

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 relates the analysis, specification and development of the Install&Go software. This software 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.

The solution to this problem comes with the development of a forum where companies can register their professionals, and these can then expose their questions so that other professionals from other companies or Motorline itself can help. This also allows that if a professional has a problem that was already solved, he can search for the solution on the platform.

The development of this solution provided the acquisition of new technical skills such as cross-platform development and its frameworks, of which flutter was explored. It also allowed the assimilation of skill such as project analysis, specification and communication with clients. At the end, it was possible to fully develop the solution 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

1	Trabalho desenvolvido	1
1.1	Web scraper	1
1.1.1	Implementação web scraper	2
1.1.1.1	Implementação no website	7
1.1.1.2	Melhoria de implementação	8
1.1.1.3	Armazenamento de dados	12
1.2	Backend	13
1.2.1	Organização do projeto	13
1.2.2	Definição de rotas base	14
1.2.3	Middlewares	14
1.2.3.1	Linguagem	14
1.2.3.2	Autenticação	15
1.2.3.3	Validação de Papel	16
1.2.4	Controllers	16
1.2.4.1	Estruturação dos controllers	16
1.2.4.2	Execução da lógica de negócio	17
1.2.4.3	Validação dos dados	17
1.2.4.4	Formulação da resposta	17
1.2.4.5	Processamento de erros	18
1.2.4.6	Envio de <i>emails</i>	18
1.2.4.7	Logging de erros	19
1.2.4.8	Logging de pedidos com morgan	19
1.2.4.9	Agendamento de tarefas	19
1.2.4.10	Cifragem de <i>passwords</i>	20
1.2.4.11	Cifragem de configurações do servidor	20
1.2.4.12	Documentação Typedoc	20
1.2.4.13	Documentação Swagger	21

1.3	Frontend	22
1.3.1	Organização do projeto	22
1.3.2	Extensions	23
1.3.3	Handlers	23
1.3.4	Providers	23
1.3.5	Helpers	23
1.3.6	Gestão de utilizadores	24
1.3.7	Fórum	25
1.3.7.1	Filtragem de tópicos	26
1.3.7.2	Carregamento de tópicos	27
1.3.7.3	Detalhes de tópico	28
1.3.8	Firestore	29
1.3.9	Apresentação de Imagens	30
1.3.10	Apresentação de Imagens em carrossel	31
1.3.11	Carregamento de Imagens	31
1.3.12	Videos	32
1.3.13	Links	32
1.3.14	Notificações	33
1.3.15	Permissões	34
1.3.16	Ios	35

1. Trabalho desenvolvido

1.1 Web scraper

Após uma reunião com o cliente foi percebido que o catálogo de produtos Motorline não se encontra em um servidor. Estes dados encontram-se apenas diretamente no *website* da empresa, sendo assim viu-se a necessidade de criar um *web scraper*.

Nesta foi decidido que o *web scraper* iria apenas correr 1 vez por mês para evitar a sobrelotação do servidor. Para agilizar a realização do *web scraper* foi disponibilizado pela empresa a estrutura do *website* a seguir para obter as informações.

1.1.1 Implementação web scraper

A implementação e testagem do *web scraper* sem sobrelootação o servidor, foi alcançada pois foi préviamente descarregado todo o website localmente, o que permite assim simular o catálogo.

Para implementar o *web scraper* foi optado pela abordagem mais simples, realizar um pedido para obter a página *web*, ler a página 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 percebido que dados seriam necessários, sendo estes então:

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

Para guardar estes dados foi utilizado um dicionário que contém primeiramente 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 com é obtida a lista de todos os produtos, sendo cada produto representado também por um dicionário, que contém como chaves os atributos do mesmo. A utilização dicionários e listas para guardar estes dados deve-se a que o objetivo será guardar estes dados em *json* e a transformação é simplificada com a utilização destas estruturas devido à sua 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 video",
          "url": "link do video"
        }
      ]
    },
    "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 percebido que a página geral de produtos possui todas as categorias, assim como também as subcategorias com *urls* para as páginas que contém todos os produtos das subcategorias.

Sendo assim foi em primeiro lugar, percebido que cada conjunto(categoria, subcategorias) é uma secção, pelo que, são obtidas todas as secções de categorias. Para cada uma destas secções é obtido o título que equivale ao nome da categoria e também todos os elementos a 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

Sendo assim já é possível identificar cada categoria e subcategoria, assim como também o *url* da página de produtos para cada subcategoria. Mas após alguma análise dos dados foi percebido que estas não contêm acentuação devido à sua formatação no *website*. Para resolver este problema foi pesquisado por ferramentas capazes de corrigir estes erros ortográficos. Pelo que foi descoberto que a biblioteca mais utilizada em *python* para resolver este problema é a biblioteca *spellchecker*, esta ferramenta é a mais utilizada devido à sua capacidade de corrigir erros ortográficos em diversas linguagens. Sendo assim 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 são obtidos os *urls* de produtos e imagens de amostra dos produtos. Em cada página foi obtido 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 do produto, pois todos os produtos seguem a mesma estrutura, sendo esta, /produtos/nome-produto. Como em *urls* não é permitido utilizar acentuação e espaços, então todos os nomes foram corrigidos com a mesma ferramenta de correção ortográfica mencionada anteriormente.

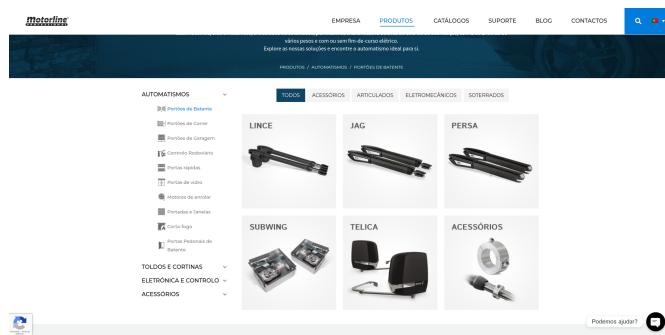


Figura 1.3: Página de produtos de uma subcategoria

Neste momento após correr o código foi percebido que existem algumas páginas de produtos em que este não conseguia obter dados, pelo que um erro era atirado, para perceber exatamente que páginas de produtos este erro acontecia, sempre que um erro era detetado este *url* seria 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. Foi então nesta ocasião percebido que nem todas as páginas de produtos são iguais e após uma reunião com o cliente este expôs que existem páginas de produtos e de detalhes de produtos que são muito diferentes das restantes.

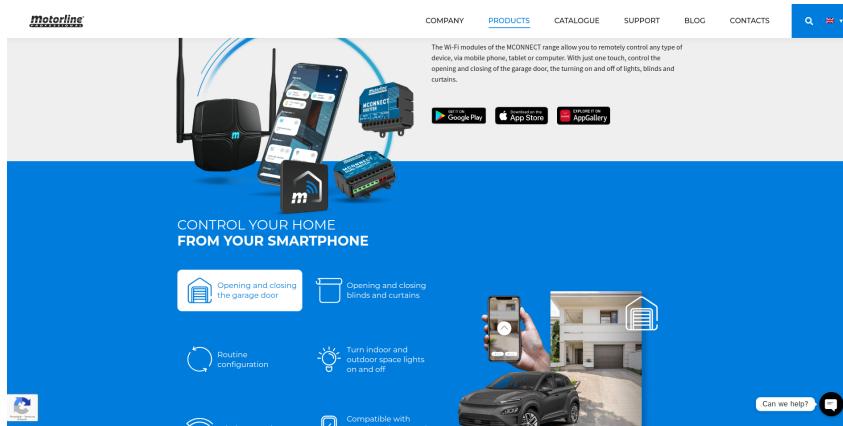


Figura 1.4: Página de produtos de uma subcategoria distinta

Com o objetivo de não atrasar o restante do projeto foi decidido em primeiro lugar obter todos os produtos que contêm páginas semelhantes. Para cada página de produto foi obtido o título que corresponde ao nome do produto, o elemento que contém o id produto-descrição, que corresponde à descrição do produto. 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 de *url*, o que permite que estes sejam obtidos através de todos os *urls* da secção de documentação, assim como os seus nomes e todas as opções de documentação do *dropdown* se este existir.

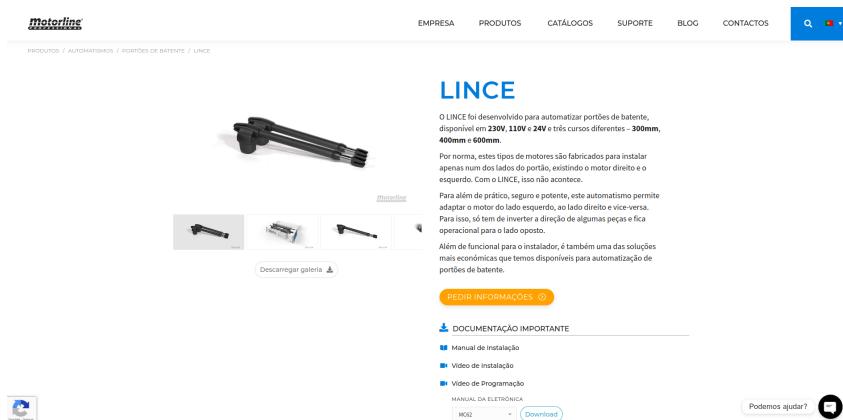


Figura 1.5: Página de detalhes de produto, secção inicial

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

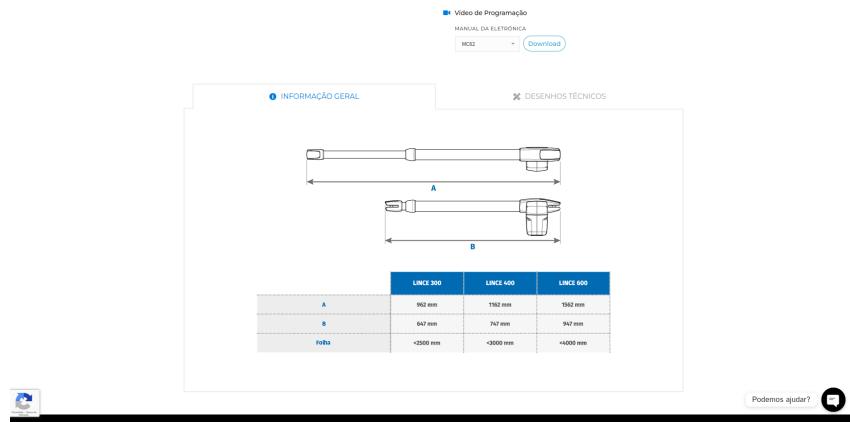


Figura 1.6: Página de detalhes de produto, secção de informações

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 video, mas após tentar visualizar este *url*, foi percebido que não é possível obter o vídeo a partir deste. Sendo assim foi investigada a plataforma *vimeo*, esta contém todos os videos de produtos, pelo que para cada um é gerado um *id* único e este poderá ser acedido através do *url* geral da plataforma seguido do *id*. O *id* está também colocado no elemento *iframe*, pelo que é obtido e acrescentado ao *url* da plataforma o que permite assim guardar todos os vídeos de produtos.

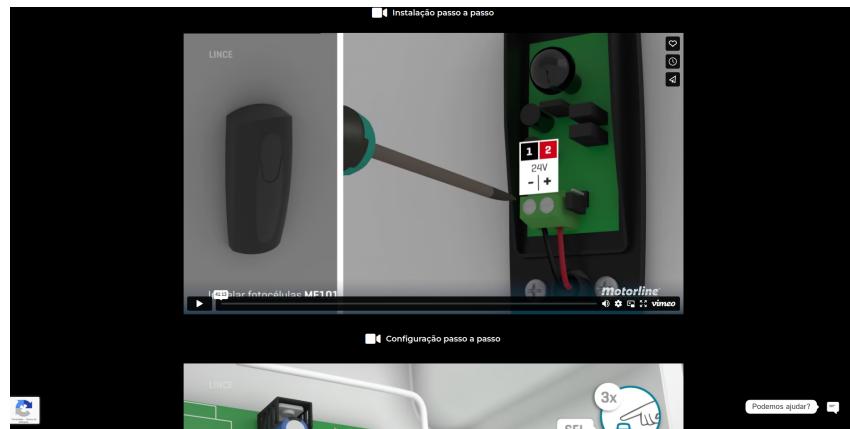


Figura 1.7: Página de detalhes de produto, secção de videos

1.1.1.1 Implementação no website

Após se verificar que eram obtidos pelo menos 80% dos produtos totais foi então decidido testar no *website*. Para isto foi utilizada a biblioteca *requests*, com a qual é realizado um pedido *GET* a cada *url* necessário para se obter a página *web*. Assim que o código foi corrido e a resposta analisada foi percebido que o *website* bloqueia este tipo de solução como indicado pela 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 então percebido que seria necessário alterar a abordagem visto que a anterior não seria possível utilizar. 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 foi descoberto 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, foi optado pela segunda opção, visto que seria possível continuar com trabalho em paralelo com o processo de obter dados. Sendo assim a ferramenta mais recomendada para realizar esta operação é a biblioteca *selenium*, esta permite realizar exatamente o processo referido anteriormente com a possibilidade de escalar com *multi threading*, o que permite abrir diversas janelas do navegador simultaneamente e obter dados destas o que diminui drasticamente o tempo de execução, esta funcionalidade não foi explorada devido a limitações de *hardware*, mas seria uma importante implementação futura.

Com a utilização a biblioteca *selenium* foi indicado qual o navegador a utilizar, neste caso foi escolhido o *chrome* devido a este já se encontrar instalado no dispositivo. Após se indicar qual o navegador a utilizar, é necessário para cada página indicar qual elemento esperar que carregue, pois por vezes existem elementos que demoram mais tempo a carregar do que a página. Visto que a página carregada poderá não conter o elemento a obter, foi então implementado um tempo de espera máximo de 5 segundos, assim que este tempo expira a operação é abortada e o *url* é adicionado à lista de *urls* com erros.

1.1.1.2 Melhoria de implementação

Após se completar 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 foi então direcionado 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/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(mb17)/",
    "https://motorline.pt/produto(mbn25)/",
    "https://motorline.pt/produto(mim1100u)/",
    "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

Após uma primeira análise foi possível perceber que grande maioria dos erros provém 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. Foi percebido que nos *urls* a palavra accessórios está sempre contida, o que permite que sempre que esta é detetada em um *url*, seja corrido o código referente à obtenção 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 contém uma imagem, titulo e descrição, esta descrição por vezes contém 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.

Acessórios para Motores de Batente

Batente Lince

Batente em alumínio, para aplicação em toda a linha **LINCE**, para limitação na abertura, evitando assim a utilização de batente no portão.



RL180

Acessório que permite a abertura 180° de um motor **SUBWING**.



KIT BATENTE

Batente mecânico para um motor **SUBWING**.



Figura 1.10: Exemplo de página de acessórios

Na iteração seguinte foi percebido que a quantidade de urls com erros diminuiu, mas existiam produtos com erro, pelo que estes foram analisados e foi percebido que o erro ocorria devido a por vezes as páginas não conterem vídeos ou imagens de documentação. Para resolver este problema o código foi alterado para apenas obter estes dados se os elementos existirem na página.

Após correr novamente o código foi percebido que a quantidade de falhas obtidas diminuiu drasticamente (Figura 1.11). Mas mesmo assim ainda existiam 3 falhas a ocorrer e após uma análise foi percebido que estas falhas ocorriam devido a:

1. Uma página de subcategoria de produtos conter um serviço;
2. Um produto conter uma página de detalhes de produto com sub produtos;
3. Existir uma página de adaptadores de produtos;
4. Um produto conter uma página de detalhes diferente das demais;

```
"misses": [
    "https://motorline.pt/produtos/elettronica-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: Exemplo de página de acessórios

A página de adaptadores de produtos segue uma estrutura similar à estrutura dos acessórios pelo que foi a primeira a ser abordada e resolvida através do código de obter detalhes de acessórios. Pelo que 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 sejam ser combinadas para construir os detalhes do produto. Após se obter estes dados foi percebido que as imagens do produto têm dados escritos que não se encontram na imagem original, em solução a este problema o cliente recomendou obter estas com *screenshots* e guardar num servidor de imagens.



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 a alteração na base de dados, foi alterado o código para obter os dados do catálogo. 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.

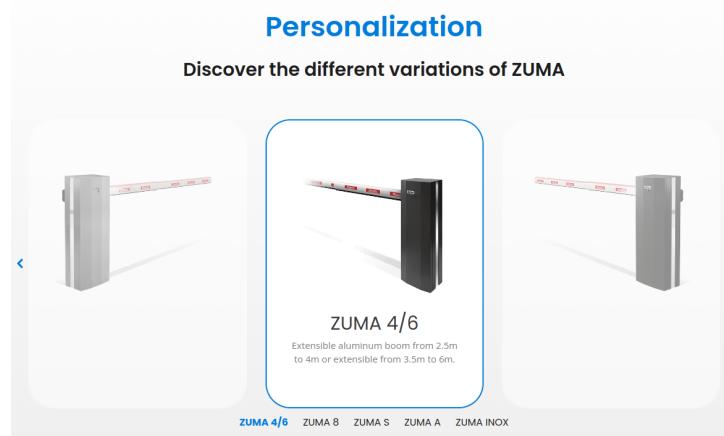


Figura 1.13: Exemplo de página de produto com subprodutos

O último problema a solucionar é a existência de serviços, iniciou-se então 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 e suas ofertas, 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 as obter sem alteração manual, o que levou a que estas tabelas também existam para os produtos. De seguida foi percebido que existem dados que não são necessários na descrição. O cliente recomendou guardar estes dados como *screenshots*.

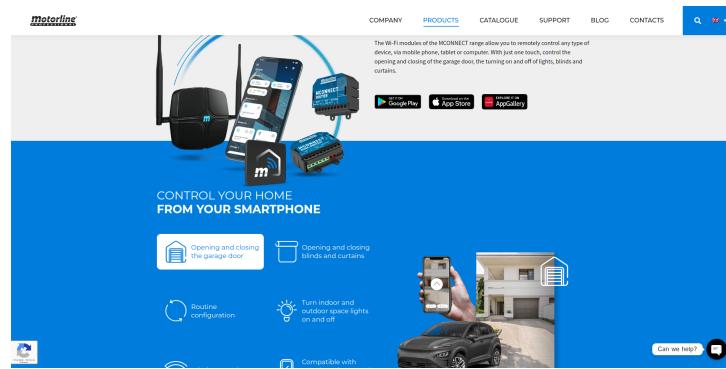


Figura 1.14: Exemplo de página de serviço

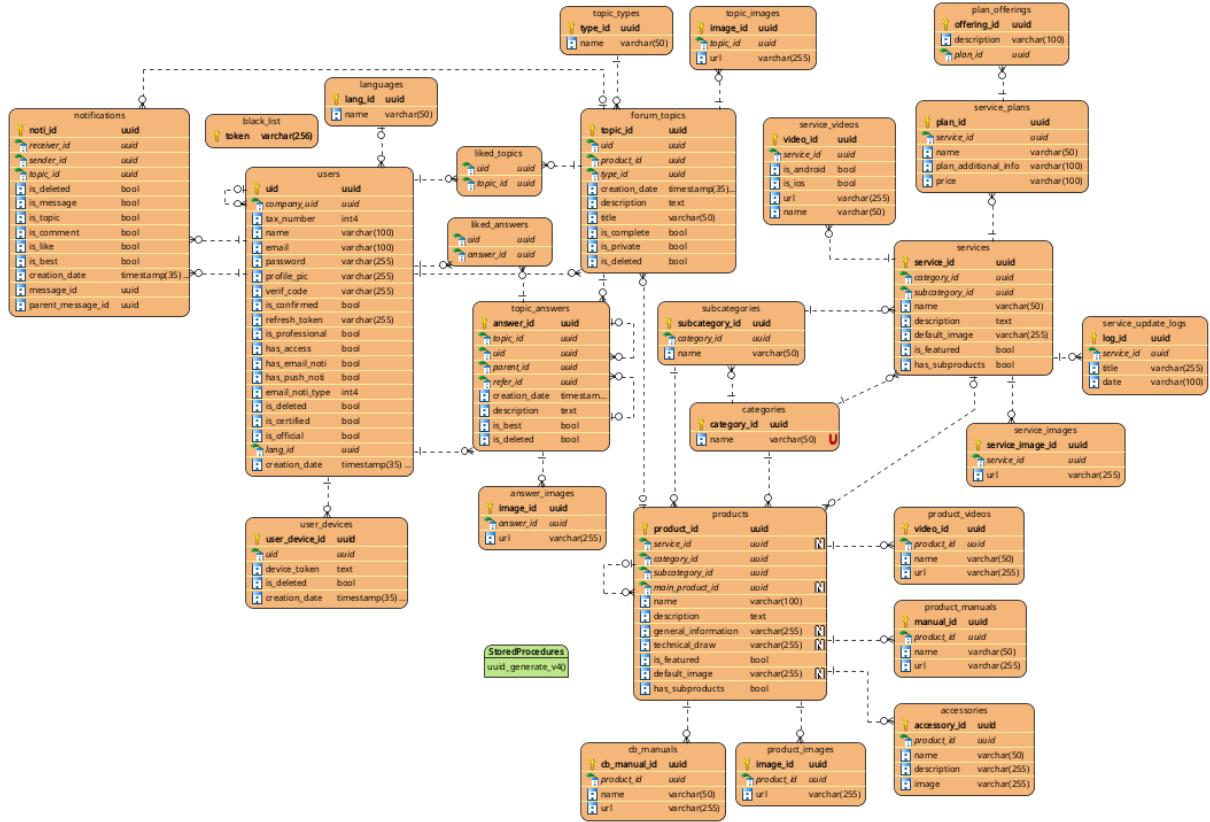


Figura 1.15: atualização da base de dados

1.1.1.3 Armazenamento de dados

Após se obter os dados dos produtos, é necessário guardar na base de dados para disponibilizar para a utilização para *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. Visto que não seria de grande interesse conectar diretamente à base de dados, 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 entre estes, o que levou a que o limite máximo de conexões com a base de dados fosse extrapolado. Isto acontece porque para cada serviço chamado é criada 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 em um *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 e todas são inseridas com a referência para a sua categoria.

1.2 Backend

Após o desenvolvimento do *webscraper*, foi procedido com desenvolvimento do suporte *backend* do projeto.

1.2.1 Organização do projeto

Antes de iniciar a implementação, foi definida a estrutura de projeto a seguir, a escolhida foi *MVC* devido a ser a mais comum e bem estabelecida. Sendo assim a organização do projeto segue a seguinte estrutura:

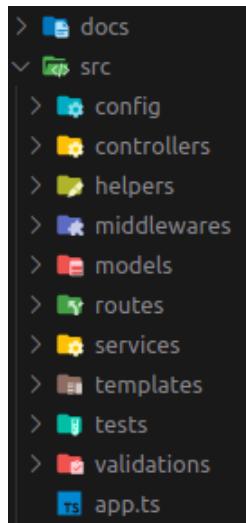


Figura 1.16: Exemplo de página de produto incomum

- **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 para as entidades de resposta;
- **routes** - Rotas existentes;
- **services** - Serviços para cada pedido;
- **templates** - Templates de *email* a serem enviados;
- **tests** - Testes de código realizados;

- **validations** - Validações a realizar a nível de modelo de negócio e validação de dados;
- **app** - Ficheiro de início do projeto;

1.2.2 Definição de rotas base

Após a definição da estrutura do projeto foram definidas as rotas base a existir, estas são rotas que se referem a cada ator. Para melhor organização destas rotas e 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 então 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 também executar código.

1.2.3.1 Linguagem

O bem essencial em 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*, o objetivo deste é verificar se existe a chave *language* no cabeçalho do pedido, caso esta exista é então obtida a linguagem e guardada nas variáveis locais do pedido. Na eventualidade desta não existir, foi decidido que a aplicação responderá 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 JsonWebToken, 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 e assim que este *token* expirar, este terá de se autenticar novamente para obter um novo *token*. A utilização de *tokens* permite também 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.

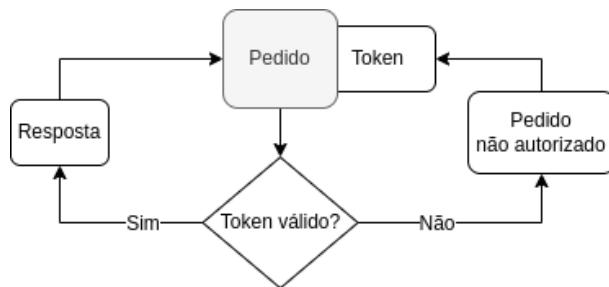


Figura 1.17: Utilização de tokens

A grande valia 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, e sempre que um *token* de sessão expira o utilizador necessita de 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*. Sempre que o utilizador termina a sua 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 também ser validado e apenas após esta verificação o utilizador estará autenticado. Na eventualidade de o *token* de sessão ou de *refresh* se encontrarem expirados, este estará sem autorização para realizar o pedido, mas poderá pedir 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 jwt*, esta funcionalidade foi utilizada para enviar o *id* e cargo do utilizador a qual pertence este *token*.

1.2.3.3 Validação de Papel

Com finalidade de garantir que apenas utilizadores com cargos suficientes conseguem realizar as ações regradas, foi criado 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, como mencionado, o cargo do utilizador em questão é enviado no *token*, pelo que é obtido este é obtido realizada uma comparação, com o cargo desejado. Para isto foram criados 2 *middlewares* diferentes, um valida o cargo de empresas e o outro o cargo de técnicos. Visto que as empresas podem realizar operações de técnicos então 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.

1.2.4 Controllers

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

1.2.4.1 Estruturação dos controllers

Para evitar variação de código destes *controllers* em termos de estrutura, foi então decidido desenhar uma estrutura de *controller* e aplicar esta 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 sendo apenas necessário escrever a palavra-chave e toda a estrutura é aplicar, 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, de onde 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 em primeiro lugar verificado se todos os dados são recebidos, de seguida estes são enviados para um validador. O validador executa todas as verificações necessárias a nível de regras de negócio e na possibilidade de alguma regra não ser cumprida, é então atirado um erro.

1.2.4.4 Formulação da resposta

Assim como mencionado anteriormente o bem mais importante numa boa comunicação é a utilização da mesma linguagem, sendo assim a resposta do servidor deverá utilizar a linguagem indicada pelo utilizador. De forma a realizar esta tradução foi utilizado o mesmo conceito que é utilizado para a tradução de aplicações android onde nestas é criado um ficheiro que contém um conjunto de chaves e a cada chave corresponde um texto, para cada tradução estas chaves têm de existir de forma a ser possível obter o texto correto para cada chave. Sendo assim foi utilizado um ficheiro json contento as chaves das linguagens suportadas, a cada linguagem corresponde um conjunto de outras chaves que contém todas as traduções necessárias, utilizando neste caso numeração, em vez de palavras. Esta abordagem permite que de forma fácil futuramente seja possível adicionar outras linguagens ao servidor.

Para dar suporte a este ficheiro foi criada uma operação que recebe a chave desejada e a linguagem desejada, devolvendo o texto correspondente, sendo assim na Formulação da resposta esta operação é executada indicando a chave da resposta a enviar e a linguagem desejada obtendo o texto traduzido, sendo então este devolvido para o utilizador.

1.2.4.5 Processamento de erros

Visto que não é de interesse enviar para o utilizador erros do próprio servidor, foi então decidido controlar estes, para isso foi criado um erro customizado, tendo este por base o erro da própria linguagem. Este erro recebe por parâmetro o código da tradução da mensagem de erro. Esta abordagem permite também evitar que sempre que um erro é lançado o sistema pare. Mesmo com esta abordagem acontece que sempre que um erro é lançado por base de dados, erro de código ou de biblioteca, o erro original é chamado, pelo que foi decidido que sempre que é detetado um erro que não é do tipo do erro customizado, então será devolvido um erro com mensagem de erro de servidor evitando que dados sensíveis e desnecessários para o utilizador sejam devolvidos.

1.2.4.6 Envio de *emails*

Como mencionado na apresentação da ferramenta tabular, esta não permite a utilização de acentuação na escrita de *emails*, sendo assim estes erros necessitam de ser resolvidos. Para permitir que os dados dos *emails* sejam personalizáveis, como por exemplo dados de utilizador e link para clicar, estes *emails* são colocados dentro de métodos que recebem por parâmetro os dados personalizáveis, sendo estes então colocados dentro do html do *email*, este método por fim devolve em string o html do *email* a enviar.

Para o envio de *emails* é necessário um servidor, sendo assim para vias de teste foi utilizado um servidor gratuito de *email* sendo este hospedado por DIZER QUEM HOSPEDAVA, após a fase de testes foi então alterado para o servidor de *email* da empresa permitindo assim que este *email* seja identificado como empresa Motorline.

A configuração do servidor de *emails* do nodemailer é realizada através de uma chamada ao objeto de servidor indicando as configurações do mesmo que estão no ficheiro .env, esta chamada ao servidor por sua vez devolve um objeto que fornece métodos para enviar *emails*, sendo este então criado e utilizado sempre que se deseja enviar *emails* no servidor.

De forma a evitar que sempre que se deseja enviar um *email* seja necessário indicar todos os dados, foi então criado um método que cria e devolve o objeto de servidor sempre que necessário. Sendo assim sempre que se deseja enviar um *email* é então 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 do *email*.

1.2.4.7 Logging de erros

A identificação dos erros no servidor é difícil, pois uma vez que os erros são tratados, os dados referentes aos mesmos não são guardados ou utilizados, para resolver este problema foi então decidido 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 e assim como também a descrição original do erro e serviço referente.

Para implementar esta solução foi então primeiramente 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, sendo então estes dados acrescentados em uma nova linha no ficheiro de erros do servidor.

1.2.4.8 Logging de pedidos com morgan

Para realizar o logging de pedidos a serviços foi então utilizado o morgan, esta ferramenta apenas precisa que seja indicado o ficheiro onde escrever os logs, sendo utilizado um ficheiro chamado log, este também precisa de saber qual o tipo de log a realizar, sendo estes tipos indicados pela ferramenta, o tipo utilizado foi o combinado, este é o tipo de log mais completo da ferramenta, visto que é o que obtém mais dados, sendo estes, a data e hora do pedido, o tipo de pedido, o serviço pedido, os dados recebidos, a resposta devolvida e também a descrição do sistema utilizado.

1.2.4.9 Agendamento de tarefas

Utilizando a ferramenta node-cron foi inicialmente programado para este enviar o relatório de notificações todos os dias às 22 horas, para realizar esta configuração é necessário indicar primeiramente a programação de horário de envio, para isso é utilizada a estrutura segundo, minuto, hora, dia do mês, mês, dia da semana. Visto que o objetivo é 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, significando assim todos os dias. A configuração final obtida foi então "0 0 22 * * *".

Quando o método do agendamento é invocado, é então obtido todos os utilizadores com a configuração de relatório de notificações ativa, sendo de seguida para cada um destes obtidas todas as notificações do dia. De seguida é criado o objeto de servidor e invocado o método para obter o *email* de notificações enviando todas as notificações do utilizador para o mesmo. Por fim é enviado o *email* para o utilizador com todas as suas notificações e um link para aceder rapidamente ao ecrã de notificações.

1.2.4.10 Cifragem de *passwords*

Aquando o registo de clientes é indicada a *password*, pelo que esta deverá ser guardada cifrada. Para isto é utilizada a biblioteca bcrypt, sendo que no processo de registo, antes do envio dos dados do utilizador para o servidor, é invocado o método de cifragem da *password*, indicando como salt o valor 8, após a execução é então obtida a *password* cifrada, sendo esta enviada para a base de dados com conjunto com os restantes dados.

1.2.4.11 Cifragem de configurações do servidor

Para a cifragem das configurações do servidor, foi então executado o comando de cifragem da biblioteca secure-env, sendo pedido a *password* da mesma, após esta indicação, foi criado o ficheiro cifrado, pelo que o ficheiro original deverá ser eliminado ou em caso de necessidade guardado em um local seguro de forma a evitar que durante algum ataque seja possível obter os dados do mesmo.

Após a cifragem do ficheiro o servidor foi configurado para aquando a sua inicialização, este deverá pedir para o utilizador indicar a *password* do mesmo, sendo esta a indicada na cifragem do ficheiro de configurações do servidor, para esta configuração foi utilizada a biblioteca readline-sync no modo de *password*. Quando o utilizador indica a *password* em caso de esta estar errada o servidor irá falhar, pois a biblioteca de cifragem irá tentar decifrar o ficheiro de configurações e irá falhar não concluindo o processo de inicialização, caso contrário, este continuará o processo de inicialização, sendo o primeiro passo a utilização da *password* para decifrar o ficheiro de configurações e carregar as variáveis de ambiente.

1.2.4.12 Documentação Typedoc

Durante a implementação desta ferramenta foi detetado que a categorização de documentação não se encontrava funcional devido a um problema encontrado pelos desenvolvedores da ferramenta, foi decidido então reduzir a versão da mesma para uma com a funcionalidade ativada, mas esta não se encontrava compatível com a versão mais atualizada de typescript, pelo que não foi possível explorar esta funcionalidade. Para contornar o problema foi então decidido explorar outra funcionalidade menos utilizada da ferramenta, esta funcionalidade permite converter qualquer documentação em módulos, estes módulos podem então ser categorizados, o problema destes módulos é que cria um modelo genérico do código não sendo facilmente identificado as tipagens de scripts. Estes módulos permitem também a categorização dos mesmos permitindo assim um nível de organização da documentação gerada.

1.2.4.13 Documentação Swagger

Apesar da ferramenta disponibilizar a geração automática de documentação a partir de comentários de código, foram encontrados alguns problemas com esta funcionalidade, acabando por não gerar a documentação, pelo que foi optado 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 para os mesmos, ambas estas funcionalidades foram exploradas conseguindo assim um bom suporte de documentação para qualquer utilizador.

Figura 1.18: Documentação swagger

Figura 1.19: Exemplo de documentação de serviço

1.3 Frontend

Após o desenvolvimento do suporte backend, foi então prosseguido para o desenvolvimento frontend do projeto.

1.3.1 Organização do projeto

Assim como o backend o modelo a seguir para o frontend foi o MVC.

Como recomendado de boas práticas de código limpo da framework as cores do tema da aplicação foram todas colocadas em um ficheiro separado de forma a garantir que a troca de tema da aplicação é facilitada. Outras aplicações de boas práticas de código limpo foram, sempre que possível partitionar o código das páginas em vários widgets de forma a ser de fácil navegação e também a criação de widgets reutilizaveis de forma a evitar a repetição de código e também para agilizar o desenvolvimento de código. Sendo assim a estrutura do projeto foi organizada da seguinte forma:

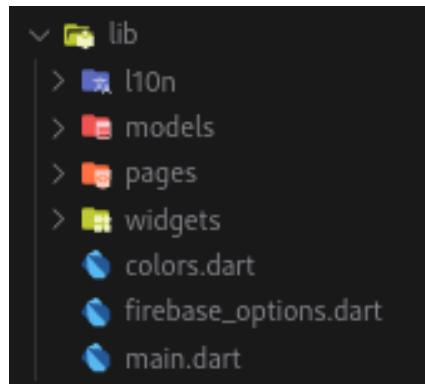


Figura 1.20: Organização do projeto

- **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 Extensions

A linguagem de programação dart, assim como outras linguagens de programação orientadas as objetos permite alterar e adicionar métodos aos objetos base da linguagem, para realizar isso são criadas extensões do objeto, neste caso foi criada uma extensão para o objeto string de forma a facilmente capitalizar um texto.

1.3.3 Handlers

Os handlers são porções de código que permitem a execução de código perante um evento como por exemplo realizar uma ação perante um clique no ecrã, neste caso os handlers foram utilizados para detetar o estado da aplicação e realizar ações perante estes 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 perante estes estados, como por exemplo tratar de notificações da aplicação.

1.3.4 Providers

Os providers são classes criadas para ajudar com comunicações externas, neste caso chamdas à API, estes providers foram criados de acordo com os diversos modelos de dados que se recebem, como por exemplo, uma publicação do fórum. Um provider oferece perante um modelo um conjunto de métodos para as diferentes chamadas necessárias à API, utilizando o exemplo anterior, para uma publicação existem métodos para eliminar, adicionar, editar e obter dados, sendo que acada método terá os seus requisitos do serviço que invocam.

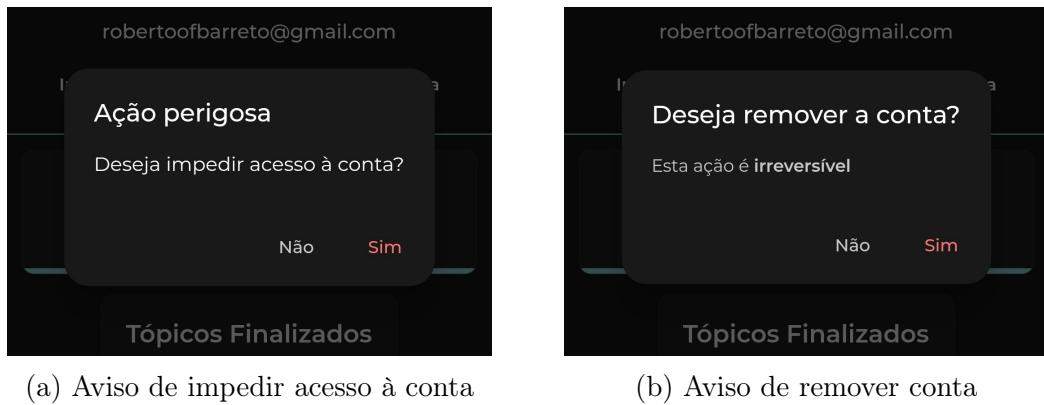
Estes providers automaticamente detetam a linguagem da aplicação e realizam o pedido ao serviço utilizando essa linguagem.

1.3.5 Helpers

Os helpers como o próprio nome indica são classes que ajudam com a realização de uma tarefa, neste caso os helpers, foram utilizados no auxílio de mensagens de notificações, estas mensagens deveriam conter o nome do utilizador e também a ação do mesmo traduzida na linguagem da aplicação, sendo assim foi criada a classe de helper de notificação que contém um método estático para obter a mensagem de uma notificação de acordo com a ação da mesma.

1.3.6 Gestão de utilizadores

Um requisito da aplicação e que apenas as empresas têm a possibilidade de se registarem, pelo que apenas estas poderão registar os seus técnicos. Para isso foi criada a página de gestão de utilizadores apenas acessível às empresas, nestas páginas estas poderão pesquisar pelos seus técnicos, ou registar novos técnicos, sendo que tem-se ser obrigatoriamente indicado o nome, *email* e tipo de utilizador. Outras ações que este utilizador poderá realizar é a visualizar um técnico onde poderá bloquear o acesso ou até apagar a conta do mesmo, sendo que esta ação é irreversível.



Sempre que um utilizador sem acesso ou com a conta apagada efetua o login, este receberá uma mensagem de erro mencionando que a sua conta não possui acesso à conta impedindo assim o acesso.

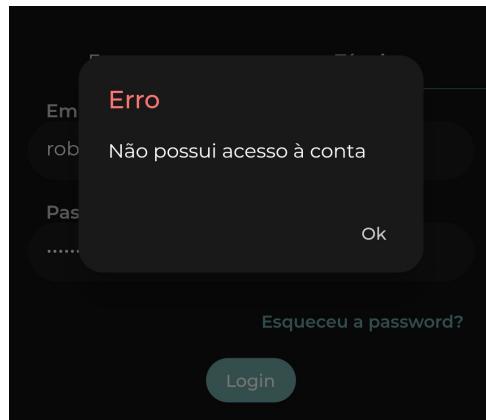


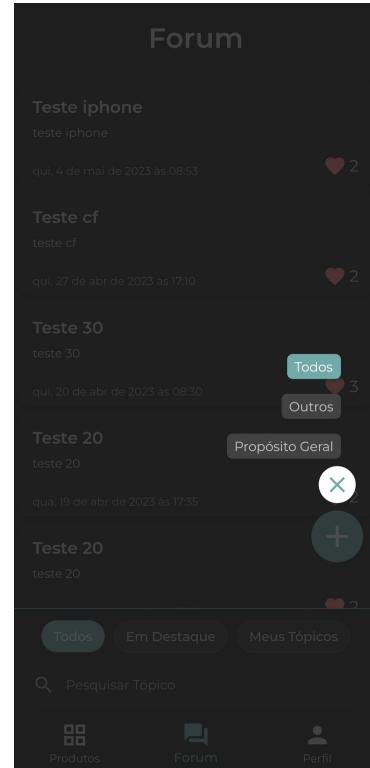
Figura 1.22: Aviso de login a conta sem acesso

1.3.7 Fórum

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



(a) Página de forum



(b) Filtragem de tipo

1.3.7.1 Filtragem de tópicos

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

Uma vez que algum filtro seja alterado, os filtros seguintes deverão ser novamente executados de forma a garantir que todos estes se encontram aplicados, inicialmente este tipo de filtragem não era executado, o que levava a problemas como por exemplo sempre que se efetua uma pesquisa, esta não era efetuada sobre as publicações filtradas o que levava a que a pesquisa fosse efetuada por todas as publicações.

Outro problema encontrado era na troca de categoria, por vezes acontecia que os filtros de categoria adicionavam-se o que levava a que estes não mostrassem publicações.

Sendo assim foram criados métodos para auxilio na filtragem, assim como também uma prioridade, sendo que a cada método invoca o método seguinte em forma de encadeação. Primeiramente aplica-se o filtro de categoria, de seguida este envia o resultado da filtragem para o método de filtragem por tipo e por fim se existir algum tipo de pesquisa os tópicos serão filtrados pela mesma.

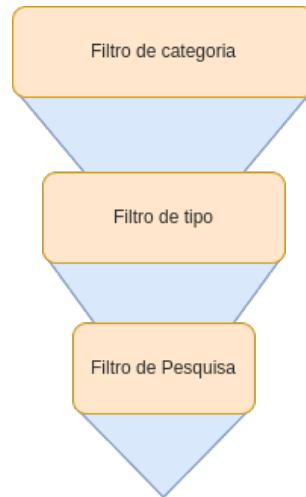


Figura 1.24: Filtragem do forum

1.3.7.2 Carregamento de tópicos

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

De forma a resolver este problema primeiramente pensou-se em uma técnica de sliding window no total, o problema é que esta solução iria levar a que se o utilizador desejar voltar para trás nos tópicos a partir de uma quantidade escolhida de tópicos estes teriam de ser carregados novamente, o que não levaria a uma boa experiência de utilização.

Para resolver este problema foi utilizada a ideia de sliding window, mas os tópicos iriam se manter 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 utilizando 3 valores, quantidade de tópicos a obter, índice inicial e data do primeiro tópico. O valor de quantidade de tópicos a obter, inicialmente 10, permite limitar a quantidade de tópicos que a API irá processar reduzindo o tempo de resposta, o índice inicial, permite indicar qual o índice do primeiro elemento que se deseja obter da lista e a data do primeiro tópico permite manter uma referência temporal para obter tópicos, garantindo assim que a lista que se está a visualizar é sempre a mesma.

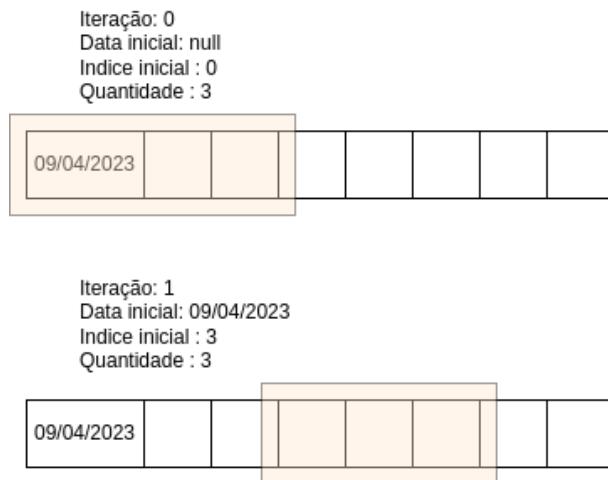


Figura 1.25: Carregamento de tópicos

Foi também reduzido a quantidade de dados carregados por cada tópico, sendo assim apenas os comentários diretos à publicação são carregados não carregando as respostas a estes, o que também contribuiu para a melhoria de performance.

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

1.3.7.3 Detalhes de tópico

A página de detalhes de tópico sofreu os mesmos problemas que a página anterior sendo necessário aplicar a mesma solução sobre os comentários de tópico e também sobre as respostas ao mesmo. Sendo assim são carregados os primeiros 10 comentários e por fim mostrado ao utilizador quantos mais comentários existem que poderá carregar sendo o limite 10 comentários.

Estes comentários podem também conter respostas sendo que estas podem ser carregadas também 10 de cada vez conseguindo o utilizador esconder ou mostrar estas.

Um problema que surgiu no desenvolvimento da página de detalhes de tópico foi também o destaque de uma mensagem para por exemplo destacar alguma resposta de um notificação. Este foi um grande problema pois com a nova implementação as mensagens não se encontram carregadas no momento de destacar a mensagem, pelo que é necessário procurar a mesma dentro das mensagens carregadas, expandindo assim as respostas do comentário que contém a mensagem a destacar.

O destacamento de mensagens também continha um erro no qual sempre que algo no ecrã é atualizado, este recarregava a animação pelo que este código teve de ser movido de forma a apenas ser executado no momento de inicialização do ecrã após todos os elementos se encontrarem devidamente carregados.



Figura 1.26: Destaque de mensagens

1.3.8 Firestore

Visto que a desenvolvedora do Flutter e do Firebase é a mesma, esta disponibilizou recursos que facilitam a utilização desta ferramenta pelo Flutter, sendo assim todas as imagens e vídeos de utilizadores, publicações e comentários são guardadas diretamente da aplicação para o firestore, assim como o acesso às mesmas é realizado diretamente.

Para permitir este tipo de acesso o Firebase disponibiliza de um configurador de projeto que permite através do terminal configurar a ligação entre o projeto e o servidor do firebase, sendo no final apenas necessário importar a biblioteca do serviço do firebase que se deseja invocar a classe do mesmo sempre que se deseja realizar alguma ação.

Os ficheiros ficam então organizados de acordo com o seu contexto, no caso de utilizadores existe a pasta utilizadores, no caso de tópicos existe a pasta tópicos e no caso de comentários existe a pasta comentários.

Dentro dos utilizadores como cada utilizador apenas contém uma imagem então estas são guardadas com o nome do id do utilizador substituindo a imagem anterior se existir. No caso de tópicos e comentários como podem conter várias imagens e vídeos, então estes são guardados em pastas com ids dos mesmos contendo dentro destas os ficheiros referentes.

The screenshot shows the 'Storage' interface in the Firebase console. At the top, there are tabs for 'Files', 'Rules', 'Usage', and 'Extensões NOVO'. Below the tabs, there's a URL 'gs://install-and-go.appspot.com'. A blue button labeled 'Fazer upload do arquivo' (Upload file) is visible. The main area is a table with columns: Name, Tamanho (Size), Tipo (Type), and Última modificação (Last modification). The table contains three entries: 'Products/' (Pasta), 'topics/' (Pasta), and 'users/' (Pasta).

Name	Tamanho	Tipo	Última modificação
Products/	—	Pasta	—
topics/	—	Pasta	—
users/	—	Pasta	—

(a) Raiz do firestore

This screenshot shows the 'Storage' interface again, but now the 'topics' folder is selected under the URL 'gs://install-and-go.appspot.com > topics'. The table structure is identical to the root, showing subfolders '1ddd308-9fd5-41ac-8d17-b2d0410486fd/' and '256c6ac4-7ce4-495c-a42a-97cd2270f3e3/'. There is also a third entry starting with '2c77661f-ded5-488b-943c-d35e667b9194/'.

Name	Tamanho	Tipo	Última modificação
1ddd308-9fd5-41ac-8d17-b2d0410486fd/	—	Pasta	—
256c6ac4-7ce4-495c-a42a-97cd2270f3e3/	—	Pasta	—
2c77661f-ded5-488b-943c-d35e667b9194/	—	Pasta	—

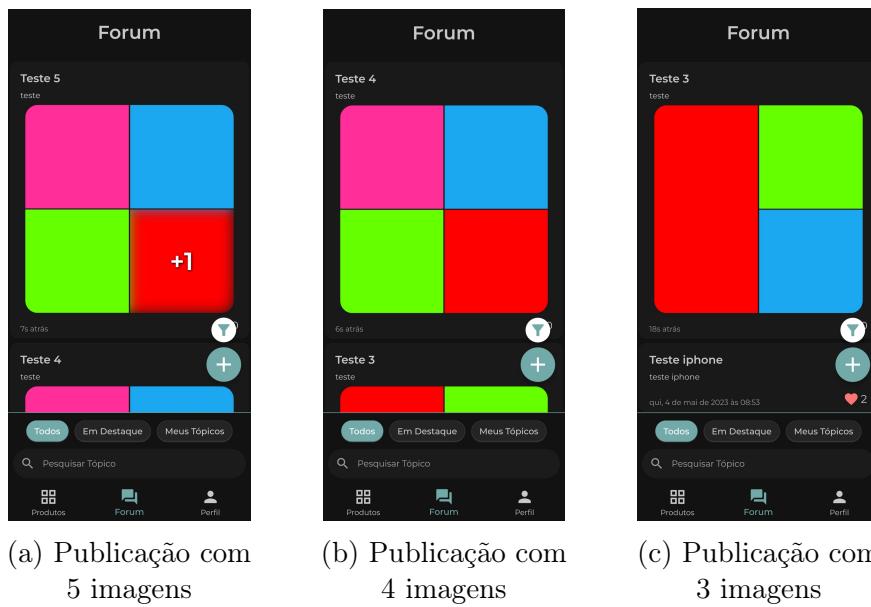
(b) Pasta topics do firestore

1.3.9 Apresentação de Imagens

A apresentação de imagens é definida em 2 níveis, a nível de pré-visualização, por exemplo ver a miniatura da imagem de uma publicação no forum e a nível de detalhe, onde é possível ver a imagem em ponto grande e realizar zoom na mesma.

Para a apresentação de miniatura da imagem foi decidido mostrar até 4 imagens, sendo que acima de 4 imagens seria mostrado apenas 3 imagens significando assim que a quarta imagem indicaria quantas mais imagens existem para mostrar.

Para a implementação da apresentação das miniaturas de imagens, foi utilizada a biblioteca staggered_grid_view, esta biblioteca permite organizar imagens em grelha. Neste contexto desejava-se organizar estas imagens em diferentes aspetos, e disposições, pelo que esta biblioteca permite indicar quantas colunas e linhas existem na grelha ao criar o agrupamento de imagens, sendo assim foi decidido 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 4 ou mais então as 4 imagens dividem a grelha por igual. Quando se encontram existentes mais do que 4 imagens foi então decidido colocar um filtro de desfoque sobre a ultima imagem e colocar por cima desta quantas mais imagens existem para mostrar.



1.3.10 Apresentação de Imagens em carrossel

A apresentação de imagens deverá permitir que o utilizador as visualize em ponto grande conseguindo também realizar diversas ações sobre estas, para isso foram experimentadas diversas bibliotecas, mas nunca se conseguia o comportamento desejado, sendo assim foi decidido 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, sendo assim 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 que visto 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 em scroll horizontal. Para resolver tal problema foi decidido que quando dois dedos são detetados no ecrã a navegação horizontal fica bloqueada e assim que estes são levantados, a navegação horizontal é ativada novamente.

1.3.11 Carregamento de Imagens

O carregamento de imagens do dispositivo poderá ser realizado por meio de seleção de imagens da galeria, para realizar este tipo de seleção primeiramente foi testada a biblioteca `image_picker`, mas esta biblioteca utiliza o seletor de ficheiro do dispositivo, o problema é que este tipo de seleção permite também a seleção de ficheiros também o que faz com que seja necessário um conjunto de outras verificações para garantir que apenas as imagens são selecionadas o que levaria a uma possível perda de performance e perda de qualidade na experiência de utilização do utilizador.

Sendo assim foi de seguida testada a biblioteca `advance_image_picker`, mas o problema desta biblioteca é que não permite selecionar vídeos, pelo que foi decidido experimentar a biblioteca `wechat_asset_picker`, esta biblioteca cria uma página de seleção de imagens própria para seleção de imagens e vídeos diretamente da galeria, esta permite indicar o limite máximo de seleção e os tipos de ficheiros que o utilizador poderá selecionar de forma a que apenas os tipos aceites são mostrados, acrescentando também que esta biblioteca permite visualizar a tipagem de cada ficheiro no próprio objeto, o único ponto desvantajoso é que não é possível traduzir o botão de confirmação de seleção.

Após a carregar os ficheiros para memória, estes são então enviados para o `firestorage` como mencionado anteriormente.

1.3.12 Videos

A grande dificuldade encontrada com os vídeos foi o facto de ser necessário um comportamento diferente nestes quando se encontram no ecrã de visualização de imagens e vídeos, pois ao contrário de imagens, os vídeos necessitam de um player, sendo sempre necessário verificar qual o tipo de ficheiro antes de carregar o widget do mesmo.

Para resolver o problema de utilizar um player foi primeiramente testada uma biblioteca que permite a utilização do player nativo do dispositivo, ou seja, android utilizaria o player do android e ios utilizaria o player de ios. O grande problema com esta solução é que o player de android tem os botões completamente brancos, sem nenhum tipo de fundo para os destacar significando isto que se um video branco fosse visualizado, o utilizador não conseguiria visualizar os botões do player.

Após o problema anterior foi decidido desenvolver o player próprio, após a implementação de funções como, pausar, resumir video, avançar 5 segundos e voltar 5 segundos, assim como também esconder e mostrar os botões, existiam 2 grandes problemas, sendo estes mostrar o video em ponto grande e retornar para o mesmo tempo de video em ponto pequeno e também o comportamento do player não era completamente fluido.

Após alguma pesquisa foi percebido que o player de ios resolvia os problemas do player de android através da colocação de um fundo nos botões do player. Através da biblioteca appinio, é possível utilizar o player nativo de ios em android, sendo também possível configurar este. Sendo assim foi decidido utilizar o player de ios em ambos os sistemas operativos resolvendo assim todo o problema. Este player também permitiu a resolução de um problema menor que era a visualização de vídeos em ponto grande sendo que dependendo a orientação do vídeo o player alterava a orientação da aplicação voltando à orientação vertical uma vez que se termina a visualização do vídeo em ponto grande.

1.3.13 Links

Uma funcionalidade da aplicação necessária é também a utilização de links, para isto programação mobile oferece duas soluções, 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 2 soluções, implementação do firebase no próprio backend ou então uma chamada ao firebase utilizando uma chave de pedido. Primeiramente foi testado a implementação do firebase no próprio backend, mas esta implementação surge com alguns problemas pois existem diversas configurações específicas necessárias, sendo então recomendado pelos colegas de trabalho a chamada ao firebase devido à sua simplicidade.

Sendo assim para a realização de chamadas ao firebase foi utilizado o axios, realizando assim um método post para o serviço de dynamic links do firebase, indicando o prefixo do projeto. De forma a permitir a reutilização deste código, este foi então colocado em um método onde este é chamado indicando o link desejado e os dados a enviar. O link é utilizado para por exemplo como em uma 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, sendo assim estes dados são colocados na string do link permitindo a configuração do mesmo perante diversas situações. Por fim a firebase retorna o link criado sendo este colocado no *email* desejado.

Para a implementação de frontend foi necessário primeiramente importar a biblioteca de dynamic links do firebase, sendo de seguida no código de inicialização da aplicação colocado um método para em caso de a aplicação ser aberta a partir de um link, este o ler. Quando este método lê o link primeiramente este extrai a página indicada pelo link e de seguida extrai a lista de parametros recebidos, sendo então o utilizador redirecionado para a página do link indicando como parametros os dados recebidos no link.

Aquando a testagem da implementação diversas tentativas de abertura de links foram realizadas mas sem sucesso, a grande dificuldade desta implementação é que os links dinamicos não permitem realizar debug, sendo que ou funcionará na totalidade ou então não funcionará levando que seja complicado identificar bugs, pelo que a documentação do serviço foi de grande auxilio pois 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.

1.3.14 Notificações

A implementação de notificações revelou-se ser a implementação de maior dificuldade devido a diversos imprevistos na implementação da mesma. Para implementação de notificações foi utilizado o serviço de notificações do firebase, esta implementação surge assim como os links em 2 secções, primeiramente backend e de seguida frontend.

A nível de 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 qualquer informações sobre erro, apenas o dispositivo poderá ou não receber a notificação.

Primeiramente foi utilizado o conteúdo a enviar indicado pela documentação do serviço, mas o serviço apenas retornava uma mensagem de erro, se seguida foi pesquisado outras implementações de outros utilizadores e testado, mas novamente surgia um erro, pelo que foi decidido utilizar o conteúdo indicado pelo professor de programação de dispositivos móveis, tendo este conteúdo funcionado sem qualquer indicação de erro.

No conteúdo da notificação é enviado em formato json a mensagem a mostrar na notificação e como as notificações serão sempre referentes a tópicos ou comentários, então é indicado o id do tópico, do comentário e em caso de necessidade o id do comentário pai.

Este processo de notificação foi também 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 e colocado no *email*.

A nível de frontend foi então importada a biblioteca do serviço de notificações do firebase, sendo então esta implementada de acordo com a documentação do mesmo. Sendo que é necessário detetar notificações no iniciar da aplicação, durante a utilização e quando esta se encontra em segundo plano. Sendo assim aproveitado estas deteções para implementar o direcionamento do utilizador para as páginas referentes às notificações, utilizando os dados enviados na notificação.

O grande problema encontrado com as notificações é que o icon de notificação não era mostrado corretamente sendo que ou era mostrado um quadrado escuro ou nenhum icon. A biblioteca de notificações do firebase não permite a customização do icon da mesma, pelo que foi decidido utilizar a biblioteca flutter_local_notifications, esta permite a total customização das notificações sendo então enviado como icon o icon desenhado para a

aplicação, mas mesmo assim as notificações continuavam com o mesmo erro, pelo que foi decidido realizar uma pesquisa sobre o mesmo e foi percebido que android possui um novo sistema para os icons das notificações pelo que é necessário transformar estes icons em preto ou branco com fundo transparente, sendo então de seguida tratados pelo próprio android.

Sendo assim foi realizada a transformação e novamente alterados os icons das notificações, após uma nova testagem foi percebido que mesmo assim apenas o icon da notificação expandida teria sido alterado, após uma nova pesquisa foi percebido que android necessita de 2 icons de notificação para aplicar a ambas as situações de notificação, tendo sido assim resolvido o problema em questão.

Nesta implementação apenas um problema continuou sem resolução, o problema em questão é a abertura de notificações quando a aplicação se encontra terminada, foram testadas várias soluções, mas após a leitura de documentação e de soluções de outros utilizadores, o flutter não permite a reconfiguração do comportamento de notificações quando a aplicação se encontra terminada, pelo que este problema não contém solução de momento. 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.

1.3.15 Permissões

De forma a 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 para pedir 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 para acesso à galeria, sendo que em caso de recusa não será possível aceder à galeria.

Para acesso à internet, foi realizado o passo anterior, mas esta permissão é pedida no ato de inicialização da aplicação em conjunto com as permissões de notificações, sendo que as permissões de notificações são encarregues da biblioteca de notificações do flutter.

1.3.16 Ios

Após o desenvolvimento completo da aplicação para dispositivos android, foi proposto pela Motorline testar como esta se comportava em ambiente ios, para isso esta disponibilizou acesso a um dispositivo móvel apple, um computador, uma conta de desenvolvedor e um colega de trabalho que já tinha a 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.

Após o processo de configuração o projeto foi compilado para ios, nesta primeira compilação todas a aplicação funcionava corretamente, sendo apontado pelo colega de trabalho algumas configurações de design comuns em ios como por exemplo os botões de cancelar e confirmar se encontrarem no topo do ecrã e também a explicação de como lidar com navegação uma vez que ios não dispõem de função de voltar para trás.

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

Após esta implementação foi decidido testar as notificações do sistema, pelo que foi percebido que estas não funcionavam. Para isto foi pesquisado qual seria a possível fonte do problema pelo que foi detetado que as permissões poderiam não estar configuradas, para realizar a configuração das mesmas foi necessário atribuir as mesmas permissões de android para ios, mas sendo estas agora implementadas através de um ficheiro com o nome de info.plist.

Após esta adição a aplicação foi compilada novamente, mas mesmo assim as notificações não funcionavam pelo que foi indicado pelo colega de trabalho que no envio do pedido de notificações para o firebase é necessário indicar as configurações de ios também, e foi então procurado na documentação como se enviava as configurações de ios e utilizando estas foi então alterado o pedido de notificações e também o pedido de links dinâmicos, uma vez que é um pedido do mesmo estilo, evitando assim problemas futuros.

Após esta resolução foi testado novamente, mas mesmo assim o problema persistia e após alguma pesquisa foi percebido que as notificações e links têm também de ser configurados no XCode, após esta configuração foi novamente testado e todas as notificações, links de *emails* e permissões funcionaram como planeado.

Bibliografia

- Bracha, Gilad. 2015. The dart programming language.
- vanden Broucke, Seppe & Bart Baesens. 2018. *Practical web scraping for data science*. Apress. doi:10.1007/978-1-4842-3582-9.
- Chuvakin, Dr. Anton A., Kevin J. Schmidt & Christopher Philips. 2012. Logging and log management.
- Developers, Android. 2023. Handling android app links. <https://developer.android.com/training/app-links#android-app-links>. [Abril-2023].
- Firebase. 2023. Firebase dynamic links | firebase documentation. <https://firebase.google.com/docs/dynamic-links>. [Abril-2023].
- Gamma, Erich, Richard Helm, Ralph Johnson & John Vlissides. 2009. *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.
- Halili, Festim & Erenis Ramadani. 2018. Web services: A comparison of soap and rest services. *Modern Applied Science* 12. 175. doi:10.5539/mas.v12n3p175.
- Jin, Brenda, Saurabh Sahni & Amir Shevat. 2018. *Designing web apis*. O'Reilly Media, Inc.
- Mainkar, Prajyot & Salvatore Giordano. 2019. *Google flutter mobile development quick start guide : Get up and running with ios and android mobile app development*. Packt Publishing Ltd.
- Masse, Mark. 2011. *Rest api design rulebook*. O'Reilly Media, Inc.
- Mead, Andrew. 2018. *Learning node.js development : learn the fundamentals of node.js, and deploy and test node.js applications on the web*. Packt Publishing Ltd.
- Moroney, Laurence. 2017. *The definitive guide to firebase*. Apress. doi:10.1007/978-1-4842-2943-9.
- Snell, James., Doug. Tidwell & Pavel. Kulchenko. 2002. *Programming web services with soap*. O'Reilly & Associates.
- Vanderkam, Dan. 2019. Effective typescript 62 specific ways to improve your typescript.