



SiGeCur – Sistema de Gestão de Recursos

2022 / 2023

1080777 José Carlos Santos



SiGeCur – Sistema de Gestão de Recursos

2022 / 2023

1080777 José Carlos Santos



Licenciatura em Engenharia Informática

Janeiro 2023

Orientador ISEP: **Mário Miguel Cordeiro**

«Família »

Agradecimentos

Com a finalização deste Relatório de Estágio Curricular não posso deixar de agradecer a algumas pessoas que, direta ou indiretamente, me ajudaram nesta caminhada tão importante da minha vida pessoal e profissional.

Começo por agradecer aos meus familiares e amigos pelo seu apoio. Ao longo destes três anos que, enquanto trabalhador-estudante, sempre compreenderam as minhas ausências e me apoiaram neste percurso que irá contribuir para o meu sucesso pessoal e profissional.

Um agradecimento especial a minha namorada, Sofia Pereira, pelo companheirismo, paciência, disponibilidade e incentivo ao longo desta árdua caminhada.

De seguida, um agradecimento ao Instituto Superior de Engenharia do Porto e a todos os colaboradores que nele empregam e que tive a oportunidade de conhecer e que direta ou indiretamente, influenciaram o meu percurso académico.

Resumo

Este projeto surge da necessidade de implementar um sistema informático para melhorar o processo de gestão e recursos (tecnológicos, intangíveis, de informação, humanos e de uma forma global também financeiros) de uma empresa prestadora de serviços no sector automóvel, usando a tecnologia para tornar mais eficiente e rentável a sua forma de trabalhar.

- Recursos tecnológicos: software e sistemas de informação.
- Recursos intangíveis: relações com clientes.
- Recursos de informação: dados, estatísticas, pesquisas dos serviços efetuados para o cliente
- Recursos humanos: funcionários
- Recursos financeiros: dinheiro, investimentos.

Os colaboradores desta empresa trabalham com regtos em papel, deslocações ao escritório para entregar relatórios, como também é realizado o processamento dos mesmos para os passar para ficheiros de Excel, isto faz com que por vezes por excesso de trabalho os relatórios efetuados no fim de cada serviço só sejam entregues ao cliente cerca de um a dois dias depois de ter sido efetuado o trabalho, acabando causar constrangimentos para o cliente por não receber a informação atempadamente.

No âmbito deste projeto foi desenvolvido um sistema de gestão de recursos tirando partido de tecnologias existentes de forma a melhorar a eficiência do processo e reduzir os custos finais e tempo despendido quer por auditores como por administrativos.

Este projeto que será utilizado como prova de conceito permite ao utilizador, de acordo com o seu perfil de utilizador, criar relatórios, adicionar pedidos de serviço e ajudas visuais, bem como funcionalidades gerais como consultar, editar ou eliminar relatórios. Para além disso o sistema permite também a consulta por parte do cliente todos os serviços efetuados.

Como prova de conceito a solução desenvolvida é funcional, rápida e de fácil utilização. Além disso, proporciona uma forma mais eficiente de reduzir tempo e custos, pretendendo realizar uma validação de tecnologia que permitirá avaliar e estimar com maior detalhe o desenho e implementação de uma solução final por parte do departamento de IT da empresa.

Palavras-chave (Tema): Web Application, Gestão de recursos, ERP (Enterprise Resource Planning), Redução de custos

Palavras-chave (Tecnologias): ASP.NET, RestFull, Angular, Clean Arquitheture,

Índice

1	<i>Introdução</i>	1
1.1	Enquadramento/Contexto	1
1.2	Descrição do Problema.....	2
1.3	Estrutura do relatório.....	5
2	<i>Estado da arte</i>	7
2.1	ERP.....	7
2.2	Trabalhos relacionados	8
2.3	Tecnologias existentes	18
3	<i>Análise e desenho da solução</i>	35
3.1	Domínio do problema	35
3.2	Requisitos funcionais e não funcionais.....	37
3.3	Desenho	42
4	<i>Implementação da Solução</i>	49
4.1	Descrição da implementação.....	49
4.2	Testes.....	58
4.3	Avaliação da solução.....	62
5	<i>Conclusões</i>	62
5.1	Objetivos concretizados	63
5.2	Limitações e trabalho futuro	64
5.3	Apreciação final	65
Referências		67
Anexo A	<i>Modelo de Domínio</i>	72

Índice de Figuras

Figura 1 - Exemplo de relatório utilizado	2
Figura 2 - Diagrama de Gantt, planeamento projeto	5
Figura 3 - História ERP [1].....	8
Figura 4 - Magic Quadrant Gartner ERP [3].....	9
Figura 5 - Pesquisa ERP produzido em Portugal.....	10
Figura 6 - Pesquisa alargada sem necessidade de idioma português. [5]	11
Figura 7 - Pesquisa alargada com idioma português.....	12
Figura 8 - Comparação entre as 4 primeiras soluções.	13
Figura 9 - Padrão de arquitetura em camadas [15].....	19
Figura 10 - Pedido de acesso por camadas fechadas (closed layers) [15].....	20
Figura 11 -Fluxo de solicitação, camadas abertas (Open layers) [15]	21
Figura 12 - Event-driven architecture, topologia mediator [15] [16].....	23
Figura 13 - Event-driven architecture: Topologia Broker [15].....	23
Figura 14 - Onion/Clean Architecture [17]	24
Figura 15 - The clean architecture cone [17].....	25
Figura 16 - Popularidade dos protocolos REST vs SOAP [21]	27
Figura 17 - Exemplo em JSON [22]	28
Figura 18 - Exemplo em XML [22].....	29
Figura 19 - Resultado obtido pela TechEmpower dos 3 melhores Frameworks [24]	31
Figura 20 - Popularidade Frameworks Front-end [25]	32
Figura 21 - Modelo Domínio.....	36
Figura 22 - Diagrama de casos de uso utilizador nível 0	40
Figura 23 - Diagrama de casos de uso utilizador nível 1 e 2.....	40
Figura 24 - Diagrama de casos de uso utilizador nível 3	40
Figura 25 - Diagrama de casos de uso utilizador nível 4 e 5.....	41

Figura 26 - Diagrama de Vista Logica Nível 2.....	43
Figura 27 - Diagramas de vista de implementação e lógica Nível 3	43
Figura 28 - Diagramas de vista de lógica Nível 3, Visual_UI	45
Figura 29 - Diagrama vista de processos nível 3, UC1	46
Figura 30 - Diagrama vista de processos nível 3, UC9	47
Figura 31 - Controller WorktAuthorization	50
Figura 32 - Service WorktAuthorization	51
Figura 33 - Repository Employee.....	52
Figura 34 - EntityTypeConfiguration Employee.....	53
Figura 35 - ValueObject PhoneNumbertype.....	53
Figura 36 - Class Employee	54
Figura 37 - Component Report (Angular)	55
Figura 38 - CustomerService (Angular).....	56
Figura 39 - View Adicionar Employee.....	57
Figura 40 - View Pesquisar Report.....	57
Figura 41 - Testes unitários classe Employee	58
Figura 42 - Testes unitários classe WorkAuthorizationService	59
Figura 43 - Testes unitários classe ReportService	59
Figura 44 - Resultados teste de Integração	60
Figura 45 - Exemplo de um HTTP(POST).....	61
Figura 46 - Exemplo de um HTTP(GET).....	61
Figura 47 - Modelo de Domínio.....	72

Índice de Tabelas

Tabela 1 - Glossário de termos.....	36
Tabela 2 - Tipos de utilizador e aceso ao sistema.	39
Tabela 3 - Objetivos gerais concretizados	63
Tabela 4 - Objetivos específicos concretizados.....	64

Notação e Glossário

API	Application Programming Interface.
ERP	Planeamento de Recursos Empresariais (do inglês, Enterprise Resource Planning)
Framework	Framework é um modelo de códigos já existentes para uma função específica necessária ao desenvolvimento de outros softwares.
HTTP	Protocolo de comunicação utilizado para comunicações entre computadores.
MRP	Planeamento de necessidades de material (do inglês, Material Requirement Planning)
MVP	Minimum Viable Product
PME	Pequena Media empresa
REST	Representational State Transfer
SaaS	Software como serviço (do inglês, Software as a Service)
SiGeCur	Sistema de Gestão de Recursos
SOAP	Simple Object Access Protocol
SO	Sistema operativo
URI	Uniform Resource Identifier
WS	Serviço Web (do inglês <i>Web service</i>)

1 Introdução

O projeto/estágio (PESTI) é uma unidade curricular da Licenciatura em engenharia informática (LEI) que pertence ao Instituto Superior de Engenharia do Porto (ISEP). Esta Unidade Curricular é bastante importante pois permite consolidar os conhecimentos obtidos no decorrer do percurso académico, como também a aquisição de novos conhecimentos, já num ambiente diferente.

Dadas as especificidades do problema, no âmbito do projeto, foi equacionada a a criação de uma prova de conceito de forma a poder tornar mais eficiente uma empresa que trabalha como prestadora de serviços na área de auditoria de qualidade no ramo automóvel, esta aplicação será apelidada de Sistema de Gestão de Recursos (SiGeCur).

No final deste capítulo é apresentada a estrutura do relatório em capítulos e a respetiva descrição.

1.1 Enquadramento/Contexto

A motivação para este projeto centra-se no facto de uma grande generalidade das pequenas e medianas empresas em Portugal não terem capacidade técnica, conhecimento e/ou capacidade monetária para dar o salto na direção da indústria 3.0 ou 4.0, trabalhando por vezes de uma forma mais antiquada. Neste tipo de empresas ainda se utiliza muitos registos em papel e formas de envio dos mesmos tornando os processos demasiado burocráticos e lentos. Os ERP no mercado geralmente têm de ser adaptados para a necessidade de cada empresa o que os torna ainda mais dispendiosos e com necessidades de ter técnicos formados para aquele programa ou área em específico. Ao trabalhar numa empresa que também sofre do mesmo problema e que não tem capacidade financeira para adquirir ou manter um ERP, surgiu a ideia de criar um programa conceptual de acordo com as necessidades da mesma que pudesse automatizar processos e diminuir perdas de tempo e custos. Nomeadamente criação de relatórios de serviços efetuados, histórico, ajudas visuais, autorizações de trabalho e acesso direto do cliente aos relatórios no mesmo dia em que são efetuados os serviços.

1.2 Descrição do Problema

O trabalho foi realizado tendo em conta a necessidade de uma empresa prestadora de serviços de qualidade dentro do ramo automóvel. O objetivo no imediato para esta organização é o de se atualizar para a indústria 3.0, isto é automatização dos processos de produção e introdução de tecnologia e sistemas de informação. Sendo que a médio tem em mente a indústria 4.0.

Para todos os serviços efetuados são elaborados relatórios de serviços em papel que depois são preenchidos manualmente linha a linha por uma administrativa para ficheiros de Excel como ilustra a Figura 1, são enviados ao cliente no dia seguinte. Dependendo da quantidade de relatórios por vezes obriga a necessidade de horas extra ou de atraso no envio dos mesmos.

XVISION		Folha de Trabalho/ Working Sheet										WO no. 315121											
												Data: 09/04/21											
Cliente/Client:		SK326hKd109 - Trigem Pur																					
Nome/Name CQ:																							
Designação do serviço/Service Designation:		Referência/Part Number	Lote	ID	Qty.OK	Qty.NOK	Qty Re-Work	NOK1	NOK2	NOK3	NOK4	NOK5	NOK6	NOK7	NOK8	NOK9	NOK10	NOK11	NOK12	NOK13	NOK14	NOK15	
Met 000003		€4 2105947	0 0006		30																		
			11		30																		
			13		30																		
			25		30																		
			21		30																		
			20		30																		
			19		30																		
			23		30																		
			12		30																		
			25		30																		
Met 000003		€4 2105947	0 0022		30																		
			23		30																		
			25		30																		
			42		30																		
			39		30																		
			34		30																		
			33		30																		
			40		30																		
			32		30																		
			42		30																		
		TOTAL			600														1	2			
Descrição de Defeitos/Defects Discrimination:		Observações/Observation:																					
NOK 1- Incomplete		NOK 6- Rubber 8 plastic																					
NOK 2- Bolhas		NOK 7- Arezinhos menores																					
NOK 3- Excesso Pur		NOK 8- Excesso de "Pelicula" Pur																					
NOK 4- Filtre/Defeito Pur		NOK 9- Crux 2 Tampas completas																					
NOK 5- Superfície Tack Solvente		NOK 10- R. 250722														Horário CO's:							

Figura 1 - Exemplo de relatório utilizado

Os auditores para fazerem determinado serviço tem de ter formação e serem atualizados sempre que exista novas ordens do cliente, acontecendo por vezes que o funcionário não recebe essa atualização ou não atempadamente e faz o seu trabalho com informações

desatualizadas, podendo originar em defeitos a chegar ao cliente final ou materiais fora de índice entre outros problemas.

Sempre que é iniciado um novo pedido de serviço tem de se entrar em contato com uma administrativa que por sua vez irá verificar qual foi o último “work order” para indicar ao auditor o número seguinte de pedido de serviço. Existindo uma interação desnecessária que poderá ser automatizada.

1.2.1 Objetivos

Os objetivos gerais deste projeto centrão-se na criação de um protótipo funcional (*MVP - Minimum Viable Product*), digitalização e automatização do processo de gestão relatórios e introdução nos processos de tecnologias de informação

Os objetivos específicos deste projeto são o de criar uma aplicação web conceptual com uma interface de utilizador e um *backend*. A aplicação deverá suportar vários papéis de utilizador com funcionalidades distintas:

- O team leader pode fazer todos os tipos (funcionários, clientes, pedidos de serviço, ajudas visuais e autorizações de funcionários), não é permitido eliminar clientes, pedidos de serviço e ajudas visuais.
- O supervisor e ou Gestor tem acesso a todas as funcionalidades, inserir, editar, eliminar e pesquisar qualquer um dos recursos.
- Controlador e/ou Auditor apenas podem inserir os relatórios dos trabalhos efetuados, pesquisar por ajudas visuais, efetuar pedidos de serviço, relatórios e autorizações de serviço. Também podem visualizar histórico de relatórios criados.
- Estagiário apenas pode fazer pesquisas por ajudas visuais, pedidos de serviço e relatórios.
- A Aplicação ser *userfriendly* e de fácil compreensão.

1.2.2 Abordagem

Para responder ao problema descrito anteriormente desenvolveu-se uma “prova de conceito” limitando-se a solução a um conjunto de casos de uso e funcionalidades de forma a conseguir apresentar dentro do tempo útil de projeto um protótipo funcional (*MVP - Minimum Viable Product*). Para este projeto efetuou-se um estudo sobre possíveis aplicações que poderiam ser uma solução como também se questionou na própria empresa em questão quais seriam os

casos de uso mais importantes para uma possível prova de conceito. Do ponto de vista de arquitetura, design e tecnologia, foram escolhidas tecnologias e *frameworks* com alguma aceitação em áreas de aplicação similar. A escolha de *frameworks* recaiu: ao nível de *backend*, a preferida foi Onion/Clean Architecture devido a melhor testabilidade, manutenção e confiabilidade nas infraestruturas sendo o ASP.NET preferido pelas mesmas razões; para *frontend* foi escolhido o *framework* Angular de forma a ter uma interação com o utilizador mais intuitiva e ágil e por ser robusto, reutilizável e de ser menos complexo ao nível que se aumenta a sua estrutura.

1.2.3 Contributos

A realização deste projeto a nível pessoal/profissional foi uma grande mais-valia, pois, permitiu a oportunidade de adquirir novos conhecimentos de forma autónoma e sem a intervenção direta do corpo docente do ISEP, como também aplicar alguns dos conhecimentos adquiridos durante a licenciatura, foi também possível ganhar uma maior experiência.

A nível técnico este projeto (prova de conceito) foi dado um contributo importante para a eliminação de desperdícios quer monetários como temporais para esta empresa como restantes que ainda trabalhem da mesma forma.

Tendo em conta a altura em que se vive, uma empresa que trabalhe de forma mais rudimentar e que ainda aplica técnicas de sucesso, mas do fim do séc. XX início do séc. XXI, acabam por perder clientes ou ter lucros reduzidos por acharem que sigla JIT (*Just in Time*) significa *Just in Tomorrow*.

Sendo que um ERP pode ser o “sistema nervoso central de uma empresa” e como cada empresa, principalmente as PME’s tem características singulares e muitas vezes necessidades únicas, o que os ERP’s generalistas não conseguem colmatar. Esta prova de conceito é uma solução desenvolvida em casa que permitirá suprir algumas necessidades imediatas, e à posteriori avaliar os ganhos obtidos. A solução ERP embora mais completa, requere outros recursos humanos e financeiros. Esta opção será uma solução *silver bullet*, com objetivos concretos, bem definidos, e rápida implantação.

1.2.4 Planeamento do trabalho

No início do projeto/estágio foi realizado um planeamento no qual foram delineadas as tarefas a realizar para concluir o projeto, assim como a duração estimada destas. Estas tarefas foram documentadas recorrendo a um diagrama de *Gantt* que está dividido em 6 atividades:

Tarefa 1: Estudo da arte e identificação de características e parâmetros de avaliação de soluções a ser utilizados.

Tarefa 2: *Backend*, serviços e desenho do modelo de dados.

Tarefa 3: Desenvolvimento de *Frontend*.

Tarefa 4: Desenvolvimento de documentação sobre a implementação e operação da aplicação.

Tarefa 5: Testes.

Tarefa 6: Elaboração do relatório.

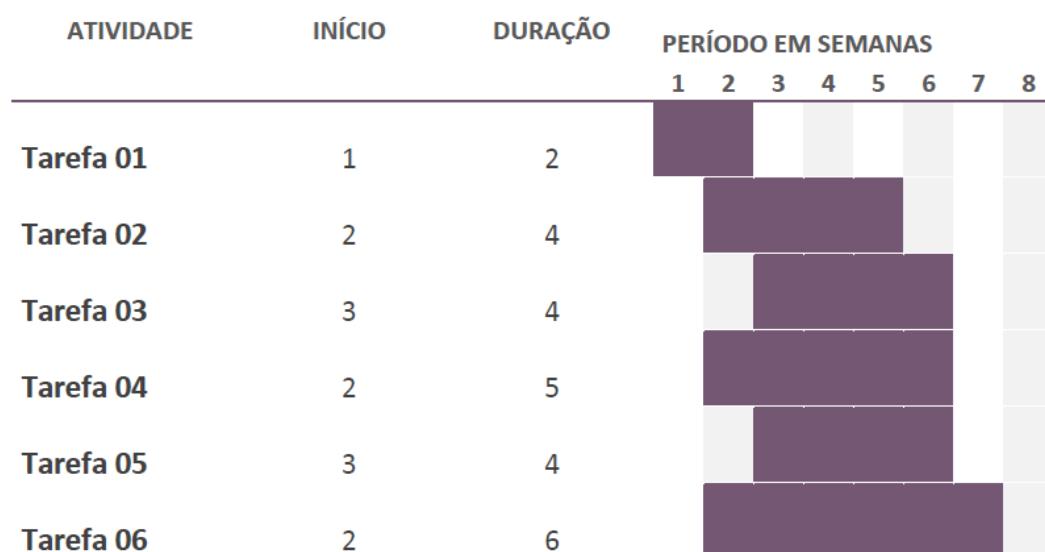


Figura 2 - Diagrama de Gantt, planeamento projeto

1.3 Estrutura do relatório

Este relatório está organizado em cinco capítulos e tem como principal propósito descrever e demonstrar o trabalho desenvolvido ao longo do projeto/estágio.

No **capítulo 1**, é efetuada a introdução com uma breve apresentação ao projeto, explicação geral do que este capítulo contém, enquadramento e uma apresentação do projeto, realçando os objetivos do trabalho e a abordagem tomada para os atingir, os contributos que este trabalho pretende ter e o planeamento do mesmo, por fim, a estruturação do presente documento.

No **capítulo 2**, é apresentado o estado de arte e a pesquisa bibliográfica para a implementação deste projeto onde se descrevem algumas tecnologias existentes no mercado, com a respetiva análise crítica entre as mesmas, as tecnologias escolhidas e as respetivas razões de escolha.

No **capítulo 3**, descreve-se a análise e desenho da solução proposta, o domínio do problema, os requisitos a cumprir e uma descrição da solução proposta.

No **capítulo 4**, são apresentados os detalhes relacionados com o enquadramento e implementação das soluções preconizadas no capítulo anterior, isto é, como foi implementada a proposta de solução ao problema inicialmente e apresentam-se e os seus testes unitários e de integração.

No **capítulo 5**, são apresentadas as conclusões, tecem-se reflexões sobre os objetivos conseguidos, as limitações para trabalhos futuros e uma apreciação final sobre o trabalho efetuado.

2 Estado da arte

Este capítulo surge como um estudo sobre arquiteturas em sistemas de software. Também foi incluída uma pesquisa sobre ERP (*Enterprise Resource Planning*), que permitiu obter inspirações e ideias na resolução de problemas, como também de possíveis funcionalidades a implementar futuramente.

2.1 ERP

A história do ERP começou no início da década de 1960 quando surgiram os primeiros conceitos de aplicações empresariais computorizadas no mundo da contabilidade e das finanças, utilizando mainframes. Estas aplicações vanguardistas eram mais rápidas e mais exatas do que os processos manuais existentes, mas eram dispendiosas, limitadas na sua funcionalidade e ainda lentas.

No início da década de 1970 a expansão económica e maior disseminação computacional deram origem ao desenvolvimento de soluções dedicadas e autónomas para o processamento de ordens de venda e o planeamento de necessidades de material (MRP) predecessor do ERP.

Na década de 1980 com o início de redes de computadores ligadas a servidores mais baratos e fáceis de usar que os mainframes, a concorrência no setor do fabrico passou por uma expansão e, por isso, eram necessárias novas ferramentas. O novo software de MRP II integrava contabilidade e finanças, vendas, compras, inventário e planeamento e agendamento de fabrico, dotando o fabricante de um sistema integrado.

Na década de 1990 foi introduzido o ERP. Este transformou o setor da tecnologia, servindo um conjunto mais abrangente de setores de atividade e combinando MRP II, recursos humanos, contabilidade de projeto e relatórios para utilizadores finais. Este salto evolutivo foi conseguido entre outras razões graças a uma evolução das redes de comunicação entre computadores com a arquitetura cliente-servidor e preços ainda mais competitivos.

A partir do ano 2000, a crescente velocidade da Internet e novas ferramentas de desenvolvimento voltaram a revolucionar os conjuntos de aplicações de ERP. A introdução do software baseado em browser pavimentou o caminho para o software de ERP na *Cloud*, uma inovação que expandiu o alcance e a funcionalidade das soluções de ERP. [1] [2]

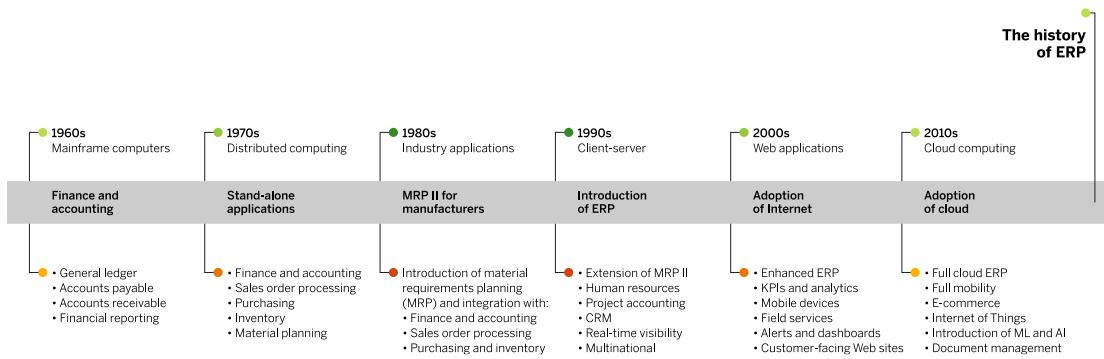


Figura 3 - História ERP [1]

Posteriormente são apresentadas algumas tecnologias existentes, como também as tecnologias utilizadas e a respetiva justificação da utilização.

2.2 Trabalhos relacionados

Neste subcapítulo são apresentadas algumas soluções existentes no mercado que possam suprimir as necessidades da empresa (XVISION) para qual esta a ser desenhado esta prova de conceito.

Tendo em conta que esta solução visa agilizar a gerir recursos da empresa XVISION, começou-se por fazer um estudo de soluções de softwares existentes no âmbito de gestão e planeamento de recursos, neste caso específico direcionados para empresas que prestem serviços de consultadoria e qualidade, foi efetuada uma pesquisa por aplicações que possuíssem uma estrutura o mais semelhante possível à do presente projeto e que pudesse responder as necessidades desta empresa e área em questão, tecnológica como do ponto de vista monetário.



Figura 4 - Magic Quadrant Gartner ERP [3]

Nenhum dos ERP indicados pela Gartner no Magic Quadrant foi considerado devido aos seus elevados custos. Apenas foi considerada na análise o NetSuite (quadrante dos visionários).

2.2.1 Soluções desenvolvidas em Portugal

Inicialmente, devido à potencial facilidade suporte, foi dado preferência a soluções desenvolvidas em Portugal.

Para auxiliar nesta pesquisa foi utilizado o site www.capterra.pt, no qual se pode pesquisar por software empresarial e fazer comparações e avaliações do software, neste caso a primeira pesquisa foi com a palavra-chave “ERP” e utilizou-se um filtro para mostrar só soluções de ERP que fossem produzidas em Portugal, e como se pode ver pelo resultado da imagem a seguir só foram encontrados dois. Para além do uso do capterra foram utilizadas pesquisas online que tentassem corroborar as afirmações efetuadas no mesmo.

The screenshot shows the Capterra platform interface. At the top, there's a navigation bar with the Capterra logo, a search bar containing 'Procurando software...', and a 'Categorias de software' link. Below the header, a banner defines what an ERP is and links to more information. The main content area has a 'Todos os produtos' button and a 'Por que o Capterra é gratuito?' link. On the left, there are several filter sections: 'Filtrar resultados (2)' with 'Produto local (Portugal) X' and 'Portugal X'; a 'LIMPAR TUDO' button; 'Países disponíveis' with 'Portugal' checked; 'Opções de preços' with 'Teste grátis'; 'Recursos' including CRM, Gestão de RH, Gestão de pedidos, etc.; and 'Implantação' options like Nuvem, SaaS, baseado na web. The right side lists two products: 'Keyword' and 'myBusiness365', each with a brief description, a 'PRODUTO LOCAL' badge, and a 'VER PERFIL' button.

Figura 5 - Pesquisa ERP produzido em Portugal

Ao se fazer uma análise as soluções apresentadas, nenhuma das duas consegue responder às necessidades identificadas no problema inicial. Estas soluções ERP apenas foram incluídas na análise para demonstrar que também existem empresas portuguesas a criar soluções de ERP.

Keyword é um software de subscrição pago por utilizador e por ano, que se foca bastante na área de RH (Recursos Humanos), para recrutamento, gestão de projetos e equipas. Desta solução pode se tomar alguma inspiração para o projeto, no que concerne a gestão dos recursos (funcionários) para se aplicar na atribuição de autorizações de auditorias, mas excetuando essa funcionalidade não responde as necessidades nem a solução desenhada neste projeto.

Mybusiness365 é um software de subscrição e por cada utilizador, e que está mais direcionado para gestão de vendas e gestão de finanças. Este software é assente em

tecnologia Microsoft Dynamics365 e como tal trabalha 100% a partir da *cloud*. [4] Tal como o *Keyword* esta solução também não responde nem a necessidade nem a área em que a XVISION trabalha.

2.2.2 Soluções desenvolvidas fora de Portugal

De seguida efetuou-se uma pesquisa mais alargada, em que a palavra-chave continua a ser “ERP”, mas remove-se a restrição de ser um software produzido em Portugal, mas limita-se os resultados com os seguintes filtros que se podem ver no canto superior esquerdo da imagem da [figura 5](#), o principal é que seja vendido em Portugal, mas pode não estar no idioma português e obtém se 33 resultados como se pode ver no mesmo local da imagem.

The screenshot shows the Capterra website interface. At the top, there is a navigation bar with the Capterra logo, a search bar containing 'Procurando software...', and a 'Categorias de software' link. Below the header, the main title is 'Sistemas ERP'. A sub-section definition follows: 'O sistema ERP refere-se a softwares que integram todos os departamentos e funções da organização em um único sistema. Um ERP online lida com processos de manufatura, logística, distribuição, estoque... [Leia mais](#)'.

The main content area displays search results for 'Todos os produtos'. On the left, there is a sidebar with filtering options: 'Filtrar resultados (33)', 'Portugal X', 'Banco de dados do cliente X', 'Gestão de pedidos X', 'Relatórios e estatística X', 'Relatórios e análise de dados X', 'LIMPAR TUDO', 'Países disponíveis', 'Portugal', 'Mostrar outros países', 'Idiomas', 'Português', 'Mostrar outros idiomas', 'Opções de preços', 'Versão gratuita', 'Teste grátis', 'Classificação do produto', and rating filters for 5 stars, 4 ou mais stars, 3 ou mais stars, and 2 ou mais stars. The results are listed in a grid:

- monday.com** (4,6 (3 261)) - Description: Use o Work OS da monday.com para simplificar e centralizar o planejamento de recursos empresariais, melhorar o trabalho em equipe e fazer mais. Esa plataforma personalizável de gestão de projetos e ... [Leia mais](#). Resources: CRM, Gestão de RH, Gestão de pedidos, Gestão financeira, Gestão de estoques. Buttons: 'ACESSE O SITE'.
- Odoo** (4,1 (545)) - Description: O Odoo é um software integrado, personalizável e de código aberto, repleto de aplicativos de negócios projetados por especialistas. O Odoo oferece tudo de que uma empresa precisa para funcionar com ... [Leia mais](#). Resources: CRM, Gestão de RH, Gestão de pedidos, Gestão financeira, Gestão de estoques. Buttons: 'ACESSE O SITE'.
- NetSuite** (4,1 (1 155)) - Buttons: 'VER PERFIL'.

Figura 6 - Pesquisa alargada sem necessidade de idioma português. [5]

A última pesquisa efetuada foi similar á anterior, mas desta vez efetuou-se uma pesquisa por software que tivesse disponível com idioma em português, o que reduziu o resultado para 8, como se pode ver na imagem da [figura 8](#).

The screenshot shows the Capterra website interface. At the top, there's a navigation bar with the Capterra logo, a search bar containing 'Procurando software...', and a 'Categorias de software' link. Below the header, a banner for 'ERP' software is displayed, followed by a 'Todos os produtos' button and a 'Por que o Capterra é gratuito?' link.

Filtrar resultados (8)

- Portugal português
- Controles/permissões de acesso
- Banco de dados do cliente
- [LIMPAR TUDO](#)

Países disponíveis

- Portugal
- [MOSTRAR OUTROS PAÍSES](#)

Idiomas

- Português
- [MOSTRAR OUTROS IDIOMAS](#)

Opções de preços

- Versão gratuita
- Teste grátis

Classificação do produto

- 5 estrelas
- 4 ou mais estrelas
- 3 ou mais estrelas
- 2 ou mais estrelas
- 1 ou mais estrelas

Recursos

- Acompanhamento de atividades
- Alertas/notificações
- Banco de dados do cliente
- CRM
- Campos personalizáveis
- Controle do processo de aprovação
- Controles/permissões de acesso
- Gerenciamento de documentos
- Importação/exportação de dados
- Painel de atividades

Software resultados:

- monday.com**   (3 261) [ACESSE O SITE](#)
- Bitrix24**   (581) [ACESSE O SITE](#)
- Odoo**   (545) [ACESSE O SITE](#)
- NetSuite**  [VER PERFIL](#)

Figura 7 - Pesquisa alargada com idioma português

Por fim realizou-se uma comparação entre as quatro primeiras soluções obtidas nas duas pesquisas, por serem as mais utilizadas na área pelo qual foram filtradas e as que obtêm mais opiniões dos utilizadores das próprias.

The screenshot shows a comparison table on Capterra for four software solutions: Bitrix24, monday.com, NetSuite, and Odoo. The table compares them across various features and includes screenshots of their interfaces.

	Bitrix24	monday.com	NetSuite	Odoo
Preço Inicial	61,00 US\$/mês	8,00 €/mês	Sem informação do fornecedor	Sem informação do fornecedor
Classificações	Geral ★ 4,1 (581) Ver todas as pontuações ▾	Geral ★ 4,6 (3 262) Ver todas as pontuações ▾	Geral ★ 4,1 (1 155) Ver todas as pontuações ▾	Geral ★ 4,1 (545) Ver todas as pontuações ▾
Recursos	<ul style="list-style-type: none"> ✓ Compartilhamento de arquivos ✓ Controle de marcos ✓ Extração de dados ✓ Gestão de clientes ✗ Gestão de despesas ✓ Gestão de edição ✓ Gestão de recursos ✓ Integração de redes sociais ✓ Interações de terceiros ✓ Visualização de Gantt/linha do tempo Ver todos os recursos ▾	<ul style="list-style-type: none"> ✓ Compartilhamento de arquivos ✓ Controle de marcos ✓ Extração de dados ✓ Gestão de clientes ✓ Gestão de despesas ✓ Gestão de edição ✓ Gestão de recursos ✓ Integração de redes sociais ✓ Interações de terceiros ✓ Visualização de Gantt/linha do tempo Ver todos os recursos ▾	<ul style="list-style-type: none"> ✓ Compartilhamento de arquivos ✓ Controle de marcos ✓ Extração de dados ✓ Gestão de clientes ✓ Gestão de despesas ✓ Gestão de edição ✓ Gestão de recursos ✓ Integração de redes sociais ✓ Interações de terceiros ✓ Visualização de Gantt/linha do tempo Ver todos os recursos ▾	<ul style="list-style-type: none"> ✓ Compartilhamento de arquivos ✓ Controle de marcos ✓ Extração de dados ✓ Gestão de clientes ✓ Gestão de despesas ✓ Gestão de edição ✓ Gestão de recursos ✓ Integração de redes sociais ✓ Interações de terceiros ✓ Visualização de Gantt/linha do tempo Ver todos os recursos ▾
Implantação	<ul style="list-style-type: none"> ✓ Baseado na nuvem ✓ Instalação local 	<ul style="list-style-type: none"> ✓ Baseado na nuvem ✗ Instalação local 	<ul style="list-style-type: none"> ✓ Baseado na nuvem ✗ Instalação local 	<ul style="list-style-type: none"> ✓ Baseado na nuvem ✓ Instalação local
Formação	<ul style="list-style-type: none"> ✗ Pessoalmente ✗ Ao vivo online ✓ Webinars ✓ Documentos ✓ Vídeos 	<ul style="list-style-type: none"> ✓ Pessoalmente ✓ Ao vivo online ✓ Webinars ✓ Documentos ✓ Vídeos 	<ul style="list-style-type: none"> ✓ Pessoalmente ✓ Ao vivo online ✓ Webinars ✓ Documentos ✓ Vídeos 	<ul style="list-style-type: none"> ✓ Pessoalmente ✓ Ao vivo online ✓ Webinars ✓ Documentos ✓ Vídeos
Assistência	<ul style="list-style-type: none"> ✓ Email/Help Desk ✓ FAQs/Fórum ✓ Base de conhecimento ✗ Suporte por telefone ✓ Assistência 24/7 ✓ Bate-papo 	<ul style="list-style-type: none"> ✓ Email/Help Desk ✓ FAQs/Fórum ✓ Base de conhecimento ✓ Suporte por telefone ✓ Assistência 24/7 ✓ Bate-papo 	<ul style="list-style-type: none"> ✓ Email/Help Desk ✓ FAQs/Fórum ✓ Base de conhecimento ✓ Suporte por telefone ✓ Assistência 24/7 ✓ Bate-papo 	<ul style="list-style-type: none"> ✓ Email/Help Desk ✓ FAQs/Fórum ✓ Base de conhecimento ✓ Suporte por telefone ✗ Assistência 24/7 ✓ Bate-papo
Capturas De Tela	 Ver 5 capturas de tela	 Ver 5 capturas de tela	 Ver 2 capturas de tela	 Ver 5 capturas de tela

Figura 8 - Comparação entre as 4 primeiras soluções.

BITRIX24 [6].[7]

O Bitrix24 remonta ao ano de 2010 e como uma plataforma interna para ser utilizado na própria empresa, como verificaram que essa solução trouxe um aumento de produtividade quase imediato decidiram vender essa solução e em 2012 saiu a primeira versão beta. Este software oferece uma solução completa de gestão de relacionamento com o cliente (CRM) especializada na geração de leads (resposta a uma demonstração de interesse). O produto também permite gerir projetos de curto ou longo prazo, configurar e administrar um *help desk* e muito mais. É uma solução complexa que oferece vários recursos para utilizadores complexos, mas faz as pequenas coisas extremamente bem, como gerir o fluxo de leads para pequenas empresas. Infelizmente, as complexidades também dificultam o uso e não é de forma alguma *plug-and-play*. Os utilizadores tendem a gastar uma boa quantidade de tempo para aprender a configurar e descobrir como colocar o software a ser benéfico.

Prós:

- Ampla gama de recursos, como tarefas, CRM, Drive, comercio eletrónico e construtor de sites;
- Ferramentas robustas de listagem e gestão de leads e projetos;
- Notificações para manter toda a equipa atualizada;
- Avaliação gratuita ilimitada (número de utilizadores);

Contras:

- Interface de utilizador complexa;
- Pouco intuitivo, difícil aprender a usar, devido a não ter plano de formação;
- Mais caro que os principais concorrentes;
- Lista de recursos “interminável” e mal-organizados;
- Recursos adicionais desnecessários para muitas empresas que acabam por ocupar espaço e tornar o software mais confuso.

MONDAY.COM [8].[9]

Monday.com é uma plataforma baseada em nuvem que permite que as empresas criem as suas próprias ferramentas e aplicativos de gestão de trabalho. Começou como uma ferramenta de colaboração de trabalho em 2014 e expandiu-se ao longo dos anos para se tornar uma ferramenta de colaboração para muitos fluxos de trabalho, incluindo gestão de

relacionamento com o cliente (CRM), gestão de projetos, desenvolvimento de software, recursos humanos, marketing e muito mais. Dependendo dos seus requisitos, pode-se usar os modelos existentes da empresa para esses fluxos de trabalho ou criar seus próprios desde o início.

Prós

- Plano gratuito para sempre disponível para até dois utilizadores;
- Painel com visual moderno e intuitivo;
- Altamente personalizável;
- Mais de 200 modelos para criar quadros e automações.

Contras

- Planos de preços desnecessariamente confusos devido ao preço por utilizador;
- Plano gratuito demasiado básico e limitado;
- Tamanho mínimo da equipe de três utilizadores para planos pagos;
- Teste de 14 dias para planos pagos;
- Falta recursos avançados de rastreamento de tarefas.

NETSUITE [10]_[11]_[12]

NetSuite é frequentemente reconhecida como a primeira empresa de software em cloud. Tornou-se o líder da indústria de *cloud ERP* porque pode responder às necessidades das empresas de “todas” as formas e tamanhos.

O sistema *NetSuite* destaca-se por ser uma plataforma de gestão de projetos, gestão empresarial e comercial ágil a fim de dimensionar e preparar seus negócios e ele evolui de acordo com as necessidades da empresa.

Utiliza o software como um modelo de serviço (SaaS): Os clientes pagam uma taxa de subscrição para aceder à tecnologia, mas não são responsáveis por qualquer infraestrutura subjacente ou manutenção do sistema, incluindo a compra e instalação de servidores, instalação de software ou teste e implementação de *patches* e atualizações.

A *Oracle NetSuite* trata de tudo isso para os clientes, com duas atualizações de software por ano.

Sendo uma das mais antigas e de certa forma a mais utilizada pelas grandes empresas pode se citar as seguintes vantagens desvantagens geradas pelo sistema ERP *NetSuite*.

Prós:

- É escalável, personalizável, e concebida para apoiar a empresa à medida que cresce e muda;
- Software rico em funcionalidades adequado para organizações que realizam negócios a uma escala global;
- É um serviço de nuvem confiável, seguro e de fácil acesso;
- O software é adequado para organizações que conduzem negócios em escala global;
- Fluxos de trabalho de processos personalizados;

Contras:

- Não é uma solução ideal para pequenas empresas;
- Pode ser muito caro para algumas organizações. Há também custos adicionais para formação, atendimento ao cliente e outras opções;
- Para clientes que desejem falar com o suporte *NetSuite* 24x7, terão de adquirir o *NetSuite Premium Support* juntamente com o contrato de licença *NetSuite*;
- Tem uma longa curva de aprendizagem, que pode frustrar ou sobrecarregar os seus utilizadores;
- Tem um longo processo de implementação (de alguns meses a anos);
- Utilizadores relataram experiências de suporte ao cliente insatisfatórias, sendo que este é pago;
- Funções específicas podem ser difíceis de configurar no sistema;
- Painel desatualizado, que precisa ser redesenhado.

ODOO [13].[14]

Odoo, em 2005 nasceu como um produto criado em código aberto chamado *TinyERP*, em 2009 graças a uma reunião com os diretores da DANONE em que “ridiculizaram” o nome do software, renomearão no para OpenERP. Depois de vários recursos e módulos adicionados, em 2014 decidem mudar de novo de nome por o software não se restringir só as funções de ERP e passa-se a chamar *Odoo*. Está disponível para ser utilizado na *cloud* ou em servidor próprio, é um software abrangente de planeamento de recursos empresariais (ERP) de código aberto composto por um conjunto integrado de módulos de negócios, incluindo gestão de relação com o cliente (CRM), comércio eletrônico, contabilidade, cobrança, gestão de stock,

gestão de projetos, gestão de armazém, gestão financeira, produção e compras. Esses módulos visam-se comunicar de forma eficiente e transparente entre si para trocar informações.

Prós:

- Tem uma arquitetura modular e flexível;
- É feito em código aberto;
- Ampla gama de recursos;

Contras:

- Requer uma curva de aprendizagem bastante grande, por ser pouco documentado;
- Versão gratuita sem licença e difícil de manter;
- A versão gratuita embora ilimitada no número de utilizadores, só contem um módulo.
- Serviço de suporte ao cliente demasiado lento

2.2.3 Sumário

As aplicações obtidas através da pesquisa efetuada resolvem parte do problema apresentado e em alguns dos casos até superam através de resultados que devolvem e interação com os clientes, e também pela grande variedade de módulos e que podem vir a ser utilizados futuramente no caso criação de páginas web com a apresentação dos serviços e a venda dos mesmos, faturação e chat interno da empresa.

Mas não dão resposta a alguns dos pontos:

- O supervisor dar autorização para os auditores poderem realizar as suas tarefas;
- Os auditores poderem verificar uma ajuda visual para efetuarem um serviço;
- Os clientes terem acesso aos relatórios emitidos pelos auditores;
- Conceito de ajuda visual e autorização associados a um pedido de trabalho.

Não se pode afirmar categoricamente que não possa existir algum software que responda totalmente à proposta do projeto submetido devido a extensa lista de software empresarial existente, mas não foi encontrado nenhum através das pesquisas efetuadas. Pôde se verificar sim que geralmente estão divididos por módulos e estes direcionados na grande maioria para vendas, finanças, logística e recursos humanos. A prova de conceito apresentada neste projeto embora esteja direcionada para uma empresa prestadora de serviços e que em específico trabalha na área de auditoria de qualidade, acaba por responder as necessidades

de outras empresas que trabalham na área e também a algumas outras empresas prestadoras de serviços.

2.3 Tecnologias existentes

2.3.1 Arquitetura de Software

Os padrões arquiteturais expressam formas de organizar a estrutura basilar do sistema, permitindo a construção de uma arquitetura aderente a certas propriedades. O conhecimento de padrões arquiteturais ajuda na definição da arquitetura do sistema.

Estes padrões definem um conjunto de subsistemas, as responsabilidades de cada subsistema e as regras de relacionamentos entre os subsistemas. Por esta razão, influencia aspectos como a performance, qualidade, facilidade de manutenção e escalabilidade, assim, essa decisão tem grande impacto no sucesso do projeto, principalmente a longo prazo, pelo que esta deverá ser cuidada e adequada ao problema em questão. As dificuldades começam a aparecer quando se trata de projetos de grande dimensão. O problema aumenta de escala quando os projetos já existem e não foram sequer pensados desde o seu início para poderem ser estendidos.

Layered Architecture (N-Layer) [15]

O padrão de arquitetura mais comum é o padrão de arquitetura em camadas (*Layered Architecture*), também conhecido como (*N-Layer*). Este padrão é amplamente utilizado devido ao facto de se aproximar da comunicação tradicional de TI e das estruturas organizacionais encontradas na maioria das empresas.

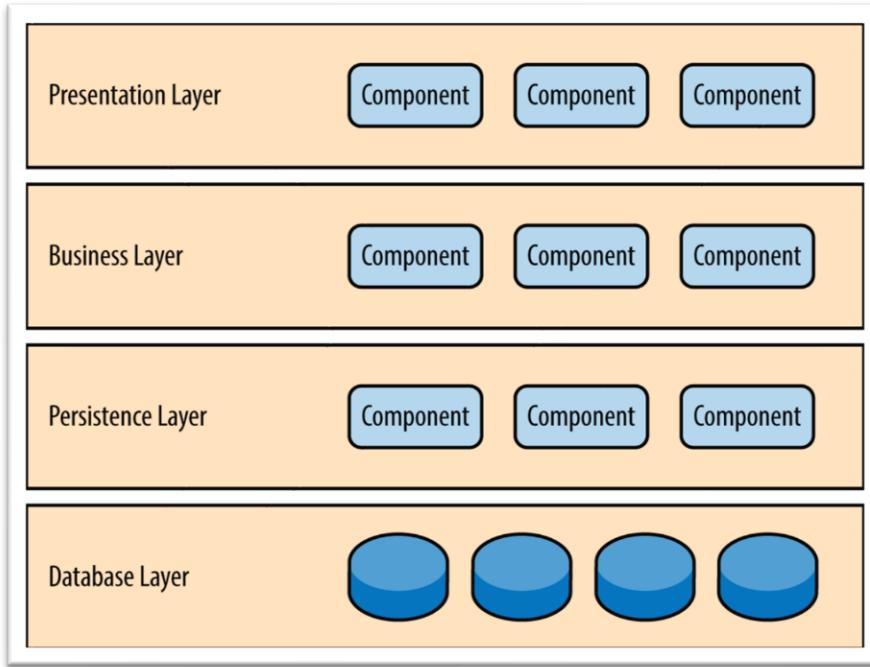


Figura 9 - Padrão de arquitetura em camadas [15]

Apesar de o padrão de arquitetura em camadas não especificar o número e os tipos de camadas que devem existir, a maioria das implementações consiste em quatro camadas: apresentação, negócios, persistência e base de dados (figura 6). Os componentes dentro desta arquitetura são organizados e apresentados em camadas como se pode ver na figura 7, todas estas camadas estão fechadas e organizadas de forma a só poderem solicitar serviços a camada imediatamente abaixo, e estas só fornecem os serviços solicitados a camada imediatamente acima.

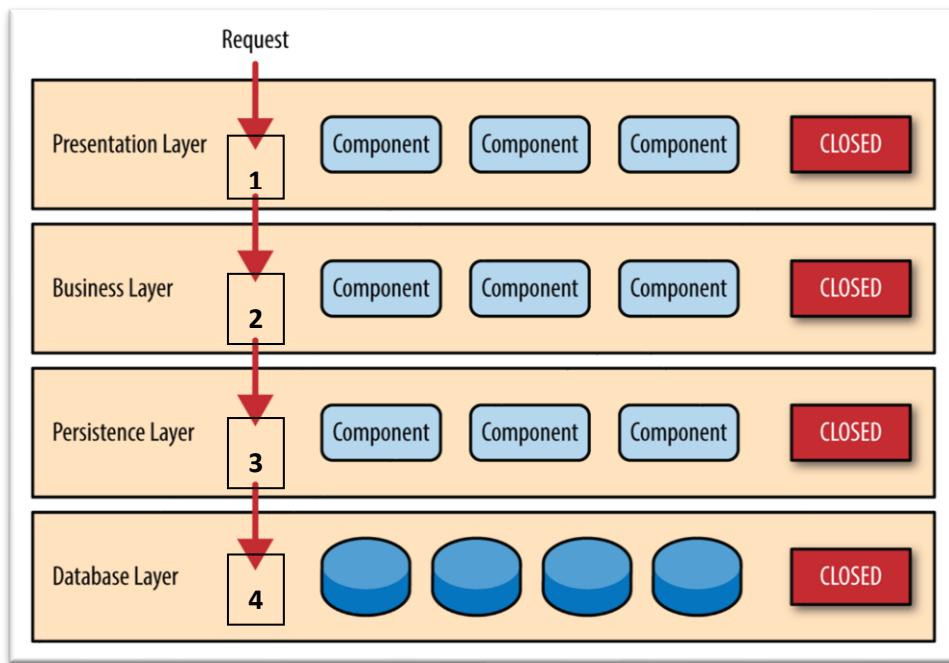


Figura 10 - Pedido de acesso por camadas fechadas (closed layers) [15]

No entanto pode haver a necessidade de existirem camadas abertas, por exemplo ao ser adicionado a camada de serviços entre a camada de negócios e de persistência (figura 8), no entanto, este não é um cenário ideal, pois agora a camada de negócios deve passar pela camada de serviço para chegar à camada de persistência. E se essa solicitação não ganhar nenhum valor passando pela camada de serviço, a única maneira de resolver isto é tornar a camada de serviços e opcional em uma camada aberta, em que a solicitação só passara por essa camada se adicionar algo de relevante ao pedido, caso contrário irá diretamente para a camada de persistência. O que cria um anti padrão na arquitetura de camadas conhecido por *sinkhole*, o que acaba por retirar os benefícios de ter as camadas isoladas.

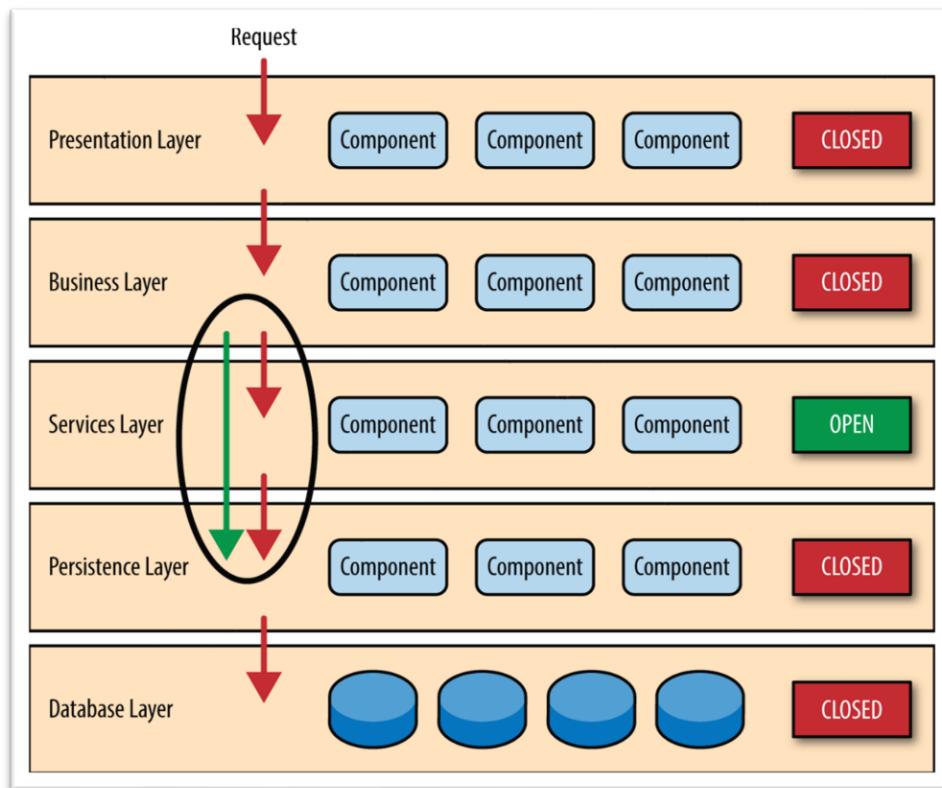


Figura 11 -Fluxo de solicitação, camadas abertas (Open layers) [15]

Prós:

- É fácil testar, pois os componentes pertencem a camadas específicas. Como tal, eles podem ser testados separadamente;
- É simples e fácil de implementar porque, naturalmente, a maioria dos aplicativos funciona em camadas.

Contras:

- Embora as alterações possam ser feitas em uma camada específica, não é fácil porque o aplicativo é uma unidade singular. Além disso, o acoplamento entre as camadas tende a torná-lo mais difícil;
- Ele deve ser implantado como uma unidade singular, portanto, uma alteração em uma camada específica significa que todo o sistema deve ser reimplantado.
- Quanto maior, mais recursos serão necessários para que as solicitações passem por várias camadas e, portanto, causem problemas de desempenho.

***Event-Driven Architecture* [15]**

A arquitetura orientada a eventos (EDA) é um padrão de design de software no qual aplicativos desacoplados podem publicar e assinar eventos de forma assíncrona por meio de um agente de eventos.

O padrão de arquitetura orientada a eventos consiste em duas topologias principais, o *mediator* e o *broker*. A topologia do *mediator* é comumente usada quando é necessário orquestrar várias etapas dentro de um evento por meio de um mediador central, enquanto a topologia do Broker é usada quando se necessita encadear eventos sem o uso de um mediador central.

Topologia do Mediator

Existem quatro tipos principais de componentes de arquitetura na topologia do mediador (*mediator*): filas de eventos, um mediador de eventos, canais de eventos e processadores de eventos. O fluxo de eventos começa com um cliente a enviar um evento para uma fila de eventos, que é usada para transportar o evento para o mediador de eventos. O mediador de eventos recebe o evento inicial, elabora esse evento e envia eventos assíncronos adicionais aos canais de eventos para executar cada etapa do processo. Os processadores de eventos, que escutam nos canais de eventos, recebem o evento do mediador de eventos e executam a lógica de negócios específica para processar o evento. A figura 9 ilustra a topologia geral do *mediator* dentro do padrão EDA.

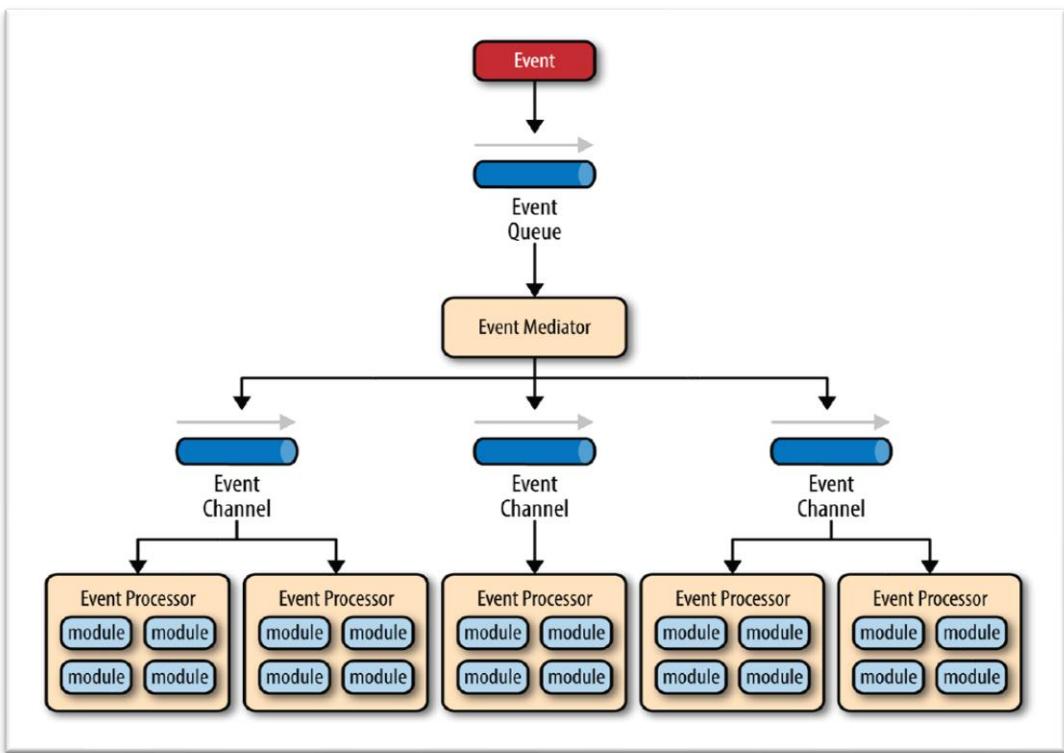


Figura 12 - Event-driven architecture, topologia mediator [15] [16]

Topologia do Broker

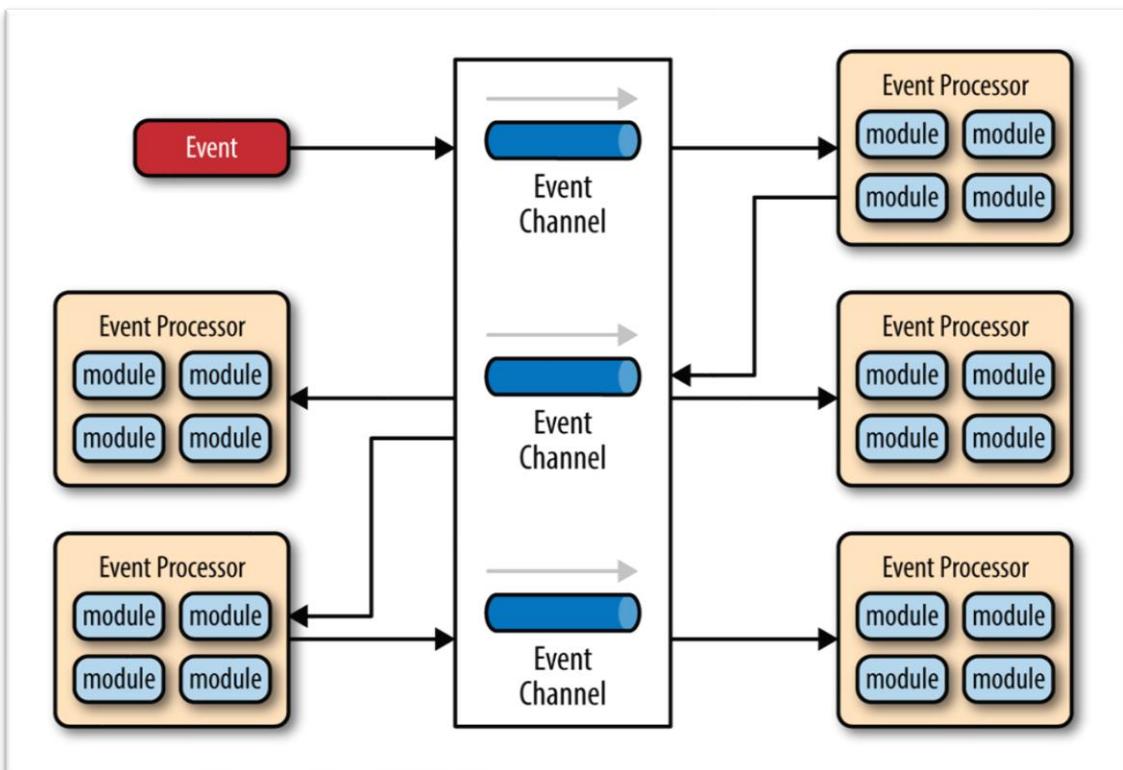


Figura 13 - Event-driven architecture: Topologia Broker [15]

Na topologia Broker existem dois tipos principais de componentes, o broker e o *event processor*. Nesta topologia não temos presença de um elemento central que faça o controlo (a orquestração) do processo. Aqui, o fluxo do processo é gerido pelos próprios *event processors*. Cada *event processor* após terminar a execução do seu evento é responsável por lançar um novo evento para o broker indicando a ação que acabou de fazer.

Prós:

- Alta performance e escalabilidade pela sua natureza assíncrona;
 - Capacidade de resposta, alta demanda e alta velocidade dos dados.

Contras:

- O tratamento de erros pode ser difícil de estruturar, especialmente quando vários módulos devem manipular os mesmos eventos;
 - Impossibilidade de prever eventos que ocorrem em períodos diferentes devido a sua natureza assíncrona.

Onion/Clean Architecture [17] [18] [19]

A arquitetura *Onion* ou *Clean Architecture* é o resultado do Princípio de Inversão de Dependência. Assim, a camada de domínio e a camada de interface do aplicativo tornam-se independentes da interface do utilizador ou de acesso aos dados. Como resultante, quando há uma alteração nesses componentes, UI, Banco de Dados, Web Services, infraestrutura de mensagens etc, as alterações não são refletidas no domínio principal de forma alguma.

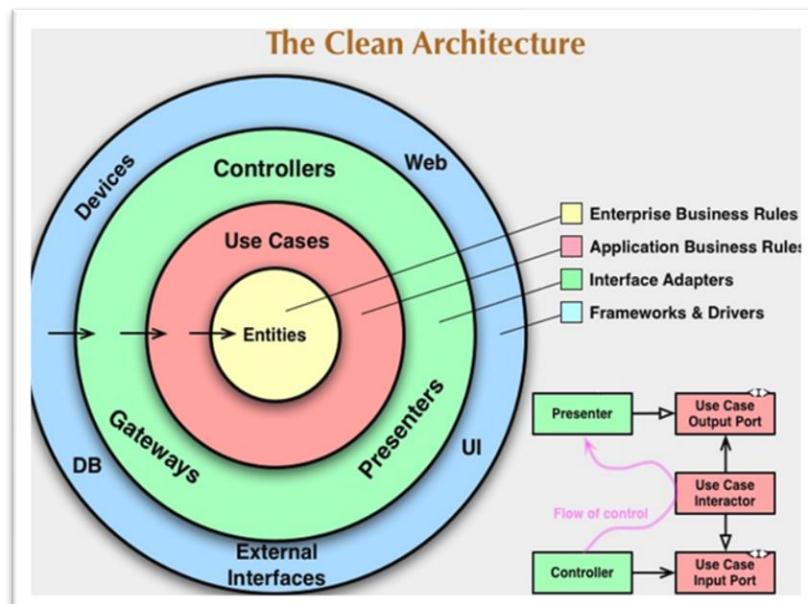


Figura 14 - Onion/Clean Architecture [17]

Os círculos concêntricos representam diferentes áreas de software. Em geral, quanto mais se avança, mais alto o nível de software se torna. Os círculos externos são mecanismos. Os círculos internos são políticos. A regra primordial que faz esta arquitetura funcionar é a regra de dependência. Esta regra diz que as dependências do código-fonte só podem apontar para dentro. O que se encontra no círculo interno pode saber absolutamente nada sobre algo que se encontre no círculo externo. Em particular, o nome de algo declarado em um círculo externo não deve ser mencionado pelo código no círculo interno. Isso inclui, funções, classes, variáveis ou qualquer outra entidade de software nomeada. Da mesma forma, os formatos de dados usados em um círculo externo não devem ser usados por um círculo interno, especialmente se esses formatos forem gerados por uma estrutura em um círculo externo. Não se quer que o que se encontra num círculo externo afete os círculos internos.

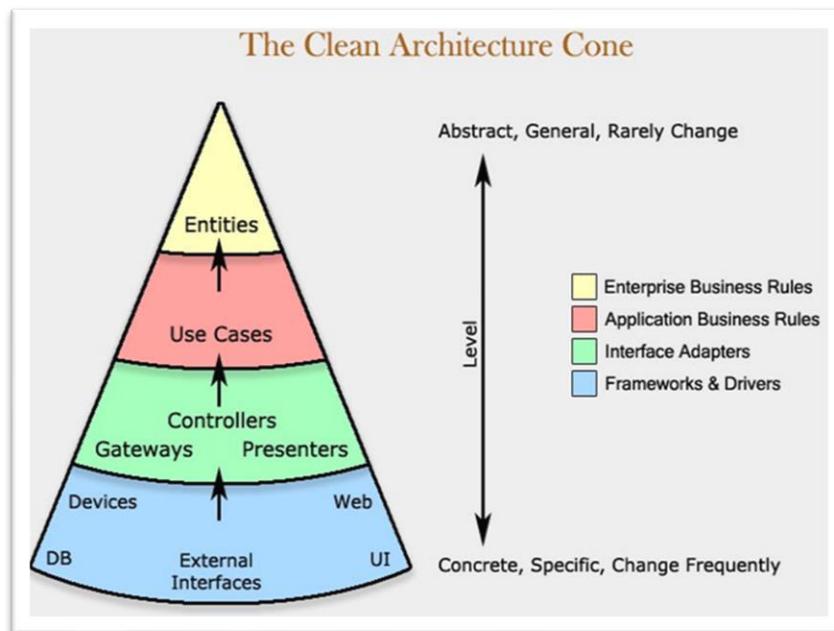


Figura 15 - The clean architecture cone [17]

Entities

As entidades encapsulam as regras de negócios. Uma entidade pode ser um objeto (entidade ou domínio), ou pode ser um conjunto de estruturas de dados e funções. Estes são os menos afetados e propensos a mudar por alterações externas.

Use cases

O software nesta camada contém regras de negócios específicas do aplicativo. Ele encapsula e implementa todos os casos de uso do sistema. Esses casos de uso orquestram o fluxo de dados, de, e para as entidades, e direcionam a essas entidades, as regras de negócios para atingir as metas do caso de uso. Pode-se usar algumas bibliotecas, no entanto, apenas como ferramentas.

Interface Adapters

Esta camada atua como um comunicador para converter dados no formato desejado para armazenamento em fontes externas, como banco de dados, sistema de arquivos e converter dados para casos de uso ou lógica de negócios.

O software nesta camada é um conjunto de adaptadores que convertem dados do formato mais conveniente nos dois sentidos. Também implementa as interfaces dos casos de uso necessários para os componentes externos.

External Interfaces

Esta é a camada mais externa e muda frequentemente com base nas tecnologias, atualizações como banco de dados, estruturas *Web Front-end*. Nesta camada, apresenta-se os dados para a interface do utilizador ou banco de dados

Prós:

- Do ponto de vista da camada interna, pode-se trocar qualquer coisa em qualquer uma das camadas externas e esta continuar a funcionar;
- Facilmente testável, as regras de negócios podem ser testadas independentemente de qualquer elemento externo;
- Pode-se mudar facilmente a UI, sem alterar o resto do sistema;
- A arquitetura do software é construída sobre um modelo de domínio.

Contras:

- Curva de aprendizagem acentuada;
- Ligeira propensão para erro ao dividir responsabilidades entre as camadas, o que prejudica nomeadamente o modelo de domínio.

2.3.2 Web Apis

Os serviços da Web são responsáveis pela comunicação online máquina a máquina. Os computadores usam serviços da Web para se comunicarem através de uma conexão com a Internet. Basicamente são as interfaces *front-end* de sites e aplicativos que residem nos dispositivos dos utilizadores finais. Os dados relacionados são armazenados em um servidor remoto e transmitidos para a máquina cliente por meio de *APIs* que fornecem serviços da Web para os seus utilizadores. As *APIs* podem usar diferentes arquiteturas, como SOAP e REST, que atuam como diretrizes de API para transferir dados do servidor para o cliente.

2.3.2.1 REST vs SOAP

REST (*Representational State Transfer*) e SOAP (*Simple Object Access Protocol*) são duas abordagens diferentes de transmissão de dados online, ambos definem como construir (*APIs*), que permitem que os dados sejam comunicados entre aplicações web. SOAP foi por muito tempo a abordagem padrão para interfaces de serviço da web, no entanto REST tem se destacado acentuadamente nos últimos anos sendo o mais pesquisado e porventura utilizado.

[20]

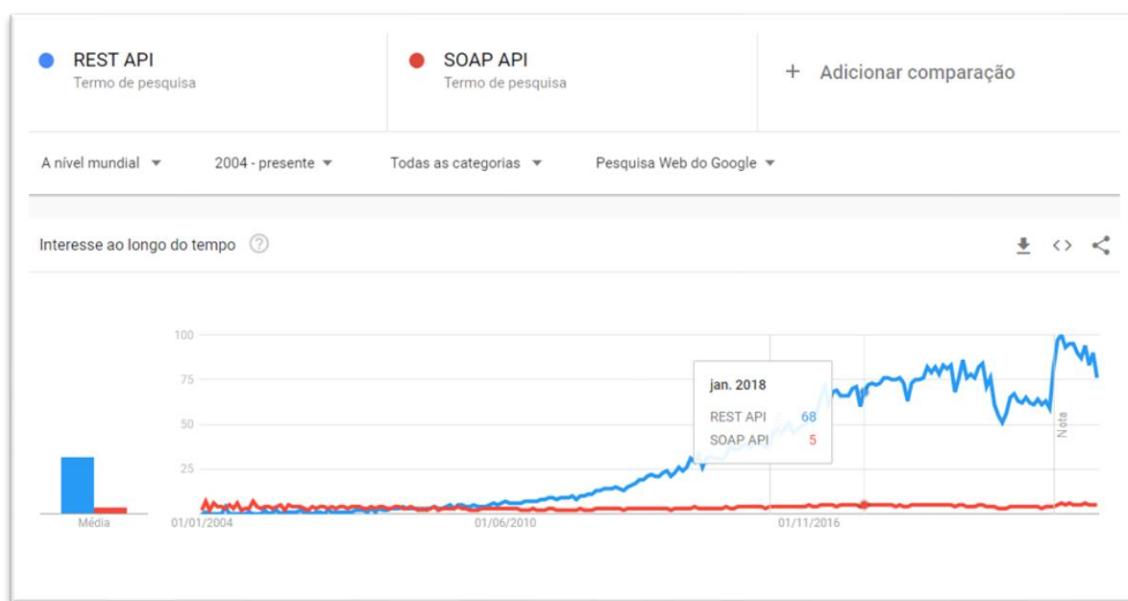


Figura 16 - Popularidade dos protocolos REST vs SOAP [21]

REST

REST (*Representational State Transfer*) é um estilo de arquitetura para criação de APIs. É um conjunto de boas práticas para o desenvolvimento de *web services* escaláveis, confiáveis e de fácil manutenção. Os API REST seguem as convenções REST também são chamados de *RESTful*. O REST pode funcionar com uma variedade de protocolos, como *HTTP*, *SMTP* ou *FTP*. Pode ainda usar vários formatos para transferir os dados, por exemplo, *JSON*, *XML*, *HTML*, texto, imagens entre outros. Embora, na maioria das vezes, os serviços da Web *RESTful* funcionem com *HTTP* e *JSON*. *JSON* (notação de objeto JavaScript) é preferido como formato de mensagem porque pode ser lido por qualquer linguagem de programação, é legível por humanos e máquinas e é leve. Dessa forma, as APIs *RESTful* são mais flexíveis e podem ser mais fáceis de configurar.



Figura 17 - Exemplo em JSON [22]

SOAP

SOAP (*Simple Object Access Protocol*) é um protocolo de mensagens mantido pelo *World Wide Web Consortium (W3C)* inicialmente projetado para que aplicativos construídos com diferentes linguagens e em diferentes plataformas pudessem se comunicar. Por ser um protocolo, ele impõe regras internas que aumentam sua complexidade e sobrecarga, o que pode levar a tempos de carregamento de página mais longos. No entanto, esses padrões também oferecem conformidades integradas que podem torná-los preferíveis para cenários empresariais. Quando uma solicitação de dados é enviada para uma API SOAP, ela pode ser

tratada por meio de qualquer um dos protocolos da camada de aplicativo: HTTP (para navegadores da Web), SMTP (para e-mail), TCP e outros. No entanto, assim que uma solicitação é recebida, as mensagens SOAP de retorno devem ser retornadas como documentos XML — uma linguagem de marcação legível por humanos e máquinas. Uma solicitação concluída para uma API SOAP não pode ser armazenada em cache por um navegador, portanto, não pode ser acedida posteriormente sem ser reenviada para a API. SOAP é um protocolo pesado. Apesar dessas desvantagens, o SOAP é um protocolo seguro e confiável.

Através das figuras 14 e 15 pode se ver a diferença entre XML e JSON, o código de exemplo dos documentos da API do *Crowd Server da Atlassian* que permite solicitar e aceitar dados nos formatos *XML e JSON*:



```
<?xml version="1.0" encoding="UTF-8"?>
<authentication-context>
    <username>my_username</username>
    <password>my_password</password>
    <validation-factors>
        <validation-factor>
            <name>remote_address</name>
            <value>127.0.0.1</value>
        </validation-factor>
    </validation-factors>
</authentication-context>
```

Figura 18 - Exemplo em XML [22]

2.3.2.2 Frameworks para Back-end

Estruturas de *back-end* são essenciais para o desenvolvimento de aplicativos para inúmeras empresas em todo o mundo e existem diversas *frameworks* disponíveis para tal. Para cobrir três das linguagens mais populares (*Java*, *Javascript*, *C#*) é essencial abordar uma *framework* baseada em cada uma delas.

ASP.NET core [23]

ASP.NET core é um *framework* de código aberto, multiplataforma, desenvolvido pela Microsoft. Amplamente usado em *back-end*, é mantido e atualizado pela .NET Foundation. ASP.NET é uma estrutura modular que pode ser executada em todo o .NET Framework no Windows e .NET. A maior vantagem de usar a estrutura ASP.NET é a melhoria de desempenho que ela oferece como também a opção de usar padrões de programação assíncrona. Uma aplicação WEB utilizando ASP.NET corre num servidor web IIS. Uma instância de IIS pode albergar vários websites. Quando se corre a partir do visual studio, este automaticamente cria uma instância do IIS Express, que é uma versão mais leve destinada ao desenvolvimento.

ExpressJS [23]

Express JS é uma *framework* para o Node JS e software de código aberto disponível sob a licença do MIT. É usado para construir *APIs* e aplicativos da web, é considerado uma estrutura de servidor Node.js padrão, ExpressJS é um componente de *back-end* da pilha MEAN junto com a estrutura de *front-end AngularJS* e base de dados.

Node.js é um ambiente JavaScript que corre do lado servidor. É leve, escalável, e permite executar código em multiplataformas. Não necessita de muito esforço para se aprender e é uma das linguagens de programação mais utilizadas.

Spring [23]

Spring Framework é uma estrutura de aplicativo de código aberto e o recipiente de inversão de controle da plataforma Java. Os aplicativos Java podem utilizar os principais recursos desta estrutura. Os usuários também podem usar muitas extensões para criar aplicativos da web baseados na plataforma Java EE. Um dos produtos que aparecem mais destacados é o Spring Boot, e segundo o website é de simples e rápida configuração permitindo criação de *WEB APIs* e ligação com vários tipos de base de dados. Algumas das vantagens são o *springboot starters* e um simples gerenciamento de dependências.

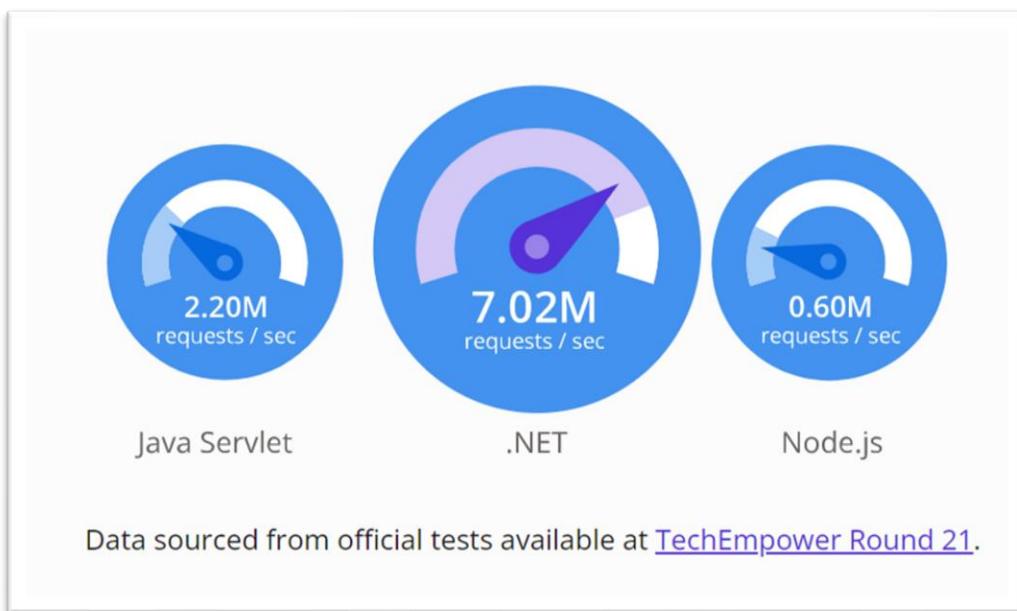


Figura 19 - Resultado obtido pela TechEmpower dos 3 melhores Frameworks [24]

Segundo a empresa *TechEmPower* que disponibiliza, testes de performance a várias *frameworks*, é possível perceber que ASP.NET se destaca em relação aos seus principais concorrentes, como a própria Microsoft destaca no seu website.

2.3.2.3 Frameworks para Front-end

O desenvolvimento web *front-end* é o processo de transformar os dados em uma interface gráfica, por meio do uso de CSS, HTML e JavaScript, para que os utilizadores possam observar

e interagir com esses dados. E a utilização de *frameworks* ajuda e simplifica esse trabalho, foram selecionadas quatro das mais utilizadas por grandes empresas a nível mundial.

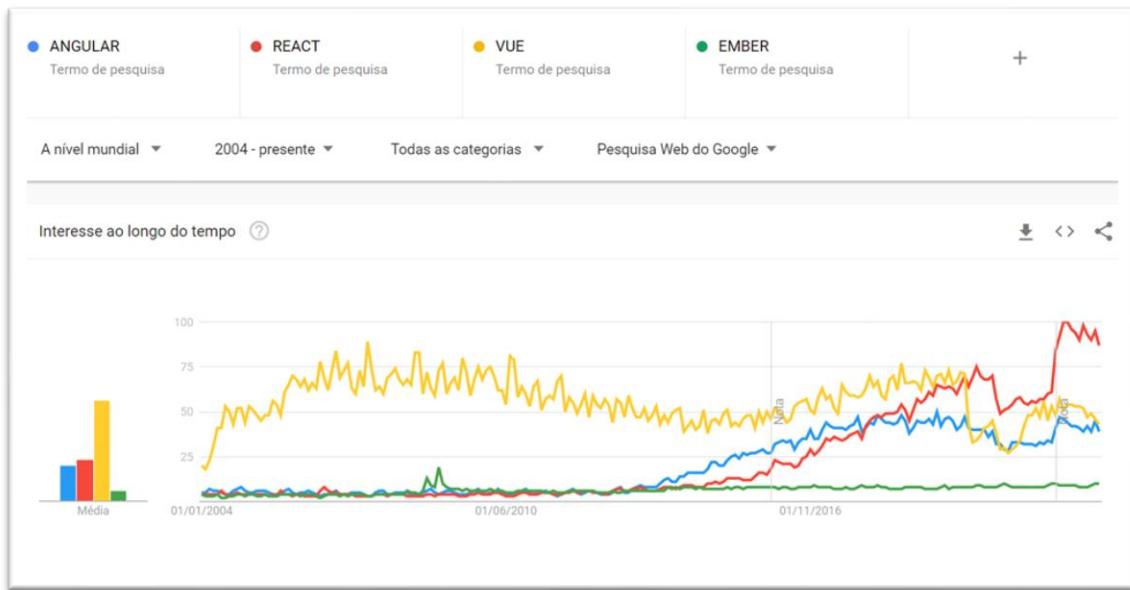


Figura 20 - Popularidade Frameworks Front-end [25]

Angular [23].[26]

É um *framework* de *front-end* moderno e de código aberto que funciona com base no *TypeScript*, com o objetivo de desenvolver aplicações em projetos em pequena ou larga escala. Esta estrutura é mantida e desenvolvida pelo Google. Tendo seu suporte totalmente orientado ao mobile, este é um *framework* cujo código pode ser reescrito para web, web mobile, mobile *native* e desktop *native*. Seu conjunto de bibliotecas possibilita o desenvolvimento de aplicações com qualidade surpreendente. Além disso, o Angular amplia o desempenho de daquelas criadas para navegadores ao usar a vinculação bidirecional de dados. Em contraste com o *React*, o Angular oferece ligação de dados bidirecional, além disso, o Angular oferece uma função de injeção de dependência hierárquica que torna os componentes de código reutilizáveis, testáveis e fáceis de controlar. É utilizado por empresas como: Microsoft office, Google, BMW, UPS, Deutsche Bank, Santander, Forbes, Samsung entre outras.

Ember [23].[26]

É um *framework* baseado em componentes, desenvolvido em 2011 uma estrutura em JavaScript de código aberto que é usada no desenvolvimento aplicativos de página únicas (*SPAs*). Pode-se criar aplicativos móveis e Web multifacetados. É um dos *frameworks front-end* mais rápidos ao renderizar o lado do servidor, também fornece vinculação de dados bidirecional, sincronizando a exibição e o modelo em tempo real semelhante ao Angular.

A curva de aprendizagem do Ember é bastante acentuada é um dos *frameworks* mais desafiadores para aprender devido à sua estrutura convencional e rígida. A comunidade deste *framework* permanece em expansão, com novos recursos e atualizações frequentes E é utilizado por empresas como: Linkedin, Tinder, Netflix, Twitch, Groupon, Square, Accenture e DigitalOcean entre outras.

React [23].[26]

O React.js surgiu em 2011 quando foi implantado no *feed* do Facebook. Tecnicamente, React.js é uma biblioteca de JavaScript de código aberto, usada para desenvolver interfaces de utilizadores, especialmente aplicações de página única (*SPAs*). Pode ser combinada com outros *frameworks* de JavaScript, como o Angular.JS. As principais características positivas são a rapidez, escalabilidade e simplicidade. Outra característica vantajosa é o DOM (*Document Object Model*) com vinculação de dados unidirecional. Graças ao DOM, o React fornece aos programadores um excelente desempenho e é considerado um dos *frameworks* mais fáceis de aprender. Essa estrutura *front-end* é agradavelmente amigável e fácil de aprender, tornando-o uma boa opção para iniciantes ou programadores menos experientes.

O React sendo uma biblioteca não mantém alguns recursos essenciais, portanto, ele foi projetado para funcionar com outras bibliotecas para tarefas como gerenciamento de estado, roteamento e interação com API. É utilizado por empresas como: Facebook, DropBox, Instagram, Walmart, Pinterest, Discord e Airbnb entre outras.

Vue [23].[26]

Atualmente, um dos *frameworks* mais simples. Tem um tamanho pequeno e apresenta dois benefícios principais – DOM visual e programação baseada em componentes. Também emprega ligação de dados bidirecional semelhante ao Angular. Permite criar pequenos aplicativos como mais complexos com facilidade. No entanto, apesar de versátil, ainda tem uma comunidade de programadores reduzida e não é popular entre as grandes empresas,

porque a maioria das empresas de grande porte ainda confia no Angular para aplicativos da Web empresariais. É utilizado por empresas como: Alibaba, Gitlab, BridgeU, Xiaomi, FindlayWebTech e Reuters entre outras.

3 Análise e desenho da solução

Neste capítulo é apresentada a análise do problema e desenho da solução. Este pretende demonstrar a evolução e concretização no desenho da solução e os conceitos apreendidos na fase da análise. Partindo do modelo de domínio, segue-se a apresentação dos requisitos funcionais e não funcionais identificados, a descrição do sistema existente, e o desenho solução proposta.

3.1 Domínio do problema

Nesta secção é representado o Modelo de Domínio do projeto com todas as entidades e as relações entre elas, bem como algumas das propriedades mais importantes dessas entidades. São identificadas visualmente (recorrendo à estrutura estática UML – diagrama de classes) classes concetuais, permitindo identificar os conceitos de domínio, atributos e possíveis relações entre os mesmos. Por adotar uma classificação baseada em objetos, este artefacto é um intermediário importante entre os requisitos e o Design orientado a objetos, pois permite a identificação de classes candidatas (possíveis classes de software).

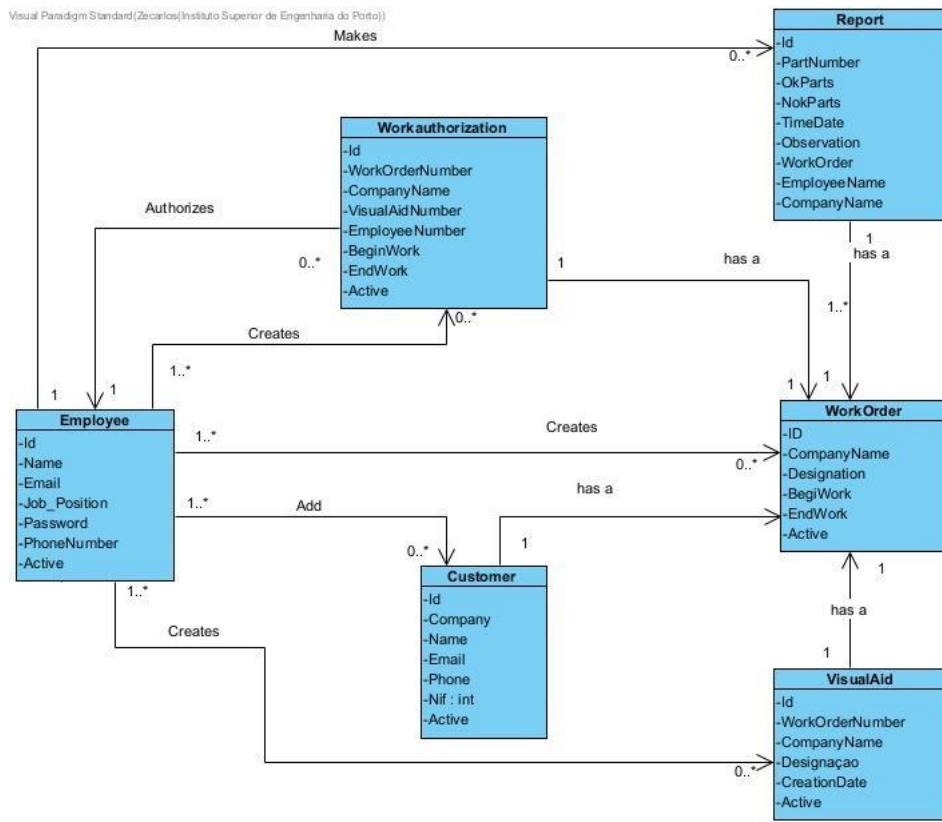


Figura 21 - Modelo Domínio

A Figura 18 encontra-se em anexo (Anexo A). Descrevem-se de seguida os conceitos de domínio para que o trabalho desenvolvido e apresentado em seções seguintes do relatório seja mais facilmente compreendido.

Termo	Descrição
Employee	Funcionário da empresa, que pode ter qualquer posto.
Customer	Cliente da empresa que pede um serviço.
Report	Relatório efetuado pelos funcionários da empresa depois de realizado o serviço pedido pelo cliente.
WorkOrder	Registo do pedido feito pelo cliente com alguma informação sobre o próprio.
WorkAuthorization	Registo de autorização com os funcionários que podem e sabem efetuar um serviço pedido por um cliente.
VisualAid	Informação de como efetuar o serviço pedido pelo cliente.

Tabela 1 - Glossário de termos.

Na solução teve-se em conta os seguintes pontos:

- Para um funcionário (*employee*) poder realizar um serviço precisa que tenha sido autorizado para tal, essa autorização fica guardada dentro do *workauthorization* com o id do funcionário, sem esta informação o funcionário não pode efetuar o serviço.
- Quando é efetuado um pedido de serviço por um cliente é criado um *workOrder*, onde fica registado a designação do mesmo, a data de início e de fim do trabalho.
- *WorkAuthorization*, tem associado um *WorkOrder*, um *VisualAid* e também os funcionários que podem efetuar o serviço.
- *VisualAid* tem a informação do que é para fazer no serviço pedido pelo cliente.
- *Report* de um serviço, só pode ser feito por funcionário que tenha autorização para tal, fica registado o *WorkOrder*, *Partnumber*, quantidade OK (válidas) e NOK (não válidas) e data de quando foi efetuado o serviço, para além do nome do funcionário e da empresa cliente.

3.2 Requisitos funcionais e não funcionais

Depois do estudo do problema na sua vertente teórica, conclui-se a necessidade dos seguintes requisitos.

3.2.1 Requisitos funcionais

UC1 - Registar funcionário (*employee*)

Para se registar um funcionário, devem fornecer-se os seguintes dados:

- Nome
- Email
- Número Telemóvel
- Cargo que vai ocupar

UC2 – Editar funcionário, alterar cargo, ativar ou desativar funcionário e eliminar.

UC3 – Pesquisar e visualizar funcionários.

UC4 –Registar cliente (*customer*)

Para se registar um cliente, devem fornecer-se os seguintes dados:

- Nome da empresa
- Nome do cliente que pede o serviço
- Email

- Número Telemóvel
- NIF da empresa

UC5 – Pesquisar e visualizar cliente.

UC6 - Ativar ou desativar cliente e eliminar registo de cliente.

UC7 – Registar um Pedido de serviço (*WorkOrder*)

Para se registar um *WorkOrder*, devem fornecer-se os seguintes dados:

- Nome da empresa
- Designação do pedido de serviço
- Indicar a data do início do serviço
- Indicar a data do fim do serviço

UC8 – Pesquisar e visualizar pedidos de serviço.

UC9 – Editar serviço, ativar ou desativar *WorkOrder*, alterar a data de fim do serviço e eliminar *workorder*.

UC10 – Registar uma ajuda visual (*VisualAid*).

Para se poder registar um ajuda visual, devem fornecer-se os seguintes dados:

- Id do *workorder*
- Nome da empresa que pediu o serviço
- Designação do serviço (*WorkOrder*)
- Data de criação da ajuda visual

UC11 – Pesquisar e visualizar ajudas visuais.

UC12 – Poder ativar ou desativar e eliminar a ajuda visual.

UC13 – Registar uma autorização de trabalho(*workAuthorization*)

Para se registar um *WorkAuthorization*, devem fornecer-se os seguintes dados:

- Id do *WorkOrder*
- Nome da Empresa que pediu o serviço
- Id da *VisualAid*
- Id do *Employee*
- Indicar a data do início do serviço
- Indicar a data do fim do serviço

UC14 – Pesquisar e visualizar autorizações de trabalho.

UC15 – Adicionar mais funcionários a uma autorização de trabalho, ativar ou desativar e eliminar o *workauthorization*

UC16 – Registar um relatório

Para se poder registar um relatório, devem fornecer-se os seguintes dados:

- Referencia do material
- Quantidade de peças Ok
- Quantidade de peças Nok
- Data da execução do serviço
- Permitir observações
- Id do *workorder*
- Nome da empresa que pediu o serviço
- Nome do funcionário que efetuou o serviço

UC17 – Pesquisar e visualizar relatório

UC18 – Eliminar relatório

Atores intervenientes no sistema

Um ator representa um conjunto coerente de funções que os utilizadores do sistema desempenham ao interagir com as suas funcionalidades. [27] O ator representa qualquer entidade que interage com o sistema, podendo ser uma pessoa ou um sistema.

Este sistema foi concebido para ser utilizado por vários tipos de utilizadores, como podemos ver com mais detalhe na tabela a seguir, estão definidos seis tipos de utilizador ou seis níveis para funcionários, onde está definido o acesso que cada nível tem ao sistema.

Nível	Nome	Descrição
0	<i>Intern</i>	O estagiário pode pesquisar e visualizar todos os dados, exceto dados de outros funcionários.
1	<i>Controler</i>	Podem criar e eliminar relatórios, mas o acesso a todas as funcionalidades do nível 0.
2	<i>Auditor</i>	
3	<i>Team Leader</i>	O perfil de team leader tem acesso a todas as funcionalidades exceto editar/eliminar cliente, pedido de serviço e ajuda visual.
4	<i>Supervisor</i>	O perfil de supervisor tem acesso a todas as funcionalidades.
5	<i>Manager</i>	O perfil de Administrador terá privilégio total sobre todos os contextos e funcionalidades disponíveis.

Tabela 2 - Tipos de utilizador e acesso ao sistema.

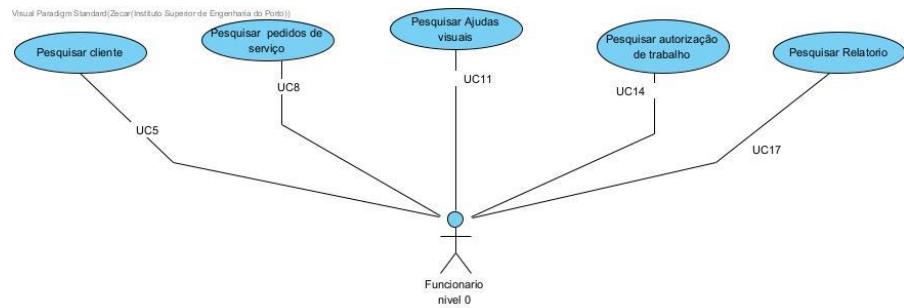


Figura 22 - Diagrama de casos de uso utilizador nível 0

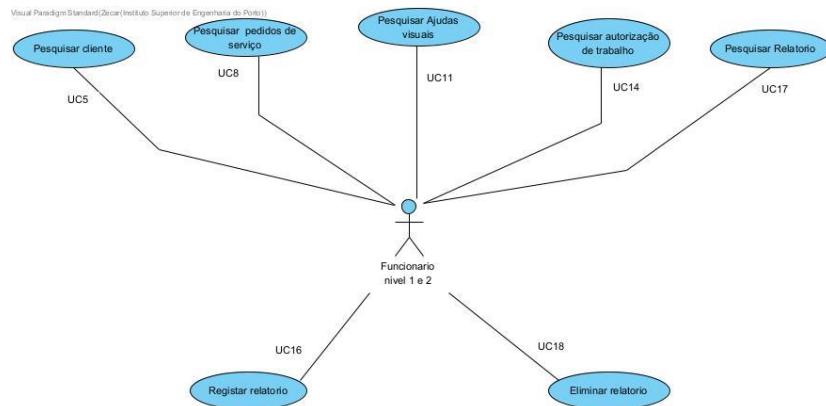


Figura 23 - Diagrama de casos de uso utilizador nível 1 e 2

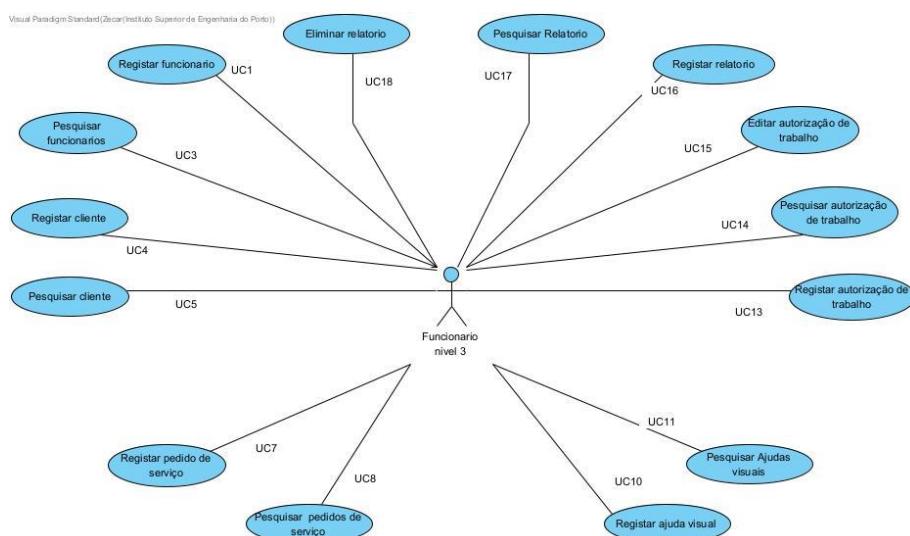


Figura 24 - Diagrama de casos de uso utilizador nível 3

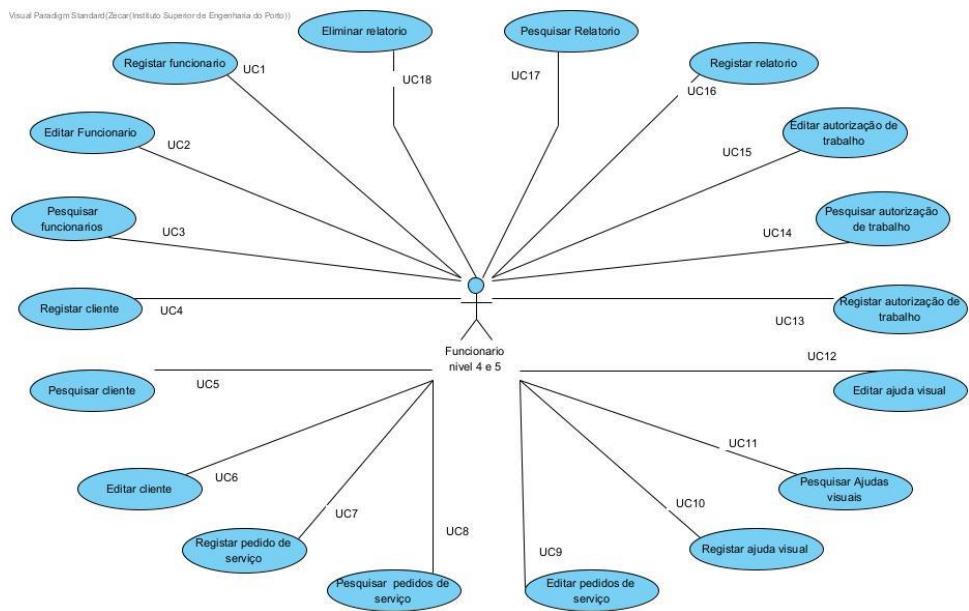


Figura 25 - Diagrama de casos de uso utilidor nível 4 e 5

3.2.2 Requisitos não funcionais

Roger Pressman [28] define os requisitos não funcionais como requisitos relacionados ao uso da aplicação em termos de desempenho, usabilidade, confiabilidade, segurança, disponibilidade, manutenção e tecnologias envolvidas. Os requisitos não funcionais possuem grande importância para o desenvolvimento do projeto e, por isso, foram baseados na classificação FURPS+.

A classificação FURPS+, sucessora do modelo FURPS, surgiu com a adição do sinal “+” e é geralmente usada para representar limitações impostas ao sistema. Essas limitações podem ocorrer, inclusivamente, a nível de desenho, implementação, comunicação (interface) ou mesmo a nível físico. [29]

O levantamento de requisitos não funcionais apresentado abaixo, segue o modelo FURPS.

Funcionalidades

Segurança/Autenticação: a utilização do sistema exige sempre autenticação.

Usabilidade

- Interface gráfica simples, intuitiva e eficiente, na aplicação web;
- O sistema (como um todo) deve ser simples e confortável de utilizar;
- Interface adaptável a diferentes resoluções de ecrã;
- O sistema deve estar dotado de mecanismos de pesquisa e ordenação de listas;

A interação entre os utilizadores e o sistema seja simples, intuitiva e completamente adaptada à ação em causa.

Fiabilidade/Confiabilidade

O sistema deve ser resiliente ao ponto de saber lidar com falhas de um ou mais componentes, o que significa que, no caso de uma falha imprevista de uma parte do sistema, o mesmo deve continuar a operar, não colapsando ou tornando-se inoperacional, utilizando para isso servidores duplos para o *frontend*, *backend* e SQL Server (*always on*).

Suportabilidade

O sistema deve ter um boa testabilidade, manutenção e confiabilidade nas infraestruturas.

Para o desenvolvimento deste projeto, foram definidos diferentes requisitos relacionados com a suportabilidade, tais como:

- O sistema deve permitir a expansão de funcionalidades sem um elevado custo de modificação;
- O sistema deve permitir a integração de novas funcionalidades, que possibilitem uma melhor experiência aos seus utilizadores.

3.3 Desenho

Após uma análise cuidada da informação gerada no âmbito das disciplinas de modelação de negócio e engenharia de requisitos, prossegue-se com o desenho arquitetural.

3.3.1 Diagrama de componentes

Os diagramas de componentes mostram a estrutura do sistema de software, que descreve os componentes do software, as suas interfaces e as suas dependências. Este tipo de diagrama suporta o desenvolvimento com base em componentes no qual um sistema de software é dividido em componentes e interfaces que são reutilizáveis e substituíveis.

De forma a entender a estrutura do sistema, serão apresentados os diagramas de componentes do sistema.

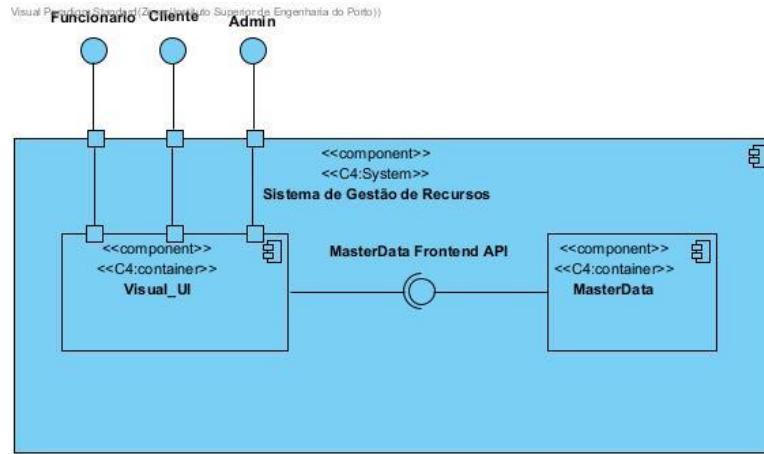


Figura 26 - Diagrama de Vista Logica Nível 2

Como se pode verificar no diagrama de vista logica nível 2, existem três tipos de acesso ao sistema, a comunicação entre os componentes *Visual_UI* e *MasterData* é efetuada através de API *RESTfull* recorrendo a métodos utilizando o protocolo HTTP (GET, POST, PUT, DELETE).

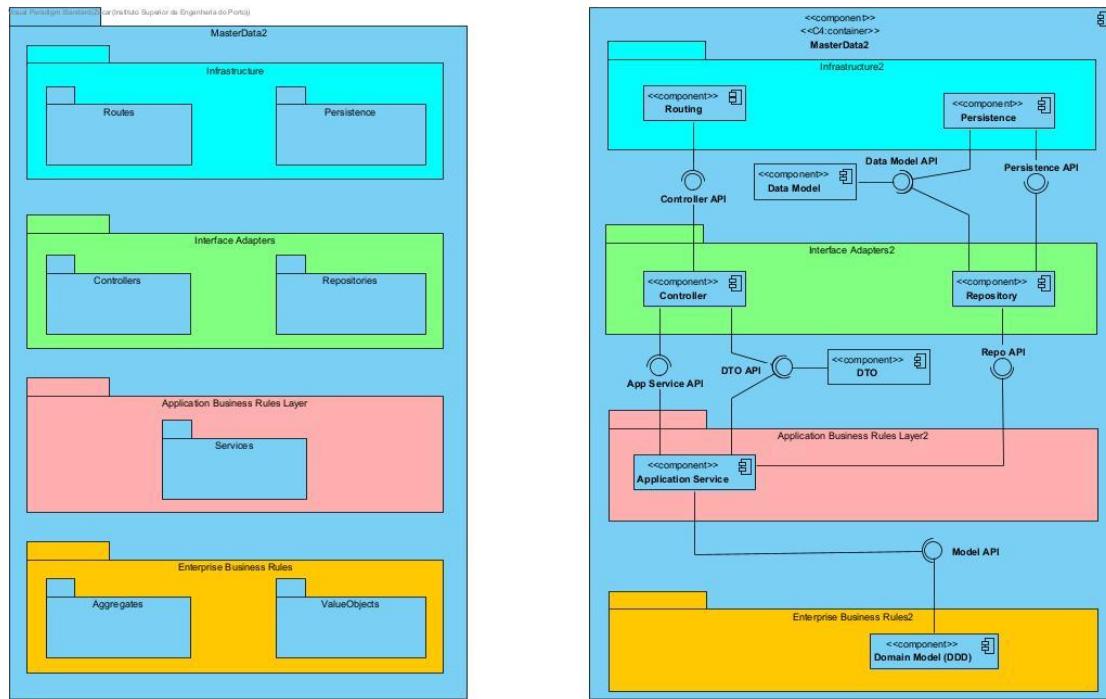


Figura 27 - Diagramas de vista de implementação e lógica Nível 3

Por se considerar o padrão mais adequado foi utilizado neste projeto a arquitetura *Onion/Clean Architecture*, entre as suas vantagens pode se encontrar:

- Manutenção, a arquitetura *Onion* ajuda a desacoplar as diferentes camadas de um sistema, o que torna mais fácil fazer alterações sem afetar outras partes.
- Melhor testabilidade, a clara separação de interesses na *Onion Architecture* facilita a escrita de testes automatizados para as diferentes camadas do sistema.
- Maior flexibilidade, como as camadas internas são independentes das camadas externas, é mais fácil fazer alterações ou adicionar novos recursos ao sistema sem interromper o restante sistema.

Este projeto foi criado para ser mutável e assim ter uma estrutura flexível, diminuindo a fragilidade que poderia ser causada por possíveis mudanças, como está a ser projetado para as necessidades de uma empresa em específico que é direcionada para prestação de serviços, este pode necessitar de adição de novas funcionalidades ou atualização das já existentes. A figura 25 mostra como foi estruturado.

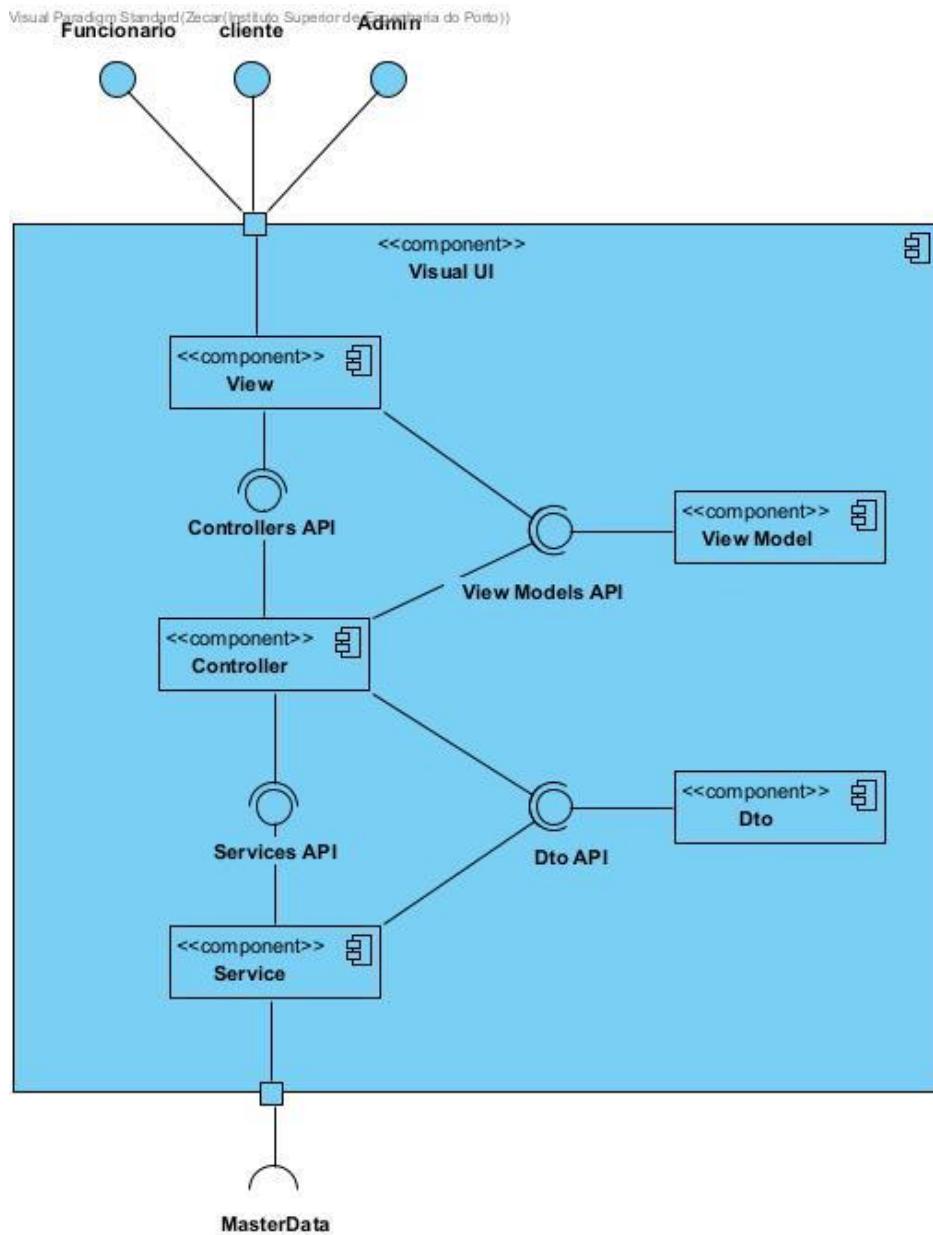


Figura 28 - Diagramas de vista de lógica Nível 3, Visual_UI

View – gera as interfaces mostradas ao utilizador, apresentando informações e interagindo com utilizador. Esta componente interage com as componentes: *Controller* e *View Model*

Controller - responsável por interpretar as ações de entrada, através do rato e teclado, realizadas pelo utilizador. O *Controller* envia essas ações para o *Service* e para a janela de visualização (*View*) onde são realizadas as operações necessárias.

Service - responsável por receber os pedidos do controller e solicitar pedidos á API *MasterData*.

3.3.2 Vista de casos de uso

Para representar a vista de casos de uso, elegeu-se o caso de uso UC1- Registar funcionário (*employee*), UC9 – Editar serviço, *workorder*. Os demais casos de excetuando algum pormenor são idênticos a estes.

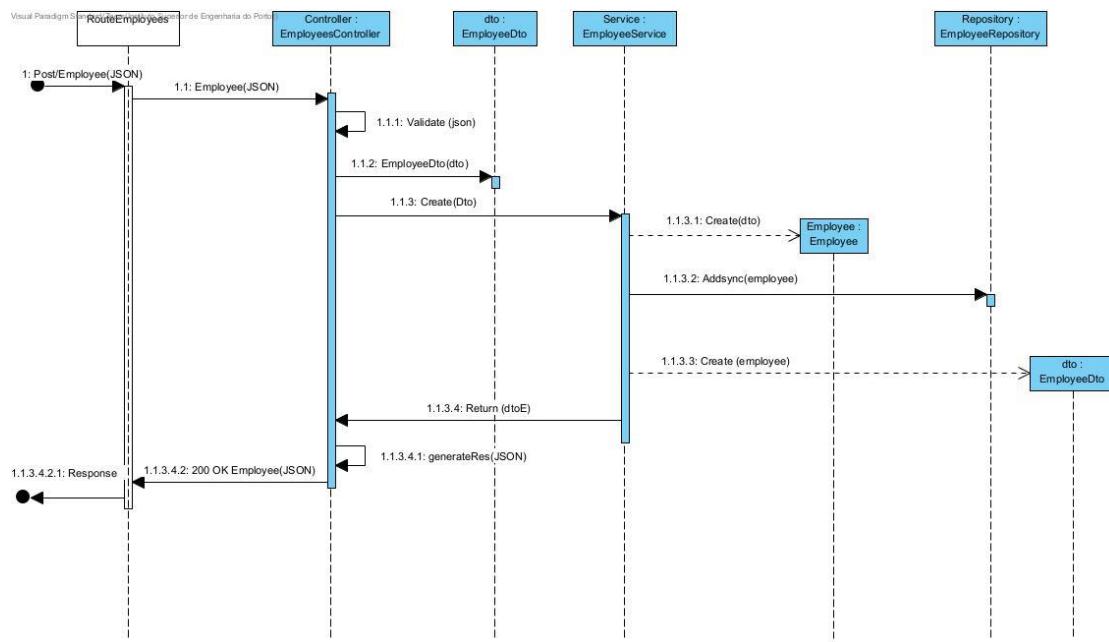


Figura 29 - Diagrama vista de processos nível 3, UC1

Para entender o relacionamento entre os componentes, tomemos como exemplo a Figura 27, que mostra a criação do *Employee* através de um diagrama de sequência. O método *Create()* da classe *EmployeeController* recebe a solicitação, cria *Dto EmployeeDto* e, em seguida, chama o método *Addsync(dto)* da classe *EmployeeService*. O método *Addsync(dto)* da classe *EmployeeService* recebe o *Dto* e tenta criar o objeto *Employee*. Na classe *Employee*, as regras de negócio são verificadas, e se as regras forem atendidas, o objeto *Employee* é criado, caso contrário, é enviada uma *BusinessRuleValidationException*. Em seguida, chama o método *AddAsync* da classe *EmployeeRepository* e adiciona a informação na base de dados por meio da *Persistence*.

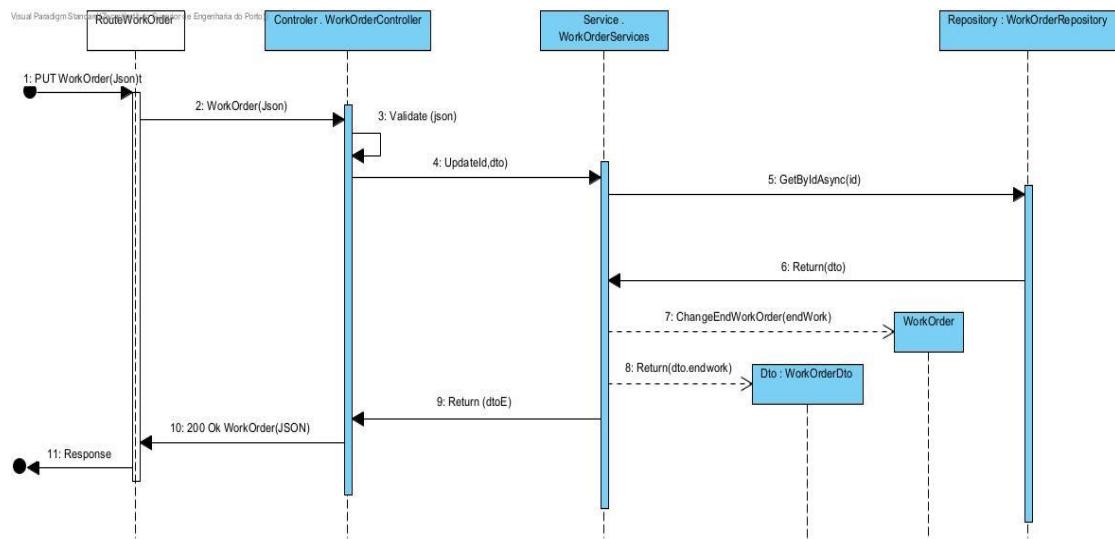


Figura 30 - Diagrama vista de processos nível 3, UC9

Para esta UC o *WorkOrderController* recebe no método *update(Id,dto)* um *Id* e um *dto* com uma nova data de fim do pedido de trabalho e o *Id* do *WorkOrder* a alterar,

de seguida chama o método *UpdateAsync(dto)* da classe *WorkOrderService*, este chama um outro método *GetByIdAsync(Id)* que retorna o objeto a alterar, por fim na classe *WorkOrder*, as regras de negócio são verificadas, e se as regras forem atendidas, o objeto *WorkOrder* é alterado, caso contrário, é enviada uma *BusinessRuleValidationException*.

4 Implementação da Solução

Este capítulo é dedicado à apresentação de detalhes relacionados com o enquadramento e implementação das soluções preconizadas no capítulo anterior, como foi implementada a proposta de solução ao problema inicialmente descrito e os seus testes unitários e de integração.

4.1 Descrição da implementação

Para se dar início ao processo de implementação, definiram-se as principais ferramentas que permitiram a concretização da solução proposta.

Bitbucket - Foi utilizado para controlo de versões e como repositório do projeto, de forma menos exaustiva foram utilizados os pipelines automatizados para testes de compilação.

Docker – Foi utilizado como *container* da base de dados, pela sua versatilidade.

Visual Studio Code – Foi utilizado como IDE devido a sua facilidade de utilização e grande variedade de extensões, sendo também leve e rápido.

Postman – Foi utilizado para a criação dos testes de integração, pela facilidade com que permite gerar e testar APIs.

Onion/ Clean Architecture -Foi empregue este tipo de arquitetura pelas razoes descritas no capítulo 3, concisamente pela manutenção, flexibilidade e testabilidade.

ASP.NET Core – Foi preferido em relação aos demais *frameworks* devido a sua flexibilidade, eficiência e eficácia, segurança e ser multiplataforma.

Angular – Foi escolhida esta *framework* devido a facilidade de aprendizagem, pela facilidade de se adaptar para web, mobile e desktop. Poder ser reutilizável, ser robusto e de ser menos complexo ao nível que se aumenta a sua estrutura.

4.1.1 BackEnd (ASP.NET Core em Clean Architecture)

4.1.1.1 Controller

O *controller* é uma camada que fica entre a camada de interface do usuário (UI) e a camada de negócios (*application services*), é responsável por intermediar a comunicação entre essas camadas, garantindo que os comandos do utilizador sejam convertidos em ações específicas no sistema. Utiliza um conjunto de princípios conhecidos como os princípios RESTful, que incluem o uso de recursos identificáveis por URI (*Uniform Resource Identifier*), o uso de verbos HTTP (como *GET*, *POST*, *PUT* e *DELETE*) para indicar a ação desejada, recebidas pela UI e direcionando-as para os respetivos métodos no *Service*. Como existe a necessidade de utilizar vários métodos, utiliza-se variações de caminhos com esses quatro URI's.

```
[Route("api/[controller]")]
[ApiController]

public class WorkAuthorizationsController : ControllerBase
{
    private readonly WorkAuthorizationService _service;

    public WorkAuthorizationsController(WorkAuthorizationService service)
    {
        _service = service;
    }

    // GET: api/WorkAuthorizations
    [HttpGet]
    public async Task<ActionResult<IEnumerable<WorkAuthorizationDto>>> GetAll()
    {
        ...
    }

    // GET: api/WorkAuthorizations/5
    [HttpGet("{id}")]
    public async Task<ActionResult<WorkAuthorizationDto>> GetById(Guid id)
    {
        ...
    }

    // GET: api/WorkAuthorization/company
    [HttpGet("workAuthorization/{company}")]
    public async Task<ActionResult<IEnumerable<WorkAuthorizationDto>>> GetWorkAuthorizationByCompanyAsync(string company)
    {
        return await _service.GetWorkAuthorizationByCompanyAsync(company);
    }

    // POST: api/WorkAuthorizations
    [HttpPost]
    public async Task<ActionResult<WorkAuthorizationDto>> Create(WorkAuthorizationDto dto)
    {
        var cust = await _service.AddAsync(dto);

        return CreatedAtAction(nameof(Get GetById), new { id = cust.Id }, cust);
    }

    // PUT: api/WorkAuthorizations/5
    [HttpPut("{id}")]
    public async Task<ActionResult<WorkAuthorizationDto>> Update(Guid id, WorkAuthorizationDto dto)
    {
        if (id != dto.Id)
        {
            return BadRequest();
        }
    }
}
```

Figura 31 - Controller WorktAuthorization

"Async" e "Await"

A programação assíncrona é útil em várias situações, como operações de rede, operações de base de dados e outras operações de I/O, que podem ser demoradas e bloquear o fluxo de execução, isso permite que a aplicação continue a responder a outras requisições enquanto aguarda a conclusão de uma operação específica.

4.1.1.2 Service

O *service* é a camada que representa as regras de negócio da aplicação. Ele é responsável por processar as requisições do *controller* e realizar as operações necessárias no *model*, é projetado para conter a lógica de negócios da aplicação, incluindo validações, cálculos, e interações com outros sistemas. Sendo também independentes do *framework* ou da plataforma, o que significa que podem ser reutilizados em outros projetos ou aplicações.

Os *services* são geralmente projetados como classes simples e fáceis de testar, que contêm métodos que correspondem a operações específicas na aplicação, como "criar *employee*", "atualizar *workOrder*" ou "eliminar *report*". São chamados pelos *controllers*, para processar as requisições dos utilizadores e preparar as respostas.

Na *Clean Architecture*, a camada de *services* é projetada para ser independente das camadas de UI e de *persistence*, o que significa que ela não tem dependência de como os dados são armazenados ou como eles são exibidos para o utilizador. Tornando a aplicação mais fácil de entender, manter e escalar.

```
public class WorkAuthorizationService
{
    private readonly IUnitOfWork _unitOfWork;
    private readonly IWorkAuthorizationRepository _repo;

    public WorkAuthorizationService(IUnitOfWork unitOfWork, IWorkAuthorizationRepository repo){ ... }

    public async Task<List<WorkAuthorizationDto>> GetAllAsync(){ ... }
    public async Task<WorkAuthorizationDto> GetByIdAsync(WorkAuthorizationId id){ ... }
    public async Task<List<WorkAuthorizationDto>> GetWorkAuthorizationByCompanyAsync(string company) { ... }
    public async Task<WorkAuthorizationDto> AddAsync(WorkAuthorizationDto dto){ ... }
    public async Task<WorkAuthorizationDto> UpdateAsync(WorkAuthorizationDto dto){ ... }
    public List<string> AddListS(List<ListOfString> list){ ... }
    public async Task<WorkAuthorizationDto> InactivateAsync(WorkAuthorizationId id){ ... }

    public async Task<WorkAuthorizationDto> DeleteAsync(WorkAuthorizationId id)
    {
        var wor = await this._repo.GetByIdAsync(id);

        if (wor == null)
            return null;

        if (wor.Active)
            throw new BusinessRuleValidationException("It is not possible to delete an active WorkAuthorization.");

        this._repo.Remove(wor);
        await this._unitOfWork.CommitAsync();

        return new WorkAuthorizationDto { Id = wor.Id.AsGuid(), WorkOrderNumber=wor.WorkOrderNumber,
                                         CompanyName = wor.CompanyName,
                                         VisualAidNumber=wor.VisualAidNumber,
                                         EmployeeNumber=AddListS(wor.EmployeeNumber),
                                         BeginWork=wor.BeginWork,
                                         EndWork=wor.EndWork,
                                         Active =wor.Active};
    }
}
```

Figura 32 - Service WorkAuthorization

4.1.1.3 Repository

O *repository* é um componente que fornece acesso aos dados de uma aplicação, separando a lógica de acesso a dados da lógica de negócios. Isso permite que a lógica de negócios seja independente das fontes de dados, permitindo que a aplicação seja facilmente testada e adaptada para usar diferentes fontes de dados. Além disso, o *repository* também pode aplicar regras de negócios específicas antes de retornar dados para a aplicação. Desta forma ajuda a manter a aplicação organizada e escalável.

```
public class EmployeeRepository: BaseRepository<Employee, EmployeeId>, IEmployeeRepository
{
    private readonly DbSet<Employee> _contextJ;

    public EmployeeRepository(DDDSample1DbContext context):base(context.Employees)
    {
        _contextJ = context.Employees;
    }

    public async Task<List<Employee>> GetByNameAsync (string Name)
    {
        var rec =await this._contextJ.Where(u=> u.Name.Contains(Name)).ToListAsync();

        return rec;
    }

    public async Task<Employee> GetByEmailPassAsync(string user, string password)
    {
        var rec = await this._contextJ.SingleOrDefaultAsync(u => u.UserEmail.Email == user && u.Password == password);

        return rec;
    }
}
```

Figura 33 - Repository Employee

4.1.1.4 EntityTypeConfiguration

O principal objetivo de *EntityTypeConfiguration* é definir como as entidades são mapeadas para a base de dados, incluindo quais propriedades são mapeadas para quais colunas e como os relacionamentos entre as entidades são representados no base de dados. Ao configurar esse mapeamento em uma classe separada, a lógica de negócios da aplicação pode ser desacoplada dos detalhes de como os dados são armazenados e recuperados, tornando-os mais flexíveis e fáceis de testar, é frequentemente usado em combinação com o padrão *Repository*, onde o *Repository* é responsável por interagir com o ORM e o *EntityTypeConfiguration* é responsável por mapear as entidades para a estrutura da base de dados.

```

public class EmployeeEntityTypeConfiguration : IEntityTypeConfiguration<Employee>
{
    public void Configure(EntityTypeBuilder<Employee> builder)
    {
        builder.OwnsOne(j => j.Job_Position).Property(pn => pn.job_Position);
        builder.OwnsOne(u => u.UserPhoneNumber, phoneNumber =>
        {
            phoneNumber.Property(pn => pn.PhoneNumber).IsRequired();
        });
        builder.OwnsOne(u => u.UserEmail, email => { email.HasIndex(e => e.Email).IsUnique(); });
        builder.HasKey(b => b.Id);
    }
}

```

Figura 34 - EntityTypeConfiguration Employee

4.1.1.5 Domain Model

O *Domain Model* faz parte da camada mais interna da aplicação, chamada Camada de Domínio, que é responsável por conter a lógica de negócio central da aplicação. É totalmente independente das demais camadas da aplicação, inclusive da camada de infraestrutura, que é responsável por interagir com sistemas externos como bases de dados e web services. Este deve ser desenvolvido para ser o mais simples e desacoplado possível, para que possa ser facilmente compreendido, testado e mantido. Ele deve ser focado nos conceitos-chave do domínio e não deve conter nenhuma lógica ou dependências desnecessárias. Isso permite que a aplicação seja facilmente adaptada a novos requisitos ou a novas regras de negócios.

Na figura a baixo segue o exemplo de uma validação de um valueobject criado na *class Employee* que se não valido lança uma *BusinessRuleValidationException*.

```

public class PhoneNumberType : ValueObject
{
    public string PhoneNumber { get; private set; }

    private PhoneNumberType()
    {
        // EFCore needs for proper working
    }

    public PhoneNumberType(string phoneNumber)
    {
        if (phoneNumber == null)
            throw new BusinessRuleValidationException("Phone number can't be null!");

        if (!IsValidPhoneNumber(phoneNumber))
            throw new BusinessRuleValidationException("Phone number is not Valid!");

        this.PhoneNumber = phoneNumber;
    }

    public static bool IsValidPhoneNumber(string phoneNumber)
    {
        string phoneNumberRegexPattern = @"^(\+|00)[0-9]{3}[0-9]{4,11}$";

        try
        {
            return Regex.IsMatch(phoneNumber, phoneNumberRegexPattern, RegexOptions.IgnoreCase, TimeSpan.FromMilliseconds(250));
        }
        catch (RegexMatchTimeoutException)
        {
            return false;
        }
    }
}

```

Figura 35 - ValueObject PhoneNumberType

```

public class Employee : Entity<EmployeeId>, IAggregateRoot {

    public string Name { get; private set; }

    public Job_PositionType Job_Position { get; private set; }

    public EmailType UserEmail { get; private set; }

    public PhoneNumberType UserPhoneNumber { get; private set; }

    public string Password { get; private set; }

    public bool Active { get; private set; }

    public Employee()
    {
        this.Active = true;
    }

    public Employee( string name, int job_Position, string userEmail, string userPhoneNumber, string password) {
        this.Id = new EmployeeId(Guid.NewGuid());
        this.Name = name;
        this.Job_Position= new Job_PositionType(job_Position);
        this.UserEmail=new EmailType(userEmail);
        this.UserPhoneNumber=new PhoneNumberType(userPhoneNumber);
        this.Password=password;
        this.Active = true;
    }

    public void ChangeJobPosition(int job_Position)
    {
        if (!this.Active)
            throw new BusinessRuleValidationException("It is not possible to change the job position to an inactive worker.");
        this.Job_Position = new Job_PositionType(job_Position);
    }
    public void MarkAsInactive()
    {
        this.Active = false;
    }
}

```

Figura 36 - Class Employee

4.1.2 FrontEnd (Angular)

4.1.2.1 Component

Em Angular o *component* é uma classe que controla uma parte da interface do utilizador, conhecida como *view*. Ele contém a lógica para exibir dados e lidar com as interações do utilizador através de vários periféricos de entrada. Um *component* é definido por uma classe, que contém propriedades e métodos, que definem a estrutura e o layout da interface do utilizador. Por exemplo na figura 35 existe um método chamado *onsubmit()* em que depois do utilizador preencher os dados necessários na *view* e clicar para submeter, é verificado se todos os campos foram preenchidos, se sim é enviado para o método *AddReport()* no *reportService*, se receber uma resposta de sucesso é criado o objeto *Report* se não é lançada uma mensagem de erro a dizer para preencher os campos em falta.

```
@Component({
  selector: 'app-report',
  templateUrl: './report.component.html',
  styleUrls: ['./report.component.css']
})
export class ReportComponent implements OnInit {
  constructor(private reportService: ReportService,private employeeService: EmployeesService,
  private fb: FormBuilder, public dialog: MatDialog,) { }
  report: any;
  isShown: boolean = false;
  registReport!: ReportDTO;
  local?:any;
  nameUser?:any;
  idType!:string;
  companyType!:any;
  hide = false;
  removable = true;
  userType!: string;
  readonly separatorKeyCodes = [ENTER, COMMA] as const;

  pesqBool() {...}
  pesqBoolFalse(){...}
}

registerUserFormGroup: FormGroup = this.fb.group({...});
});

async onSubmit(){
  if(this.registerUserFormGroup.valid){
    this.registReport = <ReportDTO>this.registerUserFormGroup.value;
    this.registReport.workOrder=this.idType;
    this.registReport.companyName=this.companyType;
    this.registReport.employerName=this.nameUser;
  if(this.registReport.employerName==null|| this.nameUser==null){
    console.log(this.companyType);

    Swal.fire('Todos os campos devem ser preenchidos');
  }else{
    (await this.reportService.addReport(this.registReport)).subscribe((log: IReport) => {
      Swal.fire('Relatório criado com sucesso.');
      this.report = log;
      this.pesqBool() ;
    }));
  }
}
}
```

Figura 37 - Component Report (Angular)

4.1.2.2 Service

O service é usado para compartilhar dados e lógica entre diferentes componentes da aplicação como comunicar com APIs externas. Ele é usado para encapsular funções e dados que são comuns a vários componentes, e pode ser injetado em outros componentes através do mecanismo de injeção de dependência do Angular. Isso permite que se mantenha o código limpo e organizado, separando a lógica de negócios da interface de utilizador. Exemplo da figura a baixo, através do método `addCustomer()` é enviado um POST para o `controller` em `backend`, se o pedido for bem sucedido é criado um novo `customer`.

```

@Injectable({
  providedIn: 'root'
})
export class CustomerService {
  baseUrl = environment.masterDataApi.ui + '/Customers';
  constructor(private http: HttpClient) { }
  httpOptions = {
    headers: new HttpHeaders({ 'Content-Type': 'application/json' })
  };

  addCustomer(dto: CustomerDTO): Observable<ICustomer> {
    return this.http.post<ICustomer>(this.baseUrl, dto, this.httpOptions).pipe(
      tap((newLog: ICustomer) => this.log(` w/ id=${newLog.id}`)),
      catchError(this.handleError<ICustomer>('AddCustomer'))
    );
  }

  /*Pesquisa Customer por company recebe infomação de backend*/
  getReportByCompany(Company: string): Observable<ICustomer[]> { ... }

  /*Pesquisa por todos customer, recebe infomação de backend*/
  getCustumerAll(): Observable<ICustomer[]> { ... }

  /*Pesquisa customer por id recebe infomação de backend*/
  getCustumerById(Id: string): Observable<ICustomer[]> { ... }

  /*Elimina Customer por id envia infomação para backend*/
  deleteCustomerById(id: string): Observable<ICustomer> { ... }

  /*Desativa customer por id envia infomação para backend*/
  deactivateCustomerById(id: string): Observable<ICustomer> { ... }

  private log(message: string) {
    if (message == 'AddCustomer failed') {
      Swal.fire('Erro - Nenhuma alteração efectuada')
    }
  }
}

```

Figura 38 - CustomerService (Angular)

4.1.2.3 View

A view é uma representação visual dos dados e lógica que estão presentes em um componente. Ela é geralmente definida como um template HTML que é associado ao componente e contém diretivas, componentes e interpolações que exibem os dados e as informações de estado. A view é atualizada automaticamente quando os dados ou o estado do componente mudam, garantindo que a interface do utilizador esteja sempre sincronizada com o estado atual da aplicação. A view é responsável por mostrar ao utilizador o resultado da lógica e dados processados pelo componente. Na figura 40 pode se ver um exemplo de criação de um novo funcionário e na figura 41 a pesquisa de relatórios.

GECUR-Gestor de Recursos

Creat employee profile

Name *
Antonio esteves

Email *
esteves@email.com

Phone number *
+312951357456

Include area code
Password *

qwertyui

Job Position
Team Leader

Submit

Figura 39 - View Adicionar Employee

GECUR-Gestor de Recursos

Search Reports

Search..

All Name Company

Search

id	employerName	companyName	timeDate	observation
9872efb9-ba7d-488a-aa3a-b4fed87ceb56	Helena Pereira	Grohe	2021-11-01T00:00:00	Nada a acrescentar
c299e027-31b1-4c5a-9e00-233f7a1a95d	Helena Pereira	Grohe	2021-11-01T00:00:00	Nada a acrescentar
d5ad50a4-3e67-4cac-8a68-41ec6aff5a4a	Helena Pereira	Grohe	2021-11-01T00:00:00	Nada a acrescentar

Items per page: 5 | 1 – 3 of 3 | < > >>

Resultado

Id : 9872efb9-ba7d-488a-aa3a-b4fed87ceb56

Employee Name : Helena Pereira

Company : Grohe

Ok Parts : 22

Nok Parts : 22

Date : 2021-11-01T00:00:00

Work Order: a3f5c030-eb74-4521-af9c-04f80af8bbbd

Observation : Nada a acrescentar

Limpar

Figura 40 - View Pesquisar Report

4.2 Testes

Para este projeto foram efetuados testes unitários e de integração. Os testes unitários foram efetuados de forma a testar o código e métodos utilizados como também todos os Services. Os testes de integração foram efetuados de forma a se poder verificar se as diferentes partes do sistema como um todo estão a funcionar corretamente, para estes testes recorreu-se a ferramenta *Postman* como mencionado anteriormente, enviando-se solicitações HTTP (*como* GET, POST, PUT e DELETE) e verificando se as respostas retomadas eram as esperadas.

4.2.1 Testes Unitários

Foram efetuados testes unitários a todas as classes para se tentar garantir que estariam a funcionar corretamente, como também aos serviços. Uma vez que muitos dos testes serão muito parecidos, irá-se dar o exemplo de alguns.

```
----- Test Execution Summary -----  
  
TesteUnitMasterD.EmployeeTests.EmployeeConstructor  
| Outcome: Passed  
  
TesteUnitMasterD.EmployeeTests.TestJobValidate:  
| Outcome: Passed  
  
TesteUnitMasterD.EmployeeTests.TestEmailValidate:  
| Outcome: Passed  
  
TesteUnitMasterD.EmployeeTests.TestPhoneValidate:  
| Outcome: Passed  
  
Total tests: 4. Passed: 4. Failed: 0. Skipped: 0
```

Figura 41 - Testes unitários classe Employee

EmployeeConstructor() – O objetivo deste teste é verificar se atendendo as regras de negócio o objeto é criado com sucesso.

TesteJobValidate() - O objetivo deste teste é assegurar que não foi violada uma regra de negócio através do “*BusinessRuleValidationException*”, neste caso que não foi inserido uma posição que não existe.

TesteEmailValidate() - O objetivo deste teste é assegurar que não foi violada uma regra de negócio através do “*BusinessRuleValidationException*”, neste caso que não foi inserido email incorreto ou vazio.

TestePhoneValidate() - O objetivo deste teste é assegurar que não foi violada uma regra de negócio através do “*BusinessRuleValidationException*”, neste caso que não foi inserido um número de telefone incorreto ou vazio.

```
----- Test Execution Summary -----  

MasterDataTest.WorkAuthorizationServiceTests.TestUpdateAsync:  

| Outcome: Passed  

MasterDataTest.WorkAuthorizationServiceTests.Test GetByIdAsync:  

| Outcome: Passed  

MasterDataTest.WorkAuthorizationServiceTests.TestGetByCompanyAsync:  

| Outcome: Passed  

MasterDataTest.WorkAuthorizationServiceTests.Test GetAllAsync:  

| Outcome: Passed  

MasterDataTest.WorkAuthorizationServiceTests.TestAddAsync:  

| Outcome: Passed  

Total tests: 5. Passed: 5. Failed: 0. Skipped: 0
```

Figura 42 - Testes unitários classe WorkAuthorizationService

TesteUpdateAsync() - O objetivo deste teste é assegurar que se podia adicionar novos funcionários a autorização de trabalho, retornando a lista atualizada.

TesteGetByIdAsync() - O objetivo deste teste é assegurar que a autorização de trabalho retornada era a selecionada pelo Id.

TesteGetByCompanyAsync() - O objetivo deste teste é assegurar que a autorização de trabalho retornada era a identificada pelo nome da empresa do cliente.

Teste GetAllAsync() - O objetivo deste teste é assegurar que retornava toda a lista de autorizações de trabalho.

TesteAddAsync() - O objetivo deste teste é assegurar que é criado o objeto *workAuthorization* com todos os dados fornecidos.

```
----- Test Execution Summary -----  

MasterDataTest.ReportServiceTests.TestAddAsync:  

| Outcome: Passed  

MasterDataTest.ReportServiceTests.TestGetByCompanyAsync:  

| Outcome: Passed  

MasterDataTest.ReportServiceTests.TestDeleteAsync:  

| Outcome: Passed  

MasterDataTest.ReportServiceTests.Test GetByIdAsync:  

| Outcome: Passed  

MasterDataTest.ReportServiceTests.Test GetAllAsync:  

| Outcome: Passed  

Total tests: 5. Passed: 5. Failed: 0. Skipped: 0
```

Figura 43 - Testes unitários classe ReportService

TesteDeleteAsync() - O objetivo deste teste é assegurar que é eliminado o relatório definido e enviado por Id.

4.2.2 Testes de Integração

Os testes de integração efetuados pretendiam encontrar falhas de integração e comunicação entre as unidades, enviando-se solicitações HTTP (como GET, POST, PUT e DELETE) e verificando se as respostas retomadas eram as esperadas ou mesmo se estaria a responder.

Na figura a baixo pode-se verificar que ambos os 129 testes de integração foram validados com sucesso

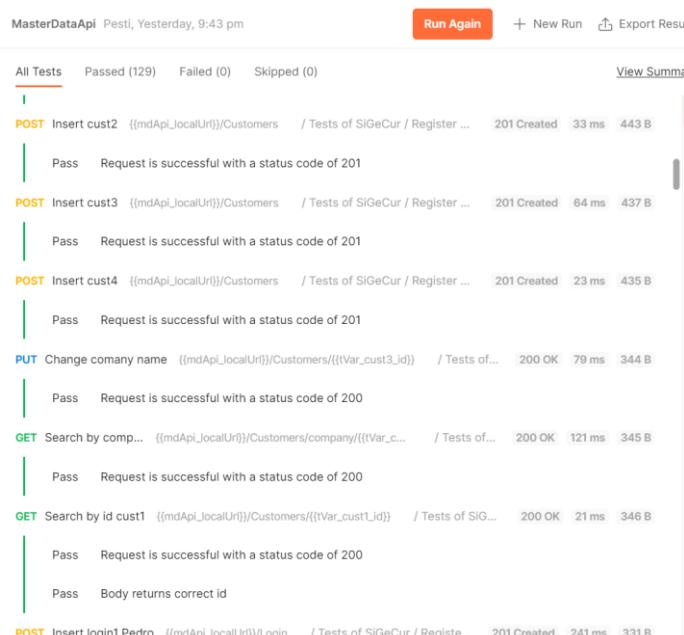


Figura 44 - Resultados teste de Integração

The screenshot shows a Postman interface with the following details:

- URL:** {{mdApi_localUrl}}/Employees
- Method:** POST
- Body:** JSON (Pretty)

```

1
2   ...
3     "Name": "Jose Santos",
4     "Job_Position": "3",
5     "UserEmail": "jose05@gmail.com",
6     "UserPhoneNumber": "+351911272421",
7     "Password": "12378678"

```

- Response Status:** 201 Created
- Response Body:** (Pretty JSON)

```

1
2   {
3     "id": "d599bd17-9d93-485a-8c33-6a8bd9364a80",
4     "name": "Jose Santos",
5     "job_Position": 3,
6     "userEmail": "jose05@gmail.com",
7     "userPhoneNumber": "+351911272421",
8     "password": "12378678",
9     "active": true

```

Figura 45 - Exemplo de um HTTP(POST)

Na figura 46 tem se um exemplo de um HTTP (POST) no qual é enviado a informação necessária para a criação de um *employee*, e na qual se recebe a resposta com a criação do mesmo.

The screenshot shows a Postman interface with the following details:

- URL:** {{mdApi_localUrl}}/Employees/{{tVar_emp3_id}}
- Method:** GET
- Pre-request Script:**

```

1 var responseJson = pm.response.json();
2
3 pm.test("Request is successful with a status code of 200", function () {
4   | pm.response.to.have.status(200);
5 });
6
7 pm.test("Check that not returns an empty array", function () {
8   | pm.expect(responseJson).to.not.eq([]);
9 });
10
11 pm.test("Body returns correct id", function () {
12   | pm.expect(responseJson.id).to.eql(pm.environment.get("tVar_emp3_id"));
13 });
14

```
- Test Results:**
 - PASS Request is successful with a status code of 200
 - PASS Check that not returns an empty array
 - PASS Body returns correct id
- Response Status:** 200 OK

Figura 46 - Exemplo de um HTTP(GET)

Na figura 33 pode se verificar o exemplo de HTTP (GET) com 3 testes, o primeiro se existe comunicação com o *Backend*, o segundo se a resposta recebida não é um array vazio e o último se o Id do *employee* enviado é o mesmo que o recebido.

4.3 Avaliação da solução

Neste subcapítulo é feita a avaliação da solução e o seu desenvolvimento ao longo do projeto de estágio.

Os resultados obtidos pelos testes unitários mostram que tanto as classes do modelo de domínio como as classes *Service*, estão a apresentar os resultados esperados e satisfatórios.

A capacidade de realização de simulações e testes de integração ao longo do desenvolvimento permitiu com que se fossem corrigindo alguns erros e os resultados dos testes finais fossem positivos.

O tempo de resposta da aplicação foram satisfatórios, porém não foram produzidos testes de desempenho em servidores externos. A aplicação foi sempre testada em *localhost* e em ambiente de desenvolvimento.

Como uma prova de conceito que visa reduzir tempos e custos e sendo também de fácil manuseamento e manutenção devido a sua simplicidade, pode se extrapolar que a aplicação proposta como solução ao problema apresentado exibe características satisfatórias quando equacionados os objetivos deste projeto de estágio.

5 Conclusões

Neste capítulo são apresentadas as conclusões de todo o trabalho realizado, todos os procedimentos envolvidos e dificuldades encontradas, ou seja, uma retrospectiva de tudo o que foi feito ao longo do projeto estágio.

O projeto iniciou-se com a realização de uma pesquisa de aplicações que pudessem servir como solução *off-the-shelf* ao problema apresentado no capítulo I. Esse estudo foi apresentado no Capítulo II e serviu também para ter alguns exemplos para este projeto.

Os primeiros passos foram a criação de diagramas para projetar (Capítulo III), analisar e modelar a aplicação e a criação da aplicação.

Durante o desenvolvimento do projeto foram se aprofundando conhecimentos sobre as tecnologias utilizadas como adquirindo novos conhecimentos, nomeadamente C#, *TypeScript*, HTML e outras demais tecnologias descritas no Capítulo IV. Durante a fase de desenvolvimento foram efetuados realizados testes unitários, que ajudaram na resolução de determinados erros, por fim foram efetuados os testes de integração. Após a conclusão do backend passou-se para o desenvolvimento do ambiente gráfico.

Houve algumas dificuldades e desafios a nível das tecnologias e métodos utilizados, que foram contornados com estudo, visualização de tutoriais, em síntese, foram realizadas as várias etapas de análise e desenvolvimento de um software de acordo com o que foi estudado durante o decorrer da licenciatura.

5.1 Objetivos concretizados

No início do projeto, foram definidos os objetivos gerais como específicos e as funcionalidades para a solução a desenvolver. Essas funcionalidades são, de seguida, apresentadas, bem como o seu grau de realização.

criação de um protótipo funcional	Cumprido
digitalização e automatização do processo de gestão relatórios	Cumprido
introdução nos processos de tecnologias de informação	Cumprido

Tabela 3 - Objetivos gerais concretizados

O team leader pode fazer todos os registo (funcionários, clientes, pedidos de serviço, ajudas visuais e autorizações de funcionários), só não pode eliminar clientes, pedidos de serviço e ajudas visuais.	Cumprido
O supervisor e ou Gestor tem acesso a todas as funcionalidades, inserir, editar, eliminar e pesquisar qualquer dos recursos.	Cumprido
Controlador e ou auditor só podem inserir os relatórios dos trabalhos efetuados, pesquisar por ajudas visuais, pedidos de serviço, relatórios e autorizações de serviço. Também poderão ver o histórico de relatórios criados.	Cumprido
Estagiário só pode fazer pesquisas por ajudas visuais, pedidos de serviço e relatórios	Cumprido
A Aplicação ser userfriendly e de fácil compreensão.	Cumprido
Efetuar Testes.	Cumprido

Tabela 4 - Objetivos específicos concretizados

5.2 Limitações e trabalho futuro

Os objetivos deste projeto foram alcançados, contudo como qualquer aplicação ou software há sempre aspetos a melhor e a adicionar. Neste caso sendo uma prova de conceito todo o desenvolvimento como testes foram feitos em ambiente local, só depois de ser testado em situações reais é que se pode garantir as suas capacidades. Como prova de conceito, e sendo um protótipo vertical, tem funcionalidades tem funcionalidades reduzidas, foi desenvolvido de forma modular para que possam ser adicionadas funcionalidades futuras a se poder e dever adicionar muitas mais funcionalidades. A título de exemplo, durante a execução do projeto, foram identificadas as seguintes novas funcionalidades: criar gráficos e estatística a partir dos

relatórios, o cliente poder comunicar diretamente através de mensagens com o funcionário que esta a fazer a auditoria.

5.3 Apreciação final

Este projeto foi uma oportunidade para consolidar e conhecer novas tecnologias, como também para conhecer as próprias capacidades de autonomia, visto ser a primeira vez que estamos sozinhos a fazer um trabalho sem um apoio mais direto dos professores, embora tendo um orientador que nos acompanhe e ajude. Destaca se também que ao conciliar este projeto de estágio com a vida profissional por vezes não permitiu uma boa gestão de tempo ou prioridades, realçando as dificuldades de um trabalhador-estudante.

Referências

- [1 SAP, “SAP,” [Online]. Available:] <https://www.sap.com/dam/application/shared/graphics/history-of-erp.svg>. [Acedido em 16 12 2022].
- [2 wikipedia, “Sistema integrado de gestão empresarial,” [Online]. Available:] https://pt.wikipedia.org/wiki/Sistema_integrado_de_gest%C3%A3o_empresarial. [Acedido em 16 12 2022].
- [3 “Infor,” 30 09 2022. [Online]. Available: <https://www.infor.com/news/infor-leader-2022-gartner-magic-quadrant-cloud-erp-product-centric-enterprises>. [Acedido em 20 01 2023].
- [4 “Mybusiness365/capterra,” [Online]. Available:] <https://www.capterra.pt/software/216420/mybusiness365>. [Acedido em 20 12 2022].
- [5 Capterra, “Capterra Pesquisa alargada,” [Online]. Available:] [https://www.capterra.pt/directory/9/enterprise-resource-planning/software?countries\[\]=PT&features\[\]=Order+Management&features\[\]=Reporting+Statistics&features\[\]=Reporting%2FAnalytics&features\[\]=Customer+Database](https://www.capterra.pt/directory/9/enterprise-resource-planning/software?countries[]=PT&features[]=Order+Management&features[]=Reporting+Statistics&features[]=Reporting%2FAnalytics&features[]=Customer+Database). [Acedido em 20 12 2022].
- [6 J. White, “Forbes,” 12 10 2022. [Online]. Available:] <https://www.forbes.com/advisor/business/software/bitrix24-review/>. [Acedido em 23 12 2022].
- [7 “Bitrix24,” [Online]. Available: <https://www.bitrix24.com/about/>. [Acedido em 26 12 2022].
- [8 R. W. Kathy Haan, “Forbes,” 16 12 2022. [Online]. Available:] <https://www.forbes.com/advisor/business/software/mondaycom-review/>. [Acedido em 25 12 2022].
- [9 J. D. & G. Sevilha, “pcmag,” 30 12 2022. [Online]. Available:] <https://www.pcmag.com/reviews/mondaycom>. [Acedido em 25 12 2022].
- [1 kasareviews, 18 01 2022. [Online]. Available: [https://www.kasareviews.com/netsuite-0\] review-pros-cons/#NetSuite_Proc_Cons](https://www.kasareviews.com/netsuite-0] review-pros-cons/#NetSuite_Proc_Cons). [Acedido em 21 12 2022].

- [1 Capterra, [Online]. Available: <https://www.capterra.pt/software/135757/netsuite>.
1] [Acedido em 21 12 2022].
- [1 “GRVPPE,” [Online]. Available: <https://www.grvppe.com/sistema-erp-netsuite/> . [Acedido
2] em 22 12 2022].
- [1 Better Buys, “betterbuys,” 01 12 2022. [Online]. Available:
3] <https://www.betterbuys.com/erp/reviews/odoo/>. [Acedido em 26 12 2022].
- [1 “Odoo,” [Online]. Available: https://www.odoo.com/es_ES/blog/nuestro-blog-5/the-4-odoo-story-56. [Acedido em 26 12 20022].
- [1 M. Richards, Architecture Patterns, 1005 Gravenstein Highway North, Sebastopol, CA:
5] O'Reilly Media, Inc., 2015.
- [1 “3PillarGlobal,” 20 09 2021. [Online]. Available:
6] <https://www.3pillarglobal.com/insights/what-is-event-driven-architecture/>. [Acedido em
28 12 2022].
- [1 R. C. Martin, “cleancoder,” 13 08 2012. [Online]. Available:
7] <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>.
[Acedido em 27 12 2022].
- [1 R. Ansari, “c-sharpcorner,” 09 02 2022. [Online]. Available: <https://www.c-sharpcorner.com/article/what-is-clean-architecture/>. [Acedido em 26 12 2022].
- [1 “codeguru,” [Online]. Available: <https://www.codeguru.com/csharp/understanding-9-onion-architecture/>. [Acedido em 18 6 2022].
- [2 C. Hunsaker, “stormpath,” 05 12 2016. [Online]. Available:
0] <https://stormpath.com/blog/rest-vs-soap>. [Acedido em 27 12 2022].
- [2 “Google.trend,” [Online]. Available:
1] <https://trends.google.com/trends/explore?date=all&q=REST%20API,SOAP%20API>.
[Acedido em 20 12 2022].
- [2 A. Monus, “Raygun,” 17 10 2022. [Online]. Available: <https://raygun.com/blog/soap-vs-2-rest-vs-json/>. [Acedido em 26 12 2022].

- [2 J. Clark, “back4app,” [Online]. Available: [https://blog.back4app.com/pt/melhores-3\] frameworks-para-desenvolver-backend/.](https://blog.back4app.com/pt/melhores-3] frameworks-para-desenvolver-backend/>.) [Acedido em 27 12 2022].
- [2 “Microsoft,” [Online]. Available: [https://dotnet.microsoft.com/en-us/platform/why-4\] choose-dotnet.](https://dotnet.microsoft.com/en-us/platform/why-4] choose-dotnet.) [Acedido em 28 12 2022].
- [2 “Google Trends FRame,” [Online]. Available: <https://trends.google.com/trends/explore?date=all&q=ANGULAR,REACT,VUE,EMBER.> [Acedido em 26 12 2022].
- [2 J. Osman, “appMaster,” 26 09 2022. [Online]. Available: <https://appmaster.io/blog/popular-frontend-frameworks.> [Acedido em 26 12 2022].
- [2 P. M. P. Petraq J. Papajorgji, Software Engineering Techniques Applied to Agricultural Systems., Boston, MA: Springer, 2006.
- [2 R. Pressman, Engenharia de software (6.^a Edição), McGraw-Hill., 2009.
- 8]
- [2 P. Eeles, “What, no supplementary specification,” 1 julho 2004. [Online]. Available: <https://www.ibm.com/developerworks/rational/library/3975.html.>
- [3 “Navigatos,” 02 03 2022. [Online]. Available: [https://blog.nbs-us.com/the-pros-and-cons-0\] of-sap-erp-systems.](https://blog.nbs-us.com/the-pros-and-cons-0] of-sap-erp-systems.) [Acedido em 01 08 2022].
- [3 Capterra, “Pesquisa ERP Portugal,” [Online]. Available: [https://www.capterra.pt/directory/9/enterprise-resource-planning/software?countries\[\]&vendor_hq=pt.](https://www.capterra.pt/directory/9/enterprise-resource-planning/software?countries[]&vendor_hq=pt.) [Acedido em 19 12 2022].
- [3 Carnegie Mellon University, “Carnegie Mellon University,” 22 01 2017. [Online]. Available: https://resources.sei.cmu.edu/asset_files/FactSheet/2010_010_001_513810.pdf. [Acedido em 26 12 2022].
- [3 T. Pal, “codeguru,” 12 02 2018. [Online]. Available: [https://www.codeguru.com/csharp/understanding-onion-architecture/.](https://www.codeguru.com/csharp/understanding-onion-architecture/>.) [Acedido em 27 12 2022].
- [3 “Google trends a R,” [Online]. Available: <https://trends.google.com/trends/explore?date=all&q=ANGULAR,REACT.> [Acedido em 27 12 2022].

[3 “Stateofjs,” [Online]. Available: <https://stateofjs.com/en-us/>. [Acedido em 26 12 2022].

5]

Anexo A Modelo de Domínio

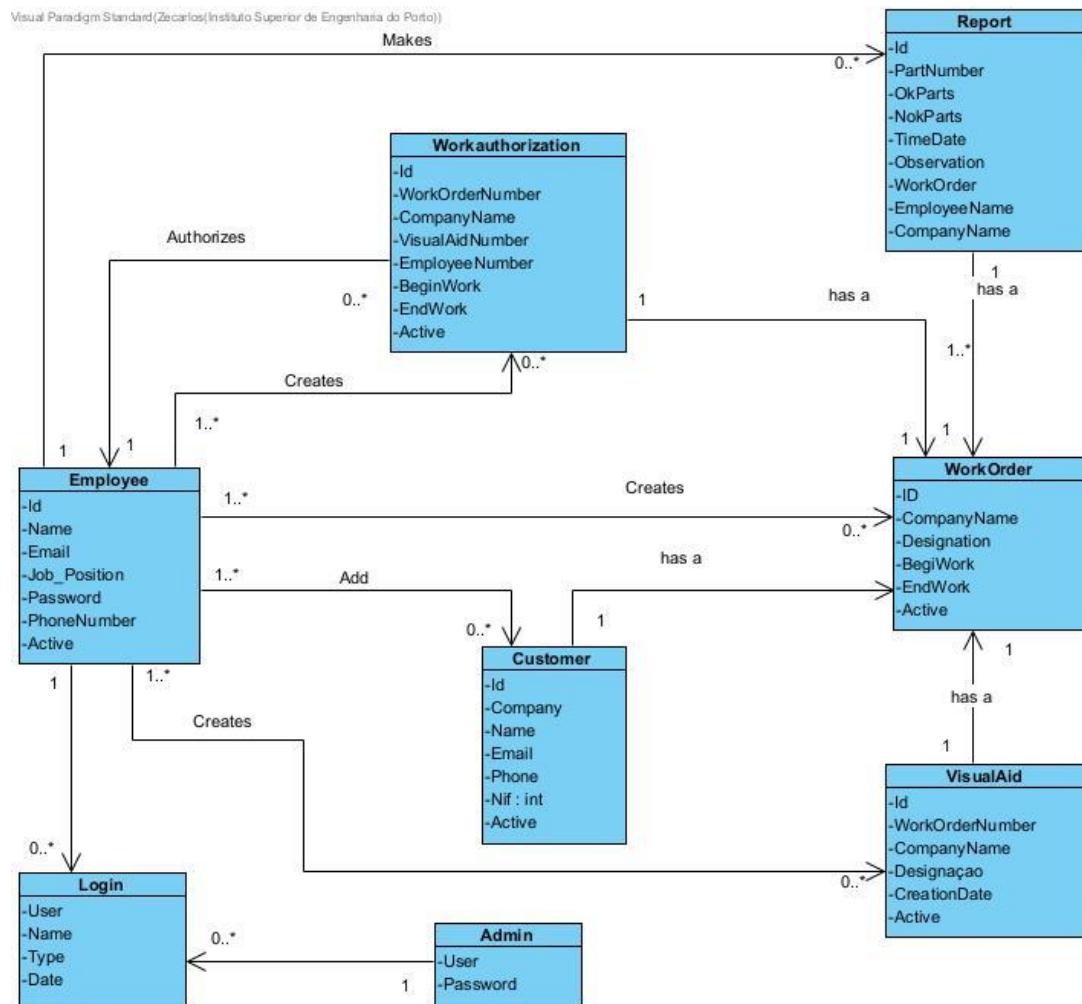


Figura 47 - Modelo de Domínio