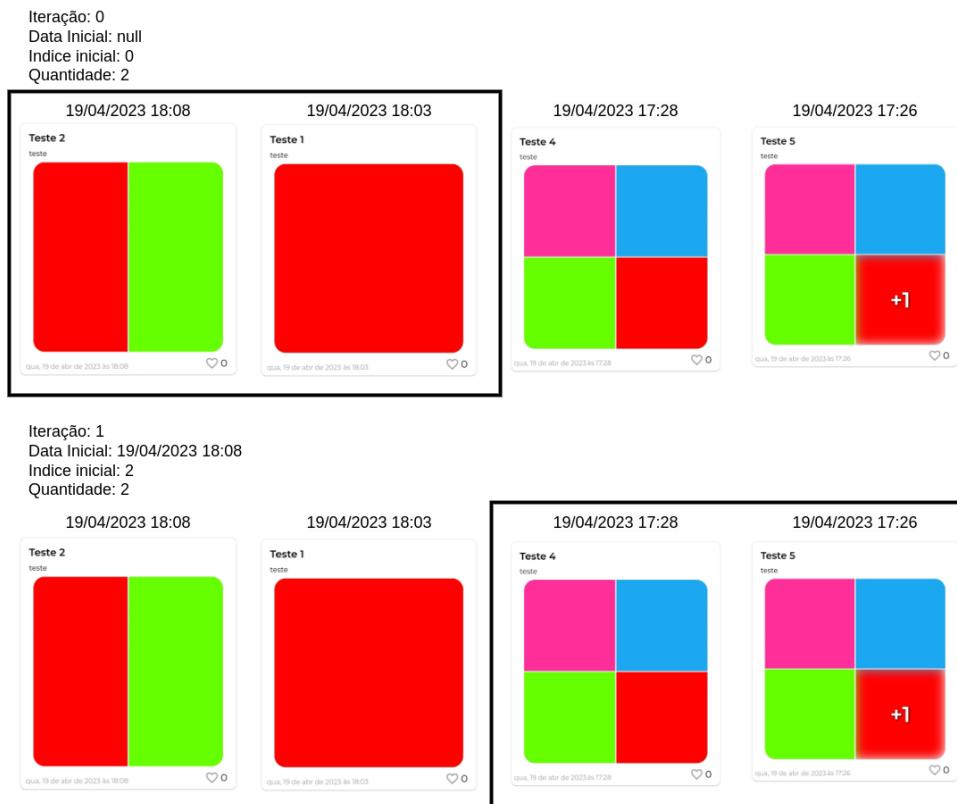


Figura 1.29: Carregamento de tópicos

Também foi reduzida a quantidade de dados carregados por cada tópico, pelo que, apenas os comentários diretos à tópico são carregados e não as respostas a estes, o que contribuiu para a melhoria do desempenho.

Por fim, sempre que o utilizador alcança o fim da lista de tópicos este poderá deslizar para carregar mais tópicos.

### 1.3.8.3 Detalhes de tópico

A página de detalhes de tópico sofreu os mesmos problemas que a página anterior, o que levou à necessidade de aplicar a mesma solução sobre os comentários de tópico e sobre as respostas aos mesmos. Sendo assim são carregados os primeiros dez comentários e por fim, demonstrado ao utilizador quantos mais existem que poderá carregar, sendo carregados dez de cada vez.

Estes comentários podem conter respostas, sendo que estas conseguem ser carregadas também dez de cada vez e o utilizador consegue esconder ou mostrar estas.

Outro problema que surgiu no desenvolvimento da página de detalhes de tópico foi o destaque de um comentário. Este foi uma grande dificuldade, pois, com a nova implementação as mensagens não estão carregadas no momento do destaque da mensagem, pelo que, é necessário procurar a mesma nos comentários carregados, o que expande as respostas do comentário que contêm a resposta a destacar.

O destacamento das mensagens também continha um erro. Sempre que algo no ecrã é atualizado, este recarregava a animação de destaque, como solução este código foi movido para apenas ser executado no momento de inicialização do ecrã após todos os elementos se encontrarem devidamente carregados.

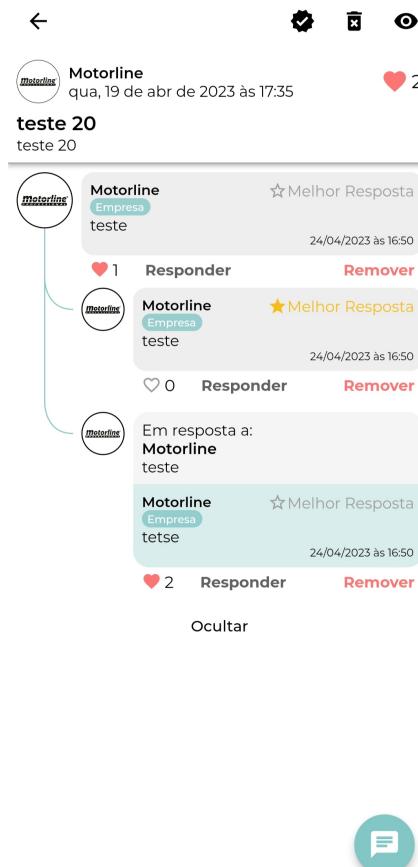


Figura 1.30: Destaque de mensagens

### 1.3.9 Firestore

A desenvolvedora do *Flutter* e do *Firebase* é a mesma, neste sentido, disponibilizou recursos que facilitam a utilização desta ferramenta pelo *Flutter*. Sendo assim, todas as imagens e vídeos de utilizadores, tópicos e comentários são guardados diretamente da aplicação para o *Firestore*, assim como, o acesso às mesmas é realizado diretamente.

Para permitir este tipo de acesso o *Firebase* disponibiliza uma ferramenta que possibilita, que através do terminal se realize a configuração da ligação entre o projeto e o servidor do *Firebase*, sendo que, no final apenas é necessário importar a biblioteca do serviço do *Firebase* que se deseja e invocar a classe do mesmo para serem realizadas ações.

Os ficheiros são organizados conforme o seu contexto. Para utilizadores, existe a pasta utilizadores, para tópicos, existe a pasta tópicos e para comentários, existe a pasta comentários.

A pasta utilizadores, como cada utilizador, apenas contém uma imagem, então são guardadas com o nome do *id* do utilizador e na eventualidade de já existir é substituída. No caso de tópicos e comentários, como podem conter várias imagens e vídeos, estes são guardados em pastas com os ficheiros referentes, que têm como nome os *id's* dos tópicos ou comentários.

The screenshot shows the Firebase Storage interface with the title "Storage". At the top, there are tabs for "Files", "Rules", "Usage", and "Extensões Novo". Below the tabs, there is a URL "gs://install-and-go.appspot.com" and a blue button "Fazer upload do arquivo" (Upload file). A table lists three folders: "Products/", "topics/", and "users/". Each folder has a checkbox, a name, a size of "-", a type of "Pasta", and a timestamp of "-".

	Name	Tamanho	Tipo	Última modificação
<input type="checkbox"/>	Products/	-	Pasta	-
<input type="checkbox"/>	topics/	-	Pasta	-
<input type="checkbox"/>	users/	-	Pasta	-

(a) Raiz do firestore

The screenshot shows the Firebase Storage interface with the title "Storage". At the top, there are tabs for "Files", "Rules", "Usage", and "Extensões Novo". Below the tabs, there is a URL "gs://install-and-go.appspot.com > topics" and a blue button "Fazer upload do arquivo" (Upload file). A table lists three subfolders under "topics": "1dddf308-9fd5-41ac-8d17-b2d04 10486fd/", "256c6ac4-7ce4-495c-a42a-97cd2 270f3e3/", and "2c77661f-ded5-488b-943c-d35e6 67b9194/". Each folder has a checkbox, a name, a size of "-", a type of "Pasta", and a timestamp of "-".

	Name	Tamanho	Tipo	Última modificação
<input type="checkbox"/>	1dddf308-9fd5-41ac-8d17-b2d04 10486fd/	-	Pasta	-
<input type="checkbox"/>	256c6ac4-7ce4-495c-a42a-97cd2 270f3e3/	-	Pasta	-
<input type="checkbox"/>	2c77661f-ded5-488b-943c-d35e6 67b9194/	-	Pasta	-

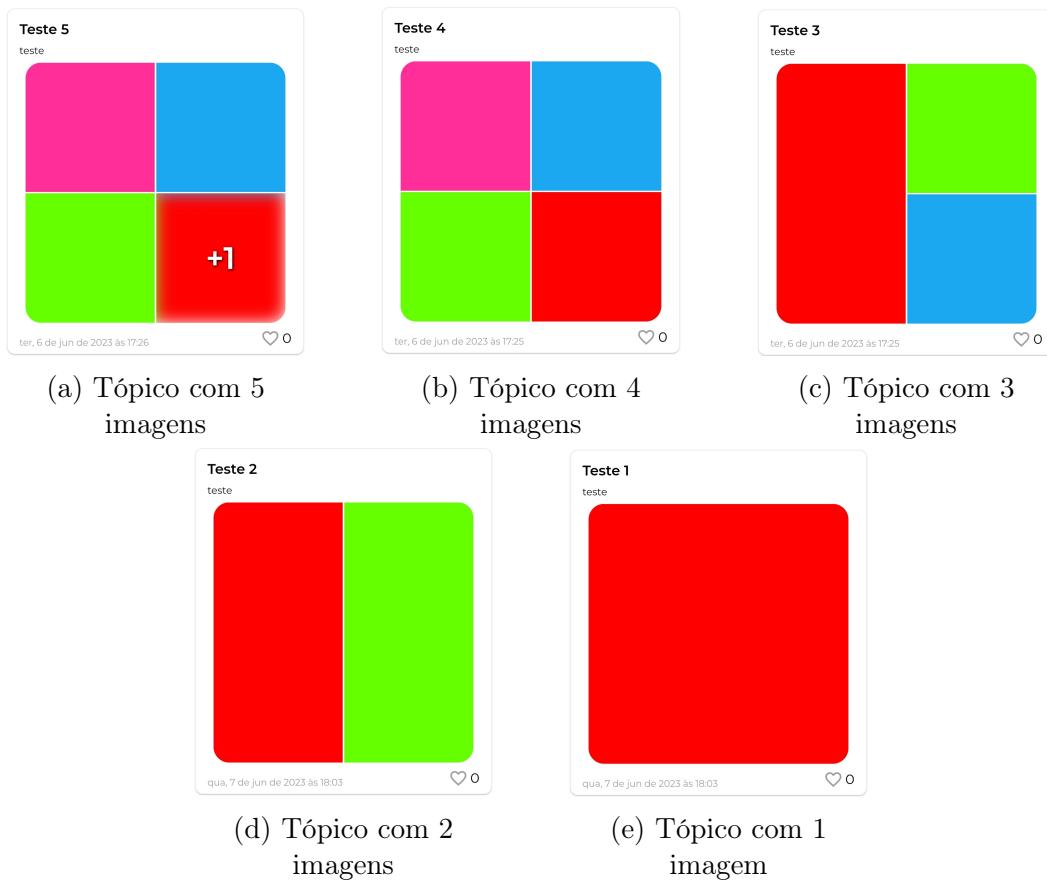
(b) Pasta topics do firestore

### 1.3.10 Apresentação de Imagens

A apresentação das imagens é definida em 2 níveis, o nível de pré-visualização, por exemplo, a miniatura da imagem de uma tópico no fórum e o nível do detalhe, onde é possível visualizar a imagem em ponto grande e realizar *zoom*.

Para a apresentação da miniatura da imagem foi decidido apresentar até quatro imagens, sendo que, acima de quatro imagens serão apenas demonstradas três, o que significa que a quarta imagem indicaria quantas mais existem para ser visualizadas.

Para a implementação da apresentação das miniaturas das imagens utilizou-se a biblioteca *staggered\_grid\_view*, que permite organizar imagens em grelha. Neste contexto desejava-se estruturar as imagens em diferentes aspectos e disposições, pelo que, esta biblioteca permite indicar quantas colunas e linhas existem na grelha ao criar o agrupamento de imagens. Deste modo, decidiu-se que quando são duas imagens, estas dividem a grelha, quando são três imagens, a primeira divide metade da grelha e as outras duas dividem a outra metade, quando são quatro ou mais, as quatro separam a grelha por igual. Quando existem mais do que quatro imagens, optou-se por colocar um filtro de desfoco sobre a última imagem e por cima desta quantas mais imagens existem para apresentar.



### 1.3.11 Apresentação de Imagens em carrossel

A apresentação das imagens deverá permitir que o utilizador visualize em ponto grande e realize diversas ações sobre estas. Para isso experimentaram-se diversas bibliotecas, mas nunca se alcançou o comportamento desejado, sendo assim, decidiu-se criar o próprio carrossel de imagens, sendo que, o próprio *Flutter* já disponibiliza um *widget* para tal.

O ponto de maior dificuldade para este processo foi a implementação de *zoom*, visto que, o *Flutter* não dispõe de *widgets* para tal. Por isso foi necessário primeiramente detetar gestos com o detetor de gestos da ferramenta e aplicado um *zoom* sobre o centro do gesto. Os gestos aceites foram o de pinça e o gesto de duplo clique.

O grande problema com esta solução é, uma vez que, é permitido um *scroll* horizontal de imagens, os gestos por vezes poderão não funcionar corretamente, principalmente o gesto de pinça que se efetuado na horizontal poderá resultar num *scroll*. Para resolver tal problema definiu-se que quando dois dedos são reconhecidos no ecrã, a navegação horizontal fica bloqueada e, assim que, estes são retirados, a navegação horizontal é ativada novamente.

### 1.3.12 Carregamento de Imagens

O carregamento de imagens do dispositivo poderá ser realizado por meio da seleção da galeria. Para realizar esta seleção, em primeiro lugar, foi testada a biblioteca *image\_picker*, contudo, esta biblioteca utiliza o seletor de ficheiros do dispositivo. Este seletor de ficheiros permite a seleção de qualquer tipo de ficheiros o que faz com que seja necessário um conjunto de outras verificações, para garantir que apenas as imagens são selecionadas. Isto levaria a uma possível perda do desempenho e da qualidade na experiência de utilização.

Sendo assim, de seguida foi testada a biblioteca *advanced\_image\_picker*, todavia, o problema desta biblioteca é que não permite selecionar vídeos. Posteriormente decidiu-se experimentar a biblioteca *wechat\_asset\_picker*. Esta cria uma página própria para seleção de imagens e vídeos diretamente da galeria. Também permite indicar o limite máximo de seleção e os tipos de ficheiros que o utilizador poderá selecionar, para que apenas, os tipos aceites sejam demonstrados. Por fim, o único ponto desvantajoso é que não é possível traduzir o botão de confirmação da seleção de ficheiros.

Após carregar os ficheiros para a memória, estes são enviados para o *firestorage*.

### 1.3.13 Vídeos

A maior dificuldade detetada nos vídeos foi a necessidade de um comportamento diferente nestes quando se encontram no ecrã de visualização de imagens e vídeos, visto que, ao contrário das imagens, os vídeos necessitam de um reprodutor, sendo sempre necessário verificar qual o tipo de ficheiro antes de carregar o *widget* do mesmo.

Para resolver o problema de utilizar um reprodutor testou-se uma biblioteca que permite a utilização do reprodutor nativo do dispositivo, ou seja, *Android* utilizaria o reprodutor do *Android* e *iOS* utilizaria o reprodutor de *iOS*. O grande problema com esta solução é que o reprodutor de *Android* tem os botões completamente brancos, sem nenhum tipo de fundo para os destacar, o que significa que se um vídeo branco for visualizado, o utilizador não conseguirá visualizar os botões do reprodutor.

Deste modo decidiu-se desenvolver um reprodutor próprio. Após a implementação de diversas funções como, pausar, resumir, avançar 5 segundos e recuar 5 segundos, esconder e apresentar os botões, existiam dois grandes problemas, demonstrar o vídeo em ponto grande, voltar para o mesmo tempo do vídeo em ponto pequeno e também o comportamento do reprodutor não ser completamente fluido.

Depois de uma vasta pesquisa compreendeu-se que o reprodutor do *iOS* resolia os problemas do reprodutor do *Android* através da colocação de um fundo nos botões do reprodutor. Através da biblioteca *appinio* é possível utilizar e configurar o reprodutor nativo de *iOS* em *Android*. Sendo assim, a utilização do reprodutor de *iOS* em ambos os sistemas operativos resolveria o problema. Este reprodutor permitiu a resolução de um problema menor, a visualização de vídeos em ponto grande, sendo que, dependendo da orientação do vídeo, o reprodutor altera a orientação da aplicação automaticamente voltando à orientação vertical, uma vez que, termina a visualização do vídeo em ponto grande.

### 1.3.14 Links

Uma das funcionalidades necessárias da aplicação é a utilização de *links*. Para isto, a programação *mobile* oferece duas soluções, *app links*, *deep links* e *dynamic links*. Como mencionado na secção de tecnologias foi decidido implementar a solução de *dynamic links* da *Firebase*.

Para implementar esta solução, primeiramente a nível de *backend* foi necessário gerar os *links*, para isto, existem duas opções, implementação do *Firebase* no próprio *backend* ou então uma chamada ao *Firebase* com a utilização de uma chave de pedido. Em primeiro lugar foi testada a implementação do *Firebase* no próprio *backend*, contudo, esta implementação surgiu com alguns problemas, uma vez que, existem diversas configurações específicas necessárias, sendo então recomendado pelos colegas de trabalho a chamada ao *Firebase* dada à sua simplicidade.

Sendo assim, para a realização de chamadas ao *Firebase* foi utilizado o *axios*. Este permitiu realizar um pedido com o método *POST* para o serviço de *dynamic links* do *Firebase*, com indicação do prefixo do projeto. Para permitir a reutilização deste código foi colocado num método onde é chamado com indicação do *link* desejado e os dados a enviar. O *link* é utilizado para, por exemplo, como numa página *web*, indicar qual página se deseja direcionar o utilizador, já os parâmetros, assim como em um *url web*, são

enviados através do próprio *link*, pelo que, estes dados são colocados na *string* do *link* o que permite a configuração perante diversas situações. Por fim, a *Firebase* retorna o *link* criado e este é colocado no *email* desejado.

Para a implementação do *frontend* foi necessário importar a biblioteca de *dynamic links* do *Firebase* e de seguida no código de inicialização da aplicação colocar um método para em caso de a aplicação ser aberta a partir de um *link*, este o ler. Quando este método lê o *link*, extraia a página indicada e a lista de parâmetros recebidos. Deste modo, o utilizador é redirecionado para a página do *link* com os dados recebidos.

Aquando o teste da implementação, diversas tentativas de abertura de *links* foram realizadas, mas, contudo, sem sucesso. A grande dificuldade desta implementação foi os *links* dinâmicos não permitem realizar *debug*, sendo que, ou funcionará na totalidade, ou não funcionará, o que leva a que seja complicado identificar *bugs*. A documentação do serviço foi de grande auxílio, uma vez que, estava em falta a indicação do nome do pacote da aplicação para *Android* e *iOS*. Após a colocação destes dados, tudo seguiu em completo funcionamento.

### 1.3.15 Notificações

A implementação das notificações revelou ser a de maior dificuldade, visto que, surgiram diversos imprevistos. Para isto foi utilizado o serviço de notificações do *Firebase*. Este surge como os *links*, em duas secções, primeiramente *backend* e de seguida *frontend*.

A nível do *backend* foi necessário utilizar *axios* para realizar um pedido ao serviço de notificações do *Firebase*. Mas assim como na criação dos *links* o serviço não indica quaisquer informações sobre erros, apenas o dispositivo poderá ou não receber a notificação.

Em primeiro lugar foi utilizado o conteúdo a enviar indicado pela documentação do serviço, mas, apenas retornava uma mensagem de erro. Posteriormente foi pesquisado outras implementações de outros utilizadores e testadas, mas, novamente surgia um erro, pelo que, decidiu-se utilizar o conteúdo indicado pelo professor de programação de dispositivos móveis, tendo este funcionado sem qualquer indicação de erro.

No conteúdo da notificação é enviado em formato *JSON* a mensagem a apresentar na notificação e como estas serão sempre referentes a tópicos ou comentários é indicado o *id* do tópico, do comentário e em caso de necessidade o *id* do comentário pai.

Este processo de notificação foi aproveitado para direcionar os mesmos dados para notificação de *email* em caso do utilizador possuir ativo as notificações por *email*, sendo gerado um *link* dinâmico com os dados na notificação.

A nível do *frontend* foi importada a biblioteca do serviço de notificações do *Firebase*, sendo esta implementada conforme a documentação. Como é necessário detetar notificações no iniciar da aplicação, durante a utilização e quando esta encontra-se em segundo plano aproveitaram-se estas deteções para implementar o direcionamento do utilizador para as páginas referentes às notificações, com a utilização dos dados recebidos.

O grande problema detetado nas notificações é que o *icon* não era apresentado corretamente, sendo que ou era demonstrado um quadrado escuro, ou nenhum *icon*. A biblioteca de notificações do *Firebase* não permite a customização do *icon*, pelo que foi decidido utilizar a biblioteca *flutter\_local\_notifications*. Esta permite a total customização das notificações, na qual é enviado o *icon* desenhado para a aplicação. Mesmo assim, as no-

tificações continuavam com o mesmo erro, pelo que, decidiu-se realizar uma pesquisa e percebeu-se que *Android* possui um novo sistema para os *icons* das notificações, deste modo, é necessário transformar estes em preto ou branco com fundo transparente e de seguida tratados pelo próprio *Android*.

Sendo assim foi realizada a transformação e novamente alterados os *icons* das notificações. Após um novo teste compreendeu-se que mesmo assim apenas o *icon* da notificação expandida teria sido alterado. Em seguida a uma nova pesquisa percebeu-se que *Android* necessita de dois *icons* de notificação para aplicar a ambas as situações, tendo sido resolvido o problema em questão.

Nesta implementação apenas um problema continuou sem resolução, a abertura de notificações quando a aplicação encontra-se terminada. Aqui foram testadas várias soluções, mas após a leitura da documentação e das soluções de outros utilizadores, compreendeu-se que o *Flutter* não permite a reconfiguração do comportamento das notificações quando a aplicação está terminada. O *Flutter* possui como futura implementação a permissão de reconfiguração do comportamento do *click* neste tipo de notificação, mas, de momento não dispõe de solução.

### 1.3.16 Permissões

Para garantir o acesso à galeria, às notificações e à *internet* é necessário pedir permissão ao utilizador.

Para pedir permissão à galeria, foi alterado o *android manifest* que pede permissão ao utilizador assim que for necessário aceder à galeria. Sendo assim, sempre que é executado algum código de acesso à galeria, pela primeira vez, o utilizador receberá um alerta a pedir permissão de acesso, sendo que, em caso de recusa não será possível aceder à galeria.

Para acesso à *internet* realizou-se o mesmo, mas, esta permissão é pedida no ato de inicialização da aplicação em conjunto com as permissões de notificações, contudo, as permissões de notificações são encarregues da biblioteca do *Firebase*.

### 1.3.17 Ios

Após o desenvolvimento completo da aplicação para dispositivos *Android*, foi proposto pela Motorline testar como esta se comportava num ambiente *iOS*, para isso, esta disponibilizou acesso a um dispositivo móvel *apple*, um computador, uma conta de programador e um colega de trabalho que já possuía experiência de desenvolvimento *iOS* com *flutter*.

Para a compilação, primeiramente foi necessário configurar a ferramenta *XCode*. Esta configuração foi realizada em conjunto com o colega de trabalho.

Depois do processo de configuração, o projeto foi compilado para *iOS*. Nesta primeira compilação toda a aplicação funcionava corretamente, mas, o colega de trabalho em questão indicou algumas configurações de *design* comuns em *iOS*, como por exemplo, os botões de cancelar e confirmar se encontrarem no topo do ecrã e a explicação de como lidar com navegação, uma vez que, *iOS* não dispõe de função de voltar para trás. Outros problemas encontrados foram as notificações e o *links* de aplicação.

Para a resolução do problema de navegação foram acrescentados botões de navegação em todas as páginas necessárias. Já para implementar a sugestão dos botões de cancelar e confirmar foi declarado que se o dispositivo em que a aplicação está a correr for um dispositivo *apple*, estes botões estarão no topo da página, caso contrário, ficarão no fundo.

A resolução do problema de notificações proveio de uma pesquisa sobre qual seria a possível fonte do problema, sendo detetado que as permissões poderiam não estar configuradas. Para realizar a configuração foi necessário atribuir as permissões de *Android* para *iOS*, contudo, estas foram implementadas através de um ficheiro com o nome de *info.plist*.

Posteriormente a esta adição, o colega de trabalho indicou que no envio do pedido de notificações para o *Firebase* é necessário também indicar as configurações de *iOS*. Deste modo foi procurado na documentação como se envia as configurações de *iOS*. Com a utilização destas configurações foi alterado o pedido de notificações e de links dinâmicos, visto que, é um pedido do mesmo estilo.

Para além das alterações anteriores foi necessário também configurar as notificações e *links* no *XCode*. Depois desta configuração foi testado e todas as notificações, *links* de *emails* e permissões funcionaram como planeado.