



**Desenvolvimento da gestão de manutenção da empresa ACATEL, Acabamentos Têxteis S.A.**

José Carlos Rodrigues Arantes,

n.º 8636

Realizado sob orientação do Professor Michael Matias

Ano letivo 2019/2020

Licenciatura de Engenharia de Sistemas Informáticos.

Escola Superior de Tecnologia do Instituto Politécnico do Cávado e Ave.

**Identificação do aluno**

José Carlos Rodrigues Arantes

Número 8636, regime diurno

Ocupação: Estudante e trabalhador.

Licenciatura em Engenharia de Sistemas Informáticos.

**Orientação académica**

Professor Michael Matias

**Informação sobre a empresa de acolhimento**

ACATEL - Acabamentos Têxteis S.A.

Rua do Barreiro, 310 - Gilmonde

4754-909 Barcelos

Orientação: Engenheiro José Carlos Catalo

“*Os analfabetos do próximo século não são aqueles que não sabem ler ou escrever, mas aqueles que se recusam a aprender, reaprender e voltar a aprender*.” Alvin Tofler (1928-2016)

### RESUMO

Toda a nossa vida está em constante mudança e também evolução. Sem exceção disso está o mundo das engenharias e tecnologias, com novas ferramentas em constante mudança.

Reflete-se também esta evolução em empresas de diversos sectores. Parte desta evolução traz consigo a informatização de sistemas obsoletos, que não cumprem normas ou regras de aceitação por parte das entidades auditoras. Assim, determinados certificados referentes ao ambiente e qualidade são necessários para o bom funcionamento de uma empresa.

A área da programação *Web* é apontada como prática fundamental para a preparação de profissionais para o mercado de trabalho.

Nesse contexto, este projeto incide na criação de uma aplicação WEB, disponível no domínio de rede da empresa, em tempo real, utilizada para facilitar a consulta, mas também alertar a necessidade de reparações dos equipamentos como por exemplo máquinas de laboração ou de ajuda à laboração.

Os objetivos estruturantes do presente relatório residem numa fase introdutória, onde é exposta a situação atual e obsoleta, que condiciona a aprovação de acreditações de futuras vistorias na empresa. De seguida, é detalhada uma análise e o processo de desenvolvimento ao ambiente apresentado. Por fim, são detalhados resultados e conclusões do projeto em questão.

**Palavras-chave (tema):** Aplicação WEB

**Palavras-chave (tecnologia):** ASP.NET, Angular, cliente - servidor

### ABSTRACT

We live in a time of great change and development. The world of engineering and technology is not an exception, with our tools constantly changing.

This progress is also reflected in many sectors of different companies. Part of this is linked with the informatization of old systems which do not comply with standards or rules of the accrediting associations. These standards and rules are necessary to keep costumers, and essentially in this case to ACATEL, which hosts the organization.

The area of Web programming is considered as a fundamental practice for the preparation of professionals for the job market.

In this context, this project focuses on the creation of a real- time Web application, available in the company’s network to facilitate consults, but also to alert to the need of equipment repairs (working machines or working aids).

The main objectives of this report focus, in an early stage, on the exposure of the current and outdated situation that conditions to the approval of accreditations for future inspections in the company. Ahead is a detailed analysis and the process of development for the environment. Finally, results and conclusions of the project are shown.

**Topic Keywords:** WEB application

**Tech Keywords:** ASP.NET, Angular, client-server

### AGRADECIMENTOS

A realização deste projeto final do curso de licenciatura, que tem como objetivo a obtenção do grau de Engenheiro de Sistemas Informáticos, contou como seria de esperar de grande apoio e suporte, sem os quais tornaria esta tarefa ainda mais difícil.

Em primeiro lugar, ao Professor Michael Matias por ter aceite o convite para me orientar nesta jornada, pelo seu conhecimento transmitido, por todas as horas que estivemos em videoconferência (em tempos de pandémicos), de modo a conseguir resolver problemas que iam surgindo. De um modo especial, à sua incansável disponibilidade de modo a que nunca me deixasse ficar mal. Ao Professor Michael, o meu grande apreço.

Ao Engenheiro José Carlos Catalo, pessoa que me acolheu na empresa durante a realização deste projeto e ajuda ao mostrar o esquema de funcionamento.

Ao amigo e colega de trabalho, José Carreira, trabalhador na empresa de acolhimento, pela força transmitida.

A todos os meus professores que me encaminharam nesta jornada, em especial ao professor Luís Ferreira por toda a ajuda enquanto seu aluno.

A todos os meus amigos e família.

E por fim, aos meus pais, pelo grande apoio que me deram, por todo o sacrifício que também eles passaram para que eu possa ter concluído todo este percurso.

A todos, saúde e muito obrigado.

**Indíce**

[RESUMO 5](#_Toc61091789)

[ABSTRACT 6](#_Toc61091790)

[AGRADECIMENTOS 7](#_Toc61091791)

[ÍNDICE DE FIGURAS 10](#_Toc61091792)

[ÍNDICE DE TABELAS 12](#_Toc61091793)

[Glossário 13](#_Toc61091794)

[Siglas e Acrónimos 14](#_Toc61091795)

[**1.** **Introdução** 16](#_Toc61091796)

[1.1. Contexto e objetivos 16](#_Toc61091797)

[1.2. Estado da arte 17](#_Toc61091798)

[1.3. Estrutura do documento 18](#_Toc61091799)

[**2.** **Análise do problema e metodologias -** 20](#_Toc61091800)

[2.1. Enquadramento da empresa 20](#_Toc61091801)

[2.2. Análise do problema 21](#_Toc61091802)

[2.2.1. Requisitos funcionais 22](#_Toc61091803)

[2.2.2. Requisitos não funcionais 24](#_Toc61091804)

[2.2.3. Diagramas de casos de uso 25](#_Toc61091805)

[2.2.4. Plataformas de desenvolvimento 29](#_Toc61091806)

[2.3. Servidor (API) 30](#_Toc61091807)

[2.3.2. Entity Framework 33](#_Toc61091808)

[2.3.3. CORS 33](#_Toc61091809)

[2.3.4. DTO (Data Transfer Object) 34](#_Toc61091810)

[2.3.5. ASP.NET Web API 35](#_Toc61091811)

[2.3.6. Controladores 36](#_Toc61091812)

[2.4. Angular (frontend) 39](#_Toc61091813)

[2.4.1. Introdução à ferramenta Angular 39](#_Toc61091814)

[2.5.2. Injeção de dependência (*Dependency injection*) 40](#_Toc61091815)

[2.5.3. Módulos (*Modules*) 41](#_Toc61091816)

[2.5.4. Diretivas e componentes 42](#_Toc61091817)

[2.5.5. Serviços (*services*) 43](#_Toc61091818)

[2.5.6. TypeScript 44](#_Toc61091819)

[**2.6. Convenções e boas práticas utilizadas** 45](#_Toc61091820)

[2.6.1. Boas práticas no frontend 45](#_Toc61091821)

[2.6.2. Boas práticas no backend 46](#_Toc61091822)

[**3.** **Implementação** 47](#_Toc61091823)

[3.1. Introdução 47](#_Toc61091824)

[3.2. As aplicações 47](#_Toc61091825)

[3.2.1. Estrutura da base de dados 47](#_Toc61091826)

[3.2.2. Estrutura da solução *backend* 50](#_Toc61091827)

[3.2.2. Estrutura da solução *frontend* 53](#_Toc61091828)

[3.3. Serviço para pedidos REST API autenticados 55](#_Toc61091829)

[3.3.1. Autenticação no servidor (Token) 56](#_Toc61091830)

[3.4. Permissões e segurança 57](#_Toc61091831)

[3.4.1. Permissão no servidor 57](#_Toc61091832)

[3.4.2. Permissão no frontend 59](#_Toc61091833)

[3.5. Paginação 61](#_Toc61091834)

[3.6. Templates (HTML + CSS) 64](#_Toc61091835)

[**4.** **Resultados e discussão** 65](#_Toc61091836)

[**5.** **Conclusão** 69](#_Toc61091837)

[5.1 Trabalhos futuros 70](#_Toc61091838)

### ÍNDICE DE FIGURAS

[Figura 1 - Diagrama de blocos geral 17](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091839)

[Figura 2 - Processo têxtil 20](#_Toc61091840)

[Figura 3 – Diagrama de caso de uso n.º 1 25](#_Toc61091841)

[Figura 4 - Diagrama de caso de uso n.º 2 27](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091842)

[Figura 5 - Logótipo Visual Studio 29](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091843)

[Figura 6 - Logótipo SQL Server 29](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091844)

[Figura 7 - Logótipo Visual Studio Code 29](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091845)

[Figura 8 - Logótipo Angular 29](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091846)

[Figura 9 - Modelo cliente-servidor 30](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091847)

[Figura 10 - Comunicação RESTful entre servidor e cliente (aplicação) 32](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091848)

[Figura 11 - Exemplo simplificado usando CORS 33](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091849)

[Figura 12 - Pasta DTOs do backend 34](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091850)

[Figura 13 - Método da classe PedidoManutCurativaController 34](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091851)

[Figura 14 - Arquitetura da framework ASP.NET 35](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091852)

[Figura 15 - Arquitetura da aplicação Web API 36](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091853)

[Figura 16 - Controllers da API 37](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091854)

[Figura 17 - Exemplo de um método de um controller (PedidoManutCurativasController.cs) 37](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091855)

[Figura 18 - Especificação do formato, num cabeçalho no frontend da aplicação 38](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091856)

[Figura 19 - Arquitetura da framework Angular 39](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091857)

[Figura 20 - Injeção de serviços numa componente 40](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091858)

[Figura 21 - Configuração simples do módulo raíz (por definição) 41](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091859)

[Figura 22 - Sintaxe da ligação de dados e direção do fluxo de dados 42](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091860)

[Figura 23 - Implementação da classe serviço PedidosPreventivosService 43](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091861)

[Figura 24 - Relação entre TypeScript e ECMAScript 44](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091862)

[Figura 25 - Exemplo da pasta de um serviço 45](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091863)

[Figura 26 - Criação de uma componente por comando 45](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091864)

[Figura 27 - Pasta referente ao conteúdo da componente 45](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091865)

[Figura 28 - Excerto da pasta Controllers 46](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091866)

[Figura 29 - Excerto da pasta DTO 46](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091867)

[Figura 30 - Definição de um método DTO, classe IntervencaoPreventivasController 46](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091868)

[Figura 31 - Visão geral da aplicação. 47](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091869)

[Figura 33- Base de dados (por definição) 48](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091870)

[Figura 34 - Base de dados da aplicação 49](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091871)

[Figura 35 - Criação do projeto (backend) 51](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091872)

[Figura 36 - Aspeto geral da solução 52](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091873)

[Figura 37 - Detalhe da pasta SRC 53](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091874)

[Figura 38 - Aspeto geral 53](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091875)

[Figura 39 - Definição da AppModule 54](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091876)

[Figura 40 - Headers frontend 55](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091877)

[Figura 41 - Esquema de um pedido HTTP 55](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091878)

[Figura 42 - Classe Startup.Auth.cs, método CreateProperties 56](#_Toc61091879)

[Figura 43 - Tabela AspNetRoles (MS SQL Server) 56](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091880)

[Figura 44 - Pedido HTTP (Postman) http://myLocalhost/Token 56](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091881)

[Figura 45 - Método Register, controller AccountController.cs 57](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091882)

[Figura 46 - Excerto do controlador PedidosManutPreventiva 58](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091883)

[Figura 47 - Definição de um método do controlador de Equipamentos 58](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091884)

[Figura 48 - Pedido HTTP autenticação - Classe auth.service.ts 59](#_Toc61091885)

[Figura 49 - Modelo do objeto Token 59](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091886)

[Figura 50 - Classe Auth.Guard.ts, UserRoleMapping 60](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091887)

[Figura 51 - AlertaPermissaoComponent 60](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091888)

[Figura 52 - Paginação (aspeto visual) 61](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091889)

[Figura 53 - Paginação (servidor) 61](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091890)

[Figura 54 - Método onChangePage, componente PedidoManutCurativa 62](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091891)

[Figura 55 - Componente alerta-about 64](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091892)

[Figura 56 - Template HTML, componente Login 64](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091893)

[Figura 57 - Form de autenticação 65](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091894)

[Figura 58 - Janela referente aos pedidos curativos 66](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091895)

[Figura 59 - Detalhe das operações de um pedido curativo 66](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091896)

[Figura 60 - Inserção de uma intervenção dentro do pedido. 66](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091897)

[Figura 61 - Pedidos de manutenção programada atribuída 67](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091898)

[Figura 62 - Excerto da listagem de equipamentos 67](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091899)

[Figura 63 - Componente dos utilizadores 68](file:///C:\Users\José%20Carlos\Desktop\Relatorio.docx#_Toc61091900)

### ÍNDICE DE TABELAS

[Tabela 1 - Requisitos funcionais 22](#_Toc59385011)

[Tabela 2 - Requisitos não funcionais 24](#_Toc59385012)

### Glossário

**Software** -traduz-se como um programa que traz a experiência de interagir com equipamentos. Em termos técnicos, é usado para designar a parte lógica que fornece diretivas para hardwares presentes num equipamento.

**Hash** - ou cifra, são usadas para fins de segurança para criptografar por exemplo palavras-passe.

**Framework** - é uma abstração que une códigos comuns entre vários projetos de software, permite reutilizar código sem necessidade de ter de reescrever.

**Frontend** - parte da aplicação que interage diretamente com o utilizador e o servidor.

**Backend** - parte da aplicação que interage entre a base de dados e o *frontend*.

**Token** - é um sistema de transferência de dados que pode ser enviado via URL ou num cabeçalho HTTP (*header*) de maneira "segura", essa informação é assinada digitalmente.

**Header** - traduzido em português como cabeçalho, é uma componente usada nos pedidos e respostas HTTP, onde definem os parâmetros destas mesmas transações.

**Authorize** - filtro de autorização no *backend*. Caso implementado num método, só permite execução desse mesmo método caso o utilizador esteja autenticado.

**Role** – propriedade utilizada no *backend*, de forma a definir que tipo de utilizador da aplicação terá acesso a determinado método/componente.

**Web Browser** - é um programa que habilita utilizadores a interagirem com documentos HTML hospedados num servidor de rede.

**Desktop** - ambiente de trabalho (*desktop environment*).

**Design pattern** - padrão de projetos de software.

### Siglas e Acrónimos

**API** Application Programming Interface (Interface de Programação de Aplicação)

**HTTP** HyperText Transfer Protocol (Protocolo de Transferência de Hipertexto).

**SQL** Structured Query Language (Linguagem de Consulta Estruturada).

**.NET** Plataforma única para desenvolvimento e execução de sistemas e aplicações.

**LINQ** Language Integrated Query (Consulta integrada à linguagem)

**RF** Requisito funcional - representa o que o software faz, em termos de tarefas e serviços.

**RNF** Requisito não funcional - relacionado ao uso da aplicação em termos de desempenho, usabilidade, confiabilidade, segurança, disponibilidade, manutenção e tecnologias envolvidas.

**VPN** Virtual Private Network (Rede privada virtual)

**DTO** Data Transfer Object, um padrão usado em linguagens orientadas a objetos para o transporte de dados entre diferentes componentes de um sistema.

**SPA** Simple Page Application (Aplicação de página única)

**JSON** JavaScript Object Notation

**XML** Extensible Markup Language

**CORS** Cross-Origin Resource Sharing

**ORM** Object-relational mapping

**URL** Uniform Resource Locator

**IP** Internet Protocol

**MVC** Model-View-Controller(Modelo-Visualização-Controlador)

**ADO** Conjunto de classes definidas pela *framework* .NET

**IIS** Internet Information Services

**REST**Representational State Transfer

# **Introdução**

Este documento descreve o processo de implementação de uma aplicação desenvolvida no âmbito do estágio do Curso de Licenciatura em Engenharia de Sistemas Informáticos.

Este projeto no contexto de um estágio, tem como objetivo a aplicação prática dos vários conhecimentos que foram adquiridos ao longo dos 3 anos de licenciatura, aplicados num contexto de trabalho real.

O projeto foi desenvolvido com vista a resolver um problema de registo e consulta de parâmetros de avarias, e suporte de equipamentos de produção da organização de acolhimento, que até aqui era feito em suporte de papel.

O projeto aqui apresentado interliga várias áreas tecnológicas, desde base de dados, aplicações e outras tecnologias WEB, sendo posto logo de parte o uso de uma aplicação *desktop*, por várias razões, nomeadamente: sistemas *web* podem ser acedidos em qualquer equipamento (*tablet* ou computador) ligado à rede interna da empresa; facilidade de fazer alterações numa aplicação *web*, sem ser necessária alteração da aplicação em todos os computadores, como acontece em aplicações de *desktop*.

## **Contexto e objetivos**

A [Acatel - Acabamentos Têxteis S.A.](https://acatel.pt/pt/) (empresa de acolhimento), presente no mercado desde 1985, presta serviços de tinturaria, mercerização, tingimento à peça, estamparia digital e tradicional e acabamentos de malhas e tecidos. Como trabalhador nesta empresa e estudante no IPCA, surgiu a oportunidade de criar esta solução.

A existência de variadas soluções que existe no mercado de trabalho, não significa que estas se adequem à empresa em questão. Embora existam algumas soluções, estas não estão moldadas ao negócio da empresa. Uma solução desenhada e moldada à imagem da mesma, tem um maior valor acrescentado.

Assim, por ter sido uma aplicação feita de raiz, o levantamento de requisitos e a criação da aplicação com base nesses requisitos, foi muito exigente, de modo a não permitir falhas durante a sua produção em tempo real.

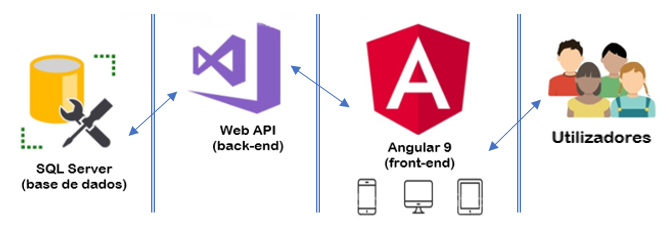
A seguinte figura representa um diagrama sobre a visão geral de todo o sistema desenvolvido.

Figura 1 - Diagrama de blocos geral

Os utilizadores interagem com uma aplicação WEB. Esta aplicação WEB, desenvolvida em Angular, é o cliente que os utilizadores utilizam para interagir com a plataforma. O cliente por sua vez faz pedidos à API, devidamente autenticados, quando num domínio privado. A API tem conhecimento de todas as regras de negócio necessárias para o registo de informação dos utilizadores na base de dados.

## **Estado da arte**

## **1.2.1 Estado da arte da empresa**

Esclarecidos os objetivos para este projeto, foi hora de analisar o que seria necessário para criar este trabalho. Como dito no capítulo anterior, o objetivo seria criar um sistema adequado às especificações da administração, num ambiente fechado (servidor interno da empresa de acolhimento, no domínio interno, afastado da rede exterior), necessário estar disponível em todos os computadores ou equipamentos móveis da empresa (telemóveis ou tablets que suportem um browser atualizado).

A empresa em questão está dividida em várias secções. Somando todo o número equipamentos fixos (computadores), são várias dezenas. A isto, soma-se o número de equipamentos móveis (*tablets, t*elemóveis).

As aplicações WEB, dependendo sempre da área a que se aplicam, estão em constante evolução, contudo, aplicações de desktop e WEB continuarão a coexistir por muito tempo, cada uma com suas vantagens e desvantagens. Neste caso, era desvantajoso instalar uma aplicação de *desktop* em todos os computadores da empresa, o mesmo também aconteceria caso fosse necessário atualizar a aplicação.

Não será vantajoso ter uma aplicação de *desktop* e querer também uma aplicação para utilizar num *tablet.* Essa solução traz consigo mais trabalho, pois é necessário desenvolver uma aplicação para cada sistema operativo móvel.

Será então vantajoso ter uma *framework* *frontend* que consiga ser utilizada através de um *browser.*

## **1.2.2 Estado da arte de tecnologias**

Foi inicialmente feita uma pesquisa sobre quais tecnologias a utilizar, de modo a satisfazer alguns requisitos: i) *frameworks* que não tragam custos extra à empresa devido a licenças; ii) *frameworks* que se adequem às restantes tecnologias da aplicação.

É importante numa empresa que sofre auditorias internas, ter certificados e licenças de softwares que usa. Existe neste momento, dois tipos de base de dados em uso na empresa, MS *Server e Pervasive* PSQL. A escolha reincidiu sobre MS *Server*, sendo um tipo de base de dados lecionada no decorrer da licenciatura. Contudo, apesar de *Pervasive* PSQLser uma boa opção, foi eliminada das opões devido ao curto intervalo de tempo para o desenvolvimento da aplicação, visto que não tenho bases sobre o funcionamento da mesma e, grande parte desse tempo terá de ser gasto a estudar uma nova *framework* *frontend*.

É necessário que a interface com o utilizador seja responsiva, dinâmica e sem a necessidade de carregar uma página inteira, sempre que há uma mudança na lógica de negócio da aplicação. Por isso, uma boa solução passa pela utilização de uma aplicação do tipo SPA (*Simple Page Application*). São várias as alternativas a escolher neste tema, entre elas: *Vue, React* ou *Angular.*

A escolha recaiu sobre a *framework* Angular. Deve-se esta escolha ao facto de o Professor Michael Matias ter conhecimentos sobre esta tecnologia, onde pôde mais tarde, transmitir-me esses mesmos conhecimentos na disciplina Multimédia e tecnologias WEB.

Houve ainda uma pequena pesquisa, para determinar uma *framework* de apoio ao desenvolvimento da API. Centraram-se duas hipóteses, sendo uma delas ASP.NET ou *NodeJS*, porém, devido a maior experiência em ASP.NET, a escolha feita foi essa.

## **1.2.3 Estado da arte de soluções existentes**

Depois de uma pequena pesquisa, há no mercado soluções para a gestão de equipamentos, como por exemplo o *software* *ValueKeep* da empresa Primavera. Contudo, falamos de soluções que estão associadas a um certo custo extra à empresa de acolhimento, e nem sempre pode fazer sentido ter algo que não se enquadre 100% nas necessidades da empresa, é isso que fez com que esta solução fosse uma necessidade. A mesma foi moldada ao negócio da empresa, trazendo um maior valor acrescentado.

## **Estrutura do documento**

Este documento está organizado em quatro capítulos principais, além do presente capítulo de introdução.

No capítulo de análise e metodologias adotadas, é feito um estudo ao problema e às ferramentas utilizadas, onde inclui uma análise dos requisitos, funcionalidades necessárias para a aplicação e apresentados os diagramas de caso de uso. Em seguida são expostas as tecnologias e metodologias utilizadas no processo de implementação do sistema.

No capítulo de implementação estão expostos todos os sistemas desenvolvidos. Este capítulo está dividido em subcapítulos que falam sobre cada um dos sistemas desenvolvidos durante o projeto, nomeadamente a base de dados, a API (*backend*) e a interface com o utilizador (*frontend*).

No capítulo de resultados, é exposto o aspeto final da aplicação, e pequenos pormenores de validação.

Por fim é apresentada uma conclusão sobre o projeto e possíveis melhorias.

# **Análise do problema e metodologias**

## **Enquadramento da empresa**

Para o desenvolvimento da aplicação é necessário identificar as necessidades da empresa, ou seja, conhecer o *modus operandi* da organização de acolhimento, para perceber a necessidade de que dados recolher e como os registar.

A Acatel Acabamentos Têxteis S.A., é uma empresa têxtil que opera na área dos acabamentos e beneficiação têxtil. A empresa presta serviços a outras empresas na área de beneficiamento têxtil, mais concretamente em tinturaria, tingimento à peça, estamparia digital, estamparia convencional e acabamentos de malhas e tecidos.

O desenvolvimento da sua atividade industrial está dependente de vários sistemas informáticos, contudo, nenhum neste momento que seja específico para a manutenção dos equipamentos da empresa. A seguinte figura mostra a fase do processo têxtil onde a empresa de acolhimento se insere (destaque a vermelho).

Figura 2 - Processo têxtil

## **Análise do problema**

A função da empresa de acolhimento contém diferentes secções. Diferentes secções, contêm diferentes grupos de máquinas, e para cada grupo de máquina, há máquinas com funcionalidades diferentes e distintas.

A realização deste trabalho, tem por função organizar a manutenção de uma empresa que contém mais de 150 (cinto e cinquenta) máquinas, de diferentes tipo, tamanhos e funcionalidades.

O esquecimento, ou o não cumprimento da manutenção destes equipamentos pode comprometer a reputação desta empresa, visto que as encomendas têm prazos de entrega, que necessitam de ser cumpridos. O não cumprimento destes prazos devido à falta de organização da manutenção, ou da não realização de tarefas de manutenção preventiva por parte de um funcionário, de modo a prevenir avarias, pode comprometer a relação entre um cliente e a empresa de acolhimento. Por isso, torna-se necessário ter um registo.

As tarefas de um funcionário pertencente à equipa de manutenção (serralheiro ou eletricista), dividem-se em duas categorias: preventivas ou curativas.

As manutenções preventivas, são aquelas que são programadas pelo Eng.º da equipa de manutenção, necessárias para o bom funcionamento destes equipamentos, de forma a minimizar o desgaste dos mesmos.   
Por outro lado, as manutenções curativas não são programadas, e acontecem quando a manutenção preventiva pode ter falhado por algum motivo alheio, e é necessária a sua reparação, podendo esta ser notificada, avaliada e realizada.

Tornar este processo mais dinâmico através desta aplicação, melhora a classificação das normas que a empresa necessitaria de cumprir (ISO 9001)[[1]](#footnote-2)*,* requisitadas por auditorias internas.

## **Requisitos funcionais**

O levantamento de requisitos foi efetuado com atenção às funcionalidades e detalhes que seriam necessários/fundamentais a serem implementados na aplicação. Os mesmos foram separados por requisitos funcionais e requisitos não-funcionais.

Tabela 1- Requisitos funcionais

|  |  |  |  |
| --- | --- | --- | --- |
| **Requisitos Funcionais** | | | |
| **ID** | **Tag** | **Descrição** | **Prioridade** |
| RF01.1 |  | O sistema deverá permitir registar novos utilizadores:   * Login; * Password; * Nome completo; * Número de trabalhador; * Secção de trabalho; * Tipo de perfil (administrador, técnico, funcionário comum); | **MUST** |
| RF01.2 | Gestão de utilizadores | O sistema poderá permitir inativar utilizadores. | **SHOULD** |
| RF01.3 |  | O sistema poderá permitir associar o utilizador a um perfil de acesso. | **MUST** |
| RF01.4 |  | O sistema deverá conter pelo menos estes três perfis:   * Administrador; * Técnico de reparação; * Funcionário (chefe de secção de cada turno, para fazer relatar as anomalias); | **MUST** |
| RF02.1 | Gestão de Equipamentos | O sistema deverá permitir adicionar equipamentos, com os seguintes campos obrigatórios:   * Código interno e descrição; * Grupo de máquina pertencente; * Marca e número de série; * Tipo; * Data de compra e desativação; * Estado do equipamento (ativo/desativo); | **MUST** |
| RF02.2 |  | O sistema deverá permitir inativar equipamentos (equipamentos que deixam de operar, vendidos ou obsoletos). | **SHOULD** |
| RF03.1 |  | O sistema deverá permitir a criação de pedidos de assistência do tipo curativa ou preventiva. | **MUST** |
| RF03.3 |  | Os pedidos e intervenções deverão ter os seguintes estados:   * Sem intervenção; * Programado/(a); * Fechado; | **MUST** |
| RF03.3 | Gestão de pedidos de assistência | O sistema deverá para o tipo curativo conter os seguintes campos:   * Hora do pedido (introduzida automaticamente pelo servidor); * Hora da intervenção (início e fim, introduzida pelo utilizador); * Tempo de intervenção por técnico (num pedido podem ocorrer várias intervenções); * Utilizador que faz o pedido (introduzido automaticamente pelo servidor, baseado no login); * Equipamento; * Descrição da avaria; * Técnico que fez a intervenção; * Descrição da intervenção realizada; | **MUST** |
| RF03.4 |  | O sistema deverá para o tipo de assistência preventiva conter os seguintes campos:   * Hora do pedido (introduzida automaticamente pelo servidor); * Hora da intervenção (início e fim, introduzida pelo utilizador); * Horas de intervenção por técnico (num pedido podem ocorrer várias intervenções); * Utilizador que faz o pedido (introduzido automaticamente pelo servidor, baseado no login); * Equipamento; * Técnico que fez a intervenção; * Equipamento; * Descrição da intervenção realizada; * Limite de intervenção (caso seja preventiva, e programada pelo Eng.º de manutenção); | **MUST** |
| RF04.1 | Gestão de Planos | O sistema deverá permitir associar um técnico a um grupo de equipamentos para manutenções preventivas programadas. | **MUST** |
| RF04.2 |  | O sistema deverá alertar o técnico de manutenções programadas a executar. | **SHOULD** |
| RF04.3 | Gestão de planos  (continuação) | O Eng.º de manutenção, emite um pedido de manutenção preventiva, com uma data limite de manutenção para grupo de máquinas (plano de prevenção). | **MUST** |
| RF04.4 |  | O sistema deverá permitir criar um plano de manutenção anual. | **SHOULD** |
| RF05.1 |  | O sistema deverá permitir a consulta de manutenção de um determinado grupo de máquinas, alojado, em formato PDF. | **MUST** |
| RF05.2 | Consulta de registos | O sistema deverá permitir consultar intervenções anteriormente realizadas. | **MUST** |
| RF05.3 |  | O sistema deverá permitir a consulta do relatório de manutenção anual. | **COULD** |

## **Requisitos não funcionais**

|  |  |  |  |
| --- | --- | --- | --- |
| **Requisitos Não Funcionais** | | | |
| **ID** | **Requisito** | **Descrição** | **Prioridade** |
| RNF01 | Desempenho | O sistema deverá apresentar bom desempenho: respostas rápidas após clique e a carregar conteúdos. | **SHOULD** |
| RNF02 | Usabilidade | O sistema deverá apresentar uma interface simples e intuitivo, facilitando a sua utilização. | **SHOULD** |
| RNF04 | Confiabilidade | O sistema deverá estar disponível durante todo o tempo de produção. | **MUST** |
| RNF05 | Idiomas | O sistema deverá suportar o idioma português. | **MUST** |
| RNF0 | Segurança | O sistema deverá oferecer segurança. Nem todas as pessoas com acesso a um computador, podem ter acesso à aplicação. | **MUST** |

Tabela 2 - Requisitos não funcionais

## **Diagramas de casos de uso**

**Diagrama n.º 1**

A seguinte figura representa o diagrama de casos de uso de forma simplificada, referente a um pedido curativo ou preventivo simples.

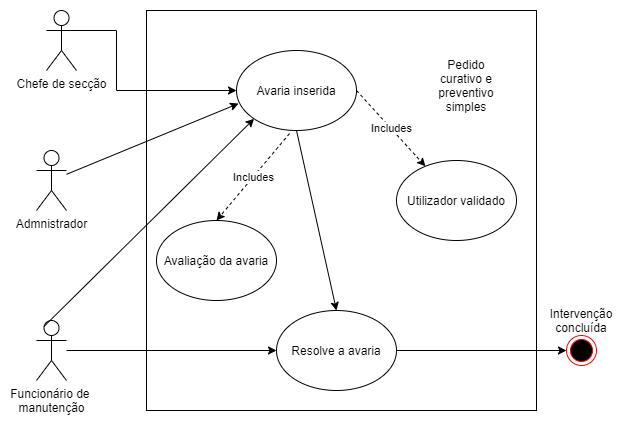


Figura 3 – Diagrama de caso de uso n.º 1

O diagrama demonstra as ações de uma parte da aplicação, nomeadamente operações de manutenção simples curativas ou preventivas.

Qualquer tipo de utilizador desta aplicação (desde que esteja registado), poderá inserir no sistema um alerta de avaria. Por outro lado, para resolver a avaria, apenas funcionários de manutenção têm acesso à componente de inserção de intervenções, de modo a deixar a avaria como resolvida.

**Tabelas descritivas do diagrama de casos de uso n.º 1**

Cada caso de uso tem uma descrição de funcionalidade que será construída no sistema proposto. De seguida são apresentadas as tabelas descritivas de cada caso de uso apresentado na figura anterior.

**AVARIA INSERIDA**

|  |  |
| --- | --- |
| **PRÉ-CONDIÇÃO** | N/A |
| **DESCRIÇÃO** | Uma entidade insere um pedido (registo) de avaria. |
| **PÓS-CONDIÇÃO** | N/A |

**UTILIZADOR VALIDADO**

|  |  |
| --- | --- |
| **PRÉ-CONDIÇÃO** | N/A |
| **DESCRIÇÃO** | O sistema verifica se esta entidade tem permissão para inserir um pedido (registo). |
| **PÓS-CONDIÇÃO** | Caso não tenha permissão, o pedido não é registado. |

**AVALIAÇÃO DA AVARIA**

|  |  |
| --- | --- |
| **PRÉ-CONDIÇÃO** | N/A |
| **DESCRIÇÃO** | Uma entidade da manutenção verifica o tipo de problema ocorrido. |
| **PÓS-CONDIÇÃO** | N/A |

**RESOLVE A AVARIA**

|  |  |
| --- | --- |
| **PRÉ-CONDIÇÃO** | N/A |
| **DESCRIÇÃO** | A intervenção foi feita e registada. |
| **PÓS-CONDIÇÃO** | O pedido é dado como resolvido no sistema. |

**Diagrama n.º 2**

A seguinte figura representa o diagrama de casos de uso de forma simplificada, referente a um pedido de manutenção preventiva programada.

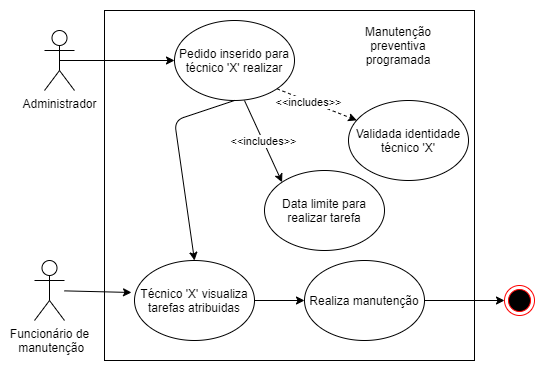


Figura 4 - Diagrama de caso de uso n.º 2

O diagrama demonstra as ações de uma outra parte da aplicação, nomeadamente operações de manutenção que o administrador solicita a determinado funcionário de manutenção. Este tipo de manutenção, é destinado para manutenção que engloba grupos de máquina com similaridades entre os equipamentos pertencentes ao mesmo.

Esta componente, é mais dedicada ao funcionário de manutenção e administrador como partes fulcrais da manutenção da empresa, sendo que um funcionário de manutenção pode e deve ver as tarefas que o administrador lhe atribuiu, para que seja realizada até uma data limite, também esta definida pelo administrador.

**Tabelas descritivas do diagrama de casos de uso n.º 2**

Cada caso de uso tem uma descrição de funcionalidade que será construída no sistema proposto. De seguida são apresentadas as tabelas descritivas de cada caso de uso apresentado na figura anterior.

**PEDIDO INSERIDO PARA TÉCNICO ‘X’ REALIZAR**

|  |  |
| --- | --- |
| **PRÉ-CONDIÇÃO** | N/A |
| **DESCRIÇÃO** | O administrador associa a necessidade de manutenção preventiva programada de um grupo de máquinas, a um funcionário de manutenção |
| **PÓS-CONDIÇÃO** | N/A |

**VALIDADA IDENTIDADE TÉCNICO ‘X’**

|  |  |
| --- | --- |
| **PRÉ-CONDIÇÃO** | N/A |
| **DESCRIÇÃO** | O sistema verifica a identidade do técnico. |
| **PÓS-CONDIÇÃO** | N/A |

**DATA LIMITE PARA REALIZAR TAREFA**

|  |  |
| --- | --- |
| **PRÉ-CONDIÇÃO** | N/A |
| **DESCRIÇÃO** | Cada pedido atribuído ao técnico ‘X’, tem uma data limite até que a manutenção seja realizada. |
| **PÓS-CONDIÇÃO** | Se ultrapassada a data limite, o técnico entra em incumprimento. |

**TÉCNICO ‘X’ VISUALIZA TAREFAS ATRIBUÍDAS**

|  |  |
| --- | --- |
| **PRÉ-CONDIÇÃO** | N/A |
| **DESCRIÇÃO** | O técnico ‘X’ tem uma área dedicada a si, onde pode consultar tarefas que lhe tenham sido atribuídas pelo administrador da manutenção. |
| **PÓS-CONDIÇÃO** | N/A |

**REALIZA MANUTENÇÃO**

|  |  |
| --- | --- |
| **PRÉ-CONDIÇÃO** | N/A |
| **DESCRIÇÃO** | O técnico ‘X’ realiza tarefas que lhe foram atribuídas. |
| **PÓS-CONDIÇÃO** | Os pedidos são dados como ‘fechados’. Técnico ‘X’ deixa ser receber alerta de pedidos em espera. |

## **Plataformas de desenvolvimento**

* O *Microsoft Visual Studio* é um pacote de programas do Microsoft para desenvolvimento de software dedicado especialmente ao .NET *Framework* e às linguagens de programação Visual Basic, C, C++, C#, e J#.



Figura 5 - Logótipo Visual Studio

* O *Microsoft SQL Server*, é um sistema gestor de bancos de dados relacional, desenvolvido em parte pela Microsoft, compatível também com *Visual Studio*.



Figura 6 - Logótipo SQL Server

* O *Visual Studio Code*, é um editor de código criado pela Microsoft, para *Windows*, *Linux* e *macOS*, sendo ideal para trabalhar em linguagens tais como *TypeScript* (TS), *JavaScript* (JS), HTML e CSS.



Figura 7 - Logótipo Visual Studio Code

* Angular é uma *framework* para aplicações WEB (frontend), baseado em *TypeScript*, reescrita de *AngularJS*, criada pela Google.



Figura 8 - Logótipo Angular

* Github é um sistema para albergar e partilhar código com controlo de versão.



Figura 9 - Logótipo da plataforma Github

## **Servidor (API)**

No início, houve uma decisão de construir uma aplicação de página única (SPA) com a *framework* angular para criar uma aplicação mais rápida e fácil de usar. Este tipo de tecnologia (SPA) trata-se de uma aplicação *WEB* que interage com o utilizador dinamicamente, atualizando uma página de acordo com novos dados do servidor, sem necessidade de carregar uma página inteira. Move grande parte da apresentação e da lógica de negócios para o *browser*, mas o servidor é, contudo, muito importante. Existem áreas em que o servidor deve estar envolvido para atingir o nível de segurança ou consistência de dados desejados

Cada aplicação trabalha com os dados que cria, modifica ou transforma e geralmente é necessário persistir esses dados para uso posterior. Além disso, nem todas as operações devem ser executadas num *browser* e, portanto, é necessário um servidor que possa lidar com as operações que a aplicação de frontend não pode ou não deve realizar. Estas são as seguintes áreas:

* Persistência de dados
* Autenticação e autorização
* Operações

Toda a aplicação é baseada num modelo de arquitetura cliente-servidor. Este modelo, descreve como um servidor fornece recursos e serviços a um ou mais clientes. Assim sendo, um servidor fornece recursos para dispositivos clientes, como ambientes *desktop* ou *smartphones*.

A maioria dos servidores tem uma relação um-para-muitos com os clientes, o que significa que um único servidor pode fornecer recursos para vários clientes ao mesmo tempo (*Christensson, Client-Server Model Definition* 2016).

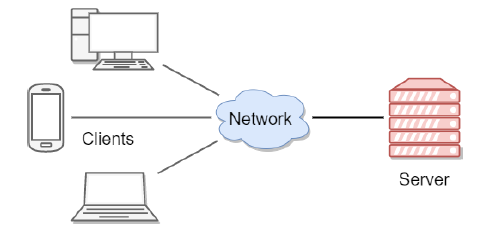
**A seguinte figura, ilustra um modelo *client-server.*

Figura 10 - Modelo cliente-servidor

O SPA é executado em dispositivos clientes para fornecer uma interface de utilizador, e a framework ASP.NET é executada no servidor para servir recursos e serviços. A comunicação entre o cliente e o servidor é realizada recorrendo ao protocolo HTTP, e a arquitetura REST (*Representational State Transfer*) é implementada para seguir as melhores práticas. A estrutura ASP.NET e a arquitetura REST serão descritas nos capítulos a seguir.

* + 1. **REST API (Representational State Transfer)**

*Representational State Transfer* (REST) é um estilo de arquitetura baseado num conjunto de princípios que descrevem como os recursos de rede são definidos e endereçados. Esses princípios foram descritos no ano 2000 por Roy Fielding como parte de sua tese de doutorado. REST é uma alternativa ao *Simple Object Access Protocol* (SOAP).

Quando falamos sobre APIs REST, palavras como URL são usadas com frequência. O *Uniform Resource Locator* (URL) é um identificador de localização deum controlador. Todos os controladores podem ser localizados por URL e deve ficar claro no URL qual controlador é endereçado.

As aplicações baseadas na arquitetura REST são frequentemente chamadas de *RESTful*. O principal objetivo REST API é ter acesso e manipular dados de maneira uniforme. Operações como GET, PUT, UPDATE ou DELETE, são frequentemente realizadas, e o nome do método HTTP define o tipo da operação desejada. A seguinte tabela mostra a relação entre o nome do método HTTP e a operação a ser realizada.

Tabela 3 - Métodos HTTP e as suas funções

|  |  |
| --- | --- |
| Método HTTP | Operação a ser executada |
| GET | Solicita a representação de um método/objeto específico. |
| POST | Utilizado para submissão de novos dados. |
| PUT | Substitui determinados dados, por novos dados inseridos. |
| DELETE | Apaga a informação relativa a determinado objeto. |

Em *RESTful*, o consumidor do serviço usa o método HTTP para definir a operação que deve ser executada. O provedor de serviços trata e processa a solicitação e envia a resposta ao consumidor para informá-lo do resultado da solicitação. Onde os códigos do estado da resposta são usados ​​para esse propósito. Os mais comuns são os seguintes:

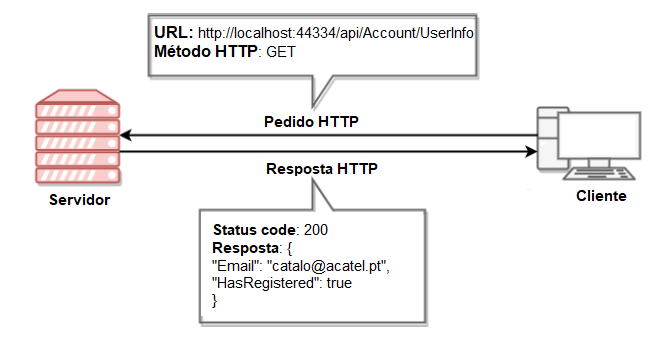
* **200 Ok** - A resposta HTTP foi concluída com sucesso.
* **201 *Created*** - A requisição foi bem-sucedida e um novo recurso foi criado como resultado. Esta é uma típica resposta enviada após um pedido POST.
* **400 *Bad Request*** - indica que o pedido do cliente não foi processado, porque o servidor não conseguiu compreender o que o cliente pede.
* **401 *Unauthorized*** - o cliente tem de efetuar *login* para obter a resposta solicitada.
* **403 *Forbidden*** - O cliente não tem direito de acesso ao conteúdo, assim o servidor rejeita dar a resposta. Diferente do código 401, aqui a identidade do cliente é conhecida.
* **404 *Not found*** - indica que o recurso pedido não está disponível.
* **500 *Internal server erro*r** - indica que a solicitação é válida, mas ocorreu um erro inesperado ao processar o pedido.

Figura 11 - Comunicação RESTful entre servidor e cliente (aplicação)

## **Entity Framework**

ADO.NET é um conjunto de classes que expõe serviços de acesso a dados para programadores da framework .NET. Este, fornece um conjunto de componentes para a criação de aplicativos de compartilhamento de dados distribuídos. É parte integrante do .NET *Framework* (ADO.NET 2017).

Por outro lado, *Entity Framework* é um ORM, um conjunto de tecnologias ADO.NET que oferece suporte ao desenvolvimento de aplicações de software orientadas a objetos. Permite consultar entidades e relacionamentos no modelo, mas também converter operações em comandos específicos da fonte de dados.

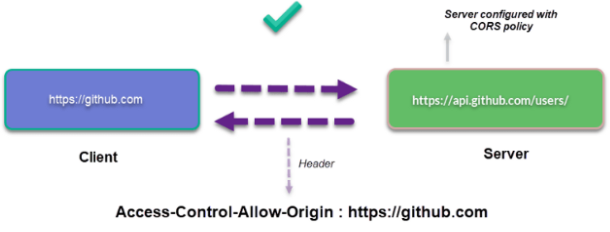
É uma ferramenta que facilita a gestão da base de dados automaticamente, sem necessidade de recorrer a outras, tais como por exemplo, *stored procedures*.

## **CORS**

*Cross-origin Resource Sharing* (CORS), é um mecanismo utilizado pelos *browsers* para partilhar recursos entre diferentes origens. Este faz parte dos cabeçalhos HTTP para informar um browser se determinado recurso pode ou não ser utilizado.

Figura 12 - Exemplo simplificado usando CORS

**4.8** / **5** ( **18** votes )

Ao enviar um pedido para uma origem diferente, o *browser* utiliza um *header* específico enviado pelo servidor chamado *Access-Control-Allow-Origin*. A partir desse e mais alguns *headers,* é determinado se esse mesmo recurso pode ser carregado ou não.

## **DTO (Data Transfer Object)**

*Data Transfer Object*, é um padrão de projetos, utilizados para transferir ou manipular dados entre subsistemas de um software.

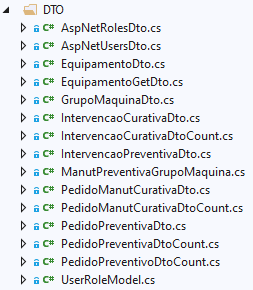
****A ideia consiste em agrupar um conjunto de atributos numa classe simples de forma a otimizar ou manipular a comunicação na forma pretendida.

Figura 13 - Pasta DTOs do backend

A imagem abaixo mostra parte de um método, que recebe informação do frontend. Neste caso trata-se de um pedido de inserção de dados, um pedido de manutenção curativa. A ajuda que um DTO nos pode oferecer neste caso, é a manipulação dos dados a nível do servidor, podendo fazer transformações ou manipulações dessa informação a ser inserida, antes que a mesma chegue ao método que insere a informação na base de dados. Trata-se neste caso da passagem de uma classe DTO (PedidoManutCurativaDto) para a classe propriamente criada pelo ADO.NET *Entity Model* (PedidoManutCurativa).

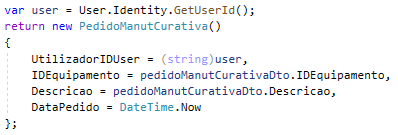


Figura 14 - Método da classe PedidoManutCurativaController

## **ASP.NET Web API**

A framework ASP.NET Web API é uma estrutura que facilita a construção de serviços HTTP, onde alcança uma ampla gama de clientes, incluindo navegadores (*browsers*) *e* dispositivos móveis. É uma plataforma ideal para construir aplicações *RESTful*.

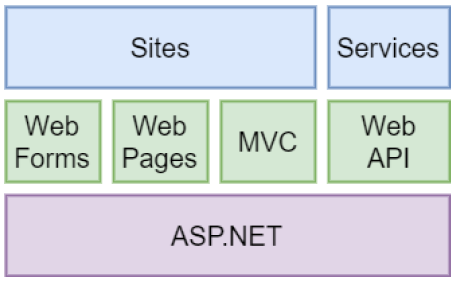


Figura 15 - Arquitetura da framework ASP.NET

API significa interface de programação de aplicação e é um conjunto de comandos, funções, protocolos e objetos que os programadores usam para criar softwares ou interagir com um sistema externo. Fornece a programadores, comandos padrão para executar operações comuns, para que não seja necessário escrever o código desde o início (*Christensson, API Definition 2016*). WEB API, como o nome sugere, é uma API na *web* e pode ser acedida por protocolo HTTP.

A ASP.NET *Web* API é uma estrutura extensível para a construção de serviços baseados em HTTP, que podem ser disponibilizados em diferentes aplicações em diferentes plataformas, tais como *web*, *Windows*, mobile, entre outros. Funciona de certa forma como ASP.NET MVC, exceto que envia dados como uma resposta em vez de uma exibição HTML (a não utilização da camada *view* referente à arquitetura MVC). É como um serviço *web* ou serviço *WCF*, mas com a exceção que só oferece suporte ao protocolo HTTP.



Figura 16 - Arquitetura da aplicação Web API

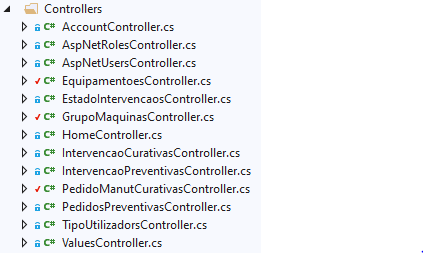
A framework ASP.NET Web API consiste em vários blocos, como definições de rota (*routing*), classes de controladores, modelo de objetos e camada de acesso a dados. Na figura acima pode-se ver como as partes da aplicação funcionam entre si. Esta arquitetura de alto nível não é fixa e pode variar de aplicação para aplicação, dependendo do efeito pretendido.

## **Controladores**

Quando a framework Web API recebe um pedido, esta encaminha o pedido para o controlador e ação (método) específicos. Um controlador é um objeto que lida com os pedidos HTTP e envia uma resposta de volta ao cliente. O controlador pode ser criado na pasta *Controllers* ou em qualquer outra pasta na pasta raiz do projeto. O nome de uma classe de controlador deve terminar em "*Controller*" e derivado da classe System.Web.Http.ApiController. Todos os métodos públicos do controlador são chamados de métodos de ação.

Para informar o cliente sobre o resultado de um pedido, uma resposta adequada deve ser enviada. Esta *framework* oferece diferentes resultados de ação integrados, que ajudam a seguir o estilo de arquitetura REST.

A seguinte imagem ilustra o conjunto de controladores presentes no projeto.

Por definição, o nosso projeto cria alguns dos controladores, sendo estes essenciais, nomeadamente o *AccountController*. Trata-se de um controlador responsável por informações relativas a utilizadores, tais como por exemplo, o registo de um novo utilizador.

Os restantes controladores foram criados de forma a responder aos pedidos HTTP, mediante a necessidade de serem utilizados.

Figura 17 - Controllers da API

A seguinte figura, retrata o exemplo de uma ação que pretende inserir informação no modelo, ou seja, na base de dados. No momento em que esta é invocada, há toda uma verificação e validação daquilo que se pretende inserir, de forma a evitar erros que possam colocar em causa a confiabilidade da aplicação, nomeadamente o vulgo *crash,* ou simples duplicação de dados previamente introduzidos. Parte desta validação pode ainda ser reforçada no *frontend* não como medida de segurança, mas como experiência do utilizador.

Figura 18 - Exemplo de um método de um controller (PedidoManutCurativasController.cs)

O modelo pode ser definido como aquilo que contém ou representa os dados com os quais a aplicação vai tratar.

A Web API do ASP.NET pode automaticamente serializar o modelo para JSON, XML (ou outro formato) e de seguida, gravar os dados serializados no corpo da mensagem de resposta HTTP. O cliente define o formato que deseja, no cabeçalho do pedido HTTP.

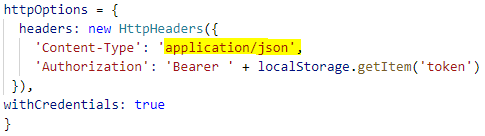


Figura 19 - Especificação do formato, num cabeçalho no frontend da aplicação

## **2.3.7 Dapper**

É uma libraria vinda do *NuGet package manager*, que pode ser associada à família de ferramentas conhecidas como um micro ORM. Estas ferramentas apenas cumprem parte das capacidades de um ORM, tal como *o Entity Framework* que é utilizado neste projeto. Contudo, é também útil para o uso de queries nativas ao invés de LINQ, mapeando queries para objetos.

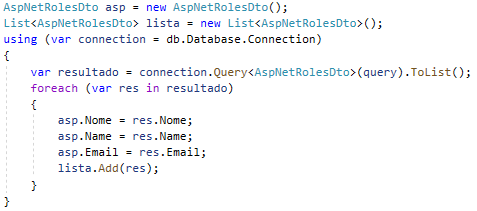


Figura 20 - Uso da libraria Dapper

## **Angular (frontend)**

## **Introdução à ferramenta Angular**

A ferramenta utilizada para interface entre utilizador e o servidor, ou seja, a camada de apresentação é baseada em *TypeScript*, é *opensource*, e é mantida pela Google e por uma comunidade de programadores no *GitHub.*

Como dito no parágrafo anterior, é baseada na linguagem *TypeScript* e fornece uma experiência mais rica, pois ajuda por exemplo a auto completar código.

Todo o processo de desenvolvimento é focado numa programação orientada a objetos, e usa os benefícios do ES2016 (abreviação de ECMAScript) tal como classes ou decoradores, ou *decorators*, são funções que modificam classes *JavaScript*. A framework Angular tem decoradores que anexam metadados às classes, para saber o que essas classes significam e como devem funcionar.

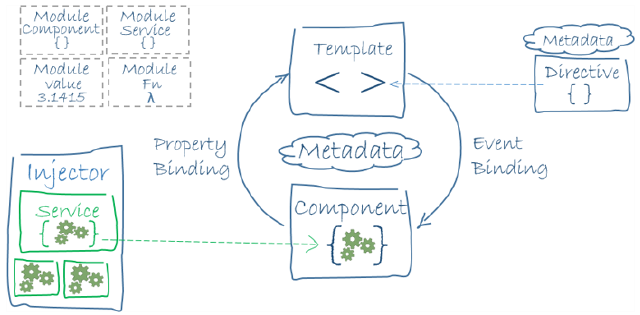
Esta framework consiste em várias bibliotecas, algumas fundamentais, outras são opcionais, mas principalmente consiste em *templates* HTML com uma marcação específica, como que “angularizada”. Consiste também em classes de componentes (*components*) para gerir os nossos templates (modelos), adicionando a lógica da aplicação a serviços (*services*), onde por fim são encaixotados as componentes e os serviços, em módulos (*modules*).

Figura 21 - Arquitetura da framework Angular

A figura acima representada, expressa os principais blocos da ferramenta Angular. Este método de separação dos *templates* HTML, classes de componentes que gerem esses templates, serviços que contêm lógica do negócio e outras componentes, leva a uma projeto arquitetónico lógico com maior reutilização e testabilidade com os blocos separados.

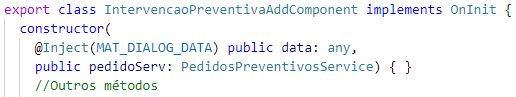
Este princípio de separação de blocos em partes distintas, é conhecido da ciência da computação como separação de conceitos (SoC, *separation of concerns).*

## **2.4.2. Injeção de dependência (*Dependency injection*)**

Injeção de dependência (DI) é uma forma de fornecer uma nova instância de uma classe, com as dependências formadas de que necessita. A maioria dessas dependências são serviços (*services*). A *framework* pode dizer quais serviços uma componente necessita, dependendo do tipo de parâmetros do construtor.

A seguinte figura mostra a injeção de serviço na componente *IntervencaoPreventivaAddComponent*.

Figura 22 - Injeção de serviços numa componente

A injeção de dependências na framework Angular consiste em duas partes chave:

* *Injector* - um serviço de gestão de dependências que cuida do registo de dependências e descobre como criá-las.
* *Dependencies* - serviços administrados pelo *injector*, são criados e fornecidos a outros serviços ou componentes.

Um *provider* é uma instrução para o sistema de [injeção de dependência](https://angular.io/guide/dependency-injection) sobre como obter um valor para uma dependência. Na maioria das vezes, essas dependências são serviços que criamos e fornecemos (*Angular guide about providers*, 2020).

Um facto sobre injeções de dependência, é que todos os serviços utilizam um *design pattern* em comum conhecido por *singleton* (existência de apenas uma instância de uma classe), o que significa que há apenas uma instância do serviço para determinado injetor.

Contudo há diferenças entre injetar *providers* num módulo ou numa componente. Quando este é registado num módulo, o serviço é fornecido em *singleton* para toda a aplicação, ao contrário, numa componente está disponível para essa mesma e para as componentes filho.

## **2.4.3. Módulos (*Modules*)**

As aplicações Angular são modulares, e Angular tem o seu próprio sistema de modularidade chamado *NgModules.* Um *NgModule*, quer seja uma raiz (root) ou não, é uma classe com um *decorator* @*NgModule* e é uma característica fundamental desta framework.

Uma classe *NgModule* descreve como é que as partes da aplicação se encaixam. Cada aplicação tem pelo menos um *NgModule*, o módulo raiz que é inicializado para a aplicação, sendo o nome convencional, *AppModule*.

A seguinte figura mostra o uso do *decorator NgModule* e a forma como o módulo é configurado.

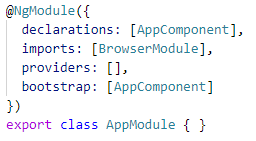


Figura 23 - Configuração simples do módulo raíz (por definição)

As propriedades mais importantes na configuração do *NgModule* são:

* *Declarations* – são as “*view classes*” que pertencem a este módulo.
* *Imports* – outros módulos (um ou mais) cujas classes exportadas são necessárias para componentes declaradas neste módulo.
* *Providers* – são criadores de serviços com os quais este módulo contribui para a coleção global de serviços (estes tornam-se acessíveis em qualquer parte da aplicação).
* *Bootstrap* – é a visualização da aplicação principal, também chamada de componente raíz.

## **2.4.4. Diretivas e componentes**

A forma como a ferramenta Angular foi criada para simplificar o aspeto de visualização, recorre a características como diretivas, elementos personalizados, interpolação, entre outros.

Os *templates* têm sintaxe HTML, enriquecida com elementos dinâmicos e, quando a ferramenta (Angular) os renderiza, transforma o DOM (*Document Object Model*) de acordo com as instruções dadas pelas diretivas.

A diretiva é uma classe com um decorador @*Directive*. São semelhantes a atributos HTML. Existem dois tipos de diretivas: as estruturais e de atributo. As diretivas estruturais alteram o *layout* adicionando, removendo e substituindo elementos no modelo de documento por objetos (DOM), enquanto as diretivas de atributo alteram a aparência ou o comportamento de um elemento existente.

Um tipo específico de diretiva é uma componente. A componente é uma diretiva com o decorador @Component, onde o *template* da componente é definido.

As componentes são como blocos principais, usados para criar a interface com o utilizador. Dentro da componente, há uma lógica definida que determina o que fazer para oferecer suporte à interface com o utilizador.

No momento de desenvolvimento de um template de uma determinada componente, há necessidade de enviar dados para a parte HTML e definir métodos para lidar com as ações do utilizador. Para simplificar, a *framework* Angular traz uma característica chamada *data bindin*g (ligação de dados).

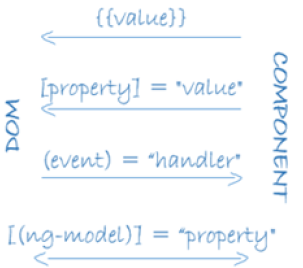
Esta ligação é um processo de sincronização entre um *template* e a sua respetiva componente.

Figura 24 - Sintaxe da ligação de dados e direção do fluxo de dados

## **2.4.5. Serviços (*services*)**

As componentes, são responsáveis por preparar os dados para a camada de apresentação e por definir métodos que serão utilizados consoantes as ações do utilizador. Por outro lado, a utilização de serviços tem o propósito de organizar o projeto de software na parte *frontend*, isolando a lógica do negócio e separando-a dos controladores.

Na prática não há diferença para o utilizador porque, provavelmente, utilizar serviços não afetará diretamente o comportamento da interface.

Um serviço é uma classe que pode ser utilizada por outros componentes do projeto para por exemplo, solicitar e manipular dados. A destacar é o que diferencia um serviço de outro tipo de componente e como um componente utiliza um serviço.

Um serviço pode ser caracterizado com o decorador @*Injectable*() e tem um propósito específico. Este *decorator* diz à *framework* Angularque o serviço em si pode ter dependências definidas no construtor de um serviço e não é necessário se um serviço não tiver nenhuma dependência.

Exemplos de serviços, podem ser os seguintes:

* Serviço de registo;
* Serviço de dados;
* Serviço de configuração;



Figura 25 - Implementação da classe serviço PedidosPreventivosService

## **2.4.6. TypeScript**

A linguagem *TypeScript* é um conjunto de funcionalidade adicionadas ao *JavaScript*. Significa que a linguagem *TypeScript* gira em torno dos avanços ECMAScript [[2]](#footnote-3) 6, onde adiciona tipagem opcional e programação orientada a objetos. Isto significa que praticamente qualquer código *JavaScript* é também um código *TypeScript*.

Benefícios desta linguagem podem ser os seguintes:

* Paradigma orientado a objetos – característica como classes, interfaces, herança, módulos, (entre outros).
* Oferece suporte a bibliotecas JavaScript - TypeScript oferece suporte a cada elemento JavaScript.
* JavaScript é também TypeScript - código escrito em JavaScript com extensão .*js* pode ser convertido em TypeScript fazendo a alteração da extensão .*js* para .*ts*.
* TypeScript é portátil - pode ser executado em qualquer *browser*, dispositivo ou sistema operativo.
* O TypeScript destaca os erros no tempo de compilação durante o momento de desenvolvimento.

Figura 26 - Relação entre TypeScript e ECMAScript

# **2.5. Convenções e boas práticas utilizadas**

## **2.5.1. Boas práticas no frontend**

Nos serviços (*services*) de uma aplicação Angular, um exemplo de convenção comum é, utilizar a funcionalidade (por exemplo nome de um objeto), seguido pelo tipo (serviço, modelo, ou outro tipo), e por fim a extensão de arquivo.

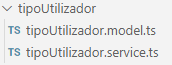


Figura 27 - Exemplo da pasta de um serviço

Na figura acima, o serviço relacionado com o tipo de utilizadores, tem dois ficheiros distintos, e fáceis de distinguir graças à convenção adotada, sendo cada ficheiro identificado com o tipo de ficheiro (se é um modelo ou um serviço).

 É importante distinguir as pastas serviços das pastas componentes, o que faz sentido para não haver confusão entre as mesmas. Assim, a prática usada, ao invés de *Camel Case,* é a utilização de hífens entre diferentes palavras, sendo esta prática ilustrada na seguinte imagem, em destaque amarelado:

Figura 28 - Criação de uma componente por comando

Assim, o resultado do comando utilizado e que é ilustrada na imagem anterior, é uma pasta com os vários ficheiros necessários para o negócio de lógica e template (HTML + CSS). A seguinte imagem ilustra a explicação daquilo que foi referido neste parágrafo.

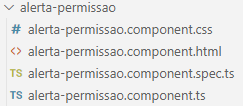


Figura 29 - Pasta referente ao conteúdo da componente

Ao contrário do que é usual em linguagens orientados a objetos, em aplicações Angular é muito usual na definição de um método ser usada a prática *lowerUpperCase.*

## **2.5.2. Boas práticas no backend**

No servidor (backend), a prática utilizada segue as normas comuns referentes a linguagens orientadas a objetos, tais como *UpperCamelCase* para classes e métodos, e *lowerCamelCase* para atributos.

Contudo, as classes controladores (*controllers)* não diferem na prática *CamelCase* porém, de seguida à funcionalidade da classe controlador, o restante nome da mesma deve terminar em ‘*controller’*, seguido da sua extensão (neste caso *.cs*).

A figura abaixo ilustra a explicação do parágrafo anterior.

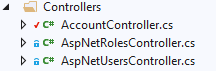


Figura 30 - Excerto da pasta Controllers

A mesma prática aplica-se às classes DTO (*Data Transfer Object*), seguida à funcionalidade da classe, o restante nome da mesma deverá terminar em ‘Dto’, seguido da sua extensão (neste caso *.cs*).

A figura abaixo ilustra a explicação do parágrafo anterior.

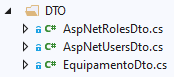


Figura 31 - Excerto da pasta DTO

Para converter objetos pertencentes a classes DTO, há que respeitar a prática adequada na criação de um método que converte esse mesmo objeto DTO para um ‘simples’ objeto.

Para isso, a boa prática utilizada consiste no nome da classe/objeto DTO que entra como parâmetro no método, seguida da palavra ‘TO’ (em português significa ‘para’), e por fim o nome da classe em queremos atribuir o DTO.

A figura abaixo ilustra a explicação anterior.

Figura 32 - Definição de um método DTO, classe IntervencaoPreventivasController

## **GitHub**

O GitHub é uma plataforma utilizada para construir e manter software, sendo também possível fazer a partilha de código para outros utilizadores.

É amplamente utilizado para manter sincronizada uma equipa de programadores, no desenvolvimento de um projeto. Assim, é utilizado para gestão de projeto, análise de código, integração e até mesmo segurança ao verificar vulnerabilidades. Contudo, também serve para manter seguro uma alteração de código, sem que diferentes versões sejam perdidas.

O uso desta plataforma permitiu guardar em segurança o projeto da aplicação, pelo que inclui base de dados, servidor e *frontend.* Serve também para outros utilizadores reagirem a modificações de código e possíveis melhorias, neste caso por parte do orientador.

Para o alojamento de código nesta plataforma, através da ferramenta *gitignore.io*, o código é então submetido pela primeira vez, sendo removidas bibliotecas das frameworks que ocupam um grande espaço e enorme número de ficheiros. Mais tarde, ao fazer *clone* do projeto, é necessário instalar de novos as dependências das frameworks.

# **Implementação**

## **3.1. Introdução**

A aplicação desenvolvida consiste em duas subpartes, devidamente distintas e separadas: a aplicação frontend, e o servidor backend. Ambas as partes usadas são independentes das tecnologias utilizadas e do ambiente de execução.

A ligação usada para a comunicação entre estas duas aplicações é a arquitetura REST API.

A aplicação do servidor utiliza *C#* como linguagem de programação utilizada, recorrendo à framework ASP.NET para criar os serviços.

Para o armazenamento de dados, foi escolhido o servidor *MS SQL* e a *framework Entity Framework* é usada como ORM para fazer a gestão da base de dados.

A aplicação frontend é escrita em *TypeScript* e usa a *framework* Angular para criar uma aplicação *user friendly*. Para tornar o desenvolvimento consistente no design da aplicação, o Angular Material foi usado.

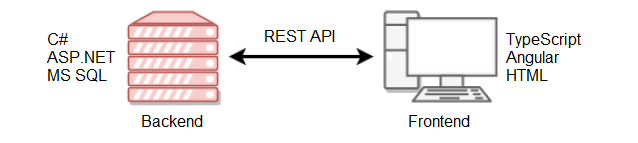
O *MS Visual Studio Code* é o compilador usado para desenvolver a aplicação frontend. É um editor de código com grande suporte para a linguagem TypeScript.

Figura 33 - Visão geral da aplicação.

## **3.2. As aplicações**

## **3.2.1. Estrutura da base de dados**

A base de dados disponível no servidor interno com licença, trata-se de SQL *Server*, tratando-se de um modelo de dados relacional. É também um bom candidato para usar em contínuo com a ferramenta *Visual Studio*, sem necessitar de drivers adicionais para estabelecer conexão entre a base de dados e a API a consumir.

Na criação de um projeto .NET *Framework* escolhido com *individual user accounts authentication,* como o nome indica tem incorporado um sistema de autenticação de utilizadores. Assim, foi escolhido um projeto do tipo Web API. Poderia ser escolhido um projeto MVC, em substituição da *view,* o usado seria a *framework* Angular.

Como referido anteriormente, por definição, este projeto acarta consigo o sistema de autenticação de contas de utilizador individuais, podendo guardar consigo dados de um utilizador, tais como email, password (encriptada sobre *hash)*.

Mais tarde neste relatório, é exposta a manipulação da classe *Register*, uma classe dedicada à criação de um novo utilizador na aplicação, para conter novos dados para além do *username* e da *password,* tais como o nome do utilizador, o tipo de permissão, entre outros.

Na figura abaixo, é mostrada a base de dados relacional para autenticação de utilizadores, contudo, não sendo necessárias todas as tabelas, esta foi adaptada de forma a coincidir com as necessidades da aplicação.

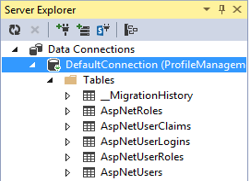


Figura 34- Base de dados (por definição)

Nesta base de dados criada por definição, as tabelas que têm aqui interesse são as seguintes:

* + - * ***AspNetUsers***: tabela que contém as informações pessoais do utilizador.
      * ***AspNetRoles****:* contém o ID do tipo de permissão e a descrição dessa permissão.
      * ***AspNetUserRoles****:* tabela que serve de interface entre as duas tabelas descritas anteriormente, tendo a informação da *role* e o IDdo utilizador.

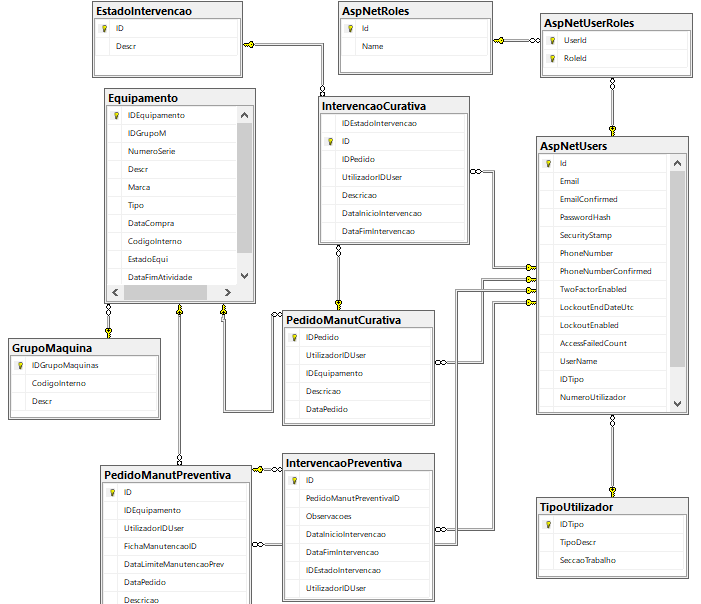
 A figura seguinte, evidencia o transporte da base de dados anteriormente citada, para a base de dados do modelo de negócio.

Figura 35 - Base de dados da aplicação

A base de dados criada de forma a servir a aplicação, para além do que já foi citado no parágrafo anterior, contém essencialmente:

* Equipamentos: são todas as máquinas presentes na empresa que têm o propósito de servir o bom funcionamento da mesma.
* Grupo de máquina: destinada a conter a informação relativa aos diferentes grupos de equipamentos da empresa, podendo haver diferentes grupos de máquina na mesma secção da empresa.
* Pedidos de manutenção curativa: destinada a conter a informação dos pedidos curativos referentes aos equipamentos.
* Pedidos de manutenção preventiva: destinada a conter a informação dos pedidos preventivos referentes aos equipamentos.
* Intervenções curativas: destinada a conter a informação relativa a um pedido curativo. Um pedido curativo pode ter mais que uma intervenção.
* Intervenções preventivas: destinada a conter informação relativa a um pedido preventivo. Um pedido preventivo pode ter mais que uma intervenção.
* Estado de intervenção: destinada a conter informação relativa ao estado em que se encontra um pedido (curativo ou preventivo). Neste caso, o estado assenta na intervenção, contudo, no *frontend,* é feita uma transformação que, baseada na última intervenção de um pedido, é mostrado o estado em que se encontra o pedido (se contém alguma intervenção, se este já foi analisado, mas ainda não reparado, ou se o problema já se encontra solucionado).
* Tipo de utilizador: destinado a conter informação relativa a um determinado utilizador, tal como a secção da empresa em que trabalha, e a sua função na empresa.

## **3.2.2. Estrutura da solução *backend***

A ferramenta usada para desenvolver o *backend*, o *Visual Studio*, permite escrever código com precisão e eficiência, sem perder o contexto do arquivo atual. É atualmente uma poderosa ferramenta de desenvolvimento.

O protocolo HTTP não é utilizado apenas para servir páginas da *web*. HTTP é também uma poderosa plataforma para construir APIs que expõem serviços e dados. Este mesmo, é simples, flexível e está ao mesmo tempo em toda a parte.

Por outro lado, ASP.NET Web API é uma estrutura para construir APIs recorrendo à .NET *Framework*.

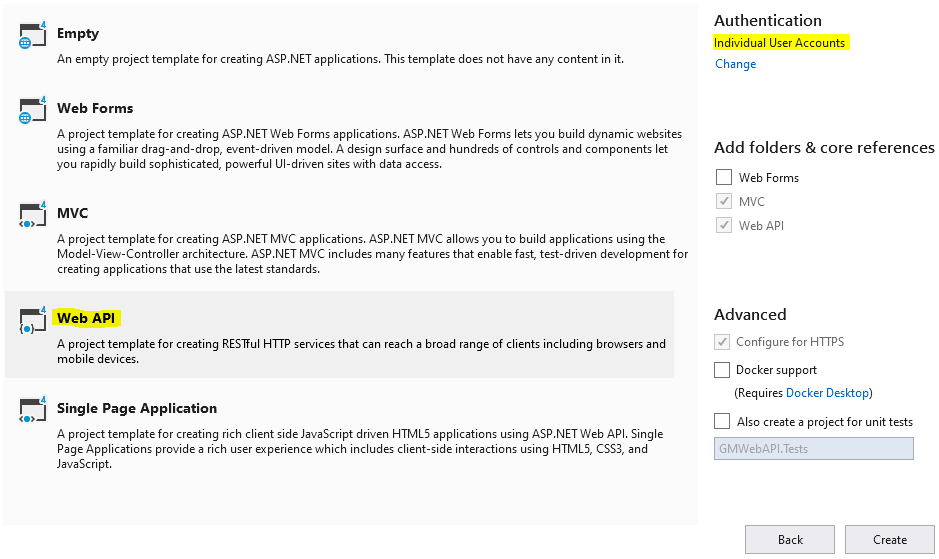
 Para este caso, o projeto é criado ao mesmo tempo que a solução, sendo atribuído o nome “GMWebAPI”*.*

Figura 36 - Criação do projeto (backend)

A figura acima, ilustra a criação da solução. De referir dois aspetos de extrema importância:

* O tipo de projeto – Web API, contém muitas semelhanças com o tipo de projeto MVC, contudo como é utilizada a *framework* Angular no frontend, não necessitamos da camada *View* referente ao tipo MVC.
* Autenticação - *Individual User Accounts*. Referente à necessidade de autenticação de utilizadores aquando da utilização da API, para aceder aos dados e métodos/recursos.

De seguida, o processo prosseguiu com a instalação do *Entity Framework*, instalado através do *NuGet package manager*. Só assim é que posteriormente é importado o nosso modelo da base de dados (ADO.NET *Entity Data Model*).

A seguinte figura ilustra o esquema da solução, de destacar as seguintes partes mais importantes para a aplicação em questão:

* Pasta *AppStart* – referente a questões relacionadas com rotas (*routing*), assim como um validador das passwords referentes aos utilizadores e por fim, protocolo *OAuth;*
* Pasta *Controllers* – contém todas as classes controladores;
* Pasta *DTO* – contém todas as classes DTOs;
* Pasta *Models* – contém o nosso modelo ADO.NET *Entity Data Model* e do objeto referente ao registo de um utilizador (propriedades);
* Pasta *Providers* – contém métodos relativos à criação de um *token* durante o tempo de execução da aplicação do lado frontend, e também características do utilizador, tais como o tipo de permissão que este tem para aceder a certas áreas da aplicação.
* Ficheiro *Web.config* – serve entre outros, para o IIS (*Internet Information Services)* ler este ficheiro e configurar quando este é hospedado. Contém também *connection string,* usado para estabelecer a ligação entre o servidor e a base de dados;

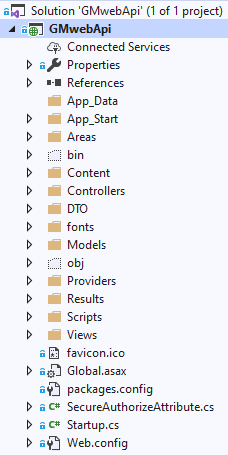


Figura 37 - Aspeto geral da solução

## **3.2.2. Estrutura da solução *frontend***

O *Visual Studio Code*, é um poderoso editor de código usado para criar a aplicação Angular. Contudo, a framework Angular precisa para compilar, construir e servir, um gestor de módulos para instalar as dependências necessárias, nomeadamente Node Package Manager (NPM[[3]](#footnote-4)).

Em primeiro lugar, o pacote Angular CLI é instalado via NPM:

* $ *npm install -g @angular/cli*

De seguida, a criação do projeto utilizando o seguinte comando:

* $ *ng new GM\_CLIENT*

O comando anterior gera a aplicação Angular, estruturada em pastas como mostrado nas figuras abaixo. A pasta *src* contém a aplicação Angular e todos os arquivos, como componentes, estilos, modelos HTML e ficheiros de configuração. Os ficheiros fora da pasta *src* destinam-se a oferecer suporte à construção da aplicação.

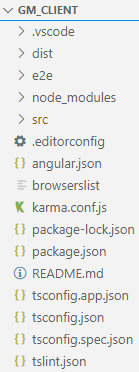
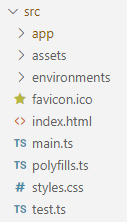


Figura 38 - Detalhe da pasta SRC

Figura 39 - Aspeto geral

Alguns dos ficheiros mais relevantes que não têm propriamente um sentido explicito no seu nome, são descritos na seguinte tabela.

Tabela 4 - Ficheiros do projeto e breve explicação

|  |  |
| --- | --- |
| **main.ts** | O principal ponto de entrada que inicializa o módulo raiz da aplicação para ser executado no *browser*. |
| **tsconfig.ts** | Ficheiro de configuração para o compilador TypeScript. |
| **index.html** | Ficheiro que é carregado primeiro quando alguém visita o *site*. A CLI adiciona todos *CLI* adiciona todos *assets* quando a aplicação é compilada. |
| **package.json** | Configuração do Node Package Manager, listando por exemplo dependências quando a aplicação é usada por terceiros. |
| **angular.json** | Ficheiro de configuração para Angular CLI. Aqui podem ser definidos vários padrões e também configurar quais ficheiros são incluídos quando o projeto é construído. |

Todos as componentes, serviços, módulos e tudo que criado por mim está localizado na pasta / src / app. É importante escolher a estratégia de estrutura certa para esta pasta, de forma a torná-la organizada e fácil de modificar a sua estrutura interna.

A pasta *src* contém um ficheiro de seu nome *AppModule*. Este deve permanecer o mais simples possível, isto porque contém a componente mãe numa hierarquia de componentes. Aqui é também feita a importação dos módulos necessários para toda a aplicação. A seguinte figura ilustra a *AppModule* da aplicação (formato reduzido, nomeadamente as *declarations* e *imports*).

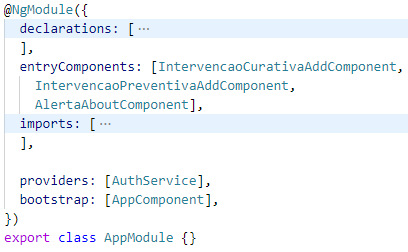


Figura 40 - Definição da AppModule

## **3.3. Serviço para pedidos REST API autenticados**

A maior parte da aplicação frontend é interligada a um servidor via HTTP. Todos os *endpoints* da API, exceto para autenticação, requerem um pedido HTTP com autenticação (email e palavra passe). Para esses pedidos, a autenticação utilizando *bearer token* foi a escolhida. Um *header* (cabeçalho) de autorização é anexado a cada pedido HTTP.

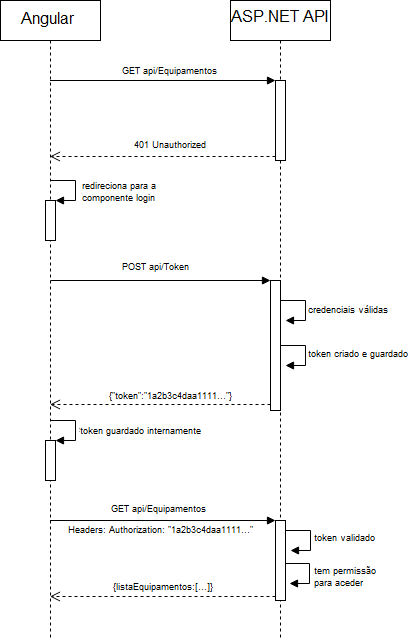
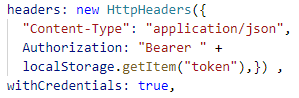
**Na figura abaixo (esquerda) é ilustrado o esquema da transição de pedidos HTTP, mostrando também o caso de autenticação, onde neste *endpoint,* não é necessária permissão para aceder.

Figura 41 - Headers frontend

Figura 42 - Esquema de um pedido HTTP

## **3.3.1. Autenticação no servidor com Token**

O sistema de autenticação (por definição) da framework ASP.NET Web API, necessitava de sofrer alterações para alcançar resultados pretendidos, nomeadamente a adição de mais propriedades relativas ao objeto que envia de retorno para o *frontend.* Tal propriedade consiste no nível de permissão que um utilizador tem, consoante a que o administrador da manutenção lhe atribuiu.

Sempre que um utilizador faz autenticação, no header do pedido HTTP, é também enviado o tipo de permissão. Esta informação extra permite que no frontend haja um duplo reforço de proteção (em capítulos posteriores, será ilustrada a proteção de conteúdos no servidor e no frontend, roles). A seguinte figura mostra parte do código necessário para alterar as propriedades enviadas no *header,* do pedido HTTP.

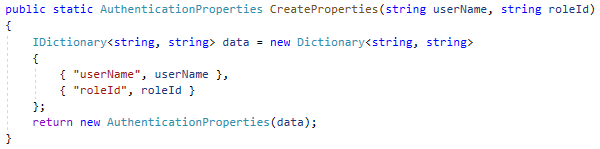


Figura 43 - Classe Startup.Auth.cs, método CreateProperties

Um teste feito com a ferramenta Postman[[4]](#footnote-5), mostra o sucesso alcançado com o código produzido, ilustrado nas seguintes imagens.

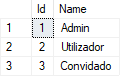


Figura 44 - Tabela AspNetRoles (MS SQL Server)

De referir, que a propriedade *roleId* ilustrada na imagem ao lado, corresponde à propriedade *Id* ilustrada na imagem acima.

Figura 45 - Pedido HTTP (Postman) http://myLocalhost/Token

## **3.4. Permissões e segurança**

## **3.4.1. Permissão no servidor**

O registo de um novo utilizador necessitou de alterações, quer seja no objeto utilizador (tabela AspNetUsers da base de dados), quer seja no modelo e objeto importado através do ADO.NET *Entity Data Model*, visto que inicialmente (por definição do controlador) para o registo de um utilizador, as propriedades necessárias eram apenas um email, password e a sua confirmação. Assim, de acordo com o desejado, novas propriedades foram acrescentadas na classe RegisterBindingModel, de forma a produzir o resultado pretendido.

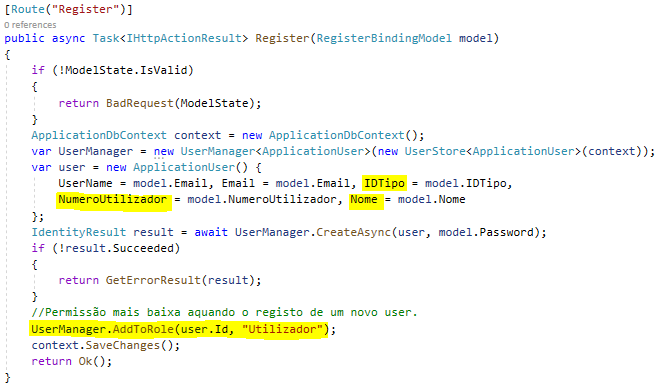
 Contudo, neste momento de registo de um novo utilizador, é também iniciado o processo de permissões que um tipo de utilizador possa vir a ter. Esta mesma permissão é que dará acesso a componentes, métodos e dados da aplicação (ilustrado a amarelo nas últimas linhas de código da figura acima). Mais tarde, apenas o administrador pode decidir qual o tipo de permissão que determinado utilizador tem. Se for o caso de atribuição de permissão mais baixa, nada necessita ser feito.

Figura 46 - Método Register, controller AccountController.cs

Estas permissões têm o nome de ‘*role’,* sendo que no servidor temos três tipos de níveis, sendo ilustrado de seguida, do maior nível de permissão, até ao mais baixo.

1. Role ‘Administrador’: Como o nome da *role* indica, é o administrador da manutenção, tendo acesso a todos os métodos e controladores principais que serão necessários na aplicação frontend.
2. Role ‘Utilizador’: Esta regra é destinada a todos os funcionários que integram a equipa de manutenção dos equipamentos da empresa de acolhimento.
3. Role ‘Convidado’: Utilizada para permitir que os chefes de secção possam inserir pedidos de equipamentos com avarias. O nome escolhido para esta regra deve-se ao facto que os chefes de secção dos diferentes setores da empresa são exteriores à equipa de manutenção, daí serem convidados.

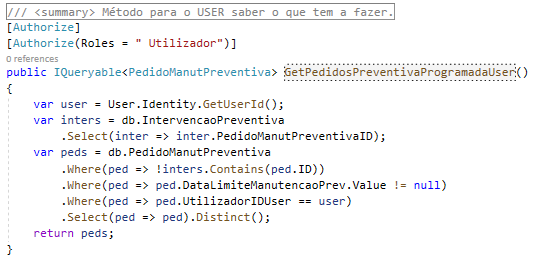
 A imagem abaixo mostra como se aplicam as regras de níveis e permissão para aceder a métodos dentro dos controladores.

Figura 47 - Excerto do controlador PedidosManutPreventiva

As duas etiquetas *Authorize* na definição do método da figura acima, são referentes às políticas de permissão e autorização no servidor.

A primeira etiqueta indica que é necessária autenticação para aceder a este método, neste caso *Bearer Token.* Por outro lado, o segundo *Authorize* dá a indicação que apenas os funcionários da manutenção podem ter acesso a este método (role utilizador). Ao contrário e ilustrado figura abaixo, todos os que utilizam a aplicação, têm acesso para aceder a esse método.

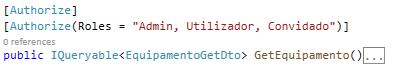


Figura 48 - Definição de um método do controlador de Equipamentos

## **Permissão no frontend**

A primeira linha de defesa contra o acesso à aplicação em geral ou a diferentes componentes, começa na camada de apresentação (frontend).

Nesta parte da aplicação, as permissões e segurança podem dividir-se nas seguintes partes:

1. Autenticação;
2. *Roles* e *CanActivate;*

A autenticação refere-se à necessidade de estar registado e autenticado para ter acesso à aplicação. A seguinte figura ilustra um excerto do método que faz um pedido HTTP, de forma a verificar no servidor se as credenciais do utilizador são válidas. Caso o pedido seja afirmativo, é enviado o *token* gerado pelo servidor e guardado localmente no frontend, assim, sempre que seja necessário efetuar um novo pedido HTTP, o *token* é enviado juntamente de forma a validar a autenticação no servidor.

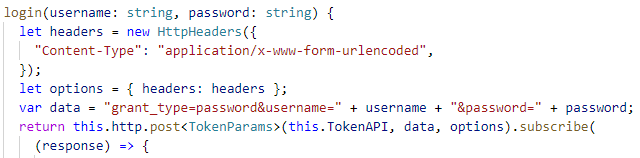


Figura 49 - Pedido HTTP autenticação - Classe auth.service.ts

Cada *token* criado pelo servidor, vem acompanhado por várias propriedades. Algumas das propriedades mais importantes são a data da sua criação e o limite de validade que este tem. Por questões de boa prática, o intervalo de validade do *token* pode ser definido no servidor mas neste caso é também manipulado na framework, de forma a pedir um novo *token* caso o intervalo de validade for ultrapassado.

A seguinte figura ilustra as propriedades do objeto ‘Token’, destacam-se as propriedades referentes à data de criação e validade do *token.*

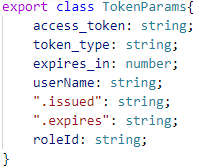
**

Figura 50 - Modelo do objeto Token

Nem todos os utilizadores têm acesso a todas as componentes. O mesmo modelo de permissões utilizado para o servidor, é também aqui utilizado no frontend, sendo que está definido para cada componente que tipo de utilizador pode ter acesso à mesma.

Para isso, é criado um tuplo no serviço *AuthGuard.ts,* especificando o *selector* da componente e o tipo de utilizador. É ilustrado na imagem abaixo, um excerto deste mapeamento.

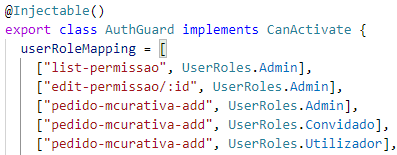


Figura 51 - Classe Auth.Guard.ts, UserRoleMapping

Caso o utilizador com sessão iniciada não tenha permissão para aceder a determinada área (componente) da aplicação, o seguinte aviso com recurso à biblioteca *MatDialog* é mostrado ao utilizador.

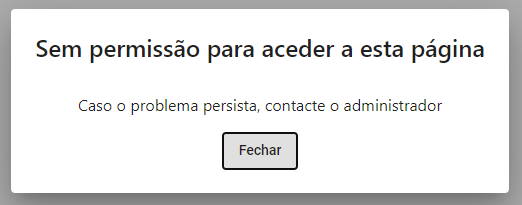


Figura 52 - AlertaPermissaoComponent

Por questões de boa prática e segurança, nenhuma das chaves primárias da tabela de utilizadores é transportada para fora do servidor. A única parte mais suscetível que é enviada para o frontend, trata-se dos emails dos utilizadores. Caso haja alguma necessidade de gerir dados de um funcionário em específico (por exemplo definir as permissões dos utilizadores), o servidor procura na base de dados qual a chave primária da tabela utilizadores, correspondente a determinado email.

Contudo, é preciso salvaguardar que o *backend* não confia em nada que venha do *frontend*. Todas as medidas de “segurança” do *frontend*, são para dar uma boa experiência de utilização ao utilizador. Sendo assim, a segurança encontra-se no servidor.

## **Paginação**

A paginação permite que grandes quantidades de informação sejam divididas entre páginas. O objetivo é evitar apresentar uma grande quantidade de dados em apenas uma página. Assim, o utilizador da aplicação não necessita de fazer *scrolldown* ou *scrollup* para conseguir ver essa grande quantidade de dados.

Traz também o benefício de melhorar o desempenho da aplicação, isto porque nas componentes, a lógica do negócio a tratar para a apresentação é menor. A quantidade de dados a transportar nos pedidos HTTP é também mais reduzida.

A seguinte imagem ilustra o aspeto visual deste assunto, implementado na aplicação.



Figura 53 - Paginação (aspeto visual)

O processo de paginação começa por um pedido HTTP ao servidor. Neste pedido, o método contém vários parâmetros de entrada, parâmetros estes que chegarão ao servidor, e mais especificamente ao controlador e ação pretendida. Os parâmetros referidos, consistem no número de página onde o utilizador pretende navegar, assim como o número de objetos pretendidos para visualizar (exemplificado na imagem acima como ‘*items per page*’).

A seguinte figura ilustra os parâmetros vindos no pedido HTTP, são eles *pedidosPerPage* e *currentPage.* São também utilizados os operadores *Skip* e *Take.*

O operador *Skip* é usado para retornar um determinado número de elementos de uma sequência (vindos anteriormente da pesquisa efetuada), e salta os restantes dessa sequência.

O operador *Take* é usado ignorar um determinado número de elementos de uma sequência e retornar o restante.

 No frontend é feita a importação do módulo *PageEvent*, na componente pretendida:

Figura 54 - Paginação (servidor)

import { PageEvent } from "@angular/material/paginator";

No momento em que o utilizador interage com o paginador, um *PageEvent* é lançado, e pode ser usado para atualizar qualquer aspeto visual de dados associado. Assim, o *PageEvent* permite alterar o *object event* que é emitido quando o utilizador seleciona um tamanho de página diferente ou navega para a outra página.

A seguinte figura ilustra o método que trata da paginação no frontend.

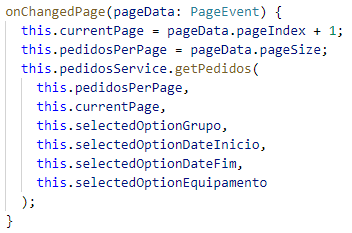


Figura 55 - Método onChangePage, componente PedidoManutCurativa

## **Templates (HTML + CSS)**

Na *framework* Angular, um *template* consiste na formação visual de interface com o utilizador. Dentro de um template, é possível usar uma sintaxe especial, aliada à parte de negócio de cada componente, de forma a aproveitar os recursos de Angular.

No momento de criação de uma componente, através do comando *ng generate class <nome>, é* criada uma classe por definição, constituída por distintos ficheiros. A principal onde é implementada a lógica de negócio relativa a essa componente e ainda, dois ficheiros relativos ao template dessa componente, um ficheiro relativo a HTML e o ficheiro CSS, que dá a formatação desse código HTML. A seguinte figura ilustra o seletor de uma componente (como se tratasse de uma chave primária da componente), seguida dos caminhos identificadores do template HTML e customização CSS.

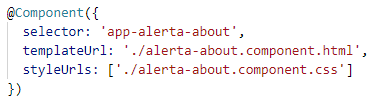


Figura 56 - Componente alerta-about

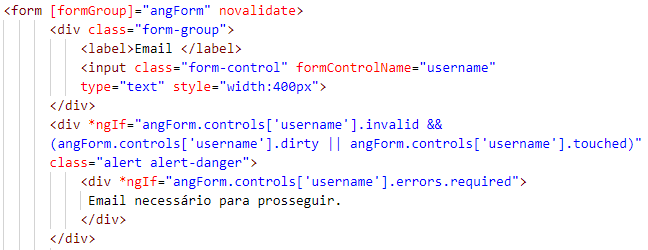
 Um *template* Angular, renderiza a camada de apresentação, ou interface do utilizador no navegador, tal como o código HTML comum, porém com funcionalidades extra. Tais funcionalidades trazem assim uma interface responsiva e agradável ao utilizador. A seguinte figura ilustra um pequeno excerto de código HTML, utilizando como exemplo, diretivas (NgIf), bibliotecas de Angular, e também *databinding* (*username* e *password*)

Figura 57 - Template HTML, componente Login

# **Resultados e discussão**

Foram efetuados testes a eventuais pedidos HTTP por parte do frontend, sendo que a ferramenta *Postman* foi indispensável para esse processo. Os testes foram sendo efetuados ao mesmo tempo que o código foi desenvolvido, de forma a manipular da melhor forma, as classes DTO, controladores e ações dos mesmos.

A criação de código desenvolvido na aplicação frontend levou a constante execução de testes entre base de dados, servidor e *frontend*.

Para a análise realizada na aplicação frontend, foi necessário recorrer à ferramenta *DevTools,* útil para depurar código web, integrada diretamente no navegador Google Chrome.

O primeiro passo para ter acesso à aplicação, é a autenticação de um utilizador previamente registado, o resultado visual é ilustrado na imagem abaixo.

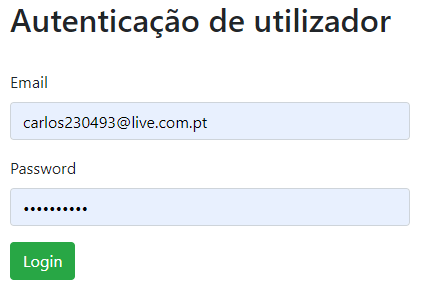


Figura 58 - Form de autenticação

Nas componentes de pedidos curativos e preventivos, o recurso da biblioteca *MatExpansionModule* e *MatAccordion,* foram peça fulcral para ilustrar esses mesmo pedidos. Ao clicar na célula correspondente a um pedido, são ilustradas as intervenções até então efetuadas nesse pedido. A seguinte figura ilustra a representação visual de um grupo de pedidos curativos, sem filtros de pesquisa ativados.

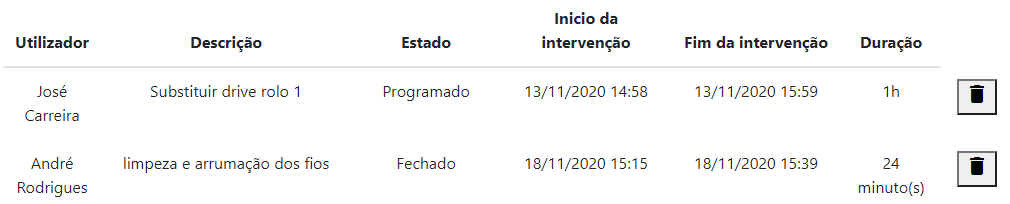
 A seguinte figura ilustra o detalhe de uma célula (cada célula corresponde a um pedido), onde cada pedido pode ter zero, uma ou mais que uma intervenção.

Figura 59 - Janela referente aos pedidos curativos

Figura 60 - Detalhe das operações de um pedido curativo

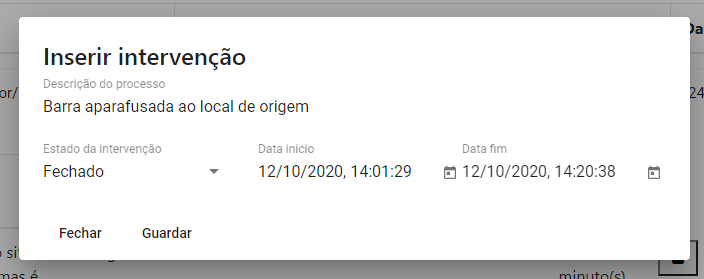
Para inserção de uma intervenção (curativa ou preventiva), é aberto uma *form*, da biblioteca *MatDialog*, inserindo essa intervenção ao pedido específico. O aspeto visual é ilustrado na imagem abaixo.

Figura 61 - Inserção de uma intervenção dentro do pedido.

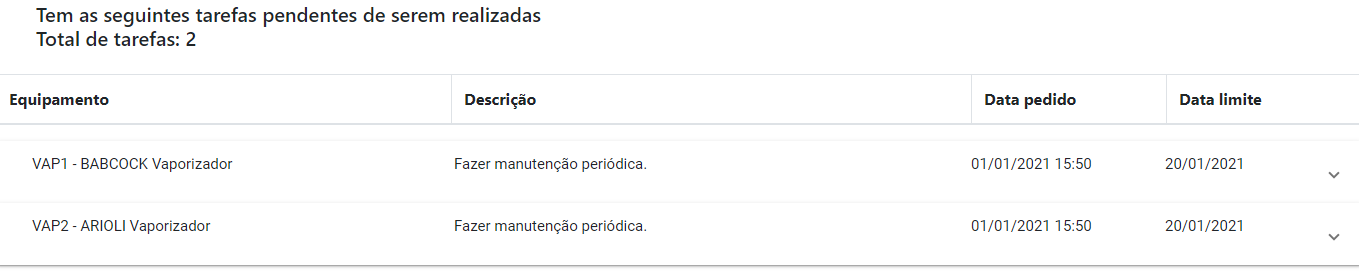
A área dedicada a pedidos de manutenção preventiva programada, referentes a tarefas que foram atribuídas a um utilizador por parte do administrador, estão redirecionadas para uma outra componente, de modo a dividir e distinguir os tipos de pedido de manutenção (preventivas, curativas ou preventivas programadas). A seguinte figura, ilustra o resultado que um funcionário de manutenção visualiza, caso tenham tarefas pendentes.

Figura 62 - Pedidos de manutenção programada atribuída

Relativamente aos equipamentos, estes podem ser consultados por qualquer funcionário, independentemente da sua permissão. Os equipamentos que já não estão em funcionamento estão destacados a vermelho e amarelo, tal como ilustra na seguinte imagem.

Figura 63 - Excerto da listagem de equipamentos

A área reservada ao registo de novos utilizadores, apenas pode ser acedida por funcionários que tenham permissão determinada pelo administrador. Aqui é feita a listagem dos utilizadores da aplicação, sendo também ponto de partida para registar novos utilizador ou mudar e visualizar permissões de cada utilizador.

A componente aqui referida é ilustrada na seguinte figura.



Figura 64 - Componente dos utilizadores

Foram encontrados problemas de compatibilidade em navegadores Web desatualizados, em computadores que estavam a tornar-se obsoletos.   
A solução tornou-se em alguns casos, atualizar as versões dos navegadores, assim como as versões .*Net* instaladas.

Inicialmente foram resolvidos problemas relativos a barramentos de *Firewall¸* resolvendo o problema com o desbloqueio das portas relativas ao IP fixo reservado à aplicação (portas 8005 do servidor e 8006 do frontend), contudo, aplicação não foi imune a interferências por parte do servidor que está alojado, visto haver uma demora de alguns segundos quando se tenta fazer *login* pela primeira vez no computador.

Neste projeto, implementado desde novembro de 2020 no servidor da empresa não se encontraram problemas ou inconvenientes até então. Foi nesse mês de novembro implementado, a pedido do supervisor da empresa de acolhimento, visto que a empresa seria alvo de auditorias internas nesse mesmo mês. Nesse mesmo momento, auditores internos relativos aos sistemas de gestão de qualidade (norma ISO 9001) tiveram acesso à aplicação, sendo recebida com entusiasmo

Como este sistema foi desenvolvido de raiz foi necessário executar imensos testes aceitação durante o desenvolvimento do mesmo, entre as três subpartes do sistema (base de dados, servidor, frontend), aplicando um maior esforço em termos de tempo, contudo, para todos os problemas encontrados durante os testes, foram sempre encontradas soluções.

# **Conclusão**

O principal objetivo deste trabalho foi desenvolver uma aplicação composta por dois projetos distintos e duas tecnologias diferentes. O objetivo da aplicação era ajudar nas reparações e inspeções de manutenção, fornecendo funcionalidade para registrar as falhas na empresa e documentá-las. Vários submódulos para o gerir utilizadores (e suas permissões), mas também equipamentos foram necessários adicionar.

No início, fui desafiado pelas novas tecnologias, como a *framework* Angular e a estrutura ASP.NET. Ao explorar e reunir novos conhecimentos sobre essas tecnologias, tais como documentações oficiais por foram também necessárias. Tanto a Google (Angular) e a *Microsoft* (ASP.NET), disponibilizaram bastante documentação API detalhada. Além da documentação oficial, muitos artigos foram encontrados sobre essas estruturas.

Nas primeiras semanas a tentar aprofundar o conhecimento em novas tecnologias, comecei a projetar e a implementar a aplicação. Inicialmente foram tomadas decisões arquitetónicas que não tiveram sucesso, sobretudo pela necessidade de conhecer melhor as frameworks. Com o tempo descobri novas soluções, que foram úteis para melhorar a estrutura de ambas as partes (ASP.NET + Angular).

Adquiri conhecimento sobre o design RESTful e arquitetura cliente-servidor, onde a ferramenta de desenvolvimento *MS Visual Studio* ofereceu um grande suporte com modelos para gerar várias classes e objetos.

O resultado de vários meses de trabalho, é uma aplicação que atende aos objetivos definidos da melhor forma encontrada. No entanto, o desenvolvimento da aplicação pode sofrer novas melhorias, que serão estruturadas no capítulo seguinte.

Como apreciação final conclui-se que este projeto foi um grande contributo para a aplicação e aperfeiçoamento dos conteúdos lecionados no decorrer da licenciatura.

## **5.1 Trabalhos futuros**

Como desafios futuros sugere-se o aperfeiçoamento dos sistemas desenvolvidos em termos de performance, segurança, funcionalidades e interação assim como a correção de problemas não detetados até ao momento.

Em geral o projeto de desenvolvimento de uma aplicação desta natureza (cliente-servidor), tem várias potencialidades para a aplicação no mercado podendo ser também útil para outras empresas, por isso é necessário garantir a funcionalidade e fiabilidade dos subsistemas. O processo de testes e melhoramento demora tempo, logo como trabalhos futuros sugere-se também testes mais exaustivos aos subsistemas desenvolvidos, bem como o seu possível melhoramento.

Especificamente, sugere-se:

* Uso de *AutoMapper,* para ajudar a resolver um problema enganosamente complexo, que é o mapeamento de um DTO para uma entidade.
* Melhoramento do *HTTPInterceptor* para interface de erros ou validações com o utilizador.
* Alojar no servidor, os guias de procedimento (.*PDF)* utilizados pelos funcionários de manutenção, durante a manutenção programada periódica (já contém a base de dados preparada esta solução).
* Remover o método de eliminação de uma intervenção. A eliminação acontece quando os dados a apagar, foram anteriormente inseridos pelo mesmo utilizador. Sugere-se avaliar se esse cenário poderá continuar a fazer sentido continuar assim ou não. Uma alternativa é manter os dados e marcar esses mesmo dados como se fossem eliminados.

**Bibliografia**

**COMPLETAR**

1. ISO 9001 – sistemas de gestão de qualidade. [↑](#footnote-ref-2)
2. ECMAScript é uma linguagem de programação, usada em tecnologias *web* para a criação de scripts executados no cliente (frontend) e no servidor (backend).  [↑](#footnote-ref-3)
3. Node Package Manager – gestor de pacotes para a linguagem JavaScript. [↑](#footnote-ref-4)
4. Postman – aplicação utilizada essencialmente para facilitar teste de código desenvolvido numa API. [↑](#footnote-ref-5)