INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

# Estágio na empresa Inetum na área de OutSystems

**Rui Ferreira Salgado**

*Relatório Final de Estágio Profissional apresentado à Escola Superior de Tecnologia e Gestão para obtenção do Grau de Mestre em Informática*

Trabalho realizado sob a orientação de:

Professor Paulo Alexandre Vara Alves

António Henrique Dias de Moura Capela

**Bragança**

Outubro 2022

Declara-se que, devido a uma cláusula de confidencialidade, não é possível apresentar dados ou qualquer material que possa identificar os clientes.

# Resumo

Com a evolução das tecnologias, houve uma maior necessidade de simplificar a criação de aplicações e soluções. Surgem então, plataformas *Low Code* como OutSystems, que têm como finalidade descomplicar este tipo de procedimentos.

O presente relatório de estágio tem como objetivo referir as aprendizagens decorridas ao longo do estágio na empresa INETUM, utilizando a plataforma OutSystems.

O estágio iniciou-se com a realização de um primeiro projeto interno "Office Ocupancy" com o objetivo de aprimorar as capacidades em OutSystems e compreender a metodologia de trabalho da empresa. O projeto seguinte destinou-se para uma empresa do setor alimentar com o objetivo de criar uma aplicação para gerir investimentos, os objetivos deste foram cumpridos todos com sucesso, apenas foi necessário alterar a data de entrega e prolongar por mais uma semana. O último projeto dedicou-se para um cliente de uma empresa de advocacia e tinha como objetivo a construção de Webservices para substituir integrações antigas de SAP para Primavera e, posteriormente manutenção da fábrica em questão. Neste último projeto foi possível concluir todos os objetivos e testar as implementações com outras equipas integrantes no projeto com sucesso.

Ao longo do período de estágio foram adquiridos bastantes conhecimentos tanto em OutSystems como na forma de comunicação com clientes.

**Palavras-chave**: INETUM, OutSystems, *Low Code*

# Abstract

With the evolution of technologies, there was a greater need to simplify the creation of applications and solutions. Low Code platforms such as OutSystems appear, which aim to simplify this type of procedure.

This internship report aims to refer to the lessons learned during the internship at the company INETUM, using the OutSystems platform.

The internship began with the completion of a first internal project “Office Occupation” with the objective of improving OutSystems capabilities and understanding the company's work methodology. The next project was destined for a company in the food sector with the objective of creating an application to manage investments, the objectives of this were all successfully met, it was only necessary to change the delivery date and extend it for another week. The last project was dedicated to a client of a law firm and aimed to build Webservices to replace old SAP integrations for Primavera and, later, maintenance of the factory in question. In this last project it was possible to complete all the objectives and successfully test the implementations with other teams that were part of the project.

During the internship period, a lot of knowledge was acquired both in OutSystems and in the way of communicating with clients.

**Keywords**: INETUM, OutSystems, Low Code

# Índice Geral

# Índice de Figuras

# Índice de Tabelas

# Capítulo 1 Introdução

Nos dias que correm a evolução das tecnologias tem sido notória, cada vez mais as empresas lutam pela evolução dos seus serviços tecnológicos disponibilizando as melhores e mais atuais soluções para os seus clientes. O papel de esta evolução tecnológica é tornar mais simples a resolução de questões essenciais para o nosso cotidiano.

Devido a esta necessidade de evoluir e oferecer melhores opções para os seus clientes muitas empresas optaram pela utilização de plataformas *Low Code* que proporcionam uma maior facilidade no momento de desenvolvimento de soluções, oferecendo rapidez e flexibilidade.

## 1.1. Enquadramento

No âmbito da conclusão de mestrado em informática lecionado na Escola Superior de Tecnologia e de Gestão (ESTiG) do Instituto Politécnico de Bragança (IPB), realizou-se o presente relatório de estágio, sendo este executado na empresa INETUM com recurso a plataforma OutSystems.

## 1.2. Objetivos

O presente relatório final de estágio tem como objetivo apresentar todo o percurso e trabalho efetuado, ao longo do período de estágio na empresa INETUM, assim como dar a conhecer um pouco das ferramentas utilizadas e tecnologias que envolveram todo o desenvolvimento de processos de negócio em ambiente empresarial.

Para além do percurso do aluno dentro da empresa, este relatório irá incidir em específico no processo de desenvolvimento de aplicações e outras tarefas levadas a cabo ao longo do período de estágio profissional.

## 1.3. Estrutura do documento

Este relatório final de estágio profissional encontra-se dividido em 4 capítulos, estando estes ainda divididos em várias secções, de modo a estruturar da melhor forma toda a informação nele contida.

- **Capítulo 1:** Introdução

É neste capítulo que é feita uma abordagem de caráter introdutória a todo o relatório.

Desta forma, pretende-se deixar claro os objetivos e o enquadramento, assim como a estrutura do documento.

- **Capítulo 2**: Tecnologias e metodologias de desenvolvimento

Neste capítulo são demonstradas as tecnologias e metodologias que foram utilizadas ao longo do estágio.

- **Capítulo 3**: Desenvolvimento de estágio

No capítulo 3 começa por uma breve descrição da empresa onde foi realizado o estágio, seguida de uma explicação mais detalhada da ferramenta OutSystems utilizada ao longo do estágio e finalmente um último capítulo onde são esclarecidos os projetos nos quais o aluno esteve presente.

- **Capítulo 4**: Conclusões

Neste capítulo final é feita uma breve análise ao período do estágio, seguida dos objetivos seguintes dentro da empresa.

# Capítulo 2 Tecnologias e metodologias de desenvolvimento

O uso das tecnologias, nos dias de hoje, é inevitável. As pessoas recorrem a estas para tarefas simples do dia a dia por exemplo, para estarem a par do que ocorre no mundo, para comunicarem e principalmente as empresas, para trabalhar.

Posto isto, o presente capítulo irá abordar as tecnologias *low code*, as metodologias de trabalho, e as linguagens de programação utilizadas ao longo do estágio.

## *2.1.* Desenvolvimento *low code*

Atualmente as empresas têm encontrado vários problemas devido ao aumento da complexidade das suas operações internas. Com isto tem se verificado que estas necessitam de respostas rápidas e mais flexíveis para os problemas que têm ocorrido e assim poderem completar os seus requisitos.

Para cumprir estes requisitos é necessário encontrar uma forma de desenvolver produtos de uma forma mais rápida e que sejam mais fáceis de manter a longo prazo. Aqui é onde entra a expressão *low code*, esta refere-se a plataformas que estão previamente configuradas para exigir menos código escrito a mão e facilitar o desenvolvimento de aplicações em menos tempo. Estas plataformas disponibilizam interfaces gráficas e desenvolvimento *drag-and-drop* que facilitam o trabalho do programador, preocupando-se apenas com lógicas um pouco mais complexas.

Estas plataformas possibilitam também às empresas alcançar o objetivo de forma mais rápida, sem grande complexidade e de forma mais ágil. Com o decorrer do tempo as plataformas oferecem também um menor custo económico de implementar novas soluções ou requisitos internos [1].

Este cenário de low-code é bastante complexo e dispõe de variadas soluções. Na figura 1 é mostrado o Quadrante Mágico da Gartner para plataformas corporativas de low-code:

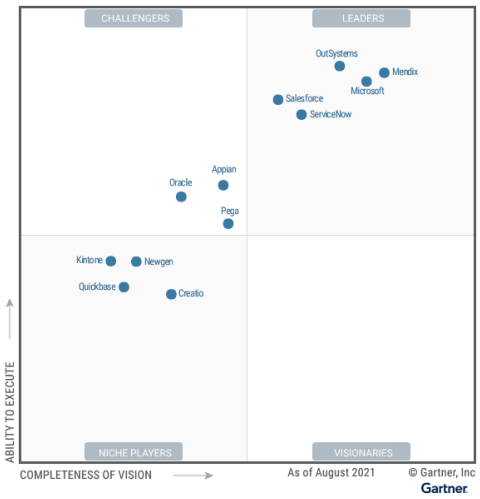


Figura 1 - Quadrante Mágico para Plataformas de Aplicativos Empresariais Low-Code (reproduzido de [15])

Como é possível ver na figura anterior algumas das plataformas low-code líderes de mercado com maior destaque são a OutSystems e o Mendix.

De seguida serão apresentadas algumas informações sobre estas plataformas.

### 2.1.1. OutSystems

OutSystems é uma plataforma *low code* (atualmente na versão 11), utilizada para construir aplicações web e mobile, permitindo que os programadores desenvolvam soluções mais rapidamente sem terem que lidar com a complexidade inerentes às linguagens de programação que finalmente implementam a solução.

Esta plataforma *low code* ajuda os programadores a criar aplicações de alta qualidade com rapidez e eficiência. Um ambiente de desenvolvimento visual orientado por modelos e assistido por IA garante que as aplicações sejam construídas em dias ou semanas em vez de meses ou anos. Os serviços da plataforma, também com IA, oferecem automação para melhorar todo o ciclo de vida das aplicações, de modo que estas possam ser implementadas com apenas um clique e geridas com uma facilidade incomparável [7].

### 2.1.2. Mendix

A Plataforma Mendix foi projetada para acelerar a entrega de aplicações corporativas em todo o ciclo de vida de desenvolvimento de aplicações, desde a conceção até o desenvolvimento, implantação e a gestão contínua do seu portfólio de aplicações na

nuvem ou no local. A Mendix oferece duas opções de programação sendo estas, *no code* (modulação visual) e *low code* (ferramentas altamente extensíveis e integradas para dar suporte a equipas multifuncionais que trabalham de forma colaborativa). Especialistas no domínio de negócios, como analistas, podem trabalhar ao lado de programadores especialistas para alcançar níveis muito elevados de agilidade nos negócios e, consequentemente, entrega acelerada, enquanto a arquitetura nativa da nuvem e as ferramentas de automação da plataforma suportam a implantação, gestão e monitorização de recursos altamente disponíveis.

## 2.2. Metodologias

Um dos pontos chave da execução de um projeto é a metodologia de gestão utilizada. Saber escolher a que melhor se encaixa, pode reduzir falhas de comunicação, diminuição de custos, atrasos e contribuir para a solução como um todo.

No desenvolvimento de software são usadas duas diferentes abordagens, a tradicional, metodologia Cascata, onde o projeto é tratado linearmente com vários eventos, e a abordagem mais moderna, conhecida como *agile*, que envolve um desenvolvimento variável, iterativo e com foco na equipa.

### 2.2.1. Metodologia *Agile*

A metodologia *agile* originou-se na indústria de desenvolvimento de software como uma nova maneira de gerir o desenvolvimento de software. Muitos projetos de desenvolvimento que falhavam ou demoravam muito tempo a serem concluídos, e os líderes do setor perceberam que era necessário encontrar uma abordagem nova e inovadora.

Em 2001, o “Manifesto para Agile Software Development” foi criado e assinado por representantes de “Extreme Programming”, “Scrum”, “DSDM”, “Adaptive Software Development”, “Crystal. Este grupo reuniu-se para encontrar uma alternativa aos métodos tradicionais de gestão de projetos para desenvolvimento de software.

Esta "Agile Alliance" foi o início das metodologias *agile* de hoje. No início, o *agile* era principalmente para gerir projetos de desenvolvimento de software. No entanto, este evoluiu e começou a lidar com projetos de todos os setores, organizações e mercados.

As metodologias ágeis defendem que, acima de tudo, é necessário procurar a satisfação do cliente por meio de entregas contínuas de software de valor agregado, mantendo a comunicação constante com o cliente e, também, focando na comunicação entre os membros da equipa. Ao contrário das práticas anteriores, a metodologia *agile* não se caracteriza pela definição completa de um produto – uma análise completa ou a definição de todas as categorias/requisitos – mas por uma interação dinâmica que permite entrega constante [2].


Figura 2 - Metodologia Agile (reproduzido de [21])

Basicamente, o desenvolvimento ágil segue um modelo incremental, que desenvolve a colaboração dentro da equipa e o planeamento contínuo, além de evolução e aprendizagens permanentes. As metodologias ágeis devem respeitar o ciclo de desenvolvimento de software – planeamento, execução e entrega final – permitindo que o software seja desenvolvido em etapas e tornando mais fácil identificar e resolver problemas.

A principal vantagem do uso de metodologias ágeis não é apenas a entrega rápida do software, mas também a entrega constante de valor ao cliente, pois as entregas são incrementais [2].

#### *2.2.1.1. Scrum*

Scrum é a framework mais utilizada das metodologias ágeis. Esta framework divide o projeto em vários pedaços com os quais é feita uma lista com estes por ordem de

prioridade. Após serem definidas as prioridades, cada tarefa da lista é desenvolvida uma a uma em um tempo específico, isto é chamado de *sprint*.

No final de cada *sprint*, o que foi feito anteriormente é revisto e se estiver correto o atual sprint é terminado e começa um novo. Normalmente, a equipa tem um *Scrum master* sendo este o responsável do projeto e que ajuda a mesma a alcançar o resultado final [4].

Todos os dias existem pequenas reuniões de 15 minutos, o *scrum* diário, que tem o papel de sincronizar as atividades e encontrar a melhor forma de planear o trabalho seguinte.

Tabela 1 - **Vantagens e Desvantagens do scrum**

| **Vantagens** | **Desvantagens** |
|---|---|
| Há muita motivação nas equipas, porque os programadores querem cumprir o prazo de cada sprint. | A segmentação do projeto e o foco na agilidade no desenvolvimento podem, às vezes, levar a equipa a perder o controlo do projeto como um todo, focando-se apenas numa parte. |
| A transparência permite que o projeto seja acompanhado por todos os membros de uma equipa ou mesmo de uma organização. | A função de cada programador pode não estar bem definida, resultando em alguma confusão entre os membros da equipa. |
| O foco na qualidade é uma constante no método scrum, resultando em menos erros. | |
| A dinâmica desse método permite que os programadores reorganizem as prioridades, garantindo que os sprints que ainda não foram concluídos recebam mais atenção. | |

### 2.2.2. Metodologia Cascata

A metodologia em cascata foi inventada pelo Dr. Winston W. Royce em 1970, sendo assim a metodologia mais antiga na área de desenvolvimento. Esta usa uma abordagem sequencial ou linear para o desenvolvimento de software. O projeto é dividido em uma sequência de tarefas, com o agrupamento de nível mais alto referido como fases. Uma verdadeira abordagem em cascata requer fases que sejam concluídas em sequência e tenham critérios de saída formais, normalmente uma aprovação das partes interessadas do projeto [5].

Na metodologia cascata antes de o programador passar para os próximos passos, é necessário rever os passos completados previamente e confirmar que foram bem executados. Normalmente o método de cascata é utilizado em empresas nas quais mudanças são caras ou demoradas, então este método propõem uma estrutura mais robusta e mais eficiente.

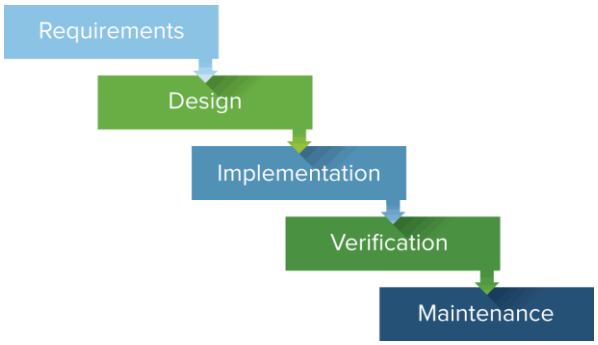


Figura 3 - Metodologia Cascata (reproduzido de [6])

De tabela seguinte são demostradas as vantagens e desvantagens de esta metodologia.

Tabela 2 - **Vantagens e desvantagens da metodologia Cascata**

| Vantagens | Desvantagens |
|---|---|
| Os requisitos são concluídos no início do projeto, permitindo que a equipa defina todo o âmbito do projeto, crie um cronograma completo e projete a aplicação geral. | Muitas vezes, é difícil, obter os requisitos de negócios completos antecipadamente num projeto, pois por vezes existem detalhes que não foram considerados previamente o que origina uma alteração dos requisitos de negócios. |

| | |
|---|---|
| Melhora a utilização de recursos porque as tarefas podem ser divididas para serem trabalhadas em paralelo ou agrupadas para alavancar as capacidades dos recursos. | Requer um grande detalhe das tarefas e entregas, que pode estar além da capacidade ou experiência da equipa do projeto no início do projeto. |
| Design de aplicações mais completas pois existe uma compreensão mais completa de todos os requisitos e entregas. | Embora os projetos em cascata não tenham inerentemente que abranger longos períodos de tempo, a probabilidade de estes se atrasarem, ultrapassarem o orçamento e não irem de encontro às expectativas aumenta à medida que o prazo do projeto de TI aumenta. |
| O status do projeto é mais facilmente medido com base num cronograma completo e plano de recursos. | |

## 2.3. Linguagens de programação

### 2.3.1. SQL

SQL é uma linguagem de programação usada para interrogar e gerir bases de dados. Esta foi desenvolvida em 1970 por investigadores da IBM e é extremamente acessível em várias plataformas e geralmente fácil de usar.

Empresas e outras organizações usam programas SQL para:

- Modificar tabelas de bases de dados e estruturas de índice;
- Adicionar, atualizar e excluir linhas de dados;
- Devolver subconjuntos de informações de dentro de bases de dados relacionais.

### 2.3.2. CSS

CSS significa “Cascading Style Sheets”. É a linguagem para descrever a apresentação de páginas da Web, incluindo cores, *layout* e fontes, tornando as páginas Web apresentáveis aos utilizadores.

É independente de HTML e pode ser usado com qualquer linguagem de marcação baseada em XML [20].

De seguida são demonstradas as vantagens e desvantagens da utilização de CSS:

Tabela 3 - **Vantagens e desvantagens de CSS (Fonte [24])**

| **Vantagens** | **Desvantagens** |
|---|---|
| Economizar tempo na criação e edição de Websites. | Por vezes não existe compatibilidade de entre navegadores. |
| Carregamento de Websites mais rápido. | |
| Maior eficiência a gerir *layout* da página. | |

### 2.3.3. C#

É uma linguagem de programação orientada a objetos desenvolvida pela Microsoft e que funciona sobre a Framework .Net. O fundador de esta linguagem de programação c# foi Anders Hejlsberg.

Esta linguagem é baseada em C++ e Java, mas possui muitas extensões adicionais usadas para executar a abordagem de programação orientada a componentes. C# evoluiu muito desde seu primeiro lançamento no ano de 2002. Este foi introduzido com o .NET Framework 1.0 e a versão atual do C# é 5.0 [16].

C# é uma linguagem de programação de uso geral usada para construir diferentes tipos de programas e aplicações. Embora seja excecionalmente versátil, existem três campos em que é comumente aplicado [17]:

- **Desenvolvimento de Aplicações Web** - Independentemente da plataforma, pode-se usar a linguagem de programação C# para criar sites dinâmicos e aplicações Web usando a plataforma .NET.
- **Aplicações Windows** - A Microsoft criou o C# essencialmente para o desenvolvimento de aplicações Windows. Isso torna o processo de desenvolvimento mais focado nas funcionalidades, não sendo necessário o programador ter preocupações com a gestão da memoria, uma vez que apresenta bom desempenho excelente. Além disso, os programadores podem contar com o suporte e a documentação da comunidade para o desenvolvimento de aplicações e programas específicos da arquitetura da plataforma Microsoft.
- **Jogos** - No mundo dos jogos, os programadores preferem a linguagem de programação C#. é uma linguagem particularmente robusta para a criação de jogos, como por exemplo no motor de jogos Unity (um dos mais populares atualmente).

### 2.3.4. JavaScript

JavaScript é uma linguagem de programação baseada em texto utilizada tanto do lado do cliente como do lado do servidor que permite criar páginas da web interativas. Enquanto HTML e CSS são linguagens que dão estrutura e estilo às páginas da web, JavaScript fornece às páginas web elementos interativos que envolvem o utilizador. A incorporação de JavaScript melhora a experiência do utilizador na página Web, passando esta de uma página estática para uma interativa. De forma mais simples, JavaScript adiciona comportamento às páginas da web [22].

Algumas das aplicações do JavaScript incluem:

- Adição de comportamento interativo a páginas web;
- Criação de aplicações web e móveis;
- Construção de servidores web e desenvolvendo de aplicações de servidor;
- Desenvolvimento de jogos;

## 2.4. Tecnologias

### 2.4.1. Teams

O Microsoft Teams é uma aplicação de mensagens utilizado em organizações, um espaço de trabalho para colaboração e comunicação em tempo real, reuniões, compartilhamento de ficheiros e aplicações.

As principais funcionalidades do Teams são as seguintes [23]:

- Mensagens Instantâneas - o software conta com uma aplicação de chat que mantém todos os membros em contato a todo o momento.;
- Chamadas de áudio e vídeos - para além de mensagens de texto, o software permite que os membros de uma equipa realizem chamadas de áudio e vídeo;
- Integração com outras ferramentas - permite integração com outros softwares como por exemplo Office 365, softwares de programação, agendas, redes sociais entre outros;
- Armazenamento de ficheiros - facilita o armazenamento e a organização de ficheiros de acordo com cada projeto;

### 2.4.2. Jira

O Software Jira faz parte de uma família de produtos desenvolvidos para ajudar equipas de todos os tipos a gerir o trabalho. Originalmente, o Jira foi projetado como um rastreador de bugs e problemas. Mas hoje, este software evoluiu para uma poderosa ferramenta de gestão de trabalho para todos os tipos de casos de uso, desde requisitos e gestão de casos de teste até desenvolvimento ágil de software [19].

Tabela 4 - **Vantagens e Desvantagens do Jira (Fonte [18])**

| **Vantagens** | **Desvantagens** |
|---|---|
| É adequado para diferentes tipos de utilizadores, como programadores, | Upload de arquivos com tamanho limitado. |

| | |
|---|---|
| gestores de projeto, engenheiros, profissionais não técnicos entre outros. | |
| Permite que os utilizadores criem qualquer tipo de problema. | Os relatórios não são reutilizáveis. |
| | Interface confusa. |

# Capítulo 3 Desenvolvimento do estágio

Neste capítulo serão apresentadas informações relativas a empresa na qual o estágio foi realizado, seguida de uma breve explicação da plataforma OutSystems, um esclarecimento de como foi elaborado o plano de estágio e finalmente são detalhados os projetos nos quais o aluno participou.

## 3.1. Inetum

A Inetum é uma empresa de TI que oferece serviços e soluções digitais. Esta trata-se de um grupo global que ajuda empresas e instituições a tirar o máximo proveito do fluxo digital. Operando em mais de 26 países, o grupo tem cerca de 27.000 funcionários e em 2020 gerou uma receita de 1,966 mil milhões de euros.

### 3.1.1. Inetum Portugal

Nos últimos anos, a Inetum cresceu muito rápido em Portugal, e hoje é uma das 5 maiores empresas de TI. A empresa está organizada em três diferentes áreas de negócio:

- **Serviços de TI e projetos de transformação** - Para apoiar os clientes a obter o máximo do seu fluxo digital, a Inetum possui ampla experiência em consultoria de TI, Serviços de Aplicação, Serviços de Infraestrutura e software.
- **SAP e OutSystems** – A Inetum é referencia em consultoria de soluções SAP nos mercados.
- **Software de seguro** - A Inetum possui uma ampla gama de soluções verticais para o setor segurador, com software de última geração dedicado a seguros de vida e não vida (CLEVA) que apoia grandes seguradoras e gestores de fundos de pensão na adaptação aos novos desafios digitais, incluindo otimização do relacionamento com o cliente, digitalização ou regulamentação de seguros de vida e não vida.

## 3.2. OutSystems

### 3.3.1 Arquitetura

A plataforma OutSystems obedece a uma arquitetura constituída pelos componentes:

- ***Plataform Server***

É o núcleo da plataforma OutSystems. Este cuida de todas as etapas necessárias para gerar, compilar, empacotar e publicar aplicações [8].

- ***LifeTime***

É uma consola centralizada para gerir os ambientes OutSystems, aplicações, utilizadores de TI e segurança, cobrindo todo o ciclo de vida da aplicação, desde o desenvolvimento até à implementação.

Uma infraestrutura OutSystems típica compreende os seguintes ambientes:

- Desenvolvimento: ambiente onde aplicações são inicialmente desenvolvidas e testadas;
- Qualidade: ambiente em que a equipa e utilizadores de testes experimentam as aplicações para realizar a garantia de qualidade;
- Produção: ambiente que hospeda a versão do aplicativo com os quais os utilizadores finais interagem;

Para o utilizador gerir a infraestrutura, este pode aceder a mesma através do link (https://<your_lifetime_server>/lifetime) [14].

- ***Integration Studio***

É uma ferramenta de desktop que permite criar e gerir as suas extensões. Este é instalado, juntamente com o *Service Studio*, como parte do ambiente de desenvolvimento, que está disponível na página de downloads do Ambiente de Desenvolvimento [13].

- ***Service Studio***

É o ambiente de desenvolvimento visual e low-code da OutSystems que permite criar aplicações e módulos no servidor, interfaces de utilizador para Web Tradicional, Web

*Reactive* e aplicações Móveis. Permite definir o modelo de dados, definir processos de negócios, temporizadores e fazer *debug* das aplicações [12].

Neste ambiente os programadores têm acesso a módulos chamados de Espaces.

O ambiente gráfico do Service Studio é constituído por separadores, sendo estes:

**Separadores de lógica** (fig.4):

Neste separador é onde a lógica da aplicação é criada dentro de ações. Estas ações podem ser de vários tipos sendo a mais importante as ações de servidor (fig. 4).

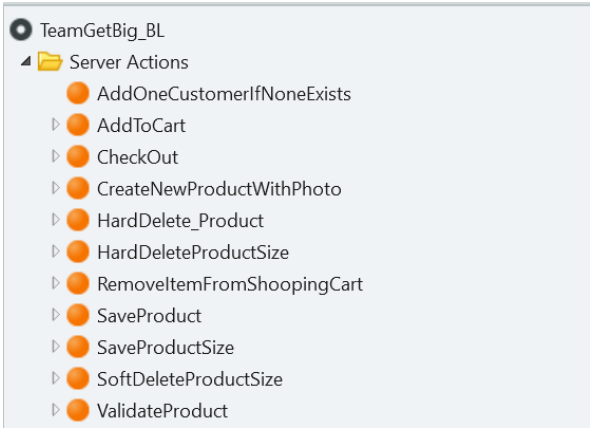


Figura 4 - Separador de lógica

Dentro de uma ação do servidor é possível utilizar várias ferramentas para a construção da lógica. Estas ações devem sempre começar com as ferramentas de *Start* e acabar com a ferramenta *End*.

Algumas das ferramentas mais utilizadas na construção de lógica são demostradas na figura 5. Como é possível ver nesta figura existe uma grande variedade de ferramentas que podem ser utilizadas para construir a lógica de uma ação, podendo chamar outras ações de servidor dentro das mesmas, fazer chamamentos à bases de dados usando agregados ou SQL, manipular listas utilizando outras ações de servidor do sistema entre outros [11].

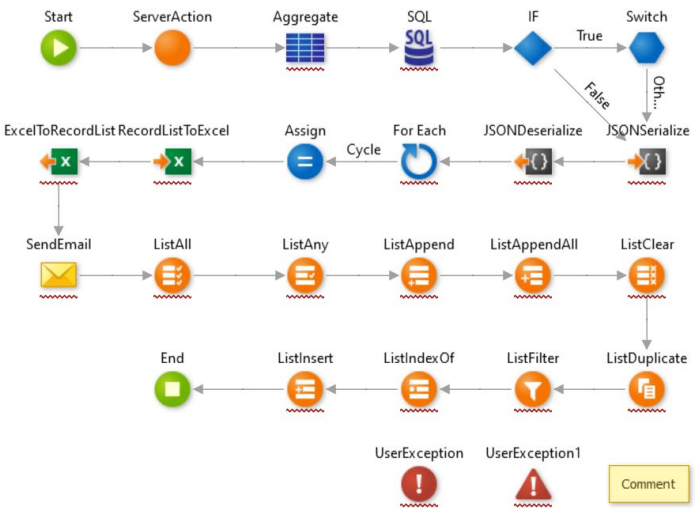


Figura 5 - Ferramentas de lógica

Na figura 6 é demonstrada uma ação utilizando algumas das ferramentas de lógica explicadas anteriormente.

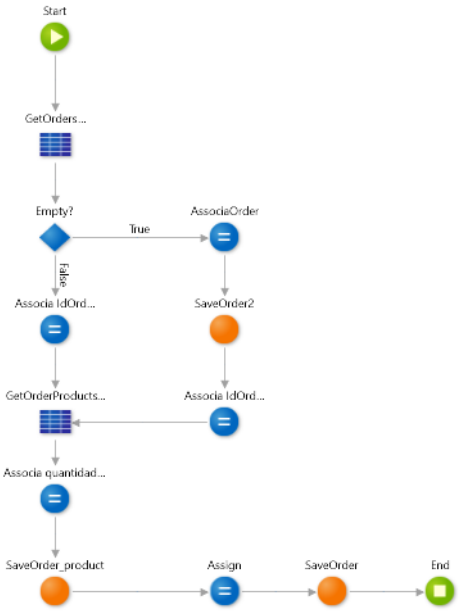


Figura 6 – Exemplo de uma ação de servidor

**Separador de interface** (fig. 7)**:**

Neste separador é onde os ecrãs são construídos de uma forma *drag-and-drop*. Para isto é possível criar fluxos de interface que agrupam ecrãs em unidades lógicas com configurações comuns.

Dentro de estes fluxos é possível criar:

- ***Web Screens*** (ecrãs) - Interface do utilizador com a qual os utilizadores finais interagem.

- ***WebBlock*** - Parte de ecrã reutilizável que pode implementar sua própria lógica (utilizada em ecrãs).

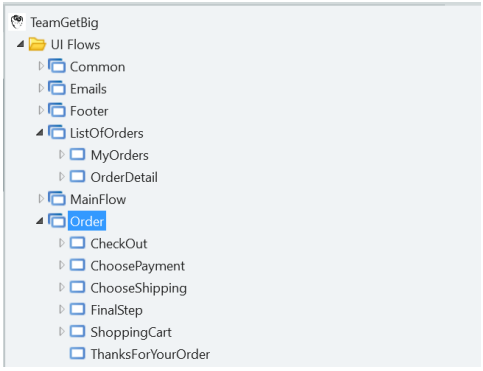


Figura 7 - Separador de interface

Ambas as opções dispõem de um *preparation*, este *preparation* trata-se de uma ação do lado do servidor que carrega dados iniciais para os ecrãs, não existe na interface *Reactive* do lado do cliente.

A ação *preparation* não necessita de ser invocada, pois esta é executada automaticamente antes de um ecrã ser renderizado. É o local adequado para adicionar toda a lógica desejada antes de executar o ecrã, como definir valores para variáveis e obter dados da base de dados. Também é possível adicionar uma ação de *preparation* a *WebBlocks*.

Ainda dentro de um ecrã ou *WebBlock* é possível ter acesso a variáveis locais e variáveis de *input*. Também é possível criar ações de ecrã que permitem executar lógica quando o utilizador interage com o ecrã, como por exemplo quando o mesmo carrega em um botão ou link. Estas ações de ecrã também estão disponíveis em *WebBlocks* e nelas é possível executar ações de servidor ou criar lógica utilizando as ferramentas de lógica referidas anteriormente na figura 5.

Todas estas funcionalidades estão demostradas na figura 8:

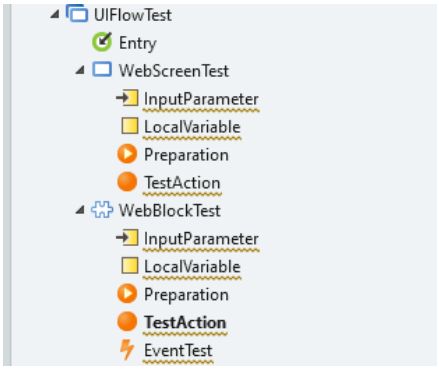


Figura 8 - *Web Screen* e *WebBlock*

Os *WebBloks* dispõem de uma outra opção chamada de evento, este tem como utilidade comunicar com o seu pai (componente que contém os *WebBlocks*), podendo este ser um outro *WebBlock* ou um ecrã. Neste evento é possível enviar variáveis para o pai poder utilizar. Quando um evento é criado é necessário associar uma ação ao *WebBlock* da parte do pai que vai executar a lógica quando o mesmo é notificado.

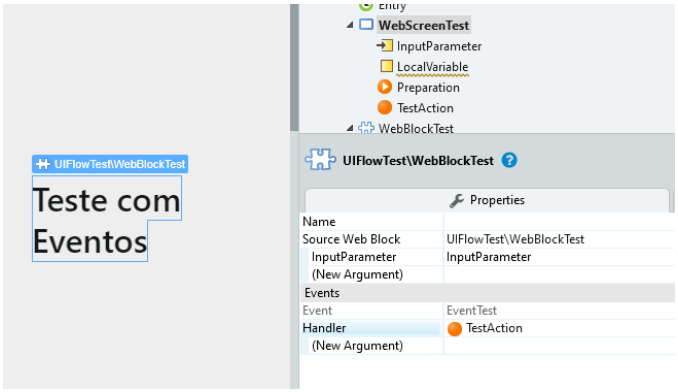


Figura 9 - Eventos

Após ter a lógica toda definida é possível fazer a construção do ecrã ou *WebBlock*. Dentro de um ecrã o programador pode optar por restringir o mesmo a certas entidades utilizando *roles* de utilizador. Estas roles têm como função restringir ou permitir que utilizadores finais possam aceder a ecrãs e operações específicas de uma aplicação [10].

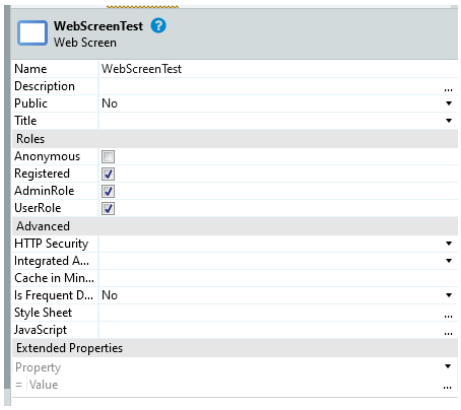


Figura 10 - Opções de roles de ecrãs

Após restringir o ecrã é possível fazer a criação da interface, para tal os componentes *Service Studio* dispõem de uma variada lista de opções que o programador pode utilizar para a criação do seu ecrã. Desde tabelas, *forms*, botões, links, imagens, *inputs*, *containers* entre muitos outros [9] como é possível ver na figura 11:

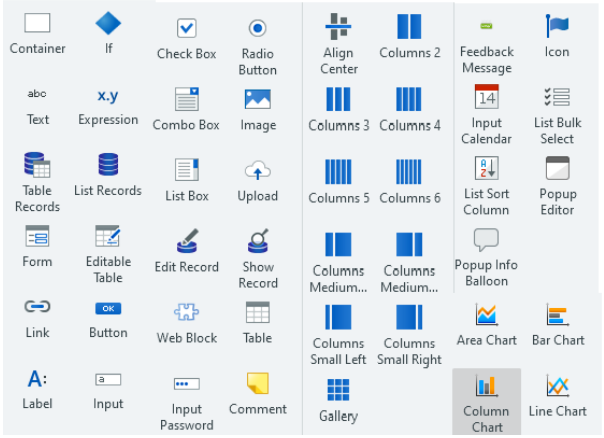


Figura 11 - Componentes gráficos

**Separador de dados** (fig. 12)**:**

Dentro de este separador o programador pode definir as suas bases de dados, criar estruturas úteis para utilizar em todo o *espace*, criar variáveis de sessão que são independentes dos ecrãs, e *site properties* que são variáveis com valores predefinidos na plataforma que podem ser utilizados em lógica ou mesmo dentro dos ecrãs.

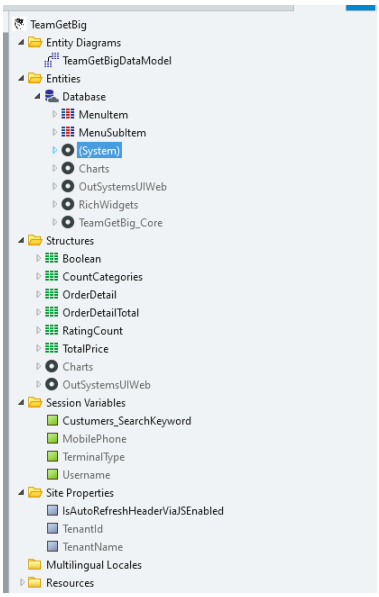


Figura 12 - Separador de dados

O programador pode aceder à base de dados utilizado a ferramenta Agregado ou *SQL*.

A ferramenta Agregado funciona utilizando uma consulta otimizada. Os Agregados podem carregar dados do servidor da base de dados local e suportam a combinação de várias entidades e filtragem avançada.

Quanto à ferramenta SQL trata-se de uma ferramenta em que o programador pode construir a sua própria *query* como ele desejar.

Tanto os Agregados como o SQl podem ser definidos do lado do cliente no caso de aplicações *reactive*, *server-side*, ou seja, dentro de ações de servidor e também dentro de *preparation* no caso de aplicações tradicionais.

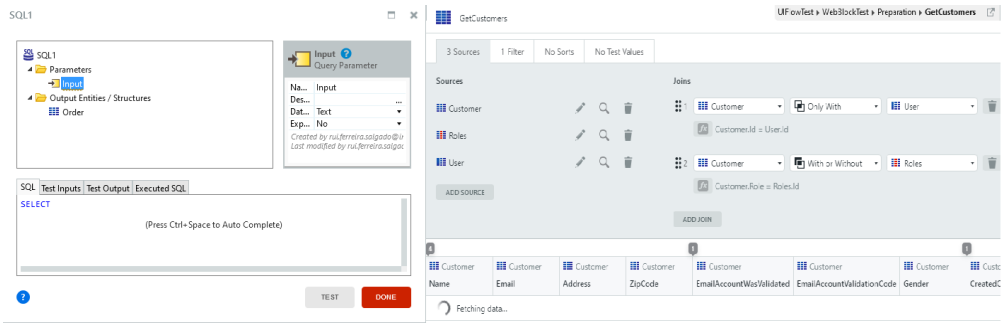


Figura 13 - Ferramenta SQL e agregado

## 3.3. Plano de estágio

O planeamento do estágio passa por executar os seguintes pontos:

- Participar em Scrums diários para falar sobre o trabalho realizado no dia anterior, o que é necessário fazer para o mesmo dia e esclarecer dúvidas;
- Utilização de ferramentas para a elaboração de requisitos e *user stories*;
- Interpretar *user stories* e realizar as tarefas técnicas associadas a estes;
- Realizar a estrutura de dados, construir o diagrama de entidades e realizar também todas as funcionalidades CRUD do modelo;
- Implementar aplicações WEB e mobile utilizando as ferramentas que a plataforma OutSystems proporciona;
- Realização tanto de tarefas de Front-End como Back-end para a implementação da solução;
- Fazer uso de linguagens de programação como SQL para base de dados, HTML, CSS e JavaScript para realizar tarefas de front-end;
- Implementar a solução usando boas praticas de programação, garantindo uma boa qualidade de dados;
- Realizar testes à solução para garantir o bom funcionamento da mesma;
- Correção de erros que possam surgir e utilizar a ferramenta de debugger da OutSystems para corrigir estes;

## 3.4. Projetos

Neste capítulo são apresentados os vários projetos realizados ao longo do estágio, sendo referida uma explicação de cada um destes, os objetivos e a implementação dos mesmos.

### 3.4.1. Office Ocupancy

Este foi o projeto no qual deu início o estágio. Este projeto trata-se de uma aplicação interna para a empresa a nível nacional para controlar a ocupação dos edifícios existentes em Portugal. Devido a situação pandémica em que todo o mundo se encontra e a necessidade da empresa reencontrar os seus trabalhadores nos escritórios foi necessário ter um controle sobre a ocupação diária dos edifícios respeitando todas as normas exigidas pela DGES.

Para tal foi então criada uma aplicação com o nome "Office Ocupancy" em OutSystems 10. Esta foi elaborada por 2 colaboradores sendo um deles o aluno, logo o trabalho foi dividido entre ambos. O trabalho correspondente ao aluno foi então elaborar CRUDS, páginas de BackOffice correspondentes a criação de novos edifícios e pisos e também a construção da página de FrontOffice "My Reservations".

#### 3.4.1.1. Página "My Reservations"

Nesta página é constituída por dois separadores, no primeiro é onde o utilizador pode visualizar todas as suas reservas futuras, e no segundo separador pode visualizar todas as reservas passadas. Para fazer isto foram usados Agregados como já foi referenciado antes e foi necessário colocar diferentes condições dependendo do separador.

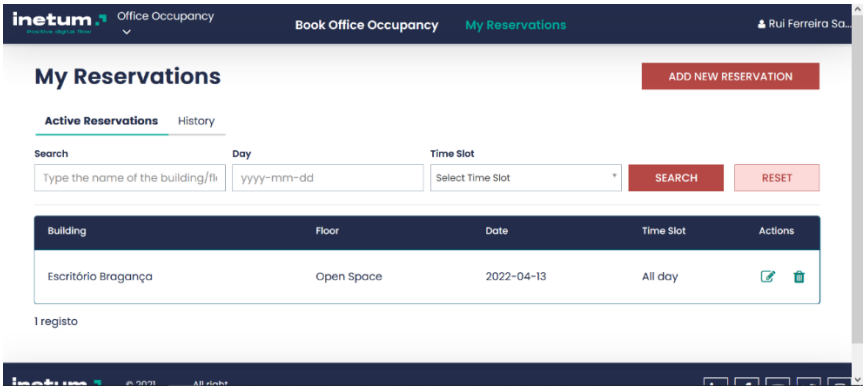


Figura 14 - "My Reservations" separadores de reservas ativas

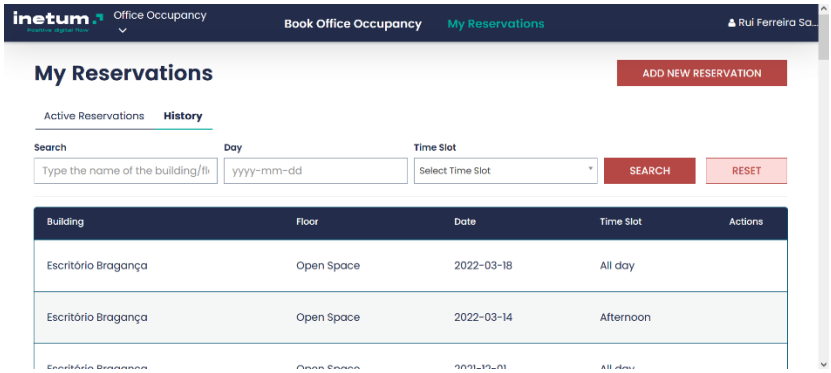


Figura 15 - "My Reservations " separador de reservas passadas

Em ambos os separadores o utilizador pode pesquisar por reservas especificas utilizando filtros diferentes como por nome de escritório, dia de reserva ou mesmo o tipo de reserva (manhã, tarde ou dia completo).

No separador de reservas ativas o utilizador pode eliminar ou editar reservas. Quando o mesmo carrega em eliminar um modal de confirmação é aberto e a reserva é apenas eliminada se o utilizador confirmar.

A outra opção é editar uma reserva, este botão é redirecionado para o mesmo link que o botão de "New Reservation", nesta navegação é enviado um *input* que no caso de a reserva já existir o mesmo será preenchido com o id da reserva e no caso de ser nova o input é preenchido com um id nulo (*NullIdentifier* em OutSystems).

O modal da reserva será preenchido com a informação *default* caso o input seja nulo e é possível ver o mesmo na figura 16:

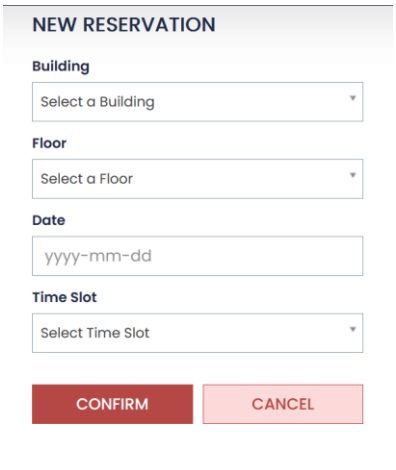


Figura 16 - Modal nova reserva

Neste modal é possível escolher o edifício no qual o utilizador pretende fazer a reserva e dependendo do edifício, a *dropdown* de "floor" será alterada e irá mostrar os pisos do

edifício selecionado. O utilizador também deve escolher uma data e o tipo de reserva e o tipo de reserva que pretende.

No caso de o utilizador não escolher pelo menos uma de estas opções uma exceção é enviada com uma mensagem diferente dependendo da situação. Após o utilizador confirmar a reserva, existe lógica que irá criar nas tabelas correspondentes os registos com esta mesma informação dando uso das CRUDS criadas previamente e também mais alguma lógica adicional.

### 3.4.2. Aplicação para gestão de investimentos

A aplicação de gestão de investimentos foi desenvolvida para uma empresa do setor alimentar. Esta teve o propósito de automatizar a forma como a empresa aprova os seus investimentos, pois antigamente isto era feito através de email.

Neste projeto foi necessário ter Scrums diários de cerca de 15 minutos onde é falado o que cada elemento da equipa fez no dia anterior e quais são os objetivos para o dia atual. Para repartir tarefas de uma forma mais fácil foi utilizado o software Jira onde as tarefas foram colocadas e apenas era necessário associar cada tarefa ao colaborador. Ainda no Jira foram colocados os mockups de cada página associados às tarefas correspondentes. Esta aplicação trata-se de uma aplicação *Reactive* em OutSystems 11.

As primeiras tarefas foram então começar por criar a base de dados com todas as tabelas e atributos necessários e com as suas respetivas restrições. Devido aos requisitos do cliente foi também necessário comentar tudo o que fosse possível como por exemplo para cada atributo criado em uma tabela era necessário deixar a descrição de qual seria o propósito do mesmo.

Para uma melhor organização do trabalho de forma a evitar referencias circulares foi criada apenas uma aplicação com os módulos referentes às bases de dados (CS) e aos BLs que são aqueles em que é feita toda lógica de servidor. Após criar as bases de dados nos respetivos CSs, foram feitas também as CRUDSs das mesmas tabelas nas respetivas BLs.

#### 3.4.2.1. BackOffice

As primeiras páginas a efetuar foram as de BackOffice. Para estas foi então criada uma outra aplicação com o modulo que contém as páginas com o nome “WorkFlowCSPsBO”. A primeira página será então uma listagem de empresas que para cada uma de estas é possível editar ou eliminar a mesma como é demonstrado na figura 17:

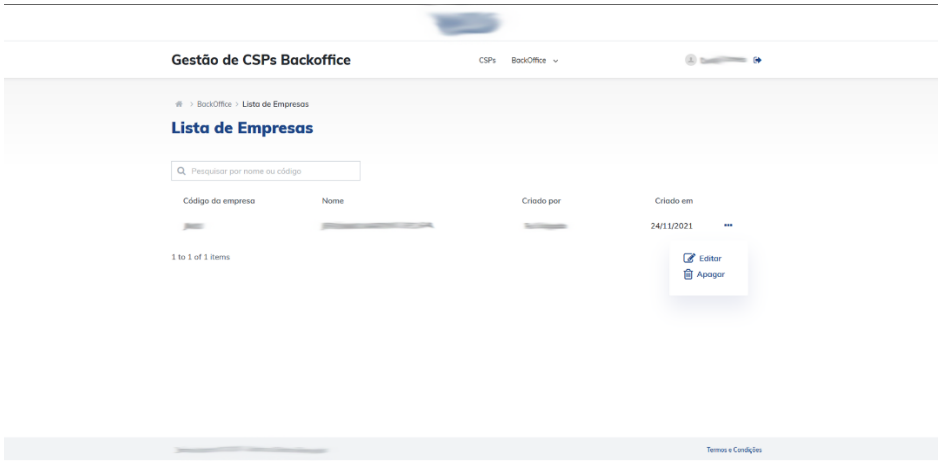


Figura 17 - Lista de empresas

Para esta página foi necessário que o cliente desse acesso a uma tabela com todas as empresas do mesmo, e, quando necessário criar uma empresa na página é criada uma cópia de uma destas empresas fornecidas. Para tal antes de ser redirecionado para a página de criação, é aberto um pop-up onde é necessário escolher a empresa (lista de empresas fornecidas) que pretende mapear. De seguida o utilizador é direcionado para a página de edição que tem a estrutura demonstrada na figura 18:

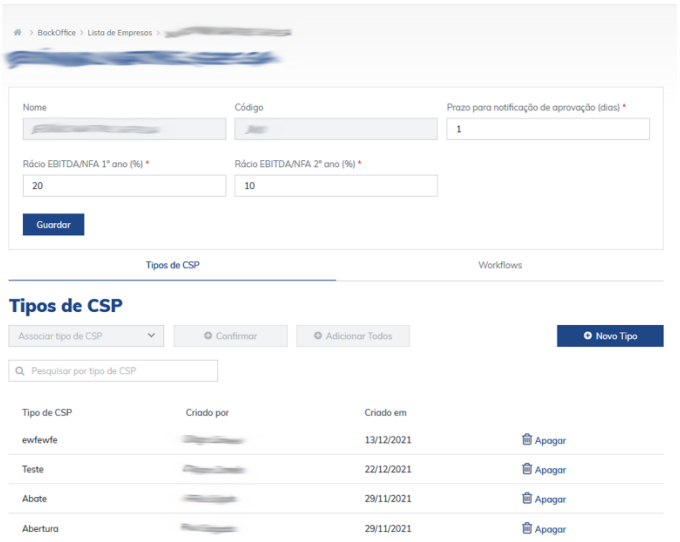


Figura 18 - Detalhe de empresa

Nesta página o nome da empresa e o código são preenchidos automaticamente com os valores da empresa escolhida no pop-up previamente. O utilizador pode guardar a mesma e quando isto acontece ficam visíveis dois separadores com o nome "Tipos de CSP" e "WorkFlows". O conteúdo de estes dois separadores foi colocado em *WebBlocks* para poderem ser reutilizados.

O segundo separador de "Workflow" (figura 19) contém a listagem de todos os *workflows* associados à empresa atual. É possível então copiar um workflow existente, criar, editar ou eliminar um.

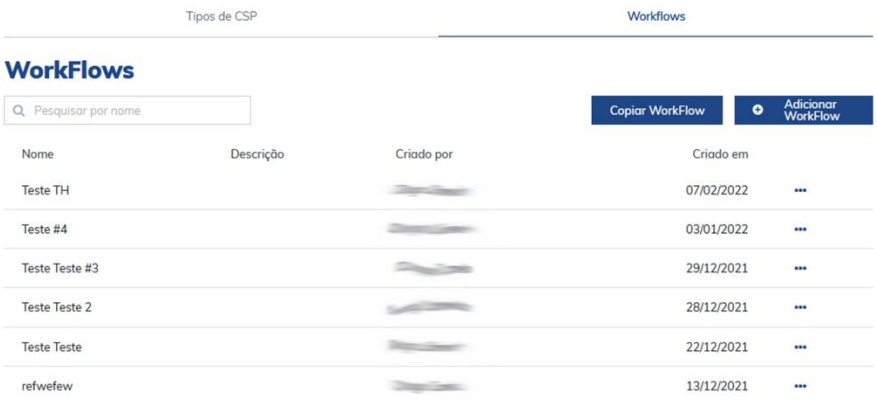


Figura 19 – Separador "Workflow"

Quando o utilizador tenta editar ou criar um workflow é direcionado para a página de detalhe do mesmo. Nesta página, quando o workflow é criado aparecem dois separadores onde é possível adicionar níveis ao workflow.

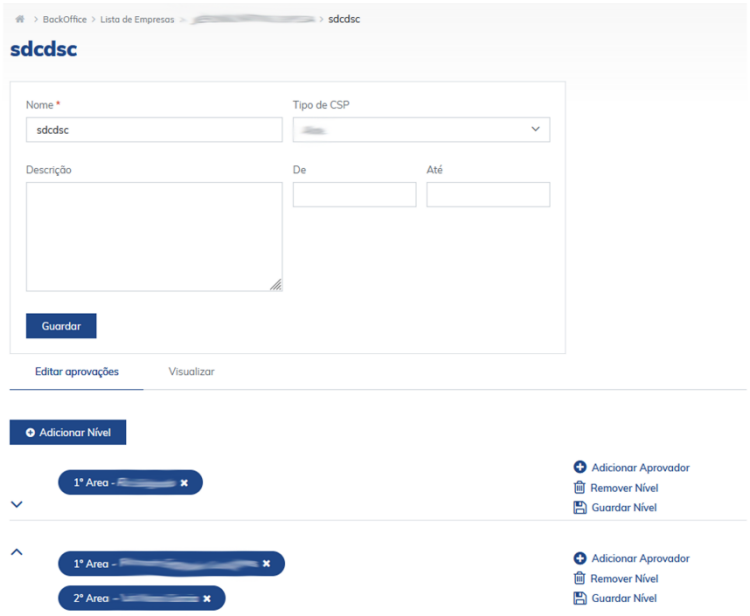


Figura 20 - Detalhe de *WorkFlow*

Cada um de estes níveis contém uma série de aprovadores (aprovador + área). Estes níveis são possíveis remover, criar, guardar ou até mesmo alterar a prioridade do mesmo. Quando um nível é criado ou alterado o botão de adicionar novo nível fica indisponível pois é sempre necessário guardar o mesmo para as alterações serem efetuadas.

Todas as alterações realizadas nesta listagem foram feitas através de uma lista local, o que originou a obrigatoriedade de gravar o nível, para as alterações serem efetuadas e gravadas na base de dados.

#### 3.4.2.2. FrontOffice

Após as páginas e lógica do BackOffice estarem finalizadas e testadas, foi necessário criar então uma outra aplicação para as páginas de FrontOffice. Nesta aplicação é onde é possível ver listagens de CSPs (designação interna da empresa para pedidos de aprovação de mobilização de recursos financeiros), ver cada CSP ao detalhe e aprovar os níveis da mesma até que esta esteja no seu estado final de aprovada ou rejeitada.

A tarefa associada no Jira foi a criação das páginas de detalhe de CSPs. Para tal foi necessário então criar a página seguindo o *mockup* disponível no Jira.

Esta página originou bastante dificuldades devido à quantidade de atributos que uma CSP pode conter e também devido aos estados pelos quais esta passa. A página tem a seguinte estrutura:

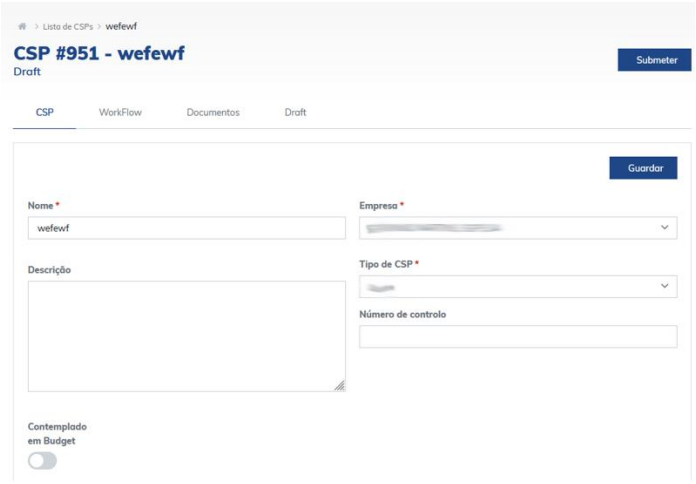


Figura 21 - Detalhe de CSP

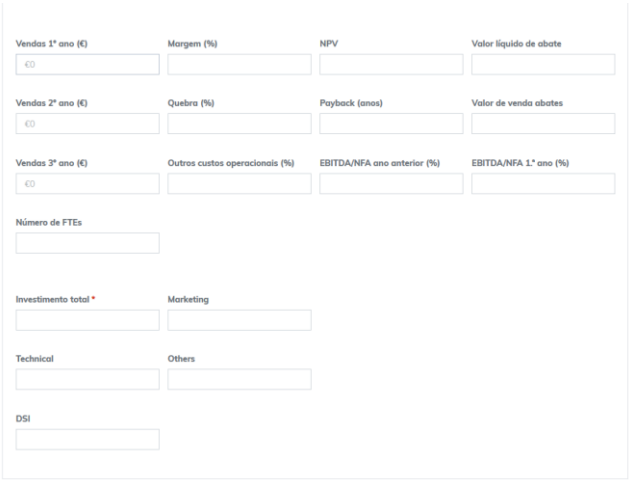


Figura 22 - Detalhe de CSP (cont.)

Como é possível ver nas figuras anteriores existe bastante informação relacionada a uma CSP e para além disso, dependendo das permissões do utilizador existem campos que ficam ocultos ou ficam obrigatórios.

Estas CSP podem passar por 5 estados, sendo estes caracterizados da seguinte forma:

- Draft - versão inicial;
- Pré-aprovação – versão criado após o utilizador guardar o Draft. Neste estado a CSP ainda pode ser alterada, mas depende das permissões dos utilizadores.
- Em aprovação - versão na qual são necessários todos os aprovadores associados aos níveis do workflow associado a esta CSP aprovem ou rejeitem a mesma;
- Aprovada - versão no qual todos os níveis do workflow já foram aprovados;

- Recusada - versão onde basta apenas um aprovador recusar um nível e a CSP passa automaticamente para este estado;

- Assinatura – versão na qual a CSP fica antes de ser aprovada. A CSP apenas fica neste estado se quando o utilizador submete a mesma colocar o campo "Assinatura Holding" (figura 23) a *true*. Quando se encontra neste estado um utilizador com uma *role* especifica deve submeter um documento assinado para a mesma ser aprovada.

Para além dos estados de uma CSP, também existem algumas permissões sendo estas:

- DAF – esta permissão é basicamente um admin, ou seja, permite a este utilizador editar sem problemas uma CSP e aprovar um nível completo de um workflow sem este ser um dos aprovadores;

- Approver – esta permissão serve para quando uma CSP se encontra no estado de Em Aprovação, sendo necessário o aprovador de cada nível ir aprovar/rejeitar este mesmo nível;

- Controller – o utilizador com esta permissão pode alterar os dados da CSP e submeter a mesma caso esta esteja no estado de Draft e Pré-aprovação;

Nesta página de CSP estão disponíveis 4 separadores. Estes apenas aparecem dependendo das permissões do utilizador e do estado da CSP. Estes separadores são:

1. Separador "**CSP**"

Este separador contém a informação da CSP sendo possível alterar a mesma e guardar. Quando a mesma é guardada, a CSP fica sempre no estado Draft.

2. Separador "**WorkFlow**"

Separador que apenas é acessível para utilizadores com as devidas permissões, sendo estes "DAF" e "Controller". Como é possível ver na figura 23 é possível escolher um workflow e depois é possível editar níveis deste mesmo.

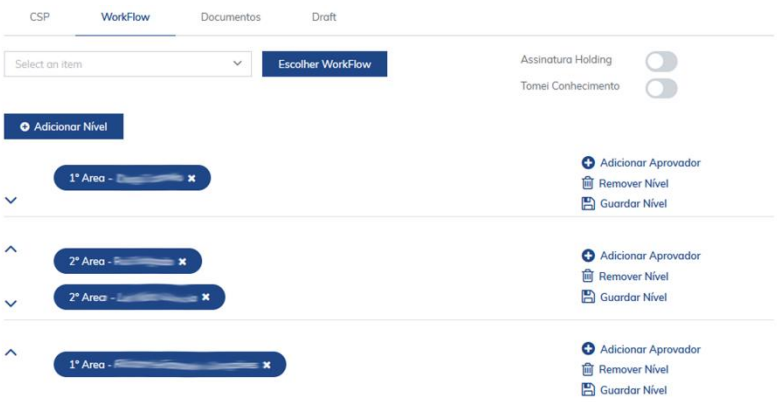


Figura 23 - Separador "WorkFlow"

Como referido anteriormente, no caso de o utilizador colocar a "Assinatura Holding" a *true*, é necessário que um utilizador DAF submeta um documento assinado para a CSP ser aprovada.

3. Separador "**Documentos**"

Neste separador é possível visualizar uma lista de documentos relacionados com a CSP. Os utilizadores com permissões de "DAF", "Controller" e "Approver" podem aceder a esta informação e adicionar/eliminar documentos.

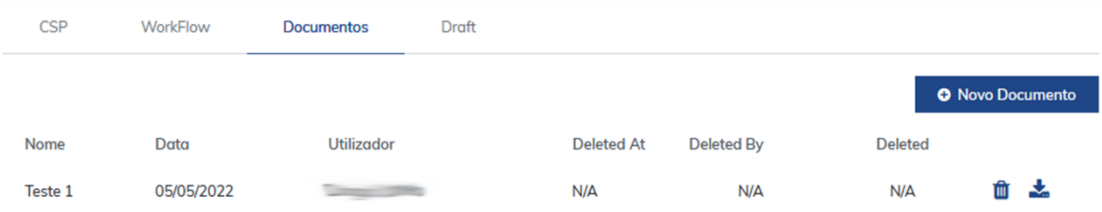


Figura 24 - Separador "Documentos"

4. Separador "**Draft**"

Este separador contém a informação da última versão guardada da CSP no estado "Draft", sendo apenas permitido visualizar e não poder alterar dados. Apenas utilizadores "DAF" têm acesso a este separador e também utilizadores "Controller", caso a CSP esteja no estado Em aprovação.

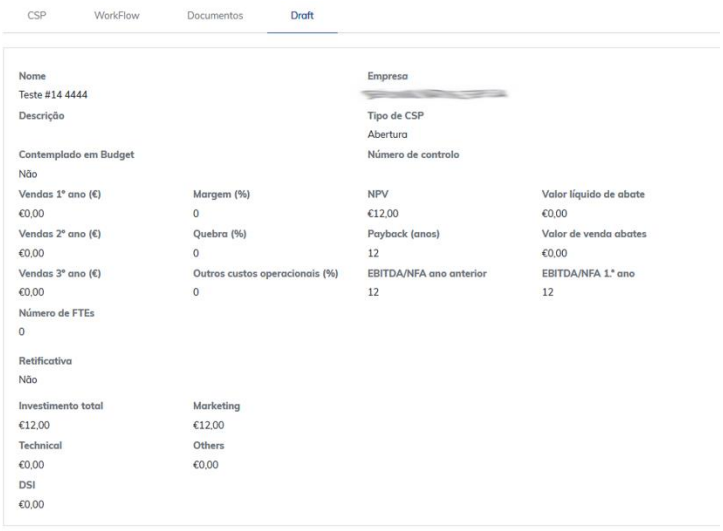


Figura 25 - Separador "Draft"

Após uma CSP ser submetida para o estado de Em Aprovação significa que o workflow já foi definido e então são enviados emails para os aprovadores associados ao primeiro nível do workflow, logo o passo seguinte é cada aprovador de cada nível ter de entrar na informação da CSP, rever se esta conformidade e aprovar. Estas aprovações são feitas por ordem de níveis, ou seja, um aprovador que apenas deva aprovar o nível dois deve esperar que o nível um tenha sido aprovado para o fazer. No final de cada nível estar aprovado são enviados emails para os aprovadores do seguinte nível para informar que existem informações por aprovar.

Para além dos aprovadores, os utilizadores com as permissões “DAF” podem também aprovar níveis, mas, estes têm a possibilidade de aprovar níveis inteiros mesmo que não estejam associados aos mesmos.

Na figura 26 é possível visualizar o workflow quando a CSP se encontra no estado de Em Aprovação. Os níveis quando aprovados ficam com a cor vermelha, o nível que se encontra em aprovação fica com a cor amarela e os restantes níveis com a cor cinzenta. Cada nível também tem uma breve descrição da sua situação atual.

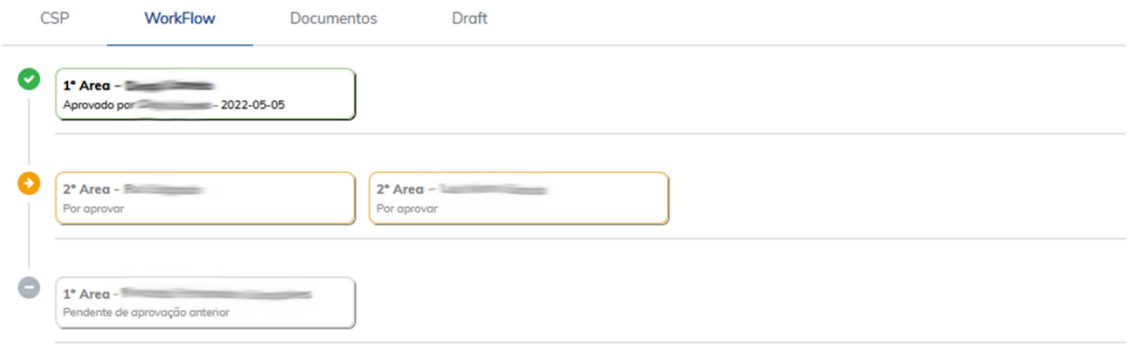


Figura 26 - WorkFlow em aprovação

Este projeto demorou algum tempo a ser realizado devido a complexidade do mesmo. Na parte de BackOffice foram cumpridas todas as datas colocadas inicialmente, mas no seguimento do projeto, mais especificamente no FrontOffice foi necessário prolongar mais um pouco o projeto pois a complexidade de criação e aprovação de CSPs era elevada e além disso foi necessário trabalhar com *Reactive*, que com a falta de experiência atrasou um pouco mais o trabalho.

### 3.4.3. Projeto para empresa de advocacia

Este projeto não se trata de uma implementação nova, mas sim da atualização de uma aplicação Web já existente em OutSystems 10. Esta aplicação fazia uso de webservices com integrações para SAP e era necessário migrar todas estas implementações para passar a utilizar Primavera. Com isto foi necessário alterar a lógica antiga para poder suportar estas alterações novas. Em algumas de estas alterações foi necessário refazer ecrãs antigos e assim aproveitar para melhorar a lógica existente de forma que os utilizadores possam dispor de uma utilização mais rápida e satisfatória.

Esta aplicação é utilizada para gestão interna de faturação, clientes, empregados entre outros. Devido a complexidade foi necessário começar primeiro por perceber o modelo de dados.

Devido à antiguidade da fábrica houve um problema de referências circulares. Na figura 27 é possível ver como funciona uma referência circular entre o modulo A e B, o problema é que à medida que cada um de estes módulos é publicado, estes passam a depender um do outro, logo as referências serão sempre incompatíveis o que origina problemas.


Figura 27 - Referência circular

Para evitar esta situação, em OutSystems é uma boa pratica a existência de vários módulos para diferenciar todas as fazes do projeto, por exemplo é de boa pratica criar apenas um módulo para a base de dados com as CRUDS das tabelas existentes, criar outro modulo onde seja realizada toda a lógica a nível de servidor e finalmente um outro módulo onde são realizados os ecrãs. Em outros casos como quando é necessário expor webservices é necessário criar outro módulo "IS". É possível existir também outros módulos para situações diferentes.

Logo, o modelo ideal para não existir referências circulares e assim não haver problemas é ilustrado na figura 28:

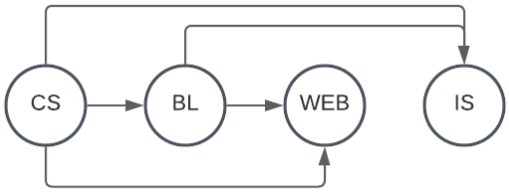


Figura 28 - Forma de evitar referências circulares

Devido a esta situação foi necessário refazer toda a lógica existente e ter em atenção este pormenor das referências e criar novos módulos onde seja implementada a nova lógica.

#### 3.4.3.1. Criação de webservices e lógica associada

Como é necessário criar diversos webservices para módulos diferentes foi necessário criar um módulo dedicado apenas a consumir webservices que a Primavera expõe, com o nome "Primavera_IS". A lógica implementada é simples, simplesmente cada ação criada recebe um *input* que será o que irá de *input* para o WebService e o *output* será igual ao *output* devolvido pelo WebService. Em cada ação foi criada uma outra ação para guardar tanto o request como a resposta de estas ações, ou seja, são chamados de logs. Assim é possível

rastrear tudo o que é enviado e recebido e no caso de acontecer algum problema ser possível visualizar tanto os *inputs* quanto os *outputs*.

Estas ações são todas parecidas e seguem o esquema representado na figura 29:

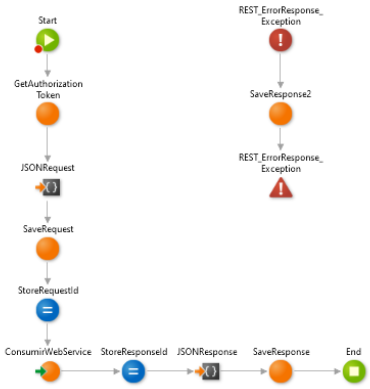


Figura 29 - Ação de WebService

Assim em cada "BL" onde seja necessário enviar informação para Primavera basta chamar a ação correspondente do modulo Primavers_IS e assim evitar referências circulares.

- **Clientes**

A primeira integração realizada foi a criação ou edição de clientes, ou seja, cada vez que um cliente é criado ou editado é necessário para além de guardar os dados na base de dados, também enviar para primavera todas estas alterações para que assim a informação esteja sincronizada corretamente.

Para enviar os dados corretamente para Primavera é necessário preencher uma estrutura do *request*, para tal foi construída uma ação que recebe o id do cliente e procura toda a informação relativa ao mesmo como é possível visualizar nas figuras seguintes:

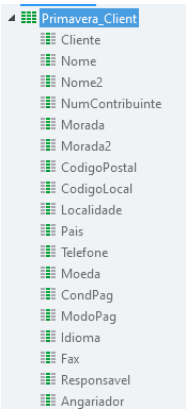


Figura 30 - Estrutura do cliente

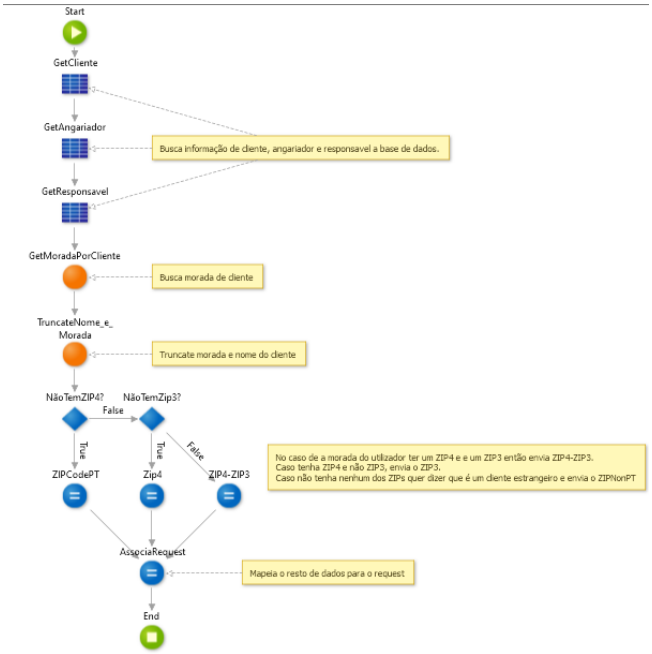


Figura 31 - Ação para mapear cliente

Na figura 31 apresenta-se a ação que procura as informações relativas ao cliente, as informações relativas ao responsável e angariador do cliente, a morada e também tanto a morada como o nome do cliente cortados até 50 carateres. Após isto é necessário verificar se o cliente é estrangeiro ou não e dependendo disso, o código postal do mesmo é mapeado de forma diferente. Finalmente existe um último "*assign*" onde toda a informação recolhida é mapeada para o *output* e de seguida é envidado como *request* para o WebService.

- **Empregados**

Esta integração refere-se a cada vez que um empregado é criado ou alterado é necessário gravar na base de dados e enviar também para o WebService da Primavera.

Nesta mesma integração foi necessário criar Webservices tanto para consumir Webservices disponibilizados por Primavera, e também expor Webservices para que cada vez que fosse feita alguma alteração do lado de Primavera, fosse guardada estas alterações na base de dados.

Inicialmente foi criado o WebService para consumir o WebService do lado de Primavera. Para isto, foi necessário criar uma ação que através de um id de Primavera ou do NIF do empregado fosse obter toda a informação do mesmo e preenchesse de seguida o request do WebService para ser enviado para o lado de Primavera. Na figura 32 é possível visualizar a ação criada para buscar e enviar os dados para Primavera.

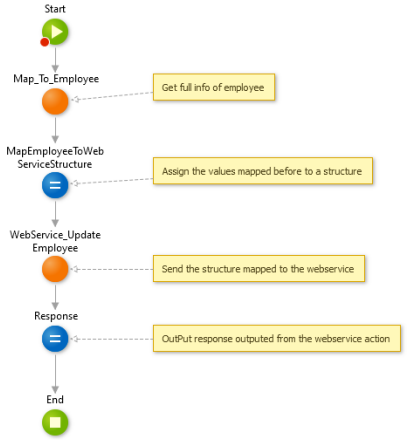


Figura 32 - Ação para mapear e enviar informação para Primavera

Nesta ação inicialmente é necessário obter toda a informação do empregado utilizando uma outra ação com o nome “Map_To_Employee”, esta é demonstrada na figura 33.

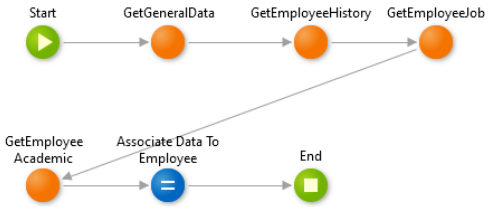


Figura 33 - Ação "Map_To_Employee"

Cada uma das ações usadas nesta mesma ação(“Map_To_Employee”) servem para ir pesquisar a informação necessária a cada uma das tabelas que contenham a informação do empregado. Todas estas são bastantes parecidas e funcionam da seguinte forma:

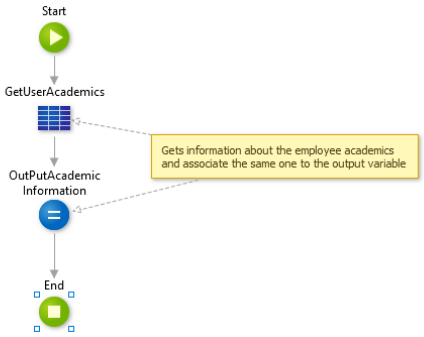


Figura 34 - Ação "GetEmployeeAcademic"

Basicamente, é necessário usar Agregados para obter a informação e depois associar a esta informação a uma variável de *ouput* para poder ser depois utilizada para preencher alguma informação da variável de *output* da ação "Map_To_Employee".

Após cada uma das ações recolherem a informação necessária é então associado a uma variável de *output* toda esta informação fazendo uso de um "*assign*" como é possível ver na figura 33. Esta variável de *output* tem a seguinte estrutura:

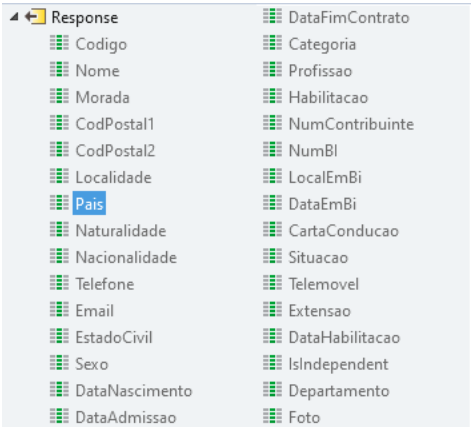


Figura 35 - Estrutura de *output*

Depois de ter toda a informação recolhida e já colocada na variável de *output* é possível então chamar a ação que consume o WebService do lado da primavera e assim enviar toda esta informação. Esta ação é muito parecida com a da figura 29, sendo as únicas diferenças as variáveis de *input*, *output* e os *endpoits*.

Após a conclusão desta ação foi então criado o WebService para poder ser consumido do lado do Primavera. Este tipo de WebServices são criados do em um módulo diferente (IS) para não criar referências circulares. Basicamente todos os WebServices que irão consumir WebServices do lado da Primavera ficam em um módulo e os WebServices que serão expostos serão colocados em módulos diferentes.

Logo, foi criado um WebService que chama uma ação chamada "CreateOrUpdateEmployeeRecord". Esta ação recebe uma estrutura como a da figura 35 que é demostrada na figura 36:

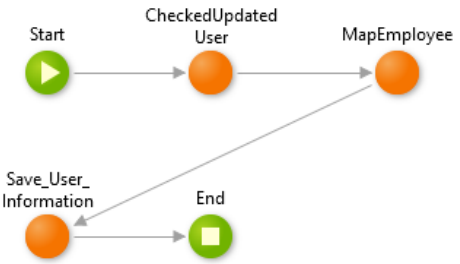


Figura 36 - Ação "CreateOrUpdateEmployeeRecord"

Nesta ação são utilizadas outras 3 ações. A primeira é a "CheckdUpdatedUser" (figura 37), na qual é necessário chamar uma outra ação como é possível ver na figura 38, esta ação "GetUserMasterRecord" recebe dois parâmetros de *input*, sendo estes o Primavera code do empregado e o email do mesmo.

A ação começa então por verificar se o código de primavera está vazio, se isto acontecer então tenta obter a informação do empregado através do email, caso o Primavera code esteja preenchido então tenta buscar a informação utilizando o mesmo, mas se não for encontrado nenhum registo na base de dados tenta também pesquisar novamente utilizando o email. Caso também não tenha sucesso a encontrar o empregado com esse email é acionada uma exceção que depois cria um erro e envia a informação toda na resposta do WebService. Caso tenha sucesso em recolher a informação do empregado é então associada essa informação à variável de *output*.

Após fazer todas as verificações e recolher a informação na ação "GetUserMasterRecord", volta a verificar se o código de primavera está preenchido e no caso de sucesso então guarda toda a informação que foi obtida da ação anterior, mas caso o código de Primavera esteja vazio não é guardada nenhuma informação.

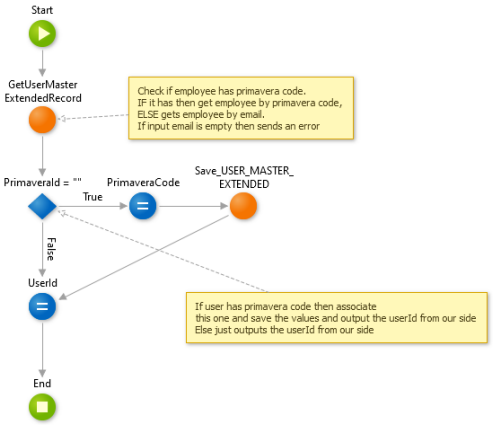


Figura 37 - Ação "CheckdUpdatedUser"

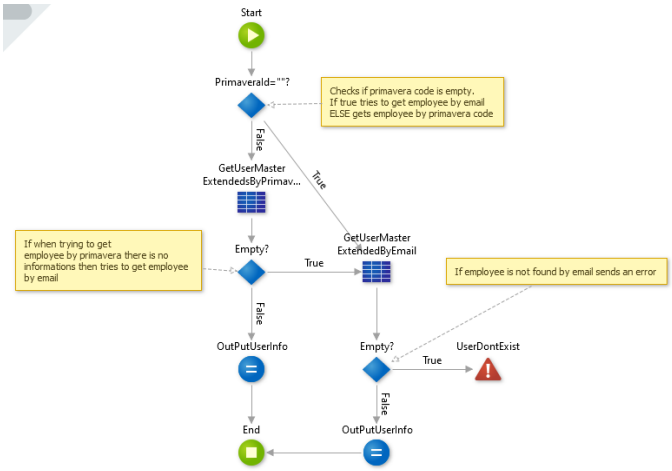


Figura 38 - Ação “GetUserMasterRecord”

Após a ação “CheckdUpdatedUser” é então chamada a ação “MapEmployee” (figura 39), na qual serve para recolher toda a informação necessária sobre o empregado e associar a mesma às variáveis de *output* e depois atualizar esta mesma informação com a nova informação para assim na ação seguinte “Save_User_Information”(figura 40) guardar os novos dados que foram enviados como *ouput* da ação anterior.

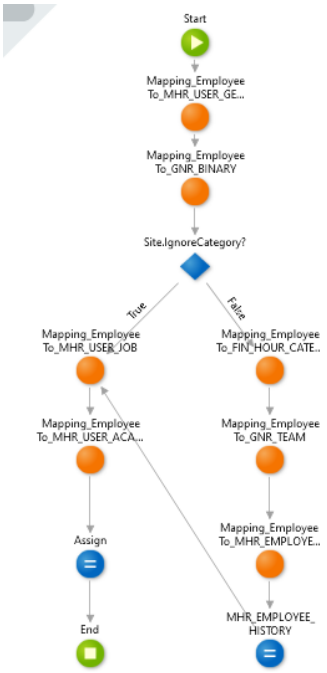


Figura 39 - Ação "MapEmployee"

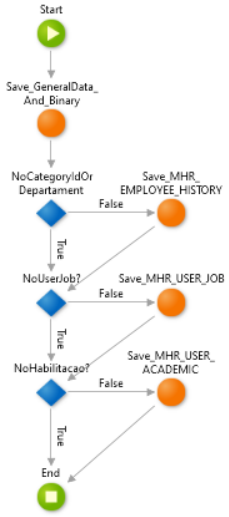


Figura 40 - Ação "Save_User_Information"

E assim, após guardar toda a informação a ação acaba.

- **Fornecedores**

Nesta integração foi necessário cria um novo módulo dedicado apenas a fornecedores. Logo foram criados 2 módulos, o Suplier_CS e Suplier_BL. No modulo Suplier_CS é onde foi necessário criar uma tabela dedicada a fornecedores. Quanto ao modulo Suplier_BL é onde foram criadas todas as cruds relativas a tabela de fornecedores e também um timer que chama um WebService que Primavera expõe e guarda os registos de todos os fornecedores disponibilizados na base de dados.

Para a construção de este timer foi então necessário criar a ação que consome o WebService de Primavera que será bastante parecida com a da figura 23, mudando apenas os *endpoits*, *inputs* e *outputs*. Após esta ação estar criada foi necessário começar com a construção do *timer*, este quando acionado executa a seguinte ação:

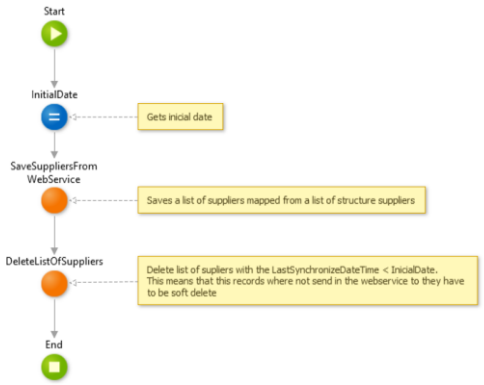


Figura 41 – Ação "ListOfSuppliers_Timer_Action" executada pelo *timer*

Esta ação começa por chamar uma outra ação chamada "SaveSuppliersFromWebService" (figura 42), que começa por consumir o WebService do lado do Primavera que retorna uma lista com todos os fornecedores.

Após ter acesso à lista onde é executada a ação "AssociateAndSaveSupplier" (figura 43) na qual a lista de fornecedores é percorrida, é feito uma chamada à base de dados para verificar se esse registo já existe e, no caso de existir associar apenas os metadados do mesmo, mas no caso de ser novo, deve criar um novo registo. Independentemente de ser um novo fornecedor ou não, é associada a variável "LastSynchronizeDateTime" que contém a *DateTime* inicial de quando o *timer* executou inicialmente.

Após guardar toda a informação existe, uma outra ação "DeleteListOfSuppliers" (figura 44) que obtém todos os fornecedores a base de dados e verifica quais destes contém através da variável "LastSynchronizeDateTime" a menor data inicial obtida no início da ação, e caso esta seja menor significa que estes registos já não se encontravam no WebService logo deve ser apagado utilizando um *soft delete*.

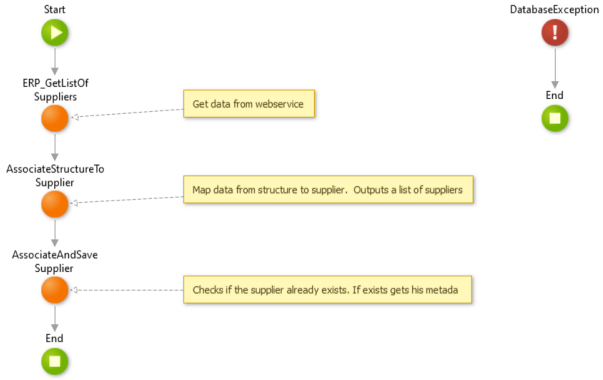


Figura 42 - Ação "SaveSuppliersFromWebService"

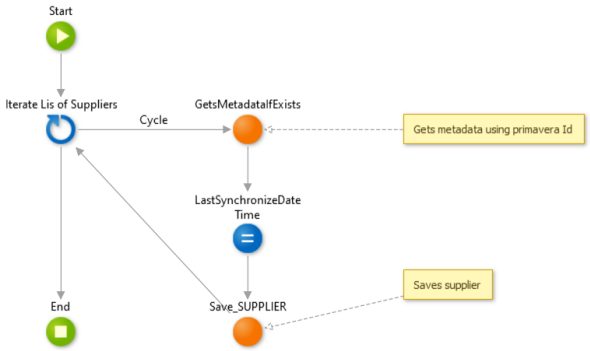


Figura 43 - Ação "AssociateAndSaveSupplier"

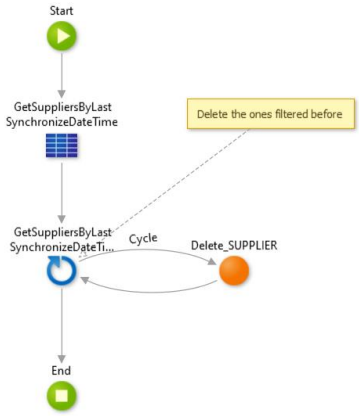


Figura 44 - Ação "DeleteListOfSuppliers"

- **Departamentos**

Nesta integração mais uma vez foi necessário criar o WebService para consumir um outro WebService disponibilizado pelo Primavera, para que, cada vez que um departamento é criado ou editado a informação seja também refletida do lado do mesmo.

Este WebService é como referido anteriormente igual aos anteriores apenas alterando os endpoits, *inputs* e os *outputs*. Foi necessário criar a lógica para sempre que um

departamentos seja criado/editado, esta informação seja enviada ara Primavera. Do outro lado apenas necessitam de um id e o nome do departamento, então a estrutura é representada na figura 45:

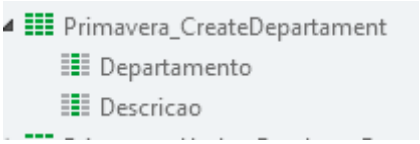


Figura 45 - Estrutura de departamento

Para tal a ação criada tem o nome de "Save_And_Send_Departament" e é a seguinte:

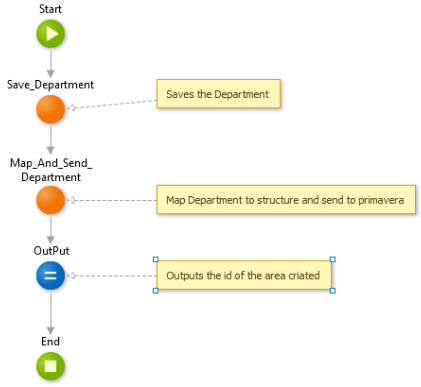


Figura 46 - Ação "Save_And_Send_Departament"

Nesta ação a informação do departamento é primeiro guardada na base de dados e de seguida existe outra ação com o nome "Map_And_Send_Department" (figura 47), na qual o departamento é mapeado para a estrutura referida anteriormente, e de seguida chamada a ação "CreateOrUpdateDepartament", onde esta informação mapeada é enviada para Primavera através do WebService.

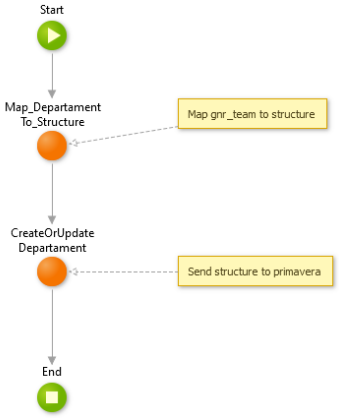


Figura 47 - Ação "Map_And_Send_Department"

Após a integração ser efetuada e testada houve a necessidade de enviar todos os departamentos existentes na base de dados para o lado de Primavera, para que a informação faça sentido os ambos lados.

Para a informação estar consolidada de ambos os lados, foi necessário criar um *timer* que executa a ação "SendAllDepatments" (figura 48), esta ação usa um agregado para buscar uma lista de todos os departamentos existentes na base de dados, e, de seguida, percorrer esta mesma lista utilizando um *for each* para que, para cada registo seja executada a ação referida anteriormente "Map_And_Send_Department" (figura 47), que mapeia o registo para a estrutura de Primavera e envia os dados.

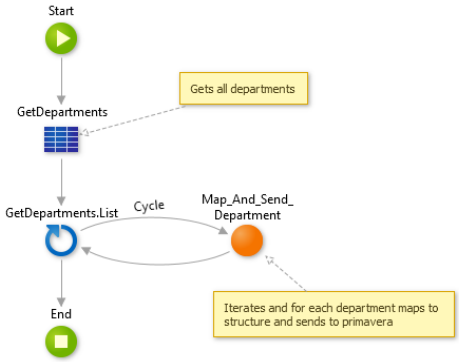


Figura 48 - Ação "SendAllDepatments"

- **Processos**

Para cada cliente existem processos associados que servem para inserir faturas. Cada um destes processos precisa de ser refletido também para Primavera. Foi então criada a ação "Map_And_Send_To_Primavera", que é utilizada cada vez que um processo é criado/editado e esta é ilustrada na figura 49:

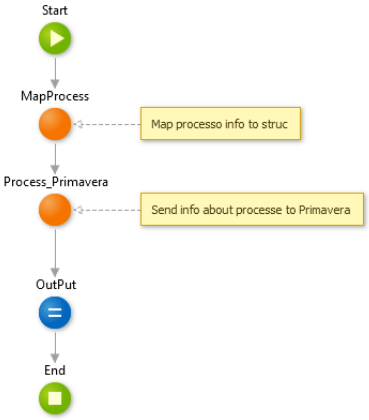


Figura 49 - Ação "Map_And_Send_To_Primavera"

Esta ação começa por mapear a informação do processo para uma estrutura representada na figura 50:

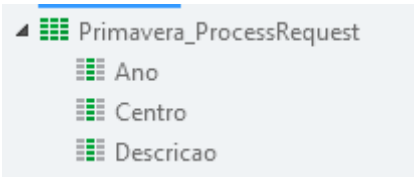


Figura 50 - Estrutura de processo

Para preencher a informação da estrutura foi necessário utilizar um *assign*.

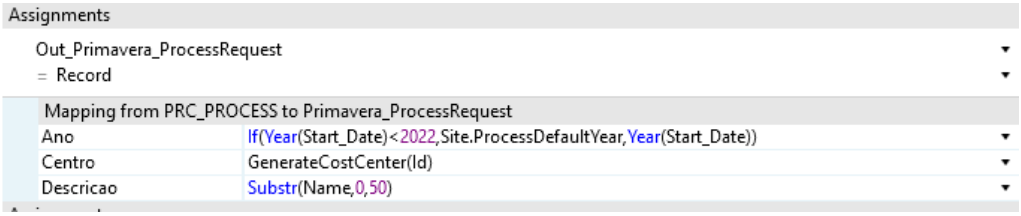


Figura 51 - *Assign* processo a estrutura

Para preencher o ano, é preciso verificar se a o ano na data inicial existente no processo é inferior a 2022, então envia o valor presente em um *site propertie* (estas *site properties* são variáveis globais que têm valores constantes*)* com o valor "2022", caso o valor seja superior então envia o ano dessa data. Quanto a descrição, o valor suportado do lado de Primavera é apenas 50 caracteres, logo houve a necessidade de utilizar uma função existente em OutSystems chamada "Substr" na qual é devolvida uma string com o tamanho desde os valores inicias e finais introduzidos sendo estes 0 e 50 respetivamente.

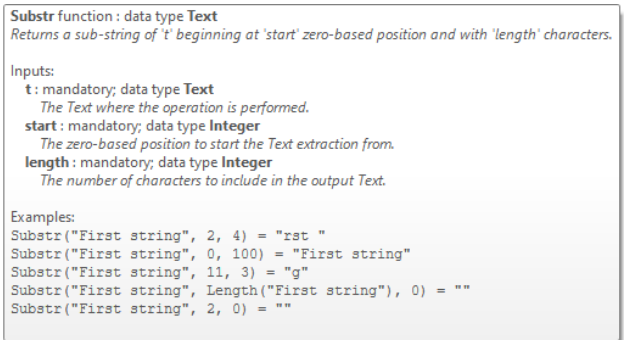


Figura 52 - Função "Substr"

Finalmente no mapeamento de "centro" foi necessário criar uma ação que pode ser usada como função (a ação passa a ser possível ser usada em *assigns*, mas apenas devolve um *output*) com o nome "GenerateCostCenter" que é possível ver na figura 53:

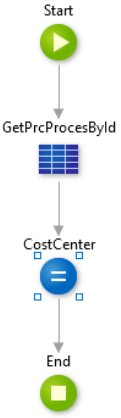


Figura 53 - Ação “GenerateCostCenter”

Esta ação recebe como input o id do processo e tem como objetivo devolver uma string com uma estrutura especifica. É então necessário primeiro ter acesso à informação do processo e de seguida fazer o mapeamento para a string de *output* como é possível ver na seguinte figura:

```
"MT"
+
FormatText(Numero de cliente,5,5,True,0)
+
FormatText(Numero de processo,5,5,True,0)
+
Segundo Numero de processo
```

Figura 54 - Mapeamento de string

Neste mapeamento foram omitidos os nomes das variáveis utilizadas pois poderiam revelar informação confidencial. O mapeamento utiliza uma função fornecida por OutSystems com o nome “FormatText” que é explicada na figura 55:

**FormatText** function : data type **Text**
*Builds a Text output of the specified Text 'value', by limiting it to the specified 'max_chars' count. If 'value' has less than the 'min_chars' characters limit, enough 'padding_char' characters are added to expand the length to that limit. In this case, 'left_padding' determines where the padding should be added.*

Inputs:
- **value** : mandatory; data type **Text**
  *The Text to be formatted.*
- **min_chars** : mandatory; data type **Integer**
  *The minimum number of characters in the output.*
- **max_chars** : mandatory; data type **Integer**
  *The maximum number of characters in the output.*
- **left_padding** : mandatory; data type **Boolean**
  *Indicates in which side the Text is padded.*
- **padding_char** : mandatory; data type **Text**
  *The character to use for padding the string to the minimum length.*

Examples:
```
FormatText("123456789", 3, 9, True, "#") = "123456789"
FormatText("123456789876", 3, 9, True, "#") = "456789876"
FormatText("123456789876", 3, 9, False, "#") = "123456789"
FormatText("12345", 10, 20, True, "#") = "#####12345"
FormatText("12345", 10, 20, False, "#") = "12345#####"
```

Figura 55 - Função "FormatText"

Após o mapeamento ser feito é então associada esta string à variável de “Centro” e assim a estrutura da figura 50 está preenchida na totalidade, sendo possível de seguida

enviar a informação para Primavera através da ação “Process_Primavera”, que envia esta mesma através de um WebService.

- **Categorias**

Cada empregado tem uma categoria associada na empresa e para tal foi necessário criar também uma integração para enviar estas mesmas categorias para Primavera.

Foi então criada a ação "Save_And_Send_Category_To_primavera" como é possível ver na figura 56:

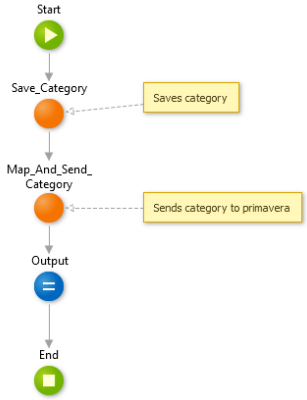


Figura 56 - Ação "Save_And_Send_Category_To_primavera"

Nesta ação começa por gravar a informação da categoria na base de dados e de seguida executa uma outra ação com o nome "Map_And_Send_Category" (figura 57) na qual é primeiro mapeada a informação da categoria para uma estrutura de Primavera (figura 58) e depois enviada para Primavera.

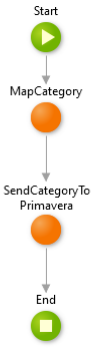


Figura 57 - Ação "Map_And_Send_Category"

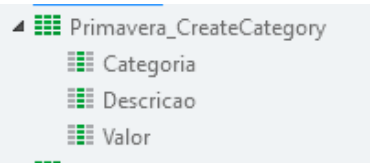


Figura 58 - Estrutura categoria

Após a criação de todas as integrações foram desenvolvidos testes integrando as equipas envolvidas. À medida que erros eram encontrados foram efetuadas as alterações necessárias para a correção dos mesmos.

Com estas integrações efetuadas foi então necessário fazer Upload das mesmas para produção. OutSystems facilita bastante este processo, pois basta criar uma solução com todos os módulos nos quais foram efetuadas as alterações e de seguida, simplesmente é necessário publicar este mesmo módulo no servidor de produção. Caso exista alguma dependência antiga, durante o upload é também mostrado um aviso de que pode ocorrer problemas durante o mesmo. Para tal basta refrescar estas mesmas dependências e publicar novamente a solução.

#### 3.4.3.2. Manutenção de fábrica

Após todo o trabalho ter sido efetuado e testado o objetivo no seguimento do estágio foi continuar nesta mesma fábrica e tratar da manutenção da mesma, corrigindo erros que fossem encontrados, e também implementar novas soluções que o cliente pretendesse.

Devido a antiguidade da fábrica foi bastante recorrente aparecerem problemas tanto com referências circulares como erros de lógica antiga.

- **Criação de novos ecrãs**

Durante a manutenção da fábrica houve a necessidade de atualizar as páginas da mesma. O objetivo foi então reformular os ecrãs que o cliente necessitasse e assim refazer também a lógica.

Um dos trabalhos propostos foi realizar novos ecrãs para os dados recebidos pelo WebService criado previamente de conta-corrente. Os dados fornecidos neste WebService poderiam ter bastantes tipos desde adiantamentos de clientes, a faturas já pagas pelo mesmo, entre outras.

Inicialmente, o objetivo era quando o utilizador tivesse a necessidade de visualizar estes dados, simplesmente chamar o WebService e tratar os dados. Mas á medida que foi feita nova lógica ocorreu situações em que o utilizador necessitava de esperar algum tempo para mostrar dados. A solução foi então criar tabelas na base de dados para guarda os dados através de um timer que é acionado todos os dias e atualiza os dados.

Assim através de chamamentos à base de dados o tempo de espera diminuiu drasticamente pois é possível filtrar os dados que são necessários, por exemplo, pesquisar apenas dados com o tipo de documento "ADC" que seriam adiantamentos. O grave problema do WebService é que, eram devolvidos todos os dados e apenas depois era possível filtrar, o que por consequência originava tempo de espera mais elevados.

Um dos ecrãs pretendidos foi dos relatórios que continha esta informação de conta-corrente. Neste ecrã é possível ver variada informação de conta-corrente,mas para não criar várias páginas diferentes foi colocada uma *combo box* que permite escolher que tipo de dados o utilizador pretende visualizar como é possível ver na figura 59.

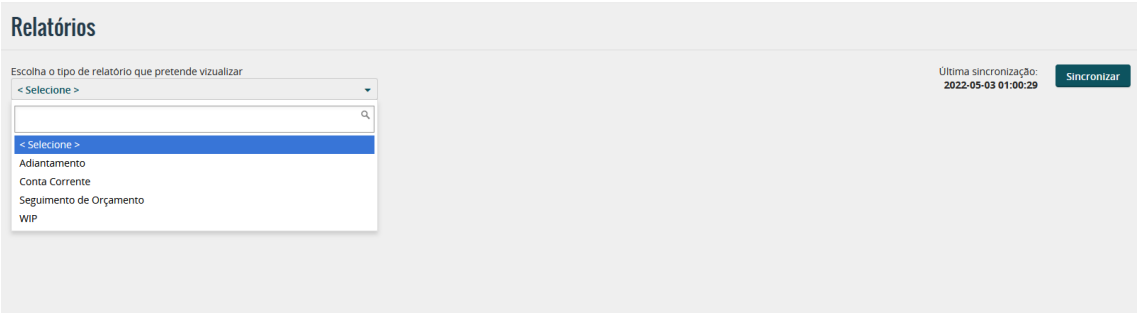


Figura 59 - *Combo-box* de relatórios

Dependendo da opção escolhida a ação associada à *combo box* é direcionada para um *WebBlock* que contém todas as páginas possíveis de destino. Este WebBlock tem como *input* o tipo de documento (selecionado na *combo box*), sendo este o que define qual será o destino da navegação. Nesta página é também possível acionar o timer que atualiza os dados da base de dados com os valores da conta-corrente vindos do WebService, pressionando o botão de "Sincronizar".

O *WebBlock* implementado foi o dos adiantamentos, esta mostra uma lista de adiantamentos (documentos com um tipo específico "ADC"), que pode ser filtrada pelos filtros também disponíveis como é possível ver na figura 60:

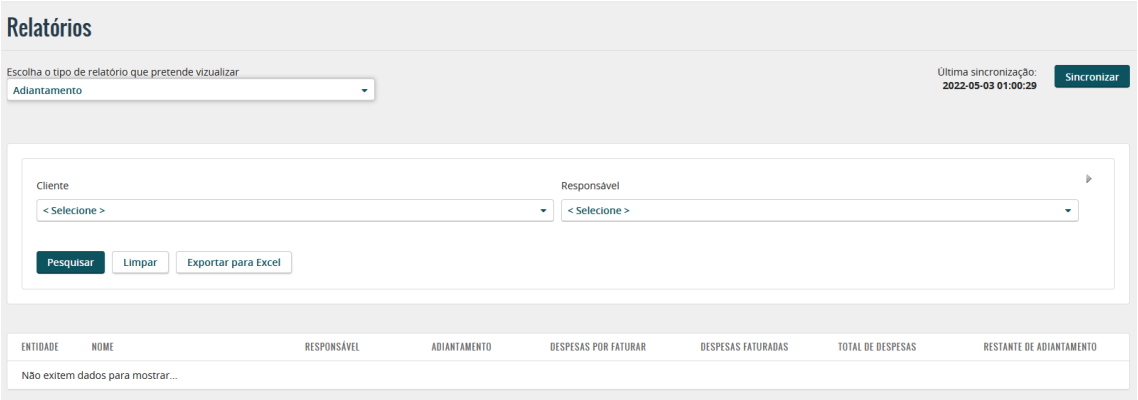


Figura 60 - *WebBlock* de adiantamentos

Após o utilizador escolher o tipo de dados que pretende visualizar (neste caso adiantamentos) o WebBlock fica visível, mas nenhum dado é carregado, isto porque, para mostrar os dados é necessário tratar os mesmos que por consequência irá causar um tempo de espera desnecessário ao utilizador. Logo, após o utilizador preencher os filtros que deseje basta carregar no botão “Pesquisar” e a lista será atualizada.

Quando o utilizador carrega então no botão de pesquisar é executada uma ação chamada "GetListOfAdvances" que é possível ver na figura 61:

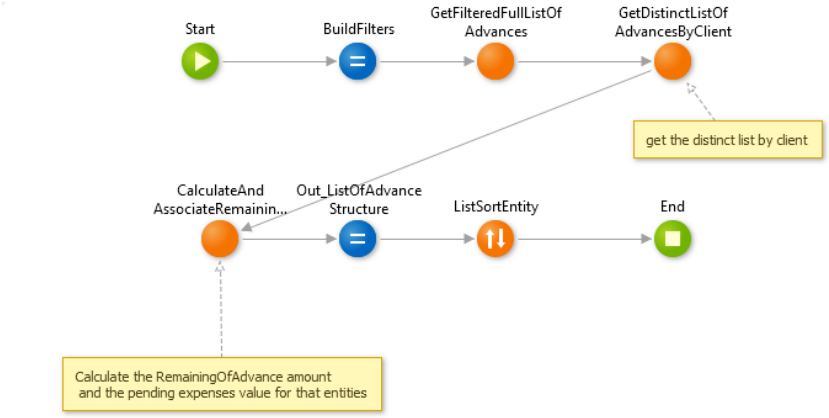


Figura 61 - Ação "GetListOfAdvances"

Esta ação começa por chamar uma ação chamada “GetFilteredFullListOfAdvances” (figura 62) que busca inicialmente uma lista às tabelas que contêm a informação da conta-corrente, filtrando esta mesma com os filtros que o utilizador colocou. No caso de esta página, a mesma tem sempre o tipo de documento como “ADC”, sendo este para adiantamentos.

No ecrã de listagem não houve a necessidade de dividir os dados por processos sendo estes agrupados por cliente, ou seja, se um cliente tiver diversos adiantamentos, estes

serão agregados ao mesmo e será mostrado na tabela uma soma total dos adiantamentos. Na figura 62 está representada a ação que obtém os dados da conta-corrente à base de dados, esta ação foi construída de forma a poder ser reutilizada e independentemente do tipo de documentos, sendo utilizada em todos os ecrãs de conta-corrente.

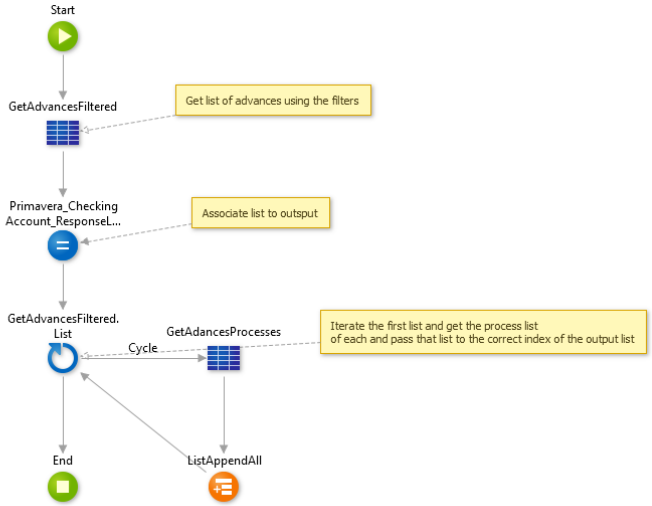


Figura 62 - Ação “GetFilteredFullListOfAdvances”

Depois da execução da ação, uma lista de adiantamentos é devolvida e é então necessário agrupar esta lista por cliente. Isto foi feito na ação seguinte chamada “GetDistinctListOfAdvancesGroupByClient”.

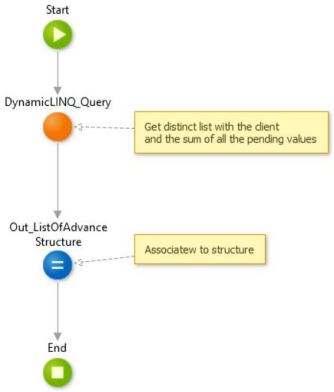


Figura 63 - Ação “GetDistinctListOfAdvancesGroupByClient”

Nesta ação é utilizada uma ação chamada “DynamicLINQ_Query” (permite consultas LINQ usando métodos de extensão que usam argumentos de string) e a *query* realizada é a seguinte:

```
"
GROUP BY
{CheckingAccount_NoProcessList}.[Entidade],
{CheckingAccount_NoProcessList}.[NomeEntidade],
{CheckingAccount_NoProcessList}.[Responsavel],
{CheckingAccount_NoProcessList}.[ResponsavelNome]
SELECT
it.key.Entidade,
SUM({CheckingAccount_NoProcessList}.[ValorTotalEUR]),
it.key.NomeEntidade,
it.key.Responsavel,
it.key.ResponsavelNome
"
```

Figura 64 - *Query* para agrupar adiantamentos por cliente

Após a execução da *query*, os dados devolvidos já estarão agregados por cliente e uma soma de adiantamentos é também devolvida. De seguida é necessário obter todas as despesas de cada cliente.

Foi criada a ação "CalculateAndAssociateRemainingOfAdvanceAndStatus", na qual é necessário iterar a lista retornada anteriormente, pesquisar todas as despesas pagas e por aprovar de cada cliente e utilizar estes valores para fazer o cálculo do restante de adiantamento (adiantamento – soma de despesas). Com o valor de restante de adiantamento é possível saber o estado atual de este adiantamento e dependendo do mesmo será retornado um status diferente para cada registo.

Na figura 65 é representado um exemplo de dados retornados nesta página:

| ENTIDADE | NOME | PROCESSO APENSO | ORÇAMENTO | POR FATURAR | FATURADAS | TOTAL | RESTANTE DE ORÇAMENTO | |
|---|---|---|---|---|---|---|---|---|
| [redacted] | [redacted] | [redacted] | 45,00 € | 2.968,26 € | 132.858,08 € | 135.826,34 € | -135.781,34 € | ⚠ |
| [redacted] | [redacted] | [redacted] | 60,00 € | 0,00 € | 0,00 € | 0,00 € | 60,00 € | ✓ |
| [redacted] | [redacted] | [redacted] | 68,00 € | 0,00 € | 0,00 € | 0,00 € | 68,00 € | ✓ |
| [redacted] | [redacted] | [redacted] | 50,00 € | 126,00 € | 0,00 € | 126,00 € | -76,00 € | ⚠ |
| [redacted] | [redacted] | [redacted] | 100,00 € | 34,00 € | 0,00 € | 34,00 € | 66,00 € | ✓ |
| [redacted] | [redacted] | [redacted] | 12,00 € | 25,00 € | 0,00 € | 25,00 € | -13,00 € | ⚠ |
| [redacted] | [redacted] | [redacted] | 23,00 € | 0,00 € | 0,00 € | 0,00 € | 23,00 € | ✓ |
| [redacted] | [redacted] | [redacted] | 12,00 € | 0,00 € | 0,00 € | 0,00 € | 12,00 € | ✓ |

Figura 65 - Lista de adiantamentos

Por fim até a data de conclusão do estágio as atividades realizadas basearam-se em correção de erros que ocorriam em produção e a atualização de páginas para novos requisitos do cliente.

# Capítulo 4 Conclusões

No decorrer do estágio foi possível adquirir diversos conhecimentos tanto em OutSystems 10 como 11 e aprender a desenvolver tanto aplicações Web como *reactive*.

Ao longo das duas primeiras etapas do estágio foi possível aprender boas praticas de programação, o funcionamento da empresa, como é o decorrer de um projeto com datas-limite de entrega, aprender a lidar com alterações de última hora que podem alterar o rumo de um *sprint* e conhecer melhor como funciona uma metodologia *agile* com entregas constantes que podem sofrer alterações dependendo do *feedback* do cliente.

A partir do terceiro projeto foi possível aprender a comunicar diretamente com clientes através de reuniões com os mesmos e a resolver vários problemas sob pressão. Graças a isto foi possível melhorar as competências de comunicação tanto com clientes como com colegas de equipa e aprimorar a organização no trabalho

Com isto é possível dizer que todos os objetivos iniciais para a execução do estágio foram cumpridos com sucesso. Inicialmente houve problemas tanto na organização de trabalho como de lógica, sendo que isto originava uma necessidade maior de tempo na resolução destes problemas ou de criação de novas soluções. Mas, ao longo do estágio, foi possível melhorar estes aspetos com a ajuda de um supervisor que demonstrava qual seria a melhor abordagem a seguir. Graças a este, foi possível entender de um modo mais simples as várias formas de criação de lógica, para que esta possa ser reutilizava em várias situações diferentes.

Com o passar do estágio surgiram derivadas situações nas quais foi necessário comunicar diretamente com o cliente, como para resolver certos problemas, testar integrações envolvendo as várias equipas e também discutir novas integrações. Nestas situações foi necessário aprender a comunicar de uma forma clara, demonstrando segurança no trabalho realizado.

Com a finalização do estágio é possível identificar uma grande evolução tanto a nível profissional como pessoal. Isto deve-se ao ambiente proporcionado pela empresa e também aos profissionais com os quais foi possível trabalhar, que possibilitaram uma grande evolução técnica e pessoal. Quanto aos projetos elaborados é possível dizer que

os objetivos para estes foram cumpridos com sucesso, tendo apenas sido necessário prolongar a data do segundo projeto para mais uma semana. É possível dizer que se adquiriu um conhecimento em criação de aplicações tanto em OutSystems 10 como no 11, também na criação de aplicações reactive e realização de WebServices e lógica de servidor associada.

Como implementações futuras, o ideal seria melhorar a lógica criada na segunda fase do estágio na página de CSPs, pois esta ficou um pouco confusa e em certas situações seria possível reutilizar a mesma. Também seria ideal melhorar a lógica associada à criação/edição de níveis de um WorkFlow, pois isto, é realizado utilizando lista locais que na situação em questão não é o ideal.

Os futuros objetivos pessoais serão melhorar os conhecimentos em OutSystems e em comunicação. Um dos objetivos principais é também estudar mais detalhadamente a criação de aplicações *reactive* e consequentemente conseguir adquirir a certificação de "Associate Reactive Developer" de OutSystems.

# Bibliografia

[1] R. Sanchis, Ó. García-Perales, F. Fraile, and R. Poler, “Low-code as enabler of digital transformation in manufacturing industry,” *Appl. Sci.*, vol. 10, no. 1, 2020, doi: 10.3390/app10010012.

[2] A. Lamela, “Top 5 main Agile methodologies: advantages and disadvantages - Xpand IT,” *Top 5 main Agile methodologies: advantages and disadvantages* , Oct. 11, 2018. https://www.xpand-it.com/blog/top-5-agile-methodologies/ (accessed Mar. 05, 2022).

[4] E. Merryweather, “What is Agile Development and Why is Everyone Talking About It? - Product School,” Aug. 03, 2020. https://productschool.com/blog/product-management-2/skills/agile-development-everyone-talking/?utm_campaign=202202-paid-b2c_ga_eu_so_pmax_bofu&utm_medium=cpc&utm_source=google_ads&utm_term=b2c_current_pm&gclid=CjwKCAiA1JGRBhBSEiwAxXblwRqCbAam2XFdhXk_Ens4g5Cr6f7oLY6lbOX_C7CTMuaNnBZKMN433xoCwPoQAvD_BwE (accessed Mar. 06, 2022).

[5] R. Sherman, “Project Management,” *Bus. Intell. Guideb.*, pp. 449–492, 2015, doi: 10.1016/B978-0-12-411461-6.00018-6.

[6] Smartsheet, “A Complete Guide to the Waterfall Project Method | Smartsheet.” https://www.smartsheet.com/content-center/best-practices/project-management/project-management-guide/waterfall-methodology (accessed Mar. 06, 2022).

[7] OutSystems, “A Plataforma de Desenvolvimento de Aplicações Modernas | OutSystems.” https://www.outsystems.com/pt-br/platform/ (accessed Mar. 06, 2022).

[8] OutSystems, “OutSystems Platform Services | Evaluation Guide | OutSystems.” https://www.outsystems.com/evaluation-guide/platform-services/ (accessed Mar. 20, 2022).

[9] OutSystems, “Actions in Web Applications - OutSystems.” https://success.outsystems.com/Documentation/11/Developing_an_Application/Implement_Application_Logic/Actions_in_Web_Applications (accessed Mar. 20, 2022).

[10] OutSystems, “User Roles - OutSystems.” https://success.outsystems.com/Documentation/11/Developing_an_Application/Secure_the_Application/User_Roles (accessed Mar. 20, 2022).

[11] OutSystems, “Logic Tools - OutSystems.” https://success.outsystems.com/Documentation/11/Reference/OutSystems_Language/Logic/Implementing_Logic/Logic_Tools (accessed Mar. 20, 2022).

[12] OutSystems, “Service Studio Overview - OutSystems.” https://success.outsystems.com/Documentation/11/Getting_started/Service_Studio_Overview (accessed Mar. 20, 2022).

[13] OutSystems, “Integration Studio - OutSystems.” https://success.outsystems.com/Documentation/11/Reference/Integration_Studio (accessed Mar. 20, 2022).

[14] OutSystems, “Manage Your OutSystems Infrastructure - OutSystems.” https://success.outsystems.com/Documentation/11/Managing_the_Applications_Lifecycle/Manage_Your_OutSystems_Infrastructure (accessed Mar. 20, 2022).

[15] J. Wong, K. , L. A. Iijima, A. Jain, and V. Paul, “Gartner Reprint,” Sep. 20, 2020. https://www.gartner.com/doc/reprints?id=1-27IIPKYV&ct=210923&st=sb (accessed Mar. 22, 2022).

[16] javaTpoint, “C# History - javatpoint.” https://www.javatpoint.com/csharp-history (accessed Apr. 11, 2022).

[17] M. Watson, “What is C# used for?,” 2020. https://stackify.com/what-is-c-used-for/ (accessed Apr. 11, 2022).

[18] Smartify, “What is Jira? Advantages and Disadvantages of Jira,” 2018. https://smartifysol.com/blog/testing/what-is-jira-advantages-and-disadvantages-of-jira-blog (accessed Apr. 11, 2022).

[19] JiraSoftware, “What is Jira Software used for? | Atlassian.” https://www.atlassian.com/software/jira/guides/use-cases/what-is-jira-used-for#Jira-for-requirements-&-test-case-management (accessed Apr. 11, 2022).

[20] F. Parvez, “Introduction to CSS | CSS Tutorial for Beginners,” Aug. 01, 2021. https://www.mygreatlearning.com/blog/css-tutorial/#t1 (accessed Apr. 11, 2022).

[21] A. Macwan, “Benefits of Working with Agile Methodologies in Software Development,” Jul. 27, 2019. http://www.alltechflix.com/agile-methodologies-in-software-development/ (accessed May 09, 2022).

[22] H. Reactor, “What is JavaScript used for? | Hack Reactor,” Aug. 26, 2021. https://www.hackreactor.com/blog/what-is-javascript-used-for (accessed May 11, 2022).

[23] M. Piccini, “Conheça as 4 principais funcionalidades do Microsoft Teams!” https://tigraconsult.com.br/2020/11/02/funcionalidades-microsoft-teams/ (accessed May 29, 2022).

[24] A. Camus, “Você sabe o que é CSS e como se relaciona com HTML?,” Oct. 14, 2020. https://www.crehana.com/pt/blog/brasil/voce-sabe-o-que-e-css-e-como-se-relaciona-com-html/ (accessed May 29, 2022).